Confiabilidade de Sistemas Distribuídos

Trabalho Prático nº 1, 2017/2018

D-SKVS - Dependable Searchable Key Value Storage

Grupo: G01

Nº 42009– João Almeida
Nº 41803 – Jorge Valadas
Nº 41959– Mário Carvalho

Resumo

O trabalho realizado implementa um serviço de armazenamento Key-Dataset suportado numa arquitetura Servidor-Cliente com suporte sobre instâncias de Redis monitorizadas através da utilização da biblioteca open source BFT-Smart. As ferramentas utilizadas permitem que operações sobre as tabelas de dados sejam iteradas e editadas, mantendo um consenso no sistema através da utilização de SMR (State Machine Replication) com tolerância a falhas bizantinas.

1. Introdução e contexto do trabalho

O objetivo com o qual a aplicação foi desenvolvida é o de disponibilizar uma interface de armazenamento confiável tendo por base uma implementação do tipo Key-Value-Store, e oferecer aos clientes a possibilidade de efectuar operações de *Read/Write* que permitem a adição e inspecção de tuplos do tipo <key value - data set> sobre a estrutura da implementação. Para garantir a consistência do sistema a aplicação gere de modo organizado várias réplicas destas tabelas distribuídas entre as várias réplicas em execução de modo a oferecer uma vista consistente sobre os dados.
A aplicação usa protocolos de *State Machine Replication* de modo a garantir a disponibilidade e integridade do sistema perante os clientes. Este protocolo é implementado através da utilização da biblioteca BFT-SMaRt que confere para além de mecanismos de recuperação de transferência de estados, tolerância a falhas bizantinas em sistemas deterministas.
A comunicação entre cliente e servidor é suportado numa arquitetura de pedidos REST sobre um canal *Transport Layer Security*, implementado através do uso de *Java Key Stores*.
No final de configuração de todos os componentes utilizados no desenvolvimento da aplicação, concebemos um sistema sobre uma interface *Client-Server* que permite a um cliente fazer pedidos *REST* a um servidor que faz a gestão de vários requests por parte de diferentes clientes, e onde o consenso sobre os diferentes pedidos é atingido através da

library BFT-SMaRt, que a cada réplica da aplicação designa uma instância do algoritmo, permitindo desse modo atingir o consenso visível no sistema.

2. Modelo de sistema e arquitetura de referência

O sistema encontra-se dividido em três componentes principais: cliente, servidor e réplica. O cliente irá efetuar pedidos REST ao servidor de forma a poder executar as operações definidas. O servidor possui uma interface REST para poder responder a estes pedidos. Os pedidos entre cliente e servidor encontram-se protegidos ao utilizar uma ligação TLS. Existe autenticação do lado do servidor, para isso foram geradas as respetivas keystores e o certificado do servidor. Ao receber um pedido, este servidor, que atua como cliente da ferramenta Bft-SMaRt irá invocar operações nas réplicas que aplicam este protocolo de State Machine Replication. Existem dois tipos de operações nas réplicas: ordered e unordered. As operações ordered representam as quais cuja ordem interessa tais como PutSet ou AddElement e as operações unordered são as cuja ordem não interessa como GetSet ou ReadElement. Na réplica, como resposta a estas invocações irão ser executadas operações na estrutura onde os dados se encontram alojados. Neste caso, os dados encontram-se num servidor Redis. Assim, a réplica irá comunicar com este servidor através de um cliente Redis, o Jedis. Este servidor Redis armazena os dados num Mapa de Mapas, isto é para cada key, que irá representar a entrada, existe um mapa de atributos sob a forma de <Key, Value>. Desta forma, facilita a indexação das entradas. De forma a que as réplicas se tornem tolerantes a falhas bizantinas foi utilizada uma ferramenta, o Bft-SMaRt. Esta ferramenta permite criar uma abstração sobre grande parte das mecânicas que tornam as réplicas tolerantes a falhas bizantinas à excepção de dois métodos: getSnapshot e installSnapshot. Estes dois métodos permitem, respetivamente, periodicamente criar uma imagem a partir do estado atual de uma réplica, ou seja, dos dados que se encontram a uma dada altura armazenados no servidor Redis e instalar essa mesma imagem numa réplica que se encontre num destes 2 casos: 1) demorem demasiado tempo a responder e assim irão saltar para um estado mais atualizado ou 2) réplicas que tenham falhado ou crashado e assim irão ser atualizadas.

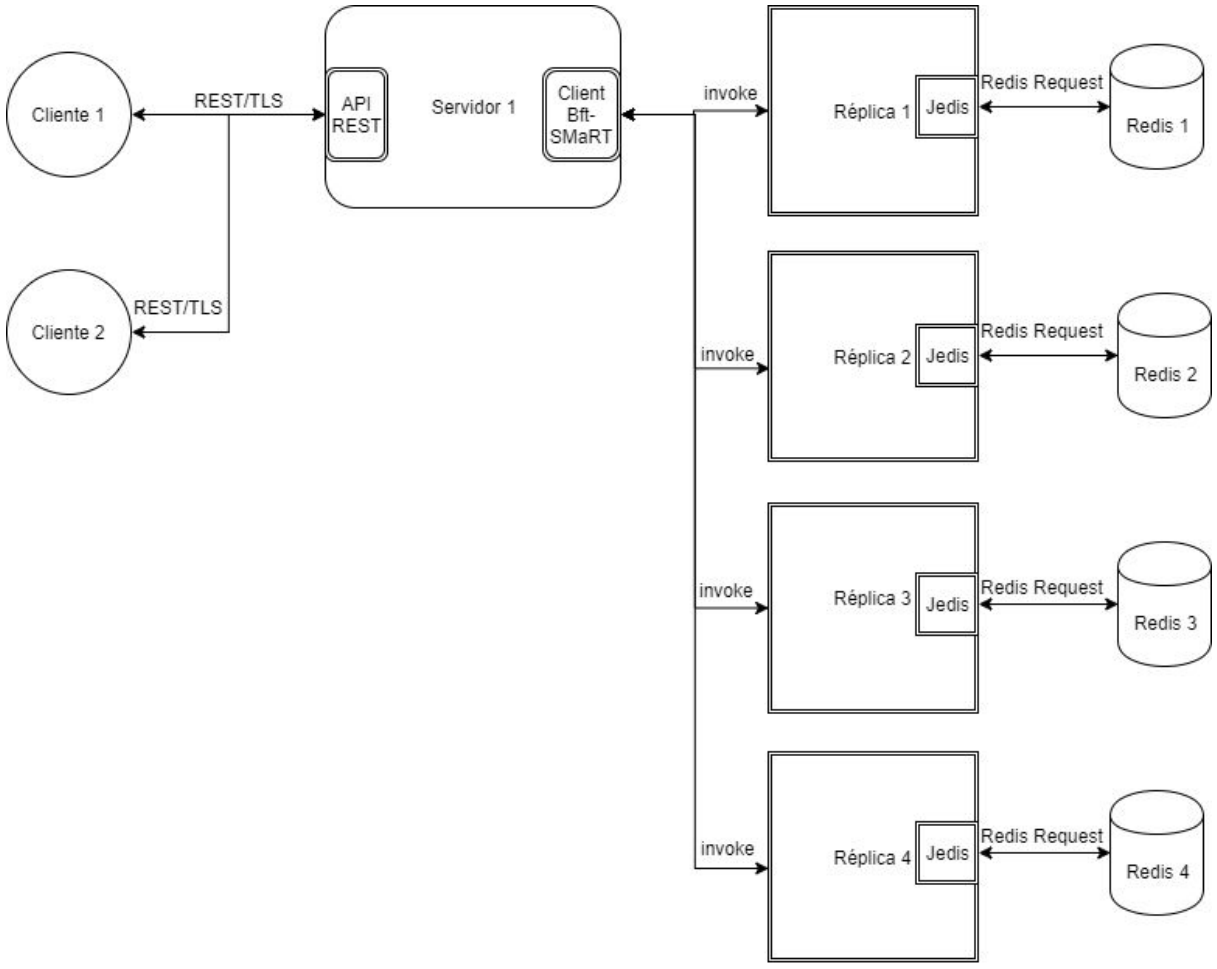

Fig. 1 - Esquema da arquitetura com 4 réplicas e 2 clientes.

3. Modelo de confiabilidade

Como já foi referido, o BFT-SMaRt é a principal ferramenta que suporta o modelo de confiabilidade do sistema. Por defeito a library suporta falhas bizantinas não maliciosas, no entanto o nosso sistema foi configurado segundo as especificações indicadas no documento de apoio do BFT [1]. Nomeadamente no ficheiro no system.config

3.1 Modelo de falhas

O BFT é um sistema tolerante a falhas por crash, até f < n/2 (minoria simples), no entanto o número de falhas suportado muda se configurarmos o sistema para lidar com falhas bizantinas.

3.2 Modelo de adversário

Desse modo é possível configurar o protocolo BFT-SMaRt de modo a suportar falhas bizantinas, definindo a variável system.bft a true, no ficheiro de system config, fazendo

com que os clientes tenham que se autenticar através de assinaturas nas suas requisições tornando assim o sistema mais resistente a falhas maliciosas

4. Arquitetura do sistema e seus componentes

De forma a tornar mais claro o funcionamento do sistema, optou-se por explicar o processamento de um pedido de PutSet. Inicialmente, são iniciados os servidores Redis. São inicializados tantos quanto o número de réplicas, pois os servidores Redis servirão para as réplicas armazenarem os seus dados. O cliente irá enviar um pedido de POST para o servidor contendo a chave e os atributos da entrada que pretende adicionar. O servidor, quando inicializado, também inicializa um objeto do tipo ServiceProxy de forma a estabelecer o contato com as réplicas. Os endereços das réplicas encontram-se definidos no ficheiro de configurações do Bft-SMaRt. O servidor, através da sua interface REST, irá receber os dados enviados e criar um OutputStream e escrevê-los. Os dados são todos transferidos em byte arrays para que haja compatibilidade com o Bft-SMaRt. De seguida, é invocada a operação de invokeOrdered na réplica. Esta operação, como definidas pelos autores, representa uma operação onde serão feitas escritas no sistema. A réplica, ao receber esta operação e os respetivos dados, efetua as verificações necessárias. Neste caso, é verificado se já foi adicionada alguma entrada com esta chave, caso não exista, esta é adicionada. Para armazenar os dados, cada réplica possui um servidor Redis. De forma a poder comunicar com este servidor é utilizado o cliente Jedis e ao inicializar cada réplica é testada esta conexão através de um ping. Os dados encontram-se armazenados através de Map<String, Map<String, String>> desta forma, a chave do mapa é o identificador de cada entrada, passada por argumento na operação de PutSet. Cada entrada terá outro mapa correspondente aos seus argumentos, sendo que para cada existe um par chave-valor. Os atributos existentes são controlados através de uma lista de fields, desta forma, torna-se mais fácil adicionar e remover campos, como também efetuar as verificações necessárias. Os dois métodos principais do Bft-SMaRt são o getSnapshot e installSnapshot, onde são tirados um snapshot ao estado atual da réplica e são instalados numa réplica. Para obter o snapshot de uma réplica são iteradas todas as chaves do mapa atual do armazenamento e são adicionadas todas as entradas num mapa local. De seguida, é criado um OutputStream de forma a poder retornar este estado sob a forma de um byte array e este é retornado. Tal como referido na Secção 2 existem duas situações em que é necessário instalar uma cópia numa réplica: 1) demorem demasiado tempo a responder e assim irão saltar para um estado mais atualizado ou 2) réplicas que tenham falhado ou crashado e assim irão ser atualizadas. Nestes casos, serão apagadas as entradas que possam existir no servidor Redis da réplica e serão percorridas todas as entradas do estado passado de forma a poder adicioná-las ao armazenamento.

5. Aspectos de implementação

O Docker foi escolhido como ambiente de execução de forma a permitir a simulação de várias máquinas.

A principal limitação do sistema é quando não existe uma maioria de nós a funcionar corretamente, o sistema não consegue recuperar deste estado.

6. Validação e resultados experimentais

6.1 Prova de correção da implementação

A implementação da linha de comandos dos clientes permitiu que, ainda durante o desenvolvimento, os diferentes métodos pudessem ser testados.

clientes/benchmarks	1	2	3	4	5
3	11915	7903	16446	28025	15982
6	23145	7141	29715	39918	37829
9	16668	8069	38753	49078	43970

Tabela 1 - Tempos de execução(ms) dos Benchmarks 1 a 5 sem falhas.

Os resultados obtidos são o resultado experimental de testes efetuados aos diferentes Benchmarks de referência utilizando 7 réplicas e 3, 6 e 9 clientes simultaneamente efetuando os pedidos. Os tempos encontram-se todos em milisegundos.

Na tabela 1 é possível observar os tempos de execução médio para cada Benchmark em testes onde não ocorrem falhas. Os tempos médios foram obtidos após 3 execuções de cada teste retirando o valor médio de todos os clientes de cada execução.

clientes/benchmark	1	2	3	4
3	18206	7060	17540	38839
6	40655	12781	13568	47963
9	76243	15045	27791	61313

Tabela 2 - Tempos de execução(ms) dos Benchmarks 1 a 5 com falhas por crash.

Na tabela 2 é possível observar os resultados dos testes efetuados para cada Benchmark em testes onde ocorrem falhas por crash. A simulação foi efetuada, tal como na primeira situação, no entanto, é simulado o crash de uma réplica, terminando a sua execução.

clientes/benchmark	1
3	15977
6	20663
9	23136

Tabela 3 - Tempos de execução(ms) dos Benchmarks 1 a 5 com falhas bizantinas.

Na tabela 3 é possível observar os resultados dos testes efetuados para cada Benchmark em testes onde ocorrem falhas bizantinas. Infelizmente, não foi possível terminar os testes, pois existiu uma excepção lançada que apenas foi detectada durante a fase de testes. E, desta forma, não foi possível terminar os testes. No entanto, estes serão efetuados a prazo da próxima entrega.

Referências

[1] A. Bessani, J. Sousa, E. Alchieri, State Machine Replication for the Masses with BFT-SMART , in Proceedings of DSN 2014 , 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Atlanta, USA, June 2014