Universidade do Minho

Reverse Proxy - Comunicação de Dados

Mestrado Integrado em Engenharia Informática

Luís Guimarães (A77004)
Gonçalo Duarte (A77508)
Luís Neto (A77763)

Braga
Novembro 2018

1 Introdução

O presente relátorio documenta a realização do trabalho proposto na unidade curricular de Comunicação de Dados.

Foi proposta a implementação de um servidor Reverse Proxy que oferece um único serviço, a gestão de uma pool de servidores capazes de dar resposta a pedidos HTTP. Este servidor apenas possui um ponto de entrada. O ponto de entrada tem um nome e um endereço IP únicos e bem conhecidos, para os seus clientes.

Neste relátorio será discutido a forma como o Reverse Proxy recolhe a informação dos servidores via UDP, para que possa ser feita a seleção de um servidor que responda ao pedido HTTP via TCP, seguindo uma métrica dinâmica.

2 Arquitetura da solução

A arquitetura desenhada é semelhante á descrita no enunciado. De seguida serão enumeradas as várias componentes e a sua função na aplicação:

- Reverse Proxy - Classe que lança duas threads que executam um objecto MonitorUDP e MonitorTCP onde ambos partilham um objecto TabelaDeEstado.
- TabelaDeEstado - Classe que armazena o estado de cada servidor. Esta classe garante que não existem problemas de Concorrência devido á utilização de um ConcurrentHashMap.
- Estado - Classe que armazena os dados relativos a um servidor. Os dados armazenados são a ram, cpu, rtt, IP, porta e largura de banda
- AgenteUDP - Classe que está constantemente á escuta num grupo e que após recebido um packet com a mensagem "Probe", recolhe os dados do servidor e envia-os para o emissor numa mensagem UDP dirgida em unicast.
- MonitorTCP - Classe que reencaminha pedidos HTTP via TCP para os servidores mediante um objecto TabelaDeEstado.
- MonitorUDP - Classe que recolhe dados via UDP dos vários servidores e popula um objecto TabelaDeEstado.
- HmacSha1Signature - Classe que permite criar um assinatura HMAC atráves de um par chave-mensagem
- Link - Classe que lê do cliente e envia para o servidor.
- LinkReader - Classe que lê do servidor e envia para o cliente.

3 Especificação do protocolo UDP

As mensagens protocolares consistem na criação de um string que resulta da junção de duas strings seperadas por ':'. A primeira string são os dados relativos a esse servidor. A segunda é gerada atráves da aplicação de uma função de hash criptográfica á primeira string.

Os dados enviados pelo servidor são a utilização do CPU e a RAM. Para além disso, sempre que o monitor UDP envia uma mensagem multicast, retira um timestamp, o mesmo acontece sempre que recebe uma mensagem, isto é feito para que seja possível calcular o RTT de um servidor. Por fim através da API do DatagramPacket é retirado o endereço IP e porta do servidor que respondeu ao Reverse Proxy.

Os dados retirados são adicionados ou atualizam a tabela de estado.

4 Implementação

De seguida serão apresentados todos os parâmetros das classes desenvoldidas, as bibliotecas utilizadas na aplicação e os detalhes relevantes de cada classe.

4.1 ReverseProxy

```
public class ReverseProxy
```

Esta classe é apenas utilizada para lançar duas threads com um objecto MonitorTCP e MonitorUDP, esta foi criada por motivos de organização, daí não possuir qualquer tipo de parâmetro ou implementação, para além da descrita acima.

4.2 MonitorTCP

```
public class MonitorTCP implements Runnable{
    private ServerSocket serverSocket;
    private int port;
    private TabelaDeEstado tab;
```

Classe que está constantemente a aceitar pedidos HTTP via TCP e que, após aceitar o pedido, seleciona o melhor servidor atráves de objecto TabelaDeEstado. Quando a seleção éfeita é lançada um thread com um objecto Link para estabelecer a comunicação cliente servidor.

4.3 LinkReader e Link

```
class LinkReader implements Runnable{
    Socket http_sock;
    BufferedReader http_out;
    PrintWriter client_in;
    TabelaDeEstado tb;
    InetAddress ip;
    int size;

...

class Link implements Runnable{
    Socket cliente_sock;
    BufferedReader cliente_out;
    PrintWriter cliente_in;
    TabelaDeEstado tb;
    InetAddress ip;
    int size;
```

LinkReader e Link são classes que permitem a comunicação servidor-cliente e cliente-servidor, respetivamente. Tratam-se de classes que leem de um objecto BufferedReader e que escrevem para um objecto PrintWriter. Quando um objecto Link é executado, este lança uma thread com um objecto LinkReader para que seja possível a leitura e escrita em simultâneo. A variável size é partilhada para efetuar o calculo da largura de banda.

4.4 MonitorUDP

```
public class MonitorUDP implements Runnable{
    private TabelaDeEstado tabela;
    private InetAddress group = null;
    private DatagramSocket socket = null;
    private byte[] buf;
    private HmacSha1Signature codec;
```

Este monitor tem como função enviar mensagens multicast para a pool de servidores. Esta mensagem tem como propósito a recolha de dados relativos aos servidores, nomeadamente, a ram e cpu. Estes dados permitem ainda, através da API do DatagramPacket, conhecer o IP e porta dos servidores.

Para garantir a fiabilidade destes dados é utilizado um objecto HmacSha1Signature. Como explicado acima, a mensagem enviada pelo Agente UDP é uma string que se encontra dividida a por ':'. Ao lado direito é aplicado a mesma função criptográfica utilizada no AgenteUDP. Caso o lado esquerdo seja igual lado direito, após a aplicação da mesma função criptográfica utilizada no AgenteUDP, assim, é possível verificar a fiabilidade da mensagem recebida.

4.5 AgenteUDP

```
public class AgenteUDP {
    String group_name;
    private InetAddress group;
    private MulticastSocket socket = null;
    private int port;
    private byte[] bufin = new byte[1024];
    OperatingSystemMXBean bean;
    HmacSha1Signature codec;
```

Este componente da aplicação é utilizado para recolher dados de um servidor e para reencaminhá-los para o ReverseProxy via UDP. Os dados são recolhidos após a receção de uma mensagem multicast enviada pelo ReverseProxy. Para garantir a autenticidade e segurança é utilizado um objecto HmacSha1Signature que aplica uma função de hash criptográfica aos dados que irão ser enviados ao ReverseProxy. Para que seja possível recolher os dados relativos ao cpu e ram é utilizado um OperatingSystemMXBean.

4.6 Estado

```
class Estado implements Comparable<Estado>, Comparator<Estado> {
    long ram;
    double cpu;
    float rtt;
    InetAddress add;
    int port;
    float bandwidth;
```

Como anteriormente referido esta classe é utilizada para armazenar o estado mais atual de um servidor. Para além disso possui a função:

```
    public static Estado melhorEst(Estado e1, Estado e2){
        int p1, p2;
        p1=p2=0;
        if(e1 == null) return e2;
        if(e2 == null) return e1;
        if(e1.ram > e2.ram) p1++;
        else p2++;
        if(e1.cpu > e2.cpu) p1++;
        else p2++;
        if(e1.rtt > e2.rtt) p2++;
        else p1++;
        return p1 > p2 ? e1 : e2;
    }
```

Esta função permite determinar retorna o melhor estados passado como argumento.

4.7 TabelaDeEstado

```
public class TabelaDeEstado {
    public Map<InetAddress, Estado> tab;
```

Classe que armazena os dados de todos os servidores da pool. Como a aplicação é concorrente esta classe, quando instanciada, cria um objecto ConcurrentHashMap para que não existe problemas de concorrência.

4.8 Bibliotecas

- java.io.IOException
- java.io.BufferedReader
- java.io.InputStreamReader
- java.io.PrintWriter
- java.security.InvalidKeyException
- java.security.NoSuchAlgorithmException
- java.security.SignatureException
- java.net.*
- java.net.InetAddress
- java.net.ServerSocket
- java.net.Socket
- java.util.Formatter
- java.util.ArrayList
- java.util.Arrays
- java.util.List
- java.util.concurrent.ThreadLocalRandom
- java.util.Comparator
- java.util.Map
- java.util.concurrent.ConcurrentHashMap
- java.util.stream.Collectors
- javax.crypto.Mac
- javax.crypto.spec.SecretKeySpec
- com.sun.management.OperatingSystemMXBean
- java.lang.management.ManagementFactory

* detalhes, parâmetros, bibliotecas de funções, etc.

5 Testes e resultados

Após a implementação foram efetuados testes com vários tipos de ficheiros. Destacamos três tipos de ficheiro:

- HTML
- Código Java compilado
- Ficheiro de música no formato mp3

Os testes realizados foram bem sucedidos isto é, o pedido realizado pelo cliente foi atendido com sucesso, sendo que o ficheiro pedido foi transferido corretamente.

6 Conclusões e trabalho futuro

No futuro poderiam ser implementadas as seguintes funcionalidades ao Reverse Proxy:

- Indicar qual ou quais os servidores que respondem a certos tipos de pedidos.
- Melhorar a métrica para que o desempenho possa ser melhor.
- Permitir que o Reverse Proxy possua uma cache para os pedidos mais frequentes.

Após a finalização do trabalho foi possível verificar que todos os objectivos foram cumpridos. Mas como se trata de uma implementação bastante simples, é possível que está contenha pequenos erros, que podem ser resolvidos caso sejam feitos testes mais rigorosos. Por fim o grupo encontra-se satisfeito com o trabalho realizado.