Byzantine Ordered Consensus without Byzantine Oligarchy

Yunhao Zhang†, Srinath Setty*, Qi Chen*, Lidong Zhou* and Lorenzo Alvisi†

†Cornell University    *Microsoft Research
Order manipulation is a scourge

Expressly forbidden... but keeps happening!

Bots have reaped from unsuspecting parties over $6M in Ethereum!
Permissioned blockchains are vulnerable

- Promise trustworthy trading platforms.
- Rely on BFT State Machine Replication...
  - ...and that’s where the vulnerability lies
Oh no! BFT!

The issue is **NOT** with this.

It’s **worse**!

It affects *correctness specification* of state machine replication.
State Machine Replication

**Ingredients: a service**

1. **Implement service as a deterministic state machine**
2. **Replicate**
3. **Provide all replicas with the same input**

**Safety:** The ledgers of correct replicas hold the same sequence of commands.

**Liveness:** Commands from correct clients eventually appear in the ledgers of all correct replicas.

**+ BFT:** S&L hold even when faulty nodes are Byzantine.
The **crux**

Ingredients: a service
1. Implement service as a deterministic state machine
2. Replicate
3. Provide all replicas with the same input

When it’s about fault-tolerance, **order does not matter**

When it’s about financial transactions, **order matters!**
Following the leader?

Most BFT RSM protocols are leader-based.

Leader has **full control** over the ledger’s order.

**Bad** if leader is Byzantine.
Rotating leaders

Yet...

- Each leader still controls order of commands in its batch.
- No way to express correctness conditions on resulting total order.
Our main contributions

- **Contribution #1**: Expand the BFT SMR specification
  - To express ordering requirements rigorously and define ordered consensus
- **Contribution #2**: Chart the boundaries of Byzantine influence
  - To understand which requirements can and cannot be enforced
- **Contribution #3**: Articulate a new architecture for BFT SMR
  - To enforce ordered consensus
- **Contribution #4**: Design, implement, and evaluate Pompē
  - To demonstrate systems based on ordered consensus are practical
#1: Byzantine ordered consensus

- My preference: cmd1 < cmd2 < cmd3
  - Node #1

- My preference: cmd1 < cmd3 < cmd2
  - Node #2

- My preference: cmd3 < cmd1 < cmd2
  - Node #n

Example: ordering unanimity
If all correct nodes prefer cmd1 < cmd2, then cmd1 < cmd2 in the output ledger.
Impossibility of unanimity

Node 1: cmd1 < cmd2 < cmd3 < cmd4

Node 2: cmd2 < cmd3 < cmd4 < cmd1

Node 3: cmd3 < cmd4 < cmd1 < cmd2

Node 4: cmd4 < cmd1 < cmd2 < cmd3
The good news: We can prevent Byzantine nodes from dictating the final total order.

The bad news: We cannot fully eliminate Byzantine influence.

my preference: cmd1 < cmd2 < cmd3

cannot distinguish correct from Byzantine

but can still express useful and natural ordering guarantees

my preference: cmd3 < cmd2 < cmd1
Ordering Linearizability

- Expresses ordering preferences as timestamps.

1. Highest timestamp by all correct nodes for cmd1
2. Lowest timestamp by all correct nodes for cmd2

1. Latest linearization point for cmd1
2. Earliest linearization point for cmd2
#3: A new architecture for BFT SMR

- **Separate Ordering from Consensus**
  - **Ordering phase** decides the relative order of commands.
    - Prevents Byzantine nodes from controlling ordering.
  - **Consensus phase** periodically decides a prefix of the ledger.
    - Can preserve performance benefits of leader-based consensus.
#4: Pompē: order-linearizable SMR

two variants of Pompē

Pompē-HS: (HotStuff)

Pompē-C:

same ordering phase

different consensus phase
Building a **Byzantine-tolerant timestamp**

- Assume $3f+1$ nodes, $f$ Byzantine

**any** 2f+1 timestamps for cmd1

timestamps by all **correct** nodes for cmd1

median

**any** 2f+1 timestamps for cmd2

timestamps by all **correct** nodes for cmd2

median
Locking the median timestamp

round-trip1: collect timestamps from any 2f+1 nodes

command & its order locked in the ledger

round-trip2: write the median timestamp to any 2f+1 nodes
Consensus phase in Pompei

- Associates each consensus slot with a time interval.
- Waits until commands issued in current time interval are locked.
- Collects newly locked commands & their timestamps.
- Uses any SMR protocol to add these commands to the ledger according to their timestamps.
Safe batching in consensus phase

Order free from Byzantine leader’s control

State-of-the-art

Order subject to Byzantine leader’s control

Pompeā
leads
[10s, 10.5s)[10.5s, 11s)
... slot#i slot#i+1 ...

leads
200 commands
... slot#201 ... slot#400 slot#401 ...
... slot#600
**Batching during the ordering phase**

- A single timestamp to a batch from the same node
- For the purposes of evaluation:

  - Baseline:
    - Batch size $\beta$
    - $\beta/n$
    - $\beta/n$

  - Pompē:
    - $\beta/n$
    - $\beta/n$
    - $\beta/n$
Pompē vs HotStuff: 4 geo-distributed nodes
Conclusion

• There is a fundamental gap between the SMR correctness spec and the threat from order manipulation in blockchains.

• We introduce a new primitive, ordered consensus, to allow rigorous expression and efficient enforcement of ordering requirements.

• We design a modular architecture for ordered consensus and built Pompē which enforces ordering linearizability with performance comparable to state-of-the-art systems.
Thanks for listening! Any questions?

• There is a fundamental gap between the SMR correctness spec and the threat from order manipulation in blockchains.

• We introduce a new primitive, ordered consensus, to allow rigorous expression and efficient enforcement of ordering requirements.

• We design a modular architecture for ordered consensus and built Pompē which enforces ordering linearizability with performance comparable to state-of-the-art.

For further questions, feel free to contact Yunhao (yz2327@cornell.edu).