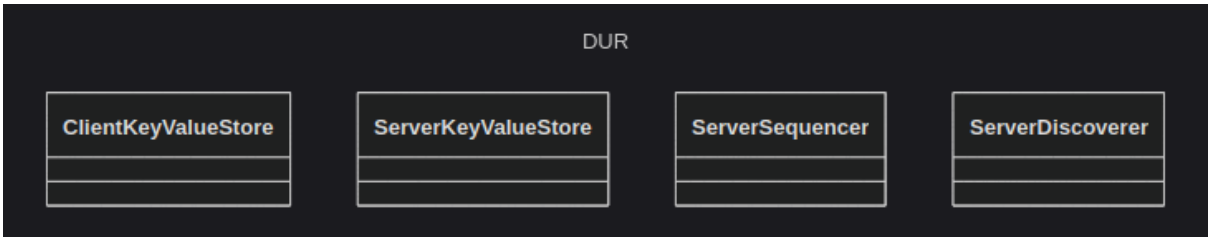
DUR

Disciplina: Computação Distribuída - INE5418
Nome do grupo: Grupo
Integrantes: Eduardo de Moraes (19203167) e Stephan Krug (21100141)
Repositório: https://github.com/stephankruggg/dur

1. Estrutura

O DUR é composto pelos seguintes componentes:



- O Server Discoverer (SD) é um servidor responsável pelas funções de entrada e saída dos servidores Key-Value Store (SKVS) do sistema. Adicionalmente, por meio dele os clientes podem buscar quais servidores estão disponíveis para a leitura de valores e para o commit das transações;
- O Server Sequencer (SS) é um servidor responsável por ordenar as transações a serem executadas pelos servidores Key-Value Store (SKVS) do sistema. Ele garante que, independentemente da ordem de chegada das requisições de confirmação (commit) dos clientes, elas sempre sejam executadas na mesma ordem em todos os servidores;
- O Server Key-Value Store (SKVS) é o servidor responsável por guardar o banco de dados chave-valor no qual o cliente consulta e escreve. Esse componente pode ser replicado sem que haja perda de segurança e validade das transações, afinal todos executarão as transações na mesma sequência;
- O Client Key-Value Store (CKVS) é uma interface com a qual o usuário pode se comunicar para requisitar valores aos servidores, realizar escritas, enviar confirmações e abortar transações.

2. Funcionamento

2.1. Requisitos

Para o funcionamento correto do sistema, é necessário que exista um componente de cada tipo descrito na seção 1 em execução. Caso contrário, não será possível realizar transações completamente.

2.2. Inicialização

Na inicialização do SD, ele cria um socket responsável por ouvir requisições, que podem ser dos tipos: conexão de um servidor, desconexão de um servidor ou busca de todos os servidores cadastrados.

Na inicialização do SS, ele realiza as seguintes ações:

- Conexão ao SD, de forma a se tornar visível aos clientes;
- Criação de um socket para recebimento de requisições de clientes por números de sequência, que serão posteriormente enviados a todos os SKVS.

Na inicialização dos SKVS, eles realizam a seguinte série de ações:

- Cada servidor cria um arquivo “db” no diretório server<id-SKVS>, no qual serão armazenados os dados do seu Key-Value Store. Caso haja um arquivo base no diretório de exemplo com o nome “data”, ele servirá como base para a criação do arquivo “db”. Nesse caso, todos os dados de “data” serão transportados ao “db”. Caso contrário, o arquivo “db” será criado sem dados;
- Cada um realiza a conexão com o server discoverer para que se torne visível aos clientes. Essa ação é realizada por meio de uma requisição ao SD, indicando qual o endereço que o servidor quer tornar visível;
- Cria-se um socket responsável por escutar conexões do servidor sequenciador. Nesse caso, sempre que é percebido o envio de um número de sequência das operações, o servidor armazena em um dicionário qual o número de sequência recebido e a qual transação ele se refere;
- Por fim, o servidor cria um socket no qual escuta por requisições de leitura ou commit dos clientes.

Na inicialização dos CKVS, eles realizam as seguintes ações:

- Criação de conjuntos de leitura e escrita vazios, além de configurar o número de transação como 0;
- Escolha de um servidor aleatório entre os servidores disponíveis no SD.

2.3. Read - Leitura

Após a inicialização do sistema, os clientes estão aptos a executarem suas funcionalidades. Dentre elas, está a leitura de um item salvo no servidor. Para isso, o seguinte procedimento é utilizado:

1. O valor para o item é retornado, caso seja encontrado no conjunto de escrita;

2. O valor para o item é retornado, caso seja encontrado no conjunto de leitura;
3. O valor é lido do servidor. Para isso, o CKVS envia uma requisição ao servidor aleatório escolhido na inicialização do sistema. O servidor realiza uma busca no seu Key-Value Store pelo item e, caso seja encontrado, responde o valor e a versão do valor. Esse valor é salvo no conjunto de leitura do cliente. Caso não seja encontrado pelo servidor, ele responde com uma mensagem vazia e o cliente retorna ao usuário que o item não foi encontrado.

2.4. Write - Escrita

A escrita é mais uma das funcionalidades disponíveis no CKVS. A ação de escrita não é comunicada aos servidores, sendo uma simples modificação do dicionário de escrita contido no cliente.

2.5. Abort - Aborto

O aborto é outra função presente no CKVS. Sua ação também não resulta em comunicação ao servidor, sendo realizada apenas a limpeza dos conjuntos de escrita e leitura do cliente. Além disso, o número de transações é incrementado para iniciar a próxima transação.

2.6. Commit - Confirmação

Na confirmação, o cliente realiza as seguintes operações:

1. Criação de um socket para receber a resposta da operação de commit dos servidores;
2. Busca de todos os servidores disponíveis pelo SD;
3. Envio de uma requisição de confirmação a todos os servidores listados. Essa requisição é composta pelo endereço e porta do cliente, ID da transação e os conjuntos de leitura e escrita serializados;
4. Assim que a operação for confirmada ou abortada pelos servidores, a resposta é recebida pelo socket criado no passo 1. Esse socket é, então, fechado e a resposta é repassada ao usuário.

Já os SKVS são responsáveis por:

1. Receber uma requisição de confirmação do cliente;
2. Aguardar o recebimento do número de sequência gerado pelo SS no socket criado durante a inicialização. Para confirmar que um número de sequência corresponde a uma transação, o servidor identifica-a por: endereço, porta e número de transação do cliente;
3. Ao receber o número de sequência, o servidor inicia a execução do algoritmo de confirmação, o qual é descrito a seguir:
    a. Todas as versões presentes no conjunto de leitura do cliente são comparadas às versões presentes atualmente no banco de dados. Caso o cliente tenha lido uma versão menor que a presente no KVS, uma requisição informando o aborto é enviada ao cliente;

b. Caso contrário, o servidor percorre o conjunto de escrita do cliente e realiza as operações listadas. Após finalizar todas as operações, uma requisição informando a confirmação é enviada ao cliente;

Por fim, o SS realiza as funções de:

1. Receber uma requisição de confirmação do cliente;
2. Requisitar ao SD todos os servidores disponíveis atualmente;
3. Enviar a todos os servidores disponíveis o número de sequência atual do SS;
4. Atualizar o número de sequência para futuras requisições.

3. Funcionamento

A especificação de como executar o código e informações complementares estão presentes no README.md do projeto. Abaixo também são listados alguns exemplos de execução.

3.1. Commit de transação

Este exemplo pode ser acompanhado por meio do vídeo.

Nesse exemplo, realizamos a leitura do item “teste”, já presente no banco de dados dos servidores. Um servidor aleatório retorna o valor correspondente ao item, que é posto no conjunto de leitura do cliente juntamente com a sua versão.

Procedemos então à escrita deste mesmo item por um valor diferente do que possuía antes. Além disso, escrevemos também um valor em um item ainda não presente no banco de dados.

Ao realizar o commit da transação, todos os servidores (SS e SKVS) recebem a mensagem. Os SKVS ao receber a mensagem, aguardam o recebimento do número de sequência do SS. A partir do momento em que o SS recebe a mensagem, processa a transação e envia o número de sequência aos servidores.

Os SKVS então realizam a checagem das versões do conjunto de leitura em comparação às versões presentes em seus bancos de dados. Como nesse caso os valores lidos estão atualizados, eles procedem à escrita dos valores do conjunto de escrita no banco de dados. Por fim, após realizarem todas as operações, enviam uma requisição ao cliente confirmando a transação, que por sua vez informa ao usuário.

3.2. Commit em um cliente + Abort em outro

Este exemplo pode ser acompanhado por meio do vídeo.

Nesse exemplo, realizamos a leitura do item “teste”, já presente no banco de dados dos servidores em dois clientes diferentes. Para cada um é retornado o valor correspondente ao item, o qual é escrito no conjunto de leitura com sua versão, por um servidor aleatório.

Procedemos à escrita de um valor diferente em um dos clientes. Nesse mesmo cliente, realizamos a operação de commit, a qual é direcionada para cada um dos servidores (SS + SKVS). O servidor SS processa a transação e envia uma mensagem a cada um dos SKVS com o número de sequência da transação. Os SKVS, ao receberem o número de sequência, comparam as versões do conjunto de leitura do cliente e do banco de dados. Após confirmarem que a transação é válida, realizam-na no banco de dados e retornam uma confirmação ao cliente.

Enquanto os servidores processam a transação do cliente 1, o cliente 2 realiza uma escrita no item lido anteriormente e então um commit. Contudo, por ter encaminhado a execução posteriormente ao SS, sua tentativa resulta em um abort, já que o cliente terá uma versão

desatualizada do item em seu conjunto de escrita. Nesse caso, os SKVS verificam que o item “teste” está desatualizado em relação à versão recém atualizada pelo cliente 1, retornando uma mensagem de abort ao usuário.

4. Trabalhos futuros

Durante o desenvolvimento do trabalho notamos algumas melhorias a serem endereçadas em trabalhos futuros:

1. Utilização de um protocolo de consenso distribuído para garantia de ordem, ao invés de utilizar um servidor sequenciador;
2. Implementação de replicação do servidor descobridor, como proposta de aumento de tolerância a falhas;
3. Incremento do SKVS para absorver as funções do servidor descobridor, atuando como um peer que realiza a conexão e desconexão de outros SKVS e realiza o envio de uma mensagem de flooding para descobrir quais servidores estão na rede;
4. Desenvolvimento de um mecanismo de checagem dos SKVS por parte do servidor descobridor, potencialmente enviando de tempos em tempos uma requisição para verificar se não houve a falha de um SKVS;
5. Conexão dos SKVS a um Key-Value Store difundido no mercado, como o Redis ou DynamoDB.