3º Trabalho Prático de Avaliação

Nuno Veloso 42181
Steven Brito 42798
Daniela Gomes 42799

12 de Junho de 2018

Conteúdo

Capítulo 1

Introdução

Com o intuito de realizar o 3º trabalho prático da unidade curricular Sistemas Distribuídos pretende-se implementar um sistema distribuído baseado em serviços Windows Comunication Foundation (WCF) usando a plataforma .NET. Este sistema terá de ser capaz de suportar armazenamento de dados seguindo o paradigma *Key-Value store*. Este paradigma consiste em guardar um valor e obter uma chave associada ao valor guardado, podendo ser possível obter novamente o valor dado essa chave.
Um desafio deste trabalho será a implementação de diferentes clientes dos serviços WCF em vários ambientes virtuais de execução, e.g. JVM, .NET.
Ao longo deste trabalho irá ser aplicado os conceitos aprendidos nas aulas. Será descrito e discutido as vantagens, os problemas e os desafios que se colocam no desenvolvimento deste sistema distribuído.

Capítulo 2

Descrição do Problema

O problema consiste em desenvolver um sistema distribuído capaz de guardar dados independentes dos utilizadores e associar a uma chave para poder aceder novamente aos dados. Este sistema deve garantir redundância, pelo que deve guardar os dados em dois ou mais serviços. A chave para os dados que foram armazenados deve evitar colisões e conter informação sobre a localização das réplicas de forma a aumentar o desempenho no acesso aos dados. Os dados que irão ser armazenados devem ser distribuídos de forma uniforme entre os vários serviços de armazenamento para evitar a sobrecarga dos mesmos. É também necessário garantir que os vários acessos concorrentes aos servidores mantenham a consistência dos dados. O sistema desenvolvido deve ter em conta tolerância a falhas, assim como a sua escalabilidade.

Capítulo 3

Requisitos

Neste capitulo é descrito os requisitos funcionais do sistema, assim como os não funcionais.

3.1 Funcionais

Os requisitos funcionais deste sistema são os seguintes apresentados:

- Evitar sobrecarga nos servidores de armazenamento;
- O cliente comunica apenas com um serviço intermediário (*broker*);
- O cliente conhece todos os servidores *brokers*;
- O cliente pode escolher a qual *broker* se quer ligar;
- O cliente pode guardar, obter e apagar dados;
- Os *brokers* devem ser *stateless*;
- Os *brokers* têm conhecimento de todos os servidores de armazenamento;
- Os *brokers* implementam um algoritmo de escolha dos servidores de armazenamento de forma a evitar a sobrecarga;
- Os servidores de armazenamento devem suportar qualquer tipo de dados independentemente das aplicações cliente;
- Os servidores de armazenamento mantêm dados em memória;
- As chaves geradas pelos *brokers* devem conter informação da localização dos dados.

3.2 Não Funcionais

Os requisitos não funcionais deste sistema são os seguintes apresentados:

- Aplicação cliente realizada em WinForm (.NET);
- Aplicação cliente realizada em Java;
- Ter um sistema escalável horizontalmente e fiável;
- Transparência à concorrência no acesso dos servidores;

- Suportar vários canais de comunicação (e.g.: HTTP, TCP);
- Tratar as falhas de forma a que o sistema continue funcional, mesmo com a existência de erros em um dos servidores.

Capítulo 4

Arquitetura

4.1 Interação entre as partes

Um utilizador apenas interage com um *broker*. O *broker* comunica-se com o utilizador e com o servidor de armazenamento. O servidor de armazenamento apenas interage com os *brokers*.

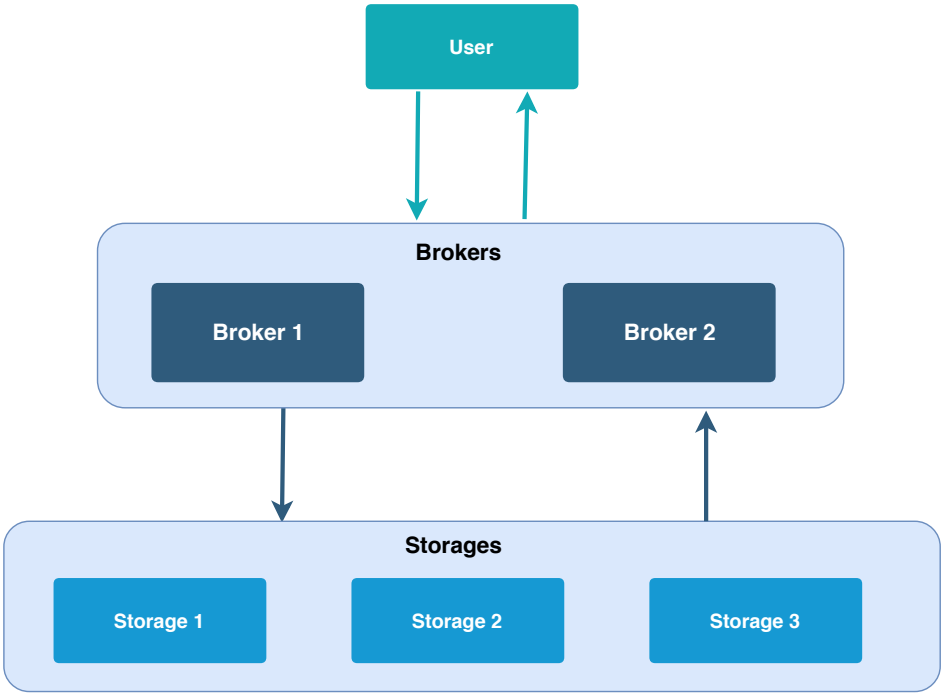

Figura 4.1: Arquitetura

O utilizador é cliente do *broker*. O *broker* é serviço do utilizador e cliente do servidor de armazenamento. O servidor de armazenamento é serviço do *broker*.

4.2 Funcionamento

Para poder utilizar o sistema, o utilizador deve escolher em qual *broker* pretende-se conectar. Estando este conectado, é possível usufruir das seguintes funcionalidades:

- Guardar dados e obter uma chave associada;

- Obter os dados através da chave associada;
- Eliminar os dados através da chave associada.

4.2.1 Funcionamento

Para um utilizador guardar um valor deve primeiro escolher qual o *broker* que deseja conectar-se. Após a escolha do *broker* o utilizador pode inserir um valor e guardá-lo. Ao guardar o valor, a aplicação cliente interage com o *broker* e este, através de um algoritmo de seleção, irá escolher qual o servidor de armazenamento com menor carga. Após guardar o valor, o *broker* gera um chave e envia à aplicação cliente. A aplicação cliente com essa chave pode obter novamente o valor ou apagar dos servidores de armazenamento.

4.3 Escalabilidade

O sistema é escalável horizontalmente. Desta forma, se um dos *brokers* ou servidores de armazenamento ficar sobrecarregado, é possível adicionar mais *brokers* e/ou servidores de armazenamento. Os *brokers* já existentes terão de ser reiniciados para reconhecer os novos servidores de armazenamento.

Capítulo 5

Implementação

A solução encontra-se dividida em várias aplicações de consola, uma aplicação WinForm e outra em Java.

5.1 KVService

Para ser implementado o servidor de armazenamento, foi necessário definir a interface IKVService, apresentada na figura 5.1.

```
[ServiceContract]
public interface IKVService
{
    [OperationContract]
    int GetCount();

    [OperationContract]
    int StoreData(string value);

    [OperationContract]
    [FaultContract(typeof(ArgumentException))]
    string RetrieveData(int key);

    [OperationContract]
    [FaultContract(typeof(ArgumentException))]
    void DeleteData(int key);
}
```

Figura 5.1: Interface IKVService

O servidor de armazenamento é *stateful* pois tem de guardar os dados de forma persistente. A forma de guardar os dados é em memória.
Para a implementação do servidor de armazenamento foi usado o padrão *singleton*. Foi usado este padrão uma vez que é preciso manter estado global. Os pedidos irão ser atendidos por threads diferentes, pelo que o *ConcurrencyMode* escolhido foi *Multiple*.

Assumindo a existência deste serviço na mesma rede com o serviço de *broker*, a comunicação é feita através do protocolo TCP. Foi escolhido este protocolo por ser mais eficiente em relação ao protocolo HTTP e como referido anteriormente, assumindo que ambos os serviços estão na mesma rede. Se mais tarde for requisito estes serviços estarem em redes diferentes, basta mudar o ficheiro de configuração e alterar o *binding* e o *base address* para suportar o protocolo HTTP.

5.2 BrokerService

Para implementar o *broker*, definiu-se a interface IBrokerService, apresentada na figura 5.2.

```
[ServiceContract]
public interface IBrokerService
{
    [OperationContract]
    [FaultContract(typeof(InvalidOperationException))]
    Key StoreData(string value);

    [OperationContract]
    [FaultContract(typeof(ArgumentException))]
    string RetrieveData(Key key);

    [OperationContract]
    [FaultContract(typeof(ArgumentException))]
    void DeleteData(Key key);
}
```

Figura 5.2: Interface IBrokerService

Os *brokers* são *stateless* por forma a facilitar o balanceamento de carga. De forma a não ter o objeto em memória foi usado o *InstanceContextMode PerCall*.
Como o modo é PerCall, ou seja, irá ser criada uma nova instância da classe *BrokerService* a cada chamada, não é necessário garantir controlo de concorrência.
Para determinar qual o servidor de armazenamento com maior disponibilidade em termos de volume de dados inseridos, o *broker* pergunta a todos os servidores de armazenamento a carga de cada um. Após ter esta informação, o *broker* ordena os servidores de armazenamento por ordem crescente do volume de dados. Irá escolher os dois primeiros para armazenar o valor nesses servidores. Se algum deles falhar o *broker* tenta guardar no terceiro servidor de armazenamento com menor volume de dados inseridos. Este processo repete-se até conseguir guardar em pelo menos dois servidores de armazenamento. Se conseguir guardar, irá construir um objeto *Key*. Caso contrário irá enviar uma mensagem de exceção a dizer que não foi possível guardar.

Para representar a chave de um valor guardado, foi criada a classe *Key* cuja definição se encontra na figura 5.3

Nesta classe é guardada o endereço dos servidores de armazenamento como forma de localização e o índice onde o valor foi guardado dentro do servidor de armazenamento.

Como foi referido na secção 5.1 é assumido a existência do *broker* na mesma rede que o serviço descrito nessa secção. O *broker* comunica diretamente com o servidor de armazena-

```
[DataContract]
public class Key
{
    [DataMember]
    public List<int> Indexes = new List<int>();

    [DataMember]
    public List<string> Storages = new List<string>();
}
```

Figura 5.3: Classe *Key*

mento, logo o *binding* usado deve ser o mesmo. Para que o serviço exposto pelo *broker* seja acessível fora da própria rede, foi usado o protocolo HTTP. O *binding* usado foi o *BasicHttpBinding* porque não é necessário manter sessão ou permitir *callbacks*.

5.3 Controlo de Concorrência

Como o servidor de armazenamento segue o padrão *singleton* irá atender os diferentes pedidos em diferentes *threads*. Como as *threads* irão aceder à mesma instância, é necessário garantir controlo de concorrência. Para garantir o acesso concorrente ao estado dessas instâncias, foi usado a biblioteca *Concurrent* do .NET.

5.4 Ficheiros de Configuração

De forma a não comprometer o código com os URLs, protocolos, implementações e *bindings*, foi usado ficheiros de configuração.

Capítulo 6

Tolerância a Falhas

Com esta arquitetura existem cenários em que um dos servidores de armazenamento possa falhar. Assim é necessário garantir que o *broker* replique os dados pelos diferentes servidores de armazenamento. A estratégia para a replicação dos dados é guardar em pelo menos dois dos servidores de armazenamento. Se não conseguir guardar em pelo menos dois o broker indica ao utilizador que não foi possível guardar o valor.

Como o *broker* é *stateless*, se este falhasse não iria prejudicar o sistema desde que houvesse mais *brokers* disponíveis.

Se o utilizador desconectar-se sem apagar os valores guardados, estes irão permanecer em memória do servidor de armazenamento. O servidor de armazenamento não tem maneira de saber a quem pertence os valores visto que não interage com os utilizadores, logo não existe forma de saber se o utilizador desconectou-se

Capítulo 7

Manual de utilização

Para usar o sistema desenvolvido, terá de executar os seguintes passos dentro da pasta Install:

1. Dentro da pasta Storages existem três pastas. Em cada uma delas deverá lançar o executável *KVStorage.exe*;
2. Dentro da pasta Brokers, existem duas pastas. Em cada uma delas, deverá lançar o executável *Broker.exe*;
3. Para executar um ou mais clientes em .NET, deverá lançar o executável *UserApp.exe* presente na pasta UserApps na sub-pasta dotNet;
4. Para executar um ou mais clientes em Java, deverá ter instalado o Java e deverá lançar o .jar *JavaClient.jar* presente na pasta UserApps na sub-paste java.

7.1 Utilização do Cliente

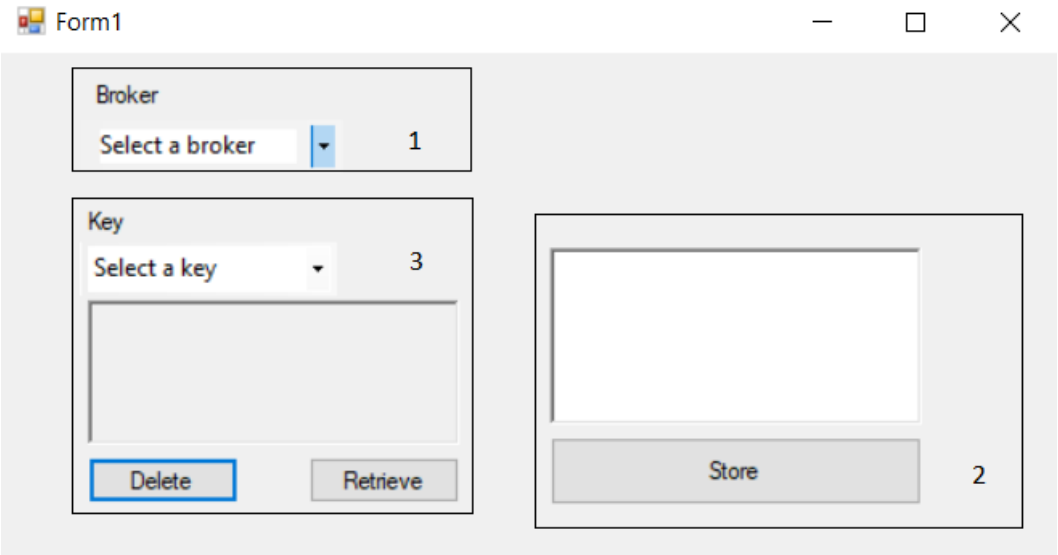

Figura 7.1: User Form

Na figura 7.1 é apresentada a interface gráfica do utilizador. Relativamente a esta figura, existem retângulos numerados. Cada retângulo é explicado de seguida.

No retângulo 1, pode selecionar o *broker* à qual o utilizador pretende-se conectar.

No retângulo 2, é possível escrever um valor para ser guardado e pressionando o botão *Store* irá comunicar com o *broker* selecionado para tentar guardar o valor.

No retângulo 3, pode selecionar uma chave e estando a chave selecionado pode pedir o valor associado a essa chave clicando no botão *Retrieve*. Ou então, pode eliminar o valor associado a essa chave clicando no botão *Delete*.

Capítulo 8

Conclusão

Com o término do trabalho, foi possível aferir os vários conhecimentos relacionados com os serviços WCF da plataforma .NET. As dificuldades encontradas estiveram relacionadas com a definição da interação entre as partes, pensar nas possíveis falhas e tentar arranjar uma melhor forma de garantir tolerância a falhas. Foi possível consciencializar os problemas encontrados ao desenvolver um sistema distribuído, tais como tratamento de falhas, concorrência e escalabilidade. A apreciação final do trabalho desenvolvido é satisfatória.