ECO-ALOC: Alocação Energeticamente Eficiente de Recursos para Roteadores de Software em Cluster

Carlo Fragni, Luís Henrique M. K. Costa e Otto Carlos M. B. Duarte*

[1]Grupo de Teleinformática e Automação, PEE/COPPE - DEL/POLI,
Universidade Federal do Rio de Janeiro

{carlo,luish,otto}@gta.ufrj.br

Resumo. *Roteadores de software em cluster são uma abordagem prática para prover a flexibilidade dos roteadores de software superando seus problemas de escalabilidade. Neste trabalho propomos o ECO-ALOC, um mecanismo de alocação de recursos energeticamente eficiente para roteadores de software em cluster. O ECO-ALOC é formado por dois módulos: o primeiro provê economia de energia de alta granularidade através do controle da frequência de operação das CPUs, o segundo provê economia de energia a longo prazo através da migração de redes virtuais para consolidar a carga e desativar servidores ociosos dentro do cluster. Simulações usando dados de tráfego real mostram que o ECO-ALOC provê até 93% de economia de energia.*

Abstract. *Cluster-based software routers are a practical approach to provide the flexibility of software routers and overcome their scalability issue. In this work we propose ECO-ALOC, an energy-efficient resource allocation mechanism for cluster-based software routers. ECO-ALOC has two modules: the first module provides fine-grained energy consumption control by switching CPU operation frequencies, while the second module provides long-term power savings by using virtual network migration to consolidate the load and shut idle servers down inside the cluster. Simulations using real traffic traces show that ECO-ALOC provides power savings of up to 93%.*

1. Introdução

Virtualização de redes é um conceito chave para suportar as abordagens pluralistas de Internet do Futuro [Moreira et al. 2009], possibilitando o compartilhamento de uma mesma infraestrutura física entre múltiplas redes concorrentes e isoladas. Roteadores convencionais priorizam desempenho, mas usam plataformas específicas de hardware e software proprietário, dificultando a experimentação de novas funcionalidades na rede. Roteadores de software, por outro lado, utilizam hardware convencional e são facilmente programáveis, mas não atingem altas taxas de encaminhamento [Dobrescu et al. 2009].

Roteadores de software em cluster superam essas limitações de desempenho ao agregar hardware convencional em clusters. Eles provêem às redes virtuais uma infraestrutura adequada de rede, mas trazem consigo o problema da distribuição dos recursos entre as redes virtuais. Visto que o consumo energético tornou-se um fator limitante para a escalabilidade e crescimento da Internet [Huang et al. 2009], outra vantagem dos roteadores de software em cluster é a possibilidade de gerenciamento mais eficiente do gasto

*Este trabalho foi parcialmente custeado pela CAPES, CNPq, FAPERJ e FINEP/FUNTTEL.

energético através do desligamento e religamento dos servidores do cluster de acordo com a demanda de tráfego.

Este trabalho propõe o ECO-ALOC, um mecanismo de alocação de recursos energeticamente eficiente para roteadores de software em cluster. O ECO-ALOC possui um Mecanismo de Dinâmica Rápida (MDR) que lida, em alta granularidade, com a economia de energia dentro de cada servidor do cluster. O ECO-ALOC também possui um Mecanismo de Dinâmica Lenta (MDL) para lidar com a economia de energia a longo prazo dentro do cluster como um todo. O MDL também é responsável pela distribuição das redes virtuais entre os servidores que compõem o cluster. O MDR proposto, chamado *Efficient/Cautious/Otiose* (ECO), consiste em uma heurística para ajustar o tempo que a CPU gasta em três estados estrategicamente escolhidos. Como MDL, propõe-se o mecanismo *Automatic Load Organizer for Cluster* (ALOC). O ALOC migra roteadores virtuais de servidores sobrecarregados, consolida os roteadores virtuais em servidores mais eficientes do ponto de vista energético e desliga servidores ociosos para economizar energia.

Para avaliar o ECO-ALOC, foi desenvolvido um simulador do roteador de software em cluster. Desenvolveu-se também um MDR alternativo baseado em técnicas de otimização para avaliar o desempenho do ECO. Os resultados mostram que, sob demanda de tráfego real de *backbone*, o mecanismo proposto reduz o consumo de energia em até 93% sem aumentar as perdas de pacotes. No cenário de pior caso, o mecanismo proposto praticamente eliminou as perdas de pacotes, economizou 48% de energia e reduziu em 50% o uso do buffer em relação a nenhum mecanismo.

Este trabalho está organizado como descrito a seguir. A Seção 2 apresenta os fatores de consumo energético de hardwares convencionais, enquanto a Seção 3 apresenta o modelo de roteador de software em cluster considerado neste trabalho. A Seção 4 descreve o ECO-ALOC, enquanto a Seção 5 avalia seu desempenho. Finalmente, a Seção 6 apresenta os trabalhos relacionados e a Seção 7 conclui este artigo.

2. Fatores de Consumo Energético

Para a construção de um roteador de software em cluster que seja energeticamente eficiente, o primeiro passo é identificar que componentes dos hardwares convencionais demandam mais energia e avaliar as possíveis ações para diminuir seu consumo. Os componentes mais utilizados por um roteador de software baseado em hardware convencional, nas tarefas associadas ao encaminhamento de pacotes, são a CPU, adaptador de rede e RAM. A CPU é, de longe, o componente que mais consome energia e foi identificado como gargalo em roteadores de software em cluster [Dobrescu et al. 2009]. Para ilustrar, um servidor convencional da Dell equipado com 2 CPUs Xeon consome 420W sob carga máxima[1] e 220W quando ocioso [ser 2010]. As CPUs Xeon consomem aproximadamente 130W sob carga máxima [xeo 2009] e respondem pela maior parte da variação do gasto energético do servidor entre os estados de ócio e de carga máxima, tornando a CPU um componente chave para a economia energética. O adaptador de rede consome aproximadamente 10W [nic 2008], gasto relativamente baixo, e por isso será ignorado em nosso mecanismo de economia de energia. O consumo energético da RAM não pode ser diretamente controlado. O resto do consumo do servidor remete à placa-mãe, fonte, sistema de refrigeração e periféricos. Visto que seus consumos não

[1]Gasto energético sob carga máxima medido utilizando o *benchmark* de CPU SANDRA Dhrystone (http://www.sisoftware.net/?d=qa&f=ben_cpu&l=en&a=).

podem ser controlados em alta granularidade, eles serão gerenciados como um bloco. Pode-se controlar o gasto energético desse bloco colocando o servidor em *standby*, consumindo aproximadamente 15W, e religando o servidor através da rede local (*wake on LAN*) [Lefevre and Orgerie 2010]. Visto que um servidor ocioso consome em torno de 50% de seu consumo máximo [Vasic et al. 2009], do ponto de vista energético, é melhor ter um servidor operando em 90% de sua capacidade e outro em *standby* do que dois servidores operando em 45% da capacidade.

Combinando os possíveis estados de frequências/voltagens de operação dos núcleos de duas CPUs Xeon, observou-se em [Bolla et al. 2009] uma relação quase linear entre a capacidade de encaminhamento de pacotes e o gasto energético. O trabalho considera 35 estados para a CPU, obtidos combinando os estados de todos seus núcleos, número que garante bom controle sobre a capacidade e consumo do servidor. Dado que a troca de estado da CPU dura poucos μs, o impacto da troca de estado da CPU é negligenciável. Uma CPU Xeon da série 5500, por exemplo, demora no máximo 2μs para efetuar a troca de estado [xeo 2009]. Considerando um enlace de 10Gb/s, no máximo 2685B de dados deixariam de ser processados durante essa troca de estados.

A virtualização provê outras possibilidades para economizar energia em um roteador de software em cluster, como o controle do tamanho dos buffers de recepção e a migração de máquinas virtuais. A migração de roteadores virtuais possibilita balancear a carga e também pode ser usada para controlar os gastos energéticos [Wang et al. 2008]. A migração de roteadores virtuais permite consolidar a carga em um menor número de servidores dentro do cluster em períodos de baixa demanda, tornando possível o desligamento dos servidores ociosos para maximizar a economia de energia. Por outro lado, durante períodos de pico, é possível obter melhor distribuição da carga entre os servidores do cluster. O controle do tamanho de buffers pode ser utilizado para melhorar o desempenho ou economia de energia. Aumentar os buffers permite o uso de um estado de CPU com capacidade inferior aos picos de demanda e com menor gasto energético, visto que o buffer absorve os picos de tráfego. Por outro lado, diminuir o tamanho dos buffers demanda estados de CPU com maior desempenho para atender os picos de tráfego, aumentando o consumo energético mas diminuindo a latência. Para efeito de simplicidade, neste trabalho usa-se um modelo com um único buffer no kernel que recebe todos os pacotes antes do processamento dos mesmos[2].

3. Modelo de Roteador de Software em Cluster

Este trabalho explora a arquitetura dos roteadores de software em cluster como uma forma de atender as demandas de encaminhamento das redes virtuais e, ao mesmo tempo, economizar energia. A Fig. 1(a) exemplifica um roteador formado por seis servidores convencionais. Nele, uma interface de rede de cada servidor é utilizada como porta externa do roteador e as interfaces remanescentes são utilizadas para a comunicação interna do cluster, idéia básica do Routebricks [Dobrescu et al. 2009].

Apesar dessa arquitetura aumentar a capacidade de encaminhamento de pacotes dos roteadores de software, ela prejudica a economia de energia. Não é possível desligar um servidor, visto que isto significaria desconectar a porta externa do roteador pela qual

[2]Na prática, há filas individuais de recepção para cada interface de rede, seguidas dentro do kernel por filas conhecidas como *backlog queues*, uma para cada CPU. O modelo utilizado é conservador, considerando que o buffer unificado possui capacidade de armazenamento equivalente à de uma única fila de recepção de interface de rede.

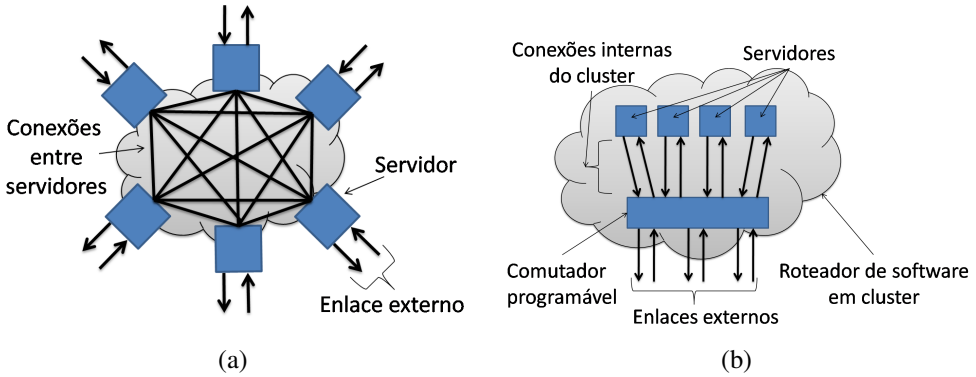


Figura 1. Modelos de roteador de software em cluster.

o servidor desligado é responsável. Por esta razão, propomos uma arquitetura diferente que possibilita o desligamento de servidores sem impactar na disponibilidade das portas externas (Fig. 1(b)). Nessa arquitetura, as portas externas do roteador estão conectadas a um comutador programável, como um comutador Openflow [McKeown et al. 2008] por exemplo, bem como todos os servidores formando o cluster. O comutador é configurado para encaminhar os pacotes de cada rede virtual para o servidor responsável pela mesma. Dessa forma, para desligar um servidor basta migrar os roteadores virtuais que estejam utilizando seus recursos para outros servidores do cluster, e, reconfigurar o comutador para contemplar a nova distribuição dos roteadores virtuais.

4. O Mecanismo ECO-ALOC

O problema de alocação de recursos em um roteador de software em cluster possui duas partes: a alocação da banda disponível nas interfaces de entrada e de saída entre as redes virtuais; e fornecer os recursos necessários para o processamento dos pacotes para cada roteador virtual. O compartilhamento da banda entre as redes virtuais pode ser tratado como compartilhamento de banda entre diferentes classes de serviço e existem diversas propostas para a solução deste problema [He et al. 2008, Cheng and Zhuang 2006]. Este trabalho foca no problema do compartilhamento de recursos associados ao processamento dos pacotes entre as redes virtuais. No escopo do cluster como um todo, o mecanismo proposto escolhe quais redes virtuais são alocadas em cada servidor e quais servidores podem ser desligados para economizar energia. A escolha dos servidores a serem desligados ou religados é efetuada pelo Mecanismo de Dinâmica Lenta (MDL). No escopo de um único servidor, o mecanismo proposto controla os estados de voltagem/frequência de operação da CPU para economizar energia enquanto atende à demanda de processamento imposta pelo tráfego da rede. O controle do estado da CPU utilizado é efetuado pelo Mecanismo de Dinâmica Rápida (MDR).

O primeiro objetivo do ECO-ALOC é diminuir o consumo energético sem prejudicar o encaminhamento de pacotes. O mecanismo possui parâmetros configuráveis que permitem regular a agressividade na economia de energia, permitindo priorizar latência ou economia de energia. Outro objetivo é facilitar a manutenção do cluster, possibilitando a retirada ou adição de um servidor ao cluster de forma transparente para as redes virtuais.

O Gerenciador de Recursos de Processamento de Pacotes

O mecanismo proposto aloca recursos baseado em dois tipos de ações. O primeiro tipo é formado por ações que atuam em uma escala de tempo de frações de segundo e é

Algorithm 1 Heurísticas do MDR ECO

Require: $T \in \mathbb{N}$, o tamanho do intervalo de tempo; e $D \in \mathbb{N}$, a demanda estimada de encaminhamento de pacotes dentro do intervalo de tempo;

Ensure: $t_e, t_c, t_o \in \mathbb{N}$, os intervalos de tempo gastos nos estados *Efficient, Cautious* e *Otiose* da CPU, respectivamente;

1: ———————————————————————————

2: Sejam $C_e, C_c \in \mathbb{N}$ as capacidades de encaminhamento de pacotes nos estados *Efficient* e *Cautious* da CPU, respectivamente;

3: $t_e, t_o, t_c \leftarrow 0$; // Inicialização

4: **if** $(D < C_e.T)$ **then**

5: // c_{eff} é suficiente para atender à demanda

6: $t_e \leftarrow \lceil D/C_e \rceil$;

7: $t_o \leftarrow T - t_e$; //A CPU pode gastar algum tempo ociosa

8: **else if** $(D > C_c.T)$ **then**

9: // A demanda é maior do que c_{caut} pode atender

10: $t_c \leftarrow T$;

11: **else**

12: // A demanda pode ser atendida combinando c_{eff} e c_{caut}

13: $t_e \leftarrow \lceil \big(C_c.T - D\big)/(C_c - C_e) \rceil$;

14: $t_c \leftarrow T - t_e$.

15: **end if**

utilizado pelo Mecanismo de Dinâmica Rápida (MDR). Já o segundo tipo é formado por ações que atuam em uma escala de tempo mais lenta, de diversos segundos ou até minutos, e é utilizado pelo Mecanismo de Dinâmica Lenta (MDL). O MDR chaveia os estados da CPU, enquanto o MDL migra roteadores virtuais e desliga ou religa servidores do cluster. Ambos os mecanismos planejam suas ações periodicamente para um intervalo de tempo, T, baseados em uma demanda estimada de encaminhamento de pacotes para o intervalo, D. Cada mecanismo possui T e D próprios, visto que atuam em escalas de tempo e com frequências diferentes. Para obter D, multiplica-se T pela demanda média de encaminhamento de pacotes no intervalo de tempo anterior. Para adicionar uma margem de segurança, soma-se o desvio padrão ao valor da demanda média.

4.1. Mecanismo de Dinâmica Rápida (MDR)

O MDR orquestra os estados da CPU para economizar energia ao mesmo tempo em que atende a demanda de processamento das redes virtuais. O MDR não se preocupa com o consumo de RAM das redes virtuais, visto que considera-se o uso de quantidade fixa de RAM pelos roteadores virtuais. Dessa forma, a preocupação com a RAM recai sobre o MDL, capaz de gerenciar o consumo de RAM através da migração de roteadores virtuais. É importante lembrar que a troca de frequencia/voltagem de operação da CPU demora poucos μs e, como o MDR troca poucas vezes o estado de operação da CPU em um período de centenas de ms, o impacto da troca de estados é negligenciável. O mecanismo proposto para o MDR é o *Efficient/Cautious/Otiose (ECO)*.

O MDR-ECO está descrito no Algoritmo 1. ECO usa três estados da CPU: *Efficient, Cautious* e *Otiose*. O estado *Efficient* é aquele com menor consumo energético por pacote encaminhado. O estado *Cautious* é o que possui a mais alta capacidade de encaminhamento. O estado *Otiose* é o que apresenta o mais baixo consumo energético, não encaminha pacotes e consome CPU somente para manutenção do Sistema Operacional. A idéia chave é economizar a máxima quantidade de energia utilizando o estado mais eficiente da CPU para encaminhar pacotes. O tempo gasto pela CPU em cada estado é escolhido baseado na demanda de encaminhamento estimada para o intervalo de tempo,

Algorithm 2 MDL-ALOC

Require: Estado atual do cluster e demandas estimadas das redes virtuais;
Ensure: Planos para consolidar as redes virtuais, desligar e religar servidores;
1: ———————————————————————
2: **for** cada servidor sobrecarregado no cluster **do**
3: $planoA \leftarrow$ "migre redes virtuais até não estar mais sobrecarregado";
4: **end for**
5: **for** cada servidor s na lista de servidores ordenados por ineficiência energética **do**
6: **for** cada servidor r na lista de servidores ordenados por eficiência energética **do**
7: **if** $(s = r)$ **then**
8: Pare o loop interno;
9: **else**
10: $planoA \leftarrow$ "migre redes virtuais para r sem sobrecarregá-lo";
11: **end if**
12: **end for**
13: **end for**
14: $planoB \leftarrow$ "desligue/religue servidores ociosos de acordo com o IDT e com o $planoA$";
15: Retorne $planoA$ e $planoB$.

D, o tamanho do intervalo de tempo, T, e a capacidade de encaminhamento dos estados *Efficient* e *Cautious*, C_e e C_c, respectivamente. Para obter maior economia de energia, o MDR-ECO configura a CPU para o estado *Otiose* assim que a CPU tenha passado tempo suficiente no estado *Efficient* para suprir a demanda estimada (linhas 4-6 do Algoritmo 1). Caso a demanda estimada seja superior à capacidade máxima da CPU, o MDR-ECO mantém a CPU no estado *Cautious* durante todo o intervalo de tempo para minimizar as perdas (linhas 7-8 do Algoritmo 1). Caso a demanda estimada seja inferior à capacidade de encaminhamento da CPU do estado *Cautious* mas superior à do estado *Efficient*, o MDR-ECO utiliza ambos para atender à demanda (linhas 9-12 do Algoritmo 1).

É importante perceber que o MDR-ECO utiliza ao máximo dois estados da CPU por intervalo de tempo, pois não é razoável utilizar o estado *Otiose* em situações de alta demanda ou o estado *Cautious* em períodos de baixa demanda. O MDR-ECO é sensível ao tamanho do buffer utilizado: aumentar o buffer possibilita avaliar a demanda das redes virtuais com menor frequência, provendo maior economia de energia em troca de aumento da latência, enquanto diminuir o buffer causa os efeitos contrários.

4.2. Mecanismo de Dinâmica Lenta (MDL)

O MDL atua no cluster como um todo, economizando energia ao consolidar as redes virtuais em alguns servidores e desligando os servidores ociosos. O mecanismo *Automatic Load Organizer for Cluster (ALOC)* é proposto para o MDL. O MDL-ALOC é periodicamente executado, com frequência muito inferior ao MDR, e possui três etapas para decidir as ações de controle a serem executadas (Algoritmo 2).

A primeira etapa (linha 2 do Algoritmo 2) elimina a sobrecarga dos servidores. Para isso, o MDL-ALOC verifica se há servidores ligados com uso de CPU acima do limiar chamado *overload threshold (OVT)*. Se houver, o MDL-ALOC cria um primeiro plano migrando redes virtuais de servidores sobrecarregados para outros com capacidade ociosa, preocupando-se em não sobrecarregar nenhum servidor de destino (linha 3 do Algoritmo 2). A segunda etapa consolida a carga nos servidores mais eficientes do cluster. A eficiência dos servidores é definida por um índice que considera os três estados da CPU

utilizados pelo MDR-ECO. O índice de eficiência do servidor, I_e, é definido pela equação

$$I_e = \left((P_e/C_e).(P_c/C_c).(P_o)\right)^{-1} \quad (1)$$

onde P_e, P_c e P_o são as potências de consumo com a CPU nos estados *Efficient*, *Cautious* e *Otiose*, respectivamente, enquanto C_e e C_c são as capacidades de encaminhamento de pacotes nos estados *Efficient* e *Cautious*, respectivamente. Nesse passo, o MDL-ALOC varre todos os servidores ligados partindo do menos eficiente para o mais eficiente (linha 5 no Algoritmo 2). Para cada servidor, o MDL-ALOC efetua um segundo plano para migrar as redes virtuais do servidor para servidores mais eficientes (linhas 6-12 do Algoritmo 2). Ao final da segunda etapa, o MDL-ALOC possui um plano com as redes virtuais consolidadas nos servidores mais eficientes, deixando os servidores mais ineficientes ociosos. A terceira etapa decide quais servidores ociosos desligar para economizar energia. Nessa etapa, o MDL-ALOC verifica o uso de CPU do último servidor com redes virtuais alocadas. Caso o uso de CPU esteja acima do limiar definido como *idle threshold (IDT)*, o MDL-ALOC mantém ligado o servidor ocioso de melhor eficiência energética e desliga os servidores ociosos restantes. Caso contrário, todos os servidores ociosos são desligados (linha 14 do Algoritmo 2). A decisão de deixar um servidor ocioso ligado serve para absorver possíveis aumento inesperados no tráfego, visto que religar um servidor é um processo demorado. É importante notar que o MDL-ALOC não efetua nenhuma migração de redes virtuais até o fim do planejamento de três etapas para evitar efetuar migrações desnecessárias. O MDL-ALOC também é a parte do mecanismo proposto responsável pelo suporte à retirada ou adição de servidores do cluster de forma transparente. Para a retirada, basta configurar manualmente o índice de eficiência do servidor desejado para um valor muito baixo. Dessa forma, todas as redes virtuais deste servidor serão migradas para outros servidores e o MDL-ALOC irá desativar o servidor em questão. Para a adição, basta informar ao MDL-ALOC sobre a inclusão dos novos servidores no cluster e estes serão automaticamente utilizados na próxima execução do MDL.

5. Avaliação de Desempenho do ECO-ALOC

Inicialmente, esta seção o simulador desenvolvido, seguido pelo detalhamento de um MDR alternativo desenvolvido para melhor avaliar o desempenho do ECO. Depois, testes preliminares para sintonia de parâmetros dos mecanismos testados são apresentados. Finalmente, apresentam-se testes efetuados com dados de 24 horas de tráfego real para mostrar a eficiência do ECO-ALOC.

5.1. O Simulador de Roteador de Software em Cluster

O simulador de roteador de software em cluster foi desenvolvido para avaliar as técnicas propostas, criando um cluster com o número desejado de servidores. Cada servidor possui uma CPU com um conjunto de estados retirados de [Bolla et al. 2009], com capacidade de encaminhamento e consumos energéticos diferentes para CPU ativa ou ociosa. Os servidores também possuem um buffer de recepção com capacidade de armazenamento de 125.000 pacotes, valor típico de buffer utilizado em enlaces do *backbone* de 2.5Gb/s considerando um tamanho médio de pacote de 500 bytes [Beheshti et al. 2008]. Cada rede virtual possui um perfil de tráfego e a RAM necessária. Os perfis de tráfego são compostos por classes de tráfego HTTP, HTTPS, FTP e e-mail extraídas de dados de roteadores do *backbone* obtidos através da CAIDA [Walsworth et al.] e da rede Ipê da RNP [ipe]. O simulador cria as redes virtuais e usa *round-robin* para escolher o perfil de tráfego associado a cada rede virtual. A distribuição inicial das redes virtuais entre os

servidores do cluster também segue o mesmo critério. Cada passo de simulação corresponde a 10ms de tempo e consiste em computar o novo estado (CPU, memória, buffer e consumo energético) de cada servidor do cluster e executar os mecanismos de alocação de recursos. O consumo de CPU e memória são calculados de acordo com a demanda das redes virtuais e com as sobrecargas geradas por eventuais migrações. As migrações duram 3s ao todo, geram sobrecarga na CPU dos servidores de origem e destino equivalente a 8% da capacidade da CPU e alocam a quantidade de RAM requerida pelo roteador virtual no servidor de destino, parâmetros baseados em [Wang et al. 2008]. A atualização do estado dos buffers considera que os pacotes que não puderam ser processados no passo de simulação ficam armazenados no buffer, e, caso o mesmo esteja cheio, os pacotes excedentes são descartados. O gasto energético de servidores ligados é computado de acordo com o estado da CPU. A execução dos mecanismos de alocação de recursos envolve o planejamento e controle dos estados da CPU (dinâmica rápida); e as ações de longo prazo (dinâmica lenta), como a migração de roteadores virtuais, o desligamento ou religamento de servidores. Um servidor demora 110s para ser religado e 10s para ser desligado, tempos durante os quais o servidor não processa nenhum pacote e possui consumo igual ao estado ativo da CPU de mais baixo consumo [Lefevre and Orgerie 2010]. Após o desligamento, os servidores entram em *standby* e consomem 15W.

5.2. MDR Alternativo

Para melhor avaliar o desempenho do MDR-ECO, um MDR alternativo foi desenvolvido baseada em otimização. Primeiro, define-se uma função custo para modelar o problema da escolha de estados da CPU e economia de energia. A função custo deve apresentar um termo que modele a necessidade de prover poder de processamento suficiente para atender a demanda de tráfego. Visto que este termo tende a maximizar a capacidade de encaminhamento da CPU e, por consequência, maximizar o gasto energético, é necessário definir um termo para penalizar o gasto energético. Dessa forma, foi definida uma função de custo, $\Phi(\mathbf{t})$, para ser minimizada. $\Phi(\mathbf{t})$, mostrada em (2), é a soma de três termos, cada um multiplicado por uma constante (α, β e γ), e depende de $\mathbf{t}$, vetor $2N$-dimensional de intervalos de tempo, em segundos, gasto em cada estado i ativo ou ocioso da CPU, com $i \in \{1 \dots N\}$. O primeiro termo, $E(\mathbf{t})$, modela o consumo energético do servidor, em Joules (3). $E(\mathbf{t})$ contabiliza a energia gasta, para todos os N estados da CPU, pelo servidor enquanto ocioso, $P_i^{ocioso}.t_i^{ocioso}$, ou encaminhando pacotes (ativo), $P_i^{ativo}.t_i^{ativo}$, onde P_i^{ocioso} e P_i^{ativo} são as potências, em Watts, do servidor ocioso ou ativo com a CPU no estado i, respectivamente.

$$\Phi(\mathbf{t}) = \alpha.E(\mathbf{t}) + \beta.S(\mathbf{t}) + \gamma.R_{tempo}(\mathbf{t}) \tag{2}$$

$$E(\mathbf{t}) = \sum_{i=1}^{N} \left(P_i^{ocioso}.t_i^{ocioso} + P_i^{ativo}.t_i^{ativo}\right) \tag{3}$$

$$S(\mathbf{t}) = \left(\left(\sum_{i=1}^{N} C_i.t_i^{ativo}\right) - D \right)^{-1} \tag{4}$$

$$R_{tempo}(\mathbf{t}) = \left(T - \sum_{i=1}^{N} (t_i^{ocioso} + t_i^{ativo}) \right)^2 \tag{5}$$

O segundo termo, $S(\mathbf{t})$, modela a capacidade ociosa de encaminhamento, e é definido como o inverso da diferença entre a soma da capacidade individual de encaminhamento

de cada estado da CPU, $C_i.t_i^{ativo}$, e a demanda total de tráfego das redes virtuais no servidor, D, em pacotes, como mostrado em (4). A capacidade total de encaminhamento depende de quanto tempo foi gasto em cada estado da CPU com o servidor ativo, t_i^{ativo}, e a capacidade de encaminhamento da CPU naquele estado, C_i, em pacotes por segundo. O termo final de $\Phi(\mathbf{t})$, $R_{tempo}(\mathbf{t})$, garante que a soma dos valores de tempo escolhidos correspondam ao tamanho do intervalo de tempo T. As constantes em cada um dos termos de $\Phi(\mathbf{t})$ são responsáveis por cancelar as unidades de cada termo e servem para configurar a otimização. Aumentar α prioriza a economia de energia, enquanto aumentar β estimula o aumento de capacidade ociosa, melhorando o desempenho e aumentando o gasto energético. Finalmente, γ determina o quão restritiva é a otimização ao escolher os intervalos de tempo que compõe T.

Para executar a otimização foram consideradas as técnicas de otimização por força bruta e *Simulated Annealing* (SA). A técnica de força bruta provê os melhores resultados quando exaustivamente executada, mas apresenta tempo de convergência extremamente alto, enquanto o *Simulated Annealing* apresenta bons resultados com tempo de convergência bem menor. Foram feitas simulações com um servidor e oito redes virtuais. A técnica de força bruta testa todas as possíveis configurações da CPU dentro de T, sendo seu tempo de execução proporcional a N^T, facilmente atingindo tempos de execução inviáveis. Por esta razão, utilizou-se os mesmos 3 estados da CPU usados pelo MDR-ECO e intervalo de tempo T de 10 passos de simulação. Para o *Simulated Annealing* foram feitos testes com os mesmos três estados e também com todos os 35 estados da CPU considerados em nosso simulador. O algoritmo de força bruta apresentou consumo médio de 325,4 W, utilizando em média 12.220,5 pacotes do buffer e demorando 182s para a execução total da sua simulação. O algoritmo SA, utilizando os mesmos 3 estados da CPU, consome em média 329,4 W, usando 9.712,5 pacotes em média no buffer e demorou ao todo 4s. Ao rodar o algoritmo SA com todos os 35 estados da CPU, o consumo médio caiu para 306,8 W, aumentou o uso médio do buffer para 16.482,7 pacotes e o tempo de execução não sofreu alteração. O tempo de execução do algoritmo de *Simulated Annealing* depende dos parâmetros avaliados na subseção 5.3. Também testamos o comportamento do sistema sem MDR, com a CPU sempre no estado de maior poder de processamento, obtendo 356 W de consumo médio, 0 pacotes em média no buffer e tempo de execução de 3s. Visto que o melhor resultado para o algoritmo de *Simulated Annealing* foi obtido usando todos os estados da CPU e isto não aumentou notavelmente o tempo de execução, esta configuração será tomada como padrão para os próximos testes.

5.3. Sintonia de Parâmetros do MDR-SA

O *Simulated Annealing* (SA) funciona perturbando as variáveis de entrada, definidas no caso como o vetor $2N$-dimensional $\mathbf{t}$ de intervalos de tempo, com o objetivo de encontrar a configuração que minimiza a função de custo, $\Phi(\mathbf{t})$. O SA é um algoritmo iterativo que inicia com uma configuração arbitrária das variáveis de entrada e, a cada iteração, adiciona uma perturbação à configuração anterior, avaliando se a perturbação reduz o erro de convergência. O SA possui dois parâmetros: temperatura, que controla a probabilidade de aceitar uma nova configuração que aumenta o erro de convergência; e o número de iterações por temperatura. A temperatura é alta no início da execução do SA para evitar convergir para um mínimo local, e decai ao longo de sua execução. São simulados cinco servidores com duas redes virtuais em cada um, por 400 passos, com o MDR-SA executado a cada 20 passos e tráfego das redes virtuais baseado nos dados da CAIDA. O número de temperaturas e de iterações assume os valores 10, 100 e 1000, totalizando

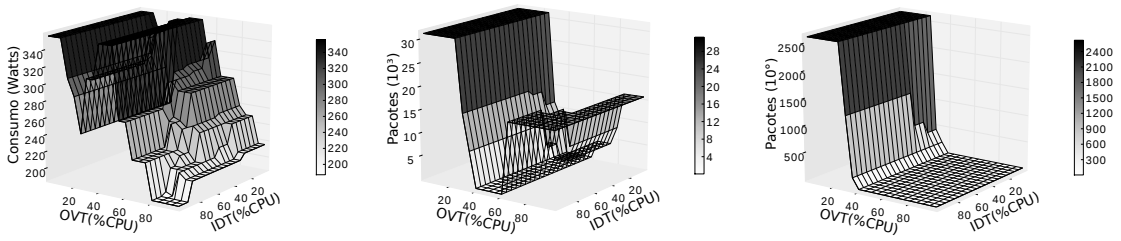
9 configurações diferentes. Para cada par (temperatura, iteração), simulações foram executadas até obter desvio padrão inferior a 10% no valor médio da potência consumida. Os resultados mostram que o MDR-SA foi capaz de economizar quase 20% de energia quando comparado ao consumo de 356W dos servidores sem o uso de mecanismos de alocação de recursos. Todas as configurações de (temperatura, iteração) praticamente não usaram os buffers, utilizando em média em torno de 5% da capacidade máxima. Como resultados similares de desempenho foram obtidos para todas as configurações testadas, a configuração de (temperatura, iteração) usando os valores (10, 10) será utilizada para os próximos testes, pois esta configuração é a que exige menor esforço computacional.

5.4. Sintonia de Parâmetros do MDL-ALOC

Para analisar o impacto dos parâmetros OVT e IDT no desempenho do MDL-ALOC, simula-se um cluster com 8 servidores e 40 redes virtuais por 150.000 passos de simulação e utilizando os dados de tráfego da CAIDA. Os valores de OVT e IDT variam de 0% até 100% com passos de 5% e o MDL-ALOC é executado a cada 30.000 passos. O número de servidores simula um cluster com alguns servidores com alta carga de redes HTTP e outros com baixa carga de redes de e-mail, FTP e HTTPS.

Os resultados são exibidos na Fig. 2. Em geral, aumentar os valores de OVT e IDT diminui o gasto energético conforme esperado. Valores de OVT acima de 40% possibilitam alta consolidação das redes virtuais, gerando maior economia de energia (Fig. 2(a)). O parâmetro IDT é altamente dependente do valor do OVT, visto que o OVT controla o nível de consolidação das redes virtuais, e, dessa forma, o número de servidores ociosos. Altos valores de IDT possibilitam mais frequentemente o desligamento de servidores ociosos, provendo maior economia de energia mas reduzindo o poder de reação do cluster a aumentos no tráfego. Quanto ao uso médio do buffer (Fig. 2(b)), valores de OVT acima de 20% permitem melhor realocação das redes virtuais de servidores sobrecarregados, diminuindo assim o uso do buffer. Como previamente explicitado, a distribuição inicial das redes virtuais é feita utilizando *round-robin*. Visto que o número de servidores em nossos testes é múltiplo do número de perfis de tráfego, todos os servidores recebem apenas um tipo de tráfego. Neste cenário, dois servidores receberam somente tráfego HTTP, outros dois somente HTTPS, e assim por diante com FTP e e-mail. Posto que o volume de tráfego HTTP é mais de 20 vezes maior que os tráfegos HTTPS, FTP e de e-mail, quando o valor do OVT é inferior a 30%, o MDL-ALOC não consegue migrar redes virtuais com HTTP, explicando a queda inicial de gasto energético para OVTs abaixo de 25%. Com OVT entre 30% e 40%, o MDL-ALOC passa a migrar redes virtuais HTTP, conseguindo retirar a sobrecarga dos servidores nos quais estavam inicialmente concentradas. Apesar do aumento no gasto energético, retirar a sobrecarga desses servidores gera grande redução no uso do buffer. É relevante notar que para valores de OVT e IDT acima de 60% e 50%, respectivamente, o uso do buffer aumenta por causa da baixa capacidade ociosa nos servidores, insuficiente para lidar com picos na demanda. A perda de pacotes (Fig. 2(c)), só ocorre nos servidores que estão inicialmente sobrecarregados com redes HTTP até que o menor valor de OVT que permite migrar redes HTTP é atingido. Após este valor ser atingido, o MDL-ALOC é capaz de eliminar a sobrecarga nesses servidores. O MDL-ALOC consegue economizar 48% de energia quando utilizando OVT e IDT iguais a 100% neste cenário com 8 servidores.

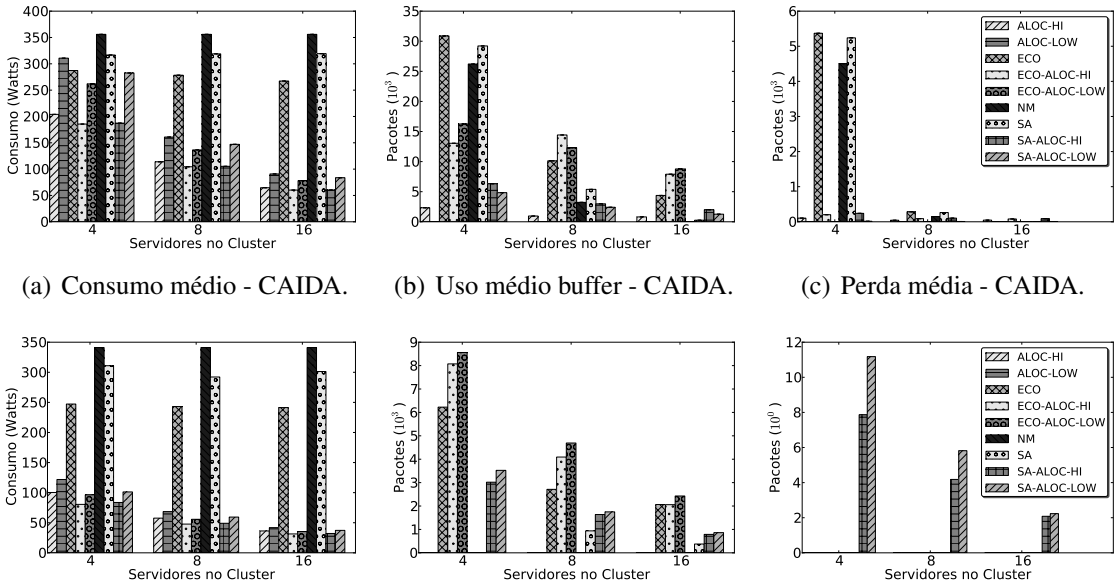
Também foram efetuados testes com 4 e 16 servidores no cluster, omitidos por falta de espaço. O cenário com 4 servidores é de sobrecarga e não possibilita o desligamento de servidores, fazendo com que o IDT não influencie. Entretanto, aumentar o OVT



(a) Potência média consumida. (b) Uso médio do buffer. (c) Perda média de pacotes.

Figura 2. Sintonia de parâmetros do MDL.

possibilita melhor distribuição da carga, diminuindo progressivamente as perdas e uso do buffer até não mais haver perdas. Já o cenário com 16 servidores apresenta abundância de recursos e o comportamento do ALOC é similar ao cenário com 8 servidores. As diferenças limitam-se à maior economia de energia e ao deslocamento da anomalia na diminuição do gasto energético para valores próximos a 20% de OVT devido à maior disponibilidade de recursos. Nos próximos testes com o MDL-ALOC duas configurações serão utilizadas: com (100% OVT , 100% IDT), a mais agressiva na economia de energia; e com (60%OVT, 65% IDT), por ter baixo uso de buffer e boa economia de energia.



(a) Consumo médio - CAIDA. (b) Uso médio buffer - CAIDA. (c) Perda média - CAIDA.

(d) Consumo médio - RNP. (e) Uso médio buffer - RNP. (f) Perda média - RNP.

Figura 3. Avaliação dos mecanismos.

5.5. Avaliação do ECO-ALOC

Nesta subseção avaliam-se o MDR-ECO, o MDR-SA e o MDL-ALOC tanto individualmente quanto combinados. O número de servidores no cluster simula três situações diferentes. O cenário de sobrecarga utiliza 4 servidores. O cenário com 8 servidores possui 2 inicialmente sobrecarregados com redes virtuais HTTP e os outros 6 com baixa carga de redes HTTPS, FTP e de e-mail. O cenário com 16 servidores apresenta abundância de recursos. As simulações utilizam 40 redes virtuais, duram 8.600.000 rodadas, equivalente a 24 horas, e utilizam perfis de tráfego baseados tanto nos dados provenientes da CAIDA quanto da RNP, originados de um enlace de 10Gb entre Rio de Janeiro e Minas Gerais. Os resultados estão na Fig 3. A sigla "NM" refere-se às simulações onde não

se utiliza nenhum mecanismo e, por simplicidade omitimos as siglas “MDR” e “MDL” na identificação dos mecanismos. As siglas “HI” e “LOW” remetem ao MDL-ALOC utilizando (100% OVT, 100% IDT) e (60% OVT, 65% IDT), respectivamente.

Considerando as simulações com os dados da CAIDA (Figs. 3(a), 3(b) e 3(c)), os resultados com 4 servidores mostram que o MDR-ECO economiza mais energia que o MDR-SA, quando comparados com o uso de nenhum mecanismo (NM), 19% e 11% respectivamente. O MDR-ECO utiliza 5% mais o buffer do que o MDR-SA e perde 2% a mais de pacotes que o MDR-SA neste cenário sobrecarregado de pior caso. Comparando com NM, tanto o ECO quanto o SA perdem em torno de 19% a mais de pacotes. Isto deve-se ao fato de tanto o ECO quanto o SA basearem suas escolhas em uma demanda estimada, que, neste cenário de sobrecarga, não contava com abundância de buffer para absorver erros nas estimativas. O MDL-ALOC foi capaz de redistribuir a carga dentro do cluster, diminuindo o consumo energético para ambas as configurações LOW e HI. Além disso, conseguiu reduzir drasticamente o uso do buffer e praticamente eliminar as perdas de pacotes. Combinando os MDRs com o MDL-ALOC continua provendo quase nenhuma perda de pacote, mas consegue aumentar a economia de energia em troca de maior uso do buffer. Finalmente, ECO-ALOC usando a configuração agressiva de (100% OVT, 100% IDT) praticamente elimina a perda de pacotes, economiza 48% de energia e usa 50% menos o buffer em comparação a nenhum mecanismo.

No cenário com 8 servidores, as duas configurações do ALOC-SDM continuam redistribuindo a carga apropriadamente. A configuração HI economizou 30% a mais de energia que a configuração LOW, mas utiliza frequentemente o buffer enquanto a configuração LOW quase não o utiliza. As duas técnicas de MDR foram capazes de reduzir o consumo energético em relação a nenhum mecanismo, mas utilizaram mais o buffer e perderam um pouco mais de pacotes nos servidores sobrecarregados com tráfego HTTP. ECO-ALOC consegui reduzir ainda mais o consumo de energia em relação ao ALOC sozinho ao custo de usar mais o buffer. ECO-ALOC também economizou mais energia do que a combinação SA-ALOC, utilizando mais o buffer para fazê-lo.

Finalmente, com 16 servidores há abundante quantidade de recursos e não há servidores inicialmente sobrecarregados. O MDL-ALOC consegue consolidar apropriadamente as redes virtuais, economizando entre 42% e 46% de energia nas configurações individuais e combinado com ECO e SA em relação às mesmas combinações de mecanismos no cenário com 8 servidores. ECO-ALOC-HI conseguiu economizar 83% de energia e o ECO-ALOC-LOW 78% em relação a nenhum mecanismo.

Nas simulações com dados da RNP (Figs. 3(d), 3(e) e 3(f)), pode-se observar que o MDL-ALOC continua eficiente na reorganização da carga dentro do cluster, tanto para eliminar a sobrecarga dos servidores quanto para economizar energia. A troca básica entre economia de energia e uso de buffer ainda pode ser observada ao modificar a configuração de (OVT, IDT) ou ao se combinar o ALOC com os MDRs. No cenário com 16 servidores, ECO-ALOC-HI é capaz de economizar 93% de energia. Também efetuamos testes com um enlace de 10Gb entre Rio de Janeiro e São Paulo, obtendo o mesmo tipo de comportamento nos mecanismos avaliados. Os resultados foram omitidos por falta de espaço. No cenário com 16 servidores, ECO-ALOC-HI economizou 83% de energia, menos que no enlace com Minas Gerais devido à maior taxa de utilização do enlace com São Paulo.

Os testes desta subseção mostram que ECO-ALOC lida bem com a reorganização da carga dentro do cluster em cenários de sobrecarga ao mesmo tempo em que obtém

boa economia de energia quando o tráfego está baixo. Os parâmetros de OVT e IDT são mais ou menos efetivos dependendo da situação do cluster, merecendo investigação sobre formas de controlá-los dinamicamente.

6. Trabalhos Relacionados

Routebricks [Dobrescu et al. 2009] analisa o desempenho de roteadores de software em cluster construídos a partir de servidores convencionais x86, provando a viabilidade de seu uso no *backbone* da Internet. DaVinci [He et al. 2008] estuda o problema de alocação de recursos em arquiteturas para a Internet do Futuro baseadas em virtualização de rede, mas foca na alocação dinâmica de banda entre as redes virtuais. DaVinci não leva em consideração os recursos de processamento dos pacotes, sendo complementar a este trabalho. DaVinci também não considera a migração de roteadores virtuais ou a problemática energética. Bufferizar pacotes para permitir o uso de estados de mais baixo consumo energético também é explorado em [Nedevschi et al. 2008], a técnica, porém, é utilizada nas interfaces de rede e não no elemento de processamento do roteador, como ocorre neste trabalho. Dessa forma, o mecanismo de Nedevschi *et al.* poderia ser combinado ao ECO-ALOC para aumentar ainda mais a economia de energia. A distribuição de cargas entre os servidores de um cluster visando desligar servidores para a economizar energia também é utilizada em [Niyato et al. 2009], mas o mecanismo por eles proposto explora o adiar o processamento de carga, recurso inviável em ambientes de rede. A troca de estados da CPU em roteadores de software também é explorada em [Bolla et al. 2009]. Para isto, Bolla *et al.* executam um algoritmo de força bruta a cada 10 minutos. Como o ECO foi desenvolvido com heurísticas computacionalmente leves, pode executar a cada 100ms, 6000 vezes mais frequentemente, adaptando a capacidade de processamento da CPU a valores muito mais próximos da demanda instantânea. Em [Wood et al. 2007], é proposto um mecanismo de balanceamento de carga baseado em migração para lidar com níveis intensos de carga em cluster, mas o mecanismo não considera a questão energética.

7. Conclusões e Trabalhos Futuros

Neste trabalho explorou-se o uso dos roteadores de software em cluster para atender a demanda de processamento dos pacotes de redes virtuais ao mesmo tempo em que economiza-se energia. Para tal, propôs-se ECO-ALOC, um mecanismo energeticamente eficiente para a alocação de recursos do cluster entre as redes virtuais. ECO-ALOC separa a dinâmica de controle em duas, de acordo com a escala de tempo das ações utilizadas. Tanto o ECO quanto o ALOC foram avaliados individualmente e combinados. Uma abordagem baseada em *Simulated Annealing* foi desenvolvida para melhor avaliar o desempenho do ECO. Em um cenário com abundância de recursos, ECO-ALOC economizou em média 93% de energia utilizando altos valores para OVT e IDT. Como trabalhos futuros, pretende-se explorar a economia de energia em componentes do hardware de menor consumo energético e investigar um mecanismo para configuração dinâmica dos parâmetros do ECO-ALOC para reconfiguração de acordo com as condições da rede e condições impostas por acordos de QoS com as redes virtuais.

Referências

Rede Ipê da RNP, http://www.rnp.br/en/backbone/index.php.

(2008). *Intel PRO/1000 PT Quad Port Server Adapter Product Brief*.

(2009). *Intel Xeon Processor 5500 Series Datasheet*.

(2010). *Dell PowerEdge R710 Energy Star Datasheet*.

Beheshti, N., Ganjali, Y., Ghobadi, M., McKeown, N., and Salmon, G. (2008). Experimental study of router buffer sizing. Em *ACM SIGCOMM IMC*, páginas 197–210.

Bolla, R., Bruschi, R., Davoli, F., and Ranieri, A. (2009). Energy-aware performance optimization for next-generation green network equipment. Em *ACM SIGCOMM PRESTO*, páginas 49–54. ACM.

Cheng, Y. and Zhuang, W. (2006). Dynamic inter-SLA resource sharing in path-oriented differentiated services networks. *IEEE/ACM Trans. Netw.*, páginas 657–670.

Dobrescu, M., Egi, N., Argyraki, K., Chun, B.-G., Fall, K., Iannaccone, G., Knies, A., Manesh, M., and Ratnasamy, S. (2009). RouteBricks: Exploiting parallelism to scale software routers. Em *ACM SOSP*.

He, J., Zhang-Shen, R., Li, Y., Lee, C.-Y., Rexford, J., and Chiang, M. (2008). Davinci: Dynamically adaptive virtual networks for a customized internet. Em *ACM CoNext*.

Huang, S., Seshadri, D., and Dutta, R. (2009). Traffic grooming: A changing role in green optical networks. Em *IEEE GLOBECOM*, páginas 1–6.

Lefevre, L. and Orgerie, A.-C. (2010). Designing and evaluating an energy efficient cloud. *J. Supercomput.*, 51(3):352–373.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: enabling innovation in campus networks. *ACM SIGCOMM Comput. Commun. Rev.*, páginas 69–74.

Moreira, M. D. D., Fernandes, N. C., Costa, L. H. M. K., and Duarte, O. C. M. B. (2009). Internet do Futuro: Um Novo Horizonte. *Minicursos do Simpósio Brasileiro de Redes de Computadores - SBRC'2009*, páginas 1–59.

Nedevschi, S., Popa, L., Iannaccone, G., Ratnasamy, S., and Wetherall, D. (2008). Reducing network energy consumption via sleeping and rate-adaptation. Em *USENIX NSDI*, páginas 323–336.

Niyato, D., Chaisiri, S., Lee Bu Sung, S., and Sung, L. B. (2009). Optimal power management for server farm to support green computing. Em *IEEE/ACM CCGRID*, páginas 84–91.

Vasic, N., Barisits, M., Salzgeber, V., and Kostic, D. (2009). Making cluster applications energy-aware. Em *ACM ACDC*, páginas 37–42.

Walsworth, C., Aben, E., Claffy, K., and Andersen, D. The CAIDA Anonymized 2009 Internet Traces - 09/17/2009.

Wang, Y., Keller, E., Biskeborn, B., van der Merwe, J., and Rexford, J. (2008). Virtual routers on the move: live router migration as a network-management primitive. *ACM SIGCOMM Comput. Commun. Rev.*, páginas 231–242.

Wood, T., Shenoy, P., Venkataramani, A., and Yousif, M. (2007). Sandpiper: Black-box and gray-box resource management for virtual machines. Em *USENIX NSDI*.