Algoritmo RAFT e LUARPC

PUC
RIO

Professora: Noemi Rodriguez
Aluno: Fernando Homem da Costa

INF2545 - Sistemas Distribuídos
Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro
12 de Abril de 2021

Conteúdo

1 Introdução

Este trabalho é a terceira tarefa da disciplina de Sistemas Distribuídos (INF2545) do primeiro semestre de 2021, ministrada pela professora Noemi Rodriguez. Nesta tarefa, deve-se implementar em Lua a parte de eleição de líder do algoritmo de consenso Raft. A implementação deve ser baseada na biblioteca LUARPC (Marcelo Costalonga).

1.1 Instruções

A execução de programa deve ser feita em duas etapas: colocar em funcionamento todas as instâncias de servidores necessárias e todos clientes. Os servidores devem ser inicializados antes dos clientes, a fim de registrar as atividades nos *logs*.

1.1.1 Arquivo de Configuração

Todas as configurações necessárias para execução dos servidores estão presentes no arquivo de configuração. O Exemplo 1 ilustra um cenário de teste de três instâncias.

- **Instances** : Números de instâncias de servidores;
- **Servers** : Lista de servidores, contendo o identificador do servidor, internet protocol (IP) e porta;
- **Interface** : Caminho do arquivo de interface.

```
{
    "Instances" : 3,
    "Servers": [
        ["HASH1","127.0.0.1", "4444"],
        ["HASH2","127.0.0.1", "4445"],
        ["HASH3","127.0.0.1", "4446"]
    ],
    "Interface" : "interface.lua"
}
```

Listing 1: Exemplo de três instâncias.

1.1.2 Servidores

Para inicializar o servidor, execute a seguinte instrução na linha de comando/terminal:

```
lua serverraft.lua [HASHID]
```

Obs: o comando deverá ser executado para cada instância presente no arquivo de configuração. O **HASHID** é um parâmetro obrigatório e deve respeitar o arquivo de configuração, conforme o Exemplo 1.

1.1.3 Clientes

Para executar um cliente, basta inserir na linha de comando/terminal:

```
lua clientrpc.lua [IP] [PORT]
```

Obs: o comando deverá ser executado para cada instância a ser inicializada. Os parâmetros **IP** e **PORT** são obrigatório, e devem respeitar as informações do arquivo de configuração, conforme o Exemplo 1.

2 Desenvolvimento

O desenvolvimento desse projeto é baseado no trabalho *"In search of an understandable consensus algorithm"* [1]. Todas as situações apresentadas no trabalho estão presentes nessa solução proposta. Vale a pena destacar que foi utilizado o simulador do algoritmo Raft [2], a fim de verificar o comportamento da solução.

Outro ponto a destacado é que o esquema de votos não é feito a todas as instâncias de replicas, mas até alguma réplica informou que ganhou a maioria dos votos.

Cada instância de uma réplica possui uma variável que responsável por armazenar o valor do tempo de espera máximo de um *"heartbeat"*. Existem em algumas situações em que essa variável é atualizada toda vez que: uma instância receber um voto de alguma réplica com um *term* maior o que o seu; uma instância receber um *heartbeat* de alguma réplica que possua um *term* maior ou igual que o seu; uma instância mudar o seu estado de candidato/líder para seguidor; finalizar um eleição, evitando assim o cada de um empate.

3 Testes

Nessa seção executaremos um teste contendo três instâncias, mas esse número pode ser estendido, apenas alterando o arquivo de configuração e o *script* bash para clientes. A fim de melhor verificar o funcionamento do programa, as ações executadas serão registradas em arquivos, ao invés de estarem disponível na tela.

A Figura 1 ilustra o momento em que os servidores são inicializados para teste.

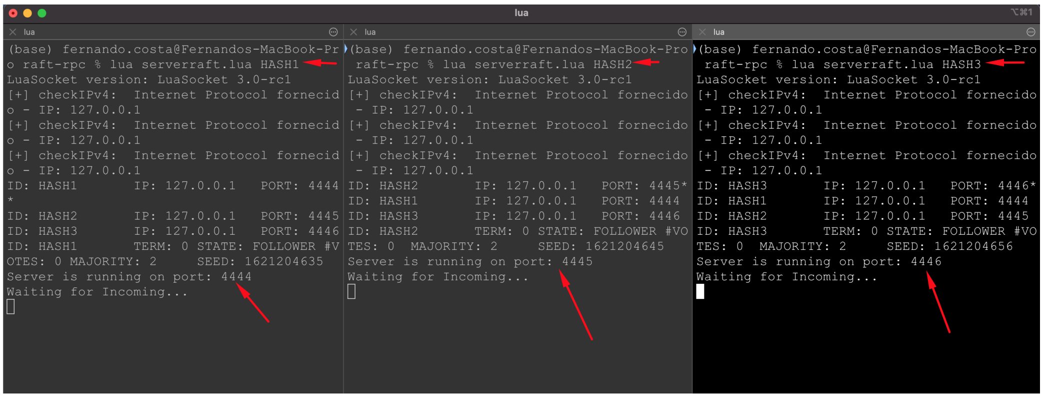

Figura 1: Inicializando Três Instâncias de Servidor

Na Figura 2 temos o momento em que os clientes são inicializados e ações começam a serem registradas nos servidores.

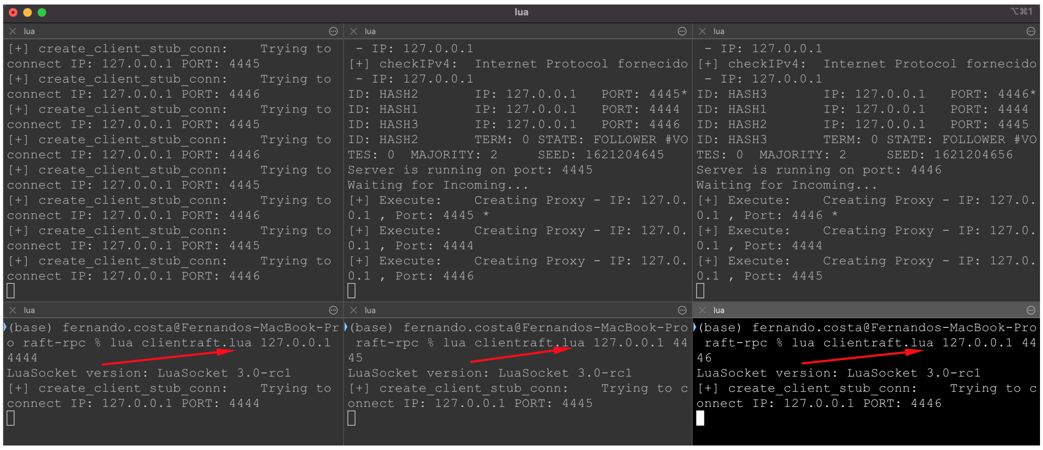

Figura 2: Inicializando Três Instâncias de Cliente

Nesse caso de teste, a primeira instância a se tornar Líder foi a **HASH3**, conforme mostra a Figura 3.

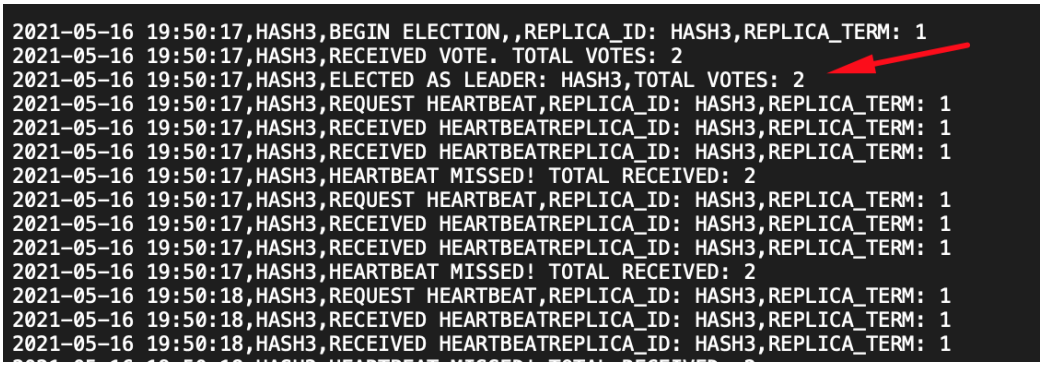

Figura 3: HASH3 é eleito Líder

Para simular uma falha de uma das instância, é necessário finalizar o processo na linha de comando/terminal por meio de **"Ctrl + C"**. Na Figura 4, o *log* mostra que o HASH3 e é removido. Após isso, surge uma nova eleição com *TERM* igual a três, que tem como resultado um empate. Por fim, uma novamente uma eleição é feita e o HASH1 é eleito líder.

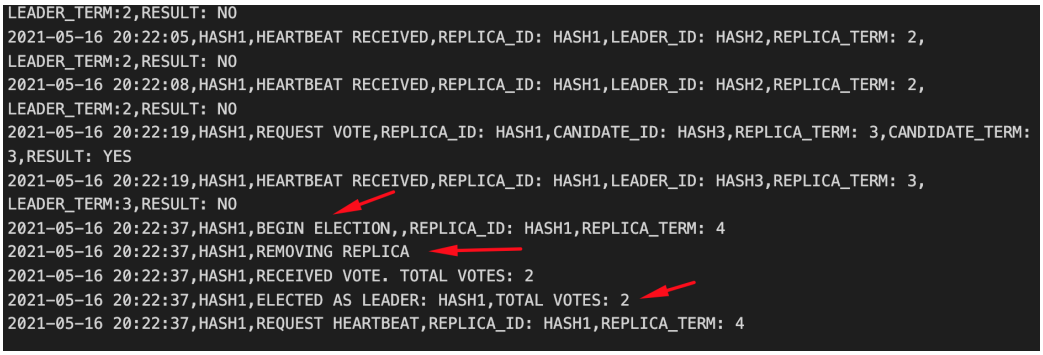

Figura 4: HASH3 falha, causa empate e HASH1 é eleito

4 Conclusão

Durante o projeto foram encontradas algumas dificuldades, tais como: entendimento do código fornecido, *"debuggar"* o código para encontrar os erros e verificar o funcionamento correto da solução da proposta.

Para evitar modificações no código que pudessem impactar o funcionamento da biblioteca luarpc, foi criado um arquivo de configuração que contém a estrutura de execução do servidor. Em relação ao processo de *"debuggar"*, utilizei a biblioteca Penlight [3], permitindo visualizar o conteúdo de uma tabela e contabilizar o número de parâmetros dentro da tabela.

Como esse trabalho utiliza bibliotecas desenvolvidas por outros autores, o mesmo não está disponibilizado no GitHub. Apenas enviado por EAD da PUC-Rio.

Referências

[1] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, page 305–320, USA, 2014. USENIX Association.

[2] Diego Ongaro. The raft consensus algorithm, 2015.

[3] Lunar Modules. Penlight lua libraries, 2018.