Trabalho Prático N.º 2 – Balanceamento de carga com servidor proxy invertido

Diana Costa, Marcos Pereira e Vitor Castro

Universidade do Minho, Departamento de Informática, 4710-057 Braga, Portugal

{ a78985, a79116, a77870}@alunos.uminho.pt

Abstract. O presente relatório foi produzido para a Unidade Curricular de Comunicações por Computador do 3º ano do Mestrado Integrado em Engenharia Informática. Descreve a arquitetura da solução, os protocolos utilizados e a implementação de um serviço de proxy invertido TCP.

Keywords: Servidor, Agente, Monitor, Proxy, UDP, TCP.

1 Introdução

Um servidor de Proxy Invertido é um servidor de *front-end* que serve de ponto de ligação de um qualquer utilizador a um conjunto de servidores de *back-end*. Este tem um endereço de IP conhecido, ou seja, o ponto de contacto para o cliente. Quando recebe um contacto, desvia-o para o servidor de *back-end* com maior probabilidade de oferecer uma resposta eficiente ao pedido efetuado. Para saber a qual servidor encaminhar o pedido, é mantida uma tabela de estado que contém todas as informações sobre estes servidores.

O sistema tem, por isso, a componente de monitorização de servidores, em que o contacto é feito por UDP. É responsabilidade do Monitor do servidor *front-end* de os contactar, de modo a atualizar a sua tabela de estado, que preserva diversas informações de relevância.

Adicionalmene, surge a componente de atendimento de pedidos TCP, onde as conexões são recebidas e encaminhadas para o servidor de *back-end* selecionado, de acordo com a performance disponível.

2 Arquitetura da Solução Implementada

2.1 Agente de monitorização (MonitorUDP)

O Monitor UDP, lançado no servidor de proxy inverso, é responsável pelo contacto em *UDP MultiCast* dos servidores de *back-end* e receção das suas respostas. O Monitor é constituído pelas *threads Signal, ReceiveUDP* e *Cleaner*. *Singal* é responsável por enviar, de 5 em 5 segundos, o pedido em *Multicast* para os servidores, sendo *ReceiveUDP* responsável pela receção dos pacotes de resposta enviados pelos mesmos. *Cleaner* é responsável por, de 20 em 20 segundos, caso algum servidor não ofereça qualquer resposta (ou seja, estiver inativo), remover os mesmos da *Tabela* de monitorização.

Cada servidor de *back-end* possui um Agente UDP que, ao receber um sinal do servidor de proxy, envia um pacote com a RAM disponível e CPU utilizado. A receção do pedido de informações é feita pelo *ProbingHandler* de cada Agente. É também esta *thread* que envia a resposta que, tal como explicitado anteriormente, será recebida pela *ReceiveUDP* do Monitor.

A *Tabela* é então atualizada de 5 em 5 segundos, onde reúne as seguintes informações, acerca de cada *Servidor* detetado:

- address – endereço do servidor;
- rtt – *round trip time* do servidor nos últimos 5 segundos;
- cpuFifo – FIFO que armazena valores reportados de uso de CPU no último minuto;
- ramFifo – FIFO que armazena valores reportados de uso de RAM no último minuto;
- bandwidthFifo – FIFO que armazena valores reportados de uso de *BANDWIDTH* no último minuto;
- totalBandBytes – total de *bytes* usados nas conexões entre o servidor *back-end* e o cliente;
- pacotesTotais – número total de pacotes enviados ao servidor (com ou sem resposta);
- pacotesPerdidos – número total de pacotes aos quais o servidor não respondeu;
- nrConexoesTCP – número de conexões atuais do servidor com clientes;
- nrVezesRTT – número total de pacotes recebidos resposta do servidor;
- lastSended – tempo em nanosegundos do último pacote enviado ao servidor;
- lastReceived – tempo em nanosegundos do último pacote recebido do servidor;

2.2 Proxy Reverso (ligação Cliente – Servidores *back-end*)

Nesta segunda parte, deve ser possível que o servidor TCP seja contactado, através da porta 80, ligando o cliente ao servidor selecionado, com base num algoritmo de seleção.

Assim, proxy reverso lança uma *thread Listener80,* que fica à espera de ligações ao próprio. Quando recebe uma, verifica qual o melhor servidor de *back-end* para proceder à resposta e cria duas novas *threads ClientListener* e *ClientSpeaker*.

Estando estabelecido qual o servidor a lidar com o pedido recebido, a *thread ClientListener* fica à espera de pedidos do cliente e passa-os à *stream* do servidor selecionado. O servidor selecionado processa este pedido e responde, estando a *thread ClientSpeaker* a aguardar essa mesma resposta, passando-a para a *stream* do cliente. Em ambas as *threads* se faz a atualização da *Tabela*, relativamente ao número de *bytes* que foram transferidos, bem como em relação ao número de conexões no caso de *ClientSpeaker*.

3 Arquitetura da Solução Implementada

Para o sucesso da implementação, definiu-se um protocolo, que é descrito facilmente pelos seguintes pontos, que representam uma interação completa entre Monitor e Agente UDP:

- Monitor UDP envia, através de *Signal*: “INFO”.
- Agente UDP recebe, através de *ProbingHandler* a mensagem;
- Agente UDP verifica se essa mensagem é “INFO”;
- Agente UDP envia, através de *ProbingHandler* a mensagem: “DATA,valor_de_cpu,valor_de_ram,security”. valor_de_cpu é o valor médio de uso de cpu da máquina, valor_de_ram é o valor livre de ram, security é uma string de bytes que representa a frase composta pelo valor de cpu e ram atuais, após passagem por uma *hash* partilhada entre Agente UDP e Monitor UDP.
- Monitor UDP recebe, através de *ReceiveUDP* o pacote e verifica se o valor security, concatenando os valores recebidos de cpu e ram, e passando-os pela mesma função de *hash*, cuja *seed* é partilhada.
- Monitor UDP atualizada a *Tabela*.

De 4 em 4 ciclos de interação, ou seja, 20 em 20 segundos, a *thread Cleaner* trata de verificar se algum *Servidor* está inativo há mais do que 30 segundos, removendo-o da *Tabela* caso se verifique.

Todas as outras informações necessárias conseguem ser extraídas através de funções existentes na biblioteca *DatagramSocket*, que trabalha sobre *PDUs* em *UDP*.

4 Implementação

Com vista à resolução dos problemas propostos, foram implementadas algumas funcionalidades com importantes características para o bom funcionamento do sistema, que se enumeram:

- *Thread Signal:* responsável por, de 5 em 5 segundos, enviar em *Multicast* mensagens para o grupo com endereço IPv4 239.8.8.8;
    - Atualiza campo que regista o tempo do último contacto, para todos os servidores da tabela (para calcular o *RTT*).
- *Thread ReceiveUDP:* responsável por receber as respostas dos servidores no grupo anterior;
    - Atualiza campo que regista o tempo da última resposta recebida, para todos os servidores da tabela (para calcular o *RTT, neste momento*);
    - Verifica integridade das mensagens recebidas.
- *Thread Cleaner:* responsável por, de 20 em 20 segundos proceder à limpeza dos servidores que estejam inativos (sempre que entre duas mensagens enviadas não haja uma resposta, vamos assumir que o servidor não responde);
- *Thread ProbingHandler:* responsável por receber os pedidos enviados em *multicast*, em cada servidor;
    - Calcula RAM e CPU disponível e utilizados (para cálculo de performance);
    - Cria string para verificação de integridade.
- *Thread Listener80:* responsável por receber pedidos TCP dos clientes;
    - Escolhe o melhor *Servidor* dos listados na *Tabela*;
    - Cria a *thread ClientListener* e *ClientSpeaker*, com o propósito de unir as streams de ambos os lados;
    - No fim da comunicação, informa a *Tabela* que o contacto foi terminado.
- *Thread ClientListener:* responsável por ouvir os pedidos, após estabelecida conexão com um servidor *back-end*.
    - Depois de receber um pedido, adiciona os *bytes* utilizados para o mesmo ao *Servidor*.
- *Thread ClientSpeaker:* responsável por enviar respostas aos clientes, após estabelecida a conexão ao servidor *back-end*.
    - Depois de receber uma resposta do servidor, adiciona os *bytes* utilizados para o mesmo ao *Servidor*.
    - No fim da comunicação, informa a *Tabela* que o contacto foi terminado.
- *Thread BandwidthCalculator:* recalcula a bandwidth utilizada para todos os *Servidor*, um a um, de 5 em 5 segundos.

5 Testes e resultados

Como o projeto tinha duas fases, o grupo foi fazendo testes consecutivos sobre o sistema desenvolvido, que decorrem com normalidade. Foi utilizado, para a segunda fase, o emulador CORE da máquina virtual fornecida, com recurso ao servidor *mini-httpd*, que se viu ser de fácil uso.

Concluiu-se que todos os pedidos feitos foram tratados com sucesso, através de pedidos com o comando *wget* ao servidor proxy, em busca de ficheiros *html* e *pdf*.

6 Conclusão

O presente trabalho permitiu ao grupo desenvolver e solidificar competências na linguagem Java, e suas bibliotecas de comunicação UDP e TCP. A análise de PDUs e de *performance* dos servidores foi um trabalho interessante e que permite a realização do que são necessidades de sistemas reais.

O grupo acredita que há melhores implementações para o contacto com o Monitor UDP, por exemplo, sendo os Agentes UDP a reportar a sua situação ao Monitor no momento da chegada e nos momentos posteriores, o que reduziria o *overhead* de comunicações.

Por fim, e apesar do trabalho ter sido concluído inteiramente e com sucesso, o grupo acredita que seria preferível ter implementações mais pequenas, sem tantos pequenos pormenores, e que permitissem maior celeridade na realização do trabalho prático. De facto, a sobrecarga de tempo a que o trabalho obrigou não é equivalente à quantidade de conhecimentos adquiridos.