CONCURRENCY TESTING OF THE HOTSTUFF CONSENSUS ALGORITHM

AUTHOR
Wenkai Li
w.li-13@student.tudelft.nl

SUPERVISORS
Ege Berkay Gulcan
E.B.Gulcan@tudelft.nl

Burcu Kulahcioglu Ozkan
b.ozkan@tudelft.nl

AFFILIATION
EEMCS, Delft University of Technology
The Netherlands

1 INTRODUCTION

Why concurrency testing (of HotStuff) ?

- Concurrency is pervasive, but concurrency bugs can be difficult to find and reproduce.
- HotStuff [1] is a popular distributed consensus algorithm, adopted by Meta for the Diem blockchain project.

What makes concurrency testing possible?

- Controlled scheduler. It takes over the scheduling of threads and explore different interleavings [2].
- But the number of interleavings can be exponential.

What makes *effective* concurrency testing possible?

- Most concurrency bugs can be found by enforcing the order of a small number of events [3]. Only need to explore a small subset of the schedule.
- Probabilistic concurrency testing (PCT) [4] bounds the minimum number of order constraints necessary to expose the bug.
- Delay bounding (DB) [5] bounds the number of deviations from a deterministic scheduler.

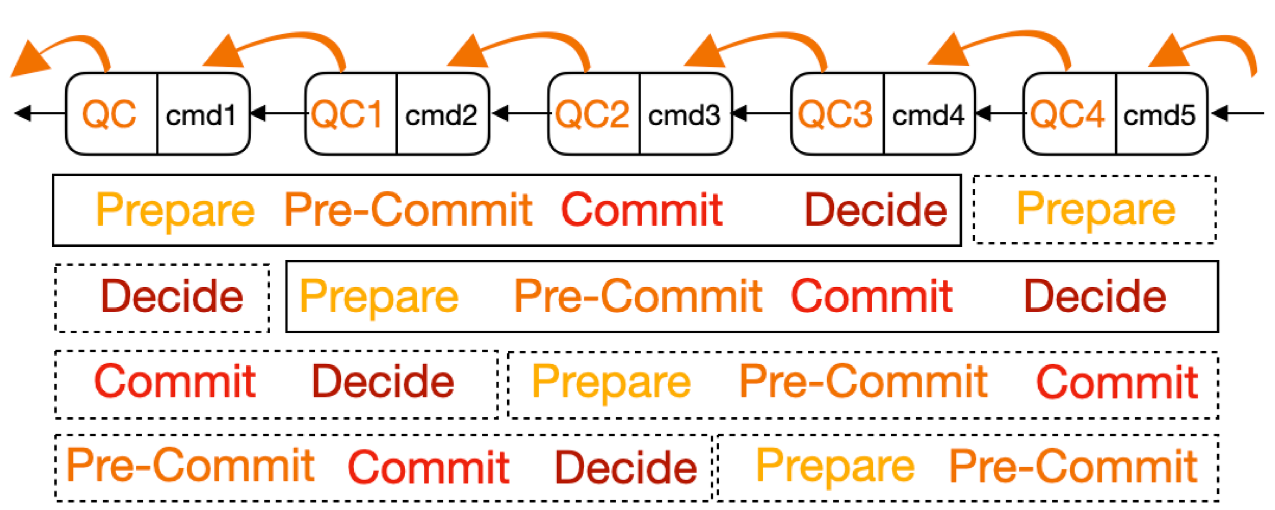

Figure 1: Execution of pipelined HotStuff algorithm

2 RESEARCH QUESTIONS

- Can PCT and delay bounding strategy find bugs more frequently in our HotStuff implementation, than the baseline random walk scheduler (RW)?
- Which bound parameter gives the best performance in PCT and delay bounding?

3 EXPERIMENTAL SETUP

Coyote framework [2] is used for the implementation of HotStuff and benchmarking of PCT and DB strategy.
Three concurrency bugs are seeded as test benchmarks:

- B1 (Safety violation). Duplication of client requests.
- B2 (Liveness Violation). Event reordering.
- B3 (Liveness Violation). Event reordering.

Two experiments to answer the research questions:

- Exp1 compares the number of buggy schedules found in 1000 explorations by RW, PCT and DB (Table 1). "F" indicates fair scheduling version used due to starvation.
- Exp2 compares the different bounding parameters for PCT (Table 2) and DB (Table3).

```
—— Thread A ——                    —— Thread B ——
                                  exec(1)
                                  ...
                                  exec(k-1)
                                  x = 1
assert(x==0)
```

Figure 2: Example bug where PCT provides a much better probabilistic guarantee (1/2) than random strategy (1/2^k)

4 RESULTS

	RW	PCT	F-PCT	DB	F-DB
B1	609	9	982	0	**<u>1000</u>**
B2	507	837	832	987	**<u>990</u>**
B3	0	9	**<u>22</u>**	0	0

Table 1: # buggy schedules found by RW, PCT and DB.

d	1	2	3	4	6	8	10
B1-F	989	988	988	991	**<u>995</u>**	990	982
B2	842	831	839	**<u>857</u>**	846	831	837
B2-F	845	848	839	**<u>850</u>**	842	832	832
B3	18	16	20	13	14	**<u>22</u>**	9
B3-F	9	**<u>26</u>**	21	12	20	15	22

Table 2: # buggy schedules found by PCT using different bounds.

d	1	2	3	4	6	8	10
B1-F	1000	1000	1000	1000	1000	1000	1000
B2	**<u>997</u>**	996	996	994	995	992	987
B2-F	**<u>998</u>**	996	993	994	994	992	990
B3	0	0	0	0	0	0	0
B3-F	0	0	0	0	0	0	0

Table 3: # buggy schedules found by DB using different bounds.

5 CONCLUSION

- PCT found buggy schedules on all three benchmarks. RW and DB failed to find any for B3.
- (Fair) PCT and DB indeed found bugs more frequently than RW on B1 and B2. (Fair) DB found bugs on B1 and B2 in almost every schedule.
- Using different values for the bound parameter did not make a significant difference on the number of buggy schedules found on our benchmarks. This could be related to the characteristics of the bugs we seeded.
- Fair implementation of PCT and DB in Coyote has comparable performance to the unfair version and can adjust performance in case of starvation.

6 LIMITATION & FUTURE WORK

- Coyote and HotStuff makes it easy to write correct code, but difficult to seed concurrency bugs.
- The performance implications of using different d parameter for PCT are not very consistent across existing work, and could be studied more extensively.
- There may not be a "One strategy for all". Performance of different strategies could depend a lot on the characteristics of bugs in different code bases.
- Deeper investigation of characteristics of different concurrency bugs could give more insight to the evaluation of different exploration strategies.

RELATED LITERATURE

[1] Maofan Yin et al. "HotStuff: BFT Consensus with Linearity and Responsiveness". In: Proceed- ings of the 2019 ACM Symposium on Principles of Distributed Computing. 2019
[2] Pantazis Deligiannis et al. "Industrial-Strength Controlled Concurrency Testing for C# Pro- grams with COYOTE". In: Tools and Algorithms for the Construction and Analysis of Systems. 2023
[3] S. Lu, S. Park, E. Seo, and Y. Zhou, "Learning from mis- takes: a comprehensive study on real world concurrency bug characteristics," in *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*, 2008.
[4] S. Burckhardt, P. Kothari, M. Musuvathi, and S. Na- garakatte, "A randomized scheduler with probabilistic guarantees of finding bugs," ACM SIGARCH Computer Architecture News, vol. 38, no. 1, pp. 167–178, 2010.
[5] M. Emmi, S. Qadeer, and Z. Rakamaric´, "Delay- bounded scheduling," in *Proceedings of the 38th an- nual ACM SIGPLAN-SIGACT symposium on principles of programming languages*, pp. 411–422, 2011.