Universidade do Minho

Mestrado Integrado de Engenharia Informática

Tolerância a Faltas

Serviço Distribuído de Escalonamento de Tarefas

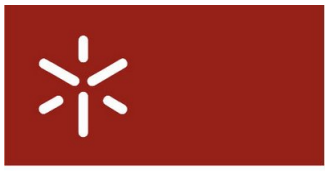
Autores:

A74817 Marcelo António Caridade Miranda

A75428 Bruno Miguel Sousa Cancelinha

A74576 José António Dantas Silva

5 de Setembro de 2018



Universidade do Minho

1 Introdução

Foi-nos proposto, no âmbito da unidade curricular de Tolerância a Faltas, a implementação de um sistema distribuído de escalonamento de tarefas; onde cada uma destas tarefas é identificada por uma URL única. Tal sistema deve garantir que a ordem de chegada das tarefas é respeitada, bem como garantir que todas as tarefas são executadas (mesmo que numa ordem diferente). Estas propriedades devem ser respeitadas mesmo quando as réplicas dos servidores ou os clientes falham.

O relatório que se segue vem expor os problemas que emergem dum sistema como este, propor soluções e, finalmente, apresentar uma implementação em JAVA usando o protocolo de comunicação em grupo *Spread*.

2 Arquitetura da solução

O serviço a desenvolver deve garantir a conclusão de todas as tarefas, suportando a falha de até f servidores desde que haja sempre pelo menos 1 cliente ativo. Sempre que um dos clientes falha, as tarefas não terminadas devem ser atribuídas a outro cliente. A arquitetura do sistema é composta por vários servidores e clientes que interagem através de um protocolo de comunicação de grupo fornecido pelo *Spread Toolkit*, terminando assim com a arquitetura do sistema tal como representada na Figura 1.

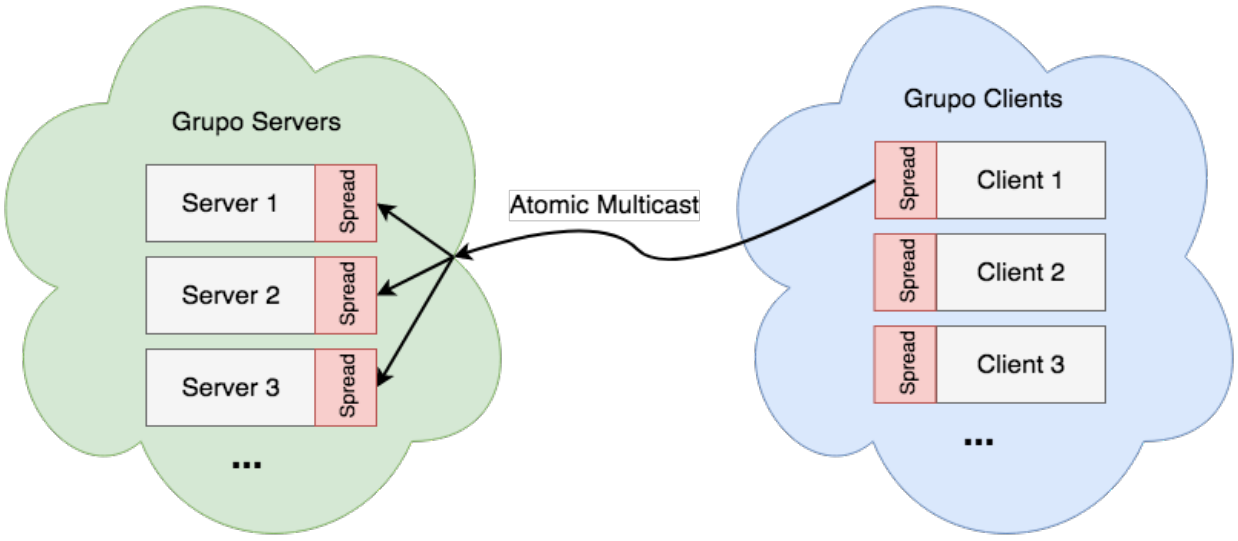


Figura 1: Visão geral da arquitetura do sistema

2.1 Servidor

O servidor tem como objetivo responder aos pedidos efetuados pelo cliente, sendo responsável por armazenar, atribuir e terminar tarefas. Perante a existência de vários servidores para garantir redundância e assegurar tolerância a faltas, é necessário lidar com problemas como a consistência dos dados entre as várias réplicas.

Para suportar a falha de servidores e sendo o serviço fornecido de caráter determinista, decidimos utilizar o método *replicação ativa* devido à simplicidade da sua implementação. Ao efetuar uma operação, o cliente envia o pedido para todos os servidores ativos e aguarda pela primeira resposta.

2.1.1 Transferência de estado

O servidor é inicializado em modo *inativo*, não processando pedidos de clientes. Após juntar-se ao grupo *servers*, este recebe uma mensagem *MembershipInfo* que indica quais os servidores que também se encontram naquele grupo. Se o servidor é o único no grupo, este passa a *ativo* e começa a processar pedidos. Caso contrário, se existem já outros servidores no grupo, é necessário obter o estado mais atual antes de começar a processar pedidos dos clientes.

Para requisitar o estado, o servidor *s1* envia um *StateReq* para o grupo *servers* utilizanto a primitiva *total order multicasting*. Ao receber a mensagem, os restantes servidores do grupo partilham o seu estado (*StateRep*) através do grupo privado de *s1*. Entre o processamento do *StateReq* e *StateRep* podem ocorrer novos pedidos. Assim sendo, ao receber o seu próprio *StateReq* o servidor *s1* deve começar a guardar tais pedidos para futuro processamento. Devido ao uso da primitiva *total order multicasting* temos garantias que o estado recebido não vai refletir estes novos pedidos.
Ao receber o primeiro *StateRep*, *s1* atualiza-se e processa todos os pedidos que tenha acumulado. Por fim, marca-se como *ativo* e começa a processar os pedidos normalmente, ignorando os restantes *StateRep* recebidos dos outros servidores.

2.2 Cliente

O cliente é inicializado juntando-se ao grupo *clients*, grupo onde estão inseridos os clientes do sistema, podendo a partir desse momento efetuar pedidos aos servidores.

Os pedidos emitidos pelos clientes são enviados para o grupo de servidores utilizando a primitiva *total order multicasting*, a qual garante que todos os servidores ativos recebem os pedidos pela mesma ordem. Desta forma garantimos a consistência na execução dos pedidos entre os servidores. Esta primitiva é fornecida pelo *spread* através do *delivery method 'agreed'*.

Os pedidos efetuados aos servidores por parte de um cliente são síncronos e registados com

uma *tag* única (**reqID**), para que possamos corresponder as respostas por parte dos servidores. Tal se deve à implementação de replicação ativa na qual todos os servidores respondem ao cliente sendo necessário garantir a transparência, obtida com o incremento do *reqID* a cada operação do cliente, não interpretando as mensagens repetidas.

2.2.1 Falha do cliente

Existe a necessidade perante a falha de um cliente da re-atribuição das suas tarefas a outros clientes. Como descrito anteriormente, um cliente ao iniciar junta-se ao grupo *clients* passando a receber informações relativamente às mudanças nesse grupo através de mensagens *MembershipInfo*.

Nestas mensagens podemos verificar a origem e causa da mudança no grupo. Verificando os casos de *Disconnection* ou *Leave*, os restantes clientes são responsáveis por informar os servidores da saída através de um pedido *ReallocateTasks*, a qual contém informação relativa ao cliente que falhou. O servidor processa esta mensagem movendo todas as tarefas atribuídas ao cliente que falhou para o inicio das tarefas pendentes.

3 Conclusão

Terminado o trabalho, o grupo ficou surpreendido com a facilidade que as funcionalidades do *spread* trouxeram na resolução dos problemas. Depois de bem equacionado, resolvido o problema da transferência de estado e da falha dos clientes, a implementação em JAVA foi trivial. Conseguimos, com sucesso, construir um sistema distribuído tolerante a falhas que cumpre os pré-requisitos.