UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

RELATÓRIO TRABALHO PRÁTICO - BACKUP PRIMÁRIO

Marcos Silva Laydner
Sadi Júnior Domingos Jacinto

Professora orientadora: Patrícia Della Méa Plentz

Florianópolis
2020

1 INTRODUÇÃO

Para o trabalho, foi feita a implementação simples de um sistema de replicação passiva (backup primário), seguindo as cinco fases de comunicação entre *front-end* e gerenciadores de réplica.

2 IMPLEMENTAÇÃO

O projeto foi feito usando a linguagem Python, usando *sockets* para a comunicação entre os processos. Ao todo, existem três processos, cada qual contido em seu respectivo *script* .py, que simulam os atores do sistema:

- **client.py:** Uma representação de um cliente. Possui uma interface de usuário em modo texto, com a possibilidade de receber *inputs* do usuário, além de também ser possível receber parâmetros da linha de comando. Usado para estabelecer a comunicação com o *front-end*. Facilmente subistituído por uma interface gráfica ou *website*. Por fim, é possível rodar os testes unitários, para checar a saúde do programa (recomendado quando for rodado pela primeira vez numa máquina), e abrir um menu com mais detalhes sobre cada opção.

```
-> % python client.py
Tentando iniciar conexão com o front-end no host 127.0.0.1 e porta 8889
Conexão com o front-end estabelecida com sucesso!

|----------------------------------------|
        Menu:

        [u] - Fazer upload de arquivo
        [d] - Deletar arquivo
        [h] - Histórico
        [r] - Rodar Testes Unitários
        [t] - Menu Detalhado
        [s] - Sair
|----------------------------------------|
```

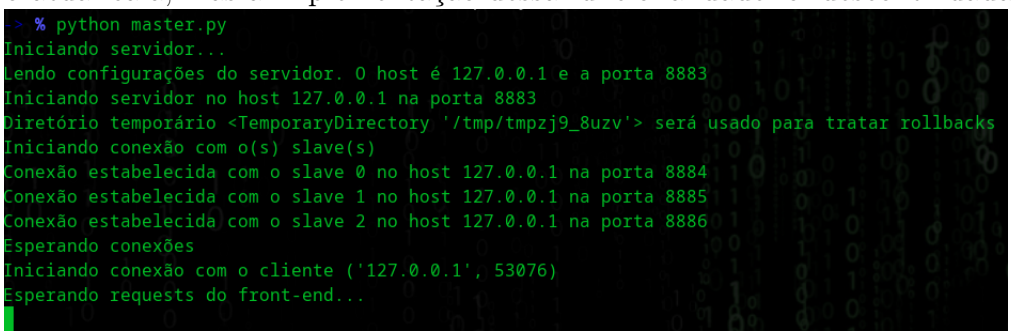
- **front_end.py:**
  Simula o *front-end*, realiza a comunicação do cliente com o servidor principal. Para isso, tal processo se conecta ao servidor principal[1] através de *sockets* e das configurações lidas no arquivo *ips.conf*. O *front-end* permite fazer requisições de envio (*upload*), atualização (*update*) e deleção (*delete*) de arquivos para o servidor principal (*master.py*). Além disso, é possível pedir para visualizar requisições já feitas.

```
-> % python front-end.py
Iniciando servidor...
Lendo configurações do servidor. O host é 127.0.0.1 e a porta 8889
Iniciando servidor no host 127.0.0.1 na porta 8889
Tentando iniciar conexão com o master no host 127.0.0.1 e porta 8883
Conexão com o master estabelecida com sucesso!
Esperando conexões
Iniciando conexão com o cliente ('127.0.0.1', 54570)
Esperando requests do client...
```

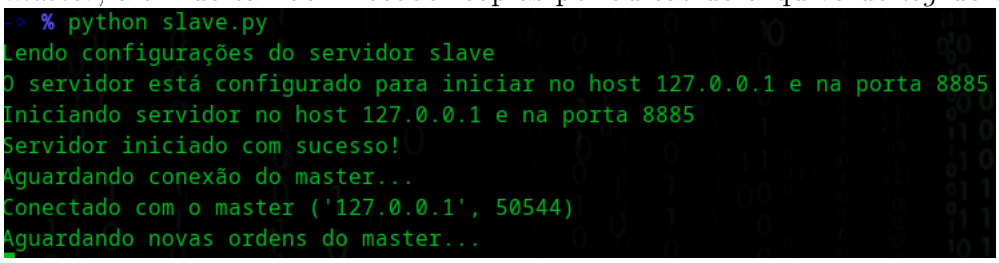
- **master.py:**
  Simula o gerenciador de réplica primário. O *master* é configurado por meio de

[1] A partir desse ponto, o servidor principal será chamado de servidor *master* ou apenas *master*

um arquivo de configuração chamado *ips.conf*. Ele busca tenta conectar-se aos servidores de backup (*slaves.py*) antes de começar a receber conexões de clientes. Caso ele apenas consiga se conectar com menos da metade dos backups esperados, ele cancela o processo e desliga. Caso isso não aconteça, o servidor inicia, e aguarda requisição dos clientes, sendo que apenas um cliente pode executar uma operação por vez. As requisições recebidas (*update*, *upload*, *delete*) são executadas no *master* e mandadas para os backups, para que eles as executem também. O resultado da operação é mandado de volta para o cliente que fez a requisição. Caso a operação resulte em erro, é feito um *rollback* para o estado anterior ao da operação, seguido de uma ordem para os *slaves* de os mesmos fazerem também um *rollback*. Finalmente, a cada 10 requisições executadas, o *master* envia para os *slaves* seu arquivo de *log*, isso foi feito visando implementar um algoritmo de eleição de um novo master, caso o atual caia, mas a implementação dessa funcionalidade foi descontinuada.



- **slave.py:**
  Simula um gerenciador de réplica secundário. Ele roda como um servidor, seguindo as especificações de um arquivo de configuração personalizado (*slave.conf*), e espera a conexão do *master*. O *slave* recebe requisições do *master*, as executa, e retorna seu resultado para o *master*. Também pode fazer *rollback*, caso receba tal ordem do *master*, além de também receber cópias periódicas do arquivo de *log* do *master*.



- **test.py:**
  Executa testes automatizados. Para isso, precisa que todos os serviços (*slaves*, *master* e *front-end*) estejam de pé. Para facilitar tal execução, no diretório de *tests* existem os arquivos necessários, já separados em diretórios e já configurados.

3 INSTRUÇÕES DE USO

O projeto foi feito pensando em executá-lo em máquinas diferentes, mas, também é possível rodar a aplicação em uma única máquina, desde que cada processo esteja em um diretório diferente, e os processos que servem como servidores de backup utilizem portas disponíveis e não repetidas.

Independente de como será feita a execução da aplicação, localmente ou remotamente, é importante ter em mente que os arquivos de configuração de cada um dos processos desse ser editado, para satisfazer a configuração real na qual a aplicação irá rodar[2].

4 FASES DE COMUNICAÇÃO

1. **Requisição:** O usuário, comunicando-se com o *front-end* através da interface do *client*, escolhe o comando que deseja executar, digitando a letra do comando e o nome ou o caminho do arquivo. Quando a requisição chega no *front-end*, o mesmo a trata de acordo com o tipo de requisição. De forma geral, requisições que envolvem chamadas ao *master* são precedidas por uma requisição especial, *get_last_id* que pede ao *master* o *id* da última requisição, para que esta nova operação tenha seu próprio *id* único. Assim que o *id* é recebido, a requisição do *front-end* é enviada para o *master* com um novo *id*[3] único.

2. **Coordenação:** O *master*, após ter enviado o último *id*, recebe a requisição do *front-end*. Então, começa a executar localmente tal requisição, após ter finalizado seu processo local, o *master* então envia a requisição para os gerenciadores secundários (*slaves*).

3. **Execução:** Os os gerenciadores secundários recebem as ordens do *master* e as executam, enviando o resultado da execução, seja ele um sucesso ou falha, para o *master*.

4. **Acordo:** O *master*, recebendo os resultados dos *slaves*, e tendo tido sucesso em sua própria execução, decide se a operação no geral foi um sucesso ou não da seguinte forma: Se pelo menos a maioria dos servidores (mais da metade), responderem com falha, a operação falha, se responderem com sucesso, é um sucesso.

5. **Resposta:** O *master* envia ao *front-end* o resultado da operação baseada no acordo, e este envia o resultado ao *client*.

[2]Isso quer dizer, editar os arquivos de configuração para indicar corretamente os *hosts* utilizados e as portas nas quais os servidores de *backup* serão inicializados.

[3]id = get_last_id + 1