Economia de Energia em *Clusters* de Servidores *Web* Visando sua Aplicação em Larga Escala: Uma Arquitetura Hierárquica.

Leandro S. de Sousa[1] **e Anna Dolejsi Santos**[1]

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

lsousa@ic.uff.br, annads@ic.uff.br

Abstract. *The growth in the demand for web services requires a larger processing capacity and, consequently, an increase in the energy consumption to support this infrastructure. The reduction of this consumption involves technical challenges and environmental aspects, because in the energy generation tons of carbon are thrown in the atmosphere. As additional restriction, the quality of service, offered to the customers, should be maintained above an acceptable minimum level. This work is directed to the energy saving in web server clusters, in the direction of the "green" data centers' construction. Our solution involves optimization techniques, the Dynamic Voltage Scaling technology (DVS), and the application of an hierarchical architecture on the support of the decision making process for large scale clusters.*

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 infraestrutura. 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 data centers "verdes". Esta solução envolve técnicas de otimização, a tecnologia Dynamic Voltage Scaling (DVS) e a construção de uma arquitetura hierárquica para a sua aplicação em clusters de servidores em larga escala.*

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, consequentemente, um aumento no consumo de energia para suportar esta infraestrutura. A redução deste consumo envolve desafios técnicos e aspectos ambientais, pois na geração desta energia toneladas de carbono são lançadas na atmosfera [EPA 2007].

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 frequência 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 frequência de operação). Para detalhes, ver, por exemplo [P. Pillai and K. G. Shin 2001, B.A. Novelli et al. 2005]. Isto possibilita, em conjunto com a medição da ocupação do servidor, ajustar a frequência de operação do processador às necessidades dos sistemas de forma a economizarmos energia.

Os *clusters* de servidores são normalmente projetados para atender à 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 frequência de operação dos servidores nestes períodos e, consequentemente, diminuímos o consumo de energia. Com esse objetivo capturamos periodicamente a frequência 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 o sistema que define a configuração das frequências 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 [P. Pillai and K. G. Shin 2001], que serão apresentados na Seção 3, neste trabalho é apresentada uma alternativa para resolver esta questão através de uma arquitetura hierárquica.

Para a validação de nosso modelo efetuamos experimentos e comparamos os resultados com a solução exata para o problema, implementada através do GLPK (*GNU Linear Programming Kit*) [GLPK 2010], e, também, com estratégia implementada no sistema operacional Linux para a redução do consumo de energia [Linux 2010], conhecida como *ondemand*.

Na sequência, 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 da arquitetura hierárquica. Na quinta seção tratamos da questão da escala dos *clusters* de servidores na arquitetura em hierarquia, quando comparada com a tradicional, e na seção seguinte apresentamos os experimentos realizados. Nas Seções 7 e 8 temos as conclusões e indicações para trabalhos futuros, respectivamente.

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 [Linux 2010] e faz parte da implementação do sistema operacional Linux. Em [E.N. (Mootaz) Elnozahy et al. 2002] os autores propõem uma política local, *Independent Voltage Scaling* (IVS), bem como uma política para o atendimento à um *cluster* de servidores homogêneos, *Coordinated Voltage Scaling* (CVS). Neste mesmo artigo 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 [T. Heath et al. 2005], no qual a DVS foi utilizada como a principal técnica. Outro trabalho, apresentado em [M. Elnozahy et al. 2003], relaciona a DVS com técnicas para capturar e tomar ações em relação ao volume de requisições efetuadas a *clusters*. Em [L.Bertini et al. 2007] os autores também tratam de *clusters* de servidores heterogêneos destinados ao atendimento de aplicações *web*, mas seus critérios, para a variação na frequência de operação dos processadores, estão baseados no controle direto da QoS e não na ocupação dos processadores, como neste trabalho. Um dos autores do presente trabalho também investigou a utilização de Redes Neurais Artificiais no artigo [L.S. Sousa et al. 2008], visando a questão de escala do problema. Um trabalho visando *clusters* de maiores dimensões pode ser encontrado em [Rountree et al. 2007], que trata de uma questão mais específica, no caso, voltada para aplicações MPI.

3. Definição do problema

A arquitetura do *cluster* para aplicações *web* mais frequentemente utilizada está indicada na Figura 1 e no texto é designada por "arquitetura tradicional". Nela os usuários efetuam requisições por páginas *web* ao servidor *front-end* que as distribui entre os 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 frequência de operação. Este percentual de ocupação é capturado diretamente no sistema operacional.

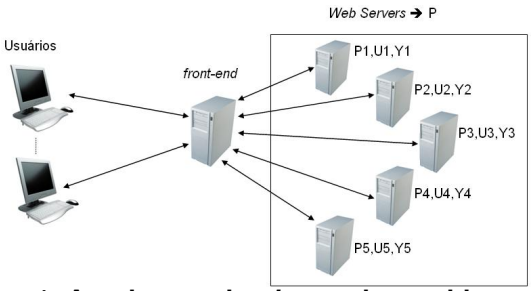

Figura 1. Arquitetura do *cluster* de servidores *web*.

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 frequência de operação dos processadores de cada um dos N servidores.

Neste texto, designaremos por "configuração de frequências" ao conjunto composto pela frequência 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 frequências, a ocupação em cada um dos servidores ultrapasse 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 frequência 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 frequências 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 frequências de operação S_i dos N processadores, o que buscamos é a configuração de frequências com as menores frequências de operação individuais tal que a ocupação de cada um dos servidores seja inferior a U_m. A Equação 2 indica, através da variável Y_i, que apenas uma das possíveis frequências de operação S_i, para cada um dos N processadores, pode fazer parte da solução. Na Equação 3 temos que a ocupação, dada a frequência selecionada, deve estar limitada a U_m.

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

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

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

Este problema de escalonamento pertence à classe de problemas NP-Difíceis, para maiores detalhes indicamos a leitura de [M.R. Garey and D.S. Johnson 1979]. Na solução utilizamos o *software* GLPK, em sua versão 4.9. Assim, dado um conjunto de pares {frequência, ocupação}, capturados em cada um dos N servidores *web*, é obtida a melhor configuração para as frequências de operação, que minimiza a Equação 1 e obedece às restrições das Equações 2 e 3.

4. Arquitetura hierárquica

Essa arquitetura foi construída a partir da idéia de "dividir e conquistar". Nela o *cluster* foi dividido em segmentos (vide a Figura 2), cada segmento é responsável pela execução da estratégia DVS e por ligar ou desligar seu próprio conjunto de servidores. Os segmentos de *cluster* repassam suas informações de carga {frequência e ocupação} para um nó raiz na hierarquia, que é responsável pela distribuição das solicitações dos usuários entre os segmentos. Desta forma, com um número pequeno de servidores, para a aplicação do algoritmo exato sobre o problema (construído através do GLPK), o tempo de resposta é reduzido e viabiliza a aplicação das técnicas DVS para um grande número de servidores.

Na Figura 2 temos um esquema da arquitetura proposta, mas antes de prosseguirmos, vamos introduzir os principais elementos dessa arquitetura, para os quais utilizamos letras e índices na figura:

- **Client (C)**: é a parte da aplicação que executa nos servidores do *cluster* que efetivamente atendem às solicitações dos clientes por páginas *web*. Este é responsável pela aplicação da mudança na frequência de operação dos processadores e pela captura das informações de ocupação e frequência necessárias à execução da estratégia baseada em DVS.

- **Server (S)**: é responsável pela execução do algoritmo da estratégia DVS bem como ligar e desligar servidores do tipo **C**.
- **Root (R)**: este é a raiz da hierarquia, e é responsável por definir a porção da carga de processamento destinada à cada um dos segmentos do *cluster*.
- **Load (L)**: esta parte da aplicação distribui a carga definida em **R** entre os segmentos do *cluster* e funciona como o *front-end* da arquitetura tradicional.

No texto, deste ponto em diante utilizamos **Client**, **Server**, **Root** e **Load**, para designar os componentes da arquitetura.

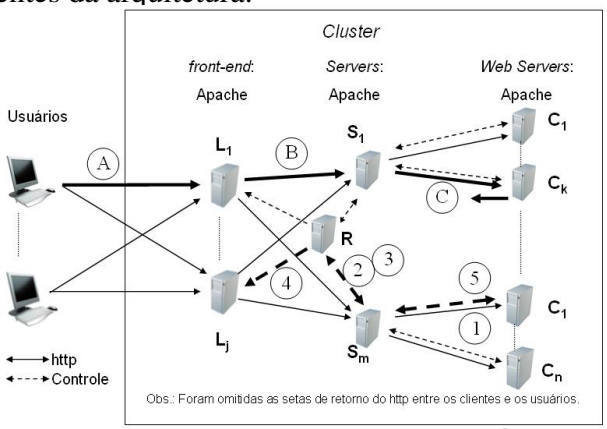

Figura 2. Esquema da arquitetura hierárquica.

Na Figura 2 temos de um lado os usuários, que solicitam a execução das páginas, e do outro o *cluster*, que atende tais solicitações. Usamos as linhas contínuas para definir as interações entre os componentes que atendem à aplicação *web*, já as linhas tracejadas correspondem às interações de controle da arquitetura hierárquica. As linhas mais espessas, tanto contínuas quanto tracejadas, indicam um fluxo completo de interações, que foram destacados para serem utilizados na descrição dos mesmos. Os círculos contendo letras destacam os passos da solicitação por páginas *web* feitas pelos usuários (fluxo http) e os que contém números acompanham o fluxo de controle da arquitetura hierárquica (fluxo de controle). Na sequência temos a descrição destes fluxos.

Fluxo de solicitações http:

- **A**: os usuários enviam suas solicitações para algum servidor de distribuição de carga (**Load**);
- **B**: o servidor **Load** às distribui convenientemente entre os servidores **Server**; e
- **C**: os servidores **Server**, por sua vez, distribuem a carga recebida entre seus próprios servidores *web*, nos quais executam os **Client**, que são responsáveis pelo atendimento.

Fluxo de controle da arquitetura hierárquica:

- **1**: cada um dos servidores **Server** solicita e recebe dos servidores **Client**, sob sua responsabilidade, seus pares de ocupação e frequência de operação;
- **2**: com estes dados os servidores **Server**, calculam a carga total em seus segmentos do *cluster* e repassam esta informação ao servidor **Root**;
- **3**: o servidor **Root** executa um algoritmo de distribuição da carga entre os segmentos do *cluster* e repassa aos servidores **Server** a informação sobre a carga à que serão submetidos. Desta forma, para o **Root** os servidores **Client** não existem e os servidores **Server** são apenas equipamentos com "grande capacidade";

- **4**: neste passo, o servidor **Root** envia a nova configuração de distribuição de carga aos servidores **Load**, que distribuem as requisições dos usuários entre os segmentos do *cluster*, através dos servidores **Server**; e
- **5**: os servidores **Server** definem como esta carga será distribuída entre os servidores **Client** (estratégia DVS), criando uma nova configuração de frequências para seus próprios servidores **Client**.

Note que, diferente do que ocorre na arquitetura tradicional para esse tipo de aplicação (veja a Figura 1), podemos ter mais de um ponto de entrada das solicitações dos usuários no *cluster*, ou seja, vários *front-end*. Isto é comum nos *clusters* de maiores dimensões que atendem aplicações *web*, pois somente um ponto de entrada seria um risco para o atendimento aos clientes, que poderiam ser impedidos de acessar às páginas contidas no *cluster* com a falha de um único servidor.

Os servidores do tipo **Server (S)** também incorporam servidores de **Load**, para que a carga recebida dos *front-end* seja distribuída convenientemente entre os seus servidores **Client**. Essa "indireção" no encaminhamento das solicitações pode causar a impressão que isto retarda o atendimento aos usuários, mas feitas as medições constatamos este "retardo" fica abaixo de um milissegundo, ou seja, é desprezível.

5. A questão de escala

De forma a avaliar a performance e a correção das soluções, pois ainda não nos foi possível avaliar a arquitetura em um *cluster* com a dimensão necessária, a arquitetura desenvolvida foi preparada para trabalhar em modo de simulação. Nesta forma de trabalho, que pode funcionar em hierarquia (com um servidor **Root**) ou na arquitetura tradicional, os componentes do tipo **Client**, vide a Figura 2, são configurados para ler de um arquivo os pares de dados (frequência e ocupação). Na simulação as ações finais não são tomadas, ou seja, não são alteradas as frequências de operação dos processadores nem o balanceamento na distribuição das solicitações dos usuários em **Load** e **Server**. O restante do processo, de geração da nova configuração de frequências, é idêntico à execução real do sistema, conforme descrito na Seção 4. Estas ações não têm impacto significativo no tempo total de geração da solução, pois são efetuadas através da escrita de alguns poucos *bytes* em arquivos texto e nas medições efetuadas esses tempos foram desprezíveis.

As simulações foram realizadas em três computadores, conforme apresentado na Figura 3 e suas configurações estão listadas na Tabela 1. A arquitetura foi configurada de duas formas diferentes:

- em hierarquia, Figura 2, na qual todos os **Client** foram colocados na máquina "B", na máquina "N" ficou o servidor **Root** e na máquina "P" ficaram os servidores **Load** e **Server**. Esta distribuição dos componentes foi proposital, pois todas as mensagens trocadas entre os diferentes módulos do sistema foram feitas entre máquinas e nunca localmente. Com os **Server** em uma única máquina perdemos o paralelismo na execução da estratégia. O **Root** executou na máquina de menor capacidade, o que criou uma sobrecarga na solução em hierarquia. Todos os **Client** localizados em um mesmo equipamento dificultou a comunicação, pois toda ela foi efetuada através de uma única interface de rede e não se beneficiou do paralelismo na comunicação.

- na arquitetura tradicional, Figura 1, todos os **Client** foram colocados na máquina "B" e na máquina "P" ficaram os servidores **Load** e **Server**. Neste caso a máquina "N" não foi utilizada.

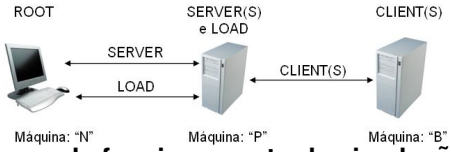

Figura 3. Esquema de funcionamento da simulação das soluções.

Tabela 1. Equipamentos utilizados nas simulações.

Equipamentos	Descrição
N	Intel Atom N270 1.6 GHz Memória de 1GB
B	AMD Althlon 64 X2 Dual 5200+ 2.7 GHz Memória de 2GB
P	AMD Althlon 64 X2 Dual 6000+ 3.0 GHz Memória de 4GB

As três máquinas foram interligadas através de um *switch* de 100Mbps e executavam o Linux Ubuntu (9.4). Foram medidos os tempos de geração da solução e os resultados foram sempre comparados com a versão gerada a partir do GLPK, para analisar a correção dos mesmos, pois a distribuição da carga de processamento entre os segmentos do *cluster* é baseada em uma heurística, desenvolvida pelos autores, que ordena os servidores **Client** de acordo com a capacidade de processamento dos mesmos em ordem não crescente. Esta heurística tenta ocupar ao máximo os equipamentos de maior capacidade, possibilitando o desligamento de um maior número de servidores. Isto porque, conforme descrito em [Rusu et al. 2006], as maiores reduções no consumo total de energia do *cluster* ocorrem quando aplicamos em conjunto as técnicas DVS e ligando/desligando servidores de acordo com a demanda por processamento dos usuários. Sua complexidade é da ordem de O(n), portanto é eficiente na busca da solução, mas introduz um certo nível de erros em relação à solução exata. Este erro decai com o aumento do número de servidores no *cluster*, ou seja, o erro é reduzido quando ganhamos em escala, essa questão pode ser observada nos resultados da simulação. A medição de tempo durante a simulação foi feita desde à requisição pelos dados de ocupação e frequência (Figura 2 no passo 1) do primeiro **Client** até que a nova configuração de frequências de operação fosse entregue ao último **Client** (Figura 2 no passo 5).

Na forma tradicional da arquitetura utilizamos apenas um **Server** e um conjunto de **N Client** (indicado na linha **Client** da Tabela 2), que se comunicaram apenas com este **Server**. No caso da arquitetura hierárquica seus dados de configuração foram listados na Tabela 2.

Tabela 2. Dados das seis configurações utilizadas em relação aos elementos da arquitetura em hierarquia.

Elementos/Configuração	1	2	3	4	5	6
Root	1	1	1	1	1	1
Load	2	2	2	2	2	2
Server	2	2	4	6	8	10
Client	10	20	40	60	80	100

Neste ponto vamos definir como foram gerados os registros de frequência e ocupação para cada um dos **Client** dos *clusters* utilizados nas simulações. As simulações usaram, para cada uma das quantidades de servidores do tipo **Client** (vide a Tabela 2)

1000 registros para cada um dos **Client**, que foram separados em 20 conjuntos de 50 registros e gerados de forma pseudo-aleatória. Para cada um dos 20 conjuntos foi selecionado um novo conjunto de equipamentos para o *cluster*, que foram ordenados em ordem não crescente de acordo com suas frequências máximas disponíveis, assumindo que tenham maior capacidade de processamento, sem perda de generalidade. Estes equipamentos foram selecionados de forma pseudo-aleatória dentro da lista de processadores da Tabela 3. Após a seleção dos equipamentos que compõem um determinado *cluster*, foram gerados os conjuntos de frequência e ocupação para o *cluster*. Esta forma de seleção dos registros e *clusters* foi definida para que tivéssemos uma variação nas configurações, mais homogênea ou heterogênea em relação aos processadores que constituem o *cluster*, e assim evitar alguma possível tendência nas soluções encontradas. Nesta simulação comparamos os resultados gerados na arquitetura implementada (DVS Hierárquica) com os gerados pelo GLPK, utilizando a arquitetura tradicional (DVS GLPK).

Tabela 3. Equipamentos utilizados no *cluster*.

Processadores (AMD)	Frequências disponíveis (MHz)
E4500	1000/1800/2000/2200
Athlon6 4 X2 4450e	1000/1800/2000/2200/2300
LE-1620	1000/1800/2000/2200/2400
Athlon X2 4850e	1000/1800/2000/2200/2400/2500
Atlhlon X2 5000	1000/1800/2000/2200/2400/2600
Athlon 64 X2 5200+	1000/1800/2000/2200/2400/2600/2700
Athlon 64 X2 5600+	1000/1800/2000/2200/2400/2600/2800
AthlonII X2 AM3 245	1000/1800/2000/2200/2400/2600/2800/2900
Athlon 64 x2 6000	1000/1800/2000/2200/2400/2600/2800/3000

Consideradas todas essas variáveis e configurações podemos apresentar os resultados obtidos, listados na Tabela 4, que correspondem ao tempo médio expresso em milissegundos na aplicação das estratégias. Podemos observar que a partir de 20 servidores a solução exata *on-line* é de difícil aplicação prática, pois o intervalo entre as intervenções para a busca de uma nova configuração de frequências gira em torno de um segundo para as aplicações *on-line*, dada a flutuação da carga de requisições imposta pelos usuários na Internet, por exemplo. Nesta mesma situação a solução em hierarquia continua com uma performance bastante interessante, ou seja, nunca esteve acima de 50 milissegundos. As casas decimais dos tempos foram arredondadas para não trabalharmos com frações de milissegundos. Outra questão é que, na geração das novas configurações, as estratégias não foram aplicadas aos resultados triviais, que resultam em todos os servidores com suas frequências mínimas ou máximas, pois estes não demandam cálculos para a configuração das frequências de operação das CPUs.

Tabela 4. Resultados expressos em milissegundos.

Quantidade de servidores	10	20	40	60	80	100
DVS hierárquica	34	36	38	41	44	49
DVS GLPK	35	322	1570	8795	20870	58234

Na Tabela 5 temos os percentuais de erro médio de todos os experimentos realizados, através do modo de simulação da arquitetura implementada, em relação à solução ótima. Este erro é gerado no segmento que recebe carga, mas não em sua frequência máxima, destacado por DVS na Figura 4 na qual usamos um *cluster* composto por três

segmentos, como um exemplo ilustrativo. Esta situação ocorre quando a melhor solução, para minimizar o somatório das frequências, envolve equipamentos do segmento de *cluster* anterior (mais à esquerda na figura), que não é alcançado pela estratégia DVS executada no servidor **Server** dedicado ao segmento destacado por "DVS" na figura. Como podemos observar o erro decai a medida que aumentamos o número de servidores que atendem às aplicações *web*.

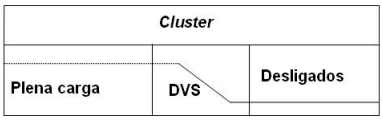

Figura 4. Esquema de funcionamento da simulação das soluções.

Tabela 5. Percentual de erros em relação à quantidade de servidores no *cluster*.

Quantidade de servidores	**10**	**20**	**40**	**60**	**80**	**100**
Em relação apenas à CPU (%)	0.13	0.09	0.04	0.02	0.01	0.01

6. Experimentos

Nos experimentos avaliamos a frequência 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, foi utilizado, sem perda de generalidade, como ambiente para os experimentos, os equipamentos listados na Tabela 1 e adicionamos um outro de configuração idêntica ao equipamento "**N**", que chamamos de "**E**", que foram interligados através de um *switch* de 100Mbps. Os **Client** ficaram nos equipamentos "**P**" e "**B**", os componentes **Server** e **Load** foram localizados em "**N**" e o **Root** em "**E**". O equipamento "**E**" também foi utilizado nos experimentos como origem das solicitações dos usuários. Este *cluster* de pequenas proporções foi escolhido propositalmente para verificar se a arquitetura hierárquica é competitiva em relação à solução ótima, mesmo para configurações nas quais sua aplicação não seria necessária, devido à pequena escala.

Foi utilizado o Apache 2.2.6 como servidor *web* no *front-end* (**Load**), nos servidores *web* e nos **Server**, e o php 5.2.4 como linguagem de desenvolvimento da página *web* executada. As requisições dos usuários foram geradas através do *software* httperf [HTTPERF 2010], 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 e cargas de execução de páginas *web*. O httperf também retornou informações relativas a QoS para compararmos as três estratégias avaliadas: a arquitetura hierárquica, a tradicional com o *ondemand* e com o algoritmo gerado pelo GLPK na arquitetura tradicional, que foram destacadas nos resultados por DVS-H, *Ondemand* e DVS-T, respectivamente.

Na apresentação dos resultados utilizamos quatro gráficos e uma tabela, por experimento, contendo os dados capturados durante a execução dos experimentos. Nas Figuras 5 e 9 temos a frequência média alcançada pelas CPUs e nas Figuras 6 e 10 a ocupação percentual média das mesmas. As medidas percentuais de qualidade de serviço são apresentadas nas Figuras 7 e 11, sempre em comparação com o *ondemand* do Linux (marco zero do gráfico), os valores são percentuais, que quando negativos indicam uma

piora na QoS da solução e positivos caso contrário, a origem dos dados destes gráficos são os valores contidos nas Tabelas 7 e 8, respectivamente. Nas Figuras 8 e 12 temos o comportamento da carga expressa em requisições dos usuários por segundo ao longo dos experimentos.

Os parâmetros de QoS avaliados, capturados no httperf, foram:

- **duração (s.)**: as variações na duração de um mesmo experimento indicam se a QoS foi prejudicada;
- **requisições (req/s.)**: esta informação reflete a quantidade de requisições que foram atendidas, em média, por segundo durante todo o experimento. Este dado indica a capacidade de execução do servidor em relação a carga submetida;
- **resposta (resp./s.)**: esta informação indica o comportamento do servidor em relação às respostas aos usuários; e
- **sessões (sess./s.)**: esta taxa indica se ocorreu alguma dificuldade na conexão dos usuários com o servidor.
- **frequência (MHz)**: indica o comportamento da frequência média de operação dos servidores diante da carga à que foram submetidos pelo httperf;

Neste trabalho apresentamos dois experimentos. No gerador de carga httperf variamos o "peso" do programa a ser executado (o que chamamos apenas de "peso" é um número que é submetido à uma página php que verifica se este é um número primo, quanto maior for o número maior será a carga de processamento gerada nesta avaliação), a quantidade de sessões de usuários abertas, a taxa na qual essas sessões foram iniciadas, a quantidade de solicitações dos usuários por sessão e o intervalo entre as solicitações dos usuários em cada uma das sessões. Estes dados foram listados na Tabela 6.

Tabela 6. Definição da carga dos experimentos.

Variáveis/Experimentos	Experimento 1	Experimento 2
Peso	370	600
Usuários	100	40
Taxa de início das sessões (sess./s.)	0.5	0.5
Solicitações por sessão	422	100
Intervalo entre as solicitações (s.)	0.475	0.200

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 nos experimentos foi de 70% no primeiro experimento e de 75% no experimento 2. Estes valores foram escolhidos para demonstrar sua influencia nos resultados da QoS e da frequência média do *cluster* durante os experimentos. A informação relativa à frequência média de operação do processador foi capturada diretamente do sistema operacional, em intervalos de um segundo. Devido à esta forma de coleta de dados, encontramos frequências não disponíveis diretamente nos processadores, pois capturamos a média das frequências utilizadas em cada intervalo. É importante lembrar que a medida da frequência indica, indiretamente, o gasto de potência, já que esse é aproximadamente uma função cúbica da frequência de operação.

6.1. Experimento 1

Note que neste experimento obtivemos um bom ganho em termos de QoS e mesmo assim reduzimos a frequência média do processador 6.23% e 7.94% nas duas estratégias. Estes

dados são apresentados na Figura 7 e listados na Tabela 7. A redução da frequência média foi um pouco tímida, mas devemos lembrar que neste caso o U_m foi ajustado para 70%. A diferença entre os resultados DVS-H e DVS-T se deve a execução da heurística no **Root**, pois o *cluster* utilizado tem dimensões bastante reduzidas. A DVS-H alcançou uma ocupação maior que a observada em DVS-T isto resultou em melhores resultados na frequência média, mas prejudicou seus resultados na QoS, mesmo assim foram bastante similares e competitivos.

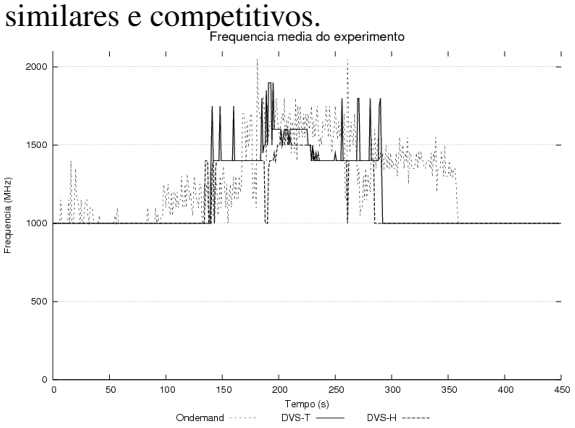

Figura 5. Frequências durante a execução do experimento 1.

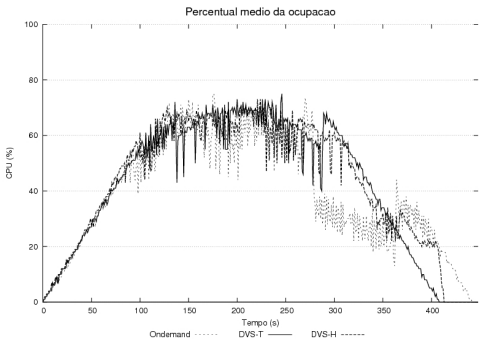

Figura 6. Ocupação da CPU durante a execução do experimento 1.

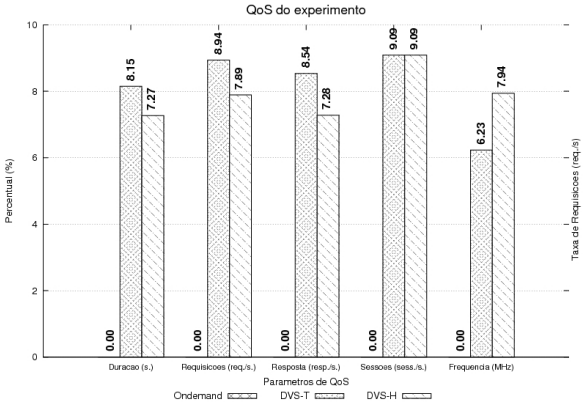

Figura 7. QoS durante a execução do experimento 1.

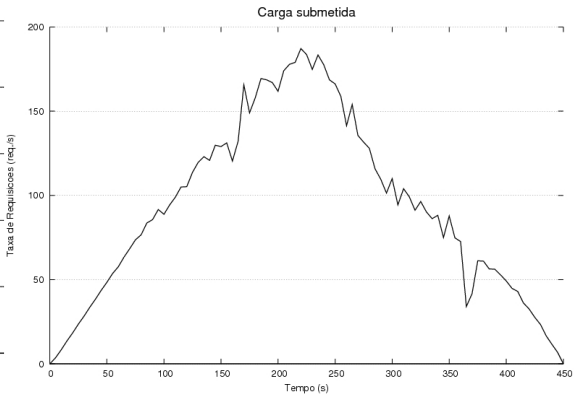

Figura 8. Carga submetida durante a execução do experimento 1.

Tabela 7. Resultados obtidos no experimento 1.

QoS/Experimento	Ondemand	DVS-T	DVS-H
Duração do experimento (s.)	447.89	411.39	415.35
Taxa de requisições (req./s.)	95.1	103.6	102.6
Taxa de resposta (rep./s.)	94.8	102.9	101.7
Taxa de sessões (sec./s.)	0.22	0.24	0.24
Frequência média (MHz)	1235	1158	1137

6.2. Experimento 2

Neste experimento, podemos destacar o comportamento bastante estável (Figura 9) da estratégia hierárquica em relação à carga submetida ao *cluster*, apresentada na Figura 12. Este comportamento teve muita influência nos resultados em termos de QoS, apresentados na Figura 11 e listados na Tabela 8. Com U_m ajustado para 75% melhoramos, em relação

ao experimento anterior, na questão da frequência média, mas isto reduziu os ganhos em termos de qualidade de serviços em relação ao *Ondemand*, mesmo assim esta não foi inferior em nenhum dos critérios avaliados. Cabe ressaltar que a definição do U_m tem influência direta nos resultados, ou seja, quanto maior for este valor limite melhores serão os resultados na economia de energia, mas temos uma redução nos índices de QoS. Portanto, na definição desse limite devemos levar em consideração as características das aplicações disponibilizadas no *cluster* e nossas espectativas em termos de QoS.

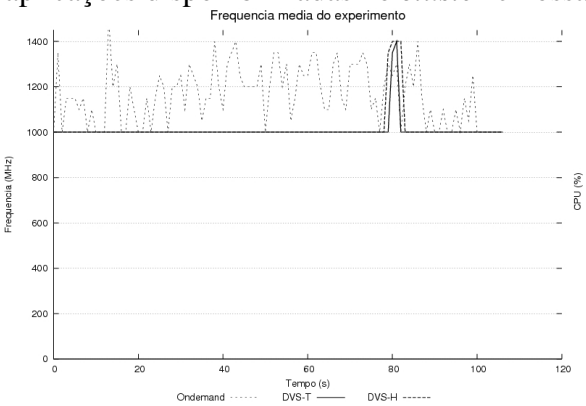

Figura 9. Frequências durante a execução do experimento 2.

Figura 10. Ocupação da CPU durante a execução do experimento 2.

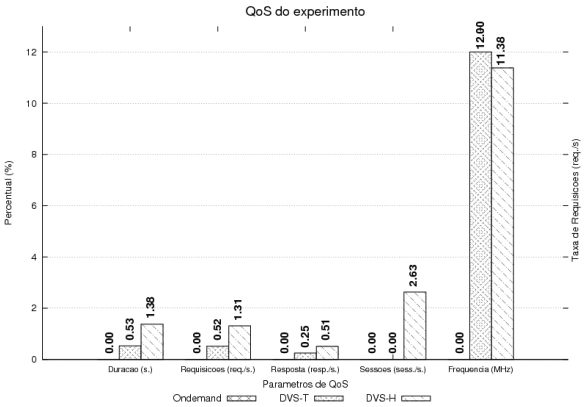

Figura 11. QoS durante a execução do experimento 2.

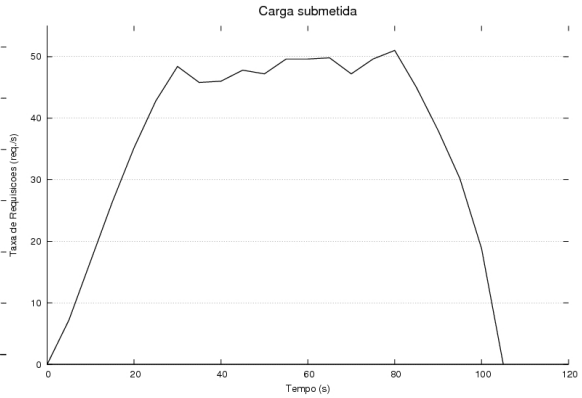

Figura 12. Carga submetida durante a execução do experimento 2.

Tabela 8. Resultados obtidos no experimento 2.

QoS/Experimento	Ondemand	DVS-T	DVS-H
Duração do experimento (s.)	104.74	104.18	103.29
Taxa de requisições (req./s.)	38.2	38.4	38.7
Taxa de resposta (rep./s.)	39.6	39.7	39.8
Taxa de sessões (sec./s.)	0.38	0.38	0.39
Frequência média (MHz)	1142	1005	1012

7. Conclusões

A principal conclusão deste trabalho é que a arquitetura hierárquica em conjunto com a heurística 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*. A arquitetura se demonstrou eficaz para a estimação das frequências de operação e suficientemente rápida para aplicações *soft real-time*. Nos experimentos efetuados o tempo de resposta para a solução foi desprezível em relação ao intervalo de 1 segundo entre as intervenções na frequência de operação dos processadores. Se observarmos os resultados obtidos durante a simulação para diferentes configurações de *cluster* (Tabela 2), os resultados, tanto na performance (Tabela 4) quanto na qualidade (Tabela 5), indicam que esta estratégia pode prover uma alternativa escalável para este tipo de aplicação.

Os experimentos demonstraram que a estratégia aqui apresentada foi competitiva em relação as outras abordagens avaliadas, mesmo na configuração de um *cluster* mínimo, e reduziu a frequência média de operação do processador em relação ao *ondemand* do Linux. Esta redução na frequência média permitiu uma economia no consumo de potência e foi alcançada com, na maioria das vezes, um ganho em QoS.

8. Trabalhos futuros

Pretendemos avaliar a arquitetura hierárquica em *clusters* com maiores dimensões, de forma a aprofundarmos os estudos sobre sua performance e qualidade dos resultados. Outra questão é avaliar a heurística implementada em uma arquitetura tradicional, pois os dados indicam que mesmo nesta situação a performance e qualidade nos resultados devem viabilizar sua aplicação.

Uma linha interessante para trabalhos futuros é investigar, como introduzido em [J. Huang et al. 2007], a tendência de redução ou crescimento da carga de processamento em um futuro próximo dentro de um *cluster*. Com esta informação podemos fornecer à arquitetura mais um dado de entrada para ajustar as frequências ou ligar/desligar servidores.

Finalmente, podemos avançar nesta arquitetura através da seleção direcionada dos servidores que compõem os segmentos de *cluster*, de forma a possibilitar que segmentos distintos possam ser tratados de forma diferenciada. Esta diferenciação pode ocorrer devido às 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. Este direcionamento pode ter um forte impacto na economia de energia nos *data centers*, pois cerca de 50% de seu consumo [EPA 2007] é relativo a infraestrutura para os servidores. Tomando como exemplo a refrigeração, se conseguirmos desligar todos os servidores em um mesmo ambiente podemos reduzir o consumo no ar condicionado.

Referências

B.A. Novelli, J.C.B. Leite, J.M. Urriza, and J.D. Orozco (Julho, 2005). Regulagem dinâmica de voltagem em sistemas de tempo real. *SEMISH, São Leopoldo, RS, Brasil*, pages 1772–1786.

E.N. (Mootaz) Elnozahy, M. Kistler, and R. Rajamony (Fevereiro, 2002). Energy-efficient server clusters. *Workshop on Power-Aware Computing Systems*.

EPA (Agosto, 2007). Report to congress on server and data center energy efficiency. *Technical report, Environmental Protection Agency, EUA*.

GLPK (2010). Gnu linear programming kit, free software foudation, gnu. http://www.gnu.org/software/glpk acessado em 12/11/2009.

HTTPERF (2010). httperf documentation. http://www.hpl.hp.com/research/ linux/httperf/docs.php.

J. Huang, H. Jin, X. Xie, and Q. Zhang (Agosto, 2007). Using NARX neural network based load prediction to improve scheduling decision in grid environments. *3rd International Conference on Natural Computation, Wuhan, China*, pages 718–724.

L.Bertini, J.C.B. Leite, and D. Mossé (Julho, 2007). Statistical QoS guarantee and energy-efficiency in web clusters. *19th Euromicro Conference on Real-Time Systems, Pisa, Itália*, pages 83–92.

Linux (2010). Linux kernel cpufreq subsystem. http://www.kernel.org/pub/linux /utils/kernel/cpufreq/cpufreq.html.

L.S. Sousa, J.C.B. Leite, and J.C.S. Souza (2008). Aplicação de redes neurais na construção de servidores web verdes. In *XXXIV Conferência Latinoamericana de Informática, Santa Fe, Argentina*.

M. Elnozahy, M. Kistler, and R. Rajamony (Março, 2003). Energy conservation policies for web servers. *4th USENIX Symposium on Internet Technologies and Systems, Seattle, WA, EUA*.

M.R. Garey and D.S. Johnson (1979). Computers and intractability - a guide to theory of np-completeness. In *Freeman, San Francisco*.

P. Pillai and K. G. Shin (Outubro, 2001). Real-time dynamic voltage scaling for low-power embedded operating systems. *18th Symposium on Operating Systems Principles, Banff, Alberta, Canadá*, pages 89–102.

Rountree, B., Lowenthal, D. K., Funk, S., Freeh, V. W., de Supinski, B. R., and Schulz, M. (2007). Bounding energy consumption in large-scale mpi programs. In *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, pages 1–9, New York, NY, USA. ACM.

Rusu, C., Ferreira, A., Scordino, C., and Watson, A. (2006). Energy-efficient real-time heterogeneous server clusters. In *RTAS '06: Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 418–428, Washington, DC, USA. IEEE Computer Society.

T. Heath, B. Diniz, E.V. Carrera, W. Meira Jr., and R. Bianchini (Junho, 2005). Energy conservation in heterogeneous server clusters. *10th ACM Symposium on Principles and Practice of Parallel Programming, Chicago, IL, EUA*, pages 186–195.