Disciplina: Programação Concorrente e Distribuida

Professor: João Robson

Grupo: Carlos Fernandes, Vinicius Lacerda, Wesley Pereira, Vitor Bittencurt

Data: 15 de junho de 2025

1. INTRODUÇÃO

Este relatório descreve o desenvolvimento de um sistema de busca distribuída para artigos científicos do arXiv. A implementação foi realizada em Java 17, utilizando comunicação via sockets, e o projeto seguiu princípios de computação distribuída para demonstrar escalabilidade, tolerância a falhas e processamento paralelo.

2. FUNDAMENTAÇÃO TEÓRICA

2.1 Computação Distribuída

A computação distribuída, um campo da ciência da computação, dedica-se ao estudo de sistemas compostos por múltiplos componentes computacionais autônomos que se comunicam em rede. Conforme Tanenbaum e Van Steen (2017), um sistema distribuído é definido como "uma coleção de computadores independentes que aparecem aos usuários como um único sistema coerente" (p. 2).

Sistemas distribuídos emergiram da necessidade de compartilhar recursos computacionais e dados entre diversas máquinas, possibilitando a divisão e o processamento paralelo de tarefas complexas. Essa abordagem permite o desenvolvimento de aplicações mais robustas, escaláveis e eficientes em comparação com sistemas centralizados.

No contexto deste projeto, a computação distribuída é aplicada através da segmentação do conjunto de dados de artigos científicos entre múltiplos servidores (B e C), coordenados por um servidor central (A). Essa arquitetura viabiliza a realização de buscas em paralelo, otimizando o tempo de resposta e distribuindo a carga computacional.2.2 Escalabilidade

A escalabilidade refere-se à capacidade de um sistema de gerenciar aumentos na carga de trabalho de maneira eficaz. Existem duas formas principais de escalabilidade:

- **Escalabilidade Vertical (Scale-up):** Expansão dos recursos de uma única máquina (CPU, memória, etc.).
- **Escalabilidade Horizontal (Scale-out):** Adição de mais máquinas ao sistema.

O sistema em desenvolvimento prioriza a escalabilidade horizontal. Sua arquitetura permite a integração de novos servidores de busca (D, E, F...) à medida que o conjunto de dados cresce,

sem demandar alterações significativas no código existente. O Servidor A pode ser facilmente adaptado para encaminhar requisições a N servidores, mantendo a mesma interface com o cliente.2.3 Tolerância a Falhas

A tolerância a falhas é a propriedade de um sistema de manter a operação correta mesmo diante da falha de alguns de seus componentes. Coulouris et al. (2012) enfatizam que "a tolerância a falhas é essencial em sistemas distribuídos porque a probabilidade de falha aumenta com o número de componentes" (p. 45).

Neste projeto, foram implementados mecanismos básicos de tolerância a falhas:

1. **Timeouts:** Evitam o bloqueio do sistema pela espera indefinida por respostas.
2. **Tratamento de Exceções:** Permite a captura e o tratamento de erros de comunicação.
3. **Independência dos Servidores:** A falha do Servidor B não impede o sistema de retornar resultados do Servidor C.
4. **Threads Separadas:** As falhas em uma requisição não interferem em outras em andamento.

2.4 Vantagens e Desvantagens da Arquitetura Distribuída

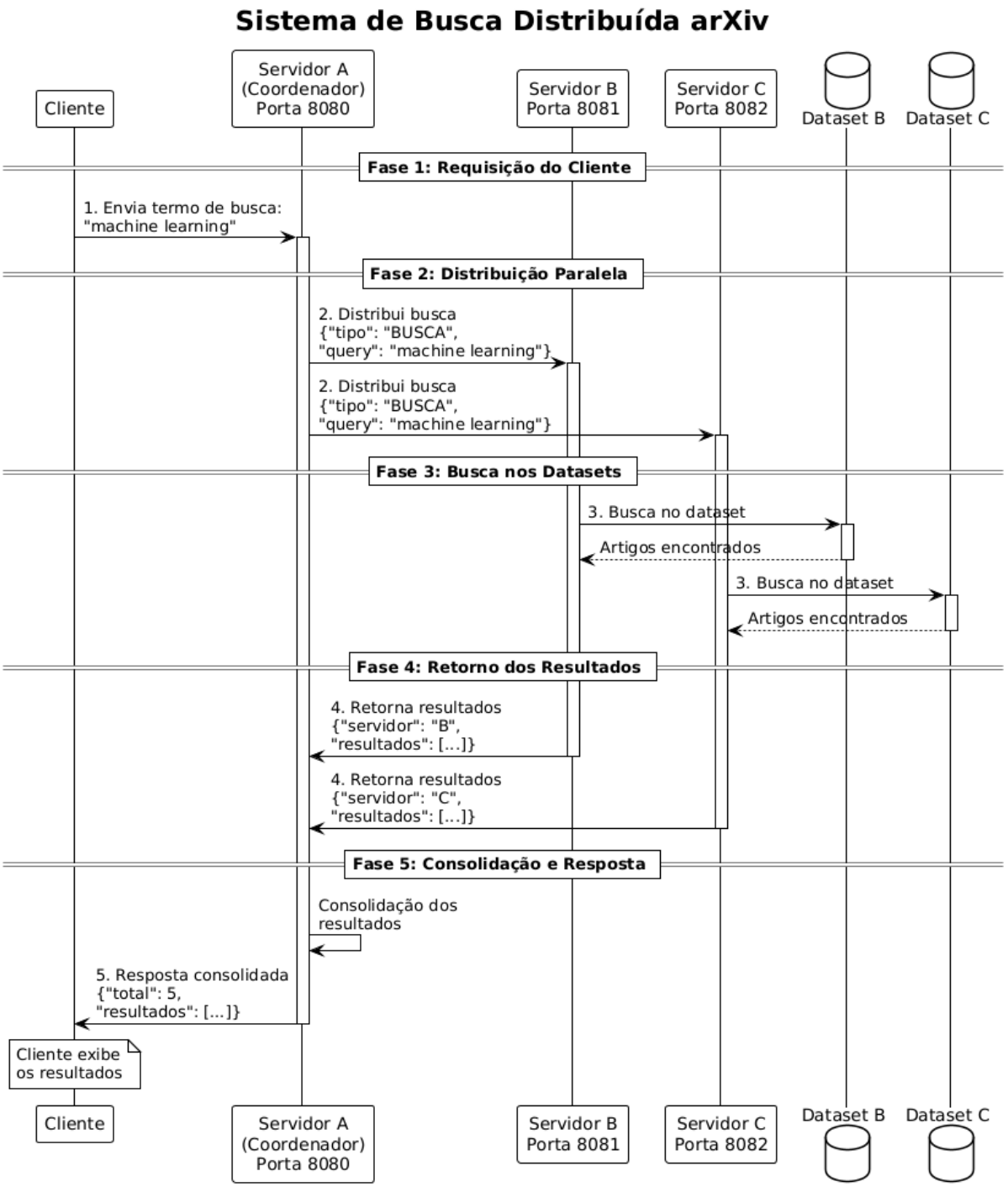
Vantagens:

- **Paralelismo:** Permite a execução simultânea de múltiplas buscas.
- **Escalabilidade:** Facilita a adição de novos servidores conforme a necessidade.
- **Balanceamento de Carga:** Distribuição equitativa do trabalho entre os servidores.
- **Modularidade:** Componentes independentes simplificam a manutenção e a evolução do sistema.

Desvantagens:

- **Complexidade:** A implementação e a depuração tornam-se mais desafiadoras.
- **Latência de Rede:** A comunicação entre servidores introduz um custo adicional.
- **Consistência:** Manter os dados sincronizados entre os servidores apresenta dificuldades.
- **Ponto Único de Falha:** O Servidor A é um componente crítico para a operação do sistema.

3. ARQUITETURA DA SOLUÇÃO
3.1 Diagrama de Comunicação



3.2 Formato de Dados3.2.1 Cliente → Servidor A

```
Unset
Formato: String simples
Exemplo: "machine learning"
```

3.2.2 Servidor A → Servidores B/C

```
Unset
{
 "tipo": "BUSCA",
 "query": "machine learning"
}
```

3.2.3 Servidores B/C → Servidor A

```
Unset
{
 "servidor": "Servidor B",
 "total": 2,
 "resultados": [
  {
   "title": "Deep Learning for Computer Vision",
   "abstract": "This paper presents...",
   "label": "cs.CV",
   "servidor": "Servidor B"
  }
 ]
}
```

3.2.4 Servidor A → Cliente

```
Unset
{
 "total": 5,
 "resultados": [
  {
   "title": "Deep Learning for Computer Vision",
   "abstract": "This paper presents...",
   "label": "cs.CV",
   "servidor": "Servidor B"
  },
  // ... mais resultados
 ]
}
```

3.3 Algoritmo de Busca: Boyer-Moore

3.3.1 Justificativa da Escolha

O algoritmo Boyer-Moore foi selecionado devido às suas vantagens para a busca de substrings em textos extensos:

1. **Eficiência:** Apresenta uma complexidade de O(n + m) no melhor caso, onde 'n' é o tamanho do texto e 'm' o tamanho do padrão.
2. **Heurísticas Inteligentes:** Utiliza duas heurísticas (Bad Character e Good Suffix) para otimizar a busca, evitando comparações desnecessárias.
3. **Performance com Padrões Longos:** Sua eficiência aumenta com o comprimento do padrão de busca.
4. **Adequação ao Contexto:** É ideal para a busca de termos técnicos e científicos em abstracts longos.

3.3.2 Funcionamento

O algoritmo Boyer-Moore compara o padrão de busca da direita para a esquerda, permitindo saltos maiores no texto quando encontra caracteres incompatíveis.

- **Bad Character Rule:** Ao encontrar um caractere no texto que não corresponde ao caractere atual no padrão, o algoritmo verifica a última ocorrência desse caractere no padrão e desloca o padrão para a direita de acordo.
- **Good Suffix Rule:** Quando uma parte do padrão corresponde ao texto, mas ocorre uma falha em outro ponto, essa regra utiliza a informação do sufixo correspondente para determinar o próximo alinhamento válido do padrão.

Exemplo Simplificado:

```
Unset
Texto:  "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
Padrão: "KLMN"

1ª tentativa: Compara N com D (posição 3) - não corresponde
  - Bad Character: D não está em "KLMN", pula 4 posições

2ª tentativa: Agora alinhado corretamente, encontra o padrão
```

4. IMPLEMENTAÇÃO E TESTES

O sistema foi implementado seguindo os princípios da programação orientada a objetos, com classes bem definidas e responsabilidades distintas:

- **ServidorA:** Coordenador responsável pela distribuição das requisições de busca.
- **ServidorBusca:** Classe abstrata que serve como base para os servidores de busca.
- **ServidorB/C:** Implementações concretas dos servidores de busca que realizam a busca no dataset local.
- **Cliente:** Interface de usuário para a submissão de termos de busca.
- **BoyerMoore:** Implementação específica do algoritmo de busca Boyer-Moore.

5. CONCLUSÃO

O desenvolvimento deste projeto permitiu a aplicação prática de conceitos fundamentais de sistemas distribuídos. A implementação demonstrou tanto as vantagens quanto os desafios de trabalhar com arquiteturas distribuídas, desde o ganho de performance através do paralelismo até a complexidade adicional no tratamento de falhas e comunicação entre componentes.

O sistema desenvolvido atende aos requisitos propostos e oferece uma base sólida para futuras expansões, como a adição de novos servidores de busca, implementação de cache distribuído, ou algoritmos de busca mais sofisticados.

REFERÊNCIAS BIBLIOGRÁFICAS

BOYER, R. S.; MOORE, J. S. A fast string searching algorithm. **Communications of the ACM**, v. 20, n. 10, p. 762-772, 1977.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T.; BLAIR, G. **Distributed Systems: Concepts and Design**. 5. ed. Boston: Addison-Wesley, 2012.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Introduction to Algorithms**. 3. ed. Cambridge: MIT Press, 2009.

TANENBAUM, A. S.; VAN STEEN, M. **Distributed Systems: Principles and Paradigms**. 3. ed. Boston: Pearson, 2017.

KNUTH, D. E.; MORRIS, J. H.; PRATT, V. R. Fast pattern matching in strings. **SIAM Journal on Computing**, v. 6, n. 2, p. 323-350, 1977.