Relatório Técnico: Sistema de Votação Distribuído

ALUNOS: Arthur Renato Normando Vasconcelos, Bruno Vaz Ferreira

1. Introdução

O objetivo principal deste projeto foi desenvolver um protótipo funcional de um sistema de votação distribuído. O sistema foi projetado para demonstrar conceitos essenciais de sistemas distribuídos, incluindo comunicação em rede, tratamento de concorrência e gerenciamento de estado centralizado, atendendo a um conjunto predefinido de requisitos técnicos. Este documento serve como uma documentação formal da solução implementada.

2. Arquitetura do Sistema

2.1. Modelo Adotado: Cliente-Servidor

A arquitetura implementada é a do tipo Cliente-Servidor. Esta é composta por um nó servidor central, que detém a autoridade sobre o estado e os dados da aplicação, e múltiplos nós clientes, que interagem com o servidor para utilizar seus serviços.

- **Nó Servidor:** Um processo único responsável por gerenciar o estado da votação (aberta/fechada) e armazenar a contagem de votos de forma segura e consistente.
- **Nós Clientes:** Processos independentes que atuam como terminais de votação, provendo a interface para o eleitor e comunicando as intenções de voto ao servidor.

2.2. Justificativa da Escolha

A escolha da arquitetura Cliente-Servidor foi motivada pelos seguintes fatores:

- **Consistência de Dados:** A centralização da lógica de negócio e do armazenamento de votos no servidor é crucial para garantir que não haja inconsistências na contagem.
- **Controle de Acesso:** Um servidor central facilita a implementação de um ponto único de controle para habilitar ou desabilitar o processo de votação para todos os clientes.
- **Escalabilidade e Separação de Responsabilidades:** O modelo permite que novos clientes sejam adicionados sem qualquer modificação no servidor. As responsabilidades são claramente divididas: o servidor gerencia a lógica e os dados, enquanto os clientes gerenciam a interação com o usuário.

3. Tecnologias e Ferramentas

- **Linguagem de Programação:** Foi utilizada a linguagem **Java**, devido ao seu robusto ecossistema e suporte nativo para programação de rede (pacote `java.net`) e concorrência (API de Threads), que são os pilares tecnológicos do projeto.
- **Tecnologia de Comunicação:** A comunicação em rede foi implementada utilizando a **API de Sockets Java**, que provê um mecanismo para a troca de dados baseada em streams sobre o protocolo TCP/IP.

4. Análise da Tecnologia de Comunicação: Socket vs. RPC

A decisão pela utilização de Sockets, em detrimento de alternativas de mais alto nível como RPC (Remote Procedure Call), baseou-se nos seguintes critérios:

1. **Controle e Didatismo:** Sockets oferecem um controle de baixo nível sobre a comunicação. Isso exigiu a definição de um protocolo de aplicação textual (`VOTE:SYNC:20`), o que é pedagogicamente valioso por tornar explícitos os mecanismos de serialização, envio e interpretação de mensagens, conceitos que seriam abstraídos por um framework RPC.
2. **Adequação ao Problema:** A comunicação no sistema se resume à troca de mensagens simples. A complexidade inerente à configuração de um sistema RPC (que frequentemente envolve a definição de interfaces via IDL e geração de stubs/skeletons) não se justificaria, apresentando um overhead desnecessário para o escopo do projeto. Sockets proveram uma solução mais direta e leve.
3. **Flexibilidade e Interoperabilidade:** O protocolo textual definido é inerentemente agnóstico à linguagem de programação. Isso significa que clientes desenvolvidos em outras tecnologias poderiam se comunicar com o servidor Java com mínimo esforço, uma vantagem em ambientes heterogêneos.

5. Análise dos Componentes de Software (Classes)

5.1. Classe `VotingServer`

- **Função:** Componente central do sistema. É responsável por iniciar o serviço, gerenciar o ciclo de vida das conexões dos clientes e manter a integridade do estado da votação.
- **Métodos Principais:**
    - `startServer()`: Instancia o `ServerSocket`, cria um pool de threads para gerenciar clientes concorrentes e entra no loop principal de aceitação de conexões (`serverSocket.accept()`).
    - `registerVote(int)`: Método `synchronized` que centraliza a lógica de registro de votos, garantindo atomicidade na atualização da contagem.
    - `setVotingOpen(boolean)`: Implementa o controle de acesso, permitindo que o estado da votação seja alterado.

5.2. Classe `ClientHandler`

- **Função:** Atua como um delegado do servidor para cada cliente conectado. Cada instância é executada em uma thread própria, isolando a comunicação de um cliente dos demais.
- **Métodos Principais:**
    - `run()`: Contém a lógica de execução da thread, lendo e processando mensagens do cliente.
    - `processMessage(String)`: Decodifica o protocolo de aplicação para acionar as ações correspondentes no `VotingServer`.

5.3. Classe `ServerAdminConsole`

- **Função:** Fornece uma interface de linha de comando para o gerenciamento do servidor, rodando em uma thread dedicada.
- **Métodos Principais:**
    - `run()`: Loop principal que aguarda e processa a entrada de comandos do administrador (`abrir`, `fechar`, `resultados`).

5.4. Classe `VotingClient`

- **Função:** Representa o nó cliente. É um programa autônomo que simula um terminal de votação.
- **Métodos Principais:**
    - `main()`: Contém o loop de sessão do eleitor. Para cada eleitor, estabelece uma nova conexão, envia um único voto e encerra a conexão usando um bloco `try-with-resources`.

6. Descrição do Fluxo de Comunicação

O fluxo para o registro de um único voto é o seguinte:

1. **Inicialização:** O `VotingServer` é iniciado e aguarda conexões na porta TCP 12345. O administrador, via `ServerAdminConsole`, executa o comando `abrir`.

```
Servidor de Votação iniciado na porta 12345
Aguardando conexões de clientes...

--- Console de Administração do Servidor ---
Comandos disponíveis:
  abrir      - Abre a sessão de votação
  fechar     - Fecha a sessão de votação
  resultados - Mostra a contagem de votos atual
  menu       - Mostra este menu de ajuda
  sair       - Encerra o servidor
-----------------------------------------
Comando do admin> abrir

--- A VOTAÇÃO FOI ABERTA ---
Comando do admin>
```

2. **Conexão do Cliente:** O `VotingClient` é executado, solicita o nome do eleitor e instancia um `Socket`, estabelecendo uma conexão com o servidor.

```
--- Terminal de Votação Iniciado ---

================================================
Digite o nome do eleitor (ou 'sair' para encerrar): bruno
Conectado ao servidor para 'bruno'.

--- Opções para bruno ---
1. Votar (Modo Síncrono)
2. Votar (Modo Assíncrono)
Escolha uma opção:
```

3. **Aceitação e Delegação:** O `ServerSocket` no servidor aceita a conexão e delega o `Socket` resultante para uma nova instância de `ClientHandler`, executada em uma thread do pool.

```
--- A VOTAÇÃO FOI ABERTA ---
Comando do admin> Novo cliente conectado: 'bruno' (127.0.0.1)
```

4. **Identificação:** O cliente envia uma mensagem de identificação (`IDENTIFY:<nome>`).

```
Enviando voto (SYNC)...

[Servidor]: VOTO_CONFIRMADO: Seu voto para 10 foi registrado.
Voto enviado. Encerrando sessão de 'bruno'.

================================================
Digite o nome do eleitor (ou 'sair' para encerrar):
```

5. **Requisição de Voto:** O cliente envia a mensagem de voto (`VOTE:SYNC:20`).
6. **Processamento no Servidor:** O `ClientHandler` recebe, decodifica a mensagem e invoca o método `server.registerVote(20)`. A execução é

serializada (thread-safe). O método atualiza a contagem e retorna uma string de confirmação.
7. **Resposta ao Cliente:** O `ClientHandler` envia a string de confirmação de volta ao cliente.
8. **Encerramento da Conexão:** O cliente recebe a resposta e o `try-with-resources` fecha o `Socket`. A thread do `ClientHandler` no servidor é finalizada e retorna ao pool.

7. Conclusão

O sistema de votação distribuído foi implementado com sucesso, atendendo a todos os requisitos. A arquitetura cliente-servidor mostrou-se robusta e a utilização de Sockets permitiu um controle explícito e didático sobre a comunicação em rede, alinhado com o objetivo de demonstrar os fundamentos da comunicação em sistemas distribuídos em detrimento das abstrações de alto nível oferecidas por RPC.