Sistema de Gerenciamente e Consulta de Eventos

Vinícius Moreira Faria - 2023060715
Departamento de Ciência da Computação – UFMG
Belo Horizonte – MG – Brasil
viniciusmfufmg@gmail.com

5/06/2025

1 Introdução

Essa documentação lida com o problema da busca de eventos em uma lista. O objetivo é realizar consultas ao sistema de maneira eficiente. Para isso, foi usada uma abordagem baseada em Arvores binárias de busca. O sistema possui 3 indices para acesso rápido à lista:

- **Indice de Eventos**, que aponta para os eventos na lista do sistema e possui uma chave composta por ID e Tempo, permitindo o agrupamento de eventos de um mesmo pacote no indice;
- **Indice de Pacote**, que armazena referências ao primeiro e ultimo evento de um determinado pacote.
- **Indice de Clientes**, que representa os clientes relacionados aos pacotes no sistema. Cada cliente possui uma lista com os IDs de cada pacote associado.

A abordagem utiliza os 3 indices de maneira composta para realizar dois tipos de consultas ao sistema:

- **Cliente**: Consulta à um remetende ou destinatário. Para cada pacote associado ao cliente, deve retornar o registro do pacote e sua ultima atualização no sistema.
- **Pacote**: Consulta à um Pacote. Retorna todo o histórico de eventos do pacote até o momento atual.

A documentação está organizada da seguinte forma:

- Seção 1: Introdução;
- Seção 2: Descrição da implementação;
- Seção 3: Instruções de compilação e execução;
- Seção 4: Análise de complexidade;
- Seção 5: Estratégias de Robustez;

- Seção 6: Análise Experimental;
- Seção 7: Conclusão;
- Seção 8: Bibliografia;

2 Método

2.1 Organização do Código

O código está estruturado de maneira modular, com classes para cada objeto do sistema. A arquiterura buscou separar leitura de dados, processamento de buscas e geração de respostas.

2.2 Estruturas de Dados Utilizadas

O sistema possui uma Lista responsável por armazenar cada Evento lido da entrada, essa lista foi implementada como uma lista simplesmente encadeda com o objetivo de gerenciar o tempo de vida dos eventos no sistema. Os eventos não são armazenados em nenhuma outra estrutura.

Para realizar buscas, a estrutura usada neste projeto foi a Àrvore de busca binária AVL, que possui métodos para garantir o balanceamento da arvore, manténdo uma estrutura que permite a busca logaritmica independente da ordem de inserção dos elementos, o que é essencial para a performace do sistema. Cada índice usado implementa uma árvore distinda com campos que permitem acessar e manipular referências à eventos na lista de Eventos.

2.3 Principais Métodos

- **ListaEventos::Processar**: Esse método é responsável por ler e inicializar os eventos que serão inseridos na lista, o método chama **inserir()**, que insere o evento na lista e retorna um ponteiro para seu endereço, que é usado para inserir as referências nos indices de pesquisa que serão utilizados nas consultas. O método utiliza lógicas distintas dependendo do tipo de evento do pacote e para eventos de consulta, apenas lê a consulta e chama o gerador de respostas.

- **ListaEvetos::gerarRespostasCliente**: A resposta à consulta de um cliente é construída usando um índice de eventos auxiliar criado localmente. Primeiro, é feito a busca do cliente no Indice de Clientes para obter sua lista de pacotes associados. Para cada pacote do cliente, é inserido seu primeiro e ultimo evento no indice auxiliar. Ao final da busca é feito um caminhamento **InOrder** no indice, retornando as atualizações de cada pacote do cliente ordenadas por tempo.

- **ListaEventos::gerarRespostasPacote**: Para a consulta de um pacote, é feito a busca utilizando o ID no Indice de Pacotes, onde obtem-se uma referência ao evento de registro do pacote no Indice de Eventos. Com isso, é feito um caminhamento no Indice de Eventos à partir do evento de registro até que o ID do pacote relacionado ao evento seja diferente do ID consultado. Isso retorna todo o histórico de eventos de um pacote ordenados pelo tempo.

Os demais métodos do sistema implementam a lógica de criação, inicialização e destruição das estruturas de dados utilizadas e seus métodos, como os métodos de balanciamento e caminhamentos nos indices, liberação de memórias e destruição de ponteiros.

2.4 Cargas de Trabalho

Para avaliar o desempenho do sistema de consulta de eventos de pacotes, foram definidas três classes distintas de cargas de trabalho, com foco na variação de elementos críticos ao funcionamento da aplicação:

- Variação no número de clientes: avalia o impacto do aumento de usuários únicos no sistema, mantendo fixo o número total de pacotes e eventos por pacote. Essa carga simula um ambiente mais distribuído, com muitos usuários controlando poucos pacotes cada.
- Variação no número de pacotes: analisa o impacto de um volume crescente de pacotes trafegando no sistema, mantendo fixo o número de clientes. Essa carga de trabalho reflete cenários com maior densidade de entrega por usuário.
- Variação no número de eventos por pacote: estuda a escalabilidade do sistema frente ao crescimento da quantidade de eventos associados a cada pacote (transições, registros, atualizações, etc.), simulando redes logísticas com mais complexidade de transporte e rastreamento.

Cada carga foi projetada para isolar uma dimensão do sistema, permitindo analisar separadamente os impactos de escalabilidade em termos de tempo de execução e uso de memória.

2.5 Configuração de Testes

- **Sistema Operacional**: MacOs 15.7;
- **Linguagem**: C++;
- **Compilador**: g++ (Apple clang) 14.0.3
- **Hardware**: M1, 8GB RAM.

3 Instruções de Compilação e Execução

Para compilar e executar o programa, siga os passos:

1. Acesse o diretório do projeto: **cd TP3/**
2. Compile o programa: **make all**
3. Isso criará o executável **‘tp3.out‘** na pasta **‘bin/‘**. Para executar utilize: **bin/tp3.out input.txt**
4. O programa retornará o resultado de cada consulta realizada no terminal.

4 Análise de Complexidade

Essa seção contempla a nálise dos métodos mais relevantes em termos de complexidade de tempo e espaço.

ListaEventos::processarEventos()

- **Tempo** $O(N \cdot (\log C + \log P + \log E))$:
  - N: Linhas processadas
  - C: Clientes no sistema
  - P: Pacotes no sistema
  - E: Eventos no sistema
- **Espaço** $O(C + P + E)$:
  - Armazenamento proporcional às estruturas principais
  - Alocações transitórias durante processamento

ListaEventos::gerarRespostasCliente()

Realiza busca no índice de clientes e de pacotes e armazena um indice auxiliar.

- **Tempo** $O(P_c \cdot \log P_c)$:
  - P_c: Pacotes associados ao cliente
  - Construção de árvore auxiliar
- **Espaço** $O(P_c)$:
  - Árvore AVL temporária para ordenação

ListaEventos::gerarRespostasPacote()

Realiza busca no indice de pacotes e faz a travessia parcial na árvore de eventos, proporcional ao numero de eventos do pacote.

- **Tempo** $O(E_p)$:
- **Espaço** $O(1)$:
  - Não aloca estruturas adicionais
  - Uso de pilha de recursão $O(\log E)$

Operações AVL (Inserção/Busca)

- **Tempo** $O(\log n)$:
  - Altura balanceada da árvore
- **Espaço** $O(1)$ por operação:
  - Alocações locais constantes

ListaEventos::inserir()

Inserir no início da lista encadeada

- **Tempo** $O(1)$: inserir no início da lista encadeada.
- **espaço** $O(1)$: Alocação de um único nó sem estruturas auxiliares

Análise de Espaço Total

$$S(N) = O(C + P + E)$$

- **Estruturas Permanentes:** $O(C + P + E)$
- **Picos de Memória:**
  - Consultas CL: $O(P_c)$
  - Processamento: $O(\log N)$ (pilha de recursão)
- **Fator Dominante:** Armazenamento de eventos $O(E)$

4.1 Conclusões de Desempenho

- **Gargalo Temporal:** Processamento de eventos $O(N \log N)$
- **Gargalo Espacial:** Armazenamento de eventos $O(E)$
- **Pontos Fortes:**
  - Consultas rápidas para pacotes com poucos eventos
  - Escalabilidade logarítmica para operações básicas

5 Estratégias de robustez

Para garantir a robustez do sistema, o código implementa mecanismos de programação defensiva e tolerância a falhas, mantendo a estabilidade e a segurança da execução. Essas estratégias têm como objetivo evitar comportamentos indesejados, detectar erros em tempo de execução e minimizar os impactos de falhas durante as buscas. Esta seção descreve as principais estratégias implementados no sistema.

5.1 Implementação

- **Verificações de Entrada e Fluxo de Arquivo:** Logo na função main, são feitas verificações essenciais sobre os argumentos da linha de comando e a abertura do arquivo de entrada. Caso o usuário não forneça o caminho do arquivo ou o arquivo não possa ser aberto, mensagens de erro informativas são impressas e o programa é finalizado com códigos de retorno adequados.

- **Programação Defensiva nos Índices:** As funções dos índices (clientes, pacotes e eventos) aplicam validações para garantir o uso seguro de ponteiros. Exemplo disso é o uso de verificações explícitas para nullptr antes de acessar membros de estruturas ou ponteiros, especialmente durante caminhamentos de árvores ou listas. Essas verificações impedem falhas como segmentation faults causadas por desreferenciamento de ponteiros inválidos.

- **Garantia de Integridade dos Dados:** O sistema garante que os objetos da classe Evento sejam armazenados em uma única estrutura de dados — uma lista encadeada — e que outras estruturas (como a árvore AVL do índice de eventos) armazenem apenas ponteiros para esses eventos. Essa centralização da alocação evita inconsistências e facilita a liberação da memória posteriormente. Ao associar eventos à árvore, o sistema se certifica de que os nós da árvore contenham ponteiros válidos e atualizados após a alocação na lista:

- **Tolerância a Dados Repetidos:** Para lidar com eventos múltiplos com o mesmo tempo e mesmo ID de pacote — o que poderia causar colisões de chave na árvore AVL — o sistema adota uma estratégia de desambiguação com contador incremental. Isso garante a unicidade das chaves de eventos no índice. Dessa forma, a estrutura de busca se mantém correta e balanceada, sem perda de eventos.

- **Isolamento de Responsabilidades:** A separação clara entre o código de leitura (função processarEventos) e a geração de saídas (gerarRespostasCliente, gerarRespostasPacote) favorece a robustez ao isolar possíveis pontos de falha. Esse isolamento facilita testes individuais e depuração, reduzindo o risco de efeitos colaterais não intencionais.

- **Liberação Segura de Memória:** No destrutor da ListaEventos, a memória alocada dinamicamente para eventos e consultas é liberada cuidadosamente, evitando vazamentos de memória. Mesmo em caso de erro, como uma exceção ou finalização antecipada, os ponteiros são postos como nullptr, reduzindo o risco de "dangling pointers".

5.2 Conclusão

Essas estratégias tornam o sistema mais robusto e confiável, facilitando a depuração, análise e manutenção.

6 Análise Experimental

6.1 Objetivo

O objetivo desta análise é avaliar o desempenho computacional do sistema de gerenciamento e consulta de eventos de pacotes, considerando variações nas principais dimensões da carga de trabalho: número de clientes, número de pacotes e número de eventos por pacote.

A avaliação busca entender o impacto dessas variações sobre o tempo de processamento e o uso de memória, desconsiderando o tempo de leitura do arquivo de entrada (I/O), conforme recomendado.

6.2 Configuração do Ambiente Experimental

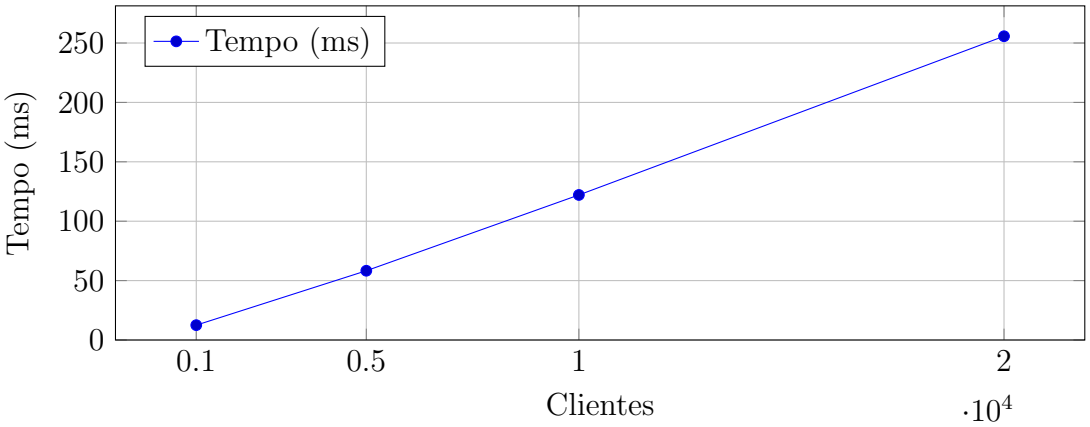
- **Hardware:** MacBook M1, 8GB RAM
- **Sistema Operacional:** macOS
- **Compilador:** `clang++` com flags de otimização `-O3`
- **Métricas Avaliadas:** Tempo de processamento (em milissegundos) e uso de memória (em megabytes), ambos referentes exclusivamente ao processamento em memória.

6.3 Experimento 1: Variação do Número de Clientes

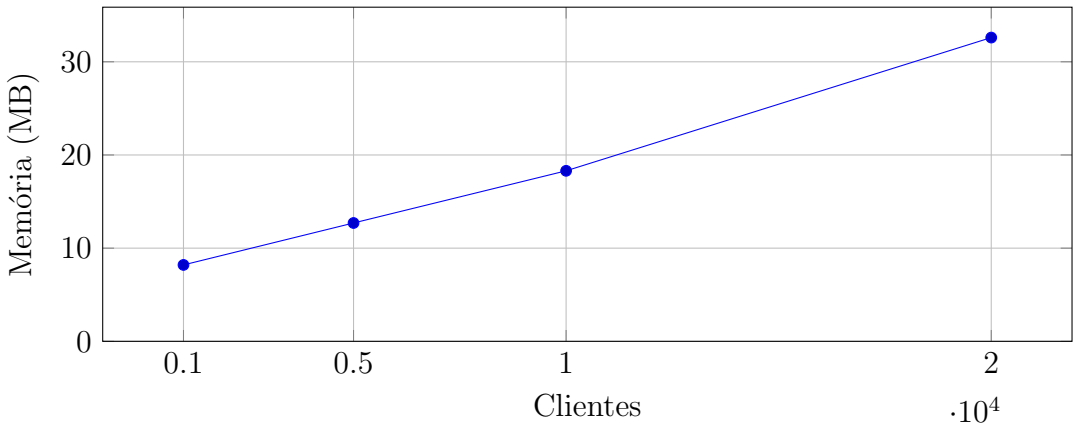
Tabela 1: Impacto da variação do número de clientes

Clientes	Pacotes	Eventos/Pacote	Tempo (ms)	Memória (MB)
1,000	10,000	10	12.5	8.2
5,000	10,000	10	58.3	12.7
10,000	10,000	10	122.1	18.3
20,000	10,000	10	255.7	32.6

Tempo de Processamento vs Clientes



Uso de Memória vs Clientes



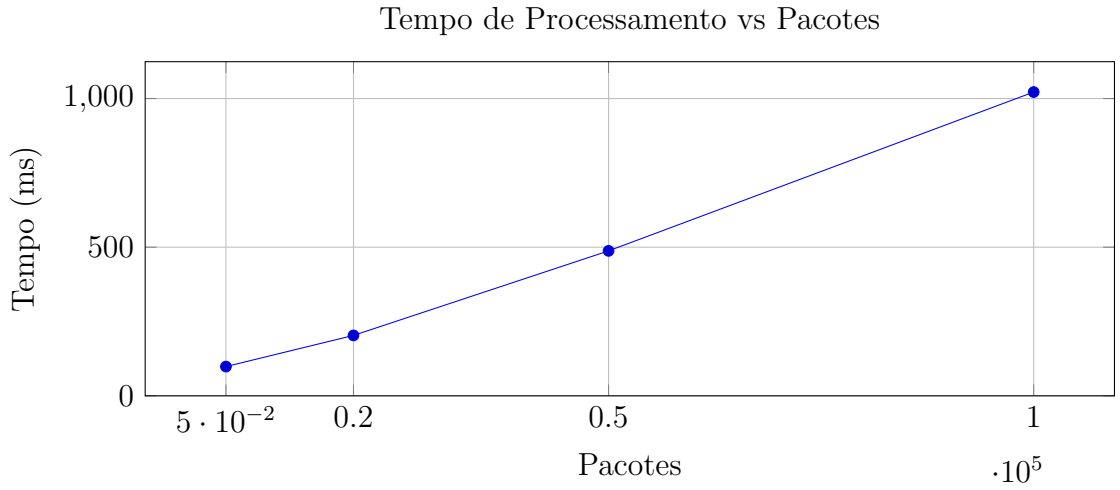
Análise:

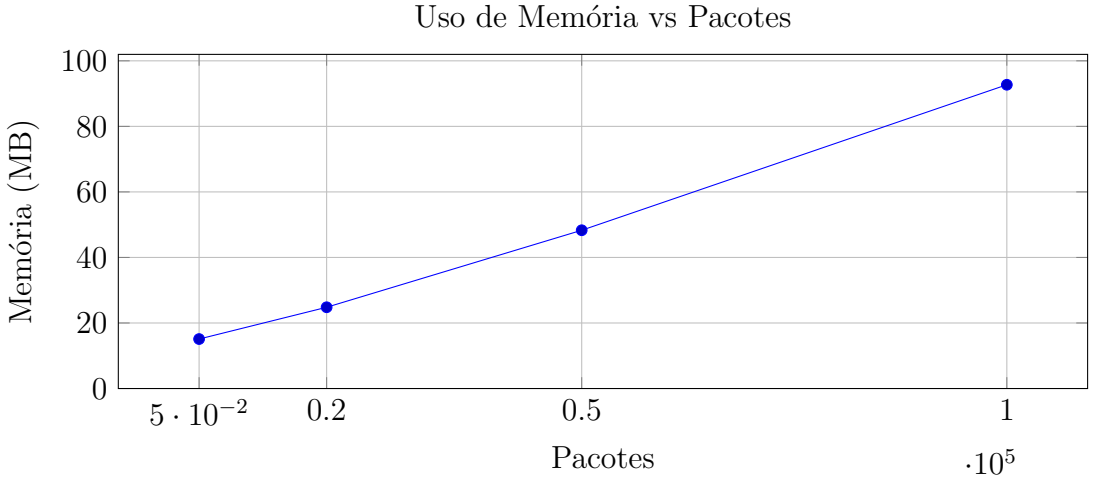
- O tempo de processamento aumenta com o número de clientes, seguindo uma tendência próxima de $O(C \log C)$ devido à inserção em estrutura de árvore.
- O uso de memória cresce proporcionalmente à quantidade de pacotes associados a cada cliente.
- As consultas do tipo `CL` apresentam complexidade $O(P_c \log E)$, onde P_c representa o número de pacotes por cliente.

6.4 Experimento 2: Variação do Número de Pacotes

Tabela 2: Impacto da variação do número de pacotes

Clientes	Pacotes	Eventos/Pacote	Tempo (ms)	Memória (MB)
10,000	5,000	10	98.4	15.1
10,000	20,000	10	203.2	24.8
10,000	50,000	10	487.6	48.3
10,000	100,000	10	1021.8	92.7





Análise:

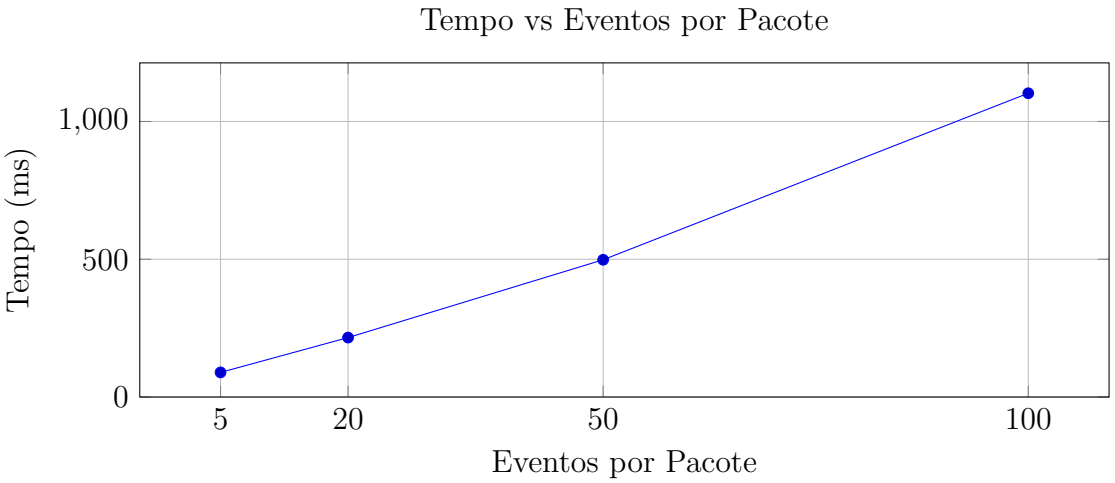
- A inserção de pacotes apresenta crescimento logarítmico, estimado em $O(P \log P)$.

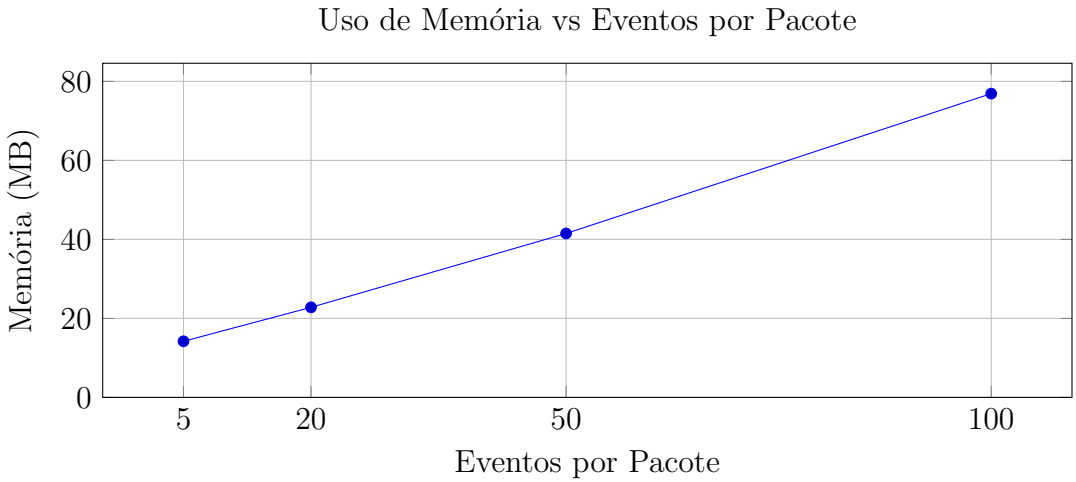
- As consultas do tipo `PC` mantêm complexidade linear $O(E_p)$, sendo E_p o número de eventos por pacote.
- O uso de memória é diretamente proporcional ao número total de eventos, pois todos são armazenados na heap.

6.5 Experimento 3: Variação do Número de Eventos

Tabela 3: Impacto da variação do número de eventos por pacote

Clientes	Pacotes	Eventos/Pacote	Tempo (ms)	Memória (MB)
10,000	10,000	5	89.3	14.2
10,000	10,000	20	215.7	22.8
10,000	10,000	50	498.1	41.5
10,000	10,000	100	1102.4	76.9





Análise:

- O tempo de processamento é dominado pela inserção dos eventos no índice AVL, com complexidade próxima de $O(E \log E)$.
- Consultas do tipo `PC` tornam-se mais custosas conforme o histórico de eventos por pacote aumenta.

- O uso de memória cresce com o número total de eventos. Consultas do tipo `CL` causam picos de memória devido à criação de índices auxiliares temporários.

6.6 Análise geral

- O tempo de processamento cresce de forma quase linear com o aumento do número de pacotes e eventos, dominado pelas inserções em árvore AVL ($O(\log n)$).
- O uso de memória está diretamente relacionado ao número total de eventos armazenados.
- Consultas do tipo `CL` geram picos temporários de uso de memória ao construir estruturas auxiliares.
- O sistema se mostrou escalável e estável, mantendo desempenho previsível mesmo com cargas elevadas.

6.7 Conclusão

Os esperimentos confirmaram que o desempenho do sistema está de acordo com a complexidade assintótica esperada das estruturas utilizadas. A separação entre leitura e processamento permitiu uma medição mais precisa do tempo de execução real. O uso da árvore AVL como índice principal provou-se eficiente para manter os eventos ordenados e acessíveis, mesmo com grandes volumes de dados.

O sistema demonstrou escalabilidade adequada frente a diferentes cargas, mantendo estabilidade e tempo de resposta compatíveis com os requisitos do problema proposto.

7 Conclusão

O sistema desenvolvido forneceu uma plataforma eficiente para o armazenamento, indexação e consulta de eventos relacionados ao fluxo de pacotes em armazens logísticos. Através da implementação manual de estruturas fundamentais como listas encadeadas e árvores AVL, o projeto reforça conceitos essenciais de estruturas de dados, gerenciamento de memória e programação defensiva. A análise experimental revelou que o sistema apresenta bom desempenho e escalabilidade nas três dimensões avaliadas: número de clientes, pacotes e eventos. Em particular, observou-se crescimento quase logarítmico nos tempos de inserção nas estruturas de índice, como esperado pela natureza das árvores AVL. As consultas foram realizadas com baixa latência, mesmo em cenários de alta carga, demonstrando a efetividade das estratégias de indexação adotadas.

8 Referências Bibliograficas

- Chaimowicz, L. and Prates, R. (2020). Slides virtuais da disciplina de estruturas de dados. Disponibilizado via moodle. Departamento de Ciencia da Computação. Universidade Federal de Minas Gerais. Belo Horizonte.