Sistemas Distribuídos

2013/14

Projeto 5

1. Descrição geral

A parte teórico-prática da cadeira de sistemas distribuídos está divida em cinco projetos, sendo que cada um deles corresponde a uma parte necessária para a realização dos seguintes. O objetivo final será concretizar um serviço de armazenamento de pares chave-valor similar ao utilizado pela *Amazon* para dar suporte aos seus serviços Web, nos moldes explicados nos enunciados anteriores.

Nos projetos anteriores foram construídas várias funções para serializar estruturas de dados e para gerir uma tabela local que suporte um subconjunto dos serviços definidos pela tabela *hash*. Esta tabela foi também já concretizada num servidor, tendo sido ainda concretizada uma aplicação cliente que permite invocar operações que se realizarão na tabela mantida no servidor. O modelo de comunicação oferecido à aplicação é do tipo RPC (*Remote Procedure Call*) e garante que o servidor suporta múltiplos clientes simultaneamente através de multiplexação de I/O.

Foi já concretizado, no projeto 4, um mecanismo muito simples de tolerância a faltas – se o servidor falhar, o cliente tenta contactá-lo uma única vez. No entanto, esse mecanismo muito simples não evita a perda dos dados existentes na tabela (o servidor *crasha*, perdemos tudo o que estava em memória). O objetivo do projeto 5 é construir um sistema tolerante a faltas 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.

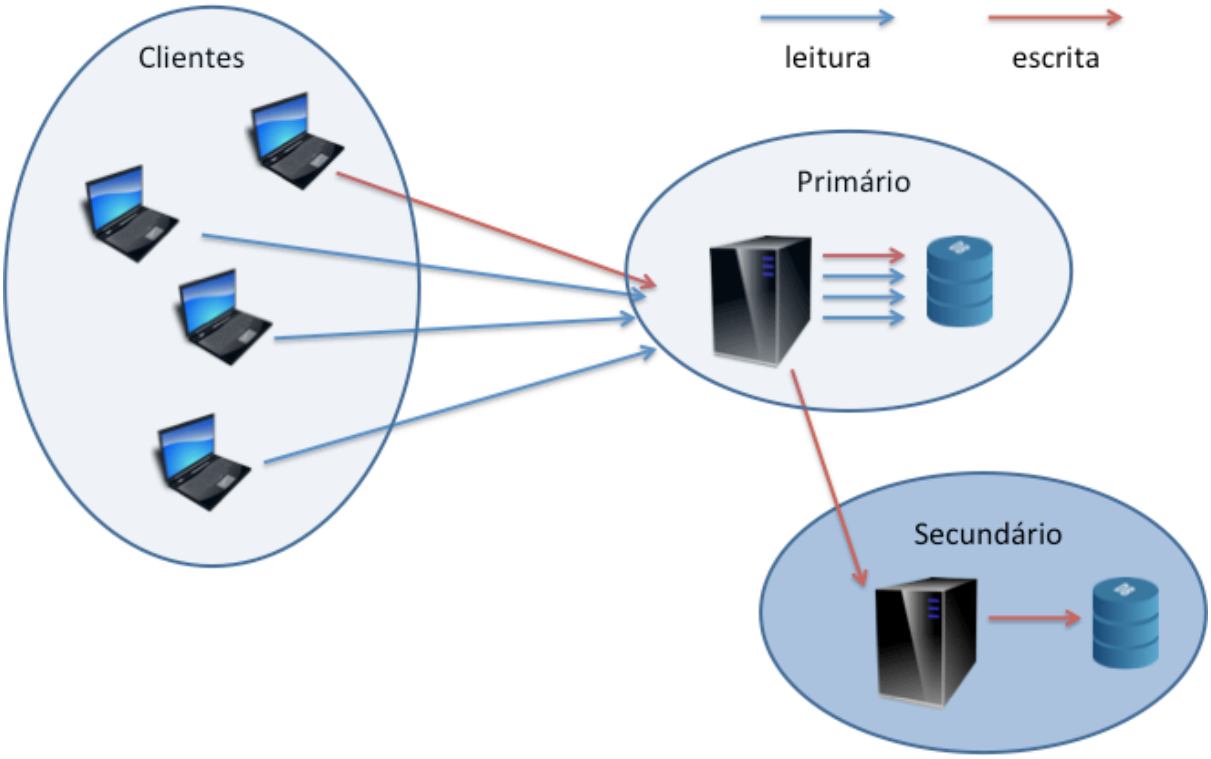

Figura 1. Replicação passiva com primário

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 *key-value store*. No caso de escritas (*PUT* e *CPUT* em alguns casos) 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 ambas estão consistentes. Os passos de comunicação apresentam-se representados na Figura 2, mais uma vez ilustrando pedidos de escrita (W) e de leitura (R).

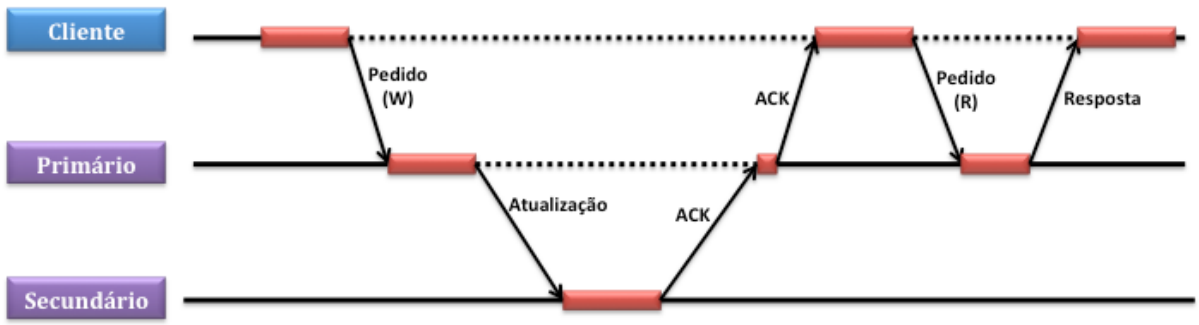

Figura 2. Passos de comunicação em pedidos de leitura (R) e escrita (W)

2.1. Funcionamento base

Em detalhe, ponto a ponto:

1. O servidor passa a ser replicado. Passamos a ter um primário e um secundário.
    a. *Nota: os servidores devem passar a estar à escuta também de eventos do teclado. Em particular, no terminal do servidor vamos poder escrever "print" (sem aspas) para imprimir o conteúdo da tabela e informação acerca da função do servidor naquele momento (se é primário ou secundário).*
2. Quando se executa uma leitura o primário:
    a. executa a operação na sua tabela
    b. envia resposta ao cliente.
3. Quando o pedido é de escrita:
    a. o primário atualiza a sua tabela e
    b. faz o pedido de atualização ao secundário. Notem que ao fazer esse pedido o servidor atua como cliente do servidor secundário. O secundário faz a escrita na sua tabela.
    c. Nota: deve realizar-se o passo anterior usando uma segunda *thread* no primário. É preciso, por isso, garantir correta sincronização entre *threads* antes de enviar a resposta ao cliente. A utilização desta segunda *thread* deve ser feita de forma a reduzir ao máximo o tempo de resposta ao cliente.
4. Quando o primário recebe OK do secundário (quando a segunda *thread* termina) e já recebeu também o OK da sua escrita, manda OK 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 acordar e o contactar de novo.
        ii. Quando o secundário acordar 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 como "UP" de novo 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 faltas do primário

Em detalhe, ponto a ponto:

1. Quando o primário falha é um dos clientes que deteta este facto. Nesse momento:
    a. faz o mesmo pedido para o secundário.
        i. Se o secundário está "morto", o serviço está em baixo. O cliente tenta mais uma vez TIMEOUT segundos depois (novamente tentando primeiro no primário, e só depois no secundário), e caso persista o erro desiste.
    b. Se está ativo, então o secundário responde normalmente ao cliente e o cliente marca esse servidor como primário a partir daí.
2. No momento em que o secundário percebe que vai ser primário, passa a atuar sozinho.
3. Quando o antigo primário "acorda" contacta o novo primário, atualiza o seu estado e passa a ter a função de secundário.
    a. *Nota: mais uma vez, 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 no lado do cliente

No cliente o `table_client` vai sofrer uma pequena alteração: vai ser adicionado como argumento 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 */

/* Funcao 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:

1. No servidor o `table_skel` não vai sofrer alteração.
2. 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 secundário deve receber como argumento o seu porto e o endereço IP e porto do primário. O primário deve receber como argumento apenas o seu porto.

A Figura 3 ilustra a forma como devem estar organizados os vários módulos do sistema.

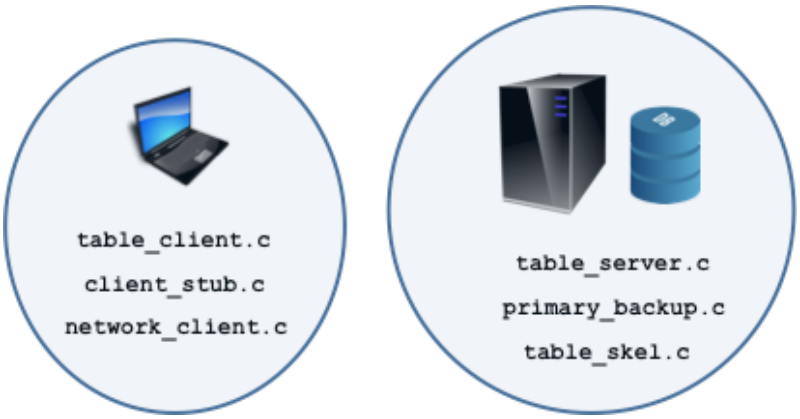

Figura 3. Organização geral do sistema

4. Discussão

Nesta secção deixamos algumas questões para reflexão. Estes tópicos serão discutidos na discussão final do projeto (que irá decorrer em Dezembro e incidirá sobre o trabalho na sua generalidade) e serão tomados em consideração na classificação final (2 valores).

1. Qual o modelo de consistência (*consistency model*) oferecido pela solução implementada?
2. Numa escrita, se o primário enviasse a resposta ao cliente logo após a atualização da sua tabela (i.e., sem esperar pela resposta do secundário), como se apresenta na Figura 4, qual o modelo de consistência oferecido?
3. Quais as vantagens e desvantagens de cada um dos modelos?
4. Durante o processo de atualização o processamento de novos pedidos deve ser interrompido. Porquê?

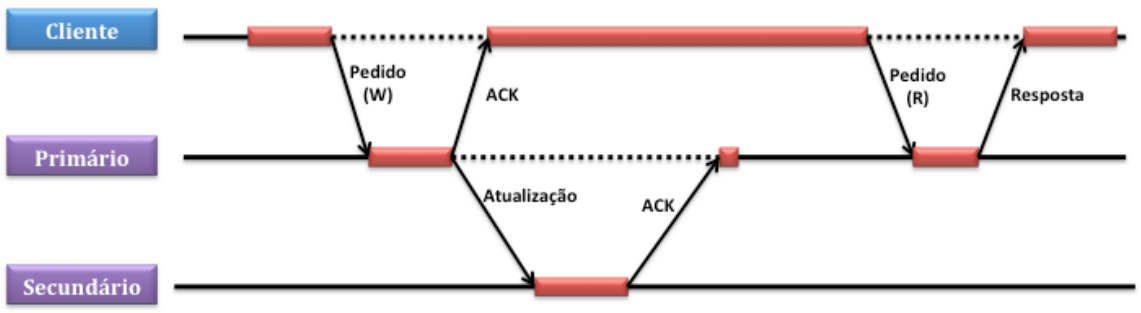

Figura 4. Passos de comunicação da solução alternativa

5. Entrega

A entrega do projeto 5 consiste em colocar todos os ficheiros .c e .h do projeto na diretoria **$HOME/areas_de_grupo/sdNNN/projeto5** (note-se que não é *projecto-5*, nem *proj5*, nem *projecto5,* nem *Projeto5,* nem *projeto 5*). Se acharem necessário, podem incluir um documento de suporte, com um máximo de duas páginas, a explicar resumidamente a vossa implementação ou alguns detalhes que julguem úteis.

Os alunos devem também incluir um ficheiro `makefile` que permita a correta compilação de todos os ficheiros entregues. **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**. O `makefile` deverá criar dois executáveis, `table-client` e `table-server`.

O prazo de entrega é terça-feira, dia 10/12/2013, até às 22:00hs.

Posteriormente, cada grupo ficará sem acesso de escrita à diretoria de entrega.