Trabalho Prático 1

Sistemas Distribuídos

João Ricardo Lourenço, Nº 2011151194
Joaquim Pedro Bento Gonçalves Pratas Leitão, Nº 2011150072

26 de Outubro de 2013

Relatório

1 Introdução

No presente projecto pretende-se criar uma aplicação responsável por gerir e organizar ideias e tópicos, submetidos por utilizadores da mesma. A aplicação permite ao utilizador realizar um conjunto de acções/shares, como é o caso de:

- Registar um utilizador na aplicação.
- Permitir a um utilizador registado efectuar o login na aplicação.
- Criar uma nova ideia. O utilizador pode criar uma ideia que será adicionada ao conjunto de ideias da aplicação, a qual terá de se encontrar associada a pelo menos um tópico.

    1. Aquando da criação de uma ideia é necessário definir um número de shares da ideia e um preço por share da ideia.
    2. É ainda possível estabelecer relações com outras ideias já existentes na aplicação, sendo estas relações de concordância (que indica que a ideia a criar suporta a ideia seleccionada), discordância (que indica que a ideia a criar não suporta a ideia seleccionada), neutral (que indica que a ideia a criar não tem qualquer tipo de relação com a ideia seleccionada), ou inexistente.
    3. Por fim, o utilizador poderá optar por associar um ficheiro à ideia a criar, indicando esse mesmo ficheiro que será descarregado para a base de dados, onde será armazenado. Posteriormente é possibilitado ao utilizador descarregar esse mesmo ficheiro (ou outro qualquer ficheiro associado a uma qualquer ideia) para o seu computador.

- Criar uma nova ideia directamente como resposta a uma ideia já existente na aplicação.
- Listar todos os tópicos da aplicação. Após ter sido feita uma listagem dos tópicos, o utilizador pode escolher um tópico para consultar, inspeccionando as ideias que o constituem. Ao inspeccionar uma ideia dentro de um tópico, podemos criar uma nova ideia directamente como resposta à ideia que estamos a inspeccionar.
- Criar um novo tópico. Esta funcionalidade encontra-se implícita na criação de uma nova ideia. Aqui, ao especificarmos a associação da ideia um tópico inexistente na base de dados é automaticamente criado um tópico com o nome indicado, onde é inserida a ideia que pretendemos criar.

- Remover uma ideia. Um utilizador pode remover uma ideia existente na base de dados, desde que seja portador da totalidade das suas shares, e se a ideia em questão não possua qualquer ideia que a suporte, contrarie ou que lhe seja neutra (ideias-filho).

- É permitido a um utilizador realizar compras e vendas de shares de ideias, através do preço por share e número de shares de cada ideia na posse de um utilizador. Assim, quando uma ideia é criada, a totalidade das suas shares são atribuídas ao utilizador que a criou, podendo este garantir um número mínimo de shares da ideia que não pretende vender. Posteriormente, caso exista um segundo utilizador interessado em comprar shares dessa ideia, se dispuser de dinheiro suficiente para a compra, poderá comprar a totalidade (ou parte) das shares do utilizador.

- Aquando da execução de uma ordem de compra de shares de uma ideia, caso a compra se realize com sucesso, os utilizadores envolvidos na operação (comprador e o vendedor, ou vendedores) recebem uma notificação, informando-os da realização da transacção. Caso esta não possa ser realizada, é armazenado o pedido de compra, esperando um momento em que seja possível a sua realização. Quando uma compra é realizada, se um (ou mais) dos utilizadores nela envolvido não estiver ligado à aplicação (isto é, se se encontrar offline), apenas receberá uma notificação da realização da transacção na próxima vez que se ligar.

- Sempre que o utilizador pretender consultar o seu histórico de transacções poderá fazê-lo no menu principal da aplicação, sendo-lhe mostrada a lista de transacções que o envolveram, ordenada por data de realização da transacção.

Assim, a aplicação desenvolvida apresenta-se como uma espécie de forum de ideias, com um mercado de acções associado, onde é permitida a compra e venda de acções de ideias, bem como a actualização dos preços das shares das acções das diferentes ideias, distribuídas por vários utilizadores.

Desta forma, pretende-se apresentar uma forma de organizar ideias de maneira clara e simples, contrariando a sua organização actual na Internet, que apresenta alguns problemas de estruturação que visamos ultrapassar.

2 Arquitectura Interna do Serviço

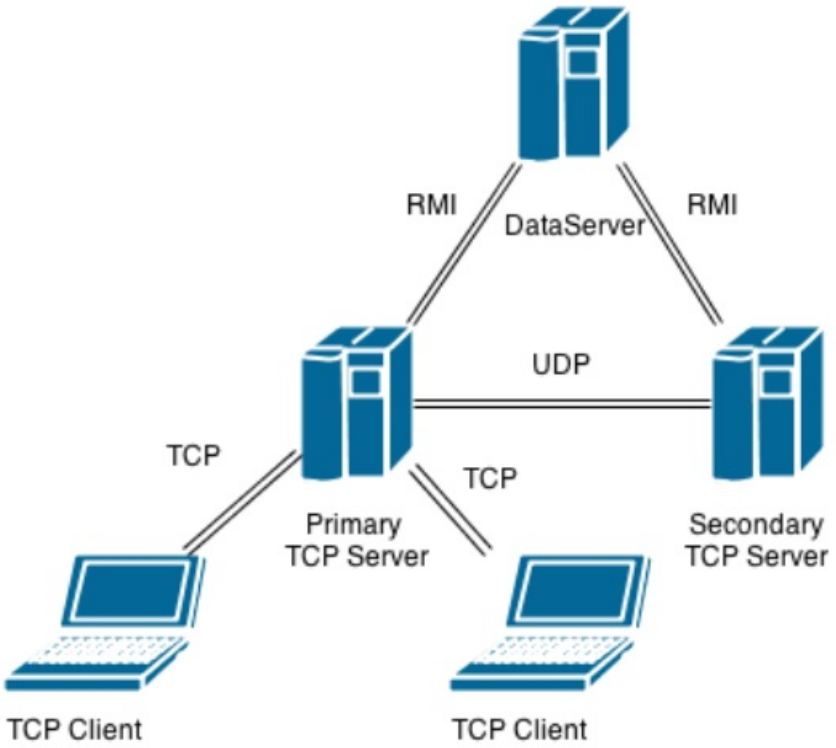

Figura 1: Arquitectura Interna do Serviço

A aplicação é constituída por 5 entidades diferentes. São elas:

2.1 Aplicação Cliente

A aplicação cleinte é responsável por fazer a ligação ao servidor primário, controlando e tratando a interacção com o utilizador. Recolhe todos os dados necessários para realizar as acções pretendidas pelo utilizador, enviando pedidos para o servidor primário e recebendo as suas respostas, que transmite ao utilizador. Em caso de falha do servidor primário, estabelece uma ligação com o servidor secundário, continuando a realizar as mesmas acções.

Internamente, esta aplicação possui duas threads em execução. Uma thread lida com os inputs do utilizador, a comunicação com o servidor para todos os pedidos e o output dos mesmos no ecrã. A segunda thread é executada paralelamente e liga-se à porta de notificações do servidor, sendo responsável por notificar o cliente de transacções que tenham acabado de ser feitas. De destacar que o cliente é responsável por garantir que nunca estamos ligados a servidores distintos (um para comandos e outro para notificações).

2.2 Servidor

O servidor tem a função de intermediário entre a aplicação cliente (e o próprio cliente) e o servidor RMI (que tem acesso aos dados). Recebe pedidos da aplicação cliente e os dados necessários para a sua realização e recorre a métodos disponibilizados pelo servidor RMI para executar os pedidos da aplicação cliente, transmitindo-lhes as respectivas respostas.

Para cada cliente, o servidor possui duas threads correspondentes – uma para a porta de comandos e outra de notificações. Deste modo, cada cliente possui dois sockets de ligação ao servidor.

A thread de notificações periodicamente verifica o RMI para ver se existe alguma notificação que o utilizador precise de receber.

O servidor pode ser considerado "secundário" ou "primário". Se for "secundário", em caso de falha do servidor primário assume o papel deste, respondendo a pedidos da aplicação cliente, tal como é feito no servidor primário.

Enquanto o servidor primário está activo, os servidores primário e secundário comunicam entre si, com o objectivo do servidor secundário poder tomar conhecimento de quando ocorre uma falha do servidor primário e informar os clientes que por algum motivo se ligam a si de que o servidor primário não é ele, terminando a ligação. Deste modo, nunca será possível que tanto servidor primário como secundário aceitem clientes.

2.3 Servidor RMI

Este é o servidor que implementa métodos remotos necessários para realizar operações sobre a base de dados (inserção, remoção e selecção de dados). Estes métodos remotos são invocados pelos servidores primário e secundário sendo, no entanto, executados no fluxo de execução do servidor RMI.

O servidor RMI possui uma fila de pedidos de compra de acções para garantir "Transactional Trading" – quando uma operação de compra de shares falha, é adicionada, de forma ordenada, a esta fila para que possa ser novamente tentada quando há alterações nas shares de ideias.

2.4 Base de dados

Na base de dados são armazenados todos os dados dos utilizadores, ideias, tópicos, etc. As operações sobre a base de dados são realizadas pelo Servidor de RMI, com o qual a base de dados estabelece uma ligação. O servidor TCP desconhece a existência da Base de Dados e, por isso, todas as operações devem ser transparentes e independentes de uma falha da mesma.

Destacamos o facto de o servidor primário e o servidor secundário trocarem de papéis.

3 Modelo de Dados da Aplicação

Para armazenar os dados da aplicação optámos por criar uma base de dados, cujo modelo de dados apresentamos na figura que se segue:

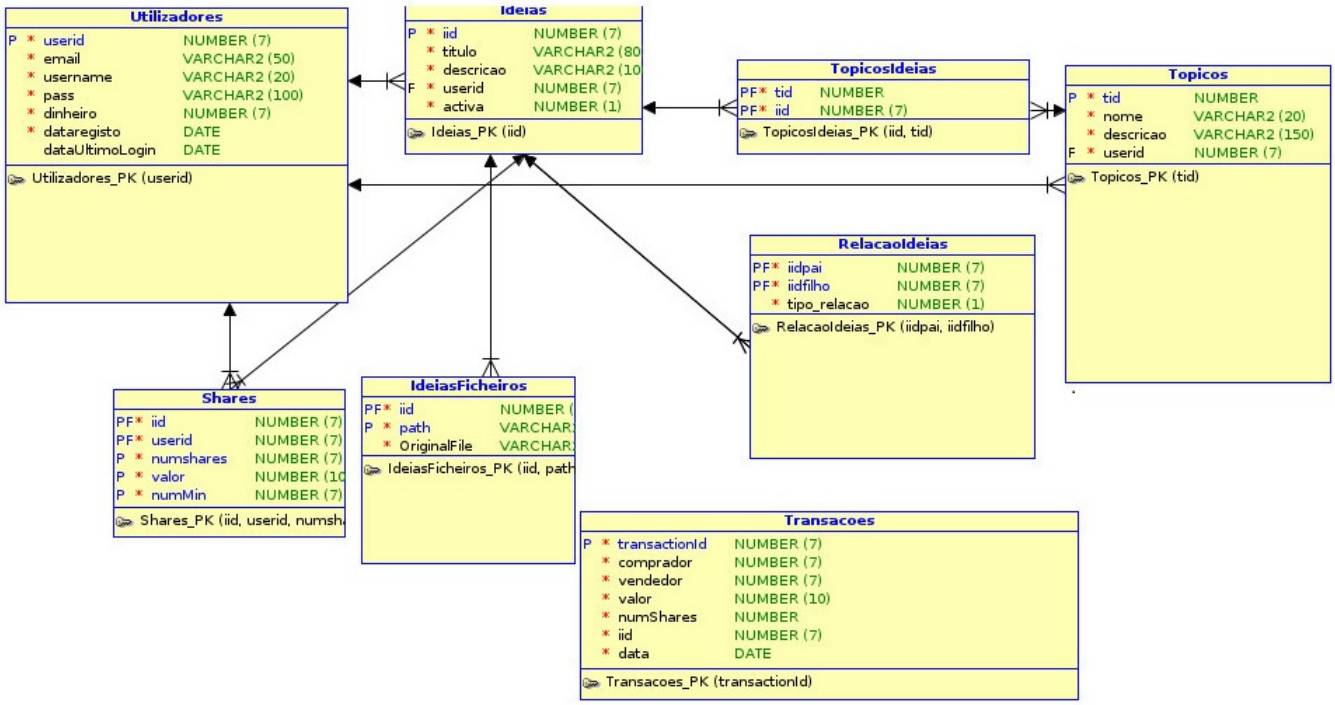

Figura 2: Modelo de Dados da Base de Dados Implementada

O nosso modelo prevê a expansão da funcionalidade de anexar um ficheiro, possibilitando, mais tarde, a adição de vários ficheiros.

O campo "activa" da tabela Ideias foi adicionado para quando queremos "remover" uma ideia. A escolha pela não remoção de ideias da base de dados deve-se à existência do histórico de transacções, em que queremos ser capazes de consultar os dados, quanto mais não seja o nome, de uma ideia agora inexistente, mas para a qual houve transacções.

4 Especificação dos Protocolos Usados para Comunicação

4.1 Ligação Cliente-Servidor (Protocolo TCP)

Para a ligação entre o cliente e o servidor primário, utiliza-se o protocolo TCP com uma abordagem por mensagens. Uma mensagem é um inteiro identificativo que pode indicar um pedido (por exemplo, "quero a lista de ideias"), um erro (por exemplo, "não pode executar isso porque não está autenticado") ou uma simples confirmação.

A par do mecanismo de mensagens, que serve maioritariamente para iniciar ou terminar um processo mais complexo, a maior parte dos dados é transmitida recorrendo a Input/OuputDataStreams para escrever inteiros, strings e arrays em bruto. Deste modo, apesar de não utilizarmos as funcionalidades da ObjectOutput/InputStream do Java para facilitar o envio de mensagens, somos capazes de compartimentalizar mais o nosso processo de troca de dados, permitindo a cliente e servidor terem uma maior noção do que já foi enviado ou não e lidar com esta situação em caso de erro.

Conforme mencionado, este mecanismo por mensagens permite ao cliente e ao servidor progredirem na sua comunicação faseadamente, sabendo onde cada um está. Por exemplo, no processo de login, é enviada uma mensagem MSG_LOGIN, seguida do username e da password (como strings UTF) e uma subsequente mensagem do servidor para o cliente a indicar o sucesso ou insucesso do login. No envio de uma lista de tópicos a associar a uma ideia, por exemplo, cada tópico é enviado individualmente e apenas quando o que o precede já foi recebido e confirmado pelo servidor. Deste modo, o servidor pode inclusivamente indicar a existência de um tópico inválido, mas proceder com o tratamento dos restantes.

4.1.1 Reacção do cliente a uma perda de ligação

Quando o cliente se apercebe de que perdeu uma ligação ao servidor, inicia instantaneamente o processo de reconecção, começando por tentar ligar-se novamente a servidor actual (por forma a contornar uma possível falha transiente) e, caso esta ligação falhe, tentando todos os outros servidores da sua lista. A partir do momento em que uma ligação é obtida com sucesso, o programa cliente re-autentica o utilizador (se necessário) e reinicia a transmissão de dados que estava a ser efectuada. Destacamos que se reinicia a transmissão de toda a acção e não de um campo em particular. Por exemplo, no caso de uma falha no login, voltar-se-ia a enviar a mensagem MSG_LOGIN, seguida do username e password, mesmo que a falha tivesse sido detectada apenas no envio da password. Cabe, pois, aos servidores, garantir que este funcionamento não danifica a estrutura dos dados – torna-se necessário garantir idempotência, algo que abordaremos em tópicos

seguintes.

Se o cliente não conseguir ligar-se a nenhum servidor disponível, continuará a tentar até que haja uma ligação.

4.1.2 Reacção do servidor a uma perda de ligação

Quando o servidor detecta que a ligação a um cliente foi terminada, termina a thread deste cliente, garantindo que quaisquer operações que tenham sido iniciadas e que necessitem de ser posteriormente retomadas (inclusivamente para se tornarem idempotentes), têm o seu estado armazenado de alguma forma no servidor de RMI. Este mecanismo é apelidado no código de "Request Queue" – uma fila de "pedidos críticos que o cliente fez ao servidor e que necessitam de acesso ao RMI".

4.2 Ligação Servidor Primário-Servidor Secundário (Protocolo UDP) – Mecanismo de Failover

Por forma a serem capazes de monitorizar o estado um do outro, bem como a sua tarefa (primário vs secundário), os servidores utilizam um mecanismo de pings UDP.

Aquando da sua inicialização, o administrador terá indicado a cada um dos servidores qual o hostname do outro, bem como a porta UDP para a qual deverá enviar pings e a porta UDP na qual se deverá escutar por pings.

Um servidor tem então duas threads adicionais responsáveis por enviar e receber pacotes UDP (uma para enviar e outra para receber). Os pacotes são enviados de X em X tempo (no caso 1 segundo) e espera-se que sejam recebidos de 2X em 2X tempo (um valor que teria de ser ajustado manualmente num cenário real). Se um pacote de ping do outro servidor não for recebido, então há um "retry", isto é, uma nova espera de mais 2X segundos por outro pacote. Se também durante este intervalo nenhum pacote UDP for recebido, assumimos que a falha não é transiente e concluímos que há uma quebra ou na nossa ligação, ou na ligação do servidor remoto.

Para distinguir entre uma quebra na nossa ligação e uma quebra no servidor remoto, faz-se um ping ao servidor RMI. Se for possível fazê-lo, então concluímos que foi o outro servidor que perdeu a sua conectividade. Neste caso, se formos o servidor secundário, assumimos o papel de primário. Se formos o primário, mantemo-nos nesse estado e passamos a saber que existe um servidor secundário a correr.

Por não estar nos objectivos do projecto, os nossos servidores nunca assumem que apenas a ligação entre eles foi quebrada, isto é, que ambos podem estar perfeitamente contactáveis por RMI e clientes, mas incapazes de comunicarem um com o outro. Para resolver este problema, poderíamos manter o estado de cada servidor no RMI, mas por constrangimentos de tempo na resolução do projecto optámos por não implementar esta solução.

4.3 Ligação Servidor-Servidor RMI (Java RMI)

A maior parte da lógica do software é mantida no servidor RMI, acessível através de Java RMI. Por questões de facilidade de organização e rapidez de programação do projecto, existe uma interface apenas, que nos fornece todas as funcionalidades do servidor RMI. Acreditamos que uma maior modularização das responsabilidades, com várias interfaces e objectos remotos beneficiaria um projecto profissional e facilitaria a manutenção do projecto. No entanto, a nossa solução parece-nos funcional e plausível para o tempo disponível e a diversidade de tarefas necessárias.

4.3.1 Falhas do servidor RMI

A existência de apenas um objecto remoto permite-nos encapsular o acesso a este mesmo objecto numa só classe RMIConnection. Esta classe garante-nos que a interface RMI que nos disponibiliza é válida, executando um método simples no RMI (uma espécie de método 'ècho") sempre que é necessário um acesso ao servidor RMI. No entanto, caso, mesmo assim (num evento altamente improvável), uma falha na ligação ao servidor RMI seja detectada fora do âmbito desta classe, esta é notificada.

Relembrando que o acesso ao servidor RMI é sempre precedido por uma ciclo até que a ligação ao mesmo seja correctamente estabelecida, percebemos como as falhas transientes do servidor RMI são transparentes para o utilizador. De facto, quando é detectada uma falha na ligação ao servidor RMI, são feitas 3 tentativas, espaçadas por um tempo proporcional ao número da tentativa de reestabelecimento da ligação. Se estas tentativas tiverem sucesso, a ligaçãp é reestabelecida e o utilizador não se apercebe da falha transiente. Se não forem, então o servidor quebra a ligação com todos os clientes, indicando-lhes que, dada a falha de ligação ao servidor RMI, está inoperacional.

5 Arquitectura da Transmissão de Ficheiros

Para realizarmos a transmissão de ficheiros da aplicação cliente para o RMI, e em sentido inverso, optámos por implementar uma classe à qual chamámos 'NetworkingFile'.

Esta classe, que também recorre a métodos disponibilizados pela classe 'RandomAccessFile', permite-nos receber e enviar um ficheiro através de um stream de dados.

Desta forma, conseguimos enviar um ficheiro em conjunto com a informação necessária para criar uma ideia, da aplicação cliente até ao servidor RMI, e do servidor RMI até à aplicação cliente, onde de seguida o ficheiro é armazenado localmente (no formato de um ficheiro binário no servidor RMI e no formato original na aplicação cliente).

O envio dos dados é garantido através da nossa implementação da classe 'NetworkingFile' e da implementação presente na classe 'RandomAccessFile'. A entrega dos dados e a detecção de erros é garantida quer pelo protocolo TCP utilizado nas comunicações entre a aplicação cliente e o servidor (primário e secundário), quer pelo mecanismo por nós utilizado na comunicação entre as diferentes entidades (que já descrevemos no quarto ponto deste relatório).

Esta classe actua como uma classe que encapsula a tarefa de ler todos os conteúdos de um ficheiro para o array e a sua escrita. Sendo um objecto serializável, garante a sua fácil transmissão entre cliente, servidor e servidor RMI.

6 Total Order Implementation

No que respeita à garantia de ordem na realização das operações demos especial atenção às operações cuja realização condiciona e afecta o resultado de outras operações que se realizem depois destas. É o caso da compra de acções e da alteração do preço por share de uma ideia.

Assim, quando é comunicado um pedido ao servidor (primário ou secundário) cuja realização condiciona e afecta o resultado de outras operações que se realizem depois destas, este é inserido numa fila (queue) presente no servidor RMI (chamada "Request Queue"), e que contém todos os pedidos desse tipo, bem como o método remoto do servidor RMI a invocar em cada situação e os seus parâmetros de chamada, a par de um Timestamp que é utilizado para ter a garantia absoluta do ordenamento dos pedidos.

Assim, existe uma Thread responsável por verificar a fila por pedidos que ainda não tenham sido processados. Executa-os por ordem (invocando os métodos RMI) e armazena o resultado. A função que fez o pedido à Request Queue é responsável por obter o seu resultado e, quando tiver terminado a operação com o utilizador, retirar este pedido da fila. Deste modo, o nosso mecanismo permite que, apesar de algum pedido já ter sido satisfeito, mesmo que a sua resposta ainda não tenha atingido o cliente, ele continue na fila e possa atingir o cliente mesmo depois de um reinício total da infraestrutura. Correctamente utilizado, este mecanismo permite a execução de pedidos duplicados, bastando, para isso, que os métodos que adicionam pedidos à fila o façam apenas se não existir já esse mesmo pedido na fila. É esta a chave para garantir a idempotência que implementámos e que apenas falhará se a fila não for correctamente armazenada no RMI (o que pode acontecer tanto se a ligação servidor-RMI falhar mesmo após a adição do pedido, como se o RMI tiver um crash).

Assim, como mencionámos, antes de ser executado um método remoto no servidor RMI, é consultada a fila a fim de se aferir qual a próxima operação a realizar. No caso de operações que não afectam a realização de outras a cha-

mada ao método remoto é feita automaticamente, sem ser realizada qualquer interacção com a fila (pois não há necessidade de garantir idempotência)

Como cada novo pedido que é colocado na fila está a ser inserido ordenadamente de acordo com o momento em que chegou ao servidor, conseguimos garantir uma ordem na realização dos pedidos feitos pelos utilizadores, garantindo essa mesma ordem no caso do servidor primário falhar: Em caso de falhar do servidor primário, todos os utilizadores que estavam ligados a ele passam a ligar-se ao servidor secundário.

Se não implementássemos esta fila poderíamos ter pedidos de utilizadores a serem executados primeiro, quando na realidade, existiam outros à espera de serem executados e que tinham chegado ao servidor primeiro. Para além disso, caso o servidor principal falhe, como todos as aplicações cliente se vão ligar ao servidor secundário, não faria sentido estas voltarem a enviar os seus pedidos novamente, porque assim não teríamos garantias de que estes pedidos chegariam ao servidor secundário pela mesma ordem que chegaram ao servidor principal, antes de este falhar, nem tão pouco garantias da idempotência das operações cruciais.

7 Exception Handling

O tratamento de excepções que fazemos assenta-se na ideia de que o melhor a fazer é apanhar as excepções o mais depressa possível, por forma a evitar bugs (entendemos que as excepções mudam bruscamente o fluxo do código e introduzem complexidade desnecessária a um programa). No entanto, é graças ao mecanismo de excepções que somos capazes de detectar que uma ligação TCP foi quebrada (quer cliente-servidor, quer servidor-cliente), que a ligação ao RMI foi perdida, que um pedido à base de dados não foi satisfeito, que o utilizador inseriu inputs inválidos ou até que um dado ficheiro ainda não existe e tem de ser criado.

No caso das excepções relativamente a acessos de leitura e escrita de sockets, levam.nos à detecção de uma falha na ligação entre o cliente e o servidor e consequente término da ligação entre os mesmos.

No caso das excepções no acesso ao servidor RMI, utilizamos o mecanismo de retry e de quebra de ligação com todos os clientes que já antes abordámos.

7.1 Reconecção do cliente ao servidor em caso de falha e auto-re-login

A reconecção do cliente ao servidor em caso de falha, como já documentado, é conseguida através dos métodos connect() e reconnect() da classe ClientConnection. Esta reconecção, caso o user esteja logado, garante ainda o auto-relogin automático, sem que o utilizador se aperceba disso.

7.2 Reconecção do servidor TCP ao RMI em caso de falha e falhas do RMI

Novamente, como já abordado, o servidor TCP possui um mecanismo de reconecção ao RMI que lida com as falhas transientes de forma transparente ao cliente e, apenas em caso de uma falha permanente, desconecta todos os clientes.

7.3 Ideias duplicadas

O nosso sistema de "Request Queue", a par do nosso funcionamento de reconnect() do cliente garante que não serão criadas ideias duplicadas. De facto, a partir do momento em que um cliente envia os dados relativos à criação de uma ideia ao servidor, o pedido de criação da ideia é armazenado na "Request Queue" e todos os pedidos seguintes, mesmo que reenviem a informação, nunca serão recolocados na fila. A fila executa uma e uma só vez este pedido, estando armazenada no servidor RMI para garantir que a sua consistência no caso da falha de um servidor. Destaquemos ainda que os itens podem ficar na fila após processados e só são removidos quando o cliente recebe a confirmação da correcta adição da sua ideia. É após este momento que são removidos os pedidos da fila e futuras tentativas de adicionar uma ideia já serão novamente possíveis.

8 Manual de Instalação e Configuração

Os ficheiros de código fornecidos podem ser compilados com o script build.sh em sistemas Unix. De seguida, os scripts *run_client.sh*, *run_server.sh* e *run_rmi_server.sh* permitem executar o cliente, o servidor e o servidor RMI. Se estes scripts forem executados sem parâmetros, então assumem que todos vão ser executados na mesma máquina e assumem certas configurações por defeito. No entanto, os scripts podem receber parâmetros:

run_client.sh (host 1) (porta comandos host 1) (porta notificações host 2) ... (host N¿ (porta comandos host N) (porta notificações host N)

Este script permite, portanto, um número variável de hosts, cada um com uma porta de comandos e de notificações especificada.

run_server.sh (porta de comandos) (porta de escuta pings UDP) (porta de envio pings UDP) (hostname do outro host) (porta de notificações) (hostname do servidor RMI) (campo que a existir indica que o servidor deve iniciar como secundário)

Este script permite definir a porta de escuta de comandos, a porta de escuta de pings UDP, a de envio de pings UDP, a de escuta de notificações, o hostname do outro servidor, do servidor RMI e ainda um campo opcional que, a existir (com qualquer valor) indica que este servidor deve iniciar-se como secundário.

run_rmi_server.sh (hostname da BD)

Este script possibilita a escolha do hostname da Base de Dados, assumindo-se que executa na porta 1521 e que existe um utilizador com o username "sd" e password "sd".

Ainda que possam ser executadas por qualquer ordem, a ordem preferencial de arranque das aplicações é:

1. Servidor RMI
2. Servidor Primário
3. Servidor secundário
4. Cliente

As tabelas da base de dados podem ser criadas com os scripts fornecidos: 'criaTabelas.sql' e 'insereDados.sql'.

9 Testes Realizados

Teste	Estado
Servidor e cliente correm em máquinas diferentes	Concluído com Sucesso
Registo e Login	Concluído com Sucesso
Criar Tópico	Concluído com Sucesso
Listar Tópicos	Concluído com Sucesso
Listar Ideas pertencentes a um tópico	Concluído com Sucesso
Criar uma ideia (com relações a favor e contra outras ideias)	Concluído com Sucesso
Remover uma ideia	Concluído com Sucesso
Comprar shares de uma ideia (com notificação instantânea para os vendedores online)	Concluído com Sucesso
Transactional trading	Concluído com Sucesso
Definir preço de venda de shares de uma ideia	Concluído com Sucesso
Mostrar histórico de transações	Concluído com Sucesso
Receber notificações de transações anteriores aquando do login	Concluído com Sucesso
Criar uma ideia (com anexo de ficheiro binário)	Concluído com Sucesso
Cliente tenta ligar-se ao mesmo servidor em caso de falha temporária do Servidor TCP Primário	Concluído com Sucesso
Falhas temporárias são transparentes para o utilizador final	Concluído com Sucesso
A aplicação cliente não perde a ideia a ser criada	Concluído com Sucesso
O servidor TCP volta a ligar-se caso perca conectividade ao servidor RMI	Concluído com Sucesso
Falhas temporários do servidor RMI são transparentes para o utilizador final	Concluído com Sucesso
Não existem ideias duplicadas em "handover scenarios"	Concluído com Sucesso
Pings UDP entre os servidores primários e secundários	Concluído com Sucesso
Distinção entre uma falha permanente e uma falha temporária	Concluído com Sucesso
Servidor secundário é activado quando o servidor primário falha	Concluído com Sucesso
Dados recebidos dos servidores TCP são os mesmos	Concluído com Sucesso
"Fail-over" é transparente para o utilizador	Concluído com Sucesso
Novos utilizadores vão utilizar o novo servidor primário	Concluído com Sucesso
Quando um servidor primário que tinha falhado retoma a sua actividade, passa a assumir o papel de servidor secundário	Concluído com Sucesso
As ordens de compra de shares devem ser rigorosas, mesmo com "handover"	Concluído com Sucesso