Ciências ULisboa | Informática

2021/22

Sistemas Distribuídos

Projeto 4

1. Descrição Geral

A componente teórico-prática da disciplina de sistemas distribuídos consiste no desenvolvimento de quatro projetos, utilizando a linguagem de programação C [4], 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 hipotecar os projetos seguintes**.

O objetivo geral do projeto será concretizar um serviço de armazenamento de pares chave-valor (nos moldes da interface *java.util.Map* da API Java) similar ao utilizado pela *Amazon* para dar suporte aos seus serviços Web [1]. Neste sentido, a estrutura de dados utilizada para armazenar esta informação é uma **lista encadeada simples** [2] e uma **tabela *hash*** [3], dada a sua elevada eficiência ao nível da pesquisa.

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 gerir uma *tabela hash* local que suporte um subconjunto dos serviços definidos pela tabela *hash*. No Projeto 2 implementaram-se as funções necessárias para serializar e de-serializar estruturas complexas usando Protocol Buffer, um servidor concretizando a *tabela hash*, e um cliente com uma interface de gestão do conteúdo da *tabela hash*. No Projeto 3 foi criado um sistema concorrente que aceita e processa pedidos de múltiplos clientes em simultâneo através do uso de múltiplas *threads*.

No Projeto 4 iremos suportar tolerância a falhas através de replicação do estado do servidor, seguindo o modelo de replicação *primary/backup* e usando o serviço de coordenação *ZooKeeper* [5]. Mais concretamente, vai ser preciso:

- Implementar coordenação de servidores no ZooKeeper, de forma a suportar o modelo de replicação *primary/backup*;
- Alterar o funcionamento do servidor para:
    - No momento do arranque, verificar se já existe um servidor primário (*primary*) a funcionar (utilizando o ZooKeeper).
        - Em caso afirmativo, o servidor assume o papel de servidor de *backup*. O *backup* apenas aceita operações vindas do servidor primário.
        - Em caso negativo, o servidor assume o papel de servidor primário. O servidor primário será o único a servir os clientes. As operações de escrita (e só estas) devem ser propagadas para o servidor de *backup*.
    - No caso de ser servidor primário:
        - Fazer *watch* no *ZooKeeper* de forma a ser notificado se houver alguma alteração relativamente ao servidor de *backup* (falha ou arranque do servidor).
        - Apenas executar operações de escrita se existir um servidor de *backup* ativo. Se não existir, deve enviar uma mensagem de erro ao cliente (pode ser definido um novo código de erro).
        - Quando receber uma operação de escrita, enviá-la para o servidor de *backup*, para fazer a replicação. Quando receber confirmação do servidor de *backup*, executar também a operação e enviar confirmação

ao cliente. Se a operação no servidor de *backup* falhar, retornar erro ao cliente sem executar a operação.

- No caso de ser servidor de *backup*:
  - Fazer *watch* no *ZooKeeper* de forma a ser notificado se o servidor primário deixar de estar ativo. Se isso acontecer, o servidor de *backup* deve assumir o papel de servidor primário.
  - Aceitar pedidos apenas do servidor primário, que deverão ser apenas pedidos de escrita.
  - Quando receber um pedido de ligação de um cliente, terminar de imediato essa ligação.

- Alterar funcionamento do cliente para:
  - Perguntar ao *ZooKeeper* a informação sobre o endereço do servidor primário, que será contactado para realizar todas as operações.
  - Fazer *watch* no *ZooKeeper* de forma a ser notificado de alterações nos servidores, realizando uma nova ligação no caso do servidor primário ter falhado e de existir um novo.

Como nos projetos anteriores, espera-se uma grande fiabilidade por parte do servidor e cliente, portanto não podem existir condições de erro não verificadas ou gestão de memória ineficiente a fim de evitar que estes sofram um *crash*.

2. Descrição Detalhada

O objetivo específico do projeto 4 é desenvolver um sistema de replicação com base no modelo de replicação *primary/backup*. Para tal, para além de aproveitarem o código do servidor e cliente desenvolvido nos projetos 1, 2 e 3, os alunos devem fazer uso de novas técnicas ensinadas nas aulas, incluindo o serviço *ZooKeeper* [5] para coordenação de sistemas distribuídos. A figura 1 ilustra a arquitetura final do sistema a desenvolver.

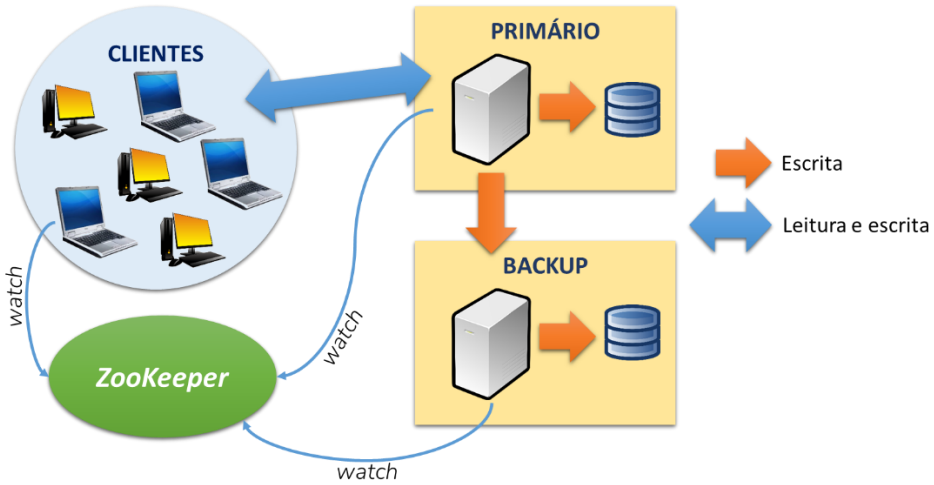

Figura 1- Arquitetura geral do projeto 4.

Neste modelo de replicação os clientes contactam apenas o servidor primário para realizar as operações na tabela, quer sejam operações de escrita (*put*, *delete*) ou de leitura (*get*, *size*, *getkeys*, *table_print*, *stats*). Desta forma, o servidor primário funciona como um sequenciador, determinando a ordem pela qual as operações de escrita (que mudam o estado) são realizadas, garantindo que esta ordem é a mesma no servidor primário e no servidor backup (ou seja, garantindo que o estado dos dois servidores é coerente).

Quando recebe uma operação de escrita, o servidor primário deve começar por garantir que a operação é realizada em exclusão mútua (não esquecer que existem várias *threads* que podem estar a tratar pedidos de vários clientes em paralelo), deve depois enviar a operação para ser

executada no servidor de *backup*, e quando receber a confirmação de que a operação foi executada com sucesso, então executa a operação localmente e responde ao cliente.

Se receber uma operação de leitura, então executa a operação localmente sem contactar o servidor de *backup*, mas garantindo sempre que nenhuma operação de escrita é realizada de forma concorrente.

2.1. ZooKeeper

Um elemento central na arquitetura anterior é o *ZooKeeper*, pois irá gerir a disponibilidade dos servidores e irá notificar tanto clientes como servidores de alterações no sistema distribuído. O *ZooKeeper* será usado para se manter uma visão completa do sistema (por exemplo, saber que servidores estão ativos, e quais os seus endereços IP e portos), estando sempre disponível para registar todas as alterações e informar os clientes e/ou os servidores sobre as mesmas. Assim, tanto os clientes como os servidores apenas terão de conhecer a localização/IP do *ZooKeeper*, ligando-se a este para obter as informações sobre o estado e configuração do sistema.

O *ZooKeeper* pode ser descrito como um serviço (eventualmente replicado, embora neste projeto se use apenas um servidor) que armazena informação organizada de forma hierárquica, em nós que são designados *ZNodes*. A imagem seguinte representa uma solução possível para implementar a coordenação necessária para a *key-value store* (kvstore) replicada através do *ZooKeeper*.

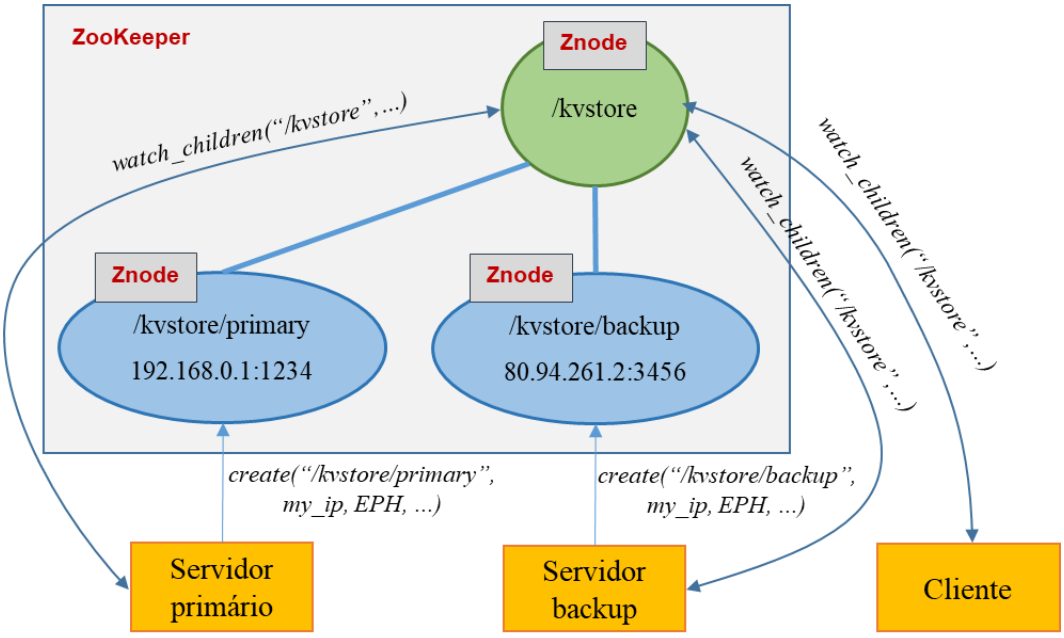

Figura 2 - Modelo de dados do ZooKeeper para replicação *primary/backup*.

Na Figura 2 existem dois tipos de *ZNodes*: o *ZNode* "/*kvstore*" e os seus *ZNodes* filhos "/*primary*" e "/*backup*". O /*kvstore* é um *ZNode* normal. É criado pelo primeiro servidor que se ligar ao *ZooKeeper*, se ainda não existir, e deve continuar a existir mesmo que todos os servidores se deliguem do *ZooKeeper*. Quando um servidor inicializa, ele contacta o *ZooKeeper* e verifica os filhos de /*kvstore*. Se existirem, ambos */kvstore/primary* e */kvstore/backup*, o servidor deve terminar. Se não existir nenhum filho, ele deve criar o nó */kvstore/primary* e assumir o papel de servidor primário. Se existir apenas */kvstore/primary*, ele deve criar o nó */kvstore/backup* e assumir o papel de *backup*. Se existir apenas */kvstore/backup*, deve aguardar um pouco e voltar a verificar os filhos de /*kvstore*, de modo a dar tempo ao servidor de *backup* existente de se autopromover a primário. Aquando da criação dos nós, os servidores devem armazenar nos mesmos o seu endereço IP e porto TCP como meta-dados.

Esta informação será usada pelos clientes e pelo servidor primário para estabelecer as ligações necessárias (os clientes ligam-se ao primário, e o primário liga-se ao *backup*).

Como também pretendemos lidar com falhas dos servidores e detetar as mesmas de forma automática, vamos criar os nós */kvstore/primary* e */kvstore/backup* como *ZNodes* efémeros. Assim, se um servidor falhar, o *ZooKeeper* deteta que a ligação entre os dois foi interrompida e remove o seu nó da lista de filhos de /*kvstore*, notificando todos os servidores e clientes que fizeram *watch* aos filhos de /*kvstore*. Isto significa que todos os servidores e clientes, quando arrancam, devem contactar o *ZooKeeper* de forma a estabelecer esse *watch*.

2.2. Mudanças a efetuar no servidor

O servidor passa a guardar uma ligação ao *ZooKeeper*, bem como a identificação do outro servidor (*primary* ou *backup*) no sistema replicado, se este estiver ativo. Se o servidor estiver a funcionar como primário, então terá também de ter um *socket* para comunicar com o servidor de *backup*, logo que este esteja ativo. A forma como esta informação é guardada fica ao critério dos alunos.

Quando um servidor é iniciado e assume o papel de *backup*, é possível que a tabela do servidor primário já contenha informação. Assim, é necessário realizar uma *transferência de estado* do primário para o *backup*, antes do primário começar a aceitar pedidos de escrita. Esta transferência pode ser feita de uma forma bastante simples, utilizando a função *getkeys* para obter todas as chaves, e depois, para cada uma das chaves, usando as funções *get* (na tabela local) e *put* (na tabela do servidor de *backup*).

Os novos passos a implementar na lógica do módulo table_skel.c, quando um servidor inicia, são os seguintes:

- Ligar ao *ZooKeeper*;
- Se não existir o *Znode* /*kvstore*, criar esse *Znode* normal e criar o nó efémero /*kvstore*/*primary*, assumindo-se como servidor primário;
- Se o *Znode /kvstore* existir e tiver nós filhos /*primary* e /*backup*, terminar;
- Se não existirem nós filhos de /*kvstore*, criar o nó efémero /*kvstore*/*primary*, assumindo-se como servidor primário;
- Se existir o nó filho /*primary* e não existir o nó filho /*backup*, criar o nó efémero /*kvstore*/*backup*, assumindo-se como servidor de *backup*;
- Se existir o nó filho /*backup* e não existir o nó filho /*primary*, esperar um pouco e voltar a consultar o *ZooKeeper*, repetindo o processo;
- Obter e fazer *watch* aos filhos de */kvstore;*
- Se for servidor de *backup*, guardar os meta-dados (IP:porto) do servidor primário;
- Se for servidor primário, deixar a variável com os meta-dados do servidor de *backup* a NULL.

Adicionalmente:

- Quando o *watch* de filhos de */kvstore* é ativado, verificar se houve uma saída ou entrada de algum servidor:
    - Se for servidor primário e o *backup* tiver saído, não aceita mais pedidos de escrita dos clientes até que volte a haver *backup*. Volta a ativar o *watch*.
    - Se for servidor *backup* e o primário tiver saído, autopromove-se a servidor primário. Volta a ativar o *watch*.
    - Se for servidor primário e houve ativação do servidor de *backup*, guarda o seu par IP:porta, estabelece ligação, transfere todas as entradas da tabela para o servidor de *backup* (realizando uma sequência de operações *put*), e volta a aceitar pedidos de escrita dos clientes. Volta a ativar o *watch*.

- Fazer com que as *threads* secundárias, depois de obterem permissão de execução da uma escrita em exclusão mútua, enviem essa operação de escrita para o servidor de *backup*, de forma a replicar a mesma. Se a operação for realizada com sucesso, deve então ser realizada localmente. Só depois da escrita ter sido feita no servidor de *backup* e localmente é que será possível realizar outras operações de leitura ou escrita.
- Se for recebido um pedido de escrita (*put*, *get*) e não existir nenhum servidor de *backup* ativo, o servidor primário não deve executar a operação e deve devolver erro.

Neste novo modelo, o table-server passa a receber dois argumentos na linha de comandos: o que já recebia antes, mais o IP e porta do ZooKeeper - <IP>:<porta>.

2.3. Mudanças a efetuar no cliente

O cliente, nomeadamente o client_stub.c, passa a ligar-se ao *ZooKeeper* e ao servidor primário. A estrutura *rtable_t* poderá ser usada para armazenar as informações necessárias, nomeadamente o *handle* de ligação ao *ZooKeeper* e IP:porto do servidor primário.

Os novos passos a implementar na lógica do client_stub.c, quando um cliente inicia, são os seguintes:

- Ligar ao *ZooKeeper*;
- Obter e fazer *watch* aos filhos de */kvstore;*
- Ler o *ZNode /kvstore/primary* do *ZooKeeper* para obter o par IP:porto do servidor primário, guardando esta informação e estabelecendo a respetiva ligação. Se o *ZNode* não existir, terminar o cliente.

Adicionalmente:

- Quando o *watch* de filhos de */kvstore* é ativado, verificar se houve alguma saída ou entrada:
    - Se saiu o servidor primário, fechar a ligação que existia ao servidor, e atualizar a informação armazenada na estrutura rtable_t; informar o utilizador sobre a impossibilidade de executar operações na tabela; voltar a ativar o *watch;*
    - Se entrou um servidor primário, atualizar a informação relativamente ao IP:porto do servidor, iniciar uma nova ligação, e voltar a permitir a execução de operações na tabela; voltar a ativar o *watch;*

Neste modelo, o IP e porto introduzidos pelo utilizador passam a ser o IP e porto do *ZooKeeper*.

3. Makefile

Os alunos deverão manter o `Makefile` usado no projeto 3, atualizando-o para compilar novo código, se necessário.

4. Entrega

A entrega do projeto 4 tem de ser feita de acordo com as seguintes regras:

1. 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).
2. Submeter o ficheiro **`grupoXX-projeto4.zip`** na página da disciplina no moodle da FCUL, utilizando a atividade disponibilizada para tal. Apenas um dos elementos do grupo deve submeter e todos os elementos têm de confirmar a submissão.

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

- o ficheiro README, onde os alunos podem incluir informações que julguem necessárias (e.g., limitações na implementação);
- diretorias adicionais, nomeadamente:
    - include: para armazenar os ficheiros .h;
    - source: para armazenar os ficheiros *.c;*
    - object: para armazenar os ficheiros objeto;
    - lib: para armazenar bibliotecas;
    - binary: para armazenar os ficheiros executáveis.
- um ficheiro `Makefile` que satisfaça os requisitos descritos. Não devem ser incluídos no ficheiro ZIP os ficheiros objeto (.o) ou executáveis. Quaisquer outros ficheiros (por exemplo, de teste) também não deverão ser incluídos no ficheiro ZIP.

Na entrega do trabalho, é ainda necessário ter em conta que:

- **Se não for incluído um `Makefile`, se o mesmo não satisfizer os requisitos indicados, 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 disciplina, no Moodle, 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 um cabeçalho com três ou quatro linhas de comentários a dizer o número do grupo e o nome e número dos 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 dia 13/12/2021 até às 23:59hs.

Após esta data, a submissão do trabalho através do Moodle deixará de ser permitida. Também, cada grupo ficará sem acesso de escrita à diretoria de entrega.

5. Bibliografia

[1] Giuseppe DeCandia et al. *Dynamo: Amazon's Highly Available Key-value Store*. Proc. of the 21st Symposium on Operating System Principles – SOSP'07. pp. 205-220. Out. de 2007.

[2] Wikipedia. Linked List. https://en.wikipedia.org/wiki/Linked_list.

[3] Wikipedia. Hash Table. http://en.wikipedia.org/wiki/Hash_table.

[4] B. W. Kernighan, D. M. Ritchie, C Programming Language, 2nd Ed, Prentice-Hall, 1988.

[5] https://zookeeper.apache.org/