UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

Arquitetura de um *cluster* computacional de baixo consumo e com proporcionalidade energética

Sebastião Emidio Alves Filho

Orientador: Prof. Dr. Aquiles Medeiros Filgueira Burlamaqui

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e de Computação da UFRN (área de concentração: Engenharia de Computação) como parte dos requisitos para obtenção do título de Doutor em Ciências.

Número de ordem PPgEEC: D208
Natal, RN, dezembro de 2017

Universidade Federal do Rio Grande do Norte - UFRN
Sistema de Bibliotecas - SISBI
Catalogação de Publicação na Fonte. UFRN - Biblioteca Central Zila Mamede

Alves Filho, Sebastiao Emidio.
Arquitetura de um cluster computacional de baixo consumo e com proporcionalidade energética / Sebastiao Emidio Alves Filho. - 2018.
85 f.: il.

Tese (doutorado) - Universidade Federal do Rio Grande do Norte, Centro de Tecnologia, Programa de Pós-Graduação em Engenharia Elétrica e de Computação. Natal, RN, 2018.
Orientador: Prof. Dr. Aquiles Medeiros Filgueira Burlamaqui.

1. Computação verde - Tese. 2. Eficiência energética - Tese. 3. Proporcionalidade de hardware - Tese. 4. Computação distribuída - Tese. 5. Dispositivos de baixo consumo - Tese. I. Burlamaqui, Aquiles Medeiros Filgueira. II. Título.

RN/UF/BCZM CDU 004.421

Arquitetura de um *cluster* computacional de baixo consumo e com proporcionalidade energética

Sebastião Emidio Alves Filho

Tese de Doutorado aprovada em 01 de dezembro de 2017 pela banca examinadora composta pelos seguintes membros:

Prof. Dr. Aquiles Medeiros Filgueira Burlamaqui (orientador) ... DCA/UFRN

Prof. Dr. Luiz Marcos Garcia Gonçalves ..... DCA/UFRN

Prof. Dr. Samuel Xavier de Souza ..... DCA/UFRN

Prof[a] Dr. Rommel Wladimir de Lima ..... DI/UERN

Prof[a] Dr. Rafael Vidal Aroca ..... DEMec/UFSCar

À minha família, pelo amor, carinho
e paciência durante a realização
deste trabalho.

Agradecimentos

Ao meu amigo de longa data e agora orientador, professor Aquiles Burlamaqui, por ter aceitado o desafio desse doutorado junto comigo.

Aos professores Luiz Marcos - LM e Rafael Aroca, que ajudaram no desenvolvimento e na publicação do artigo. Mais que competentes, os considero exemplos de seres humanos, obrigado pelos momentos que compartilharam comigo.

Aos colegas dos Laboratórios Natalnet, especialmente do laboratório TEAM: Diogo, Kerolaine, Daniel, Paulo Henrique, Renata, Sarah, Humberto, Wellington, Bianor, Júlio, Sandro, Bruno, Orivaldo, e tantos outros que a memória não permite recordar no momento. Ao seu lado subir os degraus (literalmente) dessa conquista foi mais prazeroso.

À UERN, em especial aos colegas do DI, por possibilitar a minha liberação para realizar o doutorado.

Enfim, a todos que de alguma forma contribuíram para que esse momento fosse possível. Meu muito obrigado.

Resumo

Um dos principais desafios da Computação Verde é obter uma melhor relação entre a quantidade de trabalho realizada pela infraestrutura computacional e o gasto energético para mantê-la, isto é, uma melhor eficiência energética. Este trabalho apresenta a arquitetura de um *cluster* computacional de baixo consumo energético que é capaz de ligar ou desligar, de forma dinâmica e automática, um determinado número de máquinas. A quantidade de máquinas ligadas é proporcional à demanda de trabalho a cada momento, o que evita ligar equipamentos desnecessariamente e aumenta a eficiência do sistema. Para o seu desenvolvimento propõe-se e discute-se um modelo teórico que é implementado através de um *cluster* composto por dispositivos Raspberry Pi chamado NPi-Cluster. Para atestar a eficiência do modelo proposto são mostrados resultados experimentais nos quais o *cluster* é usado como um servidor web com balanceamento de carga. Os dados obtidos mostram que o NPi-Cluster tem um desempenho adequado quando comparado a outros servidores que rodam em arquiteturas tradicionais, mas com um consumo energético menor. Um *cluster* com 7 máquinas usando sua capacidade máxima atende a mais de 450 requisições simultâneas numa taxa de cerca de 1000 transações por segundo. Para fazê-lo o *cluster* consome cerca de 15 Watts, o equivalente a uma lâmpada econômica ou um computador em modo suspenso que não realiza qualquer atividade. Quando a demanda é baixa o consumo de energia com as máquinas é reduzido dinamicamente, chegando a menos de 2 Watts. Além de ser capaz de lidar com cargas de trabalhos com boa qualidade de serviço, o *cluster* também provê alta disponibilidade evitando pontos únicos de falha.

Palavras-chave: Computação Verde, Eficiência Energética, Proporcionalidade de Hardware, Computação Distribuída, Dispositivos de Baixo Consumo.

Abstract

One of the main challenges for the so-called Green Computing is to get a better relation between the amount of work performed by the computational infrastructure and the energy consumption to maintain it, providing better energy efficiency. This work presents the architecture of a computing cluster with low energy consumption that powers on or off a number of running machines automatically and dynamically. The quantity of enabled devices adjusts according to the actual processing demand, which avoids unnecessarily powered equipment and increases the overall system power efficiency. In order to carry out its development, a theoretical model is proposed, discussed, and implemented through the NPi-Cluster, a cluster composed of Raspberry Pi devices. To prove the proposed model feasibility, NPi-Cluster is used as a web server with load balancing. Data gathered shows that NPi-Cluster has adequate performance when compared to other web servers running on traditional server architectures, however with less power consumption. A 7-machine cluster running at maximum performance is able to handle more than 450 simultaneous requests, with about 1000 transactions per second. The power consumption required to do it is about 15 Watts, which is equivalent to a energy-saving light bulb or a computer in suspended mode that does not perform any task. When the requests demand is low, the power consumption is dynamically reduced until less than 2 Watts. Besides to being able to handle workloads with acceptable quality of service, the proposed cluster also provides high availability by avoiding single points of failure.

Keywords: Green Computing, Energy Efficiency, Hardware Proportionality, Distributed Computing, Low Power Electronics.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos e Abreviaturas

ARM: *Advanced RISC Machines*

CO_2: *Dióxido de Carbono*

CPU: *Central Processing Unit*

DCIE: *Data Center Infrastructure Efficiency*

DNS: *Domain Name System*

DVFS: *Dynamic Voltage and Frequency Scaling*

FLOPS: *FLoating-point Operations Per Second*

GPIO: *General Purpose Input/Output*

HPL: *High Performance Computing Linpack Benchmark*

HTTP: *Hypertext Transfer Protocol*

IP: *Internet Protocol*

PUE: *Power Usage Effectiveness*

RISC: *Reduced Instruction Set Computer*

RPi: *Raspberry Pi*

RRDNS: *Round Robin Domain Name System*

SD: *Sistemas distribuídos*

SDCard: *Secure Digital Card*

TCP: *Transmission Control Protocol*

TI: *Tecnologia da Informação*

TICs: *Tecnologias da Informação e Comunicação*

USB: *Universal Serial Bus*

V: *Volts*

W: *Watts*

Capítulo 1

Introdução

A Tecnologia da Informação (TI) está presente no cotidiano das pessoas nas mais diversas situações. Além do uso de aparelhos eletrônicos como computadores, telefones celulares e caixas eletrônicos também há toda uma infraestrutura computacional transparente ao usuário que é necessária para que um bem ou serviço esteja disponível. O uso de uma aplicação de vendas em um supermercado, o compartilhamento de fotos em redes sociais e a edição de um documento na nuvem têm em comum o uso de servidores que podem estar distantes geograficamente de onde os dados contidos nele são acessados.

Se por um lado isso traz vantagens para as pessoas, por outro também traz riscos com relação à degradação do meio ambiente. Ao contrário da crença comum de que a TI é ambientalmente amigável, pois permite a economia de recursos como papel, o seu uso traz consequências como o descarte de produtos eletrônicos em curtos períodos de tempo e o aumento da demanda por energia elétrica (Paek 2014). O descarte incorreto do lixo eletrônico pode poluir o solo e a água com substâncias tóxicas. Com a energia elétrica o problema é o fato do processo de produção liberar dióxido de carbono (CO_2) na atmosfera. Segundo relatório *Key World Energy Statistics* da Agência Internacional de Energia (IEA 2016), mais de 80% da produção mundial de energia no ano de 2015 usou petróleo, gás natural ou carvão. Estas fontes de energia não são renováveis, causam poluição e contribuem com o agravamento do efeito estufa.

Estudos apontam que o setor de Tecnologias da Informação e Comunicação (TICs) tem aumentado as suas pegadas de carbono, isto é, sua participação global na emissão de CO_2. Um relatório da Gartner Inc. (Gartner 2007) estimou que a indústria das TICs era responsável por aproximadamente 2% das emissões globais de CO_2. Já o relatório *SMARTer 2020* da Iniciativa Global de e-Sustentabilidade (GeSI 2012) mostra que o uso generalizado de dispositivos de tecnologia deve aumentar o percentual de emissões de gases de efeito estufa para 2,3% do global em 2020, ante os 1,3% em 2002. Outro estudo (Malmodin et al. 2013) prevê, também para 2020, que o percentual será de 1,9%.

Independente do valor, é consenso que haverá uma demanda crescente por energia e isso pode provocar danos à natureza que poderão se refletir em problemas para esta e futuras gerações. Sob esta perspectiva ganhou força um conceito chamado de Computação Verde ou TI Verde que diz respeito ao uso ambientalmente amigável de computadores e tecnologias relacionadas. Ela inclui práticas como redução do consumo de energia, busca por maior eficiência de energética de CPU e periféricos, e descarte apropriado de todos

os componentes eletrônicos (Agarwal & Nath 2011). Este trabalho tem seu foco voltado para Computação Verde, mais especificamente na redução do consumo de energia em data centers.

1.1 Consumo de energia em data centers

Um fato interessante das previsões sobre o consumo de energia no setor de TI é que o crescimento não é uniforme em todos os tipos de equipamentos. O relatório *SMARTer 2020* (GeSI 2012) mostra nos comparativos entre os anos de 2002 e o previsto para 2020 que o consumo de dispositivos para usuários finais e infraestrutura de comunicação deve dobrar enquanto o consumo previsto para os data centers deve quadruplicar nesse mesmo período.

Outro trabalho mais recente (Hintemann & Clausen 2016) tomou dados do consumo de equipamentos eletrônicos na Alemanha e faz uma previsão global para 2025. Ele prevê que o consumo de energia dos equipamentos para usuários finais cairá, enquanto o da infraestrutura de rede aumentará em 142% e o dos data centers aumentará em 107% quando comparados ao ano de 2010. Alguns fatores explicam a queda da demanda para usuários finais. Em primeiro lugar, houve uma massiva troca de TVs e monitores com tecnologia CRT para LCD e LED, que tem um consumo bem menor. Em segundo, devido à comodidade e à necessidade de mobilidade, as pessoas estão substituindo computadores do tipo desktop por clientes leves ou dispositivos móveis como notebooks, tablets e smartphones que não necessitam ficar ligados na tomada o tempo todo além de terem também um consumo menor.

O poder computacional desses outros equipamentos é mais limitado em termos de velocidade de processamento e transferência de dados quando comparado aos desktops. Essa substituição só foi possível graças à migração de muitas aplicações para a web e ao uso da computação em nuvem, onde a maior parte da carga de trabalho é concentrada em servidores mais robustos que permitem acesso remoto e simultâneo por vários usuários ao mesmo tempo. Esses servidores deveriam ter seu consumo de energia ajustado de acordo com a demanda de trabalho, isto é, um uso proporcional da energia (Barroso & Hölzle 2007). Mas isso não ocorre frequentemente. Computadores tradicionais consomem dezenas ou centenas de Watts (W) e na maioria do tempo são subutilizados (Svanfeldt-Winter et al. 2011). Quando ficam concentrados em data centers, os recursos podem ser otimizados usando uma política de gerência mais verde. Como os data centers usam dispositivos com muitos recursos, em arquiteturas com redundância e geralmente projetados para suportar picos de utilização e condições adversas, há aí um grande espaço para economia de energia (Bianzino et al. 2012).

Algumas das soluções propostas, que estão descritas posteriormente, são baseadas na criação de data centers cuja infraestrutura é composta por *clusters* de computadores com placas simples de tamanho pequeno e baixa potência energética. Um exemplo disso é a Raspberry Pi (RPi), que tem o tamanho de um cartão de crédito, mas dispõe de modelos contendo saída de vídeo, rede (cabeada e sem fio) e processador multinúcleo (Raspberry Pi Foundation 2017*b*). Um *cluster* com algumas unidades de placas RPi requer menos energia que um computador desktop típico em modo suspenso que não realiza qualquer

atividade. Contudo, esses *clusters* recaem no mesmo problema da falta de eficiência energética pois as placas ficam ligadas sem interrupção ou são desligadas diretamente pelo gestor da rede, ou seja, não há mecanismo ou política de controle para que elas sejam desativadas automaticamente.

1.2 Problema e Hipótese de Pesquisa

Diante do exposto, conclui-se que ainda existem problemas em aberto no que se refere à redução do consumo de energia em data centers. O que se observa pesquisando na literatura da área é que existem soluções para o uso proporcional do hardware, mas que ainda assim consomem uma quantidade de energia alta quando comparados a dispositivos como o Raspberry Pi, por exemplo. Por outro lado soluções baseadas em *clusters* de dispositivos de baixa potência elétrica ignoram a questão da proporcionalidade. Em ambos os casos, os trabalhos propostos não obtém a melhor eficiência energética do sistema como um todo.

Há outro ponto a ser discutido referente à qualidade de serviço (*Quality of Service* - QoS). As aplicações distribuídas migradas para servidores em data centers possibilitam o acesso simultâneo de vários usuários, o que pode ocasionar situações de sobrecarga e provocar lentidão ou até negação de serviço. Quando há um aumento no número de clientes que requer maior uso de recursos, a infraestrutura computacional deve estar dimensionada adequadamente. Isso significa que não adianta consumir menos energia e não atender à demanda ou prover um serviço cuja qualidade está abaixo da esperada.

Portanto, este trabalho tem como objetivo contribuir com a pesquisa nessa área a partir do seguinte **problema de pesquisa**:

- Como prover uma infraestrutura computacional para servidores com baixo consumo e boa eficiência energética que seja capaz de atender e se adequar à demanda por um determinado serviço?

Esse trabalho propõe e implementa uma arquitetura de *cluster*, aqui chamada NPi-Cluster, que utiliza um conjunto de placas de baixa potência energética considerando a seguinte **hipótese de pesquisa**:

- Prover uma arquitetura de *cluster* de baixo consumo de energia, com uso proporcional dos servidores, que é capaz de escalar automaticamente o número de máquinas ligadas de acordo com uma política baseada na demanda de trabalho corrente a fim de obter uma maior eficiência energética.

Os testes para validar a solução proposta e verificar a hipótese de pesquisa foram realizados por meio de dois *clusters* utilizando dispositivos Raspberry Pi de modelos diferentes. É feita uma análise de dados para medir desempenho, QoS e eficiência energética em um serviço de acesso a páginas web estáticas com balanceamento de carga entre os servidores ativos. Para verificar a demanda de trabalho corrente e decidir pela ativação ou desativação de um nó é adotado um critério para estimar o tempo em que uma nova requisição seria atendida, dado o número de conexões ativas no momento. Dessa forma, novos

servidores são ligados ou desligados automaticamente quando um estado de sobrecarga ou subutilização é detectado.

1.3 Contribuições

Neste trabalho são mostrados os resultados de dois experimentos realizados a fim de validar a arquitetura proposta. O primeiro tem como objetivo avaliar a eficiência energética dos *clusters* sem o uso proporcional dos servidores quando comparados aos resultados obtidos em um trabalho semelhante (Aroca & Gonçalves 2012). Neste experimento é observado o comportamento dos NPi-Clusters como servidores web de páginas estáticas em relação aos valores obtidos com máquinas de diferentes características, desde outras placas de baixo consumo até um servidor com processador Xeon *quadcore*. Os resultados mostram que, mesmo sem escalabilidade energética, os *NPi-Clusters* estão entre os melhores valores de maior número total de acessos com sucesso, menor tempo de resposta e maior eficiência energética.

Um aspecto importante observado é a diferença de desempenho entre os diferentes modelos de Raspberry Pi usados. Esperava-se que as placas com processador multinúcleo, modelo RPi2, apresentassem um desempenho levemente superior mas também com um consumo de energia maior. Contudo, o que se observa é que o seu desempenho foi muito superior e com um consumo semelhante ao modelo RPi1, com processador de um núcleo.

Uma vez comprovado que o uso do NPi-Cluster é viável do ponto de vista de desempenho e consumo de energia, o segundo experimento mostra o impacto do uso proporcional do hardware na eficiência do sistema como um todo. Para isso são introduzidas três mudanças em relação ao primeiro cenário, que são a base da arquitetura implementada: uma solução para ligar ou desligar os nós do *cluster*; uma metodologia para verificar o estado de um servidor em relação à demanda; e um mecanismo para gerência do cluster e balanceamento de carga.

Para controlar o número de máquinas ligadas no *cluster*, um dos nós, chamado nó gerente, é conectado a um circuito através das portas digitais da interface de entrada/saída de propósito geral (*General Purpose Input/Output* - GPIO). O balanceamento da carga é feito através da distribuição de um conjunto de endereços *Internet Protocol* (IP) entre os servidores e de forma que eles possam trabalhar independente do nó gerente. Isso evita que ele seja um ponto único de falha e contribui para uma maior tolerância a falhas.

Com relação à metodologia para verificar se um nó está sobrecarregado ou subutilizado é comum tomar como critério medidas sobre o uso de um recurso da máquina (processador, memória, tráfego de rede) ou observar empiricamente o número de acessos ao serviço desejado. Este trabalho propõe utilizar como métrica a estimativa de tempo necessário até que uma requisição seja atendida pelo servidor. Assim, é atribuído um estado de utilização ao *cluster* que determina se é necessário ou não ativar novos nós.

A fim de evitar que as máquinas fossem ligadas ou desligadas com muita frequência devido a valores anormais de situações momentâneas, há um mecanismo de gerência que verifica o estado de cada máquina periodicamente. Esse mecanismo de gerência usa um algoritmo para o provisionamento dinâmico dos nós que vai observando se um mesmo estado se repete reiteradamente e se alguma ação deve ser executada.

O segundo experimento também traz uma mudança em relação ao primeiro no que tange ao conjunto de páginas de teste. Enquanto no primeiro experimento apenas uma mesma página web é usada e acessada várias vezes, este último usa um conjunto de arquivos de tipos e tamanhos diferentes, com páginas web, figuras, *scripts*, folhas de estilo e pequenos vídeos. Os arquivos selecionados fazem parte do histórico de registro de acesso à enciclopédia virtual livre Wikipédia, tornando o experimento mais próximo de um cenário de aplicação real.

Portanto, de forma sucinta, as principais contribuições dessa tese são:

- A arquitetura geral um *cluster*, implementada através dos NPi-Clusters, que usa placas de baixo consumo energético e proporcionalidade de hardware para obter uma maior eficiência energética;
- Uma análise mais aprofundada do desempenho das placas Raspberry Pi como servidores web, explicando assim a diferença de desempenho entre os modelos diferentes e apontando um gargalo que representa um fator limitante para a execução dos serviços;
- A realização de dois experimentos em que o primeiro traz a comparação dos resultados do desempenho do NPi-Cluster com um trabalho anterior que comprova a sua eficiência energética, e um segundo que faz uma análise do comportamento do *cluster* quando utilizado um mecanismo de escalabilidade energética;
- A definição de um mecanismo de controle para ligar ou desligar máquinas baseado em distribuição de endereços e GPIO com tolerância à falha do nó gerente;
- Uma metodologia para medir a carga de trabalho de cada servidor que usa uma estimativa de tempo de espera para o atendimento de uma nova requisição;
- A criação de um algoritmo de provissão dinâmico de nós que usa o histórico das últimas medições para evitar que os equipamentos sejam ativados ou desativados por comportamentos atípicos;
- A definição de um conjunto de testes com arquivos obtidos a partir de registros de acessos a um servidor real, tornando o experimento mais verossímil.

É importante mencionar também a publicação, em revista, que foi gerada durante o desenvolvimento desse trabalho:

- S. E. Alves Filho, A. M. F. Burlamaqui, R. V. Aroca & L. M. G. Gonçalves (2017), 'Npi-cluster: A low power energy-proportional computing cluster architecture', IEEE Access 5, 16297–16313.

1.4 Estrutura da Tese

Este documento está organizado em capítulos que são descritos a seguir:

- O Capítulo 2 apresenta os conceitos fundamentais de Computação Verde e das principais técnicas utilizadas para a elaboração deste trabalho, além de uma breve descrição das placas Raspberry Pi;

- O Capítulo 3 mostra uma série de trabalhos relacionados com objetivo dessa pesquisa, onde é possível perceber as diferenças e vantagens da arquitetura mostrada nesta tese;
- O Capítulo 4 apresenta a arquitetura proposta de forma genérica, a metodologia usada para o balanceamento de carga e o algoritmo de provisão dinâmica de nós;
- O Capítulo 5 traz como foram implementados os NPi-clusters e quais os critérios utilizados na metodologia para aferição de carga de trabalho e provisionamento dinâmico de nós;
- O Capítulo 6 retrata como foram montados os experimentos realizados para sua validação e os dados obtidos neles, trazendo algumas reflexões e discussões acerca dos resultados;
- No Capítulo 7 são feitas as considerações finais e propostas de trabalhos futuros.

.

Capítulo 2

Embasamento teórico

Neste capítulo é feita uma breve apresentação sobre os principais conceitos envolvidos no desenvolvimento do trabalho. Os temas *clusters* computacionais e balanceamento de carga já foram muito debatidos na literatura e apresentam diversas classificações de acordo com o tipo de utilização a que são destinados. Na primeira seção são abordadas as classificações que mais se relacionam com o *cluster* construído e com uma maior atenção para balanceamento de carga em servidores web.

Já a temática de Computação Verde também é muito ampla e conhecida, mas o foco dado é sobre a eficiência energética. Muitas instituições abdicaram da responsabilidade e dos custos de manter servidores em cada setor ou até mesmo seus próprios servidores, o que fez com que os data centers aumentassem em número e tamanho, e consequentemente no consumo de energia. Este ainda é um dos maiores desafios da área, especialmente com o uso massivo da internet como plataforma para comunicação e as aplicações na nuvem.

Por fim, é comentado um pouco sobre as placas Raspberry Pi que foram usadas nos experimentos sobre eficiência energética nesta tese.

2.1 *Clusters* computacionais

Um sistema distribuído (SD) pode ser entendido como "um conjunto de computadores independentes entre si que se apresenta a seus usuários como um sistema único e coerente" (Tanenbaum & Steen 2007), ou ainda "um sistema em que componentes de hardware e software localizados em computadores em rede se comunicam e coordenam suas ações por passagem de mensagens" (Coulouris et al. 2013). Ou seja, os SDs aproveitam a interconexão de equipamentos separados fisicamente através de uma rede para aumentar a disponibilidade de um recurso ou serviço de forma articulada. Segundo os autores citados previamente neste parágrafo, algumas das principais características inerentes aos sistemas distribuídos são:

- Compartilhamento de recursos: os sistemas distribuídos permitem que um cliente possa acessar simultaneamente recursos que estão em máquinas separadas;
- Robustez: ao contrário de um sistema centralizado, em que há um ponto único de falha, os SDs buscam fazer a replicação e redundância dos serviços de tal forma que se um dos componentes falhar ou ficar sobrecarregado outro possa assumir seu papel;

- Transparência: como consta no conceito apresentado, o usuário enxerga o sistema como uma entidade única, ficando transparente os detalhes sobre a localização ou a forma de acesso a um recurso, ou ainda se o mesmo está replicado, migrado ou se recuperou de alguma falha;
- Escalabilidade: o sistema não deve perder significativamente seu desempenho caso aumente o número de usuários e recursos, sua distância geográfica ou sua complexidade de gerenciamento;
- Qualidade de serviço: um SD pode propiciar uma melhor divisão de acessos a um recurso de forma a buscar atender a requisitos mínimos de QoS, como *jitter* ou percentual de disponibilidade.

Três paradigmas comuns de sistemas distribuídos são os *clusters*, as grades (*grids*) e as nuvens (*clouds)*. As grades e nuvens podem ser pensadas como o uso dos SDs como utilidade, onde as grades são consideradas precursoras das nuvens e com um viés mais científico (Coulouris et al. 2013). As nuvens são definidas como um conjunto de aplicações e serviços de processamento e armazenamento remoto baseados na Internet que dispensam, em grande parte ou totalmente, a necessidade dos clientes terem uma aplicação instalada ou dados armazenados localmente. Já os *clusters*, numa abordagem tradicional, representam uma alternativa aos supercomputadores onde um grupo de máquinas independentes interligadas por uma rede de alta velocidade executam aplicações de computação de alto desempenho. Isto é, enquanto nas grades e nuvens há um objetivo mais voltado a tornar um serviço ou dado remoto disponível, os integrantes de *clusters*, chamados nós, são mais direcionados para executar tarefas com uma grande quantidade de operações de processamento ou manipulação de dados.

Numa visão mais recente a definição de *cluster* computacional vai além do processamento paralelo para também incluir os *clusters* de alta disponibilidade e de balanceamento de carga (Valentini et al. 2013). Os de alta disponibilidade têm como propósito replicar um recurso em mais de uma máquina de forma que a falha de um nó não torne o recurso inacessível. Já os de balanceamento de carga são gerenciados de forma que informações e volume de trabalho sejam divididos entre os nós a fim de evitar que um deles fique sobrecarregado e não consiga atender a requisitos de QoS. *Clusters* com essas características geralmente estão incorporados à infraestrutura computacional das grades e nuvens.

2.1.1 Balanceamento em *clusters* de servidores web

Balanceamento de carga em SDs é um problema conhecido e com soluções bem definidas. Um conjunto de métricas comumente usadas para nuvens (Ghomi et al. 2017, Milani & Navimipour 2016) pode ser usada de maneira geral para medir a eficácia e a eficiência dos algoritmos de balanceamento de carga em SDs:

- *Throughput*: número de processos completos ou quantidade de dados transferidos por unidade de tempo;
- Utilização dos recursos: grau em que os recursos do sistema são utilizados;
- Tempo de resposta: tempo que o sistema leva para concluir uma tarefa submetida;

- *Makespan*: tempo máximo para completar uma tarefa ou alocar um recurso ao usuário;
- Escalabilidade: uniformidade no balanceamento de carga de acordo com os requisitos mediante o aumento do número de nós;
- Tolerância a falhas: capacidade de executar o balanceamento mesmo em situações em que alguns nós ou conexões falhem;
- Tempo de migração: tempo necessário para transferir uma tarefa de um nó sobrecarregado para outro subutilizado;
- Eficiência: o desempenho do sistema após realizar o balanceamento de carga;
- Grau de desbalanceamento: diferença de carga entre os nós;
- Consumo de energia: total de energia consumida por todos os nós;
- Emissão de carbono: total de pegada de carbono produzida por todos os recursos.

Um bom algoritmo de balanceamento de carga consegue aumentar a *eficiência* com máxima utilização dos recursos, maior *throughput*, menor tempo de resposta e *makespan* próximo a media dos tempos de resposta. Todos esses objetivos devem ser buscados ao mesmo tempo em que se tem uma boa escalabilidade, tolerância a falhas, baixo tempo de migração, baixo grau de desbalanceamento e baixo impacto ambiental (consumo de energia e emissão de carbono).

Existem diversos algoritmos de balanceamento de carga na literatura dos quais se pode destacar: Shaw & Singh (2014); Jiang (2016); Ghomi et al. (2017); Milani & Navimipour (2016). As técnicas para este fim podem ser classificadas sob vários aspectos (Jiang 2016):

- Em relação ao modelo de controle:
  - Centralizado – há apenas um nó central que exerce funções de controle enquanto todos os outros são passivos em relação à alocação de tarefas. É ideal para *clusters* pequenos onde representa uma solução globalmente ótima;
  - Distribuído – não há controladores, todos os nós são autônomos para a distribuição de tarefas. Indicado para cenários de larga escala, cada nó obtém informações apenas sobre nós vizinhos, o que pode gerar uma solução localmente ótima;
  - Híbrido – os nós podem ter um papel ativo, agindo como controladores, ou passivo, apenas recebendo tarefas para executar. É um meio termo em relação ao tamanho e à otimalidade da solução.
- Referente à otimização dos recursos:
  - De forma autônoma – a forma de distribuição das requisições para os serviços é feita a partir dos recursos que o nó possui. Dessa forma cada um executa as tarefas de forma autônoma usando seus próprios recursos para isso;
  - Baseado no contexto – a probabilidade de uma requisição ser alocada para um nó depende também de um contexto para o qual os recursos são otimizados. Geralmente usado para SDs cooperativos, requererem um grande custo computacional.
- Quanto à confiabilidade:

  - Com redundância – implementa o mesmo serviço de forma redundante em mais de um nó de forma que ele estará disponível mesmo se um nó falhar, produzindo uma boa tolerância a falhas;
  - Sem redundância: usa algum mecanismo de otimização, confiança ou reputação para achar a melhor estratégia de alocação. Necessita de um histórico de experiências passadas do nó para definir seus parâmetros de confiabilidade.

- Quanto à coordenação de nós heterogêneos:
  - Baseado em teoria dos jogos – modela o *cluster* como um jogo, cooperativo ou não, e busca atingir um equilíbrio na alocação de tarefas de forma dinâmica e distribuída;
  - Baseado em teoria dos grafos – modela o *cluster* como um grafo, usando os custos das tarefas em cada equipamento como pesos para nós e arestas.

Quando se trata de balanceamento especificamente para servidores web, pode-se classificar as técnicas, segundo Cardellini et al. (1999), como baseadas: no cliente, no servidor web, no servidor DNS (*Domain Name System* - sistema de nomes de domínio) ou em web *dispatchers*. Apesar de terem sido aplicadas a servidores web essas metodologias poderiam ser usadas para outros tipos de serviço sem nenhum problema.

Técnicas baseadas no cliente são as menos utilizadas. Nelas os servidores web não executam qualquer ação, ficando a responsabilidade da escolha do nó para o cliente. Essa escolha pode ser feita de forma explícita quando, por exemplo, escolhe-se um *mirror* de uma página desejada através de um navegador. Ou ainda de forma implícita, um *proxy* acessado pelo cliente faz essa escolha baseada em algum critério de proximidade geográfica ou tempo de resposta.

Balanceamentos baseados em servidor ocorrem de forma distribuída entre os nós do *cluster* e geralmente têm duas fases. Numa primeira fase a requisição é recebida e então se escolhe a máquina que irá atendê-la. Na segunda, a requisição é alterada para ser encaminhada para o destino através de um redirecionamento HTTP ou reescrita do pacote. Um exemplo do uso dessa técnica é o gerenciador de recursos escalável de alta disponibilidade Pacemaker (ClusterLabs 2017). Ele trabalha com dois tipos de configuração: ativo/passivo e ativo/ativo. O modo ativo/passivo é mais usado para garantir a tolerância a falhas, onde um recurso é alocado para um nó e só é migrado para outro quando o primeiro for desativado. Já no modo ativo/ativo o serviço é replicado igualmente em todos os nós que respondem às requisições através de uma rede *multicast*.

O balanceamento baseado em despachantes intermediários, web *dispatchers*, necessita de um elemento central que faz a interface entre os clientes e o *cluster*, recebendo as requisições e encaminhando para o nó que vai atendê-la. Esse encaminhamento pode acontecer em nível de camada de rede onde os pacotes são redirecionados por regras de *firewall*, *proxy* de rede ou até mesmo no equipamento roteador. Contudo, é mais frequente que ocorra em nível de transporte ou aplicação, onde também é feito um redirecionamento da requisição TCP (*Transmission Control Protocol*) ou HTTP (*Hypertext Transfer Protocol*). Dois exemplos de *dispatchers* testados durante a execução deste trabalho foram o módulo de balanceamento do servidor web Apache (Apache Software Foundation 2017*b*) e o software de balanceamento de carga TCP/HTTP HAProxy (HAProxy Technologies

2017). Ambos trabalham de forma semelhante, recebendo requisições e enviando para os servidores disponíveis de maneira uniforme, ponderada, aleatória ou de acordo com o menor número de conexões ativas.

Por fim, o balanceamento baseado em DNS mantém no servidor de nomes um conjunto de endereços IP associados ao mesmo nome em um domínio. Com isso, toda vez que um cliente consulta o servidor DNS será devolvido como resposta um IP diferente desse conjunto. Uma das maneiras de se fazer isso é através do *Round Robin DNS* (RRDNS), no qual a cada consulta de um nome os endereços são devolvidos de forma sequencial até último da lista e depois retornando ao primeiro, e assim sucessivamente (Brisco 2013). Por exemplo, se o servidor com nome `www.natalnet.br` estivesse associado do IP 10.0.0.1 até o 10.0.0.10, no primeiro acesso ao DNS ele associaria o nome ao IP 10.0.0.1, no segundo a 10.0.0.2, e assim sucessivamente até 10.0.0.10, onde retornaria para 10.0.0.1. Esse processo se repete de forma contínua e pode usar inclusive um conjunto de endereços não sequenciais.

Cada abordagem mostrada tem suas vantagens e desvantagens. O baseado nos clientes não proporciona qualquer sobrecarga aos servidores, mas também não garante o balanceamento porque dependeria dos clientes. O baseado nos servidores web permite uma gerência distribuída e uma maior tolerância a falhas, mas em compensação gera uma alta taxa de troca de mensagens para manter o estado de cada nó do *cluster* atualizado. As técnicas com web *dispatcher* podem controlar melhor o balanceamento, mas representam um ponto único de falha além de ser o ponto crítico para o surgimento de um gargalo na comunicação. Já os baseados em DNS também não sobrecarregam os servidores, mas não lidam bem com cenários dinâmicos, uma vez que se um novo servidor ou endereço é adicionado, o servidor DNS tem que ser alterado e o serviço reiniciado.

2.2 Computação verde

Vários autores conceituaram o que é Computação Verde ou TI Verde e seus objetivos, das quais destaca-se a seguinte definição:

> É o estudo e a prática de projeto, fabricação, uso e descarte de computadores, servidores e subsistemas associados – como monitores, impressoras, dispositivos de armazenamento e redes e sistemas de comunicação – de maneira eficiente e efetiva com o mínimo ou sem impactos para o meio ambiente. TI verde também busca viabilidade econômica e melhoria no uso e desempenho, ao mesmo tempo respeitando nossas responsabilidades sociais e éticas. (Murugesan 2008, tradução nossa)

O autor levanta alguns pontos que merecem destaque. Em primeiro lugar o cuidado com o meio ambiente exige que sejam adotadas medidas para evitar o impacto da tecnologia em todas as fases do seu ciclo de vida: projeto, fabricação, uso e descarte. Em segundo, a atenção não deve estar concentrada apenas nos computadores pessoais, mas também em dispositivos que trabalham em conjunto como periféricos, servidores, dispositivos de armazenamento, sistemas de comunicação, etc. Por fim, a Computação Verde

não abandona os propósitos para os quais a computação é usada nas mais diversas áreas: viabilidade econômica e melhoria no uso e desempenho. Surge então um termo chave na área que é a eficiência, isto é, obter o melhor desempenho possível mantendo uma consciência ambiental.

A Computação Verde propõe uma mudança de paradigma principalmente quando se pensa nas pegadas de carbono geradas pelo setor das TICs com ênfase principalmente no ciclo de vida dos equipamentos e no uso eficiente da energia. Referente ao ciclo de vida dos equipamentos, por não ser exclusiva da área de TI, a preocupação com o meio ambiente na manufatura industrial já vem desde a década de 1970. Projetos verdes estenderam o conceito do ciclo de vida dos produtos para incluir também práticas de reciclagem e descarte de produtos ultrapassados (Dong et al. 2003). Além disso, parâmetros ambientais, de segurança e de uso de energia e recursos foram integrados a características básicas dos produtos como funcionalidade, qualidade e custo. Mais recentemente, tem-se usado o conceito de gerência verde de cadeias de produção (*Green Supply Chain Management*) que não inclui somente o processo de manufatura, mas também a cadeia de matéria prima, venda e uso dos produtos (Srivastava 2007). As características de uma cadeia de produção verde são a preocupação com sustentabilidade em projetos, seleção de materiais, seleção de fornecedores, produção, empacotamento e reciclagem (Gao et al. 2009). Para certificar organizações que adotam um modelo de produção verde e auxiliar consumidores a adquirirem esses produtos foram criados padrões, nacionais e internacionais, além de selos de certificações, os chamados *EcoLabels* ou *EcoSeals*. Nenhuma dessas certificações avalia todos os requisitos de forma satisfatória, mas serve para que se possa evitar práticas corporativas, projetos e produtos ruins, sendo aconselhável combinar os melhores critérios de cada um (Stevels 2001, St-Laurent et al. 2012).

Já com relação ao uso eficiente de energia em *clusters*, é um problema um pouco mais complexo porque os *clusters* foram idealizados para computação de alto desempenho. É comum encontrarmos softwares que são usados para medir o desempenho em relação ao número de operações executadas por segundo ou tempo total para completar um conjunto de tarefas. Um dos mais conhecidos é o *High Performance Computing Linpack Benchmark* (HPL), um pacote de *benchmark* para computadores com memória distribuída (Petitet et al. 2017) que usa o pacote LINPACK (Dongarra et al. 2003). Ele faz a resolução de sistemas de equações lineares densos através de fatorização LU com pivoteamento parcial. Esse pacote é usado para compor o TOP500 (*Top500 Supercomputer Sites* 2017), um projeto que lista os 500 melhores supercomputadores do mundo.

No entanto, à medida que o modelo de computação em nuvem foi se popularizando, pequenos escritórios deixaram de ter seus próprios servidores de aplicação para hospedá-los remotamente em data centers especializados. A concentração de *clusters* dedicados para prover serviços na nuvem aumentou a demanda por energia para equipamentos e também para a climatização do ambiente devido à temperatura atingida pelos processadores. Além do problema ambiental, há um aumento no custo para manutenção dessa infraestrutura. Assim, o foco na gestão de data centers passou a ser também atingir uma maior eficiência energética, ou seja, conseguir o máximo de desempenho por cada unidade de energia necessária para seu funcionamento. O projeto TOP500, por exemplo, passou a ter também uma lista GREEN500, onde o ranking é ordenado pela eficiência medida em

operações de ponto flutuantes (Floating-point Operations Per Second - FLOPS) por Watt (Feng & Cameron 2007).

Outras métricas também foram criadas para verificar a eficiência energética dos *clusters* computacionais em relação a outros equipamentos necessários para seu funcionamento no data center, como os de comunicação e climatização (Belady et al. 2008). As métricas *Data Center Infrastructure Efficiency* (DCIE) e *Power Usage Effectiveness* (PUE) são dadas pela relação entre o consumo energético somente dos equipamentos de TI e o consumo total dos equipamentos presentes nas instalações físicas do data center. Quanto mais próximos estão esses valores melhor a eficiência e efetividade.

2.2.1 Técnicas para aumentar a eficiência energética em *clusters*

Muitos estudos foram conduzidos para analisar e atenuar o problema do consumo de energia e pegadas de carbono nos data centers, e um deles (Kaur & Chana 2015) classifica as soluções como baseadas em localização, infraestrutura, hardware e software. As soluções baseadas em localização buscam os melhores pontos geográficos para implantação das instalações físicas dos data centers de forma que eles fiquem próximos a plantas de produção de energia limpa, como usinas hidrelétricas, fotovoltaicas e eólicas. As baseadas em infraestrutura focam na construção dos prédios onde os equipamentos serão instalados. Nesse caso, são considerados aspectos da engenharia e arquitetura predial que reduzam o calor ou diminuam a necessidade de refrigeração. As técnicas baseadas em hardware buscam reduzir o consumo de cada equipamento através de um melhor uso dos seus componentes físicos, enquanto as baseadas em software propõem modificações na fase de projeto e durante a utilização do software de forma a reduzir o uso dos equipamentos. O trabalho mostrado nesta tese foca no hardware e seu gerenciamento.

Há duas formas de gerenciar o consumo de energia em um nó de um *cluster*: estaticamente e dinamicamente (Valentini et al. 2013). No gerenciamento estático, realiza-se a substituição de um hardware por outro de menor consumo ou com maior eficiência. O gerenciamento dinâmico utiliza componentes energeticamente escaláveis controlados por um software que pode mudar seu comportamento em tempo de execução.

Tratando primeiro da abordagem estática, ela tem crescido em parte pelo grande aumento no uso de dispositivos móveis. Contando com aplicativos que requerem cada vez mais recursos, os fabricantes de notebooks, smartphones e tablets têm o desafio de montar placas com um bom desempenho, mas que consumam pouca energia. Uma das arquiteturas mais usadas no mundo e comumente usada na indústria de dispositivos móveis é a baseada em processadores ARM (Fitzpatrick 2011). No capítulo 3 desta tese são mostrados vários exemplos de *clusters* formados por dispositivos com arquitetura ARM que trazem uma melhor eficiência energética quando comparados a arquiteturas tradicionalmente usadas em servidores.

Outro exemplo de hardware com melhor eficiência energética são os dispositivos com processadores multinúcleo. Um processador multinúcleo possui duas ou mais unidades de processamento no mesmo processador compartilhando a mesma fonte de energia, sistema de resfriamento e outros componentes. Com isso é possível obter uma maior eficiência no processamento através da execução de instruções em paralelo que rodam em unidades

diferentes, mas com um acréscimo de energia menor que se o processamento fosse concentrado em um núcleo com maior frequência de operação, ou ainda se houvesse um servidor multiprocessador, onde há unidades de processamento separadas (Blake et al. 2009). Quando um núcleo não é usado ou é usado com menor intensidade o processador requer menos energia e obtém uma eficiência ainda maior.

Sobre o gerenciamento dinâmico do hardware, há o que se chama de escalabilidade energética, isto é, o crescimento no consumo de energia se dá de acordo com o aumento da carga de trabalho. Pensando em cada equipamento individualmente, algumas estratégias mais simples envolvem apenas reduzir o consumo ou desligar equipamentos acessórios como monitores ou impressoras quando não são usados por certo período de tempo. Algumas mais complexas são tomadas no nível de arquitetura através de hardware com proporcionalidade energética, como processadores com níveis dinâmicos e modos de potência (Kansal & Zhao 2008). Outras estratégias de uso proporcionalidade energética envolvem também o uso de *links* de comunicação e equipamentos de rede (Bianzino et al. 2012), e também gerenciamento dinâmico de memória (Tolentino et al. 2007). Há ainda técnicas que são usadas especificamente para eficiência energética em computação paralela (Jin et al. 2016).

O hardware com uso proporcional de energia é aquele onde é possível definir diferentes níveis de desempenho de acordo com algum critério de utilização. Por exemplo, se o dispositivo não tem tarefas associadas ele idealmente não deve ter consumo de energia, mas deve trabalhar com o máximo desempenho em momentos de maior utilização e com um desempenho menor, consumindo menos energia, quando não é muito acionado (Barroso & Hölzle 2007). Processadores com níveis dinâmicos possuem mecanismos para reduzir sua frequência de operação e/ou a voltagem para fazê-los funcionar, onde se define um conjunto de níveis de operação cada um com um consumo de energia diferente. Esses níveis podem ser alterados pelo sistema de forma dinâmica, como no modelo *Dynamic Voltage and Frequency Scaling* (DVFS), geralmente associados a um modo de potência (Mittal 2014). Os modos de potência controlam o hardware de acordo com alguma situação pré-determinada. Por exemplo, em um modo "economia de energia" um processador é colocado em uma frequência de operação menor, o brilho da tela de um notebook é diminuído se a carga da sua bateria chegar a menos de 15%, ou um desktop é colocado em modo de "hibernação" caso não haja interação do usuário durante 30 minutos. Quanto mais modos de potência, maiores são as possibilidades de reduzir o consumo de energia, seja de forma automática ou de forma intencional através administrador do sistema. Alguns trabalhos (Kansal & Zhao 2008, Rao et al. 2011) mostram estratégias dinâmicas de modos de potência implementadas em nível de sistema operacional. Eles buscam fazer uma predição sobre a utilização do hardware pelo usuário através da construção de perfis baseados no seu comportamento e em indicadores fornecidos pelo *kernel* do sistema. Uma vez identificado um perfil, um modo de potência é escolhido para fornecer o desempenho mais adequado com o menor nível de tensão necessário.

Contudo, é preciso usar essas políticas cuidadosamente, pois os modos com menor consumo de energia levam mais tempo para retornar ao estado normal e vice-versa. Dessa forma, deve-se avaliar se a energia a ser economizada compensa o tempo gasto com a mudança de modos (Mittal 2014). Além disso, um processador com menor frequência de

operações leva mais tempo para concluir as tarefas e ficar sem carga (*idle*), ou ainda pode afetar parâmetros de qualidade de serviço.

Outra possibilidade de usar o hardware de forma proporcional é considerando o conjunto de máquinas do *cluster* ao invés de olhar individualmente para cada equipamento. Duas técnicas usadas para isso são o balanceamento de carga e a provisão dinâmica de nós (Chen et al. 2008). Como visto na seção anterior, o balanceamento de carga distribui as tarefas entre os nós evitando que uns fiquem sobrecarregados enquanto outros fiquem subutilizados. Visando uma maior eficiência energética a distribuição das requisições entre os nós possibilita que os nós possam adotar níveis de operação do processador menores ou modos de potência que consumam menos. Também é possível escalonar tarefas com restrições temporais mais severas para máquinas mais robustas enquanto outras sem restrição podem ser executadas em máquinas com maior eficiência energética, mesmo que com uma velocidade menor.

Na provisão dinâmica de nós, as máquinas podem ser ligadas/desligadas de acordo com a demanda. Se os serviços de uma máquina estão sendo pouco usados, pode-se migrá-los para outra e deixá-la inativa até que um aumento na demanda faça com que uma máquina seja sobrecarregada e a que estava desligada seja ativada novamente. Isso pode ser feito de forma automática, sem intervenção humana, onde cada fase tem seus desafios de projeto (Qu et al. 2016):

- Monitoramento: é necessário monitorar os recursos através de indicadores de desempenho que indiquem com certo grau de confiabilidade o nível de ocupação de cada um. Também é importante definir um intervalo de monitoramento equilibrado que consiga obter o estado atual do sistema sem acrescentar uma sobrecarga que afete o desempenho dos serviços;
- Análise: nessa fase verifica-se, com base nas informações de monitoramento, se é necessário fazer alguma alteração na composição do sistema. Deve-se decidir, por exemplo, se a provisão será proativa ou reativa, se será feita com base em alguma predição de carga ou somente com a carga real, a sua adaptabilidade a mudanças e como evitar grandes oscilações onde recursos são desativados pouco tempo depois de serem ativados e vice-versa;
- Planejamento: nesse período deve-se planejar quantos/quais recursos serão ativados ou desativados para se adequar à carga atual ou estimada;
- Execução: quando as ações do provisionamento que foram planejadas são efetivamente executadas.

A aplicação desta técnica de provisão dinâmica em *clusters* baseada na demanda corrente de trabalho é chamada de *Vary-On Vary-Off* (Pinheiro et al. 2003).

2.3 Dispositivos Raspberry Pi

A história dos dispositivos ARM começou em 1983 com uma companhia chamada Acorn que buscava um microprocessador de 16-bit para sua próxima geração de desktops, mas não encontrou soluções satisfatórias no mercado para atender seus requisitos

(Ryzhyk 2006). Ela então resolveu desenvolver seu próprio processador baseado na arquitetura RISC (*Reduced Instruction Set Computer*), com um conjunto de instruções simples e pequeno. Em 1990 eles fundaram um empreendimento conjunto com a empresa Apple a fim de desenvolver um processador para um PDA que oferecesse os benefícios da arquitetura RISC, mas divergindo em alguns aspectos para um melhor desenvolvimento de sistemas embarcados. Esse empreendimento foi chamado de *Advanced RISC Machines*, de onde vem a sigla ARM.

Uma vantagem da arquitetura ARM é que ela possui processadores, inclusive multinúcleo, com frequências comparáveis a sistemas tradicionais x86, mas com um consumo de energia muito inferior (Tudor & Teo 2013). Devido a essa eficiência energética, preço e tamanho dos dispositivos, muitas empresas e institutos de pesquisas estão migrando sua infraestrutura de servidores para sistemas em chip (*Sytems on Chip* - SoC). Dentre os vários dispositivos que implementam a arquitetura ARM, o foco deste trabalho é o Raspberry Pi.

O Raspberry Pi é um computador completo do tamanho de um cartão de crédito de baixo consumo energético. Seu desenvolvimento é gerenciado por uma fundação homônima cujo objetivo é prover computadores com baixo custo e de alto desempenho com os quais as pessoas possam aprender, resolver problemas e divertir-se (Raspberry Pi Foundation 2017*b*). Existem alguns modelos de placas que se diferenciam umas das outras pela capacidade de processamento, quantidade de memória e tipos de conectividade, mas mantém um aspecto físico semelhante, como o modelo da figura 2.1. Os modelos contam também com portas *Universal Serial Bus* (USB) para acoplar dispositivos de entrada como mouse e teclado, portas para dispositivos de saída de áudio e vídeo e uma entrada para cartões *Secure Digital* (SDCard).

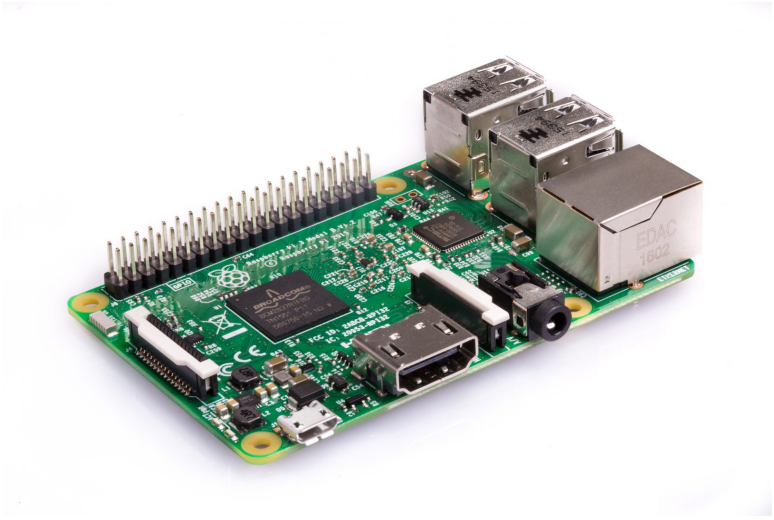

Figura 2.1: Foto do Raspberry Pi modelo 3 B (Raspberry Pi Foundation 2017*a*).

Uma das características mais importantes das placas RPi é a interface de entrada e saída GPIO que representa a principal maneira para conectar uma Raspberry a outros dispositivos eletrônicos. Ela consiste em um conjunto de pinos organizados em duas

colunas que pode aceitar comandos de entrada e saída, podendo assim receber dados de sensores e botões ou controlar lâmpadas e motores para automação residencial, por exemplo (Vujović & Maksimović 2015). Alguns pinos podem ser usados como entrada/saída digitais e como interfaces para protocolos embarcados como *Universal Asynchronous Receiver-Transmitter* (UART), *Inter-Integrated Circuit* (I^2C) e *Serial Peripheral Interface* (SPI). A ordem e a quantidade de portas variam de acordo com o modelo. A figura 2.2 mostra três possíveis configurações de portas. As indicadas com a cor vermelha são de alimentação (positivo de 3,3V ou 5,5V); as pretas são de terra (negativo); verdes, azuis e laranjas são de comunicação; e as amarelas são de propósito geral.

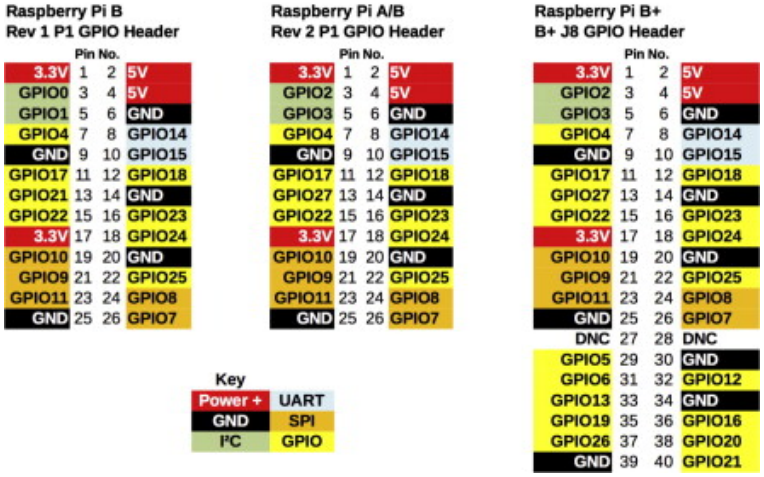

Figura 2.2: Conectores GPIO de diferentes Raspberry Pi (Vujović & Maksimović 2015).

A GPIO é mapeada em arquivos de dispositivos do diretório /dev/ no sistema operacional Linux e pode ser acessado usando qualquer linguagem de programação com suporte à arquitetura ARM. Existe uma distribuição Linux oficial baseada no Debian que é mantida pela Fundação Raspberry Pi chamada Raspbian (Raspberry Pi Foundation 2017*c*).

2.4 Principais pontos sobre o embasamento teórico

Conforme visto neste capítulo, *cluster* é um paradigma para criação de sistemas distribuídos e, como tais, eles devem manter características como robustez, transparência e escalabilidade. *Clusters* de balanceamento de carga visam evitar sobrecarga de nós individualmente através da divisão da carga de trabalho. Cada metodologia para balanceamento de carga tem suas vantagens e desvantagens, onde se deve evitar fatores que limitem ou sejam um gargalo de forma que o sistema atinja o melhor desempenho possível.

Contudo, o conceito de Computação Verde traz que o desempenho é tão importante quanto uma boa eficiência energética, isto é, conseguir o melhor desempenho por unidade de energia consumida. Uma melhor eficiência pode ser obtida estaticamente substituindo um hardware por outro mais eficiente, como os que possuem tecnologia multinúcleo ou por dispositivos baseados na arquitetura ARM, como o Raspberry Pi. De forma dinâmica,

a eficiência energética pode ser conseguida com o uso proporcional do hardware, ou seja, quanto mais o equipamento é exigido mais ele consome energia e vice-versa. Estratégias como DVFS, modos de potência e provisão dinâmica de nós *Vary-On Vary-Off* são exemplos disso.

No capítulo 3 que vem a seguir, são descritos trabalhos que tratam desses aspectos e utilizam *clusters* com eficiência energética. Eles usam dispositivos de baixo consumo de energia ou técnicas para uso proporcional do hardware. O objetivo é fazer um levantamento bibliográfico do estado da arte na área e assim diferenciar a proposta desta tese das demais já existentes.

Capítulo 3

Trabalhos relacionados

Com o avanço da tecnologia, pequenos computadores baseados em processadores de arquitetura ARM têm surgido com um desempenho cada vez melhor. Inicialmente, o suporte a um sistema operacional de rede fez com que fosse possível a implementação de servidores web embarcados com baixo consumo de energia (Roy et al. 2009, Guan & Gu 2010, Poongothai 2011). Esses servidores eram de autoria dos próprios autores dos trabalhos e estavam associados a sistemas para sensoriamento remoto. O papel dos servidores web era evitar o *overhead* de transferência de dados para outras máquinas, fornecendo um meio direto para acessá-los no próprio dispositivo via web com o sistema embarcado. Posteriormente, já com o desenvolvimento de distribuições do sistema operacional Linux, servidores web mais conhecidos como o Apache (Apache Software Foundation 2017*b*) e o Nginx (NGINX Inc. 2017) começaram a ser utilizados, assim como outros tipos de aplicação. A grande questão era saber se esses dispositivos tinham um bom desempenho e se a relação desempenho / energia compensava na obtenção de uma melhor eficiência.

Um estudo aponta que uma máquina com processador ARM Cortex A9 apresenta eficiência energética mais de 11 vezes superior a um servidor com processador Intel Xeon (Svanfeldt-Winter et al. 2011). Outro estudo mais completo (Aroca & Gonçalves 2012) verificou o desempenho de sistemas em chip baseados na arquitetura ARM em aplicações de computação intensiva com o LINPACK, acesso a páginas web e a servidor de banco de dados SQL. O estudo compara o desempenho de máquinas Pandaboard e Beagleboard com outras rodando arquitetura x86, sendo dois tipos de desktops e dois de notebooks. Exceto no primeiro experimento, as placas baseadas em ARM tiveram melhor eficiência energética que os baseados na arquitetura x86, sendo de 4 a 5 vezes melhor.

3.1 *Clusters* com dispositivos de baixo consumo

Com relação a *clusters* com dispositivos de baixo consumo, vários trabalhos aparecem na literatura mostrando as experiências, testes de avaliação de desempenho e aplicações usadas. A tabela 3.1 traz um resumo destes trabalhos, mostrando o trabalho em que foi publicado, o tipo de aplicação que foi usada, a plataforma das máquinas utilizadas, a quantidade de nós presentes e o consumo de energia por nó.

A maioria dos relatos científicos encontrados sobre *clusters* de baixo consumo apre-

senta resultados de aplicações em computação de alto desempenho. Na seção 2.2 falou-se sobre os softwares de *benchmark* LINPACK e HPL para resolução de sistemas lineares. Outros softwares também utilizados para esse fim são: STREAM, NPB, Core Mark, SPEC CPU e Dhrystone. O STREAM (*Sustainable Memory Bandwidth in High Performance Computers*) utiliza cálculos com vetores para testar o desempenho da memória em relação à velocidade da CPU, e saber se esta é um gargalo no sistema (McCalpin 1995). O NPB (*NAS Parallell Benchmark*) é uma coleção de softwares derivados de aplicações computacionais para dinâmicas de fluidos (Bailey et al. 1991). Ele foi criado pela divisão de supercomputação avançada da NASA para ajudar a avaliar o desempenho de supercomputadores paralelos. Já os softwares Core mark (EMBC 2017), SPEC CPU (Henning 2007) e Dhrystone (Weicker 1984) têm o objetivo de verificar o desempenho da CPU, sendo o primeiro mais recente e também voltado para sistemas embarcados.

Pode-se citar uma lista de trabalhos que usaram pelo menos um desses softwares de *benchmark*: Fürlinger et al. (2011); Keville et al. (2012); Balakrishnan (2012); Rajovic et al. (2013); Rajovic et al. (2014); Cox et al. (2014); Cloutier et al. (2016); e Xu & Chang (2017). Outros artigos também trazem testes de desempenho com operações matemáticas, como Pfalzgraf & Driscoll (2014) e Krpić et al. (2014). No geral, os resultados dos artigos listados dizem que os *clusters* de baixo consumo apresentaram resultados satisfatórios para essas aplicações científicas clássicas e matemáticas.

Alguns trabalhos não mostram seus *clusters* sendo usados para aplicações específicas. Eles apenas indicam a forma como foram montados, o hardware utilizado, o tipo de conectividade de rede e qual o seu propósito. Destes, destaca-se uma importante iniciativa que é o Projeto Mont Blanc (Mont Blanc 2011). Ele foi um dos pioneiros a construir *clusters* baseados em ARM, contando com hardware de diversos fabricantes buscando uma maior eficiência energética. Outro pioneiro apresentava 196 máquinas construídas com processador ARM Cortex-A8 (Brown 2011). Alguns foram construídos com propósitos específicos, como compilação distribuída de pacotes de uma distribuição Linux (Humphries 2011) ou análise de arquivos de *logs* distribuídos (Andersen et al. 2009).

Dois trabalhos (Abrahamsson et al. 2013, Tso et al. 2013) mostram a construção de uma infraestrutura com máquinas Raspberry Pi no intuito de prover serviços de computação em nuvem. Algumas características comuns nas plataformas de nuvem são encontradas, como ferramentas para configurar os serviços, monitoramento e manutenção da infraestrutura. O último também faz o gerenciamento de Linux *Containers*, que são uma alternativa mais leve e viável para virtualização em ambientes com recursos computacionais mais limitados. Essa pseudo-virtualização ocorre no nível de sistema operacional e permite rodar múltiplos servidores sob o controle de um único *host*.

Algumas pesquisas mostram a construção de servidores web e bancos de dados distribuídos: Ou et al. (2012); Xu & Chang (2017); Loghin et al. (2015); Zhao et al. (2016). As duas primeiras usaram páginas web estáticas e dinâmicas com tamanhos variados. A terceira usa um banco de dados para fazer pesquisa em *big data*, enquanto a última faz acesso a páginas da enciclopédia online Wikipédia.

Outro tipo de aplicação que tem recebido bastante atenção são os sistemas de arquivos distribuídos, especialmente o Hadoop (Shvachko et al. 2010). Alguns dos trabalhos encontrados são: Cox et al. (2014); Fox et al. (2015); Kaewkasi & Srisuruk (2014); Loghin

Trabalho	Aplicação	Plataforma	Nós	Consumo(nós)
Mont Blanc (2011)	Demonstração	Samsung	2160	4-8 W
Andersen et al. (2009)	Arquivos distribuídos	Alix	21	3-6 W
Brown (2011)	Demonstração	Gumstix	192	n/d
Fürlinger et al. (2011)	HPL e STREAM	AppleTV	4	2-3 W
Humphries (2011)	Compilação de pacotes	PandaBoard	42	3-5 W
Keville et al. (2012)	NPB	PandaBoard	40	3-5 W
Balakrishnan (2012)	HPL, NPB, CoreMark e Ping Pong	Panda/RPi	6/2	1-5 W
Ou et al. (2012)	Servidor Web e transcodificação de vídeo	PandaBoard	4	2-5 W
Abrahamsson et al. (2013)	Demonstração	RPi	300	1-4 W
Rajovic et al. (2013)	Dhrystone, STREAM e SPEC CPU	SECO Q7	256	4 W
Tso et al. (2013)	Demonstração	RPi	56	1-4 W
Rajovic et al. (2014)	STREAM, Dhrystone e SPEC CPU	NVidia	128	17-32 W
Pfalzgraf & Driscoll (2014)	Vetores e matrizes	RPi	22	1-4 W
Cox et al. (2014)	HPL e Hadoop	RPi	64	1-4 W
Kaewkasi & Srisuruk (2014)	Hadoop	CubieBoard	22	4-10 W
Krpić et al. (2014)	Multiplicação de matrizes	CubieBoard	4	4-10 W
Fox et al. (2015)	Hadoop	RPi	2	1-4 W
Schot (2015)	Hadoop	RPi2	8	1-4 W
Velthuis (2015)	*Streaming* de vídeo	RPi2	3	1-4 W
Loghin et al. (2015)	Hadoop e banco de dados	ODROID	8	1-3 W
Cloutier et al. (2016)	HPL e STREAM	RPi2	32	1-4 W
Zhao et al. (2016)	*Web services* e Hadoop	Intel	35	1-2 W
Xu & Chang (2017)	NPB, Hadoop e servidor web	Marvell	16	10 W
NPi-Cluster	Servidor Web	RPi2	7	1-4 W

Tabela 3.1: Principais trabalhos relacionados com *clusters* de dispositivos de baixo consumo de energia (adaptado de Alves Filho et al. (2017)).

et al. (2015); Schot (2015); Zhao et al. (2016); Xu & Chang (2017). Os resultados mostram que o desempenho nesses cenários não foi tão bom quanto em outros, tendo os computadores com arquitetura x86 obtido uma melhor eficiência energética. Isso é justificado pela intensa troca de mensagens e entrada/saída que as operações de *MapReduce* necessitam e esse foi apontado como um ponto fraco dos dispositivos usados. Isso também se mostra verdadeiro em experimentos que manipulam vídeo (Velthuis 2015, Ou et al. 2012). Os resultados apontam que, à medida que a demanda cresce, o desempenho da CPU e da transmissão em rede afetam a experiência do usuário, necessitando assim de mais nós e comprometendo a eficiência energética ao contrário de outras arquiteturas.

3.2 Discussão sobre os trabalhos relacionados

Alguns pontos fortes e fracos podem ser tirados acerca dos trabalhos apresentados na seção anterior. O primeiro é que o objetivo é atingir eficiência energética de maneira estática, isto é, somente substituindo as máquinas de arquiteturas tradicionais pelas de

baixo consumo. Em todos os trabalhos citados não há proporcionalidade energética, isto é, os nós ficam ligados o tempo todo independente da carga de trabalho atual. Quando se quer adicionar ou remover um nó do *cluster* isso é feito de forma manual e individual.

Por outro lado, existem trabalhos na literatura (Schall & Hudlet 2011, Costa 2013, Xie et al. 2015) em que se busca um uso proporcional do hardware colocando as máquinas em modo suspenso, onde há um consumo menor de energia e se gasta menos tempo para retornar à ativa. O mesmo se aplica a máquinas com suporte ao mecanismo *Wake on Lan*, no qual um computador pode ser ligado ou acordado do modo suspenso através de pacotes de rede. Cada máquina neste modo precisa usar um pouco de energia pra manter a placa de rede funcionando para identificar a mensagem que indica que ela deve acordar.

O problema é que nesses modos nem todos os componentes do equipamento são desligados e o consumo ainda é grande se comparado com os dispositivos usados nos *clusters* mostrados que estão operando com sua total capacidade. Por exemplo, um computador com a arquitetura Intel Atom, que já foi desenvolvido com o objetivo de economia de energia, consome pouco mais de 3 W em modo suspenso, o que corresponde a uma Raspberry Pi funcionando e provendo algum tipo de serviço.

Há ainda outro aspecto relativo aos cenários onde os *clusters* com nós menos potentes não têm um bom desempenho. De forma geral, os maus resultados são apresentados de forma superficial, mostrando apenas os valores obtidos e os justificando pela limitação da CPU ou do alto tráfego na rede. Mas não se indica o que causa o alto consumo da CPU, porque o alto tráfego causa uma degeneração no desempenho ou ainda a partir de que momento é vantajoso em termos de eficiência energética utilizar nós de baixo consumo.

3.3 Considerações sobre os trabalhos relacionados

Este capítulo apresenta uma lista de trabalhos relacionados ao uso de máquinas de baixo consumo como meio para se obter uma melhor eficiência energética. Os primeiros trabalhos buscm comparar o desempenho individual e o consumo de energia em relação a máquinas de outras arquiteturas. Também é apresentada uma lista de trabalhos com *clusters* e que tipos de aplicações foram usados para testá-los.

Mas também é apontado que, apesar dessas soluções melhorarem a eficiência dos sistemas, ainda há espaço para um consumo bem menor se for adotada alguma estratégia dinâmica para o uso proporcional do hardware. A solução apresentada neste trabalho para este problema descrito é detalhada no próximo capítulo.

Capítulo 4

Arquitetura proposta

A partir das lacunas observadas nas pesquisas relacionadas a esta temática, este trabalho define e implementa uma arquitetura de *cluster* que utiliza um conjunto de placas de baixa potência energética. Essa arquitetura permite o uso proporcional dos servidores, escalando automaticamente o número de máquinas ligadas de acordo com a demanda de trabalho corrente a fim de obter uma maior eficiência energética. A solução proposta conserva as principais características de um sistema distribuído:

- Permitir o compartilhamento de recursos e serviços com usuários externos de maneira simples e com o melhor desempenho possível;
- Criar um sistema robusto com tolerância a falhas, evitando que o mau funcionamento em um dos nós inviabilize a disponibilidade do recurso;
- Funcionar de forma transparente, no qual o usuário externo não precise saber em que máquina o recurso ou serviço está hospedado;
- Proporcionar a escalabilidade do sistema, aumentando a oferta de recursos à medida que a demanda cresce;
- Buscar atingir uma qualidade de serviço condizente com as expectativas do usuário.

A forma como isso foi feito é explicado nas próximas seções.

4.1 A arquitetura NPi-Cluster

A arquitetura geral proposta, chamada NPi-Cluster, é mostrada na figura 4.1. A designação NPi foi dada em homenagem à rede de laboratórios Natalnet na qual a pesquisa foi realizada, e às máquinas Raspberry Pi que foram usadas na sua implementação.

A arquitetura é divida em três camadas de atuação: os nós provedores de serviço; o controle de alimentação de energia; e a gerência do *cluster*. Trata-se de um modelo de arquitetura centralizada em relação ao controle do número de nós ativos. Cada nó provê o serviço da aplicação de forma autônoma e independente, com redundância entre todos. Ou seja, o serviço disponibilizado é espelhado em todas as máquinas, que para a implementação atual conta com um hardware homogêneo.

Pode-se ver na parte superior da figura da arquitetura que cada nó provedor de serviço tem três componentes. Em primeiro o serviço da aplicação que está sendo disponibilizada

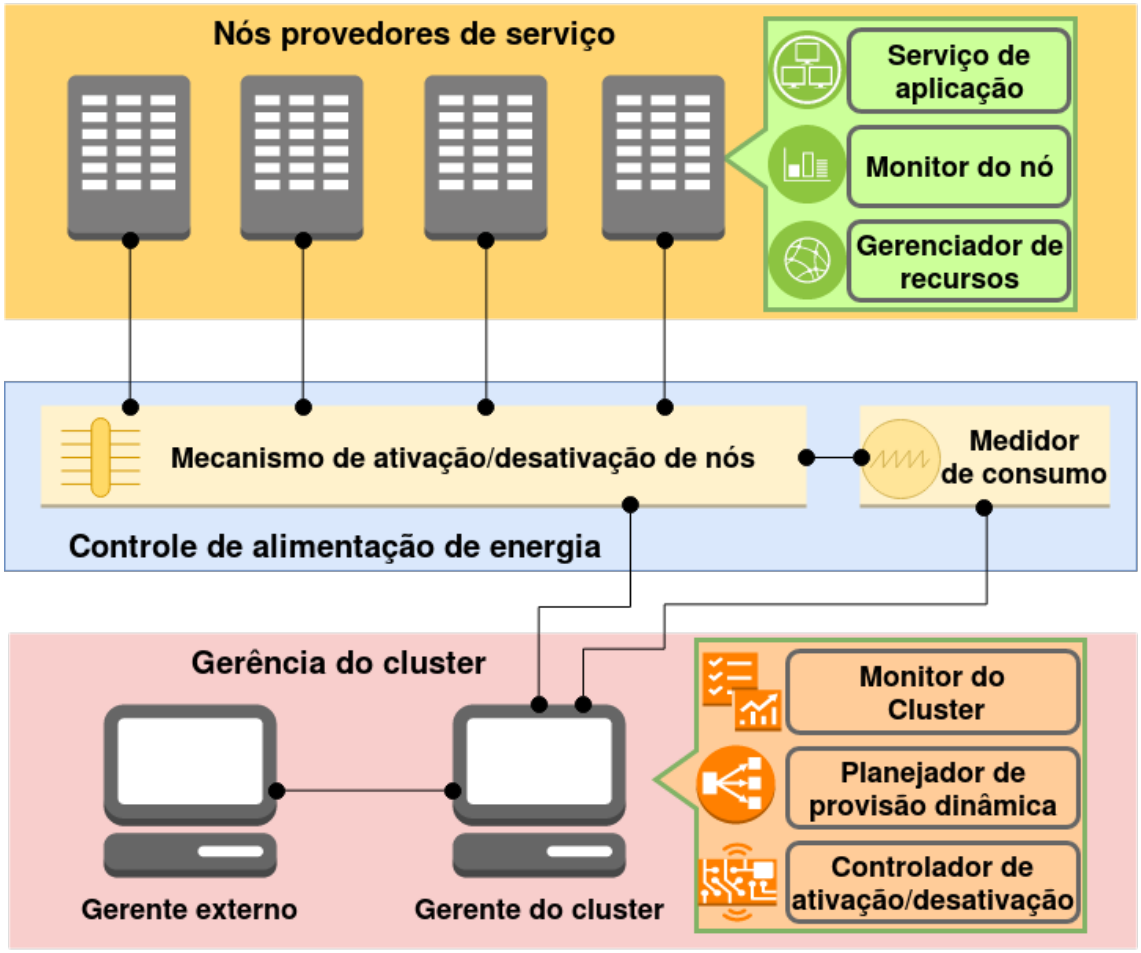

Figura 4.1: Arquitetura geral do NPi-Cluster.

para as máquinas clientes. Em segundo lugar um serviço para monitorar o nó no que concerne ao uso de seus recursos. Este monitor tem o objetivo de verificar periodicamente os indicadores que poderão servir de base para definir se ele é necessário ou não dada a demanda de trabalho atual. Em terceiro há um componente que gerencia os recursos do nó que são usados para definir a quantidade de trabalho que ele vai receber no balanceamento da carga.

Na parte intermediária está o controle de alimentação de energia, que é a interface responsável pelo fornecimento de corrente elétrica para os nós. O mecanismo de ativação/desativação de nós trata-se de um dispositivo físico que fornece ou interrompe o fornecimento de energia para os nós de serviço, sendo ele comandado pelo nó gerente. Além disso, também há um equipamento responsável por medir o consumo de energia, que também fornece os resultados para o nó gerente.

No que se refere à gerência do *cluster* há um nó principal que é definido como o

gerente, que também tem três componentes para o seu funcionamento. O monitor do *cluster* é responsável por coletar periodicamente os dados fornecidos pelo monitor de cada nó. Esses dados são consultados pelo segundo componente o planejador de provisão dinâmica. Com base no histórico dos dados de monitoramento ele define em que momento um novo nó deve ser ativado ou desativado. Quando for necessária alguma ação, o planejador aciona o controlador de ativação/desativação, que interage com o controle de alimentação de energia e com o controlador de recursos de cada nó que ficará ativo após a modificação para que a carga de trabalho seja rebalanceada.

Além do nó gerente também existe a figura de um gerente externo que é um componente opcional colocado como mecanismo de tolerância a falhas caso haja algum problema. Ele possui uma conexão direta com o gerente do *cluster* sendo capaz de reiniciá-lo caso ele pare por algum motivo.

4.2 Balanceamento de carga no NPi-Cluster

Na seção 2.1.1 foram discutidas três principais estratégias para balanceamento de carga baseados em: servidores distribuídos, despachantes intermediários e servidores DNS. Durante a execução dos experimentos foram testadas as três soluções para saber qual delas se adequava mais ao NPi-Cluster.

Na baseada em servidores distribuídos, os próprios nós que fornecem o serviço definiriam entre si quem iria atender às demandas de acordo com a carga de trabalho corrente. Para esse fim foi usado o gerenciador de recursos de *cluster* para escalabilidade e alta disponibilidade Pacemaker (ClusterLabs 2017), um software conhecido e utilizado em *clusters* de balanceamento de carga. Foram testados os dois modos (ativo/passivo e ativo/ativo) que funcionam com e sem replicação dos recursos. Em ambos os casos o uso desse software afetou de forma significantemente negativa o desempenho, proporcionando uma grande quantidade de pacotes trocados entre os nós e o uso excessivo do processador.

Na abordagem baseada em despachantes intermediários, um dos nós ficaria responsável por receber as requisições dos clientes e encaminhar ao servidor que teria melhor condições de atendê-lo. Nos testes foi usado o software de balanceamento de carga para alta disponibilidade HAProxy (HAProxy Technologies 2017). Ele aloca uma porta na máquina para receber e encaminhar as requisições de forma equivalente, aleatória ou de acordo com o número de respostas recebidas. O software pode fazer esse encaminhamento no nível de transporte ou de aplicação. Esta metodologia também não obteve um bom desempenho pois o nó despachante tornou-se o gargalo do serviço. Nos testes realizados durante a pesquisa essa solução não se mostrou escalável pois a partir da adição do segundo nó o aumento do tamanho do *cluster* não representou um acréscimo no número de requisições atendidas pelo mesmo.

A limitação de desempenho do hardware das máquinas fez com que essas duas metodologias usadas não tivessem sucesso. Seria necessária uma forma de realizar o balanceamento de carga sem provocar um aumento na demanda de trabalho dos servidores, o que foi obtido com a solução baseada em servidores DNS. Com essa técnica, quando os clientes fazem uma consulta pelo nome do um *host*, o servidor DNS retorna endereços di-

ferentes, distribuindo as requisições entre os nós. Dessa forma os nós do *cluster* não tem qualquer sobrecarga para distribuir as tarefas e nem é necessário usar outro elemento no sistema, uma vez que a consulta ao servidor de nomes é uma etapa quase que obrigatória para o acesso a um servidor.

Entretanto existem dois problemas com esse tipo de solução. O primeiro é a sensibilidade do servidor DNS à alteração de configuração do *cluster*. Um servidor de nomes pode ficar em um equipamento fora do alcance do *cluster*. Dessa forma a cada inclusão ou remoção de nós o servidor DNS teria que ter sua configuração alterada, algo que pode ser inviável ou indesejado pelos administradores da rede. O segundo problema é que o DNS apenas associa o nome ao endereço e não sabe, por exemplo, se um cliente chegou a fazer uma requisição ao servidor, se o servidor estava funcionando, se o serviço estava disponível ou se o cliente chegou a ter sua requisição atendida. Assim esse servidor não teria como rebalancear a carga caso algum nó estivesse sobrecarregado.

Para resolver esses problemas, a solução adotada para o balanceamento de carga no NPi-Cluster é um misto entre DNS e distribuição de recursos. No caso, o servidor DNS é configurado com uma base estática de endereços IP de tamanho maior ou igual ao número máximo de nós servidores do *cluster*. Este conjunto de endereços é dividido entre os nós ativos de forma que um mesmo nó pode responder por mais de um simultaneamente. Isso é exemplificado na figura 4.2.

Nesse cenário a base conta com 6 endereços IP (etiquetas vermelhas) e o *cluster* com 6 nós servidores, estando inativos os de cor cinza. Em um primeiro momento (parte de cima) apenas um nó está ativo e todos os endereços estão associados a ele, que atende a todas as requisições. Em um segundo momento onde há uma demanda maior, três nós estão ativos e os endereços são distribuídos uniformemente com 2 IPs para cada. Quando os clientes consultam o endereço no servidor DNS ele responde com os mesmos 6 endereços, mas nesse caso as requisições são distribuídas entre os três nós ativos. Dessa forma o gerente do *cluster* pode monitorar os nós e rebalancear a carga tanto de forma estática modificando a associação de endereços sem ativar novos servidores, quanto de forma dinâmica adicionando ou removendo os nós e redistribuindo os endereços.

Para a implementação atual do NPi-Cluster, considerou-se que o servidor de nomes usa o algoritmo *Round-Robin* para dividir os endereços. Quando a demanda cresce e um novo nó é ativado a quantidade de endereços é divido igualmente entre os nós ativos. Da mesma forma, quando a demanda diminui e um nó é desativado, os endereços IP que antes estavam associados a ele são redistribuídos entre os nós restantes. Isto é, o *cluster* usa a técnica descrita como *Vary-On Vary-Off*.

4.3 Provisão dinâmica de nós

Outro aspecto importante na arquitetura NPi-Cluster é como se dá a provisão dinâmica de nós. A figura 4.1 mostra que o nó gerente do *cluster* tem um componente que monitora constantemente os nós. A provisão dinâmica, isto é, a ativação ou desativação durante o funcionamento, depende de limites que devem ser estabelecidos para considerar o sistema em um determinado estado. Quando chega nesse estado o planejador de provisão deve usar um critério para determinar se está no momento de alterar o número de nós ativos.

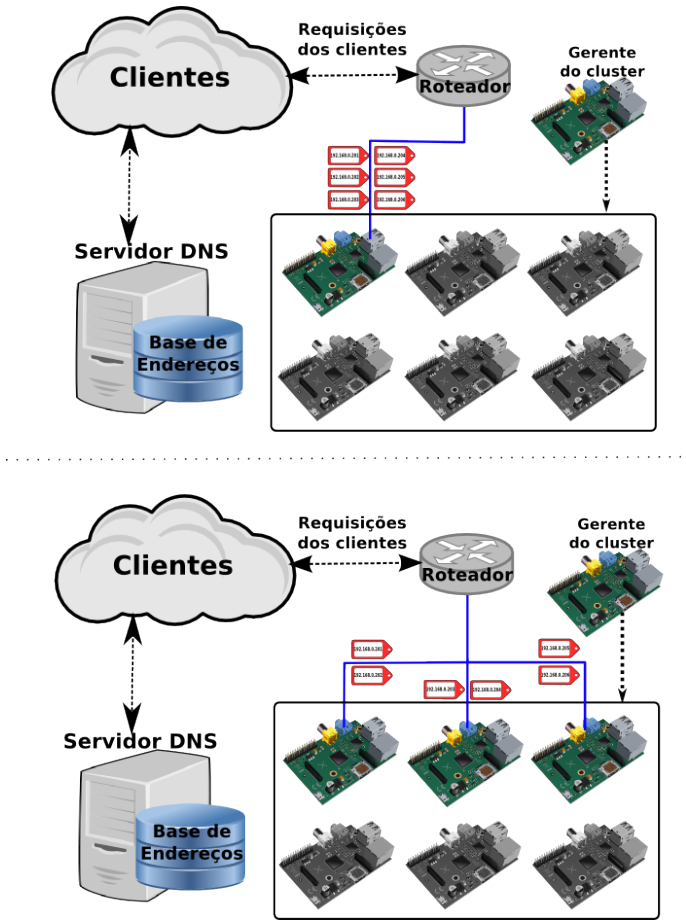

Figura 4.2: Exemplos de balanceamento de 6 endereços IP com 1 e 3 nós ligados (traduzido de Alves Filho et al. (2017))

Para evitar um comportamento oscilante onde nós sejam ligados e desligados a todo instante é preciso que um mesmo comportamento se repita um determinado número de vezes para caracterizar uma sobrecarga ou subutilização. O Algoritmo 1 detalha o funcionamento da provisão dinâmica de nós adotada.

São definidos quatro estados de demanda de acordo com a carga de trabalho. Se a carga está acima da capacidade planejada para o *cluster* é atribuído um estado de sobrecarga ou crítico. O estado crítico significa que além de estar acima da capacidade, a carga atual também pode afetar os parâmetros de QoS, como o tempo de resposta ou a porcentagem de falhas de requisição. Já quando a carga está abaixo de um limiar significa que há espaço para uma maior economia de energia, sendo atribuído um estado de subutilização. Se a demanda é considerada adequada para o número de nós ativos considera-se um estado normal e nenhuma ação é realizada.

Cada vez que um estado diferente do normal é detectado, um contador de um estado é incrementado e os demais são decrementados. No momento em que o contador de um estado chega a um limite a configuração do *cluster* é alterada com a inclusão ou exclusão de um nó, e os contadores são reiniciados.

Algoritmo 1 Provisão dinâmica de nós para o NPi-Cluster (Alves Filho et al. (2017)).

Entrada: características do *cluster*

$Nós = \{RPi_1, RPi_2, \ldots, RPi_n\}$ conjunto de n nós servidores
$Endereços = \{IP_1, IP_2, \ldots, IP_e\}$ conjunto de $e \geq n$ endereços na base do serviço DNS
$t_{sobrecarga}, t_{crítico}$ taxas para considerar o *cluster* em estado de sobrecarga ou crítico
$r_{subutilizado}, r_{sobrecarga}, r_{crítico}$ repetições de um estado para que o controlador atue
$min\_nós$ número mínimo de nós que devem permanecer ativos
$tempo\_espera$ tempo de espera entre as verificações de estado

Inicialização :

1: $Ativos = Nós$ conjunto de nós ativos
2: $Inativos = \varnothing$ conjunto de nós inativos
3: $c_{subutilizado}, c_{sobrecarga}, c_{crítico} = 0$ contadores do número de repetições de cada estado
4: Atribua IPs de *Endereços* aos nós em $Ativos$

Laço de controle :

5: **Enquanto** *verdadeiro* **faça**
6: $u_{atual} = média(u_i)$ onde u_i é a utilização do serviço no nó $RPi_i \mid RPi_i \in Ativos$
7: **Se** $(u_{atual} > t_{crítico})$ **então**
8: Incremente $c_{sobrecarga}$ e $c_{crítico}$, decremente $c_{subutilizado}$
9: **Senão Se** $(u_{atual} > t_{sobrecarga})$ **então**
10: Incremente $c_{sobrecarga}$, decremente $c_{subutilizado}$ e $c_{crítico}$
11: **Senão**
12: Compute $u^*_{atual} = \sum u_i / (|Ativos| - 1)$ a taxa de utilização se um nó for desativado
13: **Se** $(u^*_{atual} < t_{sobrecarga}$ **e** $|Ativos| > min\_nós)$ **então**
14: Incremente $c_{subutilizado}$, decremente $c_{sobrecarga}$ e $c_{crítico}$
15: **Senão**
16: Decremente $c_{subutilizado}$, $c_{sobrecarga}$ e $c_{crítico}$
17: **Fim Se**
18: **Fim Se**
19: **Se** $(c_{sobrecarga} > r_{sobrecarga}$ **ou** $c_{crítico} > r_{crítico})$ **e** $|Inativos| > 0)$ **então**
20: Ative um nó $RPi_j \in Inativos$
21: $Ativos = Ativos \cup \{RPi_j\}$
22: $Inativos = Inativos \setminus \{RPi_j\}$
23: **Senão Se** $(c_{subutilizado} > r_{subutilizado}$ **e** $|Ativos| > min\_nós)$ **então**
24: Desative um nó $RPi_k \in Ativos$
25: $Ativos = Ativos \setminus \{RPi_k\}$
26: $Inativos = Inativos \cup \{RPi_k\}$
27: **Fim Se**
28: **Se** ($|Ativos|$ foi alterado) **então**
29: Redistribua os IPs para os nós em $Ativo$ proporcionalmente
30: Reinicie os contadores $c_{subutilizado}, c_{sobrecarga}, c_{crítico} = 0$
31: **Fim Se**
32: Aguarde por $tempo\_espera$ unidades de tempo
33: **Fim Enquanto**

O algoritmo necessita de parâmetros de entrada. Para o balanceamento de carga tem-se o conjunto dos nós servidores disponíveis no *cluster* e os endereços que ficam na base do servidor DNS para o balanceamento de carga. Também é necessário estabelecer limites para a taxa de utilização dos serviços que quando ultrapassados configuram um estado crítico ou de sobrecarga e o número de vezes que cada estado deve se repetir para que seja realizada alguma ação de mudança de configuração. Define-se também o intervalo de tempo entre as verificações de monitoramento e o número mínimo de nós que devem permanecer ligados a todo tempo.

Na inicialização (linhas de 1 a 4) vê-se que o conjunto de nós Ativos começa com todos os nós do *cluster*, os contadores de contagem de estado zerados e os IPs da base do DNS distribuídos entre todos. O laço de controle, que vai até o fim do algoritmo, repete-se indefinidamente durante a execução do controlador de provisão. Primeiramente, na linha 6, é calculado o índice u_{atual} como a média das taxas de utilização do serviço em todos os nós ativos. Esse valor é comparado com os limites estabelecidos para caracterização dos estados crítico e de sobrecarga (linhas 7 a 10). Se isso não ocorrer, na linha 12 verifica-se se o *cluster* está subutilizado calculando uma taxa u^*_{atual}. Ela representa qual seria a demanda atual se o *cluster* tivesse um nó ativo a menos. Se for verificado que a taxa u^*_{atual} não deixa o *cluster* sobrecarregado significa que é possível desativar um dos nós sem comprometer o desempenho, caracterizando assim uma subutilização (linhas 13 e 14). Caso não ocorra nenhuma das situações anteriores, o *cluster* está em um estado normal de operação com uma demanda adequada à sua composição.

Após isso, a contagem de repetição dos estados é comparada com o número estabelecido como suficiente para uma alteração da configuração, podendo levar ao acréscimo (linhas 19 a 22) ou remoção (linhas 23 a 26) de um nó. Se algum desses cenários ocorrer, é necessário redistribuir os endereços IP entre os novo conjunto de nós ativos e reiniciar os contadores (linhas 28 a 30). Por fim, é feita uma pausa (linha 32) antes de reiniciar o processo de monitoramento no laço de controle.

O algoritmo pode ser considerado simples e abstrato, não exigindo muito processamento do nó gerente nem muita sobrecarga de comunicação com os nós servidores. Os parâmetros de entrada permitem que o administrador de rede possa definir um intervalo maior ou menor de tempo para reagir ou definir quando há sobrecarga ou subutilização. Ele é adaptável a qualquer aplicação na qual seja possível aferir uma taxa de utilização atual e os limites de alteração de estado. Também não há nada que impeça que os parâmetros de entrada possam ser alterados dinamicamente com o decorrer do tempo.

4.4 Resumo da arquitetura NPi-Cluster

A arquitetura NPi-Cluster é baseada em três elementos distintos: os nós provedores de serviço, o controle de alimentação de energia e a gerência do *cluster*. A distribuição de carga é feita replicando os serviços em todos os nós e usando o servidor DNS para distribuir as requisições entre endereços IP especificados. Esses IPs são flutuantes, sendo atribuídos dinamicamente de acordo com o número de nós ativos. Um algoritmo de provisão dinâmica de nós é executado de forma contínua no gerente, verificando constantemente a taxa de utilização de cada nó e atribuindo um estado de trabalho para o *cluster*. Quando

um mesmo estado é detectado repetidamente através dos contadores, a sua configuração pode ser alterada com a inserção ou remoção de nós servidores.

A solução baseada em balanceamento de clientes pelo servidor DNS e endereços flutuantes atribuídos dinamicamente tornam a gerência do *cluster* mais simples e não causa uma sobrecarga excessiva nos nós, contribuindo para o aumento de seu desempenho. A arquitetura, a distribuição de carga e o algoritmo de provisão dinâmica são apresentados de forma genérica e podem ser implementados com outros tipos diferentes de máquinas e aplicações. O próximo capítulo apresenta com mais detalhes como essa arquitetura foi implementada usando dispositivos Raspberry Pi e como aplicação de exemplo um servidor de arquivos de páginas web estáticas.

Capítulo 5

Implementação

A arquitetura NPi-Cluster foi implementada fisicamente usando dispositivos Raspberry Pi. Pesquisas sobre desempenho energético conduzidas com diferentes tipos de dispositivos ARM apontaram o Raspberry Pi como a melhor escolha para *clusters* com dispositivos de baixo consumo (Cloutier et al. 2014, Cloutier et al. 2016). Entre as razões para isso estão a boa eficiência energética e a relação custo/benefício. No Brasil, já com os impostos, uma placa RPi pode ser comprada por cerca de 200 reais, sendo ela do modelo mais atual com processador *quadcore* 64-bits, 1Gb de memória RAM, conexão de rede cabeada, Wifi e bluetooth, placa de áudio e vídeo, além de outras conexões e os pinos para GPIO. Ele também conta com uma distribuição Linux completa com uma vasta gama de aplicativos gráficos e suporte para disponibilizar diversos serviços de rede.

Outro fator importante é a temperatura máxima atingida pela placa e seu processador. Como descrito na seção 2.2, existem métricas como PUE e DCIE para aferir a eficiência de um data center que são dadas pela relação entre os equipamentos computacionais e a energia total consumida. Um dos elementos mais significativos nesse cálculo que reduzem o valor de eficiência é o gasto com refrigeração para evitar superaquecimento e queima das máquinas e seus componentes (Heller et al. 2010). Sistemas com arquitetura baseada em ARM geralmente apresentam temperaturas mais baixas sendo possível utilizar um resfriamento passivo do próprio ambiente sem necessidade de outros equipamentos para este fim (Cox et al. 2014). Nos testes realizados neste trabalho a temperatura de um processador do RPi não excedeu os $45°C$ sem qualquer componente para refrigeração. Essa é uma temperatura considerada normal para processadores mais frios da arquitetura x86 quando fazem uso de *cooler* e dissipador.

Além disso, também é importante destacar o tamanho ocupado por uma Raspberry Pi, que é praticamente igual a de um cartão de crédito. A figura 5.1 mostra a forma como um *cluster* está montado com 7 nós. As placas foram posicionadas e fixadas como um quadro e pendurado na parede.

5.1 Montagem do *cluster*

A montagem do NPi-Cluster é feita conforme mostra a figura 5.2. Os nós provedores de serviço e o nó gerente formam uma rede local (linhas pretas) conectada por um Switch Gigabit Ethernet. Esse Switch está ligado a um roteador que interliga o *cluster* aos clientes

Figura 5.1: NPi-Cluster montado com 7 nós (Alves Filho et al. 2017).

e ao servidor DNS. Foi necessário usar um Switch Gigabit porque cada Raspberry Pi possui uma placa de rede de 100 Mbits/s e a máquina que simula os clientes necessitaria de uma largura de banda maior para não ser um gargalo. De fato, nos experimentos realizados com 7 nós o tráfego nessa máquina chegou a atingir 350 Mbits/s.

Para receber alimentação de energia (linhas vermelhas) há uma fonte que fornece uma corrente contínua a uma tensão de 5 V. Antes de ser distribuída a energia passa por um medidor que está ligado ao nó 1, que é o gerente do *cluster*, através de uma conexão USB, de onde os valores de consumo são calculados. Para os cálculos usados nos experimentos o consumo do Switch não é contabilizado, mas segundo o fabricante seu consumo máximo chega a 5 W em plena utilização, o que não aconteceu durante os testes.

Após passar pelo medidor, a energia alimenta diretamente o nó gerente e uma placa de circuito com relés que permitem ou não a passagem da corrente elétrica para cada nó. Esses relés são controlados através dos pinos GPIO digitais presentes no Raspberry Pi gerente. As conexões elétricas através dos relés podem ter contatos normalmente abertos ou normalmente fechados. A diferença entre os dois é que um normalmente aberto só fornece energia quando o controlador habilita a saída digital, enquanto um normalmente fechado fornece energia continuamente e interrompe quando a entrada digital do controlador é habilitada.

Para a implementação realizada, os contatos são normalmente fechados, o que fornece mais um mecanismo de tolerância a falhas. Se o nó gerente falhar, ou se a ligação entre o pino digital e a placa de relés falhar, ou a própria placa de relés falhar o *cluster* não

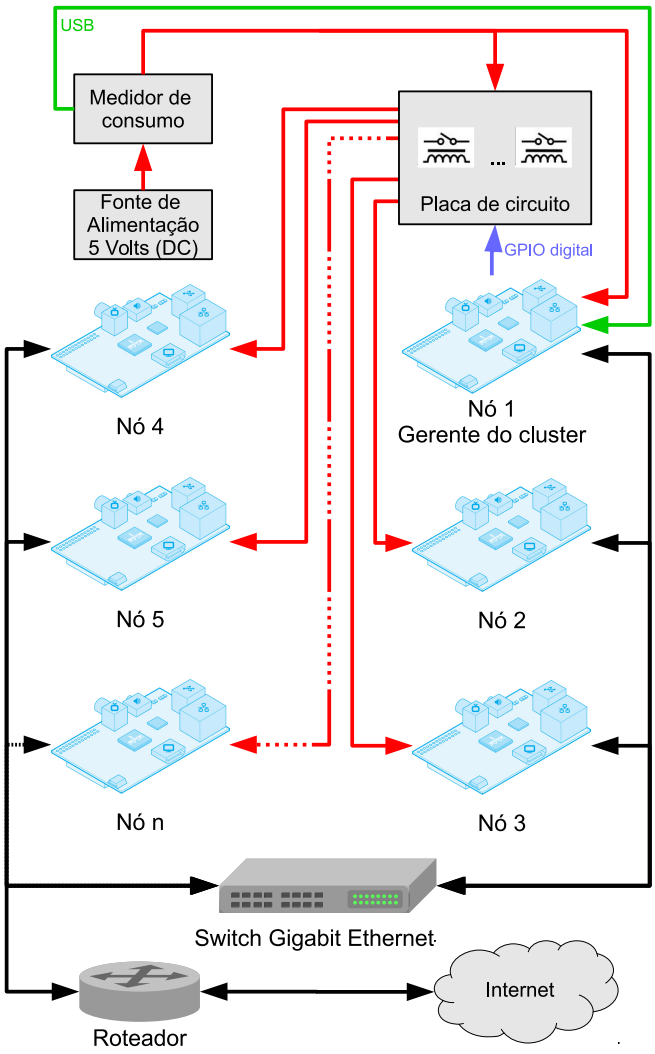

Figura 5.2: Conexões elétricas e de rede em um NPi-Cluster (traduzido de Alves Filho et al. (2017)).

vai parar completamente, mas vai perder seu aspecto dinâmico até que o problema seja resolvido. O esperado é que isso não ocorra, já que a solução proposta não sobrecarrega o gerente, e os relés da placa podem ser trocados quando estiverem próximos ao fim da vida útil esperada. Se um nó servidor apresentar falha, ele não conseguirá ser monitorado e, consequentemente pode ser desligado e os endereços associados a ele transferidos para outro nó.

Quando o *cluster* é iniciado, os contatos normalmente fechados ativam todos os nós que aos poucos são desativados pelo provisionador do gerente caso não haja demanda. Pelo menos duas máquinas ficam ligadas permanentemente, uma para prover o serviço e o gerente. Cada nó individualmente, independente da demanda, consome menos 4 W, chegando a consumir menos de 2 W quando está sem carga, confirmando resultados descritos em outros estudos, como o de Kaup et al. (2014).

5.2 Medição da carga de trabalho em servidores web

A aplicação escolhida para testar a arquitetura NPi-Cluster foi o serviço de páginas web estáticas distribuídas com balanceamento de carga. Ela é adequada para o modelo proposto, pois para todo cliente que requisita uma página web é necessário antes passar pelo servidor DNS e a quantidade de clientes pode ser balanceada entre os servidores ativos. Para fazer a provisão dinâmica é necessário estabelecer os critérios para definir o estado atual do *cluster*.

Uma possível medida para isso seria o percentual de utilização da CPU ou da memória. Entretanto, testes realizados neste trabalho mostraram que o percentual de ocupação do processador é muito inconstante e não confiável, pois varia muito em um curto intervalo de tempo. Por sua vez, a memória ocupada pelos processos do servidor web dos nós não chegou a um percentual que ameaçasse o desempenho do sistema. Procurou-se, então, outra maneira que fosse mais relacionada à aplicação de servidores web.

Um fator determinante para avaliar a qualidade da infraestrutura de um servidor web é o tempo para que a página seja transferida para o cliente. Uma pesquisa com usuários de comércio eletrônico na Inglaterra e Estados Unidos mostra que mais da metade dos entrevistados apontaram a lentidão para carregar as páginas como principal causa para desistência de uma compra (Strawson & Ayres 2012). Daí conclui-se que um tempo elevado para obter a resposta do servidor web é um bom critério para definir se o mesmo está ou não sobrecarregado.

O protocolo HTTP usado na transferência de páginas web segue um conjunto de passos bem definidos. Primeiro, o cliente envia uma requisição através do navegador que é recebida na máquina de destino, geralmente na porta 80. No servidor, a requisição é repassada ao processo que provê as páginas e, se não houver fila, sofre um processo de *parsing* para verificar sua corretude e é processado para gerar uma resposta, que é recebida pelo cliente e exibida no navegador. A figura 5.3 mostra esse processo, no qual pode-se separar dois intervalos distintos: um tempo de conexão que vai de quando o cliente solicita uma página até a requisição chegar ao processo do servidor web; e o tempo de envio de resposta em que o servidor processa a requisição e devolve a resposta ao cliente.

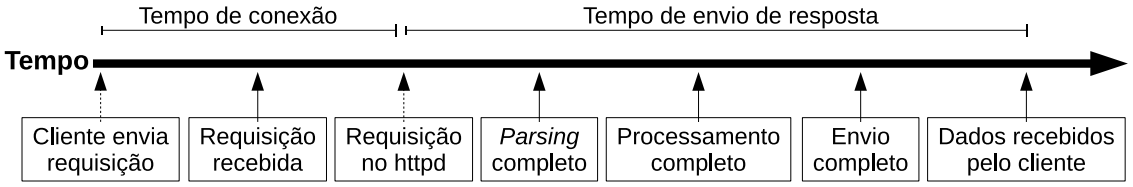

Figura 5.3: Fases do processamento de uma requisição HTTP (adaptado de Dilley et al. (1998)).

Pode-se considerar que tempos de conexão são semelhantes em uma mesma máquina, uma vez que o processo é o mesmo para todas as requisições, independente do conteúdo. Já o tempo de resposta depende se o arquivo for estático ou dinâmico. Para conteúdo estático (páginas HTML, folhas de estilo CSS, imagens, *scripts*, etc.) o tempo de resposta

depende basicamente do tamanho do arquivo que será enviado ao cliente. Para conteúdo dinâmico o tempo não pode ser medido de forma mais precisa porque depende das operações que serão feitas, como acesso a banco de dados ou processamento de informações. Mesmo assim, após o processamento o tempo de envio também depende do tamanho do arquivo que será enviado como resposta ao cliente. Pode-se aproximar o tempo total de resposta para um arquivo qualquer como:

$$t_{resposta}(arquivo) \approx t_{conexão} + t_{envio}(arquivo) \tag{5.1}$$

O tempo de conexão pode ser medido fazendo a requisição de um arquivo sem conteúdo que teria um tempo de resposta irrelevante. Assim, tomando a equação 5.1 e considerando ($t_{envio}(\varnothing) = 0$) o tempo de conexão é dado por:

$$t_{resposta}(\varnothing) \approx t_{conexão} + t_{envio}(\varnothing) = t_{conexão} \tag{5.2}$$

Para definir a relação tempo de envio x tamanho do arquivo foram feitos testes com uma das máquinas do *cluster* servindo arquivos de 3 KB a 30 KB variando de 3 em 3 KB. Outra máquina fez requisições a esse servidor para a mesma página durante 5 minutos e após isso foi calculada a média de tempo para a resposta de cada requisição. Os resultados das médias de 10 repetições deste processo estão no gráfico da figura 5.4 que foi suavizado por um filtro de Bézier (Chang & Rocchetti 1989).

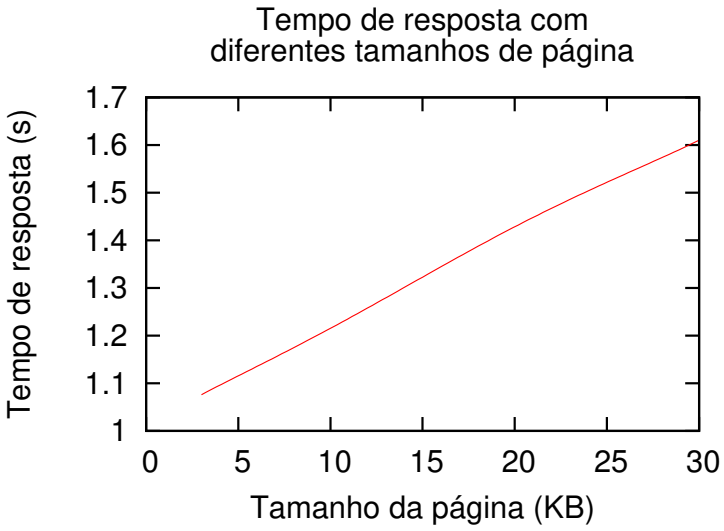

Figura 5.4: Relação tempo de resposta do servidor web x tamanho do arquivo no Raspberry Pi (traduzido de Alves Filho et al. (2017)).

Há uma relação linear entre o tempo de resposta e o tamanho do arquivo, o que torna possível estabelecer que $t_{envio}(arquivo) \approx t_{por\ byte} * tamanho(arquivo)$. Substituindo essa relação e a equação 5.2 na equação 5.1 tem-se:

$$t_{resposta}(arquivo) \approx t_{resposta}(\varnothing) + t_{por\ byte} * tamanho(arquivo) \tag{5.3}$$

Esse tempo de resposta considera que o nó servidor não está atendendo a nenhuma outra requisição. Em uma situação real um servidor pode receber dezenas ou centenas delas, e nem todas são atendidas simultaneamente. O servidor constrói uma fila para processar sequencialmente todos os pedidos. Essa fila pode ser medida pelo número de conexões estabelecidas na porta 80, que é a padrão para acesso web. Para definir o nível de utilização de um nó qualquer, calcula-se o tempo de atraso esperado para que uma próxima requisição seja atendida, isto é, a soma dos tempos de respostas de todas as solicitações que estão na fila:

$$t_{atraso} \approx \sum_{arquivo_f \in Fila\ de\ requisições} t_{resposta}(arquivo_f) \tag{5.4}$$

Para determinar o tempo de atraso é necessário calcular o tempo de resposta de todos os arquivos que estão na fila de conexões. O problema é que até ser processada pelo servidor não se sabe o tamanho do arquivo que será enviado como resposta. Mas se o administrador da rede tiver ou adquirir um conhecimento sobre o tráfego pode-se inferir um tamanho médio de arquivo e considerar que no agregado cada requisição vai levar o tempo necessário para atender a um arquivo com esse tamanho. Atualizando então a equação 5.4:

$$t_{atraso} \approx \mid Fila\ de\ requisições \mid \times t_{resposta}(arquivo\ médio) \tag{5.5}$$

A taxa de utilização do *cluster* proposta no algoritmo de provisão dinâmica de nós usa o valor da variável t_{atraso} como demanda corrente de cada nó ativo. Para atribuir um estado de sobrecarga o tempo médio de atraso deve ultrapassar um limite determinado de forma repetitiva durante um intervalo de medições.

5.3 Características da implementação da arquitetura

A arquitetura NPi-Cluster foi implementada com nós Raspberry Pi pois eles apresentarem algumas vantagens. Uma delas é a presença dos pinos GPIO que são usados pelo nó gerente do *cluster* para ativar ou desativar nós dinamicamente. Para isso, a passagem de corrente elétrica para alimentação dos nós é regulada por uma placa de relés com contatos normalmente fechados, que dão um nível a mais de tolerância a falhas.

Já o critério de detecção de sobrecarga foi adaptado para a aplicação usada, que é o acesso a páginas web. Após algumas manipulações chegou-se a uma fórmula onde o tempo de resposta de uma requisição é dado: 1) pelo tempo de conexão, que é o mesmo do tempo de resposta de um arquivo vazio; e 2) pelo tempo de envio de resposta ao cliente, calculado como uma função linear entre tamanho do arquivo solicitado e tempo esperado. Para facilitar esse processo, se calcula o tempo de espera para um nó tomando-se o tamanho médio esperado dos arquivos e o número de conexões para a porta do serviço de páginas web.

No próximo capítulo são mostrados os resultados dos testes realizados com a implementação descrita neste capítulo a fim de verificar a eficiência do *cluster* e seu comportamento em relação ao uso proporcional do hardware.

Capítulo 6

Experimentos e Resultados

Após mostrar a arquitetura e a implementação do NPi-Cluster com placas Raspberry Pi, seus objetivos em relação ao balanceamento de carga, escalabilidade e eficiência energética, este capítulo apresenta dois experimentos realizados para verificar sua viabilidade.

O primeiro testa a diferença de desempenho entre o modelo Raspberry Pi versão 2 que conta com processador multinúcleo e o seu antecessor que possui apenas um. Os resultados dos desempenhos dos dois *clusters* são comparados com os obtidos por máquinas de arquiteturas ARM e x86 publicados anteriormente na literatura. Já no segundo experimento o foco é analisar o desempenho quando se habilita a provisão dinâmica.

Em ambos os casos, os *clusters* foram montados como mostra a figura 5.1, com 7 placas Raspberry homogêneas e o Switch Gigabit Ethernet. Todas as placas foram conectadas por rede cabeada, contavam com a distribuição Linux oficial Raspbian e os softwares providos pelo seu gerenciador de pacotes com suas configurações padrão. Os arquivos hospedados no servidor web foram replicados em cada máquina e não foram alterados durante os testes. As páginas web usadas não criavam sessões nem armazenavam informações em *cookies*.

No primeiro experimento, em que não há provisão dinâmica, os 7 nós são usados como servidores web, enquanto no segundo 1 nó é usado como gerente e os outros 6 como servidores. A máquina usada para simular os clientes tinha maior capacidade de recursos de processamento, rede e memória e não representou um gargalo para os experimentos. Os componentes de software dos nós de serviço e do gerente foram implementados nas linguagens Shell Script e Python. Os arquivos referentes a estes experimentos encontram-se disponíveis na URL `https://github.com/gitlordi/npi-cluster/`.

6.1 Experimento 1 - desempenho de *cluster* estático

O trabalho de Aroca & Gonçalves (2012) comparou o desempenho de máquinas x86 e ARM em três tipos diferentes de aplicação. Em uma delas, foram geradas requisições HTTP simultâneas para uma página web de 3 KB. A taxa de requisições simultâneas testadas variava de 1 a 1001, incrementada de 25 em 25, através da ferramenta Apache Benchmark (Apache Software Foundation 2017*a*). Ao final, a ferramenta oferece como resultado algumas métricas como: desempenho em termo de número de requisições atendidas por segundo; *throughput* com a quantidade de dados transferidos por segundo; tem-

pos de atraso (latência mínima, média e máxima) para concluir as requisições; e número de requisições com falha. Além disso, o trabalho também apresenta resultados de medida de temperatura e consumo de energia. Como o número de requisições com falha foi muito baixo em todos os cenários e a temperatura não tem grandes variações, seus valores não serão mostrados.

Esse primeiro experimento compara os resultados obtidos no trabalho citado com *clusters* montados com dois modelos diferentes de Raspberry Pi, que serão chamados de RPi1 e RPi2. A tabela 6.1 mostra as configurações dos modelos RPi1 e RPi2. Nesse momento o objetivo é avaliar a viabilidade da solução usando os dois modelos de placa sem usar o mecanismo de provisão dinâmica de nós. Os resultados são apresentados usando os gráficos mostrados nas figuras a seguir, todos suavizados usando curvas de Bézier. Para extrair os valores dos gráficos os testes foram repetidos 10 vezes e em cada execução eram geradas 150.000 requisições à mesma página.

Modelo	Características
Raspberry Pi Modelo 1	Processador ARM1176JZFS single core 800 MHz, 256 MB RAM, 4 GB SDCard
Raspberry Pi Modelo 2 B	Processador ARM Cortex-A7 quadcore 1 GHz, 1 GB RAM, 16 GB SDCard

Tabela 6.1: Características das máquinas usadas no experimento 1.

O primeiro gráfico, na figura 6.1, mostra o consumo médio de energia em cada cenário. Nota-se que não há uma variação grande no valor com o aumento da demanda. Apenas nos valores do *cluster* RPi2 há um menor consumo quando a demanda é menor, em parte devido à política de DVFS do processador. O consumo de ambos os *clusters* ficou próximo aos 14 W, que é quase 4 vezes menor que a máquina com processador Xeon e abaixo da máquina com processador Turion. Ela foi superior às outras, mas isso é compreensível porque são 7 dispositivos ligados ao invés de 1 como é nos demais.

É interessante notar que apesar de possuir um processador com 4 núcleos em cada nó, com uma frequência de operação maior e também uma quantidade maior de memória o consumo dos dois modelos de máquinas Raspberry Pi foi praticamente o mesmo, divergindo apenas um pouco nas demandas menores ou maiores.

A figura 6.2 apresenta os resultados relativos ao desempenho com o número de requisições atendidas com sucesso por segundo. O desempenho de ambos os *clusters* foi considerado satisfatório, com as máquinas RPi2 tendo o melhor desempenho e as RPi1 ficando abaixo apenas do computador com processador Xeon com 4 núcleos.

Chama a atenção nesses resultados a diferença de desempenho entre os dois *clusters*, que chega a mais de 4 vezes. Apesar do RPi2 ter 4 núcleos e com uma frequência um pouco superior não há como um sistema ou processo executar 100% em paralelo em um processador multinúcleo de acordo com as leis de Amdhal e Gustafson (Gustafson 1988). Uma análise mais detalhada sobre o desempenho mostra que esta diferença acontece por causa da forma como as interrupções da placa de rede são tratadas.

As placas RPi não foram projetadas com o intuito de serem usadas como servidores de rede. O tratamento das interrupções da placa de rede envolve a cópia dos dados que são

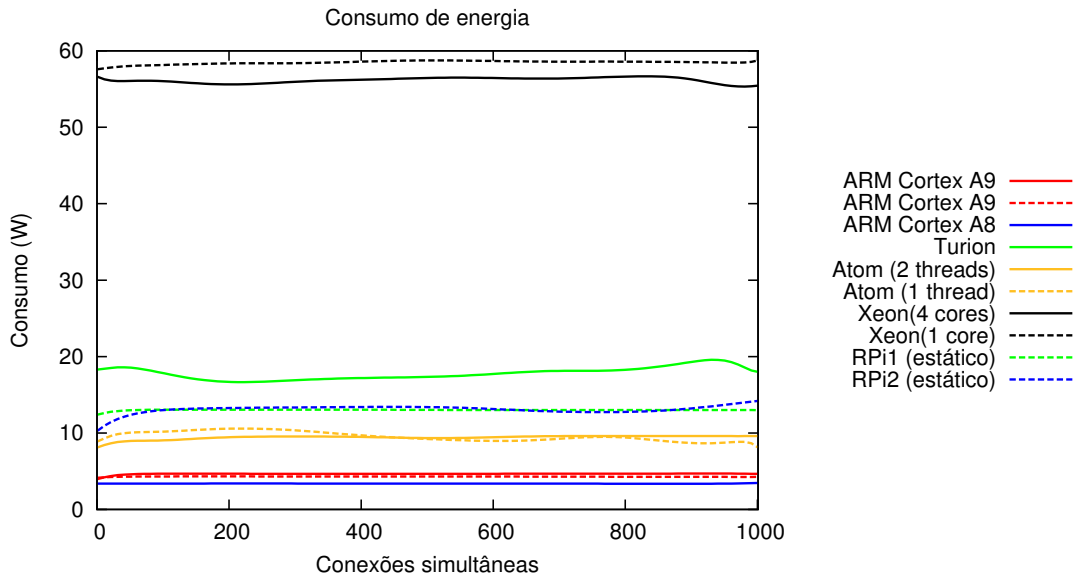

Figura 6.1: Consumo médio de energia para o experimento 1 (traduzido de Alves Filho et al. (2017)).

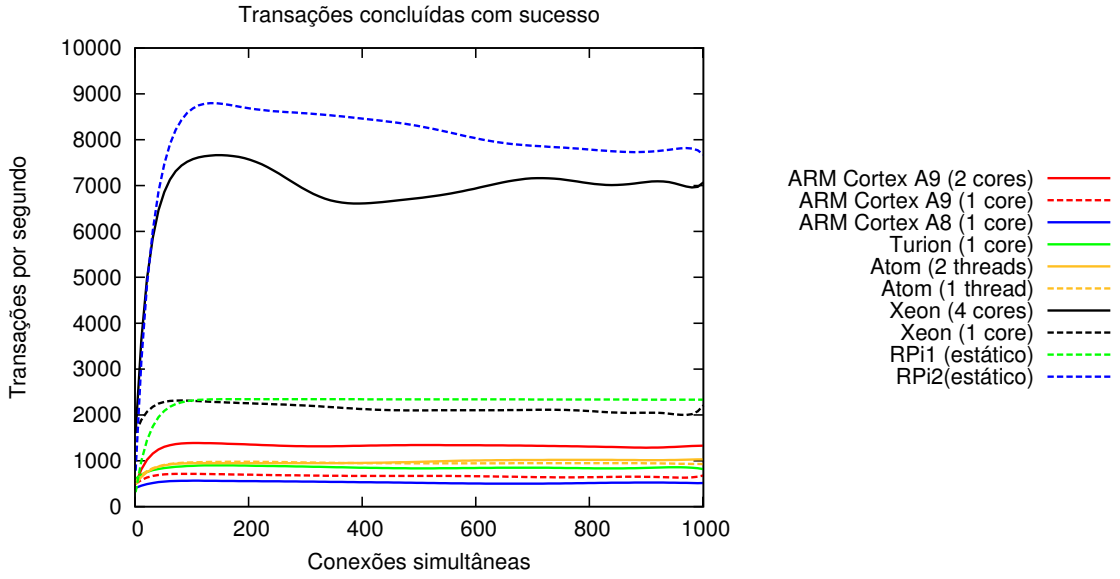

Figura 6.2: Média de transações concluídas com sucesso para o experimento 1 (traduzido de Alves Filho et al. (2017)).

mapeados pelo dispositivo na memória, para serem analisados pelo sistema operacional e então pela aplicação. Os dados trafegam por um barramento que é compartilhado por dispositivos USB. Esse processo de transferência de dados e escalonamento do uso do

barramento requer um alto uso do processador para o envio e recebimento de pacotes. Para um tráfego mais alto como nos testes, somente o processo para tratar as interrupções de rede chegou a ocupar mais de 90% da capacidade do processador no RPi1. Isso prejudicou a execução dos demais processos, entre eles os que controlavam o serviço de páginas web.

Nas máquinas RPi2 isso também acontece, mas como elas contam com 4 núcleos no processador, essa ocupação acontece em apenas um dos núcleos deixando os demais livres para executar os processos do servidor web. Mesmo acontecendo isso, a eficiência do RPi2 também foi comprometida por esse tratamento de interrupção de rede. Em sistemas Linux existe uma função no Kernel chamada *SMP affinity* (Love 2003) que pode distribuir o tratamento de interrupções entre os núcleos de um processador. Excluindo-se o núcleo que trata das interrupções da rede, os demais núcleos dos processadores das placas RPi2 obtiveram menos de 40% de sua capacidade ocupada. Se o mecanismo *SMP affinity* estivesse disponível no RPi2 uma quantidade maior de requisições poderia ser atendida. Como a memória e a largura de banda de rede não foram usadas em mais de 50%, pode-se concluir que o tratamento das interrupções de rede é o gargalo para o desempenho das máquinas Raspberry Pi e, consequentemente, do *cluster*.

O próximo gráfico, da figura 6.3, diz respeito à qualidade de serviço obtida através do tempo médio de latência. Esperava-se que os resultados dos tempos de resposta fossem piores já que as placas Raspberry possuem um hardware menos poderoso que outras máquinas e ainda têm o problema das interrupções de rede, mas não foi o que ocorreu. Devido à distribuição de carga entre os nós, os valores para as máquinas RPi2 foram semelhantes à máquina Xeon de 4 núcleos chegando a uma média de 100 ms quando há 1001 requisições simultâneas. As RPi1 ficaram com o 3º melhor desempenho, sendo melhores que a máquina Xeon com 1 núcleo ativado, com média de 250 ms.

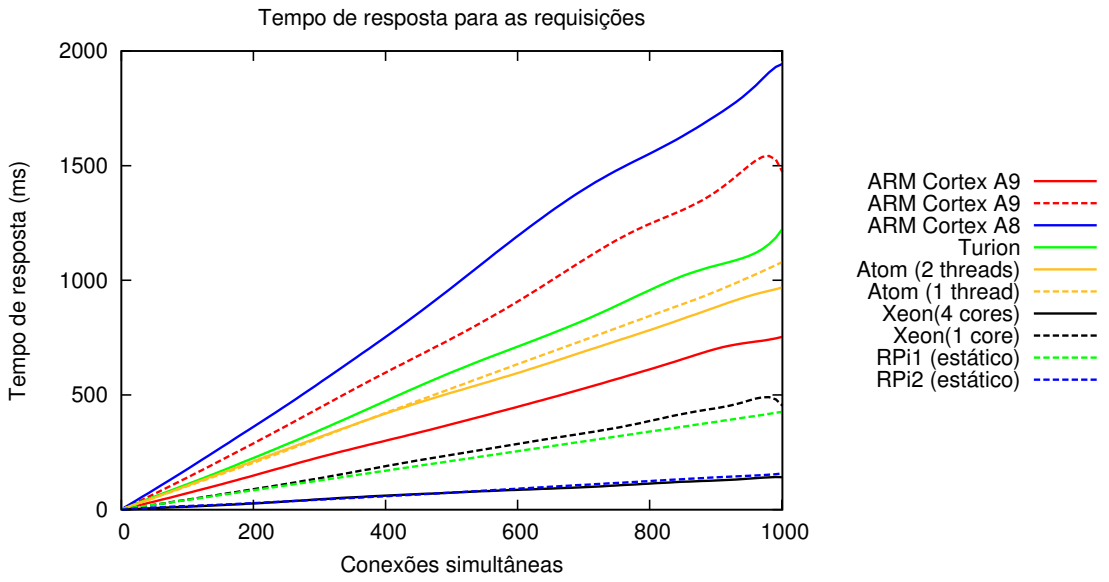

Figura 6.3: Tempo médio de resposta para cada transação no experimento 1 (traduzido de Alves Filho et al. (2017)).

O último gráfico referente a este experimento (figura 6.4) mostra a eficiência energética para cada demanda em termos de requisições por segundo / Watt. O *cluster* de máquinas RPi2 teve a melhor eficiência chegando a quase 700 requisições atendidas por segundo para cada unidade de energia consumida. Isto é mais que o dobro que a segunda maior que foi da placa PandabBoard que possui processador ARM Cortex-A9. O *cluster* RPi1 ficou com a terceira melhor eficiência mesmo com o gargalo proporcionado pelas interrupções de rede, com valores próximos aos dos outros dispositivos com arquitetura ARM. Mesmo com os melhores valores de desempenho, a eficiência dos sistemas com processadores x86 ficou abaixo de todas as máquinas ARM.

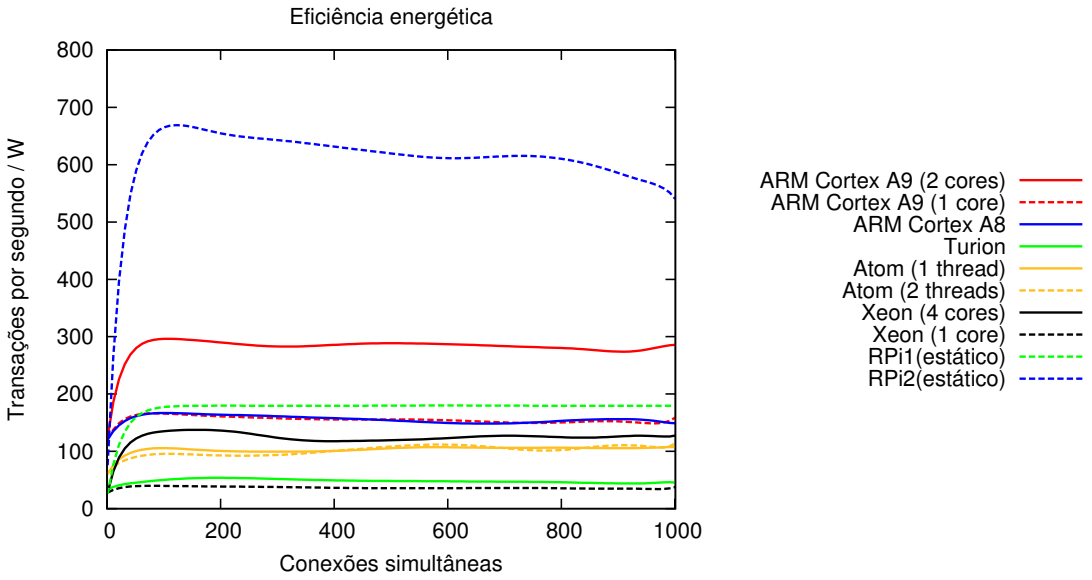

Figura 6.4: Medidas de eficiência energética para o experimento 1 (traduzido de Alves Filho et al. (2017)).

Isso demonstra mais uma vez o quão eficientes energeticamente são essas placas com processador ARM. Quanto aos *clusters*, a metodologia usada provou ser eficaz para a distribuição de carga, mesmo com os problemas enfrentados com as interrupções. Eles conseguiram estar sempre entre as melhores medidas em todos os critérios, tanto de desempenho, qualidade de serviço e eficiência energética. Na próxima seção é tratado o segundo experimento, com algumas diferenças em relação ao primeiro.

6.2 Experimento 2 - provisão dinâmica de nós

O experimento 1 mostrou a eficiência energética do NPi-Cluster, de forma especial quando implementado com o modelo RPi2. Isso ocorre porque o processador multinúcleo atenua o problema de alto processamento para tratar interrupções de rede. Mas nesse primeiro caso os nós dos *clusters* foram deixados ligados permanentemente, caracterizando um desperdício de energia para baixas demandas. Esse experimento tem o objetivo

de testar a eficácia da técnica de provisão dinâmica de nós descrita na seção 4.3 no uso proporcional do hardware. Além disso, é verificado também qual o limite de demanda em que a provisão dinâmica é mais atrativa que deixar os nós ligados de forma permanente e como ocorre a ativação ou desativação dos nós no decorrer do tempo de acordo com a demanda. Salienta-se que esse segundo experimento usa apenas as máquinas RPi2, que têm um melhor desempenho.

6.2.1 Conjunto de testes e parâmetros do algoritmo de provisão

A inclusão da provisão dinâmica não é a única alteração em relação ao experimento anterior. Uma crítica realizada ao trabalho de Aroca & Gonçalves (2012) é que ele não representa um cenário real de um servidor HTTP porque só faz acesso a uma mesma página com apenas 3 KB de tamanho. Uma página web acessada em 01 de novembro de 2017 tem em média um tamanho de 3.378 KB distribuídos em 110 arquivos diferentes (*HTTP Archive* 2017), como mostra a figura 6.5. Esses arquivos podem ser páginas HTML, folhas de estilo CSS, *scripts*, vídeos, imagens e outros tipos. Então para esse experimento foi criado um novo conjunto de testes com vários arquivos de tamanhos e tipos diferentes.

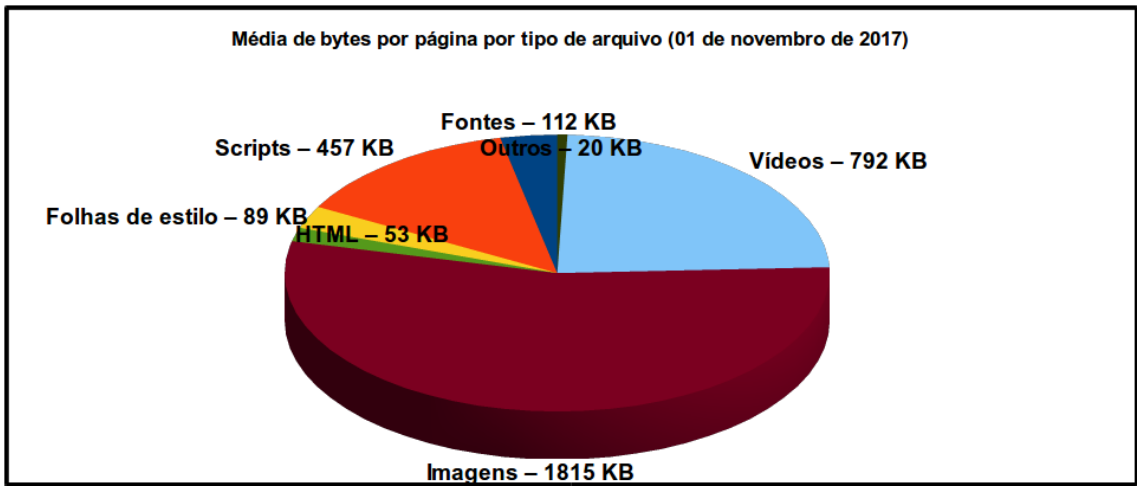

Figura 6.5: Média de tamanho para cada tipo de arquivos de uma página em 01 de novembro de 2017 (Adaptado de *HTTP Archive* (2017)).

Esse novo conjunto de testes foi criado com base no trabalho de Urdaneta et al. (2009). Nele os autores obtiveram junto à Wikimedia Foundation cerca de 10% dos *logs* de acesso a páginas da Wikipédia entre Setembro de 2007 e Janeiro de 2008. Os arquivos acessados têm tipos e tamanhos diversos. As páginas foram reconstruídas a partir dos *dumps* disponíveis com os conteúdos das páginas pela própria Wikipedia para testar um software de *benchmark* chamado Wikibench. Outros trabalhos (Toosi et al. 2017, Zhao et al. 2016) também usaram páginas da Wikipédia, mas sem usar os *logs* de acesso oficiais.

A ideia original também era utilizar os *dumps* da Wikipédia, mas a quantidade de arquivos e seus tamanhos são muito grandes em relação à capacidade do cartão de memória

dos RPis. Foi necessário adotar outra abordagem mais simplificada. Primeiro selecionou-se um subconjunto aleatório dos *logs* de acesso disponíveis para download. Esse subconjunto contém mais de 480.000 registros. Usando um *script* automático todos os arquivos desses registros foram baixados em sua versão mais atual e copiados para os cartões de cada nó do *cluster*. No total são 153.897 arquivos que ocupam aproximadamente 11,63 GB. Os detalhes sobre esses arquivos estão na tabela 6.2.

Tipo de arquivo	**% do total**	**Tamanho médio**
Áudio/Vídeo	0.02%	1.443 KB
Página HTML	54.16%	123 KB
Imagem	45.22%	23 KB
Outros	0,60%	215 KB

Tabela 6.2: Características do conjunto de arquivos usados no segundo experimento (adaptado de Alves Filho et al. (2017)).

Essa troca do conjunto de testes provocou outra mudança relacionada ao software de *benchmark*. Até o início desse experimento, o Apache Benchmark não suportava acesso a múltiplas URLs, o que é necessário para este caso. O novo software usado para realizar esse experimento foi o Siege (Fulmer 2017), que também permite especificar a demanda de clientes simultâneos e produz como resultados estatísticas similares ao Apache Benchmark, como número de transações com sucesso, throughput, latência média, entre outros. Além disso, ele pode obter as requisições que deve realizar de um arquivo, estabelecendo um período de teste sem limites do número de solicitações.

Para a realização dos testes, o arquivo das requisições foi editado para substituir o domínio `wikipedia.org` por um domínio interno criado para o experimento. Uma máquina externa ao *cluster* foi configurada com o servidor DNS bind do Linux usando o algoritmo *Round-Robin DNS*. A máquina que simula os clientes usando o software Siege consulta os registros com as requisições, faz uma consulta ao servidor RRDNS que responde a cada cliente com um IP diferente balanceando a carga.

Com relação aos parâmetros do algoritmo de provisão dinâmica nestes testes, o *cluster* usa 6 máquinas como servidores web (RPi1 ... RPi6) e 1 como gerente. Para evitar um grau de desbalanceamento elevado na distribuição das requisições, o servidor DNS está configurado com um conjunto de 12 possíveis endereços IP (192.168.0.1-12). Quando apenas um nó está atuando como servidor os 12 endereços são associados a ele. Com 2 nós ativos, cada um receberá 6 endereços e assim sucessivamente. Para os cenários com 2, 3, 4 e 6 nós ativos todos recebem o mesmo número de endereços (6, 4, 3 e 2 IPs, respectivamente). Apenas quando 5 nós estão ligados ocorre uma distribuição desigual de endereços onde algumas máquinas ficam com 2 e outras com 3 IPs. Os melhores cenários para a distribuição de IPs com o uso do RRDNS é quando o número de IPs disponíveis para distribuir possui em sua lista de divisores o maior número de elementos entre *min_nós* e o tamanho *n* do número máximo de nós disponíveis.

No início de cada experimento apenas o nó gerente e um dos nós servidores ficam ligados ($min\_nós = 1$), respondendo por todos os endereços. Com o início dos testes, o software Siege simula os clientes enviando a cada momento um número definido de requi-

sições simultâneas. Neste momento, o nó gerente já está monitorando os nós servidores para planejar se deve alterar ou não a configuração do *cluster*. Para definir o estado de utilização o algoritmo de provisão usa como taxa de utilização de cada nó o tempo médio esperado t_{atraso} de acordo com a equação 5.5 da seção 5.2.

Para realizar esse cálculo é necessário saber o tamanho da fila na porta 80, que é a do serviço HTTP e o tempo de resposta esperado para uma página com tamanho igual a média dos acessos ($t_{resposta}(\textit{página média})$). O tamanho médio da página usou os dados fornecidos pelo `HttpArchive.org` em 01 de Fevereiro de 2016, início do projeto que culminou com este trabalho. Na época o tamanho médio da página era de 2.253 KB distribuídos em 100 arquivos, dando uma média de 23.070 bytes por arquivo em média. A partir de testes semelhantes aos que foram usados para construir o gráfico da figura 5.4 os valores de $t_{resposta}(\varnothing)$ e $t_{por\ byte}$ foram computados e quando multiplicados pelo tamanho médio obtido gerou uma equação do tipo $t_{resposta}(arquivomedio) = 1,47ms$. Assim, a taxa média de atraso em um nó qualquer é dado por $t_{atraso} = 1,47\times \mid \textit{Fila de requisições} \mid$.

Não foi possível identificar na literatura um valor padrão para um tempo aceitável para aguardar que uma página web seja carregada. Fica a cargo do administrador de rede definir um valor aceitável entre seus usuários para garantir uma qualidade de serviço mínima. Para configurar o algoritmo de provisão dinâmica, foi considerado que uma página que carrega em mais de 5 segundos indica uma sobrecarga do servidor e 10 segundos indica um estado crítico. Como os dados citados anteriormente mostram que cada página é formada por cerca de 100 arquivos em média e, no pior caso, todas as requisições para estes 100 arquivos podem ser alocadas para um mesmo servidor, o valor dos limites de tempo para indicar alta utilização do *cluster* ficou com $t_{sobrecarga} = 50ms$ e $t_{crítico} = 100ms$

Para completar os parâmetros de configuração do algoritmo de provisão dinâmica, foi definido que o monitoramento iria ser feito pelo gerente a cada intervalo $tempo\_espera = 5s$. Já o número de repetições para caracterizar um estado diferente do normal foi configurado de forma que um novo nó só fosse ativado ou desativado depois de 60 segundos, o que acontecia nos testes após 6 verificações de *status*. Já o estado crítico deveria ativar um novo nó em, no máximo, 30 segundos, que dava 3 leituras seguidas. Dessa forma, define-se $r_{subutilizado} = 6$, $r_{sobrecarga} = 6$ e $r_{crítico} = 3$. A tabela 6.3 resume os parâmetros escolhidos para o algoritmo de provisão dinâmica.

Parâmetro	**Valor**
Nós	$RPi_1, RPi_2, \ldots, RPi_6$
Endereços	$192.168.0.1 - 12$
Limites	$t_{sobrecarga} = 50ms, t_{crítico} = 100ms$
Repetições	$r_{subutilizado} = 6, r_{sobrecarga} = 6, r_{crítico} = 3$
$tempo\_espera$	5 segundos
min_nos	1 nó (como servidor)

Tabela 6.3: Parâmetros de configuração do algoritmo de provisão dinâmica (traduzido de Alves Filho et al. (2017)).

Para finalizar, o número de conexões clientes concorrentes foi remodelado também para apresentar um cenário um pouco mais real. É padrão dos navegadores web definir

um número de conexões concorrentes para acelerar o carregamento das páginas, que fica entre 6 e 8. Tomando 8 como esse padrão, os testes começam com 8 conexões simultâneas, e a cada passo esse número é aumentado em 24, o que representa a chegada de 3 novos clientes. O limite máximo de conexões simultâneas obtidas com o software Siege para o cenário desse segundo experimento foi de menos de 500, que representa aproximadamente 60 clientes.

6.2.2 Resultados para o experimento 2

Cada teste para esse experimento ocorreu com a demanda entre 8 e 488 conexões simultâneas durante 10 minutos, o que permitiu observar o comportamento dinâmico do algoritmo de provisão ligando e desligando máquinas. Os resultados apresentados são a média de 10 rodadas suavizadas por curvas de Bézier. Cada rodada começa apenas com um nó servidor ativo junto com o nó gerente, e após cada rodada foi dado um intervalo de tempo necessário para retornar a esse estado inicial antes do início da rodada seguinte.

A figura 6.6 apresenta a evolução do consumo de energia no decorrer do tempo. Cada linha representa uma demanda diferente e as marcações tracejadas na horizontal mostram quantos nós estão ligados com aquele consumo de energia. Nesse gráfico pode-se notar que para as demandas mais baixas nem todos os nós são ligados, mas a partir de 200 conexões simultâneas todos os 6 nós são ativados, o que ocorre por volta dos 300 segundos. Isto significa que há um nó ativado por minuto como se desejava. Alguns comportamentos devem ser melhor analisados e, por isso foram destacados em gráficos separados.

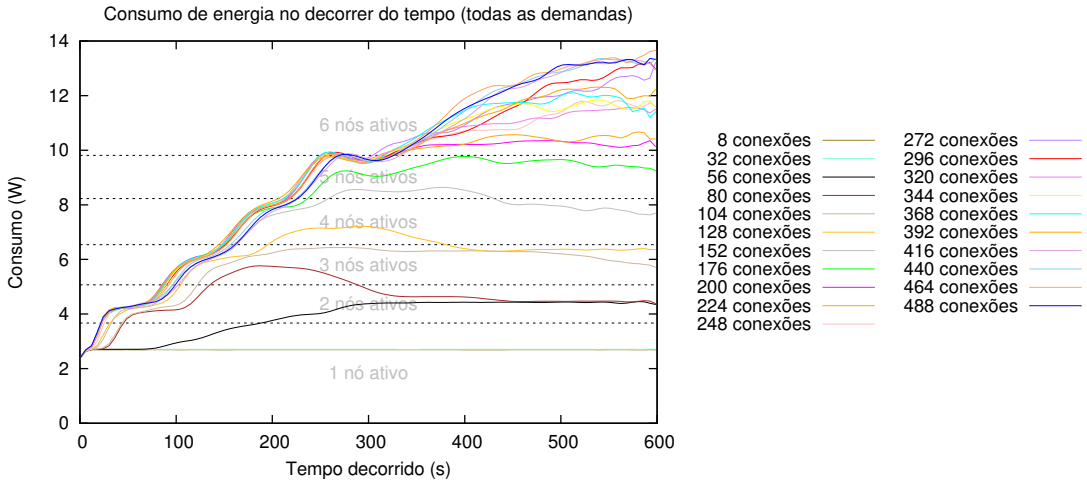

Figura 6.6: Consumo de energia no decorrer do tempo usando a provisão dinâmica proposta (traduzida de Alves Filho et al. (2017)).

A figura 6.7 traz situações nas quais, após ativar alguns nós, o monitor do gerente entende que o *cluster* achou um ponto de equilíbrio. Nesse ponto a demanda é considerada normal e não há necessidade de alterar sua configuração até o fim da execução.

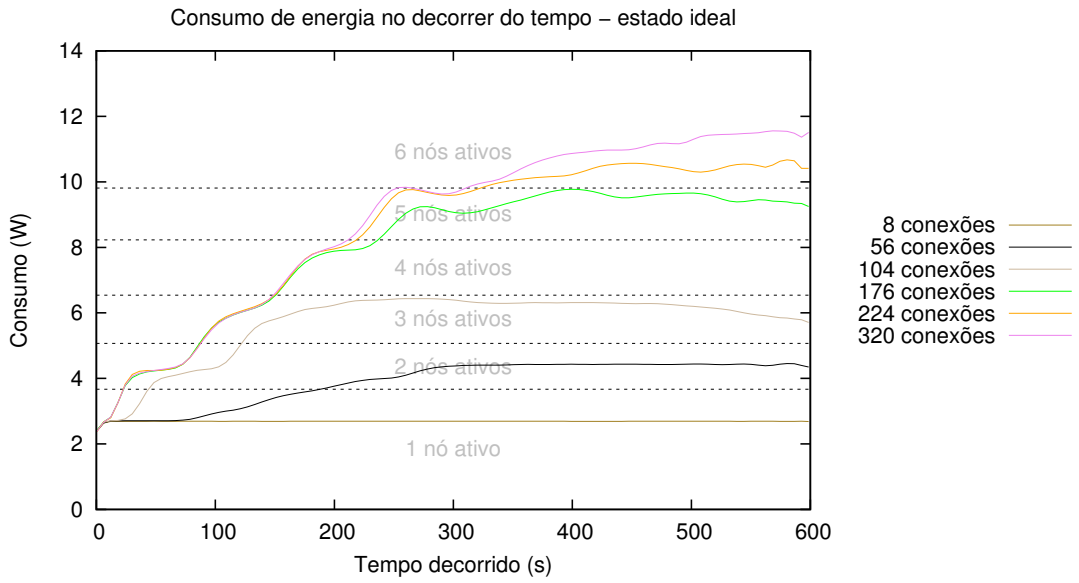

Figura 6.7: Cenários de demanda onde o *cluster* encontra o número de nós ideal (traduzida de Alves Filho et al. (2017)).

Já na figura 6.8 acontecem algumas desativações. Ou seja, após ligar um determinado nó o gerente detecta que ele não era necessário para a demanda, desligando-o e retornando para o estado anterior. Após desativar um nó, o *cluster* encontra o equilíbrio e para de ligar/desligar. Essa oscilação geralmente é provocada quando uma demanda inicia alta e, durante o tempo de ativação de uma nova máquina a fila de conexões aumenta de uma forma que mesmo com o novo nó a demanda ainda parece alta. Isso exige a ativação de outro nó, mas com o passar do tempo essa demanda vai se consolidando como menor do que a esperada e o *cluster* fica subutilizado.

Verificando a média de energia gasta neste intervalo de 10 minutos, a figura 6.9 mostra duas linhas, uma verde representando o *cluster* estático com as máquinas ligadas a todo tempo e outra azul onde há o uso da provisão dinâmica de nós. Cada marcação vermelha significa que para a demanda correspondente o *cluster* se estabilizou com a ativação de um nó a mais em relação à demanda anterior. Nota-se no gráfico que no modo estático as máquinas consomem cerca de 13 W com pequenas variações. Já no modo dinâmico a média é bem menor, pois como visto na figura 6.6 o *cluster* só atinge sua utilização máxima com aproximadamente 300 segundos. É importante considerar que nos cenários em que a demanda não chega a mais de 200 conexões simultâneas nem todos os nós são ligados. Isto significa que quanto menos o *cluster* operar no seu limite máximo, maior é vantagem de se usar o hardware proporcionalmente.

Também é interessante perceber que a linha da média de consumo começa com uma curva ascendente que depois se estabiliza. Isto acontece porque a medida que a demanda aumenta o estado de sobrecarga ou uso crítico é detectado mais cedo e os nós são ligados mais rapidamente, representando uma escalabilidade energética. Com a demanda próxima a 300 clientes simultâneos o *cluster* já liga os componentes o mais rápido que

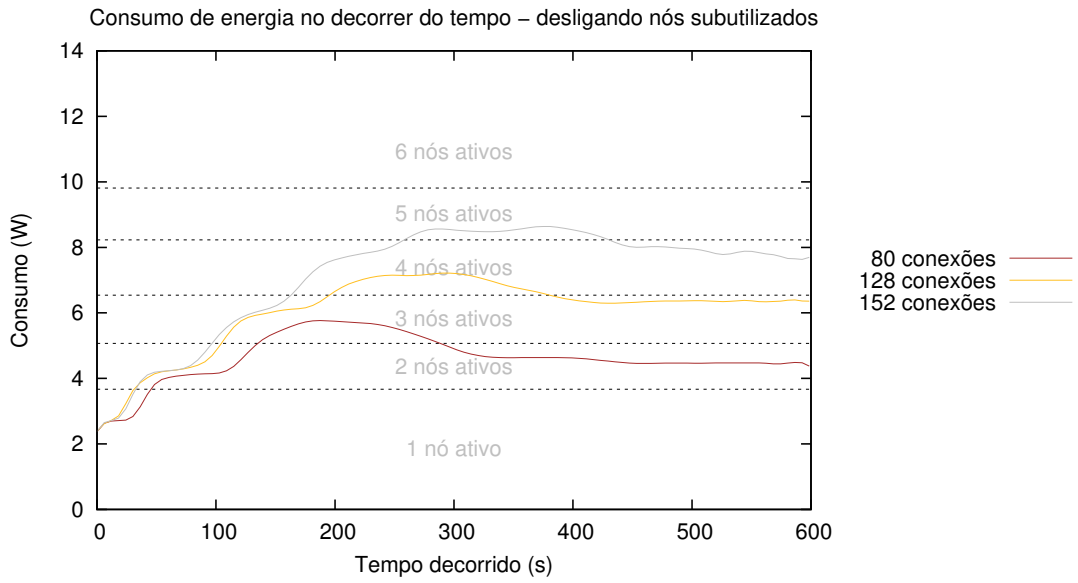

Figura 6.8: Cenários de demanda onde nós são desligados e o *cluster* desativa nós subutilizados.

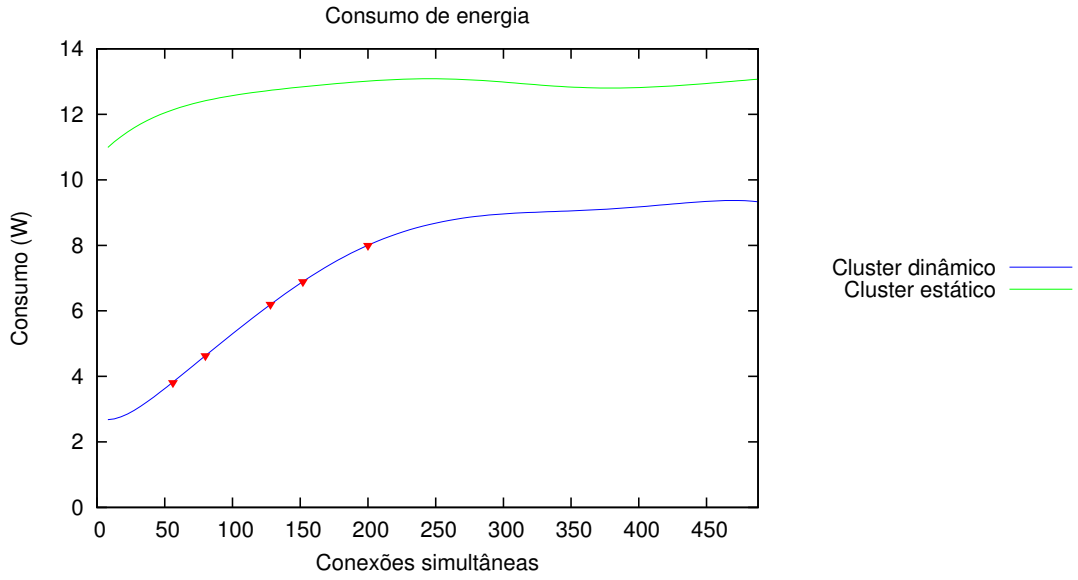

Figura 6.9: Comparação do consumo médio de energia entre os *clusters* com comportamentos estático e dinâmico (traduzido de Alves Filho et al. (2017)).

consegue e a média de consumo não aumenta mais.

Quanto ao tempo de resposta para as requisições (figura 6.10), o *cluster* estático tem sempre os menores tempos. Isso já era esperado, uma vez que todas as máquinas ficam ligadas permanentemente e oferecem o máximo de desempenho possível desde o início,

enquanto no dinâmico os nós são ligados aos poucos a partir da detecção de sobrecarga. Mesmo com o comportamento dinâmico os valores não crescem de forma descontrolada. Pelo contrário, as curvas de evolução de latência têm inclinações parecidas.

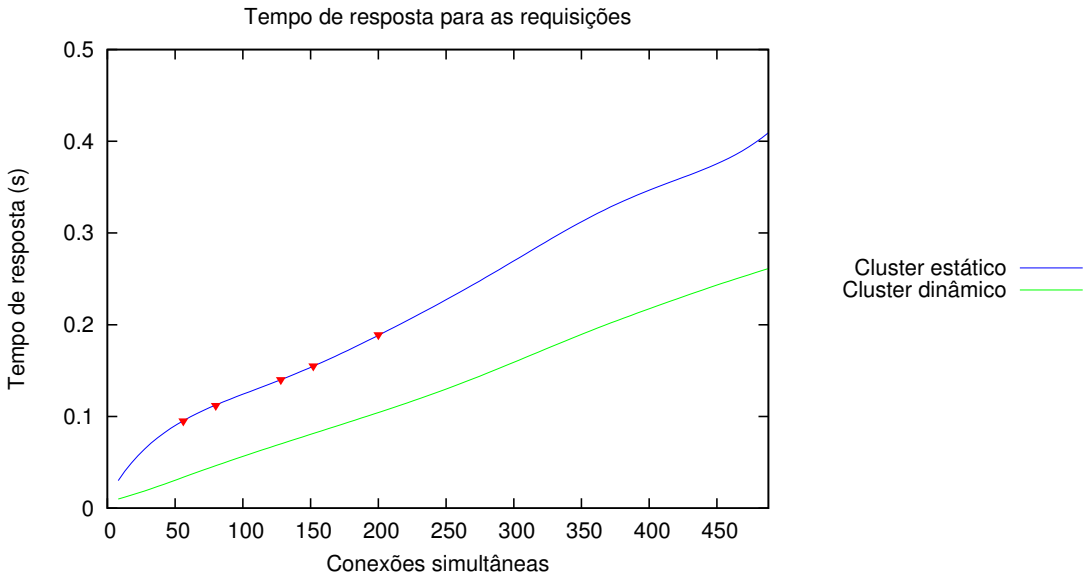

Figura 6.10: Comparação do tempo médio de resposta para as requisições entre os *clusters* com comportamentos estático e dinâmico (traduzido de Alves Filho et al. (2017)).

As médias de latências chegaram a 400 ms, ficando bem acima dos valores planejados para o algoritmo de provisão que eram de 50 ms pra sobrecarga e 100 ms para estado crítico. Mas também se deve levar em consideração que o tamanho do arquivo médio previsto era de cerca de 23 KB, enquanto os tamanhos médios dos arquivos do conjunto de teste que constam na tabela 6.2 são bem maiores que os esperados.

Com relação ao desempenho em termos de requisições atendidas por segundo a figura 6.11 mostra, também como esperado, que o *cluster* estático tem sempre um melhor desempenho. Como os nós estão sempre ligados e prontos para atender requisições a diferença de desempenho é maior para baixas demandas em que poucos nós estão ligados no *cluster* dinâmico.

Quando a demanda aumenta mais, nós são ligados e de forma mais rápida aumentando gradualmente o desempenho do *cluster* dinâmico, enquanto o estático permanece estável e sofre uma pequena queda para as demandas maiores.

Esse comportamento também é visto na figura 6.12 que traz as taxas de *throughput* em MB/s. O gráfico de transferência é praticamente uma cópia do gráfico de requisições atendidas da figura 6.11. Como os arquivos no conjunto de teste são de páginas estáticas que não necessitam de processamento no servidor web e o tempo de conexão é semelhante para todos os arquivos, o desempenho dado pela quantidade de dados transferidos é proporcionalmente equivalente às requisições atendidas.

Por fim, a figura 6.13 mostra os valores de eficiência energética para as metodologias estática e dinâmica, em que foi usada a métrica de requisições atendidas por segundo

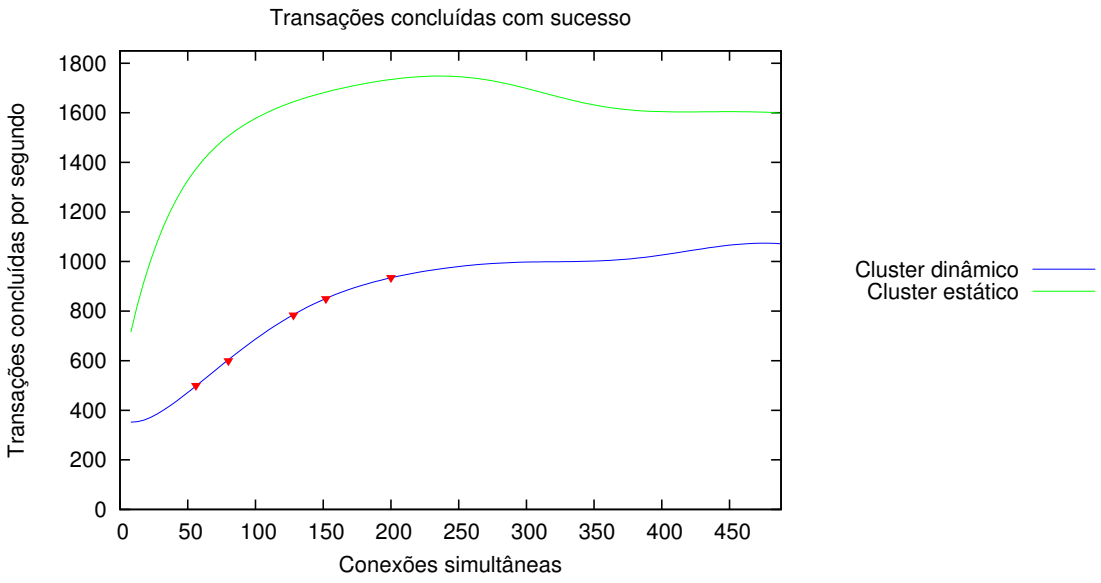

Figura 6.11: Transações concluídas por segundo para os *clusters* com comportamentos estático e dinâmico (traduzido de Alves Filho et al. (2017)).

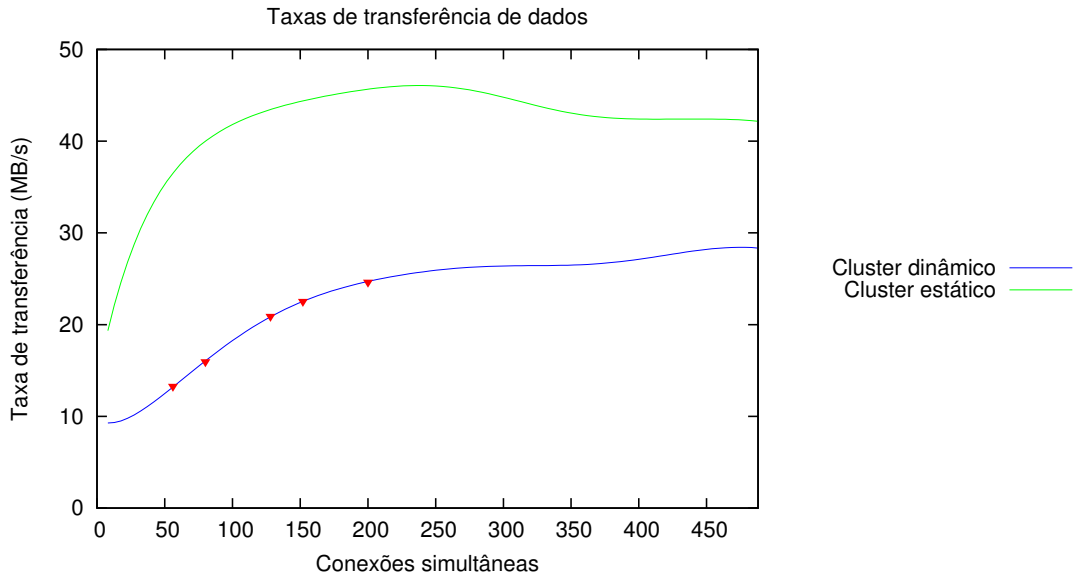

Figura 6.12: Taxas de transferência de dados pela rede para os *clusters* com comportamentos estático e dinâmico.

/ Watt. Quando a demanda é baixa, o fato de o *cluster* estático ter máquinas ligadas desnecessariamente faz com que seus valores de eficiência sejam piores que o dinâmico, mostrando um estado de subutilização. Já quando a demanda ultrapassa 152 conexões simultâneas em que o *cluster* dinâmico estabiliza com pelo menos 4 máquinas ligadas a

eficiência da abordagem estática para o período de tempo medido é mais vantajosa. Com um desempenho máximo desde o início, a maior quantidade de requisições atendidas compensa a economia de energia feita por causa do tempo necessário para o gerente detectar a sobrecarga e começar a ligar os nós.

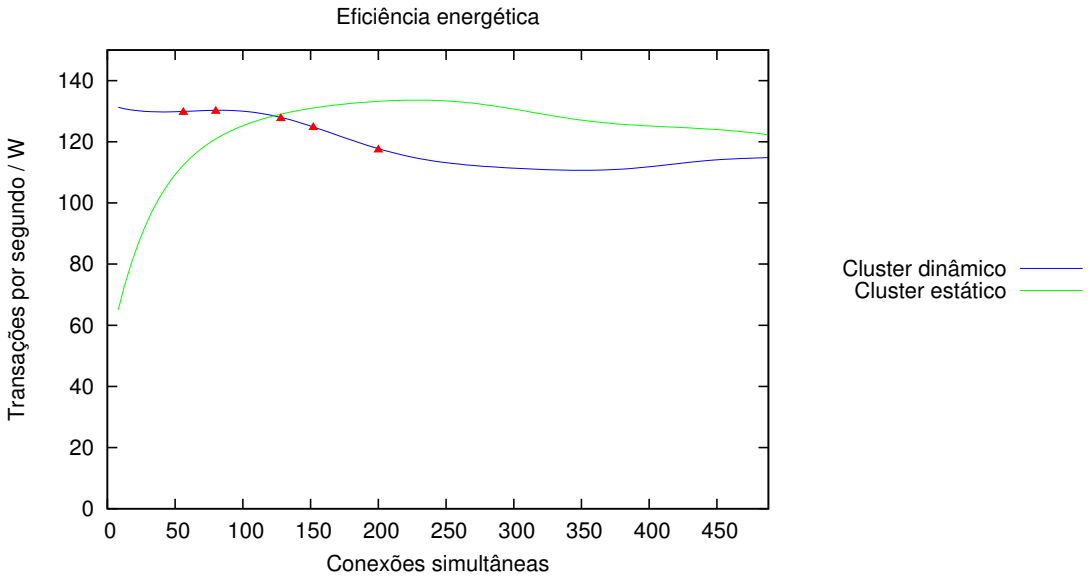

Figura 6.13: Valores de eficiência energética para os *clusters* com comportamentos estático e dinâmico (traduzido de Alves Filho et al. (2017)).

Quando a demanda obriga o *cluster* dinâmico a ligar as máquinas o mais rápido possível, o valor da eficiência energética estabiliza próximo às 120 requisições por segundo / W, não muito distante da eficiência da abordagem estática para demandas maiores. Contudo, vendo o comportamento das duas curvas, pode-se perceber que enquanto a abordagem estática tem uma variabilidade alta nos valores da eficiência entre demandas altas e baixas (o dobro), a abordagem dinâmica varia apenas em torno de 10%. Para um administrador da infraestrutura de TI esse comportamento é mais desejável, pois significa uma maneira confiável e previsível de estimar o consumo de energia com base na quantidade de requisições esperadas. Além disso, a provisão dinâmica garante que o consumo é mais baixo em períodos de baixa demanda, o que geralmente ocorre com maior frequência que períodos de sobrecarga.

6.3 Considerações sobre os experimentos e resultados

Este capítulo apresentou as configurações e os resultados dos dois experimentos montados para testar o NPi-Cluster. O primeiro experimento mostra que os *clusters* de dispositivos Raspberry Pi são soluções viáveis em termos de desempenho e eficiência energética para a aplicação de servidor web com distribuição de carga, com resultados melhores

que outras máquinas de arquiteturas de hardware ARM e x86. Este desempenho poderia ser maior se não fosse o problema do tratamento de interrupções de rede, que ocupa demasiadamente o processador e representa um gargalo para a solução.

Já o segundo experimento mostra que a provisão dinâmica de nós fornece uma solução para uma melhor eficiência energética quando a demanda é mais baixa, além de ter uma menor variação no valor da eficiência com demandas diferentes. A partir de certo limite o tempo necessário para detectar o estado de sobrecarga e ativar os nós torna a abordagem estática mais atrativa porque os nós ficam ligados de forma permanente proporcionando um desempenho máximo desde o início.

Capítulo 7

Conclusão

Este trabalho apresenta a arquitetura NPi-Cluster para a construção de *clusters* de baixo consumo de energia e proporcionalidade energética. Ele se baseia na replicação dos serviços nos nós ativos para realizar um balanceamento de carga através de servidores DNS. Quando recebe uma consulta pelo nome do servidor, o DNS distribui endereços distintos de forma que cada nó de serviço receba uma quantidade proporcional de requisições.

Para aumentar a eficiência energética, um nó gerente monitora o quanto cada nó servidor está sendo utilizado e planeja se deve aumentar ou diminuir a quantidade de nós ativos a depender da detecção de uma situação de sobrecarga ou subutilização. A fim de evitar problemas ou interferência no DNS, o *cluster* possui um conjunto de endereços IPs flutuantes que é distribuído entre os nós ativos e redistribuído a cada alteração em seu número. O algoritmo de provisão dinâmica de nós proposto tenta evitar uma oscilação em que os nós fiquem ligando e desligando constantemente, assim como busca preservar parâmetros de qualidade de serviço.

Para validar a sua viabilidade, um protótipo real foi montado com placas Raspberry Pi cuja fonte de alimentação liga-se a um circuito com relés. Esses relés são controlados pelos pinos GPIO digital de uma Raspberry que atua como nó gerente, monitorando os nós servidores e realizando a provisão dinâmica. O *cluster* montado conta com 7 nós e disponibiliza arquivos de páginas web estáticas. Para avaliar a utilização de um nó o monitor avalia o tamanho da fila de requisições do processo do servidor web e estima o tempo para atender à uma nova solicitação.

Os *clusters* montados tiveram os seus desempenhos avaliados por softwares de *benchmark* que simularam diferentes demandas de clientes. Os resultados mostram que o NPi-Cluster pode ser considerado como uma solução para data centers verdes. Eles apresentam bons resultados para a quantidade de transações concluídas por segundo e uma melhor eficiência energética quando comparados a outras máquinas com arquitetura ARM e x86. Contudo eles consumem uma quantidade reduzida de energia mesmo sem fazer o uso proporcional do hardware.

Quando habilitada a proporcionalidade, o algoritmo de provisão dinâmica consegue obter valores de eficiência energética com baixa variabilidade. Os critérios adotados para detectar sobrecarga ou subutilização também podem ser considerados satisfatórios. Com uma demanda mais baixa o número de nós ativos é mais baixo e proporciona uma melhor eficiência. Com o crescimento da demanda, os nós são ligados mais rapidamente até

encontrar uma configuração de equilíbrio.

Outros pontos positivos são a questão do espaço ocupado e da refrigeração. Como os dispositivos Raspberry Pi ocupam pouco espaço físico, os *clusters* podem ser montados como um quadro e pendurados na parede. Já quanto à refrigeração, as placas apresentam temperaturas baixas quando comparadas a outros equipamentos, e sem a necessidade de usar mecanismos de ventilação ou dissipação de calor. Isso também pode contribuir para uma maior eficiência de um data center reduzindo gastos com energia elétrica.

A implementação adotada também busca adicionar mecanismos para lidar com tolerância a falhas. Os serviços e arquivos são replicados em todos os nós servidores, aumentando assim a sua disponibilidade. O balanceamento de carga evita que um nó específico fique sobrecarregado e possa gerar falhas de conexões ou negação do serviço. O uso de conectores elétricos normalmente fechados impede que a queda do gerente comprometa o *cluster* como um todo, afetando apenas a provisão dinâmica de forma temporária. A distribuição de carga através do servidor DNS diminui o trabalho do gerente, evitando que ele seja o gargalo do sistema.

A despeito de todas as vantagens e sucesso da aplicação da arquitetura, o desempenho poderia ter sido melhor. O grande percentual de uso da CPU com o processo de tratamento de interrupções de rede faz com que esse seja o gargalo dos dispositivos e do *cluster* em si. Nos modelos RPi2 esse problema é atenuado pelos múltiplos núcleos, mas o tratamento da interrupção não é distribuído entre eles com o *SMP affinity* deixando um núcleo ocupado e os demais subutilizados, continuando a representar um gargalo.

Esperava-se que essa questão fosse resolvida com novas versões da placa Raspberry Pi, mas o lançamento do modelo 3 ainda em 2016 mostra que o problema ainda persiste. Os mesmos testes foram aplicados com um nó RPi3 e a observação do sistema mostra que um núcleo ainda continua ocupado com as interrupções e os demais mantém-se subutilizados. Há uma melhora de desempenho em relação ao RPi2 de menos de 20%, mas ocorre devido ao aumento da frequência de operação do novo modelo de processador.

É importante frisar que a distribuição de carga pode ser desbalanceada nesse modelo porque depende do número de endereços alocados em cada nó. Além disso, balancear o número de clientes por nó não garante uma distribuição de carga efetivamente igual, já que um cliente pode demandar mais carga de trabalho que outro. É preciso recordar também que o Raspberry Pi tem suas limitações de hardware como quantidade de memória RAM e velocidade de barramento. Alguns computadores permitem uma expansão, adicionando memória, trocando de processador ou outras placas, o que é mais limitado no Raspberry. É necessário encontrar aplicações ou adaptar metodologias que mais se adéquem a esse tipo de hardware.

7.1 Propostas de Trabalhos Futuros

A viabilidade da arquitetura NPi-Cluster abre uma gama de possibilidades para trabalhos futuros. Algumas estratégias podem ajudar a aperfeiçoar a implementação atual como uma interface de configuração mais amigável para o administrador da infraestrutura através de um sistema web. Também é necessário verificar a existência de soluções que contornem o problema do tratamento das interrupções de rede. Uma possibilidade é

usar outras placas de baixo consumo que tenham eficiência energética tão boa quanto os Raspberry Pi, mas sem essa limitação que se tornou o gargalo do *cluster*.

A forma abstrata como o algoritmo de provisão foi descrito também permite testar outras funções para detecção de sobrecarga, tais como redefinição dos parâmetros em tempo de execução, uso de distribuições estatísticas, agentes de software, algoritmos de predição, teoria de controle e inteligência artificial. Além disso, a aplicação de requisição de arquivos de páginas estáticas com número de conexões pré-determinadas é favorável ao modelo proposto. É necessário realizar testes que apresentam demandas com características distintas ou sazonais, além de usar outros tipos de aplicações a fim de aperfeiçoar a arquitetura proposta cada vez mais.

As técnicas usadas para melhorar a eficiência energética neste trabalho focaram no uso do hardware. Existem outras que buscam obter essa eficiência através dos softwares. É possível fazer com que os dois tipos possam trabalhar de forma complementar e/ou cooperativa fazendo surgir plataformas hardware/software eficientes e sensíveis ao contexto energético.

O baixo custo das placas Raspberry Pi, a simplicidade das configurações do algoritmo de provisão dinâmica e do mecanismo de alimentação de energia possibilita expansões do *cluster* com a inserção de novos nós. Esse crescimento também permite novas possibilidades de gerenciamento, como *clusters* de *clusters* em um modelo hierárquico ou um mecanismo de gerência distribuída. Os novos nós também podem ser outras máquinas além do Raspberry Pi, o que traria desafios adicionais como a gerência da provisão dinâmica e distribuição de carga com máquinas heterogêneas e a possibilidade de usar técnicas como os Sistemas Definidos por Software.

Por outro lado, o baixo consumo de energia pelas placas permite o uso de *clusters* como o NPi-Cluster em outros cenários além dos data centers. Paradigmas como Computação na Névoa e Internet das Coisas estendem o conceito de Computação na Nuvem trazendo serviços e aplicações para os dispositivos de borda. Isso cria a necessidade de uma infraestrutura computacional local que auxilie ou substitua os servidores remotos quando necessário. A arquitetura NPi-Cluster se encaixa nesse perfil fornecendo servidores de acordo com a demanda de maneira energeticamente eficiente.

Já saindo do contexto de servidores de rede, também é possível vislumbrar que *clusters* como os mostrados neste trabalho possam ser embarcados em robôs móveis. Ao invés de realizar certas tarefas, robôs deste tipo geralmente se comunicam com alguma estação base que possui um hardware com maior poder de processamento para ter resultados mais rápidos e consumir menos bateria. Mas uma solução como o NPi-Cluster possibilita criar dispositivos com um *cluster* embarcado de baixo consumo e proporcionalidade energética que não interfira tanto em sua autonomia. Enfim, este trabalho traz a perspectiva para a realização novas pesquisas relacionadas a sistemas distribuídos e suas aplicações.

Referências Bibliográficas

Abrahamsson, P., S. Helmer, N. Phaphoom, L. Nicolodi, N. Preda, L. Miori, M. Angriman, J. Rikkilä, X. Wang, K. Hamily & S. Bugoloni (2013), Affordable and energy-efficient cloud computing clusters: The Bolzano Raspberry Pi cloud cluster experiment, *em* 'Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on', Vol. 2, pp. 170–175.

Agarwal, S. & A. Nath (2011), Green computing - a new horizon of energy efficiency and electronic waste minimization: A global perspective, *em* 'Communication Systems and Network Technologies (CSNT), 2011 International Conference on', pp. 688–693.

Alves Filho, S. E., A. M. F. Burlamaqui, R. V. Aroca & L. M. G. Gonçalves (2017), 'Npicluster: A low power energy-proportional computing cluster architecture', *IEEE Access* **5**, 16297–16313.

Andersen, David G., Jason Franklin, Michael Kaminsky, Amar Phanishayee, Lawrence Tan & Vijay Vasudevan (2009), FAWN: A fast array of wimpy nodes, *em* 'Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles', SOSP '09, ACM, New York, NY, USA, pp. 1–14.

Apache Software Foundation (2017*a*), 'ab - apache http server benchmarking tool', On-Line. Acesso em: Maio 2017. URL: `https://httpd.apache.org/docs/2.4/programs/ab.html`.

Apache Software Foundation (2017*b*), 'Apache module mod_proxy_balancer', On-Line. Acesso em: Maio 2017. URL: `https://httpd.apache.org/docs/2.4/mod/mod_proxy_balancer.html`.

Aroca, Rafael Vidal & Luiz Marcos Garcia Gonçalves (2012), 'Towards green data centers: A comparison of x86 and arm architectures power efficiency', *J. Parallel Distrib. Comput.* **72**(12), 1770–1780.

Bailey, David H, Eric Barszcz, John T Barton, David S Browning, Robert L Carter, Leonardo Dagum, Rod A Fatoohi, Paul O Frederickson, Thomas A Lasinski, Rob S Schreiber et al. (1991), 'The nas parallel benchmarks', *The International Journal of Supercomputing Applications* **5**(3), 63–73.

Balakrishnan, Nikilesh (2012), 'Building and benchmarking a low power ARM cluster', *Master's thesis, University of Edinburgh* .

Barroso, L. A. & U. Hölzle (2007), 'The case for energy-proportional computing', *Computer* **40**(12), 33–37.

Belady, Christian, Andy Rawson, John Pfleuger & Tahir Cader (2008), Green grid data center power efficiency metrics: Pue and dcie, Relatório técnico, Technical report, Green Grid.

Bianzino, A.P., C. Chaudet, D. Rossi & J. Rougier (2012), 'A survey of green networking research', *Communications Surveys Tutorials, IEEE* **14**(1), 3–20.

Blake, G., R. G. Dreslinski & T. Mudge (2009), 'A survey of multicore processors', *IEEE Signal Processing Magazine* **26**(6), 26–37.

Brisco, Thomas P. (2013), 'DNS Support for Load Balancing', RFC 1794.
URL: *https://rfc-editor.org/rfc/rfc1794.txt*

Brown, Eric (2011), 'Sandia's mini supercomputer runs linux on 196 gumstix arm modules', On-Line. Accesso em: Maio 2017. URL: `http://linuxdevices.linuxgizmos.com/sandias-mini-supercomputer-runs-linux-on-196-gumstix-arm-modules/`.

Cardellini, V., M. Colajanni & P. S. Yu (1999), 'Dynamic load balancing on web-server systems', *IEEE Internet Computing* **3**(3), 28–39.

Chang, S.-L. & M. S. R. Rocchetti (1989), Rendering cubic curves and surfaces with integer adaptive forward differencing, *em* 'Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques', SIGGRAPH '89, ACM, New York, NY, USA, pp. 157–166.
URL: *http://doi.acm.org/10.1145/74333.74349*

Chen, Gong, Wenbo He, Jie Liu, Suman Nath, Leonidas Rigas, Lin Xiao & Feng Zhao (2008), Energy-aware server provisioning and load dispatching for connection-intensive internet services, *em* 'NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation', USENIX Association, Berkeley, CA, USA, p. 337–350.
URL: *http://research.microsoft.com/apps/pubs/default.aspx?id=76111*

Cloutier, Michael F., Chad Paradis & Vincent M. Weaver (2014), Design and analysis of a 32-bit embedded high-performance cluster optimized for energy and performance, *em* 'Proceedings of the 1st International Workshop on Hardware-Software Co-Design for High Performance Computing', Co-HPC '14, IEEE Press, Piscataway, NJ, USA, pp. 1–8.

Cloutier, Michael F., Chad Paradis & Vincent M. Weaver (2016), 'A raspberry pi cluster instrumented for fine-grained power measurement', *Electronics* **5**(4).
URL: *http://www.mdpi.com/2079-9292/5/4/61*

ClusterLabs (2017), 'Pacemaker: A scalable high availability cluster resource manager', On-Line. Acesso em: Maio 2017. URL: `http://clusterlabs.org/pacemaker.html`.

Costa, G. Da (2013), Heterogeneity: The key to achieve power-proportional computing, *em* 'Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on', pp. 656–662.

Coulouris, George, Jean Dollimore, Tim Kindberg & Gordon Blair (2013), *Sistemas Distribuídos-: Conceitos e Projeto*, Bookman Editora.

Cox, Simon J., James T. Cox, Richard P. Boardman, Steven J. Johnston, Mark Scott & Neil S. O'Brien (2014), 'Iridis-pi: a low-cost, compact demonstration cluster', *Cluster Computing* **17**(2), 349–358.

Dilley, John, Rich Friedrich, Tai Jin & Jerome Rolia (1998), 'Web server performance measurement and modeling techniques', *Perform. Eval.* **33**(1), 5–26.

Dong, Xiang, Liu Xueping, Wu Ying, Jinsong Wang & Duan Guanghong (2003), Life cycle assessment tool for electromechanical products green design, *em* 'Electronics and the Environment, 2003. IEEE International Symposium on', pp. 120–124.

Dongarra, Jack J, Piotr Luszczek & Antoine Petitet (2003), 'The LINPACK benchmark: past, present and future', *Concurrency and Computation: practice and experience* **15**(9), 803–820.

EMBC, Embedded Microprocessor Benchmark Consortium (2017), 'Coremark - processor benchmark', On-Line. Acesso em: Nov 2017. URL: `http://www.eembc.org/about/index.php`.

Feng, W. C. & K. Cameron (2007), 'The green500 list: Encouraging sustainable supercomputing', *Computer* **40**(12), 50–55.

Fitzpatrick, Jason (2011), 'An interview with Steve Furber', *Commun. ACM* **54**(5), 34–39.

Fox, Kenneth, William M Mongan & Jeffrey Popyack (2015), Raspberry HadooPI: a low-cost, hands-on laboratory in big data and analytics, *em* 'SIGCSE', p. 687.

Fulmer, Jeffrey (2017), 'Siege http load testing and benchmarking utility', On-Line. Acesso em: Maio 2017. URL: `http://www.joedog.org/siege-home/`.

Fürlinger, Karl, Christof Klausecker & Dieter Kranzlmüller (2011), Towards energy efficient parallel computing on consumer electronic devices, *em* D.Kranzlmüller & A. M.Toja, eds., 'Information and Communication on Technology for the Fight against Global Warming: First International Conference, ICT-GLOW 2011, Toulouse, France, August 30-31, 2011. Proceedings', Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 1–9.

Gao, Yongge, Jiyong Li & Yunfeng Song (2009), Performance evaluation of green supply chain management based on membership conversion algorithm, *em* 'Computing, Communication, Control, and Management, 2009. CCCM 2009. ISECS International Colloquium on', Vol. 3, pp. 237–240.

Gartner (2007), 'Gartner Estimates ICT Industry Accounts for 2 Percent of Global CO2 Emissions'.
URL: *http://www.gartner.com/newsroom/id/503867*

GeSI, Global e-Sustainability Initiative (2012), 'Gesi smarter 2020: the role of ict in driving a sustainable future', *Global e-Sustainability Initiative, Brussels, Belgium* .

Ghomi, Einollah Jafarnejad, Amir Masoud Rahmani & Nooruldeen Nasih Qader (2017), 'Load-balancing algorithms in cloud computing: A survey', *Journal of Network and Computer Applications* **88**(Supplement C), 50 – 71.
URL: *http://www.sciencedirect.com/science/article/pii/S1084804517301480*

Guan, M. & M. Gu (2010), Design and implementation of an embedded web server based on arm, *em* '2010 IEEE International Conference on Software Engineering and Service Sciences', pp. 612–615.

Gustafson, John L. (1988), 'Reevaluating Amdahl's Law', *Commun. ACM* **31**(5), 532–533.

HAProxy Technologies (2017), 'HAProxy the reliable, high performance TCP/HTTP load balancer', On-Line. Acesso em: Maio 2017. URL: `http://www.haproxy.org/`.

Heller, Brandon, Srinivasan Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee & Nick McKeown (2010), ElasticTree: Saving energy in data center networks., *em* 'NSDI', Vol. 10, pp. 249–264.

Henning, John L. (2007), 'Spec cpu suite growth: An historical perspective', *SIGARCH Comput. Archit. News* **35**(1), 65–68.
URL: *http://doi.acm.org/10.1145/1241601.1241615*

Hintemann, R & J Clausen (2016), Green cloud? the current and future development of energy consumption by data centers, networks and end-user devices, *em* 'Proceedings of ICT for sustainability'.

HTTP Archive (2017), On-Line. Acesso em: Maio 2017. URL: `http://httparchive.org`.

Humphries, Matthew (2011), 'Canonical builds a 42-core arm cluster server box for ubuntu', On-Line. Acesso em: September 2016. URL: `http://www.geek.com/chips/canonical-builds-a-42-core-arm-cluster-server-box-for-ubuntu-1390095/`.

IEA, International Energy Agency (2016), *Key world energy statistics*, International Energy Agency.

Jiang, Y. (2016), 'A survey of task allocation and load balancing in distributed systems', *IEEE Transactions on Parallel and Distributed Systems* **27**(2), 585–599.

Jin, Chao, Bronis R de Supinski, David Abramson, Heidi Poxon, Luiz DeRose, Minh Ngoc Dinh, Mark Endrei & Elizabeth R Jessup (2016), 'A survey on software methods to improve the energy efficiency of parallel computing', *The International Journal of High Performance Computing Applications* **0**, 1–31.
URL: *https://doi.org/10.1177/1094342016665471*

Kaewkasi, C. & W. Srisuruk (2014), A study of big data processing constraints on a low-power Hadoop cluster, *em* 'Computer Science and Engineering Conference (ICSEC), 2014 International', pp. 267–272.

Kansal, Aman & Feng Zhao (2008), 'Fine-grained energy profiling for power-aware application design', *SIGMETRICS Perform. Eval. Rev.* **36**(2), 26–31.
URL: *http://doi.acm.org/10.1145/1453175.1453180*

Kaup, F., P. Gottschling & D. Hausheer (2014), PowerPi: Measuring and modeling the power consumption of the raspberry pi, *em* '39th Annual IEEE Conference on Local Computer Networks', pp. 236–243.

Kaur, Tarandeep & Inderveer Chana (2015), 'Energy efficiency techniques in cloud computing: A survey and taxonomy', *ACM Comput. Surv.* **48**(2), 22:1–22:46.

Keville, K. L., R. Garg, D. J. Yates, K. Arya & G. Cooperman (2012), Towards fault-tolerant energy-efficient high performance computing in the cloud, *em* '2012 IEEE International Conference on Cluster Computing', pp. 622–626.

Krpić, Z., G. Horvat, D. Žagar & G. Martinović (2014), Towards an energy efficient SoC computing cluster, *em* 'Information and Communication Technology, Electronics and Microelectronincs (MIPRO), 37th International Convention on', IEEE, pp. 178–182.

Loghin, Dumitrel, Bogdan Marius Tudor, Hao Zhang, Beng Chin Ooi & Yong Meng Teo (2015), 'A performance study of big data on small nodes', *Proc. VLDB Endow.* **8**(7), 762–773.

Love, Robert (2003), 'Introducing the 2.6 kernel', *Linux J.* **2003**(109), 2–.
URL: *http://dl.acm.org/citation.cfm?id=770650.770652*

Malmodin, Jens, Pernilla Bergmark & Dag Lundén (2013), 'The future carbon footprint of the ict and e&m sectors', *on Information and Communication Technologies* p. 12.

McCalpin, John D (1995), 'Sustainable memory bandwidth in current high performance computers', *Silicon Graphics Inc* .

Milani, Alireza Sadeghi & Nima Jafari Navimipour (2016), 'Load balancing mechanisms and techniques in the cloud environments: Systematic literature review and future trends', *Journal of Network and Computer Applications* **71**(Supplement C), 86 – 98. **URL:** *http://www.sciencedirect.com/science/article/pii/S1084804516301217*

Mittal, Sparsh (2014), 'A survey of techniques for improving energy efficiency in embedded computing systems', *International Journal of Computer Aided Engineering and Technology* **6**(4), 440–459.

Mont Blanc (2011), 'Mont blanc project', On-Line. Acesso em: Maio 2017. URL: `http://www.montblanc-project.eu/arm-based-platforms/`.

Murugesan, San (2008), 'Harnessing green it: Principles and practices', *IT Professional* **10**(1), 24–33.

NGINX Inc. (2017), 'Nginx - high performance load balancer, web server & reverse proxy', On-Line. Acesso em: Maio 2017. URL: `https://www.nginx.com/`.

Ou, Z., B. Pang, Y. Deng, J. K. Nurminen, A. Ylä-Jääski & P. Hui (2012), Energy and cost-efficiency analysis of ARM-Based clusters, *em* 'Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on', pp. 115–123.

Paek, Min Ho (2014), An analytical framework and promotion for Green IT strategy, *em* 'Information and Communication Technology Convergence (ICTC), 2014 International Conference on', pp. 585–592.

Petitet, A., R. C. Whaley, J. Dongarra & A. Cleary (2017), 'HPL - a portable implementation of the high-performance linpack benchmark for distributed-memory computers', On-Line. Acesso em: Maio 2017. URL: `http://www.netlib.org/benchmark/hpl/`.

Pfalzgraf, A. M. & J. A. Driscoll (2014), A low-cost computer cluster for high-performance computing education, *em* 'IEEE International Conference on Electro/Information Technology', pp. 362–366.

Pinheiro, Eduardo, Ricardo Bianchini, Enrique V. Carrera & Taliver Heath (2003), Compilers and operating systems for low power, *em* L.Benini, M.Kandemir & J.Ramanujam, eds., 'Dynamic Cluster Reconfiguration for Power and Performance', Kluwer Academic Publishers, Norwell, MA, USA, pp. 75–93. **URL:** *http://dl.acm.org/citation.cfm?id=963948.963954*

Poongothai, M. (2011), Arm embedded web server based on dac system, *em* '2011 International Conference on Process Automation, Control and Computing', pp. 1–5.

Qu, Chenhao, Rodrigo N Calheiros & Rajkumar Buyya (2016), 'Auto-scaling web applications in clouds: a taxonomy and survey', *arXiv preprint arXiv:1609.09224* .

Rajovic, Nikola, Alejandro Rico, Nikola Puzovic, Chris Adeniyi-Jones & Alex Ramirez (2014), 'Tibidabo: Making the case for an arm-based hpc system', *Future Generation Computer Systems* **36**(Supplement C), 322 – 334. Special Section: Intelligent Big Data Processing Special Section: Behavior Data Security Issues in Network Information Propagation Special Section: Energy-efficiency in Large Distributed Computing Architectures Special Section: eScience Infrastructure and Applications.
URL: *http://www.sciencedirect.com/science/article/pii/S0167739X13001581*

Rajovic, Nikola, Lluis Vilanova, Carlos Villavieja, Nikola Puzovic & Alex Ramirez (2013), 'The low power architecture approach towards exascale computing', *Journal of Computational Science* **4**(6), 439 – 443. Scalable Algorithms for Large-Scale Systems Workshop (ScalA2011), Supercomputing 2011.

Rao, S.V.R.K., J. Saravanakumar, K. Sundararaman, J. Parthasarathi & S. Ramesh (2011), Intelligent green it management for enterprises through system profiling, *em* 'Green Computing and Communications (GreenCom), 2011 IEEE/ACM International Conference on', pp. 206–211.

Raspberry Pi Foundation (2017*a*), 'Raspberry pi 3 model b', On-Line. Acesso em: Maio 2017. URL: `https://www.raspberrypi.org/products/raspberry-pi-3-model-b/`.

Raspberry Pi Foundation (2017*b*), 'Raspberry Pi project home page', On-Line. Accesso em: Maio de 2017. URL: `https://www.raspberrypi.org/`.

Raspberry Pi Foundation (2017*c*), 'Raspbian - raspberry pi foundation's official supported operating system', On-Line. Acesso em: Maio 2017. URL: `https://www.raspberrypi.org/downloads/raspbian/`.

Roy, V Billy Rakesh, Sanket Dessai & SG Shiva Prasad Yadav (2009), 'Design and development of arm processor based web server', *International Journal of Recent Trends in Engineering* **1**(4), 94–98.

Ryzhyk, Leonid (2006), The arm architecture, Relatório técnico, Chicago University, Illinois, EUA.

Schall, Daniel & Volker Hudlet (2011), WattDB: An energy-proportional cluster of wimpy nodes, *em* 'Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data', SIGMOD '11, ACM, New York, NY, USA, pp. 1229–1232.

Schot, Nick (2015), Feasibility of raspberry pi 2 based micro data centers in big data applications, *em* 'Proceedings of the 23th University of Twente Student Conference on IT, Enschede, The Netherlands', Vol. 22.

Shaw, S. B. & A. K. Singh (2014), A survey on scheduling and load balancing techniques in cloud computing environment, *em* '2014 International Conference on Computer and Communication Technology (ICCCT)', pp. 87–95.

Shvachko, Konstantin, Hairong Kuang, Sanjay Radia & Robert Chansler (2010), The Hadoop Distributed File System, *em* 'Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)', MSST '10, IEEE Computer Society, Washington, DC, USA, pp. 1–10.

Srivastava, Samir K. (2007), 'Green supply-chain management: A state-of-the-art literature review', *International Journal of Management Reviews* **9**(1), 53–80.

St-Laurent, J., D. Hedin, C. Honee & M. Froling (2012), Green electronics? - an lca based study of eco-labeling of laptop computers, *em* 'Electronics Goes Green 2012+ (EGG), 2012', pp. 1–4.

Stevels, A. (2001), Product environmental care, a praxis-based system uniting iso 14001, iso 14062, ipp, eee and ecolabel elements, *em* 'Electronics and the Environment, 2001. Proceedings of the 2001 IEEE International Symposium on', pp. 52–58.

Strawson, Julie & Neil Ayres (2012), 'Adventures in retail: The other line's moving faster', On-Line. Acesso em: Maio 2017. `http://img01.thedrum.com/s3fs-public/drum_basic_article/99200/additional_media/online-retail-research-report-november-2012.pdf`.

Svanfeldt-Winter, O., S. Lafond & J. Lilius (2011), Cost and energy reduction evaluation for ARM based web servers, *em* 'Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on', pp. 480 –487.

Tanenbaum, Andrew S. & Maarten V. Steen (2007), *Sistemas Distribuídos - Princípios e Paradigmas*, 2ª edição, Editora Pearson Prentice Hall.

Tolentino, Matthew E., Joseph Turner & Kirk W. Cameron (2007), Memory-miser: A performance-constrained runtime system for power-scalable clusters, *em* 'Proceedings of the 4th International Conference on Computing Frontiers', CF '07, ACM, New York, NY, USA, pp. 237–246.
URL: *http://doi.acm.org/10.1145/1242531.1242566*

Toosi, Adel Nadjaran, Chenhao Qu, Marcos Dias de Assunção & Rajkumar Buyya (2017), 'Renewable-aware geographical load balancing of web applications for sustainable data centers', *Journal of Network and Computer Applications* **83**(Supplement C), 155 – 168.
URL: *http://www.sciencedirect.com/science/article/pii/S1084804517300590*

Top500 Supercomputer Sites (2017), On-Line. Acesso em: Nov 2017. URL: `https://www.top500.org/`.

Tso, F. P., D. R. White, S. Jouet, J. Singer & D. P. Pezaros (2013), The Glasgow Raspberry Pi cloud: A scale model for cloud computing infrastructures, *em* '2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops', pp. 108–112.

Tudor, Bogdan Marius & Yong Meng Teo (2013), 'On understanding the energy consumption of arm-based multicore servers', *SIGMETRICS Perform. Eval. Rev.* **41**(1), 267–278.
URL: *http://doi.acm.org/10.1145/2494232.2465553*

Urdaneta, Guido, Guillaume Pierre & Maarten van Steen (2009), 'Wikipedia workload analysis for decentralized hosting', *Elsevier Computer Networks* **53**(11), 1830–1845. `http://www.globule.org/publi/WWADH_comnet2009.html`.

Valentini, Giorgio Luigi, Walter Lassonde, Samee Ullah Khan, Nasro Min-Allah, Sajjad A. Madani, Juan Li, Limin Zhang, Lizhe Wang, Nasir Ghani, Joanna Kolodziej, Hongxiang Li, Albert Y. Zomaya, Cheng-Zhong Xu, Pavan Balaji, Abhinav Vishnu, Fredric Pinel, Johnatan E. Pecero, Dzmitry Kliazovich & Pascal Bouvry (2013), 'An overview of energy efficiency techniques in cluster computing systems', *Cluster Computing* **16**(1), 3–15.
URL: *https://doi.org/10.1007/s10586-011-0171-x*

Velthuis, P (2015), Small data center using raspberry pi 2 for video streaming, *em* '23th Twente Student Conference on IT'.
URL: *http://referaat.cs.utwente.nl/conference/23/paper/7515/small-data-center-using-raspberry-pi-2-for-video-streaming.pdf*

Vujović, Vladimir & Mirjana Maksimović (2015), 'Raspberry pi as a sensor web node for home automation', *Computers & Electrical Engineering* **44**(Supplement C), 153 – 171.
URL: *http://www.sciencedirect.com/science/article/pii/S0045790615000257*

Weicker, Reinhold P. (1984), 'Dhrystone: A synthetic systems programming benchmark', *Commun. ACM* **27**(10), 1013–1030.
URL: *http://doi.acm.org/10.1145/358274.358283*

Xie, Jiazhuang, Peiquan Jin, Shouhong Wan & Lihua Yue (2015), *Energy-Proportional Query Processing on Database Clusters*, Springer International Publishing, Cham, pp. 324–336.

Xu, S. S. D. & T. C. Chang (2017), 'A feasible architecture for ARM-based microserver systems considering energy efficiency', *IEEE Access* **5**, 4611–4620.

Zhao, Yiran, Shen Li, Shaohan Hu, Hongwei Wang, Shuochao Yao, Huajie Shao & Tarek Abdelzaher (2016), 'An experimental evaluation of datacenter workloads on low-power embedded micro servers', *Proc. VLDB Endow.* **9**(9), 696–707.