Projeto Final - Sistemas Distribuídos

Leonardo P. Oliveira[1]

[1] Centro de Matemática, Computação e Cognição - Universidade Federal do ABC (UFABC)
Santo André - SP – Brasil

oliveira.l@aluno.ufabc.edu.br

Disciplina:	Sistemas Distribuídos
Turma:	A2
Professor:	Rodrigo Izidoro Tinini
RA:	11201920744

Resumo. *Documentação para projeto que consiste em um sistema de Backup de arquivos o qual deve conter um ou mais clientes, um middleware e alguns servidores (nesse caso 4).*

Sumário

1. Introdução

O sistema de backup proposto teve como objetivo verificar e melhorar a qualidade de aplicação com ferramentas já vistos na disciplina de rede de computadores e aprimorados na disciplina de Sistemas Distrubuídos, como sockets, threads, comunicação Cliente Servidor, transporte UDP, protocolo de rede IP, Transparência (nesse caso principalmente a de distribuição) e afins. A idéia era fazer um sistema capaz de enviar arquivos (independentemente do tamanho), esse arquivo seria salvo em um servidor e depois disso, mesmo que o cliente desconecte da rede, será feito um backup de forma totalmente transparente ao usuário.

2. Arquitetura do Sistema

2.1. Componentes

- **Clientes**: Ao iniciar o app do cliente ele vai ter acesso a um menu "animado" o qual dá a ele 5 opções, sendo a primeira "Backup de arquivo", a segunda "Listar arquivos disponíveis", a terceira "Status das transferências", a quarta "Limpar" e a última "Sair". Cada nome já diz bastante sobre o que faz cada operação, mas basicamente, ao escolher "fazer backup", é criado um novem menu, onde ele deve digitar o nome do arquivo junto a extensão, "Listar Arquivos Disponívels", irá mostrar todos os arquivos na pasta raiz do cliente, "Status das Transferências" da ao cliente a possibilidade de verificar quais arquivos já foram salvos e quanto falta para alguns terminarem além de ver quando foi iniciada a transferência de alguns, "Limpar" limpa a tela, mostrando ao cliente o menu inicial novamente e "Sair" ele vai ter a opção de escolher se deseja realmente parar as transferências caso alguma esteja em andamento (do cliente para o servidor é claro, ele não tem visiblidade de transferências entre os servidores), o arquivo de configuração (.json) do cliente, só dá ao mesmo a visão do Middleware (via código e não log).

- **Middleware**: O Middleware tem um papel extremamente importante nesse sistema, o mesmo tem acesso a um arquivo de configuração o qual lista o endereço e diretório raiz, dos servidores e dele mesmo, ele é responsável por devolver ao cliente qual o endereço do servidor que o mesmo irá acessar, e ao servidor que será acessado, qual o endereço do servidor de backup, ele faz isso, considerando o uso de CPU e de Memória em cada um dos servidores e também (caso esses dois não estejam numa diferença tão grande) pela quantidade de threads ativas - facilitando assim também o teste caso seja feito a partir de um só computador.

- **Servidores**: Cada um dos servidores tem acesso ao endereço de si mesmo e do middleware, eles fazem o envio de "status" ao middleware cada vez que o sistema é acionado (requisição inicial do cliente), enviando como está a carga de CPU de Memória, e também a quantidade de threads ativas no momento, o mesmo diferencia se um arquivo é de backup ou de salvamento inicial, em caso de backup a thread é fechada assim que terminar a comunição enviando uma confirmação.

2.2. Fluxo de Dados

O fluxo em geral é o seguinte, o cliente envia uma requisição que chega o middleware, o middleware recebe essa requisição e devolve uma chave valor contendo a informação

apenas do servidor ao qual ele recomenda que o arquivo seja salvo, nesse mesmo momento o middleware também faz uma requisição ao servidor passando o endereço para o servidor de backup. O cliente pega o endereço e começa a fazer o envio do arquivo, mas nesse momento, por estar usando threading, ele pode fazer também outras requisições, quando um arquivo termina de ser salvo no servidor, independentemente de ter terminado a transação entre o servidor e o servidor de backup, o cliente recebe um aviso de arquivo salvo, e mesmo que ele deseje sair, essa transação de backup vai até o final finalizando então as demais threads. No servidor o arquivo pega o addr do cliente, e então salva junto ao nome do arquvivo, ao terminar de ser salvo o servidor faz a chamada para o servidor de backup e faz o envio do arquivo de backup (contendo o mesmo nome) e adicionando uma flag "isBackup" no nome do arquivo, lá no servidor de backup é retirada a flag, salvo o arquivo e então enviada uma confirmação para o servidor "primário" fechar a conexão, a idéia de pegar o addr é ter um registro relacionado ao cliente em casos de implementações futuras de download, pois se tiver mais de um cliente chamando o middleware ao mesmo tempo, e nesses clientes tiver arquivos diferentes que contenham o mesmo nome, sem esse identificador teriamos problemas.

3. Heurística de Seleção de Servidores

3.1. Critérios de Seleção

A seleção dos servidores primário e de backup é feita da seguinte forma, tem um método que pega status de cpu, memória e threads ativas do servidor, então são comparados por ordem de importância inicialmente carga de cpu e se der empate, carga de memória, caso os valores sejam muito próximos, tem uma pequena margem de erro, que chamei de "withinMargin" a qual se estiver na margem, irá selecionar os arquivos com menos threads ativas. Vale lembrar que também adicionei um caso de servidores validos apenas, ou seja, se tentar buscar o status, e der erro por algum motivo (muitas vezes ocorre no Windows - sim o método funciona tanto para Windows quanto linux) ele não adiciona esse servidor na lista de servidores válidos selecionáveis, mas caso menos de 2 servidores válidos estejam na lista, então é chamada novamente a função de buscar o status.

3.2. Algoritmo

O algorítmo utilizado para seleção de servidores previamente detalhado, faz o seguinte, menos de 2 servidores iniciados inicialmente, retornaria já erro de servidores insuficientes, se tem mais do que 2, vai validar o status, se mais do que retornarem válidos então é efetuada a seguinte lógica, se a margem de diferença de uso de cpu for até 3.0 e de memória 1.0 (margem: se a diferença for igual ou menos, essa margem é calculada entre os 2 melhores (menor uso de cpu e memória)), então vai validar qual tem menos threads ativas, feito isso, o "bestServer" é escolhido como primário, ou o que retornará ao cliente, o segundo seria o segundo melhor considerando as mesmas variáveis, apenas diferindo por "backup" diferente de "primary", esse de backup é enviado ao server.

4. Implementação

4.1. Estrutura de Diretórios

```
Raiz/
        Client/
```

```
        Client.py
        clientConfig.json
Servidores/
        Servidor1/
                ServidorOne.py
                configServerOne.json
        Servidor2/
                ServidorTwo.py
                configServerTwo.json
        Servidor3/
                ServidorThree.py
                configServerThree.json
        Servidor4/
             ServidorFour.py
             configServerFour.json
Middleware/
        SimpleMiddleware.py
        systemConfig.json
StartBackupSystem.py
```

Cada arquivo tem seu config, pois foi considerado que o diretório raiz de cada um seria como se fosse um computador diferente, logo eles acessam somente o necessário para funcionar como sistema, tendo o middleware intermediando em geral, foi criado também o arquivo "StartBackupSystem" para iniciar todos os componentes do servidor de uma vez.

4.2. Configurações

Os arquivos de configuração como previamente citados, foram separados para que cada parte do sistema seja "independente" dependendo de saber de si mesmo e do middleware, a config do cliente contém o endereço do middleware (host e porta), a do servidor vai conter o próprio endereço e seu diretório raiz, além também do endereço do middleware, já a config do middleware vai conter o endereço de cada um dos servidores, caso o mesmo suba em um pc diferente, só teria que alterar ali a configuração do mesmo passando o ip específico e verificar também se as configurações de firewall não vão acabar bloqueando essa comunicação, e também alterar na config do próprio servidor o cliente não precisaria mudar nada quanto a isso.

5. Execução do Sistema

5.1. Instruções de Instalação

Necessário ter o python (python 3) disponível no Windows ou no Ubuntu, cmd no Windows e xterm no Ubuntu, e algumas libs as quais já serão importadas ao projeto quando clonar como socket, Thread, Event, Lock, os, json, signal, sys, random, zlib, select, sys, signal, subprocess e platorm, time.

- **socket**: fornece acesso à interface de rede de baixo nível.
- **Thread**: usada para criar e gerenciar threads em Python.
- **Event**: sincronização de threads usando eventos sinalizadores.

- **Lock**: permite sincronização entre threads, garantindo que uma seção crítica seja executada por uma thread por vez.
- **os**: fornece uma forma de interagir com o sistema operacional, como manipulação de arquivos e diretórios.
- **json**: facilita a manipulação de dados no formato JSON.
- **signal**: usada para lidar com sinais do sistema, como interrupções de teclado.
- **sys**: permite acessar variáveis e funções específicas do sistema.
- **random**: gera números pseudoaleatórios.
- **zlib**: fornece funções para compressão e descompressão de dados.
- **select**: monitora múltiplos canais de I/O, como sockets, para saber quando estão prontos para leitura ou escrita.
- **subprocess**: permite a execução de comandos do sistema operacional como subprocessos.
- **platform**: fornece informações sobre o sistema operacional e a arquitetura do sistema.
- **time**: fornece funções para manipulação de tempo, como atrasos e medição de intervalos.

5.2. Instruções de Execução

Para executar os scripts em novas janelas de terminal no Ubuntu, o `xterm` é necessário. Siga os passos abaixo para instalá-lo:

1. **Atualize a lista de pacotes**:
   ```
   sudo apt-get update
   ```

2. **Instale o `xterm`:
   ```
   sudo apt-get install xterm
   ```

3. **Verifique a instalação**:
   Execute o comando abaixo para abrir uma nova janela de `xterm` e verificar se a instalação foi bem-sucedida:
   ```
   xterm
   ```

   Se a janela abrir corretamente, a instalação está concluída.
4. **Uso do `xterm`:
   Para executar um script Python em uma nova janela do `xterm`, utilize o seguinte comando:
   ```
   python3 StartBackupSystem.py
   ```

   O terminal `xterm` permanecerá aberto após a execução do script. Caso tenha algum erro, pode também executar um arquivo por vez ou executar via weblogic.

   No caso de Windows, basta rodar no cmd
   ```
   python StartBackupSystem.py
   ```

5.3. Exemplo de Uso

O cliente abre o sistema, seleciona a opção “1” Backup de arquivo, o cliente então digita o nome do arquivo junto de sua extensão, o mesmo pode a partir de momento selecionar a opção “3” para verificar o status de transferência, quando o arquivo é salvo no servidor principal, o cliente é notificado e então pode sair (caso queira, opção “5”) que as transferências vão seguir, o arquivo é salvo no servidor de backup e as threads são encerradas.

6. Desafios e Soluções

6.1. Problemas Enfrentados

Grande parte dos meus desafios vieram de conceitos que não compreendia como tranparência, além da confusão que eu mesmo causei ao ler o enunciado do projeto, onde acabei complicando bem mais do que de fato era complicado, tive muito problema para tratar também as threads e é claro, tem o fato de não conhecer muito dos métodos do python, para realizar algumas tarefas que poderiam ter sido mais simples com um certo conhecimento prévio, e acabei tendo que tirar muitas dúvidas também depois das aulas e via e-mail quanto a arquitetura do projeto. Quanto ao problema de threading tive como experiência muitos loops infinitos causados por conflitos de thread e em alguns momentos quando o servidor recebia a confirmação e acabava tratando como uma nova requisição, acabei para isso criando diversos “locks” pelo código, onde posso chamar e alterar o valor de algumas variáveis para diferir o comportamente no cenário de save inicial e de backup, quanto a transparência e a arquitetura, acabei compreendendo melhor com as dúvidas que tirei no dia a dia, só alguns pontos ficaram em aberto considerando quantas possibilidade esse projeto pode trazer, dentre elas a opção de download, para fazer com que múltiplos clientes acessem os servidores, tive a ideia inicial de simplesmente alterar o nome do arquivo usando o addr que cada cliente passa ao enviar o arquivo para download, isso fica apenas no servidor a ideia é no caso de implementar métodos para download/exibição, retirar esse prefixo e manter apenas o nome que o cliente conhece, mantendo assim a transparência. Com relação a falta de conhecimento em python, esse projeto ajudou e muito para conhecer alguns métodos que facilitam e muito o uso da linguagem.

6.2. Melhorias Futuras

Incluir o caso de backup, incluir retirada de prefixo do nome para que o cliente não perceba que ele foi alterado, tornar o sistema mais receptível (um front-end), tornar o sistema mais ágil para arquivos ainda maiores.

7. Conclusão

O projeto atende a tudo o que foi solicitado, com alguns extras como um menu com algumas opções a mais e a opção de enviar arquivos simultaneamente, utilizando conhecimento da camda de transporte de TCP e de rede IP, além de um algumas libs de python como sockets, threadings, entre outras, a comunicação entre o cliente e o sistema é totalmente transparente, principalmente o ponto solicitado que seria a transparência de distribuição, o cliente tem apenas a ciência que envia o arquivo a um servidor, mas o mesmo não sabe sobre o processo de backup entre os servidores e nem precisa saber inicialmente o endereço de nenhum, nenhuma lib que automatiza algo que foi solicitado foi usada, todo o processo, desde o código do middleware, quanto do servidor, como do cliente e

suas respectivas lógicas foram feitos do zero. Assim o projeto tem algumas melhorias futuras e pendências, antes citadas, porém para o escopo do que foi solicitado, ele atende, além também de ter ajudado no entendimento dos conceitos da disciplina.