Relátorio TP 02 - Algoritmo de Berkeley para sincronização de relógios e algoritmo de Bully para eleição de coordenador

Daniel de Souza Rodrigues - 18.2.8112, Thais Souto Damasceno - 18.2.8013

[1]Instituto de Ciências Exatas e Aplicadas – Universidade Federal de Ouro Preto (UFOP)
Rua 36, Número 115 - Bairro Loanda, João Monlevade - CEP: 35931-008

daniel.sr@aluno.ufop.edu.br

Abstract. *This work aims to present the operation and practical implementation of two objectives used in Berkeley's implementation for clock synchronization and Algorithm of attempts like them (Bully).*

1. Objetivo do trabalho

Esse trabalho tem como objetivo fixar os conhecimentos obtidos em sala de aula sobre computação distribuída, no estudo em questão iremos implementar o algoritmo de Berkeley utilizado para sincronização de relógios e o algoritmo de Bully (Valentão) que é utilizado para eleições de coordenador em sistemas distribuído.

2. Introdução

Para termos uma melhor compreensão do trabalho implementado, abordaremos nas subseções seguintes uma breve explicação sobre o funcionamento de cada algoritmo que é utilizado como base para construção do trabalho.

2.1. Algoritmo de Cristian

O Algoritmo de Cristian é um algoritmo de sincronização de relógio usado para sincronizar o tempo com um servidor de tempo por processos clientes. Esse algoritmo funciona bem com redes de baixa latência, onde o tempo de ida e volta é curto em comparação com a precisão, enquanto sistemas/aplicativos distribuídos propensos à redundância não andam de mãos dadas com esse algoritmo. Aqui Round Trip Time refere-se à duração de tempo entre o início de uma solicitação e o final da resposta correspondente.[Gee b]

1. O processo na máquina cliente envia a solicitação para buscar a hora do relógio (hora no servidor) para o Servidor de Relógio na hora $T1$.
2. O Clock Server escuta a requisição feita pelo processo cliente e retorna a resposta na forma de clock do servidor.
3. O processo do cliente busca a resposta do Clock Server na hora $T1$ e calcula a hora do relógio do cliente sincronizado usando a fórmula abaixo.

   $$Tcliente = Tserver + (T1 - T0)/2$$

   A hora no lado do cliente difere da hora real em, no máximo, $(T1 - T0)/2$ segundos. A imagem abaixo ilustra o funcionamento do algoritmo.

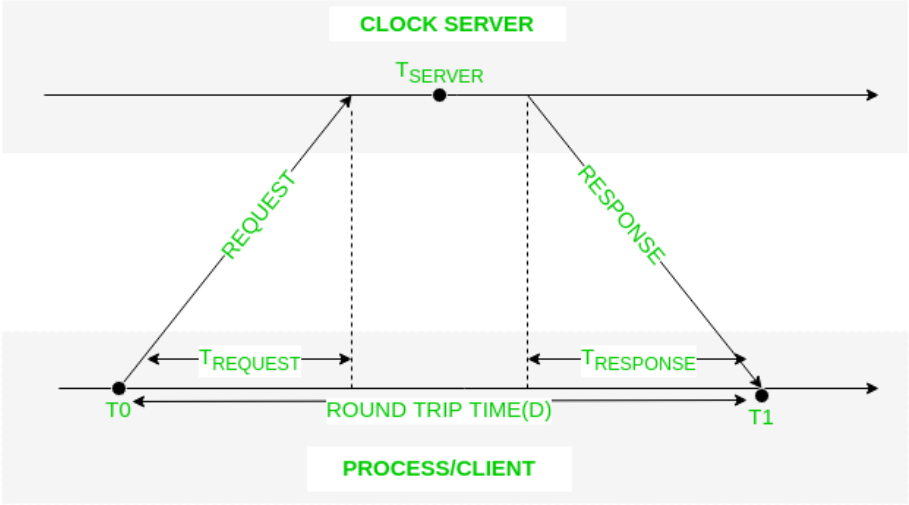

Figura 1. Algoritmo de Cristian

2.2. Algoritmo de Berkeley

O Algoritmo de Berkeley é uma técnica de sincronização de relógio usada em sistemas distribuídos. O algoritmo assume que cada nó de máquina na rede não possui uma fonte de tempo precisa ou não possui um servidor UTC.[Gee a]

1. Um nó individual é escolhido como nó mestre de um nó de pool na rede. Este nó é o nó principal na rede que atua como mestre e os demais nós atuam como escravos. O nó mestre é escolhido usando um processo de eleição/algoritmo de eleição do líder.
2. O nó mestre periodicamente pinga os nós escravos e busca a hora do relógio para eles usando o algoritmo de Cristian.
   A imagem abaixo ilustra como o mestre envia solicitações aos nós escravos:

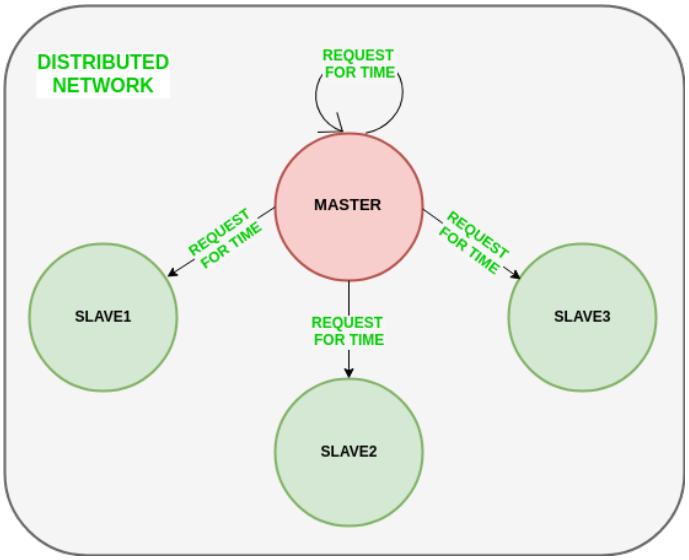

Figura 2. Algoritmo de Berkeley - Solicitações da Master

A imagem abaixo ilustra como os nós escravos enviam de volta o tempo dado pelo relógio do sistema.

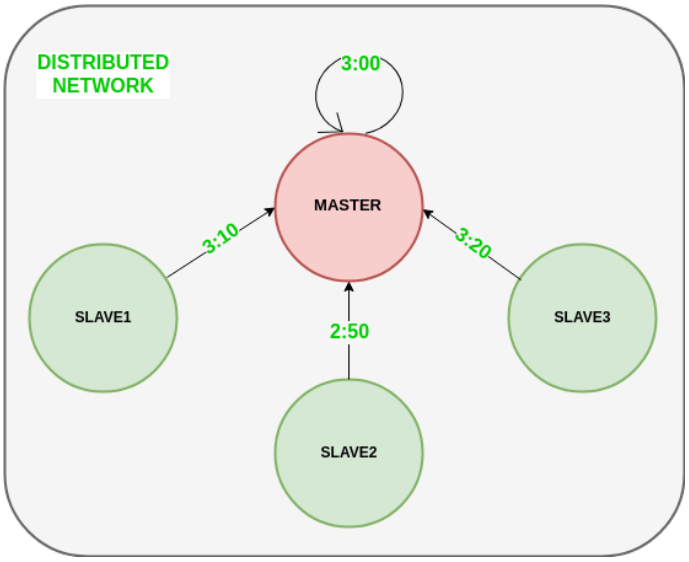

Figura 3. Algoritmo de Berkeley - Resposta dos Slaves

3. O nó mestre calcula a diferença de tempo médio entre todos os tempos de relógio recebidos e o tempo de relógio dado pelo próprio relógio do sistema do mestre. Essa diferença de tempo média é adicionada à hora atual no relógio do sistema do mestre e transmitida pela rede.

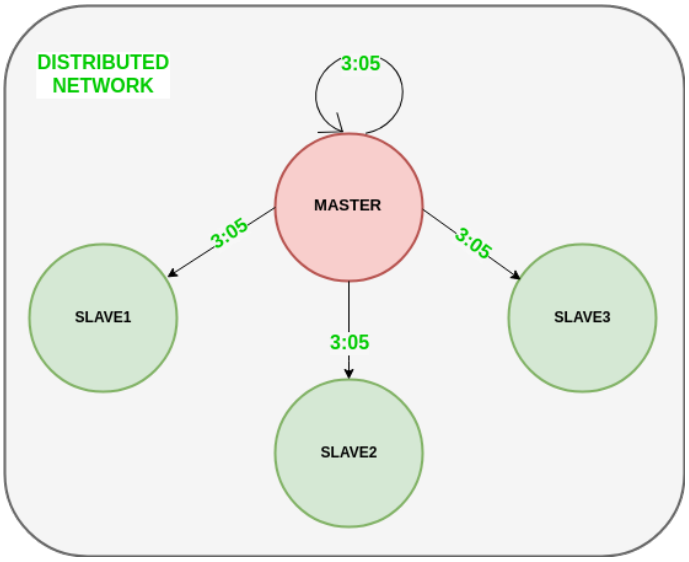

Figura 4. Algoritmo de Berkeley - Sincronização do Tempo

2.3. Algoritmo de Bully

O algoritmo de bully serve para eleger um líder entre processos identificados por um identificador numérico, único, fixo e atribuído antes do início da eleição. Entretanto neste caso a topologia não é limitada a um anel e cada um dos processos pode se comunicar com qualquer outro no sistema. Novamente a execução do algoritmo busca eleger o processo de maior identificador e fazer com que todos reconheçam o novo líder.[Gee c]

1. Se um dos processos identifica a perda de contato com o líder, inicia uma nova eleição enviando a todos os outros uma mensagem contendo seu identificador.
2. Todos os nós respondem ao processo que iniciou a eleição com os seus próprios identificadores.
3. Se o processo que iniciou a eleição verifica possuir o maior identificador entre todos os outros, proclama-se líder e avisa todos os outros. Senão aguarda que o processo de maior identificador inicie uma eleição e se torne líder.

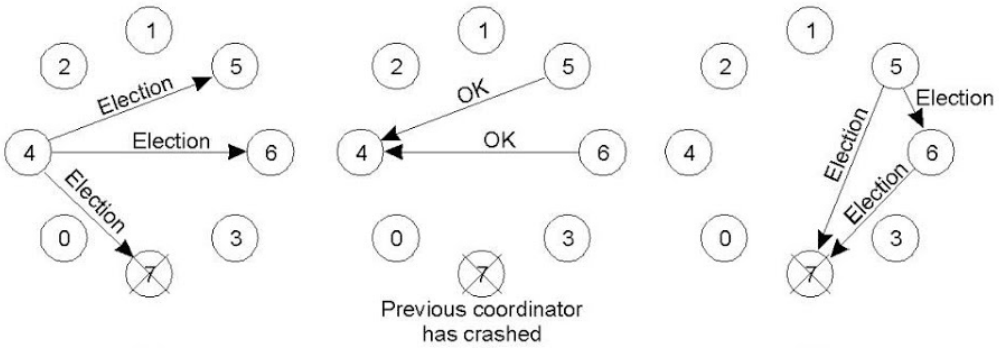

Figura 5. Algoritmo de Bully - Processo de Eleições

3. Implementação - Algoritmo de Berkeley

Nas próximas subseções será apresentado todo o passo a passo, considerações e resultados obtidos com a implementação do algoritmo de Berkeley.[Git b]

3.1. Considerações

O algoritmo foi construído de forma a ser utilizado manualmente para testes, sendo necessário ainda a implementação do algoritmo de eleição (Bully) para complementar seu funcionamento.

3.2. Tecnologias utilizadas

- Java SE 8 (JDK 8u202)
- Java RMI (Para construção das chamadas em RPC) [Dev]
- Oracle Virtual Machine (Para criação das máquinas virtuais e configuração da rede interna de computadores)
- IDE Eclipse - 2021/3 (Para execução da aplicação)
- Windows 7 - (Para realização dos testes)

3.3. Classes utilizadas

Todas as classes que foram construídas utilizando java:

- **MainClient** - Classe construída para utilização dos nodos em modo cliente, possui a chave de conexão (IP:PORT) que vincula o cliente ao servidor, utilizando Java RMI, realiza a adição do cliente a lista de registros do RMI e possui o algoritmo de Berkeley implementado. Sendo assim ela é a principal classe de requisições que são enviadas ao servidor de relógio sempre que um cliente se conecta à rede.
- **MainServer** - Classe construída para ser a ponte entre o servidor de relógio e o cliente, possui a porta do servidor e implementa o registro de RMI do servidor. Utiliza-se de um método rebind para garantir a possibilidade de novas conexões sem a necessidade de reiniciar o servidor por completo.
- **Connection** - Constrói o objeto de conexão, disponibiliza o endereço e a porta utilizada.
- **InterfaceServerTime** - Implementa a interface e declara o protótipo da função getTime().
- **ServerTimeImpl** - Classe construída para gerenciar o servidor de relógio, utiliza-se herança da classe de conexão RMI UnicastRemoteObject e implementa a interface ServerTime, implementa a função getTime() e setTime() utilizando uma instância de tempo que é gerada a partir da classe Time.
- **Time** - Classe construída para coleta e geração do tempo, utiliza-se de um cálculo baseado em números randômicos para geração do tempo do servidor e do tempo dos clientes que estão conectados, implementa a interface de serialização para complementar a utilização do LocalDateTime do java.

3.4. Como executar o projeto?

Todo o código encontra-se disponível no Github link: Algoritmo de Berkeley - Daniel Rodrigues, para execução é necessário ter o java instalado nas máquinas clientes e servidores também deve-se à configuração da variável de ambiente em cada uma delas.

1. Usar o Prompt para startar os registros do RMI na máquina servidora com o comando: start rmiregistry [Git a]
2. Definir os IPs do servidor de relógio no arquivo MainClient, basta criar uma nova connections.add(new Connection(“IP DO SERVIDOR DE RELOGIO”, PORTA DO SERVIDOR))
3. Executar os clientes em máquinas diferentes, recomenda-se utilizar máquinas virtuais para realizar os teste.

3.5. Resultados obtidos com o Berkeley

Foram utilizadas 3 máquinas virtuais configuradas com Windows 7 e uma rede interna foi construída para realização dos testes.

1. Nodo1 - 192.168.0.2:4500 - Cliente
2. Nodo2 - 192.168.0.3:4500 - Servidor de Relógio
3. Nodo3 - 192.168.0.4:4500 - Cliente

Servidor de tempo sendo executado no Nodo2 - Horário do servidor: 19:55

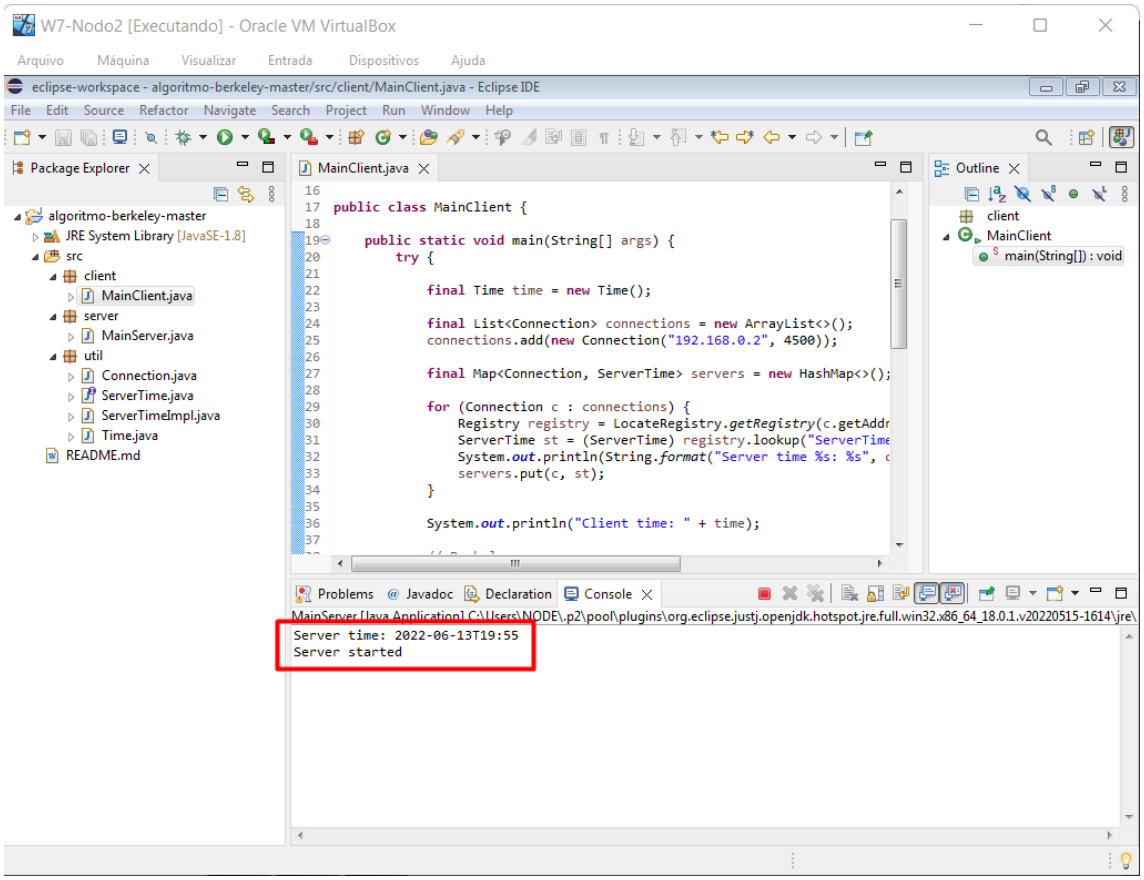

Figura 6. Algoritmo de Berkeley - Servidor de Tempo executando - Nodo2

Cliente Nodo1 entra na rede com um tempo randômico, o servidor solicita o seu tempo, realiza o ajuste e o cliente reajusta seu tempo.

- Tempo do Servidor: 19:55
- Tempo do Cliente: 03:43
- Resultado de Berkeley: 11:49
- Tempo do Servidor ajustado para: 11:49
- Tempo do Cliente ajustado para: 11:49

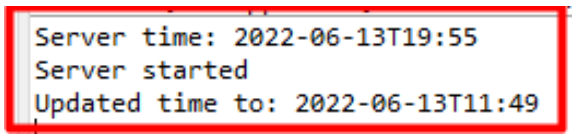

Figura 7. Algoritmo de Berkeley - Servidor ajusta seu próprio tempo para: 11:49 - Nodo2

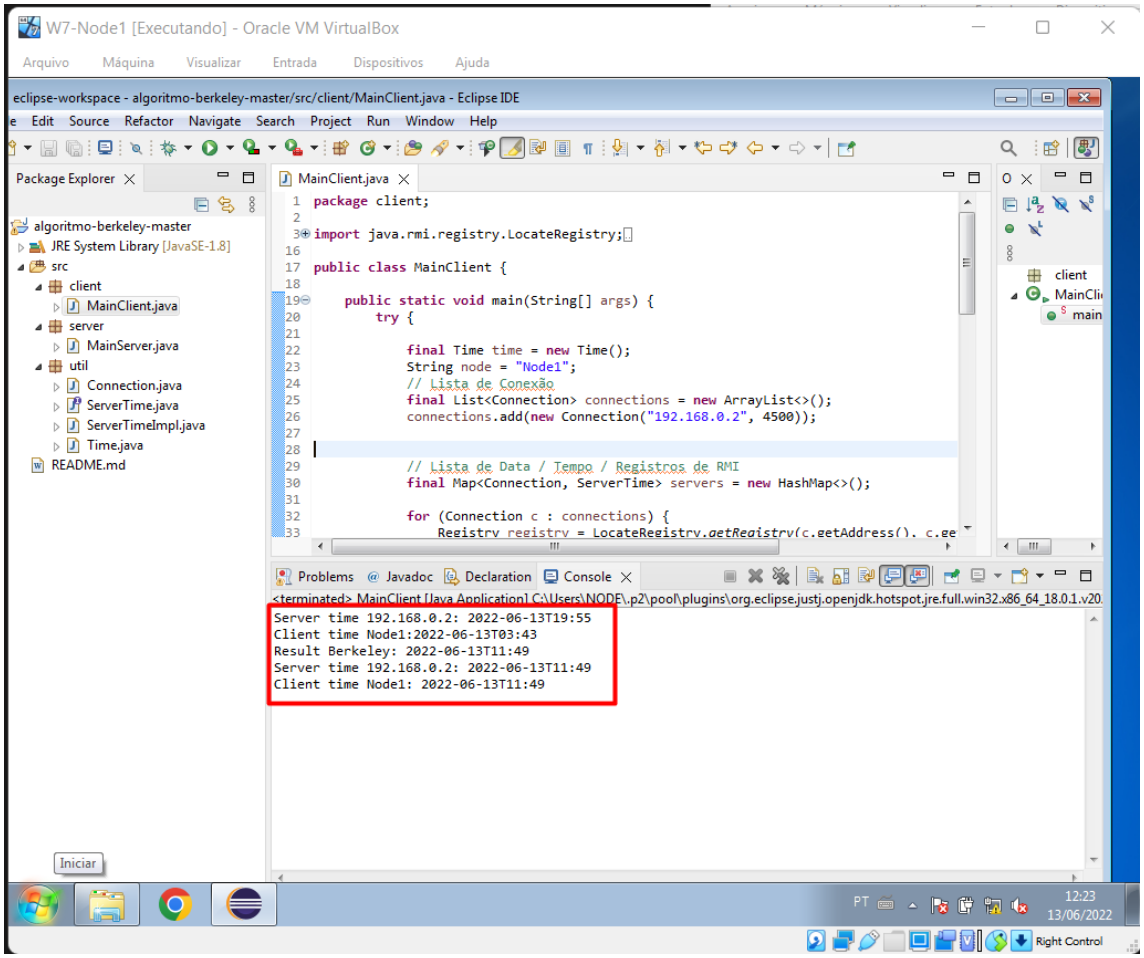

Figura 8. Algoritmo de Berkeley - Cliente entra na rede - Nodo1

O Cliente Nodo3 entra na rede com um tempo randômico, o servidor solicita o seu tempo e realiza o ajuste de tempo nos clientes e nele próprio.

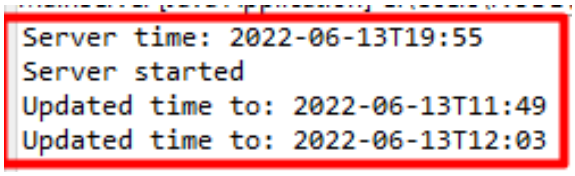

Figura 9. Algoritmo de Berkeley - Servidor ajusta seu próprio tempo e o dos clientes para: 12:03 - Nodo2

- Tempo do Servidor: 11:49
- Tempo do Cliente: 12:17
- Resultado de Berkeley: 12:03
- Tempo do Servidor ajustado para: 12:03
- Tempo do Cliente ajustado para: 12:03

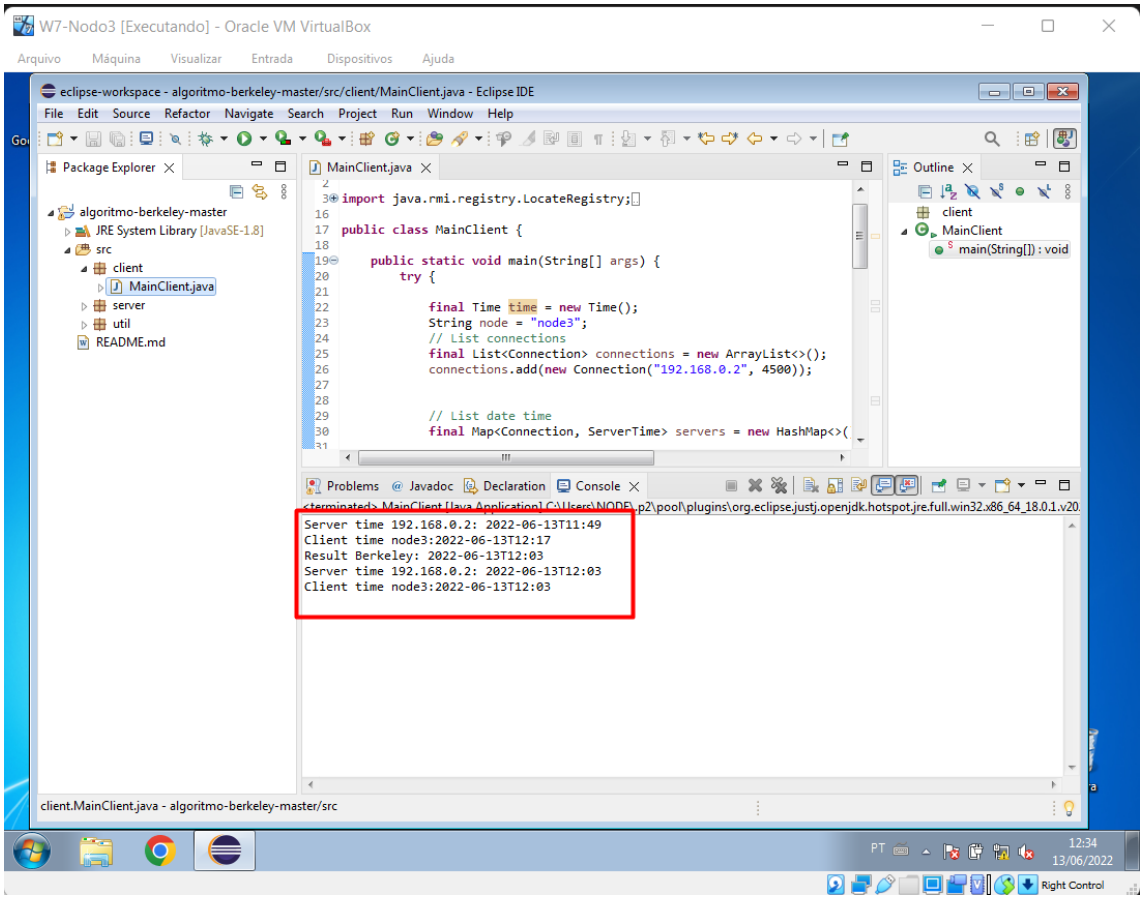

Figura 10. Algoritmo de Berkeley - Servidor ajusta seu próprio tempo e o dos clientes para: 12:03 - Nodo2

3.6. Conclusão sobre o algoritmo de Berkeley e sua implementação

O algoritmo funciona perfeitamente e é possível com poucas modificações no código utilizar e ajustar o relógio do próprio sistema operacional, o grande problema nessa situação seria que após a primeira execução, não seria possível realizar mais testes com o algoritmos e porque os nós se sincronizarão com o mesmo tempo do sistema coordenador, em contrapartida encontrar conteúdo para utilizar como base de implementação foi muito difícil o que ocasionou falhas na construção do mesmo, onde o objetivo era ter um algoritmo totalmente automatizado com o sistema de eleição incluso caso ocorrerem falhas em um dos nodos da rede.

4. Implementação - Algoritmo de Bully

Nas próximas subseções será apresentado todo o passo a passo, considerações e resultados obtidos com a implementação do algoritmo de eleição (Bully).

4.1. Considerações

O algoritmo foi construído de forma a ser utilizado manualmente para testes, onde cada nodo de rede pode ser instanciado via terminal, com algumas mudanças no código e com a criação de uma biblioteca de integração, esse algoritmo pode complementar o funcionamento do algoritmo de Berkeley apresentado anteriormente.

4.2. Tecnologias utilizadas

- Go The Go Programming Language[Go]
- Go RPC for Endpoint (Para construção das chamadas em RPC)
- Windows 11 - 21H2 (Para realização e hospedagem do código principal)
- Oracle Virtual Machine (Para criação de máquinas virtuais e configuração de rede interna de computadores)

4.3. Construção do código

Foi utilizado a linguagem de programação Go para construção do algoritmo por dois motivos, o primeiro motivo é pela escassez de conteúdo disponível sobre a implementação do algoritmo em outras linguagem e o segundo motivo é apenas para obter melhor desempenho na construção do código.

Funções utilizadas:

- **Election** - Inicia a eleição a partir de uma solicitação que é solicitada por um nodo de menor ID.
- **InvokeElection** - Invoca a eleição para os nodos superiores, envia seu identificador como parâmetro utilizando RPC.
- **NewCoordinator** - Essa função é chamada pelo novo coordenador para atualizar as informações dos outros nodos.
- **HandleCommunication** - Estabelece a comunicação entre nodos.
- **CommunicateToCordinator** - Realiza a comunicação entre o coordenador e os nodos, caso a comunição com o coordenador falhe, uma eleição é solicitada.
- **makeYourselfCoordinator** - Função utilizada após um nodo decidir que é o coordenador, informa todos os outros nodos da rede e atualiza as informações existentes.
- **random** - Utilizada para capturar uma ação, qualquer tecla apertada no prompt inicia uma comunicação entre nodo cliente e nodo coordenador.
- **main** - Invocada através de um prompt, possui um menu para realizar as tomadas de decisões, incluindo inserir um novo nodo na rede ou até mesmo decidir se um nodo está se comunicando com o coordenador ou se está se recuperando de uma falha.

4.4. Como executar o projeto?

Todo o código encontra-se disponível no Github link:Algorítimo de Bully - Daniel Rodrigues para realizar a execução do projeto é necessário ter instalado o compilador da linguagem Go, encontrado no link: Go Dowloads.

1. Usar o prompt na pasta do projeto e executar o comando ”go build”seguido pelo ”nome do arquivo.go”
2. Executar o arquivo .exe que foi criado após o build, sendo necessário abrir pelo menos 3 arquivos para realizar os testes com maior precisão.
3. Por padrão o nodo de número 5 é o coordenador primário.
4. Por padrão o arquivo possui 5 nodos pré configurados no endereço localhost:3000-3004, sendo possível adicionar novos nodos modificando a variável bully no código, exemplo na imagem abaixo:

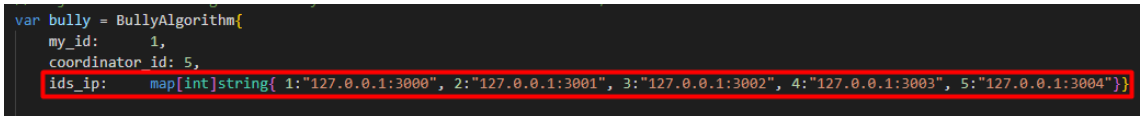

Figura 11. Algoritmo de Bully - Variável de alocação dos nodos

4.5. Resultados obtidos com o Algoritmo de eleição (Bully)

Foi utilizado 3 nodos para realização dos testes práticos com os seguintes endereços:

- Nodo1: 127.0.0.1:3000
- Nodo2: 127.0.0.1:3002
- Nodo3: 127.0.0.1:3003

O primeiro nodo é adicionado na rede com o identificador 1, sabemos que por padrão o nodo de número 5 é o coordenador mas o mesmo não está conectado, então uma nova requisição de eleição é iniciada, em multicast o nodo de identificador 1 envia uma solicitação para todos os outros nodos na rede mas nenhum nodo responde, pois apenas existe o nodo de identificador 1, após isso ele se assume como coordenador da rede. Vejamos o exemplo na imagem abaixo:

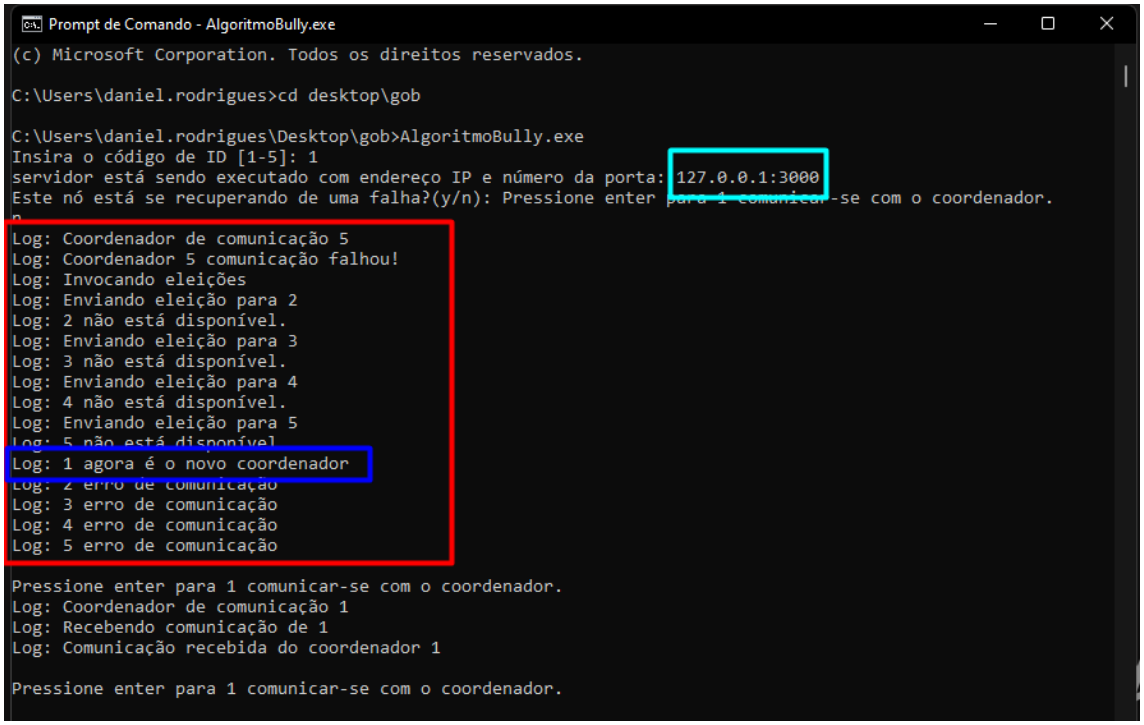

Figura 12. Algoritmo de Bully - Nodo ID = 1 - Entra na rede e inicia eleições

Após isso um novo nodo é adicionado na rede com o identificador 4, sabemos que o algoritmo de bully escolhe o coordenador a partir do maior identificador, sendo assim, uma nova eleição é solicitada pelo Nodo com o identificador 4, novamente uma mensagem é enviada em multicast para todos os outros membros, apenas o nodo com o identificador 1 está conectado e por possuir o identificador menor, o mesmo perde seu lugar de coordenador e o novo nodo com o identificar 4 passa a ser o coordenador da rede.

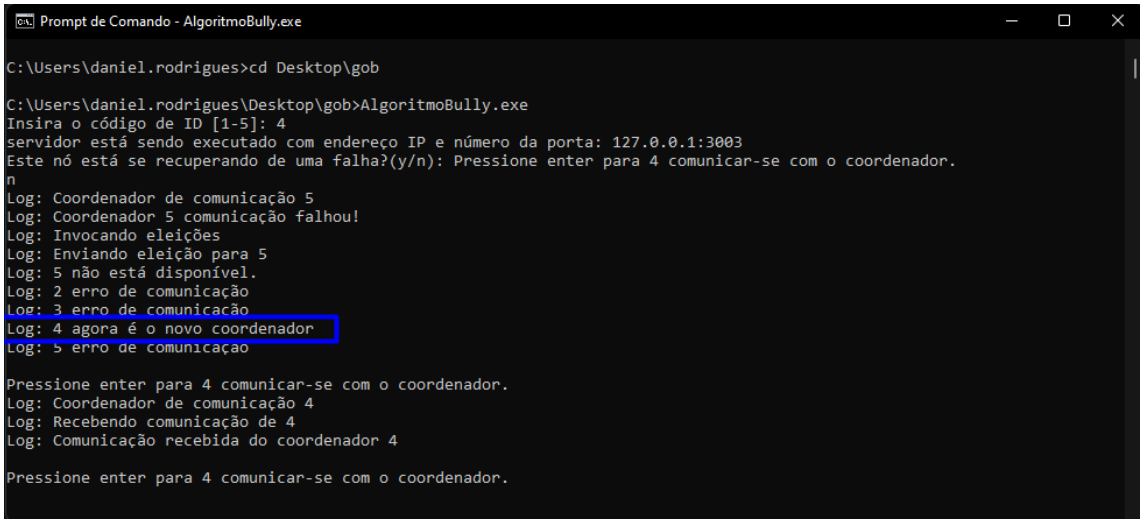

Figura 13. Algoritmo de Bully - Nodo ID = 4 - Entra na rede e assume o coordenador

Agora um novo nodo com o identificador 3, passa a fazer parte da rede e estabelece comunicação normalmente com o nodo coordenador 4 que até o momento está funcionando normalmente.

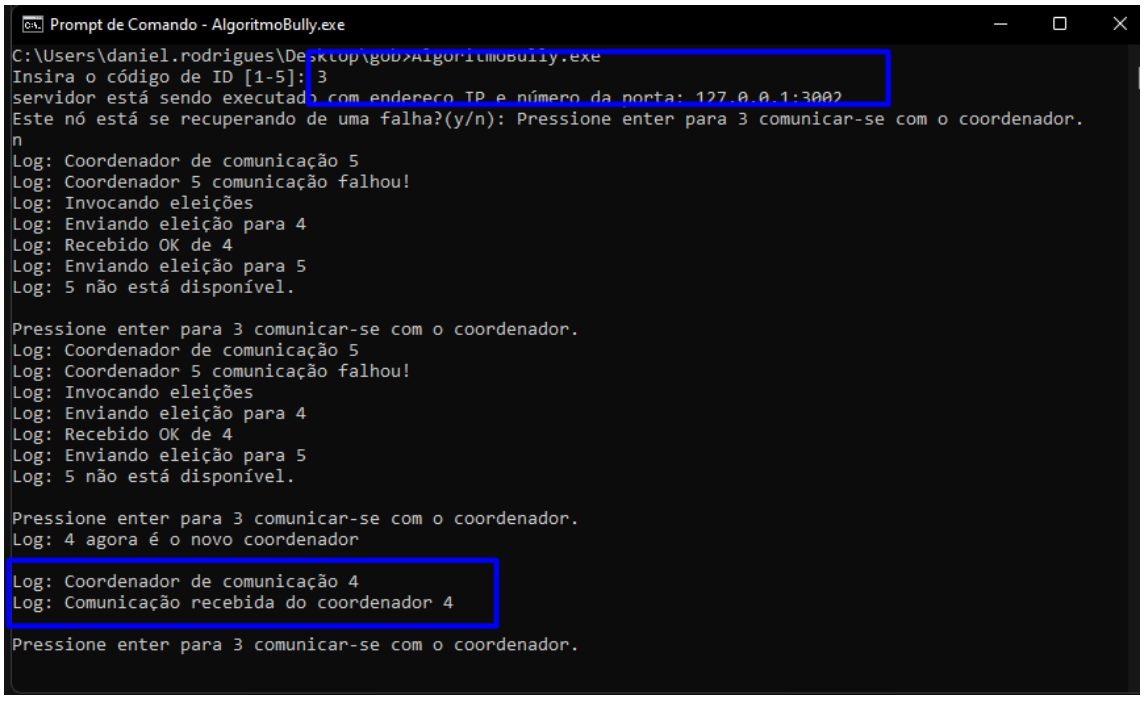

Figura 14. Algoritmo de Bully - Nodo ID = 3 - Entra na rede e realiza a comunicação com o coordenador

Para fins de testes acadêmicos e validação do algoritmo, simulamos a falha do coordenador que possui o identificador 4, a falha foi induzida através da finalização da sua execução, após isso, uma nova eleição é iniciada quando o nodo de identificador 1 tenta se comunicar com o coordenador e não recebe uma resposta, sendo assim o novo

coordenador com o identificado 3 é eleito.

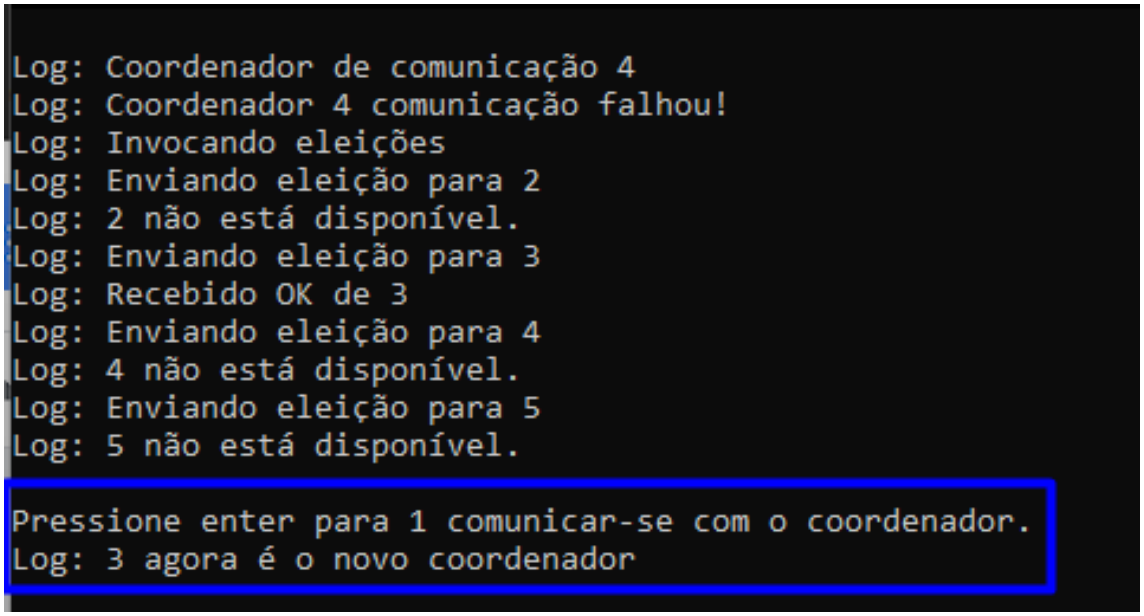

Figura 15. Algoritmo de Bully - Nodo ID = 1 - Solicita uma nova eleição

Novamente, para fins de validação, o nodo com identificador 4, se recupera de sua falha e assume novamente a coordenação.

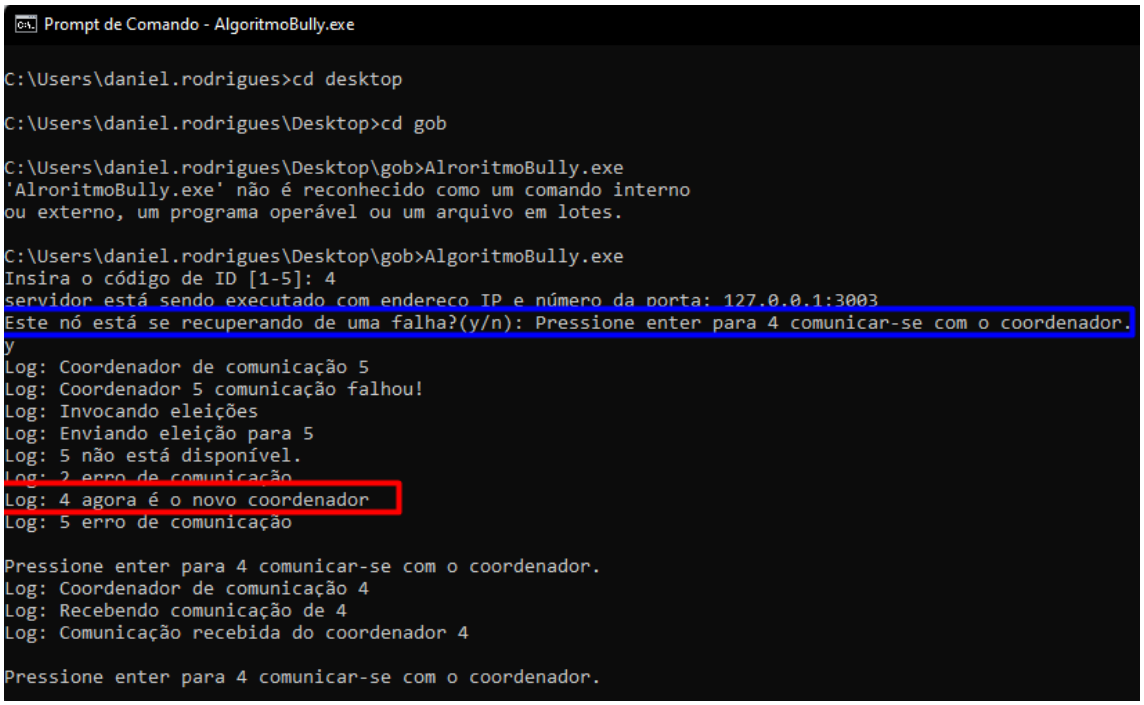

Figura 16. Algoritmo de Bully - Nodo ID = 4 - Recupera-se de uma falha

4.6. Considerações sobre o algoritmo de Bully e sua implementação

O algoritmo funciona perfeitamente e é possível ter uma compreensão clara de seu funcionamento através da realização de testes, sendo uma excelente opção para agregar na implementação do algoritmo de sincronização de relógios apresentado anteriormente. Infelizmente também existe muito pouco conteúdo implementado sobre o algoritmo, sendo todos em linguagem Go e Java, escolhi utilizar a linguagem de programação Go, pois com ela fica mais simples de entender o funcionamento de cada componente.

Referências

[Git a] Algoritmo de Berkeley git. https://github.com/topics/bully-algorithm?q=berkeley. Accessed: 2022-06-12.

[Gee a] Algoritmo de Berkeley sd. https://www.geeksforgeeks.org/berkeleys-algorithm/. Accessed: 2022-06-05.

[Git b] Algoritmo de Bully git. https://github.com/topics/bully-algorithm?l=javascript&o=asc&s=stars. Accessed: 2022-06-11.

[Gee b] Algoritmo de Cristians sd. https://www.geeksforgeeks.org/cristians-algorithm/. Accessed: 2022-06-8.

[Gee c] Algoritmo de Eleicao sd. https://www.geeksforgeeks.org/election-algorithm-and-distributed-processing. Accessed: 2022-06-12.

[Go] Go - Programação language. https://go.dev/. Accessed: 2022-06-07.

[Dev] Java RMI devmedia. https://www.devmedia.com.br/uma-introducao-ao-rmi-em-java/28681. Accessed: 2022-06-10.