Implementing and Preforming Randomized Tests on the HotStuff BFT Protocol

Author: Lubomir Marinski (L.V.Marinski@student.tudelft.nl)

Responsible professor: Dr. Burcu Kulahcioglu Özkan

Supervisor: João Miguel Louro Neto

1. Background

- Byzantine Fault Tolerant (BFT) protocols allow the non-faulty participants in a distributed system to reach consensus even if some of the participants are malicious or unreliable.
- Proposals for practical BFT protocols: PBFT, Zyzzyva, BFT-SMaRt, Tendermint, hBFT and **HotStuff**
    - HotStuff can drive consensus at the pace of actual network delay and with linear communication complexity.
    - For the pipelined HotStuff variants, each view simultaneously serves as each of the 4 Basic HotStuff phases for the 4 chained nodes. (Figures 1 and 2)
- BFT protocols may have design or implementation faults so having reliable testing tools such as **ByzzFuzz** is important.
    - ByzzFuzz uses round-based small-scope structure-aware mutations to simulate process faults
    - ByzzFuzz applies round-based network partitions to simulate network faults

Figure 1: Chained HotStuff View Change - The leader of each view collects votes for the node proposed during the previous view

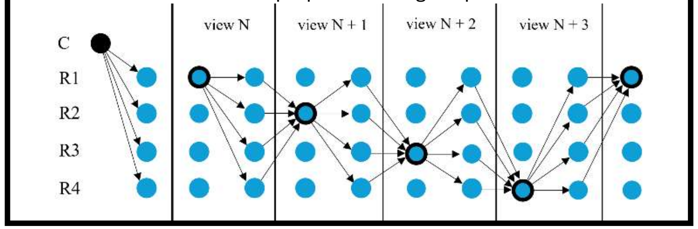
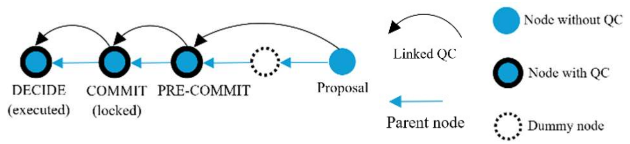


Figure 2: Chain where each node represents different protocol phase



2. Research Questions

- **RQ1** - Can ByzzFuzz find any bugs in our implementation of the HotStuff protocol?
- **RQ2** - How does the bug detection performance of ByzzFuzz compare to a baseline testing method that arbitrarily injects network and process faults?
- **RQ3** - How do small-scope and any-scope message mutations of ByzzFuzz compare in their performance of bug detection for the HotStuff protocol?

3. Methodology

- Implement the HotStuff BFT protocol in Java
- Create faulty variants of our implementation
- Test for bugs using both ByzzFuzz and 'Random' baseline testing strategy
- Test using both small-scope and any-scope mutations
- Use the obtained empirical data to answer the research questions.

Table 1: Message mutations

Message Type	**Mutations**
GENERIC	$\langle \mathbf{v'}, \langle \mathbf{h'}, p, \langle n, s\rangle, c\rangle\rangle$ $\langle v, \langle h, \mathbf{p'}, \langle n, s\rangle, c\rangle\rangle$ $\langle v, \langle h, \mathbf{p'}, \langle \mathbf{n'}, \mathbf{s'}\rangle, c\rangle\rangle$ $\langle v, \langle h, p, \langle \mathbf{n'}, \mathbf{s'}\rangle, c\rangle\rangle$
GENERIC-VOTE	$\langle \mathbf{v'}, \langle \mathbf{h'}, p, \langle n, s\rangle, c\rangle, s\rangle$ $\langle v, \langle h, \mathbf{p'}, \langle n, s\rangle, c\rangle, s\rangle$ $\langle v, \langle h, \mathbf{p'}, \langle \mathbf{n'}, \mathbf{s'}\rangle, c\rangle, s\rangle$ $\langle v, \langle h, p, \langle \mathbf{n'}, \mathbf{s'}\rangle, c\rangle, s\rangle$
NEW-VIEW	$\langle \mathbf{v'}, \langle n, s\rangle\rangle$ $\langle v, \langle \mathbf{n'}, \mathbf{s'}\rangle\rangle$

4. Implementation

Additional functionality not specified in the HotStuff paper, which affects the protocol's properties, needs to be implemented for the protocol to work in practice:

- Replica catch-up mechanism
- Client requests de-duplication
- Leader election
- Pacemaker logic
- Message validation
- Handling timeouts

5. Experimental Setup

- We use correctness invariants to detect faults
    - Agreement (safety)
    - Termination (liveness)
- Baseline implementation without intentional flaws
- 3 flawed implementations : Lower quorum, No proposal view validation, No proposal view validation
- Many experimental configurations using different parameters including number of process faults, number of network faults, max round with faults, mutation scope
- We run 1000 scenarios for each configuration

6. Results

No faults were discovered with the baseline implementation. The results for the faulty implementations when tested with ByzzFuzz are listed in Tables 2, 3 and 4.

Table 2: 'Low quorum' implementation, ByzzFuzz results

			SS		AS	
p	n	r	A	T	A	T
1	0	20	0	0	4	0
2	0	20	1	0	10	0
3	0	20	1	0	16	0
4	0	20	2	0	20	0
5	0	20	2	0	24	0
10	0	20	6	0	55	0
5	5	20	12	0	24	0
10	10	20	46	0	85	0

p	n	r	A	T
0	0	0	0	0
0	1	20	3	0
0	2	20	1	0
0	3	20	2	0
0	4	20	5	0
0	5	20	13	0
0	10	20	51	0
0	1	10	1	0
0	2	10	1	0

p	n	r	A	T
0	3	10	1	0
0	4	10	8	0
0	5	10	26	0
0	10	10	127	0
0	1	5	0	0
0	2	5	0	0
0	3	5	8	0
0	4	5	18	0
0	5	5	30	0

Table 3: 'No proposal view validation' implementation, ByzzFuzz results

			SS		AS	
p	n	r	A	T	A	T
1	0	20	0	0	0	0
2	0	20	0	0	0	0
3	0	20	0	0	0	0
5	0	20	0	0	0	0
10	0	20	0	0	0	2
15	0	20	0	0	0	2

			SS		AS	
p	n	r	A	T	A	T
20	0	30	0	0	0	5
30	0	40	0	0	0	7
10	1	20	0	0	0	2
15	1	20	0	0	0	2
20	1	20	0	0	0	0
30	1	40	0	0	0	5

p	n	r	A	T
0	0	0	0	0
0	1	20	0	0
0	10	20	0	0
0	20	30	0	0

Table 4: 'Non-monotonically increasing b_{exec}' implementation, ByzzFuzz results

			SS		AS	
p	n	r	A	T	A	T
1	0	20	3	0	83	0
2	0	20	5	0	137	0
3	0	20	12	0	172	0
4	0	20	16	0	223	0
5	0	20	23	0	245	0
1	1	20	2	0	41	0
2	1	20	7	0	98	0
3	1	20	7	0	135	0
4	1	20	17	0	173	0
5	1	20	22	0	202	0

p	n	r	A	T
0	0	0	0	0
0	1	20	0	0
0	2	20	0	0
0	3	20	1	0
0	4	20	0	0
0	5	20	1	0
0	6	20	1	0
0	7	20	0	0
0	8	20	0	0
0	9	20	0	0
0	10	20	0	0

7. Conclusions

- **RQ1** – ByzzFuzz was able to detect all introduced flaws.
- **RQ2** – For process faults there was no significant performance difference between ByzzFuzz and the 'Random' schedular. The network partitions of ByzzFuzz performed better than randomly dropped messages.
- **RQ3** – For all 3 test implementations any-scope mutations outperformed small-scope mutations. Small-scope mutations failed to detect one of the introduced bugs.