Confiabilidade de Sistemas Distribuídos

DDS Dependable Storage System

v.1.0

Neste trabalho iremos desenvolver um serviço de armazenamento confiável, tolerante a falhas bizantinas e capaz de resistir a intrusões que visem comprometer a autenticidade, integridade e confidencialidade dos dados armazenados ou que visem atacar o serviço com injeção de execuções incorretas para comprometer a disponibilidade e a fiabilidade do sistema.

1. Introdução

O serviço implementará um sistema distribuído de armazenamento do tipo big-table em que as entradas correspondem a tuplos <key value-sets>. Para garantir as propriedades desejadas o utilizará replicação consistente destas tabelas, com o armazenamento distribuído por diversas replicas organizadas e interatuando de duas possíveis formas: com base num sistema de quórums bizantinos ou com base num sistema de replicação do tipo SMR (State Machine Replication). O primeiro caso deverá ser implementado com base no algoritmo ABD, sendo o segundo caso suportado numa biblioteca que implementa o algoritmo PAXOS com suporte a galhas bizantinas e que disponibiliza mecanismos base para recuperação e transferência de estado, para tolerância a intrusões.

Os clientes interagem com o sistema através de uma API bem definida para gestão do armazenamento e pesquisa de dados disponibilizada pelo sistema DDS, suportando operações REST protegidas por TLS.

2. Aspetos de implementação

2.1 API e Operações suportadas

O sistema DDS disponibilizará na sua API operações de manipulação do sistema e armazenamento (API primária) e operações de pesquisa sobre os objetos (sets) armazenados (API estendida).

API primária (a implementa no TP1)

- key PutSet (String key, Entry set)
    - set could be NULL
- Entry = GetSet (String key)
- Status AddElement (String key) // variable part
- status RemoveSet (String key)
- status WriteElem (String key, type element, int Pos)
- Elem ReadElem (String Key, int Pos)
- boolean isELement (String key, String element)

API estendida (apenas definida no TP1, sem implementação ou com implementação "dummy")

- Int Sum (int Pos, String key, String key)
- Int SumAll (int Pos)
- Int Mult (int Pos, String Key, String Key)
- Int MultAll (int Pos)
- <set of entries> SearchEq (int Pos, value)
- <set of entries> SearchNEq (int Pos, value)
- <set of entries> SearchEntry (value)
- <set of entries> SearchEntryOR (value, value, value)
- <set of entries> SearchEntryAND (value, value, value)
- <set of entries> OrderLS (int Pos)
- <set of entries> OrderSL (int Pos)
- <set of entries> SearchEq (int Pos, value)
- <set of entries> SearchGt (int Pos, value)
- <set of entries> SearchGtEq (int Pos, value)
- <set of entries> SerachLt (int Pos, value)
- <set of entries> SearchLtEq (int Pos, value)

3. Replicação

O sistema DDS suportará uma das duas seguintes possíveis soluções a implementar:

3.1 Replicação com base em quóruns bizantinos

Neste caso o sistema organizará as réplicas num modelo de quórum bizantino, com base no algoritmo ABD (Attiya, Bar-Noy, Dolev). Este modelo deve seguir a especificação do algoritmo visto nas aulas teóricas, devendo o algoritmo ABD ser implementado de raiz de modo a assegurar as garantias e propriedades de um sistema de quóruns bizantinos.

O serviço de armazenamento DDS deve ser desenvolvido com base no seguinte modelo de referência:

- É um sistema assíncrono, sendo a interação das réplicas suportada em canais fiáveis de comunicação ponto a ponto (com base em canais TCP);
- O sistema de contemplar suporte para N=3f +1, Q=2f+1, tendo em vista poder ser avaliado experimentalmente e testado com injeção de falhas bizantinas ou por omissão (crash) com f = 1 ou 2, requerendo assim 7

réplicas com quóruns de 5 nós.

- Com base na correta implementação do algoritmo ABD, a propriedade de *Safety* deve ser garantida., enquanto a propriedade de *Liveness* será assegurada para no máximo f=2 nós maliciosos (ou incorretos);

3.2 Replicação com base num modelo de máquinas de estados

Neste caso o sistema organizará as réplicas num modelo de máquinas de estado (ou SMR – *State Machine Replication*) com base numa implementação do algoritmo PAXOS para suportar tolerância a falhas ou ataques bizantinos.

A implementação desta variante será feita com base nos seguintes requisitos:

- Será usada uma biblioteca (**BFT-smart**) que implementa o modelo SMR com implementação PAXOS para tolerância a falhas bizantinas. Este suporte disponibiliza também os mecanismos base para operações de transferência de estado para suporte de recuperação face a intrusões, com recuperação de nós que tenham sido objeto de intrusões com injeção de falhas bizantinas;
- O número de réplicas deve permitir sempre suportar até dois nós maliciosos;
- Nesta implementação que de acordo com o anteriormente referido se baseará na integração da biblioteca **BFT-smart**, os alunos deverão conduzir inicialmente um estudo e análise do funcionamento e da implementação da referida biblioteca (bem como bibliografia relacionada), disponibilizada no contexto do trabalho

Para efeitos da implementação do sistema DDS os alunos deverão começar por estender a especificação de referência do presente enunciado, de modo a completarem a mesma com as opções e aspetos específicos da sua implementação, obedecendo aos requisitos acima enunciados.

4. Demonstração e avaliação experimental

O sistema desenvolvido deverá ser verificado na sua correção e avaliado experimentalmente. Esta avaliação deverá ser feita num ambiente distribuído no laboratório, com pelo menos 3 computadores interligados em rede local, com um conjunto de 3 clientes a usar o serviço DDS e 7 réplicas (DDS), distribuídos da seguinte forma:

- Comp.1: 1 cliente, 3 réplicas
- Comp.2: 1 cliente, 3 réplicas
- Comp.3: 1 cliente, 2 réplicas

Deverão ser conduzidos os seguintes testes para obtenção das seguintes métricas quantitativas sobre a correção e desempenho da implementação:

- Prova de correção
    - Os alunos deverão conceber, implementar e estar prontos para demonstrar uma prova de correção da implementação com base em evidência experimental.

- Indicadores de *throughput§* de operações:
    - Será baseado nos seguintes cinco benchmarks:
    - Benchmark1: 100 PutSet()
    - Benchmark2: 100 GetSet()
    - Benchmark3: 50 PutSet(), 50 GetSet() , com operações alternadas
    - Benchmark4: 50 AddElement(), 50 ReadElement , com operações alternadas
    -
    - Benchmark5: Mix equilibrado de todas as operações (apenas as suportadas na API primária)

- Serão extraídos resultados que permitam ser representados num gráfico da seguinte forma:

  Gráfico sem ataques ou falhas:
    - Abcissas: benchmarks (1 a 5)
    - Ordenadas; Thoughput com media de operações / segundo

  Gráfico com ataques com crash em uma e duas réplicas:
    - Abcissas: benchmarks (1 a 5)
    - Ordenadas; Thoughput com media de operações / segundo

  Gráfico com ataques bizantinos em uma e duas réplicas:
    - Abcissas: benchmarks (1 a 5)
    - Ordenadas; Thoughput com media de operações / segundo

5. Tecnologias para implementação

O trabalho pode ser feito com base em diferentes tecnologias, de entre as seguintes, deixadas à consideração dos alunos:

- Ambiente de programação Akka, na variante dos quóruns (ABD), desenvolvido em linguagem SCALA.
- Ambiente de programação Akka, na variante dos quóruns (ABD), desenvolvido em linguagem JAVA
- Ambiente de programação Akka, na variante SMR desenvolvido em linguagem JAVA
- Ambiente JAVA (Java 7 ou Java 8)

6. Indicações para entrega do trabalho

O trabalho deve ser desenvolvido com base no sistema de gestão de código Bitbucket. Na data de entrega os alunos deverão partilhar o repositório com os docentes (hj@fct.unl.pt, nmp@fct.unl.pt). Para efeitos de entrega deverão entregar as seguintes peças:

- Relatório (de acordo com formato e *template* a disponibilizar oportunamente)
- URL com a versão congelada de entrega da implementação

7. Notas sobre a avaliação do trabalho

O trabalho será avaliado tendo em conta as seguintes pontuações indicativas para os vários elementos:

- Comunicação segura cliente/servidor [1,5 pontos]
- Comunicação segura servidor/servidor [2 pontos]
- Cobertura de todas as operações da interface (API primária) [3,5 pontos]
- Integração do protocolo de replicação [9 pontos]
- Avaliação experimental [4 pontos]

8. Referências

Materiais das aulas teóricas de CSD 16/17

BFT-Smart:

BFT-SMaRt is a replication library written in Java. It implements state machine replication. It is designed to tolerate Byzantine faults, while still being highly efficient - even if some replicas are faulty. In this page we describe how this library works, in a high level of abstraction. More details can be found in the technical report describing the system.

- **https://github.com/bft-smart/library**
- https://github.com/bft-smart/library/wiki/How-BFT-SMaRt-works
- http://bft-smart.github.io/library/
- State Machine Replication for the Masses with BFT-SMART, DSN 2014