COMPUTAÇÃO DISTRIBUÍDA

Trabalho Prático de Avaliação Final

Cursos:

MEIC Mestrado de Engenharia Informática e de Computadores
MEIM Mestrado de Engenharia Informática e Multimédia

Grupo 09:

45330 Fábio Rainho
47176 João Silva
46061 Manuel Silva

Índice

Lista de Figuras

1. Introdução

Serve o presente documento para descrever os aspetos principais do desenvolvimento do trabalho final, apresentando uma descrição do sistema implementado, descrevendo o objetivo do trabalho, principais objetivos, arquitetura do sistema, configuração dos vários componentes do sistema e finalmente conclusões sobre o trabalho desenvolvido.

1.1. Âmbito

Este relatório insere-se no âmbito do trabalho final no contexto da unidade curricular de Computação Distribuída, pondo em prática os diferentes paradigmas de interação e middleware estudados e utilizados nos Laboratórios das aulas práticas.

1.2. Objetivo

Este trabalho tem como objetivo implementar um sistema distribuído para dar suporte a uma cadeia de supermercados que registam as suas vendas numa cloud, ficando assim estas vendas disponíveis para mais tarde se fazer um resumo das vendas realizadas.

1.3. Organização do Documento

O documento está organizado da seguinte forma:

- Secção 1: Introdução;
- Secção 2: Cenário;
- Secção 3: Implementação do Sistema;
- Secção 4: Configuração do Sistema;
- Secção 5: Conclusão.

2. Cenário

O cenário que se pretende implementar neste trabalho é referente a uma plataforma informática de uma cadeia de supermercados com vários componentes que se executam em múltiplas máquinas virtuais (VM) na Google Cloud Platform (Figura 1), configuradas com um sistema de ficheiros distribuídos (Gluster file System) que permite que qualquer ficheiro criado numa VM seja replicado em todas as VM.

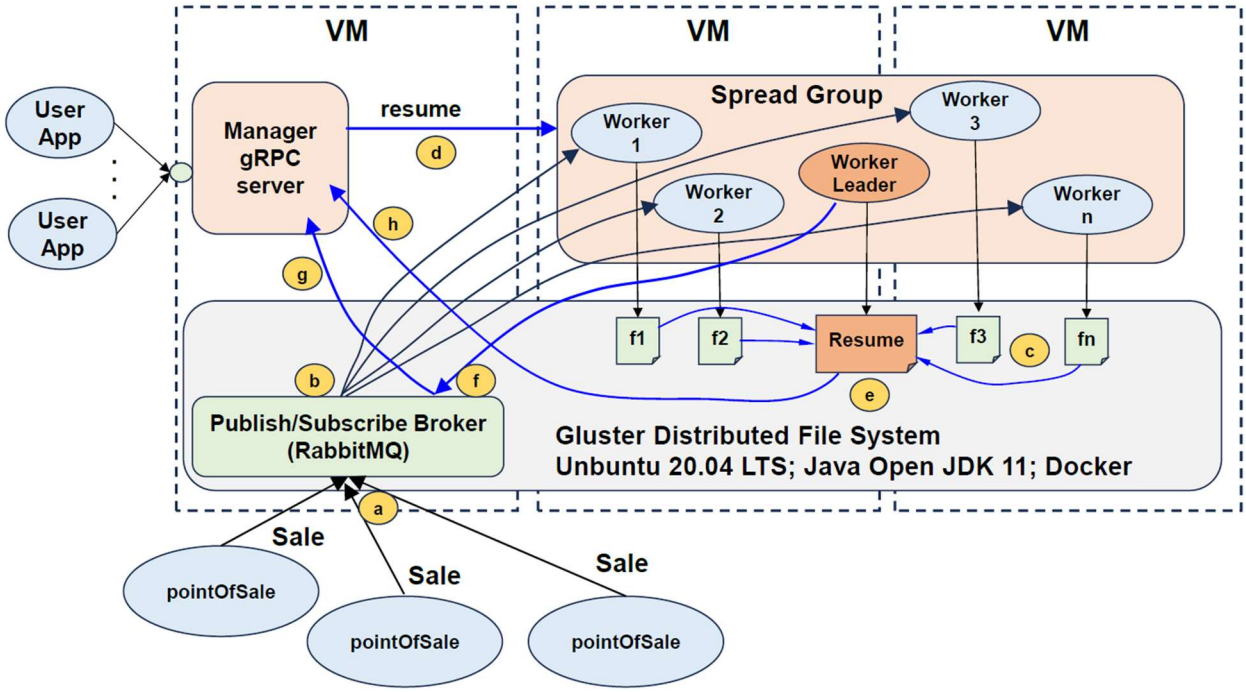

Figura 1 - Diagrama geral do sistema

3. Implementação do Sistema

Nesta secção são descritas todas as etapas realizadas durante a implementação do sistema proposto, a qual se iniciou pela configuração das VMs com os middleware e runtime necessários seguindo as instruções do guia install-configure-Vms.txt fornecido pelo docente. Alguns dos aspetos mais relevantes descritos neste relatório são os seguintes:

- Estrutura e fluxo das mensagens trocadas entre as várias partes do sistema;
- Algoritmo de eleição do worker Leader;
- Contrato UserManagerContract;

3.1. Point Of Sale

A aplicação PointOfSale, criada com o fim de simular uma caixa registadora, tem como objetivo o de enviar mensagens sobre vendas (Figura 2) que tenham sido efetuadas sobre apenas dois tipos de produtos, alimentares e de casa. Esta aplicação envia estas mensagens por eventos de rabbitMQ, para tal foi definido o exchanger "ExgSales" do tipo "Topic", a razão pelo qual foi usado o tipo "Topic" prende-se com o facto de estar a ser tirado partido dos "wildcards" que existem na "routing key" de modo a acrescentar informação sobre em que categoria dos tipos alimentar e de casa o produto vendido se insere. De forma a distinguir estas duas categorias foram então criadas duas filas no exchange referido anteriormente com os seguintes nomes e "routing keys" respetivamente, ALIMENTAR, ALIMENTAR. #, CASA, CASA.#.

```
private static void sale(Channel channel, Scanner s, String key) throws IOException {
    System.out.println("Preencha a informação do produto");
    System.out.print("Nome: ");
    String nomeProduto = s.nextLine();
    System.out.print("Subcategoria: ");
    String subcategoria = s.nextLine();
    System.out.print("Código: ");
    String codigoProduto = s.nextLine();
    System.out.print("Quantidade: ");
    double quantidade = s.nextDouble();
    System.out.print("Preço por unidade: ");
    double precoUnitario = s.nextDouble();
    System.out.print("Iva aplicado ao produto: ");
    int iva = s.nextInt();
    Sale sale = new Sale(codigoProduto, nomeProduto, quantidade, precoUnitario, iva);
    channel.basicPublish(EXCHANGE_NAME, key + subcategoria, true, null, toBytes(sale));
}

private static byte[] toBytes(Sale sale){
    Gson gson = new GsonBuilder().create();
    String jsonString = gson.toJson(sale);
    System.out.println(jsonString);
    return jsonString.getBytes(StandardCharsets.UTF_8);
}
```

Figura 2 – Simulação de uma Venda

De forma a facilitar a realização de testes foi implementada a opção de simular o envio de múltiplas vendas, tanto para o tipo ALIMENTAR como para o tipo CASA, enviando assim 10 vendas pré-definidas no sistema.

3.2. Worker

A aplicação Worker tem duas funcionalidades distintas. A primeira é processar os pedidos enviados para o rabbitMQ pelos POSs e escrever estas mensagens em ficheiros, tendo cada Worker um ficheiro próprio onde realiza a escrita. A segunda é, quando receber uma mensagem de "StartResume" pelo Spread, um dos Workers junta todos os ficheiros num ficheiro e envia a sua localização pelo RabbitMQ para o servidor "Manager GRPC Server".

Para escolher o Worker que faz o resumo, é usado um algoritmo de eleição que elege como líder o primeiro Worker a entrar no grupo. Caso este Worker saia, o novo líder é o Worker cujo nome for o primeiro alfabeticamente. Todos os membros que entram depois deste vão pedir, com a mensagem "GetLeader", informação sobre quem é o líder, que irá responder com "Leader=[nome]". Como o Spread disponibiliza uma forma de saber todos os membros de um grupo, cada um dos membros verifica se é ou não o líder. O que for líder envia uma mensagem "Leader=[nome]" para informar que ele é o líder.

No que toca ao resumo, este é iniciado ao receber a mensagem "StartResume/{exchangeName}/{bindingkey}", sendo que o "exchangeName" é o nome do Exchange para onde será enviada a mensagem a informar que o resume está feito ao Manager, usando a "bindingkey". Isto bloqueia todos os Workers, fechando a receção de mensagens e os ficheiros. Caso não exista líder, é feita uma eleição. Caso exista, ou após a eleição, o líder lê todos os ficheiros e junta-os num "resume.txt", sendo que após ler cada um dos ficheiros elimina-o. Quando o resumo é terminado, envia uma mensagem para o Manager com a localização do resumo, e uma para os Workers para sinalizar o reinício da receção de mensagens. Todas as mensagens enviadas pelo Spread estão apresentadas na Figura 3.

```
if(message.contains("StartResume")){
} else if (message.contains("GetLeader") && amILeader){
} else if (message.contains("Leader=")){
} else if (message.contains("Finished")){
}
```

Figura 3 – Mensagens diferentes enviadas no spread

3.3. UserManagerContract

Um dos passos da implementação do sistema foi a criação de um contrato que permitisse à User App ter acesso ao serviço fornecido pelo Manager gRPC server. Este serviço tem como objetivo o de fornecer uma interface que permite obter um "Resume", especificado no contrato ManagerWorkerService (Figura 4), por parte das aplicações consumidores, User App, e por parte das aplicações servidores de fornecer um "Resume", Manager gRPC Server. Para obter um "Resume" é então necessário que a aplicação cliente realize um pedido ao método getResume enviando no corpo da mensagem do tipo "Type", constituído por uma string type, que corresponde ao tipo de resumo que o utilizador deseja (ALIMENTAR ou CASA). O ficheiro de "Resume" é enviando em stream sendo este divido por blocos de 32kb, sendo cada um desses blocos constituídos por uma string "name", correspondente ao nome do ficheiro .txt, e um "bytes" data, correspondente aos dados do "Resume".

```
syntax = "proto3";

option java_multiple_files = true;
option java_package = "usermanagerservice";

package usermanagerservice; // package do proto

service UserManagerService {
    rpc getResume(Type) returns (stream Resume);
}

message Type{
  string type = 1;
}

message Resume{
  string name = 1;
  bytes data = 2;
}
```

Figura 4 – Comtrato entre UserApp e ManagerServer

3.4. Manager GRPC Server

O servidor "Manager GRPC Server" tem como principal objetivo o de pedir aos "workers" para realizarem um "resume", quando pedido por parte da "user App", e retornar esse resume à "User App". O pedido de resumo é iniciado quando um cliente, "User App", o solicita ao servidor indicado se pretende um resumo do tipo casa ou do tipo alimentar, o servidor cria então uma "spreadMessage" para pedir ao "spread group" para realizar um "resume", a conclusão deste "resume" é notificada através de um evento do "RabbitMQ" cujo exchange é gerido dinamicamente pelo servidor, sendo criado na sua inicialização e destruído no momento de desligar o servidor.

No entanto existe a possibilidade de vários clientes requisitarem um "resume" de forma concorrente, de forma a resolver este problema foi decidido que para obter o "resume" seria necessário estar na posse de um "Token" para se poder aceder à região crítica, que neste caso é o sistema de ficheiros distribuído do "Gluster", sendo que apenas é permitido que um dos clientes possua esse "Token". Se nesse momento em que foi feito o pedido um dos outros pedidos já possuir o "Token", este é então inserido numa lista de pedidos até que o "Token" seja libertado sendo estes pedidos atendidos por ordem temporal de pedidos de "resume".

3.5. UserApp

A aplicação UserApp foi desenvolvida com o propósito de criar uma interface que permitisse aos seus utilizadores a realização de comunicações com o servidor ManagerServer para deste modo requisitar um "Resume" de um dado tipo. Para tal, é necessário que primeiro a UserApp estabeleça uma ligação HTTP ao ManagerServer, para tal foi criado um canal gRPC que permite a comunicação entre a UserApp e o ManagerServer. Seguindo com a criação de um "stub" não bloqueante, do serviço ManagerWorkerService, que atua como um proxy, criando assim uma abstração que permite ao cliente realizar chamadas sobre o servidor

como se se tratasse de um objeto local, passando para o server um Stream sobre o qual os vários blocos de resposta vão ser escritos.

O utilizador começa por realizar uma escolha entre duas opções, terminar a aplicação (opção 99) ou obter um resumo (opção 1).

Tendo sido selecionada a opção de obter resumo o utilizador deve informar o tipo de resumo que deseja obter para que assim a UserApp crie uma mensagem Type com essa informação para enviar ao ManagerServer.

A seguir prepara-se parte do nome do ficheiro txt que será usado para guardar os dados do resumo, criando uma string que conterá o tipo escolhido e a data do pedido no formato "dd-MM-yyyy-HH-mm-ss" ambos separados por um "_".

Por fim é criado um streamObserver sobre o qual vão ser enviados os vários blocos de "Resume" enviados por parte do ManagerServer, é no método onNext (Figura 5) que é criado um ficheiro com auxílio do fileOutputStream (cujo nome é constituído pela string criada anteriormente e pelo nome obtido no getName da mensagem resume separados por "_") onde são escritos os dados do "Resume" recebido.

```
// Cria um streamObserver para receber os resumes
StreamObserver<Resume> resumeObserver = new StreamObserver<>() {
    FileOutputStream fileOutputStream;
    @Override
    public void onNext(Resume resume) {
        try {
            // Verifica se é o primeiro resume recebido

            // Nome do Ficheiro: (Type, Time, resume.getName)
            fileOutputStream = new FileOutputStream(fileName + resume.getName());

            System.out.println("FILENAME UNDER ME");
            System.out.println(fileName + resume.getName());
            // Escreve os dados do resume no ficheiro
            fileOutputStream.write(resume.getData().toByteArray());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

Figura 5 - onNext do StreamObserver

4. Configuração do Sistema

A configuração do sistema é feita através da execução, pela ordem indicada, dos seguintes comandos:

- Executar os seguintes comandos nas 3 vms:
  - sudo service glusterd start;
  - sudo mount -t glusterfs tf-node1:/glustervol /var/sharedfiles;
  - /usr/local/sbin/spread -c /usr/local/etc/vmsSpread.conf > /tmp/spreadlogs 2>&1 &;
- Executar o seguinte comando apenas na vm node1:
  - docker run -d --hostname rabbithost --name rabbitmg \ -p 5672:5672 -p 15672:15672 rabbitmq:management;
- Configuração do exchange com o nome “ExgSales” do tipo “Topic”;
- Configuração das queues com os seguintes nomes e routing keys, fazendo bind ao exchange referido no ponto acima:
  - ALIMENTAR, ALIMENTAR.#;
  - CASA, CASA.#;
- Os seguintes jars têm de ser corridos dentro do sistema de ficheiros partilhados:
  - java –jar ManagerServer-1.0-jar-with-dependencies.jar daemonIp brokerIp;
  - Java –jar Worker-1.0-jar-with-dependencies.jar spreadname queueName IpBroker daemonIp;
- Os seguintes jars são corridos localmente:
  - Java –jar PointOfSale-1.0-jar-with-dependencies.jar IpBroker;
  - Java –jar UserApp-1.0-jar-with-dependencies.jar managerIp.

5. Conclusão

Durante o desenvolvimento deste trabalho, foi possível explorar e aprender um pouco mais sobre diferentes ferramentas usadas no desenvolvimento de aplicações distribuídas, tais como, RabbitMq, Spread, mensagens gRPC e do sistema de ficheiros distribuído do Gluster.

Concluindo, a realização deste trabalho deu-nos uma diferente perspetiva sobre sistemas distribuídos ajudando também a entender um pouco melhor como estas ferramentas são usadas num contexto de uma aplicação profissional, possibilitando assim também obter conhecimento e bases sobre estes tópicos de modo a no futuro podermos continuar a nossa jornada por este tipo de sistemas.