Proof-of-Execution: Reaching Consensus through Fault-Tolerant Speculation

EDBT ‘21
Suyash Gupta, Jelle Hellings, Sajjad Rahnama, Mohammad Sadoghi

Presenters
Saisha Shetty, Apoorva Shete, Swati Singhvi, Chris Fernandez

Fall Quarter 2023
ECS 265 | Distributed Database Systems
University of California, Davis

Introduction

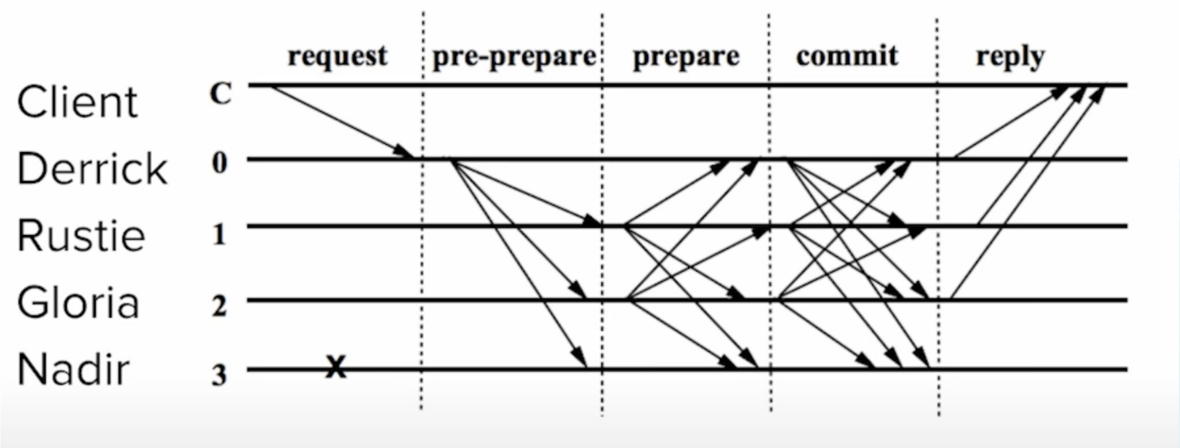
- Existing BFT (PBFT, SBFT) protocols have limitations for high-throughput applications:
    - High computational costs
    - High communication costs
    - High client latencies
    - Reliance on twin-paths and non-faulty clients

BFT

- BFT consensus objectives:
  - Order client requests within a group of replicas
  - Accommodate Byzantine faults among some replicas
  - Ensure that all non-faulty replicas unanimously agree on the order of these requests
- BFT consensus offers democratic decision-making:
  - Grants all replicas an equal vote in agreement decisions
- BFT's resilience is valuable for mitigating substantial financial losses resulting from prevalent attacks on data management systems

PBFT

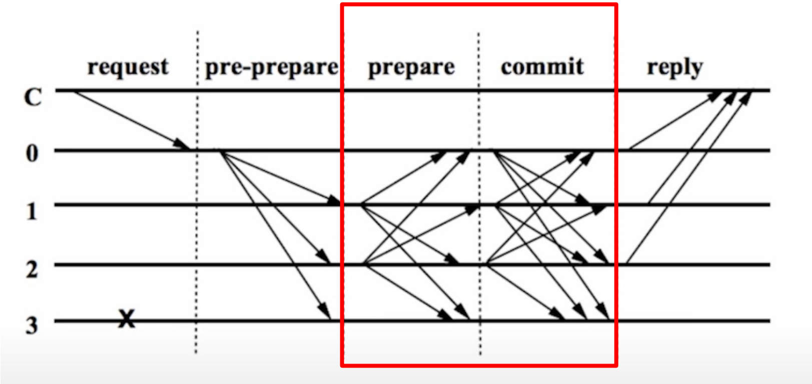
- PBFT requires at least **3f + 1** replicas to tolerate up to f faulty or malicious nodes
- PBFT operates in three communication phases: pre-prepare, prepare, and commit



Prepare vs Commit phase

Quorum - A quorum is a subset of the total number of replicas $\mathcal{R}$ that can independently reach an agreement or make a decision.

- Prepare Phase: when a replica (not necessary all replicas) observes *that a quorum* supported the client request
- Commit Phase: at least one honest replica observes that a quorum of replicas reached the prepared state; hence, the honest replica enters the committed state



Proof Of Execution (PoE)

- PoE is a novel BFT protocol achieving resilient agreement in three linear phases
- PoE builds upon PBFT and introduces four design elements:
  - Non-Divergent Speculative Execution
  - Safe Rollbacks and Robustness under Failures
  - Agnostic Signatures and Linear Communication
  - Avoidance of Response Aggregation

Notation specifications

- System S→ set $\mathcal{R}$ of replicas
- Each **Replica** r $\in \mathcal{R} \rightarrow$ unique identifier id(r) with $0 \leq id(r) < |R|$
- $\mathcal{F} \subseteq \mathcal{R} \rightarrow$ denote set of **Byzantine** replicas
- **nf** →denotes set of **non- faulty** replicas
- $n = |\mathcal{R}|$, $f = |\mathcal{F}|$, and $nf = |\mathcal{R} \setminus \mathcal{F}|$ (where \ denotes set subtraction)
- Assumptions→ **(n > 3f)** i.e (nf > 2f)
- **Byzantine** replicas: behave in arbitrary and malicious manners
- Byzantine replicas impersonate each other, but cannot impersonate non-faulty replicas

UCDAVIS

Authenticated Communication

- **Authenticated communication**
- Message Authentication Code **(MACs)**$\rightarrow$ symmetric cryptography
- Threshold signatures **(TSs)**$\rightarrow$ asymmetric cryptography
- $s\langle v\rangle_i \rightarrow$ signature share of i-th replica for signing value v
- T **=** $\{s\langle v\rangle_j \mid j \in T'\} \rightarrow$ signature shares for v from **$|T'|$=nf** replicas, aggregate T into a **single signature $\langle v\rangle$**
- Collision-resistant cryptographic hash function D(·)
- *Assumption*: Impossible to find another value v', $v \neq v'$, such that D(v) = D(v').

Requirements of PoE

- **Termination**: Each non-faulty replica executes a transaction→**Liveness**
- **Non-divergence**: All non-faulty replicas execute the same transaction→**Safety**
- **Speculative Execution**: Replicas **schedule** a transaction T for execution; they can execute and **rollback transactions**
- **Speculative non-divergence**: If **nf − f ≥ f + 1** nf replicas accept and execute the same transaction T , all nf replicas will accept and execute T

PoE Algorithm

- PoE operates in views: v= 0, 1, 2…
- View v, replica R→ id(r) = $\boldsymbol{v}$ **% n, elected as primary**
- PoE design relies on **Authenticated Communication**

Provided using

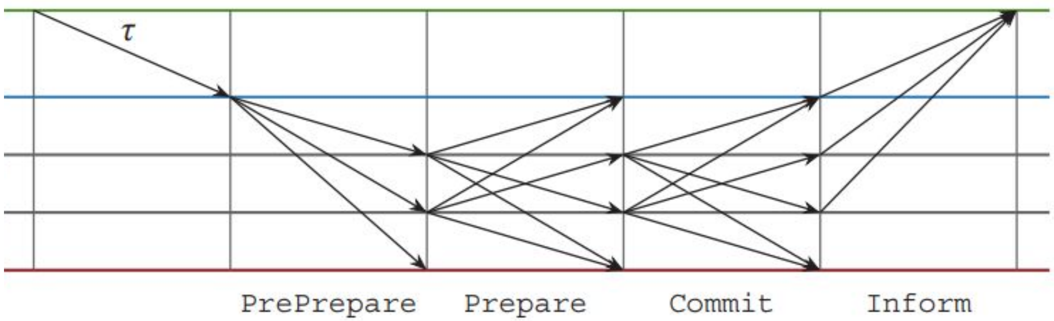
MACs

TSs: low-linear msg complexity in normal case

PBFT vs PoE (Using MACs)

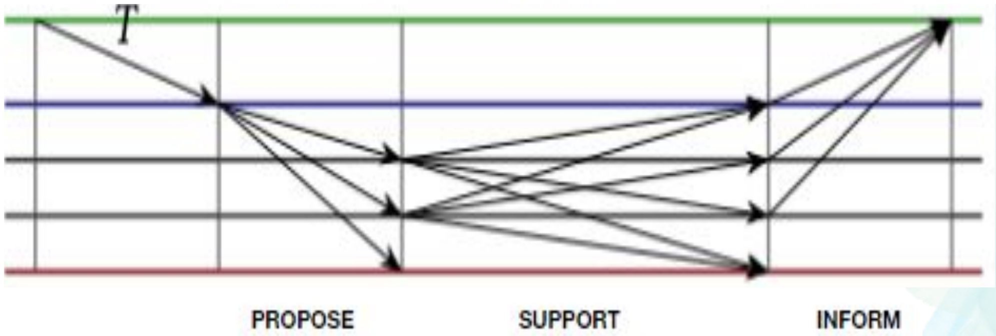
PBFT:

C
R0
R1
R2
R3



PoE:

C
R0
R1
R2
R3



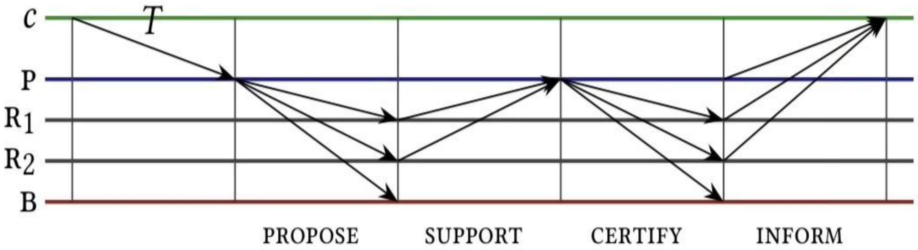
PBFT vs PoE (Using MACs)

PoE:

- Client does the work to ensure all the replicas are prepared.
- Client wait for replies from the non faulty replicas which is same as 2f+1 in PBFT.
- Prepare phase in PBFT is replaced by the support phase in PoE and client does the work of the commit phase.

The Normal-Case Algorithm of PoE (Using TSs)

O(n) O(n)

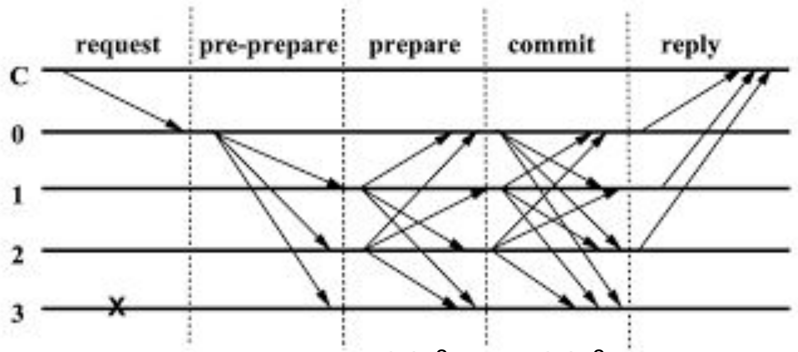


(b) PoE using TSs.

- Propose
- Support
- Certify
- Prepare phase

Support

Certify

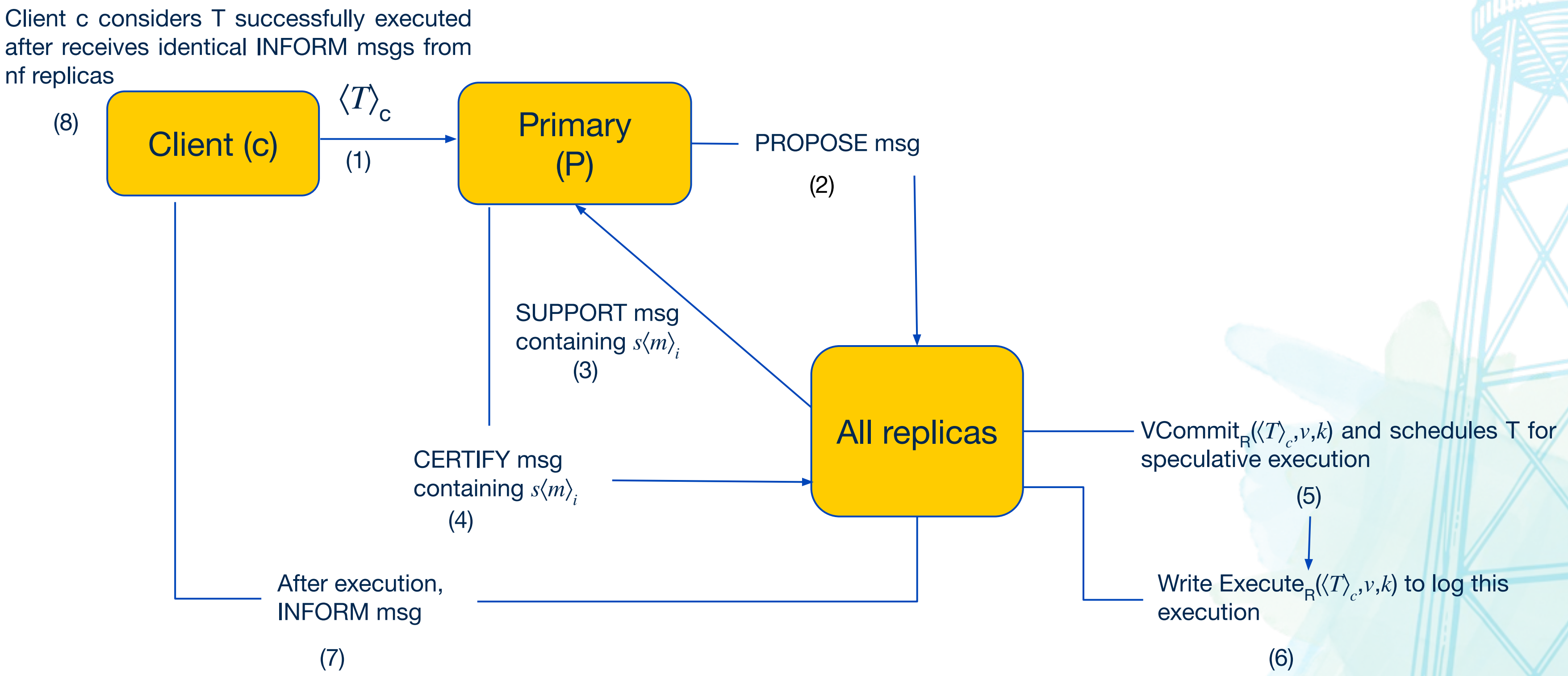


$O(n^2)$ $O(n^2)$

PBFT

- Pre-prepare
- Prepare
- Commit

Flow-Chart The Normal-Case Algorithm of PoE (Using TSs)



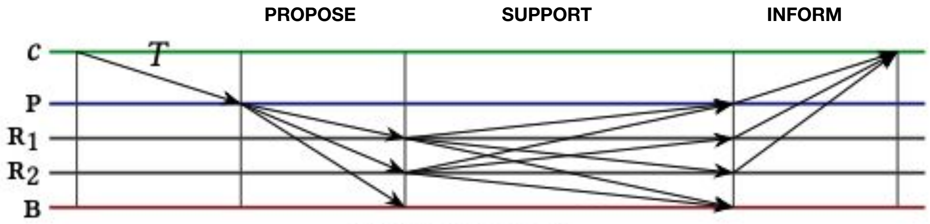
Proposition

- Statement:
  - Let R_i , $i \in \{1, 2\}$, be two non-faulty replicas that view-committed to $\langle T_i \rangle c_i$ as the k-th transaction of view v ($\mathrm{VCommit}_R(\langle T \rangle_c, v, k)$)
  - If **n > 3f**, then $\langle T_1 \rangle c_1 = \langle T_2 \rangle c_2$
  - Proof by contradiction

- Proof :
  - Key Insights:
    - Certify message contains a special signature (⟨h⟩)
    - The signature is made from non-faulty replicas.
    - Non-faulty replicas send one support message each.
    - If Replicas 1 and 2 disagree, it means their non-faulty friends don't overlap
  - Mathematical Logic:
    - X_1 , X_2 sets of non-faulty replicas for Replica 1 and Replica 2 and they don't share any replicas
    - $nf \geq |X_1 \cup X_2| \geq 2(nf - f)$
    - $nf \leq 2f$
    - $n \leq 3f$ (contradicting our initial assumption $n > 3f$)
  -

Designing PoE using MACs

- By using MACs we reduce computational complexity of PoE
- Overall communication cost increases
- Using MACs requires changes to how client requests are included in proposals (in both algorithms).
- SUPPORT and CERTIFY phases are replaced by a single all-to-all SUPPORT phase



(a) PoE using MACs

How a faulty primary can affect PoE

- Malicious primary can affect PoE in:

1. By sending proposals for different transactions to different non-faulty replicas

→Proposition 3.2 guarantees that at most a single such proposed transaction will get view-committed by any non-faulty replica.

2. By keeping some non-faulty replicas in the dark by not sending proposals to them

→The remaining non-faulty replicas can still end up view-committing the transactions as long as at least nf – f non-faulty replicas receive proposals

3. By preventing execution by not proposing a k-th transaction, even though transactions following the k-th transaction are being proposed

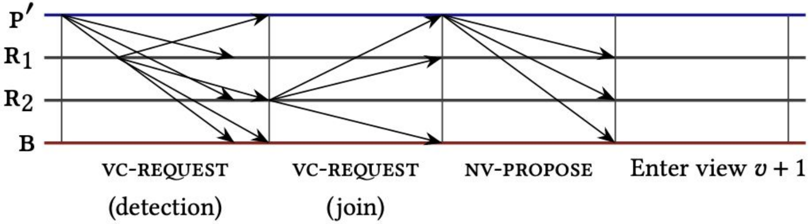
The View-Change Algorithm

- Failure of the normal-case of PoE has two possible causes:
  - primary failure
  - unreliable communication
- Goals:
  - Requests considered executed by a client are preserved
  - Replicas agree on a new view

The View-Change Algorithm

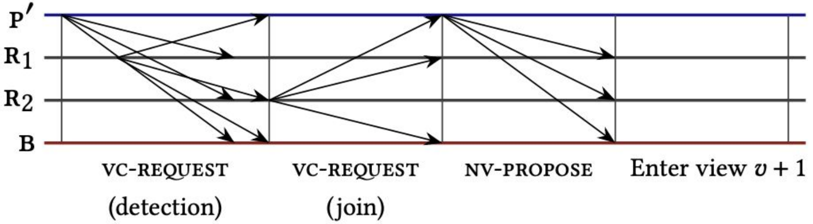
- The algorithm consists of 3 steps:

1. ***Failure Detection and View-Change Requests***
   - If a particular replica R detects failure of P in view v; it broadcasts a VC-REQUEST(v,E) message to all other replicas.
   - R detects the failure in 2 ways:
     - R timeouts
     - R receives VC-REQUEST messages



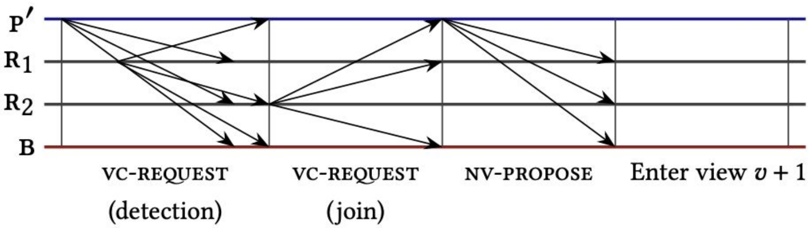
The View-Change Algorithm

1. *Failure Detection and View-Change Requests*
2. ***Proposing the New View***
    - In order to start view v+1, the new primary P’ proposes a new view after receiving valid VC-REQUEST messages from $S \subseteq \mathcal{R}$ ($|S| = nf$).
    - A VC-REQUEST is considered valid if it includes a consecutive sequence of pairs (CERTIFY($\langle h \rangle$, w, k), $\langle T \rangle_c$)
    - P’ collects nf such requests and proposes them to all replicas as a NV-PROPOSE message



The View-Change Algorithm

1. *Failure Detection and View-Change Requests*
2. *Proposing the New View*
3. ***Move to the New View***
   - R chooses for each transaction k the pair (CERTIFY($\langle h \rangle$, w, k), $\langle T \rangle_c$) from S.
   - R determines k_{max} and view-commits and executes these k_{max} requests.
   - R rollbacks any transaction not included in the proposal.
   - P’ then proposes the k_{max} + 1-th transaction.



Correctness of PoE

Theorem: *Consider a system in view v, in which the first $k - 1$ transactions have been executed by all non-faulty replicas, in which the primary is non-faulty, and communication is reliable. If the primary received $\langle T \rangle_c$, then the primary can use the normal-case algorithm to ensure that there is non-divergent execution of T.*

Proposition: *Let $\langle T \rangle_c$ be a request for which client c already received a proof-of-execution showing that T was executed as the k-th transaction of view v. If n > 3f, then every non-faulty replica that switches to a view $v' > v$ will preserve T as the k-th transaction of view v.*

Safety of PoE: *PoE provides speculative non-divergence if n > 3f*

Liveness of PoE: *PoE provides termination in periods of reliable bounded-delay communication if n > 3f.*

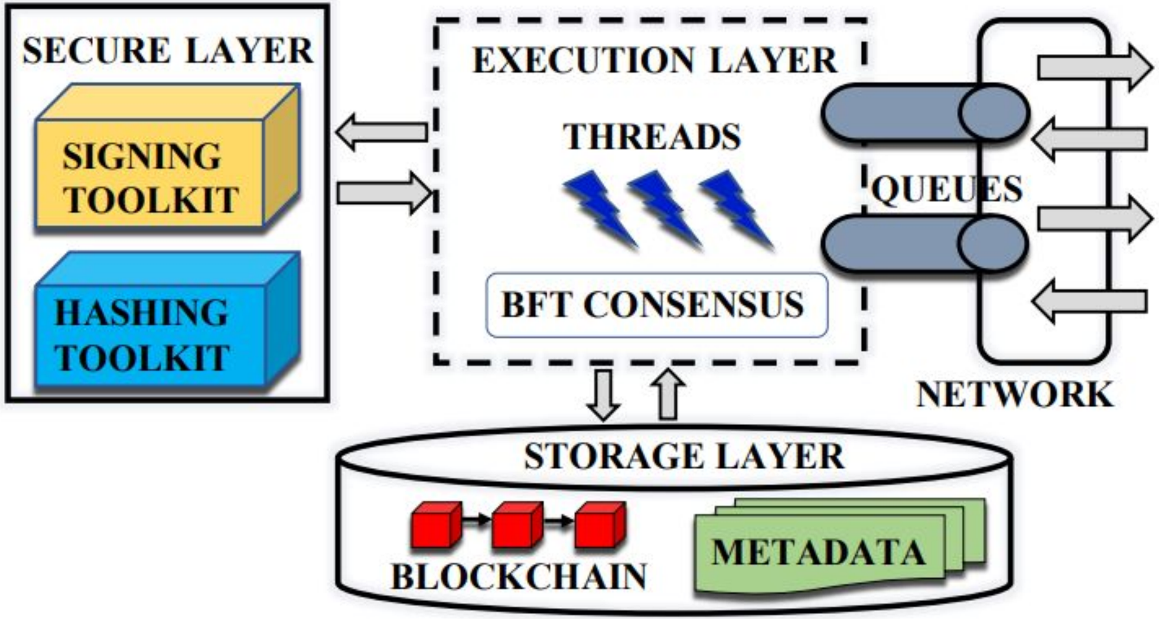
Fine-tuning and Optimization

- Optimizations:
  - **nf** signature shares also includes the primary
  - Needs **nf – 1** shares of other replicas
  - PROPOSE, SUPPORT, INFORM, and NV-PROPOSE messages→ not forwarded; signed using MACs
  - CERTIFY message are not signed while VC-REQUEST messages are signed and forwarded
  - PoE design fully compatible with out-of-order processing

ResilientDB Fabric

- Access to a state-of-the-art replicated transactional engine.
- Fulfills the need of a high-throughput permissioned blockchain fabric.
- Implement and test different consensus protocols.
- Balance the tasks done by a replica through a parallel pipelined architecture.
- Minimize the cost of communication through batching client transactions.
- Enable the use of a secure and efficient ledger.

ResilientDB Architecture



Batching

- ResilientDB facilitates batching requests at both replicas and clients.
- Multiple batch-threads that aggregate clients requests into a batch are spawned at the primary replica.
- Primary input-threads assign sequence numbers and enqueue client requests.
- Common lock-free queue shared by all batch-threads.
- When a client request is available batch-thread dequeues the request, appends client requests to a batch until size limit is met.
- Unique digest created by hashing the requests in a batch.

Ledger Management

- Blockchain- immutable ledger, blocks chained as a linked-list.

$B_i := \{k, d, v, H(B_{i-1})\}$

- First primary replica creates a genesis block.
- Each replica can independently create the next block in the blockchain.
- Each block corresponds to a batch of transactions. These transactions are executed by the system's execute-thread.
- A block is only created by the execute-thread once it completes executing a batch of transactions.
- To create a new block, the execute-thread hashes the previous block in the blockchain and then generates a new block with the necessary data for the batch of transactions it has executed.

Evaluation

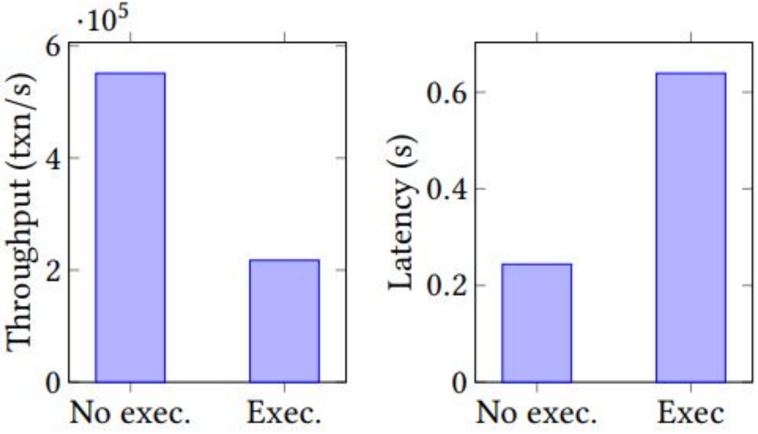
- Evaluated PoE protocol against four state-of-the-art BFT protocols.

1. Zyzzyva—absolute minimal cost in the fault-free case
2. PBFT—a common baseline
3. SBFT—safer variation of Zyzzyva
4. HotStuff—linear-communication protocol with rotating leaders

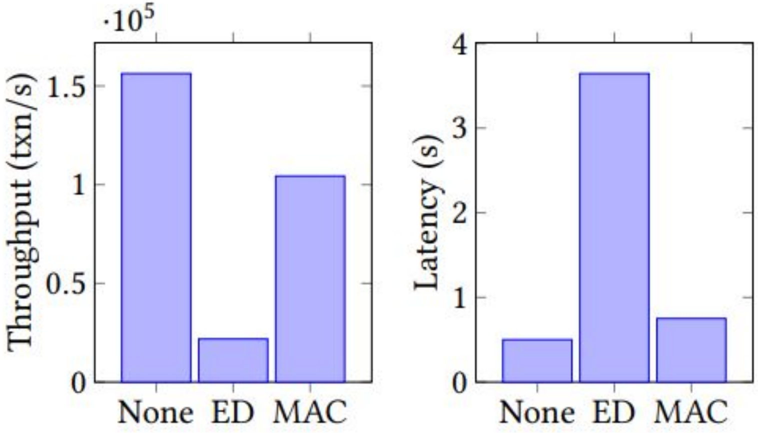
- Through experiments, following questions were answered -

1. How does PoE fare in comparison with the other protocols under failures?
2. Does PoE benefits from batching client requests?
3. How does PoE perform under zero payload?
4. How scalable is PoE on increasing the number of replicas participating in the consensus, in the normal-case?

Performance



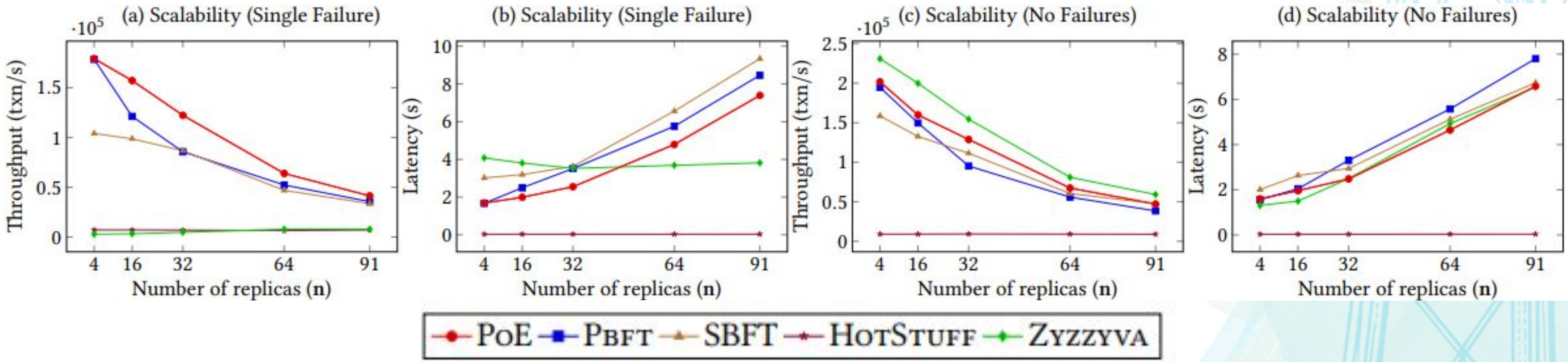
Upper Bound of performance



Performance using 3 signature schemes

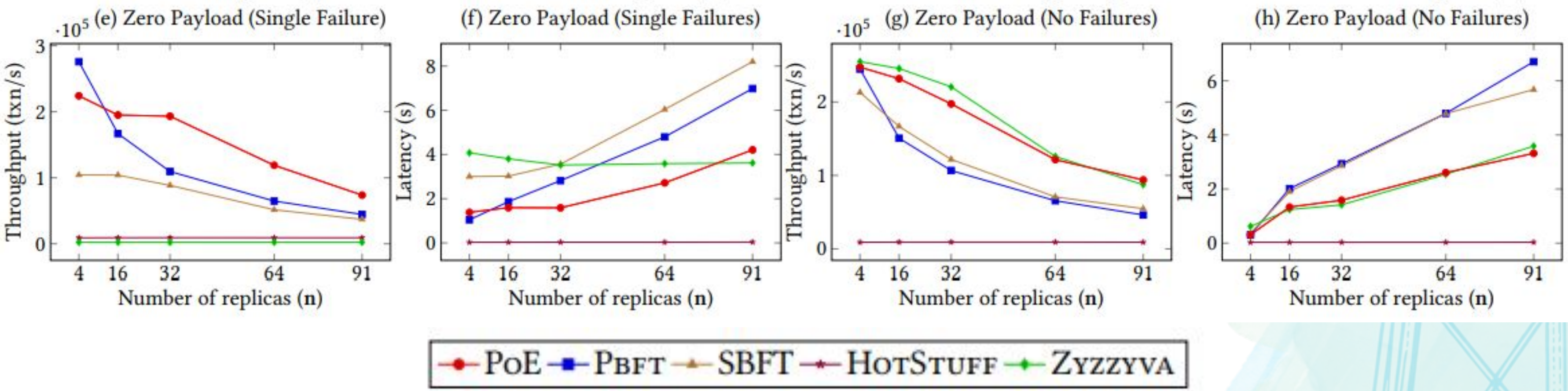
Scaling Replicas under Standard Payload

- PoE attains up to 43%, 72%, 24× and 62× more throughputs than PBFT, SBFT, HotStuff and Zyzzyva under a backup failure.
- Under no failures, PoE continues to outperform PBFT, SBFT and HotStuff.
- PoE has 20% (on 91 replicas) to 13% (on 4 replicas) less throughputs than Zyzzyva.
- PoE attains up to 35%, 27% and 21× more throughput than PBFT, SBFT and HotStuff, respectively.



Scaling Replicas under Zero Payload

- A zero payload experiment ensures that each replica executes dummy instructions.
- Zero payload leads to more throughput than the standard payload
- PoE attains up to 85% more throughput than PBFT, 62% more than SBFT, and 27× more than HotStuff.
- Under no failures, PoE's throughput is close to ZYZZYVA's.

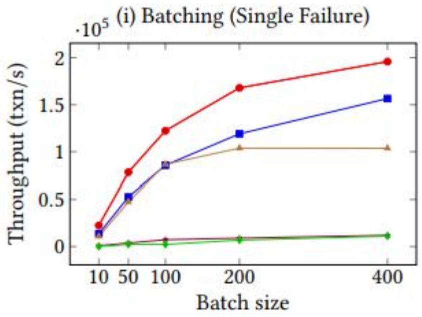


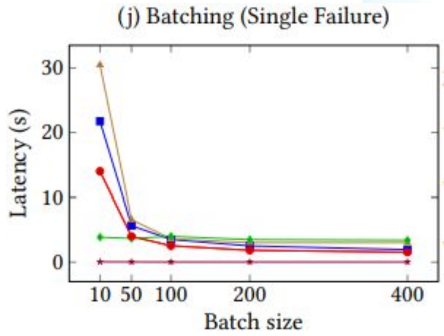
Impact of Batching under Failures

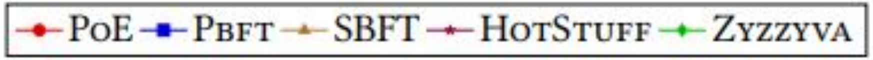
Setup: 32 replicas with one failure.

Results:

- Increasing batch size (10 to 400) increases throughput and decreases latency for all protocols.
- Throughput increase slows after batch-size 100 for PoE, PBFT, and SBFT.
- Zyzzyva underperforms due to inability to handle failures.
- HotStuff sees small throughput increase with larger batch sizes.

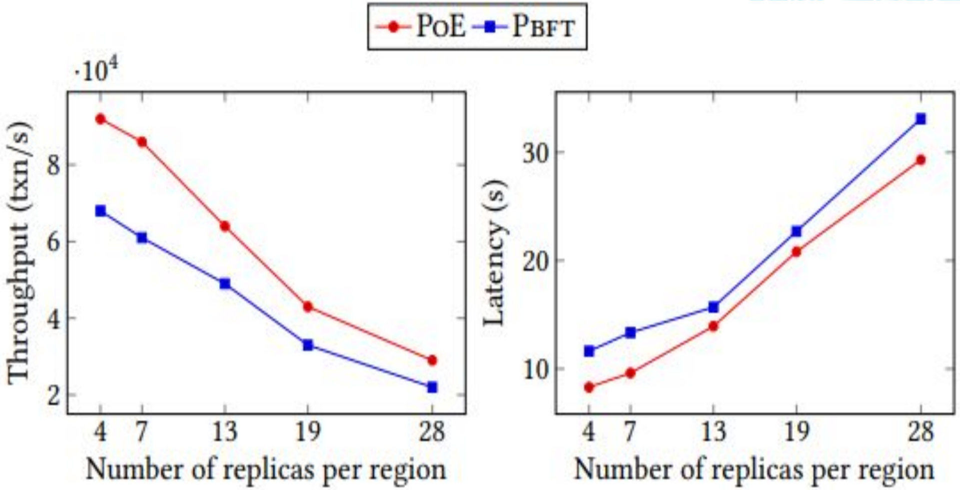






WAN Scalability

- Deployed clients and replicas across five global locations
- Varies the number of replicas from 20 to 140, distributing them evenly.
- PoE outperforms existing protocols, achieving 1.41× higher throughput and 28.67% less latency than PBFT.
- Plots for SBFT, HotStuff, and Zyzzyva are omitted due to their low throughputs in failure scenarios.



Performance of PoE and PBFT in a wide-area network with a single failure

Conclusion

- Proof-of-Execution (PoE), a novel Byzantine fault tolerant consensus protocol is presented.
- Guarantees both safety and liveness in just three linear phases.
- Out-of-order processing and speculative execution manages to reduce the costs of BFT while guaranteeing reliable consensus for clients.
- PoE implemented in ResilientDB outperforms existing BFT protocols by up to 80% in throughput in the presence of failures.
- PoE represents a significant step forward in the field of consensus protocols, offering improved performance and reliability for distributed systems.

THANK YOU