Presentation of the paper “Proof-of-Execution: Reaching Consensus through Fault-Tolerant Speculation”

Presenter: Haojun (Howard) Zhu

Background

Due to time-tested safe design of PBFT and due to the limiting designs of other BFT protocols, a large set of distributed and blockchain applications still depend on the classical PBFT protocol, even though, PBFT requires three phases of communication, of which two necessitate quadratic communication. The paper I presented introduces the Proof-of-Execution consensus protocol (PoE), which achieves consensus in just three linear phases.

To concoct POE, we start with PBFT and successively add four key ingredients:

- Non-Divergent Speculative Execution
- Safe Rollbacks and Robustness under Failures
- Agnostic Signatures and Linear Communication
- Avoid Response Aggregation

Notation specifications

Before providing a full description of the PoE protocol, let me present the system model it uses and the relevant notations.

- (R, C)
- id(R) with $0 \leq id(R) < |R|$
- $F \in R$
- $NF = R \backslash F$
- $n = |R|$
- $f = |F|$
- $nf = |NF|$
- assume that $n > 3f$ ($nf > 2f$)

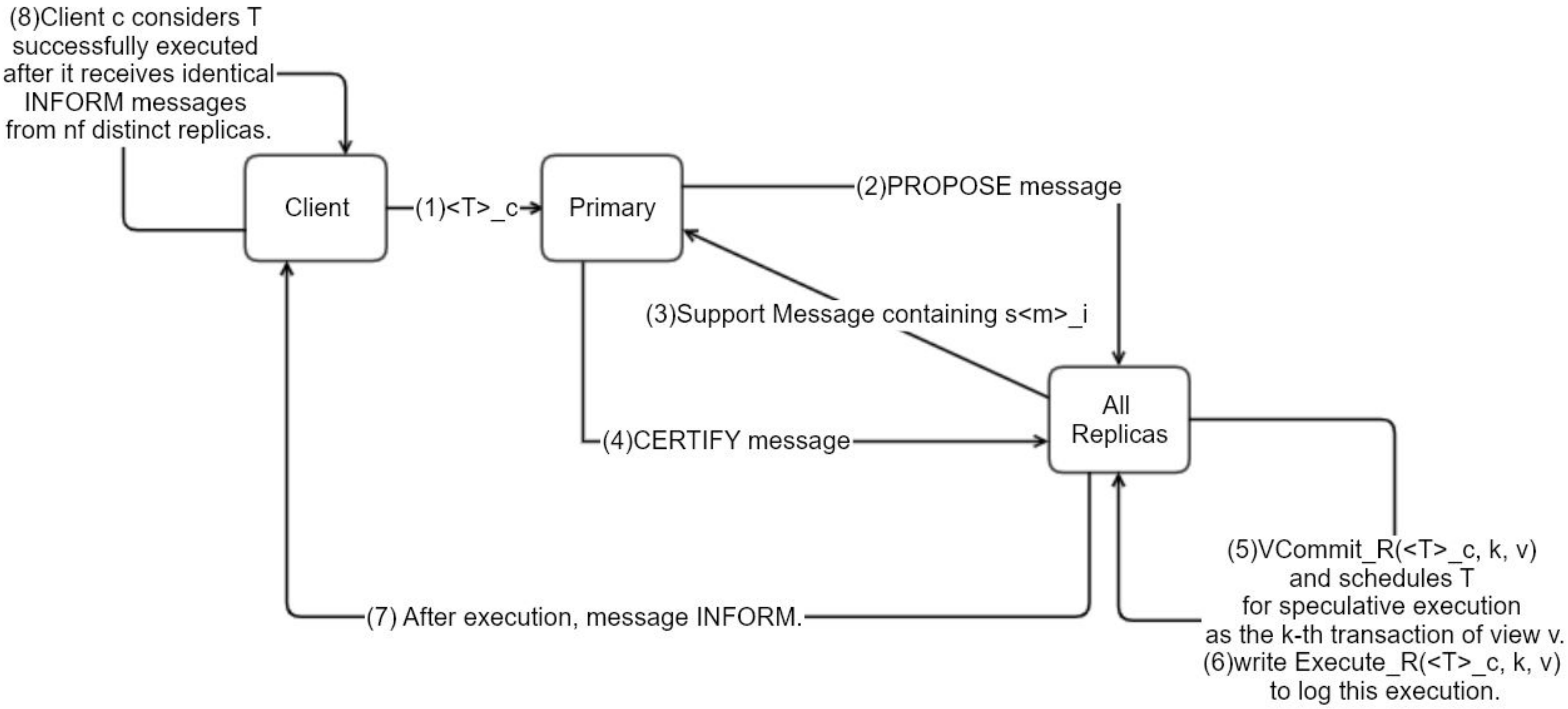
- assume authenticated communication: byzantine replicas are able to impersonate each other, but replicas cannot impersonate non-faulty replicas.
- MACs: message authentication codes
- TSs: threshold signatures
- $s\langle v\rangle_i$
- $\mathrm{D}(\cdot)$
- $\langle T\rangle_c$

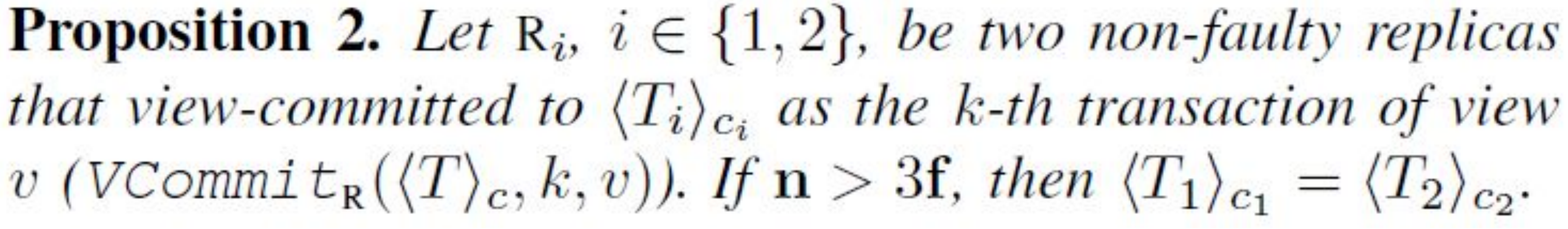
Definition

Let (R; C) be a system. A single run of any consensus protocol should satisfy the following requirements:

- Termination
- Non-divergence
- Speculative non-divergence

Normal-case algorithm of PoE


(8)Client c considers T
successfully executed
after it receives identical
INFORM messages
from nf distinct replicas.
Client
(1)<T>_c
Primary
(2)PROPOSE message
(3)Support Message containing s<m>_i
(4)CERTIFY message
All
Replicas
(5)VCommit_R(<T>_c, k, v)
and schedules T
for speculative execution
as the k-th transaction of view v.
(6)write Execute_R(<T>_c, k, v)
to log this execution.
(7) After execution, message INFORM.



Proposition 2. *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 *(*$\texttt{VCommit}_{\mathrm{R}}(\langle T \rangle_c, k, v)$*). If* $\mathbf{n} > 3\mathbf{f}$*, then* $\langle T_1 \rangle_{c_1} = \langle T_2 \rangle_{c_2}$.

Proof

In the process just described in the last page, if a replica R_i needs to view-commit to <T_i>_(c_i), it must signature shares from a set S_i of nf distinct replicas.

Define S_i := set of nf replicas that send SUPPORT message to primary

Define X_i := S_i \ F

|X_i| ≥ nf - f as S_i is a fixed value nf and the number of faulty replicas in X_i is ≤ f

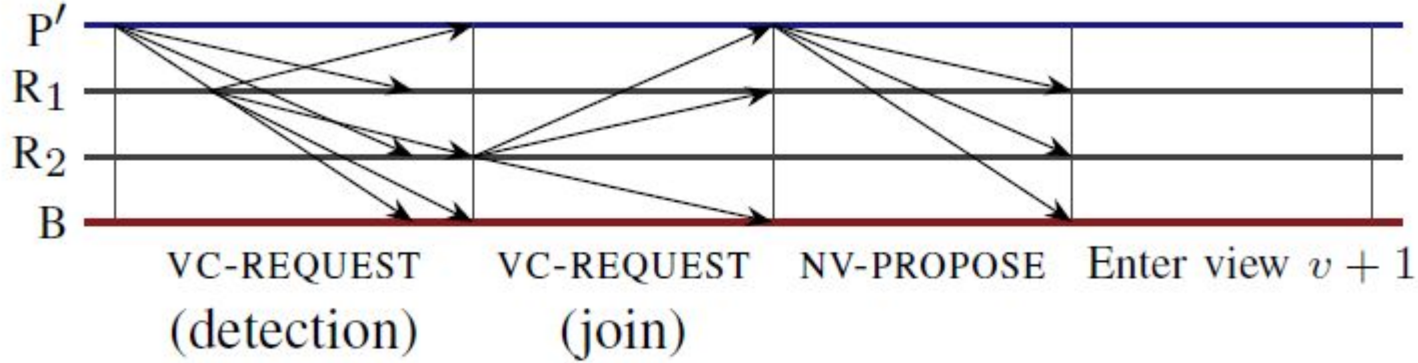
We have n ≥ |X_1∪X_2 | ≥ 2(nf - f). As n = nf + f , this simplifies to 3f ≥ n, which contradicts n＞3f.

Three typical cases a malicious primary can try to affect PoE by not conforming to the normal-case algorithm

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

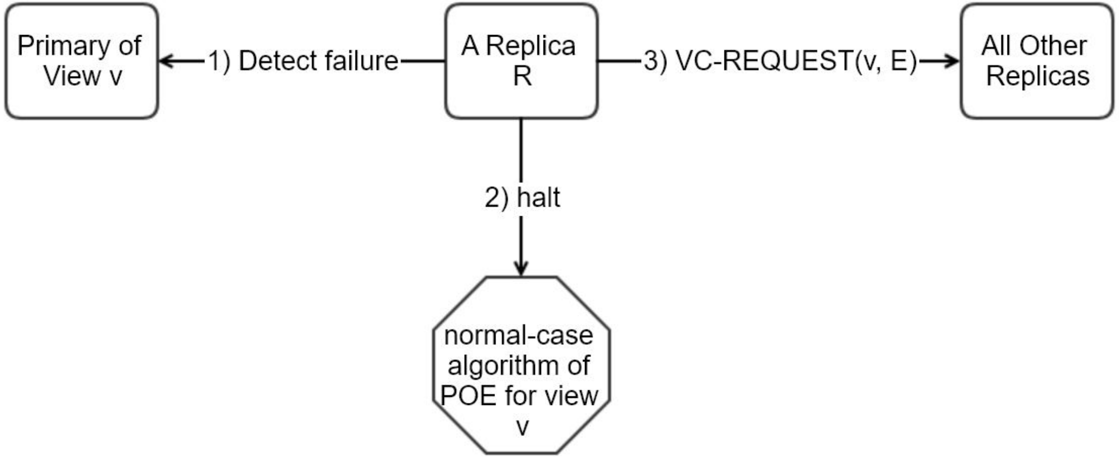
Overview



The view-change algorithm

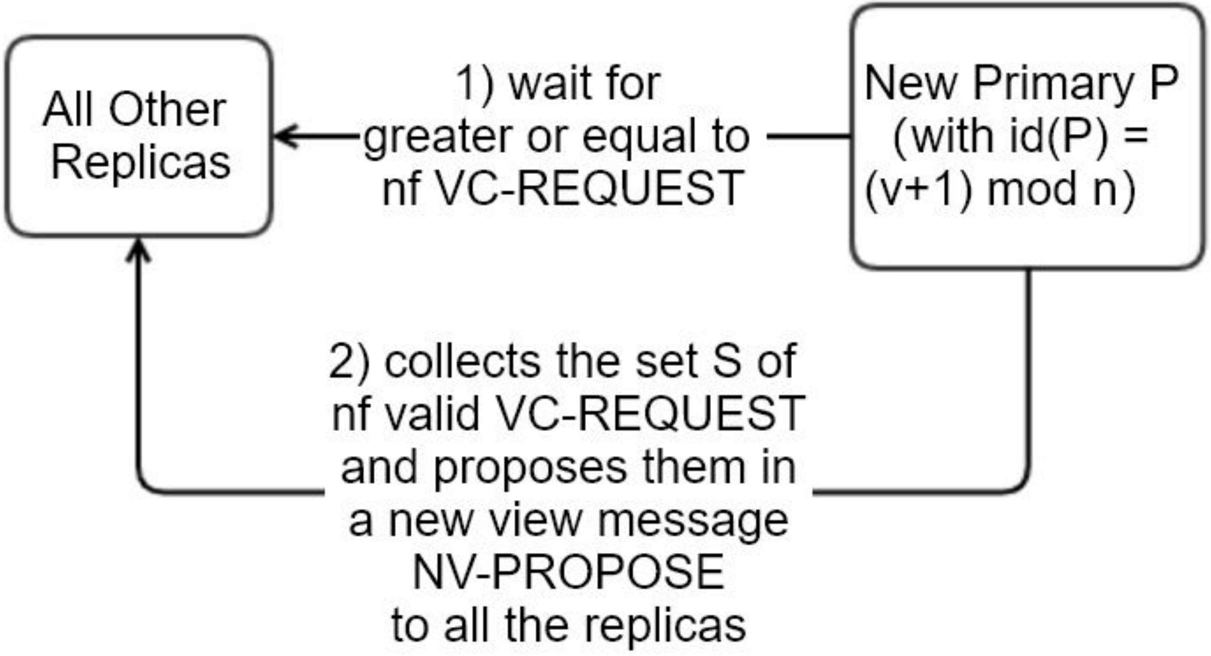
Detail 1) Failure detection and view-change requests

Each replica R can detect the failure of primary in the following two ways:
1) R timeouts while expecting normal-case operations toward executing a client request.
2) R receives VC-REQUEST messages indicating that the primary of v failed from f+1 distinct replicas.



The view-change algorithm

Detail 2) Proposing the new view



All Other Replicas

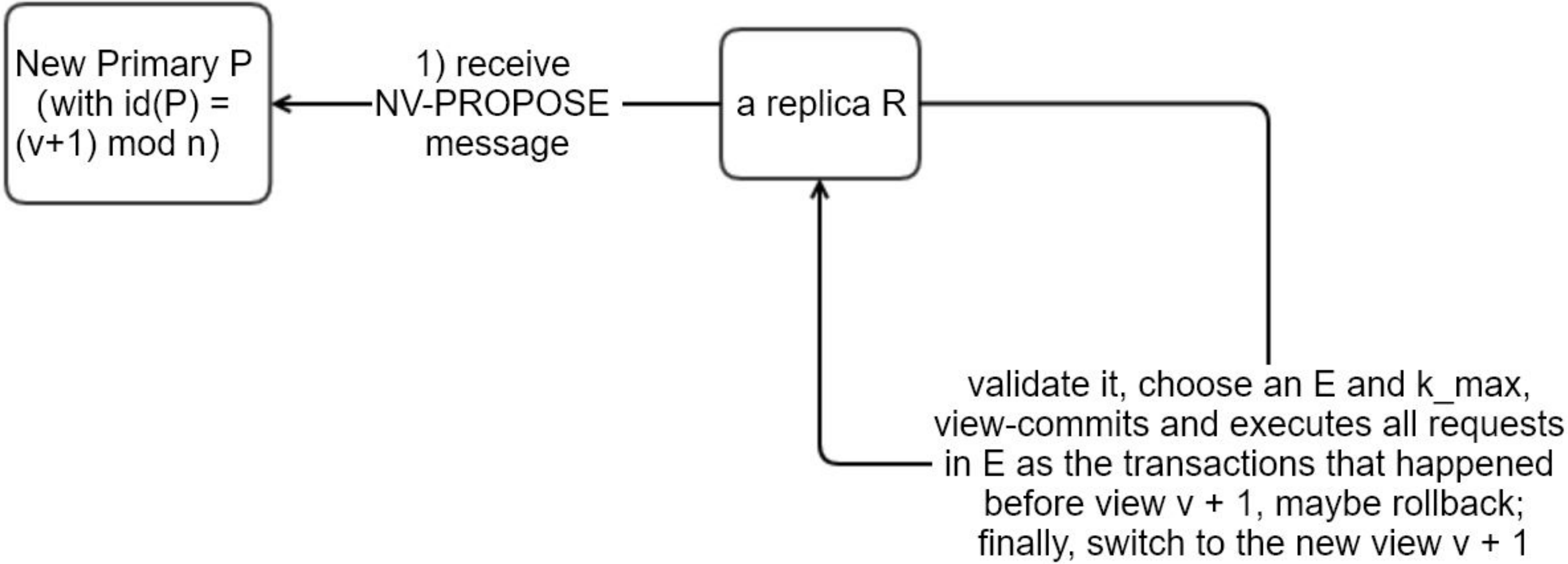
1) wait for greater or equal to nf VC-REQUEST

New Primary P (with id(P) = (v+1) mod n)

2) collects the set S of nf valid VC-REQUEST and proposes them in a new view message NV-PROPOSE to all the replicas

The view-change algorithm

Detail 3) Move to the new view



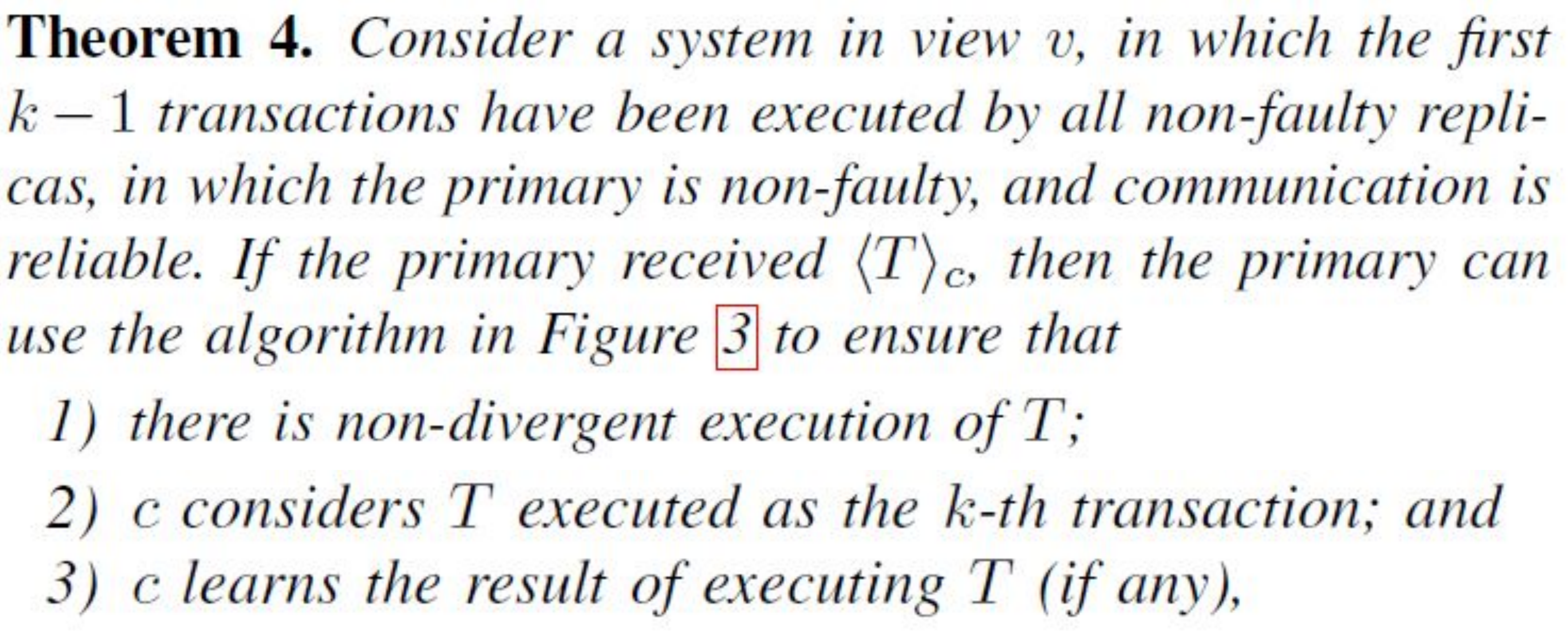
New Primary P (with id(P) = (v+1) mod n)

1) receive NV-PROPOSE message

a replica R

validate it, choose an E and k_max, view-commits and executes all requests in E as the transactions that happened before view v + 1, maybe rollback; finally, switch to the new view v + 1

Correctness of PoE



Theorem 4. *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 algorithm in Figure 3 to ensure that*

1) *there is non-divergent execution of T;*
2) *c considers T executed as the k-th transaction; and*
3) *c learns the result of executing T (if any),*

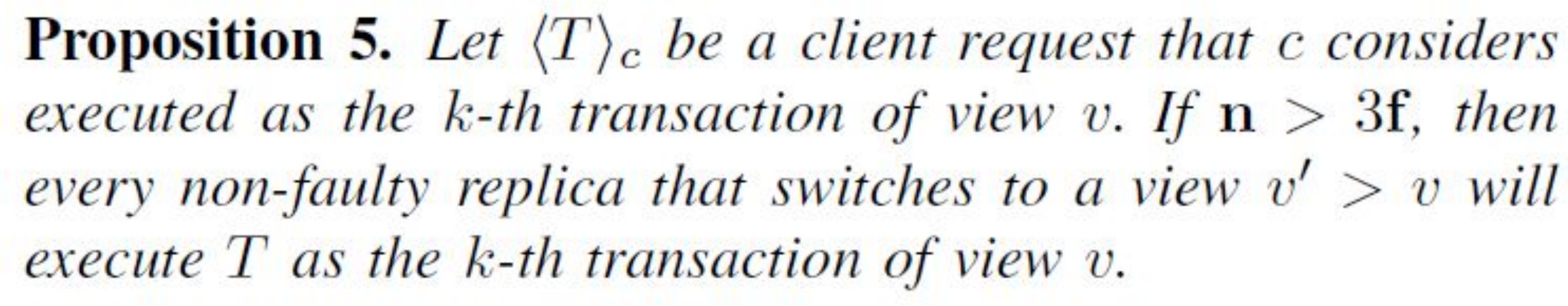
Theorem 4 proof

Because all non-faulty replica will execute k-th transaction T and behave deterministically, they will yield the same result r (if any) across all non-faulty replicas and inform client c by all sending identical INFORM messages.

As all nf non-faulty replicas executed T, we have non-divergent execution.

Because n ＞3f, f faulty replicas can not forge invalid INFORM messages to client c and c will conclude that T is executed yielding result r.

Correctness of PoE



Proposition 5. *Let $\langle T \rangle_c$ be a client request that c considers executed as the k-th transaction of view v. If $\mathbf{n} > 3\mathbf{f}$, then every non-faulty replica that switches to a view $v' > v$ will execute T as the k-th transaction of view v.*

Proposition 5 proof Part I

set A: nf distinct replicas the client receive messages from

set B: B = A\F be the set of non-faulty replicas in A

set C: The set of nf distinct replicas that provided messages to clients

set D: Let D = C\F be the set of non-faulty replicas in C

We have $|B| \geq nf - f$, $|D| \geq nf - f$ and $2(nf - f) > nf$, hence $|B \cap D| \geq 1 \Rightarrow$ there exists a non-faulty replica $Q \in (B \cap D)$ that executed <T>_c, informed c and requested a view-change. Hence, the new-view change must contains <T>_c.

Proposition 5 proof Part II

To complete the proof, we need to show that <T>_c is part of the message m_i, 1 ≤ i ≤ nf, with the longest consecutive sequence of executed transactions. Due to Proposition 2, the only possibility that <T>_c doesn’t belong to what we mentioned just now is that <T>_c is in a different view.

Without loss of generality, we can assume the view w which <T>_c belongs to satisfies w > v. As faulty replicas can only forge f signature shares, there must be a set E of nf - f non-faulty replicas that contributed to CERTIFY(<h_w>, w, k). Each of these replicas must have entered view w after processing some new-view proposal m. According to part I proof, m must contain <T>_c, so it’s impossible that <T>_c is in view w. And <T>_c will be executed by R upon entering view v' .

Corollary 6 (Safety of PoE)**.** PoE *provides speculative non-divergence if* $\mathbf{n} > 3\mathbf{f}$.

Theorem 7 (Liveness of PoE)**.** PoE *provides termination in periods of reliable bounded-delay communication if* $\mathbf{n} > 3\mathbf{f}$.

Out-of-order message processing

Single-primary protocols: PBFT, SBFT and PoE

Property: Multiple PROPOSE messages can be pipelined and parallelized.

Replicas can process messages out-of-order while they continue executing transactions in the sequence-order.

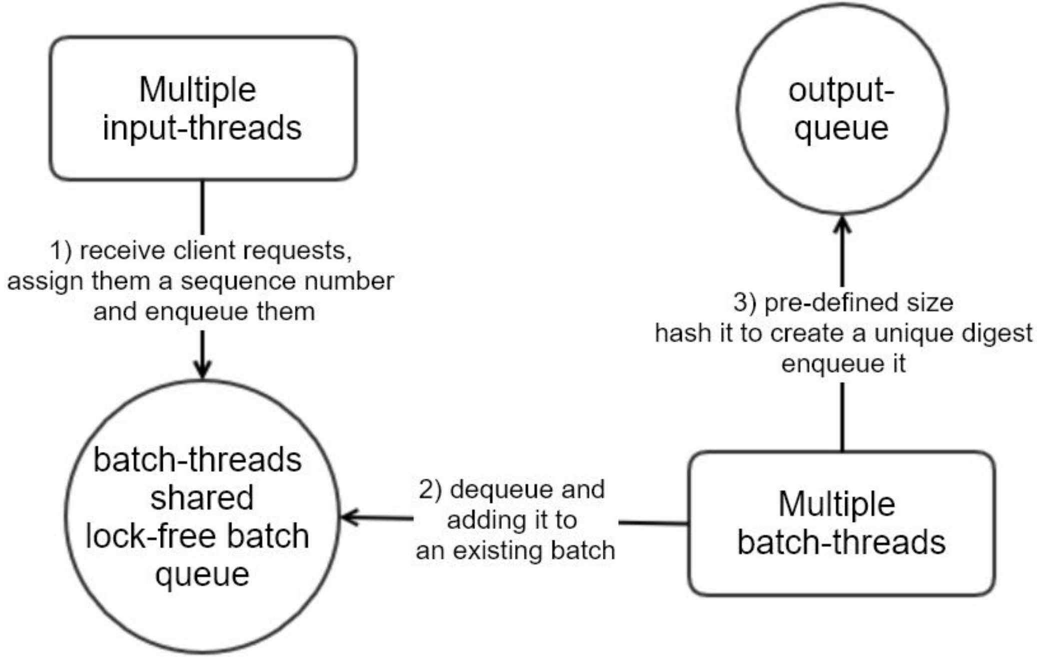
In case of PoE:

- A backup replica only accepts k-th proposal from the primary if it had not previously supported another k-th proposal.
- A replica only executes a k-th request if it has executed requests for all the previous rounds.

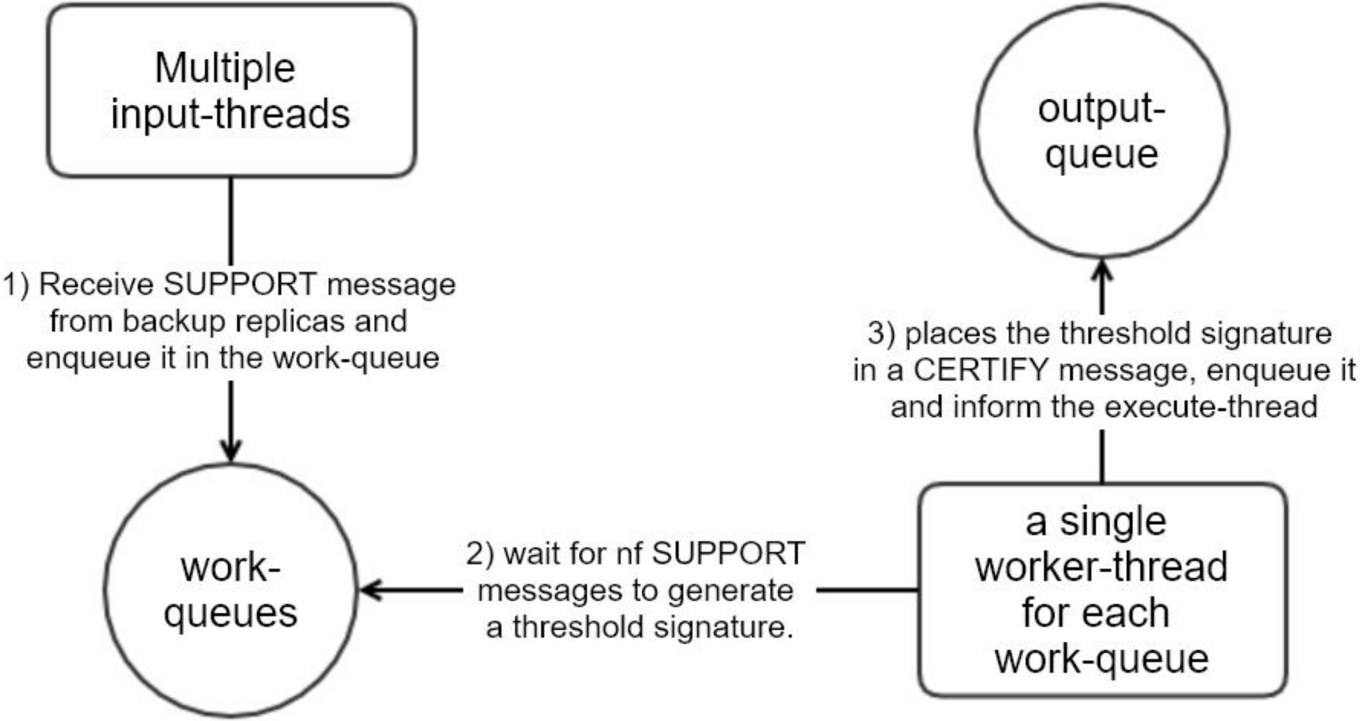
Hence, the primary can send out PROPOSE message for (k+1)-th request before the completion of the k-th request.

Implementations on ResilientDB fabric

Primary replica


Multiple input-threads
1) receive client requests, assign them a sequence number and enqueue them
batch-threads shared lock-free batch queue
2) dequeue and adding it to an existing batch
Multiple batch-threads
3) pre-defined size hash it to create a unique digest enqueue it
output-queue

Primary replica



Multiple input-threads

1) Receive SUPPORT message from backup replicas and enqueue it in the work-queue

work-queues

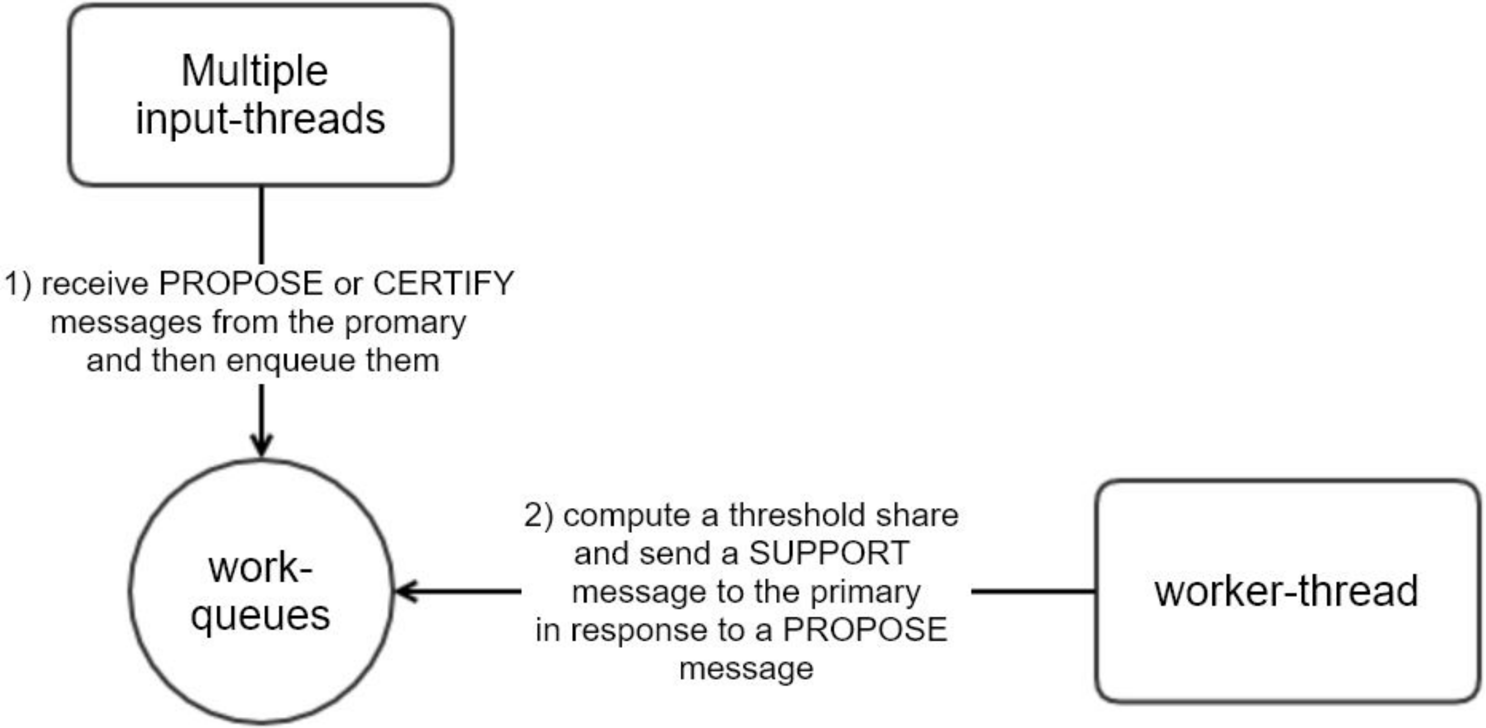
2) wait for nf SUPPORT messages to generate a threshold signature.

a single worker-thread for each work-queue

3) places the threshold signature in a CERTIFY message, enqueue it and inform the execute-thread

output-queue

Backup replicas


Multiple input-threads
1) receive PROPOSE or CERTIFY messages from the promary and then enqueue them
work-queues
2) compute a threshold share and send a SUPPORT message to the primary in response to a PROPOSE message
worker-thread

Execution

At each replica we also have an execute-thread that executes all the requests in accordance with the normal case algorithm. Once the execution is complete, the execution thread creates an INFORM message and places it in the output queue for the output-thread to send it to the client.

Evaluation

To evaluate POE’s design, the paper implements it in the ResilientDB fabric together with PBFT, Zyzzyva, SBFT and HotStuff and do many kinds of experiments and show that POE achieves up to 80% more throughput than its compared ones.

Related work and conclusion

Proof-of-Execution (PoE) is a novel Byzantine fault-tolerant consensus protocol that guarantees safety and liveness in only three linear phases. PoE may require replicas to revert executed transactions because of speculative execution.

The experiments on ResilientDB fabric show that PoE can achieve up to 80% more throughput than the currently existing BFT protocols.

Questions?