UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

CLEBER TAVARES DE MOURA

AVALIAÇÃO DE DESEMPENHO DAS ESTRATÉGIAS DE PARTICIONAMENTO DE GRANDES VOLUMES DE DADOS APLICADA AO JUMP.

RECIFE
2025

CLEBER TAVARES DE MOURA

AVALIAÇÃO DE DESEMPENHO DAS ESTRATÉGIAS DE PARTICIONAMENTO DE GRANDES VOLUMES DE DADOS APLICADA AO JUMP.

Dissertação apresentada ao Programa de Pós-Graduação em Ciências da Computação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de mestre em Ciências da Computação. Área de Concentração: Engenharia de Software e Avaliação de desempenho.

Orientador: Ricardo Massa Ferreira Lima

RECIFE
2025

FICHA CATALOGRÁFICA

.Catalogação de Publicação na Fonte. UFPE - Biblioteca Central

Moura, Cleber Tavares de.
Avaliação de desempenho das estratégias de particionamento de grandes volumes de dados aplicada ao JuMP / Cleber Tavares de Moura. - Recife, 2025.
88f.: il.

Dissertação (Mestrado) - Universidade Federal de Pernambuco, Centro de Informática, Programa de Pós-Graduação em Ciência da Computação, 2025.
Orientação: Ricardo Massa Ferreira Lima.
Inclui referências.

1. Bancos de dados relacionais; 2. Big data; 3. Metodologia de avaliação; 4. Mineração de processos; 5. Simulação de carga. I. Lima, Ricardo Massa Ferreira. II. Título.

UFPE-Biblioteca Central

CLEBER TAVARES DE MOURA

AVALIAÇÃO DE DESEMPENHO DAS ESTRATÉGIAS DE PARTICIONAMENTO DE GRANDES VOLUMES DE DADOS APLICADA AO JUMP.

Dissertação apresentada ao Programa de Pós-Graduação em Ciências da Computação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de mestre em Ciências da Computação. Área de Concentração: Engenharia de Software e Avaliação de desempenho.

Aprovado em: ___/___/_____.

BANCA EXAMINADORA

_____
Prof. Dr. Ricardo Massa Ferreira Lima (Orientador)
Universidade Federal de Pernambuco - UFPE

_____
Prof. Dr. Adriano Lorena Inácio de Oliveira (Examinador Interno)
Universidade Federal de Pernambuco - UFPE

_____
Dr. Raphael José D`Castro (Examinador Externo)
Tribunal de Justiça- PE

A Deus, que nos deu condições para terminar essa jornada, minha esposa Angelina Soares de Figueiredo Moura e filhas Alice e Aurora que com muita paciência e compreensão me possibilitaram dedicar tempo a este trabalho, a meus pais que sempre foram minha base, meus irmãos Morgana e Cleston que me deram o suporte necessário para iniciar minha carreira profissional, aos amigos que me incentivaram a perseverar e ao meu orientador pela dedicação, direção e colaboração para conclusão desse projeto. E também a todo corpo docente desta ilustre instituição de ensino. Dedico!

"Feliz aquele que transfere o que sabe e aprende o que ensina. O saber se aprende com mestres e livros. A Sabedoria, com o corriqueiro, com a vida e com os humildes. O que importa na vida não é o ponto de partida, mas a caminhada. Caminhando e semeando, sempre se terá o que colher".

Cora Coralina

RESUMO

O aumento exponencial na geração de dados, impulsionado por dispositivos conectados, redes sociais e aplicativos móveis, tem gerado grandes desafios para a análise e processamento dessas informações. No contexto do Poder Judiciário Brasileiro, o crescente volume de processos judiciais eletrônicos demanda soluções eficientes para o gerenciamento e processamento desses dados massivos. O Judiciário com Mineração de Processos (JuMP), desenvolvido em parceria entre o Conselho Nacional de Justiça (CNJ) e o V-Lab da Universidade Federal de Pernambuco (UFPE), utiliza técnicas de mineração de processos para otimizar o fluxo processual e identificar gargalos no sistema judiciário. No entanto, o aumento contínuo no número de processos resulta em um volume de dados ainda maior, o que cria novos desafios e exige estratégias adequadas de particionamento e gerenciamento. Este estudo avaliou quatro estratégias de particionamento de dados aplicadas ao JuMP: por Chave, Intervalo, Hash e Híbrido (Intervalo + Lista e Intervalo + Hash), com o objetivo de identificar a abordagem mais eficaz para o armazenamento, processamento e análise de grandes volumes de dados heterogêneos e dinâmicos. A pesquisa foi conduzida por meio de experimentos que avaliaram o desempenho das estratégias com base em parâmetros como tempo de resposta, escalabilidade, custo de redistribuição e taxa de transferência. Os resultados indicaram que a estratégia Híbrida (Intervalo + Lista) obteve o melhor desempenho, com 0,89 segundos de tempo de resposta e uma taxa de transferência de 404.061 registros por segundo, sendo capaz de suportar até 21 usuários simultâneos. Embora a estratégia Híbrida (Intervalo + Hash) também tenha mostrado bons resultados, ela apresentou custos elevados de redistribuição. Por outro lado, o particionamento por Chave revelou-se ineficiente, com falhas superiores a 20% após 13 usuários simultâneos. Conclui-se que, para o JuMP, a estratégia Híbrida (Intervalo + Lista) é a mais indicada, especialmente em cenários de crescimento das partições, desde que o impacto dos custos de redistribuição seja monitorado cuidadosamente.

Palavras-chave: bancos de dados relacionais; big data; metodologia de avaliação; mineração de processos; simulação de carga.

ABSTRACT

The exponential increase in data generation, driven by connected devices, social media, and mobile applications, has created significant challenges for analyzing and processing this information. In the context of the Brazilian Judiciary, the growing volume of electronic legal proceedings demands efficient solutions for managing and processing these massive datasets. The Judiciary with Process Mining (JuMP), developed in partnership between the National Justice Council (CNJ) and the V-Lab at the Federal University of Pernambuco (UFPE), uses process mining techniques to optimize procedural flow and identify bottlenecks in the judicial system. However, the continuous increase in the number of cases results in an even larger volume of data, which creates new challenges and requires appropriate partitioning and management strategies. This study evaluated four data partitioning strategies applied to JuMP: by Key, Interval, Hash, and Hybrid (Interval + List and Interval + Hash), aiming to identify the most effective approach for storing, processing, and analyzing large volumes of heterogeneous and dynamic data. The research was conducted through experiments that assessed the performance of the strategies based on parameters such as response time, scalability, redistribution cost, and data transfer rate. The results indicated that the Hybrid strategy (Interval + List) achieved the best performance, with a response time of 0.89 seconds and a data transfer rate of 404,061 records per second, being able to support up to 21 simultaneous users. Although the Hybrid strategy (Interval + Hash) also showed good results, it presented high redistribution costs. On the other hand, partitioning by Key proved to be inefficient, with failures exceeding 20% after 13 simultaneous users. It is concluded that, for JuMP, the Hybrid strategy (Interval + List) is the most suitable, especially in scenarios with partition growth, provided that the impact of redistribution costs is carefully monitored.

Keywords: big data; evaluation methodology; load simulation; process mining; relational databases.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

CIN	Centro de Informática da UFPE
CNJ	Conselho Nacional de Justiça
CPU	Central Processing Unit
CSV	Comma-Separated Values
JuMP	Judiciário com Mineração de Processos
NoSQL	Not Only SQL
PDPJ-Br	Plataforma Digital do Poder Judiciário Brasileiro
PJ-e	Processo Judicial Eletrônico
SQL	Structured Query Language
TI	Tecnologia da informação
TOPSIS	Technique for Order of Preference by Similarity to Ideal Solution
UFPE	Universidade Federal de Pernambuco

SUMÁRIO

1 INTRODUÇÃO

O crescimento exponencial na geração e no armazenamento de dados, impulsionado por fontes como dispositivos conectados, redes sociais e aplicativos móveis, tem gerado desafios significativos para o processamento e a análise eficiente dessas informações (Li *et al*., 2022). Desse modo, é fundamental que as organizações desenvolvam e implementem estratégias robustas para gerenciar o grande volume de dados gerado, permitindo a obtenção de *insights* valiosos que embasem a tomada de decisões estratégicas.

No âmbito do Poder Judiciário Brasileiro, a Emenda Constitucional nº 45 de 2004 conferiu ao Conselho Nacional de Justiça (CNJ) a responsabilidade de promover a modernização e aprimoramento da administração do sistema judiciário, com o objetivo de assegurar uma justiça mais célere e acessível à sociedade. Em consonância com essa reforma, a Lei nº 11.419/06 determinou a digitalização dos processos judiciais, simplificando sua tramitação e promovendo maior eficiência e transparência no acesso à justiça (Braz, 2022). Em 2010, como continuidade a esse processo de modernização, foi lançada a plataforma digital Processo Judicial Eletrônico (PJe), permitindo aos tribunais definir, monitorar e gerenciar o fluxo processual de forma eletrônica (Tomio; Robl Filho; Santos-Pinto, 2015).

Contudo, o aumento contínuo no número de processos judiciais resulta em um volume massivo de dados (CNJ, 2024a), o que exige, a adoção de soluções tecnológicas adequadas para armazenar, processar e analisar esses dados de maneira eficiente e escalável. Nesse cenário, surgiu a iniciativa do Judiciário com Mineração de Processos (JuMP), uma parceria do V-Lab, laboratório vinculado à Universidade Federal de Pernambuco (UFPE), com o CNJ (CNJ, 2024b). O JuMP utiliza técnicas de Mineração de Processos para analisar os dados transacionais dos processos judiciais, com o objetivo de diagnosticar o funcionamento do sistema judiciário, identificar gargalos no fluxo processual, propor melhorias e avaliar o impacto das modificações implementadas (Lima *et al*., 2023).

Esta ferramenta utiliza dados extraídos do *data lake Codex*, de propriedade do CNJ, que integra as bases de dados processuais dos tribunais brasileiros, totalizando 144 milhões de processos armazenados até 2023. (CNJ, 2023). Desses, mais de 23 milhões foram carregados no JuMP até o mesmo ano, e novos processos continuam sendo incorporados de forma constante. Aliado a isso, heterogeneidade dos dados (Diferentes arquivos: texto, imagem, áudio, vídeo e

diversas extensões: .PDF, .DOCX, .MP3, .MP4) e natureza dinâmica dessas informações, torna o gerenciamento e a interpretação dos dados tarefas extremamente desafiadoras.

Dessa forma, o particionamento de dados se configura como uma estratégia fundamental para garantir a eficiência e escalabilidade do sistema, permitindo a distribuição equilibrada do processamento entre os nós e otimizando os recursos computacionais disponíveis (Özsu e Valduriez, 2020). No entanto, a implementação de estratégias de particionamento adequadas exige uma análise detalhada das características dos dados e dos padrões de acesso, uma vez que uma escolha inadequada pode comprometer o desempenho do sistema.

Apesar da relevância do tema, a literatura ainda apresenta uma lacuna significativa em estudos comparativos sobre as estratégias de particionamento de dados aplicadas a sistemas judiciais eletrônicos. Embora existam estudos sobre particionamento em sistemas de grande porte (Zhang *et al*., 2019; Costa; Costa; Santos, 2019; Liu; Li; Chen, 2024), até onde sabemos, não há estudos avaliando as especificidades dos dados judiciais, que são altamente heterogêneos e exigem operações complexas.

Nesse sentido, o objetivo deste estudo foi avaliar e comparar quatro estratégias de particionamento de dados: por Chave, Intervalo, *Hash* e Híbrido (Intervalo + *Hash* e Intervalo + Lista), aplicadas ao sistema JuMP, com a finalidade de identificar a estratégia mais adequada para garantir a eficiência e escalabilidade no armazenamento, processamento e análise de grandes volumes de dados heterogêneos e dinâmicos, características típicas desses sistemas. Além disso, buscou-se fornecer *insights* sobre como essas estratégias influenciam o desempenho das operações e a qualidade das consultas realizadas nos sistemas judiciais eletrônicos.

1.1 DESAFIOS

O aumento contínuo no volume de dados gerados por sistemas modernos exige abordagens sofisticadas e escaláveis para garantir a eficiência no armazenamento, processamento e análise dessas informações. Um dos principais desafios enfrentados em ambientes distribuídos é o desequilíbrio na distribuição dos dados entre os nós de processamento, que pode resultar em sobrecarga de determinados nós ou subutilização de outros, comprometendo o desempenho global do sistema (Zaharia *et al*., 2016). Superar esse desafio requer o desenvolvimento de técnicas avançadas de particionamento de dados, que não

apenas considerem a distribuição dos dados, mas também os padrões de acesso e as variações temporais nas operações do sistema (Van Steen e Tanenbaum, 2017).

Além disso, à medida que o sistema cresce e novos dados são continuamente incorporados, a escalabilidade das estratégias de particionamento se torna uma preocupação fundamental. As soluções adotadas precisam ser suficientemente flexíveis para escalar de maneira eficiente, sem comprometer o desempenho do sistema ou a integridade dos dados, especialmente quando os dados se acumulam em volumes massivos e as operações de consulta se tornam mais complexas (Dean e Ghemawat, 2008). Nesse cenário, é essencial que as soluções de particionamento consigam se adaptar às novas demandas, garantindo a continuidade operacional sem aumento de latência.

Outro desafio relevante é a heterogeneidade dos dados. Sistemas que processam diferentes tipos de dados, com variações em formato, origem e frequência de atualização, necessitam de estratégias de particionamento que sejam capazes de lidar com essa diversidade, ao mesmo tempo em que otimizam o uso dos recursos computacionais. Para isso, soluções de particionamento devem ser dinâmicas o suficiente para se ajustar a diferentes tipos de dados e arquiteturas de hardware (Zaharia *et al*., 2010).

A segurança e a privacidade dos dados também representam um desafio significativo, especialmente em sistemas distribuídos, onde múltiplos nós interagem em ambientes frequentemente expostos a riscos de segurança. A implementação de mecanismos robustos, como criptografia e controle de acesso, é essencial para assegurar que dados sensíveis sejam protegidos contra ameaças externas (Lamport, 2019).

Ademais, a complexidade operacional e a manutenção contínua desses sistemas distribuídos apresentam desafios adicionais. Não basta implementar uma estratégia de particionamento eficaz no início do ciclo de vida do sistema, ela deve ser capaz de ser ajustada e otimizada de forma contínua, em resposta a mudanças nos padrões de dados e nos requisitos de desempenho. Esse processo de adaptação dinâmica é fundamental para garantir que o sistema continue eficiente à medida que evolui, tornando-se cada vez mais complexo (DeCandia *et al*., 2007).

Dessa forma, a identificação e implementação de estratégias de particionamento de dados eficientes e escaláveis são essenciais para o sucesso de sistemas que lidam com grandes volumes de dados heterogêneos e exigem alto desempenho e robustez. Este estudo visa investigar soluções que abordem de maneira eficaz esses desafios, propondo novas abordagens

que não só melhorem o particionamento de dados, mas também otimizem o desempenho e a escalabilidade, avançando no desenvolvimento de sistemas mais eficientes e resilientes em ambientes distribuídos.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Realizar uma avaliação comparativa das estratégias de particionamento de dados comumente utilizadas (Chave, Intervalo e *Hash*) e abordagens Híbridas (Intervalo + Lista e Intervalo + *Hash*), com a finalidade de analisar o impacto dessas técnicas no desempenho, escalabilidade e eficiência no processamento de grandes volumes de dados.

1.2.2 Objetivos Específicos

- Realizar uma análise aprofundada e uma avaliação crítica das estratégias de particionamento de dados mais frequentemente utilizadas;
- Avaliar o desempenho das estratégias de particionamento, considerando o tempo de resposta, utilização de recursos, escalabilidade e eficiência das consultas;
- Fornecer recomendações práticas para otimizar o desempenho e a eficiência de consultas de grandes volumes de dados e acessos concorrentes em sistemas de bancos de dados;
- Validar as estratégias de particionamento de dados por meio da realização de experimentos em um ambiente controlado, que simule condições reais.

1.3 HIPÓTESE

A implementação de estratégias híbridas de particionamento de dados, combinando Intervalo + *Hash* e Intervalo + Lista, resulta em uma melhora substancial na eficiência e escalabilidade no armazenamento, processamento e análise de grandes volumes de dados dinâmicos e heterogêneos, como os encontrados no contexto do Processo Judicial Eletrônico (PJ-e). Essas abordagens híbridas superam as técnicas tradicionais de particionamento (Chave,

Intervalo e *Hash* em aspectos críticos como tempo de resposta, taxa de transferência de dados e capacidade de escalabilidade, especialmente em cenários caracterizados por alta carga.

1.4 QUESTÕES DA PESQUISA

Este trabalho visa responder às seguintes questões de pesquisa, com o objetivo de investigar o impacto e a eficácia das estratégias de particionamento de dados aplicadas a sistemas que lidam com grandes volumes de dados, especialmente em contextos de ambientes distribuídos, como o JuMP.

QP-1. Quais são as principais estratégias de particionamento de dados aplicadas em sistemas distribuídos que lidam com grandes volumes de dados, especialmente no contexto da ferramenta JuMP?

Esta questão visa identificar, categorizar e detalhar as técnicas de particionamento de dados mais amplamente empregadas em ambientes de processamento de grandes volumes de dados. O estudo enfoca técnicas como particionamento por Intervalo, por *Hash* e abordagens Híbridas (Intervalo + Lista, Intervalo + *Hash*). A análise pretende examinar os cenários nos quais essas estratégias são mais eficazes no contexto da ferramenta JuMP, levando em consideração as características específicas do sistema de gestão de processos judiciais e os padrões de acesso aos dados, com ênfase nos aspectos de escalabilidade, eficiência e distribuição dos dados.

QP-2. Quais métricas são mais apropriadas para avaliar o desempenho das estratégias de particionamento de dados em sistemas distribuídos, especificamente na ferramenta JuMP?

O objetivo desta questão é definir e selecionar as métricas mais relevantes para a avaliação e comparação de estratégias de particionamento de dados, com foco na ferramenta JuMP. Entre as métricas a serem investigadas estão o tempo de processamento, o uso de recursos computacionais (CPU e memória), a escalabilidade, o balanceamento de carga e a latência das operações de consulta. A avaliação dessas métricas fornecerá uma análise detalhada sobre o impacto das estratégias no desempenho e na eficiência global do sistema, levando em

consideração tanto a capacidade de resposta quanto a utilização otimizada dos recursos dentro do contexto de uso da ferramenta JuMP.

QP-3. Como as diferentes estratégias de particionamento lidam com padrões de acesso não uniformes e distribuições desiguais de dados, particularmente no contexto da ferramenta JuMP?

Esta questão se propõe a investigar a eficácia das diversas abordagens de particionamento de dados ao lidar com padrões de acesso variados, como leituras esporádicas versus gravações frequentes, e a distribuição desigual de dados em sistemas distribuídos de grandes dimensões, focando na ferramenta JuMP. O objetivo é examinar como as diferentes técnicas de particionamento abordam problemas comuns, como hotspots e gargalos de desempenho, especialmente em cenários de alta carga de trabalho e requisitos dinâmicos de processamento, como ocorre no JuMP, que lida com dados judiciais com padrões de acesso específicos.

QP-4. Qual é a capacidade de escalabilidade das estratégias de particionamento em diferentes cenários de carga de trabalho e crescimento de dados, especificamente no contexto da ferramenta JuMP?

A quarta questão se concentra em avaliar a escalabilidade das estratégias de particionamento de dados em sistemas com aumento contínuo de dados e variação nas cargas de trabalho, especificamente no contexto da ferramenta JuMP.

QP-5. Quais são as vantagens e limitações das estratégias de particionamento de dados em sistemas distribuídos, como o JuMP, considerando fatores como complexidade de implementação, custos operacionais e impacto no desempenho de consultas?

A quinta questão busca uma análise crítica das vantagens e limitações das diferentes estratégias de particionamento de dados em sistemas distribuídos, com foco na ferramenta JuMP. O objetivo é avaliar a complexidade de implementação de cada técnica, os custos operacionais envolvidos, a facilidade de manutenção e o impacto nas operações de consulta. A pesquisa visa fornecer uma análise comparativa que permita a seleção da estratégia mais apropriada para o JuMP, levando em consideração as particularidades dos sistemas judiciais

eletrônicos, como a necessidade de integração de dados entre diferentes tribunais ou o tratamento de consultas complexas em tempo real.

Essas questões orientam a pesquisa no sentido de compreender de maneira profunda o impacto das estratégias de particionamento de dados em sistemas que processam grandes volumes de dados, proporcionando insights valiosos para a otimização do desempenho e da eficiência. Ao responder a essas questões, buscamos oferecer orientações práticas para profissionais de TI e engenheiros de sistemas, auxiliando na escolha e configuração das estratégias de particionamento mais adequadas. Além disso, a pesquisa contribuiu para o avanço do conhecimento nessa área essencial da ciência da computação, incentivando o desenvolvimento de melhores práticas e técnicas mais eficazes para ambientes distribuídos de processamento de grandes volumes de dados.

2 REVISÃO DE LITERATURA

2.1 ESTRATÉGIAS DE PARTICIONAMENTO DE DADOS

O particionamento de dados consiste na divisão de um grande conjunto de dados em partes menores, denominadas partições, com o intuito de otimizar a performance e a escalabilidade no armazenamento, acesso e processamento desses dados. Essa técnica de distribuição de dados também visa melhorar o desempenho, pois possibilita que a varredura das tabelas seja realizada apenas nas partições que atendem aos critérios especificados nas consultas (Almeida *et al*., 2017).

Desse modo, a escolha da estratégia de particionamento de dados em sistemas distribuídos que lidam com grandes volumes de dados deve considerar as características específicas do sistema, os padrões de acesso aos dados e os requisitos de desempenho a serem atendidos (O'Neil e O'Neil, 2000). Uma abordagem híbrida, que combina elementos de diferentes estratégias de particionamento, pode ser fundamental para alcançar um equilíbrio ideal entre eficiência, escalabilidade e distribuição uniforme dos dados (Sudarshan e Silberschatz, 1999). Além disso, é necessário considerar o tipo de consulta predominante e o padrão de distribuição dos dados para garantir que o sistema atenda de forma eficaz às demandas de acesso e processamento (Silberschatz; Korth; Sudarshan, 2011).

A seguir, serão apresentadas as principais estratégias de particionamento de dados, com o objetivo de fornecer uma visão geral e comparativa das abordagens mais reconhecidas.

2.1.1 Estratégias tradicionais

Dentre as estratégias mais comuns de particionamento de dados, destacam-se algumas estratégias tradicionais, que são: o particionamento por Intervalo, Chave, Lista e *Hash*. Cada uma dessas abordagens apresenta características distintas que impactam a distribuição e o acesso aos dados (Elmasri, 2008).

2.1.1.1 Particionamento por Intervalo

O particionamento por Intervalo, descrito por Tanenbaum e Van Steen (2017), divide os dados em segmentos contíguos com base em um critério de ordenação. Esta estratégia é particularmente adequada para consultas que exigem acesso sequencial a uma série de registros. No entanto, como apontado por DeCandia *et al*. (2007), o particionamento por Intervalo pode enfrentar desafios relacionados à distribuição desigual dos dados, especialmente se houver uma variação significativa na distribuição dos valores-chave.

No particionamento por Intervalo, os dados são divididos em segmentos contíguos baseados em um critério de ordenação, como intervalos de valores de uma coluna específica (por exemplo, datas ou IDs numéricos) (Van Steen e Tanenbaum, 2017). Essa técnica é particularmente útil em cenários que envolvem consultas sequenciais, facilitando o acesso a registros consecutivos. Por exemplo, em sistemas financeiros que analisam transações por períodos de tempo, essa abordagem pode otimizar o desempenho das consultas. No entanto, a técnica pode apresentar problemas de balanceamento quando os dados não são distribuídos de forma uniforme, resultando em alguns segmentos excessivamente carregados enquanto outros ficam subutilizados (DeCandia *et al*. 2007).

2.1.1.2 Particionamento por Chave

Por outro lado, o particionamento por Chave atribui dados a nós (*partições*) específicos com base em uma chave de partição. Essa abordagem, conforme mencionado por Corbett *et al*. (2012), facilita o acesso rápido e direto aos dados necessários, tornando-a adequada para sistemas que exigem recuperação eficiente de informações com base em chaves específicas. No entanto, o particionamento por Chave pode resultar em desequilíbrios na distribuição dos dados, especialmente quando um conjunto limitado de chaves é excessivamente acessado (Zaharia *et al*. 2016).

2.1.1.3 Particionamento por Lista

O particionamento por lista é uma técnica de particionamento de dados em que cada partição armazena linhas que contêm valores específicos e predefinidos de uma determinada coluna. Essa abordagem é especialmente útil quando se conhece previamente um conjunto limitado de valores possíveis para a coluna particionada, como por exemplo, códigos de

unidades, categorias, regiões ou status. (POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2023).

2.1.1.4 Particionamento por *Hash*

O particionamento por *Hash* é uma técnica que utiliza uma função de *hash* para determinar a alocação dos dados entre os nós de processamento, distribuindo-os de forma uniforme. Isso melhora o balanceamento de carga, pois a função de *hash* mapeia os dados de maneira pseudo-aleatória entre os nós, evitando a concentração excessiva de registros em um único local. Essa técnica é especialmente adequada para cenários em que o acesso aos dados é imprevisível ou quando o volume de consultas é alto (Slesarev; Mikhailov; Chernishev, 2022). No entanto, a eficácia do particionamento por *Hash* depende da função de *hash* escolhida e da uniformidade dos dados originais. Se os dados apresentarem valores de *hash* concentrados em poucas posições, pode ocorrer desequilíbrio, impactando o desempenho do sistema (Luo; Carey, 2021). Além disso, a adição ou remoção de nós no sistema pode exigir a redistribuição significativa dos dados, o que pode ser custoso em termos de desempenho.

2.1.2 Estratégias Contemporâneas

Além das estratégias tradicionais de particionamento de dados em sistemas distribuídos, novas abordagens estão emergindo para maximizar a eficiência no processamento distribuído. Essas estratégias contemporâneas focam em melhorar a distribuição dos dados de maneira adaptativa e inteligente, considerando a evolução dos requisitos e padrões de acesso.

2.1.2.1 Particionamento por amostragem

O particionamento por amostragem representa uma abordagem avançada e eficiente, empregando uma amostra representativa dos dados para identificar padrões subjacentes de distribuição e comportamento de acesso. Ao realizar uma análise detalhada da amostra, é possível extrair informações cruciais sobre a organização e a frequência de acesso aos dados, possibilitando uma distribuição mais estratégica e equilibrada entre os nós de processamento. Essa técnica revela-se especialmente valiosa em cenários dinâmicos, onde a distribuição de

dados sofre variações temporais, permitindo ajustes flexíveis e em tempo real no particionamento, de acordo com as características observadas na amostra. Ao focar na análise de uma amostra, a técnica minimiza a necessidade de processamento intensivo de grandes volumes de dados completos, resultando em uma alocação mais eficiente e otimizada de recursos. (Rong *et al*., 2020).

2.1.2.2 Particionamento baseado em aprendizagem

O particionamento baseado em aprendizado de máquina é uma estratégia que aplica algoritmos de *machine learning* para identificar características e padrões nos dados que podem orientar a divisão entre os nós distribuídos. Utilizando técnicas de aprendizado supervisionado ou não supervisionado, essa abordagem adapta dinamicamente a distribuição dos dados com base em padrões de acesso em constante mudança e nos requisitos de desempenho do sistema. Por exemplo, algoritmos podem aprender a reconhecer picos de demanda em determinados períodos ou identificar tendências de acesso específicas, ajustando a distribuição dos dados para otimizar o desempenho e evitar gargalos. Essa flexibilidade torna o particionamento baseado em aprendizado particularmente eficaz em cenários onde os padrões de uso são imprevisíveis ou variam significativamente (Cantini *et al*., 2022; Hori *et al*., 2023).

2.1.2.3 Particionamento Geoespacial

Conforme discutido por Magalhães (2022), a redundância e o particionamento de dados convencionais e geoespaciais podem impactar significativamente o desempenho analítico em data warehouses orientados a colunas. Ganhando destaque em aplicações onde a localização dos dados é um fator crítico, como em sistemas de sensoriamento remoto, análise de dados geográficos ou serviços de localização (WANG et al., 2022). Essa estratégia utiliza técnicas avançadas de particionamento baseadas em índices espaciais, como quadtrees ou R-trees, e técnicas de agrupamento geoespacial para organizar os dados com base em suas coordenadas geográficas. Segundo Zein et al. (2023), o uso de particionamento baseado em *clustering* é eficaz para melhorar o desempenho no processamento de grandes volumes de dados geoespaciais, especialmente quando os dados são organizados com base na sua localização espacial. Essa abordagem melhora a eficiência do processamento distribuído ao reduzir a

sobrecarga de comunicação entre nós e otimizar a execução de consultas baseadas em localização, como aquelas que buscam informações em regiões geográficas específicas. Em cenários onde os dados geoespaciais estão distribuídos de forma irregular, o particionamento geoespacial pode equilibrar a carga entre os nós, melhorando o desempenho do sistema (Finkel, 1974; Guttman, 1984).

2.1.2.4 Particionamento Híbrido

As estratégias de particionamento Híbrido combinam diferentes abordagens de particionamento, como Intervalo, *Hash* e Lista, para otimizar a distribuição de grandes volumes de dados e melhorar o desempenho das consultas em sistemas distribuídos. Essa abordagem permite explorar as vantagens de cada técnica individualmente, garantindo um balanceamento eficiente de carga e reduzindo gargalos no processamento de dados (Alsubaiee *et al*., 2020). O particionamento Híbrido Intervalo + *Hash*, por exemplo, é frequentemente utilizado para distribuir dados de forma uniforme entre nós, evitando hotspots e mantendo um desempenho estável mesmo com o crescimento da carga de trabalho (Zaharia *et al*., 2016). Já o particionamento Intervalo + Lista é vantajoso para sistemas que necessitam de organização lógica dos dados por categorias específicas e por períodos de tempo.

2.1.2.5 Particionamento Dinâmico

O uso de técnicas de particionamento dinâmicas está se tornando cada vez mais comum, especialmente em ambientes onde a carga de trabalho e os padrões de acesso aos dados variam significativamente ao longo do tempo. Conforme proposto por Zaharia *et al*. (2010), essas abordagens adaptativas ajustam a distribuição dos dados de forma contínua com base nas condições atuais do sistema e na demanda. Esta técnica monitora métricas como volume de requisições e uso de recursos para redistribuir os dados entre os nós, evitando sobrecargas e melhorando o balanceamento. Isso é particularmente útil em cenários com picos imprevisíveis de demanda, permitindo que o sistema se adapte rapidamente e mantenha um desempenho consistente em sistemas de processamento distribuído.

2.1.3 Considerações Finais sobre Estratégias de Particionamento de Dados

Nesta seção foram apresentados os fundamentos teóricos e as principais estratégias de particionamento de dados, destacando sua relevância para sistemas distribuídos que processam grandes volumes de informações.

As abordagens contemporâneas refletem uma busca contínua por eficiência e resiliência diante da crescente complexidade e escala dos dados modernos. Técnicas emergentes, como amostragem adaptativa, aprendizado de máquina e otimização dinâmica, pavimentam o caminho para sistemas distribuídos mais ágeis e responsivos. Por exemplo, Cutkosky e Busa-Fekete (2018) propuseram um método eficiente de otimização estocástica distribuída que combina adaptatividade com técnicas de redução de variância, resultando em aceleração linear no número de máquinas e uma redução significativa na necessidade de ajuste de hiperparâmetros. Da mesma forma, Savva, Anagnostopoulos e Triantafillou (2019) introduziram um mecanismo adaptativo de aprendizado para estimar respostas de consultas agregadas em grandes volumes de dados, demonstrando a eficácia de técnicas de aprendizado de máquina na melhoria da responsividade de sistemas distribuídos. Dessa forma, é evidente que a escolha da estratégia de particionamento ideal exige uma análise cuidadosa das características dos dados, dos padrões de acesso e dos requisitos do sistema. Além disso, a utilização de abordagens híbridas, que combinam técnicas tradicionais e contemporâneas, surge como uma alternativa promissora para equilibrar desempenho, escalabilidade e adaptabilidade.

2.2 ARQUITETURAS E ESTRATÉGIAS DE ESCALABILIDADE EM SISTEMAS DE BANCO DE DADOS DISTRIBUÍDOS

À medida que o tráfego e o volume de dados aumentam, escalar servidores de banco de dados tradicionais, que geralmente envolvem a expansão da capacidade de um único servidor (escalabilidade vertical), torna-se cada vez mais complexo e dispendioso (Gunasekaran; Patel; Tirtiroglu, 2001).

Uma alternativa mais eficiente e amplamente adotada é a escalabilidade horizontal, que distribui a carga de trabalho entre vários servidores em um *cluster*, permitindo que múltiplos nós processem solicitações simultaneamente (DeWitt e Gray, 1992). Essa abordagem é baseada na arquitetura de bancos de dados distribuídos, na qual a carga é dividida entre os servidores do *cluster*, o que não só melhora o desempenho, mas também aumenta a disponibilidade e a tolerância a falhas do sistema (Stonebraker e Cattel, 2001).

No contexto atual, existem duas principais arquiteturas para escalar sistemas de banco de dados: os sistemas de banco de dados distribuídos e os sistemas de banco de dados paralelos.

2.2.1 Sistemas de banco de dados distribuídos

Sistemas de banco de dados distribuídos envolvem a distribuição de dados entre múltiplos locais físicos, que podem estar em diferentes geografias. De acordo com Elmasri e Navathe (2011), sistemas de banco de dados distribuídos são definidos como uma coleção de bancos de dados logicamente inter-relacionados distribuídos em uma rede de computadores, geridos por um sistema de gerenciamento de banco de dados distribuído que torna a distribuição transparente para o usuário.

Esses sistemas são projetados para reduzir a carga sobre cada servidor individual, melhorando o desempenho geral e a tolerância a falhas (Coulouris *et al*., 2013). A arquitetura distribuída permite que o processamento e o armazenamento sejam realizados de maneira descentralizada, aumentando a escalabilidade e a disponibilidade ao distribuir a carga de trabalho entre vários nós independentes (Hellerstein *et al*., 2019).

Para maximizar a capacidade de um *cluster* de bancos de dados distribuídos, é necessário escolher um modelo de distribuição apropriado, que pode ser replicação ou fragmentação (*sharding*). A replicação pode ser implementada de duas formas: mestre-escravo e ponto a ponto. No esquema mestre-escravo, um nó é responsável pelas atualizações de dados, enquanto um processo em segundo plano sincroniza os dados com os outros nós, chamados de escravos. Este modelo é recomendado para aplicações com alta taxa de leitura, pois permite aumentar a capacidade de leitura ao adicionar mais escravos. No entanto, a taxa de gravação é limitada pela capacidade do nó mestre (Ceri e Pelagatti, 1984). Em uma replicação ponto a ponto, todos os nós podem aceitar solicitações de gravação, o que melhora a capacidade do sistema de lidar com operações de gravação, mas pode introduzir problemas de consistência (Ceri e Pelagatti, 1984).

2.2.2 Sistemas de banco de dados paralelos

Em contraste, sistemas de banco de dados paralelos focam em processar dados simultaneamente utilizando múltiplos processadores ou núcleos dentro de um único local físico.

A arquitetura paralela é caracterizada pela execução de operações em paralelo em uma única máquina ou em um conjunto de máquinas homogêneas, com a vantagem de reduzir o tempo de processamento ao dividir tarefas entre vários processadores (Dean e Ghemawat, 2008).

Sistemas como o *MapReduce*, um paradigma paralelo desenvolvido pelo Google, exemplificam essa abordagem ao permitir que grandes volumes de dados sejam processados de forma eficiente através da divisão de trabalho em tarefas paralelas (Dean e Ghemawat, 2008).

A escolha entre uma arquitetura distribuída ou uma paralela depende das necessidades específicas do sistema e das características do ambiente de dados. Enquanto os sistemas distribuídos oferecem flexibilidade e escalabilidade geográfica, ideal para ambientes que requerem alta disponibilidade e resiliência, os sistemas paralelos são mais adequados para aplicações que exigem processamento intensivo em um único local (O'Neil e O'Neil, 2000). Ambos os modelos têm suas aplicações e desafios, e a escolha da arquitetura deve considerar fatores como a natureza dos dados, os requisitos de desempenho e a infraestrutura disponível.

2.2.3 Arquiteturas emergentes e tendências

As novas arquiteturas de bancos de dados têm evoluído para enfrentar os desafios da escalabilidade em um mundo de dados cada vez mais volumosos e dinâmicos. Uma inovação significativa é a arquitetura de bancos de dados NoSQL, que inclui uma variedade de modelos de dados como chave-valor, colunares, orientados a documentos e grafos. Os bancos de dados NoSQL são projetados para suportar escalabilidade horizontal com alta flexibilidade e desempenho para diferentes tipos de dados e cargas de trabalho. Essa abordagem permite que os dados sejam distribuídos e replicados através de múltiplos nós em um *cluster*, facilitando o crescimento contínuo sem comprometer o desempenho (Cattell, 2011).

Outra tendência emergente é a arquitetura de banco de dados baseado em nuvem, que oferece escalabilidade dinâmica e elasticidade através de provedores de serviços em nuvem. Plataformas como *Amazon Aurora* (AMAZON WEB SERVICES, 2025) e *Google Spanner* (GOOGLE CLOUD, 2025) exemplificam essa arquitetura ao fornecer soluções de banco de dados que escalam automaticamente conforme a demanda, otimizando o desempenho e o custo. Essas soluções aproveitam a infraestrutura de nuvem para distribuir a carga e armazenar dados de forma eficiente, garantindo alta disponibilidade e recuperação rápida em caso de falhas. A

capacidade de ajustar recursos em tempo real permite que as organizações respondam rapidamente às mudanças nas necessidades de dados.

Além disso, a arquitetura de banco de dados distribuído com consistência eventual é uma abordagem inovadora para lidar com a escalabilidade em sistemas de grande escala. Sistemas como o *Amazon DynamoDB* (AMAZON WEB SERVICES, 2025) e o *Apache Cassandra* (APACHE SOFTWARE FOUNDATION(a), 2025) adotam a consistência eventual para equilibrar a escalabilidade e a disponibilidade com a consistência dos dados. Essa arquitetura permite que o sistema continue operando eficientemente mesmo diante de falhas, ao mesmo tempo em que garante que todas as atualizações eventualmente sejam visíveis em todos os nós do sistema (Vogels, 2009; Lakshman e Malik, 2010). Esse modelo é particularmente útil em aplicações de alta disponibilidade e escalabilidade, onde a consistência imediata não é crítica, mas a disponibilidade e a resiliência são prioridades.

2.2.4 Data sharding

Com a evolução das arquiteturas de bancos de dados surge uma técnica fundamental para o particionamento de dados em sistemas distribuídos, *Data Sharding*, projetada para melhorar a escalabilidade e o desempenho de bancos de dados que lidam com grandes volumes de dados. A estratégia de *sharding* divide um banco de dados em partes menores e mais manejáveis chamadas "*shards*", cada uma armazenada em um servidor ou nó separado. Esta abordagem permite que diferentes *shards* sejam processados em paralelo, reduzindo a carga em qualquer servidor único e possibilitando uma escalabilidade horizontal eficiente (Solat *et al*., 2024). A distribuição dos dados em *shards* é geralmente baseada em uma chave de particionamento, como por exemplo um identificador de cliente ou um intervalo de dados, que determina como os dados são distribuídos entre os *shards*.

Uma das principais vantagens do data *sharding* é a escalabilidade horizontal, que permite a expansão do sistema através da adição de novos *shards* e nós conforme a demanda aumenta. Em vez de aumentar a capacidade de um único servidor (escalabilidade vertical), a adição de novos shards distribui a carga de trabalho e o armazenamento de dados de forma mais equilibrada, proporcionando um desempenho aprimorado e uma maior capacidade de processamento (Solat, 2024). Além disso, o *sharding* pode melhorar a disponibilidade e a

resiliência do sistema, já que a falha de um *shard* não afeta diretamente os outros, e a replicação entre *shards* pode ser configurada para garantir a continuidade do serviço (Cattell, 2011).

No entanto, a implementação de data *sharding* apresenta desafios significativos. A gestão e coordenação de múltiplos *shards* podem ser complexas, exigindo soluções eficazes para garantir a consistência e o balanceamento da carga entre eles. A escolha adequada da chave de particionamento é crucial para evitar a formação de hotspots, onde alguns *shards* podem se tornar sobrecarregados enquanto outros permanecem subutilizados (Sadalage e Fowler, 2013). Além disso, a manutenção da consistência dos dados entre s*hards* e a execução de transações distribuídas requerem mecanismos adicionais para assegurar a integridade do sistema.

Um exemplo prático de *data sharding* pode ser observado em MongoDB, um banco de dados NoSQL que utiliza o *sharding* para gerenciar grandes volumes de dados. O MongoDB divide dados de coleções em *shards* e distribui-os por múltiplos servidores. Cada *shard* contém uma parte do banco de dados, e as consultas são roteadas para os *shards* apropriados, otimizando o desempenho e a escalabilidade (Kleppmann, 2017). A aplicação de *sharding* no MongoDB permite que ele lide eficientemente com cargas de trabalho pesadas e grandes conjuntos de dados, mantendo um desempenho elevado e uma alta disponibilidade.

2.3 MINERAÇÃO DE PROCESSOS

A Mineração de Processos (*Process Mining*) é uma disciplina emergente que combina técnicas de mineração de dados e modelagem de processos para descobrir, monitorar e melhorar processos organizacionais com base em dados reais. Conforme destacado por Van der Aalst (2016), a Mineração de Processos se fundamenta na análise de registros de eventos (*Event Logs*) para construir modelos precisos de processos reais, possibilitando não apenas a visualização dos fluxos de trabalho, mas também a identificação de desvios e ineficiências. Essas descobertas permitem que as organizações otimizem suas operações, alinhando seus processos às metas estratégicas e aos requisitos do negócio.

Na visão de Mehr (2024), a Mineração de Processos é dependente, principalmente, do conhecimento extraído de *logs* de eventos (*Event Logs*) de sistemas de informação. Onde muitas vezes existe uma lacuna entre um processo modelado e as tendências reais observadas durante a execução deste processo. O objetivo da mineração de processos é identificar e mitigar questões decorrentes desses desvios.

Este campo pode ser dividido em três principais categorias: descoberta de processos, conformidade e aprimoramento de processos. A descoberta de processos envolve a criação de modelos de processos a partir de dados de eventos, enquanto a conformidade analisa se os processos estão sendo seguidos conforme as regras estabelecidas. O aprimoramento de processos foca na identificação de áreas para melhoria com base na análise dos dados (van der Aalst, 2016).

As estratégias de particionamento de dados desempenham um papel crucial no desempenho das aplicações de Mineração de Processos, especialmente ao lidar com grandes volumes de dados gerados em ambientes organizacionais. Uma abordagem eficaz de particionamento permite que os dados sejam distribuídos entre múltiplos nós de processamento, facilitando a análise paralela e reduzindo o tempo de resposta para a extração de insights. Quando os dados são bem particionados, as operações de mineração de processos, como a descoberta e o monitoramento, podem ser realizadas de forma mais eficiente, minimizando a latência e melhorando a escalabilidade das soluções (Han; Pei; Kamber, 2011). Além disso, boas estratégias de particionamento podem contribuir para uma distribuição equilibrada da carga de trabalho, o que é essencial para manter a performance em sistemas de Process Mining que exigem processamento contínuo de dados.

Portanto, a adoção de estratégias de particionamento de dados adequadas não só otimiza o desempenho das aplicações de Mineração de Processos, mas também permite que as organizações tirem pleno proveito dos insights derivados de seus dados operacionais. Com a implementação de técnicas robustas de particionamento, as empresas podem melhorar a eficiência de suas operações e a eficácia de suas decisões, utilizando a mineração de processos como uma ferramenta para transformação organizacional (Van der Aalst., 2016).

2.3.1 Mineração de Processos no contexto do Poder Judiciário brasileiro

No contexto do Poder Judiciário Brasileiro, o CNJ (Conselho Nacional de Justiça), em parceria com outros Tribunais, deram um passo de grande importância que foi a criação da Plataforma Digital do Poder Judiciário Brasileiro (PDPJ-Br), uma rede com mais de 90 tribunais e conselhos para apoiar no desenvolvimento de soluções inovadoras e na integração dos tribunais com demais organizações que fazem parte do ecossistema judicial, com o objetivo de melhorar a eficiência dos tribunais brasileiros na gestão dos processos judiciais (Conjur, 2024).

Dentre as ferramentas, três delas se destacam para o propósito desta pesquisa, o PJ-e, CODEX e o JuMP.

2.3.2 Processo Judicial Eletrônico

O Processo Judicial Eletrônico (PJ-e), uma plataforma digital desenvolvida pelo CNJ em parceria com diversos Tribunais que objetiva, portanto, otimizar o fluxo processual e garantir a eficiência e acessibilidade no sistema judiciário por meio da conversão de esforços na a adoção de solução padronizada e gratuita aos Tribunais em âmbito nacional, e considerando características inerentes a cada ramo da Justiça (CNJ, 2024).

Em linhas gerais, o manual do PJ-e sustentado pela Lei nº 11.419/2006, que trata da informatização do processo judicial, descreve que o ciclo de vida de um processo judicial eletrônico compreende diversas fases e a atuação de vários atores. Inicia-se com o ajuizamento da ação, onde o peticionante apresenta a demanda e os documentos pertinentes ao sistema eletrônico. O processo, composto por movimentações, petições, documentos, despachos, decisões e sentenças, é gerido digitalmente, o que permite acesso remoto e simultâneo às partes envolvidas. Os principais atores são o autor da ação, o réu, os advogados, o juiz e os servidores do fórum. Após o ajuizamento, o processo é distribuído ao juiz responsável, que realiza a tramitação, incluindo a análise dos documentos, audiências e julgamentos. A tramitação é acompanhada em tempo real pelas partes e seus representantes, proporcionando maior transparência e agilidade.

Nesse contexto, os seguintes conceitos são essenciais:

2.3.2.1 Natureza

A natureza de um processo judicial eletrônico refere-se à sua classificação quanto ao tipo de ação que está sendo ajuizada. Ela define o propósito e o contexto jurídico do processo, como ações civis, criminais, trabalhistas ou administrativas. Essa classificação é fundamental para a correta tramitação e direcionamento dos processos dentro do sistema judiciário, possibilitando a aplicação adequada das normas e procedimentos específicos a cada tipo de ação (Resolução CNJ nº 185/2013).

2.3.2.2 Classes

As classes dos processos são categorias amplas que agrupam processos com características semelhantes, facilitando a organização e o gerenciamento dos mesmos dentro do PJe. Exemplos de classes incluem "Ação de Conhecimento", "Execução", "Mandado de Segurança", entre outras. A definição de classes ajuda a determinar o tipo de procedimento aplicável e as regras de tramitação, além de auxiliar na alocação de recursos e na análise estatística dos casos (BRASIL, Lei nº 11.419/2006).

2.3.2.3 Assuntos

Os assuntos referem-se às questões específicas ou temas tratados dentro de cada classe processual. Eles detalham o objeto do litígio e a natureza da demanda, como "Dívida Ativa", "Alimentos", ou "Indenização por Danos Morais". A especificação dos assuntos é crucial para a identificação precisa do tema do processo e para a correta aplicação das leis e regulamentações pertinentes (Resolução CNJ nº 185/2013).

2.3.2.4 Partes

As partes de um processo são os indivíduos ou entidades envolvidas no litígio. No PJe, cada processo deve identificar claramente as partes, que podem ser autor, réu, intervenientes ou terceiros interessados. A correta identificação e registro das partes são essenciais para garantir a comunicação adequada, a citação e a defesa, além de possibilitar o controle do andamento processual e a execução das decisões judiciais (BRASIL, Lei nº 11.419/2006).

2.3.2.5 Movimentos

Os movimentos representam as ações realizadas ao longo do processo, como a juntada de documentos, a realização de audiências, ou a intimação de partes. Cada movimento é registrado no PJe para assegurar o acompanhamento detalhado do andamento processual, garantir a transparência e a integridade dos atos processuais, e possibilitar o controle efetivo do fluxo de trabalho do processo (Resolução CNJ nº 185/2013).

2.3.2.6 Tarefas

As tarefas no PJe são as atividades específicas atribuídas aos diferentes atores do processo, como advogados, juízes e servidores. Elas podem incluir a elaboração de petições, a análise de documentos, a emissão de decisões, entre outras. A gestão eficiente das tarefas é fundamental para assegurar que todas as etapas do processo sejam realizadas de acordo com os prazos e normas estabelecidos, contribuindo para a agilidade e eficiência do trâmite processual (BRASIL, Lei nº 11.419/2006).

2.3.2.7 Documentos

Os documentos referem-se a qualquer tipo de arquivo digital que é anexado ao processo para registrar informações relevantes e necessárias para a tramitação do mesmo. Esses documentos podem incluir petições, provas, relatórios, decisões, pareceres, certidões e outros papéis essenciais que compõem o conteúdo do processo judicial. Eles são fundamentais para a validade e a integridade do processo judicial e são utilizados para formalizar a apresentação de alegações, a submissão de provas, e a comunicação de atos processuais entre as partes e o Judiciário. Cada documento anexado deve seguir os requisitos legais e regulamentares quanto à sua forma e conteúdo, garantindo que todas as informações relevantes sejam corretamente registradas e disponibilizadas para análise dos envolvidos e do juiz responsável (Resolução CNJ nº 185/2013).

2.3.3 CODEX

O CODEX é uma plataforma desenvolvida pelo Tribunal de Justiça de Rondônia em colaboração com o Conselho Nacional de Justiça. Esta ferramenta serve como um repositório centralizado de dados processuais, reunindo informações textuais e também dados estruturados provenientes do sistema do Processo Judicial Eletrônico (PJ-e). Ao operar como um data lake, o CODEX facilita o acesso a dados cruciais para a análise e a tomada de decisões, permitindo a produção de painéis de inteligência de negócios, pesquisas unificadas e a alimentação automatizada de estatísticas (CNJ, 2022). Com a consolidação desses dados, o CODEX tem o

potencial de transformar a gestão do Judiciário, promovendo uma abordagem de gestão orientada por dados.

2.3.4 JuMP

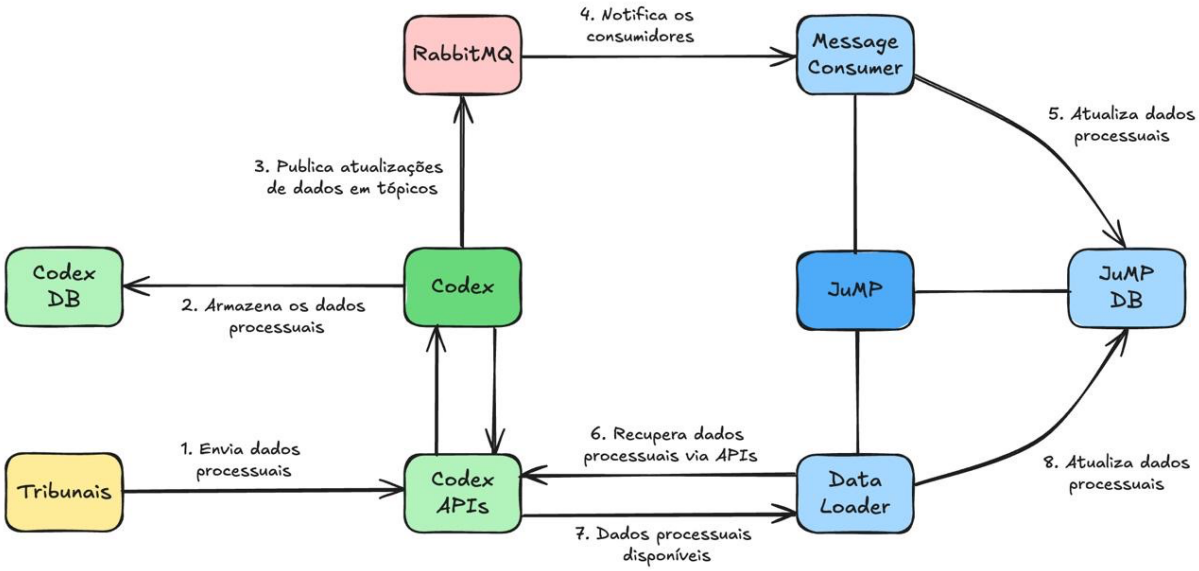
O Judiciário com Mineração de Processos (JuMP) constitui uma abordagem inovadora, desenvolvida através de uma colaboração entre o V-Lab, laboratório da UFPE e o CNJ. Em operação desde dezembro de 2022, o JuMP está integrado à Plataforma Digital do Poder Judiciário Brasileiro (PDPJ-Br), tornando-se uma ferramenta acessível aos membros dos tribunais brasileiros (Lima *et al*., 2023).

O JuMP emprega técnicas avançadas de mineração de processos para fornecer *insights* analíticos sobre os fluxos processuais, facilitando a identificação de gargalos e promovendo a otimização da gestão judicial. Esse sistema, ao ser integrado à infraestrutura digital do Judiciário, busca aprimorar a eficiência operacional e contribuir para a agilização do trâmite processual, com vistas a um serviço jurisdicional mais eficaz e transparente (CNJ 2021).

A aplicação de Mineração de Processos no contexto do Poder Judiciário Brasileiro é fundamental para a transformação digital do sistema judiciário. Essa abordagem utiliza técnicas analíticas para extrair informações valiosas a partir dos dados processuais, permitindo uma visualização clara dos fluxos de trabalho e dos pontos de estrangulamento. Ao identificar ineficiências e inconsistências, a Mineração de Processos oferece subsídios para que os gestores possam implementar melhorias direcionadas, aumentando a transparência e a *accountability* no Judiciário (Lino Júnior 2023). Essa prática não apenas melhora a gestão interna, mas também reforça a confiança da população no sistema de justiça.

Em um ambiente onde a morosidade é um dos principais desafios, a capacidade de visualizar e analisar dados em tempo real possibilita decisões mais rápidas e informadas. Como destacado por Porto (2022), essa abordagem não só identifica problemas, mas também permite que boas práticas sejam disseminadas entre diferentes unidades judiciárias, elevando a qualidade dos serviços prestados. A figura Error: Reference source not found apresenta uma visão geral da arquitetura da solução:

Figura 1 – Visão geral da arquitetura da solução JuMP.



Fonte: O autor (2025).

3 METODOLOGIA

O estudo investigou a adaptação das abordagens de particionamento frente ao crescimento do volume de dados e ao aumento das requisições, analisando o desempenho dessas soluções em termos de escalabilidade em ambientes dinâmicos e de alta demanda. O objetivo foi avaliar a resiliência e a capacidade de adaptação dessas estratégias no JuMP, um sistema de processamento de dados judiciais, em cenários nos quais o volume e a complexidade dos dados evoluem constantemente.

A metodologia experimental foi projetada para realizar uma análise comparativa do desempenho de diferentes estratégias de particionamento de dados: por Chave, por Intervalo, Hash e Híbridas (Intervalo + Lista e Intervalo + Hash). A análise foi orientada pelo principal cenário de uso do sistema, que envolve a recuperação de registros de movimentações processuais de uma unidade judiciária ao longo dos anos.

Os experimentos foram organizados em dois componentes principais: o primeiro focou na simulação de um ambiente com grandes volumes de dados, enquanto o segundo simulou diferentes cenários de carga de usuários, com o objetivo de avaliar o desempenho do sistema à medida que o número de usuários simultâneos crescia. O objetivo central foi analisar como cada estratégia de particionamento de dados responde a variáveis críticas, tais como: tempo de resposta, utilização de recursos computacionais (CPU e memória), escalabilidade, equilíbrio de carga, taxa de transferência de dados, custo de redistribuição e eficiência de consultas. Essa abordagem permitiu uma avaliação detalhada do impacto de cada estratégia em cenários de consulta concorrente de dados e alta demanda.

A figura Error: Reference source not found ilustra as etapas realizadas nessa metodologia para execução dos experimentos, desde a sua preparação até a coleta dos resultados.

Fonte: o autor (2025).

3.1 COLETA DE DADOS

A coleta de dados foi realizada a partir de um backup da base de dados do JuMP, contendo uma amostra representativa dos dados utilizados pela ferramenta. Esta base incluiu registros processuais, movimentações, documentos e metadados associados, como informações sobre classes processuais, assuntos, unidade judiciária e tribunal. Os dados, fornecidos pela equipe do Centro de Informática (CIn) da Universidade Federal de Pernambuco (UFPE), foram previamente anonimizados para garantir a proteção da privacidade, sendo destinados exclusivamente para esta pesquisa. Além da base de dados mencionada, a equipe do Centro de Informática da UFPE também forneceu a consulta SQL (*Structured Query Language*) mais utilizada pelo JuMP, que apresenta o tempo de resposta mais elevado.

3.1.1 Banco de Dados

O backup da base de dados do JuMP foi restaurado no PostgreSQL 16.2, sistema de gerenciamento de banco de dados utilizado pelo JuMP. Para a realização dos experimentos, foram ajustados os parâmetros de configuração do PostgreSQL, permitindo a execução de consultas SQL (*queries*) paralelas e de longa duração. No entanto, é importante ressaltar que o mesmo banco de dados, com os mesmos parâmetros de configuração e a mesma configuração de recursos computacionais (CPU e Memória RAM), foi utilizado em todos os experimentos, garantindo a consistência das condições de avaliação.

3.2 EXPERIMENTOS

Nesta seção, será descrita a metodologia geral aplicada nos quatro experimentos, com o objetivo de avaliar as estratégias de particionamento adotadas: por Chave, por Intervalo, *Hash* e Híbridas (Intervalo + Lista e Intervalo + *Hash*). Inicialmente, serão apresentados o conjunto comum de procedimentos e métricas utilizados para analisar a eficácia das diferentes estratégias de particionamento de dados, considerando aspectos como desempenho, escalabilidade e

eficiência operacional do sistema. Em seguida, serão detalhadas as especificidades da metodologia aplicada a cada uma das estratégias de particionamento.

3.2.1 Ambiente utilizado para os experimentos

Para realização dos experimentos foi utilizada uma infraestrutura local baseada em *Docker Containers*. Essa abordagem adotada visou garantir o controle sobre os parâmetros de concorrência, disponibilidade de recursos e desempenho.

O mesmo equipamento host foi utilizado para simular o ambiente de banco de dados do JuMP e a execução dos cenários de simulação de carga com *JMeter*. As configurações de hardware do equipamento host utilizado são as seguintes:

Modelo: MacBook Pro
Processador: Apple M2 Max
Memória RAM: 32 GB
Armazenamento: SSD de 1 TB
Sistema Operacional: Sonoma 14.6.1

Os recursos de CPU e memória do *Docker Container* que executou o banco de dados PostgreSQL foram limitados para estabelecer uma base de comparação entre as estratégias de particionamento (Figura Error: Reference source not found). As reservas e limites de CPU e memória definidos estão disponíveis no arquivo *docker-compose* disponível em material complementar elaborado pelo próprio autor (MOURA, 2025).

Esses parâmetros garantiram que o ambiente de execução fosse controlado e consistente ao longo dos experimentos, permitindo uma avaliação justa do impacto de cada estratégia de particionamento no desempenho do sistema.

Figura 3 – Limites de recursos no *docker-compose*.

```yaml
services:

  postgres:
    image: postgres:16.2
    shm_size: "8g" # sets the size of the shared memory
    sysctls:
      kernel.shmmax: 8589934592
      kernel.shmall: 2097152
    deploy:
      resources:
        limits:
          cpus: "4.0"
          memory: "12g"
        reservations:
          cpus: "4.0"
          memory: "12g"
```

Fonte: O autor (2025).

3.2.2 Configurações e Execução do Banco de Dados

Para que fosse possível realizar os experimentos, foram realizadas alterações nos seguintes parâmetros de configuração do PostgreSQL (Figura Error: Reference source not found), a fim de permitir a execução de *queries* paralelas e de longa duração. Os parâmetros abaixo estavam configurados no arquivo *postgresql.conf*, disponível em material complementar elaborado pelo próprio autor (MOURA, 2025).

Figura 4 – Parâmetros de configuração do PostgreSQL.

```
max_connections = 200

shared_buffers = 1GB

work_mem = 32MB
maintenance_work_mem = 128MB

max_worker_processes = 8
max_parallel_workers_per_gather = 2
max_parallel_workers = 4

enable_partitionwise_join = on

effective_cache_size = 4GB

statement_timeout = 30000
lock_timeout = 15000
idle_in_transaction_session_timeout = 60000
idle_session_timeout = 60000
```

Fonte: O autor (2025).

Como mencionado, o banco de dados foi executado em um *Docker Container*, e para isso foi criado um arquivo *docker-compose.yml*, que possibilitou a inicialização da base de dados do JuMP fornecida de forma simples e rápida.

3.2.3 Ferramenta para execução dos experimentos e coleta dos dados

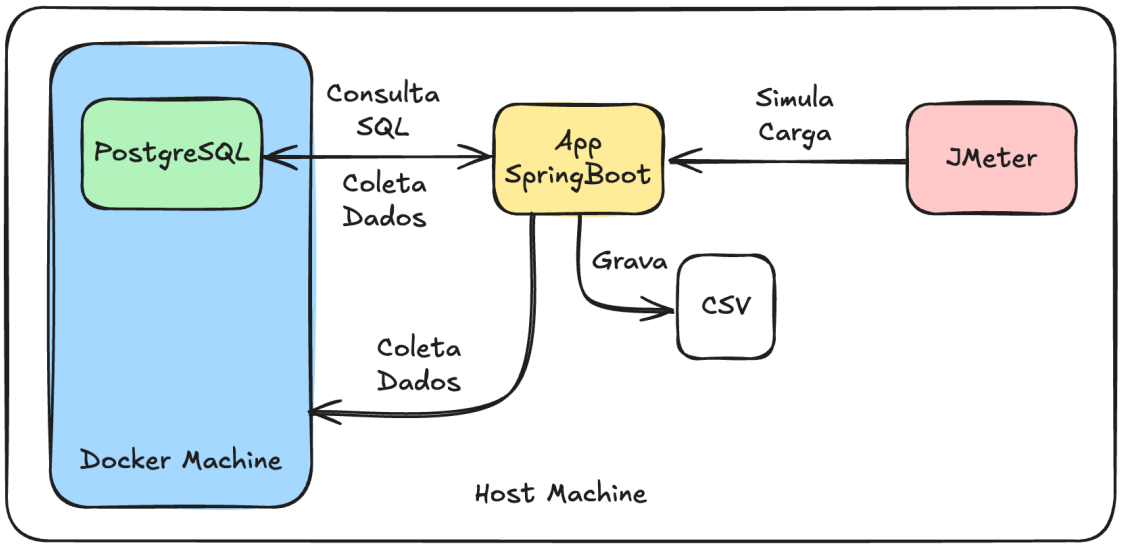
Com o objetivo de viabilizar a execução dos cenários de teste que simulavam a carga de usuários e facilitar a coleta de dados das métricas definidas, foi implementada uma aplicação em *Java* utilizando o *framework Spring Boot*. Esta aplicação gerenciou o *pool* de conexões com o banco de dados *PostgreSQL*, utilizado para a execução das consultas SQL. Dentro da aplicação, foram implementadas APIs REST, que foram consumidas pelo *JMeter* e responsáveis por disparar a execução das consultas SQL no banco de dados.

É importante destacar que o código-fonte e os recursos utilizados neste projeto estão disponíveis em repositório GIT como material complementar elaborado pelo próprio autor (MOURA, 2025).

Durante a execução das consultas, a aplicação coletou as métricas relevantes para a comparação de cada estratégia e as armazenou em um arquivo CSV. Cada linha do arquivo representava uma execução de consulta SQL, e as colunas, separadas por vírgula, continham dados importantes para as métricas, como: número de usuários, tempo de resposta, status da execução (SUCESSO ou FALHA), uso de CPU, uso de memória, entre outros. Além disso, foi desenvolvida uma integração com a Docker Engine API, permitindo a coleta do consumo instantâneo de CPU e memória, métricas essenciais para este estudo.

A Figura Error: Reference source not found demonstra uma visão geral de como os componentes da solução para a execução dos experimentos interagiram.

Figura 5 – Visão geral da solução para execução dos experimentos.



Fonte: O autor (2025).

3.2.4 Cenários de teste

Para a simulação das cargas de execução, foi utilizada a ferramenta Apache JMeter (versão 5.6.3), com o objetivo de criar um plano de testes que permitisse simular diferentes cenários de carga de usuários interagindo com a aplicação. O *JMeter,* (APACHE SOFTWARE FOUNDATION (b), 2023) possibilitou a configuração e emulação de múltiplos usuários simultâneos, criando um ambiente controlado para avaliar o desempenho da aplicação sob diferentes intensidades de requisições. Isso permitiu realizar uma análise detalhada da escalabilidade e capacidade de resposta do sistema.

Os cenários de teste foram definidos conforme o número de usuários e a duração das requisições sequenciais realizadas, conforme descrito abaixo:

Cenário 01: Um usuário realiza requisições sequenciais durante 30 segundos.

Cenário 02: Dois usuários realizam requisições sequenciais por 30 segundos.

Cenário 03: Três usuários realizam requisições sequenciais por 30 segundos.

Cenário 04: Cinco usuários realizam requisições sequenciais por 30 segundos.

Cenário 05: Oito usuários realizam requisições sequenciais por 60 segundos.

Cenário 06: Treze usuários realizam requisições sequenciais por 60 segundos.

Cenário 07: Vinte e um usuários realizam requisições sequenciais por 60 segundos.

Cenário 08: Trinta e quatro usuários realizam requisições sequenciais por 90 segundos.

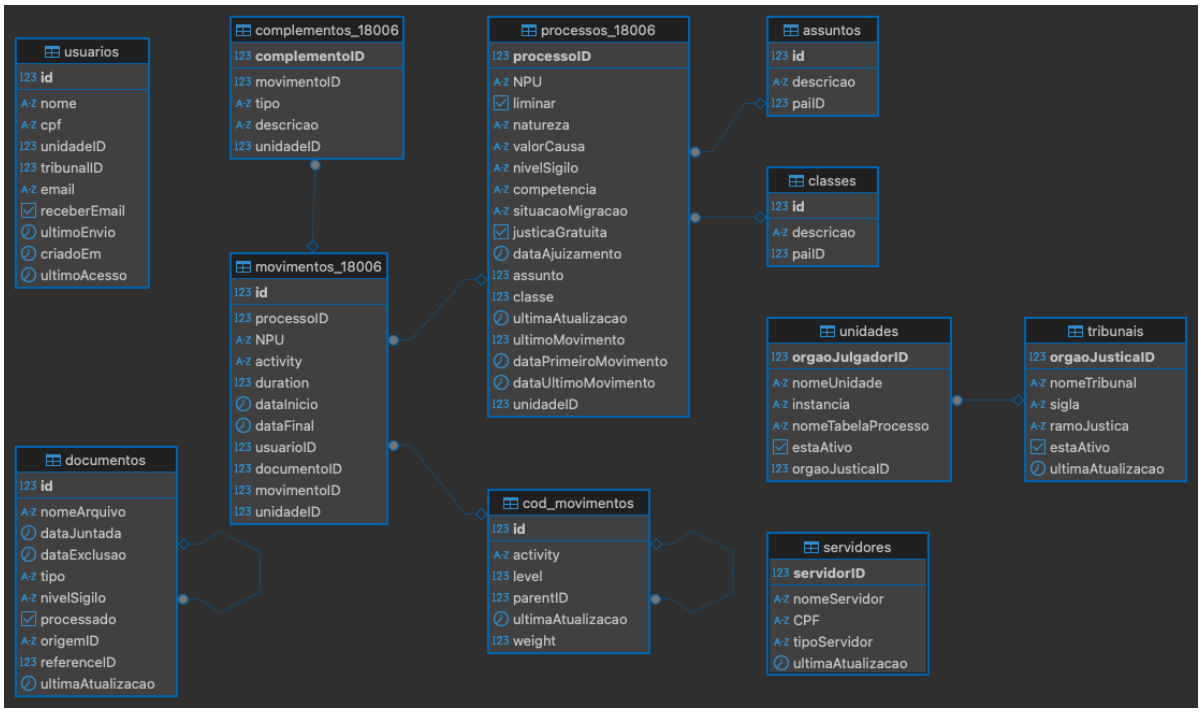
Cenário 09: Cinquenta e cinco usuários realizam requisições sequenciais por 120 segundos.

Esses cenários foram projetados para testar o desempenho do sistema em cargas crescentes de usuários simultâneos e tempos de execução.

3.2.5 Experimento controle

Este experimento serviu como base para a comparação com as demais abordagens, representando o ponto de partida para a análise da eficácia das estratégias de particionamento. Foram coletados os resultados das métricas previamente estabelecidas, que permitiram uma avaliação comparativa da estratégia de particionamento utilizada no sistema JuMP. Contudo, é importante ressaltar que, neste cenário, não houve intervenção na estratégia de particionamento, no modelo de dados ou na arquitetura do sistema, de forma a garantir uma avaliação precisa da situação atual.

Figura 6 – Modelo de dados atual do JuMP.



Fonte: O autor (2025).

No modelo de dados do JuMP, apresentado na figura Error: Reference source not found, é possível perceber que a estratégia de particionamento adotada foi o particionamento por Chave, aplicada em nível físico das tabelas, no qual, para cada Unidade Judiciária, eram criadas novas tabelas para os dados de processos, movimentos e complementos. Cada uma dessas tabelas recebia um sufixo correspondente ao identificador da respectiva unidade judiciária. Por exemplo, para um tribunal que possuía uma unidade judiciária com o ID 1000, eram geradas as tabelas: *processos_1000*, *movimentos_1000* e *complementos_1000*. Esse modelo de particionamento visava isolar os dados por unidade, mas mantinha a estrutura atual de tabelas independentes para cada unidade judiciária, sem qualquer modificação ou alteração no processo de particionamento.

3.2.5.1 Preparação

A base de dados que foi fornecida precisou ser enriquecida com mais dados. Para isso, foi realizado o clone da estrutura e dos registros das tabelas processos_18006, movimentos_18006 e complementos_18006, e efetuadas as devidas correções nas referências de chaves primárias e estrangeiras. Esse processo teve como objetivo viabilizar a simulação de carga de usuários realizando consultas de dados de diferentes unidades judiciárias.

A documentação disponível no material complementar (MOURA, 2025) referente a este experimento, descreve todas as etapas e comandos utilizados para realizar o enriquecimento dos dados mencionado.

3.2.5.2 Consulta SQL de referência

O comando descrito no quadro Error: Reference source not found se trata da consulta SQL de referência fornecida pela equipe do Centro de Informática da UFPE, como exemplo da consulta mais onerosa que é utilizada pelo JuMP para leitura e manipulação dos dados. Apesar não estar implementada uma estratégia de particionamento nativa do banco de dados, é possível perceber, que esta consulta faz uso de tabelas “particionadas” por unidade judiciária, com o objetivo de otimizar a performance.

Quadro 1 – Consulta SQL de referência.

```
SELECT
   p."NPU",
   p."processoID",
   p."ultimaAtualizacao",
   c.descricao AS classe,
   a.descricao AS assunto,
   m.activity,
   m."dataInicio",     m."dataFinal",
   m."usuarioID",     m.duration,
   m."movimentoID",
   com.descricao AS complemento,
   s."nomeServidor",     s."tipoServidor",
   d.tipo AS documento
FROM
   processos_18006 AS p
INNER JOIN
   movimentos_18006 AS m
   ON m."processoID" = p."processoID"
INNER JOIN
   classes AS c ON p.classe = c.id
LEFT JOIN
   assuntos AS a ON p.assunto = a.id
LEFT JOIN
   complementos_18006 AS com
   ON com."movimentoID" = m."id"
LEFT JOIN
   servidores AS s ON s."servidorID" = m."usuarioID"
LEFT JOIN
   documentos AS d ON d."id" = m."documentoID"
WHERE
   p."dataPrimeiroMovimento" >= '2020-01-01'
ORDER BY
   p."processoID", m."dataFinal";
```

Fonte: O autor (2025).

3.2.6 Experimento 01 – Particionamento Por Intervalo

3.2.6.1 Preparação dos dados

Nesta etapa, a base de dados foi preparada para o particionamento das tabelas por intervalo, partindo da premissa de que a etapa de preparação do Experimento de Controle já havia sido realizada.

3.2.6.2 Análise da distribuição dos dados

Para a realização do particionamento por Intervalo, primeiramente foi realizada uma análise da distribuição dos dados ao longo dos anos, com o objetivo de definir a quantidade de partições. No quadro Error: Reference source not found destaca-se a consulta SQL que retorna como os processos estão distribuídos em cada ano, e em seguida, o seu resultado (Tabela Error: Reference source not found) que permite constatar que os processos estão distribuídos ao longo de 13 anos, ou seja, 13 partições.

Quadro 2 – Consulta SQL da distribuição de processos por ano.

```
SELECT
    DATE_TRUNC('year', "dataPrimeiroMovimento") as ano,
    count(*) as qtd_processos
FROM processos_18006
GROUP BY DATE_TRUNC('year', "dataPrimeiroMovimento")
ORDER BY DATE_TRUNC('year', "dataPrimeiroMovimento");
```

Fonte: O autor (2025).

Tabela 1 – Processos distribuídos ao longo de 13 anos em 13 partições.

Ano	**Qtd_processos**
2013-01-01 00:00:00	5324
2014-01-01 00:00:00	465
2015-01-01 00:00:00	25329
2016-01-01 00:00:00	76213
2017-01-01 00:00:00	11179
2018-01-01 00:00:00	76895
2019-01-01 00:00:00	17404
2020-01-01 00:00:00	16263
2021-01-01 00:00:00	25782
2022-01-01 00:00:00	50124
2023-01-01 00:00:00	30581
2024-01-01 00:00:00	14876
2025-01-01 00:00:00	2
NULL	60

Fonte: O autor (2025).

3.2.6.3 Criação das tabelas com o Particionamento por Intervalo

Nesta etapa, foram criadas as tabelas processos_exp01, movimentos_exp01 e complementos_exp01, com o particionamento por Intervalo ativado. Cada uma delas foi configurada com 13 partições, uma para cada ano. Para isso, foi utilizada a técnica de particionamento do PostgreSQL chamada *Range Partitioning*, aplicada à coluna *dataPrimeiroMovimento.*

Os comandos necessários para a criação das referidas tabelas estão descritos na documentação disponível no material complementar (MOURA, 2025).

3.2.6.4 Migração de dados

Para que as novas tabelas fossem devidamente populadas com os dados presentes nas tabelas originais, foi necessário executar scripts de migração de dados. Esses scripts possibilitaram a redistribuição dos dados, que anteriormente estavam armazenados em uma única tabela, para as partições correspondentes, com base no ano especificado na coluna *dataPrimeiroMovimento*.

Os detalhes sobre os comandos utilizados para a migração dos dados e a criação das tabelas podem ser consultados na documentação disponível no material complementar (MOURA, 2025).

3.2.6.5 Consulta SQL de referência

A consulta SQL foi ajustada para referenciar as novas tabelas. Conforme descrito no quadro Error: Reference source not found.

Quadro 3 – Consulta SQL do experimento 01.

```
SELECT
   p."NPU",
   p."processoID",
   p."ultimaAtualizacao",
   c.descricao AS classe,
   a.descricao AS assunto,
   m.activity,
   m."dataInicio",
   m."dataFinal",
   m."usuarioID",
   m.duration,
   m."movimentoID",
   com.descricao AS complemento,
   s."nomeServidor",
   s."tipoServidor",
   d.tipo AS documento
FROM
   processos_exp01 AS p
INNER JOIN
   movimentos_exp01 AS m
   ON m."processoID" = p."processoID"
INNER JOIN
   classes AS c ON p.classe = c.id
LEFT JOIN
   assuntos AS a ON p.assunto = a.id
LEFT JOIN
   complementos_exp01 AS com
   ON com."movimentoID" = m."id"
LEFT JOIN
   servidores AS s ON s."servidorID" = m."usuarioID"
LEFT JOIN
   documentos AS d ON d."id" = m."documentoID"
WHERE
   p."dataPrimeiroMovimento" >= '2020-01-01'AND p."unidadeID" = 18006
      AND m."dataPrimeiroMovimento" >= '2020-01-01' AND m."unidadeID" = 18006
      AND com."dataPrimeiroMovimento" >= '2020-01-01' AND com."unidadeID" = 18006
ORDER BY
   p."processoID", m."dataFinal";
```

Fonte: O autor (2025).

3.2.7 Experimento 02 – Particionamento Por *Hash*

3.2.7.1 Preparação dos dados

Nesta etapa, a base de dados foi preparada para o particionamento das tabelas por intervalo, partindo da premissa de que a etapa de preparação do Experimento de Controle já havia sido realizada.

3.2.7.2 Definição da função *Hash*

As partições das tabelas processos, movimentos e complementos foram configuradas com base no resultado de uma função *hash*, que utilizou como entradas os valores das colunas unidadeID e anoPrimeiroMovimento. Entretanto, o particionamento por *Hash* não garantiu que todos os registros dentro de uma partição fossem associados à mesma unidade judiciária ou ao mesmo ano de primeiro movimento. Em vez disso, o particionamento foi realizado de forma pseudo-aleatória, distribuindo os dados entre as partições com base no cálculo da função *hash*. A função *hash* combinou os dois valores de entrada, conforme exemplificado a seguir:

hash (18006, 2023) % 39 resultou em 3.
hash (25000, 2020) % 39 também resultou em 3.

Desse modo, ambos os registros foram alocados na partição *tabela_particao_3*, evidenciando que dados referentes a diferentes unidades judiciais e anos puderam ser armazenados na mesma partição. Esse tipo de distribuição teve como objetivo equilibrar o armazenamento nas partições, sem garantir o agrupamento lógico dos dados, característica presente em estratégias de particionamento por Lista (*List Partitioning*) ou por intervalo (*Range Partitioning*) que asseguram que registros de uma mesma categoria sejam armazenados de forma contígua. Ao contrário, o particionamento por *Hash* distribuiu os dados de maneira uniforme entre todas as partições disponíveis.

3.2.7.3 Definição da quantidade de partições

Considerando que a base de dados continha registros de três unidades judiciárias distintas, com processos distribuídos ao longo de 13 anos, o experimento adotou a criação de 13 partições, visando distribuir os processos de maneira equilibrada entre as unidades judiciárias. Cada partição foi configurada para comportar os registros das diferentes unidades, de forma a garantir uma distribuição homogênea de dados ao longo do tempo.

Os comandos necessários para a criação das referidas tabelas estão descritos na documentação disponível no material complementar (MOURA, 2025).

3.2.7.4 Migração de dados

Para que as novas tabelas fossem devidamente populadas com os dados existentes nas tabelas originais, foi necessário executar scripts de migração de dados. Esses scripts permitiram que os dados, anteriormente armazenados em uma única tabela, fossem redistribuídos para diferentes partições, conforme o cálculo da função *hash*, que utilizou como parâmetros as colunas *unidadeID* e *anoPrimeiroMovimento*.

Os comandos necessários para a criação e migração das referidas tabelas estão descritos na documentação disponível no material complementar (MOURA, 2025).

3.2.7.5 Consulta SQL de referência

A consulta SQL foi ajustada para referenciar as novas tabelas. Conforme descrito no quadro Error: Reference source not found.

Quadro 4 – Consulta SQL do experimento 02.

```
SELECT
   p."NPU",
   p."processoID",
   p."ultimaAtualizacao",
   c.descricao AS classe,
   a.descricao AS assunto,
   m.activity,
   m."dataInicio",
   m."dataFinal",
   m."usuarioID",
   m.duration,
   m."movimentoID",
   com.descricao AS complemento,
   s."nomeServidor",
   s."tipoServidor",
   d.tipo AS documento
FROM
   processos_exp02 AS p
INNER JOIN
   movimentos_exp02 AS m
   ON m."anoPrimeiroMovimento" = p."anoPrimeiroMovimento"
        AND m."unidadeID" = p."unidadeID"
        AND m."processoID" = p."processoID"
INNER JOIN
   classes AS c ON p.classe = c.id
LEFT JOIN
   assuntos AS a ON p.assunto = a.id
LEFT JOIN
   complementos_exp02 AS com
   ON com."anoPrimeiroMovimento" = p."anoPrimeiroMovimento"
        AND com."unidadeID" = m."unidadeID"
        AND com."movimentoID" = m."id"
LEFT JOIN
   servidores AS s ON s."servidorID" = m."usuarioID"
LEFT JOIN
   documentos AS d ON d."id" = m."documentoID"
WHERE
   p."anoPrimeiroMovimento" >= 2020 AND p."unidadeID" = 18006
        AND m."anoPrimeiroMovimento" >= 2020 AND m."unidadeID" = 18006
        AND com."anoPrimeiroMovimento" >= 2020 AND com."unidadeID" = 18006
ORDER BY
   p."processoID", m."dataFinal";
```

Fonte: O autor (2025).

3.2.8 Experimento 03 – particionamento Híbrido (Intervalo + Lista)

Neste experimento, avaliamos a combinação de particionamento por Intervalo, aplicada à coluna *anoPrimeiroMovimento* no primeiro nível, e particionamento por Lista, aplicada à coluna *unidadeID* no segundo nível. A escolha dessa combinação foi motivada pela

flexibilidade e facilidade de expansão das partições ao longo do tempo, bem como pela criação de novas unidades judiciárias. Essa abordagem permite que novas partições sejam adicionadas de maneira eficiente sem a necessidade de redistribuir dados existentes ou causar interrupções no sistema.

3.2.8.1 Preparação dos dados

Nesta etapa, a base de dados foi preparada para o particionamento das tabelas por intervalo, partindo da premissa de que a etapa de preparação do Experimento de Controle já havia sido realizada.

3.2.8.2 Definição da quantidade de partições

Considerando que a base de dados contém registros de três unidades judiciárias distintas, com processos distribuídos ao longo de 13 anos, o experimento adotou, no primeiro nível, a criação de 13 partições por intervalo, aplicadas à coluna *anoPrimeiroMovimento***,** e no segundo nível, a criação de 3 partições por lista, aplicadas à coluna *unidadeID*, correspondendo a uma partição para cada unidade judiciária.

Os comandos necessários para criação das referidas tabelas estão descritos na documentação disponível no material complementar (MOURA, 2025).

3.2.8.3 Migração de dados

Para que as novas tabelas fossem corretamente populadas com os dados já existentes nas tabelas originais, foi necessário executar scripts de migração de dados. A execução desses scripts possibilitou a redistribuição dos dados, que antes estavam armazenados em uma única tabela, para diferentes partições, com base nos valores das colunas *anoPrimeiroMovimento* e *unidadeID*.

Os comandos necessários para criação das referidas tabelas estão descritos na documentação disponível no material complementar (MOURA, 2025).

3.2.8.4 Consulta SQL de referência

A consulta SQL foi ajustada para referenciar as novas tabelas (Quadro Error: Reference source not found).

Quadro 5 – Consulta SQL do experimento 03.

```
SELECT
   p."NPU",
   p."processoID",
   p."ultimaAtualizacao",
   c.descricao AS classe,
   a.descricao AS assunto,
   m.activity,
   m."dataInicio",
   m."dataFinal",
   m."usuarioID",
   m.duration,
   m."movimentoID",
   com.descricao AS complemento,
   s."nomeServidor",
   s."tipoServidor",
   d.tipo AS documento
FROM
   processos_exp03 AS p
INNER JOIN
   movimentos_exp03 AS m
   ON m."unidadeID" = p."unidadeID"
        AND m."anoPrimeiroMovimento" >= p."anoPrimeiroMovimento"
        AND m."processoID" = p."processoID"
INNER JOIN
   classes AS c ON p.classe = c.id
LEFT JOIN
   assuntos AS a ON p.assunto = a.id
LEFT JOIN
   complementos_exp03 AS com
   ON com."unidadeID" = m."unidadeID"
        AND com."anoPrimeiroMovimento" >= p."anoPrimeiroMovimento"
        AND com."movimentoID" = m."id"
LEFT JOIN
   servidores AS s ON s."servidorID" = m."usuarioID"
LEFT JOIN
   documentos AS d ON d."id" = m."documentoID"
WHERE
   p."unidadeID" = 18006 AND p."anoPrimeiroMovimento" >= 2020
        AND m."unidadeID" = 18006 AND m."anoPrimeiroMovimento" >= 2020
        AND com."unidadeID" = 18006 AND com."anoPrimeiroMovimento" >= 2020
ORDER BY
   p."processoID", m."dataFinal";
```

Fonte: O autor (2025).

3.2.9 Experimento 04 – particionamento Híbrido (Intervalo + *Hash*)

Neste experimento, avaliou-se a combinação de particionamento por Intervalo, aplicada à coluna *anoPrimeiroMovimento* no primeiro nível, e particionamento por *Hash*, aplicado à coluna *unidadeID* no segundo nível.

3.2.9.1 Preparação dos dados

Nesta etapa, a base de dados foi preparada para o particionamento das tabelas por intervalo, partindo da premissa de que a etapa de preparação do Experimento de Controle já havia sido realizada.

3.2.9.2 Definição da quantidade de partições

Considerando que a base de dados contém registros de três unidades judiciárias distintas, com processos distribuídos ao longo de 13 anos, o experimento adotou no primeiro nível a criação de 13 partições por intervalo, aplicadas à coluna *anoPrimeiroMovimento*, e no segundo nível a criação de 3 partições por *Hash*, aplicadas à coluna *unidadeID*.

Os comandos necessários para criação das referidas tabelas estão descritos na documentação disponível no material complementar (MOURA, 2025).

3.2.9.3 Migração de dados

Para que as novas tabelas fossem populadas com os dados já existentes nas tabelas originais, foi necessário executar scripts de migração de dados. A execução desses scripts permitiu que os dados, previamente armazenados em uma única tabela, fossem redistribuídos e armazenados em diferentes partições, de acordo com os valores das colunas **anoPrimeiroMovimento** e **unidadeID**.

3.2.9.4 Consulta SQL de referência

A consulta SQL foi ajustada para referenciar as novas tabelas (Quadro Error: Reference source not found).

Quadro 6 – Consulta SQL do experimento 04.

```
SELECT
   p."NPU",
   p."processoID",
   p."ultimaAtualizacao",
   c.descricao AS classe,
   a.descricao AS assunto,
   m.activity,
   m."dataInicio",
   m."dataFinal",
   m."usuarioID",
   m.duration,
   m."movimentoID",
   com.descricao AS complemento,
   s."nomeServidor",
   s."tipoServidor",
   d.tipo AS documento
FROM
   processos_exp04 AS p
INNER JOIN
   movimentos_exp04 AS m
   ON m."unidadeID" = p."unidadeID"
        AND m."anoPrimeiroMovimento" >= p."anoPrimeiroMovimento"
        AND m."processoID" = p."processoID"
INNER JOIN
   classes AS c ON p.classe = c.id
LEFT JOIN
   assuntos AS a ON p.assunto = a.id
LEFT JOIN
   complementos_exp04 AS com
   ON com."unidadeID" = m."unidadeID"
        AND com."anoPrimeiroMovimento" >= p."anoPrimeiroMovimento"
        AND com."movimentoID" = m."id"
LEFT JOIN
   servidores AS s ON s."servidorID" = m."usuarioID"
LEFT JOIN
   documentos AS d ON d."id" = m."documentoID"
WHERE
   p."unidadeID" = 18006 AND p."anoPrimeiroMovimento" >= 2020
        AND m."unidadeID" = 18006 AND m."anoPrimeiroMovimento" >= 2020
        AND com."unidadeID" = 18006 AND com."anoPrimeiroMovimento" >= 2020
ORDER BY
   p."processoID", m."dataFinal";
```

Fonte: O autor (2025).

3.3 MÉTRICAS DE AVALIAÇÃO

Ao se comparar as diferentes estratégias de particionamento de dados é essencial considerar uma variedade de métricas para avaliar o desempenho, a escalabilidade e a eficiência operacional do sistema. Assim, o quadro Error: Reference source not found apresenta as métricas utilizadas para avaliar o desempenho das diferentes estratégias de particionamento aplicadas ao sistema JuMP.

Quadro 7 – Métricas utilizadas na avaliação do desempenho das estratégias aplicadas ao sistema JuMP.

	Métrica	Descrição	Relevância
1	Tempo de Resposta	Mede o tempo necessário para executar operações de consulta a dados no sistema.	Avalia a eficiência das estratégias para lidar com grandes volumes de dados. Estratégias mal implementadas podem aumentar a latência.
2	Utilização de Recursos	Mede o uso de recursos computacionais, como CPU, memória e disco, durante a execução de operações no sistema.	Identifica gargalos que possam ser introduzidos por estratégias de particionamento. Mostra a eficiência no uso da infraestrutura.
3	Escalabilidade	Avalia como a estratégia se comporta ao aumentar o volume de dados ou a quantidade de nós no sistema.	Sistemas com crescimento contínuo de dados exigem estratégias que possam escalar sem comprometer o desempenho.
4	Equilíbrio de Carga	Mede a uniformidade na distribuição de dados e tarefas entre os nós do sistema.	Os desequilíbrios resultam em nós sobrecarregados e outros subutilizados, reduzindo a eficiência do sistema.
5	Transferência de Dados	Mede a quantidade de operações processadas por unidade de tempo (ex.: consultas por segundo ou registros inseridos por segundo).	Avalia o desempenho global do sistema em cenários com alta concorrência.
6	Custo de Redistribuição	Avalia o impacto de redistribuir dados em tempo real devido a alterações na carga ou na configuração do sistema.	Estratégias dinâmicas, como particionamento adaptativo, podem causar interrupções ou consumir recursos durante redistribuições.
7	Eficiência de Consultas	Mede a eficácia da estratégia para consultas específicas, como aquelas que atravessam várias partições.	Importante para sistemas como o JuMP, onde dados são frequentemente acessados de diferentes tribunais e regiões.

Fonte: O autor (2025).

3.3.1 Tempo de Resposta

Esta métrica, expressa em milissegundos, indica o tempo total necessário para que uma consulta SQL seja executada por completo pelo banco de dados. O tempo de resposta foi obtido durante os cenários de teste que simulavam a carga de usuários. A aplicação de testes uma vez implementada é capaz de registrar individualmente a duração de execução de cada consulta, permitindo a coleta detalhada do tempo de resposta de cada operação realizada.

3.3.2 Utilização de Recursos

Durante a execução dos cenários de teste, também foram coletadas informações sobre o consumo instantâneo de CPU e memória do container Docker onde o servidor de banco de dados é executado. Isso foi feito para demonstrar como o aumento no consumo de recursos computacionais se relaciona ao crescimento da carga de usuários simultâneos e para avaliar qual estratégia foi mais eficiente no uso desses recursos.

A coleta dessas informações do *container* foi realizada por meio da Docker Engine API, integrada à aplicação de testes, disponível no material complementar (MOURA, 2025).

3.3.3 Escalabilidade

A métrica de escalabilidade foi medida com base na capacidade do banco de dados de atender adequadamente à crescente demanda de usuários simultâneos, realizando consultas SQL em paralelo, sem ocorrência de falhas. Para simular esse comportamento, implementamos cenários de teste utilizando a ferramenta *JMeter*, com um aumento progressivo de usuários a cada ciclo de execução.

O incremento da quantidade de usuários seguiu a sequência de *Fibonacci*, que foi escolhida por seu crescimento exponencial controlado. Isso permitiu avaliar o impacto da carga no banco de dados de forma gradual, evitando um aumento abrupto e desproporcional da concorrência. Esse método proporciona uma transição mais suave entre os cenários e facilita a identificação de gargalos de desempenho. Além disso, a sequência de Fibonacci reflete um padrão comum em sistemas dinâmicos e redes complexas, tornando-a uma estratégia eficaz para medir a escalabilidade e a eficiência das diferentes abordagens de particionamento sob cargas de trabalho crescentes, (El Abdalaoui *et al*., 2013).

É importante destacar que, em todos os cenários, foi utilizado o mesmo filtro de dados na consulta SQL, restringindo os dados aos últimos 5 anos e relacionados a uma única unidade judiciária. Porém, a unidade judiciária utilizada no filtro era alternada a cada execução, variando entre as três unidades disponíveis na base de dados.

Para avaliar a escalabilidade de cada estratégia, adotamos a seguinte fórmula, que se baseia na relação entre a quantidade de usuários simultâneos, o total de consultas realizadas, a

quantidade de consultas bem-sucedidas e o tempo médio de resposta das consultas bem-sucedidas. O resultado dessa fórmula reflete a quantidade de consultas processadas por usuário, ajustada pelo tempo de resposta ideal (ou seja, consultas abaixo do tempo ideal).

$$E = \left(\frac{Q_sucesso}{Q_total}\right) \times \left(\frac{Q_total}{U}\right) \times \left(\frac{T_ideal}{T_resposta}\right)$$

Onde:

Q sucesso = Número de consultas realizadas com êxito;

Q total = Total de consultas realizadas;

U = Número de usuários simultâneos;

T ideal = Tempo ideal máximo, definido como uma constante no valor de 3 segundos;

T resposta = Tempo médio de resposta das consultas realizadas com êxito.

A fórmula apresentada é fundamentada em conceitos clássicos de desempenho de sistemas distribuídos, escalabilidade de bancos de dados e métricas de eficiência computacional. Embora não exista uma fórmula única e padronizada na literatura científica para medir a escalabilidade de estratégias de particionamento de dados exatamente da maneira descrita, a definição proposta pode ser embasada em estudos e teorias consolidadas nas áreas de bancos de dados distribuídos, benchmarks e avaliação de desempenho. Um exemplo fundamental é a Lei de Amdahl (1967), que descreve como a concorrência e o paralelismo afetam o desempenho de sistemas, destacando os benefícios e limitações desses fatores no contexto de processamento paralelo.

Além disso, a abordagem de escalabilidade discutida pelos autores Stonebraker e Cattell (2011), sobre a medição da escalabilidade de bancos de dados distribuídos, fornece uma base importante. Esses autores propõem uma análise que considera aspectos como o número de usuários, a taxa de consultas e o tempo de resposta, fornecendo um framework robusto para a avaliação de desempenho em sistemas distribuídos. Assim, a fórmula proposta neste estudo está alinhada com essas abordagens, buscando mensurar de maneira prática e eficiente como diferentes estratégias de particionamento de dados lidam com o aumento da carga de trabalho em um ambiente de banco de dados distribuído.

3.3.4 Equilíbrio de Carga

Para possibilitar a análise dessa métrica, os cenários de teste que simulavam a carga de usuários foram aprimorados com a adição de um controle que garantiu que as consultas SQL disparadas pelos testes fossem alternadas de maneira equitativa entre as unidades judiciárias disponíveis na base de dados. Esse controle foi fundamental para garantir uma distribuição balanceada da carga entre as unidades, permitindo uma avaliação mais precisa do desempenho do sistema sob diferentes cenários de carga.

A avaliação do equilíbrio de carga, com base na consulta SQL utilizada pelo JuMP, foi obtida a partir da taxa de uso das partições. Isso é feito dividindo o número de partições acessadas (P_Acessadas) pelo número total de partições disponíveis (P_Disponíveis) e multiplicando o resultado por 100, para expressar o valor em percentual. Uma taxa de 100% indica que todas as partições foram utilizadas de forma equilibrada, enquanto valores inferiores refletem o grau de subutilização das partições disponíveis.

Taxa (%) = (P_Acessadas / P_Disponíveis) * 100

3.3.5 Transferência de Dados

Para calcular a taxa de transferência de dados, inicialmente foi utilizado o comando *EXPLAIN ANALYSE* do PostgreSQL, aplicado na consulta SQL de referência do JuMP. Esse comando nos forneceu o tempo total que a consulta levou para ser executada no banco de dados, além da quantidade de registros retornados pela consulta.

Com essas informações, a taxa de transferência de dados é calculada dividindo-se a quantidade de registros retornados pelo tempo total de execução da consulta, expresso em segundos. O resultado dessa operação nos fornece a taxa de transferência de dados, que é apresentada em registros por segundo. Essa métrica nos permite avaliar a eficiência do banco de dados em relação a quantidade de dados processados por unidade de tempo durante a execução das consultas.

3.3.6 Custo de redistribuição

O custo de redistribuição foi avaliado em uma escala de BAIXA, MÉDIA e ALTA, com base na complexidade envolvida na redistribuição dos dados entre as partições, considerando as particularidades do modelo de dados do JuMP e da estratégia de particionamento utilizada. Por exemplo, quando novas partições precisaram ser adicionadas, a complexidade dessa operação variou dependendo da estratégia de particionamento adotada e da natureza dos dados. Em alguns casos, essa operação foi altamente complexa, pois envolveu a migração de dados, impacto na performance do sistema e possíveis desafios quanto à disponibilidade do banco de dados durante o processo de redistribuição. Assim, a avaliação do custo de redistribuição considerou não apenas o tempo necessário para mover os dados, mas também os efeitos colaterais que isso teve no desempenho e na estabilidade do sistema.

3.3.7 Eficiência da Consulta

A eficiência foi expressa como uma relação entre o número de partições acessadas (P_Acessadas), o total de partições disponíveis (P_Disponíveis), o tempo de execução da consulta (T_Query) e o tempo ideal da consulta (T_Ideal):

Eficiência (%) = (P_Acessadas / P_Disponíveis) * (1 - (T_Query / T_Ideal)) * 100

Para calcular a eficiência, estabelecemos como tempo ideal da consulta o valor constante de **3 segundos**, presumindo que esse seria o tempo máximo aceitável pelos usuários do JuMP para obter uma resposta do sistema. Assim, a eficiência reflete tanto o aproveitamento das partições disponíveis quanto à adequação do tempo de execução da consulta em relação ao tempo ideal preestabelecido.

3.4 AVALIAÇÃO DE ESTRATÉGIAS DE PARTICIONAMENTO DE DADOS UTILIZANDO O MÉTODO TOPSIS

A técnica *Technique for Order of Preference by Similarity to Ideal Solution* (TOPSIS) foi amplamente aplicada na análise multicritério em diversos contextos, incluindo a otimização

de desempenho de bancos de dados relacionais (Hwang; Yoon, 1981). O método se fundamenta na seleção da alternativa que apresenta a menor distância em relação à solução ideal positiva e a maior distância em relação à solução ideal negativa. Neste contexto, foi detalhada a aplicação do TOPSIS na avaliação comparativa das estratégias de particionamento de dados em bancos de dados relacionais.

3.4.1 Definição dos Critérios e Pesos

A primeira etapa consistiu na definição dos critérios de avaliação, os quais foram classificados como critérios de custo ou benefício. Os critérios de custo foram aqueles para os quais valores menores são desejáveis, enquanto os critérios de benefício são aqueles para os quais valores maiores são preferíveis. Os critérios de desempenho e os pesos foram estabelecidos com base em sua influência positiva (benefício) ou negativa (custo) na eficiência das estratégias avaliadas.

As seguintes premissas foram adotadas:

- **Nº Usuários**: Quanto maior a quantidade de usuários simultâneos, maior é a influência positiva;
- **(%) Sucesso:** Quanto maior o valor da taxa, maior é a influência positiva;
- **(%) Erros:** Quanto maior o valor da taxa, maior é a influência negativa;
- **Tempo de Resposta (Mínimo):** Quanto maior o valor, maior é a influência negativa;
- **Tempo de Resposta (Máximo):** Quanto maior o valor, maior é a influência negativa;
- **Tempo de Resposta (Médio):** Quanto maior o valor, maior é a influência negativa;
- **Conexões Ativas:** Quanto maior a quantidade de conexões, maior é a influência negativa;
- **Tam. Arq. Temp. (GB):** Quanto maior o valor, maior é a influência negativa;
- **Cache Hit (%):** Quanto maior a taxa de uso da cache, maior é a influência positiva;
- **Uso Máx. CPU:** Quanto maior a taxa de uso de CPU, maior é a influência negativa;
- **Uso Máx. Memória (GB):** Quanto maior o consumo de Memória, maior é a influência negativa;

A Figura Error: Reference source not found apresenta os critérios utilizados na avaliação das estratégias de particionamento, onde os critérios de benefício estão identificados com o valor "True", enquanto os critérios de custo estão identificados com o valor "False".

Figura 7 – Critérios de benefício e custo da avaliação TOPSIS.

```
criteria_info = {
    "Nº Usuários": True,                # Quanto maior número de usuários, melhor
    "(%) Sucessos": True,               # Quanto maior a taxa de sucesso, melhor
    "(%) Erros": False,                 # Quanto menor a taxa de erros, melhor
    "Menor Duração": False,             # Quanto menor a duração minima, melhor
    "Maior Duração": False,             # Quanto menor a duração máxima é melhor
    "Duração Média": False,             # Quanto menor a duração média é melhor
    "Conexões Ativas": False,           # Quanto menos conexoes ativas é melhor
    "Tam. Arq. Temp. (GB)": False,      # Quanto menos arquivo temporario é melhor
    "Cache Hit (%)": True,              # Quanto maior uso da cache é melhor
    "Uso Máx. CPU (%)": False,          # Quanto menos uso de CPU é melhor
    "Uso Máx. Memória (%)": False       # Quanto menos uso de memória é melhor
}
```

Fonte: O autor (2025).

Cada um dos critérios recebeu um peso proporcional a sua importância relativa, apresentados na figura Error: Reference source not found.

Figura 8 – Pesos estabelecidos aos critérios de avaliação TOPSIS

```
weights = {
    "Nº Usuários": 2.0,                # Quanto maior número de usuários, melhor
    "(%) Sucessos": 2.0,               # Quanto maior a taxa de sucesso, melhor
    "(%) Erros": -1.5,                 # Quanto menor a taxa de erros, melhor
    "Menor Duração": -1.5,             # Quanto menor a duração média, melhor
    "Maior Duração": -0.5,             # Quanto menor a duração máxima é melhor
    "Duração Média": -2.0,             # Quanto menor a duração média é melhor
    "Conexões Ativas": 1.0,            # Quanto mais conexoes ativas é melhor
    "Tam. Arq. Temp. (GB)": -1.0,      # Quanto menos arquivo temporario é melhor
    "Cache Hit (%)": 1.0,              # Quanto maior uso da cache é melhor
    "Uso Máx. CPU (%)": -1.0,          # Quanto menos uso de CPU é melhor
    "Uso Máx. Memória (%)": -1.0       # Quanto menos uso de memória é melhor
}
```

Fonte: O autor (2025).

3.4.2 Normalização da Matriz de Decisão

Para que os critérios possam ser comparados, aplicamos a normalização Euclidiana:

Figura 9 – Normalização Euclidiana.

$$norm_{ij} = \frac{x_{ij}}{\sqrt{\sum_{i=1}^{m} x_{ij}^2}}$$

```
norm_decision = decision_matrix.copy()
for crit in criteria_cols:
    norm = np.sqrt((decision_matrix[crit]**2).sum())
    norm_decision[crit] = decision_matrix[crit] / norm
```

Fonte: O autor (2025).

3.4.3 Cálculo das Soluções Ideais Positiva e Negativa

A solução ideal positiva (α*) foi determinada considerando-se os maiores valores para os critérios de benefício e os menores valores para os critérios de custo. Por outro lado, a solução ideal negativa (α-) foi obtida de forma inversa, ou seja, com os menores valores para os critérios de benefício e os maiores valores para os critérios de custo.

Figura 10 – Cálculo das soluções ideais positiva e negativa

```
ideal_positive = {}
ideal_negative = {}
for crit in criteria_cols:
    if criteria_info[crit]:
        ideal_positive[crit] = weighted_matrix[crit].max()
        ideal_negative[crit] = weighted_matrix[crit].min()
    else:
        ideal_positive[crit] = weighted_matrix[crit].min()
        ideal_negative[crit] = weighted_matrix[crit].max()
```

Fonte: O autor (2025).

3.4.4 Cálculo das distâncias Euclidianas

A distância Euclidiana entre cada alternativa e as soluções ideais é calculada pela fórmula:

Figura 11 – Cálculo das distâncias Euclidianas.

$$d_i^+ = \sqrt{\sum_{j=1}^{n} (x_{ij} - \alpha_j^*)^2}$$

$$d_i^- = \sqrt{\sum_{j=1}^{n} (x_{ij} - \alpha_j^-)^2}$$

```
def euclidean_distance(row, ideal):
    return np.sqrt(np.sum((row - ideal) ** 2))

dist_pos = []
dist_neg = []
for index, row in weighted_matrix.iterrows():
    d_pos = euclidean_distance(row, ideal_positive_series)
    d_neg = euclidean_distance(row, ideal_negative_series)
    dist_pos.append(d_pos)
    dist_neg.append(d_neg)
```

Fonte: O autor (2025).

3.4.5 Cálculo do TOPSIS Score

O coeficiente de similaridade é determinado conforme:

$$S_i = \frac{d^-}{d^+ + d^-}$$

```
topsis_score = []
for d_pos, d_neg in zip(dist_pos, dist_neg):
    score = d_neg / (d_pos + d_neg)
    topsis_score.append(score)
```

Fonte: O autor (2025).

3.4.6 Ordenação e Análise dos Resultados

Os resultados foram classificados em ordem decrescente, permitindo a identificação da melhor estratégia de particionamento.

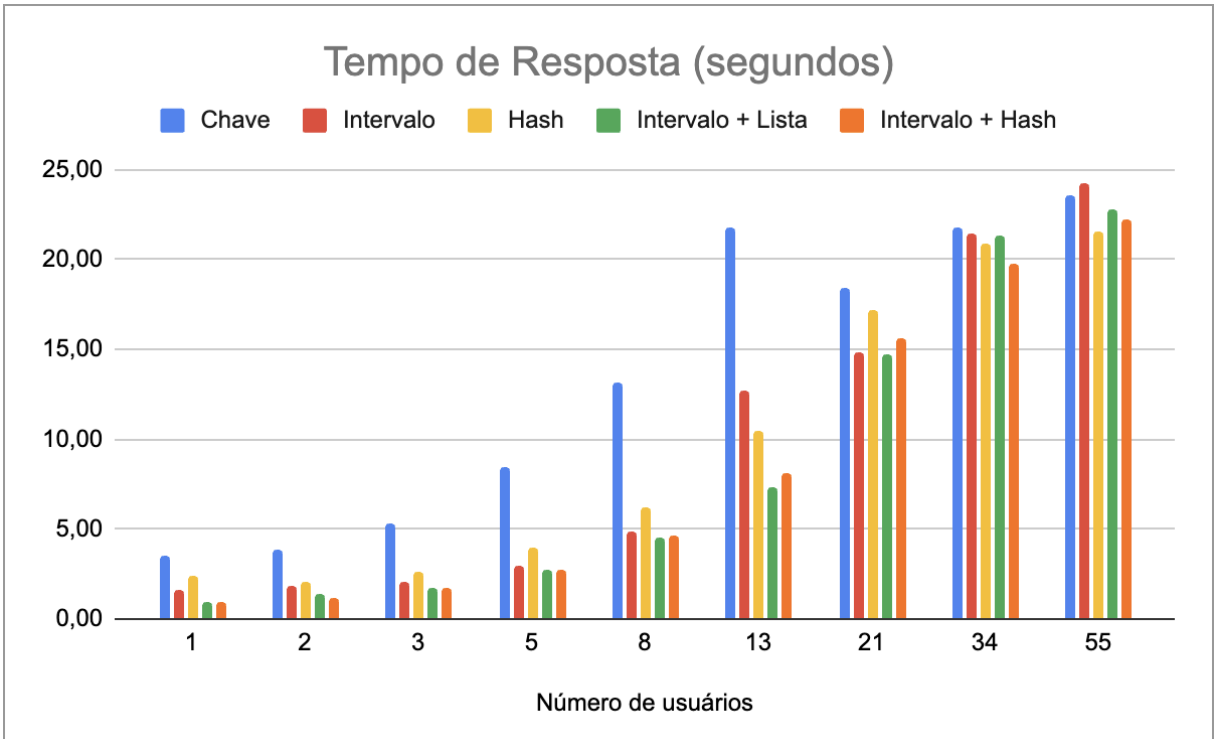
Figura 13 – Ordenação e análise dos resultados.

```
df_topsis = pd.DataFrame({
    "Experimento": df["Experimento"],
    "TOPSIS Score": topsis_score
})
df_topsis = df_topsis.sort_values(by="TOPSIS Score", ascending=False) \
    .reset_index(drop=True)
```

Fonte: O autor (2025).

4 RESULTADOS

O tempo de resposta das diferentes estratégias de particionamento foram analisados, e os resultados obtidos foram: particionamento por Chave com 3,49 segundos, particionamento por Intervalo com 1,57 segundos, particionamento por *Hash* com 2,35 segundos, particionamento Híbrido (Intervalo + Lista) com 0,89 segundos e particionamento Híbrido (intervalo + *Hash*) com 0,90 segundos. A figura Error: Reference source not found apresentada um gráfico comparativo das estratégias de particionamento a partir dos dados de resultados que constam na tabela Error: Reference source not found.



Fonte: O autor (2025).

Fonte: O autor (2025).

Tabela 2 – Tempo de resposta das diferentes estratégias de particionamento de dados aplicadas ao sistema JuMP

	Tempo de Resposta Médio				
Nº Usuários	**Estratégias de particionamento**				
	Chave	**Intervalo**	**Hash**	**Intervalo + Lista**	**Intervalo + Hash**

1	3,50	1,58	2,36	0,89	0,90
2	3,81	1,81	2,09	1,37	1,22
3	5,32	2,07	2,62	1,77	1,78
5	8,51	3,00	3,93	2,75	2,78
8	13,20	4,83	6,23	4,55	4,59
13	21,77	12,67	10,44	7,38	8,11
21	18,45	14,79	17,23	14,76	15,67
34	21,82	21,46	20,93	21,38	19,79
55	23,59	24,24	21,51	22,84	22,27

Fonte: O autor (2025).

Considerando que o ambiente simulado dispõe de 4 *CPUs* e 12 *gigabytes* de memória, os resultados apresentados em percentuais refletem o consumo de recursos em relação à capacidade total disponível. Cada *CPU* pode ser utilizada até 100%, o que significa que um consumo de 400% corresponde ao uso completo das 4 *CPUs*. Esses resultados foram normalizados para apresentar valores em percentual de até 100%. Assim, os resultados de consumo de recursos mostram que o particionamento por Chave utilizou 41,23% da *CPU* disponível e 3,22 GB de memória. O particionamento por Intervalo consumiu 25,33% de *CPU* e 1,22 GB de memória. O particionamento por *Hash* usou 37,36% de *CPU* e 1,69 GB de memória. O particionamento Híbrido (Intervalo + Lista) consumiu 35,66% de *CPU* e 1,05 GB de memória, enquanto o Híbrido (Intervalo + *Hash*) utilizou 36,13% de *CPU* e 1,05 GB de memória (Figura Error: Reference source not found).

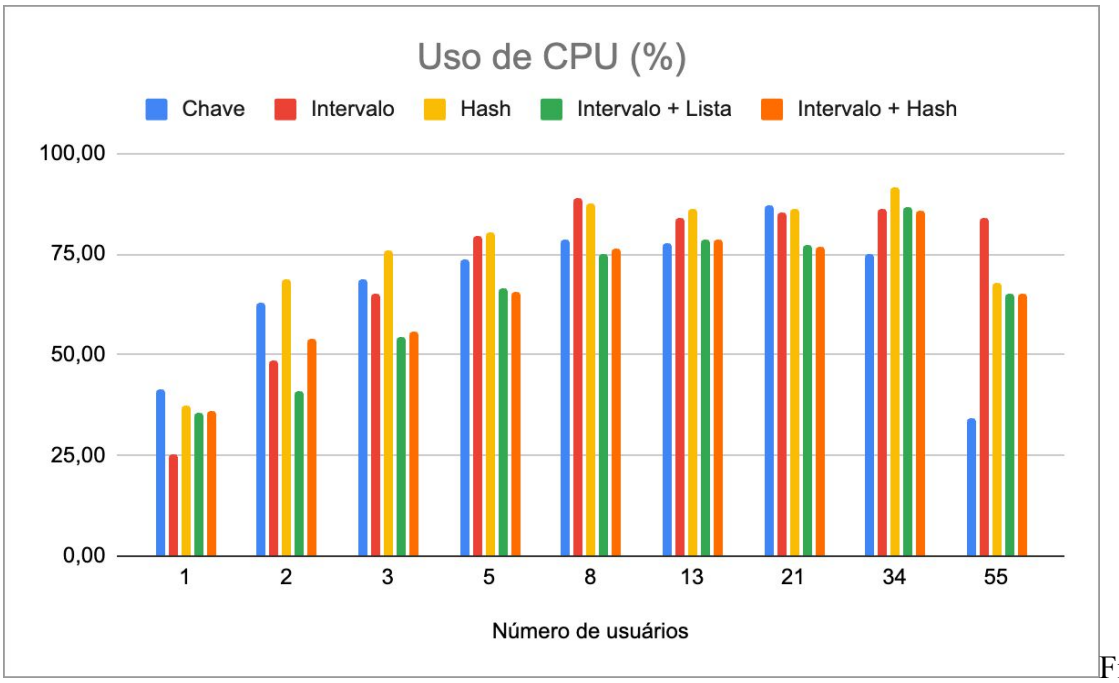


Figura 15 – Consumo de CPU das diferentes estratégias de particionamento de dados aplicadas ao sistema IoMP

Fonte: O autor (2025).

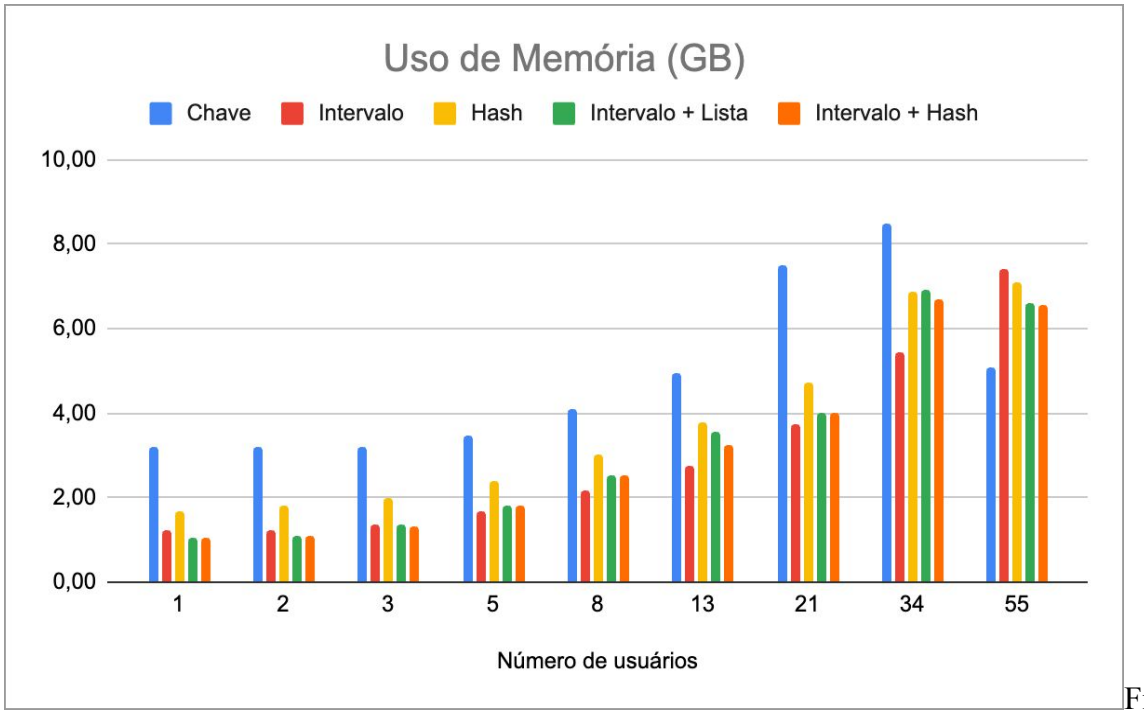


Figura 16 – Consumo de memória (GB) das diferentes estratégias de particionamento de dados aplicadas ao sistema IoMP

Fonte: O autor (2025).

Em relação a escalabilidade, o particionamento por Chave suportou até 8 usuários simultâneos e 40 consultas com sucesso, apresentando falhas de 20,2% a partir de 13 usuários. O particionamento por Intervalo escalou até 13 usuários e 91 consultas com sucesso, apresentando falhas de 0,7% a partir de 21 usuários. O particionamento por *Hash* também suportou até 13 usuários porém com 77 consultas com sucesso, e falhas de 10,3% a partir de 21 usuários. O particionamento Híbrido (Intervalo + Lista) permitiu escalar até 21 usuários e 90 consultas com sucesso, e falhas de 37,0% a partir de 34 usuários, enquanto o particionamento Híbrido (Intervalo + *Hash*) suportou até 21 usuários e 92 consultas com sucesso, mas com falhas de 46,2% a partir de 34 usuários. A Figura Error: Reference source not found apresenta o gráfico comparativo dos tempos de resposta, e a Figura Error: Reference source not found demonstra a taxa de erros das estratégias avaliadas.

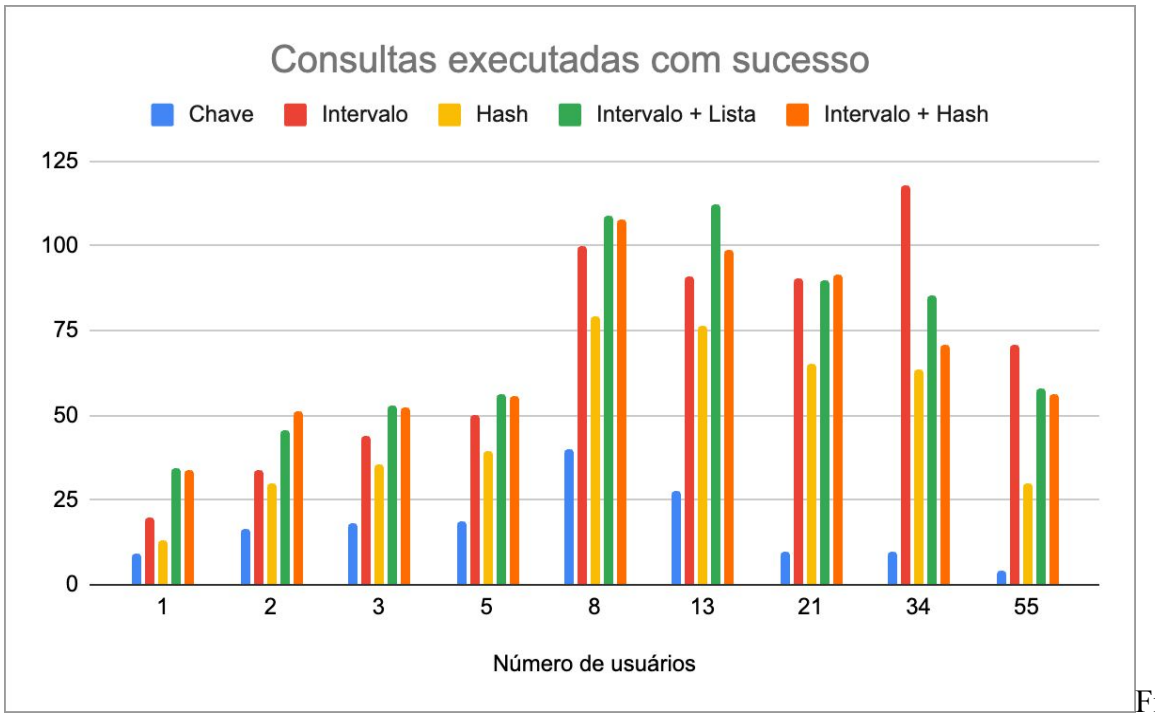


Figura 17 – Consultas com sucesso das diferentes estratégias de particionamento de dados aplicadas

Fonte: O autor (2025).

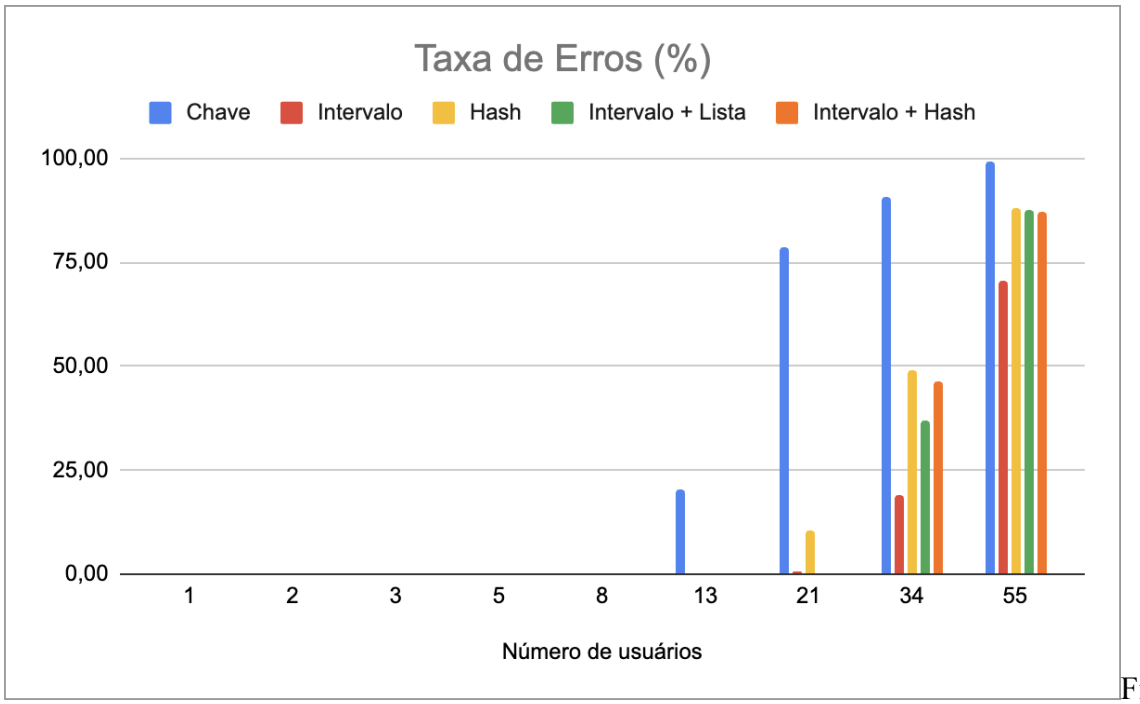


Figura 18 – Taxa de Erros (%) nas consultas das diferentes estratégias de particionamento de dados

Fonte: O autor (2025).

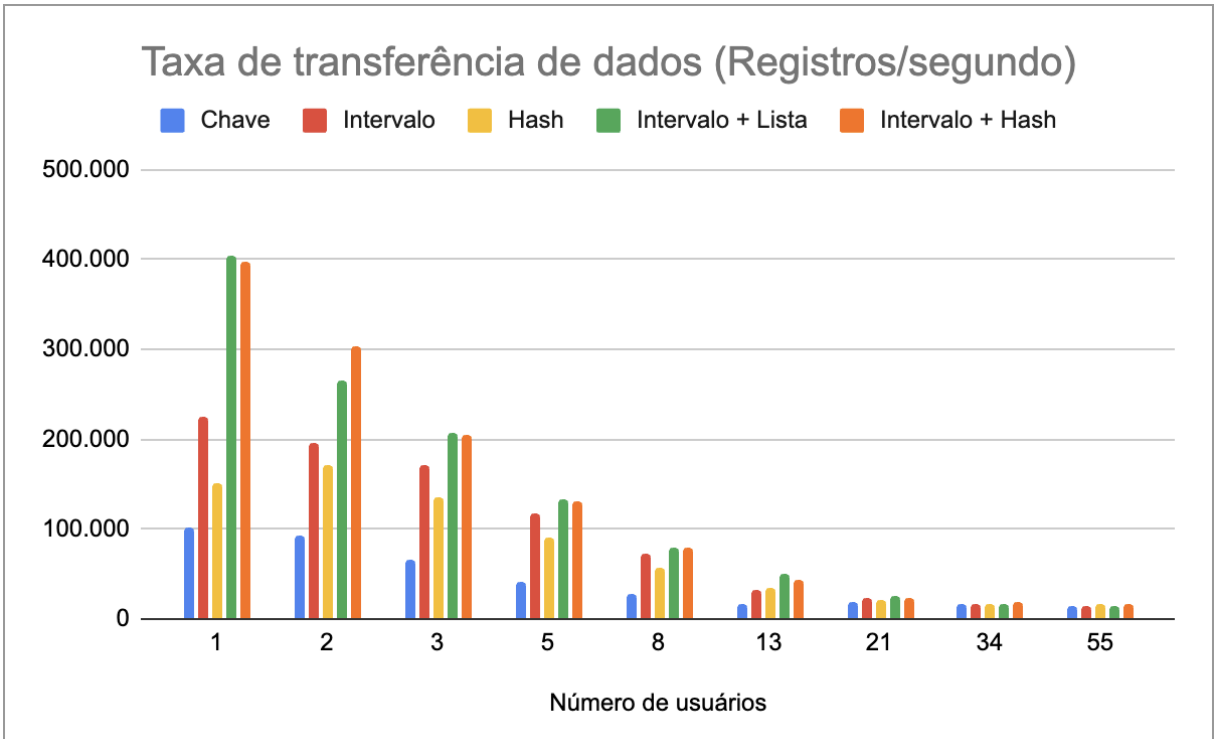
O particionamento por Chave apresentou um equilíbrio de carga de 33,33%**,** indicando uma distribuição desigual dos dados, o que pode levar a algumas partições sobrecarregadas. O particionamento por Intervalo teve um equilíbrio de carga de 46,15%, demonstrando uma distribuição mais equilibrada, embora ainda com algumas variações. O particionamento por *Hash* alcançou um equilíbrio de carga de 100%, o que sugere uma distribuição extremamente uniforme dos dados entre as partições. Já os particionamentos Híbridos (Intervalo + Lista e Intervalo + *Hash*) apresentaram equilíbrio de carga de 46,15%**,** mostrando uma distribuição razoavelmente equilibrada (Tabela Error: Reference source not found).

Tabela 3 – Equilíbrio de carga das diferentes estratégias de particionamento de dados aplicadas ao sistema JuMP.

	Equilíbrio de Carga (%)				
Nº Usuários	**Chave**	**Intervalo**	***Hash***	**Intervalo + Lista**	**Intervalo + *Hash***
1	33,33	46,15	100,00	46,15	46,15

Fonte: O autor (2025).

Os resultados da taxa de transferência de dados para as diferentes estratégias de particionamento foram os seguintes: particionamento por Chave com 101.384 registros por segundo, particionamento por Intervalo com 224.909 registros por segundo, particionamento por *Hash* com 150.353 registros por segundo, particionamento Híbrido (Intervalo + Lista) com 404.061 registros por segundo e particionamento Híbrido (Intervalo + *Hash*) com 396.982 registros por segundo (Figura Error: Reference source not found).



Fonte: O autor (2025).

O custo de redistribuição variou conforme o tipo de particionamento adotado. O particionamento por Chave apresentou um custo de redistribuição alto, devido à necessidade de movimentar grandes volumes de dados quando há mudanças nas chaves, o que pode ser ineficiente. O particionamento por Intervalo teve um custo de redistribuição baixo, já que a divisão por intervalos tende a ser mais estável, com menos movimentação de dados entre as partições. O particionamento por *Hash* também gerou um custo de redistribuição alto, pois mudanças nas funções de *hash* ou nas partições podem exigir um grande movimento de dados.

As abordagens híbridas, como o particionamento Intervalo + Lista, apresentaram um custo médio de redistribuição, indicando que, embora mais eficientes que o *hash*, ainda exigem algum esforço na redistribuição. O particionamento Intervalo + *Hash* teve um custo de redistribuição alto, devido à complexidade combinada de manter a consistência entre intervalos e partições por *Hash*, o que pode demandar um esforço significativo para redistribuir os dados (Tabela Error: Reference source not found).

Tabela 4 – Custo de redistribuição das diferentes estratégias de particionamento de dados aplicadas ao sistema JuMP.

Custo de Redistribuição					
Nº Usuários	**Chave**	**Intervalo**	***Hash***	**Intervalo + Lista**	**Intervalo + *Hash***
1	ALTO	BAIXO	ALTO	MÉDIO	ALTO

Fonte: O autor (2025).

Os resultados de eficiência das consultas para cada estratégia de particionamento foram os seguintes: o particionamento por Chave apresentou uma eficiência de 2,32%, enquanto o particionamento por Intervalo obteve um aumento de 22,41% em relação à situação atual. O particionamento por *Hash* demonstrou a maior melhoria, com uma eficiência 34,26% superior à situação atual. Já o particionamento Híbrido (Intervalo + Lista) teve um aumento de 32,49% na eficiência, e o particionamento Híbrido (Intervalo + *Hash*) apresentou uma melhoria de 32,66% (Tabela Error: Reference source not found).

Tabela 5 – Eficiência da consulta das diferentes estratégias de particionamento de dados aplicadas ao sistema JuMP.

Eficiência da Consulta (%)					
Nº Usuários	**Chave**	**Intervalo**	***Hash***	**Intervalo + Lista**	**Intervalo + *Hash***
1	2,32	24,73	36,58	34,81	34,98
2	-3,21	21,41	62,61	33,55	33,97
3	-7,13	21,05	60,52	32,25	33,12
5	-19,37	22,54	50,14	33,51	33,55
8	-24,22	20,40	51,90	33,35	32,87
13	-48,63	20,51	55,57	32,70	32,71
21	-75,65	1,06	27,69	28,04	27,38
34	-112,30	-1,82	-4,23	24,05	21,09
55	-84,35	-28,62	13,43	16,41	9,66

Fonte: O autor (2025).

A análise dos dados utilizando o método *TOPSIS* (*Technique for Order Preference by Similarity to Ideal Solution*) permitiu ranquear as estratégias de particionamento avaliadas com base em múltiplos critérios de desempenho. Os resultados indicaram que a estratégia Híbrida (Intervalo + Lista) obteve o maior escore *TOPSIS*, demonstrando maior eficiência ao apresentar uma melhor relação entre taxa de sucesso das *queries*, menor tempo de resposta e maior suporte a usuários simultâneos. Em contrapartida, a estratégia Híbrida (Intervalo + *Hash*) apresentou um desempenho inferior, sendo penalizada pelo maior tempo médio de resposta e pelo maior consumo de recursos computacionais, resultando em um menor índice de escalabilidade. A ordenação final das estratégias, baseada no escore *TOPSIS*, apresentada na tabela Error: Reference source not found evidencia as diferenças quantitativas entre os métodos e aponta a solução mais equilibrada em termos de eficiência e desempenho.

Tabela 6 – Ranking TOPSIS Score das estratégias de particionamento avaliadas.

Ranking	**Estratégia**	**TOPSIS Score**
1	**Intervalo + Lista**	0,820287
2	**Intervalo + *Hash***	0,819713
3	**Intervalo**	0,819712
4	***Hash***	0,797472
5	**Chave**	0,753501

Fonte: O autor (2025).

5 DISCUSSÃO

O particionamento de dados é uma técnica fundamental em sistemas de grande escala, pois permite otimizar a gestão de grandes volumes de dados, melhorar a disponibilidade e balancear a carga de trabalho (Han; Pei; Kamber, 2011). No entanto, quando mal implementado, pode resultar em problemas como alta latência, sobrecarga na comunicação entre partições e dificuldades em consultas complexas, impactando negativamente a escalabilidade e o desempenho do sistema (Özsu, 2011; Phansalkar; Ahirrao, 2016). Neste contexto, este estudo explora diferentes estratégias de particionamento de dados aplicadas ao JuMP, visando avaliar sua eficácia na otimização do desempenho e da escalabilidade da solução.

A análise dos resultados do tempo de resposta revelou que as estratégias híbridas (Intervalo + Lista e Intervalo + *Hash*) apresentaram desempenho superior em relação às demais abordagens avaliadas. Dentre elas, o particionamento Híbrido (Intervalo + Lista) se destacou com os melhores resultados, alcançando um tempo de resposta de 0,89 s, o que representa uma redução significativa no tempo de processamento. A combinação de intervalo e lista oferece uma estrutura eficiente para consultas em faixas de dados e, ao mesmo tempo, permite um controle refinado sobre a alocação dos dados, otimizando a distribuição e reduzindo a latência. Porém, em cenários de alta carga, a complexidade do modelo de lista pode afetar a escalabilidade, demandando uma análise detalhada sobre o crescimento do sistema e o aumento de partições.

Por outro lado, a análise de escalabilidade revelou que o particionamento Híbrido (Intervalo + Lista) foi a estratégia mais eficiente, sendo capaz de suportar até 21 usuários simultâneos e um total de 90 consultas sem falhas, com médio custo de redistribuição. Em seguida, a estratégia Híbrida (Intervalo + *Hash*) que também conseguiu suportar a mesma quantidade de usuários simultâneos e 92 consultas sem falhas, porém obteve uma taxa de falhas 9,12% maior que a estratégia (Intervalo + Lista) no cenário com 34 usuários simultâneos, onde a primeira conseguiu executar mais consultas.

Essas abordagens se mostraram adequadas para cenários de carga elevada, onde as consultas são centradas em faixas de dados bem definidas. Destacando a abordagem (Intervalo + Lista) por apresentar um bom desempenho em termos de distribuição de dados e custo de redistribuição, embora a flexibilidade possa ser impactada conforme a carga cresce. A

abordagem por intervalo também permitiu escalar até 21 usuários simultâneos e 90 consultas com êxito, porém com uma taxa mínima de falhas e com tempo médio de resposta ~4 segundos acima que a Abordagem Híbrida (Intervalo + Lista), que respondeu em tempo médio de 14 segundos.

Já o particionamento Híbrido (Intervalo + *Hash*), apesar de oferecer uma distribuição uniforme dos dados, demonstrou um alto custo de redistribuição, o que pode prejudicar a performance em sistemas que necessitam de frequentes adições ou ajustes nas partições. Por outro lado, o particionamento por Chave se mostrou limitado, com falhas superiores a 20% a partir de 13 usuários, sendo mais eficiente em cenários com menor número de usuários simultâneos, mas com escalabilidade restrita. O particionamento por *Hash*, por sua vez, proporcionou uma boa distribuição de dados, mas enfrentou dificuldades similares em termos de custo de redistribuição, comprometendo sua eficiência em sistemas com grande carga de consultas. Assim, as estratégias híbridas apresentam um bom equilíbrio entre escalabilidade e desempenho, mas exigem um gerenciamento cuidadoso do custo de redistribuição para evitar perdas de eficiência em cenários de alta carga.

A estratégia de particionamento por *Hash*, que se destaca pela sua distribuição uniforme dos dados, obteve o melhor desempenho em termos de equilíbrio de carga (100%). Esse balanceamento ideal é possível porque o particionamento por *Hash* distribui os dados aleatoriamente entre as partições, prevenindo sobrecargas em partições individuais (Stein *et al*., 2014; Goulart, 2023). No entanto, essa técnica não é a mais adequada para consultas que envolvem faixas específicas de dados, pois a distribuição aleatória pode dificultar o acesso eficiente a intervalos de interesse. Além disso, o custo de redistribuição de dados quando novas partições são adicionadas é elevado, o que pode impactar o desempenho em cenários de grande escala.

As abordagens híbridas (Intervalo + Lista e Intervalo + *Hash*) combinaram as vantagens do particionamento por Intervalo com as da Lista e do *Hash*, proporcionando uma combinação eficiente entre otimização de consultas e balanceamento de carga. O particionamento Híbrido (Intervalo + Lista) obteve uma melhoria de 32,49%, enquanto o particionamento Híbrido (Intervalo + *Hash*) apresentou um ganho de 32,66%. Essas combinações permitem uma maior flexibilidade ao mesmo tempo em que mantêm um bom balanceamento de dados, mas ambos os modelos enfrentam desafios no que diz respeito ao custo de manutenção e escalabilidade em cenários de carga elevada.

A necessidade de redistribuir dados ao adicionar novas partições pode gerar um impacto considerável no desempenho, especialmente em sistemas com alta variação no volume de consultas e dados. Ainda assim, a combinação (Intervalo + Lista) é a mais indicada para cenários de crescimento de partições com o surgimento de novas chaves, pois ela permite a implementação de rotinas automatizadas para a criação dinâmica de novas partições, otimizando esse processo.

Por fim, a estratégia de particionamento por Chave, atualmente implementada, obteve um desempenho inferior, com eficiência de 2,32%. Isso sugere que, embora os dados estejam distribuídos de forma estática, a alocação fixa de dados nas partições pode resultar em um desbalanceamento, principalmente quando determinadas chaves são acessadas com maior frequência, o que gera hotspots e compromete o desempenho (Zaharia *et al*., 2016). Além disso, apresenta problemas de desbalanceamento de carga e alto custo de redistribuição, especialmente em cenários com grandes volumes de dados e consultas intensivas.

No que diz respeito à taxa de transferência de dados, o particionamento Híbrido (Intervalo + Lista) se destacou, com 404.061 registros por segundo, seguido pelo particionamento Híbrido (Intervalo + Lista), com 396.982 registros por segundo. O particionamento por Intervalo e por *Hash*, embora eficientes, apresentaram desempenho inferior em comparação com as abordagens híbridas. A taxa de transferência do particionamento por Intervalo foi de 224.909 registros por segundo, enquanto a do particionamento por *Hash* foi de 150.353 registros por segundo. Esses resultados indicam que, embora o particionamento por Intervalo seja eficiente para consultas que envolvem faixas de dados, ele não é tão eficaz para a transferência de grandes volumes de dados, ao passo que as abordagens híbridas se destacam nesse aspecto.

A análise dos custos de redistribuição também evidenciou o impacto significativo desse fator na eficiência de cada estratégia. O particionamento por Chave apresentou custo alto de redistribuição quando necessário unificar ou dividir dados das partições, envolvendo a migração de dados e impactando diretamente na performance e disponibilidade do sistema. O particionamento por Intervalo, por outro lado, apresentou baixos custos de redistribuição e foi flexível o suficiente para lidar com a adição de novos intervalos, mas ainda pode ser afetado por hotspots se os dados não forem distribuídos uniformemente. O particionamento por *Hash* garantiu uma boa distribuição de dados, mas seus custos de redistribuição são elevados devido à necessidade de recalcular os valores de *hash* ao adicionar novas partições. As abordagens

híbridas, embora combinassem as vantagens dos métodos por Intervalo e *Hash*, também apresentaram desafios relacionados ao custo de manutenção e escalabilidade, especialmente em cenários de alta carga.

A interpretação dos resultados obtidos pelo TOPSIS reforça a importância de avaliar estratégias de particionamento considerando múltiplas métricas simultaneamente, em vez de métricas isoladas. Embora a estratégia (Intervalo + Lista) tenha sido a mais bem ranqueada, é necessário considerar que seu desempenho pode ser influenciado por diferentes fatores, como a carga de trabalho aplicada e o volume de dados armazenado. Além disso, a penalização da estratégia (Intervalo + *Hash*) pelo maior tempo de resposta ressalta a sensibilidade do modelo a critérios de latência, o que pode impactar sua aplicabilidade em sistemas com requisitos de tempo real. Dessa forma, os resultados demonstram que a escolha da estratégia de particionamento mais adequada deve levar em conta o balanço entre escalabilidade, eficiência computacional e impacto no tempo de execução das *queries*, garantindo que a solução selecionada seja condizente com o ambiente operacional do sistema.

De um modo geral, os resultados indicam que escolha da estratégia de particionamento de dados depende diretamente do tipo de carga de trabalho e do volume de consultas. No caso do JuMP, o particionamento Híbrido (Intervalo + Lista) mostrou-se a abordagem mais eficiente em termos de tempo de resposta e transferência de dados, permitindo uma maior escalabilidade em cenários de alta carga de usuários.

6 CONCLUSÃO

Os resultados deste estudo evidenciam a relevância das estratégias de particionamento de dados para otimizar o desempenho e a escalabilidade em sistemas de grande escala, como o JuMP. As análises demonstraram que as abordagens híbridas, em especial a combinação de particionamento por Intervalo e Lista, apresentaram os melhores resultados em relação ao tempo de resposta e taxa de transferência de dados, destacando-se com 0,89 segundos de tempo de resposta e taxa de transferência de 404.061 registros por segundo, tudo isso com um consumo de eficiente dos recursos de CPU (35%) e de memória (1,06 GB).

Embora o particionamento por Intervalo tenha se mostrado eficiente para cenários de carga moderada, suportando até 13 usuários simultâneos sem falhas e oferecendo maior flexibilidade na distribuição de dados e controle de redistribuição. As abordagens híbridas foram capazes de escalar até 21 usuários sem que fossem identificados erros nas consultas. No entanto, os custos de redistribuição, especialmente na estratégia Híbrida (Intervalo + *Hash*), impactaram significativamente sua eficácia, limitando a sua aplicabilidade em sistemas com grande volume de consultas e necessidade de ajustes frequentes nas partições.

A estratégia de particionamento por Chave, atualmente implementada no JuMP, embora simples e eficiente em cenários com menor número de usuários simultâneos, revelou-se ineficiente para cenários de maior escalabilidade, apresentando falhas superiores a 20% após 13 usuários simultâneos. Assim, as abordagens híbridas (Intervalo + Lista e Intervalo + *Hash*) mostraram-se mais adequadas para o JuMP, recomendando-se a combinação (Intervalo + Lista) para cenários de crescimento das partições, pois oferece maior flexibilidade no gerenciamento de novas chaves, desde que acompanhada de uma análise cuidadosa do crescimento dos dados para minimizar os custos de redistribuição e garantir a manutenção do desempenho.

Os resultados obtidos por meio da análise *TOPSIS* demonstraram que a escolha da estratégia de particionamento de dados impacta significativamente o desempenho do sistema, especialmente em cenários de alta concorrência. A estratégia melhor ranqueada apresentou maior eficiência na execução das *queries*, menor tempo de resposta e melhor escalabilidade, reforçando a importância de considerar múltiplos critérios na avaliação do desempenho. No entanto, a análise também evidenciou que o desempenho de cada estratégia pode variar conforme o volume de dados e a carga de trabalho aplicada, tornando essencial a realização de testes específicos para cada contexto operacional. Assim, este estudo contribui para a tomada de

decisão baseada em dados, oferecendo uma metodologia estruturada para a escolha da estratégia de particionamento mais adequada a diferentes cenários de aplicação.

7 CONTRIBUIÇÕES

7.1.1 Avaliação detalhada e comparativa das estratégias de particionamento de dados:

A análise comparativa das estratégias de particionamento de dados apresentada nesta dissertação é uma contribuição substancial para a área, uma vez que proporciona uma visão aprofundada das vantagens, limitações e desempenho de cada abordagem em cenários diversos. Este estudo detalha as técnicas mais tradicionais, como particionamento por Intervalo, Chave e *Hash*, e também explora abordagens mais avançadas, como a de particionamento Híbrido, revelando a aplicabilidade dessas estratégias em contextos específicos.

7.1.2 Análise da escalabilidade e adaptabilidade das estratégias de particionamento:

A avaliação da escalabilidade e adaptabilidade das estratégias de particionamento realizada neste trabalho é particularmente relevante, dado o foco de muitos sistemas modernos em crescer de forma eficiente à medida que a carga de trabalho aumenta. A dissertação oferece uma análise sólida sobre como as estratégias lidam com o aumento no volume de dados e com padrões de acesso dinâmicos.

7.1.3 Ferramental e orientações práticas para otimização de sistemas de análise de dados em grande escala:

A ferramenta implementada e as orientações práticas fornecidas neste estudo para a otimização de sistemas que processam grandes volumes de dados são uma valiosa contribuição para profissionais da área de TI e Bancos de Dados. A ferramenta pode ser facilmente ajustada para avaliar estratégias de particionamento aplicadas a qualquer cenário de bancos de dados relacionais. E as recomendações sobre a escolha e a configuração das

estratégias de particionamento são particularmente úteis, fornecendo uma base sólida para a tomada de decisões.

8 TRABALHOS FUTUROS

Realizar novos experimentos utilizando uma infraestrutura distribuída para o banco de dados, com bases replicadas e balanceamento da carga de conexões. Esta abordagem pode melhorar significativamente a performance do sistema, permitindo o processamento paralelo de dados em múltiplas instâncias do banco de dados. Associado às estratégias de particionamento de dados, isso pode resultar em maior escalabilidade e eficiência operacional.

Além disso, investigar o desempenho das estratégias de particionamento em arquiteturas de dados *multi-tenant*, especialmente no contexto de tribunais. Como cada tribunal pode ter diferentes volumes de dados e consultas, avaliar como as abordagens de particionamento lidam com essa diversidade e como mantêm a eficiência em cenários de alta carga e múltiplos usuários será essencial para aprimorar a escalabilidade e a gestão dos dados em sistemas complexos.

Desse modo, esses trabalhos podem fornecer insights valiosos para a melhoria contínua das estratégias de particionamento de dados no JuMP, visando não apenas otimizar a performance, mas também garantir a flexibilidade necessária para se adaptar a diferentes cenários e demandas.

REFERÊNCIAS BIBLIOGRÁFICAS

Abdalaoui, E. H. E et al. On the Fibonacci complex dynamical systems. **arXiv preprint arXiv:1304.4864**, 2013.

ALMEIDA, Ana Carolina et al. Particionamento como Ação de Sintonia Fina em Bancos de Dados Relacionais. In: **Simpósio Brasileiro de Banco de Dados (SBBD)**. SBC, 2017. p. 112-123.

AMAZON WEB SERVICES. **Amazon Aurora User Guide**. Seattle: Amazon, 2023. Disponível em: https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/Welcome.html. Acesso em: 22 fev. 2025.

AMAZON WEB SERVICES. **Amazon DynamoDB Developer Guide**. Seattle: Amazon, 2023. Disponível em: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html. Acesso em: 22 fev. 2025.

BRASIL. **Lei nº 11.419, de 19 de dezembro de 2006**. Dispõe sobre a informatização do processo judicial e dá outras providências. Disponível em: http://www.planalto.gov.br/ccivil_03/leis/2006/l11419.htm. Acesso em: 4 de março. 2025.

BRAZ, Graciela Farias. **Decisão judicial por meio da inteligência artificial e o desrespeito à garantia fundamental da motivação no judiciário brasileiro**. 2022. Dissertação (Mestrado) – Universidade Federal de Alagoas, Alagoas, 2022.

CANTINI, Riccardo et al. Block size estimation for data partitioning in HPC applications using machine learning techniques. **Journal of Big Dat**, v. 11, n. 1, p. 19, 2024.

CATTELL, R**.** Scalable SQL and NoSQL data stores. **ACM Computing Research Repository (CoRR***)*, 2011.

MOURA, Cleber Tavares de. [Repositório de código-fonte com o projeto dos experimentos]. [Recife], 2025. Disponível em: https://github.com/ctmoura/jump_data_partitioning. Acesso em: 4 de março. 2025.

CNJ – CONSELHO NACIONAL DE JUSTIÇA**. Ferramenta mapeia gargalos no andamento de processos judiciais**. Disponível em: https://www.cnj.jus.br/ferramenta-mapeia-gargalos-no-andamento-de-processos-judiciais/. Acesso em: 7 set. 2024.

CNJ – CONSELHO NACIONAL DE JUSTIÇA. **Lançamento do CODEX, 2022**. Disponível em: https://www.cnj.jus.br/sistemas/plataforma-codex/.

CNJ – CONSELHO NACIONAL DE JUSTIÇA. **Mineração de processos inova gestão com identificação gráfica dos gargalos**. *Conselho Nacional de Justiça*, 17 dez. 2021. Disponível em: https://www.cnj.jus.br/mineracao-de-processos-inova-gestao-processual-com-identificacao-grafica-dos-gargalos/. Acesso em: 4 de março. 2025.

CNJ – CONSELHO NACIONAL DE JUSTIÇA. **Processo Judicial Eletrônico – PJ-E**. Disponível em: https://www.cnj.jus.br/programas-e-acoes/processo-judicial-eletronico-pje/. Acesso em: 7 set. 2024.

CONJUR. **Justiça 4.0 apresenta soluções digitais em evento sobre inovação no Judiciário**. *Conjur*, 1 set. 2024. Disponível em: https://www.conjur.com.br/2024-set-01/justica-4-0-apresenta-solucoes-digitais-em-evento-sobre-inovacao-no-judiciario/. Acesso em: 28 set. 2024.

CONSELHO NACIONAL DE JUSTIÇA (CNJ). **CNJ lança curso inédito para ferramenta de mineração de processos judiciais**. *Portal CNJ*, 17 mar. 2025. Disponível em: https://www.cnj.jus.br/cnj-lanca-curso-inedito-para-ferramenta-de-mineracao-de-processos-judiciais/. Acesso em: 20 mar. 2025.

CORBETT, J. D. J. C.; DEAN, Jeffrey. ME et al. Spanner: Google's globally distributed database. In: **Proceedings of OSDI**. 2012. p. 251-264.

COSTA, E; COSTA, C; SANTOS, M. Y. Evaluating partitioning and bucketing strategies for Hive-based Big Data Warehousing systems. **Journal of Big Data**, v. 6, n. 1, p. 34, 2019.

COULOURIS, G. et al. **Sistemas Distribuídos-: Conceitos e Projeto**. Bookman Editora, 2013.

CUTKOSKY, A.; BUSA-FEKETE, R. Distributed Stochastic Optimization via Adaptive SGD. 2018.

DEAN, J.; GHEMAWAT, S. MapReduce: simplified data processing on large clusters. **Communications of the ACM**, v. 51, n. 1, p. 107-113, 2008.

DECANDIA, G. et al. Dynamo: Amazon's highly available key-value store. **ACM SIGOPS operating systems review**, v. 41, n. 6, p. 205-220, 2007.

DENG, Y.; ZHANG, Y.; WU, Y**.** An overview of data partitioning techniques in distributed systems. **IEEE Access**, v. 9, p. 124245-124259, 2021.

DEWITT, D.; GRAY, J. Parallel database systems: The future of high performance database systems. **Communications of the ACM**, v. 35, n. 6, p. 85-98, 1992.

ELMASRI, R.; NAVATHE, S. B. Database systems: models, languages, design, and application programming. 6. ed. Boston: Addison-Wesley, 2011.

ELMASRI, Ramez. **Fundamentals of database systems**. Pearson Education India, 2008.

FINKEL, Raphael A.; BENTLEY, Jon Louis. Quad trees a data structure for retrieval on composite keys. **Acta informatica**, v. 4, p. 1-9, 1974.

GOOGLE CLOUD. **Cloud Spanner Documentation**. Mountain View: Google, 2023. Disponível em: https://cloud.google.com/spanner/docs. Acesso em: 22 fev. 2025.

GOULART, HENRIQUE DOS SANTOS. **Uma estratégia de rebalanceamento de estados para checkpoints particionados**. Dissertação (Mestrado no Programa de Pós-Graduação em Ciência da Computação) — Universidade Federal de Santa Catarina, 2023.

GUNASEKARAN, Angappa; PATEL, Chaitali; TIRTIROGLU, Ercan. Performance measures and metrics in a supply chain environment. **International journal of operations & production Management**, v. 21, n. 1/2, p. 71-87, 2001.

GUTTMAN, Antonin. R-trees: A dynamic index structure for spatial searching. In: **Proceedings of the 1984 ACM SIGMOD international conference on Management of data**. 1984. p. 47-57.

HAN, J.; PEI, J.; KAMBER, M. Data mining: concepts and techniques. [S.l.]: Elsevier, 2011.

HORI, Keizo et al. Learned spatial data partitioning. In: **Proceedings of the sixth international workshop on exploiting artificial intelligence techniques for data management**. 2023. p. 1-8.

HWANG, Ching-Lai et al. Methods for multiple attribute decision making. **Multiple attribute decision making: methods and applications a state-of-the-art survey**, p. 58-191, 1981.

KLEPPMANN, Martin. **Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems**. " O'Reilly Media, Inc.", 2017.

LAKSHMAN, Avinash; MALIK, Prashant. Cassandra: a decentralized structured storage system. **ACM SIGOPS operating systems review**, v. 44, n. 2, p. 35-40, 2010.

LAMPORT, Leslie. The part-time parliament. In: **Concurrency: the works of Leslie Lamport**. 2019. p. 277-317.

LI, Ruiyuan et al. Apache shardingsphere: A holistic and pluggable platform for data sharding. In: **2022 IEEE 38th International Conference on Data Engineering (ICDE)**. IEEE, 2022. p. 2468-2480.

LI, Zhongzhi; WANG, Xuegang. Spatial clustering algorithm based on hierarchical-partition tree. **International Journal of Digital Content Technology and its Applications**, v. 4, n. 6, p. 26-35, 2010.

LIMA, R. M. F. et al. Mineração de processos no Judiciário Brasileiro. **Computação Brasil**, n. 49, p. 39-43, 2023.

LINO JUNIOR, Daniel Alves. **Transparência Ativa na Justiça**: o Modelo Gerencialista do CNJ e sua incidência na criação do Banco Nacional de Dados da Justiça - Datajud. 2023. Dissertação (Mestrado) – Universidade de São Paulo, São Paulo, 2023. Disponível em: https://www.teses.usp.br/teses/disponiveis/100/100134/tde-24102023-184608/. Acesso em: 15 jan. 2025.

LIU, Peng-Ju; LI, Cui-Ping; CHEN, Hong. Enhancing storage efficiency and performance: A survey of data partitioning techniques. **Journal of Computer Science and Technology**, v. 39, n. 2, p. 346-368, 2024.

LUO, Chen; CAREY, Michael J. Dyna*Hash*: Efficient Data Rebalancing in Apache AsterixDB. In: **2022 IEEE 38th International Conference on Data Engineering (ICDE)**. IEEE, 2022. p. 485-497.

MEHR, Azadeh Sadat Mozafari. **Multi-perspective conformance checking: identifying and understanding patterns of anomalous behavior**. 2024.

O'NEIL, Patrick; O'NEIL, Elizabeth. Database: Principles, programming, and performance, chapter 8, section 6. 2000.

ÖZSU, M. T.; VALDURIEZ, P. **Principles of distributed database systems**. Englewood Cliffs: Prentice Hall, 1999.

ÖZSU, M. Tamer et al. NoSQL, NewSQL, and polystores. **Principles of distributed database systems**, p. 519-558, 2020.

PHANSALKAR, Shraddha; AHIRRAO, Swati. Survey of data partitioning algorithms for big data stores. In: **2016 Fourth International Conference on Parallel, Distributed and Grid Computing (PDGC)**. IEEE, 2016. p. 163-168.

PORTO, F. **Apresentação da ferramenta JuMP no CNJ, 2022**. Disponível em: https://www.cnj.jus.br/ferramenta-mapeia-gargalos-no-andamento-de-processos-judiciais/.

POSTGRESQL GLOBAL DEVELOPMENT GROUP. **PostgreSQL 16 Documentation: 5.12. Table Partitioning.** [S. l.], 2023. Disponível em: https://www.postgresql.org/docs/current/ddl-partitioning.html. Acesso em: 31 mar. 2025.

RESOLUÇÃO CNJ Nº 185, DE 18 DE DEZEMBRO DE 2013. **Dispõe sobre o processo judicial eletrônico e dá outras providências.** Disponível em: https://www.cnj.jus.br/wp-content/uploads/2014/07/RESOLUCAO_185_2013.PDF. Acesso em: 31 mar. 2025.

RONG, Kexin et al. Approximate partition selection for big-data workloads using summary statistics. **arXiv preprint arXiv:2008.10569**, 2020.

SADALAGE, Pramod J.; FOWLER, Martin. **NoSQL distilled: a brief guide to the emerging world of polyglot persistence**. Pearson Education, 2013.

SAVVA, Fotis; ANAGNOSTOPOULOS, Christos; TRIANTAFILLOU, Peter. Adaptive Learning of Aggregate Analytics under Dynamic Workloads. 2019.

SILBERSCHATZ, Abraham; KORTH, Henry F.; SUDARSHAN, Shashank. **Database system concepts**. 2011.

SLESAREV, Alexander; MIKHAILOV, Mikhail; CHERNISHEV, George. Benchmarking hashing algorithms for load balancing in a distributed database environment. In: **International Conference on Model and Data Engineering**. Cham: Springer Nature Switzerland, 2022. p. 105-118.

SOLAT, Siamak. Sharding Distributed Databases: A Critical Review. **arXiv preprint arXiv:2404.04384**, 2024.

STONEBRAKER, Michael; CATTELL, Rick. 10 rules for scalable performance in'simple operation'datastores. **Communications of the ACM**, v. 54, n. 6, p. 72-80, 2011.

APACHE SOFTWARE FOUNDATION (a). **Apache Cassandra Documentation**. 2023. Disponível em: https://cassandra.apache.org/doc/latest/. Acesso em: 22 fev. 2025.

APACHE SOFTWARE FOUNDATION (b). **Apache JMeter**. [S. l.], [s. n.], 2023. Disponível em: https://jmeter.apache.org/. Acesso em: 06 abr. 2025.

TOMIO, F. R.L; ROBL FILHO, I. N; SANTOS-PINTO, R. **The National Judicial Council** (CNJ) and the creation of digital procedural platforms (PJe): methodology for compared research of judicial efficiency. **Revista da Faculdade de Direito** – *UFPR*, Curitiba, v. 60, n. 2, p. 97-114, maio/ago. 2015.

VAN DER AALST, W. M. P. Process mining: Data science in action. 2. ed. Cham: Springer, 2016.

VAN STEEN, Maarten; TANENBAUM, Andrew. Distributed Systems Pearson Education. 2017.

VOGELS, W. Eventually consistent. **Communications of the ACM**, v. 52, n. 5, p. 40-44, 2009.

WANG, Lei et al. **A Cluster-Based Partition Method of Remote Sensing Data for Efficient Distributed Image Processing**. Remote Sensing, Basel, v. 14, n. 19, p. 1–22, 2022. Disponível em: https://www.mdpi.com/2072-4292/14/19/4964. Acesso em: 6 abr. 2025.

ZAHARIA, M. et al. Apache spark: a unified engine for big data processing. **Communications of the ACM**, v. 59, n. 11, p. 56-65, 2016.

ZAHARIA, M. et al. Spark: *Cluster* computing with working sets. In: **2nd USENIX workshop on hot topics in cloud computing (HotCloud 10)**. 2010.

ZEIN, Alaa Aldin et al. **Clustering-based method for big spatial data partitioning.** Intelligent Systems with Applications, [S. l.], v. 17, p. 200267, 2023. ISSN 2665-9174. Disponível em: https://www.sciencedirect.com/science/article/pii/S2665917423000673. Acesso em: 6 abr. 2025.

ZHANG, Zuyu; DESHMUKH, Harshad; PATEL, Jignesh M. **Data Partitioning for In-Memory Systems: Myths, Challenges, and Opportunities**. In: **CIDR**. 2019.