PGCOMP - Programa de Pós-Graduação em Ciência da Computação
Universidade Federal da Bahia (UFBA)
Av. Milton Santos, s/n - Ondina
Salvador, BA, Brasil, 40170-110

https://pgcomp.ufba.br
pgcomp@ufba.br

A computação em nuvem oferece uma infraestrutura centralizada e orientada a serviços para usuários em todo o mundo. Um ambiente em nuvem permite executar aplicações críticas sensíveis à interação com os usuários (como uma busca online na Web), e também é usado para executar aplicações em lote orientadas a melhor esforço (como compressão de áudio/vídeo, indexação das páginas Web). Data centers são projetados em larga escala para suportar os diversos serviços em nuvem, distribuídos em centenas de milhares de servidores, com expressivos gastos operacionais e de capital. Assim, otimizar o uso dos recursos dos servidores resulta em economia significativa nesses sistemas. Como oportunidade de pesquisa, há a possibilidade de explorar momentos de ociosidade dos servidores que executam os serviços críticos, em especial durante períodos de baixa demanda. Esta pesquisa visa permitir coalocar diferentes tipos de processos de aplicações em nuvem (sensíveis à latência e de melhor esforço) em um mesmo servidor de forma a aumentar sua utilização, melhorando assim sua eficiência. O trabalho demonstra que realizar coalocação de processos sensíveis à latência e de melhor esforço proporciona maior eficiência dos servidores em nuvem, desde que seja garantido o desempenho das processos das aplicação críticas. O trabalho explorou o uso de escalonadores específicos para cada tipo de processo, as processos sensíveis à latência foram escalonadas e parametrizadas via escalonamento de tempo real e processos de aplicações em lote utilizaram escalonamento justo de melhor esforço. Os ganhos em relação ao escalonador padrão do sistema operacional foram consideráveis, chegando a melhores resultados de até 150% quando comparado ao proposto neste trabalho, aqui chamado de EDFCoaloc.

Palavras-chave: computação em nuvem, escalonadores, aplicações sensíveis à latência.

MSC | 134 | 2022

Melhorando a eficiência de sistemas em nuvem através de coalocação de processos sensíveis à latência e de melhor esforço

Alan Teixeira de Oliveira

UFBA

Melhorando a eficiência de sistemas em nuvem através de coalocação de processos sensíveis à latência e de melhor esforço

Alan Teixeira de Oliveira

Dissertação de Mestrado

Universidade Federal da Bahia

Programa de Pós-Graduação em
Ciência da Computação

Abril | 2022

Universidade Federal da Bahia
Instituto de Computação

Programa de Pós-graduação em Ciência da Computação

MELHORANDO A EFICIÊNCIA DE SISTEMAS EM NUVEM ATRAVÉS DE COALOCAÇÃO DE PROCESSOS SENSÍVEIS À LATÊNCIA E DE MELHOR ESFORÇO

Alan Teixeira de Oliveira

DISSERTAÇÃO DE MESTRADO

Salvador
01 de Abril de 2022

ALAN TEIXEIRA DE OLIVEIRA

MELHORANDO A EFICIÊNCIA DE SISTEMAS EM NUVEM ATRAVÉS DE COALOCAÇÃO DE PROCESSOS SENSÍVEIS À LATÊNCIA E DE MELHOR ESFORÇO

Esta Dissertação de Mestrado foi apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal da Bahia, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: Vinicius Petrucci

Salvador
01 de Abril de 2022

Ficha catalográfica elaborada pela Biblioteca Universitária de Ciências e Tecnologias Prof. Omar Catunda, SIBI – UFBA.

O48 Oliveira, Alan Teixeira de

Melhorando a eficiência de sistemas em nuvem através de coalocação de processos sensíveis à latência e de melhor esforço / Alan Teixeira de Oliveira. – Salvador, 2022.

64 f.

Orientador: Prof. Dr. Vinícius Tavares Petrucci

Dissertação (Mestrado) – Universidade Federal da Bahia. Instituto de Computação, 2022.

1. Computação em nuvem. 2. Escalonadores. 3. Aplicações sensíveis às latências. I. Petrucci, Vinícius Tavares. II. Universidade Federal da Bahia. III. Título.

004.45

"*Melhorando a eficiência de sistemas em nuvem através de co-alocação de tarefas sensíveis à latência e de melhor esforço*"

Alan Teixeira de Oliveira

Dissertação apresentada ao Colegiado do Programa de Pós-Graduação em Ciência da Computação na Universidade Federal da Bahia, como requisito parcial para obtenção do Título de Mestre em Ciência da Computação.

Banca Examinadora

DocuSigned by:
Vinicius Petrucci
4DE4BB4DB5134E1...

Prof. Dr. VINICIUS TAVARES PETRUCCI (Orientador PGCOMP)

DocuSigned by:
George Marconi F. de Araujo Lima
26F6B9686F09424...

Prof. Dr.GEORGE MARCONI DE ARAUJO LIMA (PGCOMP/UFBA)

DocuSigned by:
E52C4F7FA0D348C...

Prof. Dr.Esbel Tomás Valero Orellana(UESC)

Para Daiane, Lucas e Felipe, amo vocês! E a todos aqueles que acreditaram e me incentivaram em todo esse processo. Muito Obrigado!

AGRADECIMENTOS

Gostaria de expressar meus sinceros agradecimentos a todas as pessoas que contribuíram para a realização deste trabalho científico. É com imensa gratidão que reconheço o apoio e o incentivo recebidos ao longo dessa jornada. Em especial, quero agradecer ao meu orientador,Vinícius, pela sua orientação, paciência e dedicação em me guiar durante todo o processo. Suas sugestões, conhecimento e apoio foram fundamentais para o desenvolvimento deste trabalho. Agradeço também à minha família, em especial a minha esposa Daiane pelo constante encorajamento e apoio incondicional. Seu amor, compreensão e suporte foram pilares essenciais para superar as dificuldades encontradas ao longo do caminho. Não posso deixar de mencionar meus pais, Marinália e Aldemar, que sempre acreditaram em mim e sempre buscaram me proporcionara as melhores oportunidades de educação. Seu amor, incentivo e sacrifícios foram fundamentais para eu chegar até aqui. Ao amigo Thiago e ao meu irmão Leonan colegas de curso e companheiros de estudos, obrigado pelas viagens e madrugadas de estudo. Agradeço aos meus amigos, colegas de trabalho, que estiveram ao meu lado durante essa jornada desafiadora. Suas palavras de encorajamento, discussões construtivas e momentos de descontração fizeram toda a diferença para manter minha motivação e perseverança. Não posso deixar de mencionar as dificuldades encontradas ao longo do percurso. Os obstáculos e desafios enfrentados me fizeram crescer como pesquisador e como pessoa. No entanto, com o apoio e incentivo de todos mencionados anteriormente, pude superar essas dificuldades e seguir em frente. Por fim, gostaria de expressar minha profunda gratidão a todas as pessoas que contribuíram, direta ou indiretamente, para o sucesso deste trabalho científico. Seu apoio, compreensão e encorajamento foram fundamentais para alcançar os resultados obtidos. Obrigado a todos!

"Feliz de quem passa pela vida; tendo mil razões para vivê-la..."

—D. HÉLDER CÂMARA

RESUMO

A computação em nuvem oferece uma infraestrutura centralizada e orientada a serviços para usuários em todo o mundo. Um ambiente em nuvem permite executar aplicações críticas sensíveis à interação com os usuários (como uma busca online na Web), e também é usado para executar aplicações em lote orientadas a melhor esforço (como compressão de áudio/vídeo, indexação das páginas Web). *Data centers* são projetados em larga escala para suportar os diversos serviços em nuvem, distribuídos em centenas de milhares de servidores, com expressivos gastos operacionais e de capital. Assim, otimizar o uso dos recursos dos servidores resulta em economia significativa nesses sistemas. Como oportunidade de pesquisa, há a possibilidade de explorar momentos de ociosidade dos servidores que executam os serviços críticos, em especial durante períodos de baixa demanda. Esta pesquisa visa permitir coalocar diferentes tipos de processos de aplicações em nuvem (sensíveis à latência e de melhor esforço) em um mesmo servidor de forma a aumentar sua utilização, melhorando assim sua eficiência. O trabalho demonstra que realizar coalocação de processos sensíveis à latência e de melhor esforço proporciona maior eficiência dos servidores em nuvem, desde que seja garantido o desempenho das processos das aplicação críticas. O trabalho explorou o uso de escalonadores específicos para cada tipo de processo, as processos sensíveis à latência foram escalonadas e parametrizadas via escalonamento de tempo real e processos de aplicações em lote utilizaram escalonamento justo de melhor esforço. Os ganhos em relação ao escalonador padrão do sistema operacional foram consideráveis, chegando a melhores resultados de até 150% quando comparado ao proposto neste trabalho, aqui chamado de EDFCoaloc.

Palavras-chave: Computação em Nuvem, Escalonadores, Aplicações Sensíveis à Latência

ABSTRACT

Cloud computing offers a centralized and service-oriented infrastructure for users worldwide. A cloud environment allows for the execution of critical user interaction-sensitive applications(such as online web searches). And it is also used to run best-effort batch-oriented applications, such as audio/video compression and web page indexing. Data centers are designed on a large scale to support the various cloud services distributed across hundreds of thousands of servers, with significant operational and capital expenses. Hence, optimizing the use of server resources can result in substantial cost savings in these systems. The possibility of exploring idle moments of servers that perform critical services, especially during periods of low demand, presents itself as an encouraging research opportunity. Therefore, this research aims to allow the co-alocation of different cloud application processes (latency-sensitive and best-effort) on the same server to increase utilization, thus, improving efficiency. The study demonstrates that co-locating processes, being latency-sensitive or best-effort, deliver greater efficiency of cloud servers, provided that the performance of critical application processes is guaranteed. The study explored using specific schedulers for each type of process. Latency-sensitive ones were scheduled and parameterized via real-time scheduling. Batch application processes used fair best-effort scheduling. The proposed scheduler, EDFCoaloc, showed promising results, up to 150% better than the standard operating system scheduler.

Keywords: Cloud Computing, Schedulers, Latency Sensitive Applications

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

Capítulo 1

INTRODUÇÃO

A popularização e crescimento de serviços como e-mail, busca online, redes sociais, associados ao aumento das velovidades de acesso a internet fornecidos pelos provedores de serviços, impulsionaram o modelo de comunicação centrada no servidor (*service-ride*). Esses serviços demandam uma grande quantidade de armazenamento e processamento de dados. Por conta dessa demanda, computadores específicos em larga escala são destinadas a esse fim. A organização e hospedagem destes computadores em locais centralizados e em uma estrutura destinada para tal é chamada de *data center* (ZATS et al., 2012).

Com a tendência de centralização dos serviços, a computação em nuvem oferece infraestrutura de tecnologia da informação (TI) orientada a serviços para usuários em todo o mundo. Com base em um modelo de pague e use (*pay-as-you-go*), a computação em nuvem permite a hospedagem de aplicativos dos mais diversos, com diferentes características de consumo de recurso e com diferentes aplicações. (BUYYA; BELOGLAZOV; ABAWAJY, 2010). Neste sentido, sendo esse tipo de serviço acessível, podendo facilmente ser contratado e ampliado, os desenvolvedores de novos serviços de Internet não precisam de aportes volumosos de capital em hardware para implantar seu serviço ou dispender de recurso para despesa em equipe para operação e manutenção de uma infraestrutura. A computação em nuvem oferece benefícios significativos para empresas em geral, libertando-as do processo de baixo nível, como a criação de infra-estruturas básicas de hardware e software e, assim, possibilitando foco na inovação e criação valor comercial para seus serviços (BUYYA; BELOGLAZOV; ABAWAJY, 2010).

Um ambiente em nuvem permite executar aplicações críticas sensíveis a respostas dos usuários (como busca na Web), bem como pode ser usado para executar aplicação em lote orientadas a melhor esforço (*batch jobs*). Exemplos de processos em lote incluem a análise de grades volumes de dados, aplicação de inteligência artificial, dentre outras e a vazão de execução dessas aplicações é o fator mais importante para essos processos (BARROSO; CLIDARAS; HÖLZLE, 2013). Como os *data centers* são extremamente caros para serem construídos, é muito importante utilizar plenamente o máximo de sua capacidade disponível com o objetivo de reduzir o custo total de propriedade (TCO, Total Cost Ownership).

Diversas características restringem a capacidade de um *data center*. Eles são projetados de modo que busque-se a melhor ocupação espacial e sua infraestrutura deve ser capaz de fornecer energia suficiente para o atendimento das demandas dos equipamentos em seu maior pico de uso. Mas o que realmente importa para as aplicações é a capacidade de processamento do *data center*, isto é, quanto de recursos computacionais (ciclos de CPU, memória, espaço de armazenamento) o *data center* pode fornecer e, em alguns casos, garantir acordos de nível de serviço (SLAs), que definem as políticas de atendimentos conforme a demanda de requisições e serviços (WANG et al., 2016).

A aquisição de novos servidores contribui para mais da metade do custo total de propriedade (TCO) dos data centers modernos (BARROSO; CLIDARAS; HÖLZLE, 2013). No entanto, a utilização do servidor é baixa em data centers que hospedam grandes serviços sensíveis à latência por dois motivos principais: os serviços sensíveis à latência são normalmente provisionados para o pico de demanda, que ocorre apenas por uma fração do tempo total de execução (LEVERICH; KOZYRAKIS, 2014) e o processo de continuidade de funcionamento determinam o limite para tolerância de paradas importantes do data center, que pode reduzir o custo operacional. O alto grau de provisionamento é determinante: um incidente no data center que causa ao menos um breve tempo de inatividade resulta em perda de receita e usuários frustrados, enquanto um tempo de inatividade prolongado pode trazer danos à lucratividade. Mesmo tempos de resposta um pouco mais altos diminuem a satisfação do usuário e impactam as receitas (DEAN; BARROSO, 2013; KIM et al., 2015). De modo que para alguns serviços pode ser extremamente prejudicial até mesmo pequenas paradas no funcionamento, ou enfrentamento de alguma restrição para atendimento das solicitações.

Na operação dos *data centers* há um gasto fixo de energia para manter os servidores, mesmo com baixo uso da capacidade de processamento. Assim, otimizar a utilização dos recursos pode resultar em economia significativa de custos. Estudos revelam que cerca de 60% dos gastos de energia nos *data centers* são utilizados para manutenção dos equipamentos em funcionamento (KONTORINIS et al., 2012), dos quais 85% desse total são utilizados por discos, memória e processadores (MEISNER et al., 2011). Construir e operar uma grande plataforma de computação é caro e a qualidade de um serviço (QoS, *quality-of-service*), que se refere a capacidade da infraestrutura em assegurar o atendimento das demandas apresentadas, depende do processamento agregado e da capacidade de armazenamento disponíveis, o que aumenta os custos e exige eficiência no uso dos dispendiosos recursos.

Algumas das aplicações críticas em nuvem, aquelas cujos tempos de respostas aos usuários são bastantes restritos, como os representados pelos grandes serviços online de busca usados pelo Google e Microsoft/Bing, as quais exigem uma grande potência de processamento, bem como possuem exigências de tempo na resposta dos serviços aos usuários são exemplos de aplicações críticas. As demandas por resposta rápida com tempos restritos fazem que esse requisito, muitas vezes, faça com que alguns dos *data centers* sejam dedicados exclusivamente a tais aplicações, porém o problema é que o consumo de energia nos *data centers* não é proporcional à carga imposta ao sistema (BARROSO; HÖLZLE, 2007). Além disso, o ambiente deve alocar recursos adicionais para um eventual aumento na demanda (KONTORINIS et al., 2012).

Alguns dos *data centers* e fornecedores dos serviços em nuvem são flexíveis quanto a alocação de recursos de hardware durante a execução de das aplicações alocadas, fazendo uso de alguns mecanismos que busquem redução dos custos de operação. Algumas das alternativas adotadas são o adiamento de uma execução por um intervalo de tempo, degradação mínima da qualidade do serviço ofertado ou transferência da requisição para um outro *data center* em localização diferente que possua maior capacidade para responder por um aumento de demanda (WIERMAN et al., 2014).

Observa-se que o gerenciamento de energia é um aspecto chave para as operações dos servidores e *data centers*, em que o foco está na redução de custos relacionados ao consumo de energia, incluindo o capital investido, as despesas operacionais e os impactos ambientais. Uma das maneiras de reduzir os custos de operação dos *data centers* é tratar o consumo de energia. Algumas técnicas de economia de energia desenvolvidas para dispositivos móveis são candidatos naturais para estes problemas, mas de modo geral, um computador no *data center* é bem diferente de um dispositivo móvel (BARROSO; CLIDARAS; HÖLZLE, 2013; LIU et al., 2014), o que requer estudos específicos e mais aprofundados neste ambiente. Mas algumas dessas possibilidades não podem ser utilizadas em todas as situações, outras alternativas podem e devem ser exploradas.

1.1 OPORTUNIDADE DE PESQUISA

Neste trabalho consideramos *data centers* que hospedam aplicações de Internet, como buscadores, redes sociais e comércio eletrônico. Neste ambiente, certas aplicações que possuem interação com os usuários devem respeitar contratos rigorosos de nível de serviço (SLA, *Service Level Agreement*), especialmente no que se refere à latência das requisições para esses serviços (LO et al., 2014). Isto é, os serviços devem fornecer respostas em tempos satisfatórios (por exemplo, dentro de até 200ms) para permitir uma experiência mais fluída e natural na perspectiva dos usuários.

Um dos serviços da Internet que demandam latência controlada são conhecidos como Online Search (OLS) (DEAN; BARROSO, 2013; BARROSO; CLIDARAS; HÖLZLE, 2013; LI et al., 2014). Tal serviço é utilizado principalmente nos mecanismos de busca como Google e Bing. Espera-se, como usuários, que esses serviços forneçam acesso instantâneo as solicitações dos usuários, enquanto processam grandes volumes de dados. Para isso, os sistemas incluem dezenas de milhares de servidores que consomem dezenas de megawatts de energia (BARROSO; CLIDARAS; HÖLZLE, 2013). Contudo, na maioria do tempo, a capacidade operacional do ambiente não é totalmente utilizada, o que gera um desperdício de processamento que influencia diretamente no uso do investimento aportado e utilizado para manutenção do ambiente.

Como oportunidade de pesquisa existe a possibilidade de explorar períodos de ociosidade dos servidores que executam os serviços críticos, em especial durante baixa demanda. Isso permite melhorar o aproveitamento dos custos operacionais dos *data centers* uma vez que os recursos já alocados estarão sendo melhor utilizados, não necessitando, por exemplo, comprar novos servidores para atender maior demanda. Infelizmente, apesar do potencial de adaptar os sistemas à demanda dos *data centers*, a realidade atual é que há pouca ou nenhuma gestão da ociosidade do uso dos recursos nesses ambientes (WI-

ERMAN et al., 2014). Isso se deve ao fato dos operadores não terem garantias do QoS das aplicações quando os servidores são compartilhados por outras aplicações que usem os períodos ociosos dos servidores.

Uma boa gestão e dimensionamento dos recursos computacionais para as aplicações, buscando coalocar diferentes aplicações, pode trazer uma melhor eficiência no uso dos recursos. Isso traz melhor aproveitamento do capital investido pois ataca o problema da ociosidade do sistema que gera um custo fixo de energia dos servidores. Esta é uma situação e possibilidade pouco explorada, visto a dificuldade de garantias do QoS das aplicações críticas quando coalocadas com outros tipos de aplicação, contundo promissora diante dos desafios impostos no cenário apresentado.

No estudo dos sistemas operacionais, um dos tópicos se refere aos diferentes escalonadores. Os escalonador é parte do sistema operacional responsável por definir o momento em que cada processo poderá ter acesso ao recurso da CPU. Eles utilizam métricas diferentes, o que faz com que a ordem e tempos de execução dos processos seja alterado de acordo com o escalonador que esteja sendo utilizado.

A coalocação das aplicações, com diferentes escalonadores sendo utilizados para cada uma delas de acordo com sua característica, pode permitir um esquema de prioridades em que os requisitos possam ser atendidos para as aplicações servidas, a dificuldade está em estabelecer os corretos parâmetros de utilização, carregamento e agendamentos das mesmas diante da heterogeneidade dos recursos computacionais.

1.2 ESCOPO DO TRABALHO

Este trabalho propõe o uso de meta escalonador que permite coalocar diferentes tipos de processos das aplicações (sensíveis à latência e de melhor esforço) em um mesmo servidor, com uso de diferentes parâmetros, de forma a aumentar sua utilização. Contudo, tal abordagem não é trivial e não pode ser feita de maneira arbitrária, pois as aplicações podem interferir severamente umas nas outras, e assim violar o SLA daquelas que são sensíveis à latência. Assim, processos do tipo melhor esforço podem ser coalocadas desde que não sejam violados os níveis de SLA exigidos pelas aplicações críticas.

Propomos que a coalocação de processos sensíveis à latência e de melhor esforço permita proporcionar maior eficiência dos servidores, garantindo SLA dos processos das aplicação críticas, através do uso de escalonadores específicos para cada tipo de processo.

Para avaliar a hipótese deste trabalho, a proposta apresenta:

- Uma rotina para caracterizar e identificar os processos críticas das aplicações sensíveis à latência;
- Uma heurística para determinar os melhores parâmetros para escalonador de tempo real para os processos sensíveis à latência, ajustando parâmetros do escalonador de acordo com as características de execução;
- Uma avaliação real em um servidor com sistema operacional Linux para quantificar os benefícios de permitir coalocação das aplicações sensíveis à latência e outras aplicações com processos de melhor esforço.

1.3 ORGANIZAÇÃO DO TRABALHO

O Capítulo 2 traz conceitos fundamentais utilizados na pesquisa, incluindo os principais trabalhos relacionados ao problema estudado.

O Capítulo 3 descreve a contribuição principal do trabalho, que é a proposta a ser implementada, descrevendo como o problema será abordado e possível solução a ser adotada para tal fim.

O Capítulo 4 apresenta a metodologia a ser adotada, com a descrição de toda a rotina a ser desenvolvida.

O Capítulo 5 apresenta os resultados encontrados. Ainda neste capítulo os resultados apresentados são discutidos sob a ótica da proposta.

Finalmente, o Capitulo 6 apresenta a conclusão e considerações finais do trabalho

Capítulo

2

CONTEXTO DA PESQUISA

2.1 SERVIÇOS DE INTERNET

Os sistemas de *data centers* hospedam muitos dos modernos serviços de Internet, como a pesquisa online, redes sociais, comércio eletrônico e computação em nuvem. *Data centers* consomem dezenas de megawatts de energia elétrica, que representam milhões de dólares em custos operacionais anuais (BARROSO; CLIDARAS; HÖLZLE, 2013). Do seu poder total, os *data centers* modernos consomem cerca de 10% sobre os gastos gerais de refrigeração e distribuição de energia (a eficiência de uso de energia é 1,12) e cerca de 5% no equipamento de rede, deixando cerca de 85% para os servidores dos quais a memória e o disco ocupam 45% e os processadores consomem 55% (ou seja, 47% do total) (GOOGLE, 2018; BARROSO; CLIDARAS; HÖLZLE, 2013; VAMANAN et al., 2015).

A pesquisa online (OLS), é um componente integral da Pesquisa na Web, mas também de muitos serviços de Internet importantes, como propagandas, comércio eletrônico (por exemplo, Amazon) e redes sociais (por exemplo, Facebook, Twitter). Tais serviços geralmente operam sob orçamentos de tempo de resposta ajustados por acordos de nível de serviço (SLAs) rígidos (por exemplo, 200 ms para responder a uma consulta de Pesquisa na Web) e processam grandes quantidades de dados da Internet (MEISNER et al., 2011).

A OLS tipicamente utiliza uma arquitetura conhecida como partição de agregação, onde diversos servidores são dispostos logicamente como uma estrutura em árvore e a consulta é repartida entre os diversos nós/folhas das árvores, após as consultas realizadas os resultados são agregados no nó raiz que organiza os resultados e apresenta ao usuário (VAMANAN et al., 2015). O ponto chave é que a resposta a uma consulta deve aguardar o resultado de todas as folhas consultadas para agregar os resultados e remetê-la ao solicitante. Consequentemente, o tempo de resposta global de uma consulta é afetado pelas folhas mais lentas. (DEAN; BARROSO, 2013; BARROSO; CLIDARAS; HÖLZLE, 2013).

2.2 SERVIDORES E CARGAS DE TRABALHO

Os servidores em *data centers* são comumente mantidos subutilizados para atender a metas de latência rigorosas. As aplicações críticos quanto à latência são comuns nos *data*

centers atuais e representam novos desafios para os designers de sistemas.

Os serviços críticos utilizam como uma de suas metas de SLA a latência de cauda (*tail latency*), que se refere as latências altas nos tempos de resposta. Na distribuição dos tempos de resposta de uma aplicação, aquelas que ocorrem com pouca frequência e que apresentam um elevado tempo na resposta reapresentam essa situação, elas diferem da maioria absoluta dos tempos de resposta e vão estar acima de um determinado limite. Elas são frequentemente expressas como tempo de resposta do percentil 95 ou 99, o que significa que 95% ou 99% dos tempos de resposta estarão abaixo de determinado valor.

A latência de cauda e não a latência média, determina a qualidade do serviço dessas aplicações. Os servidores com cargas de trabalho críticos em relação à latência são mantidos levemente carregados para atender a esses alvos de latência, com utilizações típicas entre 5% e 25%. As cargas de trabalho do *data center* geralmente podem ser vistas como modelos de solicitação/resposta e exibem um padrão de execução ligado/desligado. O núcleo da CPU consome energia durante o período ligado, quando os pedidos estão sendo processados. Durante o período desligado, onde não há solicitação, a CPU consome energia de maneira ociosa, que é menor que a potência ativa, mas ainda é significativa.

As CPUs modernas fornecem uma sucessão de estados de suspensão para economizar energia durante períodos inativos. Estes estados de suspensão alternam o estado de suspensão e ativação buscando economias de energia, desligando diferentes partes do núcleo e seus caches correntes. Como resultado, o consumo de energia ocioso é determinado pelo estado em que o núcleo entra. O período desligado entre execuções de processamento de pedidos oferece oportunidades para executar o gerenciamento de energia. É importante notar que a duração do período ligado/desligado não é determinista e depende das taxas de chegada e serviço do pedido (CHOU; WONG; BHUYAN, 2016; BARROSO; CLIDARAS; HÖLZLE, 2013). Eventualmente essa não é uma prática muito adequada para aplicações sensíveis, visto que o tempo de alternar os estados de operação do núcleo podem representar sensível alteração nos tempos de resposta e pode influir diretamente nos tempos de resposta da aplicação, proporcionado em algumas situações quebra dos acordos de SLA.

2.3 OTIMIZAÇÃO NO USO DOS RECURSOS

Do custo total de propriedade para os *data centers* modernos, os servidores são a maior fração (50-70%) (BARROSO; HÖLZLE, 2007). Contudo, devido à fraca proporcionalidade energética em servidores, relação entre a energia consumida e processamento realizado, a baixa utilização resulta em um desperdício de consumo de energia que custa milhões de dólares. O desenvolvimento de alternativas de uso mais adequado dos recursos, como por exemplo aplicações de gerenciamento dinâmico de energia nas cargas de trabalho do *data center* são desafiadoras, especialmente quando os requisitos de latência da cauda geralmente caem no nível de sub-milissegundos. A questão fundamental é a aleatoriedade dos picos de demanda devido aos horários de solicitação imprevisíveis e aos horários de pico do serviço. (CHOU; WONG; BHUYAN, 2016).

Nos servidores, o processamento de informações não é proporcional ao gasto demandado, assim os *data centers* operam no pico de eficiência energética quando são totalmente

utilizados, mas têm eficiências muito menores durante períodos de baixa utilização (BARROSO; HÖLZLE, 2007). A utilização média dos sistemas para cargas de trabalho de processamento em lote pode chegar a cerca de 75%, contudo a utilização média de sistemas compartilhados que misturam vários tipos de carga de trabalho, incluindo serviços online, pode estar entre 10% e 50%. Em utilizações médias e baixas, o desperdício de energia, em comparação com um sistema ideal, proporcional à energia é muito relevante. Para melhorar a relação consumo-eficácia dos sistemas, é importante melhorar a eficiência em utilizações baixas e moderadas (BARROSO; CLIDARAS; HÖLZLE, 2013). Além disso, aplicações críticas de latência sofrem tanto com a degradação média do desempenho do servidor quanto com a previsibilidade do tempo de conclusão reduzida quando coalocadas com processos em lote. Isso leva os operadores do sistema a sobreprovisionar recursos para garantir o QoS (*Quality of Service*) para processos críticas de latência, degradando a vazão (*Throughput*) global do sistema (ZHU; EREZ, 2016).

Algumas abordagens são adotadas a fim de reduzir o desperdício dos recursos. Para cargas de trabalho orientadas para transferência, como lote ou analítica, há várias maneiras de melhorar a eficiência em baixa utilização. Por exemplo, as cargas de trabalho podem ser consolidadas em uma fração dos servidores, desligando o resto dos servidores ou atrasando temporariamente os processos para criar períodos suficientemente longos de ociosidade para que os modos de *deep sleep*, desligamento, nos processadores sejam melhor explorados (LO et al., 2014). O dimensionamento dinâmico de tensão/frequência (DVFS), alterna a frequência do processador na execução de acordo com a demanda, pode reduzir o consumo de energia com baixa utilização, mas é difícil de aplicar sem degradar a latência da cauda. O desafio principal é lidar com a variabilidade de curto prazo de aplicações críticos de latência: os pedidos chegam em tempos imprevisíveis e geralmente são explosivos, com grandes quantidades chegando de maneira conjunta, causando grandes demandas de curto prazo e consequentes atrasos no atendimento das solicitação com restrições à latência da cauda (KASTURE et al., 2015).

2.4 ESCALONADORES NO SISTEMA OPERACIONAL LINUX

Neste trabalho investigamos dois tipos de escalonamento no Linux, descritos abaixo.

2.4.1 CFS: Escalonamento justo e de melhor esforço

O escalonador Completely Fair Scheduling (CFS) é o padrão do *kernel* Linux, seu algoritmo de agendamento é baseado em filas ponderadas (WFQ), em que os ciclos de CPU disponíveis são divididos entre *threads* em proporção aos seus pesos. O escalonador define um intervalo de tempo fixo durante o qual cada *thread* no sistema deve ser executado pelo menos uma vez. O peso de uma aplicação é a sua prioridade, processos mais complexos tem pesos mais altos e vice-versa. Quando um processo é executado, ele acumula *vruntime* (tempo de execução do processo dividido pelo seu peso). Uma vez que o *vruntime* de um *thread* excede o tempo limite a ele atribuído, o processo é colocado na CPU de maneira prioritária, mesmo se houver outras *threads* em sua frente para serem executadas (LOZI et al., 2016).

O CFS tenta dividir em parcelas justas os tempos da CPU e tornar os tempos disponíveis para cado processo no sistema. O CFS executa um relógio justo de uma fração da velocidade real do relógio da CPU. A taxa de aumento do relógio é calculada dividindo o tempo pelo número total de processos em espera. O valor resultante é a quantidade de tempo de CPU ao qual cado processo tem direito. À medida que um processo aguarda a CPU, o escalonador rastreia a quantidade de tempo ideal que ele teria usado no processador. Este tempo de espera é usado para classificar os processos para escalonamento e determinar a quantidade de tempo que o processo pode executar antes de ser antecipado. O processo com o tempo de espera mais longo (isto é, com a maior necessidade de CPU) é escolhido pelo escalonador e atribuído à CPU. Quando esse processo está sendo executado, seu tempo de espera diminui, enquanto o tempo de outros processos de espera aumenta. Usando esse princípio, após algum tempo, haverá outro processo com o maior tempo de espera e a processo em execução será suspensa para que a outra seja executada. (PABLA, 2009).

2.4.2 DEADLINE: Escalonamento com restrições temporais

Em sistemas de tempo real (RTOS) devem ser garantidos que as execuções das aplicações sejam atendidas dentro de suas restrições de tempo e gerenciados da forma mais eficiente possível. Uma restrição importante é o prazo (*deadline*) do processo, que é o instante em que a execução do processo deve ser concluída, caso contrário, seus resultados podem ser inúteis. As políticas de escalonamento padrão do Linux, como a `SCHED_NORMAL`, podem não fornecer garantias de que uma aplicação sensível à latência será atendida dentro das suas limitações, principalmente porque nesses escalonadores nenhuma restrição temporal é explicitamente associada a uma processo. A política `SCHED_NORMAL` possui uma prioridade dinâmica que é alterada pelo sistema com base nas características de cada processo, onde a quantidade de CPU que cado processo consome e a latência que ele obterá é determinada principalmente pelo valor *'nice'* - o valor *nice* é um número entre -20 (prioridade mais alta) e 19 (prioridade mais baixa). Eles são executados por curtos períodos e compartilham a CPU entre todos os outros processos em execução com a mesma política, em todos os valores. Ajustar o valor *nice* mudará a maneira como o encadeamento é tratado. Tradicionalmente, o Linux oferece políticas de prioridade fixa simples em tempo real, como `SCHED_FIFO` e `SCHED_RR`, que executam a processo pronta de prioridade máxima. Na política `SCHED_FIFO`, *first in - first out*, processos são agendados de acordo com sua prioridade e a prioridade é definida pela ordem de chegada a fila de execução. O processo de prioridade mais alta é executado indefinidamente, nunca liberando a CPU, exceto para uma processo prioridade ainda mais alta. Na política `SCHED_RR`, *round-robin*, os processos são executados de maneira semelhante ao SCHED_FIFO, exceto que, se mais de um processo tiver a mesma prioridade, eles serão executados por curtos períodos cada e compartilharão a CPU.

O agendamento de prioridade fixa é uma solução efetiva para executar processos em RTOSs onde o número de processos é muito pequeno e o comportamento de cado processo é perfeitamente conhecido. Embora seja possível atribuir um compartilhamento do tempo do processador para um processo (ou um grupo de processos), não há como especificar

que um processo deve executar por determinado tempo durante um intervalo específico. Tais ações são possíveis com escalonadores de tempo real, a exemplo do Earliest Deadline First (EDF) (FAGGIOLI et al., 2009; LELLI et al., 2016)

O Earliest Deadline First (EDF) é um algoritmo de escalonamento que se baseia em um modelo de prioridade dinâmica, que corresponde a uma atribuição dinâmica de prioridades a qual define a ordenação dos processos segundo os seus *"deadlines"* absolutos. Neste modelo, o tempo de computação de cada processo é definido antes do ingresso à CPU, e a fila é organizada de acordo com o menor tempo de execução (BRANDT et al., 2003).

No Linux, a política `SCHED_DEADLINE` fornece uma política de agendamento adequada para aplicativos em tempo real, implementando um algoritmo de reserva de recursos no kernel do Linux. Este algoritmo utiliza três parâmetros para construir a sua política de funcionamento *"runtime"*, *"period"* e *"deadline"*. Cado processo recebe tempos de execução (*"runtime"*), a cada período de tempo (*"period"*) e os tempos de execução estão disponíveis dentro de um prazo (*"deadline"*) partindo do início do período. No processo de implementação dessa política, sempre que um processo entra em execução, o escalonador calcula um prazo de escalonamento com alguma garantia, utilizando o algoritmo CBS. O CBS (Constant Bandwidth Server), implementa reservas de recursos sobre o escalonador EDF (LIU; LAYLAND, 1973). Ele atribui prazos de escalonamento aos processos para que cada processo execute no máximo seu tempo de execução a cada período, evitando qualquer interferência entre diferentes processos, enquanto o EDF verifica o processo com o prazo de agendamento mais antigo e define como o próximo a ser executado (KERNEL.ORG, 2021). Graças a esse recurso, os processos que não cumprem estritamente o modelo de processos em tempo real "tradicional"podem usar essa política.

O algoritmo EDF não é ótimo em múltiplos processadores, no sentido de satisfazer todos os prazos dos processos, quando a demanda total de utilização não excede a capacidade de utilização dos processadores. Contudo, o algoritmo EDF é bastante simples e de fácil de ser utilizado em sistemas de tempo real com características *"soft"*, que é o caso das aplicações em nuvem que toleraram alguns poucos prazos violados ao longo de sua execução, diferentemente de um sistema de tempo real *"hard"* onde nenhum prazo pode ser violado, que não é o nosso caso.

Neste trabalho, a ideia é escalonar as aplicações críticas em nuvem como processos de tempo real e atribuir a cada processo uma reserva Q_i, T_i, o que significa que o processo será permitido executar um "tempo de execução máximo" Q_i a cada período de tempo T_i. O objetivo do escalonador proposto é agendar os processos de tal forma que cada processo receba o tempo de CPU garantido (tempo de execução) Q_i a cada período T_i. Então, se Q_i e T_i são definidos de maneira adequada, as restrições temporais do processo são respeitadas. Além disso, pelo algoritmo CBS, cada processo é restrito a não usar mais do que seu compartilhamento reservado, ou seja, o máximo Q_i a cada T_i unidades de tempo (LELLI et al., 2016).

2.5 TRABALHOS RELACIONADOS

Blagodurov, Zhuravlev e Fedorova (2010) apresentam um estudo de técnicas para mitigação da contenção de recursos compartilhados. Este trabalho descreve um esquema de classificação que aborda a contenção do espaço do cache e a contenção de outros recursos compartilhados, como o controlador de memória, o barramento de memória e o hardware de pré-busca. A partir dos dados, foram propostos algoritmos de escalonamento, de modo que a taxa de falhas (*cache misses*) seja distribuída uniformemente entre os processadores. Apresentaram melhoras na estabilidade dos tempos de execução e redução do tempo de execução do pior caso em até dois fatores para as cargas de trabalho científicas e em até 40% para uma carga de trabalho comerciais em relação ao escalonador padrão do sistema. Não adotam aplicações sensíveis à latência.

O Bubble-Up (MARS et al., 2011) busca prever por meio das características da aplicação a demanda de recursos e como elas afetarão os demais processos do sistema. Sua utilização promoveu um aumento no uso dos recursos entre 50% e 90% mantendo o QoS das aplicações. O Bubble-Flux (YANG et al., 2013), uma melhora do Bubble-Up, usa uma bolha dinâmica para investigar os servidores em tempo real, medindo a demanda instantânea sobre os recursos de hardware compartilhados e prever com precisão como a QoS de um trabalho sensível à latência será afetada por potenciais concorrentes. O Bubble-Flux usa um Online Flux Engine para monitorar continuamente a QoS da aplicação sensível à latência e controlar a execução de processos em lote para se adaptar a chamada, execução e carga dinâmicas mudanças para fornecer QoS satisfatória, selecionando e mapeando os trabalhos sensíveis, privilegiando o uso dos recursos. Diferentemente da nossa pesquisa, nestes trabalhos não foram usados tipos diferentes de escalonamento para as distintas classes de aplicações em *data centers*.

Zhang et al. (2012) apresentam uma solução teórica de controle para o problema de provisionamento de capacidade dinâmica, que minimiza o custo total de energia no *data center* enquanto cumpre o objetivo de desempenho em termos de atraso de escalonamento de processos. O problema é modelado como um problema de controle ótimo de tempo discreto e usa um Modelo de Controle Preditivo (MPC) para encontrar a política de controle ideal. Este trabalho pode ser usado para controlar aplicações sensíveis à latência, porém é agnóstico a outras aplicações presentes no sistema.

A CPI^2 (ZHANG; TUNE; HAGMANN, 2013) usa dados de *cycles-per-instruction* (CPI) obtidos por contadores de desempenho de hardware para identificar problemas, selecionar os processos prováveis e em seguida, optar por acelerá-los para que eles possam retornar ao seu comportamento esperado. Ele aprende automaticamente comportamentos normais e anômalos, agregando dados de vários processos no mesmo trabalho. Com a utilização do sistema, as aplicações críticas que eventualmente possam apresentar problemas, são detectadas e solucionadas, melhorando o desempenho geral do sistema e garantido fluidez de uso.

Leverich e Kozyrakis (2014) no trabalho *Reconciling High Server Utilization and Sub-millisecond Quality-of-Service*, apresentam técnicas que abordam as vulnerabilidades decorrentes da coalocação de aplicações sensíveis a latência que podem levar ao estouro dos prazos de resposta. O trabalho propõe o provisionamento do serviço crítico de latência

de forma consciente por interferência e a substituição do escalonador CFS do Linux por um escalonador que ofereça boas garantias de latência e equidade para cargas de trabalho coalocadas. Algumas das cargas de trabalho quando coalocadas podem aumentar o *throughput* efetivo de um *data center* em até 52%.

O PEGASUS (LO et al., 2014) usa estatísticas de latência das solicitações para ajustar dinamicamente os limites de gerenciamento de energia do servidor de uma maneira fina, executando em cada servidor apenas o suficiente para atender aos objetivos globais de latência do nível de serviço. Em grandes experimentos em *cluster*, o PEGASUS reduz o consumo de energia em até 20%.

Na proposta do Rubik (KASTURE et al., 2015) é apresentada uma abordagem de ajuste das frequências da CPU utilizando a distribuição dos tempos de solicitação dos serviços para prever a menor frequência que não viola o tempo de execução (SLA), dado o número de solicitações atualmente em fila e seus horários de chegada. Lida com a variabilidade através de um modelo de desempenho estatístico. Este modelo permite que o Rubik ajuste frequências em granularidade de sub-milissegundos para economizar energia ao encontrar uma *target tail*. Cada vez que um novo pedido chega ou é atendido, o Rubik faz uma nova previsão e muda a frequência do núcleo em caso de necessidade. Para fazer essas previsões, periodicamente o sistema precompila duas pequenas tabelas de pesquisa, chamadas tabelas de *target tail*. Essas tabelas encapsulam as definições do modelo estatístico e são consultadas para fazer cada previsão. Com a técnica apresentada foi possível reduzir o consumo de energia do núcleo ativo em até 66%. Usando o Rubik foi possível projetar o RubikColoc, um esquema que permite a colocação agressiva de aplicativos críticos de latência e em lotes sem degradar a latência, reduzindo o consumo de energia do *data center* em até 31% enquanto usa 41% menos servidores do que os *data centers* convencionais e segregados.

O Adrenaline (HSU et al., 2015) apresenta uma estratégia de identificação de *querys* com latência critica, a fim de promover um aumento na frequência da CPU responsável pela respectiva consulta, acelerando o seu tempo de conclusão e impedindo o estouro do tempo limite. O Adrenaline identifica as consultas que provavelmente aumentariam a distribuição da cauda e tira proveito do aumento de tensão/frequência da CPU. Foi apresentada uma melhoria de latência da cauda 2.50x para o Memcached e até 3.03x para a Pesquisa da Web com uso do DVFS.

O TimeTrader (VAMANAN et al., 2015) propõe explorar o atraso de latência nas respostas subcríticas que chegam antes do prazo, oferecendo os tempos de processamento restante de algumas consultas a processos que ainda demandam de recurso. Ele remodela a distribuição em todas as cargas, reduzindo os nós subcríticos individuais da arvore de processamento do *data center* sem aumentar os prazos, utiliza da flexibilidade da rede e calcula os tempos médios de processamento de cado processo com base nos anteriores. O TimeTrader economiza entre 15-19% e 41-49% de energia em 90% e 30% de carregamento.

O Heracles (LO et al., 2015) gerencia dinamicamente vários mecanismos de isolamento de hardware e software, como CPU, memória e isolamento de rede, para garantir que o trabalho sensível à latência atenda as restrições, maximizando os recursos atribuídos à processos de melhor esforço. Trata-se de um controlador baseado em respostas que permite a coalocação segura de processos de melhor esforço junto com um serviço de

latência crítica. O Heracles aumenta a eficiência de custos global do sistema substancialmente através do aumento da utilização do mesmo, conseguindo manter a taxa de utilização entre 80% e 90% sem violar os tempos de cada rotina.

Zhu e Erez (2016) expõe as dificuldades encontradas quando processos sensíveis à latência são coalocadas com aplicações de melhor esforço. Partindo do exposto, propõem o Dirigent, uma aplicação que busca prever a demanda de recurso para execução de um processo de modo a controlar os recursos cedidos durante a execução e obter melhor rendimento. O Dirigent explora as variações no tempo de conclusão dos processos que causam ineficiências no sistema compartilhado porque elas ocupam recursos superprovisionados e subutilizam o sistema para alcançar as metas de latência. Sua proposta consiste em prever dinamicamente o tempo de conclusão do processo durante a sua execução e adaptar a frequência e particionamento do cache.

O DynSleep (CHOU; WONG; BHUYAN, 2016) apresenta um esquema de gerenciamento de energia para as cargas de trabalho dos *data centers* através do uso de estados de suspensão por núcleo (*C-states*). Ele reduz de forma dinâmica o processamento em alguns pedidos, criando períodos de inatividade mais longos, que permitem o uso de *C-states* mais profundos para economizar energia. O DynSleep atinge até 65% de economia de energia do núcleo e é 27% melhor do que os esquemas de gerenciamento no *data center*, mantendo as restrições de tempo nos testes realizados com o Memcached.

Capítulo 3

PROPOSTA

A proposta deste trabalho tem por objetivo maximizar a utilização dos servidores, que é um desafio importante para a escalabilidade dos serviços de Internet para um número crescente de usuários. Uma abordagem para melhorar o retorno do investimento é otimizar o nível de utilização dos *data centers*, pois a baixa utilização afeta negativamente a operabilidade e o retorno eficiência dos custos de capital. Para reduzir tais despesas, o objetivo da proposta é garantir o uso efetivo dos recursos disponibilizados no servidor através de escalonamento ciente da classe das aplicações.

3.1 OPORTUNIDADES E DESAFIOS

Diversos estudos da literatura (LO et al., 2014; MARS et al., 2011; LEVERICH; KOZYRAKIS, 2014) demonstram que a utilização dos servidores na maioria dos *data centers* é baixa, variando entre 10% e 50%. Uma razão primordial para a baixa utilização é a popularidade dos serviços de latência crítica, como redes sociais, motores de busca, *software-as-a-service*, mapas *on-line*, *webmail*, tradução automática, compras *on-line* e publicidade. Esses serviços voltados para o usuário, geralmente são dimensionados em milhares de servidores e acessam de forma distribuída os arquivos armazenados na memória RAM ou discos Flash em diversos desses servidores (BARROSO; CLIDARAS; HÖLZLE, 2013).

Embora a carga de requisições nas aplicações em nuvem varie significativamente devido a padrões das rotinas e picos imprevisíveis nos acessos de usuários, é difícil consolidar a carga em um subconjunto de servidores altamente utilizados, pois o estado da aplicação, a demanda de serviço e quantidade de dados armazenados não se encaixam em um pequeno número de servidores. O custo dessa subutilização pode ser significativo. Por exemplo, os servidores Google Websearch geralmente têm uma ociosidade média de 30% ao longo de um período de 24 horas (LO et al., 2014).

A proposta desse trabalho busca melhorar a eficiência dos *data centers* coalocando processos de melhor esforço nos mesmos servidores que executam os processos sensíveis a latência, utilizando diferentes escalonadores, explorando os recursos subutilizados por

tais cargas de trabalho. As aplicações de melhor esforço podem gerar inúmeros subprocessos e consomem recursos significativos. O principal desafio dessa proposta é definir os paramênicos de cada um dos processos, buscando reduzir o nível de interferência entre cargas de trabalho coalocadas, já que os recursos para tais rotinas são compartilhados, como *caches*, memória, interfaces de E/S e links de rede (MARS et al., 2011). A mais importante relação a ser observada é que os processos de latência crítica operam com níveis de SLA baixos, e mesmo pequenas quantidades de interferência podem causar violações desses valores (ZATS et al., 2012).

Quando duas ou mais aplicações são executadas simultaneamente em um servidor, os processos das aplicações competem por recursos compartilhados. Os principais recursos compartilhados no servidor são os núcleos e cache/memória. Não se pode simplesmente particionar os núcleos entre os processos de latência crítica e os processos de melhor esforço usando mecanismos como *cgroups* ou *cpuset* do Linux. Ocorre que os serviços voltados para o usuário, como a pesquisa online na Web, enfrentam um pico de carga, e neste momento elas precisam de todos os núcleos disponíveis para atender às demandas de transferência sem violações da latência definidas no SLA. Da mesma forma, não se pode simplesmente atribuir alta prioridade às processos de latência crítica e confiar no Sistema Operacional que não implementa garantias de tempo real. Algoritmos de escalonamento, como o Linux CFS, possuem vulnerabilidades que levam a violações frequentes do SLA quando os processos de latência crítica são coalocadas com processos de melhor esforço (LO et al., 2015).

De forma a implementar a proposta descrita neste trabalho, há alguns importantes desafios e etapas a serem enfrentados, tais como:

- Propor uma metodologia ou rotina para isolar as *threads* críticas que devem utilizar o escalonador diferente do padrão, neste caso o EDF;
- Observar métricas de carga da aplicação para melhor parametrizar o escalonador, com a correta relação dos tempos (prazos e períodos);
- Verificar limites de carga dos processos de melhor esforço, de forma a poder melhor extrair desempenho nos períodos de baixa capacidade dos processadores.
- Estabelecer uma relação geral de ambas os processos de modo a garantir os tempos de SLA da aplicação de latência crítica e o *throughput* do processo de melhor esforço ambas associadas as demandas de funcionamento básico do Sistema Operacional.

3.2 PROPOSTA DE ESCALONAMENTO MISTO

A proposta ilustrada na Figura 3.1, aqui chamada de EDFCoaloc, baseia-se em um mecanismo que envolve um **meta-escalonador** com auxílio de **monitor de CPU**, um **monitor de latência** para a aplicação sensível à latência e um **identificador de *threads* críticas**.

O **monitor da CPU** será responsável por aferir o nível de uso da CPU de todo o sistema, levando em consideração também as aplicações inerentes ao SO, monitorando

o sistema e acompanhando o comportamento das aplicações. As informações coletadas serão remetidas ao **meta-escalonador**.

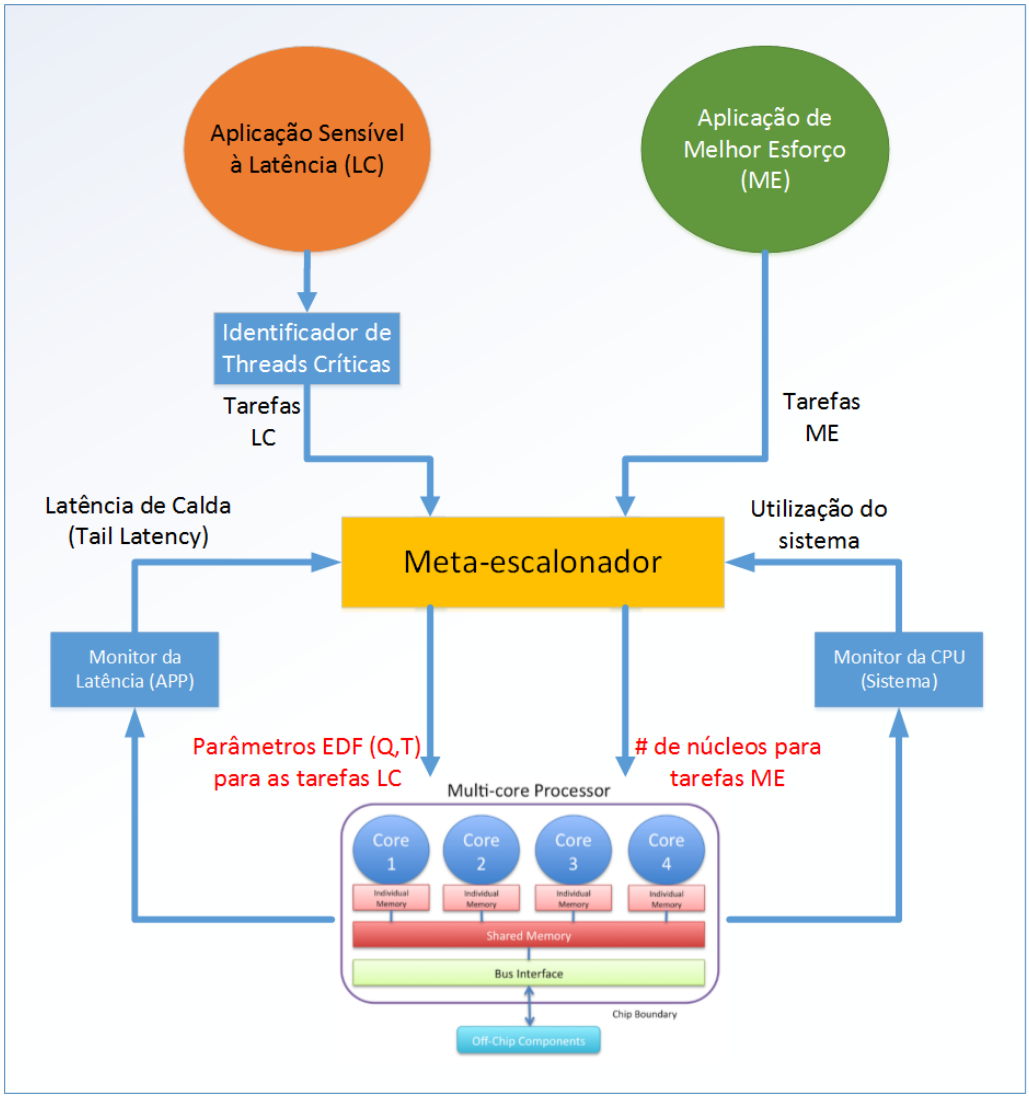

Figura 3.1 Proposta de escalonamento misto para aplicações em nuvem

As políticas de escalonamento padrão do Linux não fornecem garantias temporais exigidas pela aplicação sensível à latência, embora uma prioridade possa ser associada a cado processo ou grupo de processos. Neste sentido, o EDF pode suprir tal demanda de garantias temporais. Contudo, o desafio é usar o EDF para permitir o escalonamento do número de *threads* limitadas à ocupação da totalidade do uso do processador; caso contrário, o sistema não é escalonável com a política de tempo real.

O **identificador de *threads* críticas** é o responsável por uma avaliação prévia da aplicação sensível à latência, construindo um *profile* da aplicação, identificando as *threads* críticas e os valores iniciais de Q_i e T_i e fornecendo as informações para **meta-escalonador**. Esses valores são inerentes ao processo e uma das particularidades do

EDF, Q_i relacionado a quantidade de tempo que o processo será executada e T_i, que representa o tempo que ele pode esperar para ser executada. A identificação das threads do ElasticSearch é realizada por *script* que que executa o utilitário *jstak*, rastreado as *threads* de pesquisa executadas pela aplicação e promovendo as alterações dos valores de Q_i e T_i do escalonador para o ElasticSearch.

A correta definição desses paramentos é crucial para o bom desempenho do sistema. Na proposta apresentada, após inicializada a aplicação pretende-se ajustar o valor de Q_i por meio do **meta-escalonador** baseado em uma heurística que leve em consideração o conjunto de dados coletados sobre as rotinas já finalizadas e coletadas pelo **monitor de latência**, realizado de forma dinâmica e periódica o ajuste do valor. Enquanto que, o valor de T_i será definido conforme o tempo limite estabelecido na SLA da aplicação. Com essa rotina se pretende alcançar um ajuste fino e gradual dos paramentos, estabelecendo valores mais próximos dos ideais e que promovam um melhor desempenho, garantido os tempos de SLA requeridos pela aplicação.

Além disso, o **meta-escalonador**, com base nas informações coletadas pelo **identificador de *threads* críticas**, definirá quais as *threads* serão carregadas no EDF. O **meta-escalonador** também definirá o quanto de recurso, CPU, está disponível para uso pela aplicação de melhor esforço com as informações obtidas pelo **monitor da CPU**. O mesmo alternará a quantidade de CPU disponíveis para a aplicação de melhor esforço conforme informações recebidas. Em caso de disponibilidade maior de recurso, a quantidade de recurso será aumentada, em caso de demanda maior por parte da aplicação sensível à latência a demanda será reduzido.

Capítulo 4

METODOLOGIA

A pesquisa desenvolvida neste trabalho é de natureza experimental com abordagem quantitativa. A pesquisa de natureza experimental concentra-se em determinar um objeto de estudo, selecionar as variáveis que podem influenciá-lo e buscar formas de controle e observação dessas influencias (APPOLINÁRIO, 2012; BARROS; LEHFELD, 2000). A pesquisa quantitativa utiliza de técnicas para mensurar a medida de algo, possuem varáveis objetivas e tem ênfase na comparação de resultados e no uso de técnicas estatísticas (**??**).

Nesta seção introduzimos a metodologia da proposta, incluindo o ambiente experimental usado para validação da proposta apresentada neste trabalho.

4.1 PLATAFORMA GOOGLE CLOUD

O Google Cloud Platform (GCP) é hoje um dos mais importantes e crescentes plataformas do mercado de nuvem. O GCP consiste em um conjunto de ativos físicos, como computadores e unidades de disco rígido, e recursos virtuais, como máquinas virtuais (VMs), localizados nos centros de dados do Google em todo o mundo (GOOGLE, 2018). Ele fornece aos desenvolvedores vários produtos para criar uma variedade de programas, desde sites simples até aplicativos complexos distribuídos em todo o mundo. O GCP oferece serviços de hospedagem na mesma infraestrutura de suporte que o Google usa internamente para produtos de usuários finais, como a Pesquisa do Google e o YouTube (CHALLITA et al., 2018).

Neste trabalho, foi utilizado na infraestrutura do GCP duas instâncias de máquinas virtuais (MV), uma para trabalhar como cliente nas consultas usando a aplicação Faban e outra atuando como servidor com uma aplicação de busca na Web: ElasticSearch. A máquina cliente foi configurada para uso de duas CPU Intel Xeon E5 v3 2,3 GHz (Haswell), 7,5GB de RAM e um disco de 10GB SSD. O servidor faz uso de seis CPU Intel Xeon E5 v3 2,3 GHz (Haswell), com 25GB de RAM e 65GB de disco SSD. A tabela 4.1 traz um resumo das configurações das Máquinas Virtuais

Tabela 4.1 Configurações do GCP

Característica	VM	
	Cliente	Servidor
Processador	2 x Intel Xeon E5 v3 (Haswell)	6x Intel Xeon E5 v3 (Haswell)
Frequência	2,3 GHz	2,3 GHz
Cache	20 MB SmartCache	20 MB SmartCache
Memória	7,5 GB DDR4 1600	25 GB DDR4 1600
Disco	25 GB	65GB

4.2 APLICAÇÃO SENSÍVEL À LATÊNCIA: ELASTICSEARCH

O Elasticsearch é um mecanismo de pesquisa e análise de texto completo de código aberto altamente escalável. Permite armazenar, pesquisar e analisar grandes volumes de dados rapidamente. Geralmente, é usado como tecnologia subjacente que alimenta aplicativos que possuem recursos e requisitos de pesquisa complexos (ELASTIC, 2018). O Elasticsearch é usado em muitos casos de uso, como armazenamento analítico, auto-correção, corretor ortográfico, mecanismo de alerta e armazenamento de documentos de uso geral. A pesquisa de texto completo é uma delas, trata-se de um mecanismo de pesquisa robusto que fornece uma pesquisa rápida. É um método de pesquisa em que cada palavra da solicitação é pesquisada é comparada com palavras de documentos ou banco de dados. Ele pesquisa em campos de texto completo para localizar o documento e retornar primeiro o resultado mais relevante (KALYANI; MEHTA, 2017). Para tal situação foi criada uma base indexada para consulta com uma cópia do Wikipédia utilizado a ferramenta stream2es[1] que após todo o processamento gerou um arquivo de 25GB. Este arquivo foi a utilizado como base para realização de todos os testes.

4.3 APLICAÇÕES EM LOTE: PARSEC

O pacote de benchmark PARSEC (Princeton Application Repository for Shared-Memory Computers) é projetado para fornecer programas para o estudo de computadores microprocessados (CMPs). Ele se concentra em aplicativos emergentes de desktops/servidores. O PARSEC inclui aplicativos de reconhecimento, mineração e síntese (RMS), bem como aplicativos de sistemas que imitam programas comerciais de vários segmentos em larga escala (BIENIA et al., 2008). O PARSEC foi utilizado de forma coalocada com o ElasticSearch, gerando uma rotina de processamento de melhor esforço com objetivo de verificar o comportamento da proposta apresentada. Foram utilizadas as cargas *blackscholes*, *canneal*, *ferret*, *fluidanimate*, *freqmine* e *streamcluster* com variações no número de *threads* a serem carregadas.

Blackscholes é uma aplicação que resolve equações diferenciais para calcular os preços de um determinado número de opções financeiras. A entrada nativa tem 10.000.000

[1] https://github.com/elastic/stream2es

opções. É uma aplicação de granularidade alta, necessitando de maior tempo de processador para execução da rotina. Além disso, opera melhor com paralelismo em sua execução. Ferret é uma aplicação que faz busca de similaridades por imagens, a entrada da aplicação é um banco de dados de imagens, a configuração do número de imagens a serem encontradas e uma lista de imagens de consulta. O Ferret trabalha com pipeline de seis estágios. Como cada uma das etapas depende da anterior, um menor tempo entre as etapas tende a proporcionar um melhor desempenho. A aplicação Canneal busca minimizar o custo de roteamento de um projeto de chip com Simulated Annealing (SA) utilizando reconhecimento de cache. Ela utiliza uma rotina de troca dos elementos netlist para se aproximar da solução ótima. A entrada é um dado elemento netlist, uma definição do número de swaps por cada passo e o número de passos para toda a execução. Os valores para o tamanho nativo são 2.500.000, 15.000 e 6.000, respectivamente. O componente linear desta aplicação é o número de trocas e o número de passos.

Streamcluster é uma aplicação que executa cálculos para solução de *clustering* online. Para um fluxo de pontos de entrada, ele encontra um número predeterminado de medianas para que cada ponto seja atribuído ao seu centro mais próximo. O programa passa a maior parte do tempo avaliando o ganho da abertura de um novo centro. Esta operação usa um esquema de paralelização que emprega particionamento estático de pontos de dados. A aplicação Fluidanimate usa o método Smoothed Particle Hydrodynamics para simular a física subjacente do movimento do fluido para animação em tempo real. As entradas são o número de partículas e o número de quadros. A entrada nativa tem 500.000 partículas e 500 quadros. O número de partículas tem uma relação não linear com o tamanho do problema, e o número de quadros tem uma relação linear com o tamanho do problema .Freqmine é uma aplicação de mineração de dados. Ele identifica padrões que ocorrem com frequência em um banco de dados transacional. A entrada é uma lista de transações, com a definição de um suporte mínimo (BIENIA et al., 2008). A tabela 4.2 apresenta uma resumo de cada uma dos *workloads* utilizados.

4.4 GERADOR DE CARGA: FABAN

O Faban é um gerador de carga de trabalho de desempenho que é executado em uma máquina cliente e gera uma carga na máquina do servidor simulando um grande número de usuários simultâneos acessando a aplicação (IRMA, 2017; FABAN, 2018).

Neste estudo, o mesmo foi utilizado como gerador de consultas ao servidor executando a aplicação ElasticSearch, com tempo de RamUp (tempo de carregamento e estabilização da aplicação) fixado em um minuto, o tempo de teste foi limitado a 5 minutos e tempo de RamDown fixado em um minuto.

O Faban gera um relatório simplificado em XML de cada experimento, no qual é possível avaliar os tempos médio, máximo e mínimo de cada consulta, número total de consultas realizadas e respondidas e os valores de 90° percentil e 99° percentil das consultas realizadas, que representa, respectivamente, o tempo que levou para que 90% e 99% das consultas fossem concluídas abaixo de uma valor pré-definido, além de outras informações complementares. O Faban também gera um relatório detalhado em texto, com todas as informações sobre a execução, o que ajuda a traçar um perfil do comportamento

Tabela 4.2 Workloads PARSEC

Workload	Tipo de Aplicação	Descrição	Paralelização	
			Modelo	Granularidade
blackscholes	Financial Analysis	Calcula analiticamente os preços de uma carteira de ações com a equação diferencial parcial de Black-Scholes (PDE).	data-parallel	coarse
canneal	Engineering	Simula o custo de roteamento para designer de chips	unstructured	fine
ferret	Similarity Search	Servidor de pesquisa por similaridade de conteúdo	pipeline	medium
fluidanimate	Animation	Dinâmica de fluidos para fins de animação com o método de hidrodinâmica de partículas suavizadas (SPH)	data-parallel	fine
freqmine	Data Mining	Mineração de um conjunto de itens	data-parallel	medium
streamcluster	Data Mining	Clustering online de um fluxo de entrada	data-parallel	medium

da aplicação e produzir outras informações quanto a execução.

4.5 FERRAMENTA PARA ALTERAR ESCALONAMENTO: SCHEDTOOL

O schedtool[2] é uma ferramenta disponível para o Linux que tem por objetivo alterar ou consultar todas as políticas de agendamento de CPU no Linux, em um comando prático. Assim, o schedtool é a interface para acesso e configuração do escalonador do Linux. Ela permite definir diversos parâmetros e alterar as políticas de escalonamento. O algoritmo implementado é um CBS (Constant Bandwith Server) global, implementado como um algoritmo particionado com migração (LELLI, 2014) . Neste trabalho foi utilizado para definir os parâmetros de execução ElasticSearch com EDF.

4.6 MÉTRICAS DE AVALIAÇÃO

A avaliação da proposta será realizada levando em consideração o comportamento das aplicações em diversos cenários:

1. Aplicação sensível à latência isolada utilizando o escalonado padrão - CFS;

2. Aplicação sensível à latência isolada utilizando o escalonador EDF;

3. Aplicação sensível à latência coalocada com aplicação de melhor esforço com ambas utilizando o escalonador CFS;

[2]https://github.com/jlelli/schedtool-dl

4. Aplicação sensível à latência utilizando o escalonador EDF coalocado com aplicação de melhor esforço utilizando o escalonador CFS.

Os parâmetros a serem considerados são o nível de carga de ambas as aplicações e a quantidade de núcleos disponíveis para cada uma delas. Como métricas de desempenho e comparação serão avaliados:

1. Comportamento das requisições nos diferentes cenários apresentados e com os parâmetros especificados;

2. Vazão das aplicações: numero de requisições atendidas na aplicação crítica e número de aplicações em lote finalizadas por intervalo de tempo;

3. Nível de utilização das CPUs do sistema;

Capítulo 5

AVALIAÇÃO EXPERIMENTAL

Neste capítulo, apresentamos a descrição da execução do experimento e detalhamento do funcionamento da proposta.

5.1 CARACTERIZAÇÃO DO EXPERIMENTO

Para aplicação secundária (melhor esforço) foram utilizados seis aplicativos do Suíte de Benchmark PARSEC (BIENIA et al., 2008). O número de processadores a serem utilizados pelo aplicativo foi configurado diretamente na chamada da aplicação por meio dos parâmetros de execução. Para essa finalidade foi construído um *script* Python para automatizar as chamadas sucessivas dentro do tempo determinado para a execução.

O Faban é uma aplicação que simula as requisições geradas por aplicações cliente, ele é capaz de gerar uma carga de consultas para o servidor ElasticSearch possibilitando o ajuste no número de clientes simulados. Com o Faban é possível definir os tempos de carga da aplicação, e ao fim do tempo programado, um resumo do desempenho das consultas é apresentado, na forma de distribuição das latências das requisições.

Para definição do *baseline* de comparação, a aplicação (ElasticSearch) e o cliente (Faban) foram executadas utilizando o escalonador padrão do sistema sem nenhum tipo de alteração. Os valores aqui coletados foram utilizados como padrão para comparação com os valores obtidos com as variações e proposta descrita.

Os dados de uso das CPUs foram coletados e por meio de *script* construído para esse fim. Definidos os limites de atendimento da aplicação, foram especificados três níveis de ocupação da CPU para avaliação dos dados. Baixa utilização com 25% de ocupação, média utilização com 50% de ocupação e alta ocupação com 75% de ocupação.

Posteriormente o processo foi repetido utilizando o meta-escalonador proposto, descrito na outra sessão. E por fim a aplicação foi executada de forma coalocada com aplicações de características distintas utilizando o benchmark PARSEC, já descrito em capítulo anterior (4).

As aplicações do benchmark PARSEC foram usadas para verificar o comportamento do ElasticSearch quando coalocado com aplicações de melhor esforço. No PARSEC é

possível definir a quantidade de *threads* a ser carregada para as aplicações, e o tipo de carga de trabalho (tamanho da entrada) a ser utilizada. Ao fim da execução um resumo do processamento (tempo de execução) é apresentado.

Para obter valores referenciais de comparação da aplicação quando coalocadas ou não, as aplicações foram executadas isoladamente e seus valores de desempenho aferidos.

5.2 O META-ESCALONADOR

Para atingir o objetivo desse trabalho foi proposto o uso de um meta-escalonador repensável por monitorar e promover ajustes na execução das aplicações com alterações nos paramentos dos escalonadores do sistema.

O Meta-escalonador foi construído utilizando a linguagem Python com uso de diversas bibliotecas. O Algoritmo 1 apresenta as principais funções da aplicação. Os valores de *runtime* e de *deadline* para os processos do ElasticSearch utilizados escalonador EDF são dinamicamente ajustados. Quando o valor do 95º percentil das consultas apresenta-se maior que o valor definido na SLA o valor de *runtime* é defindo como o valor da SLA, buscando ampliar o uso da CPU pelo ElasticSearch e o valor de *deadline* é ajustado para a razão entre o valor obtido inicialmente e o valor definido na SLA, ampliando o tempo da aplicação no CPU, buscando reduzir o seu tempo de execução e fazendo aplicação ampliar o tempo de CPU. Por outro lado, quando o tempo calculado é menor que o definido na SLA, o valor do *runtime* é reduzido ampliando o uso para a aplicação em lote e o valor de deadline é definido como tempo da SLA, buscando garantir o tempo de execução máximo desejado para cada rotina, caso a aplicação termine antes do prazo a CPU é liberada para outra demanda.

O meta-escalonador opera monitorando o uso da CPU, com verificação constante da latência alvo definida. Como parte do processo ele faz ajustes periódicos nos valores de funcionamento do escalonador EDF. As alterações são realizadas com uso da ferramenta schedtool, da qual é possível modificar a execução do ElasticSearch de forma a mapear alguns processos para escalonadores CFS e EDF e também estabelecendo diferentes parâmetros de tempo real para Q_i e T_i (no caso do EDF). Os valores são ajustados utilizando frações do tempo estabelecidos pelo meta-escalonador, dessa forma é possível ajustar os tempos enquanto as aplicações são executadas.

Para operação do experimento foi criado um script responsável por inicializar cada uma das aplicações. As rotinas de execução são processadas em uma ordem definida a fim de garantir a correta padronização do experimento.

São envolvidos no processo duas máquinas virtuais na plataforma Google Cloud, já descritas na metodologia. Uma máquina remota é responsável por efetuar acesso ao cliente e ao servidor e inicializar as aplicações.

Inicialmente a VM servidora é acessada com inicialização do meta-escalonador. Esse inicializa o ElasticSearch e identifica os PIDs da aplicação alterando os parâmetros de operação do escalonador EDF com base nas configurações iniciais. Em seguida, o cliente Faban é carregado na máquina cliente para realização das consultas. O número de clientes do Faban é definido antes do processo ser iniciado. Também são configurados o tempo de RamUp e RamDown bem como o tempo de execução e coleta dos dados.

Algorithm 1 EDFCoaloc

$TailLatency = 200$ ▷ Define a latência alvo
$LogElastic =$ Caminho ▷ Especifica o caminho para o arquivo de log

function SCHEDULE($ThreadsCriticas, DeadLine, RunTime$) ▷ Função responsável por definir o *deadline* e *runtime* do escalonador EDF
    for $Thread$ in $ThreadsCriticas$ **do**
        SetDeadlineEDF ($Thread$,$DeadLine, RunTime$)
    end for
end function

function UPDATEDEADLINE($Percentil, TailLatency, RunTime, DeadLine$) ▷ Atualiza os valores de *runtime* e *deadline*
    if $Percentil \leq TailLatency$ **then**
        $RunTime = \frac{Percentil}{TailLatency} \times \frac{Percentil+TailLatency}{2}$
        $DeadLine = TailLatency$
    else if $Percentil \geq TailLatency$ **then**
        $RunTime = TailLatency$
        $DeadLine = TailLatency * P\_tail$
    end if
    return $DeadLine, RunTime$
end function

function CALCPERCENTIL($LogElastic$) ▷ Calcula o 95º *percentil* dos tempos das consultas realizadas
    AbreArquivo($LogElastic$)
    ExtrairTempos ()
    CalculaPercentil ()
    return $Percentil$
end function

function MAIN( )
    while *true* **do**
        $Percentil =$ CalcPercentil ($LogElastic$)
        $DeadLine, RunTime =$ UpdateDeadline ($Percentil, TailLatency, RunTime, DeadLine$)
        Schedule ($ThreadsCriticas, DeadLine, RunTime$)
        Aguarde (ΔS)
    end while
end function

Conforme as aplicações executam, o meta-escolador recolhe os dados e executa os cálculos ajustando os tempos de execução, proporcionando maior ou menor uso da CPU pelo ElasticSerch de modo a liberar parte do recurso para a aplicação coalocada ofertada pelo benchmark PARSEC.

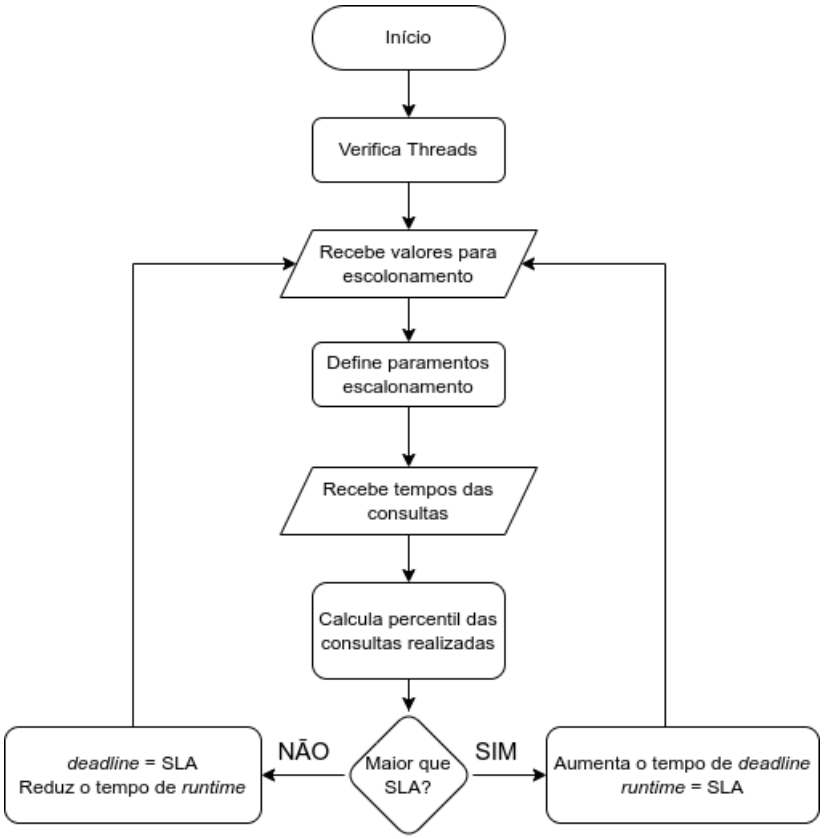

Figura 5.1 Fluxograma do funcionamento da função de ajustes dos parâmetros do escalonador.

Além disso o meta-escalonador faz a gravação de informações sobre o uso da CPU, valores das latências obtidas das consultas realizadas e paramentos de funcionamento do escalonador.

Conforme descrito em sessão anterior, diversas combinações de execução das aplicações foram utilizadas para avaliação da proposta, com alteração da aplicação PARSEC coalocada bem como do número de *threads* utilizadas por essa aplicação, variando entre duas, quatro ou seis.

O funcionamento da principal função da proposta consiste em acompanhar a execução das rotinas, monitorando os tempos das consultas realizadas, calculando, de tempos em tempos, os valores de SLA das consultas realizadas naquele intervalo. A política proposta utiliza a latência calculada para promover ajuste nos valores de *runtime* e *deadline* do escalonador EDF.

Figura 5.1 apresenta um fluxograma ilustrando o funcionamento da função de ajustes dos parâmetros do escalonador utilizado para a processo de crítica. A avaliação da latência ocorre a cada 5 segundos, o calculo é realizado extraindo informações do arquivo de

log da aplicação ElasticSearch por meio de uma função construída especificamente para essa finalidade. A verificação também considera os valores atuais antes de realizar a alteração, desse modo busca-se uma melhor eficiência no uso dos recursos, garantido a SLA e liberando recuso para a aplicação coalocada.

No experimento foi possível avaliar que o tempo destinado a este intervalo entre avaliações foi suficiente para obtenção de bons resultados. Os ajustes são graduais e conservadores de modo que o uso do recurso possa ser garantido para todas as aplicações e não comprometa a sua operação.

5.3 RESULTADOS

Nesta seção, são apresentados os resultados obtidos a partir da execução do experimento com as combinações do sistema proposto. As combinações são formadas com variação na quantidade de *threads* carregadas na aplicação do PARSEC, variando entre 2, 4 ou 6 *cores*. Combinadas com a variação do nível de carga da aplicação ElasticSearch definidos em 25%, 50% e 75% de uso do processador, conforme definido em seção anterior.

A seguir são apresentados os gráficos demonstrando os resultados, separados de acordo com cada aplicação PARSEC com as suas variações no número de *threads* coalocado com o ElasticSearch nos seus diferentes níveis de carga.

Na área superior dos gráficos são apresentadas as distribuições cumulativas dos tempos de respostas das consultas realizadas no ElasticSearch. Neles são destacadas duas linhas, uma vermelha horizontal que apresenta o tempo limite para para atendimento da SLA da aplicação com 95º percentil, onde o resultado será satisfatório caso a linha que representa a distribuição dos dados corte a linha vermelha antes de passar pela linha pontilhada em preto na vertical, que representa os 200ms de SLA da aplicação. Essa parte demonstra se aplicação viola as restrições temporais de SLA em algum cenário. As linhas contínuas representam os valores obtidos com uso do EDFCoaloc, modelo proposto neste trabalho. As linhas pontilhadas representam os valores obtidos com uso do escalonador padrão do sistema, o CFS. Ambas as linhas tem variações de cores que rementem ao número de ***threads*** em que a aplicação PARSEC foi carregada, ou a execução isolada do ElasticSearch.

Na área inferior do gráfico são apresentadas as quantidades de requisições atendidas e quanto de CPU foi utilizado em média durante a execução. Essa região do gráfico ilustra se o volume de requisições atendidas foi prejudicada com adesão de um modelo diferente de escalonamento. As barras hachuradas apresentam os valores obtidos quanto utilizado o EDFCoaloc, enquanto que o as barras sem hachuras apresentam os valores obtidos com uso do escalonado padrão CFS. Assim como na área superior, as barras apresentam variações nas cores, que rementem o número de ***threads*** em que a aplicação PARSEC foi carregada, ou a execução isolada do ElasticSearch.

A Figura 5.2 apresenta o resultado obtido da coalocação do ElasticSearch com a aplicação Blackscholes. A aplicação resolve equações diferenciais para calcular os preços de um determinado número de opções financeiras.

Observando os gráficos superiores (distribuição cumulativa) na Figura 5.2, percebe-se que o EDFCoaloc obteve melhor resultado, considerando o aumento da demanda por

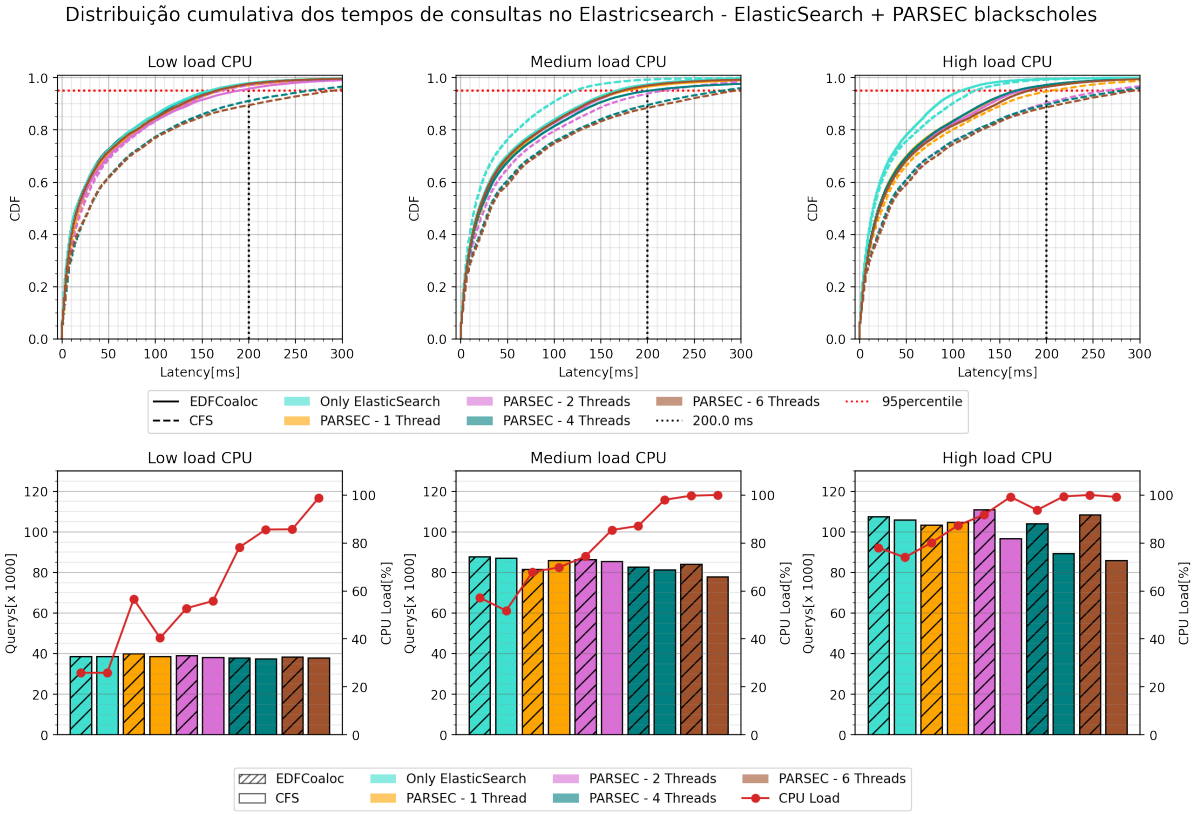

Figura 5.2 Distribuição cumulativa dos tempos de consulta com uso do ElasticSearch coalocado com a aplicação Blackscholes do benchmark PARSEC.

recurso de ambas as aplicações coalocadas e tanto pelo aumento do volume de consultas pelo ElasticSearch quanto pelo número de *threads* carregadas no Blasckscholes. Com a CPU com alta demanda, o algoritmo padrão do Linux, CFS, conseguiu permanecer dentro dos requisitos de SLA com o carregamento da aplicação secundária com apenas uma *thread*, enquanto nos demais cenários houve violação das condições de qualidade de serviço. Com a CPU em baixa e média demanda o número de consultas atendidas foram bem próximas com o EDFCoaloc e CFS. Quando a demanda por CPU aumentou, impulsionado pelo aumento das consultas do ElasticSearch, o volume de consultas atendidas foi menor quando utilizado o escalonador CFS (Figura 5.2).

Neste *workload* os melhores resultados com uso do EDFCoaloc ocorreu quando a demanda por CPU foi maior, com uso em *high load* pelo ElasticSearch. Os ganhos de performance, fazendo uma relação entre os valores de 95º percentil obtidos com uso do CFS e do EDFCoaloc, foram de 20% com uso de uma *thread*, 53% com uso de duas *threads*, 68% com uso de quatro *threads* e 62% com uso de 6 *threads*. Em todas as situações com SLA cumprido pelo EDFCoaloc para o ElasticSearch. Outra consideração importante é que com *low load*, a *tail latency* ficou mais próxima da SLA utilizando o EDFCoaloc, sendo este um comportamento esperado, pois quando a aplicação sensível à latência tem baixa demanda, a aplicação co-alocada tende a ocupar por mais espaço da CPU. Por ser

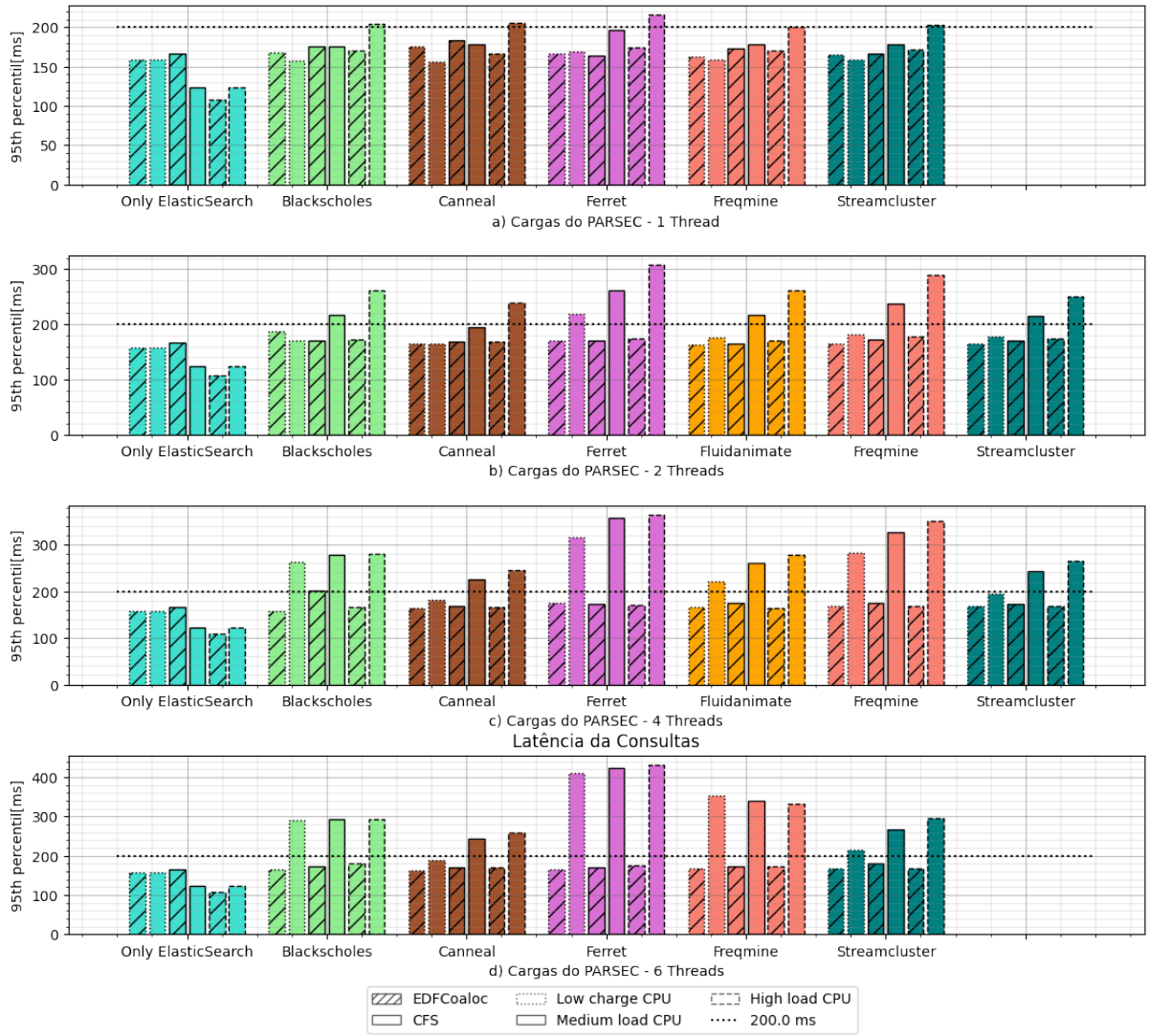

Figura 5.3 Resumo dos experimentos - 95º percentil por numero de *threads*.

uma aplicação de granularidade alta, o Blackscholes é favorecido quando um maior tempo de CPU é disponibilizado, conseguindo concluir mais rotinas do processo sem necessidade se suspensão. A diferença de maiores ganhos com o processador em maior uso demonstra a eficiência do EDFCoaloc em garantir os tempos exigidos pela SLA e dar segurança na execução de ambas as aplicações coalocadas.

Observando a Figura 5.3 é possível verificar que com aumento do *load* do ElasticSearch associado ao aumento do numero de threads pelo PARSEC com aplicação Blackscholes faz com que as violações de SLA do ElasticSearch ocorram com o escalonador CFS. Quando os cenários são observados com uso do EDFCoaloc não ocorreu violação da SLA em nenhuma situação com a coalocação das duas aplicações.

Figura 5.4 apresenta o resultado obtido da coalocação do ElasticSearch com a aplicação Ferret. Essa carga de trabalho é uma aplicação de faz busca de similaridades por imagens.

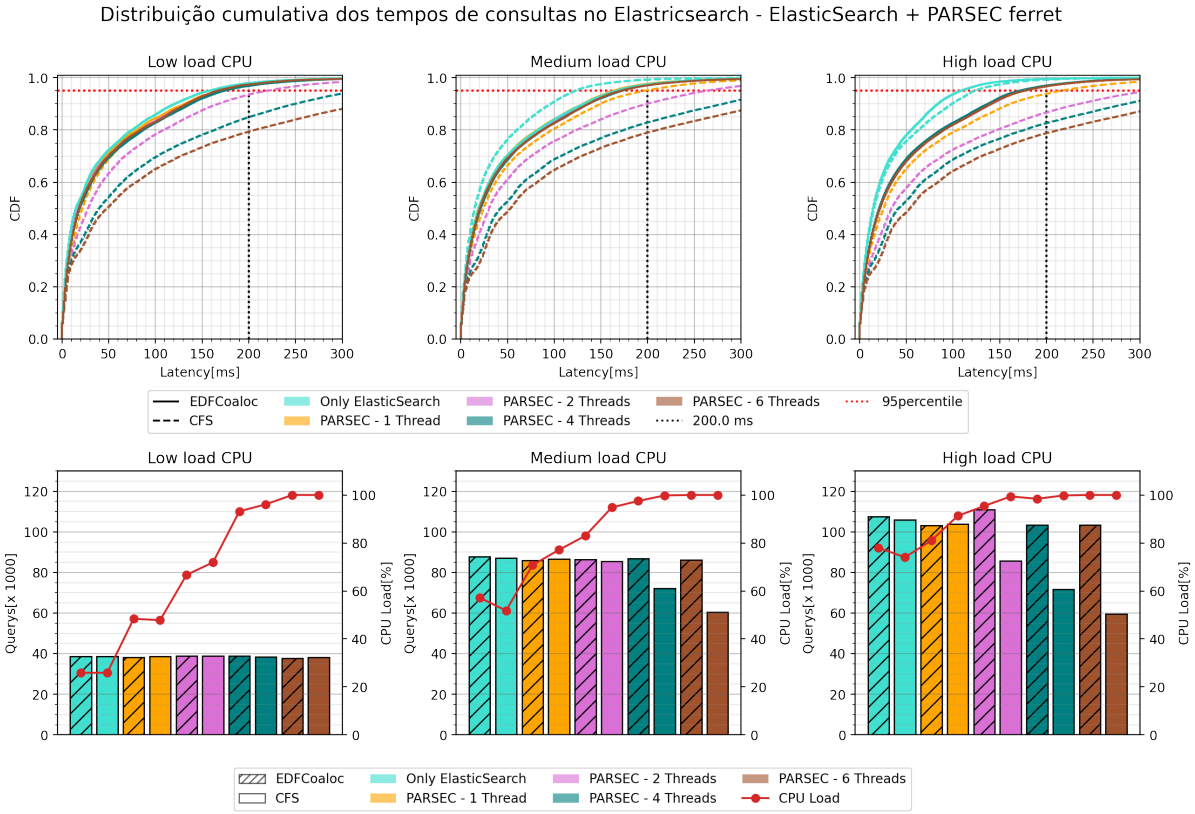

Figura 5.4 Distribuição cumulativa dos tempos de consulta com uso do ElasticSearch coalocado com a aplicação Ferret do benchmark PARSEC.

Pelo gráfico observa-se que em todas as combinações o escalonamento utilizando o EDFCoaloc obtiveram sucesso, mesmo com o aumento da demanda de CPU com aumento das consultas ao ElasticSearch quanto pelo aumento do número de *threads* na aplicação Ferret. Em todos os cenários o EDFCoaloc obteve melhor desempenho, destacando que quanto maior a demanda pelo uso da CPU pelas aplicações, melhor foi o resultado foi obtido.

Quando a coalocação ocorreu com um maior numero de threads pelo Ferret, os ganhos foram sempre próximos dos 150%, isso comparando o escolanador padrão, EDF, com o EDFCoaloc. Em *low load*, os ganhos também foram consideráveis, chegando a de 147%, com *medium load* os ganhos foram de 148% e com *high load* os ganhos foram de 146%. O ferret é uma aplicação que funciona com modelo de execução em pipeline, com execução de atividades sequenciais, gerando alta demanda de E/S e memória, o que não comprometeu o desempenho do ElasticSearch, tendo o EDFCoaloc superado o desempenho do CFS em todos os cenários de de coalocação experimentados.

Na Figura 5.3 observa-se que com a coalocação do ElasticSearch e aplicação Ferret do PARSEC o uso do escalonador CFS faz com que ocorresse a vailação da SLA em praticamente todos os cenários, com exceção do uso de uma thread para a aplicação Ferret com baixo e médio load da CPU para o ElasticSearch.

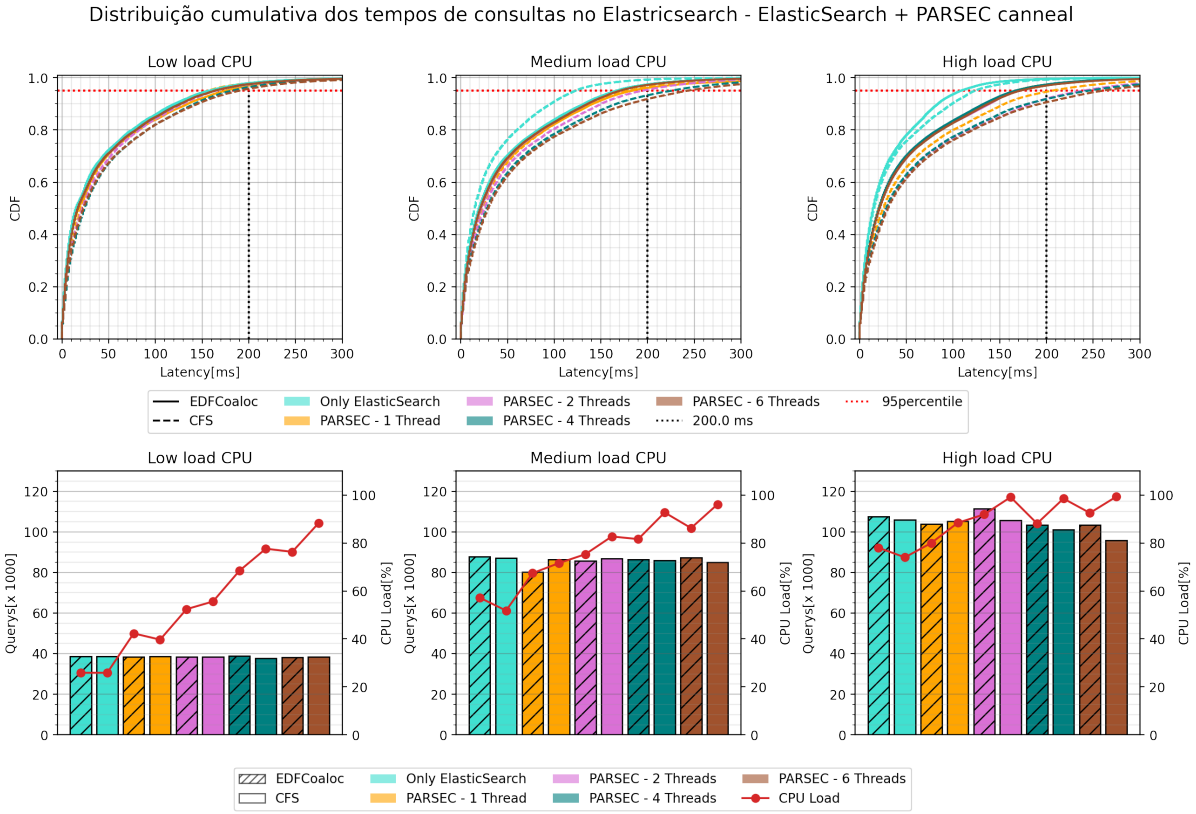

Figura 5.5 Distribuição cumulativa dos tempos de consulta com uso do ElasticSearch coalocado com a aplicação Canneal do benchmark PARSEC.

Figura 5.5 apresenta o resultado obtido da coalocação do ElasticSearch com a aplicação Canneal. A aplicação busca minimizar o custo de roteamento para designer de chips. Os ganhos mais consideráveis ocorreram quando a exigência por recursos de CPU do ElasticSearch foi maior e com 2, 4 e 6 *threads* no Canneal. O menor ganho foi 0,5% com o CPU em *low load* para o ElasticSearch e com Canneal carregada em apenas duas *threads*. A maior diferença nos valores da *tail latency* ocorreram quando 6 *threads* foram carregadas para a aplicação Canneal com demanda alta de CPU para o ElasticSearch.

Na Figura 5.3 verifica-se que essa foi a coalocação melhor se comportou em relação as demais quando coalocada com uso do CFS. Mesmo nessa situação os ganhos foram importantes para garantir a SLA do ElasticSearch com uso do EDFCoaloc em comparação com o escalonador padrão. Em todos os cenários ele manteve a aplicação dentro dos limites propostos, diferente do escalonador padrão em que ocorreram violações em alguns cenários.

A Figura 5.6 apresenta o resultado obtido da coalocação do ElasticSearch com a aplicação Streamcluster. Essa aplicação executa cálculos para solução de clustering online. Os ganhos com uso do EDFCoaloc com essa aplicação proporcionou uma melhora de até 76% quando a demanda de CPU para o ElasticSearch foi alta e foram utilizadas 6 threads para o Streamcluster. O menor ganho foi de 7% com *medium load* de CPU para o

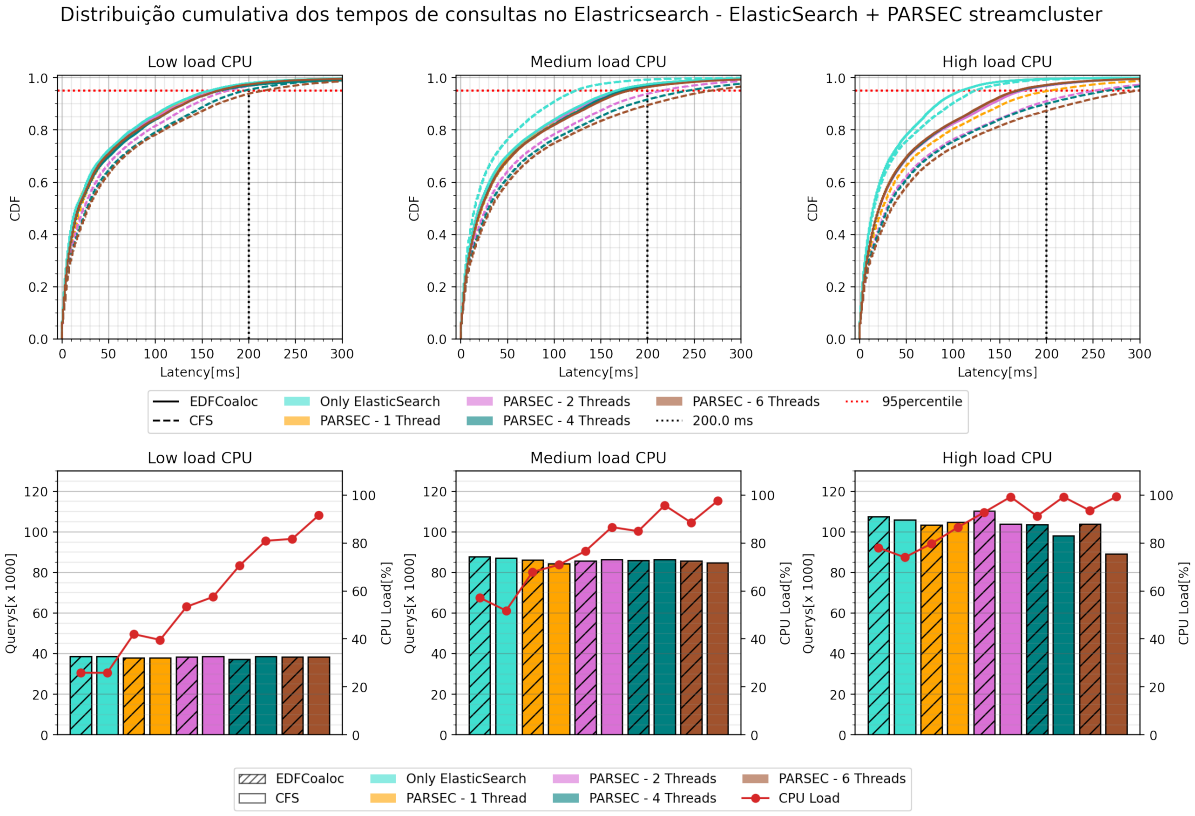

Figura 5.6 Distribuição cumulativa dos tempos de consulta com uso do ElasticSearch coalocado com a aplicação Streamcluster do benchmark PARSEC.

ElasticSearch e uma *thread* carregada para o Streamcluster. A aplicação tem como característica ter processos de granularidade média, o que não comprometeu o funcionamento da proposta.

Na Figura 5.3 pode-se observar o comportamento da aplicação com os diferentes escalonadores, CFS e EDF Coloc. O EDFColoc manteve os prazos de SLA bem próximos em todos os cenários, enquanto que com o uso do CFS ocorre violação quando existe maior demanda da CPU.

A Figura 5.7 apresenta o resultado obtido da coalocação do ElasticSearch com a aplicação Fluidanimate. A aplicação Fluidanimate simula a física do movimento dos fluido para animação em tempo real. O EDFColaoc mostrou-se eficaz em promover a execução coalocada dessa aplicação com o ElasticSearch. Os ganhos mais consideráveis ocorreram quando a exigência por recursos foi maior. Com a alta demanda de processador para o ElasticSearch e com 4 e 6 *threads* no Fluidanimate os ganhos foram 68% e 64% respectivamente. O menor ganho foi de 4% com demanda baixa de processador para o do ElasticSearch e 2 *threads* com Fluidanimate. O Fluidanimate executa em com paralelismo e necessita de um numero de *threads* par para executar. Por isso o gráfico apresenta apenas as execuções duas e quatro *threads* carregadas. Tal característica não comprometeu o desempenho das aplicações com uso do EDFCoaloc.

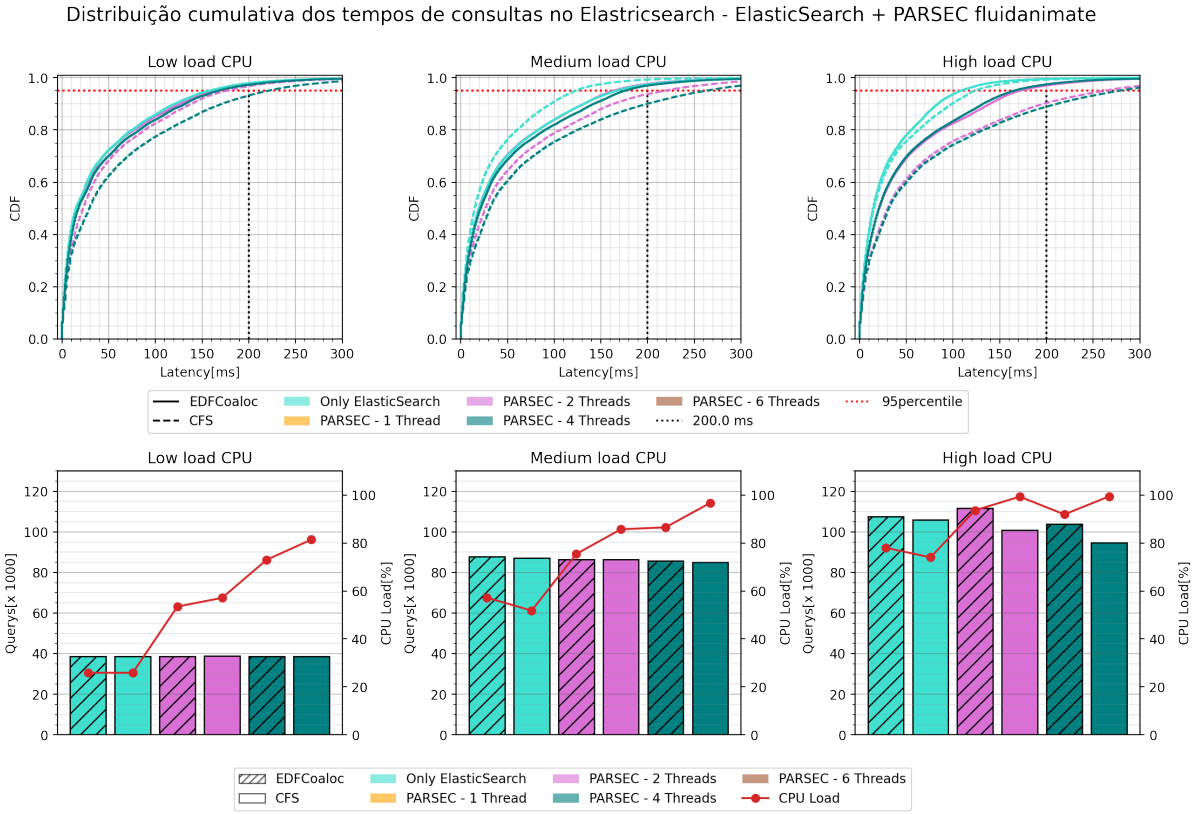

Figura 5.7 Distribuição cumulativa dos tempos de consulta com uso do ElasticSearch coalocado com a aplicação Fluidanimate do benchmark PARSEC.

Figura 5.3 apresenta violação da SLA da aplicação com uso do escalonador CFS na maioria dos cenários avaliados. O EDFCoaloc consegue garantir os prazos definidos para o ElasticSearch.

Figura 5.8 apresenta o resultado obtido da coalocação do ElasticSearch com a aplicação Freqmine. Freqmine é um aplicativo de mineração de dados, sua execução consiste em identificar padrões que ocorrem com alguma frequência. A granularidade da sua execução é média e apresenta grande demanda leitura de memória.

Os ganhos com uso do EDFCoaloc com essa aplicação proporcionou uma melhora de até 108% quando a demanda de CPU para o ElasticSearch foi alta e foram utilizadas 4 threads para o Freqmine. O menor ganho foi de 2% com baixa demanda de CPU para ElasticSearch e 1 *thread* de execução para o Freqmine.

A Figura 5.3 apresenta violação da SLA da aplicação com uso do escalonador CFS sobretudo quando utilizados 4 ou 6 threads para a aplicação Freqmine. O EDFCoaloc ajusta os prazos de CPU e garante os tempos de SLA em todos os cenários testados. São apresentados um resumo dos experimentos organizados por *thread* e por aplicação. Também é apresentado o gráfico de execução do ElasticSearch isolado a título de comparação. Verifica-se que o EDFCoaloc foi capaz de garantir os prazos de SLA em todos os cenários testados. Com ajuste dinâmico dos tempos de uso da CPU, foi possível pro-

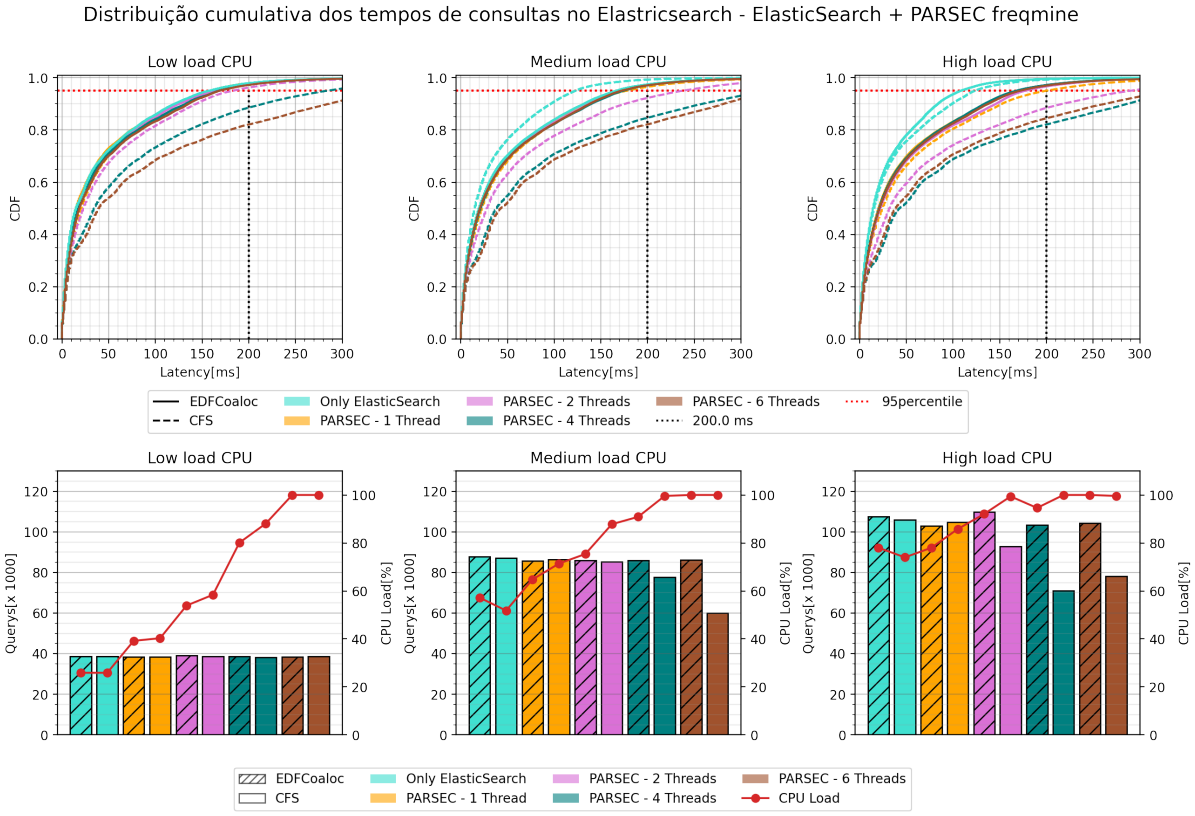

Figura 5.8 Distribuição cumulativa dos tempos de consulta com uso do ElasticSearch coalocado com a aplicação Freqmine do benchmark PARSEC.

mover essa garantia e permitir a execução da aplicação secundária sem prejuízos para o ElasticSearch.

Capítulo 6

CONCLUSÃO

Este capítulo finaliza o trabalho, apresenta uma visão do geral e sintetiza os resultados alcançados. Por fim são apresentadas as considerações finais.

Este trabalho teve como objetivo apresentar um proposta para melhorar a eficiência no uso dos recursos de sistemas em nuvem através da coalocação de processos sensíveis à latência e melhor esforço com uso de diferentes escalonadores CFS e EDF, buscando executar ambas as aplicações e garantir a SLA da aplicação sensível à latência. Para a organização da execução de ambas as aplicações foi proposto o meta escalonador, aqui denominado EDFCoaloc, que monitora as aplicações e sua execução, acompanhando a latência das requisições realizadas à aplicação crítica e ajusta parâmetros de configuração do escalonado EDF. Essas informações são utilizadas para os ajuste dos tempos de Q_i e T_i do escalonador EDF em função do tempo de processamento das consultas, o EDFCoaloc avalia os tempos das consultas já realizadas a fim de definir os melhores valores para os parâmetros.

Coalocar trabalhos em lote com aplicações sensíveis à latência é uma forma interessante de aumentar a utilização e eficiência dos *data centers*. Aplicações sensíveis à latência são complexas e seguem uma arquitetura organizada e bem distribuída, além de exigirem o respeito à SLAs. A diversidade de sistemas comerciais e de aplicações limita, em algumas situações, a aplicabilidade de algumas alternativas de redução dos custos.

Para a avaliação da proposta foram utilizadas a aplicação ElasticSearch como aplicação sensível à latência e o bechmark PARSEC, com suas diversas aplicações, como aplicação de melhor esforço. O ElasicSearch utilizou uma base gerada a partir da Wikipédia e para geração das consultas ao ElasticSearch foi utilizada a aplicação Faban. O experimento foi realizado na plataforma Google Cloud, com utilização de duas máquinas virtuais, uma para execução do ElasticSearch e PARSEC, enquanto a outra máquina foi utilizada para geração das consultas pelo Faban.

O experimento foi organizado com a combinação das variações das possibilidades de execução de cada uma das aplicações. Executando o ElasticSearch isoladamente foram definidos três níveis de carga no sistemas, *low load*, *medium load* e *high load*. Para

aplicação PAESEC foram carregadas as aplicações com variação no número de threads, variando entre 1, 2, 4, e 6. Cada cenário foi executado e os dados coletados foram apresentados em gráficos.

A coalocação com diferentes escalonadores definiu um esquema de prioridades em que os requisitos foram atendidos para as aplicações servidas. Conforme demonstrado nos dados apresentados, o EDFCoaloc apresentou melhores resultados do que escalonador padrão, sobretudo quando a carga de trabalho é composta de processos mais independentes, ou quando possui paralelismo de grão fino e utiliza um número maior de *threads*, visto que as mesmas ficam menos tempo bloqueadas aguardando algum recurso.

A proposta mostrou-se eficiente, melhorando em até 150% os resultados em comparação com o escalonador padrão do sistema, comprovando que uma boa gestão e dimensionamento dos recursos computacionais para as aplicações, com coalocação de diferentes aplicações, pode trazer uma melhor eficiência no uso dos recursos. Isso traz melhor aproveitamento do capital investido, pois ataca o problema da ociosidade do sistema, que gera um custo fixo de energia dos servidores. Esta é uma situação e possibilidade pouco explorada visto a dificuldade de garantias do QoS das aplicações críticas, contundo promissora diante dos desafios impostos no cenário de eficiência de uso dos recursos.

Tendo obtido bons resultados com a heurística mostrado-se eficiente, ainda é possível aprimorar o seu funcionamento, buscando novas métricas para cálculo dos ajustes a serem realizados no escalonador EDF. Uma avaliação possível de ser realizada é observar como métrica principal o uso do CPU e/ou *throughput* da aplicação de melhor esforço. Neste trabalho buscamos ampliar o uso do CPU, proporcionando coalocação com aplicação secundária sem deixar de garantir a SLA da aplicação principal.

Como sugestão para continuidade do trabalho pode ser considerado um aprimoramento da métrica para definição dos valores utilizados como parâmetro do escalonador EDF, uma possibilidade é a implementação de algoritmo que utilize PID para ajustes, contudo o consumo de recurso deve ser considerado, tendo em vista que pode prejudicar o desempenho do sistema.

REFERÊNCIAS BIBLIOGRÁFICAS

APPOLINÁRIO, F. *Metodologia da Ciência - Filosofia e Prática da Pesquisa*. 2ª. ed. São Paulo, SP: Cengage Learning, 2012. 240 p. ISBN 978-85-221-1471-9.

BARROS, A. J. P.; LEHFELD, N. A. d. S. *Fundamentos de Metodologia - Um Guia para a Iniciação Científica*. [S.l.]: Pearson Prentice Hall, 2000. 122 p.

BARROSO, L. A.; CLIDARAS, J.; HÖLZLE, U. The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second edition. *Synthesis Lectures on Computer Architecture*, v. 8, n. 3, p. 1–154, 7 2013. Disponível em: ⟨http://www.morganclaypool.com/doi/pdf/10.2200/S00516ED2V01Y201306CAC024⟩.

BARROSO, L. A.; HÖLZLE, U. The Case for Energy-Proportional Computing. *Computer*, v. 40, n. 12, p. 33–37, 12 2007. Disponível em: ⟨http://ieeexplore.ieee.org/document/4404806/⟩.

BIENIA, C. et al. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In: *Proceedings of the 17th international conference on Parallel architectures and compilation techniques - PACT '08*. [s.n.], 2008. p. 22. Disponível em: ⟨http://parsec.cs.princeton.edu/doc/parsec-report.pdfhttp://portal.acm.org/citation.cfm?doid=1454115.1454128⟩.

BLAGODUROV, S.; ZHURAVLEV, S.; FEDOROVA, A. Contention-Aware Scheduling on Multicore Systems. *ACM Transactions on Computer Systems*, ACM, v. 28, n. 4, p. 1–45, 12 2010. Disponível em: ⟨http://portal.acm.org/citation.cfm?doid=1880018.1880019⟩.

BRANDT, S. et al. Dynamic integrated scheduling of hard real-time, soft real-time, and non-real-time processes. In: *Proceedings. 2003 International Symposium on System-on-Chip (IEEE Cat. No.03EX748)*. [s.n.], 2003. p. 396–407. Disponível em: ⟨https://users.soe.ucsc.edu/~sbrandt/courses/Fall11/221/Papers/RT/rtss.pdfhttp://ieeexplore.ieee.org/document/1253287/⟩.

BUYYA, R.; BELOGLAZOV, A.; ABAWAJY, J. Energy-Efficient Management of Data Center Resources for Cloud Computing : A Vision , Architectural Elements , and Open Challenges Clou d Computing and D istributed S ystems (CLOUDS) Laboratory Department of Computer Science and Software Engineering The. *The University of Melbourne, Australia*, n. Vm, p. 1–12, 2010. Disponível em: ⟨https://arxiv.org/pdf/1006.0308.pdf⟩.

CHALLITA, S. et al. A Precise Model for Google Cloud Platform. In: *IEEE International Conference on Cloud Engineering - IC2E 2018*. Orlando: [s.n.], 2018. Disponível em: ⟨https://hal.inria.fr/hal-01689659/document⟩.

CHOU, C.-H.; WONG, D.; BHUYAN, L. N. DynSleep. *Proceedings of the 2016 International Symposium on Low Power Electronics and Design - ISLPED '16*, ACM Press, New York, New York, USA, p. 212–217, 2016. Disponível em: ⟨http://dl.acm.org/citation.cfm?doid=2934583.2934616⟩.

DEAN, J.; BARROSO, L. A. The Tail at Scale. *Communications of the ACM*, v. 56, n. 2, p. 74, 2013. ISSN 00010782. Disponível em: ⟨http://dl.acm.org/citation.cfm?id=2408794\%5Cnhttp://dl.acm.org/citation.cfm?doid=2408776.2408794⟩.

ELASTIC, I. *Elasticsearch Reference*. 2018. Disponível em: ⟨https://www.elastic.co/guide/en/elasticsearch/reference/current/getting-started.html⟩.

FABAN, I. *About Faban*. 2018. Disponível em: ⟨http://faban.org/about.html⟩.

FAGGIOLI, D. et al. An EDF scheduling class for the Linux kernel. *Proceedings of the 11th Real Time Linux Workshop (RTLW)*, p. 8 pp., 2009. Disponível em: ⟨https://static.lwn.net/images/conf/rtlws11/papers/proc/p16.pdfpapers/Faggioli2009.pdf⟩.

GOOGLE, I. *Eficiência: como fazemos isso – Data Centers – Google*. 2018. Disponível em: ⟨https://www.google.com/about/datacenters/efficiency/internal/⟩.

HSU, C. H. et al. Adrenaline: Pinpointing and reining in tail queries with quick voltage boosting. In: *2015 IEEE 21st International Symposium on High Performance Computer Architecture, HPCA 2015*. IEEE, 2015. p. 271–282. Disponível em: ⟨http://ieeexplore.ieee.org/document/7056039/⟩.

IRMA, I. *Application development and design : concepts, methodologies, tools, and applications*. 1 ed.. ed. [S.l.]: IGI Global, 2017. 1611 p.

KALYANI, D.; MEHTA, D. D. Paper on Searching and Indexing Using Elasticsearch. *International Journal Of Engineering And Computer Science*, v. 6, n. 6, 6 2017. ISSN 23197242. Disponível em: ⟨http://ijecs.in/index.php/ijecs/article/view/2986http://ijecs.in/issue/v6-i6/45ijecs.pdf⟩.

KASTURE, H. et al. Rubik: Fast Analytical Power Management for Latency-Critical Systems. In: *Proceedings of the 48th International Symposium on Microarchitecture - MICRO-48*. [s.n.], 2015. p. 598–610. Disponível em: ⟨http://people.csail.mit.edu/sanchez/papers/2015.rubik.micro.pdf⟩.

KERNEL.ORG. *Deadline Task Scheduling — The Linux Kernel documentation*. 2021. Disponível em: ⟨https://docs.kernel.org/scheduler/sched-deadline.html⟩.

KIM, S. et al. Delayed-Dynamic-Selective (DDS) Prediction for Reducing Extreme Tail Latency in Web Search. In: *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*. New York, NY, USA: ACM, 2015. p. 7–16. Disponível em: ⟨https://dl.acm.org/doi/10.1145/2684822.2685289⟩.

KONTORINIS, V. et al. Managing distributed UPS energy for effective power capping in data centers. In: *Proceedings - International Symposium on Computer Architecture*. [s.n.], 2012. p. 488–499. ISSN 10636897. Disponível em: ⟨http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.587.4150&rep=rep1&type=pdf⟩.

LELLI, J. *SCHED_DEADLINE – How to use it*. Pisa, Itália: Institute of Communication, Information and Perception Technologies. Scuola Superiore Sant'Anna, 2014. Disponível em: ⟨http://retis.sssup.it/~jlelli/talks/rts-like14/SCHED_DEADLINE.pdf⟩.

LELLI, J. et al. Deadline scheduling in the Linux kernel. *Software: Practice and Experience*, v. 46, n. 6, p. 821–839, 6 2016. Disponível em: ⟨http://doi.wiley.com/10.1002/spe.2335⟩.

LEVERICH, J.; KOZYRAKIS, C. Reconciling high server utilization and sub-millisecond quality-of-service. In: *Proceedings of the Ninth European Conference on Computer Systems - EuroSys '14*. [s.n.], 2014. p. 1–14. Disponível em: ⟨http://csl.stanford.edu/~christos/publications/2014.mutilate.eurosys.pdf⟩.

LI, J. et al. Tales of the Tail. *Proceedings of the ACM Symposium on Cloud Computing - SOCC '14*, p. 1–14, 2014. Disponível em: ⟨http://dx.doi.org/10.1145/2670979.2670988⟩.

LIU, C. L.; LAYLAND, J. w. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, v. 20, n. 1, p. 46–61, 1 1973. Disponível em: ⟨http://www.di.ens.fr/~pouzet/cours/systeme/bib/liu73.pdfhttp://portal.acm.org/citation.cfm?doid=321738.321743⟩.

LIU, Z. et al. Pricing Data Center Demand Response. In: *ACM SIGMETRICS*. [s.n.], 2014. p. 111–123. Disponível em: ⟨http://dl.acm.org/citation.cfm?doid=2591971.2592004⟩.

LO, D. et al. Towards Energy Proportionality for Large-Scale Latency-Critical Workloads. *ACM SIGARCH Computer Architecture News*, IEEE Press, v. 42, n. 3, p. 301–312, 10 2014. Disponível em: ⟨http://dl.acm.org/citation.cfm?doid=2678373.2665718⟩.

LO, D. et al. Heracles: Improving Resource Efficiency at Scale. In: *Proceedings of the 42nd Annual International Symposium on Computer Architecture - ISCA '15*. [s.n.], 2015. p. 450–462. Disponível em: ⟨http://csl.stanford.edu/~christos/publications/2015.heracles.isca.pdf⟩.

LOZI, J.-P. et al. The Linux scheduler. In: *Proceedings of the Eleventh European Conference on Computer Systems - EuroSys '16*. New York, New York, USA: ACM Press, 2016. p. 1–16. Disponível em: ⟨https://hal.archives-ouvertes.fr/hal-01295194/documenthttp://dl.acm.org/citation.cfm?doid=2901318.2901326⟩.

MARS, J. et al. Bubble-Up: Increasing Utilization in Modern Warehouse Scale Computers via Sensible Co-locations. *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture - MICRO-44 '11*, p. 248, 2011. Disponível em: ⟨http://www.cs.virginia.edu/~skadron/Papers/mars_micro2011.pdf⟩.

MEISNER, D. et al. Power management of online data-intensive services. *ACM SIGARCH Computer Architecture News*, v. 39, n. 3, p. 319, 7 2011. ISSN 01635964. Disponível em: ⟨http://dl.acm.org/citation.cfm?doid=2024723.2000103⟩.

PABLA, C. S. *Completely fair scheduler*. Robert F. Young, 2009. 4 p. Disponível em: ⟨https://dl.acm.org/citation.cfm?id=1594375https://en.wikipedia.org/wiki/Completely_Fair_Scheduler⟩.

VAMANAN, B. et al. TimeTrader: Exploiting Latency Tail to Save Datacenter Energy for Online Search. *Proceedings of the 48th International Symposium on Microarchitecture - MICRO-48 (2015) 585-597*, p. 585–597, 2015. Disponível em: ⟨http://dl.acm.org/citation.cfm?id=2830772.2830779⟩.

WANG, G. et al. Increasing Large-scale Data Center Capacity by Statistical Power Control. *Proceedings of the Eleventh European Conference on Computer Systems*, p. 8:1–8:15, 2016. Disponível em: ⟨https://pdfs.semanticscholar.org/f657/588b14e7e558d3e231ad3796cf043cf02105.pdf⟩.

WIERMAN, A. et al. Opportunities and challenges for data center demand response. In: *International Green Computing Conference*. [s.n.], 2014. p. 1–10. ISBN 978-1-4799-6177-1. Disponível em: ⟨http://rsrg.cms.caltech.edu/greenIT/papers/dcdrsurvey.pdf⟩.

YANG, H. et al. Bubble-Flux: Precise online QoS management for increased utilization in warehouse scale computers. *Isca'13*, p. 12, 2013. Disponível em: ⟨http://clarity-lab.org/wp-content/papercite-data/pdf/yang13isca.pdf⟩.

ZATS, D. et al. DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks. In: *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication - SIGCOMM '12*. New York, New York, USA: ACM Press, 2012. p. 139. Disponível em: ⟨http://www.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-113.html⟩.

ZHANG, Q. et al. Dynamic energy-aware capacity provisioning for cloud computing environments. In: *Proceedings of the 9th international conference on Autonomic computing - ICAC '12*. [s.n.], 2012. p. 145. Disponível em: ⟨http://dl.acm.org/citation.cfm?doid=2371536.2371562⟩.

ZHANG, X.; TUNE, E.; HAGMANN, R. CPI2: CPU performance isolation for shared compute clusters. *EuroSys'13*, p. 1–13, 2013. Disponível em: ⟨http://dl.acm.org/citation.cfm?id=2465388⟩.

ZHU, H.; EREZ, M. Dirigent: Enforcing QoS for Latency-Critical Tasks on Shared Multicore Systems. In: *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS '16*. [s.n.], 2016. p. 33–47. Disponível em: ⟨https://lph.ece.utexas.edu/merez/uploads/MattanErez/asplos2016_dirigent.pdf⟩.