Universidade Federal Fluminense

CARLOS HENRIQUE SANT' ANA DA SILVA

Previsão de Carga em Aglomerado de Servidores Web Aplicada a Economia de Energia

NITERÓI

2010

CARLOS HENRIQUE SANT' ANA DA SILVA

Previsão de Carga em Aglomerado de Servidores Web Aplicada a Economia de Energia

Dissertação de **Mestrado** *submetida* ao "Programa de Pós-Graduação em Computação" da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre. Área de concentração: Processamento Paralelo e Distribuído.

Orientador:

JULIUS C. B. LEITE

Universidade Federal Fluminense

NITERÓI

2010

Previsão de Carga em Aglomerado de Servidores Web Aplicada a Economia de Energia

Carlos Henrique Sant' Ana da Silva

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre. Área de concentração: Processamento Paralelo e Distribuído.

Aprovada por:

Prof. Orlando Loques, Ph.D. / IC-UFF

Profa. Cristina Boeres, Ph.D. / IC-UFF

Prof. Alexandre Sztajnberg, D. Sc. / IME-UERJ

Niterói, 10 de Março de 2010.

"É melhor tentar e falhar, que preocupar-se e ver a vida passar; é melhor tentar, ainda que em vão, que sentar-se fazendo nada até o final. Eu prefiro na chuva caminhar, que em dias tristes em casa me esconder. Prefiro ser feliz, embora louco, que em conformidade viver ..."

Martin Luther King

"Comece fazendo o que é necessário, depois o que é possível, e de repente você estará fazendo o impossível."

São Francisco de Assis

Dedico este trabalho aos meus pais e irmão que são os pilares da minha vida e a Deus por prover a luz do meu caminho.

Agradecimentos

Agradeço em primeiro lugar e igualmente ao meu pai José Henrique e minha mãe Maria Alice por todo carinho, dedicação e apoio dado em todos os momentos. Que todo aborrecimento e preocupação que eu causei a vocês sejam amenizados pela obtenção desse título. Ao meu irmão Mário Henrique, por todos os conselhos e preocupação comigo mesmo apesar da distância que nos separa. Ele sabe que a fraternidade que nos une é proporcional ao tempo que estamos longe. Agradeço também à toda minha família, por toda torcida e orações feitas em meu nome. Obrigado tia Penha pela ajuda e pelos conselhos pedagógicos que me deste no início da escrita desse trabalho.

Aos meus amigos de república Bruno, Edelberto, Felipe, Gustavo, Lucas e Tadeu. Cada um na sua medida sempre soube fazer do apartamento que compartilhamos não somente um local para descansar, mas sim um refúgio para as piores situações que encontrei durante essa jornada. Que a nossa amizade continue por longos anos. Aos meus amigos de Juiz de Fora: Bruno, Felipe, Fernando, Júnior, Yuri e demais. Vocês tornaram meus retornos cada vez mais revigorantes. Ao meu casal de amigos favorito: Bruno e Karen. Obrigado pela torcida e também desejo sucesso a vocês no novo desafio que enfrentam.

Também sou grato a todos companheiros do Laboratório Tempo. Ao Professor Orlando por todo incentivo e confiança. Ao Douglas, Giulio, Matheus e Rodrigo pela amizade que desenvolvemos. Ao Alessandro Copetti e Sérgio por todos os momentos de trabalho e companheirismo. Ao Vinícius, pela presença e presteza em muitas dificuldades que enfrentei. Uma menção especial ao Luciano Bertini, que na defesa desse trabalho não fazia mais parte do nosso grupo, mas que tem grande mérito no trabalho que foi desenvolvido aqui. Se o seu doutorado não estivesse em curso, ele seria forte candidato ser a co-orientador desse trabalho.

Ao meus companheiros do Instituto de Computação (IC) André, Anand (Tenso! Carroça!), Ângela, Cintia, Idálmis, Jacques, Jesuliana, Leandro, Luciana, Mônica, Marcelo, Renatha, Rômulo, Simone, Stênio e a todos os outros que não mencionei por tornarem o meu ambiente de trabalho como um todo um lugar mais humano e agradável. Às

secretárias pelo zelo e chamadas para os seminários e aos funcionários do suporte pela competência. Aos professores do IC-UFF, em especial aos membros do Colegiado desse programa, pelas oportunidades e confiança na concretização desse trabalho.

Aos meus médicos Dra. Anacely, Dr. Heraldo Vícter e Dr. Luiz Antônio por deixarem minha mente, coração e corpo preparados para o desafio que foi desenvolver este trabalho.

Ao CNPq por financiar a maior parte dos meus estudos.

Obrigado Professor Daniel Mossé por toda contribuição feita nesse e em outros trabalhos. Suas sugestões, dicas e ensinamentos tornam-me testemunha do seu brilhantismo e versatilidade. Muitas das coisas que aprendi não podem ser encontradas em publicação alguma.

Agradeço ao meu orientador Professor Julius Leite por construir e proporcionar um ambiente mais do que adequado para desenvolver esse trabalho. Se este é o caminho dos bem-sucedidos, estou aqui graças ao entusiasmo, saber e competência dele. Com certeza trabalhar com o Prof. Julius durante todo esse tempo redefiniu a abrangência da palavra *orientação*.

E agradeço a Deus, sobre todas as coisas.

Resumo

A complexidade e os requisitos das aplicações web vêm aumentando a fim de satisfazer modelos de negócios cada vez mais sofisticados (*web services* e computação em nuvem, por exemplo). Por esta razão, características como desempenho, escalabilidade e segurança são tratadas no projeto de aglomerados de servidores *web*. Entretanto, devido à crise energética, o consumo de energia neste tipo de ambiente se tornou uma grande preocupação. Para tratar esse problema, é proposta uma política de redução de consumo de energia em um aglomerado de servidores *web*, utilizando previsão de carga combinada com técnicas de DVFS (*Dynamic Voltage and Frequency Scaling*) e de configuração dinâmica. Para validar esta política preditiva, uma aplicação *web* executando um perfil de carga real foi testada em um protótipo de *cluster* de servidores, e a energia consumida e a qualidade de serviço obtidas foram usadas para avaliar o desempenho do sistema.

Palavras-chave: Aglomerados de Servidores *Web*, Gerenciamento de Energia, Consumo de Energia, Previsão de Carga, Qualidade de Serviço, Configuração Dinâmica

Abstract

The complexity and requirements of web applications are increasing in order to meet more sophisticated business models (web services and cloud computing, for instance). For this reason, characteristics such as performance, scalability and security are addressed in web server cluster design. However, due to the energy crisis, the energy consumption in this type of environment became a major concern. Aiming to avoid this adverse scenario, this dissertation shows how energy consumption reduction can be achieved in a web server clustered environment. The energy consumption reduction policy used here is based on load forecasting combined with DVFS (Dynamic Voltage and Frequency Scaling) and dynamic configuration techniques. To validate this predictive policy, a web application running a real workload profile was submitted to a server cluster testbed. In addition a criteria for quality of service was used to evaluate system's performance.

Keywords: Web Clusters, Power Management, Energy Consumption, Load Forecasting, Quality of Service, Dynamic Configuration.

Abreviações

CMOS	:	Complementary Metal-Oxide-Semiconductor
CPU	:	Central Processing Unit
DVFS	:	Dynamic Voltage and Frequency Scaling
HTTP	:	Hyper-Text Transfer Protocol
IETF	:	Internet Engineering Task Force
IP	:	Internet Protocol
NFS	:	Network File System
PDA	:	Personal Digital Assistant
PHP	:	PHP: Hypertext Preprocessor
QOS	:	Quality of Service
SSL	:	Secure Sockets Layer
TI	:	Tecnologia da Informação
TCP	:	Trasnsmission Control Protocol
TSL	:	Transport Layer Security
UDP	:	User Datagram Protocol
URI	:	Uniform Resource Identifier
URL	:	Uniform Resource Locator
USB	:	Universal Serial Bus
W3C	:	World Wide Web Consortium
WebDAV	:	Web-based Distributed Authoring and Versioning
VNC	:	Virtual Network Computing

Sumário

Lista de Algoritmos

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Com o passar dos anos, o consumo de energia elétrica vem tornando-se uma preocupação cada vez mais importante no desenvolvimento de novas tecnologias, pois sua produção representa uma importante questão ambiental e econômica. Segundo projeções feitas pela Agência Internacional de Energia [30] (*International Energy Agency* ou IEA), a emissão de gases que ocasionam o efeito estufa vai dobrar no fim deste século, causando dessa forma um aumento de 6° C na temperatura global. Como em muitos países a principal fonte de produção de energia elétrica são os combustíveis fósseis, medidas de largo alcance estão sendo tomadas para que essa crise energética seja superada.

Além de serem um dos principais causadores do efeito estufa, os combustíveis de origem fóssil (tais como derivados do petróleo e carvão) são encontrados em uma quantidade limitada, ou seja, são fontes de energia *não renováveis*. Com base nessas duas afirmações, países ao redor do mundo têm realizado medidas para renovação de sua matriz energética, a fim de diminuir o uso de combustíveis dos tipos acima citados e substituí-los por outros de fontes limpas e principalmente renováveis.

Esforços de caráter multinacional têm tido destaque com relação à busca de energias limpas e renováveis. Sob a premissa de que o deserto recebe mais energia do sol do que a humanidade consumiria em um ano inteiro, o projeto Desertec [20] é uma iniciativa ambiciosa de reunir energia limpa vinda do deserto africano para países do norte da África, Oriente Médio e Europa. Nas Ilhas Galápagos, após um acidente com um petroleiro causar o derrame de mais de 500.000 litros de combustível em uma baía da Ilha de San Cristóbal, o governo equatoriano iniciou juntamente com a ONU (Organização das Nações Unidas) a instalação de energia eólica visando diminuir riscos de acidentes oriundos do transporte de combustível [27].

Considerada a terceira maior nação de potencial hídrico no mundo, o Brasil mostra pouca dependência em relação aos combustíveis fósseis. Ao contrário de países como os EUA e China que possuem como importante fonte de geração de energia elétrica o carvão, as usinas hidroelétricas produzem 69% da capacidade total brasileira. Dados mais detalhados da matriz elétrica brasileira [4] podem ser vistos na Tabela 1.1. Mesmo com uma matriz energética sustentada por uma fonte de energia limpa, as hidroelétricas possuem um alto custo de implantação. A instalação de uma usina desse tipo exige que grandes áreas sejam inundadas, destruindo assim parte do ecossistema e demandando do governo local um gasto com indenizações e novas moradias para as pessoas que residem nessas áreas.

Tabela 1.1: Matriz elétrica brasileira

Tipo	Usinas	Capacidade (kW)	%
Hidroelétrica	802	78.016.4997	69,03
Gás	121	11.844.285	10,48
Petróleo	800	5.553.764	4,91
Biomassa	331	5.563.943	4,92
Nuclear	2	2.007.000	1,78
Carvão Mineral	8	1.455.104	1,29
Eólica	33	414.480	0,37
Importação	–	8.170.000	7,23

Outra vertente aliada aos severos prejuízos ambientais que são trazidos pela utilização de fontes de origem fóssil é a questão econômica. Recursos que possuem um ciclo de vida finito ou que de alguma forma podem ser considerados não renováveis possuem um preço de marcado muito elevado. Desta forma, a busca por novos combustíveis e fontes de energia tem se ampliado cada vez mais. Mas enquanto essa busca evolui, medidas de largo escopo estão sendo tomadas para que essa crise energética seja atenuada.

Na área de sistemas computacionais, no nível de *hardware*, essa preocupação pode ser comprovada com a criação da *Advanced Configuration and Power Interface* [2] ou ACPI. Representando a união entre empresas tidas como referências no mercado (HP, Intel, Microsoft, Phoenix e Toshiba), elas estabeleceram uma especificação para gerenciamento de energia e temperatura em computadores de maneira geral. Outra iniciativa de destaque é o programa *Energy Star* [23] da Agência de Proteção Ambiental dos Estados Unidos (*Environmental Protection Agency* ou EPA). Este programa visa a adoção de práticas e produtos com maior eficiência energética, tanto por empresas quanto pela população de maneira geral.

A EPA divulgou recentemente um relatório [22] no qual cita uma série de medidas

necessárias para promover o aumento da eficiência nos *datacenters*. Também chamados de Centros de Processamento de Dados (CPD) ou fazenda de servidores, esta infra-estrutura reúne vários computadores para a execução de processamento, armazenamento, e comunicação de dados, dentre outras funções. A justificativa para a escolha desse tipo de infra-estrutura como foco desse relatório são as projeções feitas em relação ao seu consumo de energia. Estima-se que o crescimento do consumo de energia pelas fazendas de servidores, somente nos Estados Unidos, irá efetivamente dobrar até 2011, passando de 61,4 para 124,5 bilhões de kWh gastos por ano.

As medidas contidas no relatório para a economia de energia em *datacenters* englobam desde mudanças no *hardware* (substituição para aumento de eficiência energética) até a utilização de políticas de gerenciamento de energia por parte de aplicações, servidores e equipamentos de rede e armazenamento de dados. Também são necessárias mudanças no ambiente no qual as máquinas serão armazenadas para que uma maior economia de energia seja alcançada. Como consequência dessa redução uma maior redução nos gastos e uma menor emissão de CO_2 será alcançada, conforme mostrado na Tabela 1.2.

Tabela 1.2: Previsão para economia anual alcançada em 2011

Cenário	Energia (bilhões de kWh)	Custo (bilhões US$)	Emissão CO_2 (milhões de t)
Aprimoramento	23	1,6	15
Melhores práticas	60	4,1	38
Estado da arte	74	5,1	47

O cenário de **aprimoramento** das operações inclui melhorias que envolvem somente uma melhor administração do capital, ou seja, a adoção de medidas como escolha de equipamentos mais eficientes para renovação dos já existentes. No contexto de **melhores práticas** são considerados ganhos obtidos com a adoção em larga escala de práticas e tecnologias utilizadas nas melhores infra-estruturas de TI. O último cenário prevê a maior redução do consumo de energia (assim como menor valor gasto com energia e menos emissão de CO_2 na atmosfera) através do emprego de tecnologia e gerenciamento de energia contido no **estado da arte** tanto a nível de software quanto de *hardware*.

Seguindo esta tendência de pesquisa de novos métodos para a economia de energia, esse trabalho descreve uma política para redução no consumo energético em ambientes computacionais chamados de aglomerado de servidores *Web*. Este tipo de ambiente pode apresentar uma série de camadas lógicas (n camadas ou *n-tiers*), mas o tipo mais utilizado é a uma arquitetura de 3 camadas, como ilustrado na Figura 1.1. A primeira camada nesta arquitetura representa iteração mais imediata do *cluster* com o mundo externo. O *front-*

end é responsável por atender as requisições que chegam ao aglomerado e encaminhá-las a próxima camada. A camada 2 representa a lógica do modelo de serviço provido por este ambiente. Ela também é responsável por realizar o processamento de dados, cálculos e decisões necessárias prover os recursos demandados pelo cliente. Já a camada 3 representa o diferencial dessa arquitetura para a conhecida arquitetura cliente/servidor. Ao invés de embutir serviço de armazenamento e manipulação de dados na camada anterior, nesse modelo arquitetural existe uma camada dedicada a desempenhar essas funções.

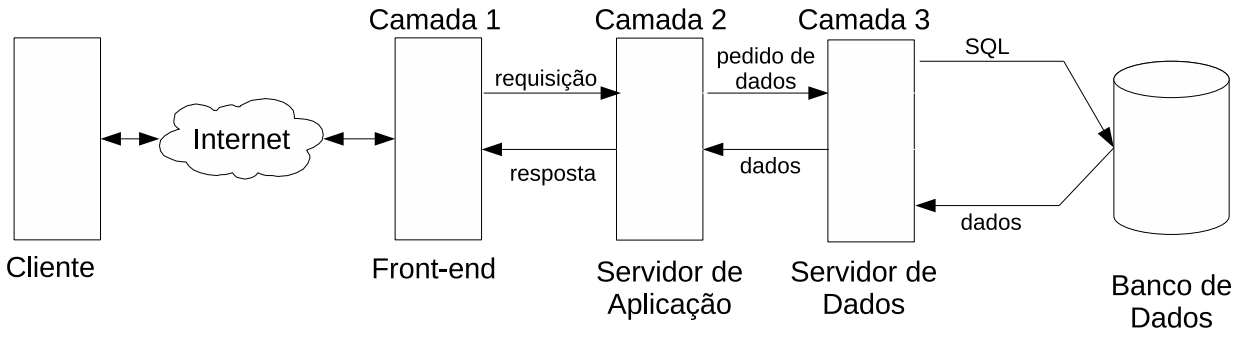


Figura 1.1: Arquitetura de 3 camadas

Empregando mecanismos de previsão de carga (uma estimativa de um valor para um instante no futuro) no ajuste da configuração do sistema computacional (ou seja, ligando e desligando servidores e ajustando a velocidade de seus processadores), a escolha daquela mais eficiente para uma dada carga pode ser feita de forma pró-ativa e não reativa. Dessa forma, um incremento no ganho com relação à economia de energia pode ser obtido, e a qualidade de serviço (QoS) contratada pela aplicação ainda ser satisfatoriamente atendida.

Uma outra contribuição desse trabalho é a forma pela qual a manutenção da qualidade de serviço está relacionada aos atributos do sistema. Assume-se que a manutenção de um determinado nível de QoS baseia-se tão somente na utilização do processador dos servidores, limitando-a a um valor máximo; essa política torna-se aplicável em uma maior variedade de ambientes, visto que essa métrica é independente do tipo de serviços executados no sistema computacional em questão.

Para desenvolver tal política com as características descritas nos parágrafos precedentes, um ambiente simplificado para a execução de uma aplicação *Web* foi inicialmente utilizado. Essa infra-estrutura, como apresentado em [42], possuía somente uma máquina nas camadas 1 e 2. A terceira camada não foi utilizada porque o foco do estudo encontra-se nos efeitos que um mecanismo de previsão tem sobre uma política de economia de energia, e deseja-se evitar os problemas de tratamento de consistência em bases de dados. Após vistos os benefícios proporcionados pela previsão de carga, o estudo realizado

em [43] expandiu o modelo anterior para uma arquitetura com mais de um computador na camada 2 tornando, desta forma, o ambiente mais realista.

Além dos resultados publicados em estudos anteriores, este trabalho apresenta melhorias realizadas no modelo. Essas mudanças visam a melhoria da qualidade de serviço da aplicação em situações nas quais uma máquina previamente desligada deve ser ligada. Outro diferencial é a realização de uma análise de sensibilidade de parâmetros. Essa análise tem como objetivo mostrar o impacto que algumas variáveis exercem sobre os resultados obtidos. É importante ressaltar que em todos os trabalhos feitos empregando o mecanismo de economia de energia preditivo aqui proposto foram testados em um ambiente real de *cluster* de servidores.

O restante dessa dissertação é apresentada como se segue: o **Capítulo 2** versará sobre os trabalhos relacionados em economia de energia em vários ambientes computacionais, bem como sobre as técnicas de economia de energia existentes. O **Capítulo 3** descreverá o modelo preditivo utilizado para a elaboração da política para redução de energia. O **Capítulo 4** discutirá mecanismos de previsão e alguns conceitos relacionados. Já o **Capítulo 5** abordará algumas das ferramentas utilizadas para construir a implementação da aplicação *Web* sobre um aglomerado de servidores, e documentar o uso das mesmas com o intuito de facilitar outros trabalhos que virão. Já o **Capítulo 6** descreverá como o ambiente de testes foi gerado, bem como o emprego das ferramentas descritas no capítulo anterior. No **Capítulo 7** serão mostrados resultados quantitativos e qualitativos obtidos nos experimentos conduzidos. Por último, o **Capítulo 8** contém as considerações finais, bem como indicações de trabalhos futuros.

Capítulo 2

Trabalhos Relacionados

Em função de sua crescente importância, a área de Economia de Energia em sistemas computacionais vem ganhando notoriedade e uma base mais abrangente de aplicações. Por exemplo, em dispositivos móveis (*laptops* e PDAs), restrições referentes ao consumo de energia desde sempre tiveram atenção. Em virtude de seu crescente poder de processamento, eles precisam maximizar o tempo de vida de sua fonte de energia já que não possuem (na maioria do tempo) uma ligação com uma fonte fixa. Com este tipo de restrição em vista, além da redução de energia obtida colocando-se alguns componentes de *hardware* em estado de baixo consumo de energia (*display* e disco, por exemplo), trabalhos tais como como [24] procuraram adicionar a alguns tipos de aplicações móveis um perfil adaptativo em relação ao estado da capacidade de fornecimento de energia pela bateria.

Além do processador, outros tipos de componentes e sistemas de *hardware* também evoluíram na questão da economia de energia. Dentre esses componentes podemos citar dispositivos de armazenamento de massa [25], sistemas de memória RAM [1, 46], dispositivos de rede [13] e outros. Contudo, como o processador possui um gasto energético substancialmente maior que o de outros componentes de um computador, em função de sua utilização [14], este e outros trabalhos que ainda serão mencionados têm a CPU (*Central Processing Unit*, Unidade de Processamento Central ou UCP) como seu principal foco.

Cenários envolvendo aplicações do tipo multimídia também foram explorados na literatura, como, por exemplo, no estudo apresentado em [50]. Este trabalho aborda a construção de um escalonador para sistemas de tempo-real não críticos (*soft real-time*) que tem como objetivo economizar energia a partir da previsão dos ciclos demandados por uma aplicação multimídia. Outro estudo contido nesse contexto é apresentado em [32], o qual descreve o uso de um algoritmo utilizando o mecanismo de DVFS aplicado a um

decodificador de áudio e vídeo.

Maiores oportunidades para economia de energia aparecem, contudo, em ambientes que utilizam grande quantidade de computadores. Devido à natureza das aplicações existentes na atualidade, ambientes tais como fazendas de servidores e aglomerados de computadores (daqui em diante *clusters* será utilizado como sinônimo) ganharam notoriedade. As técnicas desenvolvidas para esse tipo de sistema consideraram, inicialmente, somente *clusters* compostos por computadores homogêneos [21, 39], onde o uso de configuração dinâmica (ligar e desligar máquinas do *cluster* de acordo com a utilização do mesmo) baseada em uma política de gerenciamento global obteve resultados significativos.

Uma das primeiras iniciativas utilizando um ambiente computacional heterogêneo foi feita por [41]. Nesse trabalho, as técnicas de DVFS e configuração dinâmica são combinadas com o propósito de redução do consumo de energia associada à manutenção da qualidade de serviço provida para uma aplicação *Web*. Nesse caso, o *cluster* considerado é composto por duas camadas, uma sendo o *front-end*, que distribui a carga e gerencia a configuração, e a outra abrangendo as camadas de aplicação e base de dados. O trabalho pressupõe que a maioria das requisições são atendidas a partir da *cache* de dados em memória, o que realça a importância do controle de energia em nível de processador (DVFS).

O estudo apresentado em [29] discute uma infra-estrutura na qual estão presentes três camadas: interface, negócios e armazenamento de dados. Minimizando o atraso total entre camadas, um modelo utilizando um controlador com realimentação é responsável pela execução do DVFS. Nesse trabalho não há menção a configuração dinâmica porque somente um *path* foi utilizado na avaliação. Por *path* os autores entendem uma máquina na segunda camada e uma máquina na terceira camada. Esse trabalho, contudo, é baseado em um modelo de filas, o que limita os tipos da distribuição dos tempos da carga oferecida e da distribuição do tempo de processamento a modelos markovianos.

Também utilizando uma aplicação *Web* de três camadas, o trabalho apresentado em [9] mostra um modelo no qual a gerência de economia de energia leva em consideração os níveis de QoS contratados pela aplicação, assumindo um sistema do tipo *soft real-time*. Com base em uma grandeza chamada de *tardiness*, um controlador consegue manter a qualidade de serviço acima de um patamar pré-estabelecido. Por *tardiness* os autores entendem a relação entre o tempo de resposta de cada requisição (medido durante a execução) dividido pelo seu prazo de execução (estabelecido nas especificações). O trabalho apresenta duas formas de modelar a função de distribuição dessa variável, e mostra como

garantir que um dado percentil das requisições irá atender ao contratado. Posteriormente, em [11], os mesmos autores estendem o trabalho anterior e realizam o controle de QoS e de consumo de energia sem fazer nenhuma hipótese sobre a forma da distribuição da variável *tardiness*, através do uso de aproximação estocástica.

Buscando um melhor aproveitamento e racionalização dos recursos providos por um *datacenter*, o emprego do mecanismo de virtualização tem obtido grande destaque. Além de prover grandes benefícios (escalabilidade, melhor gerenciamento e utilização do *hardware*), o uso deste tipo de técnica traz novos desafios para a implementação de redução no consumo de energia [48]. Apesar de não ser o escopo deste trabalho, a metodologia aqui utilizada para realizar a previsão de carga pode servir como ferramenta para técnicas existentes de economia de energia em ambientes virtualizados [33, 38].

O estudo conduzido em [16] analisa um perfil de carga de trabalho (ou *workload*) obtido através de registros (*logs*) do mensageiro instantâneo Microsoft MSN. O foco principal desse trabalho é disponibilizar um determinado conjunto de servidores para atender a demanda da aplicação em questão (*server provisioning*) e distribuir a grande quantidade de requisições recebidas (*load dispatching*). No primeiro desses dois processos, existe o emprego de uma técnica de previsão sobre a carga enviada ao sistema. Com o resultado dessa previsão é estabelecida a quantidade de servidores necessários para atender a carga e economizar energia. O previsor SPAR (*Sparse Periodic Auto-Regression* atua como um previsor de curto alcance, ou seja, realiza previsões para instantes futuros próximos.

Dividido em duas partes, esse previsor contém uma parte periódica e outra para efetuar ajustes locais. A primeira parte do SPAR assume que dentro de uma periodicidade maior (um dia da semana, por exemplo) os trechos da carga terão um alto nível de correlação entre si, enquanto que a segunda parte faz a mesma suposição para um período menor (intervalo de uma hora dentro de um mesmo dia). Essas duas suposições revelam que o desempenho da técnica de previsão está fortemente relacionada a presença de sazonalidade no perfil de carga utilizado. O modelo proposto nesta dissertação leva em consideração uma técnica que embora seja mais simples não precisa que nenhuma inferência seja feita sobre o padrão do *workload*. Outra característica diferencial presente no estudo desenvolvido aqui é a utilização do mecanismo de DVFS para realizar a redução no consumo de energia.

Absorvendo influência de trabalhos mencionados anteriormente, o estudo desenvolvido aqui visa colaborar com o estado da arte incorporando técnicas de previsão de carga a técnicas de economia de energia. De tal forma, um sistema que consegue estimar algo

sobre a carga de trabalho a qual eventualmente será submetido, possibilita que medidas direcionadas tanto para economia de energia quanto para manutenção da qualidade de serviço possam ser tomadas. Ou seja, uma política de energia baseada nesse paradigma passará de reativa a preditiva.

Além de fatores que influenciam o método de previsão, elementos característicos que ocorrem durante o tempo de execução de um sistema são mostrados e analisados (Capítulo 7). Diferentemente de outros trabalhos, muitos deles simplesmente baseados em simulação, nossa implementação ocorre em um protótipo de *cluster*, que é submetido a um perfil de carga real: um trecho do *workload* da Copa do Mundo de Futebol de 1998 [7]. A respeito da alta autocorrelação dessa carga [8], diversos desafios existem quando se faz a implementação em um ambiente real.

Capítulo 3

Política Preditiva para Economia de Energia

3.1 Aglomerado de Servidores *Web*

A estrutura adotada para aplicar a metodologia desenvolvida nesse trabalho é um aglomerado de servidores *Web*, podendo ser referenciado daqui por diante como aglomerado *Web* ou *web cluster*. Um aglomerado *Web* refere-se a um conjunto de máquinas servidoras que estão agrupadas em um mesmo local interconectadas através de uma rede de alta velocidade e representam para o meio externo um único sistema. Cada nó servidor desse *cluster* normalmente contém seu próprio disco e um sistema operacional completo. Para concentrar a atenção em efeitos de previsão de carga, aqui será assumido que a base de dados desse *cluster* é replicada em cada um dos nós de nível 2 (como visto na Figura 1.1). A Figura 3.1 ilustra um exemplo dessa arquitetura.

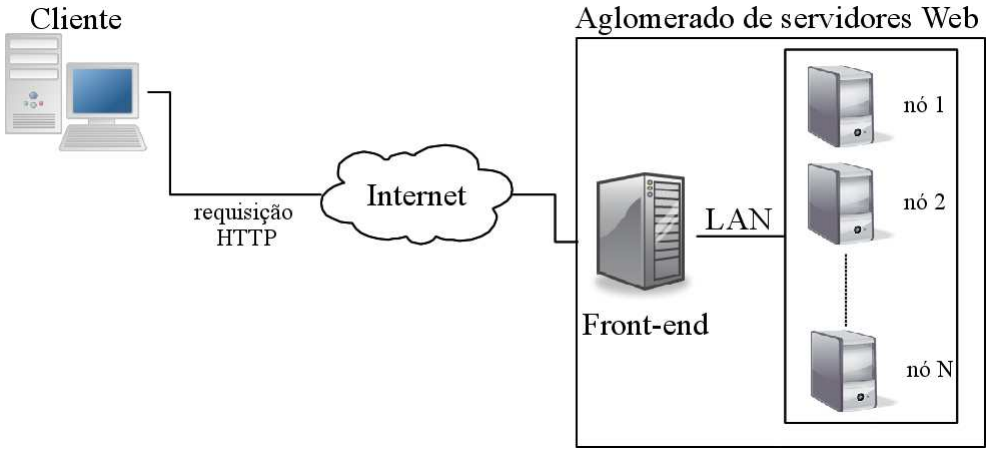


Figura 3.1: Arquitetura de um *web cluster*

Além dos nós responsáveis pela disponibilização do serviço existe uma entidade de-

nominada *front-end* (FE). O FE representa o ponto de interação entre a Internet e o *web cluster* permitindo deixar a natureza distribuída desse ambiente transparente para suas aplicações e usuários. O *front-end* recebe todos os pacotes enviados pelos clientes ao *cluster* e os encaminha para algum dos nós. Agindo dessa maneira, o FE se comporta como um distribuidor central de carga de um sistema distribuído. Além de distribuir a carga enviada ao *cluster*, outros papéis que podem ser desempenhados por esse elemento são balanceamento de carga e *proxy* reverso (caso os nós estejam em uma rede privada). Uma melhor caracterização de um ambiente dessa natureza, assim como suas variantes, pode ser encontrada em [15].

Os *datacenters* é um tipo de ambiente que favorece a implantação de políticas para redução de energia. Segundo os estudos presentes em [21], um menor consumo de energia pode ser alcançado se informações a respeito das outras máquinas do *cluster* forem usada pelo FE na utilização de uma configuração global. Aliada a este fato, essa organização facilita o balanceamento de carga e permite um tratamento adequado da heterogeneidade dos servidores. No quesito escalabilidade, um esquema hierárquico com mais de um computador ou hardware FE podem ser utilizados para expandir esta estrutura.

3.2 Modelo

Para descrever a aplicação de um modelo preditivo sobre um *web cluster*, primeiro será definido o tipo de serviço provido pela mesmo. O tipo de aplicação adotado foi construído tomando como base valores de referência contidos no TPC-W [47]. Nesse padrão é definido um conjunto de requisições *Web* destinadas a simular um ambiente de comércio eletrônico (*e-commerce*). Para cada tipo de requisição, existe um limite de tempo para que ela seja atendida. Para a modelagem da aplicação usada nesse trabalho utilizamos esse mesmo método: atribuímos um tempo de execução média e um prazo para as requisições enviadas ao aglomerado de servidores *Web*. Dessa forma, conseguimos estabelecer um padrão de tempo real não crítico para esta aplicação, bem como uma métrica de qualidade de serviço (fração das requisições que conseguem ser atendidas no prazo).

O modelo de aglomerado *Web* utilizado para avaliar a política preditiva possui as mesmas entidades ilustradas na Figura 3.1. No *cluster* em questão, o FE tem o papel de realizar o balanceamento de carga, *proxy* reverso (encaminhamento de requisições do mundo exterior para o ambiente distribuído e o caminho reverso) e outras funções descritas ao longo desse trabalho. Os nós (também chamados servidores trabalhadores,

worker servers ou simplesmente *workers*) possuem a função de processar as requisições vindas do ambiente externo e devolve-las ao FE. O lado cliente é aquele responsável por gerar a demanda ou carga de trabalho a qual o aglomerado será submetido.

Visando um emprego independente da aplicação executada no *web cluster*, a grandeza alvo para aplicação da técnica de previsão será a carga a qual o sistema está sendo submetido. Ao longo do tempo, amostras do valor da carga serão coletadas pelo FE a fim de fornecer os dados para alimentar o previsor. Após coletados um certo número de amostras, o previsor será invocado e uma nova previsão para o valor da carga calculada. O tempo necessário para que essas amostras sejam obtidas será denominado **janela de decisão**. A partir dessa estimativa para o valor futuro da carga, é possível escolher uma configuração do aglomerado que forneça desempenho suficiente para cumprir a demanda e economizar energia ao mesmo tempo.

Na escolha de uma configuração, as seguintes opções estão disponíveis: a primeira delas é ligar máquinas que são mais eficientes em termos de consumo de energia e desligar aquelas que não são necessárias. Definidas as máquinas que estão ligadas, o próximo passo é saber em qual será a sua frequência de operação. Para combinar as melhores escolhas com o intuito de minimizar o consumo de energia do *cluster*, o seguinte problema de otimização [12] é resolvido:

Minimizar

$$P = \sum_{i=1}^{N} \sum_{s=1}^{S_i-1} \left\{ \alpha_i^s P_i^s + \beta_i^s P_i^{s+1} \right\} \tag{3.1}$$

Sujeito à:

$$\sum_{i=1}^{N} \sum_{s=1}^{S_i-1} \left\{ \alpha_i^s H_i^s + \beta_i^s H_i^{s+1} \right\} \geq Hbase \tag{3.2}$$

$$\alpha_i^s + \beta_i^s - y_i^s = 0 \tag{3.3}$$

$$\sum_{s=1}^{S_i-1} y_i^s \leq 1 \tag{3.4}$$

$$\alpha_i^s \in [0,1], \beta_i^s \in [0,1], y_i^s \in \{0,1\} \tag{3.5}$$

$$\forall i \in \{1...N\}, \forall s \in \{1...S_i - 1\} \tag{3.6}$$

A função objetivo desse problema (Equação 3.1) mostra a minimização do consumo de energia do *cluster* através da escolha da frequência de operação de cada processador partindo do princípio de que a energia é expressa em função da potência ao longo do tempo e que cada frequência possui um valor de potência associado. Tendo em vista que

esse modelo emprega o uso de valores contínuos para a frequência, é necessário que o valor da potência gasta por ela seja expressa em função dos valores discretos disponíveis pelo *hardware*. Portanto, temos a potência gasta por uma frequência contínua desejada é expressa por: $\alpha_i^s P_i^s + \beta_i^s P_i^{s+1}$, onde P_i^s e P_i^{s+1} representam o consumo de potência de duas frequências da máquina i. Para maiores detalhes sobre frequências contínuas, vide Seção 5.2.1.

A variável H_i^s é a carga máxima de um determinado *worker* para uma determinada frequência, sendo que P_i^s representa a potência consumida nesta mesma frequência. Já y_i é uma variável binária e inteira a qual garante que somente uma frequência por servidor é escolhida (Equação 3.4). Os conjuntos N e S representam respectivamente o conjunto de *workers* existente no *cluster* e a quantidade total de frequências para cada uma das máquinas do outro conjunto.

Obter uma configuração do *cluster* para um determinado nível de carga é uma tarefa simples quando existe um pequena quantidade de máquinas *workers*. Para um número maior de *workers*, o tempo de cálculo gasto por um *solver* para obter uma configuração ótima pode crescer significativamente. Um *solver* é um tipo de programa utilizado para de resolver problemas de otimização utilizando modelos exatos, tais como técnicas de programação linear, inteira ou mista. Do conjunto de *solvers* existentes na literatura, foram utilizados o COIN [17], GLPK [26] e o SCIP [44], pois durante o desenvolvimento da escrita deste trabalho apresentavam-se como *softwares* livres.

Os *solvers* mencionados anteriormente obtiveram os resultados indicados na Tabela 3.1, para resolução da modelagem expressa pelas Equações de 3.1 a 3.6. Nesse caso, para uma instância (ou dados de entrada) contendo um *cluster* com N *workers*, o *solver* minimizava a potência para uma variação de carga de 10% até a carga máxima. Resolvendo instâncias com tamanho maior do que 20 (como 50 e 100, por exemplo) no pior caso observou-se tempos maiores do que 10 horas. Sendo assim, optou-se pela solução *offline* e a construção de uma tabela com valores de configuração para que somente consultas fossem feitas de maneira dinâmica, isto é, durante o funcionamento do sistema. Além do número de servidores do aglomerado, a potência e o desempenho medidos em determinada frequência para cada um dos processadores foi levada em consiredação.

Para a construção da tabela de possíveis configurações para o *cluster* foi utilizado o seu tamanho total ao invés de valores pré-determinados para a carga. Essa abordagem permite que cada linha da tabela mostre a porcentagem da carga na qual haverá mudanças de frequência ou configuração do *cluster* (quais máquinas estarão ligadas). A Tabela 3.2

Tabela 3.1: Tempo gasto para resolver o problema de minimização da potência do *cluster*

Solver	Tamanho da instância	Pior Caso (s)
GLPK	5	< 0.1
	10	0.4
	20	3636.9
COIN	5	0.05
	10	1.86
	20	9.27
SCIP	5	0.09
	10	0.53
	20	3.24

mostra um exemplo dessa construção para uma topologia que contém uma configuração (para um aglomerado de 5 servidores) referente a variação de um milésimo no valor da carga máxima. Para os computadores que devem estar desligados o valor de frequência de operação igual a zero aparece na tabela.

Tabela 3.2: Exemplo de tabela de otimização de frequências gerada com 1000 linhas

Carga	*ampere*	*coulomb*	*hertz*	*joule*	*ohm*
1	0	0	1000	0	0
2	0	0	1000	0	0
...	...	...	...	...	...
...	...	...	...	...	...
600	2000	0	1800	0	1971
...	...	...	...	...	...
1000	2000	2400	2400	2200	2600

Tendo em vista que o processo descrito até aqui leva em consideração somente a minimização do consumo de energia do *cluster*, uma solução para atender a qualidade de serviço representa o próximo passo da política a ser implementado. Nesse trabalho, estamos definindo o tempo de resposta das requisições como representativo da QoS a ser atendida. Para cada requisição pode-se associar um prazo (*deadline*), como em um sistema *soft real-time*. Por simplificação, somente um tipo de requisição é assumido, embora a extensão não seja difícil de implementar.

Experimentos preliminares indicaram que uma solução heurística, através da aplicação de uma parcela multiplicativa sobre a estimativa do valor futuro da carga, em função do atendimento ou não a uma QoS contratada, no momento de avaliação de uma nova configuração, pode ser uma solução adequada. Para isso, foi definido um fator de correção de previsão (chamado de γ, ou gama, daqui por diante).

A função desse fator é acionar uma configuração de maior desempenho de acordo com os níveis de QoS obtidos durante a execução do sistema. A fim de cumprir essa meta, esse fator é definido como $\gamma = min(\gamma_{max}, (QoS_{alvo} \div QoS_{atual}))$, caso a QoS não esteja sendo atendida. Se a qualidade de serviço estiver de acordo com a contratada, o valor de γ não será calculado pela equação acima e a ele é atribuído o valor 1. Utilizando essa definição, o fator gama irá provocar a utilização de uma configuração de maior desempenho de acordo com o nível da qualidade de serviço atual do sistema e garantir que configurações com maior poder computacional serão definidas gradualmente (de acordo com a parcela γ_{max}). O Algoritmo 1 mostra o como é feito a computação do fator γ.

```
1 if QoS atual está abaixo de QoS_alvo then
2 |  γ = min(γ_max, (QoS_alvo ÷ QoS_atual));
3 else
4 |  γ = 1.0;
5 end
6 γ = γ / (util_max);
```

Algoritmo 1: Algoritmo para ajuste da configuração utilizando parâmetro γ

Adicionalmente, é assumido que limitando-se a utilização do servidor a um valor menor do que 100%, a QoS será mais adequadamente atendida. Vários estudos constataram que níveis de utilização superiores a 60-80% tornam o sistema suscetível a ser mais influenciado por variações súbitas de carga, piorando a QoS (como em [41]). Assim, o valor de gama anteriormente obtido é dividido por uma valor de utilização limite (expresso pela variável $util_{max}$), de forma a reduzir a citada influência.

Depois de calculado o valor de γ utilizando o resultado da previsão, uma nova configuração será escolhida e aplicada ao *cluster*. Através de mensagens enviadas pela rede, o FE ajusta a frequência das máquinas que estão ligadas, liga as máquinas que possuem um valor de frequência maior que 0 na tabela e depois começa a desligar aquelas com frequência nula. A gerência desse processo é feita em intervalos regulares de tempo chamados de **janelas de controle**.

Com o intuito de impedir sucessivas operações de ligar e desligar máquinas em um intervalo muito curto, um novo intervalo de tempo foi adotado. A **janela de estabilização** é um período de tempo maior que as outras janelas definidas anteriormente, cuja meta é deixar que o sistema estabilize após mudanças sofridas na configuração (entrada e saída de *workers* no conjunto das máquinas ligadas). Isto significa que durante o tempo que a janela de estabilização estiver ativa, novas configurações não serão obtidas a partir da tabela de possíveis configurações. Entretanto, a demanda durante esse período pode

exigir que a configuração do *web cluster* seja levemente alterada. Caso seja detectado um aumento no valor da carga durante o período de estabilização, significante perda de QoS pode ser constatada. Em caso de um declínio na carga atual do sistema, energia estaria sendo gasta desnecessariamente caso as máquinas atuais pudessem ser desligadas ou suas frequências de operação diminuídas.

```
1  foreach janela de decisão do
2  |  Medição da carga;
3  |  Medição de QoS;
4  |  Adicionar amostra de carga no histórico para ajuste do previsor;
5  |  Ajuste do fator γ;
6  |  Cálculo da previsão;
7  |  Configuração preditiva := carga prevista × tamanho da tabela ×γ;
8  |  Configuração atual ← configuração preditiva;
9  end
10 foreach janela controle do
11 |  Seleção de workers ativos de acordo com configuração vigente;
12 |  if Houve alteração nos servidores ativos then
13 |  |  Ativação do período de estabilização;
14 |  |  Ajuste de frequência dos workers ativos utilizando γ;
15 |  end
16 |  Ajuste de frequência dos workers ativos de acordo com a tabela;
17 end
18 foreach janela estabilização do
19 |  if Período de estabilização está ativo then
20 |  |  Desativar período de estabilização;
21 |  end
22 end
23 foreach janela otimização do
24 |  Ajuste dos parâmetros do previsor;
25 |  Limpeza do histórico de previsão;
26 end
```

Algoritmo 2: Ajuste a cada janela de decisão, controle, estabilização e otimização

Para contornar o problema envolvendo perda de QoS, as máquinas que estiverem ligadas terão as suas frequências ajustadas pelo fator gama. Ao longo da janela de estabilização, a frequência atual de uma máquina será multiplicada por γ. Dessa maneira, sempre que houver perdas na qualidade de serviço um aumento de desempenho será gerado em resposta. Para a situação onde ocorre um gasto adicional de energia, o estado será mantido. A escolha por um estado ineficiente à nível de consumo de energia foi feita para evitar uma possível queda QoS caso o desempenho fosse diminuído. O Algoritmo 2 mostra uma visão geral do processo descrito até então.

Capítulo 4

Técnicas de Previsão

A necessidade da utilização de uma técnica de previsão em uma atividade qualquer está ligada a existência de uma diferença de tempo entre a expectativa e o acontecimento de um determinado evento, isto é, o tempo que um determinado evento demora para ocorrer a partir de um certo instante. Quando esta diferença de tempo é significativa para um processo em questão, a execução de um planejamento ou até mesmo de uma previsão torna-se aplicável.

O termo previsão, usado várias vezes nesse trabalho, trata da utilização de uma função ou modelo matemático que, com base em dados ocorridos no passado, fornece uma avaliação para um instante futuro. Antes que informações sobre os possíveis estimadores existentes sejam mostradas, algumas informações a respeito de Séries Temporais, quesito importante para o entendimento das técnicas de previsão serão vistas na próxima seção.

4.1 Séries Temporais

O termo Séries Temporais refere-se a uma de sequência observações coletadas durante intervalos (normalmente regulares) de tempo. Como exemplo desse tipo de sequência, pode-se citar dados como consumo mensal de energia elétrica de um estado, variação diária de ações na bolsa de valores, resultados de um mapa cardíaco[1] e evolução do produto interno bruto brasileiro durante uma década.

Além de uma forma muito utilizada para exibição de estatísticas, esse tipo de série pode conter padrões que possibilitam caracterizar a natureza dos dados adiante no tempo. Esses padrões são: horizontal, tendência, sazonalidade e ciclo. Um padrão **horizontal**

[1]Um mapa cardíaco contém medidas de pressão arterial e batimentos cardíacos, por exemplo, feitas (de maneira geral) de hora em hora

dentro de uma série existe quando os seus dados sempre oscilam em torno da média ao longo do tempo. A **tendência** é um padrão que confirma um acréscimo ou decréscimo dos dados ao longo do tempo. A Figura 4.1 mostra o ganho de capital bruto[2] para dois tipos de ações no Reino Unido, na qual as ações de impressão em papel mostram um padrão horizontal (os valores oscilam perto de 5 bilhões de libras esterlinas) e as de transporte e comunicações possuem uma tenência positiva (fonte [36]).

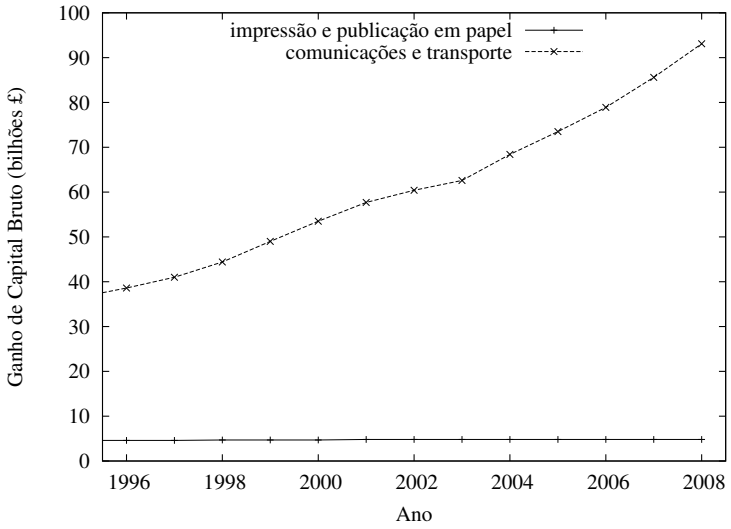
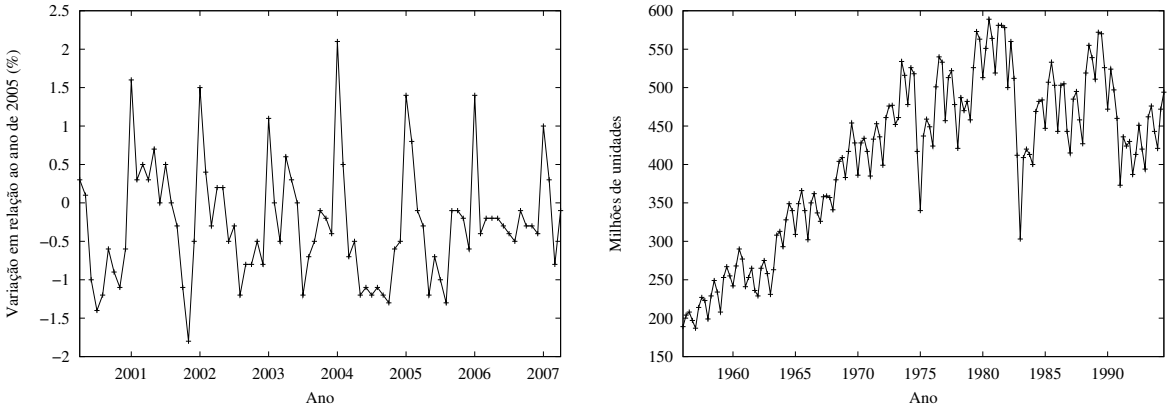


Figura 4.1: Evolução do ganho de ações no Reino Unido

O padrão de **sazonalidade** existe quando uma série é influenciada por fatores sazonais, tais como uma época do ano (estações, feriados, etc), um determinado mês ou dia da semana. Normalmente, essas influências são representadas por picos ou grandes quedas dos dados da série. Já o **ciclo** é a presença de altos e baixos nos dados dentro de um período de tempo com tamanho indeterminado. A principal diferença entre um padrão cíclico e um padrão sazonal é a sua duração. Os padrões sazonais acontecem em um intervalo regular enquanto que o intervalo de cada ciclo pode variar [45]. A Figura 4.2a mostra a variação de um dos índices de inflação no Reino Unido. A sazonalidade presente nesses dados justifica o aumento desse indicador sempre nos meses de janeiro. Já a Figura 4.2b mostra a presença de dois ciclos a partir de 1975, na produção de tijolos feitos na Austrália. Os dados das Figuras 4.2a e 4.2b podem ser encontrados respectivamente em [36] e [45]

Uma medida estatística importante ligada a series temporais é denominada autocorrelação. Ela descreve a correlação entre valores em tempos diferentes dentro de uma mesma série, isto é, quanto um determinado valor desta série influencia nos que estão adiante

[2]Ganho antes da incidência de impostos



(a) (b)

Figura 4.2: Séries temporais com sazonalidade (a) e ciclo (b)

no tempo. Um gráfico mostrando o valor desta medida para diferentes atrasos (também chamados de *time lag*) é chamado de correlograma (*correlogram*). Através da análise do correlograma é possível determinar se em uma série temporal existe sazonalidade. Como a carga submetida a um sistema de computação, em função do tempo, caracteriza uma série temporal, a aplicação dessa figura estatística [8] fornece bons indicadores de como políticas de previsão irão reagir quando submetidas a perfis reais de carga.

4.2 Previsores

Técnicas de previsão ou previsores são métodos aplicados a séries temporais que realizam uma estimativa do valor que a mesma poderá assumir em qualquer instante de tempo posterior a um determinado ponto base. Dentre as técnicas existentes para realizar previsão sobre séries temporais podem ser citadas as médias móveis, o amortecimento exponencial, regressões, redes neurais e outras. Contudo, como o foco desse trabalho não são os previsores em si, mas sim a aplicação citada no Capítulo 3.2, esta seção mostrará apenas alguns desses métodos.

4.2.1 Médias Móveis

O método de média móvel simples usa a média de todos os dados para obter a previsão [28]. Um número constante de pontos de dados pode ser especificado no início para ser calculada uma média das observações mais recentes. O termo média móvel é usado para descrever essa abordagem. À medida que uma nova observação torna-se disponível, uma nova média pode ser calculada, retirando o valor mais antigo e incluindo o novo. Essa

média móvel é então usada para prever o próximo período. A seguinte equação mostra o modelo de média móvel simples:

$$M_t = F_{t+1} = \left(\frac{Y_t + Y_{t-1} + Y_{t-2} + Y_{t-3} + ... + Y_{t-n+1}}{n} \right) \tag{4.1}$$

onde M_t mostra qual o valor da média móvel no instante t, F_{t+1} representa o valor previsto para o próximo período, Y_t é valor da série temporal no instante t e n o número de períodos da média móvel. A média móvel do período t corresponde à média aritmética das n observações mais recentes. Note que são atribuídos pesos iguais para cada observação. À medida que se torna disponível, cada novo ponto de dado é incluído na média, e o ponto de dado mais antigo é descartado. A taxa de resposta às mudanças no padrão implícito aos dados depende do número de períodos, n, incluído na média móvel.

Observe que a técnica de média móvel trabalha somente com os mais recentes n períodos dos dados conhecidos; o número de pontos de dados em cada média não muda com o passar do tempo. O modelo de média móvel funciona melhor com dados estacionários. Ele não lida muito bem com tendência ou sazonalidade, apesar de ser melhor que o método de média simples. Uma média móvel de ordem 1 tomaria a última observação, Y_t, e a utilizaria para prever o próximo período. Isto é simplesmente a abordagem de previsão *naïve* (na qual tem-se $F_{t+1} = Y_t$). Para dados trimestrais, uma média móvel de quatro trimestres gera uma média de um ano, e para dados mensais, uma média móvel de doze meses elimina os efeitos sazonais. Quanto maior a ordem da média móvel, maior o efeito amortecedor, sendo pouco absorvidas as flutuações nas séries de dados.

Uma maneira de prever séries temporais de dados que tenham uma tendência linear é usar a técnica de média móvel dupla. O método faz o que seu nome indica: um conjunto de médias móveis é calculado, e então um segundo conjunto é calculado como uma média móvel do primeiro conjunto. Depois que os valores da média móvel simples foram obtidos utilizando a Equação 4.1, o cálculo da média móvel dupla (M'_t) é realizado segundo as equação 4.2. Vale ainda lembrar que esse valor poderá ser usado como uma previsão para o próximo instante (F_{t+1}).

$$M'_t = F_{t+1} = \left(\frac{M_t + M_{t-1} + M_{t-2} + M_{t-3} + ... + M_{t-n+1}}{n} \right) \tag{4.2}$$

4.2.2 Amortecimento Exponencial Simples e Duplo

Amortecimento exponencial é um procedimento para a revisão contínua de previsões originadas a partir das experiências mais recentes. Esse método é baseado em ponderar (amortecer) valores passados de uma série de maneira decrescente (exponencial). As observações são ponderadas, com mais peso sendo dado às observações mais recentes. Os pesos usados são α para a observação mais recente, $1-\alpha$ para a segunda mais recente, $(1-\alpha)^2$ para a terceira, e assim sucessivamente.

De forma amortecida, a nova previsão para o período $t+1$ pode ser considerada como uma sendo média ponderada da observação para o período t e da antiga previsão para o período t. O peso α é dado ao último valor observado, e o peso $1-\alpha$ é dado a antiga previsão, supondo que $0 < \alpha < 1$. Assim, temos que *nova previsão* = $\alpha\times$(*última observação*) + $(1-\alpha)\times$(*antiga previsão*). De maneira mais formal:

$$F_{t+1} = \alpha \times Y_t + (1-\alpha) \times F_t$$

onde F_{t+1} é o novo valor amortecido ou de previsão para o próximo período, α constante ou coeficiente de amortização, Y_t representa o último valor da série

A técnica de amortecimento exponencial duplo, frequentemente referida como método de Brown, é usada para prever séries temporais de dados que tenham uma tendência linear. A técnica de amortecimento exponencial duplo é resumida através das Equações 4.3 a 4.7

$$L_t = \alpha Y_t + (1-\alpha) \times L_{t-1} \tag{4.3}$$

$$L'_t = \alpha L_t + (1-\alpha) \times L'_{t-1} \tag{4.4}$$

$$l_t = 2L_t - L'_t \tag{4.5}$$

$$b_t = (\alpha/(1-\alpha)) \times (L_t - L'_t) \tag{4.6}$$

$$F_{t+m} = l_t + b_t \times m \tag{4.7}$$

O fator L_t indica o valor de Y_t amortecido exponencialmente no período t e L'_t o valor de Y_t duplamente amortecido no período t. O valor amortecido de forma exponencial simples é agora calculado usando a Equação 4.3. A Equação 4.4 é usada para calcular o valor amortecido de forma exponencial dupla. Já a Equação 4.5 é usada para calcular a diferença entre os valores amortecidos exponencialmente. Representando um fator de

ajuste adicional, a Equação 4.6 é similar a uma medida de inclinação que pode mudar ao longo da série. Finalmente, a Equação 4.7 é usada para fazer a previsão de m períodos no futuro.

4.2.3 Método Linear de Holt e Winters

Uma outra técnica frequentemente usada para se trabalhar uma tendência linear é chamada método bi-paramétrico de Holt ou Método Linear de Holt. A técnica de Holt (futuramente referida pela sigla MLH) amortece o nível e a tendência diretamente usando diferentes constantes de amortecimento para cada um. Na abordagem de Brown, apenas uma constante de amortecimento era usada, e os valores de tendência estimados eram muito sensíveis a influências aleatórias. As três equações usadas nessa técnica são as seguintes:

$$L_t = \alpha Y_t + (1-\alpha)(L_{t-1} + b_{t-1}) \tag{4.8}$$

$$b_t = \beta(L_t - L_{t-1}) + (1-\beta)b_{t-1} \tag{4.9}$$

$$F_{t+m} = L_t + b_t m \tag{4.10}$$

A Equação 4.8 mostra como é calculada uma estimativa do valor da série temporal, a partir de uma amostra Y_t obtida no instante t. A Equação 4.9 calcula b_t, isto é, uma noção de tendência positiva ou negativa apresentada pelos dados. Utilizando os valores definidos nas duas equações anteriores, a Equação 4.10 calcula a previsão para o instante $t+m$, onde m representa a janela de previsão. As variáveis α e β representam os coeficientes de amortecimento. Devido a existência desses coeficientes, esse método é também classificado como um método de amortecimento duplo.

O modelo de amortecimento exponencial sazonal e linear tri-paramétrico de Winters (ou Holt-Winters), uma extensão do modelo de Holt, pode reduzir o erro da previsão. Uma equação adicional é usada para estimar a sazonalidade. Essa sazonalidade estimada é dada como um índice, sendo calculada pela Equação 4.13. Essa equação mostra que a estimativa do índice sazonal Y_t/L_t é multiplicada por γ e então somada à antiga estimativa sazonal S_{t-s} multiplicada por $1-\gamma$. A razão Y_t/L_t expressa o valor como um índice ao invés de termos absolutos, de tal forma que ela pode ser ponderada com o índice sazonal amortecido para o período $t-s$. O modelo de Holt-Winters é definido pelas seguintes equações:

$$L_t = \alpha \times (Y_t / S_{t-s}) + (1 - \alpha) \times (L_{t-1} + b_{t-1}) \tag{4.11}$$

$$b_t = \beta \times (L_t - Lt - 1) + (1 - \beta) \times b_{t-1} \tag{4.12}$$

$$S_t = \gamma \times (Y_t / L_t) + (1 - \gamma) \times S_{t-s} \tag{4.13}$$

$$F_{t+m} = (L_t + m \times B_t) \times S_{t-s+m} \tag{4.14}$$

onde a variável L_t representa o novo valor amortecido, Y_t é a última observação feita no instante t, b_t é a tendência estimada, S_t é a sazonalidade estimada, m os períodos futuros a serem previstos, s é a extensão da sazonalidade e F_{t+m} a previsão para m períodos futuros. No Método de Holt-Winters, os coeficientes de amortização para os dados, tendência e sazonalidade são representados respectivamente por α, β e γ.

A Equação 4.11 atualiza a série amortecida. Uma sutil diferença nessa equação a distingue da sua correspondente no modelo de Holt, a Equação 4.8. Na Equação 4.11, Y_t é dividido por S_{t-s}, que ajusta Y_t pela sazonalidade, removendo assim os efeitos sazonais que poderiam existir no dado original Y_t.

Depois da estimativa da sazonalidade e da tendência terem sido amortecidas nas Equações 4.12 e 4.13, uma previsão é obtida com a equação 4.14. É quase a mesma coisa do que a fórmula correspondente usada para obter a previsão no modelo de Holt 4.10. A diferença é que essa estimativa para o período futuro, $t + m$, é multiplicado por S_{t-s+m}. Esse índice sazonal é correspondente ao último valor real disponível e é usado para ajustar a previsão pela sazonalidade.

Capítulo 5

Ferramentas Utilizadas

Neste capítulo serão mostradas as ferramentas utilizadas e o suporte de hardware necessários para a aplicação prática deste trabalho. Além das características e condições de funcionamento dessas ferramentas, também será abordado o seu papel dentro do escopo global da aplicação desenvolvida.

5.1 Software

5.1.1 Httperf

5.1.1.1 Gerador de Carga

O Httperf [35] é uma ferramenta do tipo *software* livre (vide GNU GLP para maiores informações) que através do envio de requisições HTTP (simulando o lado cliente da aplicação *Web*) realiza a medição de desempenho em servidores *web*. Utilizando um controle de admissão baseado em *timeout* (tempo limite para execução de um pedido), o *httperf* determina a quantidade de requisições a serem enviadas a fim de que o servidor em questão não seja sobrecarregado. Essa ferramenta também realiza o controle de recursos locais, ou seja, aqueles que são utilizados na máquina cliente e podem tornar-se um gargalo. Durante a execução de um teste de desempenho o *httperf* limita o número de portas a serem usadas, descritores de arquivos, e o tamanho do *buffer* de rede a ser utilizado na máquina cliente.

A versão utilizada dessa ferramenta utiliza dois tipos distintos de geradores de carga. O primeiro tipo é o **gerador de carga através de URLs**, o qual em uma versão envia requisições baseando-se somente em uma URL, de maneira que vários acessos ao mesmo recurso sejam realizados durante toda a geração da carga. Já a versão mais aprimorada

deste gerador é capaz de produzir uma carga de trabalho a partir de uma lista de URLs contidas dentro de um arquivo texto.

O outro gerador de carga é o **gerador de requisições**. Esse segundo tipo gera carga a partir de chamadas do protocolo HTTP em tempos apropriados e, assim como o primeiro gerador, possui duas versões. A primeira versão cria **conexões** de maneira fixa e determinística. Vale notar que quando o número de conexões simultâneas é igual a 1, reproduz-se o comportamento da versão 1.0 do protocolo HTTP. A segunda versão desse gerador é responsável pela criação de **sessões**. Ao contrário da versão anterior (por **conexões**), as chamadas dentro de cada sessão HTTP pode ser separadas por tempos (*think-time*) diferentes entre si. Dessa forma, a ferramenta consegue reproduzir o comportamento de um usuário que clica em *links* em tempos distintos.

Ao final de cada teste, o *httperf* apresenta uma lista detalhada de informações (como ilustrado na Figura 5.1) a respeito do desempenho do teste realizado. Constam desse relatório informações tais como códigos de resposta HTTP das requisições (agrupados por centena), quantidade de requisições que excederam o tempo limite, quantidade de informação transferida e outros dados.

```
Total: connections 198 requests 1112 replies 1110 test-duration 39.520 s

Connection rate: 5.0 conn/s (199.6 ms/conn, <=31 concurrent connections)
Connection time [ms]: min 5113.2 avg 5207.4 max 6114.3 median 5188.5 stddev 136.7
Connection time [ms]: connect 2.5
Connection length [replies/conn]: 6.453

Request rate: 28.1 req/s (35.5 ms/req)
Request size [B]: 71.0

Reply rate [replies/s]: min 13.6 avg 27.8 max 30.8 stddev 6.3 (7 samples)
Reply time [ms]: response 10.8 transfer 22.4
Reply size [B]: header 174.0 content 29704.0 footer 2.0 (total 29880.0)
Reply status: 1xx=0 2xx=1110 3xx=0 4xx=0 5xx=0

CPU time [s]: user 4.15 system 35.37 (user 10.5% system 89.5% total 100.0%)
Net I/O: 821.5 KB/s (6.7*10^6 bps)

Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 4.35 max 5.00 stddev 1.88 (172/172)
Session: avg 1.00 connections/session
Session lifetime [s]: 5.2
Session failtime [s]: 0.0
Session length histogram:  0 0 0 0 0 0 172
```

Figura 5.1: Relatório do *httperf*

Uma das funções que o *httperf* desempenha nesse trabalho é a de determinar a quantidade máxima de requisições aceitas pelo *cluster*. Para obter tal informação, estipula-se uma quantidade inicial de maneira arbitrária. Em seguida inicia-se um teste *httperf* e realiza-se o monitoramento da utilização do processador da(s) máquina(s) alvo. Para que esta estimativa seja relevante deve-se observar que a utilização pode assumir 100% para

qualquer valor acima da carga máxima que o servidor suporta. Sendo assim, o valor ideal será aquele que fizer a utilização do(s) servidor(es) oscilar bem próximo ao valor de 100%.

Com a estimativa do valor da carga máxima feita, os primeiros testes de carga puderam ser realizados. Esses testes utilizaram um *workload* (carga de trabalho) em forma de rampa. Começando o teste a partir de uma carga inicial igual a zero, essa ferramenta começa a enviar requisições até chegar ao valor máximo configurado. Dependendo dos parâmetros ajustados, o envio de requisição pode continuar no valor máximo durante o tempo necessário até que a taxa de envio de requisições entre em declínio.

Para realizar o tipo de teste mencionado foi utilizada a linha de comando mostrada na Figura 5.2. A primeira diretiva nessa linha de comando é o parâmetro **hog**. Ele indica ao *httperf* que podem ser usadas quantas portas TCP estiverem disponíveis no cliente no qual a ferramenta está executando. O parâmetro **server** indica o nome ou endereço IP do servidor com o qual se deseja realizar o teste. As diretivas **port** e **uri** indicam, respectivamente, qual é a porta de funcionamento do servidor e qual o endereço do recurso alvo do teste.

```
httperf --hog --server=ampere --port=82 --uri=/home.php?200 --wsess=90,900,1 --rate 0.1 -v
```

Figura 5.2: Linha de comando httperf para *workload* rampa

Para determinar a quantidade de requisições enviadas ao servidor alvo, o parâmetro *wsess* (ou seja, o segundo tipo do gerador de requisições) foi utilizado. Para a execução do *httperf* nesse modo, é preciso especificar os seguintes dados: o número total de clientes virtuais do teste, a quantidade de chamadas que cada um dos clientes irá realizar durante o teste (quantas vezes cada cliente irá pedir o recurso especificado no parâmetro **uri**) e o intervalo de tempo entre as chamadas de um mesmo cliente. Com o ajuste ideal destes parâmetros, a quantidade de requisições estará em seu ápice assim que todos os clientes virtuais estiverem enviando requisições.

O parâmetro **rate** da linha de comando na Figura 5.2 representa a quantidade de clientes que irão ser iniciados a cada segundo. Este parâmetro influencia tanto na quantidade requisições enviadas ao servidor simultaneamente, quanto na duração do teste. O parâmetro **v** ativa a verbosidade do programa, isto é, o *httperf* irá imprimir antes de começar o teste informações adicionais (caso existam) dos parâmetros informados.

O exemplo a seguir mostra a utilização desses parâmetros. Dada uma carga máxima de 90 requisições, deseja-se executar um teste de 30 minutos com um *workload* em forma de rampa. Para proceder com esse teste será enviado ao servidor somente um tipo de

requisição: um *script* PHP. Um outro dado a ser estabelecido é a quantidade total de clientes virtuais iniciados pelo *httperf*. Por questões de simplificação aritmética será assumida a criação de 90 clientes virtuais durante todo o teste e que cada cliente enviará um pedido a cada segundo. Como o *workload* tem a forma de rampa, a carga máxima deverá ser alcançada na metade da duração total do teste (15min ou 900s). Utilizando esses dados, a taxa de criação de clientes será de 1 cliente a cada 10s. Na Tabela 5.1 segue um resumo dos dados para o teste desejado.

Tabela 5.1: Dados do teste *httperf* com *workload* rampa

Informação	Valor	Parâmetro *httperf*
Tempo do teste	1800 segundos	–
Carga máxima	90 requisições	–
Taxa de criação de clientes	0,1 cliente por segundo	–
Total de clientes	90 clientes	1° parâmetro *wsess*
Requisições por cliente	900 requisições	2° parâmetro *wsess*
Intervalo entre requisições de cada cliente	1 segundo	3° parâmetro *wsess*

Os resultados do teste descrito acima pode ser visto na Figura 5.3. Um detalhe que merece atenção é a variação na taxa de envio feita pelo *httperf*. Apesar desse teste estar sendo executado em um ambiente isolado, fatores tais como envio de dados em rajada (característica do protocolo HTTP) e o fato do período de criação dos clientes virtuais não ser plenamente representado por um número de 32 bits (dízimas periódicas) interferem na taxa de envio de requisições. Entretanto essa interferência não prejudica a integridade do teste, pois como pode ser visto nessa mesma figura a carga máxima nunca é ultrapassada.

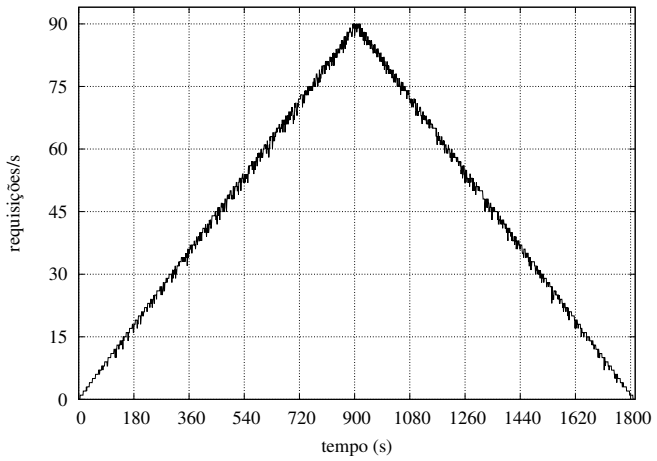


Figura 5.3: *Workload* rampa gerado a partir do exemplo numérico

5.1.1.2 Reprodução de um perfil de carga real

Para desenvolvimento desse trabalho houve a necessidade de modificar a estrutura básica dessa ferramenta para adicionar uma nova função: a reprodução de um perfil de carga enviada a servidores usados em aplicações reais. Esse perfil de carga tradicionalmente são obtidos através de aquivos de log (*traces*) feitos nesses servidores. Exemplos desse tipo de dado podem ser encontrados em [34]. Para realizar tal função foi necessária uma intervenção no código da ferramenta, pois o *httperf* em sua forma original não oferece suporte a reprodução de *traces*.

Ao invés de utilizar as distribuições disponíveis no *httperf* (determinística ou fixa, distribuição exponencial e distribuição de Poisson) a leitura de um arquivo é feita a cada segundo e a taxa desejada será retornada. No momento em que o gerador de requisições calcula o período de criação das requisições que serão enviadas ao servidor, uma chamada a essa função personalizada é feita. Para que o mesmo valor seja retornado durante o mesmo segundo, a função modificada do *httperf* somente altera o valor de requisições a serem enviadas durante um tempo pré-determinado como mostra, o Algoritmo 3.

```
1  if se não é fim de arquivo then
2  |  if se é hora de trocar o valor do período then
3  |  |  ler o valor da carga dentro do arquivo;
4  |  else
5  |  |  utilize a carga anterior;
6  |  end
7  end
8  ajuste do numero de sessões;
9  normalização da carga;
10 cálculo do período conforme a carga;
```

Algoritmo 3: Algoritmo de geração de carga com perfil real

O Algoritmo 3 leva em consideração que somente um cliente virtual será criado durante o teste. Devido ao fato do *httperf* ser uma aplicação que funciona somente com uma *thread*, essa alternativa simplificou a codificação. Como a duração do teste não é conhecida a priori, a linha 8 ajusta o valor da quantidade de pedidos a serem feitos pelo cliente virtual sempre para um número fixo e pré-determinado. Caso isso não fosse feito, esse contador de pedidos poderia chegar a zero (já que ele sempre é decrementado) antes que o final do arquivo contendo a carga a ser gerada fosse atingido.

Especificamente nessa implementação foi adotado o seguinte critério: o arquivo que o *httperf* irá consultar a cada tempo (de 1s em 1s por exemplo) irá conter somente infor-

mações sobre a carga do servidor real, isto é, não é necessário que essa carga seja redimensionada para o servidor em questão. Depois de realizado o processo de normalização (linha 9 do algoritmo de geração de carga), o cálculo do período de criação de requisições baseada na carga dimensionada para o servidor em questão é realizado e retornado para o *httperf* na linha 10 do Algoritmo 3.

Para realizar o teste da implementação feita sobre o *httperf* utilizamos como exemplo a reprodução de um trecho do *trace* da Copa do Mundo de 1998 [7]. Essa parte específica do evento corresponde ao dia 28 de junho no período de 16:53:21 até as 17:23:20 do mesmo dia. O período mencionado conta com 929837 requisições enviadas aos servidores tendo como carga máxima 674 requisições em um determinado segundo e carga mínima de 384 requisições. Na tabela 5.2 encontram-se outros dados referentes ao *trace* original, ao processo de normalização e à medição realizada na máquina servidora.

Tabela 5.2: Normalização dos dados do *trace* da Copa do Mundo

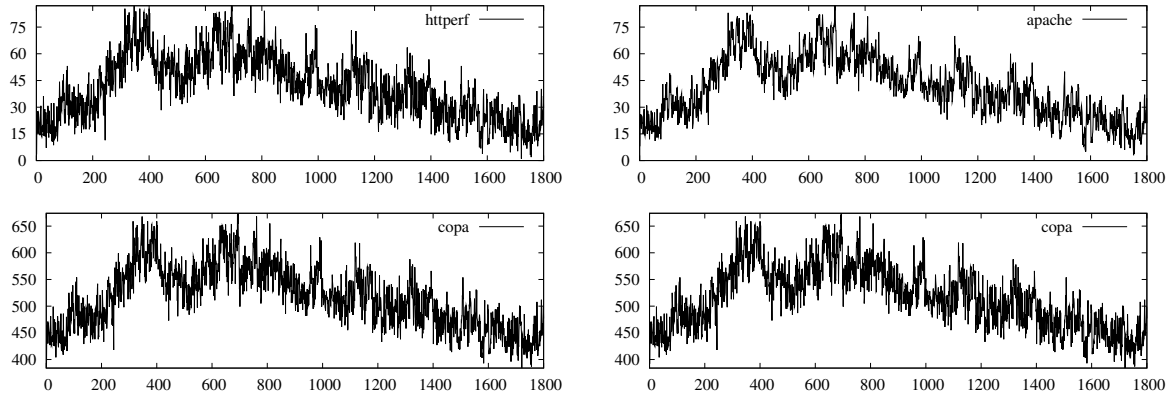
Dado	*Trace* original	*Trace* normalizado	Carga *httperf*
Total de requisições	929837	75037	74984
Carga máxima (requisições/s)	674	90	87
Carga mínima (requisições/s)	384	1	3

Na Figura 5.4a é mostrado o resultado da reprodução da carga, dimensionada apenas para uma máquina (parte superior), comparado com os dados da Copa do Mundo de 1998 (sempre na parte inferior). A Figura 5.4b mostra o mesmo perfil de dados da Copa do Mundo sendo comparado com a medição feita utilizando-se um módulo no servidor Apache (parte superior). Apesar de leves diferenças entre as curvas mostradas nessas figuras (somente percebidas em escala de segundos), o *httperf* consegue manter de forma adequada o perfil da carga original. Isto significa que tanto a máquina cliente (*httperf*) quanto o servidor (Apache) registraram valores de carga semelhantes ao longo do tempo.

5.1.2 Medição de potência

Criado e registrado pela *National Instruments*, o LabVIEW é um ambiente de programação gráfica cuja principal função é realizar instrumentação, ou seja, a medição e registro das atividades que giram em torno de um experimento. Através de componentes denominados instrumentos virtuais (*virtual instruments* ou VI) o LabVIEW proporciona vários módulos capazes de realizar a aquisição de sinais, envio de dados via rede, análises estatísticas e outras funções.

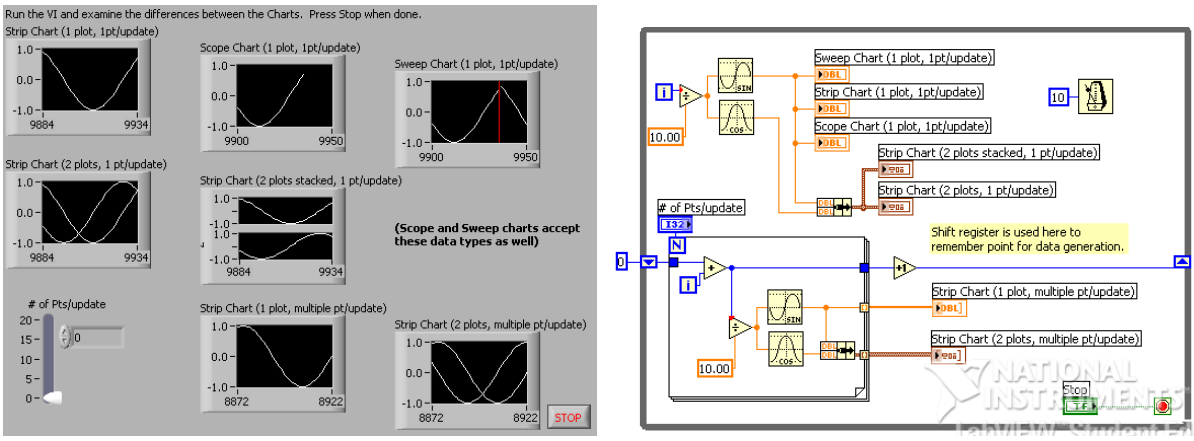
O LabVIEW basicamente está dividido em duas interfaces principais: o painel frontal



(a) (b)

Figura 5.4: Perfil de carga: (a) copa x *httperf* (b) copa x apache

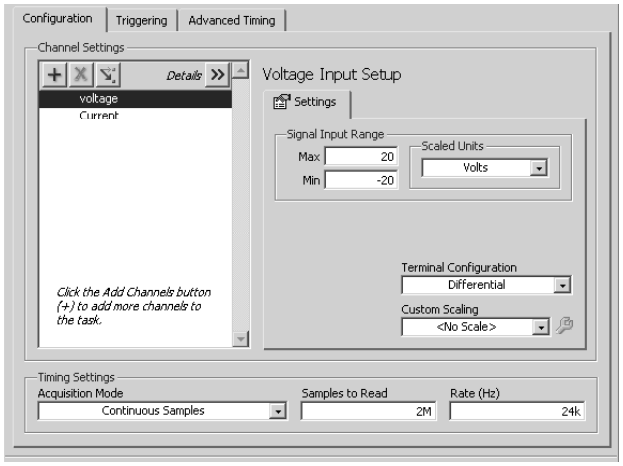
(*front panel*) e o diagrama de blocos (*block diagram*). No painel frontal são posicionados os componentes ou controles que estarão inseridos no instrumento virtual a ser construído. Para realizar acesso a funções avançadas ou a outros componentes, o diagrama de blocos deve ser utilizado. A Figura 5.5a mostra o painel frontal de um instrumento virtual enquanto que a Figura 5.5b mostra o diagrama de blocos deste mesmo VI.



(a) (b)

Figura 5.5: Interfaces para construção de VIs no LabVIEW

Para realizar a captura dos sinais de corrente e tensão foi utilizado o *software* NI-DAQmx e o hardware USB 6009 (ambos pertencentes a *National Instruments*). Nesse trabalho, o NI-DAQmx tem como função prover o *driver* para o hardware em questão, calibrar as medições de acordo com a rede elétrica do ambiente de trabalho e transmitir os sinais capturados pelo hardware ao LabVIEW. O USB 6009 é um dispositivo especializado em coletar dados (*data acquisition* ou DAQ). Ele possui 8 entradas analógicas capazes de realizar um total de 48 mil amostras por segundo. A Figura 5.6 mostra a interface do *software* e o hardware mencionados anteriormente.



(a)

(b)

Figura 5.6: NI-DAQmx (a) e USB 6009 (b)

No trabalho descrito por este documento foi construído um instrumento virtual para gerenciar uma série de funcionalidades além da captura de sinais. Essas funcionalidades são:

- Transformar os sinais de corrente e tensão em potência instantânea;
- Receber os dados do *cluster* (frequência de operação, nível de utilização, número de máquinas em funcionamento, etc);
- Monitorar a carga enviada ao *cluster*;
- Gerar registros (*logs*) para construção de gráficos;
- Exibir gráficos em tempo real.

Além das funções mencionadas acima esse VI, através da implementação de um servidor TCP/IP, também é capaz de interromper e reiniciar experimentos a partir de comandos enviados via rede. Esse recurso mostra-se útil quando deseja-se monitorar as informações e construir *logs* de experimentos em situações diferentes (i.e., variando parâmetros de entrada e funcionalidades ativas no servidor do Apache). Sendo assim, a automação de uma série de experimentos pode ser feita de maneira mais prática e eficiente.

A inserção dessa ferramenta dentro do escopo do trabalho é mostrada na Figura 5.7. O canal 1 dessa figura separa os valores da corrente e tensão obtidos diretamente da alimentação do *cluster*, para que cada um dos sinais seja enviado ao USB 6009. Usando

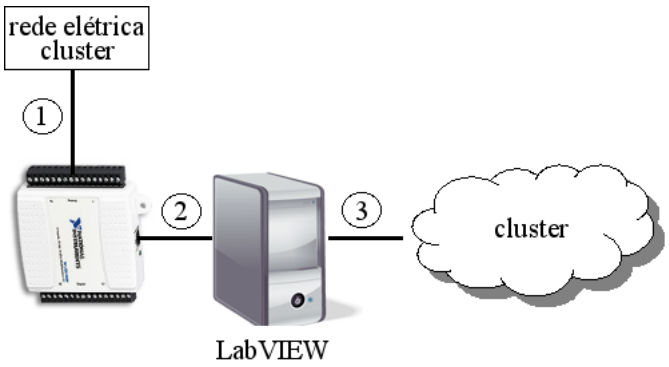


Figura 5.7: Esquema de aquisição de dados

uma conexão USB (representada por 2) esses sinais são transmitidos para a máquina na qual estão instalados o LabVIEW e o NI-DAQmx para que possam ser combinados e assim gerar o valor da potência instantânea. Já o canal 3 é uma conexão do tipo Gigabit Ethernet na qual trafegam os dados coletados do *cluster*. Esse canal de rede é diferente do utilizado na comunicação interna do *cluster* para que desta forma não ocorra interferência nos experimentos. Na Figura 5.8 tem-se parte do painel frontal do VI construído para a aplicação desenvolvida nesse trabalho.

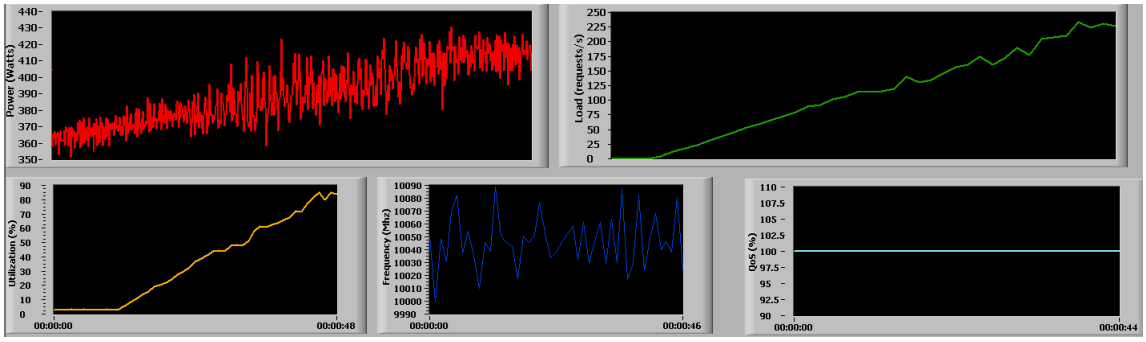


Figura 5.8: Painel frontal do instrumento virtual *power-multiconnection.vi*

5.1.3 Servidor Web Apache

Criado em sua primeira versão por Robert McCool e mantido pela *Apache software Foundation* (ASF), o Apache é um projeto colaborativo de desenvolvimento de um servidor Web de código livre e aberto [5]. Além da ASF e das centenas de desenvolvedores que contribuíram com idéias, códigos e documentação de suporte, organizações tais como a W3C e IETF possuem projetos e padrões aplicados no Apache. Durante a escrita desse documento, a versão mais atual desse servidor era a 2.2.11.

Essa ferramenta de forma nativa provê acesso a páginas estáticas (páginas HTML) utilizando o protocolo HTTP. Entretanto, as funcionalidades providas por este servidor

vão muito além. Com a adição de módulos ao componente principal do Apache, este torna-se uma ferramenta poderosa capaz de fornecer acesso a páginas dinâmicas (tais como PHP, ASP), acesso a banco de dados, autenticação, controle de acesso, protocolos de criptografia (SSL e TSL), criação e controle de versão (WebDAV) e outros recursos.

Para realizar a construção da aplicação desenvolvida nesse trabalho, os seguintes módulos foram utilizados: *mod_php*, *mod_proxy*, *mod_proxy_http*, *mod_proxy_balancer*. O *mod_php* é utilizado para prover acesso a *scripts* PHP. O *mod_proxy* tem como objetivo tornar o Apache capaz de realizar as funções de um *proxy* (redução na utilização de banda e diminuição da latência) e de se tornar um *gateway* (interligar redes e separar domínios, por exemplo). Para que as funções do *mod_proxy* estejam disponíveis para o protocolo HTTP é necessária a utilização do módulo *mod_proxy_http*.

O *mod_proxy_balancer* é responsável pela distribuição de carga. Servindo como um *gateway* entre uma aplicação e o mundo externo, e como *proxy* para cada uma das máquinas que proveem o serviço (*back-end*), o Apache (agindo como uma interface de comunicação entre a aplicação e o mundo exterior) controla o fluxo de carga às máquinas do *back-end* como mostrado na Figura 5.9. Outros dois módulos (*frontend_module* e *worker_module*) também foram utilizados, e estão descritos na Seção 6.3.

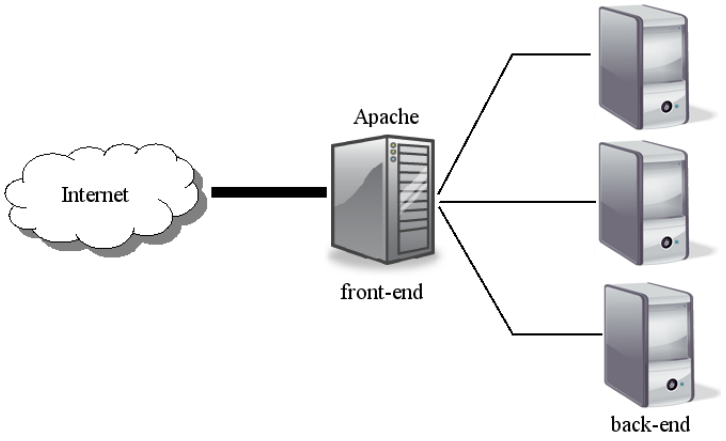


Figura 5.9: Balanceador de carga do Apache

A versão do *mod_proxy_balancer* utilizada nesse trabalho possui os seguintes algoritmos ou escalonadores para balancear a carga [6]: **contagem de requisições**, **quantidade de tráfego** ou **contagem de requisições pendentes**. Como esses métodos são bem parecidos entre si, e para um melhor entendimento, será explicado o funcionamento do método por contagem de requisições e depois disso a diferença entre eles.

A ideia por trás do balanceador de carga via **contagem de requisições** é distribuir as mesmas entre as várias máquinas (chamada de trabalhadores ou *workers*) para garantir

que cada uma obtenha a fatia de trabalho configurada [6]. Sendo assim, a cota de cada trabalhador é chamada de *lbfactor* (uma analogia para fator de balanceamento de carga ou *load balancing factor*). Como este valor é normalizado, ele representa uma parte de todo o trabalho designado a um determinado *worker*. Já a variável *lbstatus* mede quão urgentemente determinado *worker* deve receber a próxima requisição. O Algoritmo 4 mostra o funcionamento desse escalonador de carga.

```
1 foreach worker do
2 |   worker lbstatus + = worker lbfactor
3 |   total factor + = worker lbfactor
4 |   if worker lbstatus > candidate lbstatus then
5 |   |   candidate = worker
6 |   end
7 end
8 candidate lbstatus − = total factor
```

Algoritmo 4: Algoritmo de distribuição de carga por requisições

Para demonstrar o funcionamento do algoritmo acima, considere dois trabalhadores A e B com os respectivos fatores: $lbfactor_A = 70$ e $lbfactor_B = 30$. Ao receber uma requisição, o primeiro passo é adicionar o valor do *lbfstatus* de cada *worker* e a soma de todos eles. Dessa forma temos que $lbstatus_A = 70$, $lbstatus_B = 30$ e $total\_factor = 100$. Para saber qual deles será o primeiro a receber uma requisição, simplesmente escolhemos o trabalhador de maior *lbstatus*. Então A será o primeiro a receber a requisição, ou seja, ele é o *worker* candidato. Portanto $lbstatus_A$ será igual a -30 no final dessa rodada do algoritmo.

Executando mais uma rodada do algoritmo para determinar quem receberá a segunda requisição, faremos com que cada *lbstatus* receba o *lbfactor* de seu respectivo trabalhador. Logo $lbstatus_A = 40$ e $lbstatus_B = 60$. Escolhendo o de maior *lbstatus*, temos que B é o candidato dessa rodada, tendo na última linha do algoritmo seu *lbstatus* alterado para -40. Para um grande número de execuções teremos que cada um dos trabalhadores terá recebido um valor proporcional da quantidade total de trabalho a ser processado. Para os dados do exemplo mencionado anteriormente, 10 execuções já se mostram suficientes para provar a essa afirmação. Como demostrado na Tabela 5.3 o trabalhador A é escolhido como candidato em 70% das chamadas desse algoritmo e o trabalhador B 30%.

No método de contagem de requisições, a variável *lbfactor* mostra qual a quantidade de requisições que um determinado *worker* irá processar. No balanceador por quantidade de tráfego, essa variável irá definir a quantidade de *bytes* a serem transferidos. Dessa forma, outras variáveis tais como o tamanho de cada requisição serão levadas em consideração.

Tabela 5.3: Desenvolvimento do algoritmo de balanceamento por requisições

		A	B	
rodada 0	$lbstatus$	0	0	
rodada 1	$lbstatus + lbfactor$	70	30	
	candidato: $lbstatus - total\_factor$	-30	30	candidato: A
rodada 2	$lbstatus + lbfactor$	40	60	
	candidato: $lbstatus - total\_factor$	40	-40	candidato: B
rodada 3	$lbstatus + lbfactor$	110	-10	
	candidato: $lbstatus - total\_factor$	10	-10	candidato: A
rodada 4	$lbstatus + lbfactor$	80	20	
	candidato: $lbstatus - total\_factor$	-20	20	candidato: A
rodada 5	$lbstatus + lbfactor$	50	50	
	candidato: $lbstatus - total\_factor$	50	-50	candidato: B
rodada 6	$lbstatus + lbfactor$	120	-20	
	candidato: $lbstatus - total\_factor$	20	-20	candidato: A
rodada 7	$lbstatus + lbfactor$	90	10	
	candidato: $lbstatus - total\_factor$	-10	10	candidato: A
rodada 8	$lbstatus + lbfactor$	60	40	
	candidato: $lbstatus - total\_factor$	-40	40	candidato: A
rodada 9	$lbstatus + lbfactor$	30	70	
	candidato: $lbstatus - total\_factor$	30	-30	candidato: B
rodada 10	$lbstatus + lbfactor$	100	0	
	candidato: $lbstatus - total\_factor$	0	0	candidato: A

Vale ainda notar que a quantidade de requisições pode variar, desde que a quantidade de entrada e saída (E/S) feita por cada trabalhador seja a mesma. Já o algoritmo de contagem de requisições pendentes rastreia qual a quantidade de requisições em atendimento para cada *worker*. Dessa forma, a próxima requisição irá para o trabalhador que estiver menos ocupado. Esse algoritmo também pode ser chamado de balanceamento por ocupação.

5.2 Hardware

5.2.1 DVFS – *Dynamic Voltage/Frequency Scaling*

5.2.1.1 Funcionamento

Uma tecnologia existente em nível de hardware para implementar economia de energia é o Variação Dinâmica de Tensão/Frequência ou DVFS (*Dynamic Voltage/Frequency Scaling*). O DVFS é um mecanismo que através de uma alteração no valor da tensão de operação em *chips* com tecnologia CMOS (semicondutor metal-óxido complementar)

permite que uma economia no consumo de potência de ordem (aproximadamente) quadrática [40] seja obtida. O cálculo do consumo de potência total em um processador é expresso pela soma de duas parcelas: potência estática e potência dinâmica. A primeira parcela, estática, engloba todos os custos de potência relacionados a polarização dos transistores. A segunda parcela é aquela referente ao chaveamento dos sinais. A técnica de DVFS influência somente essa última parcela; a potência estática não é alterada e é uma característica do *chip*.

Esse mecanismo é desenvolvido a nível de *software*, mas possui como pré-requisitos um suporte da placa mãe como *drivers* do sistema operacional para acessar às frequências disponíveis no processador em questão. Nos sistemas operacionais derivados do GNU/Linux (ou simplesmente Linux como será chamado daqui por diante) a parte do *kernel* (ou núcleo) responsável por disponibilizar o acesso as frequências do processador é conhecida como subsistema CPUFreq [19]. Para cada geração de processadores ou fabricante, existe uma interface que precisa ser definida e disponibilizada para empregar o mecanismo de DVFS. Por exemplo, para os microprocessadores da família AMD existem as interfaces *powernow-k6* (AMD K6), *powernow-k7* (Athlon, Duron e Geode) e *powernow-k8* (Athlon 64, Opteron e Sempron). Entre as interfaces disponíveis para os processadores da família Intel, existem *speedstep-centrino* e *acpi-cpufreq*.

Para mostrar os efeitos desse mecanismo no consumo de potência em um computador, dados foram coletados a partir da execução de um programa em diferentes frequências de operação. Esse teste foi feito de maneira simplificada, uma vez que ele levou em consideração somente valores de frequências disponíveis pelo hardware utilizado.

Um fator importante na realização desse experimento é que o programa executado faça uso intenso da CPU. Dessa forma poderá ser assumido que o processador será o principal responsável pela variação no gasto de potência pelo computador. O algoritmo escolhido para a condução desse teste foi o cálculo do fatorial de um número, de maneira iterativa. O número N da função $fat(N)$ foi escolhido de forma que o resultado da função não ultrapassasse o limite de um inteiro de 64 bits e sem sinal (aproximadamente 1,8 $\times 10^{19}$). Dessa forma escolhemos o $N = 20$ cujo fatorial é aproximadamente 2,4 $\times 10^{18}$. A Tabela 5.4 apresenta os resultados obtidos executando um programa implementado na linguagem C que realiza 10^8 vezes a função de cálculo do fatorial.

Os dados que envolvem o consumo de potência do computador mostram que a diferença no gasto de potência da frequência 2,0 GHz para 1,0 GHz é de 16,81 W e quando considerados os valores de 3,0 GHz e 2,0 GHz essa diferença sobe para 47,3 W. Caso

Tabela 5.4: DVFS: consumo de um programa intensivo em CPU

Frequência (GHz)	Potência (W)
1,0	86,84
2,0	103,65
3,0	150,95

esse consumo de energia fosse linear, o valor da segunda diferença teria que ser igual à primeira. Portanto esses números conseguem mostrar uma economia aproximadamente quadrada do mecanismo de DVFS.

5.2.1.2 Utilização de Frequências Contínuas

Processadores reais só implementam um número fixo (e pequeno) de frequências (chamaremos frequências discretas). Visando um aumento na quantidade de frequências de operação disponíveis para uma determinada máquina, para melhor operar um mecanismo de economia de energia, valores contínuos para tal grandeza podem ser gerados através do chaveamento entre duas frequências discretas fornecidas pelo hardware em questão [31]. Esse processo é feito da seguinte forma:

$$F_{cont} = \alpha \times F_{baixa} + (1 - \alpha) \times F_{alta} \tag{5.1}$$

$$F_{cont} \in [F_{min} : F_{max}] \tag{5.2}$$

$$\alpha \in [0 : 1] \tag{5.3}$$

Na simulação de uma frequência contínua F_{cont} (Equação 5.1), as variáveis F_{baixa} e F_{alta} representam as frequências discretas imediatamente mais baixa e mais alta disponíveis no processador em questão, no entorno da frequência contínua em que se deseja operar. O fator α representa a influência que F_{baixa} e F_{alta} têm na geração da frequência contínua desejada. A Equação 5.2 limita os possíveis valores para frequência contínua aos limites de frequência mínima e máxima disponibilizadas pelo hardware. Já a Equação 5.3 estabelece limites para o fator α. A Tabela 5.5 mostra alguns exemplos numéricos para geração de frequências contínuas a partir dos valores discretos 1000 MHz, 2000MHz e 3000 MHz.

O efeito real na utilização de frequências contínuas no mecanismo de DVFS é que, com mais frequências disponíveis para operar, haverá um refinamento na economia de energia, já que a frequência ideal de operação para uma determinada aplicação poderá ser selecionada com maior precisão. Para implementação da técnica de DFVS chaveada ou

Tabela 5.5: Utilização de frequências contínuas

Frequência contínua (MHz)	α	F_{baixa} (MHz)	$(1-\alpha)$	F_{alta} (MHz)
1438	0,562	1000	0,438	2000
1726	0,274	1000	0,726	2000
2381	0,619	2000	0,381	3000
2859	0,141	2000	0,859	3000

com frequências contínuas, foi utilizado um *daemon* capaz de acessar a interface provida pelo *driver cpufreq* [18].

Com a utilização de um modo no qual a frequência atual do processador é definida pelo usuário (*userspace governor*), esse *daemon* promove o chaveamento entre duas frequências discretas presentes no processador. O tempo escolhido entre esse chaveamento foi de 100ms. Tipicamente, processadores reais gastam dezenas de microssegundos para chavear frequências. Assim, com o período de chaveamento escolhido, as mudanças de frequência serão asseguradas.

5.2.2 Estados de energia: configuração ACPI

Através da padronização de mecanismos de hardware promovida pela ACPI, vários estados de consumo de energia podem ser mantidos em um computador, através do controle do sistema operacional (SO). Ao invés de simplesmente estar ligado com todos os seus componentes ativos, o computador pode assumir outros estados nos quais estão disponíveis somente alguns componentes. A Tabela 5.6 mostra os possíveis estados de gerenciamento de energia contidos na especificação 3.0b da ACPI [3], seguidos de uma breve descrição.

Tabela 5.6: Estados de energia - ACPI 3.0b

Estado	Descrição
S0	Sistema ligado
S1	*Sleep*, processador desligado e memória RAM ligada e sendo atualizada
S2	Similar a S1, porém energia da CPU é desligada
S3	*Suspend to RAM* ou *Standby*. Todos os componentes desligados, exceto RAM
S4	*Suspend to disk* ou hibernação. Contexto armazenado no disco.
S5	Sistema desligado

A principal diferença entre esses estados é que quanto maior o número que acompanha o nome do estado, menor é o gasto de energia. Portanto, em períodos de inatividade, é possível que a rotina de gerenciamento de energia coloque o computador em um estado de menor consumo de energia. Entretanto, quanto maior a economia de energia obtida

por colocar a máquina em um estado de baixo consumo de energia, maior será o tempo que o computador em questão levará para reassumir o estado ligado.

Tecnologias capazes de trazer o computador de um estado de menor consumo de energia (tradicionalmente S3 ou S5) para o funcionamento normal tiveram oportunidades de surgir. Para que isso aconteça, uma transição chamada *wake-up event* ou evento de acordar deve ocorrer. Além de teclado e mouse, dispositivos tais como placa de rede *Ethernet* e *modems dial-up* tornaram-se capazes de realizar *wake-up events* dando origem ao *Wake-On-Lan* e *Wake-On-Ring* respectivamente.

Capítulo 6

Implementação

Este capítulo irá descrever os detalhes do ambiente de testes construído em torno do modelo proposto para a aplicação da previsão de carga em aglomerados de servidores *web*.

6.1 Previsor: MLH

Como o foco deste trabalho é examinar o impacto gerado pela aplicação de uma técnica de previsão, o previsor escolhido dentre aqueles citados na Seção 4.2 foi o Método Linear de Holt [45] (MLH ou *Holt Linear Method*). A adoção do previsor MLH foi feita baseada nas razões que se seguem. A primeira razão para a escolha desse método foi sua simplicidade de implementação. Com esse processo facilitado, estudos e análises feitas fora do *cluster web* puderam ser realizados. Esses estudos concretizaram a perspectiva de que seria possível realizar previsão de carga empregando esse método.

Outro fato revelado por esses estudos foi a necessidade de adicionar um mecanismo para refinamento dessa previsão. Além desse fato, o método também precisa de um pequeno histórico para realizar previsões o que torna a sua complexidade computacional bem baixa. Mais uma vez a simplicidade da técnica facilitou a escolha do método otimização e sua implantação no *front-end*. Adicionalmente, MLH é indicado na literatura como um previsor adequado [28, 45] para situações nas quais existe presença de tendência e ausência de sazonalidade, situação típica dos *traces* utilizados em estudos, como o dessa dissertação.

6.2 Otimização do previsor: biblioteca LMFIT

Com o intuito de melhorar os resultados ao longo da execução dos experimentos, os coeficientes de amortização do MLH passaram a funcionar de maneira adaptativa. Essa estratégia foi adotada para que o método de previsão aumentasse sua precisão, alterando esses coeficientes ao longo do tempo. Assumindo considerações mínimas sobre a carga (como somente presença de tendência), essa estratégia tem como objetivo eliminar a influência de dados que ocorreram em um passado muito distante.

O critério empregado para modificação das variáveis α e β foi o ajuste de curva (*curve fitting*) usando os pontos contidos em uma janela de dados recentes (**histórico de otimização**). Esse ajuste leva em consideração a minimização do MSE (*Mean Square Error* ou erro médio quadrático) entre a curva que representa os dados contidos nessa janela e a saída do previsor para esses mesmos dados. Após realizado esse procedimento, novos valores dos coeficientes de amortização são obtidos. Em nível de implementação, a biblioteca LMFIT [49] foi empregada na geração dos novos valores de α e β.

Uma preocupação que surge ao adicionar um recurso desse tipo é saber o impacto no ambiente de implementação. Para avaliar o peso computacional gerado pela utilização da biblioteca LMFIT um teste realizado sobre os dados do *trace* da Copa do Mundo de 98 foi feito. Agrupando 600 pontos de um conjunto de 45 mil, foram geradas 75 chamadas a biblioteca. Para cada uma destas chamadas foi medido um tempo de execução. A fim de analisar as possíveis variações do tempo de execução esse teste foi repetido por 870 vezes, produzindo um total de 65250 chamadas. A Tabela 6.1 apresenta os resultados obtidos em uma máquina AMD Athlon 64 3200+ utilizando 2GB de memória principal. Para a obtenção dos tempos mostrados nesta tabela foi atribuído ao método um número de chamadas máximo (300) como critério de convergência. Com base nesses resultados, pode ser verificado que esse mecanismo de ajuste para os coeficientes de amortecimento pode ser realizado durante o tempo de execução do sistema.

Tabela 6.1: Tempo de execução da biblioteca LMFIT

Medida	Tempo (μs)
Tempo Médio	10211,46
Desvio padrão	5006,13
Melhor tempo	2848
Pior tempo	28542

6.3 Ambiente de Testes

Na arquitetura utilizada para a construção do ambiente de testes deste trabalho existem três categorias de máquinas: o *front-end* (FE), os trabalhadores (*workers*) e as máquinas de suporte. A máquina *front-end* é aquela responsável por executar os procedimentos contidos no modelo, (como o gerenciamento das máquinas trabalhadoras, por exemplo), balanceamento de carga e comunicação com as máquinas de suporte. Como estamos interessados em avaliar um esquema de relacionamento entre QoS e utilização do *cluster*, um sistema de duas camadas é aqui considerado.

Os *workers* são responsáveis pelo processamento da carga enviada ao *cluster*. No caso do ambiente adotado nesse trabalho o processamento de requisições *web*. Já as máquinas de suporte são aquelas que proveem funções para que tanto o FE quanto os *workers* possam realizar suas funções e monitoramento do ambiente. Na Figura 6.1 elas representam o gerador de carga e o medidor de potência.

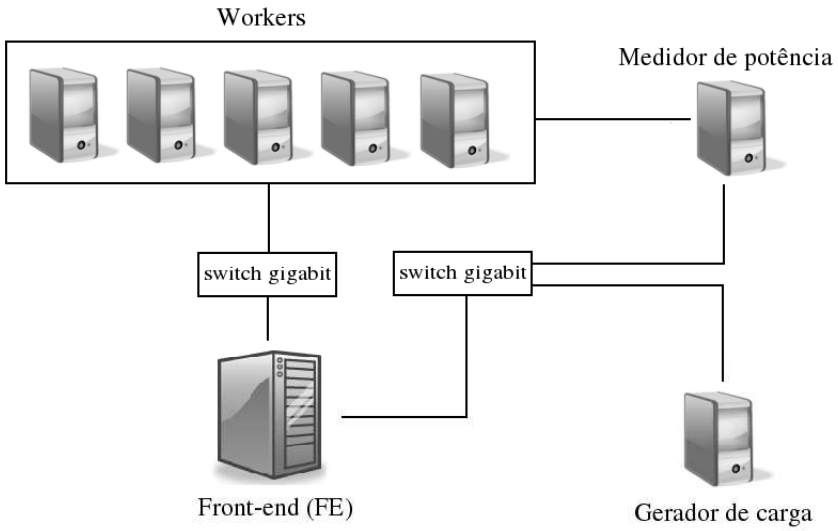


Figura 6.1: Organização e comunicação das máquinas do *cluster*

O FE e todos os *workers* executam GNU/Linux (referido como Linux daqui por diante) e servidores Apache que são responsáveis pelo processamento dos pedidos feitos pelo gerador de carga. O que difere o FE de cada trabalhador são os módulos ativados em cada um dos servidores. Nos *workers* estão ativados os módulos de processamento de requisições PHP e o *worker_module*. A necessidade da utilização deste último módulo é devido a informações que estão contidas somente no escopo da aplicação *web* (como número de requisições pendentes em um determinado *worker*, por exemplo).

O *front-end* por sua vez possui o *frontend_module*, o qual é responsável por imple-

mentar todas as funções do FE, exceto pelo balanceamento de carga (feito pelo módulo *mod_proxy_balancer*). Além do módulo mencionado por último, também estão habilitados os outros módulos (citados na Seção 5.1.3) responsáveis pelo balanceamento de carga da aplicação. Na máquina usada como *front-end* existe um servidor de NFS que é responsável por armazenar os códigos-fonte dos módulos e *shell scripts*. Desta forma, o conteúdo torna-se acessível por todas as máquinas.

O lado cliente da aplicação é representado pelo gerador de carga usando a versão do *httperf* que executa o Algoritmo 3 mostrado na Seção 5.1.1.2. Para que o programa fosse executado da maneira mais rápida possível, o arquivo que contém o número de requisições a serem enviadas por segundo é gravado em um sistema de disco do tipo RAMFS. O medidor de carga é a única máquina que executa o sistema operacional Microsoft Windows XP, pois o a distribuição do programa LabVIEW executa nesse SO. Mas mesmo assim, foi instalado nessa máquina um servidor OpenSSH e um VNC para que a comunicação com o ambiente Linux ocorresse, respectivamente, via console ou *desktop* remoto. Outras características a respeito das máquinas empregadas nessa arquitetura podem ser vistas na Tabela 6.2.

Tabela 6.2: Máquinas utilizadas na arquitetura e suas funções

Nome	Função	Frequências	Arquitetura
Watt	FE	–	AMD Athlon 64 3200+
Ampere	*worker*	3	AMD Athlon 64 X2 3800+
Coulomb	*worker*	5	AMD Athlon 64 3800+
Hertz	*worker*	5	AMD Athlon 64 3800+
Joule	*worker*	4	AMD Athlon 64 3500+
Ohm	*worker*	6	AMD Athlon 64 X2 5000+
Camburi	gerador de carga	–	Pentium 4 2.80GHz
Putiri	medidor de potência	–	Pentium 4 2.80GHz

No processo de ligar e desligar uma máquina, o tempo de *boot* (ou iniciação) é um fator que deve ser levado em consideração. A utilização do mecanismo de *Suspend-to-RAM* (em conjunto com *Wake-on-LAN*), em contraposição ao uso de *Suspend-to-disk* [3], faz esse tempo de *boot* diminuir cerca de 8 vezes (de 56s para 7,5s) de acordo com experimentos realizados em nosso ambiente de testes.

Para promover a configuração dinâmica do *cluster*, a quantidade de máquinas que permanecerão ligadas é determinada pelo nível de utilização do *cluster*. Ou seja, de acordo com o nível de desempenho exigido pela aplicação, computadores podem ser colocados em estado de hibernação em RAM (estado S3), ou (re)ligados através do mecanismo de *Wake-On-Lan*. O diagrama descrito na Figura 6.2 demonstra como ocorrem as transições

descritas acima. Os estados BOOT e SHUTDOWN representam passos intermediários assumidos para que haja algum controle antes da máquina estar respectivamente ativa e inativa, ou seja, representam o estado de "sendo ligado" e "sendo desligado".

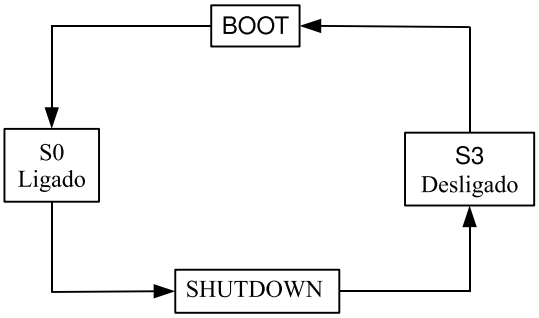


Figura 6.2: Máquina de estados dos *workers*

Outro fato que deve ser levado em consideração nesse processo é a variação do tempo de *boot*. Para que uma máquina não receba carga enquanto ela não estiver iniciada por completo, o FE não irá enviar carga para o *worker* que está sendo ligado até que esse faça confirmação (*ack* ou *acknowledment*) de que está completamente preparado para operação. Isso que evita perdas de requisições e problemas de balanceamento de carga.

O ajuste dos pesos do balanceador de carga é feito de acordo com a situação atual das máquinas no *web cluster*. Para realizar esse processo, empregou-se o seguinte estratégia:

$$lbfactor_i = \lceil \frac{freq_i}{fmax_{cluster}} \times 100 \rceil \tag{6.1}$$

onde $lbfactor_i$ representa o peso do *worker* i, $freq_i$ indica a frequência escolhida pela política preditiva e $fmax_{cluster}$ é a frequência máxima entre todos os *workers* pertencentes ao *cluster web*. O uso da Equação 6.1 permite que para qualquer frequência utilizada o peso do balanceador de carga tenha valores maiores ou iguais a 1 e menores ou iguais a 100, respeitando dessa forma uma restrição imposta pelo *mod_proxy_balancer*. Na infraestrutura adotada nesse trabalho a constante $fmax_{cluster}$ possui o valor de 5200MHz, que representa o maior valor obtido através da multiplicação da maior frequência de uma máquina pelo seu número de núcleos. Para uma máquina com dois núcleos operando a 1500 MHz o peso do balanceador de carga seria 58 e para uma outra máquina com um núcleo operando a 2100 MHz o *lbfactor* seria de 41.

A quantidade de núcleos do processador de um *worker*, assim como a sua arquitetura, tem influência direta no desempenho mostrado por ele e na quantidade de potência

consumida. Apesar dos computadores usados na implementação serem todos da família AMD, eles possuem um conjunto diferente de frequências de operação. Essa última afirmativa comprova que o conjunto de nós responsáveis pelo atendimento de requisições no aglomerado pode ser classificado como heterogêneo. A Tabela 6.3 reúne dados adicionais dos *workers*.

Tabela 6.3: Frequências, desempenho e gasto de potência dos *workers*

Nome	Frequência (MHz)	Potência* (ocioso)	Potência* (ocupado)	Requisições por segundo
ampere	1000	66,3	81,5	99,8
	1800	70,5	101,8	179,6
	2000	72,7	109,8	199,6
coulomb	1000	67,4	75,2	53,8
	1800	70,9	89,0	95,4
	2000	72,4	94,5	104,8
	2200	73,8	100,9	113,6
	2400	75,2	107,7	122,3
hertz	1000	63,9	71,6	53,6
	1800	67,2	85,5	92,9
	2000	68,7	90,7	103,4
	2200	69,9	96,5	112,4
	2400	71,6	103,2	122,8
joule	1000	66,6	74,7	51,2
	1800	73,8	95,7	91,2
	2000	76,9	103,1	101,4
	2200	80,0	110,6	111,4
ohm	1000	65,8	82,5	99,4
	1800	68,5	99,2	177,4
	2000	70,6	107,3	197,2
	2200	72,3	116,6	218,0
	2400	74,3	127,2	234,6
	2600	76,9	140,1	255,2
*Medida em watts				

A potência ociosa é o valor medido pelo *LabVIEW* quando o utilitário *vmstat*, presente no ambiente Linux, mostrava 100% de ociosidade. Para medir a potência ocupada, foi gerada uma carga através do programa *httperf* (representada pela coluna requisições por segundo) suficiente para colocar a máquina próxima de 100% de atividade e com um menor número de processos na fila de processos prontos do sistema operacional. Esse procedimento foi adotado para evitar uma medição com uma máquina em estado de sobrecarga. Os mesmos dados contidos na Tabela 6.3 foram utilizados para processar o problema de minimização da potência gasta pelo *cluster web* descrito na Seção 3.2.

Capítulo 7

Resultados

Este capítulo faz uma análise quantitativa e qualitativa do modelo proposto nesse trabalho através de comparações feitas entre a política preditiva desenvolvida aqui, sua versão não preditiva e políticas existentes no sistema operacional Linux. A Seção 7.1 mostra a reprodução de um perfil de carga real, o que permite ressaltar os ganhos energéticos e questões voltadas a manutenção da qualidade de serviço provida pelo sistema. A Seção 7.2 deste capítulo, além da comparação com as mesmas políticas da seção anterior, faz uma análise de sensibilidade de alguns parâmetros chaves do modelo: o fator gama e a variável $util_{max}$.

7.1 Comparação entre Políticas de Gerenciamento de Energia

Esta seção do trabalho apresenta alguns resultados para a política de economia de energia confrontada com três outras políticas: (i) Modo[1] *Performance* (MP) [18] (padrão do sistema Linux que possui todas as máquinas ligadas em frequência máxima); (ii) Modo *Ondemand* (MO) [37], o qual representa um gerenciador de energia que altera a frequência e tensão do processador baseado na carga do sistema (disponível em sistemas Linux com *kernel* 2.6 ou superior); e, (iii) uma política de *On-Off* Otimizado (OOO) que possui as mesmas características do MLH exceto pela presença do previsor, isto é, possui o mesmo esquema de configuração dinâmica e DVFS que a política preditiva.

Todos os resultados são para o aglomerado utilizando os cinco trabalhadores e o *front-end* mostrados na Figura 6.1 e Tabela 6.2 (ambos descritos na Seção 6.3). O tipo de *trace* utilizado para realizar a avaliação contida nesta seção baseia-se na reprodução do perfil

[1] Tradução adotada para a palavra *governor*

da Copa do Mundo de Futebol de 1998 [34]. Utilizando o período compreendido entre 11h30min do dia 29 junho até 00h do dia 30 do mesmo mês, o programa *httperf* foi utilizado para reproduzir o mesmo padrão contido na Figura 7.1. Uma normalização da taxa de envio de requisições existentes no *trace* original foi feita a fim de ajustar os valores à capacidade máxima do *cluster*.

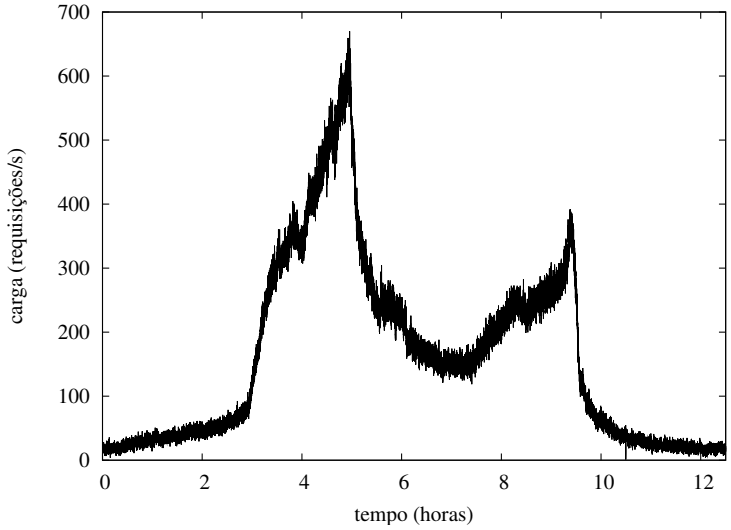


Figura 7.1: Perfil de carga da copa do mundo (12h)

Alguns dos parâmetros utilizados para a execução dos experimentos conduzidos nesta seção estão indicados da Tabela 7.1. Todos eles já foram definidos em seções anteriores (3.2 e 4.2), exceto pelo intervalo de otimização. Este parâmetro representa o tempo existente entre as sucessivas chamadas do otimizador LMFIT, responsável por realizar de maneira adaptativa o ajuste dos coeficientes de amortização do Método Linear de Holt. Para a definição da janela de decisão, valores entre 5s e 20s foram utilizados. Devido a uma maior resposta do sistema, os melhores níveis de QoS e menor interferência na geração de carga da ferramenta *httperf* foram obtidos usando o menor deles.

Tabela 7.1: Parâmetros dos experimentos com duração de 12h

Parâmetro	**Valor**
carga máxima do *cluster*	670 requisições/s
prazo das requisições	4 s
γ_max	1,4
$util_{max}$	0,8
janela de decisão	5 s
janela de controle	1 s
janela de estabilização	2 min
intervalo de otimização	10 min
histórico de otimização	600 amostras

A primeira comparação de resultados é feita confrontando o MLH contra o MP e MO. Esse duelo mostra o poder das técnicas voltadas para redução de energia perante mecanismos padrões. O total de energia gasta nesses experimentos foi aproximadamente 5.267 Wh para o Modo *Performance*, 4.730 para o Modo *Ondemand* e 2.168 para a política que emprega o Método Linear de Holt. Esses dados mostram uma redução de energia de 3.099 Wh (do MP para o MLH), ou aproximadamente 59%; a redução obtida comparando MP com MLH foi de 2.562 Wh, ou 54%.

Comparando-se a política MLH contra OOO, os resultados indicaram uma economia de energia por volta de 1,5% (indicando um consumo aproximado de 2200 Wh para o último método). Essa diferença não apresentou a mesma grandeza da comparação feita contra as outras políticas devido ao fato de que o OOO é um mecanismo de economia de energia que representa uma solução otimizada. A Figura 7.2 mostra a quantidade de potência consumida pelo aglomerado em função do tempo de experimento. Devido a situação de carga máxima, o valor de pico é o mesmo para todas as políticas. A curva de potência para o método OOO não foi mostrada nessa figura devido a sua grande semelhança com a do método MLH na escala utilizada. Como indicado por [10] e outros trabalhos publicados, o efeito de desligar e ligar máquinas provoca o ganho de maior ordem na economia de energia.

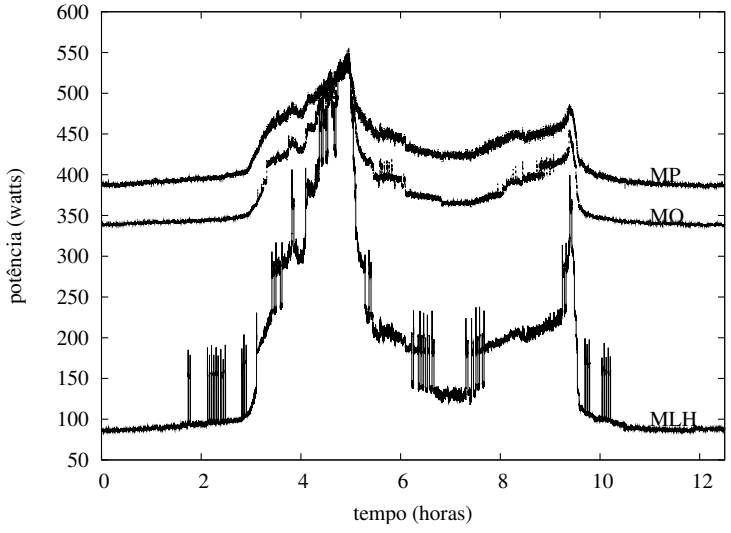


Figura 7.2: Gasto de potência: MLH, MO, MP

Além da redução do consumo de energia obtido pelo MLH, a política de energia modelada nesse trabalho mostra resultados superiores que o OOO[2] em situações nas quais a manutenção da qualidade de serviço provida pelo sistema deve ficar acima de

[2]Tanto o MP quanto o MO não apresentaram nenhuma perda de QoS, ao custo de gastarem mais energia.

um determinado alvo (95% para estes experimentos). Deve-se notar que esse valor faria parte de um contrato entre cliente e provedor do serviço (SLA - *Service Level Agreement*), onde normalmente uma QoS menor do que 100% é acertada. A Figura 7.3 apresenta os níveis de QoS demonstrados pelo MLH e OOO durante toda a duração do experimento. No início do experimento (perto da segunda hora) e no fim (perto da décima hora), as grandes perdas mostradas pelo OOO são explicadas por uma reação excessiva a rajadas de cargas.

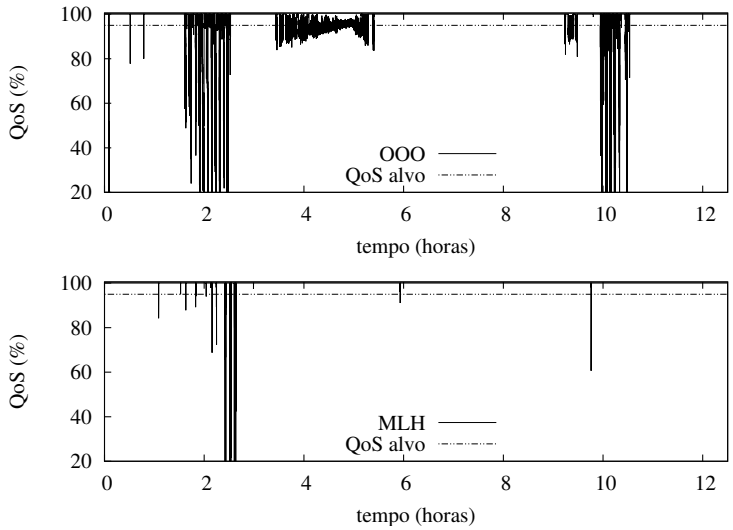


Figura 7.3: Níveis de QoS: MLH (baixo) *vs.* OOO (cima)

Isto significa que existem mudanças significativas de configuração (seja via DVFS ou configuração dinâmica) em um curto espaço de tempo, conforme observações registradas nos *logs* do sistema, e em uma escala de tempo muito menor do que a mostrada na Figura 7.3). Nota-se que o método preditivo se comporta muito melhor nessa situação.

Perto da quarta hora, quando a carga está próxima do valor máximo, uma situação de sobrecarga ocorre: requisições são acumuladas pelos computadores *workers* para processamento tardio. Este enfileiramento de requisições atrasadas, juntamente com a flutuação da carga, causa várias mudanças de configuração no esquema OOO. O MLH, como descrito previamente, foi capaz de lidar com esta situação de uma maneira mais suavizada, ou seja, com menos perdas de QoS. É oportuno lembrar que perdas de QoS são permitidas (e inevitáveis) em sistemas *soft real-time*. É importante ressaltar que no esquema MP não houve perda de qualidade de serviço devido ao super dimensionamento do desempenho do sistema, acarretando um consumo de energia duas vezes maior que o MLH.

A política MLH apresentou 30 ocorrências de perda de qualidade de serviço durante

todo o experimento. Levando em consideração a enorme quantidade de requisições (perto de 7,28 milhões) durante as 12 horas do experimento, este número torna-se inexpressivo. Entretanto, a concentração das perdas de QoS em alguns instantes particulares (perto das horas 2 e 10) valem ser comentadas. Esses momentos são, na maioria das vezes, aqueles em que a configuração em vigor possui somente um servidor ativo que pode não ser capaz de lidar com uma eventual rajada de requisições presente no perfil de carga. O MLH é capaz de filtrar algumas das rajadas, evitando configurações indesejáveis (o que não acontece com o esquema OOO).

A Figura 7.4 demonstra essa situação. Nesta figura, o eixo y representa a configuração atual do *cluster* através da primeira letra do nome de cada máquina *worker* em operação (por exemplo, O indica que somente o servidor *ohm* está ligado; O+A representa uma configuração na qual tanto a máquina *ampere* quanto a *ohm* estão ligadas). A variação presente perto da hora 2 mostra trocas de configuração efetuadas em função de uma flutuação mais acentuada da carga. A análise dos registros obtidos em cada uma das execuções mostram que o OOO realiza um maior número de trocas que o esquema MLH.

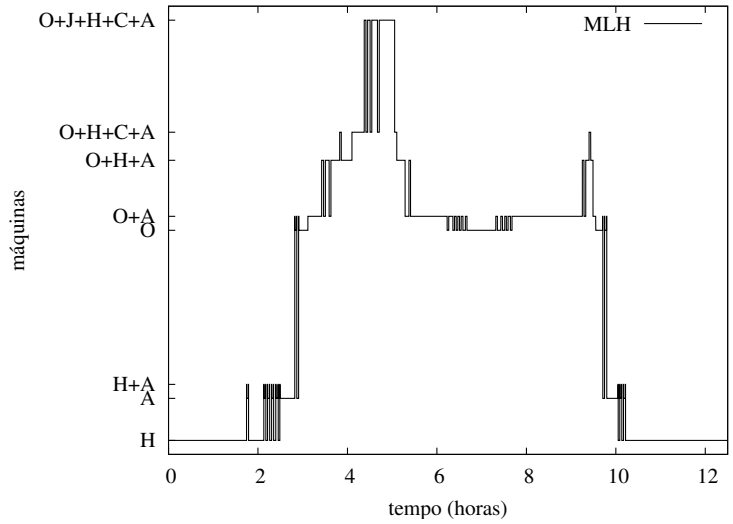


Figura 7.4: MLH: trocas de configuração. No eixo y: inicias do nome de cada *worker*

O esquema OOO sofre um maior número de trocas que a política MLH pela sua falta de capacidade em analisar a tendência da carga. Examinando o cenário contido na Figura 7.5[3] tem-se que logo após o instante de 143,6 min o método MLH reage ao aumento de carga ativando a máquina *ampere* para substituir *hertz*. Logo após essa transição (um pouco antes do minuto 144), o esquema OOO realiza a troca inversa devido a influência de valores imediatos do *workload*. Como a carga é crescente nessa parte do perfil de carga,

[3]A escala dessa figura foi aumentada para minutos para melhorar a legibilidade

a política OOO é penalizada com excessivas perdas de QoS a partir do minuto 145, sendo algumas delas bem severas. Já o método preditivo opta por um estado de maior custo de energia, privilegiando dessa forma uma maior qualidade de serviço.

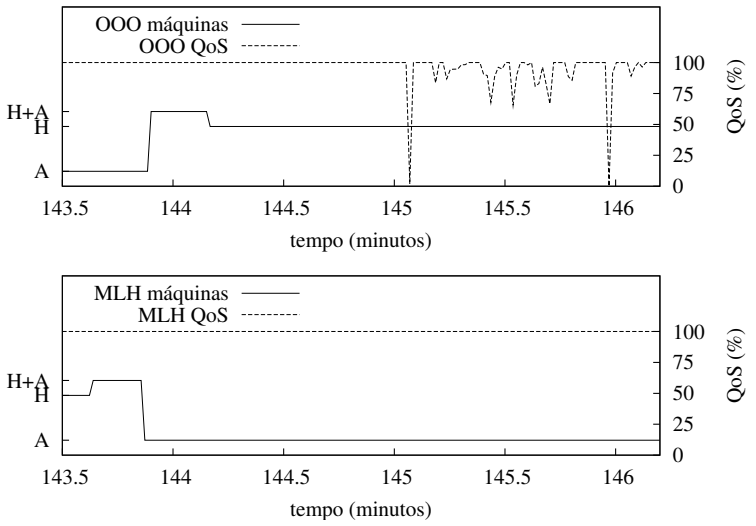


Figura 7.5: Troca de máquinas: reativa (cima) x preditiva (baixo)

As curvas de potência mostradas na Figura 7.2, quando relacionadas com as configurações na Figura 7.4, claramente mostram o impacto das configurações dinâmicas feitas no sistema quando existem mudanças na carga. Assim que uma máquina precisa ser ligada, um pico no valor da potência aparece na curva. Para melhor entender os benefícios do método MLH, foi computada a energia gasta por cada política durante os momentos críticos, quando a QoS está sendo afetada, e isso está indicado na Tabela 7.2.

Tabela 7.2: Energia consumida durante picos de carga (Wh)

	Período (horas)		
Política	**1.7-2.4**	**3.6-5.5**	**10-10.5**
MP	277	937	198
MO	241	885	173
OOO	74	728	53
MLH	70	716	51
MLH *vs.* MP (%)	74.7	24.1	74.2
MLH *vs.* MO (%)	71.0	19.1	70.5
MLH *vs.* OOO (%)	5.4	1.6	3.8

Por fim, a Figura 7.6 mostra a frequência agregada do cluster para as políticas MO e MLH. A frequência agregada é a soma das frequências de todos os servidores que estão ligados. Para máquinas que possuem mais de um núcleo, a multiplicação do número

de núcleos pela sua frequência atual foi usada como parcela dessa soma. Dessa forma, o desempenho de máquinas com mais de um núcleo (*multicores*) pode ser diferenciado daquelas que possuem somente um quando estão na mesma frequência. Vê-se claramente nessa figura os benefícios da técnica aqui proposta quando comparada ao *governor* Linux disponível para aplicações que requeiram economia de energia.

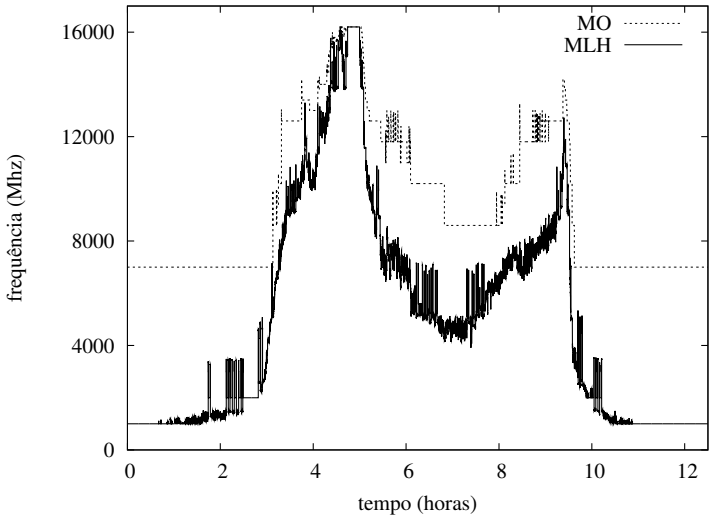


Figura 7.6: MLH *vs* MO: frequência do *cluster*

7.2 Experimentos com Análise de Sensibilidade

Os experimentos conduzidos para realizar a análise de sensibilidade de parâmetros também utiliza o perfil de carga da Copa do Mundo de 1998 indicado na Seção anterior. Com o intuito de aumentar o número de experimentos realizados e diminuir a probabilidade de interferências externas (e.g., falha no fornecimento de energia pela concessionária local), a taxa de reprodução desse *trace* foi acelerada em 3 vezes. Isto implica que a duração de cada um dos experimentos foi reduzida de 12 horas e meia para 4 horas e 10 minutos (ou 250 min).

A forma da carga submetida ao sistema pode ser observada na Figura 7.7. Adicionalmente, aqui também foi considerado um único tipo de requisição, com prazo de execução limitado a 4s, de forma a tornar o atendimento à demanda realizável no pico da carga. Esse valor é típico das requisições definidas no *benchmark* TPC-W [47]. Uma outra observação a ser feita é que a curva contida no gráfico da carga de trabalho acelerada tem a mesma forma da figura reproduzindo o *trace* sem nenhuma aceleração (Figura 7.1). Esta constatação mostra que a aceleração do *workload* não mudou o seu perfil.

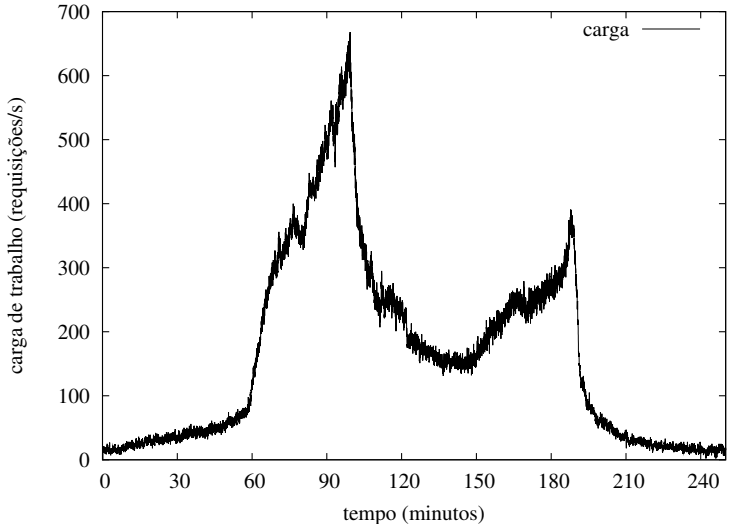


Figura 7.7: Perfil de carga de trabalho enviada ao *cluster*

Alguns dos parâmetros de configuração para o experimento estão indicados na Tabela 7.3. Aqueles que não constam nessa tabela possuem o mesmo valor da tabela utilizada para compor os experimentos da seção anterior (ver Tabela 7.1). A maior razão para a mudança desses valores é a diferença presente em cada *workload* para uma mesma janela de tempo. Por causa da escala em que as Figuras 7.1 e 7.7 foram apresentadas, este fato não é percebido de maneira trivial. Sendo assim, a Figura 7.8, com o objetivo de esclarecer esta afirmação, mostra a os valores da carga do minuto 60 ao minuto 120 de cada um dos experimentos. Ou seja, a carga de trabalho foi aumentada quando é comparada uma mesma janela de tempo.

Tabela 7.3: Parâmetros usados nos experimentos com análise de sensibilidade

γ_max	1,0 a 1,4
$util_{max}$	0,8 e 1,0
janela de decisão	4 s
janela de estabilização	1,5 min
intervalo de otimização	5 min
histórico de otimização	300 amostras

Outro ponto que deve ser abordado é que a redução dos valores feita nessas variáveis não foram realizadas9 de maneira proporcional à aceleração sofrida pela carga de trabalho. Isto é devido ao fato de que os valores inicialmente escolhidos já sofreram ajustes os quais permitiram um melhor desempenho em nível de qualidade de serviço ou prevenção de situações indesejáveis (tais como sucessivas trocas de máquinas em curto espaço de tempo). Portanto, os novos valores foram ajustados para que as modalidades do MLH (testes utilizando γ_max e $util_{max}$ diferentes entre si) realizassem um maior número de decisões

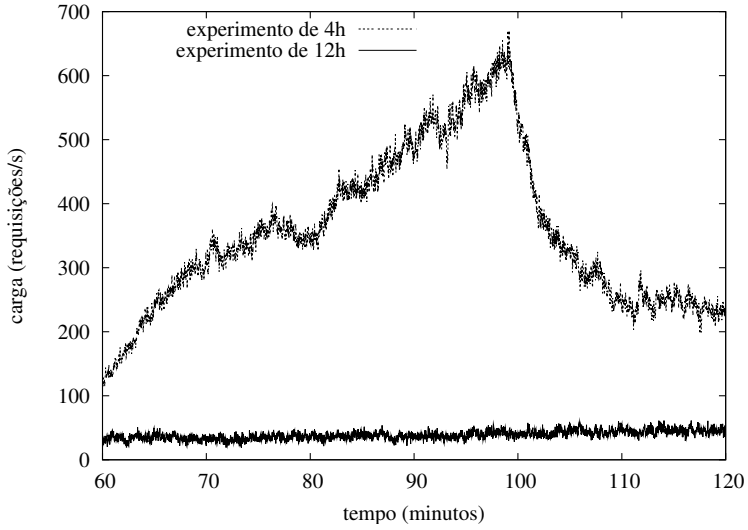


Figura 7.8: Perfil da carga de trabalho enviada ao *cluster*

durante o experimento, realçando dessa forma o impacto das diferentes configurações. Essas mudanças também foram feitas de forma o custo computacional da implementação não introduzisse nenhum (*overhead*) na execução do modelo proposto pelo *front-end*.

Para mostrar o impacto da variável $util_{max}$ (vista no Algoritmo 1) em uma situação real realizamos os mesmos testes para o valor $util_{max} = 1$. A Tabela 7.4 reúne os dados desses experimentos. Os dados relativos a $\gamma\_max = 1{,}0$ nessa tabela indicam uma maior perda de requisições. Aproximadamente 10% dessas requisições (de um total de $\approx$ 2,4 milhões), devido à ação do controle de geração de carga do *httperf*. Quando um grande número de requisições ultrapassa o prazo de atendimento um mecanismo é ativado para diminuir a geração de carga, até que a situação de sobrecarga desapareça. Como esse mecanismo reduz o número de requisições enviadas ao aglomerado *web*, o consumo de energia também torna-se menor, mas de maneira artificial. Ou seja, em função de uma característica do *httperf* que não ocorreria em uma situação real.

Tabela 7.4: MLH: $util_{max} = 1{,}0$ x $util_{max} = 0{,}8$

$\boldsymbol{util_{max} = 0{,}8}$					
γ_max	1,0	1,1	1,2	1,3	1,4
QoS $< 95\%$	1027	913	781	522	421
Requisições perdidas	28875	27292	26793	19684	17419
Energia (Wh)	735	741	752	783	791
$\boldsymbol{util_{max} = 1{,}0}$					
γ_max	1.0	1.1	1.2	1.3	1.4
QoS $< 95\%$	3300	1124	747	656	574
Requisições perdidas	261617	40115	27769	25493	23447
Energia (Wh)	650	727	739	756	769

A variação do parâmetro gama tem maior influência em situações nas quais há um aumento súbito no valor médio da carga dentro de um pequeno intervalo de tempo (se comparado com a duração de todo o experimento). Como o previsor não teve tempo para reagir a essa rajada de carga e capturar corretamente a ocorrência de uma tendência mais acentuada, perdas de QoS estão suscetíveis a acontecer. Se essas perdas se concretizarem, o fator gama será acionado e provocará a escolha de uma configuração com maior desempenho. Dependendo da seriedade dessas perdas, um maior valor para γ será necessário para contornar essas perdas. Dessa forma, os maiores valores mostram melhores índices de QoS, menor perda de requisições e um consumo de energia mais elevado.

O parâmetro $util_{max}$ tem sua influência no sistema durante toda a execução do sistema independentemente da variação da variável gama, bem com o seu valor máximo. Um maior desempenho é necessário nos momentos que existe substituição de máquinas. Em um cenário no qual a carga é crescente, o conjunto de máquinas ativas aumenta o seu desempenho gradativamente para atender a demanda. No momento que o previsor escolhe uma configuração na qual uma máquina deve ser substituída por outra com maior desempenho ou mais econômica em nível de energia para a demanda atual, uma determinada quantidade de tempo será necessária para que a máquina desligada seja iniciada. Como esse tempo é variável e a carga está aumentando, um maior desempenho escolhido para a configuração anterior (devido a um maior valor de $util_{max}$) atenua este impacto de transição.

Capítulo 8

Considerações Finais

No trabalho descrito aqui, a utilização de uma política preditiva visando economia de energia foi empregada em um ambiente de *cluster* de servidores. É importante enfatizar que esta aplicação não foi feita através de simulação e sim utilizando um aglomerado de servidores *web* real.

A aplicação *web* desenvolvida possui características de um sistema de tempo real não-crítico, isto é, uma parcela das requisições enviadas ao aglomerado deve obedecer um prazo a fim de satisfazer um contrato de qualidade de serviço. Baseada neste ambiente, a modelagem do mecanismo de economia de energia proposto relaciona uma métrica de QoS com a utilização da CPU dos computadores contidos no aglomerado. Tal característica torna o emprego desta política preditiva transparente em nível de aplicação.

Também foram mostrados os benefícios obtidos através da incorporação de uma técnica de previsão (HLM ou Método Linear de Holt) da evolução da carga a qual o sistema está sendo submetido. Ganhos energéticos superiores a 51% foram obtidos quando HLM é comparado ao esquema Linux chamado de *Performance*, e de aproximadamente 46% quando comparado ao também esquema padrão Linux *Ondemand*. Foi observado que esta economia de energia pode ser obtida respeitando-se contratos negociados para a qualidade de serviço.

Quando comparada a um sistema otimizado (OOO), a técnica aqui apresentada mostrou que atinge patamares de ganho energético superiores, e ainda se comportando melhor em termos de qualidade de serviço. A adoção de um método de previsão proporcionou ao sistema atuar de maneira pró-ativa, que mostrou ser uma característica importante para sistemas nos quais são exigidos altos níveis de qualidade de serviço.

A análise de sensibilidade feita com parâmetros da heurística utilizada, apresentada

na Seção 7.2, mostrou a sua influência tanto na economia de energia, quanto na qualidade de serviço apresentada pelo sistema. A variável gama tem com objetivo melhorar a QoS de forma dinâmica, ou seja, na medida que o nível de QoS do sistema apresenta-se abaixo do valor alvo, ela é ajustada para fazer uma correção na configuração do aglomerado. Já a variável $util_{max}$ ajusta o *cluster* para um estado de maior desempenho, para que o efeito de rajadas de carga e o *overhead* de transição seja atenuado, principalmente em situações com um pequeno número de servidores ativos. Uma possibilidade futura é também atuar nessa variávél tendo como base medidas e estatísticas que possam ser feitas *on-line*.

Em síntese, o trabalho proposto nessa dissertação mostrou a redução no consumo de energia em um aglomerado de servidores *Web* através da aplicação de uma polítca preditiva. Perante ao fato de que esse ambiente está sujeito à medidas para controlar a qualidade de serviço provida, a política em questão contém uma heurística capaz de aliar um aumento na qualidade de serviço do sistema sem deteriorar a economia de energia obtida.

Como trabalhos futuros, sugerimos a realização de experimentos para avaliar o impacto ocasionado por outras variáveis do modelo, como, por exemplo, a janela de controle. Um outro estudo a ser feito será a execução de testes com outros perfis de carga, para tentar avaliar características como sazonalidade e cargas com valores de autocorrelação mais baixos.

Para melhorar os tempos de execução na solução do problema de otimização da potência consumida pelo *cluster*, sugere-se a introdução de métodos heurísticos, bem como a utilização de soluções que não busquem necessariamente o ótimo global, mas que apresentem uma solução viável. Com uma solução dessa natureza, um menor esforço computacional será necessário, tornando a escolha *on-line* de uma melhor configuração um processo viável e aumentando a escalabilidade. Para avaliar a eficiência dessa nova estratégia uma comparação envolvendo o método *off-line* empregado nesse trabalho contra o *on-line* poderá ser feita. Finalmente, sugere-se avaliar o impacto de outras técnicas de previsão, além do método de Holt aqui utilizado. Por exemplo, soluções basseadas em Filtragem de Kalman já estão sendo investigadas em nosso grupo de pesquisa.

Referências

[1] Nevine AbouGhazaleh, Bruce Childers Daniel Mossé e Rami Melhem. Near-memory caching for improved energy consumption. *IEEE Transactions on Computers*, 56(11):1441–1455, Novembro 2007.

[2] ACPI. Advanced configuration and power interface, 2008. `http://www.acpi.info/`.

[3] ACPI 3.0b. Advanced configuration and power interface – Especificação 3.0b, 2008. `http://www.acpi.info/spec30b.htm`.

[4] ANEEL. Matriz elétrica brasileira. Agência Nacional de Energia Elétrica, Brasil, 2009. `http://www.aneel.gov.br/aplicacoes/capacidadebrasil/OperacaoCapacidadeBrasil.asp`.

[5] Apache. The apache HTTP server project, 2009. `http://httpd.apache.org/ABOUT_APACHE.html`.

[6] Apache. Módulo mod_proxy_balancer, 2009. `http://httpd.apache.org/docs/2.2/mod/mod_proxy_balancer.html`.

[7] Martin Arlitt e Tai Jin. Workload characterization of the 1998 Soccer World Cup web site. Relatório Técnico – Laboratórios Hewlett-Packard, 1999. `http://www.hpl.hp.com/techreports/1999/HPL-1999-35R1.pdf`.

[8] Yuliy Baryshnikov, Ed Coffman, Guillaume Pierre, Dan Rubenstein, Mark Squillante e Teddy Yimwadsana. Predictability of web-server traffic congestion. In *International Workshop on Web Content Caching and Distribution*, pp. 97–103, Washington, DC, EUA, Setembro 2005.

[9] Luciano Bertini, J. C. B. Leite e Daniel Mossé. Statistical QoS guarantee and energy-efficiency in web server clusters. In *19th Euromicro Conference on Real-Time Systems*, pp. 83–92, Pisa, Itália, Julho 2007.

[10] Luciano Bertini, J. C. B. Leite e Daniel Mossé. Dynamic configuration of web server clusters with QoS control. In *WiP Session, 20th Euromicro Conference on Real-Time Systems*, Praga, Rep. Tcheca, Julho 2008.

[11] Luciano Bertini, J. C. B. Leite e Daniel Mossé. Generalized tardiness quantile metric: Distributed DVS for soft real-time web clusters. In *21st Euromicro Conference on Real-Time Systems*, pp. 227–236, Dublin, Irlanda, Julho 2009.

[12] Luciano Bertini, J. C. B. Leite e Daniel Mossé. Power optimization for dynamic configuration in heterogeneous web server clusters. *Journal of Systems and Software*, aceito para publicação, 2010.

[13] Davide Bertozzi, Luca Benini e Bruno Ricco. Power aware network interface management for streaming multimedia. In *IEEE Wireless Communications and Networking Conference*, volume 2, pp. 926–930, Orlando, FL, EUA, Março 2002.

[14] Pat Bohrer, E.N. Elnozahy, Tom Keller, Michael Kistler, Charles Lefurgy, Chandler McDowell e Ram Rajamony. *Power aware computing*. Kluwer Academic Publishers, 2002.

[15] Valeria Cardellini, Emiliano Casalicchio, Michele Colajanni e Philip S. Yu. The state of the art in locally distributed web-server systems. *ACM Computing Surveys*, 34(2):263–311, 2002.

[16] Gong Chen, Wenbo He, Jie Liu, Suman Nath, Leonidas Rigas, Lin Xiao e Feng Zhao. Energy-aware server provisioning and load dispatching for connection-intensive internet services. In *USENIX Symposium on Networked Systems Design and Implementation*, pp. 337–350, São Francisco, EUA, Abril 2008.

[17] COIN-OR. Computational INfrastructure for Operations Research, 2009. `http://www.coin-or.org`.

[18] CPUFreq Governors. Linux Kernel Documentation, 2009. `http://www.mjmwired.net/kernel/Documentation/cpu-freq/governors.txt`.

[19] CPUFreq Subsystem. Linux Kernel Documentation, 2009. `http://www.mjmwired.net/kernel/Documentation/cpu-freq`.

[20] Desertec. Fundação desertec, 2009. `http://www.desertec.org/en/`.

[21] E.N. Elnozahy, Michael Kistler e Ramakrishnan Rajamony. Energy-efficient server clusters. In *Workshop on Power-Aware Computing Systems*, pp. 179–196, Cambridge, MA, EUA, Fevereiro 2002.

[22] EPA. Report on server and data center energy efficiency, 2007. `www.energystar.gov`.

[23] EPA. Energy star, 2009. `http://www.energystar.gov/index.cfm?c=prod_development.server_efficiency_study`.

[24] Jason Flinn e Mahadev Satyanarayanan. Energy-aware adaptation for mobile applications. *ACM SIGOPS Operating Systems Review*, 33(5):48–63, Junho 1999.

[25] Lakshmi Ganesh, Hakim Weatherspoon, Mahesh Balakrishnan e Ken Birman. Optimizing power consumption in large scale storage systems. In *USENIX Workshop on Hot Topics in Operating Systems*, pp. 1–6, San Diego, CA, EUA, Maio 2007.

[26] GLPK. Gnu linear programming kit, 2009. `http://www.gnu.org/software/glpk`.

[27] Erico Guizzo. Wind power in paradise. IEEE Spectrum Online, 2008. `http://spectrum.ieee.org/mar08/6020`.

[28] John E. Hanke e Arthur G. Reitsch. *Business Forecasting*. Prentice Hall, 1995.

[29] Tibor Horvath, Tarek Abdelzaher, Kevin Skadron e Xue Liu. Dynamic voltage scaling in multitier web servers with end-to-end delay control. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 444–458, San Jose, CA, EUA, Abril 2006.

[30] IEA. World energy outlook 2008 edition – executive summary, 2008. `http://www.iea.org/weo/2008.asp`.

[31] Tohru Ishihara e Hiroto Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *International Symposium on Low Power Electronics and Design*, pp. 197–202, Monterey, CA, EUA, Agosto 1998.

[32] Byoung Il Kim e Tae Gyu Chang. A power reduction technique based on the microscopic dynamic voltage scaling (DVS) of multimedia processors in wireless communication terminal. In *The First IEEE and IFIP International Conference in Central Asia on Internet*, pp. 1–4, Bishkek, República do Quirguistão, Setembro 2005.

[33] Dara Kusic. *Combined Power and Performance Management of Virtualized Computing Environments Using Limited Lookahead Control*. PhD thesis, Drexel University, Philadelphia, PA, EUA, Junho 2008.

[34] LBNL. The internet traffic archive, 2009. `"http://ita.ee.lbl.gov/index.html"`.

[35] David Mosberger e Tai Jin. httperf: A tool for measuring web server performance. In *Internet Server Performance Workshop*, pp. 59–67, Madison, WI, EUA, Junho 1998.

[36] ONS. Estatísticas nacionais online do reino unido. Office for National Statistics, Reino Unido, 2009. `http://www.statistics.gov.uk/statbase/tsdlistfiles.asp`.

[37] Venkatesh Pallipadi e Alexey Starikovskiy. The ondemand governor: past, present and future. In *Linux Symposium*, pp. 223–238, Ottawa, Canadá, Julho 2006.

[38] Vinicius Petrucci, Orlando Loques e Daniel Mossé. A framework for dynamic adaptation of power-aware server clusters. In *ACM Symposium on Applied Computing*, pp. 1034–1039, Honolulu, HI, EUA, Março 2009.

[39] Eduardo Pinheiro, Ricardo Bianchini, Enrique V. Carrera e Taliver Heath. Load balancing and unbalancing for power and performance in cluster-based systems. Relatório Técnico DCS-TR-440, Department of Computer Science, Rutgers University, EUA, 2001.

[40] Gang Qu. What is the limit of energy saving by dynamic voltage scaling? In *IEEE/ACM International Conference on Computer-aided Design*, pp. 560–563, San Jose, Califórnia, EUA, Novembro 2001.

[41] Cosmin Rusu, Alexandre Ferreira, Claudio Scordino e Aaron Watson. Energy-efficient real-time heterogeneous server clusters. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 418–428. San Jose, CA, EUA, Abril 2006.

[42] Carlos Santana, Luciano Bertini, J. C. B. Leite e Daniel Mossé. Applying forecasting to interval based DVS. In *Brazilian Workshop on Real-Time Systems*, pp. 13–20, Rio de Janeiro, RJ, Brasil, Maio 2008.

[43] Carlos Santana, J. C. B. Leite e Daniel Mossé. Load forecasting applied to soft real-time web clusters. In *ACM Symposium on Applied Computing*, Sierre, Suíça, Março 2010.

[44] SCIP. Solving constraint integer programs, 2009. `http://scip.zib.de/`.

[45] Steven C. Wheelwright Spyros Makridakis e Rob J. Hyndman. *Forecasting: Methods and Applications*. Kluwer Academic Publishers, 2002.

[46] Matthew E. Tolentino, Joseph Turner e Kirk W. Cameron. Memory miser: Improving main memory energy efficiency in servers. *IEEE Transactions on Computers*, 58(3):336–350, Março 2009.

[47] TPC. Transaction Processing Performance Council, 2009. `http://www.tpc.org/`.

[48] Yefu Wang, Xiaorui Wang, Ming Chen e Xiaoyun Zhu. Power-efficient response time guarantees for virtualized enterprise servers. In *IEEE Real-Time Systems Symposium*, pp. 303–312, Barcelona, Espanha, Dezembro 2008.

[49] Joachim Wuttke. lmfit – A C/C++ routine for Levenberg-Marquardt minimization with wrapper for least-squares curve fitting, 2009. `http://www.messen-und-deuten.de/lmfit/`.

[50] Wanghong Yuan e Klara Nahrstedt. Energy-efficient soft real-time CPU scheduling for mobile multimedia systems. In *ACM Symposium on Operating Systems Principles*, pp. 149–163, Bolton Landing, NY, EUA, Outubro 2003.