INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO RIO DE JANEIRO

CURSO SUPERIOR DE TECNOLOGIA EM REDES DE COMPUTADORES

UMA INVESTIGAÇÃO SOBRE A EFICIÊNCIA ENERGÉTICA EM SERVIDORES WEB VIRTUALIZADOS UTILIZANDO O HAPROXY

VIERNANRYCK LUCIANO

Arraial do Cabo, 23 de novembro de 2023.

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO RIO DE JANEIRO

CURSO SUPERIOR DE TECNOLOGIA EM REDES DE COMPUTADORES

CURSO SUPERIOR DE TECNOLOGIA EM REDES DE COMPUTADORES

VIERNANRYCK LUCIANO

Parte escrita do Projeto Interdisciplinar apresentada à Banca Examinadora do Projeto Final do Curso Superior de Tecnologia em Redes de Computadores, como requisito parcial à obtenção do título de Tecnólogo em Redes de Computadores.

Arraial do Cabo, 23 de novembro de 2023.

Ficha catalográfica elaborada por
Monica de Oliveira Tinoco
CRB7 4850

L937

Luciano, Viernanryck.

Uma investigação sobre a eficiência energética em servidores web virtualizados utilizando o HAproxy / Viernanryck Luciano – Arraial do Cabo, RJ, 2023.

42 f. : il. ; 21 cm.

Projeto Interdisciplinar (Curso Superior de Tecnologia em Redes de Computadores) – Instituto Federal de Educação, Ciência e Tecnologia do Rio de Janeiro, 2023.

Orientador: Prof. Dr. Carlos Roberto de Oliveira Junior.

1. Servidores da web. 2.Redes de computadores. I. Oliveira Junior, Carlos Alberto de Oliveira. III. Título.

IFRJ/CAC/CoBib CDU 004.774

CURSO SUPERIOR DE TECNOLOGIA EM REDES DE COMPUTADORES

VIERNANRYCK LUCIANO

Projeto Final de curso aprovado em 23 de novembro de 2023 pela banca examinadora composta pelos seguintes membros:

Carlos Roberto de Oliveira Junior

Prof. CARLOS ROBERTO DE OLIVEIRA JUNIOR – Orientador

IFRJ

gov.br Documento assinado digitalmente
ARMANDO MARTINS DE SOUZA
Data: 29/01/2024 12:04:41-0300
Verifique em https://validar.iti.gov.br

Prof. ARMANDO MARTINS DE SOUZA

IFRJ

gov.br Documento assinado digitalmente
RAFAEL DE OLIVEIRA COSTA
Data: 12/12/2023 15:44:20-0300
Verifique em https://validar.iti.gov.br

Prof. RAFAEL DE OLIVEIRA COSTA

IFRJ

Arraial do Cabo, 23 de novembro de 2023.

AGRADECIMENTOS

Agradeço sinceramente a todos que contribuíram para a realização deste trabalho, tornando possível a conclusão deste desafio acadêmico. Meus agradecimentos especiais vão para:

À minha querida mãe, Sandra, e ao meu pai, André, pela constante inspiração, apoio incondicional e amor inabalável ao longo dessa jornada. Suas palavras de encorajamento foram a luz que me guiou nos momentos mais desafiadores.

À instituição IFRJ e seu programa de bolsas de auxílio permanência, que desempenharam um papel crucial ao fornecer recursos financeiros essenciais para a continuidade dos meus estudos. Agradeço pelo compromisso em apoiar os alunos e promover a igualdade de oportunidades.

Aos professores do curso, cuja dedicação e expertise enriqueceram minha formação acadêmica. Suas orientações foram fundamentais para meu crescimento intelectual.

Ao Professor Armando Martins, que gentilmente sugeriu o tema deste trabalho. Sua orientação inicial foi fundamental para definir o foco da pesquisa.

Ao meu orientador, Carlos Roberto, pelo apoio incansável, valiosas sugestões e orientação precisa ao longo de todo o processo de elaboração deste trabalho. Sua contribuição foi essencial para a qualidade e rigor deste estudo.

Ao Professor Anderson Albuquerque, pela colaboração significativa, sugestões valiosas e pelo auxílio no uso da biblioteca em Python para a geração de gráficos. Sua expertise foi crucial para o desenvolvimento técnico deste trabalho.

À equipe do Instituto, em especial ao pessoal do CSTI, que manteve a infraestrutura funcionando eficientemente, proporcionando um ambiente propício para a realização deste trabalho.

Cada um de vocês desempenhou um papel vital, e sou profundamente grato por todas as contribuições que tornaram este trabalho possível. Muito obrigado.

RESUMO

Este artigo trata de um estudo sobre a eficiência energética em servidores Web virtualizados. Sabendo que a CPU é o componente que mais consome energia em um servidor Web, o objetivo deste trabalho foi verificar o consumo de energia dos processadores em uma série de experimentos de balanceamento de carga utilizando o HAProxy como balanceador. Nesses experimentos foram simulados clientes acessando um serviço Web hospedado em servidores virtualizados. Foram utilizados diferentes algoritmos de balanceamento de carga fornecidos pelo HAProxy e coletados dados referentes ao consumo de energia dos processadores e a experiência do usuário. Foi possível observar que um dos algoritmos proporciona um menor consumo de energia e melhor experiência do usuário. Esse menor consumo de energia pode ser significativo para o caso de datacenters com vários servidores.

Palavras-chave: Alta disponibilidade, Balanceador de carga, Failover, HAProxy, jMeter, Servidor Web.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

No cenário atual da tecnologia da informação, a eficiência e o desempenho dos servidores web desempenham um papel fundamental na entrega eficaz de conteúdo online. O balanceamento de carga emerge como fator crítico para garantir a disponibilidade contínua de serviços web e aprimorar a experiência do usuário.

No entanto, à medida que a complexidade das aplicações web e o trafego de rede continuam a crescer, surgem desafios significativos no gerenciamento eficiente desses servidores. Um desses desafios é o balanceamento de carga, que envolve a distribuição equitativa de solicitações de usuários entre vários servidores web para evitar sobrecarga e maximizar a capacidade de resposta do sistema. Embora existam várias abordagens e algoritmos de balanceamento de carga disponíveis, há uma necessidade contínua de avaliar e aprimorar essas soluções para atender às demandas crescentes.

Este artigo tem como objetivo investigar o desempenho do balanceamento de carga em um ambiente virtualizado de servidores web usando o **HAProxy** (HAProxy Community Edition, 2023) como balanceador. Nosso estudo visa analisar o impacto de diferentes algoritmos de balanceamento de carga, como Round Robin e Leastconn, nas métricas de desempenho do sistema. Além disso, pretendemos avaliar a eficácia desses algoritmos em lidar com cargas de trabalho variáveis e identificar as melhores práticas para otimizar o balanceamento de carga em ambientes similares.

Para fazer essas avaliações realizamos experimentos utilizando uma infraestrutura cuidadosamente configurada, incluindo servidores web replicados, um ambiente virtualizado e ferramentas de coleta de dados, para oferecer uma análise aprofundada do balanceamento de carga em servidores web. Acreditamos que a compreensão das implicações de diferentes algoritmos de balanceamento de carga e sua adaptação às exigências dinâmicas do ambiente poderá´ contribuir com os leitores para a melhoria contínua da eficiência energética e do desempenho dos seus servidores.

O restante deste artigo está organizado da seguinte maneira. Na Seção 2 apresentamos a arquitetura do nosso ambiente de teste, detalhando os elementos-chave que compõem nossa investigação. Os experimentos realizados são apresentados na Seção 3. Na Seção 4 apresentamos uma discussão a partir dos dados dos experimentos. Na Seção 5 apresentamos trabalhos relacionados e a conclusão e trabalhos futuros são apresentados na Seção 6.

2. DESCRIÇÃO DO CLUSTER VIRTUALIZADO

2.1. ARQUITETURA

Nossa arquitetura (apresentada na Figura 1) consiste em um cluster de servidores web replicados. O cluster apresenta uma visão única aos clientes por meio de duas máquinas front-end., que distribuem as solicitações recebidas entre os servidores que processam as solicitações. Esses servidores executam a distribuição Linux **Debian 11** x64 sem interface gráfica.

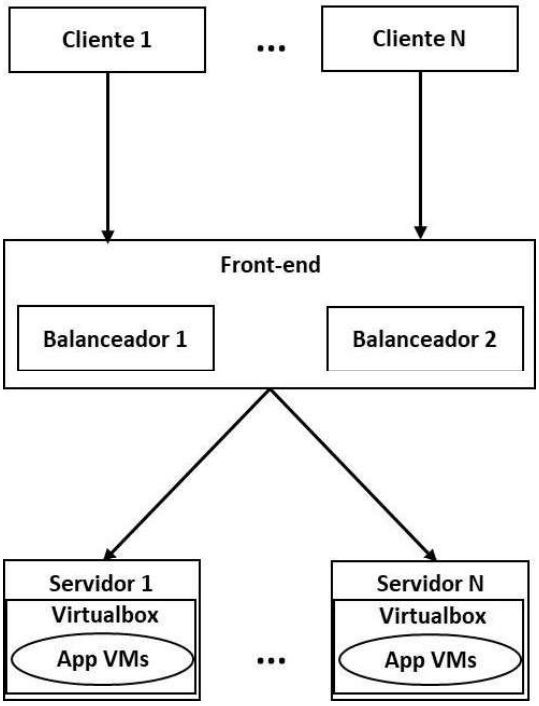

Figura 1. Arquitetura do cluster de servidores

2.1.1. BALANCEADOR DE CARGA

Segundo HAProxy Technologies (2023) ”Balanceamento de carga consiste em agregar múltiplos componentes para atingir uma capacidade total de processamento superior à capacidade individual de cada componente, sem qualquer intervenção do usuário final e de forma escalável. Isso resulta em mais operações sendo realizadas simultaneamente no tempo que um componente leva para realizar apenas uma. No entanto, uma única operação ainda será executada em um único componente por vez e não será mais rápida do que sem o balanceamento de carga. Sempre requer pelo menos tantas operações quanto componentes disponíveis e um mecanismo eficiente de balanceamento de carga para aproveitar todos os componentes e obter plenos benefícios do balanceamento de carga. Um bom exemplo disso é o número de faixas em uma rodovia, o que permite que o mesmo número de carros passe durante o mesmo período de tempo sem aumentar sua velocidade individual. ”

Um balanceador de carga é uma peça fundamental da arquitetura de redes e servidores modernos. Sua principal função é distribuir o tráfego de rede entre vários servidores ou recursos de forma eficiente, garantindo que o sistema funcione de maneira confiável, escalável e com alta disponibilidade

À medida que as empresas e organizações aumentam sua presença online e expandem suas operações digitais, a demanda por serviços de TI confiáveis e com bom desempenho se torna crucial. Para atender a essa demanda, muitas vezes é necessário dimensionar horizontalmente, adicionando mais servidores ou recursos a` infraestrutura existente. No entanto, essa expansão traz consigo o desafio de distribuir o tráfego de forma uniforme para evitar sobrecarregar qualquer servidor específico, minimizando as- sim os tempos de inatividade e garantindo uma experiência de usuário consistente.

Entre as diversas opções de balanceadores de carga disponíveis (**Nginx**, **Microsoft Azure Load Balancer** e **Kemp LoadMaster**), o HAProxy se destaca como uma solução robusta e amplamente adotada. Ele é um software de código aberto que oferece um conjunto poderoso de recursos de balanceamento de carga, o que o torna uma escolha popular para equilibrar o tráfego em uma variedade de cenários.

Algumas das características distintivas do HAProxy incluem:

- **Alta Disponibilidade:** O HAProxy suporta configurações de alta disponibilidade, garantindo que, mesmo em caso de falha de um nó, o tráfego seja redirecionado de maneira transparente para servidores funcionais.
- **Flexibilidade de Configuração:** Sua configuração é altamente personalizável, Permitindo que os administradores definam políticas de balanceamento precisas e ajustem-nas conforme necessário.
- **Monitoramento e Estatísticas avançadas:** O HAProxy oferece ferramentas abrangentes de monitoramento e geração de relatórios, permitindo que os operadores acompanhem o desempenho do sistema em tempo real.
- **Suporte a Protocolos Diversos:** Além de equilibrar o tráfego HTTP/HTTPS, o HAProxy oferece suporte para uma variedade de protocolos, tornando-o versátil o suficiente para atender a uma ampla gama de aplicações.

Além dessas características, o HAProxy oferece diferentes algoritmos de balanceamento. Durante os experimentos, alternamos o balanceador de carga entre os algoritmos de balanceamento Round Robin e Leastconn.

2.1.2. ROUNDROBIN E LEASTCONN

Conforme informado pelo HAProxy Technologies (2023), a principal distinção entre esses dois algoritmos reside na abordagem de distribuição da carga. O RoundRobin (Round- Robin) distribui as solicitações de forma igualitária, independentemente da carga de trabalho de cada servidor, enquanto o Leastconn (Least connection) prioriza servidores com menos conexões ativas no momento. Portanto, o Leastconn seria o mais adequado quando a carga de trabalho não é uniforme, e alguns servidores podem estar mais sobrecarrega- dos do que outros. Isso ajuda a evitar a sobrecarga de servidores individuais, assegurando uma distribuição mais equilibrada da carga.

2.1.3. FAILOVER

A busca incessante por disponibilidade contínua em ambientes de servidor nos levou a` implementação de estratégias de Failover utilizando o Keepalived (Alexandre Cassen, 2023) , uma solução valiosa para manter a integridade dos serviços. No cerne

dessa abordagem, temos o balanceador de carga principal, identificado pelo endereço IP 10.0.0.1, e seu contraparte de backup, configurado com o endereço IP 10.0.0.2. O balanceador de backup, estratégica- mente mantido em standby, representa uma peça-chave em nosso plano de contingência para lidar com falhas inesperadas em nossa infraestrutura.

Embora nossos servidores individuais estejam bem preparados para oferecer alto desempenho e confiabilidade, é importante reconhecer que, mesmo com vários servidores em operação, o balanceador de carga principal pode, de certa forma, se tornar um ponto único de falha. Se o balanceador de carga principal parar de funcionar, todos os outros servidores sob sua gestão podem se tornar inacessíveis, resultando em uma interrupção significativa dos serviços.

É aqui que entra o conceito de ”Failover” ativo em nosso balanceador de carga de backup. O Failover é projetado para detectar falhas em nosso balanceador de carga principal. Quando ocorre uma falha o Failover assume um papel crucial. Ele redireciona automaticamente o tráfego para o servidor de backup, ativando o IP virtual (10.0.0.1), garantindo assim a continuidade dos serviços web.

Essa configuração inteligente e altamente resiliente é fundamental para nossa missão de fornecer disponibilidade contínua. Ela garante que mesmo no caso improvável de uma falha no balanceador de carga principal, a infraestrutura de servidor seja capaz de se recuperar rapidamente, minimizando qualquer impacto nas operações e na experiência do usuário. O Failover em nosso balanceador de carga de backup é um exemplo de como estratégias cuidadosamente planejadas e implementadas podem garantir a confiabilidade e a disponibilidade dos serviços, independentemente das adversidades que possam surgir.

2.2. TESTBED

O testbed utilizado para implementar a arquitetura proposta é apresentado na Figura 2. As solicitações web dos clientes são redirecionadas para as máquinas virtuais (VMs) corres- pendentes que executam os servidores web. Cada VM possui uma cópia de um script PHP simples vinculado que faz uso da CPU para caracterizar uma aplicação web. Para gerar carga para a aplicação web nós utilizamos uma máquina com jMeter (Apache Software Foundation, 2023). Todas as máquinas do cluster possuem a mesma configuração (Intel Pentium Gold 5400 (Dual core 3,7Ghz. Quad core núcleos lógicos), Memória DDR3 8GB, Disco rígido 500GB, Rede em 1Gbs full duplex)). As máquinas que rodam os balanceadores e os servidores executam a distribuição Linux Debian 11 x64 sem interface gráfica, enquanto o gerador de carga executa o Windows 10.

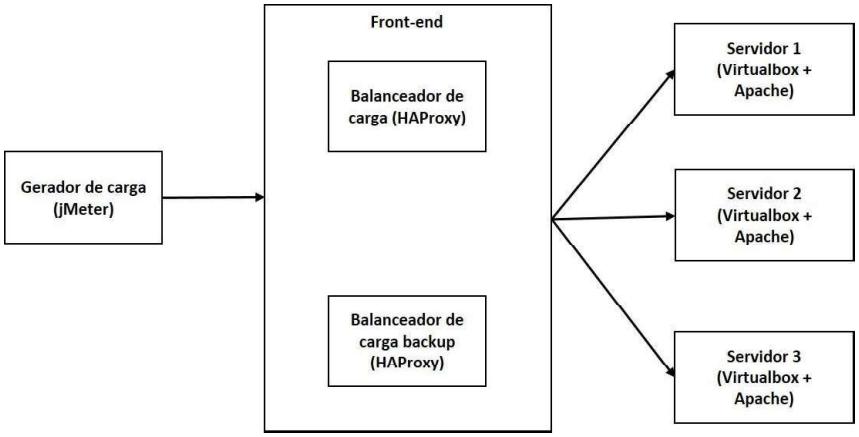

Figura 2. Configuração do cluster

2.3. COLETA DE DADOS

Todos os servidores tem o HWiNFO (HWiNFO Diagnostic Software, 2023) rodando e configurado com uma hotkey para iniciar a coleta de dados da máquina física quando a tecla HOME for pressionada. Ainda para a coleta de dados, foi escrito por um dos autores um serviço em Java que fica ouvindo uma porta (34569) e quando recebe um comando para iniciar pressiona a tela HOME e o HWiNFO começa a captar e salvar os dados (Consultar Apêndice A) . Dentro das VMs rodando o servidor web também existe um agente do Perfmon (Tableau, 2023) que monitora o uso de memória RAM da máquina virtual. O Perfmon deixa uma porta aberta (4444) para receber o pedido e enviar o percentual de memória utilizado.

Então quando o experimento é iniciado, com a geração de carga no JMeter, são enviados os comandos para todos os servidores iniciarem as coletas de dados localmente. Depois de coletar todos os dados eles são processados com um código em Python que lê os diferentes arquivos .CSV (com a biblioteca Pandas) e faz as análises iniciais também gerando gráficos (com a biblioteca Matplotlib). Com esses dados em mãos é feita uma melhor análise e comparação dos dados com os gráficos e arquivos de log existentes (Consultar Apêndice B).

3. EXPERIMENTOS

Para analisar o desempenho de nossa infraestrutura de servidores, realizamos uma série de experimentos em cenários variados, alternando entre diferentes estratégias de balanceamento de carga e modos de teste. Estes experimentos, apresentados nas subseções a seguir, visam oferecer uma visão abrangente de como nosso sistema opera em diferentes condições, destacando sua capacidade de manter um serviço confiável e de alta disponibilidade.

Em nossos experimentos, empregamos a poderosa ferramenta JMeter (Apache Software Foundation, 2023) para simular a interação de diversos usuários com nossos servidores, resultando em uma significativa carga de processamento para nossa infraestrutura. Através dessas simulações, buscamos avaliar o desempenho e a capacidade de resposta de nossos sistemas sob condições de alto tráfego, replicando um cenário realista de utilização.

Nos testes realizados, simulamos a atividade de um grupo composto por 1000 usuários que acessam uma página web em PHP. Essa página é projetada para gerar carga de CPU por meio de um simples loop, conforme mostrado abaixo:

```
for($i = 0; $i < 1000000; $i++) {
  $a += $i;
  $b+=rand();
}
```

Cada usuário faz 10 acessos, resultando em um total de 10.000 acessos. Configuramos o JMeter para distribuir esses acessos ao longo de 60 segundos. No contexto desses testes, consideramos que ocorre um erro quando: (1) o tempo de conexão com o servidor excede 21 segundos (21000ms); ou (2) o servidor demorar mais de 50 segundos (50000ms) para criar a página solicitada.

Para calcular o quanto foi gasto com energia elétrica levamos em consideração a tarifa residencial normal (B1) vigentes para o Rio de Janeiro pela concessionária Enel (Enel Spa, 2023) que no momento do teste era de R$ 0,88834 kWh na bandeira verde (os valores de impostos não foram levados em consideração para os cálculos). O monitoramento do consumo de energia das CPUs em cada servidor é realizado em intervalos de um segundo, e os valores são registrados regularmente. Para calcular os custos em reais associados a cada servidor, o código utiliza a seguinte fórmula:

$$Gasto = \frac{\text{Potencia\_Watts}}{1000} * \text{Tempo\_Em\_Horas} * \text{taxa\_energia\_em\_KWh}$$

A função apresentada permite calcular o custo de energia elétrica ao usar um dispositivo elétrico com base em três principais fatores: a potência do dispositivo em Watts, o tempo de uso em horas e a taxa de energia elétrica em quilowatt-hora (kWh). Primeiramente, a potência é convertida de Watts para quilowatts (kW) dividindo por 1000. Em seguida, multiplica-se a potência convertida pelo tempo de uso para calcular a quantidade total de energia consumida em quilowatt-hora. Por fim, multiplica-se essa quantidade pela taxa de energia elétrica para determinar o gasto financeiro estimado.

Observe que os dados de uso do balanceador de carga não foram considerados, pois ele apenas "encaminha" as solicitações e seu consumo de energia foi muito baixo (não ultrapassando 6W). Portanto, concentraremos nossa análise no uso dos servidores, que realizam o processamento e consomem uma quantidade significativa de energia.

3.1. CENÁRIO 1: SERVIDOR ÚNICO

Neste primeiro teste, não houve uso de balanceador; em vez disso, um único servidor recebeu toda a carga. O teste teve uma duração de 307 segundos, o equivalente a cerca de 5 minutos, e obteve uma taxa de sucesso de 72% nas respostas, com uma média de 12 segundos (12544 ms) para cada resposta como visto na figura 5. Durante esse período, A CPU atingiu uma temperatura máxima de 43°C (figura 4). trabalhando em uma potência de média de 16.17 Watts, operando em carga máxima(figura 6) , O que se traduz em um custo aproximado de R$ 0.001226 centavos durante esse teste.

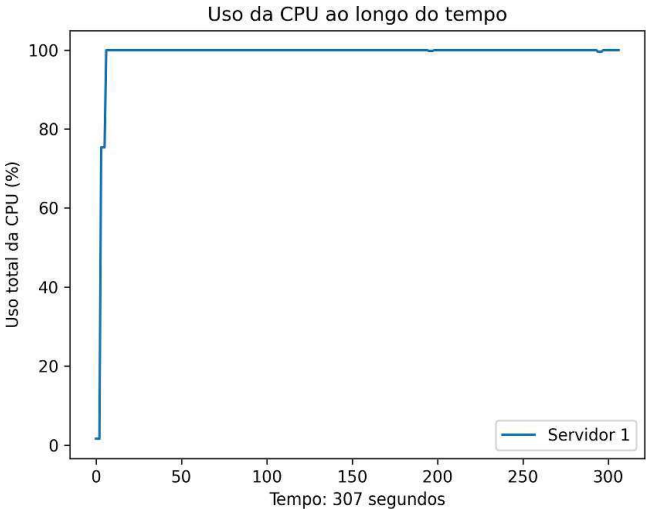

Figura 3. Uso de CPU, Cenário 1

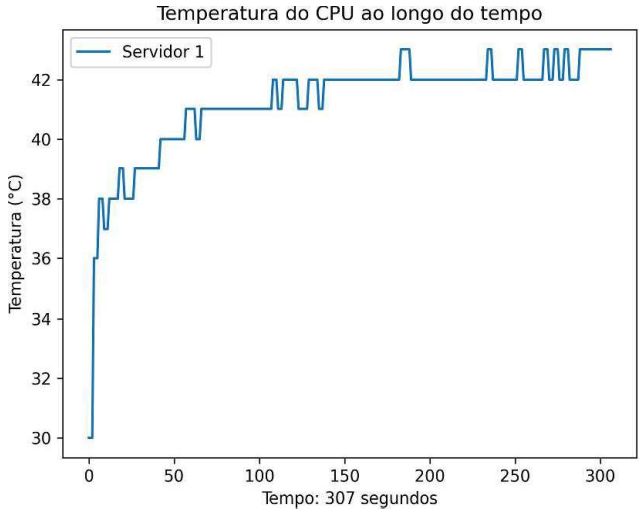

Figura 4. Temperatura da CPU, Cenário 1

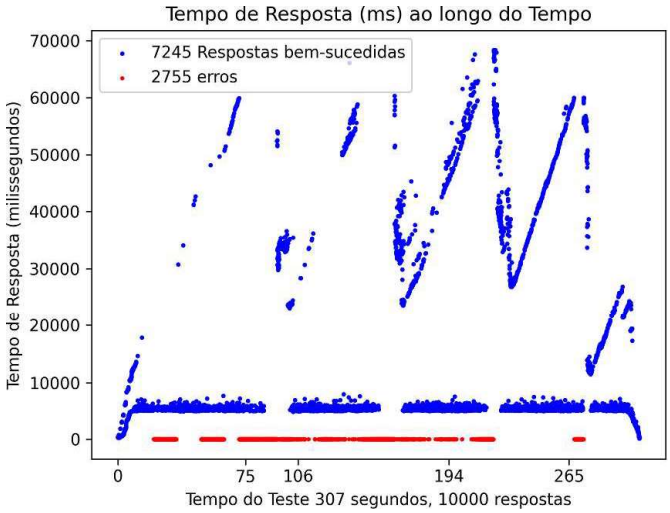

Figura 5. Tempos de resposta, Cenário 1

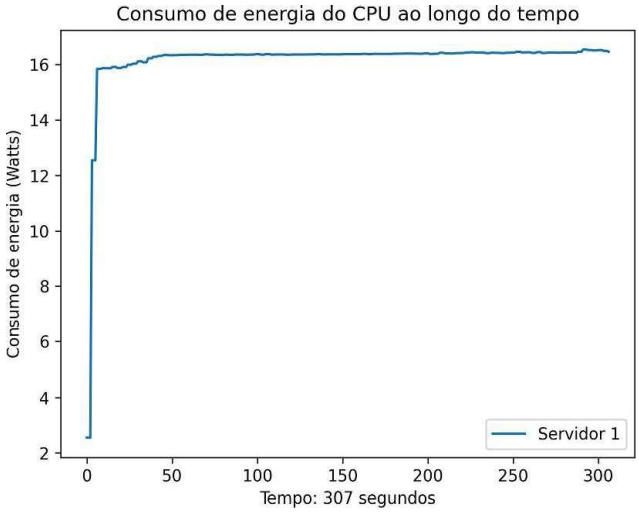

Figura 6. Consumo de energia, Cenário 1

3.2. CENÁRIO 2: ROUND-ROBIN

Nesta série de testes, nosso balanceador de carga entrou em operação, distribuindo o trabalho entre nossos três servidores por meio do algoritmo Round-Robin (consulte a seção
2.1.2 para mais detalhes). A duração total do teste foi de 126 segundos, com um índice de sucesso de 95% ou seja 23% a mais que o primeiro teste, obtivemos apenas 494 erros contra 2755 no teste anterior. A média do tempo de resposta foi de aproximadamente 6 segundos (5931 ms) metade do tempo anterior, conforme ilustrado na Figura 9. Durante

esse período, o consumo médio de energia pelos processadores foi de 16 watts por segundo apresentado na Figura 10, operando em carga máxima, e a temperatura máxima atingida foi de 44°C no Servidor 3 como demonstrado na Figura 8..A potência somada de todos os CPUs foi de 48.70 Watts, Esse consumo se traduz em um custo aproximado de R$ 0.001515 centavos durante o teste.

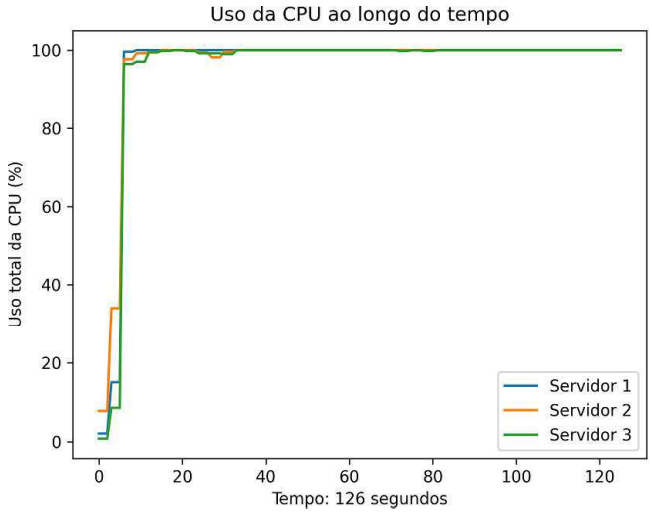

Figura 7. Uso de CPU, Cenário 2

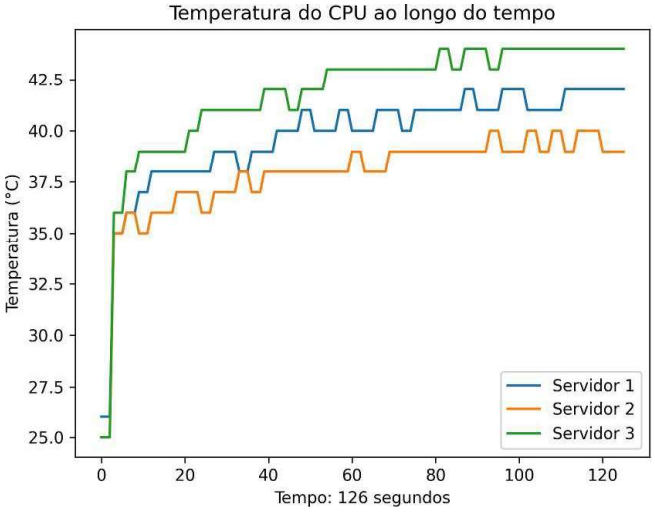

Figura 8. Temperatura da CPU, Cenário 2

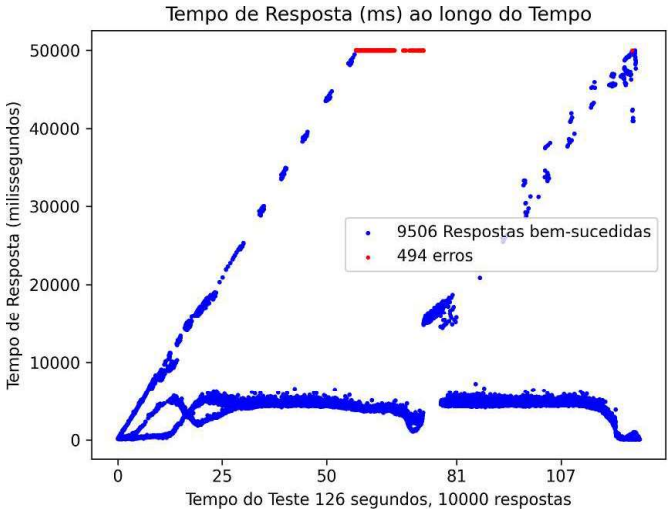

Figura 9. Tempos de resposta, Cenário 2

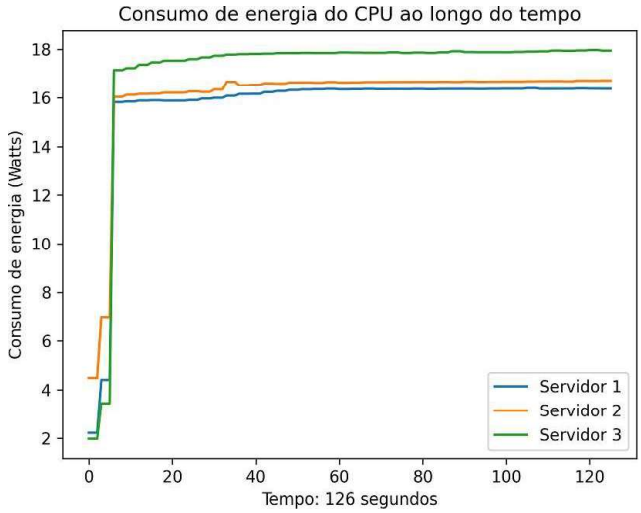

Figura 10. Consumo de energia, Cenário 2

3.3. CENÁRIO 3: FAILOVER

Neste teste, aplicamos os mesmos parâmetros utilizados no teste anterior (ver Cenário 2, na Seção 3.2). No entanto, desta vez, deliberadamente desativamos nosso balanceador de carga principal (10.0.0.1) para simular uma interrupção. Como resultado, o nosso servidor de backup (10.0.0.2) entrou em ação, ativando o Failover (consulte a Seção 2.1.3). Assim, ele assumiu o endereço IP 10.0.0.1 e começou a responder às requisições, distribuindo a carga.

Nesse cenário, nosso teste teve uma duração de 166 segundos, um pouco mais de três minutos, apenas 40 segundos a mais em relação ao teste anterior (ver Cenário 2)

Alcançamos uma taxa de sucesso de 87%, com uma média de tempo de resposta de aproximadamente 6 segundos (5950 ms), conforme ilustrado na Figura 13. Entre os 1278 erros registrados, 923 deles foram identificados como 'Non HTTP response code: java.net.SocketTimeoutException/Non HTTP response message: Read timed out', correspondendo às requisições perdidas devido ao desligamento do balanceador principal. Os outros 295 erros indicaram que as requisições foram atendidas, mas com um tempo de resposta significativamente alto.

A temperatura máxima registrada nos CPUs atingiu 44°C (conforme a Figura 12), enquanto a potência média de consumo dos CPUs foi de 11 watts (ver Figura 14), com uma potência pico de 18 watts no servidor 3. A potência somada de todos os servidores foi de 33.01 Watts, resultando em um custo de energia de aproximadamente R$ 0.001352 centavos durante o período de execução do teste.

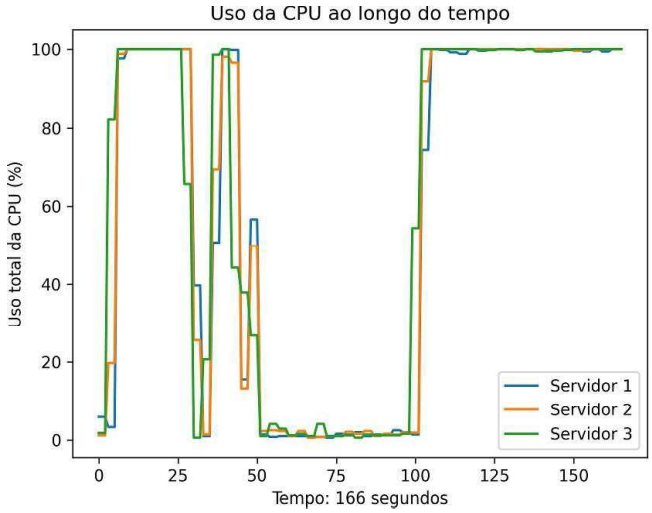

Figura 11. Uso de CPU, Cenário 3

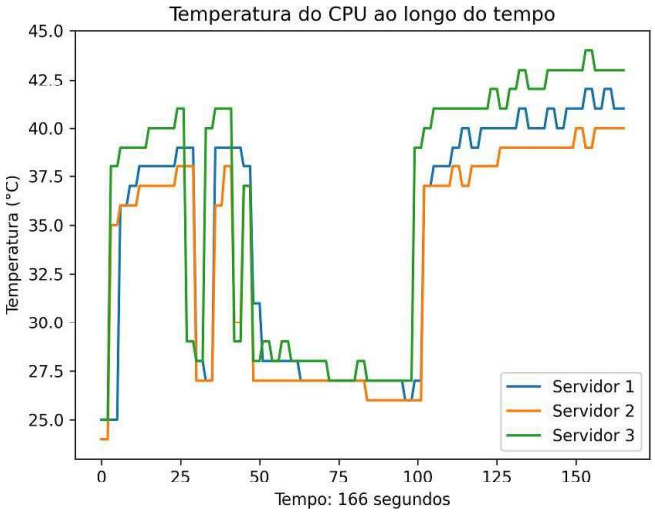

Figura 12. Temperatura da CPU, Cenário 3

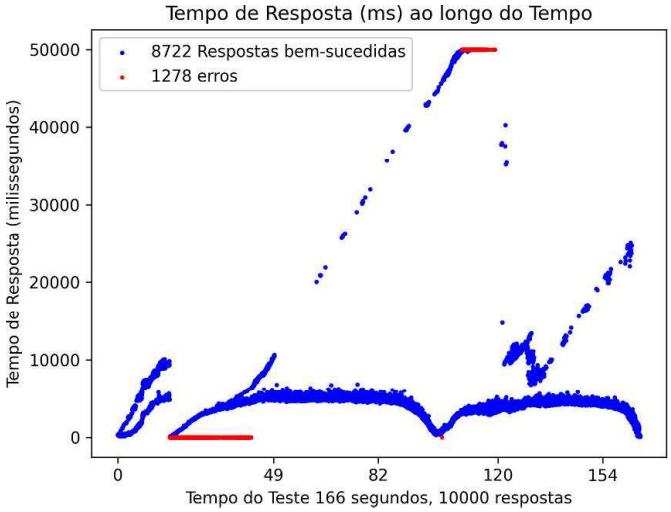

Figura 13. Tempos de resposta, Cenário 3

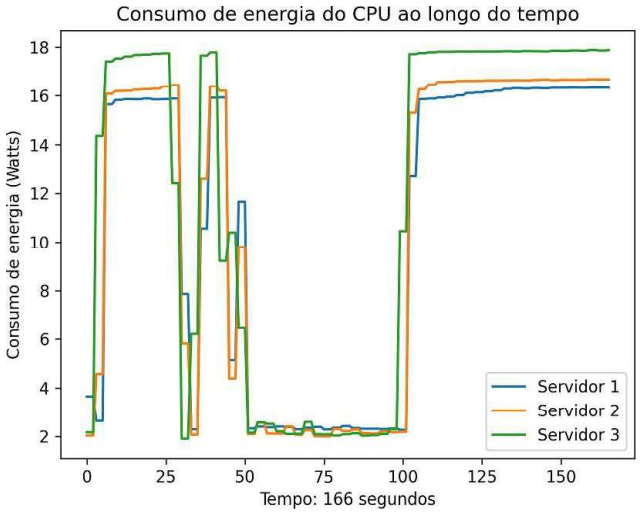

Figura 14. Consumo de energia, Cenário 3

3.4. CENÁRIO 4: LEASTCONN

A configuração do HAProxy foi modificada para utilizar o algoritmo Leastconn (consulte a seção 2.1.2). Em seguida, conduzimos uma bateria de testes sob essa nova configuração. Este teste teve uma duração de 130 segundos, o que equivale a pouco mais de dois minutos, e apresentou uma taxa de sucesso de 94,5%, registrando somente 544 erros ao longo desse período. Além disso, a média do tempo de resposta foi de aproximadamente cinco segundos (5458ms), conforme demonstrado na Figura 17.

Durante essa avaliação, o Servidor 3 alcançou a temperatura mais alta, atingindo 45°C Figura 16, enquanto os processadores mantiveram uma potência média de 16 watts por segundo Figura 18. A potência somada foi de 49.41 Watts, Como resultado, o consumo de energia foi calculado em R$ 0.001593 centavos.

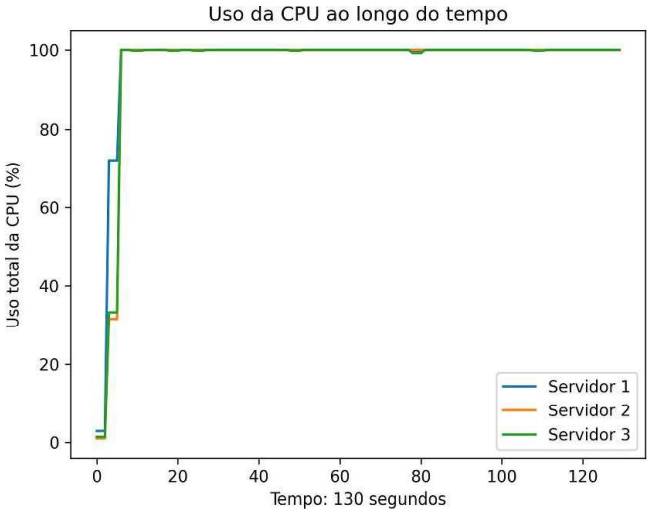

Figura 15. Uso de CPU, Cenário 4

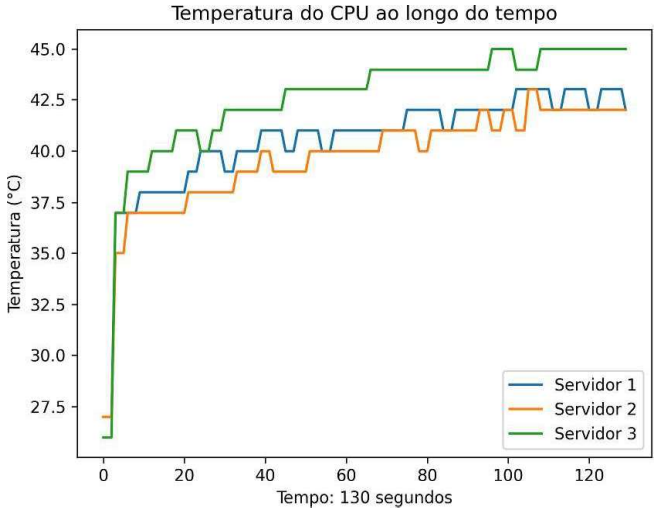

Figura 16. Temperatura da CPU, Cenário 4

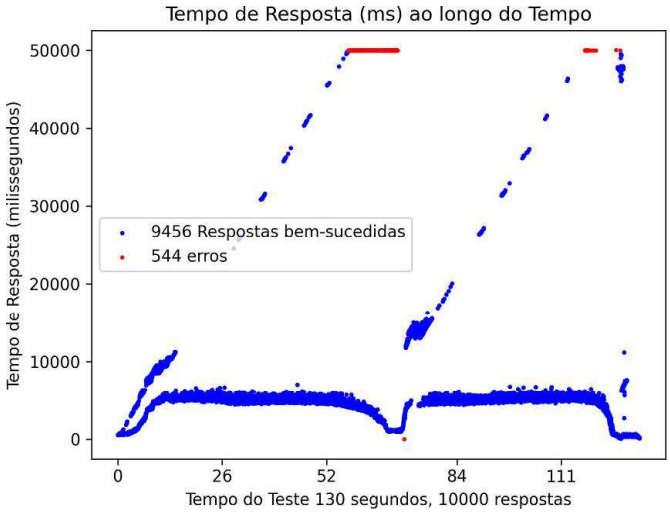

Figura 17. Tempos de resposta, Cenário 4

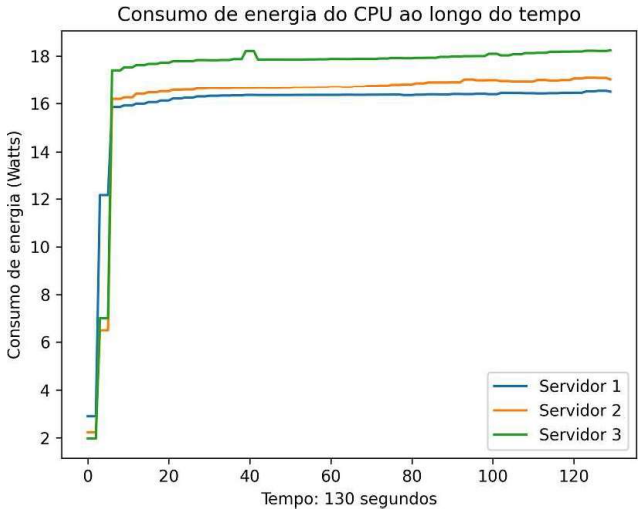

Figura 18. Consumo de energia, Cenário 4

4. DISCUSSÃO

Tabela 1. Comparativo entre os testes

Cenário	Tempo	Sucesso	Erros	Média Latência	Potência CPUs	Gasto
1	307s	72,45%	2755	12544ms	16.17 W	R$ 0.001226
2	126s	95,06%	494	5931ms	48.70 W	R$0.001515
3	166s	87,22%	1278	5950ms	33.01 W	R$ 0.001352
4	130s	94,56%	544	5458ms	49.41 W	R$ 0.001593

O foco deste trabalho está no consumo energético dos servidores web. Como era de se esperar, no cenário 1, em que menos máquinas estão ligadas, o consumo de energia foi menor. Contudo, a experiência do usuários foi muito pior (como pode ser observado na tabela 1). Além do consumo de energia, a empresa que tem um serviço Web precisa cuidar da experiência do usuário ou o mesmo não retornará ao site e o serviço perderá seu propósito segundo Guerra(2023) . Os usuários tiveram uma experiência significativamente melhor nos cenários em que as requisições foram atendidas por mais de um servidor. Isso pode ser observado nas colunas sucesso, erros e média latência na tabela 1.

Como era de se esperar, houve um impacto na experiência do usuário quando o balanceador de backup precisou ser acionado. Contudo, a variação foi pequena (com uma taxa de sucesso apenas 8% menor que o cenário 2 e tempo de resposta igual ao cenário 2). No entanto, quando se compara o cenário em que o balanceador caiu (cenário 3) com o cenário 1 (em que havia um único servidor), as métricas para avaliar a experiência do usuário também são melhores no cenário 3.

Ao comparar os dois cenários em que a única diferença nos experimentos foi o algoritmo utilizado para o balanceamento de carga (cenários 2 e 4), observamos que a diferença foi mínima. No entanto, notou-se um desempenho ligeiramente inferior com o algoritmo Leastconn. Contudo, é preciso observar que, neste trabalho, estamos comparando experimentos realizados em um intervalo de tempo muito curto. Em um cenário real em que o serviço Web é prestado 24 horas por dia, a diferença pode ser relevante no consumo de energia elétrica.

Foi observado que a` medida que a carga de trabalho aumenta nos servidores, a temperatura da CPU também aumenta, o que resulta em um maior consumo de energia. Isso é evidenciado na Figura 19, onde ao comparar os gráficos lado a lado, fica perceptível que o aumento do consumo de energia segue o aumento da temperatura. Além disso, quando a CPU começa a gerar mais calor, o sistema automaticamente aumenta a rotação das ventoinhas para resfriá-la, o que, por sua vez, gera ainda mais consumo de energia. Portanto, nos testes realizados com o uso do balanceador, observamos que as CPUs passam menos tempo processando quando a carga de trabalho é distribuída entre elas. Isso reduz o tempo de operação das ventoinhas em alta velocidade. O que indica que um maior resfriamento da CPU deverá resultar em um menor consumo de energia (por parte da CPU). Além disso, é importante notar que CPUs gerando calor por menos tempo também contribuem para a redução da temperatura no ambiente. Consequente- mente, se houver um sistema de ar condicionado, ele consumirá menos energia, uma vez que terá menos calor para dissipar. No entanto, é preciso observar que também há um custo envolvido em resfriar a CPU e isso estava fora do escopo deste trabalho.

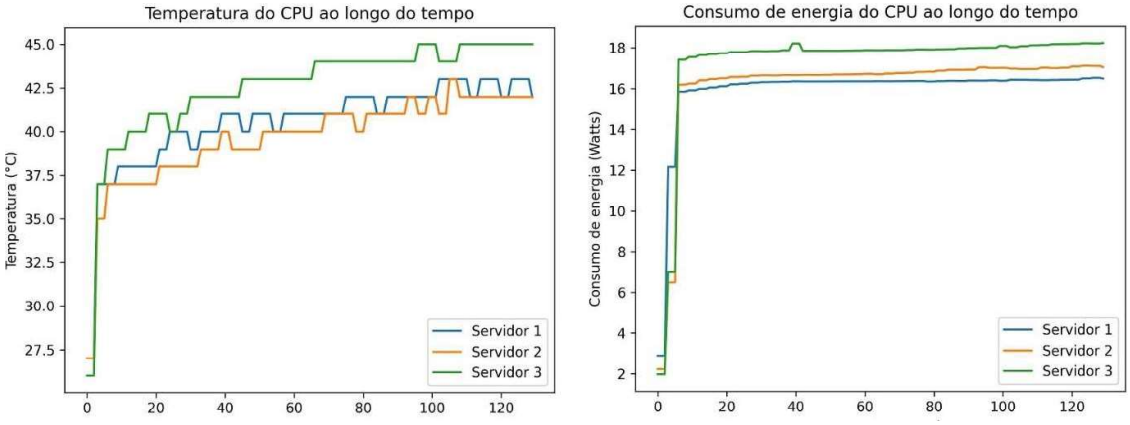

Figura 19. Comparativo: Temperatura e Uso de energia.

5. TRABALHOS RELACIONADOS

O grande consumo de energia elétrica nos datacenters é um problema para as empresas e o balanceamento de carga direcionada a servidores Web pode ser feito de forma a considerar o consumo de energia e, até mesmo, buscar reduzi-lo. Nesse contexto, realizamos uma pesquisa por trabalhos relacionados que tratam do balanceamento de carga com eficiência energética. Em seu trabalho Liu et al. (2011) fazem o balanceamento de carga pensando não na redução do consumo de energia, mas na utilização de energias de fontes renováveis distribuídas geograficamente em diferentes áreas. Em nosso trabalho consideramos que há apenas um fornecedor de energia e que o consumo dessa energia deve ser o menor possível para gerar um menor custo a empresa dona dos servidores.

Alguns autores buscam fazer o balanceamento de forma a desligar máquinas que não estão sendo utilizadas (Andrews et al., 2010; Wierman et al., 2009) este é também o caso de Francois et al. (2013) que buscam concentrar a carga em um menor número de servidores para permitir que outros servidores possam ser desligados. No entanto, é importante observar que uma alta utilização de um CPU pode fazer com que o servidor proporcione altos tempos de resposta para os usuários (Oliveira et al. 2010). Já Zhao et al. (Zhao et al. 2013) argumentam que essa não é uma solução realista e focam na redução do uso dos servidores para buscar uma utilização média dos diferentes servidores.

Pérez et al. (Pérez et al. 2017) estudaram a eficiência energética em um cluster formado por servidores com configurações heterogêneas. Esses autores utilizaram o Maat, uma biblioteca do OpenCL (Pérez et al. 2016), para testar três diferentes algoritmos de balanceamento de carga para uma aplicação que utiliza processamento paralelo. Eles executaram três diferentes benchmarks e observaram que cada um dos três algoritmos de balanceamento conseguiu reduzir o tempo de execução do experimento e, consequentemente, gastar menos energia para cada tipo de carga. A configuração do cluster é uma diferença importante entre o nosso trabalho e o de Pérez et al. (Pérez et al. 2017) já que esses autores utilizaram tanto CPUs como GPUs, enquanto nós utilizamos apenas CPUs. Contudo, em nosso trabalho foi possível observar que o consumo de energia pode variar significativamente dependendo dos algoritmos utilizados para balancear a carga.

6. CONCLUSÃO E TRABALHOS FUTUROS

Neste trabalho realizamos diversos experimentos de balanceamento de carga entre servidores Web virtualizados. Esses experimentos nos permitiram comparar o consumo de energia e a experiência do usuário realizando o balanceamento de carga com diferentes algoritmos. Observamos que o consumo de energia varia, dependendo do algoritmo de balanceamento utilizado, o que está de acordo com o verificado por outros autores (e.g., (Pérez et al. 2017). Neste trabalho, o cenário em que a carga foi balanceada utilizando o algoritmo Round-Robin apresentou a melhor experiência para o usuário (com a maior taxa de sucesso dos servidores no atendimento as requisições) e ainda um menor consumo de energia para o atendimento dessas requisições. É preciso levar em conta que os experimentos que realizamos duraram poucos minutos. Portanto, em um cenário real de servidores rodando 24 horas por dia, pode-se ter uma diferença muito grande no consumo de energia simplesmente mudando o algoritmo que está sendo utilizado para balancear a carga entre os servidores.

É sabido que a CPU é o componente que mais consome energia elétrica nos servidores (Mills, 2021) e que as CPUs atuais permitem a sua utilização em diferentes frequências (Ibrahim et al. 2016). Portanto, em um trabalho futuro seria interessante alternar entre as diferentes frequências dos servidores para verificar se é possível minimizar o consumo de energia enquanto se mantém uma melhor experiência para o usuário. Talvez seja possível melhorar a relação consumo de energia x experiência do usuário ao se olhar para o consumo individual dos componentes do servidor. Mesmo que se alcance a redução de um pequeno percentual no consumo de energia, isso pode ter um grande impacto na conta de energia elétrica do datacenter.

Outra observação que fizemos em nossos experimentos é que há uma relação entre o aquecimento da CPU e o seu consumo de energia. Levando em conta que um melhor resfriamento irá levar a um menor aquecimento e, consequentemente, menor consumo de energia, é interessante investigar melhor a relação valor gasto com resfriamento x valor economizado com o consumo da CPU.

REFERÊNCIAS

Andrews, M., Anta, A. F., Zhang, L., & Zhao, W. (2010). Routing for energy minimization in the speed scaling model. In Anais do IEEE INFOCOM 2010, páginas 1–9. IEEE.

Cassen, A. (2023). **Manpage do Keepalived**. Disponível em: https://www.keepalived.org/manpage.html. Acessado em 20/04/2023.

Edition, H. C. (2023). **HAProxy.** Disponível em: https://www.haproxy.org/. Acessado em 22/04/2023.

Foundation, A. S. (2023). **Apache JMeter.** Disponível em: https://jmeter.apache.org/. Acessado em 22/03/2023.

Francois, F., Wang, N., Moessner, K., Georgoulas, S., & Xu, K. (2013). Green IGP link weights for energy-efficiency and load-balancing in IP backbone networks. In Anais do IFIP Networking Conference 2013, páginas 1–9. IEEE.

Guerra, B. (2023). A importância da experiência do usuário para sua empresa. Disponível em: https://www.mazag.com.br/marketing-digital/a-importancia-da-experiencia-do-usuario-para-sua-empresa/. Acessado em 19/10/2023.

Ibrahim, S., Phan, T.-D., Carpen-Amarie, A., Chihoub, H.-E., Moise, D., & Antoniu, G. (2016). Governing energy consumption in Hadoop through CPU frequency scaling: An analysis. Future Generation Computer Systems, 54, 219–232.

Liu, Z., Lin, M., Wierman, A., Low, S. H., & Andrew, L. L. (2011). Geographical load balancing with renewables. ACM SIGMETRICS Performance Evaluation Review, 39(3), 62–66.

Mills, M. (2021). Componentes de hardware de PC com maior consumo elétrico. Disponível em: https://itigic.com/pt/pc-hardware-components-with-highest-electrical-consumption/. Acessado em 12/06/2023.

Oliveira, C., Petrucci, V., & Loques, O. (2010). Impact of server dynamic allocation on the response time for energy-efficient virtualized web clusters. In Anais do XXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos-12th Brazilian workshop on real-time and embedded systems (WTR).

Pérez, B., Bosque, J. L., & Beivide, R. (2016). Simplifying programming and load balancing of data parallel applications on heterogeneous systems. In Proceedings of the 9th Annual Workshop on General Purpose Processing using Graphics Processing Unit, páginas 42–51.

Pérez, B., Stafford, E., Bosque, J. L., & Beivide, R. (2017). Energy efficiency of load balancing for data-parallel applications in heterogeneous systems. The Journal of Supercomputing, 73(1), 330–342.

Software, H. D. (2023). HWiNFO. Disponível em: https://www.hwinfo.com/. Acessado em 22/05/2023.

Spa, E. (s.d.). Taxas, tarifas e impostos. Disponível em: https://www.enel.com.br/pt/Tarifas_Enel.html. Acessado em 15/06/2023.

Tableau. (s.d.). Collecting Performance Data Using Windows Performance Monitor. Disponível em: https://help.tableau.com/current/server/pt-br/perf_collect_perfmon.htm. Acessado em 20/04/2023.

Technologies, H. (s.d.). HAProxy documentation. Disponível em: https://www.haproxy.com/blog/fundamentals-load-balancing-and-the-right-distribution-algorithm-for/. Acessado em 22/05/2023.

Technologies, H. (s.d.). Quick Introduction to Load Balancing. Disponível em: http://docs.haproxy.org/2.8/intro.html#2. Acessado em 22/05/2023.

Wierman, A., Andrew, L. L., & Tang, A. (2009). Power-aware speed scaling in processor sharing systems. In Anais do IEEE INFOCOM 2009, páginas 2007–2015. IEEE.

Zhao, Y., Wang, S., Xu, S., Wang, X., Gao, X., & Qiao, C. (2013). Load balance vs energy efficiency in traffic engineering: A game theoretical perspective. In Anais do IEEE INFOCOM 2013, páginas 530–534. IEEE.

APÊNDICE A – CÓDIGO FONTE PARA INICIAR COLETA

Neste apêndice, apresentamos o código fonte em Java utilizado para a coleta de dados mencionados na Sessão 2.

Código 1: Enviando comando para o servidor iniciar a coleta dos dados.

```java
public class CommandSender {

    public static void main(String[] args) {
        // Obtendo o endereço IP do primeiro argumento da linha de comando
        String ipAddress = args[0];

        // Definindo a porta do servidor
        int port = 34569;

        // Comando a ser enviado ao servidor
        String commandToSend = "executar";

        try {
            // Criando um socket para se conectar ao servidor no endereço e porta especificados
            try (Socket socket = new Socket(ipAddress, port)) {
                // Obtendo o stream de saída do socket
                OutputStream out = socket.getOutputStream();

                // Convertendo o comando em bytes e enviando ao servidor
                out.write(commandToSend.getBytes());

                // Forçando a escrita imediata dos dados
                out.flush();
            }
        } catch (IOException e) {
            // Lidando com exceções de entrada/saída (IO)
            e.printStackTrace();
        }
    }
}
```

Código 2: Recebendo o comando para iniciar gravação.

```
import java.awt.Robot;
import java.awt.event.KeyEvent;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.ServerSocket;
import java.net.Socket;

public class PortListener {

    public static void main(String[] args) {
        // Porta na qual o servidor estará ouvindo
        int port = 34569;

        try {
            // Criando um ServerSocket para ouvir conexões na porta especificada
            try (ServerSocket serverSocket = new ServerSocket(port)) {
                System.out.println("Ouvindo a porta " + port);

                // Aguardando por conexões
                while (true) {
                    // Aceitando uma conexão do cliente
                    Socket clientSocket = serverSocket.accept();
                    System.out.println("Conexão recebida");

                    // Configurando um BufferedReader para ler o comando enviado pelo
                    cliente
                    BufferedReader in = new BufferedReader(new InputStreamReader(
                    clientSocket.getInputStream()));
                    String command = in.readLine();

                    // Verificando se o comando recebido é "executar"
                    if (command.equals("executar")) {
                        System.out.println("Comando 'executar' recebido. Simulando
                        pressão da tecla HOME...");

                        // Criando uma instância de Robot para interagir com o sistema
                        operacional
                        Robot robot = new Robot();

                        // Simulando a pressão e liberação da tecla HOME
                        robot.keyPress(KeyEvent.VK_HOME);
                        robot.keyRelease(KeyEvent.VK_HOME);

                        System.out.println("Pressionou a tecla HOME");
                    }

                    // Fechando a conexão com o cliente
                    clientSocket.close();
                    System.out.println("Conexão fechada");
                }
            }

        } catch (IOException e) {
            // Lidando com exceções de entrada/saída (IO)
            e.printStackTrace();
        } catch (Exception e) {
            // Lidando com outras exceções que podem ocorrer
            e.printStackTrace();
        }
    }
}

```

APÊNDICE B – CÓDIGO FONTE PARA ANÁLISE E CRIAÇÃO DE GRÁFICOS

Neste apêndice, apresentamos o código fonte em Python utilizado para processar os dados coletados gerando estatísticas e gráficos, mencionados na Sessão 2.

Código 1: Processando dados e criando gráfico sobre uso de CPU.

```
import datetime
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Ler o arquivo "inicio_fim.txt" e obter a string com horário de início e fim
with open('inicio_fim.txt', 'r') as file:
    data = file.read().strip()

# Extrair o horário de inicio e fim da string
inicio = data[data.find('inicio_') + len('inicio_'):data.find('_fim_')]
fim = data[data.find('_fim_') + len('_fim_'):]

# Converter as strings em objetos de data e hora
try:
    inicio_datetime = datetime.datetime.strptime(inicio, '%H:%M:%S,%f')
    fim_datetime = datetime.datetime.strptime(fim, '%H:%M:%S,%f')
except ValueError:
    print("Formato de horário inválido. Certifique-se de que o formato seja
    HH:MM:SS,SSS.")
    exit()

# Calcular a diferença de tempo em segundos
diferenca = fim_datetime - inicio_datetime
segundos = diferenca.total_seconds()

# Ler os dados dos arquivos CSV
df_ser1 = pd.read_csv('ser1.csv', encoding='cp1252')
df_ser2 = pd.read_csv('ser2.csv', encoding='cp1252')
df_ser3 = pd.read_csv('ser3.csv', encoding='cp1252')

# Obter os dados do eixo x
x = np.arange(int(segundos))

# Obter os dados do eixo y para cada servidor
cpu_ser1 = df_ser1['Uso total da CPU [%]'].values
cpu_ser2 = df_ser2['Uso total da CPU [%]'].values
cpu_ser3 = df_ser3['Uso total da CPU [%]'].values

```

```
# Ajustar os dados do eixo y ao mesmo número de pontos do eixo x
num_points = len(x)

# Verificar se cpu_ser1 tem mais ou menos pontos
if len(cpu_ser1) != num_points:
    cpu_ser1 = np.repeat(cpu_ser1, num_points // len(cpu_ser1) + 1)[:num_points]

# Verificar se cpu_ser2 tem mais ou menos pontos
if len(cpu_ser2) != num_points:
    cpu_ser2 = np.repeat(cpu_ser2, num_points // len(cpu_ser2) + 1)[:num_points]

# Verificar se cpu_ser3 tem mais ou menos pontos
if len(cpu_ser3) != num_points:
    cpu_ser3 = np.repeat(cpu_ser3, num_points // len(cpu_ser3) + 1)[:num_points]

# Plotar o gráfico
plt.plot(x, cpu_ser1, label='Servidor 1')
plt.plot(x, cpu_ser2, label='Servidor 2')
plt.plot(x, cpu_ser3, label='Servidor 3')
plt.xlabel("Tempo: " + str(int(segundos)) + " segundos")
plt.ylabel('Uso total da CPU (%)')
plt.title('Uso da CPU ao longo do tempo')
plt.legend()
plt.savefig('grafico_cpu.png', dpi=300)
#plt.show()
plt.close()

# Calcular as estatísticas


uso_min_ser1 = np.min(cpu_ser1)
uso_min_ser2 = np.min(cpu_ser2)
uso_min_ser3 = np.min(cpu_ser3)

uso_max_ser1 = np.max(cpu_ser1)
uso_max_ser2 = np.max(cpu_ser2)
uso_max_ser3 = np.max(cpu_ser3)

media_ser1 = np.mean(cpu_ser1)
media_ser2 = np.mean(cpu_ser2)
media_ser3 = np.mean(cpu_ser3)

media combinada = np.mean([media ser1, media ser2, media ser3])

# Adicionar informações ao arquivo Respostas.txt
with open('Respostas.txt', 'a') as file:
    file.write("\n\nUso de CPU dos servidores:\n")
    file.write("Uso mínimo de CPU dos servidores:\n")
    file.write(f"Servidor 1: {uso_min_ser1}%\n")
    file.write(f"Servidor 2: {uso_min_ser2}%\n")
    file.write(f"Servidor 3: {uso_min_ser3}%\n\n")

    file.write("Uso máximo de CPU dos servidores:\n")
    file.write(f"Servidor 1: {uso_max_ser1}%\n")
    file.write(f"Servidor 2: {uso_max_ser2}%\n")
    file.write(f"Servidor 3: {uso_max_ser3}%\n\n")

    file.write("Médias de uso de CPU:\n")
    file.write(f"Servidor 1: {int(media_ser1)}%\n")
    file.write(f"Servidor 2: {int(media_ser2)}%\n")
    file.write(f"Servidor 3: {int(media_ser3)}%\n\n")

    file.write(f"Média combinada: {int(media_combinada)}%\n")

print("Informações adicionadas ao arquivo Respostas.txt.")
```

Código 2: Processando dados e criando gráfico sobre uso de energia da CPU.

```
import datetime
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Ler o arquivo "inicio_fim.txt" e obter a string com horário de inicio e fim
with open('inicio_fim.txt', 'r') as file:
    data = file.read().strip()

# Extrair o horário de inicio e fim da string
inicio = data[data.find('inicio_') + len('inicio_'):data.find('_fim_')]
fim = data[data.find('_fim_') + len('_fim_'):]

# Converter as strings em objetos de data e hora
try:
    inicio_datetime = datetime.datetime.strptime(inicio, '%H:%M:%S,%f')
    fim_datetime = datetime.datetime.strptime(fim, '%H:%M:%S,%f')
except ValueError:
    print("Formato de horário inválido. Certifique-se de que o formato seja
    HH:MM:SS,SSS.")
    exit()

# Calcular a diferença de tempo em segundos
diferenca = fim_datetime - inicio_datetime
segundos = diferenca.total_seconds()

# Ler os dados dos arquivos CSV
df_ser1 = pd.read_csv('ser1.csv', encoding='cp1252')
df_ser2 = pd.read_csv('ser2.csv', encoding='cp1252')
df_ser3 = pd.read_csv('ser3.csv', encoding='cp1252')

# Obter os dados do eixo x
x = np.arange(int(segundos))

# Obter os dados do eixo y para cada servidor
energia_ser1 = df_ser1['Consumo de energia total da CPU [W]'].values
energia_ser2 = df_ser2['Consumo de energia total da CPU [W]'].values
energia_ser3 = df_ser3['Consumo de energia total da CPU [W]'].values

# Ajustar os dados do eixo y ao mesmo número de pontos do eixo x
num_points = len(x)

# Verificar se energia_ser1 tem mais ou menos pontos
if len(energia_ser1) != num_points:
    energia_ser1 = np.repeat(energia_ser1, num_points // len(energia_ser1) + 1)[:
    num_points]

# Verificar se energia_ser2 tem mais ou menos pontos
if len(energia_ser2) != num_points:
    energia_ser2 = np.repeat(energia_ser2, num_points // len(energia_ser2) + 1)[:
    num_points]

# Verificar se energia_ser3 tem mais ou menos pontos
if len(energia_ser3) != num_points:
    energia_ser3 = np.repeat(energia_ser3, num_points // len(energia_ser3) + 1)[:
    num_points]

# Plotar o gráfico
plt.plot(x, energia_ser1, label='Servidor 1')
plt.plot(x, energia_ser2, label='Servidor 2')
plt.plot(x, energia_ser3, label='Servidor 3')
plt.xlabel("Tempo: "+str(int(segundos))+" segundos")
plt.ylabel('Consumo de energia (Watts)')
plt.title('Consumo de energia do CPU ao longo do tempo')
plt.legend()
plt.savefig('grafico_energia.png', dpi=300)
#plt.show()
```

```
64
65  # Calcular o uso mínimo, máximo e médio de energia para cada servidor
66  uso_min_ser1 = (np.min(energia_ser1))
67  uso_max_ser1 = (np.max(energia_ser1))
68  uso_medio_ser1 = (np.mean(energia_ser1))
69
70  uso_min_ser2 = (np.min(energia_ser2))
71  uso_max_ser2 = (np.max(energia_ser2))
72  uso_medio_ser2 = (np.mean(energia_ser2))
73
74  uso_min_ser3 = (np.min(energia_ser3))
75  uso_max_ser3 = (np.max(energia_ser3))
76  uso_medio_ser3 = (np.mean(energia_ser3))
77
78  # Calcular o uso total de energia e o gasto em reais durante o teste para cada servidor
79
80  taxa_energia = 0.88834  # taxa em reais por kWh
81  gasto_ser1 = ((uso_medio_ser1 / 1000) * (segundos / 3600))* taxa_energia
82  gasto_ser2 = ((uso_medio_ser2 / 1000) * (segundos / 3600))* taxa_energia
83  gasto_ser3 = ((uso_medio_ser3 / 1000) * (segundos / 3600))* taxa_energia
84
85
86
87  # Calcular a média de uso e o uso total de energia combinado dos 3 servidores
88  uso_medio_combinado = (uso_medio_ser1 +uso_medio_ser2+ uso_medio_ser3)
89  gasto_combinado = gasto_ser1 + gasto_ser2 + gasto_ser3
90
91
92  # Gravar as informações no arquivo Respostas.txt
93  with open('Respostas.txt', 'a') as file:
94      file.write("\n\nUso de energia (Watts):\n")
95      file.write("Potência mínima (Watts):\n")
96      file.write(f"Servidor 1: {uso_min_ser1:.2f} Watts\n")
97      file.write(f"Servidor 2: {uso_min_ser2:.2f} Watts\n")
98      file.write(f"Servidor 3: {uso_min_ser3:.2f} Watts\n\n")
99
100     file.write("Potência máxima (Watts):\n")
101     file.write(f"Servidor 1: {uso_max_ser1:.2f} Watts\n")
102     file.write(f"Servidor 2: {uso_max_ser2:.2f} Watts\n")
103     file.write(f"Servidor 3: {uso_max_ser3:.2f} Watts\n\n")
104
105     file.write("Potência média (Watts):\n")
106     file.write(f"Servidor 1: {uso_medio_ser1:.2f} Watts\n")
107     file.write(f"Servidor 2: {uso_medio_ser2:.2f} Watts\n")
108     file.write(f"Servidor 3: {uso_medio_ser3:.2f} Watts\n\n")
109
110     file.write(f"Uso de energia (Watts) durante o teste:\nR$: {taxa_energia} taxa por
        kWh no RJ(jun 2023, bandeira verde)\n\n")
111     file.write(f"Servidor 1: {uso_medio_ser1:.2f} Watts\n")
112     file.write(f"Gasto em reais (Servidor 1): R$ {gasto_ser1:.6f}\n\n")
113     file.write(f"Servidor 2: {uso_medio_ser2:.2f} Watts\n")
114     file.write(f"Gasto em reais (Servidor 2): R$ {gasto_ser2:.6f}\n\n")
115     file.write(f"Servidor 3: {uso_medio_ser3:.2f} Watts\n")
116     file.write(f"Gasto em reais (Servidor 3): R$ {gasto_ser3:.6f}\n\n")
117
118     file.write("Potência (Watts) combinando os 3 servidores:\n")
119     file.write(f"Potência somada: {uso_medio_combinado:.2f} Watts\n")
120     file.write(f"Gasto em reais combinado dos 3 servidores: R$ {gasto_combinado:.6f}\n\n"
        )
121
```

Código 3: Processando dados e criando gráfico sobre tempo de resposta das solicitações.

```
import datetime
import pandas as pd
import matplotlib.pyplot as plt

# Ler o arquivo "inicio_fim.txt" e obter a string com horário de início e fim
with open('inicio_fim.txt', 'r') as file:
    data = file.read().strip()

# Extrair o horário de início e fim da string
inicio = data[data.find('inicio_') + len('inicio_'):data.find('_fim_')]
fim = data[data.find('_fim_') + len('_fim_'):]

# Converter as strings em objetos de data e hora
try:
    inicio_datetime = datetime.datetime.strptime(inicio, '%H:%M:%S,%f')
    fim_datetime = datetime.datetime.strptime(fim, '%H:%M:%S,%f')
except ValueError:
    print("Formato de horário inválido. Certifique-se de que o formato seja
    HH:MM:SS,SSS.")
    exit()

# Calcular a diferença de tempo em segundos
diferenca = fim_datetime - inicio_datetime
segundos = diferenca.total_seconds()

# Ler os dados do arquivo CSV
df = pd.read_csv('resultados.csv')

# Filtrar os dados para respostas verdadeiras
respostas_verdadeiras = df[df['success'] == True]

# Filtrar os dados para respostas falsas
respostas_falsas = df[df['success'] == False]

# Obter os tempos de resposta das respostas verdadeiras
tempos_resposta_verdadeiros = respostas_verdadeiras['Latency'].values

# Obter os tempos de resposta das respostas falsas
tempos_resposta_falsas = respostas_falsas['Latency'].values

# Obter as posições (em segundos) das respostas verdadeiras
posicoes_verdadeiras = respostas_verdadeiras.index.values

# Obter as posições (em segundos) das respostas falsas
posicoes_falsas = respostas_falsas.index.values

# Definir a quantidade de amostras desejada
quantidade_amostras = 5

# Realizar amostragem dos valores e rótulos do eixo x
x_values = posicoes_verdadeiras.tolist() + posicoes_falsas.tolist()
x_labels = [str(int(segundos * x / len(x_values))) for x in x_values[::len(x_values) //
quantidade_amostras]]
x_values = x_values[::len(x_values) // quantidade_amostras]

# Plotar o gráfico
plt.scatter(posicoes_verdadeiras, tempos_resposta_verdadeiros, marker='o', color='blue',
label=f'{len(posicoes_verdadeiras)} Respostas bem-sucedidas',s=3)
plt.scatter(posicoes_falsas, tempos_resposta_falsas, marker='x', color='red', label=f'{
len(posicoes_falsas)} erros',s=3)

# Definir os valores e rótulos do eixo x
plt.xticks(x_values, x_labels)

# Obter a quantidade de respostas
quantidade_respostas = len(posicoes_verdadeiras) + len(posicoes_falsas)

```

```
# Modificar o rótulo do eixo x
plt.xlabel(f'Tempo do Teste {int(segundos)} segundos, {quantidade_respostas} respostas')

plt.ylabel('Tempo de Resposta (milissegundos)')
plt.title('Tempo de Resposta (ms) ao longo do Tempo')
plt.legend()
plt.savefig('grafico_tempo_resposta.png', dpi=300)
#plt.show()
```

Código 4: Processando dados e criando gráfico sobre temperatura da CPU.

```
import datetime
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Ler o arquivo "inicio_fim.txt" e obter a string com horário de início e fim
with open('inicio_fim.txt', 'r') as file:
    data = file.read().strip()

# Extrair o horário de início e fim da string
inicio = data[data.find('inicio_') + len('inicio_'):data.find('_fim_')]
fim = data[data.find('_fim_') + len('_fim_'):]

# Converter as strings em objetos de data e hora
try:
    inicio_datetime = datetime.datetime.strptime(inicio, '%H:%M:%S,%f')
    fim_datetime = datetime.datetime.strptime(fim, '%H:%M:%S,%f')
except ValueError:
    print("Formato de horário inválido. Certifique-se de que o formato seja
    HH:MM:SS,SSS.")
    exit()

# Calcular a diferença de tempo em segundos
diferenca = fim_datetime - inicio_datetime
segundos = diferenca.total_seconds()

# Ler os dados dos arquivos CSV
df_ser1 = pd.read_csv('ser1.csv', encoding='cp1252')
df_ser2 = pd.read_csv('ser2.csv', encoding='cp1252')
df_ser3 = pd.read_csv('ser3.csv', encoding='cp1252')

# Obter os dados do eixo x
x = np.arange(int(segundos))

# Obter os dados do eixo y para cada servidor
temperatura_ser1 = df_ser1['CPU Inteira [°C]'].values
temperatura_ser2 = df_ser2['CPU Inteira [°C]'].values
temperatura_ser3 = df_ser3['CPU Inteira [°C]'].values

# Ajustar os dados do eixo y ao mesmo número de pontos do eixo x
num_points = len(x)

# Verificar se temperatura_ser1 tem mais ou menos pontos
if len(temperatura_ser1) != num_points:
    temperatura_ser1 = np.repeat(temperatura_ser1, num_points // len(temperatura_ser1) +
    1)[:num_points]

# Verificar se temperatura_ser2 tem mais ou menos pontos
if len(temperatura_ser2) != num_points:
    temperatura_ser2 = np.repeat(temperatura_ser2, num_points // len(temperatura_ser2) +
    1)[:num_points]
```

```
# Verificar se temperatura_ser3 tem mais ou menos pontos
if len(temperatura_ser3) != num_points:
    temperatura_ser3 = np.repeat(temperatura_ser3, num_points // len(temperatura_ser3) + 1)[:num_points]

# Encontrar a temperatura mínima e máxima de cada servidor
temperatura_minima_ser1 = np.min(temperatura_ser1)
temperatura_maxima_ser1 = np.max(temperatura_ser1)
temperatura_minima_ser2 = np.min(temperatura_ser2)
temperatura_maxima_ser2 = np.max(temperatura_ser2)
temperatura_minima_ser3 = np.min(temperatura_ser3)
temperatura_maxima_ser3 = np.max(temperatura_ser3)

# Calcular a taxa de aquecimento de cada servidor em porcentagem
taxa_aquecimento_ser1 = (temperatura_maxima_ser1 - temperatura_minima_ser1) / temperatura_minima_ser1 * 100
taxa_aquecimento_ser2 = (temperatura_maxima_ser2 - temperatura_minima_ser2) / temperatura_minima_ser2 * 100
taxa_aquecimento_ser3 = (temperatura_maxima_ser3 - temperatura_minima_ser3) / temperatura_minima_ser3 * 100

# Adicionar as informações ao arquivo "Respostas.txt"
with open('Respostas.txt', 'a') as file:
    file.write("\nTemperatura mínima do Servidor 1: {}°C\n".format(temperatura_minima_ser1))
    file.write("Temperatura máxima do Servidor 1: {}°C\n".format(temperatura_maxima_ser1))
    file.write("Taxa de aquecimento do Servidor 1: {:.2f}%\n".format(taxa_aquecimento_ser1))
    file.write("\nTemperatura mínima do Servidor 2: {}°C\n".format(temperatura_minima_ser2))
    file.write("Temperatura máxima do Servidor 2: {}°C\n".format(temperatura_maxima_ser2))
    file.write("Taxa de aquecimento do Servidor 2: {:.2f}%\n".format(taxa_aquecimento_ser2))
    file.write("\nTemperatura mínima do Servidor 3: {}°C\n".format(temperatura_minima_ser3))
    file.write("Temperatura máxima do Servidor 3: {}°C\n".format(temperatura_maxima_ser3))
    file.write("Taxa de aquecimento do Servidor 3: {:.2f}%\n".format(taxa_aquecimento_ser3))

# Plotar o gráfico
plt.plot(x, temperatura_ser1, label='Servidor 1')
plt.plot(x, temperatura_ser2, label='Servidor 2')
plt.plot(x, temperatura_ser3, label='Servidor 3')
plt.xlabel("Tempo: {} segundos".format(int(segundos)))
plt.ylabel('Temperatura (°C)')
plt.title('Temperatura do CPU ao longo do tempo')
plt.legend()
plt.savefig('grafico_temperatura.png', dpi=300)
#plt.show()

```