# Trabalho Final Programação Concorrente e Distribuída

## Relatório Técnico: Aplicação Distribuída para Consulta de Dados em JSON com Java Sockets

### 1. Resumo Executivo

Este relatório detalha a arquitetura e a implementação de uma aplicação distribuída desenvolvida em **Java**. O sistema simula um ambiente de busca de dados em larga escala, utilizando **Java Sockets** para comunicação TCP, **Threads** para processamento concorrente e **JSON** como formato para a base de dados. O objetivo principal é demonstrar um modelo de processamento paralelo onde uma consulta é dividida, processada por múltiplos servidores autônomos e, por fim, consolidada antes de ser entregue ao usuário final.

---

### 2. Arquitetura do Sistema

O sistema é composto por quatro componentes principais que interagem de forma orquestrada.

#### 2.1. Cliente (`Client.java`)

- **Responsabilidade:** Interface do usuário e ponto de entrada da consulta.
- **Funcionamento:**
  1. Estabelece uma conexão via socket **TCP** com o Servidor Central.
  2. Envia uma consulta textual (palavra-chave), fornecida como argumento de linha de comando.
  3. Aguarda e, ao receber a resposta do servidor, exibe os resultados formatados no console.

#### 2.2. Servidor Central (`AServer.java`)

- **Responsabilidade:** Atuar como orquestrador e gateway de comunicação.
- **Funcionamento:**
  1. Opera em uma porta de rede predefinida, aguardando conexões de clientes.
  2. Ao receber uma consulta, a repassa simultaneamente para os dois servidores de busca (`BServer` e `CServer`).
  3. Coleta e agrega as respostas parciais de cada servidor.
  4. Formata os dados consolidados em uma única string legível.

5. Retorna a resposta final ao cliente através da conexão socket estabelecida.

### 2.3. Servidores de Busca / Workers (`BServer.java` e `CServer.java`)

- **Responsabilidade:** Realizar a busca em um subconjunto específico dos dados.
- **Funcionamento:**
  1. Ambos os servidores herdam de uma classe base (`WorkerServer`) e operam em portas distintas.
  2. Durante a inicialização, cada servidor carrega uma partição da base de dados JSON em memória:
     - `BServer`: Carrega a primeira metade do array de dados.
     - `CServer`: Carrega a segunda metade do array de dados.
  3. Aguardam requisições do Servidor Central.
  4. Ao receber uma consulta, realizam uma busca local e retornam, linha por linha, os objetos JSON que contêm o termo pesquisado nos campos `"title"` ou `"abstract"`.

---

## 3. Estrutura de Dados e Fluxo da Consulta

### 3.1. Base de Dados

A fonte de dados é um arquivo único, `dados_servidor_c.json`, contendo um array de objetos JSON. Cada objeto possui a seguinte estrutura:

JSON

```
[
  {
    "title": "Título da Obra",
    "abstract": "Resumo descritivo da obra..."
  },
  ...
]
```

A leitura e manipulação dos dados em memória são realizadas com o auxílio da biblioteca **org.json**.

### 3.2. Fluxo de Execução da Consulta

O processo de busca ocorre em seis etapas principais:

1. **Inicialização e Particionamento:** Os servidores `BServer` e `CServer` são iniciados e cada um carrega sua respectiva metade da base de dados.
2. **Requisição do Cliente:** O usuário executa o `Client.java` com uma palavra-chave como argumento.
3. **Distribuição da Consulta:** O Servidor Central ( `AServer` ) recebe a palavra-chave e a encaminha para os dois workers.
4. **Processamento Paralelo:** Cada worker executa a busca em sua partição de dados de forma independente e concorrente. A busca não diferencia maiúsculas de minúsculas e verifica a presença do termo nos campos `title` e `abstract` .
5. **Agregação e Formatação:** O Servidor Central coleta os resultados (strings JSON) de ambos os workers, interpreta cada string como um `JSONObject` , e monta uma resposta final formatada, apresentando o título e o resumo de cada resultado de forma legível.
6. **Retorno ao Cliente:** A string com os resultados consolidados é enviada de volta ao cliente, que a exibe no terminal.

---

## 4. Tecnologias Empregadas

- **Java Sockets:** Utilizados para a comunicação de baixo nível baseada em TCP/IP entre todos os componentes da aplicação.
- **Threads:** Essenciais para a concorrência. O Servidor Central utiliza threads para lidar com múltiplos clientes, e a própria arquitetura permite que os workers processem em paralelo.
- **Biblioteca `org.json` :** Empregada para o parsing eficiente e a manipulação de dados no formato JSON.
- **Entrada/Saída (I/O) de Arquivos:** Utilizada para a leitura inicial da base de dados `dados_servidor_c.json` .

---

## 5. Conclusão

A aplicação implementada demonstrou com sucesso a viabilidade do uso de **Java Sockets** para construir uma arquitetura distribuída funcional, modular e escalável. O paralelismo na busca, a centralização da lógica de orquestração e o uso de JSON como formato de dados flexível simulam com eficácia os desafios e as soluções encontradas em sistemas distribuídos do mundo real. O projeto serve como uma excelente base prática para o estudo de conceitos de computação distribuída e concorrência.