Practical Byzantine Fault Tolerance

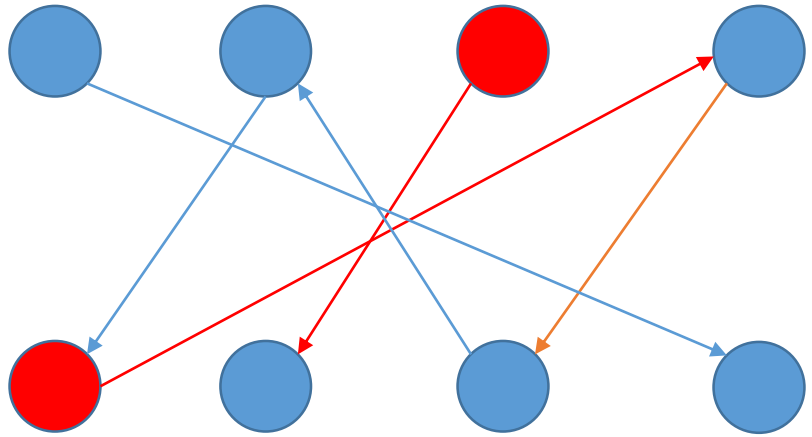
Tal Dov Kopelevitch

Content

- The Byzantine model for distributed computations.
- The service properties for our algorithm.
- An extended explanation of the algorithm.
- Optimizations for the initial algorithm.
- The performances of a tested implementation.
- Conclusions.

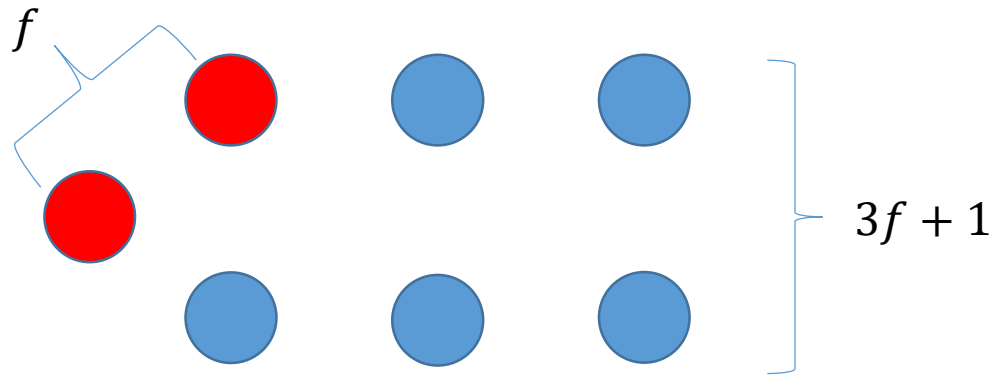
The Byzantine Model

- Each component(replica) can be faulty or non-faulty.
- Faulty component is one which doesn’t operate as expected.
- A faulty component can try to sabotage the safety of the information or the correction of the computation.



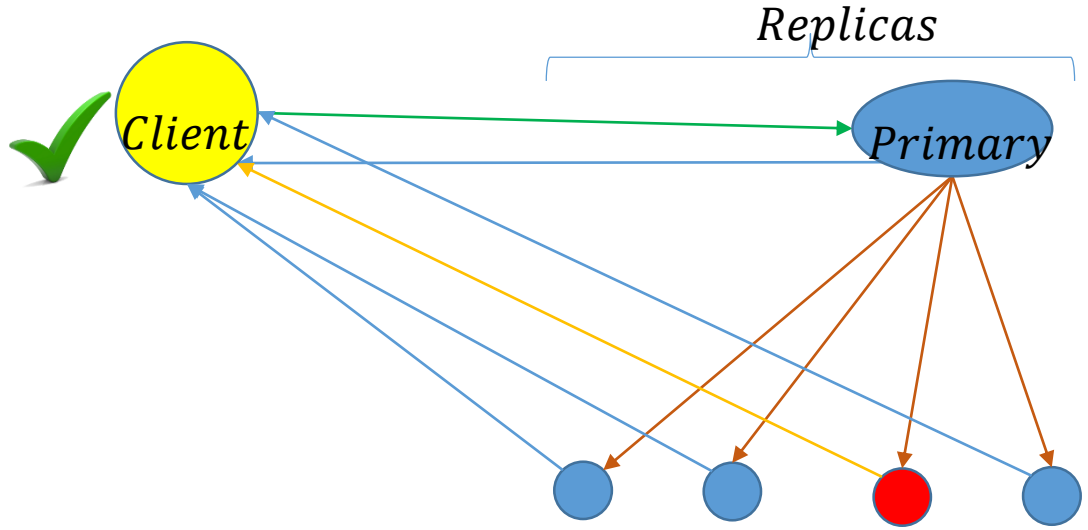
The Service Properties

- The network might not work as expected – asynchronous system.
- A replicated system – operates as one computer.
- Includes read-only and write-read operations.
- We will assume that if we have $3f + 1$replicas, the number of faulty ones is bounded by f.
- The algorithm provide both safety and liveness.



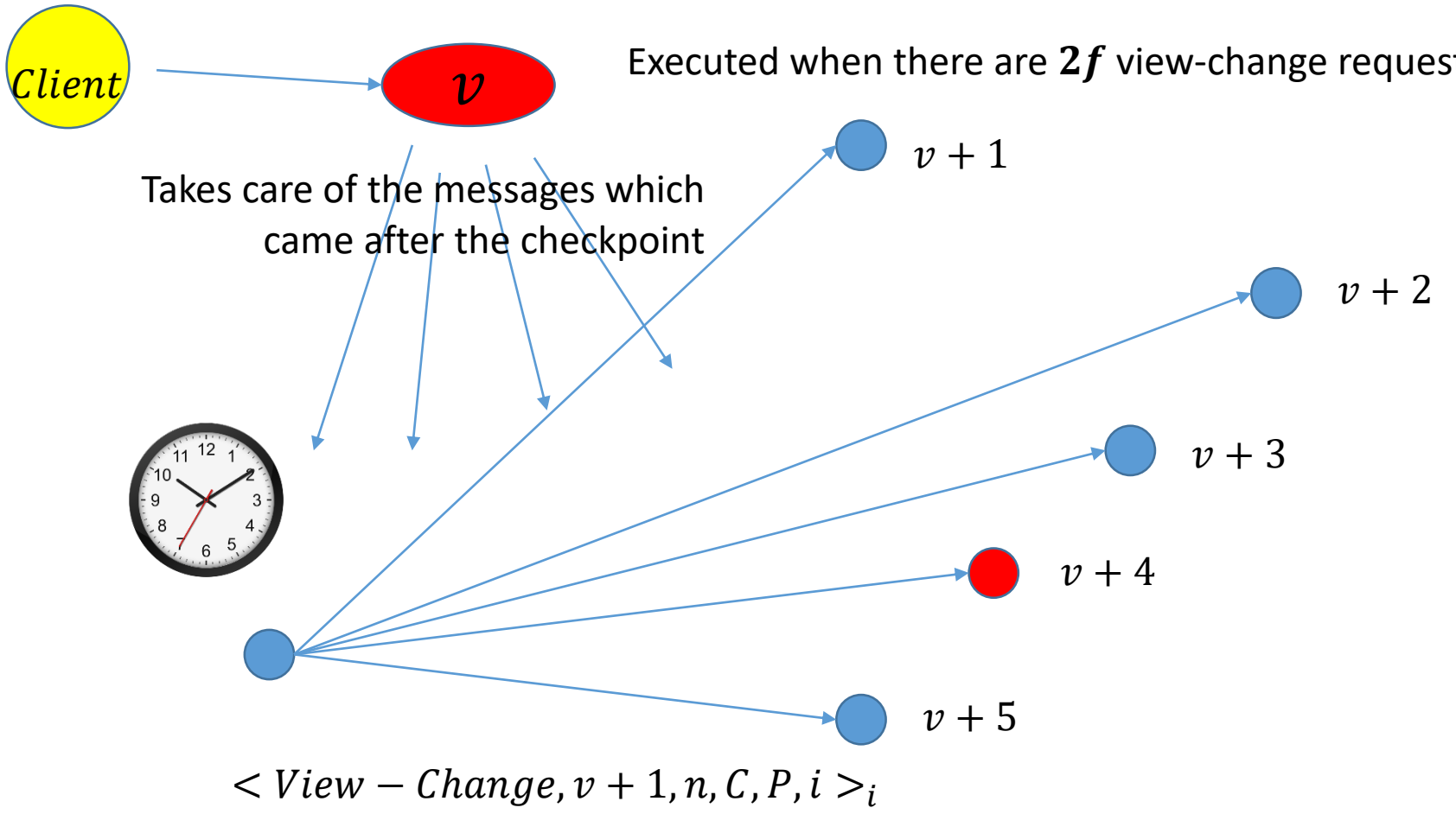
The Algorithm

- We use a primary replica when the others are backups.
- For each client request the algorithm will work as follows:



- The client receives a result when $f + 1$ replicas replayed the same.

View Changes



The Client

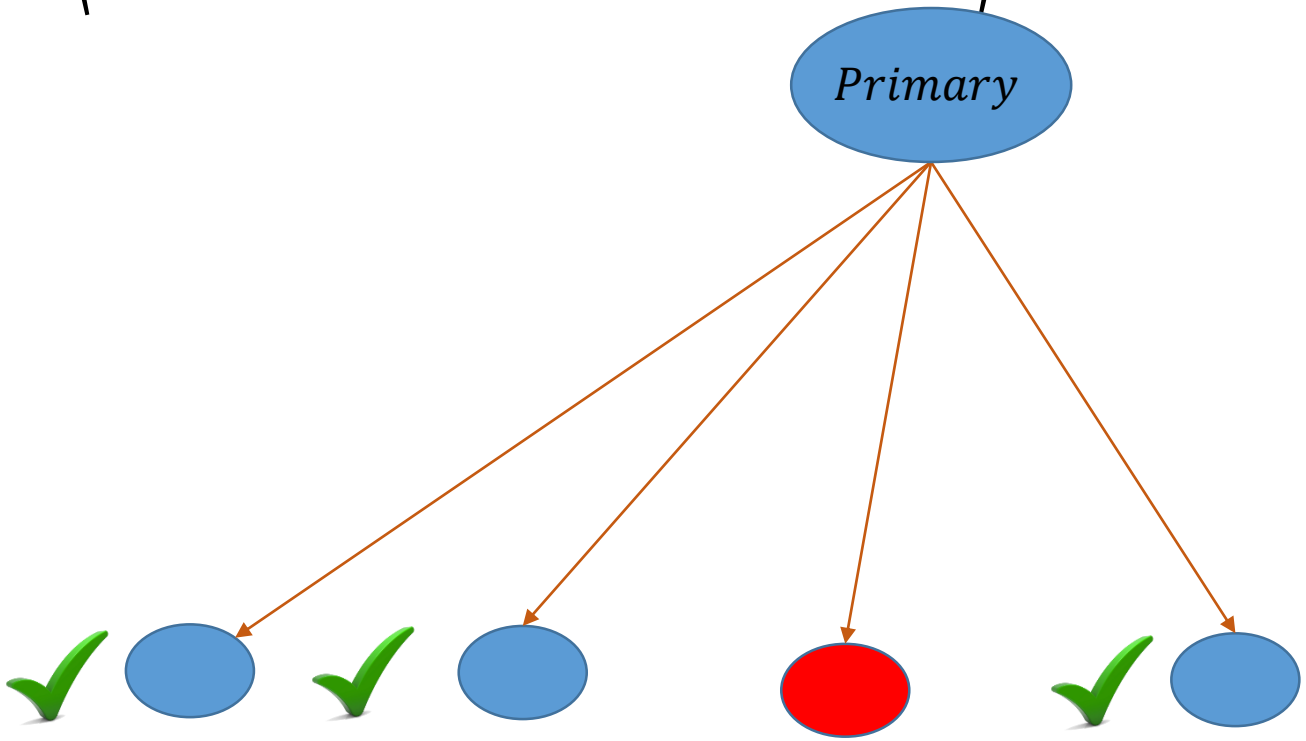
- A client c sends a $Request$ message, $\langle Request, o, t, c \rangle_c$ to the primary.
- The replicas sends a $Reply$ message containing the view, the replica's id i, c, the result r and t.
- If the client did not receive $f + 1$ replies soon enough, it broadcasts the request to all the replicas, which replies directly to it.

Normal Case Operation Protocol

- When the primary receives the request it starts a three phase protocol: Pre-prepare, prepare and commit.
- The two first phases are used to order requests even when the primary is faulty.
- The prepare and commit phases used to overcome view changes, so it do not disorder committed requests.

Pre Prepare

- The primary assign a sequence number n to the request.
- It multicasts $\left\langle \langle Pre - Prepare, v, n, d \rangle_p, m \right\rangle$ to the backups.

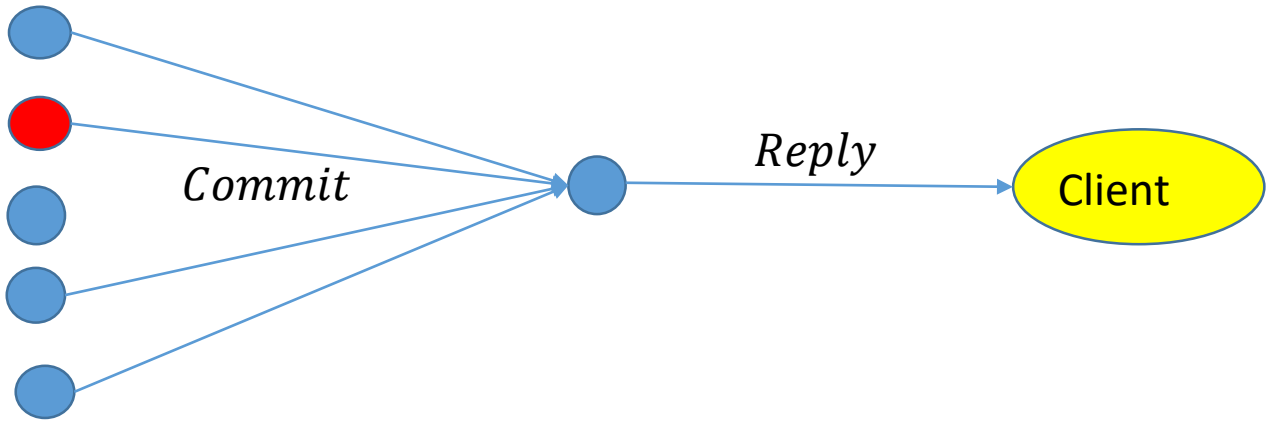


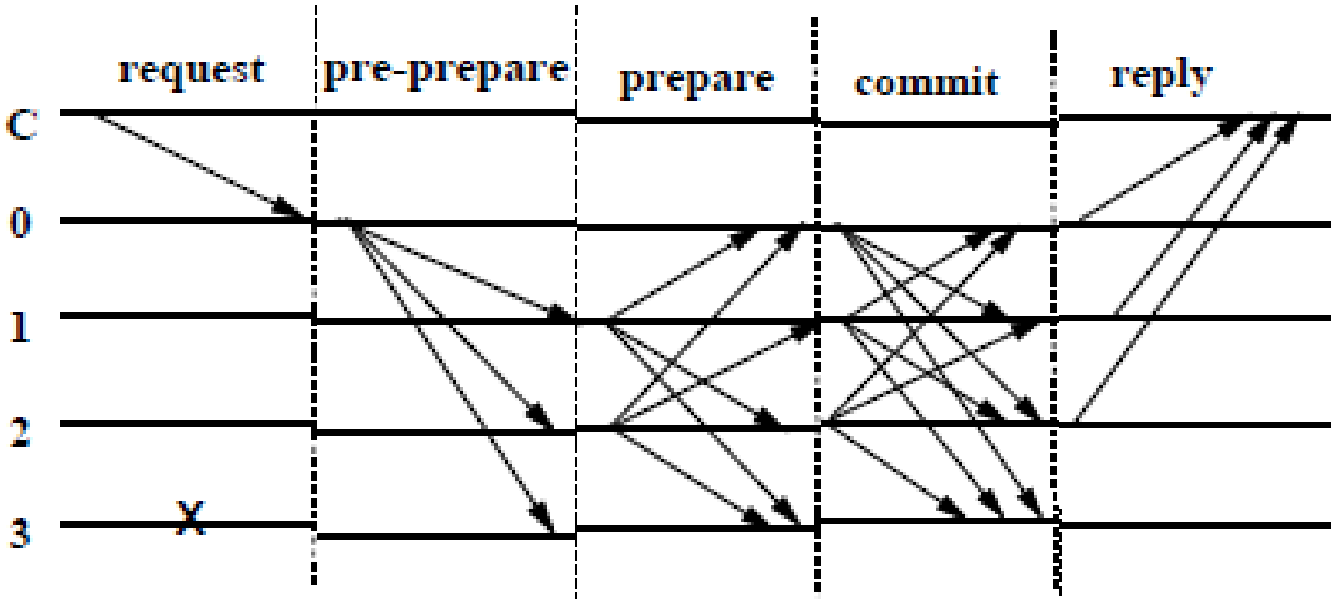
Prepare

- When backup i accepts pre-prepare message it's sends a prepare message $\langle Prepare, v, n, d, i \rangle_i$ to all the replicas and also sends it to the replica's message log.
- When a replica accepts a prepare message it provides that the signature, view and sequence number is valid.
- When a replica accepts $2f$ prepare messages with the same view and sequence number, it insert it to it's log, and starts the commit phase.

Commit

- Each replica i multicasts a commit message, $\langle Commit, v, n, D(m), i \rangle_i$ to the other replicas once it accepts $2f$ prepare messages for v and n.
- When a replica receives a commit message it's ensures that it's valid.
- A replica i executes the request if it accepted commit messages for m, v, and n from $2f + 1$ replicas.




request
pre-prepare
prepare
commit
reply
C
0
1
2
3
X

Garbage Collection

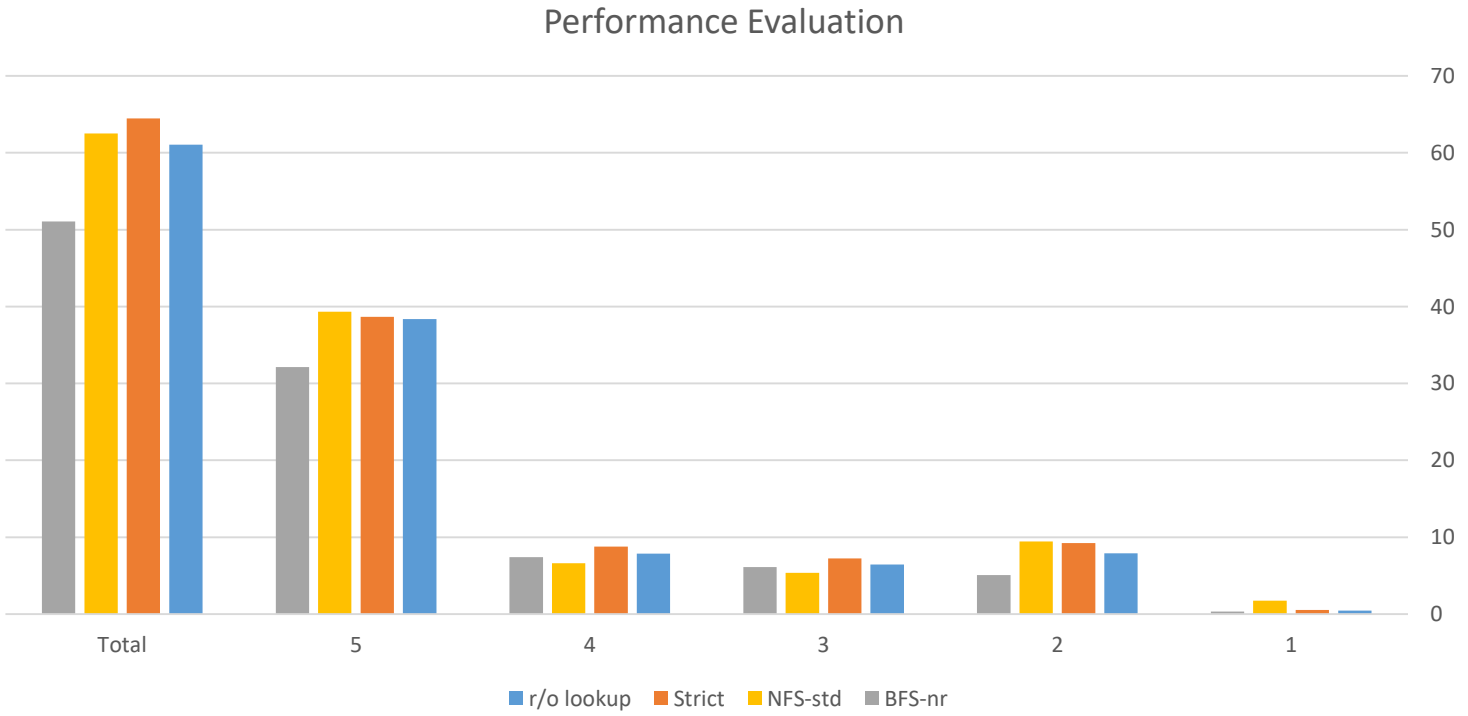
- We will clear the log every k commits, when k is predetermined.
- Every replica will remember the last $stable\ checkpoint$, the last service state that proved correctly.
- When a replica i produced a checkpoint, it send it to the other replicas, in a message containing the state, the last request number n, and i.
- If a replica gets $2f + 1$ it clears all the messages with sequence number less or equal to n.

Optimizations

- Only the first reply for a request is a full reply message.
- When a replica execute a request it sends a reply to the client, whose waiting for 2f+1 replies.
- The client sends read-only requests directly to the replicas.
- We do not check the correctness of the full sign, but only the lest-significant bytes.
- We use the digital signing only in new-view and change-view messages.

Performances

- To compare the performances we use Andrew benchmark, which has five phases, two initialize the system, two checking the files in all the replicas, and one which compiles and links the replicas.



Conclusions

- The algorithm works in an asynchronous system, although it a replicated state machine.
- The algorithm avoids using digital signatures, which sufficiently improves the performance.
- The algorithm tolerates software errors occur independently.