Ciências ULisboa | Informática

2017/18

Sistemas Distribuídos

Projeto 4

1. Descrição geral

A componente teórico-prática da disciplina de sistemas distribuídos está divida em quatro projetos, sendo que a realização de cada um deles é necessária para a realização do projeto seguinte. Por essa razão, é muito importante que consigam ir cumprindo os objetivos de cada projeto, de forma a não hipotecarem os projetos seguintes.

O objetivo geral do projeto será concretizar um serviço de armazenamento de pares chave-valor similar ao utilizado pela *Amazon* para dar suporte aos seus serviços Web [1], utilizando para tal uma **tabela *hash*** [2]. No projeto 1 foram definidas estruturas de dados e implementadas várias funções para lidar com a manipulação dos dados que vão ser armazenados na tabela, bem como para implementar a tabela recorrendo à técnica de *coalesced chaining* para resolver o problema das colisões de chaves no mesmo índice da tabela. No projeto 2 implementaram-se as funções necessárias para serializar e de-serializar estruturas complexas em mensagens, concretizou-se um conjunto de tabelas *hash* num servidor, e a implementou-se um cliente com uma interface similar à das funções que interagem com as tabelas na memória. No projeto 3 garantiu-se que o servidor suporta múltiplos clientes simultaneamente, implementou-se um modelo de comunicação **tipo RPC** (*Remote Procedure Call*) para as aplicações cliente, e foi também concretizado um mecanismo muito simples de tolerância a faltas.

No entanto, esse mecanismo muito simples não evita a perda dos dados existentes na tabela (se o servidor *crashar*, perdemos tudo o que estava em memória). O objetivo do projeto 4 é construir um sistema tolerante a falhas para evitar este problema. A ideia é replicar o servidor usando um esquema de replicação passiva com primário (*primary backup*).

2. Desenho do sistema

O modelo de replicação passiva com primário é uma das técnicas clássicas de replicação. Uma das réplicas, o primário, desempenha o papel principal. É esta réplica que recebe, executa e responde aos pedidos dos clientes. As outras réplicas, secundárias (os servidores de backup), interagem apenas com o primário, guardando uma cópia do seu estado. Neste projeto vamos considerar apenas um servidor secundário, portanto o sistema tolera apenas 1 falta. Quando o primário falha, o secundário toma o seu lugar. Na Figura 1 ilustramos esquematicamente o sistema.

Como se pode verificar na figura, os pedidos de leitura (no nosso caso: GET, SIZE, etc.) são tratados somente pelo servidor primário, dado não implicarem alteração na tabela de pares chave/valor. No caso de escritas (PUT e UPDATE) há uma atualização da tabela, e portanto é necessário fazer a escrita em ambas as réplicas, para garantir que quando o cliente recebe a resposta ao pedido ambas estão consistentes. Os passos de comunicação apresentam-se representados na Figura 2, ilustrando pedidos de escrita (W) e de leitura (R).

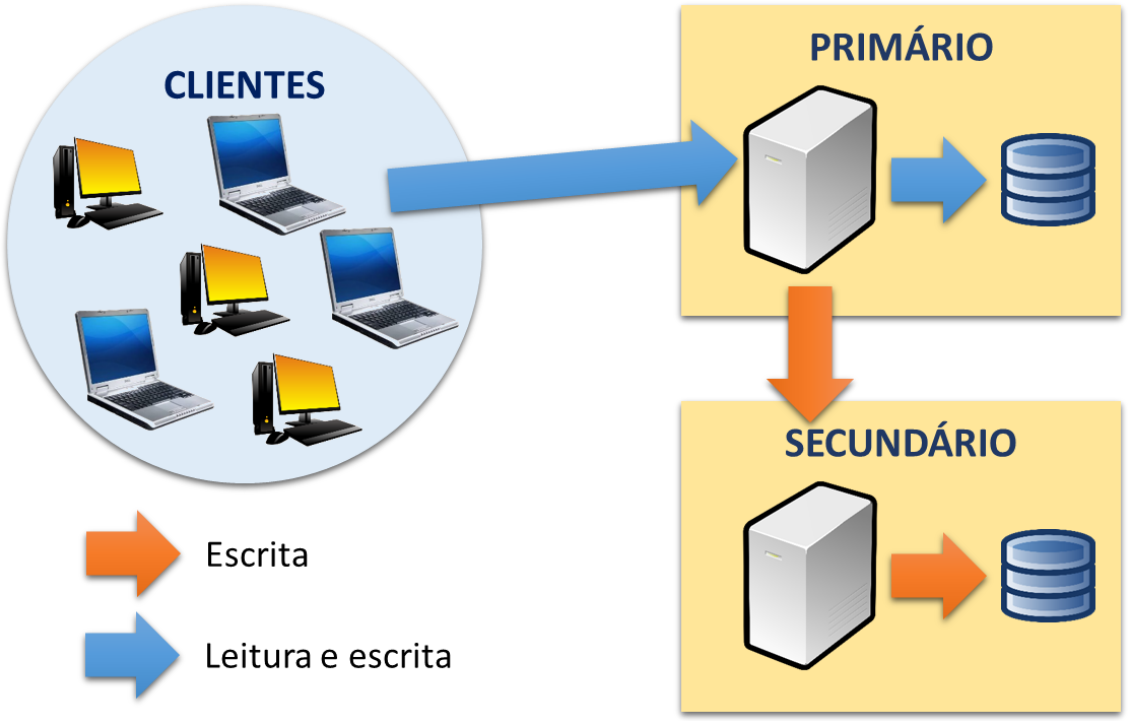

Figura 1 – Arqiutetura da replicação primário/secundário

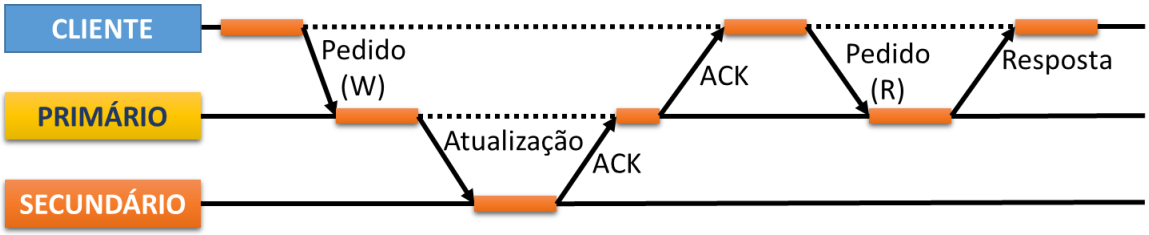

Figura 2 – Sequência de comunicação entre cliente, primário e secundário, em operações de leitura e escrita

2.1. Funcionamento base (com falhas do servidor secundário)

Descreve-se de seguida, em detalhe, o funcionamento base entre cliente, primário e secundário.

1. O servidor passa a ser replicado. Passamos a ter um servidor primário e um servidor secundário.
2. Quando um cliente pede uma operação de leitura ao primário, este:
    a. executa a operação na sua tabela;
    b. envia a resposta ao cliente.
3. Quando um cliente pede uma operação ao primário, que envolve alteração da tabela, o primário:
    a. altera a sua tabela;
    b. faz o pedido de alteração ao secundário. Ao fazer esse pedido o primário atua como cliente do secundário, fazendo com que o secundário proceda à alteração da sua tabela.
    No primário este passo deve ser implementado através de uma nova *thread*. Assim, é preciso garantir uma correta sincronização entre a *thread* principal e a *thread* que interage com o secundário. É também necessário reduzir ao máximo o tempo de resposta ao cliente.
4. Quando o primário recebe confirmação do secundário e já recebeu também a confirmação da sua escrita na tabela, envia a confirmação ao cliente.

a. se der erro na escrita do primário este reporta o erro ao cliente.
b. se a sua escrita correr bem mas não receber OK do secundário (por ele estar "morto" ou enviar erro), envia OK ao cliente e assume que o secundário está "morto" marcando-o como "DOWN".
    i. nesse caso, o primário passa a atuar sozinho até o secundário recuperar e o contactar de novo.
    ii. quando o secundário recuperar deve pedir para atualizar o seu estado.
    iii. nota: durante o processo de atualização o processamento de novos pedidos deve ser interrompido.
    iv. quando tiver a garantia de que o estado do secundário está atualizado o servidor primário marca-o de novo como "UP" e continua a tratar pedidos.

Um ponto importante é conseguir que os servidores distingam se o pedido recebido vem de um cliente ou do outro servidor. Os alunos têm total liberdade para decidirem como tratar esta situação (por exemplo, criando novas operações, se entenderem necessário).

Ter tudo funcional até este ponto equivale a uma classificação de 12 valores. Se estiver funcional apenas até ao ponto 4.a (inclusive) a classificação será de 10 valores.

2.2. Funcionamento com falhas do servidor primário

Nesta secção descrevem-se os requisitos relativos ao funcionamento com falhas no servidor primário.

1. Quando um cliente deteta a falha do primário:
    a. faz o mesmo pedido para o secundário.
        i. se o secundário também não responde então o serviço não está disponível. Nesta situação o cliente tenta outra vez RETRY_TIME depois (novamente tentando primeiro o servidor primário e só depois o servidor secundário). Caso a falha de serviço persista o cliente desiste da operação.
        ii. Se o secundário está ativo, então o secundário responde normalmente ao cliente e o cliente marca esse servidor como primário a partir daí.
2. Quando o servidor secundário percebe que passou a ser servidor primário, passa a atuar sozinho.
3. Quando o antigo primário inicia de novo a execução, contacta o atual primário para atualizar o seu estado, e depois passa a funcionar como servidor secundário.
    a. nota: durante o processo de atualização o processamento de novos pedidos deve ser interrompido.

Ter o ponto 1 completamente funcional vale 3 valores; ter o ponto 2 vale 1 valor; e, finalmente, o ponto 3 vale 4 valores.

3. Implementação

3.1. Alterações do lado do cliente

No cliente o módulo table_client vai sofrer uma pequena alteração: vai ser adicionado como argumento de entrada o endereço e porto do servidor secundário. O client_stub não deverá sofrer alterações. O network_client vai ser alterado para fazer o pedido ao secundário sempre que detete que o primário está em baixo (e passando esse secundário a ser o novo primário que vai contactar a partir daí).

3.2. Alterações no lado do servidor

Os alunos devem criar um novo módulo, definido no cabeçalho primary_backup.h.

A interface a ser oferecida é a seguinte:

```
#ifndef _PRIMARY_BACKUP_H
#define _PRIMARY_BACKUP_H

struct *server_t; /* Para definir em primary_backup-private.h */

/* Função usada para um servidor avisar o servidor "server" de que
 * já acordou. Retorna 0 em caso de sucesso, -1 em caso de insucesso
 */
int hello(server_t *server);

/* Pede atualizacao de estado ao server.
 * Retorna 0 em caso de sucesso e -1 em caso de insucesso.
 */
int update_state(server_t *server);

#endif
```

Relativamente aos módulos atuais do sistema, no servidor o table_skel não vai sofrer alteração. A maior parte da lógica a implementar neste projeto vai ser concretizada no table_server.

A distinção (inicial) entre primário e secundário é efetuada através dos argumentos passados ao programa. O primário deve receber como argumentos o seu porto e o endereço IP e porto do secundário. O secundário deve receber como argumento apenas o seu porto

4. Entrega

A entrega do projeto 4 consiste em colocar todos os ficheiros do projeto, bem como o ficheiro README mencionado abaixo, num ficheiro com compressão no formato ZIP. O nome do ficheiro será **`grupoXX-projeto4.zip`** (XX é o número do grupo). Este ficheiro será depois entregue na página da disciplina, no moodle da FCUL.

O ficheiro ZIP deverá conter uma diretoria cujo nome é **`grupoXX`**, onde **`XX`** é o número do grupo. Nesta diretoria serão colocados:

- o ficheiro de texto README, onde os alunos podem incluir informações que julguem necessárias (e.g., limitações na implementação);
- diretorias adicionais:
  - include: para armazenar os ficheiros .h;
  - source: para armazenar os ficheiros *.c;*
  - object: para armazenar os ficheiros objeto;
  - binary: para armazenar os ficheiros executáveis.
- um ficheiro `Makefile` que permita a correta compilação de todos os ficheiros entregues. Não devem ser entregues ficheiros objeto (.o) ou executáveis. Os executáveis a gerar deverão ter o mesmo nome do ficheiro .c correspondente (sem a extensão .c). Espera-se que o Makefile compile os módulos bem como o cliente e o servidor. Os ficheiros executáveis resultantes devem ficar na diretoria **`grupoXX/binary`**.
  Se não for incluído um `Makefile`, se o mesmo não compilar os ficheiros fonte, ou se houver erros de compilação (isto é, se não forem criados os ficheiros objeto e executáveis), o trabalho é considerado nulo.

  Na página da cadeira podem encontrar vídeos e documentos do utilitário `make` e dos ficheiros `Makefile` (cortesia da disciplina de Sistemas Operativos).

Todos os ficheiros entregues devem começar com três linhas de comentários a dizer o número do grupo e o nome e número de seus elementos.

Os programas são testados no ambiente dos laboratórios de aulas, pelo que se recomenda que os alunos testem os seus programas nesse ambiente.

O prazo de entrega é sexta-feira, dia 8/12/2017, até às 12:00hs.