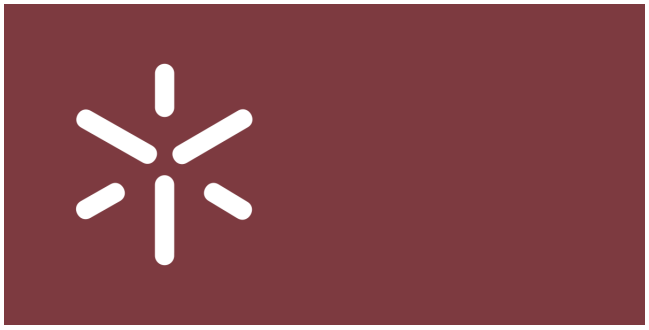
Universidade do Minho

Mestrado Integrado em Engenharia Informática



Sistemas Distribuídos

Trabalho realizado por:	*Número*
Tiago Baptista	A75328
Frederico Pinto	A73639
Filipe Fortunato	A75008
Ricardo Canela	A74568

12 de Fevereiro de 2019

Conteúdo

1 Contextualização do problema

Foi pedido como objetivo deste trabalho o desenvolvimento de um serviço de alocação de servidores na nuvem e de contabilização do custo do aluguer dos mesmos.Tendo como inspiração as plataformas já existentes que permitem reservar vários tipos de Servidores,cada um deles com uma certa configuração de hardware e um preço nominal predifinido.

Sendo que o funcionamento do trabalho poderia ser inpirado no serviço prestado pela Amazon, no qual existem diversos tipos de Servidores com um preço nominal fixo que podem ser requisitados,se disponíveis, pelos Utilizadores.

A esta funcionalidade acrescenta-se ainda a possibilidade de existirem leilões efetuados sobre um servidor, no qual os interessados poderiam efetuar "bids"para tentarem adquirir os serviços de um servidor

O serviço desenvolvido permite :

- Autenticação e registo de utilizador;
- Reservar um servidor a pedido;
- Reservar uma instância em leilão
- Libertar um servidor;
- Consultar a sua conta corrente;

2 Desenvolvimento do Servidor e Cliente

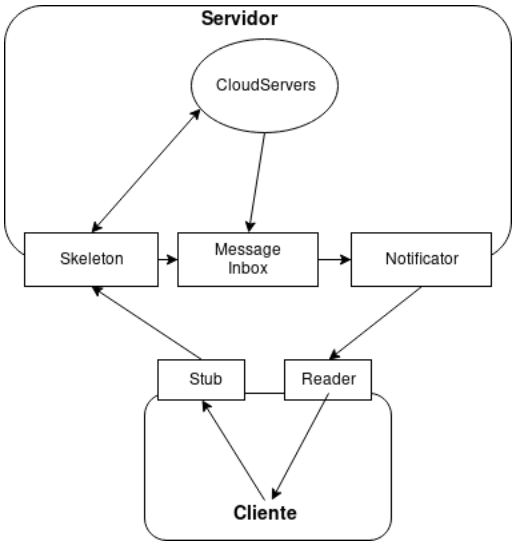


Figura 1: Arquitetura do Sistema

2.1 Servidor

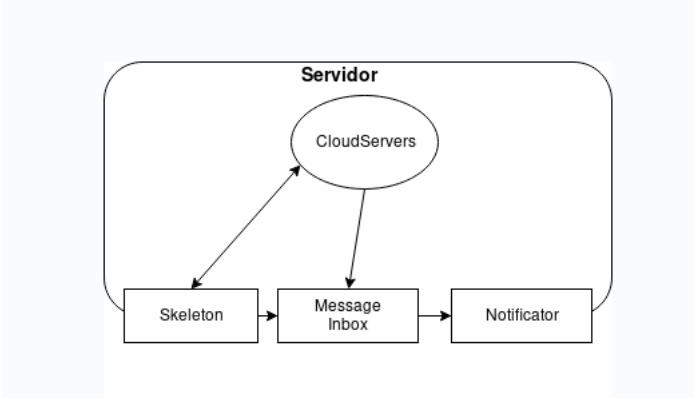


Figura 2: Arquitetura do Servidor

O servidor foi criado de maneira a suportar a ligação de vários clientes ao mesmo tempo e também a criação de leilões.

O funcionamento do servidor assenta em 3 classes fundamentais a CloudServers, Skeleton e Notificator.

Sendo que a classe Skeleton funciona como um listener, estando sempre à escuta de inputs vindo do Cliente, processa-os verificando o que o cliente pretende

e comunica com a classe Notificator através de uma variável partilhado do tipo MessageInbox(que tem a função de guardar as mensagens enviadas a cada cliente). Após receber uma "mensagem"do Skeleton, a classe Notificator responde ao Cliente.

A classe CloudServers possui as estruturas de dados com toda a informação sobre Clientes,Servidores,Leilões e Mensagens por Cliente.Possui também todos os métodos de controlo do Servidor,como por exemplo o SignIn, atribuição de leilões, adicionar fundos, entre outros.

2.2 Cliente

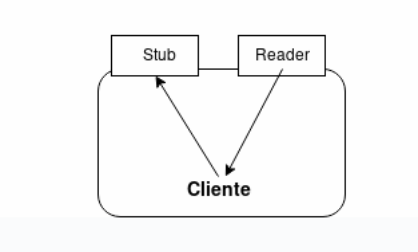


Figura 3: Arquitetura do Cliente

O funcionamento do Cliente assenta em 2 classe a Reader e Stub.

A classe Reader é o listener do cliente, responsável por estar à escuta de mensagens de resposta enviadas pelo Servidor e dependendo da situação mostrar ao Cliente a resposta obtida no ecrã ou menus para que o Cliente continue a interação com o serviço.

A classe Stub é a responsável por processar a informação que o Cliente pede e enviar esse mesmo pedido para o Servidor.

3 Comunicação Servidor-Cliente

O Servidor foi implementado com recurso a Threads e Sockets TCP.

Como explicado anteriormente a comunicação efetiva entre o Cliente e Servidor é da Responsabilidade da Notificator(do lado do Servidor) e da Stub (do lado do Cliente). O envio de mensagens para o Servidor e do Servidor é efetuado através de um PrintWriter, que envia a informação(neste caso Strings) pela Socket.A leitura de mensagens é efetuada através de um inputStreamReader, que recebe as informações enviadas(Strings) pela socket.

De seguida passa-se a exemplificar o processo de envio de uma mensagem do Cliente, Resposta do Servidor e receção da mesma pelo Cliente.

3.1 Exemplo

O cliente indica que pretende efetuar o login, o Stub envia a informação para o Servidor que é lida pelo Skeleton do Server, que recebe a info e processa o pedido enviando-o para o Notificator através de uma MessageInbox. A partir daí o notificator analisa a MessageInbox e envia a resposta para o Reader do Cliente.

4 Leilões

Para a gestão dos leilões decidimos por bem ter apenas um leilão de cada tipo de server disponível. Para resolver o problema de gerir os leilões, decidimos criar um método que gere um tipo de leilão. Esse método presente no *CloudServers* verifica se está a acontecer um leilão daquele tipo, se não estiver, cria um objeto *Auction* que contêm já o server que irá para leilão e coloca esse objeto na estrutura que contêm os leilões ativos. Após isso fica a dormir(tempo de duração do leilão), e quando acorda verifica se o objeto Auction criado ainda se encontra na estrutura, se estiver, termina o leião reservando o server para o cliente com o preço que ele licitou. Caso o leilão tenha sido cancelado pela nuvem, este método descrito acima não irá encontrar o Auction criado na estrutura, então irá avançar para a próxima iteração do ciclo.

Estes métodos estão presentes no *CloudServers* e no *Server* e é criada uma thread por cada tipo de leilão que executa apenas o método criado que gere esse tipo de leilão.

No nosso caso como temos dois tipos de leilão, temos dois métodos e consequentemente duas threads.

5 Concorrência

O serviço desenvolvido está sujeito a vários utilizadores a interagir com apenas um Servidor,ora de maneira a garantir a integridade e consequente bom funcionamento do serviço, foi necessário implementar controlo de concorrência. Para tal usou-se ReentrantLocks, Conditions(controlar as Threads) e métodos synchronized.

Detetou-se que existiam diversas operações que necessitavam de controlo para evitar problemas com o funcionamento geral do serviço assim como transmissão de dados incorretos. Como o cliente contêm duas threads, o *Stub* e o *Reader*, elas têm que "comunicar"entre si, para isso utilizamos uma condição para permitir organizar a comunicação das mesmas. Também utilizamos esse método com o *Notificator* que partilha uma condição com a classe *MessageInbox* de maneira a possibilitar a comunicação entre ambos.

Já na aplicação principal, utilizamos *ReentrantLocks* para bloquear certas zonas críticas como as estruturas de dados, para assim permitir uma consistência da informação nelas presente. Utilizamos métodos synchronized com o mesmo intuito dos *ReentrantLocks* para manter a ordem nos acessos às estruturas em certos objetos. Como exemplo temos um método synchronized na classe *Auction* que nos permite um cliente faça uma licitação, mas um de cada vez para o *Auction* em questão, mantendo assim a ordem.

6 Conclusão

Este trabalho pratico permitiu por em prática conhecimentos lecionados na unidade curricular de *Sistemas Distribuidos* sobre a comunicação entre um servidor multithreaded e varios clientes atraves de sockets TCP em Java

A gestão de concorrência dentro do projeto foi o maior desafio encontrado, pois houve necessidade de analisar e identificar zonas críticas de maneira a não existirem conflitos,evitando também por outro lado situações de *deadlock*.

Em suma ficaram muitos conhecimentos e experiências que seguramente serão úteis caso,no futuro, caso algum dos membros se deparar com um projeto do mesmo género.