Sistemas Distribuídos
Projeto de Programação – KV Store

Revisão 1: 17/09/2022.

1. Definição do Sistema

Crie um sistema distribuído com três servidores que permita armazenar pares chave-valor (também denominado Sistema KV ou *Key-Value Store*) de forma replicada e consistente, utilizando TCP como protocolo da camada de transporte.

O sistema funcionará de forma similar (porém muito simplificada) ao sistema **Zookepeer**, um sistema distribuído que permite a coordenação de servidores.

2. Recomendação Inicial

Se nunca programou com TCP ou UDP, recomendo assistir o vídeo do link https://www.youtube.com/watch?v=nysfXweTl7o e implementar os exemplos mostrados. Só assistir o vídeo não lhe será de utilidade quando tiver que implementar funcionalidades mais complexas.

3. Visão geral do sistema

O sistema será composto por 3 servidores (com IPs e portas conhecidas) e muitos clientes. Os clientes poderão realizar requisições em qualquer servidor, tanto para inserir informações key-value no sistema (i.e., PUT) quanto para obtê-las (i.e., GET). Os servidores, por sua vez, deverão atender as requisições dos clientes. Dentre os três servidores, escolhe-se inicialmente um deles como líder, o qual será o único que poderá realizar um PUT. Por outro lado, qualquer servidor poderá responder o GET.

4. Funcionalidades do Cliente (considere o ponto de vista de um ClienteX)

a) Inicialização: o cliente deve capturar do teclado os IP e portas dos três servidores. As portas default dos servidores (como mencionado na Seção 5) são 10097, 10098 e 10099. Cabe destacar que o cliente não conhece (e não deve conhecer) quem é o líder.

b) Envio do PUT: o cliente deve capturar do teclado a key e value a ser inserida. A seguir, envia uma requisição PUT a um servidor escolhido de forma aleatória. Deve receber a mensagem PUT_OK com um timestamp[1]. Para detalhes do timestamp, ver Seção 5.
   - Na requisição do PUT, envie a key e a value.

[1] Timestamp denota um valor em que determinado evento ocorreu. Pode ser uma data/hora, um contador, etc.

c) Envio do GET: o cliente deve capturar do teclado a key a ser procurada. A seguir, envia uma requisição GET a um servidor escolhido de forma aleatória. Deve receber uma resposta do servidor escolhido. Para detalhes da resposta, ver Seção 5.
    - Na requisição do GET, envie a key e o último timestamp que o cliente tem associado a essa key. Note que o timestamp não deve ser capturado do teclado.

Observações:

- Cada cliente possui seu(s) próprio(s) timestamp(s), inicializado(s) em zero.
- Toda comunicação entre cliente servidor será por TCP e deverá obrigatoriamente transferir a classe Mensagem criada por você.
- Considere que o cliente não vai sair do sistema nem morrer.

5. Funcionalidades do Servidor (considere o ponto de vista de um Servidor S)

a) Inicialização: o servidor deve capturar do teclado o IP e a porta dele e o IP e a porta do líder. O endereço IP a ser inserido será o 127.0.0.1 se estiver realizando o projeto na mesma máquina. As portas default (que permitirão aos clientes conectar-se com algum servidor) serão as 10097, 10098 e 10099.

b) Recebe e responde simultaneamente (obrigatório com threads) requisições dos clientes. Por 'simultaneamente' entenda-se que o servidor deverá poder realizar outras funcionalidades enquanto está fazendo requisições ou esperando por elas.

c) Recebe Requisição PUT. Caso o servidor não seja o líder, deverá encaminhar a requisição para o líder. Caso o servidor seja o líder:

1. Insere a informação em uma tabela de hash local, associando um timestamp para essa key.
    - Se a key já existir, atualize tanto o value quanto o timestamp associado.
2. Replique a informação (key, value, timestamp) nos outros servidores, enviando-a na mensagem REPLICATION.
3. Envie para o cliente a mensagem PUT_OK junto com o timestamp associado à key (ver regra de envio no item: Requisição REPLICATION_OK).

d) Recebe Requisição REPLICATION. Insira na sua tabela de hash local as informações e responda para o líder a mensagem REPLICATION_OK.

e) Recebe Requisição REPLICATION_OK. Assim que o líder receber essa mensagem de **todos** os servidores, envie para o cliente a mensagem PUT_OK junto com o timestamp associado à key.

f) Recebe Requisição GET. Considere o cliente Cx com seu timestamp Tx, requisitando pela key Kx ao servidor S. Caso a chave não exista, o value devolvido será NULL. Caso exista, o value a ser devolvido para Cx será aquele cujo timestamp associado a Kx (no Servidor S) seja igual ou maior ao Tx. Se o timestamp da key em S for menor que Tx, devolva para Cx a mensagem TRY_OTHER_SERVER_OR_LATER.

- Em outras palavras, o cliente **NUNCA** deverá obter um value anterior ao que já viu.

Além do value, devolva para o cliente o timestamp (do servidor S) associado a Kx

Observações:

- Toda comunicação entre servidores será por TCP e deverá obrigatoriamente transferir uma classe Mensagem criada por você.
- Considere que os servidores não vão sair do sistema, não vão morrer e não haverá troca de líder.
- Considere que o líder somente executará um PUT por vez, ou seja, não há requisições PUT concorrentes.

6. Mensagens (prints) apresentadas na console

Na console de cada cliente deverão ser apresentadas “exatamente” (nem mais nem menos) as seguintes informações

- Menu interativo (por console) que permita realizar a escolha somente das funções INIT, PUT e GET.
    - No caso do INIT, realize a inicialização do cliente.
    - Envio da requisição PUT, capturando do teclado as informações necessárias.
    - Envio da requisição GET, capturando do teclado as informações necessárias.
- Quando receber o PUT_OK, print “PUT_OK key: [key] value [value] timestamp [timestamp] realizada no servidor [IP:porta]”. Substitua a informação entre os parênteses com as reais.
- Quando receber a resposta do GET, print “GET key: [key] value: [valor devolvido pelo servidor] obtido do servidor [IP:porta], meu timestamp [timestamp_do_cliente] e do servidor [timestamp_do_servidor]”

Na console de cada servidor deverão ser apresentadas “exatamente” (nem mais nem menos) as seguintes informações

- Quando receber o PUT:
    - Se for o líder, print “Cliente [IP]:[porta] PUT key:[key] value:[value].
    - Se não for o líder, print “Encaminhando PUT key:[key] value:[value]”.
- Quando receber o REPLICATION, print “REPLICATION key:[key] value:[value] ts:[timestamp].
- Quando o líder receber o REPLICATION_OK de todos, print “Enviando PUT_OK ao Cliente [IP]:[porta] da key:[key] ts:[timestamp_do_servidor].
- Quando receber o GET, print “Cliente [IP]:[porta] GET key:[key] ts:[timestamp]. Meu ts é [timestamp_da_key], portanto devolvendo [valor ou erro]”.

7. Teste realizado pelo professor

O professor compilará o código usando o javac da JDK 1.8.

Após a compilação, o professor abrirá 5 consoles (no Windows, seria o CMD.EXE, também conhecido como prompt). Três serão servidores e os outros dois serão clientes. A partir das consoles, o professor realizará os testes do funcionamento do sistema. Exemplo das consoles pode ser observado no link: https://www.youtube.com/watch?v=FOwKxw9VYqI

Cabe destacar que:

- Seu código não deve estar limitado a 2 clientes, suportando mais do que 2.
- Inicialmente, o professor levantará (executará) os servidores. A seguir, levantará os clientes.
- Você não precisará de 5 computadores para realizar a atividade. Basta abrir as 5 consoles.

8. Código fonte

- Deverá criar somente as classes Servidor, Cliente e Mensagem. A última deverá ser utilizada **obrigatoriamente** para o envio e recebimento de informações (nas requisições e respostas).
    - Caso não use, envie e receba a classe Mensagem, serão descontados 3 pontos da nota final.
    - Caso envie novas classes, serão descontados 3 pontos da nota final.
    - A única exceção é a criação das classes para Threads, mas elas deverão ser criadas dentro da classe Cliente ou Servidor (e.g, classes aninhadas).

- O código fonte deverá apresentar claramente (usando comentários) os trechos de código que realizam as funcionalidades mencionadas nas Seções 4 e 5.

- O código fonte deverá ser compilado e executado por uma JDK 1.8 (também denominada JDK 8). Você poderá utilizar todas as classes e pacotes existentes nela. O link destas é https://docs.oracle.com/javase/8/docs/api/

- O uso de bibliotecas que realizem parte das funcionalidades pedidas não será aceito. Caso tenha dúvidas de alguma específica, pergunte ao professor.

9. Entrega Final

A entrega é individual e consistirá em: (a) um relatório [SeuRA].pdf; (b) o código fonte do programa com as pastas e os .java respectivos (c) o link para um vídeo, dentro do relatório, que mostre o funcionamento. A entrega será realizada como definida no plano de ensino, não sendo aceita por outras formas.

- Caso queira utilizar outra linguagem de programação, deve enviar um email ao professor até a terceira semana de aula e esperar a aceitação do professor. Após essa data, será obrigatório o envio em Java.
- Caso tenha utilizado uma biblioteca permitida pelo professor, envie-a também.

O relatório deverá ter as seguintes seções:

I. Nome e RA do participante

II. Link do vídeo do funcionamento (*screencast*). O vídeo deverá conter no máximo 5 minutos, mostrando a compilação e o funcionamento do código nas cinco consoles. **Não envie o vídeo**, envie só o link do vídeo, o qual pode disponibilizar no Youtube ou em outro lugar semelhante (como vimeo). Lembre-se de dar permissão para visualizá-lo.

III. No vídeo do ponto anterior, realize e mostre um exemplo de quando o GET acontece normalmente e quando acontece o TRY_ANOTHER_SERVER_OR_LATER

IV. Para cada funcionalidade do servidor, uma breve explicação em “alto nível” de como foi realizado o tratamento da requisição. Na explicação DEVE mencionar as linhas do código fonte que fazem referência.

V. Para cada funcionalidade do cliente, uma breve explicação em “alto nível” de como foi realizado o tratamento da requisição. Na explicação DEVE mencionar as linhas do código fonte que fazem referência.

VI. Explicação do uso das threads. Separe a explicação de servidor e do cliente, mencionado as linhas do código fonte que fazem referência.

VII. Links dos lugares de onde baseou seu código (caso aplicável). Prefiro que insira os lugares a encontrar na Internet algo similar.

10. Observações importantes sobre a avaliação

A seguir mencionam-se alguns assuntos que descontarão a nota.

- Código fonte não compila ou usa bibliotecas externas à JDK sem aprovação do professor (nota zero).
- Não transfere a classe Mensagem no envio/recebimento (menos 3 pontos)
- Não usou Threads (menos 2 pontos).
- Não enviou o relatório (menos 2 pontos).
- Não enviou o link do vídeo, o link não está disponível ou o vídeo não mostra o funcionamento pedido (menos 2 pontos)

- Enviou outros arquivos do código fonte além dos citados na Seção 8, por exemplo os .class ou outras classes (menos 2 pontos).
- Código fonte sem comentários que referenciem as funcionalidade das seções 4 e 5 (menos 2 pontos)

11. Links recomendados

- Para baixar a JDK 8
  https://www.oracle.com/br/java/technologies/javase/javase-jdk8-downloads.html

- Vídeo explicativo sobre programação UDP, TCP e Threads:
  https://www.youtube.com/watch?v=nysfXweTI7o

- Informações sobre programação com UDP podem ser encontradas em:
  https://www.baeldung.com/udp-in-java
  https://www.geeksforgeeks.org/working-udp-datagramsockets-java/
  https://docs.oracle.com/javase/tutorial/networking/datagrams/index.html

- Informações sobre programação com TCP podem ser encontradas em:
  https://www.baeldung.com/a-guide-to-java-sockets

- Informações sobre Threads, que permitem que o servidor ou peer receba e envie informações de forma simultânea:
  https://www.baeldung.com/a-guide-to-java-sockets (Seção 6: TCP com muitos clientes)
  https://www.tutorialspoint.com/java/java_multithreading.htm

12. Ética

Cola, fraude, ou plágio implicará na nota zero a todos os envolvidos em todas as avaliações e exercícios programáticos da disciplina.

Perguntas Frequentes (FAQ)

1. Como faço para enviar o objeto Mensagem em vez do String?

Existem algumas formas. Você pode serializar o objeto (não recomendo) ou usar o formato JSON (recomendo). JSON permite transformar um objeto Java a uma String e vice-versa. Caso queira usar o JSON, deve ser através da biblioteca https://github.com/google/gson. Um link interessante: http://tutorials.jenkov.com/java-json/gson.html (só até "Generating JSON From Java Objects").

2. Tenho a estrutura no servidor e criei as Threads que atendem os Peers. Estou tendo problemas de concorrência ao atualizar a estrutura. Como resolvo isso?

Existem diversas formas, como o uso de *locks*, semáforos, *synchronized*, etc. A mais simples é usar uma estrutura que já trate/implemente a concorrência. Para isso, o Java 1.8 possui o pacote *concurrent*. Por exemplo, pode usar o ConcurrentHashMap ou o ArrayBlockingQueue. Um link interessante: https://www.baeldung.com/java-concurrent-map