Ciências ULisboa | Informática

**2020/21**

# 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 [3], 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 árvore de pesquisa binária [2], 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 árvore. No Projeto 2 implementaram-se as funções necessárias para serializar e de-serializar estruturas complexas usando *Protocol Buffer*, um servidor concretizando a árvore binária, e um cliente com uma interface de gestão do conteúdo da *árvore binária*. No projeto 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 básico de replicação passiva com primário fixo e usando o serviço de coordenação *ZooKeeper* [4]. Mais concretamente, vai ser preciso:

- Implementar coordenação de servidores no ZooKeeper, de forma a suportar o modelo de replicação passiva com primário fixo;
- Alterar funcionamento do servidor para:
  - No momento do arranque é necessário verificar se já existe um servidor primário a funcionar (utilizando ZooKeeper).
    - Em caso afirmativo: o servidor assume o papel de servidor de *backup*. O *backup* apenas aceita operações de escrita vindas do servidor primário e operações de leitura vindas dos clientes.
    - Em caso negativo: o servidor assume o papel de servidor primário. O servidor primário apenas aceita operações de escrita provenientes dos clientes. As operações de escrita devem ser propagadas para o servidor *backup*.
  - No caso de ser servidor primário:
    - Fazer *watch* no *ZooKeeper* de forma a ser notificado de alteração de servidor *backup*, e ligar-se ao novo servidor caso este tenha mudado.
    - Apenas executa 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).
    - Depois de executar com sucesso uma operação de escrita, deverá enviá-la para o servidor de *backup* para fazer a replicação. Quando

receber confirmação ou erro do servidor de *backup*, envia essa confirmação ou erro ao cliente.
    - 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.
        - Apenas aceita pedidos de escrita do servidor primário e aceita pedidos de leitura dos clientes. Caso um cliente envie um pedido de escrita, dever-lhe-á ser enviada uma mensagem de erro (pode ser definido um novo código de erro).
- Alterar funcionamento do cliente para:
    - Perguntar ao *ZooKeeper* a informação sobre os endereços do servidor primário e do servidor de backup.
    - Enviar operações de escrita para o servidor primário.
    - Enviar operações de leitura para o servidor de *backup*.
    - Fazer *watch* no *ZooKeeper* de forma a ser notificado de alterações nos servidores.

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 replicação com base no modelo de replicação passiva com primário fixo (visto na aula 14 – Replicação). 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* [4] para coordenação de sistemas distribuídos. A figura 1 ilustra a arquitetura final do sistema a desenvolver.

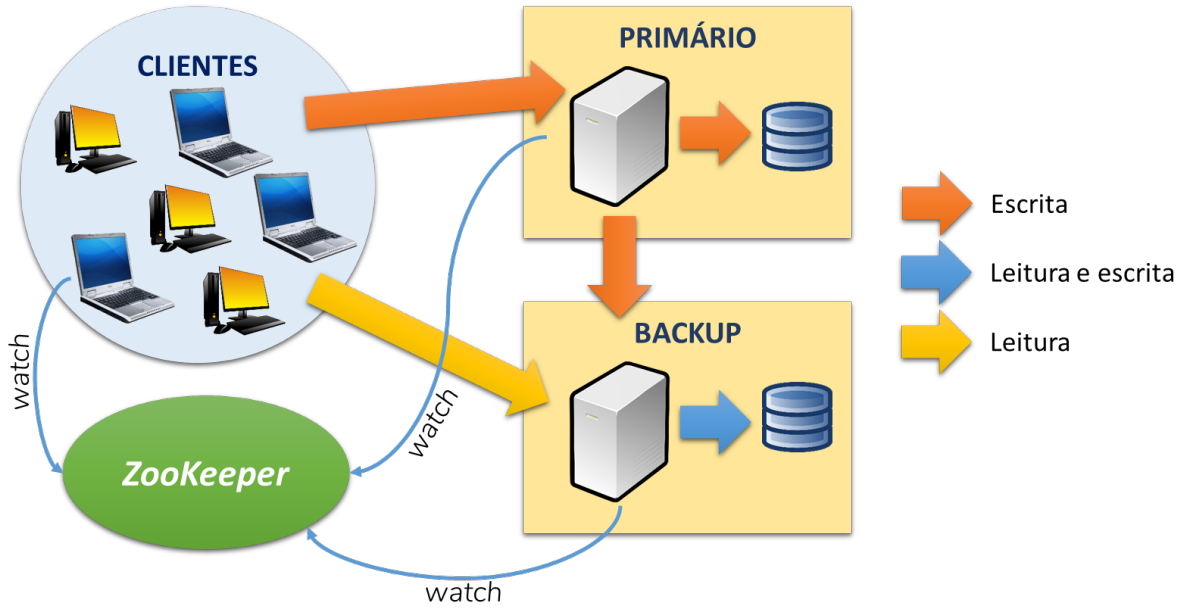


Figura 1- Arquitetura geral do Projeto 4

Neste modelo de replicação as operações de mudança de estado, i.e. operações de escrita (*put*, *delete*), são enviadas apenas ao servidor primário. Este, após execução local da operação, em caso de sucesso replica a operação para o servidor de *backup* (ou secundário). Para garantir que quando leem um estado, este estado já foi replicado, os clientes enviam as operações de leitura (*get*, *size*, *getkeys*, *verify*) para o servidor de *backup*. Inclusive, a operação *verify*, quando executada no *backup*, passa a permitir verificar que uma operação já foi ou não propagada para o servidor secundário.

### 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. 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 para a *key-value store* (kvstore) replicada através do *ZooKeeper*.

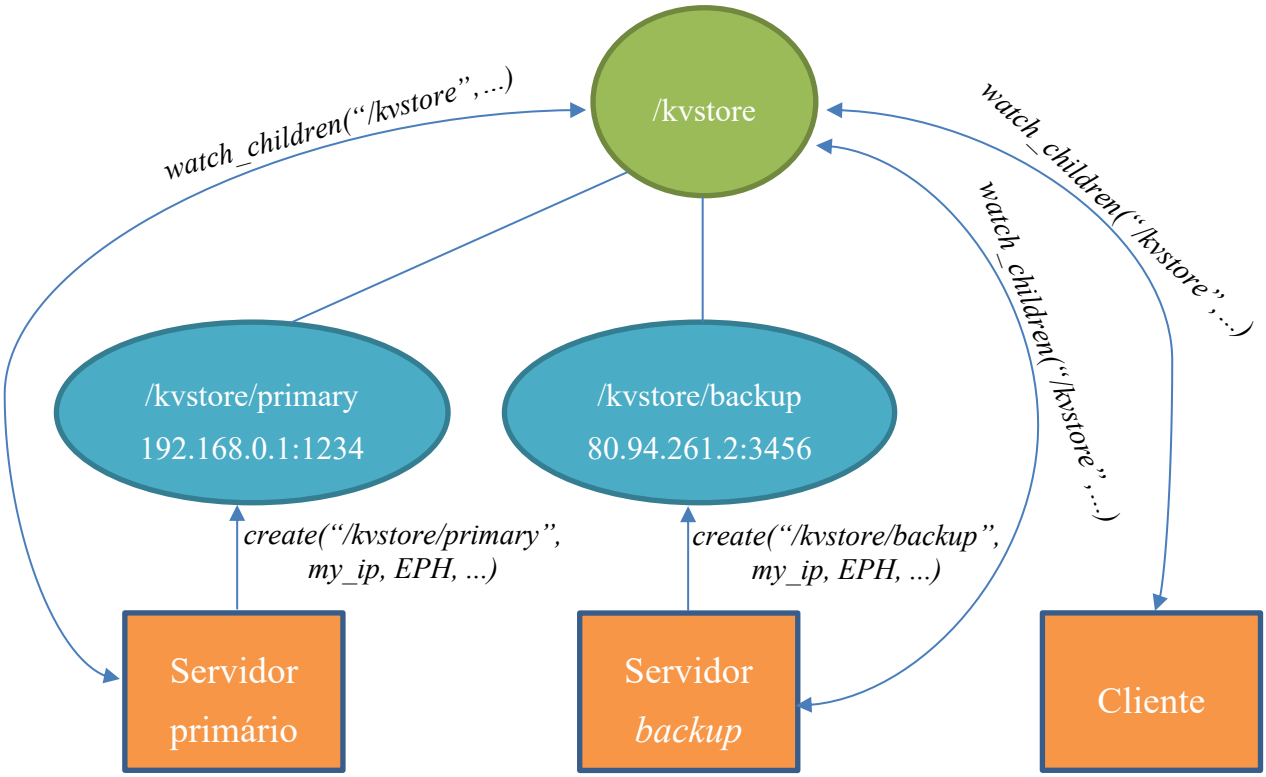


Figura 2. Modelo de dados do *ZooKeeper* para replicação passiva

Na figura anterior existem dois tipos de *ZNodes*: a "/*kvstore*" e os seus filhos "/*primary*" e "/*backup*". A /*kvstore* 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*. Quando um servidor inicializa, ele contacta o ZooKeeper e verifica os filhos de /*kvstore*. Se existirem, ambos /*primary* e /*backup*, o servidor deve terminar. Se não existir nenhum filho, ele deve criar o nó /primary e assumir o papel de servidor primário. Se existir apenas /primary, ele deve criar o node /*backup* e assumir o papel de *backup*. Se existir apenas /backup, ele deve aguardar (em princípio não deve acontecer porque o *backup* ao detetar que não há primário autopromove-se a primário). Aquando da criação dos nós, os servidores devem armazenar nos mesmos o seu IP e porta como meta-dados. O IP e porta servirão para outros servidores e clientes se poderem ligar a eles.

Como também pretendemos lidar com falhas dos servidores e detetar as mesmas de forma automática, vamos criar os nós 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, nomeadamente o tree_skel.c, passa a guardar: uma ligação ao *ZooKeeper*; o identificador do seu *node* no *ZooKeeper*; e o identificador do *node* do outro servidor (*primary* ou *backup*) no sistema replicado, bem como um *socket* para comunicação com o mesmo. Para tal os alunos podem reutilizar a estrutura *rtree*, modificando-a para guardar as novas informações necessárias.

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

- Ligar ao ZooKeeper;
- Se não existir o *Znode* /*kvstore*, criar esse *Znode* normal e criar o nó efémero /*kvstore*/*primary*, assumindo-se este servidor como 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 este servidor como primário;
- Se existir o nó filho /*primary* e não existir o nó filho /*backup*, criar o nó efémero /*kvstore*/*backup*, assumindo-se este servidor como *backup*;
- Obter e fazer *watch* aos filhos de */kvstore;*
- Obter os meta-dados do outro servidor no sistema;
- Guardar esse servidor como primário ou backup (consoante seja backup ou primário), ou deixar a variável a NULL se o servidor for primário e ainda não existir backup.

Adicionalmente:

- Quando *watch* de filhos de */kvstore* é ativada, 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 *watch*.
    - Se for servidor *backup* e o primário tiver saído, autopromove-se a servidor primário. Volta a ativar *watch*.
    - Se for servidor primário e houve ativação de *backup*, guarda o seu par IP:porta, estabelece ligação e volta a aceitar pedidos de escrita dos clientes. Volta a ativar *watch*.
- Fazer com que a thread secundária, depois de executar uma tarefa de escrita, envie essa tarefa para o servidor de *backup*, de forma a propagar a replicação das operações dos clientes.

Neste novo modelo, o tree-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 a dois servidores, o primário para oprações de escrita e o *backup* para operações de leitura. Para tal, os alunos podem usar a estrutura *rtree* como descrito na Secção 2.2 e as variáveis *rtree primary* e *rtree backup*.

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

- Ligar ao ZooKeeper;
- Obter e fazer *watch* aos filhos de */kvstore;*
- Dos filhos de */kvstore*, obter do ZooKeeper os pares IP:porta dos servidores primário e secundário, guardá-los como *primary* e *backup* respectivamente, e ligar a eles;

Adicionalmente:

- Quando *watch* de filhos de */kvstore* é ativada, verificar se houve alguma saída ou entrada:
  - Se saiu um dos servidores, atualizar a estrutura de dados correspondente, e não enviar comandos para os servidores; voltar a ativar a *watch;*
  - Se entrou um servidor, atualizar a estrutura de dados correspondente, e, no caso de haver primário e *backup* ativos, voltar a enviar comandos aos servidores; voltar a ativar a *watch;*
- O cliente passa a enviar:
  - Pedidos de escrita (put e delete) para o *primary*, de forma a serem propagados para o *backup*;
  - Pedidos de leitura (get, size, getkeys, e verify) para o *backup*, de forma a garantir que o estado que é obtido já foi replicado;

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 o *primary*, esperar um dado *timeout* e depois fazer *verify* dessa operação no *backup* para garantir que a operação foi replicada. Se a verificação da operação falhar no *backup*, ele volta a tentar executar a operação no *primary*. Mesmo que a operação entretanto tenha executado, não há problema porque na nossa aplicação executar a mesma operação duas vezes tem o mesmo efeito que executar apenas uma vez (operações idempotentes).

## 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 13/12/2020 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 . *Binary Search Tree*. https://en.wikipedia.org/wiki/Binary_search_tree

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

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