Aluno: Ícaro Pires de Souza Aragão **Matrícula:** 15/0129815
Aluno: Matheus de Sousa Bernardo **Matrícula:** 14/0028340
Disciplina: Programação para Sistemas Paralelos e Distribuídos
Professor: Fernando William Cruz
Tema: Bag of tasks utilizando REST e SOAP

OBJETIVOS

O objetivo deste projeto é construir uma aplicação que permita que processos executando em diferentes máquinas possam realizar partes de uma tarefa em paralelo. A idéia geral é que um processo mestre coloque sub-tarefas de uma computação em um repositório de tarefas (task bag) e processos escravos selecionem tarefas do repositório para executá-las, retornando os resultados para o mesmo repositório. O mestre, então, coleta os resultados e os combina para produzir o resultado final.

E aproveitando desse contexto para se familiarizar com as arquiteturas SOAP e REST, suas diferenças e vantagens na construção de *webservices*, além de explorar um o problema da concorrência.

SOAP

DESCRIÇÃO DOS ELEMENTOS

O repositório de tarefas possui três componentes principais, o Master, Slave e o Repository.

O repository foi criado utilizando Spring SOAP, esse repository tem três endpoints *pairIn, pairOut, readPair*. o *pairIn* insere um par dentro do repositório, já o *pairOut* tira um par e retorna o valor associado a chave e o *readPair* ler o valor associado a chave sem apagar do repositório. Esses endpoints utilizam as classes de domínio gerados pelo Spring SOAP.

O Master é um cliente criado utilizando Spring SOAP, ele possui a capacidade de chamar os endpoints do repositório. Ele é responsável por inserir os dados das matrizes e também as tasks a serem executadas. No fim ele mostra o resultado da multiplicação para o usuário.

O Slave possui capacidade de fazer uma multiplicação de linha por coluna, ele pega os dados referentes a linha e a coluna daquela task e então faz a multiplicação, depois ele retorna esses dados ao repositório.

DESCRIÇÃO DA EXECUÇÃO

O processo para fazer uma multiplicação envolve iniciar o repositório de tarefa nesse ponto o XML Schema é lido e é gerado um WSDL então as classes de domínio são geradas e então é exposto os endpoints na classe PairEndpoint. Existe três endpoints já previamente citados.

Depois que o repositório estiver rodando é necessário iniciar os escravos em processos separados, esses slaves possuem um processamento simples eles ficam fazendo chamadas pairOut com a chave *nexttask* se não houver esse par no repositório a exceção é ignorada e ele continua a fazer as chamadas. Quando há esse par no repositório ele lê o valor associado e busca no repositório a linha e a coluna necessárias para o cálculo, então ele calcula o resultado e devolve para o repositório por meio de uma chamada *pairIn.*

Aqui é importante dizer que os slaves não conseguem executar um pairOut ao mesmo tempo pois foi usado a *syncronized* no momento da retirada de um pair então essa condição de corrida é tratada.

O Master então pode ser executado, o papel dele é receber do usuário as matrizes que querem ser multiplicadas e depois quebrar essas matrizes em pequenas tarefas a serem enviadas ao repositório. O master então fica enviando requisições para o repositório a fim de receber o resultado calculado, quando todo o resultado estiver no Master ele imprime a matriz resultado.

DESCRIÇÃO TECNOLOGIA UTILIZADA

Spring SOAP e JAXB, o spring é um framework java, nele podemos utilizar annotations para expor os métodos referentes ao framework, ele permitiu junto com o JAXB criar classes domínios rapidamente por meio de um XML Schema. O SOAP trouxe algumas vantagens e outras desvantagens, como vantagem o podemos citar a facilidade de gerar clientes pois o WSDL descreve a interface de comunicação entre o serviço e o cliente, também é interessante o Schema como documentação, por outro lado é relativamente verboso o uso de XML e o suporte a SOAP é cada vez menor.

GRAU DE TOLERÂNCIA A FALHAS

O projeto possui um tolerância a falhas a nível de Escravos apenas. Porém se o Master ou o repositório falharem o sistema fica inconsistente. Os escravos porém podem ser adicionados e retirados sem maior prejuízo já que eles são meros workers.

ALTERNATIVAS PARA A SINCRONIZAÇÃO ENTRE OS CLIENTES

Os clientes poderiam utilizar outras propostas de comunicação como webhooks e websockets. De forma a ter uma comunicação mais realtime/full-duplex com o repositório.

ANÁLISE DA CONCORRÊNCIA

O método pairOut pode ser chamado por mais de um cliente ao mesmo tempo e isso pode causar alguns problemas como por exemplo a perda de alguma task, para resolver tal problema foi utilizado synchronized no método de retirar pares do repositório dessa forma é alcançado exclusão mútua. Percebemos que synchronized poderia ter sido utilizado em outros métodos e também talvez em um nível de abstração acima para o endpoint não poder ser chamado em uma situação de corrida

ANÁLISE DE DESEMPENHO SIMPLES

1 Slave	**3 Slaves**
18.814s	15.603s
18.321s	12.804s
16.540s	14.071s
14.387s	11.159s
15.673s	11.350s

REST

DESCRIÇÃO DOS ELEMENTOS

Master

O master foi implementado também como uma API REST e pode-se interagir com ele a partir de requisições HTTP, isso foi feito para que facilitasse a manipulação das matrizes salvas, aproveitando a interação com o banco de dados provida pelo framework.

Ele também é responsável pela interação com o usuário, isso foi implementado através de um script em python, que é executado no terminal e interno à API, disponibilizando um pequeno menu para interação. Além de interagir com o usuário, ele também interage com o repositório, depositando nele tasks necessárias para se concluir a multiplicação das matrizes.

O menu disponibiliza as seguintes operações: registrar uma matriz, multiplicar duas matrizes, limpar o banco de matrizes e limpar o repositório.

Os arquivos que o implementam estão na pasta *master*, que contém toda a API. Já o script para interação com o usuário, é o arquivo *master/master/api/management/commands/run_interface.py*

Repository

O repositório também é uma API à parte que guarda pares no formado {chave: valor}, além disso, é totalmente passivo na interação. O master e os slaves sempre irão começar a interação e então o repositório fará alguma alteração interna.

As operações disponíveis são: *pairOut, pairIn* e *readPair*. A *pairOut* remove um par do repositório e o retorna a quem fez a requisição. A *pairIn* insere um par no repositório. E, por último, a operação *readPair* retorna um par a quem o pediu mas não o remove do repositório. Também foi adicionado um endpoint extra, para limpar o repositório.

Da mesma forma que o master, os arquivos desta api estão em pasta separada, na pasta *repository*.

Slave

Diferente dos elementos anteriores, um slave não é uma api, ele foi dividido em apenas dois scripts em python. Um deles, o *Slave.py*, tem a implementação da classe Slave, que é utilizada no segundo arquivo, o *slaves.py*, usa a classe Slave e gera vários processos escravos.

O papel do slave é o de solicitar tasks ao repositório, processá-las e depositar os respectivos resultados de volta no repositório.

DESCRIÇÃO DA EXECUÇÃO

Master

1. O script é executado e um menu é disponibilizado para o usuário e cada opção tem o seu fluxo:
    - Cadastrar matriz
        1. primeiro deve ser fornecido um nome para a matriz, o nome da matriz deve ser único;
        2. depois deve ser inserido cada linha da matriz com os elementos separados por espaço, no final, uma linha em branco deve ser adicionada para indicar o fim da matriz;

3. o master irá tentar salvar a matriz no banco, se for uma matriz válida, ela será convertida para uma string e se o nome for único, ela será salva no banco de dados.

- Multiplicar matrizes
    1. o nome de duas matrizes deve ser fornecido, separados por espaço, a multiplicação ocorrerá na ordem que os nomes forem fornecidos e uma matriz pode ser multiplicada por si mesmo;
    2. se as matrizes forem multiplicáveis entre si, cada linha da primeira matriz e cada coluna da segunda serão enviadas ao repositório;
    3. cada termo da matriz resultante será convertido numa task que será adicionada ao repositório;
    4. uma lista será gerada com o nome da chave de cada resultado que deverá ser buscado no repositório posteriormente
    5. uma busca por cada resultado será feita iterando sobre a lista com os nomes das chaves;
    6. caso o resultado já esteja disponível no repositório, ele será obtido e inserido na matriz resultante e o nome da chave será removido da lista, este procedimento será feito até que a lista dos nomes das chaves dos repositórios fique vazia;
    7. quando a lista estiver vazia, a matriz resultante estará completa, então ela é exibida ao usuário.
- Limpar banco de matrizes
    1. Todas as matrizes são buscadas no banco de dados e removidas.
- Limpar repositório
    1. É feita uma requisição no endpoint clearDatabase do repositório.

Repository

- pairIn
    1. os dados do par são extraídos da requisição;
    2. o par é salvo no banco de dados.
- pairOut
    1. os dados do pair são extraídos da requisição;
    2. o par é buscado no banco de dados;
    3. o par é colocado na resposta da requisição;
    4. o par é apagado do banco de dados;
    5. o par é enviado na resposta da requisição a quem o solicitou
- readPair
    1. os dados do pair são extraídos da requisição;
    2. o par é buscado no banco de dados;

3. o par é colocado na resposta da requisição;
4. o par é enviado na resposta da requisição a quem o solicitou

- clearDatabase
    1. todos os pares são buscados no banco de dados e apagados;

Slave

1. n slaves são instanciados;
2. cada slave irá requisitar pela Next Task ao repositório;
3. se uma task for retornada, o slave irá iniciar a multiplicação dos vetores responsáveis pelo resultado da coordenada da task;
4. cada resultado será colocado no repositório;

DESCRIÇÃO TECNOLOGIA UTILIZADA

Para a criação das APIs foi utilizado o framework Django, em python, e a dependência django rest. Sendo o django rest responsável pela serialização e processamento das requisições.

Na escrita dos scripts foi utilizado a biblioteca requests, disponível para download no PyPi.

Além disso, foi utilizado o docker e docker-compose para isolamento dos ambientes de desenvolvimento e facilitação da portabilidade.

GRAU DE TOLERÂNCIA A FALHAS

A aplicação resiste a falhas de validade dos dados, como por exemplo, ele não deixa que matrizes inválidas sejam salvas e não tenta multiplicar matrizes que não multiplicáveis entre si.

A aplicação só funcionará corretamente para matrizes com no máximo 10 linhas e 10 colunas, porque no nome da chave do resultado é usado apenas um algarismo para referenciar o índice da linha e um algarismo para referenciar o índice da coluna.

Com relação ao processamento dos resultados, podem ter vários escravos inativos, desde que haja um escravo ainda em execução.

Como o servidor não escrito por nós, não sabemos exatamente quantas requisições ele suporta e nem quantos escravos solicitando requisições simultaneamente, mas foi testado com 100 escravos simultâneos e o servidor funcionou bem.

ALTERNATIVAS PARA A SINCRONIZAÇÃO ENTRE OS CLIENTES

- os clientes poderiam se comunicar entre si antes de realizar ações;

- se todos os clientes tivessem acesso ao estado completo do repositório, também seria possível que se organizassem melhor;
- eles poderiam ter acesso em fatias de tempo específicas;
- etc.

ANÁLISE DA CONCORRÊNCIA EXISTENTE

A concorrência é um problema real nesse contexto. Quando vários escravos estão solicitando por tasks ao mesmo tempo, pode acontecer que vários recebam a mesma task antes que ela seja apagada pelo primeiro cliente a terminar a requisição.

Pode acontecer também que tasks desapareçam e nenhum cliente retorne o resultado respectivo.

Para controlar o problema do desaparecimento de tasks, foi implementado um mecanismo que se o resultado não for encontrado no repositório e nenhuma task a ser feita, ela é reenviada ao repositório, até que seu resultado seja calculado.

ANÁLISE DE DESEMPENHO SIMPLES

O tempo mostrado nas tabelas abaixo, incluem o tempo de inicializar o container, executar o script do master, limpar o repositório, enviar as requisições e receber a matriz resposta completa:

Com 1 escravo:

Caso de teste	**Tempo**
1	16,878 s
2	16,686 s
3	19,247 s
4	18,462 s
5	18,445 s
6	17,210 s
7	16,745 s
8	16,376 s
9	18,185 s
10	16,700 s

Com 2 escravos:

Caso de teste	**Tempo**
1	16,143s
2	18,522s
3	16,221s
4	17,922s

5	17,017s
6	15,296s
7	15,377s
8	15,370s
9	15,160s
10	16,105s

Com 5 escravos:

Caso de teste	**Tempo**
1	19,042s
2	20,197s
3	18,149s
4	17,510s
5	16,014s
6	18,735s
7	17,197s
8	14,987s
9	12,752s
10	14,475s

Com 10 escravos:

Caso de teste	**Tempo**
1	16,878s
2	19,247s
3	18,462s
4	18,445s
5	17,210s
6	16,745s
7	16,424s
8	16,376s
9	18,185s
10	16,700s

Pode-se observar a partir das medições, mesmo sem tratamentos estatísticos, que o melhor desempenho registrado foi com apenas um escravo.

Isso pode ser devido a alguns fatores, como:

- o servidor usado nos testes não era um servidor de produção, mas um servidor de desenvolvimento, e portanto não é preparado para um grande número de requisições;
- a matriz não era tão grande a ponto de fazer grande diferença no tempo multiplicar os vetores de forma paralela;
- há muitos resultados repetidos no repositório devido a problemas de concorrência (uma task é pega por vários processos antes de ser apagada).

CONSIDERAÇÕES

Ícaro

Foi um trabalho interessante no sentido de se praticar alguns dos conceitos aprendidos até o momento na matéria, além de evidenciar algumas das diferenças

entre SOAP e REST, que são bastante populares no contexto da Engenharia de Software.

Também foi possível perceber alguns problemas de concorrência não previstos, desse ponto de vista, ficaram mais claros os objetivos e algumas estratégias de alguns algoritmos e protocolos estudados tanto nesta disciplina quanto em outras relacionadas.

Além dos problemas de concorrência, também ocorreram dificuldades que não achei que irão acontecer por estarmos utilizando frameworks modernos, principalmente na solução em REST, mas mesmo o framework fazendo bastante coisa, ainda há implementações que tem que ser feitas que não são específicas do problema.

Por último, ficou evidente a simplicidade do REST com relação ao SOAP, este que exigia alterações em arquivos bem mais carregados de informações e texto, o que por consequência já deixa o código mais suscetível a erros.

Matheus

O trabalho demandou bastante tempo e esforço eu particularmente percebi uma verbosidade muito grande com XML e Java porém há seus benefícios como a geração automática de código.

Percebi que o REST consegue com sua simplicidade um maior controle do código porém com essa liberdade existem problemas por exemplo a conversão de strings para o banco de dados, essa liberdade permite você testar mais facilmente seu código.

Já o SOAP eu percebi que gerar código é uma faca de dois gumes, é muito bom ter código gerado porém a complexidade para entender o código deve ser considerado e também a capacidade de testar esse código já que as vezes ele não é gerado seguindo boas práticas.

O atual funcionamento tem algumas limitações, os problemas de concorrência não foi pensada de forma sistemática e podem existir problemas não detectados, também percebemos que a entrada de dados e leitura pode ser problemática.

REFERÊNCIAS

Documentação do django rest: http://www.django-rest-framework.org/
Documentação do django: https://docs.djangoproject.com/en/2.0/
Documentação do requests.py: http://docs.python-requests.org/en/master/
Documentação do spring soap server: https://spring.io/guides/gs/producing-web-service/
Documentação do spring soap client: https://spring.io/guides/gs/consuming-web-service/