Phase-3 Distributed Systems

Team SAC - Abhinav Chawla, Ashutosh Das, Kumar Shivam

November 2021

1 Introduction

This report aims to convey our approach and design by leveraging the Twins-BFT Testing framework for the DiemBFT protocol as developed in Phase 2. This is a systematic approach to test the DiemBFT protocol as compared to the fault injection method which we used in Phase 2. Using the twins approach, we try to create a faulty twin of one of the nodes, create partitions and try to verify if the BFT protocol works fine or not. We use three new components to complete this task.

1. Test Generator
2. Test Executor
3. Network Playground

2 Test Generator

The test generator generates multiple scenarios for which we will test our Diem-BFT protocol upon. We mention nodes as numbers in our case for a given n and f. We mark the nodes as $0, 1, \cdots, n-2, n-1$ and the twins for $0, 1, \cdots, f-1$ will be marked as $n, n+1, \cdots, n+f-2, n+f-1$. For example, for $n = 4$ and $f = 1$, we have $0, 1, 2, 3$ as the honest nodes and 5 as the twin for node 0 It employs the 3 steps as mentioned in the paper.

First, it creates all possible partition scenarios for the given number of nodes and twins as a parameter. We create all possible non empty partitions, while also ensuring that at least one partition has a super majority quorum(2f+1). Note that the node which has a corresponding twin will be counted as a single node in the partition. Eg $\{0, 1, 4\}, \{2, 3\}$ is an invalid partition for n=4, f=1 as 0 and 4 are twins for each other.

Second, we assign leaders to all the partitions from step 1. We could assign all nodes as possible leader to all the partitions or we could do it for only the nodes which have twins(as they are the ones causing imbalance in the system). We provide this as a parameter.

Third, we permute all the possible leader-partition combos over a given number of rounds - R. We can do the permutations with replacement or without replacement according to the parameter.

As the number of possible scenarios are huge in all the three steps, we provide a mechanism to prune at the end of steps. Pruning can be done in a deterministic way by filtering first X values from the given list or in a randomized way by filtering X random values from the given list.

Partition Change Triggers - For each scenario, we mention the leader and partiton for that round. When the scenario is being executed, and it receives block for round r, the partition for round r is selected. So the network playground implicitly takes care of the partition change between the rounds. For intra partition, we propose to provide a message drop config by the user and we will pass it directly to the network playground to verify the message drops. The message drop configs can be considered as a list of configs, where each config mentions sender node, message type and range of rounds for the message drops. Another approach which we have thought is to partition at the generator level and create a different partition for each message type. So for a specific round, for a specific message type, we have a partition. By default, for all message types for a specific round, the partitions will be the same. The partition can be split into a isolated node for a message drop according to config and store it for that specific message type. We have written pseudocode for approach 1, according to the feasibility of the second approach we will accordingly accommodate in the actual code.

3 Test Executor

The test generator provides one particular scenario to the test executor, with the total nodes, total twins, leaders for r rounds, partitions for r rounds and the intra partition config. The test executor starts up the processes and assigns an id number to all of the processes. The twin process and its corresponding node, are started with the same public key and private key as well as id. It also initiates a network playground which we describe ahead. The test executor also verifies given the logs and messages, the safety and the verification properties. We check these properties on an offline basis. The online approach would require repeatedly at every commit/message/log a message being sent to the executor which would seem like an overhead. At an offline basis, we can check the logs after the whole process gets over. The properties we are checking for are -

- Safety Property 1 - If a block is certified in a round, no other block can gather f + 1 non-Byzantine votes in the same round. Hence, at most one block is certified in each round.
- Safety Property 2 - For every two globally direct-committed blocks B, B0 , either B $\leftarrow$ B0 or B0 $\leftarrow$ B.
- Safety Property 3 - Check that commits are consistent at all heights.

- Liveness Property 1(Checked in network playground) - For a given liveness bound, we are checking the system progresses by an existence of a timeout for a message of a higher round for that bound

4 Network Playground

Our design approach is to include a centralized hub which will have the role of orchestration of messages. To be precise, when a node A sends a message to node B, it will be intercepted by the playground, the playground will verify whether B is in the partition for that specific round r and then accordingly retransmits the message. Hence the nodes are unaware of the partitions, the playground will take control of it. The nodes also dont know the existence of a twin explicitly i.e. if a node B sends a message to A, but A is not in its partition but A' is, then the playground will send the message to A'. As A' and A have the same id, public key and private key, node B if receives a message from A', it will look like it received message from A.

To achieve this, we have created a flat map from the scenario partitions received initially. The map stores for each round, for each node to which all nodes, the message will be dropped, hence making it easier to verify if a message has to be dropped or not.

DiemBFT original algorithm requires minimal changes while sending messages. Rather than checking from the $from\_$ attribute when receiving a message, we will be checking from the *sender* attribute which will be sent by the playground(the $from\_$ attribute will always have playground, we want to actual sender). While sending messages, the *sender* attribute we will put the sender's id.

References

[1] S. Bano, A. Sonnino, A. Chursin, D. Perelman, and D. Malkhi. Twins: White-glove approach for BFT testing. *CoRR*, abs/2004.10617, 2020.

[2] M. Baudet, A. Ching, A. Chursin, G. Danezis, F. Garillot, Z. Li, D. Malkhi, O. Naor, D. Perelman, and A. Sonnino. State machine replication in the libra blockchain. 2019.