Delft University of Technology

Distributed Systems (Group 7)

IN4391

Report Milestone 2: Byzantine-CP

Authors:
Douwe Brinkhorst, Patrik Kron, Michael Leichtfried, Miguel Lucas

2020-03-27

Evaluation ↑

> Please send me your codebase along side a short report that discusses how you are going to evaluate the implemented system. (e.g., what metrics, what is the setup, which experiments, etc.)

In this report we will describe how we are going to evaluate the BFTDC[ˆ1] Protocol system described in *"A Byzantine Fault Tolerant Distributed Commit Protocol"* by Wenbing Zhao (Department of Electrical and Computer Engineering, Cleveland State University).

Functional requirements

Functional requirements were evaluated using Scala tests (`ScalaTestWithActorTestKit`). We considered:

- Basic Committing
- Aborting
- Unilateral aborting
- Byzantine behavior tolerance

Along with the development we have built a set of tests which tested every feature we implemented. This way we ensured that every module did its work properly.
We have built a total of 15 tests through which Coordinators and Participants exange messages and perform the corresponding message verification and decision making processes. These tests ensure the implementation correctness by creating protocol instances and making coordinator replicas and participants conduct several distributed commit protocols. A different number of transactions, coordinator replicas and participants is used to test the system's resilience to multiple message passing. Further participant behaviour is tested by sending abort messages in the middle of a commit transaction.

The following tests were implemented using tests in/with Scala/Akka:

- **Test 1:** Initiate the protocol and commit with 1 coordinator replica and 1 participant.

- **Test 2:** Initiate the protocol and commit with 4 coordinator replicas and 1 participant.

- **Test 3:** Initiate the protocol and commit with 1 coordinator replica and 4 participants.

- **Test 4:** Initiate the protocol and abort with 1 coordinator replica and 1 participant.

- **Test 5:** Initiate the protocol and abort with 4 coordinator replicas and 1 participant.

- **Test 6:** Initiate the protocol and abort with 1 coordinator replica and 4 participants.

- **Test 7:** Initiate the protocol with 1 coordinator replica and 1 participant and make the participant abort the transaction.

- **Test 8:** Initiate the protocol with 1 coordinator replica and 5 participants and make one participant unilaterally abort the transaction.

- **Test 9:** Initiate the protocol with 4 coordinator replicas and 5 participants and make one participant unilaterally abort the transaction.

- **Test 10:** Initiate 2 instances of the protocol and succeed committing in both.

- **Test 11:** Initiate a commit with 1 coordinator replica and 1 participant which is followed by initiating an abort for this transaction, resulting in the in-flight commit being aborted.

- **Test 12:** Initiate the protocol and commit with 4 coordinator replicas and 1 participant, where one of the coordinator replicas is nonresponsive.

- **Test 13:** Initiate the protocol and commit with 4 coordinator replicas and 1 participant, where one of the nonprimary coordinator replicas exhibits some byzantine behaviour.

- **Test 14:** Initiate the protocol and commit with 4 coordinator replicas and 1 participant, where the primary coordinator replica exhibits some byzantine behaviour.

- **Test 15:** Initiate the protocol and force a view change by creating a participant and a slow coordinator which will exceed the timeout.

Further evaluations planned in the near future:

- expanding the simulation of byzantine behaviour. The current implementation of byzantine behaviour only covers a fraction of the byzantine faulty space. Expanding this could be interesting, but simulating more byzantine behaviours would have a large impact on code complexity. Ultimately, we believe simulating all possible byzantine behaviours is impossible. If we simulate anything less, we can only prove the system is not byzantine fault tolerant, not that it is. It might be better to show byzantine behaviour tolerance by showing that our implementation conforms to the one described in the paper.

- running the system in a distributed manner: actors on different hosts should be able to communicate with each other

Non-Functional Requirements

Further evaluations planned in the near future:

- measuring throughput and latency in continuous operation

For this, it might be interesting to implement a commit payload, as well as running the system in some distributed fashion (giving each actor its own JVM would help to show some of the distributed difficulties/overhead).

Footnotes