Licenciatura em Engenharia Informática e de Computadores

SISTEMAS DISTRIBUÍDOS, VERÃO 2016/2017

RELATÓRIO DO 1º TRABALHO PRÁTICO DE AVALIAÇÃO

GRUPO 3

41839 – ANDRE CARVALHO

41482 – GONÇALO VELOSO

39134 – RÚBEN TABORDA

ÍNDICE

ÍNDICE DE FIGURAS

INTRODUÇÃO

No âmbito da unidade curricular de Sistemas Distribuídos foi proposto o desenvolvimento de sistemas distribuídos usando objetos distribuídos na plataforma .NET.

Neste documento é descrita como foi concebida a solução para o problema proposto.

O cenário apresentado na Figura 1 é caracterizado por uma rede de múltiplos servidores, onde poderão ser guardados pares (Chave, Valor) emitidos pelos diversos clientes. Os servidores (Server)

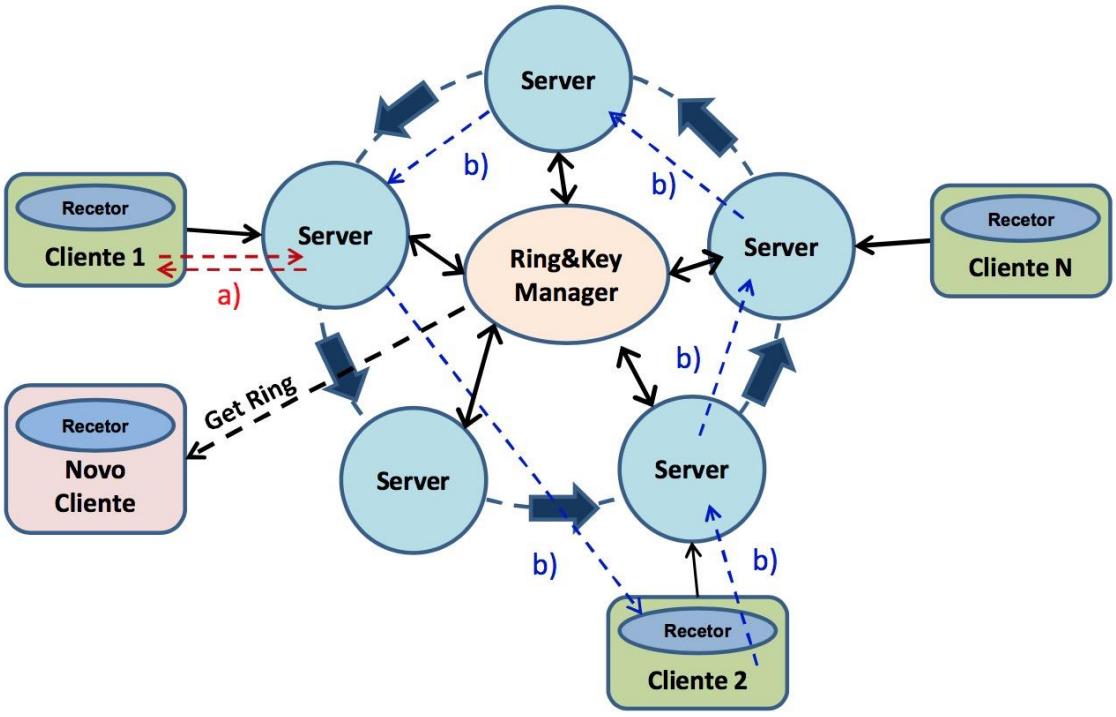

Figura 1 – Esboço geral do sistema

estão ligados em anel a um servidor central.

ARQUITECTURA DO SISTEMA

Apesar de ser explícito no enunciado que a solução pretendia que existisse 5 servidores, graças a ficheiros de configuração é possível ter N servidores.

Contudo existem algumas restrições de implementação:

- Cada servidor pode ser conectado a mais do que um cliente, caso o número de clientes seja superior ao número de servidores disponíveis.
- Cada Servidor tem uma ligação indireta com os restantes Servidores, formando assim um anel.

REGISTO DE UMA PAR CHAVE-VALOR

Quando um cliente efetua a operação de push de um par-valor para o servidor associado, este irá efetuar as seguintes operações:

1. Colocar um pedido ao Manager, a perguntar se a chave já existe nalgum servidor.
2. Caso a chave não exista, este irá guardar a chave nele mesmo. Depois disso irá colocar um pedido ao Manager a pedir que ele ordene os servidores adjacentes a realizarem uma cópia.

ELIMINAÇÃO DE UMA PAR CHAVE-VALOR

Quando um cliente efetua a operação de delete de um par-valor para o servidor associado, este irá efetuar as seguintes operações:

1. Apagar a informação local, se esta existe.
2. Enviar um pedido ao servidor central para que este apague a informação correspondente de outros servidores.

Ao receber o pedido, o anel efetua as seguintes operações:

1. Verifica na lista local de chaves quais os servidores que possuem informação para essa chave.
2. Pedir a esses servidores que apaguem essa informação.
3. Apagar essa chave da sua lista de chaves.
4. Retornar com sucesso ao servidor que efetuou o pedido ou lançar exceção em caso de falha. O servidor por sua vez faz o mesmo ao cliente que efetuou o pedido.

ESTRUTURA E DESCRIÇÃO DA SOLUÇÃO

Neste capítulo descrevemos a estrutura adotada pelo grupo. Temos a plena noção que existem outras soluções para o problema, mas falaremos apenas da nossa solução, sabendo que em engenharia informática não existem soluções perfeitas.

A estrutura da nossa solução está organizada em sete projetos embebidos numa solução. Tentámos separar a parte lógica de negócio com a parte da user interface. Temos um projeto de nome, Interfaces, que corresponde à uma biblioteca com todas as interfaces necessárias para a solução. O assembly desta biblioteca é partilhado pelo manager, pelo servidor como pelo cliente. Os projetos com o nome ServerClass, ManagerClass e ClientClass tratam da lógica de negócio da solução, implementando as interfaces da biblioteca referida anteriormente. Os projetos Cliente, Server e Ring são referentes à user interface respetivamente do cliente, do server e do RingManager.

De seguida é apresentado uma descrição mais pormenorizada de cada projeto e as relações existentes em cada um deles.

COMUNICAÇÃO ENTRE SERVIDORES

Durante o desenvolvimento do trabalho considerámos 3 possibilidades que vamos enumerar agora e descrever brevemente. Nestes esquemas, o círculo Ring representa o servidor central enquanto cada servidor ServerX corresponde ao servidor número X. As setas a tracejado representam as ligações entre os servidores e as setas normais representam uma tentativa de ligação.

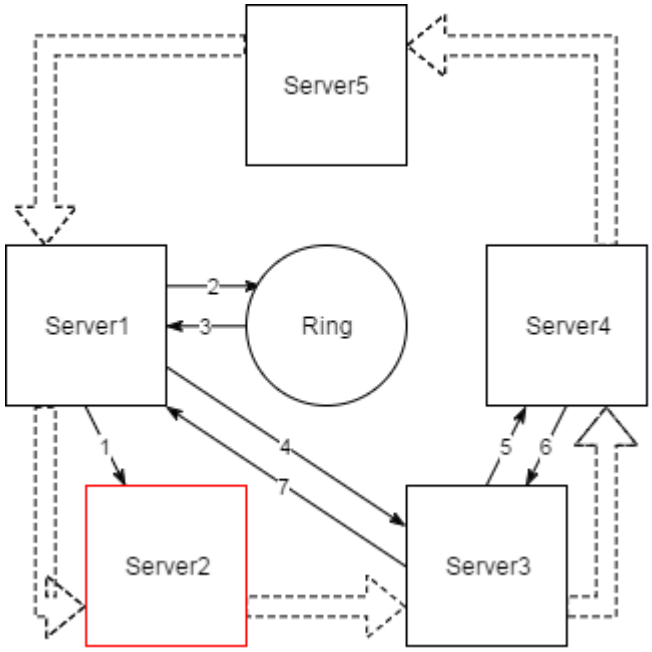

Figura 2.a - Comunicação tipo 1

Nesta primeira implementação, quando a comunicação entre servidores falha (entre o Server1 e o Server2, através de seta 1), o servidor que efetuou o pedido falhado ao próximo servidor pede ao Ring para lhe fornecer o próximo servidor na sequência (setas 2 e 3). A sequência é continuada normalmente a partir deste ponto, assumindo que mais nenhum servidor falha, nesse caso o processo de pedir ao Ring é repetido. A vantagem desta estrutura é a independência que os servidores têm do ring exceto quando ocorrem erros. A desvantagem é a necessidade de circular a lista de servidores para verificar em quais servidores existe uma determinada chave.

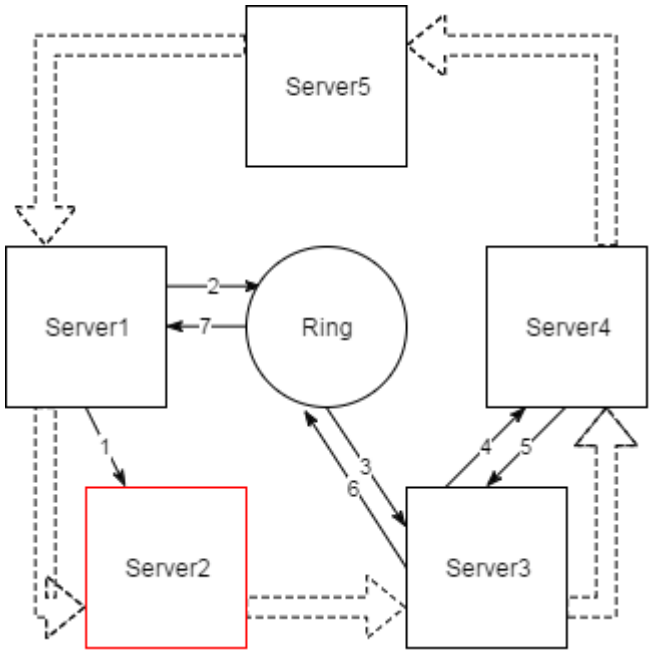

Figura 2.b - Comunicação tipo 2

O segundo caso é semelhante ao primeiro. A diferença significativa é que, em vez de pedir ao Ring um novo servidor, o servidor original pede ao Ring para processar o pedido por si. O Ring irá, portanto, efetuar esse pedido ao próximo servidor disponível e esse próximo servidor irá continuar a sequência. Devido às semelhanças com a primeira implementação, as desvantagens e vantagens desta implementação são as mesmas.

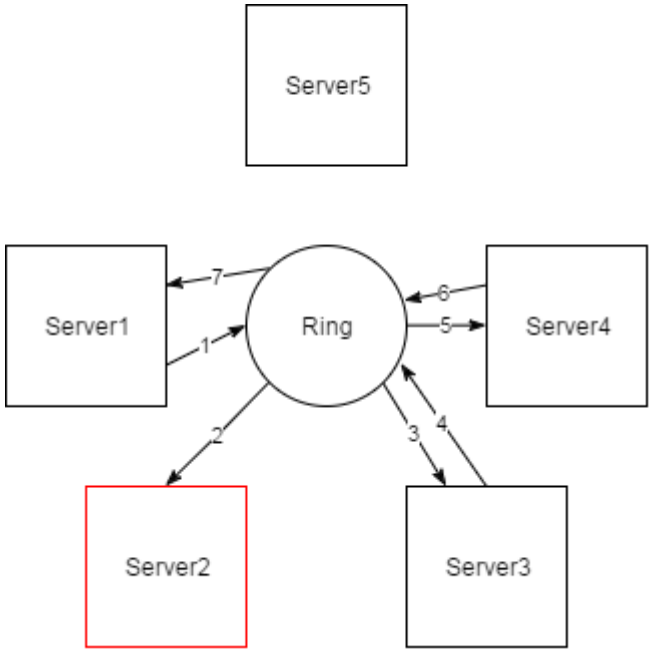

Figura 2.c - Comunicação tipo 3

Esta última implementação é a que escolhemos usar. Um servidor não conhece os outros servidores, portanto pede sempre ao Ring para efetuar o pedido por si. O Ring contém uma lista de servidores disponíveis assim como as chaves contidas nesses servidores. Desta forma o Ring sabe exatamente a quais servidores efetuar os pedidos. Esta é a principal vantagem: a capacidade de saber a quais servidores da lista efetuar o pedido permite evitar a pesquisa dos servidores todos. A desvantagem é o possível peso adicional colocado no Ring.

SERVER

Os servidores, após estabelecerem ligação com o Ring e com um ou mais clientes, estão encarregados de receber pedidos do cliente, processa-los localmente e envia-los para o Ring. Todos os pedidos possíveis correspondem a alterações locais e alterações remotas, excepto o método **deleteKeylocally** que apenas é chamado pelo Ring para apagar a chave localmente e não propagar essa informação, visto que o Ring já se está a encarregar disso quando chama esse método.

RING&KEY MANAGER

O Ring&Key Manager, estabelece uma conexão entre ele e um servidor contém a informação sobre as chaves atribuídas e a que servidores estas estão associadas. Contém métodos de verificação, obtenção e tratamento de falhas por parte dos servidores.

O método **checkIfKeyExists** trata de verificar se para o servidor desejado existe a chave.

O método **ReplicateInformationBetweenServers** trata de verificar se é possível a replicação no servidor +1, +2 e trata as falhas destes porque se um destes falhar replica para o seguinte, mantendo sempre 2 réplicas para além do original. Para tal este método exige o uso do método **storePairLocally** para guardar os pares pelos seguintes servidores.

O método **deleteInformation** como o nome indica serve para eliminar os pares chave/valor para os servidores que contiverem a chave a ser eliminada, este usa o método **deletePairLocally** para cada servidor que contenha tal chave.

O método **searchServersForObject** procura pelo servidor que contenha a chave desejada e encarrega-se por obter o valor desta.

TEMPO DE VIDA DOS OBJECTOS

No decorrer da realização do trabalho optamos por estabelecer que todos os objetos servidores seria do tipo Singleton, uma vez que estes iriam partilhar informação entre si. O caso mais notável neste trabalho é o objecto de Manager, que iria guardar uma lista de todas as chaves já guardadas. Caso este fosse Single-call a abordagem optada não seria possível realizar, visto que assim cada servidor teria acesso a um Manager com chaves únicas e não partilhadas.

TRATAMENTO DE EXCEPÇÕES

No trabalho encontra-se implementado a gestão de exceções a quando a realização de um store. Caso o servidor n + 1, ou o servidor n + 2, se encontrarem indisponíveis o Manager continuará a propagar o pedido de store, para os diferences servidores, até encontrar dois servidores disponíveis. Caso não exista servidores disponíveis, ou este dê a volta completa ao anel este retornará a resposta de que não é possível realizar o store.

CONCLUSÃO

Optamos por uma solução sem muitos requisitos funcionais para concentrar o nosso trabalho maioritariamente nos requisitos não funcionais.

Mesmo assim tivemos partes que não conseguimos implementar como o dinamismo do anel e não conseguimos garantir todas as falhas que possam ocorrer estão tratadas na nossa solução.