Ciências ULisboa | Informática

2019/20

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, as estruturas de dados utilizadas para armazenar esta informação são 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 Projecto 3 foram suportados múltiplos clientes através da chamada ao sistema *Poll* e foi separado o tratamento de I/O do processamento de dados através do uso de Threads.

No Projeto 4 iremos suportar tolerância a falhas através de replicação do estado do servidor, seguindo o modelo *Chain Replication* (Replicação em Cadeia) [5] e usando o serviço de coordenação ZooKeeper [6]. Mais concretamente, vai ser preciso:

- Implementar coordenação de servidores no ZooKeeper, de forma a suportar o modelo de replicação em cadeia (*Chain Replication*);
- Alterar funcionamento do servidor para:
  - Perguntar ao ZooKeeper qual o próximo servidor a seguir a este na cadeia de replicação;
  - Depois de executar uma operação de escrita, enviá-la para o próximo servidor de forma a propagar a replicação;
  - Fazer *watch* no ZooKeeper de forma a ser notificado de alterações na cadeia e ligar-se ao seu novo servidor, caso este tenha mudado.
- Alterar funcionamento do cliente para:
  - Perguntar ao ZooKeeper que servidores estão à cabeça e à cauda da cadeia;
  - Mandar operações de escrita para o servidor que está à cabeça da cadeia;
  - Mandar operações de leitura para o servidor que está na cauda da cadeia;
  - Fazer *watch* no ZooKeeper de forma a ser notificado de alterações na cadeia e ligar-se à nova cabeça e cauda, se estes tiverem mudado.

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.

2. Descrição Detalhada

O objetivo específico do projeto 4 é desenvolver um sistema de *Chain Replication* (Replicação em Cadeia) [5] com múltiplos clientes e servidores. Para tal, para além de aproveitarem o código desenvolvido nos projetos 1, 2 e 3, os alunos devem fazer uso de novas técnicas ensinadas nas aulas, incluindo o serviço ZooKeeper [6] para coordenação de sistemas distribuídos. A figura abaixo ilustra a arquitetura final do sistema a desenvolver.

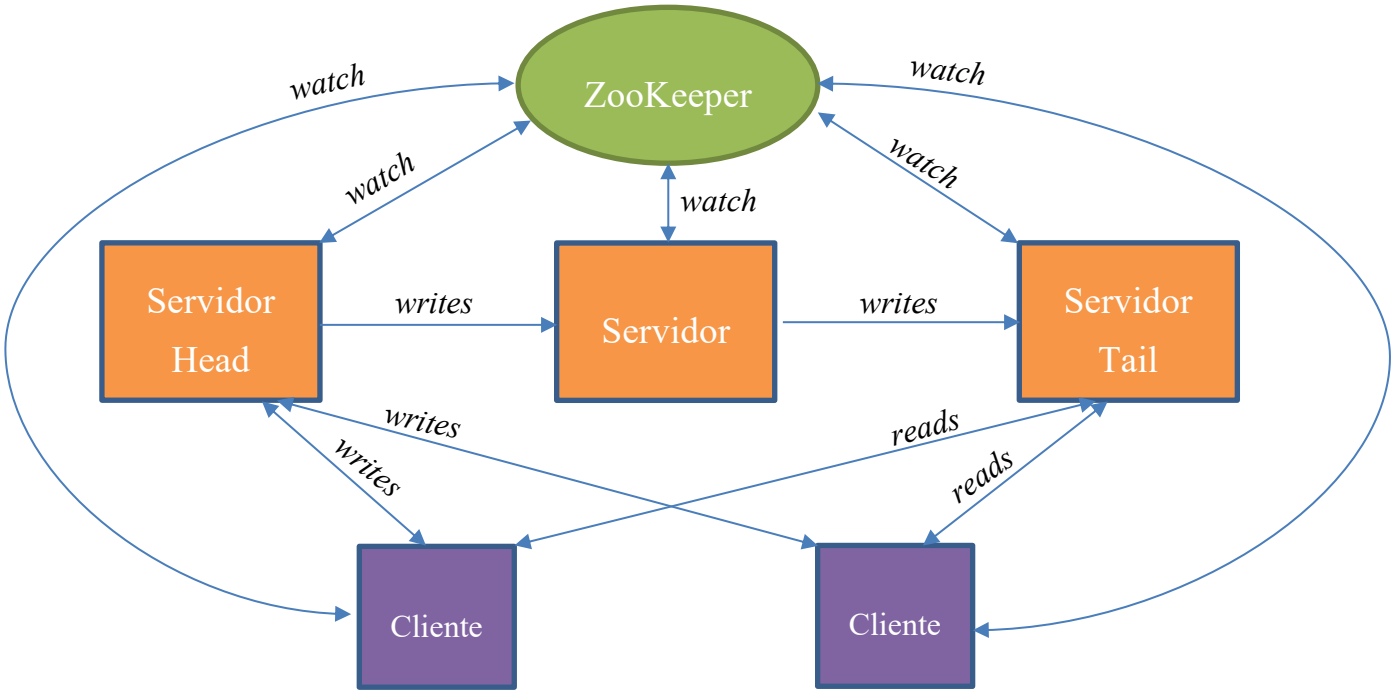

Figura 1. Arquitetura geral do Projeto 4

Num modelo de replicação em cadeia, os servidores ligam-se entre si formando uma sequência, i.e. cada servidor apenas comunica com um outro servidor, que é o que lhe segue na cadeia. Todas as operações de mudança de estado, i.e. operações de escrita (*put*, *delete*), sejam enviadas por clientes ou pelo servidor que lhe antecede na cadeia, são enviadas para o próximo servidor depois de serem executadas localmente. Adicionalmente, de forma a garantir que todos os servidores recebem as mesmas operações e guardam o mesmo estado (consistência dos dados), os clientes devem enviar todas as operações de escrita para o servidor que está à cabeça da cadeia, sendo estas então propagadas pelos servidores até chegarem ao servidor que está na cauda. Para garantir que quando leem um estado, este estado já foi replicado por todos os servidores, os clientes enviam as operações de leitura (*get*, *size*, *getkeys*, *verify*) para o servidor que está na cauda. Inclusive, a operação *verify*, quando executada na cauda, passa a permitir verificar que uma operação já foi ou não propagada por toda a cadeia de servidores.

2.1. ZooKeeper

Um elemento central na arquitetura anterior é o ZooKeeper, pois irá gerir a disponibilidade de todos os servidores e irá notificar tanto clientes como servidores de alterações no sistema distribuído. Através do uso do ZooKeeper, tanto os clientes como os servidores precisam de manter uma visão parcial do sistema (nomeadamente, só precisam de conhecer a localização/IP do ZooKeeper), delegando no ZooKeeper a responsabilidade de manter uma visão completa do sistema e de se manter sempre disponível para registar todas as alterações e informar os clientes e/ou os servidores sobre as mesmas.

No entanto, o ZooKeeper é uma ferramenta bastante flexível, que deve ser configurada de forma a fornecer a funcionalidade que se pretende para cada aplicação. Essa configuração é feita através dos ZNodes. A imagem seguinte representa uma solução possível para implementar a coordenação necessária em Chain Replication através do ZooKeeper.

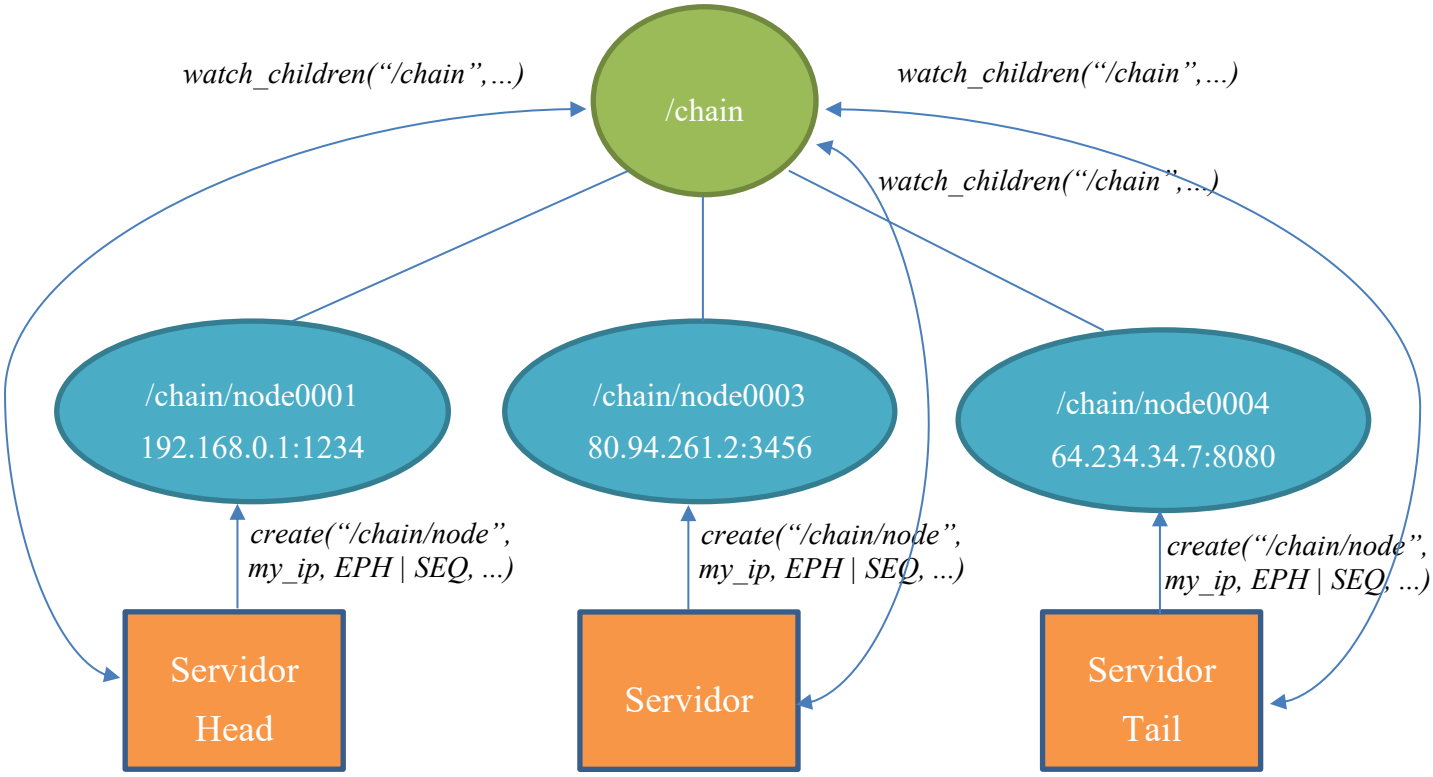

Figura 2. Modelo de dados do ZooKeeper para Chain Replication

Na figura anterior existem dois tipos de ZNodes: a "/*chain*" e os seus filhos "/*node*". A /*chain* é um ZNode normal. É criada 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. Os */node* são os ZNodes filhos de *chain*. Existe um */node* para cada servidor do nosso sistema. Quando um servidor inicializa e contacta o ZooKeeper, ele pede para criar um novo ZNode filho de /*chain* (i.e. o seu /*node*) com o seu IP e porta como meta-dados. O IP e porta servirão para outros servidores e clientes se poderem ligar a ele.

Como queremos uma ordenação global entre servidores de forma a que exista sempre um (e apenas um) servidor *head* e um *tail*, vamos criar os *node* como ZNodes sequenciais. Assim, o ZooKeeper atribui um número de sequência único e crescente a cada /*node*. Quando se obtem a listagem de filhos de /*chain*, podemos ordenar os mesmos por ordem lexicográfica.

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

Resumindo: deve haver um ZNode normal /*chain*; todos os servidores e clientes devem fazer *watch* aos filhos de /*chain*; e cada servidor, quando inicializa, deve criar um ZNode efémero e sequencial, chamado *node* e guardando o IP e porta desse servidor, como filho de /*chain*.

2.2. Mudanças a efetuar no servidor

O servidor, nomeadamente o table_skel.c, passa a guardar: uma ligação ao ZooKeeper; o identificador do seu *node* no ZooKeeper; e o identificador do *node* do próximo servidor na cadeia de replicação, assim como um socket para comunicação com o mesmo. Para tal os alunos podem reutilizar a estrutura *rtable*, modificando-a para guardar as novas informações necessárias.

Novos passos a implementar na lógica do table_skel.c, quando um servidor inicia:

- Ligar ao ZooKeeper;
- Criar novo ZNode efémero sequencial no ZooKeeper, filho de */chain*, como descrito na Secção 2.1;
- Guardar o id atribuído ao ZNode pelo ZooKeeper;
- Obter e fazer *watch* aos filhos de */chain;*
- Ver qual é o servidor com id mais alto a seguir ao nosso, de entre os filhos de */chain;*
- Obter os meta-dados desse servidor do ZooKeeper (i.e. o seu IP);
- Guardar esse servidor como *next_server*, ou deixar a variável a NULL se o nosso id for o mais alto (quer dizer que nós somos a cauda da cadeia).

Adicionalmente:

- Quando *watch* de filhos de */chain* é ativada, ver qual é o servidor com id mais alto a seguir ao nosso, pedir o seu IP ao ZooKeeper, guardar e ligar a ele se o *next_server* tiver mudado, e voltar a ativar a *watch;*
- Fazer com que a thread secundária, depois de executar uma tarefa, envie essa tarefa para o *next_server*, de forma a propagar a replicação das operações dos clientes.

Neste novo modelo, o table-server passa a receber três argumentos na linha de comandos: os dois 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 a dois servidores, a cabeça e a cauda da cadeia de replicação. Para tal, os alunos podem usar a estrutura *server_t* descrita na Secção 2.2 e as variáveis *server_t head* e *server_t tail*.

Novos passos a implementar na lógica do client_stub.c, quando um cliente inicia:

- Ligar ao ZooKeeper;
- Obter e fazer *watch* aos filhos de */chain;*
- Dos filhos de */chain*, obter do ZooKeeper o IP do que tem id mais baixo e do que tem id mais alto, guardá-los como *head* e *tail* respectivamente, e ligar a eles;

Adicionalmente:

- Quando *watch* de filhos de */chain* é ativada, ver quais são os servidores com id mais baixo e com id mais alto, pedir os seus IPs ao ZooKeeper, guardá-los como *head* e *tail* respetivamente, ligar a eles se tiverem mudado, e voltar a ativar a *watch;*
- O cliente passa a enviar:
    - Pedidos de escrita (put e delete) para a *head*, de forma a serem propagados por toda a cadeia;
    - Pedidos de leitura (get, size, getkeys, e verify) para a *tail*, de forma a garantir que o estado que é obtido já foi replicado por todos os servidores da cadeia;

Neste modelo, o IP e porta introduzidos pelo utilizador passam a ser o IP e porta do ZooKeeper. Adicionalmente, o cliente pode, por exemplo, enviar um put para a *head*, esperar um dado timeout e depois fazer verify dessa operação na *tail* para garantir que a operação foi replicada por todos os servidores. Se a verificação da operação falhar na *tail* (e.g. um servidor a meio da cadeia falhou), ele volta a tentar executar a operação na *head*. Mesmo que metade dos servidores tenham executado a operação, mas a outra metade não tenha, não há problema, porque na nossa aplicação executar a mesma operação duas vezes tem o mesmo efeito que executar apenas uma vez.

3. Makefile

Os alunos deverão manter o `Makefile` usado no Projecto 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 15/12/2019 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] R. V. Renesse and F. B. Schneider. Chain Replication for Supporting High Throughput and Availability. OSDI. Vol. 4. No. 91–104. 2004.

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