Relatório - Raft

Grupo 5
fc51033 Jorge Martins
fc51088 Pedro Silva
fc51101 Lívio Rodrigues

Conteúdo

1. *Principais Componentes*

1.1 Cliente

1.1.1 Client

Classe Cliente que envia em loop infinito requests ao chamar o método request() da classe ClientRaft. Esta classe permite criar várias Threads que fazem multiplos pedidos ao servidor de forma a simular vários clientes

1.1.2 ClientRaft

Classe responsável por se conectar ao líder dos Servidores. Primeiro tenta conectar-se a um servidor aleatório e se esse não for o Leader, ser-lhe-ão enviados os dados do Leader(endereço ip e porto) e de seguida conecta-se ao Leader. Depois de conectado ao Leader, recebe o seu clientID(único) gerado pelo Servidor. Se o líder for abaixo ou mudar, esta mudança é detetada e o cliente tenta-se conectar de novo a um servidor e irá receber o novo líder eventualmente.
Esta Classe vai ainda ser responsável por gerar os IDs das operações(incrementa o id a cada operação realizada).

1.1.3 Command

Classe auxiliar usada para enviar toda a informação para o Líder. Esta Classe contem a string referente ao request enviado pelo Client, o ID do Client em questão e o ID da operação.

1.2 Servidor

1.2.1 Service

Classe recebe o ID do servidor que queremos iniciar e liga esse servidor através da classe ServerRaft. Nesta fase é necessário que o id dado a esta classe seja

válido, i.e, exista no ficheiro "config.properties".

1.2.2 ServerRaft

Classe responsável por aceitar conexões de clientes e servidores, que implementa a lógica do algoritmo Raft. Foi usada uma arquitetura MultiThreading de modo a paralelizar o funcionamento do servidor, como iremos explicar mais à frente.

1.2.3 ServerConnection

Classe responsável por guardar uma conexão entre o servidor atual e outro servidor. Esta classe contem o ID to servidor atual, o ID do servidor da conexão atual, o socket da conexão, o ObjectInputStream e o ObjectOutputStream para enviar e receber objetos pela rede.

1.2.4 StateMachine

Classe que executa os comandos depois do seu LogEntry correspondente ser commited. Esta classe tem um HashMap que guarda a ultima operacão realizada para cada cliente para que, caso esta se perca e seja requested novamente, a state machine não tenha de realizar novamente a computação.

1.2.5 LogEntry

Entry que vai ficar guardada no Log. Contém o termo, o seu index no Log, o seu comando, o id do cliente que fez o request e o id da operação.

1.2.6 Command

Classe auxiliar usada para enviar toda a informação para o Líder. Esta Classe contém a String referente ao request enviado pelo Client, o ID do Client em questão e o ID da operação.

1.2.7 AppendEntries

Objeto que vai ser enviado do Leader para os Followers. Contém, o termo atual do Leader, o ID do Leader, o último index no Log, o último termo no Log, uma lista de LogEntries referente ao pedido e o último index de commit no Leader. No caso de se tratar de um heartbeat, a AppendEntry não vai conter LogEntries.

1.2.8 AppendEntryResponse

Resposta aos AppendEntriesRPC. Esta Classe vai servir como resposta dos Followers para o Leader. Contém apenas o termo atual do follower e se o Append foi realizado com sucesso ou não.

1.2.9 ServerState

Enumerado que identifica o estado do servidor. LEADER quando o servidor é leader, FOLLOWER quando o servidor é follower e CANDIDATE quando o servidor é candidato a eleição.

1.2.10 RequestVote

Classe que representa um pedido de votação. Esta classe contem o termo, o ID do candidato, o ultimo index do log e o ultimo termo do log.

1.2.11 RequestVoteResponse

Classe que representa uma resposta a um pedido de votação. Esta classe contem o termo e se o voto foi concedido.

1.2.12 Election Timer

Para iniciar uma eleição é criado um Timer que irá executar um TimerTask ao fim de um tempo random gerado entre os valores MIN_ELECTION_TIMEOUT e MAX_ELECTION_TIMEOUT. O setup deste Timer é feito dentro do método resetTimer() que também é usado para dar reset a esse Timer sempre que é recebida uma AppendEntry do líder ou um RequestVote de um servidor com um termo igual ou superior ao do que está a receber esse pedido.

1.3 Ficheiros de Configuração

Existem dois ficheiros de configuração, um para os Cliente e outro para os Servidores.

1.3.1 No Cliente

O ficheiro no lado do Cliente contem a informação de todos os servidores disponíveis para conexão. Contem o ID, Ip e porto usado por cada servidor.
Exemplo:
servers=1-127.0.0.1:40000;2-127.0.0.1:40001;3-127.0.0.1:40002;4-127.0.0.1:40003;

1.3.2 No Servidor

O ficheiro no lado do Servidor contem a informação de todos os servidores disponíveis para conexão. Contem o ID, Ip, o porto usado por cada servidor para comunicar com clientes e o porto para comunicar entre servidores.
Exemplo:
servers=1-127.0.0.1:40000:50000;2-127.0.0.1:40001:50001;3-127.0.0.1:40002:50002;

2. *Decisões Arquiteturais*

As principais classes envolvidas na nossa implementação estão representadas no esquema seguinte. Existem duas classes (Client e Service) que servem apenas para executar as aplicações. A implementação do RAFT é feita nas classes ClientRaft e ServerRaft. Temos a classe State Machine que executa as operações a pedido do cliente e, por fim, temos também classes que representam objetos relevantes para o algoritmo RAFT, como a LogEntry, a AppendEntry e a AppendEntryResponse.

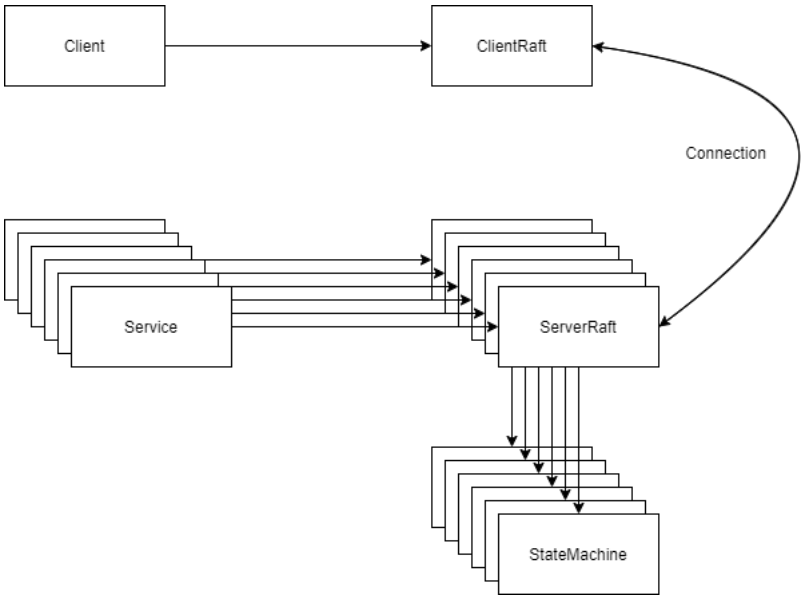

Figura 2.1: Modelo de Classes

2.1 Arquitetura Multi Thread

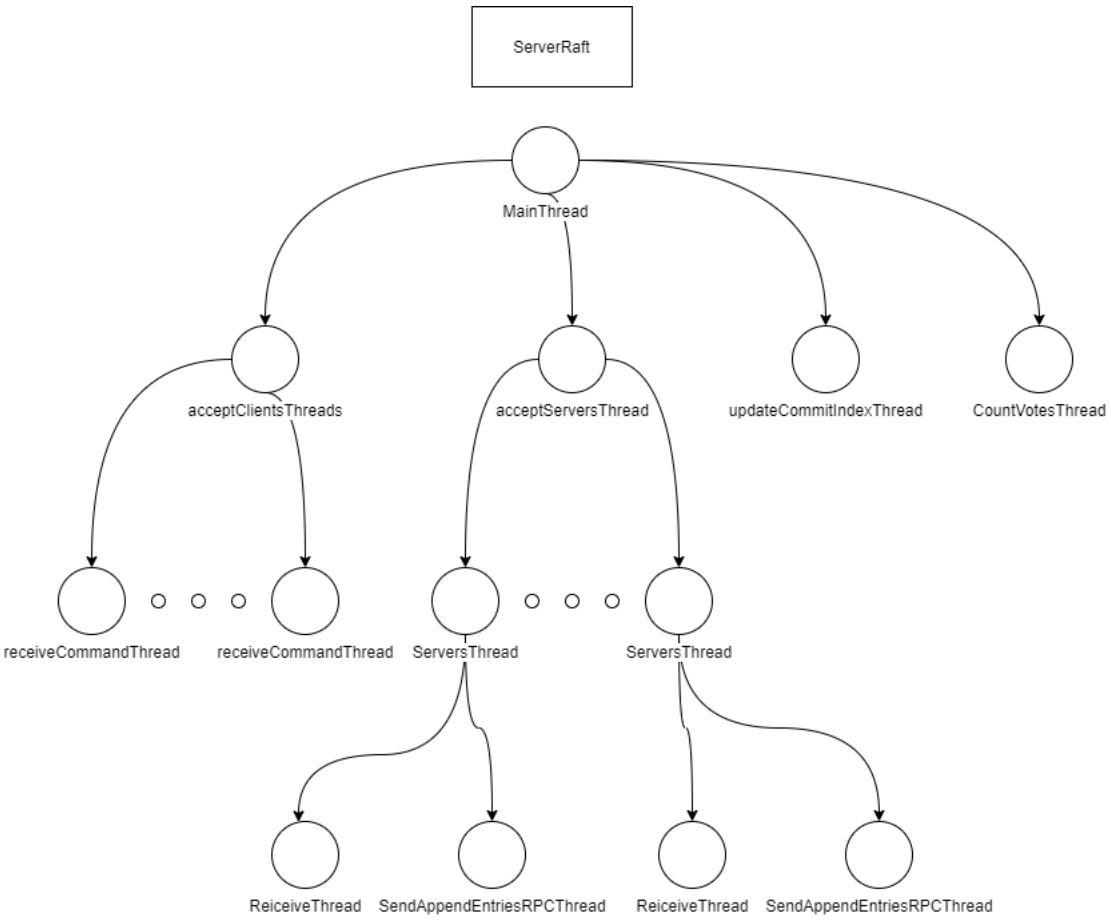

Figura 2.2: Esquema de Threads num Servidor

2.1.1 Aceitar conexões de clientes

Implementámos uma thread específica para aceitar conexões de clientes. Uma vez que uma ligação seja aceite é criada uma thread ReceiveCommandsThread para cada cliente aceite.

2.1.2 ReceiveCommandsThread

Esta thread é responsável por receber os pedidos do cliente. Ao receber um pedido cria e adiciona ao Log uma LogEntry referente ao pedido. A thread

depois espera que o pedido seja executado para responder ao cliente com o valor computado e guardado na State Machine.

2.1.3 Aceitar conexões de servidores

Para que os servidores sejam capazes de comunicar uns com os outros implementámos uma thread que, à semelhança da thread que aceita conexões de clientes, irá aceitar conexões de outros servidores (de notar que na nossa implementação os servidores apenas se ligam aos servidores que tenham um id superior ao deles, de modo a evitar tentativas de ligação bilaterais). Esta thread irá criar uma thread do tipo ServersThread para cada servidor aceite.

2.1.4 ServersThread

Esta Thread independentemente do estado do servidor, vai receber AppendEntries, AppendEntriesResponce, RequestVote e RequestVoteResponce. Para este efeito é criada mais uma Thread que vai ficar responsável apenas por receber estes objetos.
Enquanto a ServerThread estiver ativa se o servidor for Candidate, vai enviar RequestVotes. Se for Leader vai enviar os AppendEntriesRPC. Para isto cria-se mais uma Thread auxiliar, que enquanto o servidor seja Leader vai ao log ver se existem logEntries para enviar e se sim envia. É nesta Thread que também recebemos as respostas a esses appendEntriesRPCs. Se não existirem logEntries para enviar o servidor envia um HeartBeat.

2.1.5 Update ao Commit Index

Por fim temos uma thread que verifica constantemente os commit index dos followers, através da estrutura de dados "matchIndex". Se o número de servidores que têm um commit index superior ao do Leader for maior que metade dos servidores do cluster, então o commit index do Leader irá ser incrementado. Caso o commit index do servidor seja maior que o lastApplied iremos executar o pedido na posição lastApplied do Log.

2.1.6 Contar Votos

Temos uma Thread cuja única função é quando recebemos votos, contar os votos e verificar se já temos a maioria, se sim executa as modificações necessárias ao Servidor. Esta Thread é bloqueada e só desbloqueia quando recebemos um voto.

3. *Instruções de Uso*

Para executar o nosso programa foram entregues dois scripts Client.sh e Service.sh nas pastas Client e Server, respetivamente. Deste modo para executar o cliente basta abrir uma consola na pasta Client e correr o comando *./Client.sh <NumberOfClients>*, sendo NumberOfClients o numero de Cliente que se quer simular. Para executar o servidor, é necessário abrir uma consola na pasta Server e correr o comando *./Service.sh <serverID>*, sendo serverID um dos ids presentes no config.properties.
No caso de estar em Windows, também existem ficheiros bat equivalentes. A forma de executar os bat é semelhante aos sh.
Também é possível importar a pasta Client e Server como projetos separados no Eclipse e correr normalmente. No entanto deve ser tido em conta que a correr o servidor é necessário colocar o id desejado nos argumentos da run configuration.