Universidade do Minho
Escola de Engenharia
Mestrado integrado em Engenharia Informática

Unidade Curricular de Comunicações por Computador

Ano Lectivo de 2016/2017

Proxy TCP reverso com monitorização proativa

Gustavo Afonso Andrez – a27748

Rogério Gomes Lopes Moreira – a74634

Samuel Gonçalves Ferreira – a76507

maio, 2017

Índice

Introdução

Para muitos serviços hoje em dia, um servidor não é suficiente para dar resposta a todos os pedidos de clientes. É necessário por isso sistemas que garantam a resposta a todos os pedidos fazendo uso de vários servidores. Trata-se assim de um sistema que tem um único ponto de entrada (servidor front-end) e que distribui a carga pelos vários servidores de back-end que respondem ao pedido do cliente. O escalonamento da carga pelos diversos servidores de back-end é feito em função de uma métrica dinâmica bem definida.

Neste trabalho, para além da recolha de informação dos servidores via UDP é também implementado um front-end TCP que recebe as ligações dos clientes, escolhe um dos servidores disponíveis e intermedeia a ligação TCP para o servidor escolhido.

O programa foi implementado recorrendo à linguagem Java e testado na topologia CORE fornecida pela equipa docente.

Arquitetura da Solução

A arquitetura inicial da aplicação compreende duas componentes essenciais. Cada uma destas componentes corresponde a uma fase do trabalho prático.

- **Protocolo de monitorização do pool de servidores** – cada servidor de back-end tem um agente de monitorização instalado. Este agente comunica com o servidor de ReverseProxy. Esta comunicação funciona sobre UDP na porta 5555. Quando o Monitor entra em execução envia de 2 em 2 segundos uma mensagem de registo a informar o servidor de ReverseProxy de que está ativo. Para além disso responde também às mensagens do servidor principal. Este servidor pode ser inicializado e fechado a qualquer momento.

- **Proxy TCP reverso** – O servidor atende pedidos TCP na porta 80. Depois de aceitar uma determinada ligação de um cliente, o servidor principal escolhe o servidor de back-end mais adequado para atender aquele pedido naquele momento, consultando a tabela com a informação de estado. Abre depois uma ligação TCP na porta 80 com o servidor de back-end e intermedeia a ligação entre o servidor de back-end e o cliente, ou seja, tudo o que o for recebido do cliente é enviado para o servidor de back-end escalonado para a determinada ligação.

Especificação do protocolo de monitorização

Primitivas de comunicação

A comunicação correspondente à monitorização é feita através de sockets UDP entre o Reverse Proxy e todos os servidores de back-end. Existem três primitivas de comunicação relativas à monitorização:

- **HELLO** – os servidores de back-end enviam para o servidor de Reverse Proxy hello's para assinalarem que se encontram ativos;
- **Probe Sender** – o servidor de Reverse Proxy envia para cada servidor de back-end um pacote que contém um número de sequência (gerado aleatoriamente);
- **Probe Response** – cada servidor de back-end responde ao Probe Sender enviado pelo Reverse Proxy com o número de sequência do pacote a que estão a responder e o número de ligações ativas que tem nesse momento.

Formato das mensagens protocolares (PDU)

O formato das mensagens protocolares é o seguinte:

Número de sequência do pacote	Número de ligações ativas no servidor

O número de sequência do pacote serve não só para fazer corresponder cada resposta recebida a cada pedido solicitado, como para controlar o round trip time (RTT). O número de ligações ativas no servidor serve para controlo das ligações, de forma a permitir um adequado escalonamento dos clientes pelos diversos servidores back-end. Por fim, nesta mensagem recebida, é também consultado o endereço IP que servirá como controlo para o preenchimento da tabela de servidores do Reverse Proxy.

Interações

De 7 em 7 segundos cada servidor de back-end envia um “HELLO” ao Reverse Proxy para que este saiba que ainda está ativo. Por sua vez, o servidor principal envia um “Probe Request” a cada servidor de back-end, ao qual estes respondem através de um “Probe Response”. Esta operação é feita sequencialmente sobre os servidores back-end constantes da tabela, com um tempo de intervalo fixo (2 segundos).

Implementação

Foram implementados três programas distintos:

- BackendServer - responsável por controlar o servidor de back-end e todas as comunicações com o servidor de Reverse Proxy;
- Client - responsável pela comunicação entre o cliente e o Reverse Proxy;
- ReverseProxy - que controla todo o servidor de Reverse Proxy, gerindo todas as comunicações quer de monitorização quer de comunicação entre o cliente e o servidor back-end.

BackendServer

No momento em que o servidor de back-end começa a sua execução este já tem acesso ao IP do Reverse Proxy que é bem conhecido por todo o sistema.

O BackendServer é composto por 4 classes distintas:

- **BackEndServer** – classe principal do sistema, faz toda a gestão;
- **Beeper** – Llgo que o servidor inicia a execução é criada uma instância dessa classe, responsável por enviar “HELLO” de 7000 em 7000ms ao Reverse Proxy.
- **MonitorUDP** – criado também logo quando o servidor inicia a sua execução. É responsável por ficar à espera de receber o “Probe Request” (PR). Recebida uma mensagem, testa se o pacote recebido é iniciado por “*probe*”. Em caso positivo envia um “Probe Response” contendo o mesmo número de sequência do PR a que está a responder e o número de ligações ativas.
- **Server** – comporta-se como um handler de cada ligação TCP ao servidor. Para cada ligação vai existir uma thread Server que fica ativa até ao cliente sair. É responsável por tratar dos pedidos dos clientes.

Client

Classe que gere a interação entre um cliente e o Reverse Proxy e por sua vez o servidor de back-end.

- **Client** – faz a ligação TCP com o Reverse Proxy. A interação é feita através de palavras chave que o cliente envia ao Reverse Proxy (o IP deste é bem conhecido), ou seja, o cliente tenta ligar-se ao socket do RP e fica à espera da mensagem de confirmação. Se não começar por "*erro*" prossegue, caso contrário ocorreu um erro na ligação e termina a ligação. Entra no ciclo e enquanto o input for diferente de "*sair*", continua a enviar as mensagens e a receber respostas.

ReverseProxy

Programa principal que gere o servidor de Reverse Proxy, escalonando as ligações TCP entre os vários servidores de back-end e gerindo as informações de monitorização dos servidores. É composta por várias classes:

- **BackendInfo** – classe que contém a informação de monitorização de cada servidor de back-end - número de medições do RTT, soma dos RTT, ligações ativas, o número de perdas e o número de perdas consecutivas. Existem tantos BackendInfo como o número de servidores back-end ativos;
- **Handler** – classe que intermedeia a conversa entre o cliente e o servidor back-end. Quando inicia a execução tem presente no seu construtor a informação do servidor escalonado para a ligação e o socket TCP do cliente. Cria depois um socket TCP com o servidor, lê a frase de confirmação e retransmite para o cliente.;
- **Listener** – classe que permite que apenas uma thread em cada momento faça as leituras do socket. Quando lê um pacote "HELLO", vê o IP contido no pacote. Se o IP não existir na tabela é então adicionado à tabela que contém a informação sobre todos os servidores de back-end. Caso contrário ignora o pacote. Quando o pacote recebido é iniciado dois números consecutivos, separados por uma vírgula, trata-se de um Probe Response e por isso adiciona-o à variável probeResponse e o probeSender acede ao pacote;
- **Pacote** – implementação com exclusão mútua de um *datagrampacket*;
- **ProbeSender** – itera sobre todos os *backendinf*o da tabela e, para cada um, envia um probeRequest com um número de sequência aleatório. Fica à espera da resposta e quando é recebida, faz o cálculo da média dos RTT e atualiza a tabela;
- **ReverseProxy** – executa duas threads: probeSender e listener. Em seguida entra num ciclo "while(true)", ficando disponível para aceitar a ligações de clientes. Para cada ligação de um cliente escolhe um dos back-end segundo o algoritmo de escalonamento. Depois de selecionar

o melhor servidor de back-end cria um handler com acesso ao servidor de back-end e ao cliente e executa a ligação entre os dois.

Algoritmo de Escalonamento

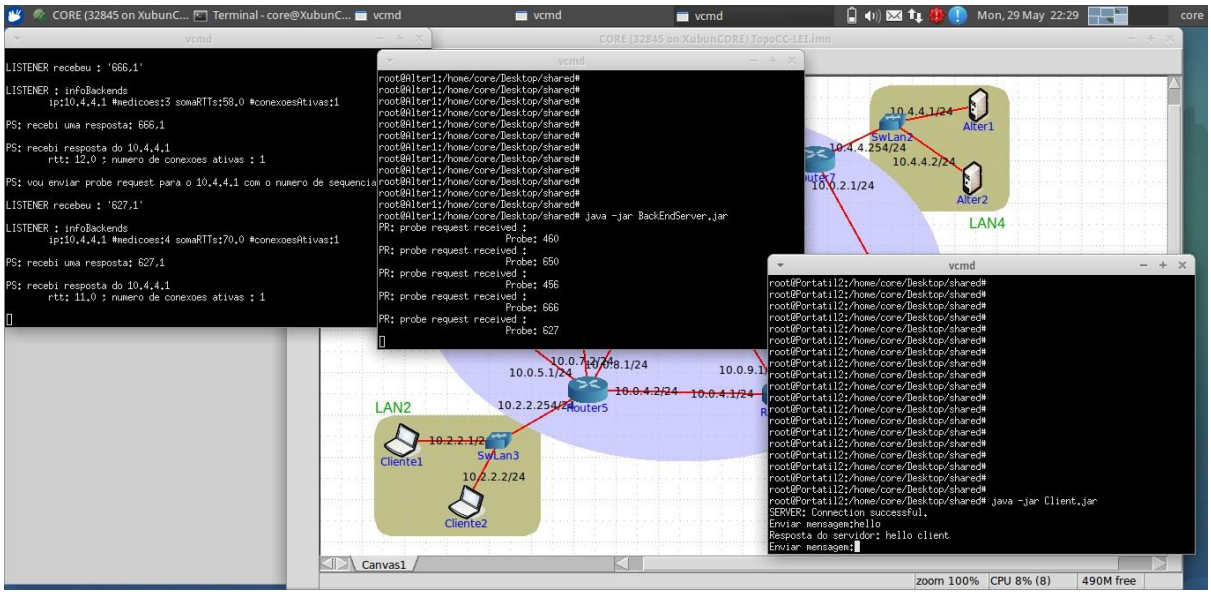
O nosso algoritmo de escalonamento, usado quando um cliente envia um pedido ao RP e este tem que decidir qual o melhor servidor de back-end nesse momento para responder a esse pedido, tem em conta os três critérios referidos abaixo na seguinte ordem:

1. Taxa de perdas;
2. Número de ligações ativas;
3. Média dos Round Trip Time;

De realçar ainda que os RTT são calculados a partir de médias, ou seja, quando um pedido de informação é enviado ao servidor de back-end, o pacote contém um número de sequência. O número de sequência está associado a uma data e hora do envio do pedido correspondente. Quando a resposta ao pacote é recebida é calculado o Round Trip Time usando essa data de envio. O intervalo de tempo determinado é somado ao somaRTT e a variável nMedicoesRTT é incrementada. Para cálculo do RTT médio a soma total de RTT é dividida pelo número de medições de RTT (RTT=somaRTT/ nMedicoesRTT).

Testes e Resultados

O programa foi desenvolvido tendo como princípio correr na tipologia CORE disponibilizada pela equipa docente.



Conclusões e trabalho futuro

Com a elaboração deste projeto concluímos não só a importância de um sistema deste género para os sistemas de hoje em dia, mas também a complexidade do desenvolvimento deste tipo de sistemas já que existem diversas variáveis a ter em consideração. Concluímos também que é conveniente efetuar testes exaustivos de modo a conseguir um bom aproveitamento do sistema.
Relativamente ao trabalho desenvolvido deixamos algumas observações:

- foi implementado na parte TCP apenas comunicações via string. Contudo, recorrendo por exemplo a projetos como o mini-http, podia ser implementado um servidor http completo;
- caso fosse implementado um servidor http de back-end, o programa Client deixaria de ser necessário, uma vez que a comunicação poderia ser feita através de um browser.

Considerando o desenvolvimento do trabalho, consideramos que os objetivos do projeto foram cumpridos na totalidade.