Aplicação de Redes Neurais na Construção de Servidores *Web* "Verdes"

Leandro S. de Sousa[1], J.C.B. Leite[1], and J.C. Stacchini de Souza[2]
{lsousa, julius, julio}@ic.uff.br *

[1] Instituto de Computação
[2] Departamento de Engenharia Elétrica
Universidade Federal Fluminense (UFF), Niterói, RJ, Brasil

Resumo O crescimento na demanda por serviços *web* requer uma maior capacidade de processamento e, assim, um aumento no consumo de energia para suportar esta infra-estrutura. A redução deste consumo envolve desafios técnicos e aspectos ambientais, pois na geração desta energia toneladas de carbono são lançadas na atmosfera. Como restrição adicional, a qualidade de serviço oferecida aos clientes deve ser mantida acima de um nível mínimo aceitável. Este trabalho está voltado para a economia de energia em *clusters* de servidores *web*, na direção da construção dos chamados servidores "verdes". Nossa solução envolve técnicas de otimização, a tecnologia *Dynamic Voltage Scaling* e a aplicação de redes neurais para suporte à tomada de decisão.

1 Introdução

A área de interesse deste trabalho está na economia da energia consumida em *clusters* de servidores que atendem aplicações *web*. Este tipo aplicação faz parte, cada vez mais, do cotidiano das pessoas e instituições, e é representado por, citando apenas alguns exemplos, sistemas de busca, comércio eletrônico e transações bancárias. O crescimento na demanda por estes serviços impõe uma maior capacidade processamento para atendê-los e, conseqüentemente, um aumento no consumo de energia para suportar esta infra-estrutura. A redução deste consumo envolve desafios técnicos e aspectos ambientais, pois na geração desta energia toneladas de carbono são lançadas na atmosfera [1].

Atualmente, temos a possibilidade de efetuar o gerenciamento do consumo de energia nos servidores em diversos pontos, tais como: no processador, na memória principal e no sistema de armazenamento em disco. Neste trabalho, atuamos no gerenciamento do consumo do processador através de uma estratégia baseada em *Dynamic Voltage Scaling* (DVS). A DVS utiliza a possibilidade, que está presente em vários processadores, de variar através de *software* a tensão de alimentação e a freqüência de operação. De forma abreviada, a DVS permite uma economia no gasto de potência, dado que esse é aproximadamente proporcional ao quadrado da tensão de alimentação (ou, ainda, ao cubo da freqüência de operação). Para

* Os autores agradecem ao CNPq e à Faperj pelo financiamento parcial desse trabalho.

detalhes, ver, por exemplo [2,3]. Isto possibilita, em conjunto com a medição da ocupação do servidor, ajustar a freqüência de operação do processador às necessidades dos sistemas de forma a economizarmos energia.

Os *clusters* de servidores são normalmente projetados para atender a um pico de demanda das aplicações. Desta forma, na maior parte do tempo estes servidores permanecem com grande parte de sua capacidade de processamento ociosa, ou seja, consumindo uma quantidade de energia que é parcialmente desperdiçada. Na estratégia aqui apresentada, reduzimos a freqüência de operação dos servidores nestes períodos e, conseqüentemente, diminuímos o consumo de energia. Com esse objetivo capturamos periodicamente a freqüência de operação e o percentual de utilização do processador (que designaremos no restante do texto apenas por "ocupação") de cada um dos servidores. Estes dados servem de entrada para uma rede neural que define a configuração das freqüências destes processadores. Esta configuração deve manter a ocupação abaixo de um certo limite, de forma a oferecer uma qualidade de serviços adequada e minimizar a energia consumida. A solução *on-line* exata para este problema sofre com problemas de escala, pois para um grande número de servidores esta se torna inviável devido a seu tempo de resposta. Desta forma, optamos pela aplicação de uma rede neural na solução deste problema, dado que o treinamento da rede neural acontece *off-line*.

Para a validação de nosso modelo efetuamos experimentos e comparamos os resultados com a estratégia implementada no sistema operacional Linux para a redução do consumo de energia [4], conhecida como *ondemand*.

Na seqüência, a seção 2 apresenta trabalhos relacionados. Na seção 3 apresentamos a definição do problema. Na seção 4 descrevemos como este problema foi equacionado através da utilização de redes neurais, e na seção seguinte tratamos da implementação da proposta. Na seção 6 apresentamos os experimentos realizados e, em seguida, as conclusões e indicações para trabalhos futuros.

2 Trabalhos relacionados

A redução no consumo de energia em *clusters* de servidores tem se tornado um ponto chave para as corporações que os utilizam. Uma política local, que está associada à carga de processamento de um único servidor, é destacada em [4] e faz parte da implementação do sistema operacional Linux. Em [5] os autores propõem uma política local, *Independent Voltage Scaling* (IVS), bem como uma política para o atendimento a um *cluster* de servidores homogêneos, *Coordinated Voltage Scaling* (CVS). No caso do artigo [5] também é proposto um esquema para ligar ou desligar servidores de acordo com a demanda por processamento.

Cabe ressaltar que, no trabalho aqui apresentado, tratamos de um *cluster* de servidores heterogêneos, ou seja, cada um dos servidores pode oferecer distintas capacidades de processamento. A primeira caracterização deste tipo de *cluster* heterogêneo e para aplicações *web* surgiu no artigo [6], no qual o DVS foi utilizado como a principal técnica. Outro trabalho, apresentado em [7], relaciona o DVS com técnicas para capturar e tomar ações em relação ao volume de requisições efetuadas a *clusters*. Em [8] os autores também tratam de *clusters* de servidores

heterogêneos destinados ao atendimento de aplicações *web*, mas seus critérios, para a variação na freqüência de operação dos processadores, estão baseados no controle direto da QoS e não na ocupação dos processadores, como neste trabalho.

3 Definição do problema

A arquitetura do *cluster* para aplicações *web* utilizada neste trabalho está indicada na Figura 1. Nela os usuários efetuam requisições por páginas *web* ao servidor *front-end* que as distribui aos servidores que efetivamente as executam. A execução destas páginas nos servidores gera a carga de processamento, a qual nomeamos por "ocupação" (U), que é o percentual de utilização do processador para uma determinada freqüência de operação. Este percentual de ocupação é capturado diretamente no sistema operacional.

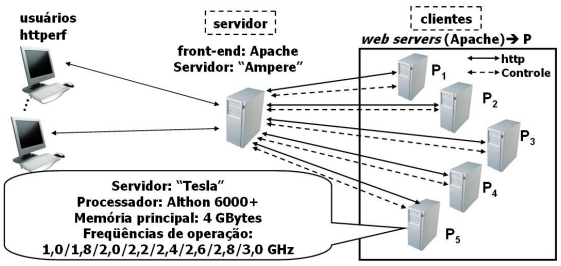

Figura 1. Arquitetura do *cluster* de servidores *web*.

A potência P consumida pelo *cluster* é o somatório da potência consumida por cada um dos N servidores P_i. A técnica DVS nos possibilita reduzir este consumo de potência alterando a freqüência de operação dos processadores de cada um dos N servidores.

Neste texto, designaremos por "configuração de freqüências" ao conjunto composto pela freqüência de operação, em um dado momento, para cada um dos processadores dos servidores *web*. Nosso objetivo é não permitir que, para uma determinada configuração de freqüências, a ocupação em cada um dos servidores ultrapasse a um determinado limite U_m. Este limite de ocupação nos permite manter uma capacidade de processamento ociosa para prover um certo nível médio de QoS.

De forma simplificada, quanto menor for a freqüência de operação do processador, para uma determinada carga de processamento, menor será o consumo de potência e maior será a ocupação. Desta forma, devemos buscar uma configuração de freqüências que maximize a ocupação, desde que abaixo de U_m, e ao mesmo tempo minimize a potência consumida. Isto por um lado nos provê um certo nível médio de QoS e por outro reduz o consumo de potência.

Para definir o problema apresentamos a Equação 1 a ser minimizada para obter o menor consumo de potência para os servidores, desde que obedeçamos às restrições impostas pelas Equações 2 e 3. Dado que a Equação 1 é uma

função da ocupação U_i em cada uma das possíveis freqüências de operação S_i dos N processadores, o que buscamos é a configuração de freqüências com as menores freqüências de operação individuais tal que a ocupação de cada um dos servidores seja inferior a U_m. A Equação 2 indica que apenas uma das possíveis freqüências de operação S_i, para cada um dos N processadores, pode fazer parte da solução. Na Equação 3 temos que a ocupação, dada a freqüência selecionada, deve estar limitada a U_m.

$$P = \sum_{i=1}^{N} \sum_{s=1}^{S_i} P_i^s U_i^s Y_i^s \tag{1}$$

$$\sum_{s=1}^{S_i} Y_i^s = 1, \forall s \in \{1...S_i\}, Y_i^s \in \{0,1\} \forall i \in \{1...N\}, \forall s \in \{1...S_i\} \tag{2}$$

$$Y_i^s U_i^s \leq U_m, \forall i \in \{1...N\}, \forall s \in \{1...S_i\} \tag{3}$$

Para a solucionar este problema utilizamos o *software* GLPK (*GNU Linear Programming Kit*), em sua versão 4.9. Assim, dado um conjunto de pares {freqüência, ocupação}, capturados em cada um dos N servidores *web*, é obtida a melhor configuração para as freqüências de operação, que minimiza a Equação 1 e obedece às restrições das Equações 2 e 3.

4 Aplicação de redes neurais artificiais

Neste trabalho é proposta a utilização de uma rede neural artificial para a estimação das freqüências de operação de cada servidor. Uma rede neural artificial é uma estrutura de processamento massivamente paralelizada, não algorítmica, capaz de aprender e armazenar conhecimento sobre um dado problema, a partir de casos históricos ou simulados. Vários modelos de redes neurais têm sido propostos na literatura, diferindo basicamente quanto à arquitetura, modelo dos neurônios, tipo de treinamento e regra de aprendizagem.

As redes neurais do tipo *Perceptron* de Múltiplas Camadas (MLP – *Multi-Layer Perceptron*) [9] têm sido as mais utilizadas em problemas de reconhecimento de padrões. A rede MLP é não-realimentada e utiliza treinamento supervisionado, sendo capaz de aproximar qualquer região de decisão. O modelo de neurônio mais empregado utiliza uma função de ativação do tipo sigmóide [9].

Na fase de aprendizado, padrões de entrada são apresentados à rede e propagados para frente (através das camadas intermediárias) até que as saídas dos neurônios da última camada sejam obtidas (saídas da rede). Os erros entre as saídas obtidas e as saídas desejadas (previamente conhecidas) são então utilizados para ajustar os pesos das conexões entre neurônios de modo a melhorar o desempenho computacional da rede. A regra de aprendizagem usualmente empregada é a da retropropagação de erros.

A rede neural substitui o módulo construído a partir do GLPK, pois a execução *on-line* deste módulo só é possível para um número pequeno de servidores. Na implementação das redes neurais utilizamos o *software* SNNS (*Stuttgart Neural Network Simulator*), versão 4.1 [10].

A arquitetura da RN utilizada é composta por uma camada de entrada, que é responsável pela recepção dos pares {freqüência,ocupação} de cada um dos servidores, uma camada intermediária e uma camada de saída, responsável por estimar o ajuste a ser realizado na freqüência de operação de cada servidor. Assim, o número de variáveis de entrada a serem consideradas será duas vezes o número de servidores.

Como mecanismo de aprendizado é empregado o aprendizado supervisionado, que se caracteriza pela utilização de um agente externo que indica à rede a resposta desejada para cada padrão de entrada. Uma função pseudo-aleatória gerou uma grande quantidade de pares de entrada que continham, para cada um dos servidores, um valor para a ocupação (entre 0% e 100%) e uma freqüência válida para cada um dos servidores. Esses dados foram submetidos à rotina criada através do GLPK, a qual retorna a solução que minimiza a Equação 1 obedecendo às restrições representadas pelas Equações 2 e 3. Dessa maneira, tem-se as saídas desejadas que, juntamente com os padrões de entrada contendo os pares (freqüência, ocupação) de cada máquina, formam a base de treinamento da RN. Esta base de treinamento foi composta por oitocentos mil conjuntos de padrões de entrada e na validação dos resultados obtidos utilizamos cinqüenta mil padrões de entrada.

Uma questão que devemos colocar é que os resultados válidos como respostas são discretos, ou seja, freqüências de operação para as CPUs envolvidas, mas os resultados obtidos pela RN são contínuos. Por este motivo é necessário construir uma função que faça o mapeamento dos resultados obtidos pela RN para os dados válidos para o problema. Por exemplo, considerando um servidor que disponibilize as freqüências de 1,0 e 1,8GHz e que a RN retorne o resultado 1,567GHz, a princípio somos levados a acreditar que 1,8GHz seria a freqüência mais apropriada, mas isto nem sempre é verdade. As fronteiras de decisão para, a partir da resposta da RN, atribuir um valor válido de freqüência imediatamente superior ou inferior foram definidas com base nas simulações realizadas e de forma a maximizar o número de acertos. Nestas simulações variamos a quantidade de neurônios na camada intermediária das RNs, entre um e trinta neurônios, e desprezamos as RNs que resultaram em uma taxa de erros superior a 1%. Dentre as que obtiveram taxas de erros inferiores a 1% foram escolhidas as RNs utilizadas no trabalho. A RN para um servidor tem 3 neurônios em sua camada intermediária e a com cinco servidores conta com 13 neurônios, suas taxas de erros foram de 0,82% e 0,87%, respectivamente.

5 Implementação

A arquitetura da aplicação é a indicada na Figura 1. Nela podemos observar que os usuários enviam suas requisições http ao servidor *front-end*, que utiliza o *software* Apache para distribuí-las entre os servidores *web*, que são os responsáveis pelo seu atendimento. Estas interações são representadas através das linhas contínuas na Figura 1.

Os dois módulos implementados neste trabalho são: o **Servidor** e o **Cliente**, que funcionam no servidor *front-end* e em cada um dos servidores *web*, respec-

tivamente. O módulo **Servidor** é ativado e recebe parâmetros que informam as possíveis freqüências de operação dos processadores dos servidores *web*, bem como o intervalo de tempo entre a captura dos dados de ocupação e freqüência. Em seguida o módulo **Cliente** é iniciado em cada um dos servidores *web*. Periodicamente, o **Servidor** requisita aos **Clientes** as informações de ocupação e freqüência. Estas são repassadas à rede neural que retorna as novas freqüências de operação. Com a nova configuração de freqüências, o **Servidor** a compara com a anterior e verifica se modificações são necessárias. Em caso positivo, os **Clientes** cujas freqüências devem ser alteradas recebem uma mensagem indicando suas novas freqüências de operação e realizam as alterações necessárias. Neste momento, o procedimento é repetido.

Na terceira seção, na qual apresentamos a definição do problema, indicamos que nossa estratégia deve manter a ocupação abaixo de um certo limite U_m, mas não definimos um valor. O U_m utilizado na implementação foi de 80%. Este valor foi escolhido por que também é o alvo para a ocupação na estratégia *ondemand* do Linux [4,11], que utilizamos para efeito de comparação.

6 Experimentos

Nos experimentos avaliamos a freqüência de operação média dos processadores dos servidores *web*, que nos indica a redução ou aumento no consumo de potência, e parâmetros de QoS. Estes pontos foram avaliados com o objetivo de verificar se nossa estratégia consegue reduzir o consumo de potência mantendo um certo nível médio de QoS.

Para validar os conceitos aqui apresentados, utilizamos, sem perda de generalidade, como ambiente para os experimentos, um único servidor na camada *web* (servidor *tesla* na Figura 1). Portanto, o sistema aqui implementado serviu como uma política DVS local. Utilizamos também o Apache 2.2.6 como servidor *web*, tanto no *front-end* quanto no servidor *web*, e o php 5.2.4 como linguagem de desenvolvimento da página *web* executada. As requisições dos usuários foram geradas através do *software* httperf [12], em sua versão 0.8. O httperf simulou requisições efetuadas por um certo número de usuários, buscando por uma página php que executa um procedimento que permite simular diferentes tempos de execução de páginas *web*. O httperf também retornou informações relativas a QoS para compararmos as duas estratégias avaliadas. Estas informações foram:

- duração do experimento (DM): as variações na duração de um mesmo experimento indicam se a QoS foi prejudicada (indicada em *s* na Tabela 1);
- duração média das conexões (DMC): esta informação indica se o usuário levou um tempo maior ou menor para executar todas as suas requisições por páginas *web* (indicada em *ms* na mesma tabela); e,
- requisições por segundo (RS): esta informação reflete a quantidade de requisições que foram atendidas, em média, por segundo durante todo o experimento. Este dado indica a capacidade de "resposta" do servidor em relação a carga submetida.

A informação relativa a freqüência média de operação do processador (FM) foi capturada diretamente do sistema operacional, em intervalos de um segundo. Devido a esta forma de coleta de dados, encontramos freqüências não disponíveis no processador, pois capturamos a média das freqüências utilizadas em cada intervalo. É importante lembrar que a medida da freqüência indica, indiretamente, o gasto de potência, já que esse é aproximadamente uma função cúbica da freqüência de operação. Estes quatro dados, apresentados na Tabela 1, serviram para comparar a estratégia implementada neste trabalho com a estratégia *ondemand* do Linux, nos experimentos realizados.

Na simulação da carga de processamento, 100 usuários fizeram requisições e aguardaram por um intervalo de tempo entre essas requisições. Estas duas variáveis diferenciaram os dois experimentos realizados, e foram: 200 e 422 usuários com intervalos de 0,2 e 0,475s, respectivamente, para os experimentos 1 e 2. Durante um experimento, os usuários requisitam a execução da página *web* até que a quantidade de requisições por usuário seja completada, quando a sessão do usuário é finalizada. Quando todos os usuários finalizam suas sessões o experimento é encerrado. Outra variável é a taxa na qual os usuários iniciam suas sessões, que nos experimentos foi de um usuário iniciando uma sessão a cada dois segundos.

Nos dois experimentos é gerada uma carga de processamento de forma a que, na estratégia *ondemand*, o processador atinja, em algum momento, a sua freqüência de operação máxima (3,0GHz). Eles diferem na forma como esta carga é alcançada. Na Figura 2 apresentamos um gráfico da taxa de requisições gerada pelo httperf durante os experimentos. Esta taxa indica a carga de processamento a que foi submetido o servidor nos dois experimentos e também o comportamento da mesma ao longo do tempo. No experimento 1, reduzimos a carga à metade, comparado ao segundo experimento, através da redução do número de usuários. Isto teve como objetivo verificar o comportamento das duas estratégias em situações nas quais a ocupação não é concentrada na capacidade máxima de processamento do servidor. No segundo experimento avaliamos uma carga, que cresce até o pico de processamento, estabiliza por um breve período e depois decai.

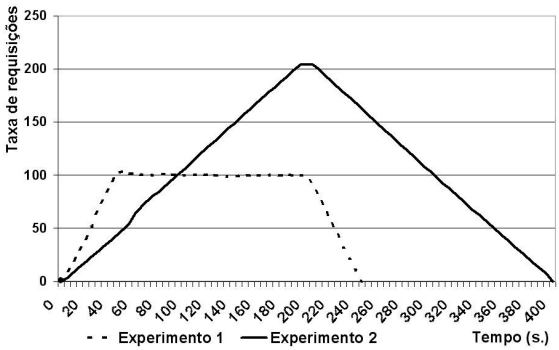

Figura 2. Taxa de requisições dos experimentos.

No primeiro experimento, Figura 3, observamos que ambas as estratégias flutuaram entre as freqüências intermediárias, mas nossa estratégia conseguiu flutuar em uma faixa inferior a alcançada pelo *ondemand*. Isto se refletiu nos resultados apresentados na Tabela 1, onde observamos que ocorreu uma forte redução na freqüência média de operação do processador (33,3%). Notamos também que o nível médio de QoS de ambas as estratégias se manteve bastante próximo. Estes dados indicam que nossa estratégia foi mais eficiente na adequação da capacidade de processamento em relação à carga recebida.

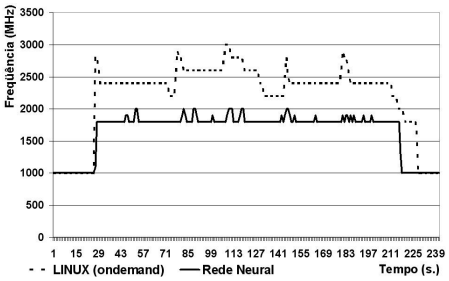

Figura 3. Freqüências durante a execução do experimento 1.

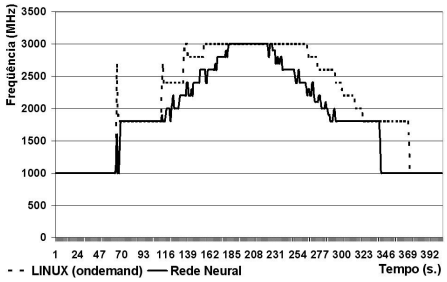

Figura 4. Freqüências durante a execução do experimento 2.

No segundo experimento, Figura 4, notamos que as duas estratégias aumentaram e, depois, reduziram a freqüência de operação do processador de forma gradual e em forma de "degraus", seguindo a carga de processamento recebida (Figura 2). Podemos observar que nossa estratégia aumentou a freqüência de operação de forma mais gradual, quando comparada a estratégia do Linux, ou seja, estabilizou em um número maior freqüências. Este melhor aproveitamento das freqüências disponíveis resultou em uma queda de 13,0% em relação à freqüência média alcançada pelo Linux, como apresentado na Tabela 1. Nesta mesma tabela observamos que a perda média de QoS se manteve abaixo de 0,5%.

Tabela 1. Resultados obtidos nos experimentos.

	Experimento 1			**Experimento 2**		
	Linux	**RN**	**Comp.(%)**	**Linux**	**RN**	**Comp.(%)**
FM	2.196	1.647	(33,3)	2.106	1.863	(13,0)
DE	241,7	242,5	0,4	403,9	405,1	0,3
DMC	21.277,1	21.848,3	2,6	41.042,6	41.202,7	0,4
RS	83,2	82,9	0,4	105,5	105,2	0,3

Durante os dois experimentos também capturamos as taxas de erros da rede neural, que se mantiveram próximas àquelas alcançadas durante a seleção da

RN. Os valores foram de: 1,41% e 0,81%, respectivamente, para cada um dos experimentos. Estes dados indicam que o treinamento da RN foi adequado ao problema a ser resolvido.

7 Conclusões

A principal conclusão deste trabalho é que as redes neurais podem ser utilizadas e tem potencial para gerar boas soluções *on-line* na questão da economia de energia em ambientes de processamento intensivo, como no caso de um *cluster* que atende aplicações *web*. As RNs se demonstraram eficazes para a estimação das freqüências de operação e suficientemente rápidas para aplicações *soft real-time*. Em nosso caso, o tempo de resposta da RN foi desprezível em relação ao intervalo de 0,5 segundo entre as intervenções na freqüência de operação do processador. Esta velocidade se deve ao fato de que os cálculos necessários para a resposta da RN são simples, dado que o peso das ligações é ajustado *off-line*, e dependem apenas do número de nós envolvidos.

Os experimentos demonstraram que a estratégia aqui implementada reduziu entre 13,0% e 33,3% a freqüência média de operação do processador em relação ao *ondemand* do Linux. Esta redução na freqüência média permitiu uma economia no gasto de energia, dado que esse é aproximadamente proporcional ao quadrado da tensão de alimentação ou, ainda, ao cubo da freqüência de operação (dado capturado durante os experimentos). Esta redução foi alcançada com uma perda na QoS, em média, inferior a 3%.

8 Trabalhos futuros

Nos experimentos avaliamos a estratégia que implementa uma política DVS local. Dando seguimento a este trabalho, vamos explorar um *cluster* composto por cinco servidores, cuja rede neural já foi selecionada, conforme apresentado na seção 4. Neste mesmo *cluster*, pretendemos avaliar, para realizar um gerenciamento mais fino na energia consumida, a questão de ligar e desligar servidores de acordo com os dados de ocupação. Na literatura, a utilização desta técnica, como apresentado em [5], propicia um incremento significativo na economia de energia. Além disto, nos experimentos, utilizaremos a medição direta do consumo de potência dos servidores e não a freqüência média de operação do processador, como no presente trabalho.

Uma linha interessante para trabalhos futuros é investigar, como introduzido em [13], a tendência de redução ou crescimento da carga de processamento em um futuro próximo dentro de um *cluster*. Com esta informação podemos fornecer à rede neural mais um dado de entrada para ajustar as freqüências ou ligar/desligar servidores.

Finalmente, podemos avançar para uma arquitetura na qual teremos uma rede neural de aprendizado contínuo, como descrito em [13], que indicaria tendências de ocupação e, subordinada a ela, de forma hierárquica, um conjunto de outras RNs que controlariam segmentos do *cluster*, contendo um conjunto de

servidores. Esta forma de tratar o problema pode resolver a questão da escala para esta aplicação, bem como possibilitar que segmentos distintos em um *cluster* possam ser tratados de forma diferenciada. Esta diferenciação pode ocorrer devido a características físicas dos servidores (localização, capacidade de processamento, refrigeração do ambiente, etc.) ou pela sua destinação lógica dentro do conjunto de servidores na solução de um determinado problema.

Referências

1. EPA: Report to congress on server and data center energy efficiency. Technical report, Environmental Protection Agency, EUA (Agosto, 2007)
2. P. Pillai, K. G. Shin: Real-time dynamic voltage scaling for low-power embedded operating systems. 18th Symposium on Operating Systems Principles, Banff, Alberta, Canadá (Outubro, 2001) 89–102
3. B.A. Novelli, J.C.B. Leite, J.M. Urriza, J.D. Orozco: Regulagem dinâmica de voltagem em sistemas de tempo real. SEMISH, São Leopoldo, RS, Brasil (Julho, 2005) 1772–1786
4. Linux: Linux kernel cpufreq subsystem. http://www.kernel.org/pub/linux /utils/kernel/cpufreq/cpufreq.html.
5. E.N. (Mootaz) Elnozahy, M. Kistler, R. Rajamony: Energy-efficient server clusters. Workshop on Power-Aware Computing Systems (Fevereiro, 2002)
6. T. Heath, B. Diniz, E.V. Carrera, W. Meira Jr., R. Bianchini: Energy conservation in heterogeneous server clusters. 10th ACM Symposium on Principles and Practice of Parallel Programming, Chicago, IL, EUA (Junho, 2005) 186–195
7. M. Elnozahy, M. Kistler, R. Rajamony: Energy conservation policies for web servers. 4th USENIX Symposium on Internet Technologies and Systems, Seattle, WA, EUA (Março, 2003)
8. L.Bertini, J.C.B. Leite, D. Mossé: Statistical QoS guarantee and energy-efficiency in web clusters. 19th Euromicro Conference on Real-Time Systems, Pisa, Itália (Julho, 2007) 83–92
9. S. Haykin: Neural networks: A comprehensive foundation. Prentice-Hal (1994)
10. SNNS: Stuttgart neural network simulator. http://www.nada.kth.se/ orre/snns-manual/.
11. K. Flautner, T. Mudge: Vertigo: Automatic performance-setting for Linux. 5th Symposium on Operating Systems Design and Implementation, Boston, MA, EUA (Dezembro, 2002) 105–116
12. HTTPERF: httperf documentation. http://www.hpl.hp.com/research/linux/ httperf/docs.php.
13. J. Huang, H. Jin, X. Xie, Q. Zhang: Using NARX neural network based load prediction to improve scheduling decision in grid environments. 3rd International Conference on Natural Computation, Wuhan, China (Agosto, 2007) 718–724