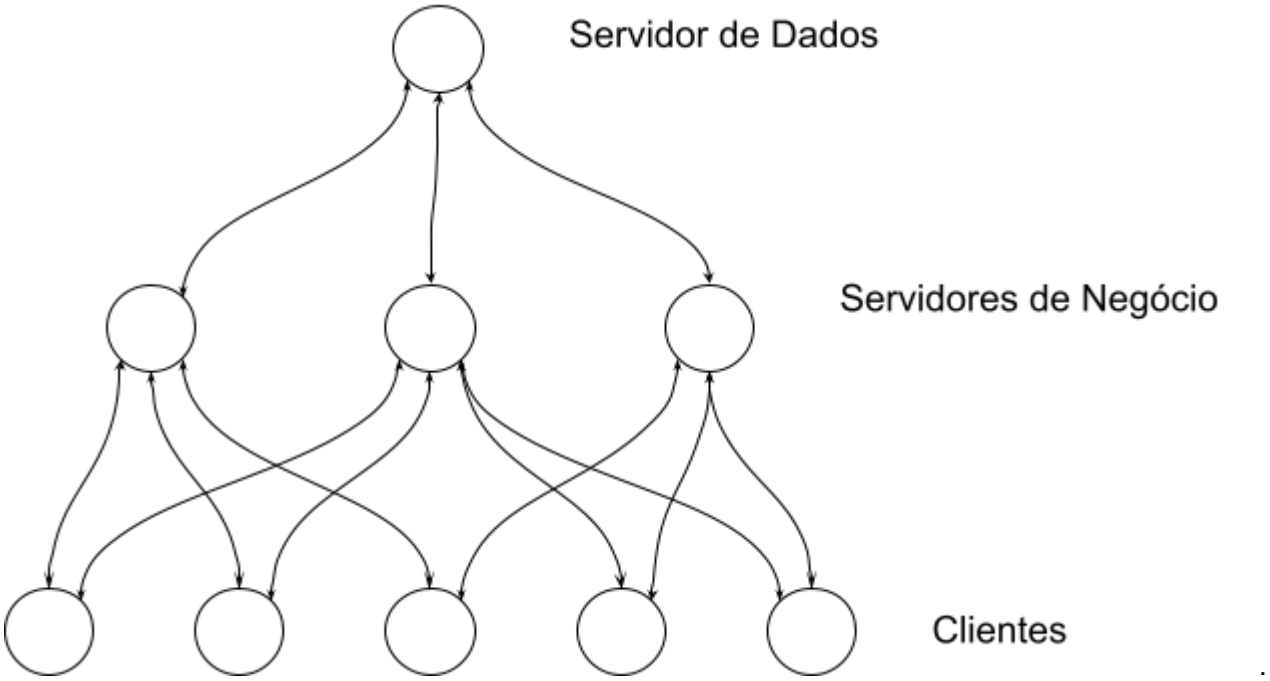
Projeto - Banco Distribuído com WebServices

Objetivo

Implementar um Sistema de Banco Distribuído que consistirá em:

- um servidor de dados (banco de dados)
- três (ou mais) servidores de negócio (responsáveis pela conexão com o cliente e com o servidor de dados)
- vários clientes, implementados em Python (e opcionalmente, em mais uma linguagem de programação diferente)



Responsabilidades e comportamentos

Servidor de Dados

O servidor de dados é responsável por implementar o mecanismo de exclusão mútua no acesso às contas dos clientes. **Deve ser implementado um modelo baseado no conceito de leitores e escritores, onde a exclusão mútua é imposta para o caso de escritores mas não para leitores.** O servidor de dados deverá implementar apenas serviços básicos de atualização de saldo e de lock das contas:

- getSaldo {id_negoc, conta} → retorna -1 se a conta estiver travada por outro servidor de negócio
- setSaldo {id_negoc, conta, valor} → retorna -1 se a conta estiver travada por outro servidor de negócio
- getLock {id_negoc, conta} → retorna -1 se a conta estiver travada por outro servidor de negócio
- unLock {id_negoc, conta} → retorna -1 se a conta estiver travada por outro servidor de negócio

O servidor de dados também deve manter um log de todas as operações/transações. O log deve ser mantido em um arquivo específico, no qual todas as operações são gravadas em modo “append”. Como sugestão de campos de cada registro de log, pode-se pensar em:

TIMESTAMP, NumOperação, ID Servidor Negócio, TipoOperação, Conta, Valor

- TIMESTAMP - momento em que a operação foi efetivamente realizada;
- NumOperação - Número da operação realizada pelo Servidor; deverá ser iniciado quando o servidor é executado, é um número crescente, sem pular.
- ID Servidor Negócio
- TipoOperação - tipo da operação sendo efetuada (dentre as acima)
- Conta - sendo operada
- Valor da transação

O servidor de dados deve ser capaz de atender múltiplas conexões dos servidores de negócio; no caso de uma operação tentar acessar uma conta bloqueada por outra transação, a resposta padrão deve ter um valor -1; outras conexões a contas não bloqueadas devem operar normalmente.
Cada servidor de negócio deve ser autenticado previamente antes de poder operar. Para isso, o servidor de dados terá uma listagem de códigos de autorização; cada servidor de negócio deverá apresentar um destes códigos para receber a autorização (*um token que deverá ser apresentado em cada operação/conexão*) de realizar transações. **Você deverá implementar esta funcionalidade de autenticação como um serviço também.**
Deverão ser feitas simulações em que um servidor de negócio apresenta um código inválido e o servidor de dados não o autoriza.

Servidor de Negócio

O servidor de negócio é responsável por receber as conexões dos clientes, gerenciar as transações e prover os serviços de manipulação das contas:

void deposito(int acnt, int amt) URI: /deposito/{acnt}/{amt}	Aumenta o saldo da conta acnt pelo valor amt e retorna nada
void saque(int acnt, int amt) URI: /saque/{acnt}/{amt}	Diminui o saldo da conta acnt pelo valor amt e retorna nada
int saldo(int acnt) URI: /saldo/{acnt}	Retorna o saldo da conta acnt
void transferencia(int acnt_orig, int acnt_dest, int valor) URI: /transferencia/{acnt_orig}/{acnt_dest}/{amt}	Transferência da conta acnt_orig para a conta acnt_dest do valor amt

O servidor de negócio deve enfileirar sempre 5 operações, que podem ser em contas distintas, antes de enviá-las ao servidor de dados. Para efetuar as operações, ele precisa obter os locks das contas que vai manipular junto ao servidor de dados; só depois inicia o processamento das mesmas.
O servidor de negócio deve manter um log de todas as transações que efetua, registrando informações do tipo:

TIMESTAMP, NumOperação, ID Cliente, Operação, Conta 1, Conta 2, Valor

O servidor de negócio deve ser capaz de atender múltiplos clientes, e para cada cliente deve manter um controle das operações que ele realiza, registrando tudo no log. Cada cliente deverá ser autenticado antes de poder iniciar a realização das operações.

Para isso, o servidor de negócio terá uma listagem de códigos de autorização; cada cliente deverá apresentar um destes códigos para receber a autorização (*um token que deverá ser apresentado em cada operação/conexão*) de realizar transações.
Deverão ser realizadas simulações em que um cliente apresenta um código inválido e o servidor de negócio não o autoriza a operar.
Para tratar os múltiplos clientes, pode ser necessário adotar um paradigma de múltiplas threads, onde cada thread serve um cliente e se comunica com o servidor de dados. Em cada conexão com o servidor de dados, o servidor de negócio deve enviar o **token** de autorização que recebeu na autenticação.
A conexão com o servidor de dados deve ser implementada com Webservices.
O servidor de negócio oferecerá uma interface REST para os clientes, **devendo ser capaz** de enviar respostas ~~tanto em formato XML como~~ JSON, dependendo do que for solicitado pelo cliente no Accept.

Clientes

Os clientes serão basicamente os mesmos do projeto anterior, utilizando sempre Webservices e **poderão** se conectar a mais de um servidor de negócio simultaneamente, realizando em cada um transações/operações distintas ou a mesma (para fins de teste de consistência).
Opcionalmente poderão ser feitas duas implementações dos clientes, com as mesmas funcionalidades, feitas em linguagens de programação diferentes, que sejam multiplataforma, ou seja, possam ser executadas no Linux, Windows ou Mac.
Os clientes executarão as mesmas transações/operações do projeto anterior, listadas abaixo.
Os clientes se comunicarão com os servidores de negócio **exclusivamente** através de Webservices, podendo escolher receber respostas em ~~XML o~~u JSON.
Deve ser implementada uma rotina de teste no cliente que executa diversas operações automaticamente, se conectando a mais de um servidor de negócio simultaneamente, acessando, através destes servidores de negócio diferentes, a mesma conta; isto será feito para verificar a confiabilidade das medidas de sincronismo (exclusão mútua) implementadas no sistema.
Nesta rotina de teste deverá haver tentativas de acesso sem autenticação (na fase de Autenticação) para verificar a segurança do sistema.

Generalidades

O servidor de dados administra todas as informações de conta dos usuários. Uma pessoa cliente poderá invocar as seguintes operações em uma ATM:

1. void deposito(int `conta`, int `valor`)
2. void saque(int `conta`, int `valor`)
3. void transferencia(int `conta_orig`, int `conta_dest`, int valor)
4. int saldo(int `conta`)

Cada uma das operações acima deverá ser transformada em um webservice, cujas URI's estão especificadas no servidor de negócios.

Requerimentos de Programação

Os servidores de dados e de negócio deverão ser implementados preferencialmente na mesma linguagem de programação (~~Java,~~ Python com Flask ~~ou Go~~), observados os requisitos de comunicação descritos acima.

~~Você é livre para usar qualquer linguagem das opções dadas (Java, Python ou Go) contanto que seu código realize as mesmas operações mencionadas acima.~~

Este projeto utilizará exclusivamente **REST** como protocolo de acesso aos serviços web (webservices). As mensagens poderão ser trocadas em ~~XML ou~~ JSON.

Funcionamento

Inicialmente, o servidor de dados vai criar 10 contas numeradas de 1 a 10 com saldo inicial 1000

Os servidores de negócio são uma mesma implementação (mesmo código executável), rodando em máquinas diferentes. Eles recebem como parâmetro de execução o endereço IP do servidor, isto é, a URL completa do servidor de dados.

Os clientes recebem como parâmetros de entrada os seguintes:

- *server_address:* o endereço do **servidor de negócio**
- *authentication_server:* o endereço do **servidor de autenticação** (quando estiver implementado)
- *operacao:* uma dentre “deposito”, “saque”, “transferencia” e “saldo”
- *conta:* a conta do usuário; no caso de “transferencia”, duas contas, origem e destino.
- *valor:* somente para as operações “deposito”, “transferencia” e “saldo”

Notas adicionais

O sistema deverá ser desenvolvido em uma das plataformas de nuvem (AWS ou Azure), em máquinas rodando Linux

Demo (faça as adaptações necessárias para Python)

1. Inicie o servidor de dados
2. Inicie os servidores de negócio (cada um)
3. Finalmente, teste os clientes, realizando diversas operações.

Submissão

O projeto deverá ser entregue na data especificada no Moodle e deve ser submetido eletronicamente pelo Moodle, com todos os arquivos zipados. Os seguintes arquivos devem ser submetidos:

- Todos os **códigos fonte**
- Um arquivo **Readme**, que descreve brevemente todos os arquivos submetidos. No arquivo Readme você deve prover também informação acerca do ambiente de execução e instruções de execução, etc. Você deve prover tanta informação detalhada quanto você julgue que possa ajudar o processo de correção e execução do seu código.
- Um arquivo **Amostra**, que descreve os testes que você executou no seu programa para se certificar de sua correção. Também descreva quaisquer casos para os quais o seu programa não esteja funcionando corretamente. Enviar os arquivos de logs dos servidores (dados e negócio). Também adicione *prints* das telas demonstrando a execução.
- Um arquivo **Projeto**, que descreve o projeto global do programa, uma descrição verbal de “como ele funciona”, incluindo as bases do que o sistema está fazendo (por trás das interfaces) e as decisões de projeto consideradas e tomadas. Também descreva possíveis melhorias e extensões ao seu programa (e rascunhe como elas poderia ser feitas).

Notas

As notas serão (mais ou menos) baseadas nos seguintes elementos:

- Todos os arquivos necessários foram submetidos? (5%)
- O código fonte é fácil de entender? (i.e., boa estrutura e comentários) (25%)
- O sistema roda corretamente? (25%)
- O sistema foi testado por você com sucesso? (prints e logs das execuções) (20%)
- Cobertura dos casos de teste (5%)
- Documentação e Relatório do Projeto (20%)

COMECE O QUANTO ANTES. É UM PROJETO DESAFIADOR.
Leia atentamente o artigo "How Not to Go About a Programming Assignment". Busque no Google.