Relatório TP2

Carlos Silva, João Coelho e Miguel Silva

Universidade do Minho, Departamento de Informática, 4710-057 Braga, Portugal
e-mail: {a75107,a74859,a74601}@alunos.uminho.pt

Resumo Este trabalho expõe uma situação de necessidade de escolha perante vários servidores Web, procurando não só propor uma solução para situações semelhantes, como também expor a importância deste tipo de seleção, de modo a aumentar a eficácia da resposta aos pedidos. Desde a arquitetura à implementação e posteriores testes, neste documento apresenta-se o relatório de todo o processo de desenvolvimento do tema.

1 Introdução

1.1 Contextualização

Em muitos serviços Web, um único servidor não é suficiente para dar vazão aos pedidos dos clientes. Nesses casos, é necessário ter uma pool de N servidores de back-end, capazes de dar resposta aos pedidos. Ainda assim, o serviço terá um único ponto de entrada, comum a todos os clientes, que se trata de um servidor de front-end, com nome e endereço IP únicos e bem conhecidos. A tarefa deste servidor, habitualmente designado por *Reverse Proxy*, é atender as conexões dos clientes e desviá-las para um dos servidores de back-end disponíveis.

A escolha do servidor pode ser cega, baseada por exemplo num algoritmo de Round-Robin, que faz uma distribuição equitativa das conexões pelos N servidores de back-end, mas será esta a melhor solução?

1.2 Exposição do problema

O principal objetivo deste trabalho é desenhar e implementar um serviço simples de proxy reverso TCP, em que a escolha do servidor a usar se faz com base em parâmetros dinâmicos, como por exemplo o RTT, as perdas e número de conexões TCP do servidor. Para tal, é necessário haver monitorização dos dados do estado do servidor e da rede, redirecionando em função de uma métrica dinâmica bem definida. Pretende-se desenhar e implementar um protótipo simples desta abordagem, separando-a em dois momentos: numa primeira fase temos um protocolo sobre UDP, para criar e manter atualizada uma tabela com dados recolhidos por agentes de monitorização; depois, numa segunda fase, pretende-se implementar um servidor proxy TCP genérico, que fique à escuta na porta 80 e redirecione, automaticamente, cada conexão TCP que recebe para a porta 80 de um dos servidores de back-end disponíveis (o que aparente estar em melhores condições para o fazer).

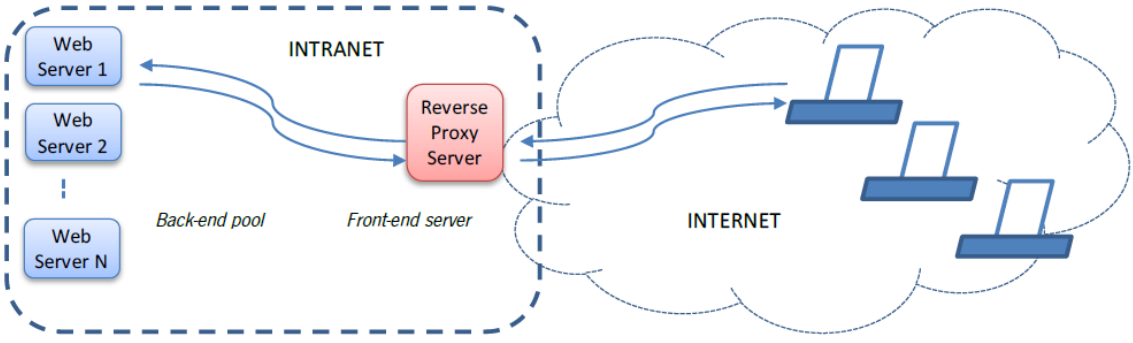
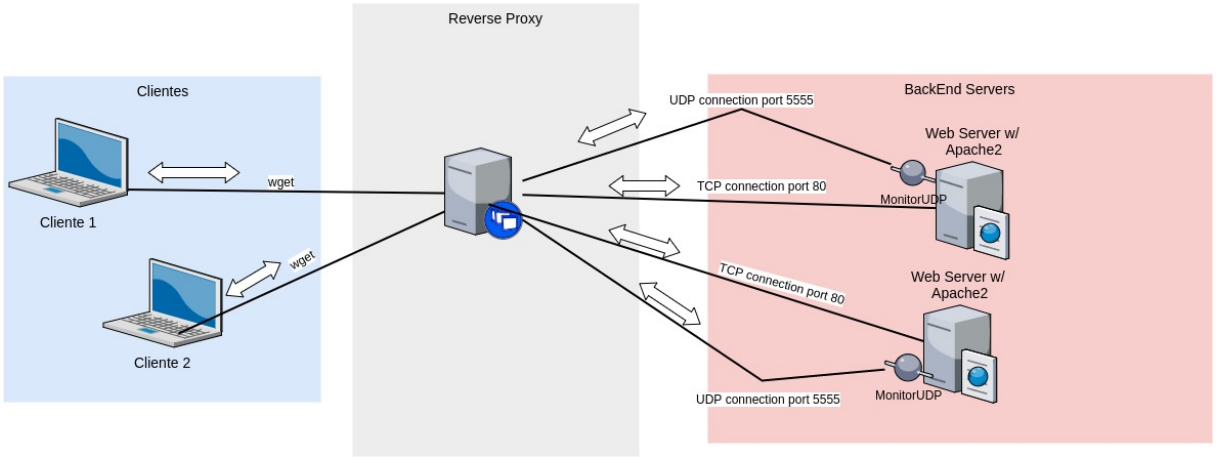


Figura 1. Esquematização da arquitetura do problema.

2 Arquitetura da solução implementada(em ambiente de teste)



3 Especificação do protocolo de monitorização

- **Primitivas de comunicação:** Sockets UDP.

- **Formato do PDU:** constituído pelo número do ACK do próximo pacote, que o servidor pretende receber, e pelo timestamp que permitirá calcular o RTT. Sobre o ACK, importa referir que é utilizado no cálculo do taxa de perdas e de duplicados. Já sobre o timestamp, nota para o cuidado especial que se teve para que contabilizasse apenas o tempo de deslocamento de um pacote entre servidores. Aquando da receção de um pacote, o monitor marca o momento. A esse valor temporal subtrai o timestamp recebido, obtendo assim o tempo da deslocação até si. Depois, no momento em que se prepara para enviar a resposta ao pedido de probing, volta a calcular o tempo atual, subtraindo o valor do tempo da deslocação. Assim, ao receber a resposta, o *Reverse Proxy* só tem de calcular o momento atual e subtrair o timestamp recebido na resposta, obtendo assim o RTT.

- **Interações:** Os servidores *back-end* disponíveis enviam *hellos* de 5 em 5 segundos, com o *Reverse Proxy* a responder aos mesmos com pedidos de *probing*.

4 Implementação

– **Monitor UDP:** Cada *back-end* terá associado a si um monitor implementado via UDP, que monitoriza o estado da ligação com o *Reverse Proxy*. O monitor envia *hellos* de 5 em 5 segundos para o *Reverse Proxy* e responde ao pedido de *probing* por parte do servidor. Para o tratamento do envio dos *hellos*, foi implementada uma thread que se dedica somente a isso.

A thread principal do monitor é a responsável por responder aos pedidos que o *back-end* faz quando deseja atualizar a entrada da tabela que correponde ao presente monitor. Assim, sempre que é recebido um pacote no monitor vindo do *Reverse Proxy*, é recebido o número de sequência desse pacote, sendo a esse número adicionada uma unidade que advém de agora o monitor esperar pelo próximo pacote. É, portanto, enviado no PDU o número de sequência que corresponde ao próximo pacote que o monitor está à espera. É também calculado e enviado no PDU um *timestamp*, já esclarecido na secção acima.

Esta comunicação por UDP foi implementada com o auxílio da API de *DatagramPackets*.

– ***Reverse Proxy*:** O *Reverse Proxy* tem a função de selecionar um servidor *back-end* para dar resposta a pedidos de clientes de forma mais rápida e eficiente.

Assim, é necessário enviar pedidos de *probing* aos *back-ends* que estão ligados, de forma a que a escolha do servidor seja a melhor possível. O envio e receção de *probing* são assegurados por uma nova thread criada para o paralelismo dos processos. Estes pedidos são enviados logo que um *hello* vindo de um monitor é recebido, ou seja, de 5 em 5 segundos também. O pedido enviado é tratado pelo monitor da forma referida no ponto anterior, e depois recebido novamente pelo *Reverse Proxy*, que retira informação do PDU que lhe chega. Esta informação ajuda a atualizar a tabela com os dados dos monitores, como o RTT (*round-trip-time*), número de pacotes duplicados e a *loss rate* (percentagem de pacotes perdidos).

O RTT é calculado através do timestamp que traz consigo o PDU. Este timestamp já conta com o deslocamento incial *Reverse Proxy –> back-end*, sendo que quando recebido, agora é somente necessário subtrair o timestamp atual pelo que veio com o PDU. A cada pacote com a resposta/pedido de *probing*, são incrementados o número de pacotes enviados e o número de pacotes recebidos. Este controlo de pacotes que são enviados e recebidos é usado para ser calculada a *loss rate*. O número de pacotes duplicados são calculados com recurso aos ACKs que são enviados e recebidos. Quando é recebido mais do que um pacote com o mesmo ACK é incrementado o número de pacotes duplicados, para o monitor em questão. Toda esta informação é atualizada para a entrada da tabela, correspondente a cada monitor.

O *Reverse Proxy*, além de tratar de toda a monitorização dos *back-ends* via UDP utilizando DatagramPackets, como se acabou de expôr, trata também da receção e encaminhamento de um pedido de um cliente para o melhor *back-end* no momento, via TCP.

Para a escolha do melhor *back-end* para encaminhamento, é utilizado um score que é calculado utilizando informação presente na tabela deste. Assim, para o cálculo do score, o *loss rate* é multiplicado por 100, o número de conexões TCP é multiplicado por 2 e o número de pacotes duplicados é dividido pelo número total de pacotes enviados e recebidos, de modo a que a *loss rate* e o número de conexões TCP tenham mais significância e os duplicados não tenham tanto peso. O score é então a soma destes três parâmetros mais o RTT. A cada 5 segundos, o score é reiniciado.

A receção do pedido do cliente, o encaminhamento do pedido do cliente para o melhor *back-end* disponível e a receção da resposta do *back-end* são tratados numa nova thread criada para este propósito.

Finalmente, se um *back-end* não mandar *hellos* durante um período contíguo de 20 segundos, o *Reverse Proxy* retira a entrada da tabela que é correspondente ao

back-end que não se mostra disponível. Esta funcionalidade é também tratada numa nova thread, criada para este mesmo propósito.

5 Testes e resultados

Para cenário de teste foi utilizado Apache2 em cada um dos servidores *back-end*(dois), uma máquina a correr o *Reverse Proxy* e ainda dois clientes a efetuar pedidos ao *Reverse Proxy*. Os testes foram realizados na topologia core da unidade curricular, a máquina do *Reverse Proxy* é o Servidor 1, as máquinas de *back-end* são os portáteis 1 e 2 e, por último, os clientes que são o 1 e 2.

Deixou-se a topologia correr, durante alguns segundos, o *Reverse Proxy* e os 2 servidores *back-end* para que a tabela do *Reverse Proxy* estabilizasse. Depois, efetuou-se o comando wget no Cliente 1 para o *Reverse Proxy* obtendo o index.html do servidor *back-end* com melhor score naquele instante. Verificou-se que o número de conexões TCP (na tabela) do servidor *back-end*, naquele momento, aumentou para 1, então testou-se um wget de cada cliente e o número de conexões TCP do *back-end* escolhido atualizou para 2. Como os pedidos dos clientes não foram efetuados exatamente ao mesmo tempo, também existiu a situação em que cada pedido foi atendido por um servidor *back-end* diferente, ficando ambos com uma conexão TCP naquele momento.

6 Conclusões e trabalho futuro

Tendo em conta que os resultados obtidos foram satisfatórios, considera-se que a solução implementada é plausível e de aceitável eficiência. Todavia, para uma maior fiabilidade dos resultados, seria necessário testar o programa fora da topologia CORE utilizada. Uma situação mais realista, com máquinas diferentes a correrem o programa, era recomendável. Isto permitiria, por exemplo, tornar mais rigorosa a contagem do número de conexões TCP, permitindo o uso de comandos como o netstat.

Para além do aprofundar dos testes, outra secção a explorar no futuro seria o algoritmo de seleção do servidor *back-end*. Com mais alguma reflexão, inclusão de outras variáveis e, também, melhores testes, antecipa-se que os resultados seriam melhores. Porém, sublinhando, a solução implementada foi satisfatória a este nível.

Referências

1. Kurose, J., Ross K.: Computer Networking . A Top Down Approach Featuring the Internet, 6ª edição (2012)