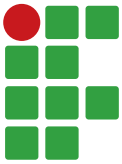


INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Goiás

Bacharelado em Sistemas de Informação

Ítalo Freire Santos

Redução do Tempo de Processamento na Importação de Dados Bancários com Computação Paralela: Uma Simulação

Luziânia
2025

INSTITUTO FEDERAL
Goiás

MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
SISTEMA INTEGRADO DE BIBLIOTECAS

TERMO DE AUTORIZAÇÃO PARA DISPONIBILIZAÇÃO NO REPOSITÓRIO DIGITAL DO IFG - ReDi IFG

Com base no disposto na Lei Federal nº 9.610/98, AUTORIZO o Instituto Federal de Educação, Ciência e Tecnologia de Goiás, a disponibilizar gratuitamente o documento no Repositório Digital (ReDi IFG), sem ressarcimento de direitos autorais, conforme permissão assinada abaixo, em formato digital para fins de leitura, download e impressão, a título de divulgação da produção técnico-científica no IFG.

Identificação da Produção Técnico-Científica

[ ] Tese	[ ] Artigo Científico
[ ] Dissertação	[ ] Capítulo de Livro
[ ] Monografia – Especialização	[ ] Livro
[x] TCC - Graduação	[ ] Trabalho Apresentado em Evento
[ ] Produto Técnico e Educacional - Tipo: _____	

Nome Completo do Autor: Ítalo Freire Santos
Matrícula: 20181080080210
Título do Trabalho: Redução do Tempo de Processamento na Importação de Dados Bancários com Computação Paralela: Uma Simulação

Autorização - Marque uma das opções

1. (x) Autorizo disponibilizar meu trabalho no Repositório Digital do IFG (acesso aberto);
2. ( ) Autorizo disponibilizar meu trabalho no Repositório Digital do IFG somente após a data ___/___/_____(Embargo);
3. ( ) Não autorizo disponibilizar meu trabalho no Repositório Digital do IFG (acesso restrito).

Ao indicar a opção **2 ou 3**, marque a justificativa:

( ) O documento está sujeito a registro de patente.
( ) O documento pode vir a ser publicado como livro, capítulo de livro ou artigo.
( ) Outra justificativa: _____

DECLARAÇÃO DE DISTRIBUIÇÃO NÃO-EXCLUSIVA

O/A referido/a autor/a declara que:

i. o documento é seu trabalho original, detém os direitos autorais da produção técnico-científica e não infringe os direitos de qualquer outra pessoa ou entidade;
ii. obteve autorização de quaisquer materiais inclusos no documento do qual não detém os direitos de autor/a, para conceder ao Instituto Federal de Educação, Ciência e Tecnologia de Goiás os direitos requeridos e que este material cujos direitos autorais são de terceiros, estão claramente identificados e reconhecidos no texto ou conteúdo do documento entregue;
iii. cumpriu quaisquer obrigações exigidas por contrato ou acordo, caso o documento entregue seja baseado em trabalho financiado ou apoiado por outra instituição que não o Instituto Federal de Educação, Ciência e Tecnologia de Goiás.

Luziânia - GO, 30/06/2025.

gov.br Documento assinado digitalmente
ITALO FREIRE SANTOS
Data: 30/06/2025 15:17:00-0300
Verifique em https://validar.iti.gov.br

_____
Assinatura do Autor e/ou Detentor dos Direitos Autorais

Instituto Federal de Educação, Ciência e Tecnologia de Goiás
Bacharelado em Sistemas de Informação

Ítalo Freire Santos

Redução do Tempo de Processamento na Importação de Dados Bancários com Computação Paralela: Uma Simulação

Trabalho de Conclusão de Curso apresentado à Coordenação do Curso de Bacharelado em Sistemas de Informação do Instituto Federal de Educação, Ciência e Tecnologia de Goiás - Câmpus Luziânia, como requisito para obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Daniel Vitor de Lucena

Luziânia
2025

S237r Santos, Ítalo Freire

Redução do tempo de processamento na importação de dados bancários com computação paralela: uma silulação / Ítalo Freire Santos. – Luziânia: IFG, 2025.
51 f. : il. color.

Orientador: Prof. Dr. Daniel Vitor de Lucena.
TCC (Graduação - Bacharelado em Sistemas de Informação).
Instituto Federal de Goiás - IFG, Câmpus Luziânia, 2025.

1. Ciência da computação. 2. Processamento eletrônico de dados 3. Programação paralela (Computação) I. Título. II. Lucena, Daniel Vitor de, orient.

Bibliotecário: Helio Lino Delfino – CRB-1/3031

ATA DE DEFESA DE TRABALHO DE CONCLUSÃO DE CURSO

Na presente data realizou-se a sessão pública de defesa do Trabalho de Conclusão de Curso intitulada **Redução do Tempo de Processamento na Importação de Dados Bancários com Computação Paralela: Uma Simulação**, sob orientação de Daniel Vitor de Lucena , apresentada pelo aluno **Ítalo Freire Santos (20181080080210)** do Curso **Bacharelado em Sistemas de Informação (Câmpus Luziânia)**. Os trabalhos foram iniciados às **17:00** do dia **19/02/2025** pelo Professor presidente da banca examinadora, constituída pelos seguintes membros:

- **Daniel Vitor de Lucena** (Presidente)
- **Lucas de Almeida Ribeiro** (Examinador Interno)
- **Audir da Costa Oliveira Filho** (Examinador Interno)
- **Christiane Borges Santos** (Examinadora Suplente Interna)

A banca examinadora, tendo terminado a apresentação do conteúdo do Trabalho de Conclusão de Curso, passou à argüição do candidato. Em seguida, os examinadores reuniram-se para avaliação e deram o parecer final sobre o trabalho apresentado pelo aluno, tendo sido atribuído o seguinte resultado:

[X] Aprovado [ ] Reprovado Nota : 8,5

Observação / Apreciações:

Realizar as observações sugeridas pela banca.

Proclamados os resultados pelo presidente da banca examinadora, foram encerrados os trabalhos e, para constar, eu **Daniel Vitor de Lucena** lavrei a presente ata que assino juntamente com os demais membros da banca examinadora.

Documento assinado eletronicamente por:

- **Lucas de Almeida Ribeiro** (028.179.561-41), em **02/06/2025 21:42:53** com chave **af1639de401311f08925005056a537a4**.
- **Audir da Costa Oliveira Filho** (524.878.062-49), em **03/06/2025 08:49:12** com chave **c494e348407011f0bb18005056a537a4**.
- **Christiane Borges Santos** (022.564.481-93), em **02/06/2025 22:21:22** com chave **0f2334f8401911f0a376005056a537a4**.
- **Daniel Vitor de Lucena** (986.409.841-15), em **02/06/2025 21:41:08** com chave **7066a200401311f0a695005056a537a4**.

Este documento foi emitido pelo SUAP. Para comprovar sua autenticidade, faça a leitura do QrCode ao lado ou acesse https://suap.ifg.edu.br/comum/autenticar_documento/ e informe os dados a seguir.

Tipo de Documento: Ata de Projeto Final

Data da Emissão: 03/06/2025

Código de Autenticação: 6fdb4a

Dedico este trabalho a Deus. Sem Ele, nada seria possível.

AGRADECIMENTOS

Primeiramente, agradeço a Deus, pela força e sabedoria concedidas ao longo desta jornada.

Agradeço à minha família, pelo apoio e incentivo em todos os momentos, sem os quais esta conquista não seria possível.

Aos meus amigos, que estiveram ao meu lado nos desafios e compartilharam comigo cada etapa deste percurso.

Meu profundo agradecimento ao meu orientador, Prof. Dr. Daniel Vitor de Lucena, pela paciência, dedicação e valiosas orientações, que foram fundamentais para a realização deste e dos futuros trabalhos.

Agradeço também aos membros da banca, Audir da Costa Oliveira Filho e Lucas de Almeida Ribeiro, pelas contribuições e considerações que enriqueceram ainda mais este estudo.

RESUMO

Ítalo Freire Santos. **Redução do Tempo de Processamento na Importação de Dados Bancários com Computação Paralela: Uma Simulação. Luziânia, 2025. 51p. Metodologia Científica. Instituto Federal de Educação, Ciência e Tecnologia de Goiás**

A crescente demanda por processamento eficiente de dados bancários devido ao aumento de transações financeiras impulsionou a pesquisa de técnicas de programação paralela. Este trabalho teve como objetivo desenvolver uma solução escalável para importação de dados financeiros, reduzindo o tempo de processamento e garantindo a integridade das operações. O estudo começou com uma revisão bibliográfica sobre programação paralela e distribuiu tarefas em um modelo que utiliza subdivisão de arquivos e múltiplas *threads*. A solução foi implementada em Java e integrada ao framework Spring, sendo validada por meio de testes com dados simulados. Os resultados mostraram uma redução de até 95,7% no tempo de processamento em comparação ao modelo sequencial, comprovando a eficiência do paralelismo. A contribuição deste estudo inclui uma metodologia replicável que pode ser aplicado em sistemas financeiros e outros contextos de alto volume de dados.

Palavras-chave: Programação Paralela, Processamento de Dados Bancários, Escalabilidade.

ABSTRACT

The growing demand for efficient processing of banking data due to the increase in financial transactions has driven research into parallel programming techniques. This work aimed to develop a scalable solution for importing financial data, reducing processing time and ensuring the integrity of operations. The study began with a literature review on parallel programming and distributed tasks in a model that uses file subdivision and multiple threads. The solution was innovative in Java and integrated with the Spring framework, being validated through tests with simulated data. The results showed a reduction of up to 95.7% in processing time compared to the sequential model, proving the efficiency of parallelism. The contribution of this study includes a replicable methodology that can be applied in financial systems and other high-volume data contexts.

Keywords Parallel Programming, Banking Data Processing, Scalability.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SGBD	Sistema de Gerenciamento de Banco de Dados
MPI	Message Passing Interface
MIMD	Multiple Instruction, Multiple Data
SISD	Single Instruction, Single Data
SIMD	Single Instruction, Multiple Data
MISD	Multiple Instruction, Single Data
JPA	Java Persistence API
WORA	Write Once, Run Anywhere
JVM	Java Virtual Machine
GIL	Global Interpreter Lock

SUMÁRIO

1 INTRODUÇÃO

Avanços tecnológicos desempenham um papel preponderante na expansão das transações bancárias, sobretudo aquelas efetuadas mediante o uso de cartões de crédito, débito e pré-pagos. No Brasil, a utilização de meios digitais para operações financeiras apresentou crescimento acelerado, impulsionado por inovações tecnológicas e alterações regulatórias (BRASIL, 2022). Esse crescimento é impulsionado pelas vantagens que esses meios de pagamento oferecem, destacando-se a segurança, praticidade e conveniência que proporcionam aos usuários em seu dia a dia financeiro (EMBRACON, 2023). Contudo, o aumento no volume de transações implica um fluxo expressivo de dados bancários, evidenciando a necessidade de processos de alta eficiência por parte das instituições financeiras.

Entre 2023 e 2024, o volume de transações de pagamento no Brasil cresceu aproximadamente 21,72% no primeiro semestre, passando de 17,01 bilhões em 2023 para 20,71 bilhões em 2024(BRASIL, 2025a). O crescimento persistiu ao longo de ambos os anos, refletindo a maior adoção de pagamentos por cartões no país (BRASIL, 2024). Este crescimento apresentou desafios consideráveis ao processamento de dados, principalmente em sistemas que ainda operam com processamento sequencial. O processamento sequencial é um modelo em que as operações são realizadas em sequencia, ou seja, cada tarefa precisa aguardar a finalização da anterior para dar início à próxima (TANENBAUM, 2007). Essa abordagem sequencial, pode configurar um elemento restritivo em sistemas de grande porte, uma vez que ambientes com elevado volume de transações podem apresentar pontos críticos que comprometem a eficiência do sistema como um todo.

Para enfrentar os desafios inerentes aos processos baseados no processamento sequencial, as abordagens de computação paralela e distribuída apresentam soluções promissoras. O processamento paralelo consiste na execução simultânea de múltiplas operações ou tarefas, dividindo um problema complexo em partes menores que podem ser processadas simultaneamente por vários processadores (PACHECO, 2011a). Isso reduz significativamente o tempo total de processamento ao aproveitar a capacidade de múltiplos núcleos ou processadores. Por outro lado, o processamento distribuído implica a segmentação do sistema em várias unidades computacionais independentes, que cooperam na resolução de um problema. Cada unidade processa uma parcela do problema e compartilha os resultados com as demais unidades, permitindo que grandes volumes de dados sejam administrados e processados de forma eficiente por meio da rede(TANENBAUM, 2007). Essas abordagens podem ajudar a mitigar os gargalos associados ao processamento sequencial e oferecer a escalabilidade necessária para lidar com o crescimento acelerado das transações financeiras.

O aumento contínuo no volume de transações bancárias, aliado à necessidade de um processamento mais ágil e escalável, evidencia a importância de uma abordagem eficiente para lidar com dados críticos. Para instituições financeiras, o crescimento das transações apresenta desafios, especialmente no que diz respeito à importação e integração desses dados em seus sistemas internos.

Dentre os principais entraves enfrentados destaca-se o tempo necessário para processar grandes volumes de informações provenientes de parceiros comerciais. Esse cenário demanda soluções que reduzam os tempos de execução e contribuem para a escalabilidade do sistema, assegurando o cumprimento das exigências operacionais e os elevados padrões de eficiência requeridos pelos clientes das instituições bancárias.

Com base no historico de crescimento do volume de transações (BRASIL, 2025a), estima-se que

a utilização de uma abordagem de processamento sequencial poderá exceder o tempo máximo permitido para a conclusão oportuna de todos os processos dependentes desse processamento dentro da instituição bancária. Dessa forma, a implementação de soluções que reduzam o tempo de processamento torna-se crucial para garantir que os dados sejam tratados dentro dos prazos estipulados. É crucial considerar os impactos que essa degradação na eficiência operacional pode ter sobre diversas áreas da instituição financeira. Os efeitos não se restringem apenas ao aumento do tempo de processamento, mas também se estendem a aspectos críticos como a geração de relatórios financeiros, a satisfação do cliente e a integridade de processos relacionados. Atrasos na importação de dados podem comprometer a precisão e a pontualidade dos relatórios contábeis, dificultando a tomada de decisões gerenciais e operacionais. Além disso, a confiança e a reputação da instituição financeira podem ser seriamente afetadas por falhas no processamento de transações, o que pode gerar um aumento no volume de consultas e reclamações ao serviço de atendimento ao cliente, sobrecarregando a equipe de suporte e afetando a qualidade do atendimento. Esses impactos se refletem ainda em processos relacionados, como o fechamento de faturas e o estorno de transações errôneas, onde os atrasos podem causar inconvenientes tanto para os clientes quanto para a própria instituição financeira.

Para mitigar esses problemas, aprimorar a eficiência operacional no setor financeiro é fundamental. A integração ágil de grandes volumes de transações diárias é essencial para melhorar a experiência do cliente e permitir uma resposta rápida às condições de mercado. Nesse contexto, desenvolver uma solução de importação de dados bancários em formato texto para um Sistema de Gerenciamento de Banco de Dados (SGBD), com foco na redução do tempo de processamento e na capacidade de lidar com o aumento contínuo de transações financeiras, torna-se um objetivo estratégico. Essa abordagem não apenas otimiza a execução dos processos, mas também fortalece a posição competitiva da instituição no dinâmico setor financeiro.

O problema do alto tempo de processamento baseia-se nas hipóteses de que a implementação de técnicas de processamento paralelo permitirá uma redução no tempo necessário para importar dados bancários em formato texto. Além disso, a distribuição da carga de trabalho entre múltiplos núcleos ou processadores contribuirá para a otimização das operações. Por fim, a utilização de técnicas de computação paralela garantirá o uso pleno e eficiente dos recursos computacionais disponíveis, maximizando a utilização do hardware e minimizando a ociosidade dos processadores.

A implementação de soluções escaláveis em sistemas financeiros apresenta uma série de benefícios estratégicos essenciais. Primeiramente, a escalabilidade proporciona uma capacidade aprimorada para lidar com picos de carga, assegurando que os sistemas mantenham um desempenho consistente mesmo durante períodos de alta demanda. Este atributo é crucial para preservar a integridade das operações financeiras diárias, garantindo que os serviços aos clientes não sejam interrompidos em momentos críticos.

Investir em escalabilidade significa estar preparado para o futuro crescimento do volume de transações. Ao desenvolver e implementar soluções eficientes agora, as instituições financeiras se posicionam para lidar com esse aumento sem comprometer a qualidade do serviço oferecido. Isso não apenas fortalece a infraestrutura tecnológica, mas também melhora a resiliência e a adaptabilidade do sistema frente a novos desafios e oportunidades no mercado financeiro.

A capacidade de oferecer serviços ágeis e confiáveis através de uma infraestrutura escalável coloca as instituições financeiras em uma posição competitiva mais forte. Em um mercado dinâmico e exigente, a eficiência operacional e a confiabilidade dos serviços são diferenciais decisivos para atrair e reter clientes. Portanto, investir em escalabilidade não apenas protege as operações atuais, mas também prepara para uma posição de liderança no mercado financeiro global.

Observa-se uma lacuna nas práticas do mercado no que diz respeito à aplicação eficiente de

técnicas de computação paralela e distribuída para enfrentar a crescente demanda por processamento de dados em larga escala. Essa lacuna se reflete na necessidade de otimizar os sistemas existentes para que possam lidar com volumes de dados cada vez maiores de maneira eficaz. A motivação para abordar esse tema reside na oportunidade de impulsionar a inovação tecnológica, desenvolvendo soluções que não apenas melhorem o desempenho dos sistemas atuais, mas também estabeleçam bases para futuras pesquisas e práticas avançadas.

Este trabalho visa alinhar o interesse nas tecnologias emergentes com a necessidade prática das instituições financeiras de aprimorar seus processos. Espera-se que os resultados deste estudo possam ser diretamente aplicados para otimizar o processamento de dados bancários, proporcionando uma contribuição significativa para o setor financeiro. A relevância e a atualidade do tema são fundamentais para orientar a pesquisa e análise ao longo de todo o processo, buscando um impacto positivo no desempenho dos sistemas de processamento de dados.

1.1 Objetivos Específicos

1. Realizar uma revisão bibliográfica abrangente sobre soluções de computação de alto desempenho e processamento paralelo aplicadas à importação de dados bancários em formato texto.
2. Elaborar uma análise estatística abrangente do tempo de processamento da importação de dados, bem como uma previsão dos requisitos computacionais para os próximos cinco anos, com base no aumento esperado de transações.
3. Pesquisar e analisar soluções tecnológicas e estratégias que possam melhorar o desempenho e a eficiência na importação de dados bancários.
4. Projetar uma solução proposta, nova versão, que inclua uma arquitetura de sistema e algoritmos para a importação de dados em formato texto.
5. Implementar a solução proposta com base no projeto elaborado, garantindo que a solução seja funcional e eficiente.
6. Realizar simulações da importação de dados, comparando o desempenho da solução que utiliza processamento sequencial com a solução proposta e coletando dados para análise.

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção apresenta um referencial teórico que fundamenta a execução deste projeto, explorando os principais conceitos e abordagens relevantes. Serão abordados temas essenciais, como processamento paralelo e técnicas de otimização de desempenho, proporcionando uma base teórica sólida para a compreensão e desenvolvimento da solução proposta.

2.1 Estrutura dos Arranjos de Pagamento

As transações com cartões de pagamento são amplamente utilizadas no varejo e podem ser classificadas em três categorias principais: crédito, débito e pré-pago, cada uma com características e funcionalidades específicas. O cartão de crédito possibilita o pagamento posterior à compra, enquanto o cartão de débito desconta imediatamente o valor da conta do titular. Já o cartão pré-pago exige recarga antecipada com um valor pré-determinado. Esses meios de pagamento são essenciais para o comércio, facilitando compras tanto em estabelecimentos físicos quanto em plataformas online (GARIBALDI, 2023).

Para que essas transações ocorram de maneira segura e eficiente, elas são regulamentadas dentro de um sistema denominado arranjo de pagamento. Esse sistema organiza e gerencia a troca de valores entre usuários de cartões, garantindo a liquidação das transações financeiras. Desde a solicitação do pagamento até sua confirmação, o arranjo de pagamento desempenha um papel fundamental no funcionamento da economia moderna. Seu objetivo principal é assegurar que as operações financeiras sejam realizadas conforme as normas estabelecidas para cada tipo de serviço. Um exemplo disso ocorre no uso de cartões de crédito, onde o comprador e o vendedor são conectados por meio de uma bandeira de pagamento, responsável por intermediar e viabilizar a transação (GARIBALDI, 2023).

Os arranjos de pagamento podem ser categorizados conforme os serviços prestados e a natureza das transações. Entre os principais componentes, destacam-se as bandeiras de cartões, como Visa, MasterCard, Elo e American Express, que conectam emissores, instituições financeiras e empresas aos estabelecimentos comerciais e usuários. Cada bandeira estabelece suas próprias regras para garantir que as transações sejam conduzidas com eficiência, segurança e confiabilidade. A diversidade de bandeiras no mercado proporciona aos consumidores mais opções de pagamento, enquanto os comerciantes podem escolher quais delas aceitar, considerando tarifas e condições específicas (GARIBALDI, 2023).

Outro elemento fundamental são os emissores de cartões, instituições financeiras responsáveis por disponibilizar e administrar os cartões. Esses emissores desempenham um papel central na cadeia do mercado de pagamentos, sendo responsáveis por identificar e habilitar os titulares, autorizar transações, definir limites de crédito ou saldo disponível e realizar a cobrança de faturas. Além disso, podem oferecer programas de benefícios, como pontos ou milhas, para fidelizar os consumidores. Dessa forma, os emissores estabelecem uma conexão direta com os clientes, influenciando a aceitação dos cartões no mercado e garantindo a segurança das operações financeiras (GARIBALDI, 2023).

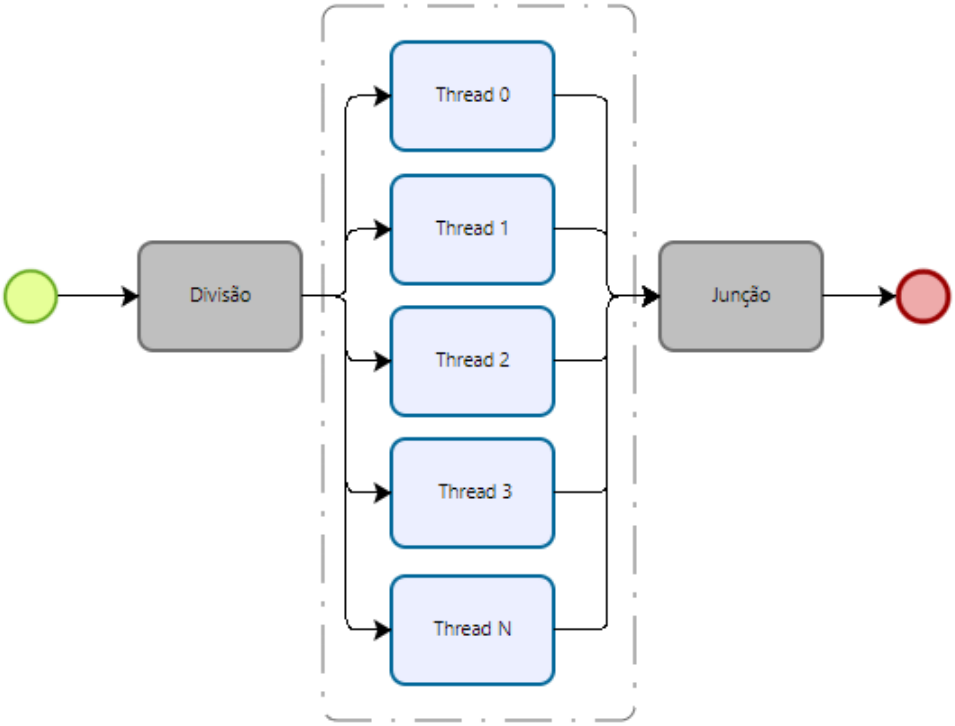
A abordagem sobre a estrutura de arranjos de pagamento estabelece um entendimento das características fundamentais das operações realizadas no mercado financeiro, como crédito, débito e pré-pago. Esses conceitos são essenciais para compreender os requisitos operacionais e tecnológicos necessários para garantir a eficiência e segurança nas transações financeiras. A descrição dos arranjos de pagamento também foi integrada para contextualizar como diferentes agentes, como emissores e bandeiras, interagem dentro desse ecossistema para facilitar e regular as transações por meio de métodos de estatística preditiva

para mitigar riscos operacionais e fraudes.

2.2 Processamento Paralelo

Programação paralela é uma estratégia que permite obter o resultado de um algoritmo ao dividir e distribuir sua execução em partes específicas. Esse processo pode ser realizado através de pequenas parcelas do algoritmo, executadas por *threads* distintas, com a junção dos resultados ao final da execução. Esse modelo é conhecido como fork-join, conforme ilustrado na Figura 1.

Figura 1 – Representação de Algoritmos Paralelos Utilizando o Modelo Fork-Join.



Fonte: Adapdado de (PACHECO, 2011a; MAROWKA, 2007)

Por outro lado, a Figura 2 ilustra a execução de um algoritmo de forma sequencial, conduzido por um único processo, sem divisão em tarefas paralelas. Essa abordagem é mais simples, mas pode ser menos eficiente em comparação com o modelo paralelizado, especialmente para tarefas que podem ser divididas em subtarefas independentes (BRESHEARS, 2009; PACHECO, 2011a).

Figura 2 – Representação da Execução de Algortimo Sequencial.

Fonte: Adaptado de (PACHECO, 2011a)

Outro modelo comum na programação paralela é a divisão de tarefas em processos, onde cada processo é responsável por executar um trecho do código previamente dividido (PACHECO, 2011a). De acordo com a taxonomia de Flynn (FLYNN, 1972), um dos modelos de processamento paralelo é o Multiple Instruction, Multiple Data (MIMD). Nesse modelo, vários processadores autônomos executam simultaneamente diferentes instruções com dados compartilhados ou distintos (EL-REWINI; ABD-EL-BARR, 2005).

Existem dois principais tipos de sistemas MIMD:

1. **Memória compartilhada**: Um conjunto de processadores autônomos conectados a um sistema de memória através de uma rede, permitindo que cada processador acesse qualquer localização da memória.
2. **Memória distribuída**: Cada processador é emparelhado com sua própria memória, formando um par, processador e memória.

Além do modelo MIMD, Flynn definiu outros três modelos:

1. ***Single Instruction, Single Data (SISD)***: Um único fluxo de instruções opera sobre um único fluxo de dados, característico de processamento sequencial, com uma única unidade de controle.
2. ***Single Instruction, Multiple Data (SIMD)***: Vários dados são processados sob um único comando de uma instrução. Embora execute operações com múltiplos dados, mantém características de programas sequenciais, oferecendo compartilhamento de resultados.
3. ***Multiple Instruction, Single Data (MISD)***: Múltiplas etapas ou processos de controle executam operações distintas sobre um mesmo ponto. Este modelo é considerado impraticável com a tecnologia atual.

A solução implementada baseou-se no modelo MIMD, destacado na taxonomia de Flynn. Esse modelo permite que múltiplos processadores autônomos executem instruções distintas sobre diferentes conjuntos de dados simultaneamente. Essa abordagem foi aplicada por meio da divisão dos arquivos de entrada em partes menores, cada uma processada de forma independente em *threads* separadas. O uso de *threads* foi gerenciado pelo framework Spring, utilizando o ThreadPoolTaskExecutor para criar e gerenciar *pools* de *threads*, garantindo o balanceamento da carga e a eficiência no uso dos recursos computacionais.

Além disso, a estratégia de processamento adotou o modelo fork-join, no qual as tarefas são subdivididas em subtarefas menores e, posteriormente, seus resultados são combinados para obter o resultado final. Essa técnica foi aplicada na fase de fracionamento dos arquivos, onde cada bloco de dados era processado de forma independente, permitindo a execução simultânea em múltiplos núcleos do processador.

2.2.1 O Surgimento da Computação Paralela

A resolução de diversos problemas computacionais tornou-se inviável com o aumento da carga de dados e a complexidade dos algoritmos, estendendo significativamente o tempo de execução quando executados de forma serial. Nesse cenário, pesquisadores e instituições, principalmente dos Estados Unidos e Europa, iniciaram estudos que resultaram na criação do Message Passing Interface (MPI) (FOSTER, 1995). Muitos fabricantes de processadores também participaram dessa pesquisa, padronizando a troca de mensagens em ambientes de memória distribuída, liderados pelo Center for Research on Parallel Computing em 1992 (PACHECO, 2011b).

Entre 1986 e 2002, o desempenho dos microprocessadores aumentou em média 50% ao ano. No entanto, a partir de 2002, houve uma desaceleração de cerca de 20% ao ano nos processadores single-processor. Esta desaceleração foi associada a uma mudança dramática na arquitetura dos novos processadores(HENNESSY; PATTERSON, 2011). Na tentativa de aumentar o número de registradores em um único processador, a indústria enfrentou desafios relacionados ao aquecimento e ao tamanho dos registradores (SIEGEL, 1990).

Desde 2005, a maioria das empresas responsáveis pela construção de microprocessadores decidiu adotar o paralelismo na execução de processos computacionais para obter ganhos significativos, em vez de continuar tentando melhorar os processadores monolíticos (GRAMA et al., 2003). Essa decisão foi crucial para o desenvolvimento da computação moderna, pois permitiu obter melhorias significativas em programas que anteriormente eram executados de forma serial, ou seja, construídos para rodar em uma única *thread*.

Com a estratégia adotada pela indústria de microprocessadores, foi possível realizar operações muito complexas, como a decodificação do genoma humano, o desenvolvimento de máquinas de busca, e a criação de jogos computacionais com inteligência artificial. Sem essa abordagem, esses avanços seriam impossíveis.

A inclusão da história do processamento paralelo na fundamentação teórica tem como objetivo contextualizar o surgimento e a evolução dessa abordagem no campo da computação, evidenciando sua relevância para a resolução de problemas complexos e de alta demanda computacional. Ao apresentar os avanços históricos, buscou-se demonstrar como o processamento paralelo emergiu como uma resposta às limitações do processamento serial, especialmente diante do crescimento de dados e da complexidade dos sistemas modernos. Esse panorama histórico reforça a importância de adotar técnicas de paralelismo no trabalho, conectando os desafios atuais enfrentados no processamento de dados bancários com soluções consolidadas e eficazes desenvolvidas ao longo do tempo.

2.3 Linguagem de Programação Java

Java é uma linguagem de programação de alto nível amplamente utilizada, reconhecida por sua portabilidade, desempenho robusto e vasto ecossistema de ferramentas que suportam diversas áreas do desenvolvimento de software. Criada pela Sun Microsystems em 1995 (GOSLING BILL JOY, 1995) e atualmente mantida pela Oracle Corporation (CORPORATION, 2024), Java é baseada no paradigma de orientação a objetos, oferecendo uma estrutura modular e reutilizável que facilita o desenvolvimento e a manutenção de sistemas.

A filosofia *Write Once, Run Anywhere* (WORA) destaca-se como um dos principais atributos de Java, possibilitando a execução de programas em qualquer plataforma que possua uma Máquina Virtual Java (JVM) (SCHILDT, 2021). Essa característica transformou Java em uma escolha natural para o desenvolvimento de aplicações empresariais robustas e sistemas escaláveis (BLOCH, 2017).

Além disso, o Java destaca-se como uma linguagem de programação que fornece um amplo suporte para o gerenciamento eficiente de *threads* e o processamento paralelo através de ferramentas como o ExecutorService (GOETZ et al., 2006). A abstração do gerenciamento de *threads* por meio de *pools* otimiza os recursos do sistema e reduz os custos associados à criação e destruição frequente de *threads*, além de simplificar o ciclo de vida das mesmas (LEA, 2000). Adicionalmente, o pacote `java.util.concurrent` e frameworks como Fork-Join fornecem mecanismos robustos para dividir e executar tarefas simultaneamente, enquanto bibliotecas modernas, como a Stream API, oferecem interfaces de alto nível para manipulação paralela de coleções de dados (ORACLE, 2024).

No contexto deste trabalho, Java foi escolhido devido à sua robustez, confiabilidade e integração com as demais tecnologias. A linguagem também oferece suporte nativo para manipulação de arquivos, fundamental para o processamento paralelo implementado. Além disso, sua natureza multiplataforma e vasta comunidade de desenvolvedores garantem suporte contínuo e a possibilidade de expandir as soluções desenvolvidas.

2.4 Spring Framework

O Spring Framework é um dos mais amplamente utilizados *frameworks* para o desenvolvimento de aplicações Java, oferecendo um ecossistema robusto e flexível para criar soluções modernas e eficientes. Originalmente criado por Rod Johnson em 2003, o Spring visa simplificar o desenvolvimento de aplicações Java ao fornecer suporte para integração, gerenciamento de dependências e escalabilidade. Sua arquitetura modular permite que os desenvolvedores escolham apenas os componentes necessários, promovendo flexibilidade e eficiência no desenvolvimento de software (JOHNSON, 2003).

A arquitetura do Spring é organizada em torno de vários módulos que fornecem funcionalidades específicas. Esses módulos são agrupados em camadas, cada uma projetada para atender a diferentes aspectos do desenvolvimento de software (FRAMEWORK, 2024). A seguir, destacam-se os principais módulos:

1. **Spring Core** O módulo Spring Core fornece funcionalidades essenciais, como a Inversão de Controle e a Injeção de Dependências, que permitem criar aplicações desacopladas e mais fáceis de testar. Ele é o núcleo do framework e serve de base para os outros módulos (FRAMEWORK, 2024).
2. **Spring Data** O Spring Data é um módulo que simplifica o acesso a dados, oferecendo uma abstração sobre tecnologias de persistência, como Java Persistence API (JPA), MongoDB, Cassandra, entre outras. Ele reduz a necessidade de escrever código repetitivo ao lidar com operações de banco de dados (FRAMEWORK, 2024).
3. **Spring Batch** O Spring Batch é um módulo do ecossistema Spring projetado especificamente para o desenvolvimento de aplicações de processamento em lote robustas, escaláveis e reutilizáveis. Este módulo fornece uma estrutura extensível para a criação de aplicações que executam tarefas em lotes, como processamento de grandes volumes de dados, geração de relatórios, transferências de arquivos e operações de integração de dados (FRAMEWORK, 2024). A seguir, destacam-se os principais recursos do Spring Batch:

    a) **Configuração Declarativa**: O Spring Batch utiliza o modelo de programação do Spring Framework, permitindo que as tarefas sejam configuradas de forma declarativa através de XML ou anotações Java. Isso facilita o desenvolvimento e a manutenção do código.

    b) **Modelo de Processamento em Três Etapas**: O Spring Batch divide o processamento em lote em três etapas principais:

    - **Leitura**: Responsável por obter dados de fontes como bancos de dados, arquivos ou APIs externas.
    - **Processamento**: Aplica lógica de negócio ou transforma os dados lidos.
    - **Gravação**: Insere os dados processados em destinos como bancos de dados, arquivos ou outras aplicações.

    c) **Gestão de Execuções e Estado**: O Spring Batch oferece suporte à persistência de metadados sobre as execuções de tarefas em um banco de dados. Isso permite que tarefas sejam pausadas, retomadas e monitoradas, garantindo a consistência e rastreabilidade dos processos.

d) **Controle de Transações**: Por meio da integração com a plataforma Spring, o Spring Batch fornece suporte completo para transações, permitindo rollback e commit para garantir que os dados sejam processados corretamente mesmo em caso de falhas.

e) **Escalabilidade**: O Spring Batch é projetado para lidar com grandes volumes de dados, com suporte para execução paralela e processamento distribuído em clusters, permitindo que as tarefas sejam executadas de forma eficiente em ambientes com alta demanda.

Com o crescimento e a complexidade das aplicações modernas, é essencial contar com mecanismos eficientes para gerenciar a execução de tarefas assíncronas. O `ThreadPoolTaskExecutor` do Spring permite otimizar o uso de recursos do sistema, melhorando a performance e a escalabilidade das aplicações. Para um controle eficiente do *pool* de *threads*, alguns parâmetros são fundamentais:

1. ***Core Pool Size***: Número mínimo de *threads* que serão mantidas no *pool*, mesmo que não haja tarefas a serem executadas. Este valor garante que sempre haja um conjunto básico de *threads* prontas para execução.
2. ***Max Pool Size***: Número máximo de *threads* que podem ser criadas no *pool*, determinando o limite superior de *threads* ativas para processar as tarefas. Esse valor deve ser ajustado conforme os recursos disponíveis no sistema.
3. ***Queue Capacity***: Capacidade da fila de tarefas que aguardam execução. Quando o número de *threads* ativas atingir o limite máximo, as tarefas adicionais serão colocadas na fila até que uma *thread* esteja disponível para processá-las.
4. ***Thread Name Prefix***: Prefixo utilizado para nomear as *threads* gerenciadas pelo `ThreadPoolTaskExecutor`, facilitando o monitoramento e rastreamento das *threads* no ambiente de execução.

A escolha do Spring Framework como base tecnológica deve-se à sua flexibilidade, modularidade e capacidade de escalar soluções robustas. O uso do Spring Batch foi essencial para estruturar o processamento em lote, permitindo a execução paralela de tarefas e o aproveitamento eficiente dos recursos computacionais. Além disso, recursos como controle de transações e gestão de estado garantiram a confiabilidade e consistência do sistema.

A integração com módulos como o Spring Data simplificou o acesso e a persistência dos dados, reduzindo a complexidade no desenvolvimento. Dessa forma, o Spring Framework proporcionou uma base tecnológica eficiente, atendendo aos objetivos de escalabilidade e desempenho do trabalho.

2.5 Modelo Holt-Winters

A análise preditiva desempenha um papel fundamental na estimativa do crescimento do volume de transações bancárias e na avaliação de seu impacto sobre os sistemas de processamento de dados. Para este estudo, utilizou-se o método Holt-Winters, também conhecido como Suavização Exponencial Tripla, amplamente empregado para previsão de séries temporais devido à sua capacidade de modelar tendências e padrões sazonais.

O software Microsoft Excel implementa o modelo Holt-Winters por meio das funções PREVISÃO.ETS e PREVISÃO.ETS.CONFINT, que permitem realizar projeções baseadas em séries temporais e calcular intervalos de confiança para as estimativas (MICROSOFT, 2024). Essas funções foram aplicadas aos dados históricos de transações bancárias para identificar tendências futuras e avaliar a demanda

esperada sobre o sistema de importação de dados bancários, descrito pela Equação 2.1 (CHATFIELD, 1978).

$$F(t+m) = (L_t + m \cdot T_t) \times S_{t-p+((m-1) \bmod p)+1} \tag{2.1}$$

onde L_t é o nível estimado no período t, T_t é a tendência estimada no período t, S_i é o índice sazonal correspondente, p representa o período sazonal e m é o número de períodos futuros que se deseja prever.

O uso do modelo Holt-Winters por meio das funções PREVISÃO.ETS e PREVISÃO.ETS.CONFINT possibilitou uma análise quantitativa do crescimento das transações bancárias e sua influência sobre o desempenho do sistema. Essa abordagem permitiu estimar a demanda futura e justificar a adoção de técnicas de processamento paralelo para atender ao volume crescente de dados, garantindo a eficiência e escalabilidade do sistema de importação de dados bancários.

3 DADOS E MÉTODOS

Este capítulo apresenta os dados utilizados no estudo e descreve os procedimentos metodológicos adotados no desenvolvimento deste trabalho. A metodologia está estruturada em etapas sequenciais, englobando desde a análise da abordagem de processamento sequencial até a implementação e validação das soluções propostas.

3.1 Dados Bancários

Os dados utilizados neste estudo foram simulados com base nos registros quantitativos de transações de meios de pagamento disponibilizados pelo Banco Central do Brasil. Esses registros abrangem informações detalhadas sobre a frequência de operações realizadas no sistema financeiro nacional, fornecendo uma base relevante para a análise de desempenho em ambientes de alta demanda.

A Tabela 1 apresenta a quantidade total de transações realizadas em cada trimestre, juntamente com a média de instituições financeiras envolvidas no processamento. Os dados evidenciam a complexidade do sistema financeiro nacional, destacando trimestres com volumes superiores a 12 bilhões de transações, como observado no terceiro trimestre de 2024, e sua distribuição entre mais de 150 instituições financeiras. Dessa forma, constata-se que o aumento progressivo do volume de transações impõe desafios consideráveis à eficiência do processamento, especialmente em termos de escalabilidade e desempenho do sistema. Além disso, o número elevado de instituições envolvidas reflete a diversidade e a abrangência do sistema financeiro brasileiro, com cada instituição lidando com uma carga média significativa de transações diárias.

Tabela 1 – Quantidade Total de Transações e Instituições Bancárias

Trimestre	**Quantidade de Transações**	**Quantidade de Instituições**
2024-03	12.183.371.776	157
2024-02	9.854.617.801	157
2024-01	10.854.032.086	158
2023-04	9.798.973.102	159
2023-03	10.993.581.275	160
2023-02	8.912.667.049	160
2023-01	8.100.794.578	161
2022-04	9.641.715.980	161
2022-03	9.321.007.479	162
2022-02	9.474.520.428	162
2022-01	8.285.514.381	162
2021-04	8.474.506.544	162
2021-03	7.028.773.066	162
2021-02	6.945.362.589	162
2021-01	6.063.962.431	161
2020-04	6.879.837.979	161
2020-03	5.266.506.753	161
2020-02	4.621.529.095	159
2020-01	4.818.605.622	157
2019-04	5.912.798.469	158
2019-03	4.739.490.099	157
2019-02	4.102.341.218	156
2019-01	2.945.665.845	157

Fonte: (BRASIL, 2025a; BRASIL, 2025b)

As transações apresentadas na Tabela 1 incluem operações realizadas com cartões de débito, crédito e pré-pago. Esses dados trafegam diariamente por meio de arquivos enviados entre parceiros comerciais, como bandeiras e emissores de cartão, até as instituições financeiras. Os arquivos são fornecidos no formato de texto e necessitam de integração precisa e eficiente em um SGBD. Cada registro de transação contém informações detalhadas, como a data e hora da operação, o valor movimentado, a identificação do cliente e o código da operação, que classifica a natureza da transação.

Para identificar a média diária de transações realizadas por instituição no sistema financeiro, foi considerada a divisão do total de transações trimestrais pelo número aproximado de 90 dias em um trimestre. Dessa forma, o cálculo da média diária de transações é formalizada pela Equação 3.1:

$$\text{Transações diárias} = \frac{\text{Total de transações}}{\text{Número de dias no trimestre}} \tag{3.1}$$

Posteriormente, calculada pela divisão do total de transações diárias pelo número de instituições bancárias, conforme mostrado na Equação 3.2:

$$\text{Transações por instituição por dia} = \frac{\text{Transações diárias}}{\text{Número de instituições}} \tag{3.2}$$

Aplicando as Equações 3.1 e 3.2, foi gerada a Tabela 2, a qual apresenta a média de transações diárias realizadas por cada instituição financeira, com base nos dados fornecidos pela Tabela 1. A Tabela 2 destaca quantidade aproximada de carga de trabalho enfrentadas pelas instituições ao longo dos trimestres analisados.

Tabela 2 – Média Diária de Transações por Instituição

Trimestre	**Quantidade de Transações**
2024-03	862.234
2024-02	697.425
2024-01	763.293
2023-04	684.764
2023-03	763.443
2023-02	618.935
2023-01	559.061
2022-04	665.405
2022-03	639.301
2022-02	649.830
2022-01	568.279
2021-04	581.242
2021-03	482.083
2021-02	476.362
2021-01	418.493
2020-04	474.799
2020-03	363.458
2020-02	322.958
2020-01	341.020
2019-04	415.809
2019-03	335.420
2019-02	292.190
2019-01	208.469

Fonte: Autor

A Tabela 3 foi gerada com base nos dados históricos apresentados na Tabela 2, utilizando a Equação 2.1, do método Holt-Winters para prever os valores futuros.

Tabela 3 – Expectativa de Transações por Instituição

Trimestre	**Quantidade de Transações**
2029-04	1.335.600
2029-03	1.294.213
2029-02	1.248.135
2029-01	1.261.164
2028-04	1.219.777
2028-03	1.173.699
2028-02	1.186.729
2028-01	1.145.342
2027-04	1.099.263
2027-03	1.112.293
2027-02	1.070.906
2027-01	1.024.827
2026-04	1.037.857
2026-03	996.470
2026-02	950.392
2026-01	963.421
2025-04	922.034
2025-03	875.956
2025-02	888.985
2025-01	847.598
2024-04	801.520

Fonte: Autor

Para avaliar a eficiência do sistema no processamento do volume diário de transações, é essencial calcular o tempo médio necessário para concluir cada execução do processo. Esse indicador permite identificar se o sistema conseguirá acompanhar o crescimento projetado nas transações diárias sem enfrentar atrasos ou sobrecarga. A fórmula utilizada para a estimativa leva em consideração o número médio de transações por dia e a taxa de processamento em transações por minuto. A metodologia de cálculo do tempo médio diário de processamento ao longo dos próximos anos é baseada na Equação 3.3.

$$P_{\text{diário}} = \frac{T_{\text{diário}}}{R} \tag{3.3}$$

Fonte: Adapdado de (SLACK; CHAMBERS; JOHNSTON, 2009)

onde:

- $P_{\text{diário}}$ é o tempo médio diário de processamento,
- $T_{\text{diário}}$ é o número médio de transações por dia,
- R é a taxa de processamento em transações por minuto.

A Figura 3 apresenta as estimativas projetadas para o volume médio diário de transações, fundamentadas na análise histórica dos dados da média diária de transações por instituição calculados a partir dos dados disponíveis (BRASIL, 2025a; BRASIL, 2025b). O gráfico evidencia a evolução das transações bancárias por instituição entre 2019 e 2029. A linha contínua representa os valores efetivamente registrados até o terceiro trimestre de 2024, enquanto a linha tracejada indica a projeção para os anos

seguintes. Já as linhas pontilhadas superior e inferior delimitam o intervalo de confiança, destacando a faixa de variação esperada para as transações futuras.

Figura 3 – Quantidade de Transações Diárias por Trimestre

Fonte: Autor

3.1.1 Estrutura do Arquivo

Esta subseção apresenta a organização do arquivo de transações bancárias e seus principais componentes. O formato adotado é estruturado, com registros de tamanho fixo dispostos em linhas. Cada linha contém campos organizados conforme posições pré-definidas, de acordo com o tipo de registro. A estrutura geral do arquivo está dividida nas seguintes seções:

3.1.2 Header

O header contém informações de identificação e metadados do arquivo, incluindo código de identificação, data e hora da geração, além do nome da instituição responsável pelo processamento.

```
1 9099997125061      2506100400      0000      00000000000000000INSTCRED      001
```

1. **90** – Identificação do registro tipo header.
2. **99997125061** – Identificação do arquivo.
3. **2506100400** – Data e hora de geração.
4. **INSTCRED** – Nome da instituição responsável.
5. **001** – Código sequencial do arquivo.

3.1.3 Registros de Transações

Cada transação é composta por múltiplos registros, organizados da seguinte forma:

1. **Registro 0500** – Contém detalhes principais da transação, como valores, data e identificação do comércio.
2. **Registro 0501** – Informações adicionais, como códigos internos e status da transação.

3. **Registro 0502** – Informação de moeda e país da transação.

4. **Registro 0505** – Detalhes do processamento, como taxas e identificadores internos.

5. **Registro 0507** – Códigos de segurança e autenticação da transação.

```
1 0500999999000813016100 0Z
    24198955060050885017680100725030301000000001600986000000001600986COMERCIO
    DAS UTILIDADES   CONSELHEIRO PBR 594900000     1008D6560235 1 0750610
2 0501             000000  000000001600
    16707529300000050 1N0P0J000000000000  5061 0  0 V 3
                               0000000000
3 0502             BR    02700000000000000
4 0505305060446140935000000001600986           0000 000000001600
                                 000000000235200C00000000000000000000000000000P
                  0000000000000000 C
5 050700000250301E0F8C8076        5
    CBF5C6B0803200015067DA3F5090FFD010A000000000003A02000000000001600
                      06                                    20700000
```

3.1.4 Divisor de Bloco de Transações

Este registro atua como um separador entre diferentes blocos de transações, contendo informações agregadas sobre as transações processadas até aquele ponto.

```
1 9100999999250610000000000897320000000000160021640000000000061000000
    0000000170000000000000000000000000247780833
```

1. **9100** – Código do registro divisor.

2. **9999995061** – Identificação do bloco.

3. Outros campos – Informações agregadas, como número total de transações e valores somados.

3.1.5 Trailer

O trailer contém informações consolidadas do arquivo, incluindo totais de transações, somatório de valores e metadados de fechamento.

```
1 9200999999506100000404146126300000035051500216400000174562600000 0
    0003802610000000000000000000000005153925583
```

1. **9200** – Código do registro trailer.

2. **9999995061** – Identificação do arquivo.

3. Outros campos – Contagem total de transações, somatório de valores e validações finais.

3.2 Metodologia

A metodologia deste trabalho baseia-se em uma revisão bibliográfica e na escolha de tecnologias e ferramentas para implementar as soluções. A primeira etapa envolveu a análise dos conceitos de transações e arranjos de pagamento, essenciais para entender o contexto financeiro. Em seguida, foram analisados modelos de processamento paralelo, com foco nas arquiteturas MIMD e SIMD, para identificar abordagens eficientes para lidar com grandes volumes de dados. A escolha do Java como linguagem de programação foi motivada pela sua robustez, escalabilidade e ampla adoção em sistemas bancários, garantindo desempenho e integração adequados. O Spring foi selecionado como framework devido à sua capacidade de gerenciar operações em lote de forma eficiente, enquanto o modelo Fork-Join foi explorado como uma estratégia eficaz para o particionamento e execução paralela de tarefas, otimizando o processamento.

3.2.1 Levantamento Bibliográfico e Seleção de Ferramentas de Desenvolvimento

Nesta etapa, o objetivo foi consolidar os fundamentos teóricos que sustentam o desenvolvimento do trabalho. Inicialmente, foi realizada uma revisão sobre Tipos de Transações e Arranjos de Pagamento para compreender o impacto estratégico e negocial do processo no contexto em que está inserido na Seção 2.1. Em seguida, aprofundou-se no estudo da literatura sobre modelos de processamento paralelos, com destaque para os modelos MIMD e SIMD, baseados na taxonomia de Flynn mostrado na Seção 2.2. Esses modelos foram investigados para avaliar como diferentes arquiteturas computacionais podem ser aplicadas na otimização do processamento de grandes volumes de dados.

Durante a revisão, foi realizada uma análise das linguagens disponíveis para o processamento paralelo de grandes volumes de dados, levando à escolha do Java como tecnologia principal para este trabalho. O Java se destacou em relação a outras opções, como Python e C++, devido ao seu equilíbrio entre desempenho, escalabilidade e compatibilidade com sistemas bancários existentes. Diferente do Python, que possui limitações devido ao GIL (*Global Interpreter Lock*) em operações multithread, o Java oferece um modelo robusto de concorrência, permitindo o uso eficiente de múltiplos núcleos de processamento. Já em relação ao C++, embora este tenha alto desempenho, a complexidade do gerenciamento de memória e a falta de um ecossistema padronizado para processamento paralelo tornam o Java uma escolha mais prática para aplicações corporativas. Além disso, a linguagem conta com um amplo suporte da Oracle, uma comunidade ativa e um ecossistema consolidado para o desenvolvimento de aplicações distribuídas. As pesquisas relacionadas a esse tema utilizaram chaves como *Melhores Linguagens de Programação para Processamento Bancário*, *Processamento paralelo em Java*, *Framework de concorrência em Java* e *ExecutorService vs ForkJoinPool no Java*. A Seção 2.3 detalha características importantes da linguagem Java.

A partir da definição da linguagem, foram pesquisadas ferramentas para agilizar o desenvolvimento das implementações, em especial, o frameworks que pudessem auxiliar nessa etapa. O Spring foi selecionado devido à sua capacidade de gerenciar tarefas em lote de forma eficiente, característica essencial para o escopo deste trabalho, como apresentado na Seção 2.4. Para a realização dessa pesquisa, foram utilizadas chaves de busca como *Processamento paralelo no Spring Batch* e *Suporte a múltiplas threads no Spring*.

Além disso, abordagens como o modelo de programação Fork-Join demonstraram alta relevância para o particionamento e execução paralela de tarefas, como apresentado na Seção 2.2. Para esse tópico, as pesquisas foram conduzidas utilizando palavras-chaves como *Exemplo de Fork-Join em Java*, *Parallel Stream vs ForkJoinPool* e *Melhores práticas para computação paralela em Java*.

3.2.2 Implementação e Análise do Modelo de Processamento Sequencial

A avaliação do processo inicia-se com uma implementação do processamento sequencial e análise detalhada do seu funcionamento com o propósito de avaliar o comportamento dessa abordagem e identificar limitações que possam comprometer o desempenho dessa solução em função do crescimento do volume de transações. Essa etapa envolve a implementação de uma solução tradicional de software senguindo o fluxo sequencial de processamento. O objetivo é mensurar o tempo médio de processamento das transações no volume atual e projetar o tempo necessário para atender ao crescimento esperado nos próximos anos, conforme as projeções apresentadas na Figura 3.

Paralelamente é necessário mapear os principais componentes da arquitetura e os fluxos de dados existentes, proporcionando uma visão do funcionamento do sistema de processamento de transações bancárias. Essa abordagem busca verificar se problemas de desempenho estão relacionados a eventuais limitações da infraestrutura.

Essa etapa inicial é essencial para estabelecer uma base sólida para as próximas fases da metodologia, garantindo que os esforços de otimização sejam direcionados de maneira eficaz e que as soluções propostas considerem tanto as limitações técnicas quanto o crescimento projetado do sistema.

3.2.3 Modelagem e Desenvolvimento de Modelos de Processamento Paralelo

A partir a compreensão das limitações inerentes ao processamento sequencial, foram elaborados dois modelos para processamento paralelo com o propósito de minimizar o tempo de processamento. Os modelos propostos foram:

1. **Processamento com múltiplas instâncias operando de forma sequencial com divisão de arquivo de transações:** Consistiu na criação de uma solução para importação e processamento paralelo de arquivos. O objetivo dessa etapa foi adaptar a solução de processamento sequencial, que processava os dados utilizando apenas uma instância e uma *thread*, para operar com múltiplas instâncias simultâneas. O modelo proposto será detalhado na Seção 4.2.
2. **Processamento com uma instância de múltiplas *threads* com divisão de arquivo de transações:** Baseando-se na solução de múltiplas instâncias, foi projetada e implementada uma solução multi-tread, descrito na Seção 4.3, utilizando o framework Spring. Essa abordagem proporcionou maior abstração, modularidade e escalabilidade, além de integrar serviços gerenciados pelo Spring para orquestrar o processamento e assegurar maior controle na execução das tarefas.

Após o desenvolvimento das soluções, foram realizados testes para avaliar o funcionamento dos modelos propostos, conforme mostrado na Subseção 4.4.2.

3.2.4 Preparação e Organização dos Dados

Os dados foram organizados em um arquivo de transações simuladas, representando o volume diário do quarto trimestre de 2024, conforme apresentado na Tabela 3. A criação desse arquivo envolveu diversas etapas do processamento de dados bancários para garantir sua representatividade. Primeiramente, foram identificados e selecionados os dados como números de autorização, informações comerciais, detalhes dos cartões e outros atributos críticos. A seleção cuidadosa desses dados foi essencial para criar um cenário de teste realista que refletisse as condições encontradas no ambiente de produção. Com isso, os dados foram formatados e organizados em um arquivo de teste estruturado. Esse arquivo foi projetado para ser facilmente interpretado pelo sistema, com a correta disposição dos campos e registros para simular um fluxo típico de transações financeiras. A representatividade foi garantida pela inclusão de uma

variedade de transações, como compras, estornos, e outras operações financeiras comuns, abrangendo diferentes valores, datas e condições.

A criação desse arquivo viabilizou o teste das três soluções desenvolvidas: a sequencial, a de múltiplas instâncias e a de múltiplas *threads*. No processamento sequencial, foi possível estimar o tempo de execução para os próximos anos. Já nas abordagens com múltiplas instâncias e multithreading, a análise possibilitou a identificação da configuração mais eficiente. Finalmente, a comparação entre as três soluções evidenciou as diferenças de desempenho e escalabilidade de cada modelo.

3.2.5 Execução de Experimentos e Avaliação de Resultados

Por fim, foram realizados experimentos controlados em um ambiente de teste padronizado sob as mesmas configurações de hardware com o objetivo de avaliar e comparar o desempenho entre os modelos de processamento sequencial e processamento paralelo. Para garantir a confiabilidade das medições, cada modelo foi testado em três execuções consecutivas, com os tempos registrados em cada execução. Para cada teste, foi preparado um arquivo contendo dados que simulam o cenário real e utilizado como entrada padrão para todos os experimentos.

A execução dos experimentos, seguiu o seguinte fluxo: 1) a solução sequencial foi executada, registrando-se o tempo necessário para processar o arquivo por completo; 2) Testou-se a solução baseada em múltiplas instâncias sequenciais com diversas configurações de divisão de arquivos, variando o número de instâncias de 1 a 11; e 3) Avaliou-se a solução de processamento paralelo utilizando *threads*, variando o número de *threads* de 1 a 11. Todas as configurações de experimentos foram executadas 3 vezes.

Ao final dos experimentos, os resultados foram tabulados, analisados e discutidos. Os tempos de execução foram compilados e transformados em gráficos, proporcionando uma visualização clara e objetiva. Esses gráficos permitiram identificar tendências, avaliar o impacto do aumento do número de instâncias e *threads* no tempo de processamento, e evidenciar os ganhos de desempenho da arquitetura proposta em relação à solução de processamento sequencial. A Seção 5.4 apresenta a comparação dos resultados dos experimentos.

4 DESENVOLVIMENTO

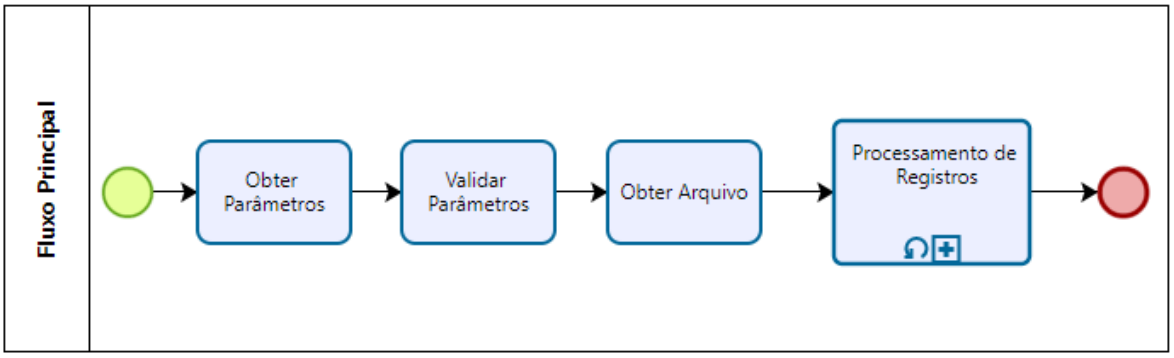
Este capítulo apresenta as soluções desenvolvidas para otimizar o processamento de transações financeiras.

4.1 Desenvolvimento de Processamento Sequencial

O modelo de processamento sequencial foi a primeira abordagem implementada para a importação e tratamento de dados bancários. Nesse modelo, as transações são processadas de forma linear, uma após a outra, sem distribuição da carga entre múltiplos núcleos de processamento. Essa abordagem é ilustrada na Figura 6.

A Figura 4 apresenta o fluxo principal do sistema de importação de dados bancários utilizando o modelo sequencial. O processo inicia-se com a obtenção dos parâmetros necessários para a execução, como diretórios de entrada e saída para o processamento dos arquivos. Em seguida, ocorre a validação desses parâmetros para garantir que as configurações estejam corretas. Após essa validação, o sistema obtém o arquivo contendo os registros a serem processados.

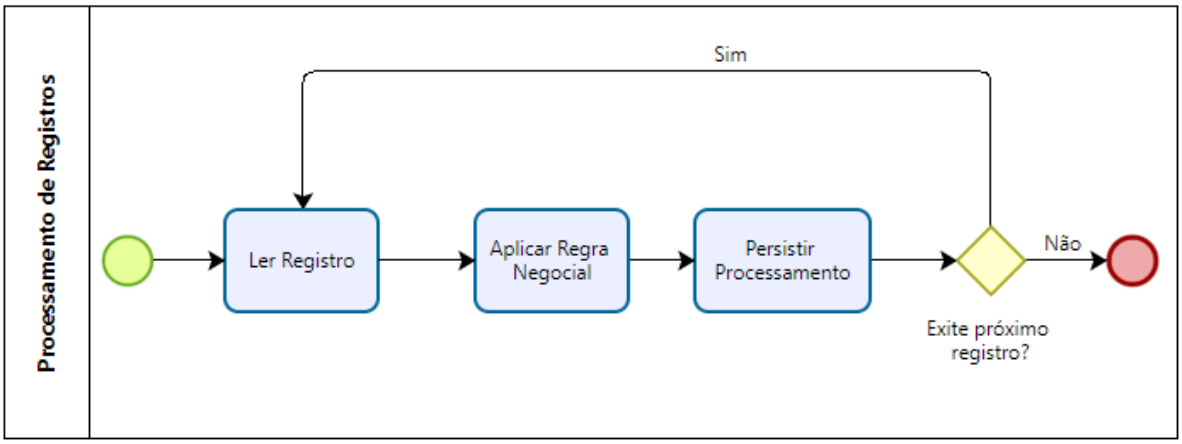
Figura 4 – Fluxo de Processamento Sequencial



Fonte: Autor

A Figura 5 detalha o fluxo de processamento de registros dentro do modelo sequencial. O processo começa com a leitura de um registro do arquivo de entrada. Em seguida, aplica-se uma regra de negócio específica, e os dados são armazenados no banco de dados. Após essa etapa, verifica-se se ainda existem registros pendentes. Caso afirmativo, o fluxo retorna ao início para continuar o processamento. Se não houver mais registros, o processo é finalizado.

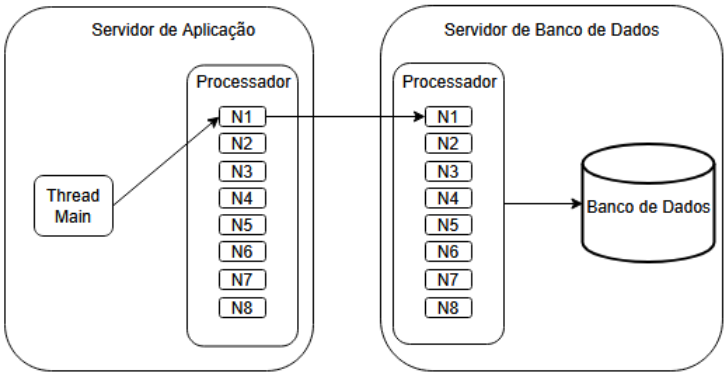
Figura 5 – Subfluxo de Processamento Sequencial



Fonte: Autor

A Figura 6 representa o fluxo sequencial de processamento entre um servidor de aplicação e um servidor de banco de dados. O processo inicia-se no servidor de aplicação, onde uma *thread* principal organiza e envia as tarefas para um dos núcleos disponíveis, representados por N1, N2, N3, N4, N5, N6, N7 e N8. Após o término do processamento inicial, as tarefas são encaminhadas para o servidor de banco de dados, onde um núcleo realiza operações como armazenamento, consulta ou manipulação de dados.

Figura 6 – Arquitetura do Modelo Sequencial de Importação de Dados

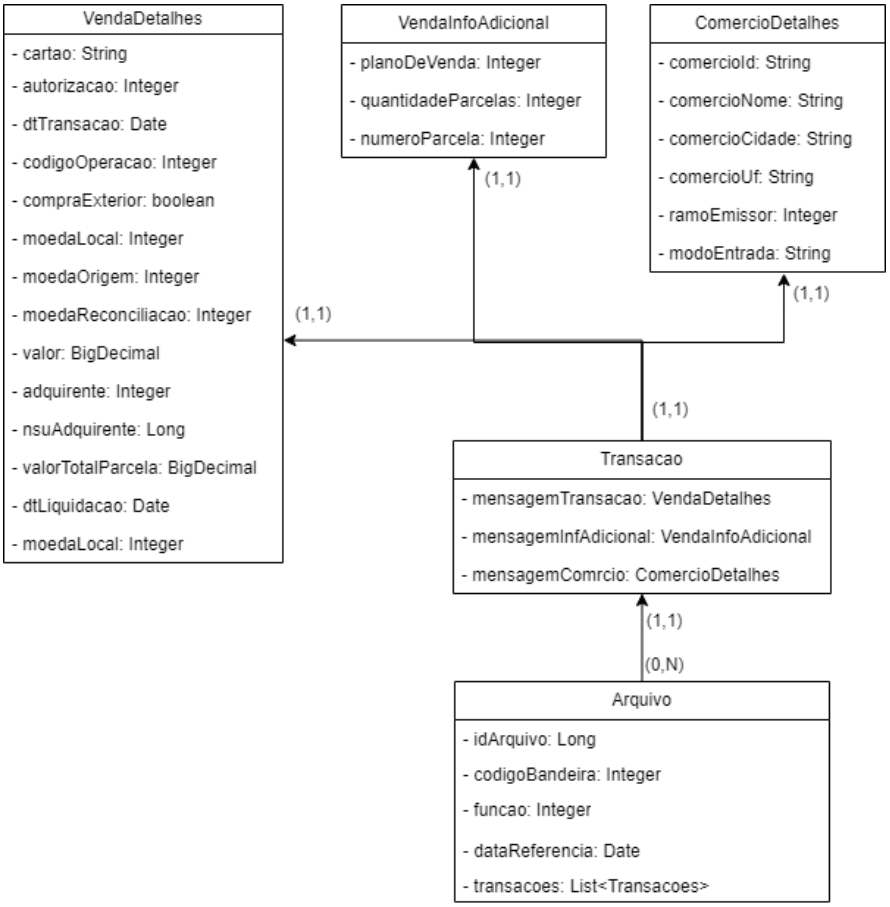


Fonte: Autor

4.1.1 Implementação

O código-fonte do sistema sequencial inicia sua execução lendo um arquivo de entrada, cujo modelo de dados está ilustrado na Figura 7. Esse arquivo de transações bancárias contém os registros financeiros a serem processados. Para a leitura, utiliza-se a classe nativa do Java `BufferedReader`, que permite a leitura eficiente dos dados linha por linha. Cada linha corresponde a um registro bancário, que deve ser processado individualmente.

Figura 7 – Diagrama de Classes do Modelo dos Registros do Arquivo



Fonte: Autor

O modelo apresentado na Figura 7 é composto por cinco classes principais: `VendaDetalhes`, `VendaInfoAdicional`, `ComercioDetalhes`, `Transacao` e `Arquivo`. A classe `VendaDetalhes` captura informações essenciais da venda, como dados do cartão e detalhes da transação. A classe `VendaInfoAdicional` complementa esses dados, detalhando o plano de pagamento e parcelamento. `ComercioDetalhes` registra informações sobre o estabelecimento onde a transação ocorreu. Essas classes estão associadas à classe `Transacao`, que consolida todas as informações de uma transação específica. Por fim, a classe `Arquivo` gerencia os arquivos que armazenam essas transações, permitindo que um único arquivo contenha múltiplas operações. Para a leitura desses registros, foi utilizado a classe `BufferedReader`, que percorre o conteúdo linha por linha, realizando o mapeamento de cada linha para um objeto `Transacao`.

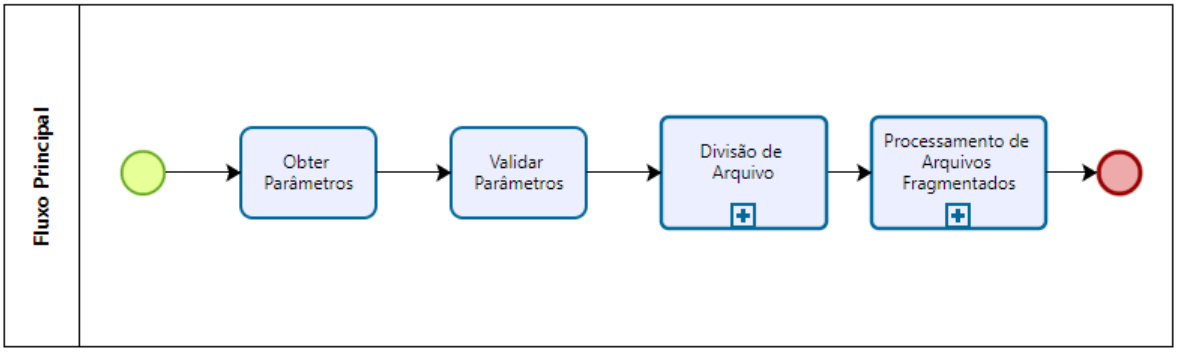
Após a leitura, cada transação passa por validações para garantir a integridade das informações. Inicialmente, é verificada a estrutura dos dados, conferindo se o número de campos está correto e se os valores seguem o formato esperado. Em seguida, os dados são transformados conforme necessário, incluindo a conversão de datas e a normalização de códigos de moeda. Também são aplicadas regras de negócio para determinar se a transação pode ser processada, verificando possíveis inconsistências e a existência de operações duplicadas. Com os dados validados, a etapa seguinte consiste na persistência das transações no banco de dados. Após essa etapa de processamento de regras, a transação é persistida no banco de dados.

4.2 Desenvolvimento da Solução com Múltiplas Instâncias

Após o desenvolvimento da solução com um modelo de processamento sequencial, a importação de arquivos foi redesenhada para explorar o processamento paralelo, com o objetivo de melhorar o desempenho é reduzir o tempo necessário para processar as transações financeiras. No modelo de múltiplas instâncias, os arquivos de entrada são divididos em partes menores, permitindo que essas partes sejam processadas ao mesmo tempo por várias instâncias.

A Figura 8 ilustra o fluxo principal da solução que utiliza múltiplas instâncias de processamento. O processo tem início com a obtenção dos parâmetros essenciais para a execução, que são então validados para garantir que atendem aos requisitos necessários para o correto funcionamento do sistema. Após a validação, o arquivo original é fragmentado em partes menores, o que permite que o processamento subsequente seja realizado de forma paralela e otimizada. A divisão do arquivo é uma etapa crucial, pois prepara os dados para serem distribuídos entre as instâncias de processamento. Finalmente, as porções fracionadas do arquivo são submetidas ao processamento, completando assim o ciclo do fluxo principal da solução.

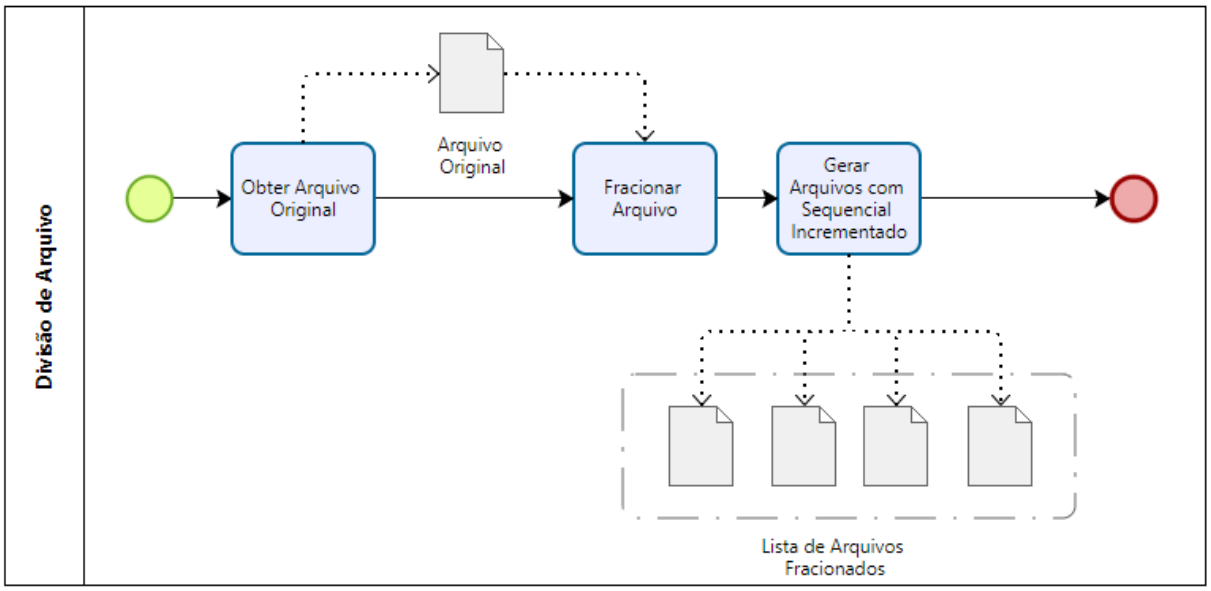
Figura 8 – Fluxo Principal do Modelo de Processamento com Múltiplas Instâncias



Fonte: Autor

Na Figura 9, é apresentado o processo de fracionamento do arquivo original, que representa um subfluxo do modelo principal descrito na Figura 8. O processo tem início com a obtenção do arquivo original, que é preparado para a etapa de fracionamento. Durante essa fase, o arquivo é dividido em partes menores de acordo com parâmetros pré-estabelecidos, como tamanho ou número de registros, garantindo que cada fração esteja otimizada para o processamento paralelo subsequente. Após o fracionamento, cada um dos arquivos resultantes é gerado e recebe um identificador sequencial único, o que facilita seu rastreamento e processamento em etapas posteriores. Ao final, uma lista organizada de arquivos fracionados é formada, pronta para ser utilizada no processamento paralelo descrito no fluxo principal.

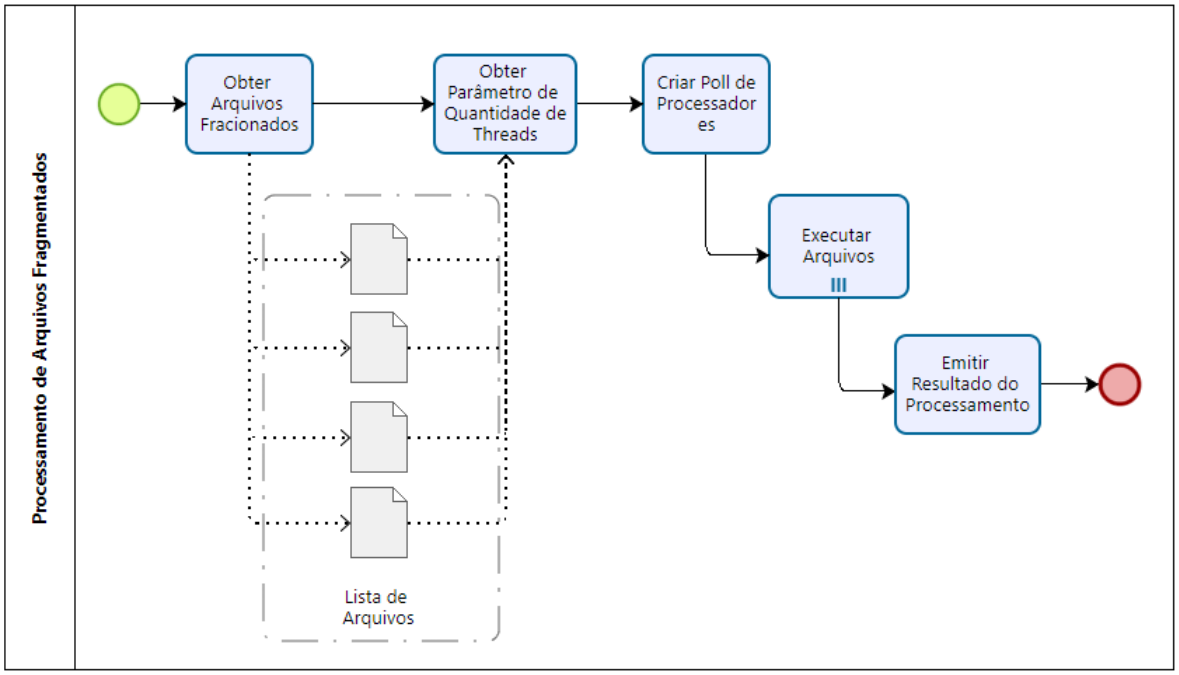
Figura 9 – Subfluxo do Modelo de Processamento com Múltiplas Instâncias - Fracionador de Arquivo



Fonte: Autor

A Figura 10 descreve o funcionamento do processador paralelo dentro do modelo de processamento paralelo, que é um subfluxo do modelo principal apresentado na Figura 8. O processo inicia com a obtenção dos arquivos fracionados, que foram previamente gerados na etapa anterior. Esses arquivos são organizados em uma lista, a qual será utilizada para distribuir as tarefas de processamento de maneira eficiente. O próximo passo envolve a obtenção do parâmetro de quantidade de *threads*, que define a quantidade de unidades de processamento a ser utilizada. Com base nesse parâmetro, um *pool* de processadores é criado, ou seja, um conjunto de *threads* independentes que serão responsáveis pela execução paralela das tarefas. Cada *thread*, ou processador, recebe uma parte dos arquivos fracionados e realiza o processamento de forma simultânea, o que contribui para uma significativa otimização do tempo total de execução. Ao término do processamento paralelo, os resultados são agregados e emitidos, concluindo a execução do subfluxo e preparando os dados para a etapa final do fluxo principal.

Figura 10 – Subfluxo do Modelo de Processamento com Múltiplas Instâncias - Processador Paralelo



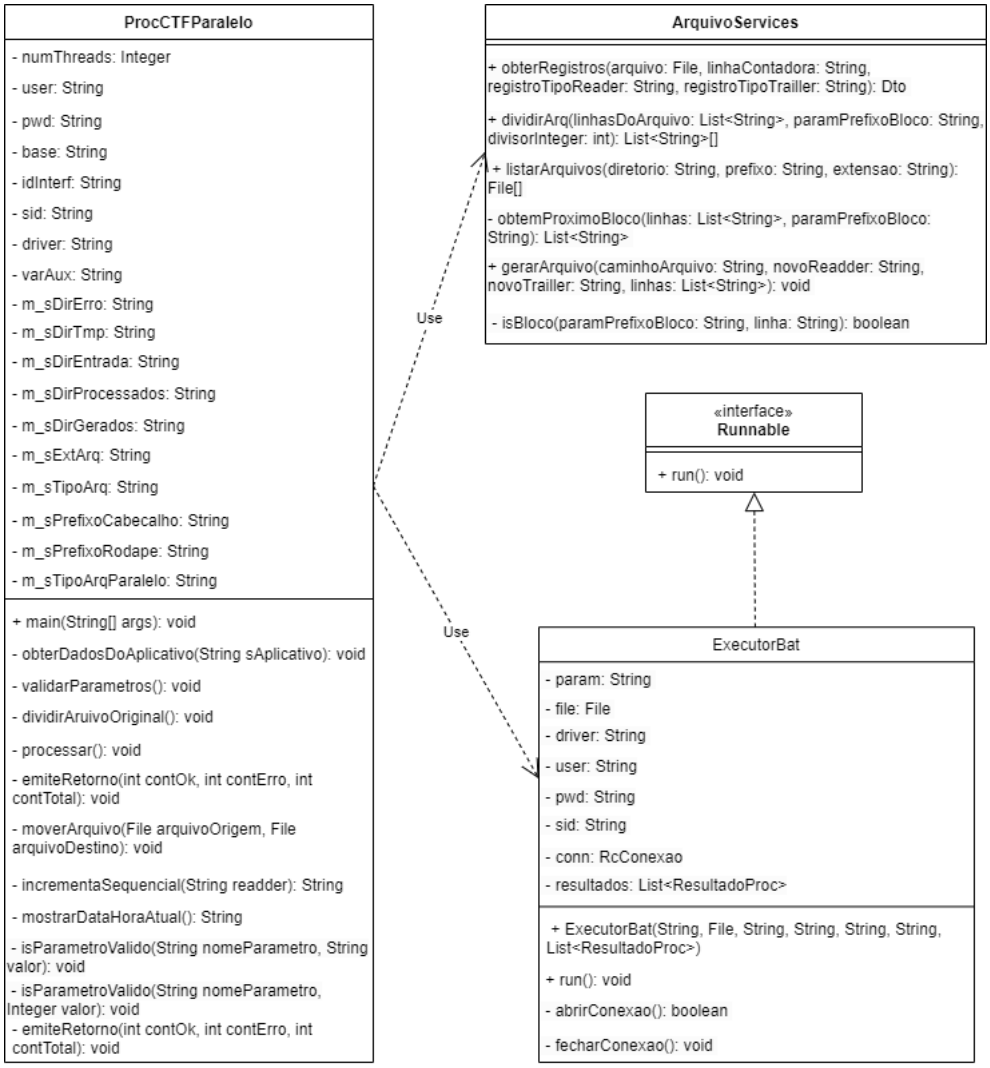
Fonte: Autor

4.2.1 Implementação

A implementação teve início com a criação da classe `ProcCTFParalelo`, que foi projetada especificamente para viabilizar o processamento paralelo de arquivos. Para isso, utilizou-se o `ExecutorService`, uma classe que facilita a criação e o gerenciamento de um *pool* de *threads*, permitindo a execução simultânea de múltiplas tarefas e otimizando o tempo de processamento.

A Figura 11 ilustra em detalhes a interação entre as classes responsáveis pela execução paralela, fornecendo uma visão clara do fluxo de execução e da coordenação entre os componentes do sistema.

Figura 11 – Diagrama de Classe para o Modelo de Processamento Paralelo



Fonte: Autor

O diagrama apresentado na Figura 11 foca em três classes: `ProcCTFParalelo`, `ArquivoServices`, e `ExecutorBat`. A classe `ProcCTFParalelo` é responsável por orquestrar o processo de execução, utilizando as configurações necessárias, como o número de instâncias e os diretórios de entrada e saída. Ela faz uso da classe `ArquivoServices` para gerenciar operações relacionadas a arquivos, como listar, dividir e obter registros de arquivos. A classe `ExecutorBat` implementa a interface `Runnable`, permitindo que cada arquivo seja processado de forma independente e simultânea. A interação entre essas classes é representada no diagrama por setas pontilhadas, indicando que a classe `ProcCTFParalelo` utiliza métodos das classes `ArquivoServices` e `ExecutorBat`, demonstrando uma relação de uso.

Antes de submeter os arquivos para processamento, a classe ArquivoServices divide o arquivo original em partes menores. Esse processo de divisão é fundamental para permitir o processamento paralelo, pois cada bloco de dados resultante é tratado como uma tarefa separada. Isso facilita a distribuição das tarefas entre múltiplas instâncias, maximizando a eficiência do sistema.

Os arquivos a serem processados são então submetidos ao ExecutorService como tarefas executáveis através do método submit. Cada tarefa é encapsulada na classe ExecutorBat, que armazena todas as informações relevantes para o processamento de um arquivo específico, incluindo variáveis auxiliares,

credenciais de acesso e detalhes do banco de dados.

A classe ExecutorBat assegura que cada arquivo seja processado de forma independente e simultânea. A distribuição do trabalho entre múltiplas instâncias permite que o sistema lide com um grande volume de dados de maneira paralela, otimizando o tempo de processamento.

A classe ProcCTFParalelo também inclui funcionalidades para a validação dos parâmetros de configuração, garantindo que todas as condições necessárias estejam devidamente configuradas antes do início das operações. Isso inclui a verificação dos diretórios de entrada, saída e temporários, além da validação de outros parâmetros críticos para o funcionamento correto do sistema.

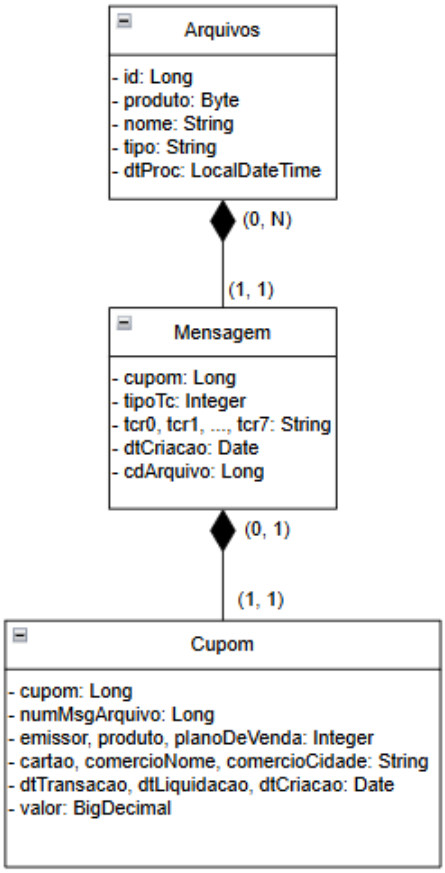
Após a submissão das tarefas ao ExecutorService, o fluxo de execução é mantido em espera até que todas as tarefas sejam concluídas. Esse controle é realizado por meio de um mecanismo que monitora o progresso das tarefas, garantindo que o processamento ocorra de forma ordenada e completa, preservando a integridade dos dados ao longo de todo o processo.

4.3 Desenvolvimento da Solução com Múltiplas *Threads*

Diante das limitações do processamento sequencial e dos desafios encontrados na abordagem baseada em múltiplas instâncias, tornou-se necessário explorar técnicas mais avançadas para otimizar o desempenho do sistema. Embora o uso de múltiplas instâncias tenha melhorado a escalabilidade, ele apresentou dificuldades, como o alto consumo de memória. Para superar essas limitações, a adoção de múltiplas *threads* dentro de uma única instância surge como uma solução mais eficiente. Essa abordagem permite que diferentes partes do arquivo de entrada sejam processadas simultaneamente dentro da mesma aplicação, reduzindo a sobrecarga de memória e processamento.

Na solução com mútiplas *threads*, os dados estão estruturados em três entidades principais: Arquivos, Mensagens e Cupons. Essas entidades representam a organização dos dados durante o fluxo de execução, sendo que os Arquivos são o ponto de entrada, as Mensagens correspondem às informações extraídas e processadas, e os Cupons refletem os resultados gerados a partir das Mensagens. O relacionamento entre essas entidades é apresentado no Figura 12, que detalha a forma como elas se conectam para garantir a integridade e a consistência do processamento.

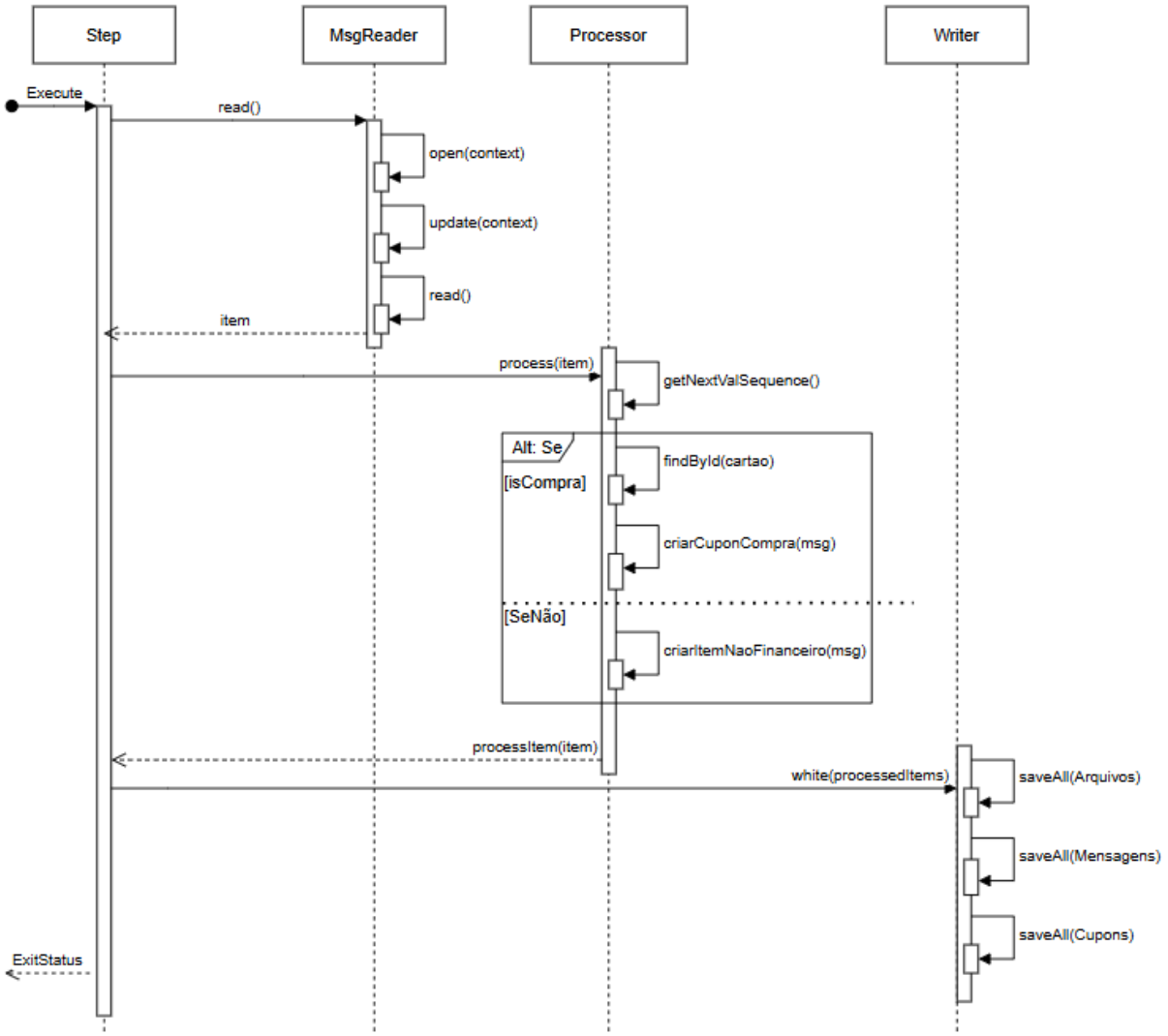
Figura 12 – Diagrama de Classes



Fonte: Autor

Com base nessa estruturação, a solução organiza o fluxo em três etapas principais: leitura, processamento e escrita, garantindo eficiência e modularidade no sistema. Cada etapa desempenha um papel específico e crucial, assegurando que os dados fluam de maneira eficiente entre os componentes. A adoção do framework *Spring* possibilitou que essas fases fossem configuradas de forma modular, facilitando o gerenciamento e a integração entre as etapas. Esse fluxo de execução e as interações entre os componentes estão representados na Figura 13, que fornece uma visão detalhada de como os dados são manipulados ao longo do processo.

Figura 13 – Diagrama de Sequência



Fonte: Autor

4.4 Descrição das Fases de Processamento

Na fase de leitura, o componente `MsgReader` é responsável por iniciar o processo de coleta dos dados necessários para o processamento subsequente. Esse processo segue algumas etapas. Primeiramente, o método `open(context)` é utilizado para inicializar o contexto de leitura. Em seguida, o método `update(context)` atualiza o estado atual do contexto, permitindo que a leitura continue sem interrupções. Finalmente, o método `read()` realiza a leitura de cada item da transação e o encaminha para a próxima fase, garantindo que os dados estejam prontos para serem processados.

Durante a fase de processamento, o `Processor` manipula cada item de forma condicional, seguindo uma sequência de etapas. Primeiramente, o método `getNextValSequence()` gera um identificador sequencial único para cada transação, garantindo a unicidade de cada operação. Em seguida, uma estrutura condicional verifica o tipo de transação usando o método `isCompra`. Dependendo do resultado dessa verificação, o sistema segue diferentes fluxos: se a transação for uma compra, o cartão associado é recuperado através do método `findById(cartao)`, e o cupom de compra é criado usando o método `criarCuponCompra(msg)`. Caso contrário, se a transação for não financeira, o sistema cria um item não financeiro com o método `criarItemNaoFinanceiro(msg)`. Essa etapa condicional permite que o sistema

diferencie entre tipos de transações e realize o tratamento adequado para cada uma. Após a definição do tipo de transação, o método `processItem(item)` finaliza o processamento do item, assegurando que ele esteja devidamente formatado e pronto para a fase de escrita.

Na fase final, o `Writer` grava os itens processados, encerrando o ciclo de processamento. O método `write(processedItems)` recebe os itens processados e os envia ao `Writer`, que grava os dados em diferentes repositórios. As operações de gravação incluem o uso de `saveAll(Arquivos)`, que salva as informações dos arquivos gerados, `saveAll(Mensagens)`, que grava as mensagens processadas no banco de dados, e `saveAll(Cupons)`, que armazena os cupons gerados a partir das transações de compra. Essas operações de gravação garantem que todos os itens processados estejam devidamente registrados, o que possibilita consultas futuras e assegura a integridade dos dados no sistema.

4.4.1 Configuração do Paralelismo

A principal configuração que garante o paralelismo na solução proposta é o uso do `TaskExecutor`, um componente do Spring Framework que facilita a execução assíncrona de tarefas. O `TaskExecutor` é responsável por gerenciar um *pool* de *threads* para a execução das tarefas, otimizando o uso dos recursos do sistema e permitindo que múltiplas operações sejam realizadas de forma paralela. O Spring oferece o `ThreadPoolTaskExecutor`, que configura e gerencia o *pool* de *threads*, fornecendo uma maneira eficiente de processar tarefas simultaneamente.

O uso do `TaskExecutor` no Spring é fundamental para o processamento paralelo eficiente. No contexto da solução desenvolvida, as operações de leitura, processamento e gravação dos itens de transação são realizadas de forma independente em múltiplos lotes, utilizando *threads* diferentes para cada lote. Isso garante que o sistema aproveite melhor os recursos do hardware, reduzindo o tempo total de execução.

A implementação do paralelismo permite que as tarefas sejam processadas de maneira assíncrona, ou seja, enquanto uma *thread* está processando um lote de transações, outras *threads* podem estar processando outros lotes simultaneamente. Esse comportamento não apenas melhora o desempenho, mas também contribui para uma maior escalabilidade da solução, permitindo que o sistema lide com volumes maiores de dados sem comprometer sua performance.

Além disso, o `TaskExecutor` é facilmente configurável, permitindo ajustar os parâmetros conforme a necessidade do sistema. Isso torna a solução flexível, adaptando-se a diferentes cenários de carga e garantindo a execução eficiente das tarefas paralelizadas. O gerenciamento do *pool* de *threads* pelo Spring também proporciona maior controle sobre a alocação de recursos e a distribuição de tarefas, contribuindo para uma execução mais eficiente e com menor risco de sobrecarga no sistema.

Todo o código-fonte das principais configurações está disponível no repositório do GitHub: <https://github.com/italofsantos/paralelismo>.

4.4.2 Validação das Implementações das Soluções Desenvolvidas

Inicialmente, foram realizados testes unitários detalhados na solução de processamento sequencial para garantir que o sistema estivesse funcionando corretamente e gerando os resultados esperados a partir dos dados de entrada fornecidos. Esses testes foram implementados utilizando o framework JUnit, amplamente reconhecido por sua eficiência na criação, execução e automação de testes para aplicações Java. O JUnit permitiu estruturar os testes de maneira organizada, com métodos específicos para cada funcionalidade avaliada e a geração de relatórios detalhados dos resultados.

Em seguida, os mesmos testes unitários foram aplicados às novas soluções implementadas, utilizando os mesmos conjuntos de dados de entrada. O objetivo dessa abordagem foi assegurar que ambas

as soluções produzissem resultados idênticos para cenários equivalentes, garantindo a consistência e a integridade do sistema para possibilitar a migração. No JUnit, foram configurados testes parametrizados para comparar diretamente os resultados das duas soluções, assegurando que as saídas fossem idênticas em diferentes condições de execução.

Além disso, o uso do JUnit facilitou a detecção de possíveis discrepâncias ao longo do desenvolvimento, permitindo o ajuste rápido de comportamentos divergentes e garantindo que as alterações realizadas na nova arquitetura não comprometessem a funcionalidade esperada. Esse processo foi essencial para validar a compatibilidade funcional entre as versões, além de identificar melhorias nas respostas geradas pela nova arquitetura.

5 RESULTADOS

Este capítulo apresenta os resultados dos modelos apresentados no Capítulo 4. Todos os experimentos foram realizados utilizando a seguinte configuração de ambiente: Sistema operacional Windows 10 Pro, processador 11th Gen Intel Core i7-1165G7 2.80GHz com 4 núcleos e 8 Processadores Lógicos, 16 GB de RAM e memória virtual total de 40,8 GB. As métricas analisadas abrangem o tempo médio de processamento de transações e a taxa de redução proporcionada pela nova arquitetura.

5.1 Resultados do Modelo de Processamento Sequencial

Com base nos dados obtidos pelo método Holt-Winters, apresentados na Tabela 3, foi gerada a Tabela 4, a partir da aplicação do modelo sequencial de processamento descrito na Seção 4.1. O tempo total necessário para a importação dos registros diários foi estimado multiplicando-se o volume médio de transações por instituição pelo tempo médio de processamento unitário de cada transação. Esse cálculo foi realizado para cada trimestre projetado, permitindo avaliar a evolução da demanda e os impactos no desempenho do sistema ao longo do tempo.

Tabela 4 – Expectativa de Tempo com Modelo Sequencial

Trimestre	**Tempo de Processamento (hh:mm)**
2029-04	390:31
2029-03	378:25
2029-02	364:57
2029-01	368:45
2028-04	356:39
2028-03	343:11
2028-02	346:59
2028-01	334:53
2027-04	321:25
2027-03	325:13
2027-02	313:07
2027-01	299:39
2026-04	303:28
2026-03	291:21
2026-02	277:53
2026-01	281:42
2025-04	269:36
2025-03	256:07
2025-02	259:56
2025-01	247:50
2024-04	234:21

Fonte: Autor

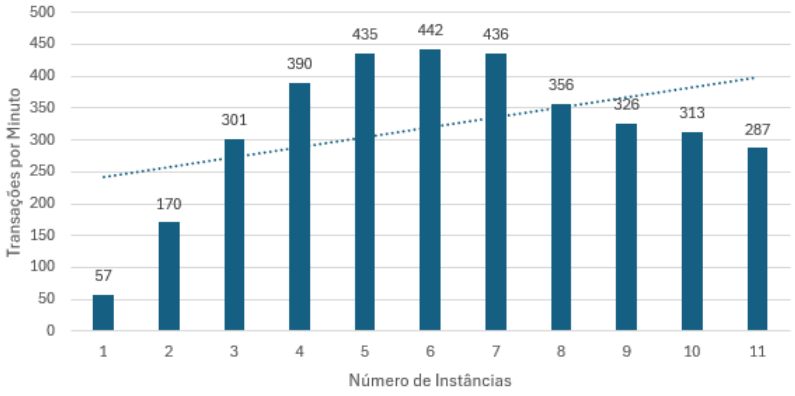
A análise dos dados apresentados na Tabela 4 revela que há uma relação proporcional entre o aumento da quantidade de transações e o tempo necessário para o processamento. Conforme o número de transações cresce, o tempo total de processamento também aumenta, refletindo diretamente as limitações de um modelo de processamento sequencial. Essa relação pode ser observada, por exemplo, no quarto trimestre de 2029, que registrou o maior volume de transações e apresentou o maior tempo de

processamento. Em contrapartida, no terceiro trimestre de 2024, com o menor número de transações, o tempo foi significativamente menor.

5.2 Análise de Desempenho do Sistema com Múltiplas Instâncias

A Figura 14 apresenta uma análise detalhada do desempenho do sistema ao utilizar múltiplas instâncias, descrito na Seção 4.2. O desempenho é medido em termos de transações processadas por minuto, considerando diferentes quantidades de instâncias durante o processamento. Este gráfico é fundamental para visualizar como o paralelismo impacta diretamente na eficiência do sistema, revelando o ponto em que o aumento do número de instâncias proporciona ganhos significativos de desempenho, bem como o ponto de saturação onde esses ganhos começam a se estabilizar ou até mesmo a declinar. Essa representação gráfica permite uma compreensão clara dos limites e das potencialidades do uso do paralelismo nas operações de escrita e leitura, sendo essencial para a determinação da configuração ótima de instâncias para maximizar o desempenho sem comprometer a estabilidade do sistema.

Figura 14 – Desempenho de Transações por Minuto em Função do Número de Instâncias



Fonte: Autor

Os resultados apresentados na Figura 14 validaram a eficácia da abordagem de processamento paralelo, evidenciando um aumento significativo no número de transações processadas ao utilizar múltiplas instâncias em comparação com uma única instância. Esse desempenho superior comprova a viabilidade da estratégia adotada, estabelecendo uma base sólida para sua consolidação como uma solução definitiva, destacada por sua maior robustez e escalabilidade.

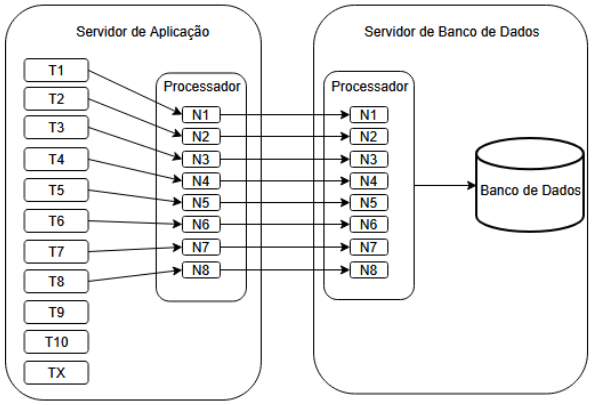
5.3 Avaliação de Desempenho da Solução Definitiva com Múltiplas *Threads*

Esta seção apresenta os resultados da implementação do modelo descrito na Seção 4.3. A utilização do framework Spring permitiu o desenvolvimento de uma solução com maior modularidade e controle. Essa etapa incorporou serviços gerenciados pelo Spring para orquestrar tarefas paralelas, otimizando o uso de recursos.

A Figura 15 ilustra como a solução implementada aproveitou melhor os recursos da máquina ao utilizar o processamento paralelo. Cada *thread*, representada como T no servidor de aplicação, é mapeada para um núcleo, identificado como N no servidor, garantindo o uso eficiente do hardware disponível e acelerando o processamento das transações. A figura também demonstra que, quando todos os núcleos estão ocupados, as *threads* excedentes aguardam na fila até que um núcleo fique disponível para pro-

cessamento. Essa arquitetura permitiu um escalonamento dinâmico das tarefas, otimizando o tempo de resposta e aproveitando ao máximo os recursos computacionais disponíveis.

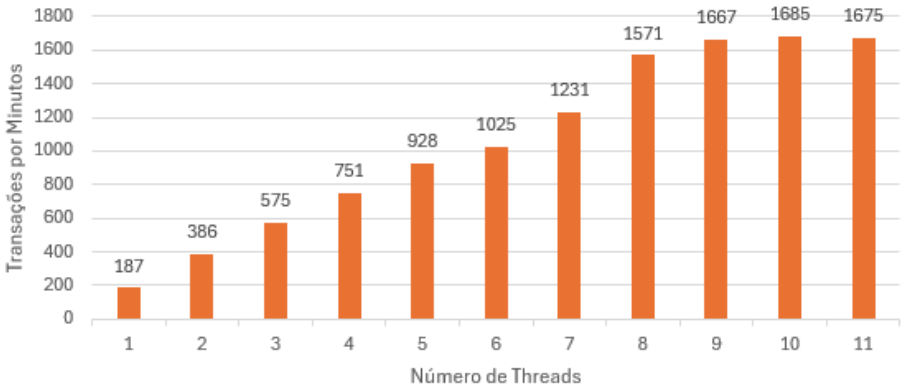
Figura 15 – Utilização de Processamento Paralelo com Melhor Aproveitamento de Recursos



Fonte: Autor

A Figura 16 apresenta os resultados obtidos com a solução utilizando múltiplas *threads*. A integração do Spring possibilitou a utilização eficiente de 10 *threads*, identificadas como a configuração ideal no cenário testado. O gráfico destaca o impacto do aumento no número de *threads* sobre a capacidade de processamento, evidenciando a eficiência e a escalabilidade da nova arquitetura.

Figura 16 – Desempenho do Sistema em Função do Número de *Threads*



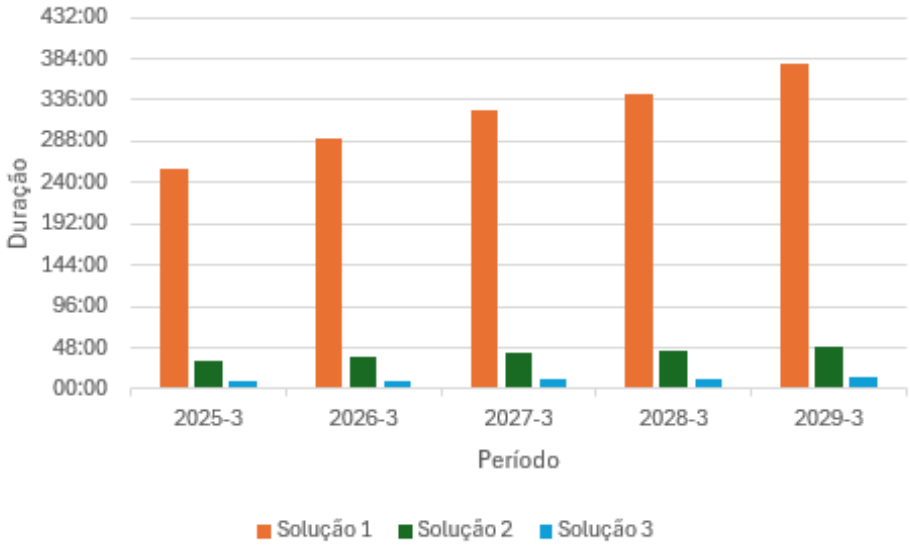
Fonte: Autor

5.4 Comparação de Desempenho

A Figura 17 apresenta a comparação do tempo de processamento diário de transações para três abordagens distintas: solução sequencial, solução multi-instância e solução com múltiplas *threads*, representadas respectivamente por, Solução 1, Solução 2 e Solução 3. A quantidade de transações foi estimada com base nos dados apresentados na Tabela 3. O eixo horizontal representa os períodos projetados, enquanto o eixo vertical indica o tempo total necessário para o processamento. Nota-se que a solução sequencial apresenta os tempos mais elevados, com um crescimento acentuado à medida que o volume de transações aumenta. A solução multi-instância reduz significativamente o tempo de processamento, mas ainda apresenta uma tendência de crescimento ao longo dos anos. Já a solução com múltiplas *thre-*

ads demonstra o melhor desempenho, mantendo tempos reduzidos mesmo com o aumento da carga de trabalho, evidenciando sua eficiência e escalabilidade.

Figura 17 – Comparação de Desempenho



Fonte: Autor

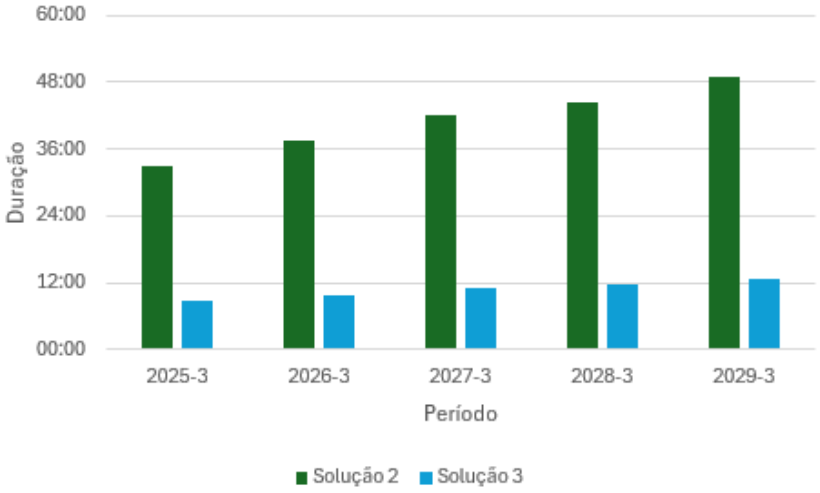
A solução sequencial de importação de dados bancários apresenta limitações expressivas no tempo de processamento, especialmente quando comparada às abordagens que exploram paralelismo. No terceiro trimestre de 2025, o tempo médio necessário para processar 875.956 transações diárias no ambiente de teste seria de 256 horas e 7 minutos. Esse tempo aumenta consideravelmente ao longo dos anos, atingindo 378 horas e 25 minutos no terceiro trimestre de 2029.

Já a solução multi-instância, que executa múltiplas instâncias do processamento em paralelo, apresenta uma melhora significativa. No terceiro trimestre de 2025, o tempo médio de processamento diário é reduzido para 33 horas e 1 minuto, enquanto no terceiro trimestre de 2029, mesmo com o aumento no volume de transações, o tempo necessário é 48 horas e 48 minutos. Embora essa abordagem traga ganhos relevantes, o tempo de processamento ainda pode se tornar um gargalo operacional conforme a demanda continua crescendo.

Por outro lado, a solução com múltiplas *threads* demonstra o melhor desempenho entre as três abordagens. No terceiro trimestre de 2025, o tempo necessário para processar o mesmo volume de transações diárias é reduzido para 8 horas e 39 minutos. Mesmo no terceiro trimestre de 2029, com um volume diário projetado de 1.294.213 transações, o processamento é concluído em 12 horas e 48 minutos, garantindo um tempo de resposta dentro de um intervalo operacional eficiente. Essa melhoria representa uma redução de 95,7% no tempo de processamento em comparação com a solução sequencial e 73,7% em relação à solução multi-instância, demonstrando a eficácia e a escalabilidade do modelo implementado para atender às demandas futuras de forma confiável.

Para uma análise mais detalhada da eficiência das abordagens paralelas, a Figura 18 compara diretamente o desempenho da solução multi-instância com a solução com múltiplas *threads*, representadas respectivamente por, Solução 2 e Solução 3

Figura 18 – Comparação entre Solução Multi-instância e Solução com Múltiplas *Threads*



Fonte: Autor

A Figura 18 evidencia que, ao longo dos anos, a Solução 3 mantém tempos de processamento significativamente menores do que a Solução 2, demonstrando maior escalabilidade. Enquanto a Solução 2 apresenta um crescimento contínuo no tempo necessário para processar as transações diárias, a Solução 3 se mantém mais estável, suportando um volume crescente de transações sem um aumento proporcional no tempo de processamento.

6 CONCLUSÃO

Durante a execução do presente trabalho, foram analisados três modelos distintos de processamento para a importação de dados bancários: sequencial, multi-instância e multi-threading. Cada um desses modelos apresenta características específicas que impactam diretamente a eficiência e escalabilidade do sistema.

O modelo sequencial foi a abordagem inicial adotada, no qual as transações são processadas de forma linear. Essa abordagem se destaca pela simplicidade de implementação e manutenção, além de proporcionar maior previsibilidade no fluxo de execução. No entanto, sua principal limitação é o tempo elevado de processamento, que se torna um gargalo à medida que o volume de dados cresce. Além disso, esse modelo não aproveita os recursos de hardware modernos, como múltiplos núcleos de processamento, o que resulta em um desempenho inferior e limita sua escalabilidade.

Para mitigar essas limitações, foi implementado o modelo multi-instância, no qual múltiplas instâncias do processo são executadas simultaneamente, cada uma processando uma parte do arquivo de entrada. Esse modelo apresentou uma redução do tempo de execução em relação ao modelo sequencial, uma vez que diferentes instâncias trabalham em paralelo, distribuindo a carga de processamento. No entanto, a execução simultânea de várias instâncias também trouxe desafios, como o maior consumo de recursos computacionais, já que cada instância requer memória e poder de processamento individuais. Além disso, foi necessária a implementação de mecanismos eficientes para gerenciar a comunicação e sincronização entre as instâncias, garantindo a integridade dos dados processados.

Buscando otimizar ainda mais o desempenho, foi implementado o modelo multi-threading, que substitui a execução de múltiplas instâncias por um único processo que gerencia diversas *threads*. Nesse modelo, a carga de trabalho é distribuída dinamicamente entre as *threads*, aproveitando melhor os núcleos de processamento disponíveis e reduzindo a sobrecarga de memória em comparação ao modelo multi-instância. O uso de *threads* permitiu uma redução no tempo de processamento, tornando o sistema mais eficiente e responsivo. Entretanto, essa abordagem também introduziu desafios, como a necessidade de gerenciar concorrência entre as *threads* e a sincronização de acessos simultâneos aos recursos compartilhados.

Com base nos resultados obtidos, foi possível observar que o modelo multi-threading apresentou a melhor eficiência, reduzindo o tempo de processamento em comparação ao modelo sequencial e ao multi-instância. No entanto, a escolha do modelo mais adequado depende do contexto e das limitações de infraestrutura. Enquanto o modelo sequencial pode ser viável para pequenos volumes de dados e sistemas legados, o modelo multi-instância demonstrou ser uma alternativa intermediária, permitindo maior paralelismo ao custo de um consumo maior de recursos. Já o modelo multi-threading combinou eficiência e escalabilidade, tornando-se a solução mais adequada para lidar com a crescente demanda de processamento de dados bancários.

Entre as contribuições do trabalho, destacam-se a validação do uso de técnicas de paralelismo para transações financeiras e a criação de uma metodologia replicável em outras áreas que enfrentam desafios similares. Embora a nova solução tenha alcançado resultados positivos, há espaço para melhorias, como a utilização de computação distribuída.

Os objetivos estabelecidos no início deste trabalho foram atendidos. A revisão bibliográfica forneceu uma base sólida para compreender as soluções e propor inovações. A análise da solução utilizando

um modelo sequencial permitiu identificar os gargalos e as oportunidades de melhoria. As projeções estatísticas destacaram os desafios futuros e reforçaram a necessidade de uma solução mais robusta. A implementação do sistema com múltiplas instâncias confirmou a viabilidade da solução proposta, evidenciando ganhos de desempenho.

Por fim, este trabalho contribui para o avanço do conhecimento sobre processamento paralelo aplicado a sistemas financeiros e abre caminho para estudos futuros que possam expandir e melhorar as soluções apresentadas, atendendo às crescentes demandas do setor bancário e de outras áreas intensivas em dados.

REFERÊNCIAS

BLOCH, J. *Effective Java*. 3. ed. [S.l.]: Addison-Wesley, 2017.

BRASIL, B. C. do. *Evolução de meios digitais para realização de transações de pagamento no Brasil*. 2022. <https://www.bcb.gov.br/content/publicacoes/boxe_relatorio_de_economia_bancaria/reb2022b7p.pdf>. Acessado em: 04-01-2025.

_____. *Estatísticas de Meios de Pagamentos*. 2024. <https://www.bcb.gov.br/estatisticas/spbadendos>. Acessado em: 04-01-2025.

_____. *Estatísticas de Meios de Pagamento - Dados Abertos*. 2025. Acesso em: 18 jan. 2025. Disponível em: <https://dadosabertos.bcb.gov.br/dataset/estatisticas-meios-pagamentos>.

_____. *Relação de Instituições em Funcionamento no País*. 2025. Acesso em: 18 jan. 2025. Disponível em: <https://www.bcb.gov.br/estabilidadefinanceira/relacao_instituicoes_funcionamento>.

BRESHEARS, C. *The Art of Concurrency: A Thread Monkey's Guide to Writing Parallel Applications*. Sebastopol, CA: O'Reilly, 2009.

CHATFIELD, C. The holt-winters forecasting procedure. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, Wiley Online Library, v. 27, n. 3, p. 264–279, 1978.

CORPORATION, O. *Documentação Oficial do Java*. 2024. <https://docs.oracle.com/javase/>.

EL-REWINI, H.; ABD-EL-BARR, M. *Advanced Computer Architecture and Parallel Processing*. New Jersey: John Wiley Sons, 2005.

EMBRACON. *Cartão de crédito ou cartão de débito: suas diferenças e qual usar*. 2023. <https://www.embracon.com.br/blog/cartao-de-credito-ou-cartao-de-debito-suas-diferencas-e-qual-usar>. Acessado em: 29-07-2024.

FLYNN, M. J. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, p. 948–960, September 1972.

FOSTER, I. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. [S.l.]: Addison-Wesley, 1995.

FRAMEWORK, S. *Spring Documentation*. 2024. [Acesso em: 22 dez. 2024]. Disponível em: <https://spring.io/projects/spring-framework>.

GARIBALDI, F. *Sistema de Pagamentos Brasileiro*. [S.l.]: Lumen Juris, 2023.

GOETZ, B. et al. *Java Concurrency in Practice*. [S.l.]: Addison-Wesley, 2006.

GOSLING BILL JOY, G. S. J. *The Java Language Specification*. [S.l.]: Addison-Wesley, 1995.

GRAMA, A. et al. *Introduction to Parallel Computing*. 2. ed. [S.l.]: Pearson, 2003.

HENNESSY, J. L.; PATTERSON, D. A. *Computer Architecture: A Quantitative Approach*. 5. ed. [S.l.]: Morgan Kaufmann, 2011.

JOHNSON, R. *Spring Framework*. [S.l.]: SpringSource, 2003. Originalmente criado em 2003, o Spring Framework visa simplificar o desenvolvimento de aplicações Java.

LEA, D. *Concurrent Programming in Java: Design Principles and Patterns*. [S.l.]: Addison-Wesley, 2000.

MAROWKA, A. Parallel computing on any desktop. *Communications of the ACM*, New York, NY, USA, v. 50, n. 9, p. 75–78, September 2007.

MICROSOFT. *Criar uma previsão no Excel para Windows*. 2024. [Acesso em: 28 ago. 2024]. Disponível em: <https://support.microsoft.com/pt-br/office/criar-uma-previs%C3%A3o-no-excel-para-windows-22c500da-6da7-45e5-bfdc-60a7062329fd?ns=EXCEL&version=90>.

ORACLE. *Java Platform, Standard Edition Documentation*. 2024. Acesso em: 18 jan. 2025. Disponível em: <https://docs.oracle.com>.

PACHECO, P. S. *An Introduction to Parallel Programming*. San Francisco: Elsevier, 2011.

_____. *An Introduction to Parallel Programming*. [S.l.]: Morgan Kaufmann, 2011.

SCHILDT, H. *Java: The Complete Reference*. 12. ed. [S.l.]: McGraw-Hill, 2021.

SIEGEL, H. J. *Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies*. 2. ed. [S.l.]: McGraw-Hill, 1990.

SLACK, N.; CHAMBERS, S.; JOHNSTON, R. *Administração da Produção*. 3. ed. [S.l.]: Atlas, 2009.

TANENBAUM, A. S. *STEEN, M. V. Sistemas Distribuídos: Princípios e Paradigmas. 2. ed.* São Paulo: Pearson Prentice Hall, 2007.