Universidade do Minho
Mestrado Integrado em Engenharia Informática
Departamento de Informática

Sistemas Distribuídos

Trabalho Prático
Alocação de Servidores na Nuvem

Grupo 41
Bernardo Manuel Ribeiro Marques Soares Silva
A77230
José Francisco Gonçalves Petejo e Igreja Matos A77688
Nuno Filipe Maranhao Dos Reis A77310
Paulo Jorge Costa Conceição Mendes A78203
6 de Janeiro de 2019

Conteúdo

1 Introdução

O trabalho de Sistemas Distribuídos deste ano lançou como desafio uma pequena simulação de interações entre clientes e um sistema de armazenamento de Cloud, com base neste problema seriam expectáveis vários desafios relativos a sincronização de threads a competirem por espaço na Cloud o que daria um bom desafio para aplicar os conhecimentos adquiridos durante as aulas desta unidade curricular. Antes de mais, o projeto foi dividido em partes competentes de modo a minimizar a carga por aluno e a obter uma vista mais abrangente sobre o que seria necessário neste projeto. Deste modo, foram identificadas as partes de Cloud, de Cliente e de Business ou mais facilmente descrito como o "rental process" de cada servidor. Foi imediatamente visto que o maior desafio seria o design da cloud e métodos ligados a ela quer poderiam ser concorrentes entre vários Workers que serviam como "bridges" de conexão a clientes que acediam a esta e que a boa implementação de locks seria confinada maioritariamente a processos inerentes a estes workers.

2 Desenho da Solução

De forma a melhor entender a solução escolhida pelo grupo, apresenta-se um desenho desta:

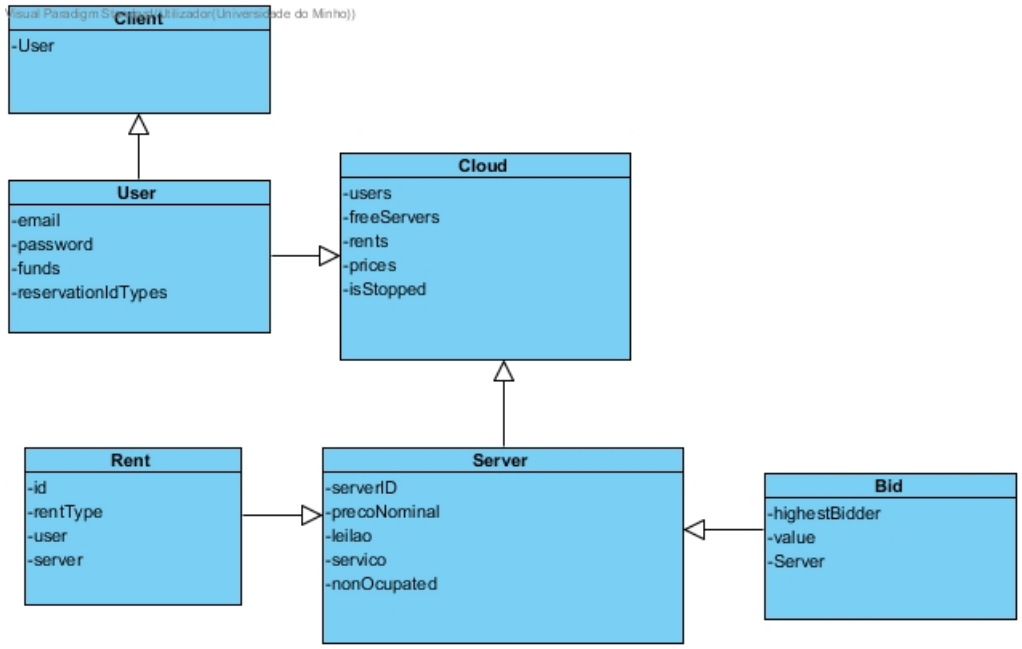

Figura 1: Desenho da Solução

Nesta imagem é possivel observar as classes fundamentais para o funcionamento do trabalho, começando pelo lado do Cliente, que é representado pelas classes *Client* , que realiza a comunicação com a Cloud através de *ReadFromCloud* apresentando ao utilizador o sucesso/insucesso do programa (o processo de comunicação será visto mais à frente) e *User* (que representa o utilizador em si).

Do outro lado da comunicação temos o servidor, que não representa o servidor típico a que estamos habituados, mas sim um produto a arrendar, como foi apresentado no enunciado. Esta classe apresenta o identificador do servidor, o seu preço nominal, caso o utilizador pretenda arrendar o servidor diretamente. Caso pretenda entrar num leilão, *Bid*, esta é guardada, para se saber qual o utilizador que está a ganhar o leilão e qual o seu valor. Por fim, também apresentá-mos uma classe *Rent* que indica o serviço que está a ser oferecido, se é uma alocação por leilão, que pode ser retirada ao utilizador, ou por preço nominal, que tem prioridade.

A última classe, é a *Cloud*, que funciona como o servidor do cliente. Esta armazena a informação relativa aos dados do projeto, ou seja:

- users - Um *HashMap* que associa o email a um user especifico

- freeServers - Um *HashMap* que associa o id de um servidor à quantidade de servidores desse tipo disponiveis
- rents - Um *HashMap* que associa um id a uma *rent* especifica
- prices - Um *HashMap* que associa o tipo de servidor ao seu preço

Para além de armazenar eta informação, também é responsável por estabelecer comunicação com o cliente, que irá ser abordado de seguida.

3 Comunicação Cliente Cloud

A comunicação entre cliente e a Cloud foi implementada da mesma forma como uma conexão cliente-servidor apresentada durante as aulas praticas desta UC. Um cliente inicia um pedido de conexão para o IP e a porta relevantes, esperando que exista um acesso livre para iniciar a ligação para com a cloud. Esta ligação será fornecida por um conjunto de "workers", ou seja threads individuais, que apresentam um connection socket oriundo do método accept() do Server Socket. A partir do momento a que o Worker aceita a conexão de um cliente, este ganhara acesso a comandos que o deixarão interagir com o processo de arrendamento e com outros métodos relevantes. Quando a conexão termina, o worker será libertado da sua carga e o processo inicia novamente. Para ilustrar o funcionamento da comunicação entre o cliente e a cloud foi realizado um esboço:

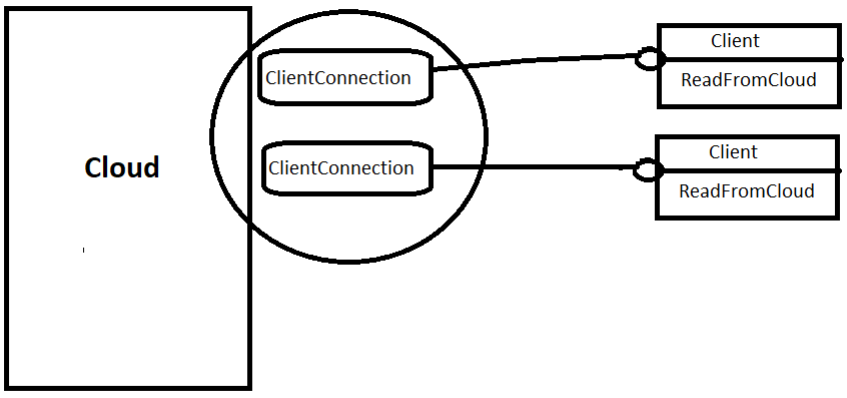

Figura 2: Comunicação entre Cliente e a Cloud

Como se pode observar, a *Cloud* possui um *ServerSocket* onde se encontram os sockets que comunicam com o cliente (*ClientConnection*) e do outro lado possuimos os vários clientes que possuem um socket cada, e interpretam a informação recebida pela *Cloud* para apresentar ao utilizador (como mensagens de sucesso ou insucesso) e enviam para a *Cloud* mais especificamente para o *ClientConnection* o que o utilizador pretende fazer, como *login*, arrendar um servidor ...

4 Concorrência

De forma a garantir que o programa funcionasse sem ocorrer nenhuma dos problemas de sistemas distribuidos estudados nas aulas, como *Deadlocks*, *Starvation*, Exclusão mútua, etc, foi realizado um controlo de concorrência nas classes externas ao *Client-Cloud*.

Como por exemplo, a classe de *User*, oferece *locks*, para cada um dos atributos que podem vir a ser acedidos, e sempre que for necessário realizar um acesso, tanto de leitura como de escrita nesses atributos, é primeiro bloqueado o acesso, para apenas uma thread poder entrar, e depois de ser realizado o acesso, esta é desbloqueada.

Esta ideologia foi seguida durante o trabalho e garante um controlo eficaz das secções críticas do código desenvolvido

5 Conclusão

A conclusão que foi obtida na realização deste trabalho foi curiosa. Especialmente visto que o objetivo nele encapsulado era extremamente diferente de qualquer outro trabalho realizado anteriormente. Com este trabalho concluímos que, por vezes, o desafio de um sistema pode não vir na sua implementação pratica no que toca a conexões individuais e que o código encapsulado nestas conexões se torna trivial com a experiencia obtida por um programador e que os desafios começam a ser outros que por vezes se visam mais difíceis de interpretar e resolver ou talvez simplesmente impossíveis de prever. Estes problemas são originados pela necessidade de distribuir carga para otimizar um sistema, distribuindo a "workload" por varias threads de um processador, seja este de um computador local ou de um servidor com vários processadores interligados e visam as matérias aprofundadas durante

as aulas como deadlocks, acessos concorrentes e conflitos read/write, entre outros. Se não tratados, estes originam resultados extremamente inconsistentes e, por muitas vezes, resultam no colapso de um sistema distribuído. Apesar de considerarmos o trabalho desenvolvido um sucesso, existe uma funcionalidade que não foi possível implementar, e que no futuro pode ser desenvolvida, notificar os clientes autenticados que um leilão foi cancelado. No entanto, todos os outros requisitos apresentados no enunciado, foram cumpridos com sucesso.