Project Presentation:

Formal Specification and verification of the BFT protocol Prime

Huzaifah Nadeem

University of Pittsburgh

Project Description

Prime [ACKL11]

Prime: Byzantine Replication under Attack

Yair Amir, *Member, IEEE Computer Society*, Brian Coan,
Jonathan Kirsch, *Member, IEEE*, and John Lane, *Member, IEEE*

Abstract—Existing Byzantine-resilient replication protocols satisfy two standard correctness criteria, safety and liveness, even in the presence of Byzantine faults. The runtime performance of these protocols is most commonly assessed in the absence of processor faults and is usually good in that case. However, faulty processors can significantly degrade the performance of some protocols, limiting their practical utility in adversarial environments. This paper demonstrates the extent of performance degradation possible in some existing protocols that do satisfy liveness and that do perform well absent Byzantine faults. We propose a new performance-oriented correctness criterion that requires a consistent level of performance, even with Byzantine faults. We present a new Byzantine fault-tolerant replication protocol that meets the new correctness criterion and evaluate its performance in fault-free executions and when under attack.

Index Terms—Performance under attack, Byzantine fault tolerance, replicated state machines, distributed systems.

[ACKL11] Y. Amir, B. Coan, J. Kirsch and J. Lane, "Prime: Byzantine Replication under Attack," in IEEE Transactions on Dependable and Secure Computing, vol. 8, no. 4, pp. 564-577, July-Aug. 2011, doi: 10.1109/TDSC.2010.70.

Prime [ACKL11]

- Prime is BFT protocol
- Aims to address a performance-impacting attack on PBFT [CL99]
  - Not an attack in the traditional sense as the correctness and liveness properties hold
  - If you control a malicious server, you can really slow down the system by delaying messages at just the right time

[ACKL11] Y. Amir, B. Coan, J. Kirsch and J. Lane, "Prime: Byzantine Replication under Attack," in IEEE Transactions on Dependable and Secure Computing, vol. 8, no. 4, pp. 564-577, July-Aug. 2011, doi: 10.1109/TDSC.2010.70.

[CL99] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine fault tolerance. In Proceedings of the third symposium on Operating systems design and implementation (OSDI '99). USENIX Association, USA, 173–186.

Formal Spec of Prime [ACKL11]

- Prime has an implementation that seems to work just fine
- But, there is no formal specification
- The English Language description does not exist for two subprotocols.

[ACKL11] Y. Amir, B. Coan, J. Kirsch and J. Lane, "Prime: Byzantine Replication under Attack," in IEEE Transactions on Dependable and Secure Computing, vol. 8, no. 4, pp. 564-577, July-Aug. 2011, doi: 10.1109/TDSC.2010.70.

Prime Subprotocols

- 9 subprotocols:
  - Client
  - Preordering
  - Global Ordering
  - Reconciliation
  - Suspect-Leader
  - Leader Election
  - View Change
  - Proactive Recovery
  - Reconfiguration

English description and implementation code exists

Prime Subprotocols

- 9 subprotocols:
  - Client
  - Preordering
  - Global Ordering
  - Reconciliation
  - Suspect-Leader
  - Leader Election
  - View Change
  - Proactive Recovery
  - Reconfiguration

Only the implementation code exists

Goal of the project

- Write a formal spec of 4 subprotocols Prime in TLA$^+$.
  - Preordering ✅
  - Global Ordering (partial)
  - Reconciliation
  - Suspect-Leader
  - Client subprotocol (partial)
- Model-Check these using TLC
  - An attempt for PO, never got around to the rest
  - Some problems (later)
- Learn about formal specs and how to write and verify them ✅

Learning TLA+

TLA+

- Learn to use TLA$^+$.
  - First reference: Leslie Lamport's video course [TLA1]
  - Second reference: Hillel Wayne's "Learn TLA+" [TLA2]
- More or less understand most concepts
- Confident: Writing Specs (behavior, actions, etc)
- Not confident: Models, Proofs, Assertions, Refinement

[TLA1] https://lamport.azurewebsites.net/video/videos.html

[TLA2] https://old.learntla.com/introduction/

Written Specs

Preordering

EXTENDS *Integers*, *FiniteSets* need *Integers* for: *Nat* (set of Natural numbers), ... Need *FiniteSets* for: *Cardinality*(*set*)

CONSTANT

R, the set of servers (the paper uses 'R' to refer to the set of servers (server *IDs*: $R = \{1, 2, \ldots, N\}$)

OPERATIONS, the set of possible client operations

f, number of malicious servers

f_ids server ids that are malicious

VARIABLES

preorderSummary,
preorderSeqNum,
preorderCertificates,
msgs,
lastPreorderSummaries,
blacklist

Preordering

$$
\begin{aligned}
&Next \triangleq \\
&\quad \lor \exists\, s \in R : \\
&\qquad \lor RecvAndProcessClientMsg(s) \\
&\qquad \lor RecvAndProcessPOReq(s) \\
&\qquad \lor RecvPOACK(s) \\
&\qquad \lor preorderOp(s) \\
&\qquad \lor GenSummaryAndSend(s) \\
&\qquad \lor RecvAndProcessPOSUMMARY(s)
\end{aligned}
$$

Preordering

this specifies how a server sends a summary message

$$
\begin{aligned}
&GenSummaryAndSend(s) \triangleq \\
&\quad \text{LET} \\
&\qquad summary \triangleq [type \mapsto \text{“PO-SUMMARY”}, vec \mapsto preorderSummary[s], i \mapsto s] \\
&\quad \text{IN} \\
&\qquad \wedge\ mx\_more\_uptodate\_than\_my(lastPreorderSummaries[s][s], summary) \\
&\qquad \wedge\ mx\_and\_my\_are\_consistent(lastPreorderSummaries[s][s], summary) \\
&\qquad \wedge\ send(summary) \\
&\qquad \wedge\ lastPreorderSummaries' = [lastPreorderSummaries \text{ EXCEPT } ![s][s] = summary] \\
&\qquad \wedge\ \text{UNCHANGED } \langle preorderSeqNum, preorderSummary, preorderCertificates, blacklist \rangle
\end{aligned}
$$

from figure 1 of the paper:

$$
\begin{aligned}
mx\_atleast\_uptodate\_as\_my(m1, m2) &\triangleq \forall j \in R : m1.vec[j] \geq m2.vec[j] \\
mx\_more\_uptodate\_than\_my(m1, m2) &\triangleq \exists j \in R : m1.vec[j] > m2.vec[j] \\
mx\_and\_my\_are\_consistent(m1, m2) &\triangleq mx\_atleast\_uptodate\_as\_my(m1, m2) \vee \\
&\qquad mx\_more\_uptodate\_than\_my(m1, m2)
\end{aligned}
$$

Preordering

- Invariants

$$
\begin{aligned}
NoHolesInSeqNums \triangleq\ & \forall\, s \in R : \\
& \forall\, j \in R : \\
& \quad \forall\, n \in 1\,..\,preorderSummary[s][j] : \\
& \quad\quad \vee \exists\, m \in preorderCertificates[s] : \\
& \quad\quad\quad m.seq\_i \leq n
\end{aligned}
$$

$$
\begin{aligned}
cardinality\_blacklist\_under\_f \triangleq\ & \forall\, s \in R : \\
& Cardinality(blacklist[s]) \leq f
\end{aligned}
$$

Other subprotocols

- Global Ordering
  - Incomplete, only part of the behavior is there
- Client
  - Basic sketch: Client sends a request and gets responses back
  - The idea was that I will write this then connect to PO
  - Prove something about Client and PO collectively:
    - (Client + PO) => Higher-level Client + PO
  - But due to being stuck on writing models for PO, did not get far

Lessons

Lessons

- Limitations/Problems:
  - Spec writing takes time
  - Have to go back and forth on details
  - Multiple sources (paper, code)
- Specifying behavior is much simpler than specifying invariants, proofs, models, etc
- I never got around to the refinement part
  - Likely a challenge by itself

Lessons

- Main one:
  - Easy to follow written specs, writing them is hard
  - Obvious in hindsight
- Very different skill-set compared to programming
- Need to write specs and practice

Next

- I would keep working on this
- A good exercise to improve this skill
- Go over examples and recreate them
  - Especially the parts I mentioned Im not confident about

Questions?

h.nadeem@pitt.edu

University of Pittsburgh