EECS 262a
Advanced Topics in Computer Systems
Lecture 25

Byzantine Agreement
April 25th, 2016

John Kubiatowicz
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www.eecs.berkeley.edu/~kubitron/cs262

Today's Papers

- The Byzantine Generals Problem, Leslie Lamport, Robert Shostak, and Marshall Pease. Appears in *ACM Transactions on Programming Languages and Systems* (TOPLAS), Vol. 4, No. 3, July 1982, pp 382-401
- Practical Byzantine Fault Tolerance, M. Castro and B. Liskov. Appears In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation* (OSDI), 1999.

- Thoughts?

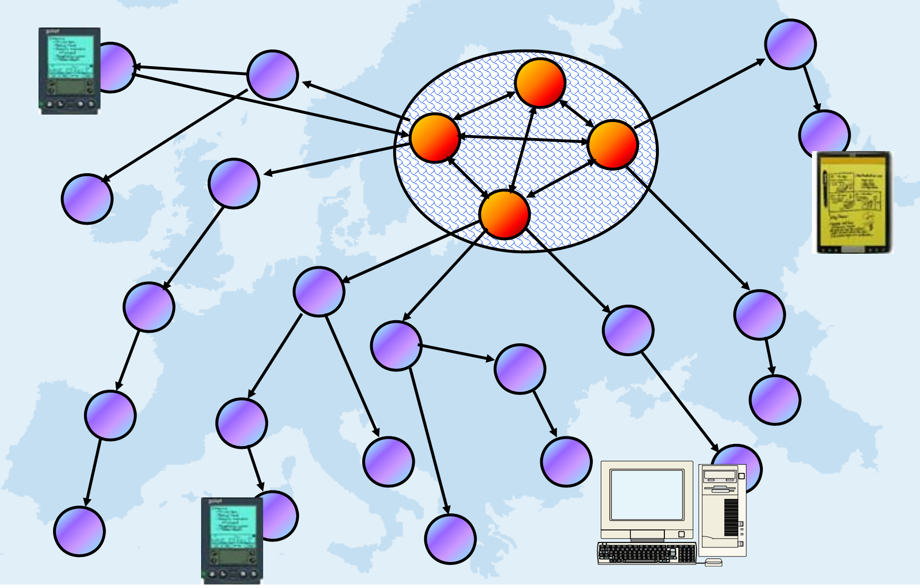
Motivation

- Coping with failures in computer systems
- Failed component sends conflicting information to different parts of system.
- Agreement in the presence of faults.
- P2P Networks?
  – Good nodes have to "agree to do the same thing".
  – Faulty nodes generate corrupted and misleading messages.
  – Non-malicious: Software bugs, hardware failures, power failures
  – Malicious reasons: Machine compromised.

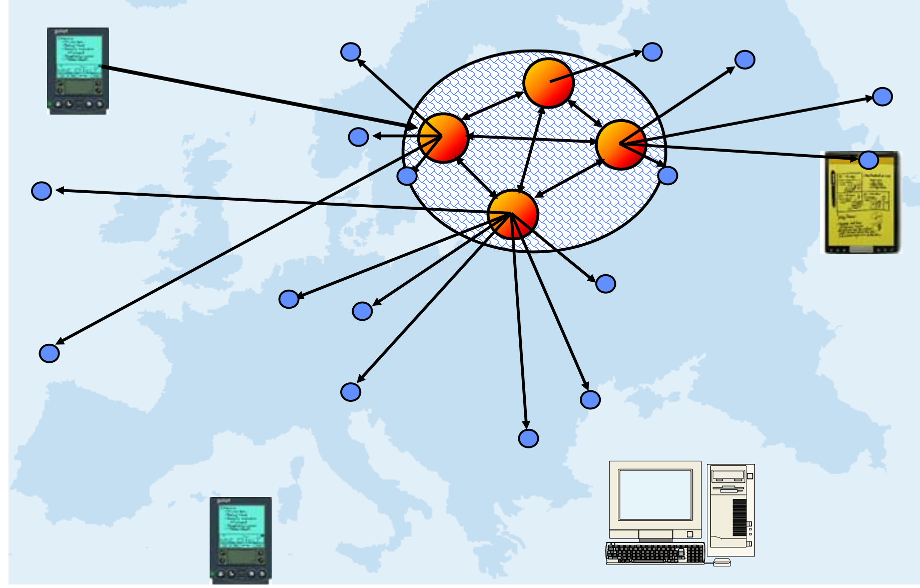
Problem Definition

- Generals = Computer Components
- The abstract problem…
  – Each division of Byzantine army is directed by its own general.
  – There are n Generals, some of which are traitors.
  – All armies are camped outside enemy castle, observing enemy.
  – Communicate with each other by messengers.
  – Requirements:
    » G1: All loyal generals decide upon the same plan of action
    » G2: A small number of traitors cannot cause the loyal generals to adopt a bad plan
  – Note: We **do not** have to identify the traitors.

Recall: The Path of an OceanStore Update



Recall: Archival Dissemination of Fragments



Differing Degrees of Responsibility

- Inner-ring provides quality of service
  - Handles of live data and write access control
  - Focus utility resources on this vital service
  - Compromised servers must be detected quickly
  - Byzantine Agreement important here!
- Caching service can be provided by anyone
  - Data encrypted and self-verifying
  - Pay for service "Caching Kiosks"?
- Archival Storage and Repair
  - Read-only data: easier to authenticate and repair
  - Tradeoff redundancy for responsiveness
- Could be provided by different companies!

Naïve solution

- i^{th} general sends v(i) to all other generals
- To deal with two requirements:
  - All generals combine their information v(1), v(2), .., v(n) in the same way
  - Majority (v(1), v(2), …, v(n)), ignore minority traitors
- Naïve solution does not work:
  - Traitors may send different values to different generals.
  - Loyal generals might get conflicting values from traitors
- **Requirement:** Any two loyal generals must use the same value of v(i) to decide on same plan of action.

Reduction of General Problem

- **Insight:** We can restrict ourselves to the problem of one general sending its order to others.
- **Byzantine Generals Problem (BGP):**
  – A commanding general (commander) must send an order to his n-1 lieutenants.
- **Interactive Consistency Conditions:**
  – IC1: All loyal lieutenants obey the same order.
  – IC2: If the commanding general is loyal, then every loyal lieutenant obeys the order he sends.
- Note: If General is loyal, IC2 $\Rightarrow$ IC1.
- Original problem: each general sends his value v(i) by using the above solution, with other generals acting as lieutenants.

3-General Impossibly Example

- 3 generals, 1 traitor among them.
- Two messages: Attack or Retreat
- Shaded – Traitor
- L1 sees (A,R). Who is the traitor? C or L2?
- Fig 1: L1 has to attack to satisfy IC2.
- Fig 2: L1 attacks, L2 retreats. IC1 violated.

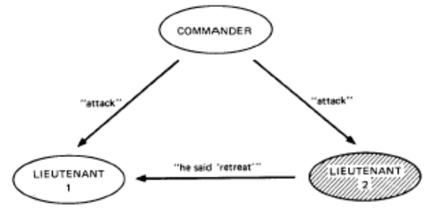


Fig. 1. Lieutenant 2 a traitor.

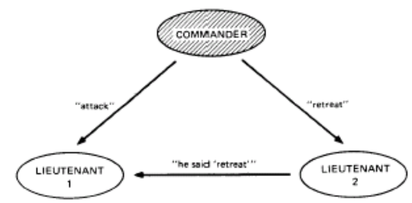
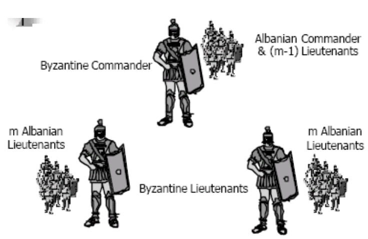


Fig. 2. The commander a traitor.

General Impossibility

- In general, no solutions with fewer than 3m+1 generals can cope with m traitors.
- Proof by contradiction.
  – Assume there is a solution for *3m Albanians* with m traitors.
  – Reduce to 3-General problem.

- Solution to 3m problem => Solution to 3-General problem!!



Solution I – Oral Messages

- If there are 3m+1 generals, solution allows up to m traitors.
- Oral messages – the sending of content is entirely under the control of sender.
- Assumptions on oral messages:
  – A1 – Each message that is sent is delivered correctly.
  – A2 – The receiver of a message knows who sent it.
  – A3 – The absence of a message can be detected.
- Assures:
  – Traitors cannot interfere with communication as third party.
  – Traitors cannot send fake messages
  – Traitors cannot interfere by being silent.
- Default order to "retreat" for silent traitor.

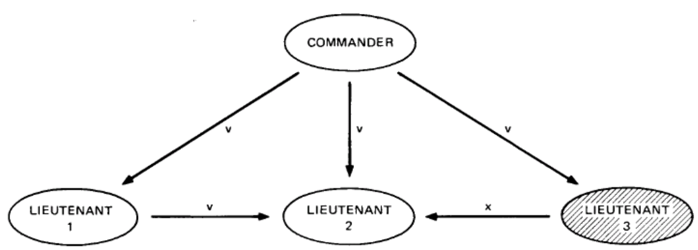
Oral Messages (Cont)

- Algorithm OM(0)
  - Commander send his value to every lieutenant.
  - Each lieutenant (L) use the value received from commander, or RETREAT if no value is received.
- Algorithm OM(m), m>0
  - Commander sends his value to every Lieutenant (v_i)
  - Each Lieutenant acts as commander for OM(m-1) and sends v_i to the other n-2 lieutenants (or RETREAT)
  - For each i, and each j ≠ i, let v_j be the value lieutenant i receives from lieutenant j in step (2) using OM(m-1). Lieutenant i uses the value majority ($v_1, \ldots, v_{n-1}$).
  - Why j ≠ i? “Trust myself more than what others said I said.”

Restate Algorithm

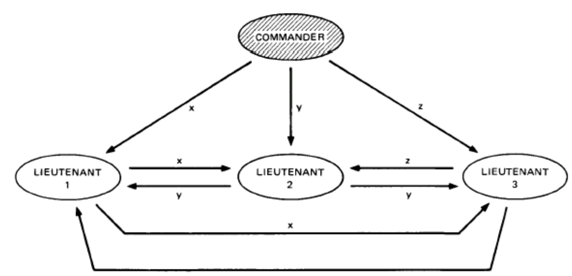
- OM(M):
  - Commander sends out command.
  - Each lieutenant acts as commander in OM(m-1). Sends out command to other lieutenants.
  - Use majority to compute value based on commands received by other lieutenants in OM(m-1)
- Revisit Interactive Consistency goals:
  - IC1: All loyal lieutenants obey the same command.
  - IC2: If the commanding general is loyal, then every loyal lieutenant obeys the command he sends.

Example (n=4, m=1)



- Algorithm OM(1): L3 is a traitor.
- L1 and L2 both receive v,v,x. (IC1 is met.)
- IC2 is met because L1 and L2 obeys C

Example (n=4, m=1)



- Algorithm OM(1): Commander is a traitor.
- All lieutenants receive x,y,z. (IC1 is met).
- IC2 is irrelevant since commander is a traitor.

Expensive Communication

- OM(m) invokes n-1 OM(m-1)
- OM(m-1) invokes n-2 OM(m-2)
- OM(m-2) invokes n-3 OM(m-3)
- …
- OM(m-k) will be called (n-1)…(n-k) times
- $O(n^m)$ – Expensive!

Solution II: Signed messages

- Previous algorithm allows a traitor to lie about the commander's orders (command). We prevent that with signatures to simplify the problem.
- By simplifying the problem, we can cope with any number of traitors as long as their maximum number (m) is known.
- Additional Assumption A4:
  - A loyal general's signature cannot be forged.
  - Anyone can verify authenticity of general's signature.
- Use a function *choice(…)* to obtain a single order
  - *choice*(V) = v if v if the only elem. in V
  - *choice*(V) = RETREAT if V is empty

Signed Messages (Cont)

- Each lieutenant maintains a set V of properly signed orders received so far.
- The commander sends a signed order to lieutenants
- A lieutenant receives an order from someone (either from commander or other lieutenants),
  - Verifies authenticity and puts it in V.
  - If there are less than m *distinct* signatures on the order
    - » Augments orders with signature
    - » Relays messages to lieutenants who have not seen the order.
- When lieutenant receives no new messages, and use choice(V) as the desired action.
- If you want to protect against more traitors, increase m

Algorithm's Intuition

- All loyal lieutenants compute the same set of V eventually, thus choice(V) is the same (IC1)
- If the commander is loyal, the algorithm works because all loyal lieutenants will have the properly signed orders by round 1 (IC2)
- What if the commander is not loyal?

V = "attack, retreat" => Commander is a traitor.

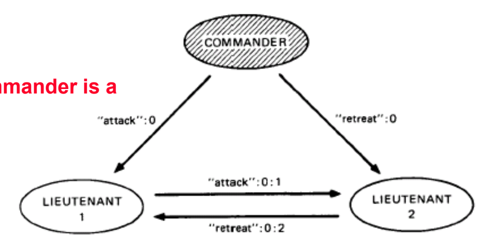


Fig. 5. Algorithm SM(1); the commander a traitor.

Missing Communication Paths

- What if not all generals can reach all other generals directly?
- P-regular graph – Each node has p regular neighbors.
- 3m-regular graph has minimum of 3m+1 nodes
- Paper shows algorithm for variant of oral message algorithm – OM(m,p). Essentially same algorithm except that each lieutenant forwards orders to neighbors.
- Proofs that OM(m,3m) solves BGP for at most m traitors.
- I.e. if the communication graph is 3m-regular, and there are at most m traitors, the problem can still be solved.

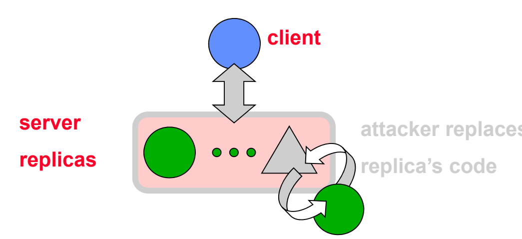
Is this a good paper?

- What were the authors' goals?
- What about the evaluation/metrics?
- Did they convince you that this was a good system/approach?
- Were there any red-flags?
- What mistakes did they make?
- Does the system/approach meet the "Test of Time" challenge?
- How would you review this paper today?

BREAK

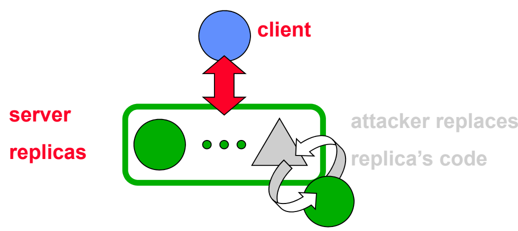
Bad Assumption: Benign Faults

- Traditional replication assumes:
  - replicas fail by stopping or omitting steps
- Invalid with malicious attacks:
  - compromised replica may behave arbitrarily
  - single fault may compromise service
  - decreased resiliency to malicious attacks



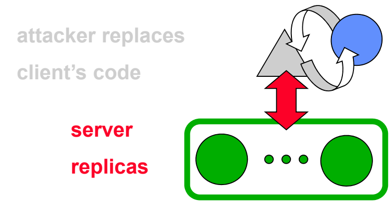
BFT Tolerates Byzantine Faults

- Byzantine fault tolerance:
  - no assumptions about faulty behavior
- Tolerates successful attacks
  - service available when hacker controls replicas



Byzantine-Faulty Clients

- Bad assumption: client faults are benign
  - clients easier to compromise than replicas
- BFT tolerates Byzantine-faulty clients:
  - access control
  - narrow interfaces
  - enforce invariants



- Support for complex service operations is important

Bad Assumption: Synchrony

- Synchrony ≡ known bounds on:
  - delays between steps
  - message delays
- Invalid with denial-of-service attacks:
  - bad replies due to increased delays
- Assumed by most Byzantine fault tolerance

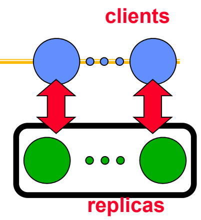
Asynchrony

- No bounds on delays
- Problem: replication is impossible

Solution in BFT:

- provide safety without synchrony
  - guarantees no bad replies
- assume eventual time bounds for liveness
  - may not reply with active denial-of-service attack
  - will reply when denial-of-service attack ends

Algorithm Properties

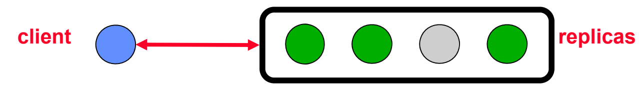


- Arbitrary replicated service
  - complex operations
  - mutable shared state
- Properties (safety and liveness):
  - system behaves as correct centralized service
  - clients eventually receive replies to requests
- Assumptions:
  - *3f+1* replicas to tolerate *f* Byzantine faults (optimal)
  - strong cryptography
  - **only for liveness:** eventual time bounds

Algorithm

State machine replication:

- deterministic replicas start in same state
- replicas execute same requests in same order
- correct replicas produce identical replies

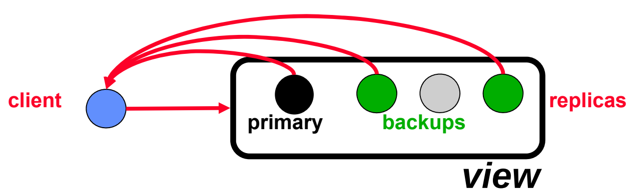


- **Hard: ensure requests execute in same order**

Ordering Requests

Primary-Backup:

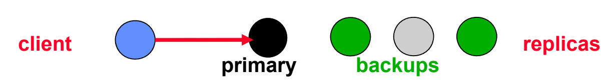
- View designates the primary replica



- Primary picks ordering
- Backups ensure primary behaves correctly
  - certify correct ordering
  - trigger view changes to replace faulty primary

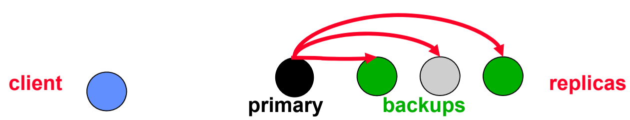
Rough Overview of Algorithm

- A client sends a request for a service to the primary



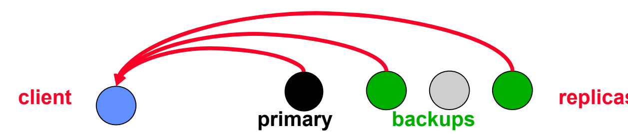
Rough Overview of Algorithm

- A client sends a request for a service to the primary
- The primary mulicasts the request to the backups



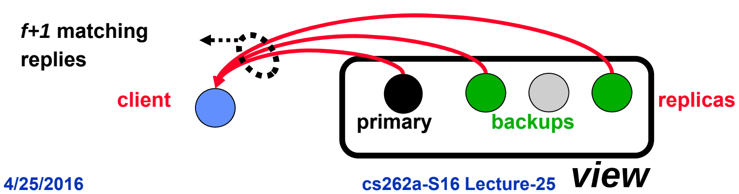
Rough Overview of Algorithm

- A client sends a request for a service to the primary
- The primary mulicasts the request to the backups
- Replicas execute request and sent a reply to the client

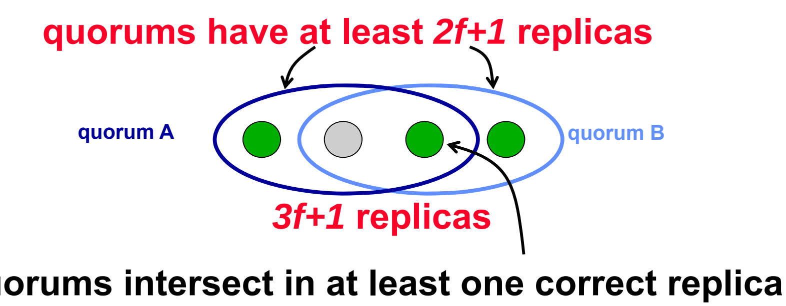


Rough Overview of Algorithm

- A client sends a request for a service to the primary
- The primary mulicasts the request to the backups
- Replicas execute request and sent a reply to the client
- The client waits for f+1 replies from different replicas with the same result; this is the result of the operation



Quorums and Certificates



- **Certificate = set with messages from a quorum**
- **Algorithm steps are justified by certificates**

Algorithm Components

- Normal case operation
- View changes
- Garbage collection
- Recovery

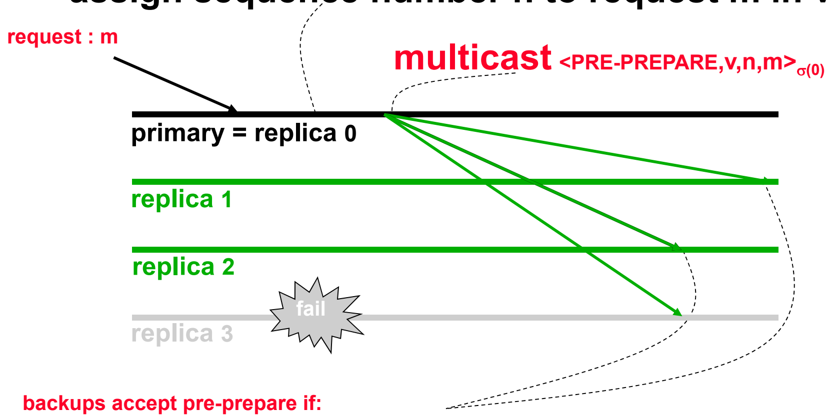
All have to be designed to work together

Normal Case Operation

- Three phase algorithm:
  - *pre-prepare* picks order of requests
  - *prepare* ensures order within views
  - *commit* ensures order across views
- Replicas remember messages in log
- Messages are authenticated
  - $\langle \bullet \rangle_{\sigma(k)}$ denotes a message sent by k

Pre-prepare Phase

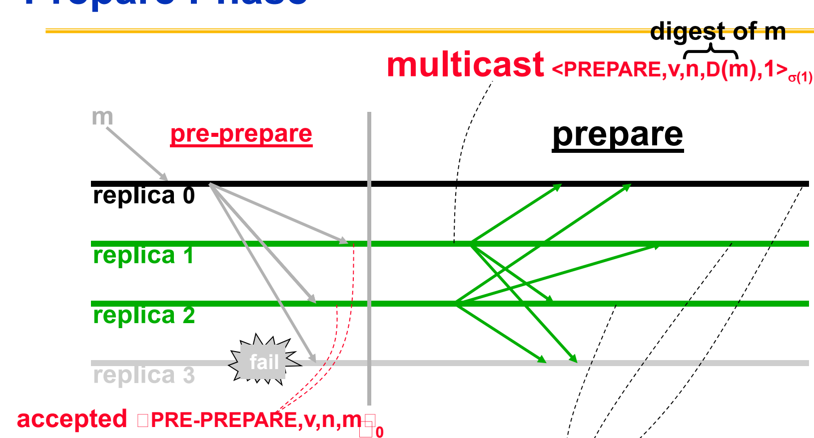
assign sequence number n to request m in view v



backups accept pre-prepare if:

- in view v
- never accepted pre-prepare for v,n with different request

Prepare Phase



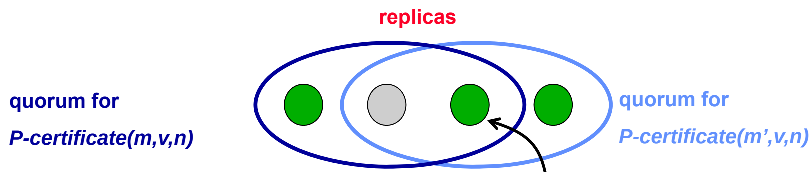
all collect pre-prepare and *2f* matching prepares

P-certificate(m,v,n)

Order Within View

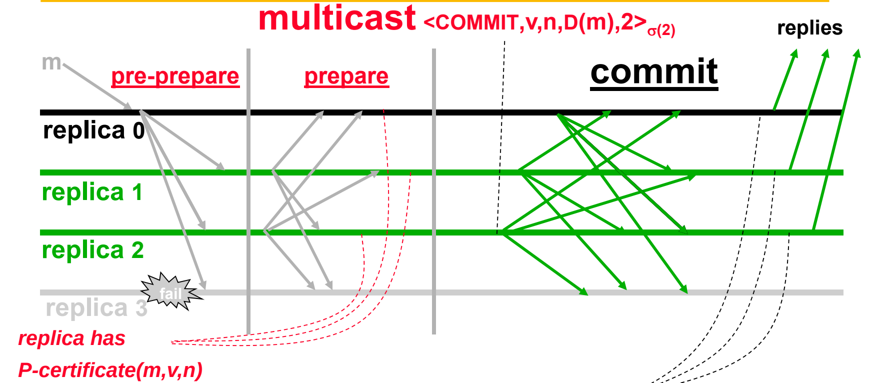
No *P-certificates* with the same view and sequence number and different requests

If it were false:



one correct replica in common ⇒ m = m'

Commit Phase



all collect 2f+1 matching commits

C-certificate(m,v,n)

Request m executed after:

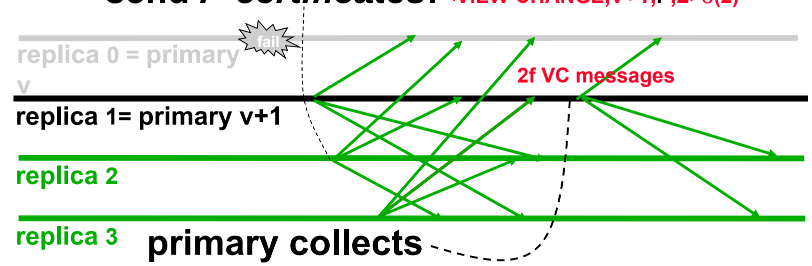
- **having *C-certificate(m,v,n)***
- **executing requests with sequence number less than n**

View Changes

- Provide liveness when primary fails:
  - timeouts trigger view changes
  - select new primary (≡ view number mod *3f+1)*
- But also need to:
  - preserve safety
  - ensure replicas are in the same view long enough
  - prevent denial-of-service attacks

View Change Protocol

send *P-certificates*: <VIEW-CHANGE,v+1,P,2>σ(2)



primary collects *VC-messages in X*: <NEW-VIEW,v+1,X,O>σ(1)

pre-prepares messages

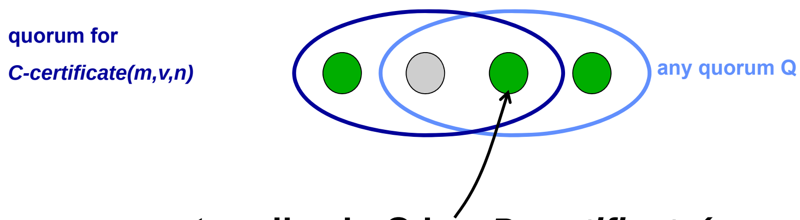
for v+1 view in O with the same sequence number

backups multicast prepare messages for pre-prepares in O

View Change Safety

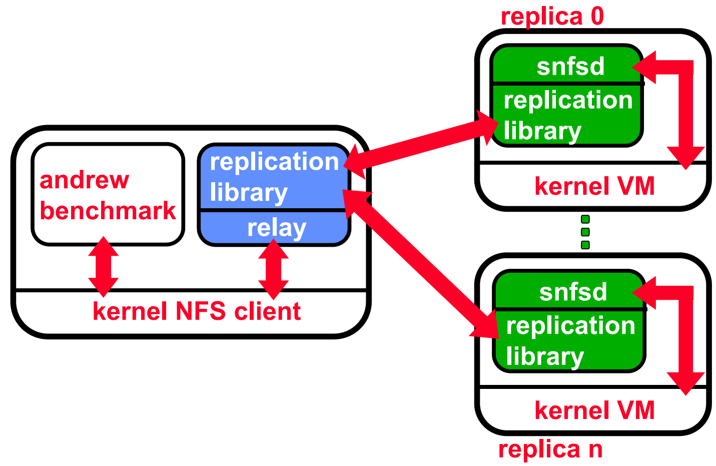
Goal: No *C-certificates* with the same sequence number and different requests

- **Intuition: if replica has *C-certificate(m,v,n)* then**



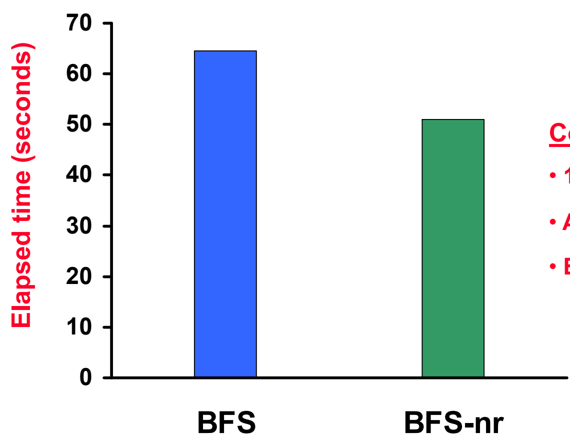
correct replica in Q has *P-certificate(m,v,n)*

BFS: A Byzantine-Fault-Tolerant NFS



No synchronous writes – stability through replication

Andrew Benchmark

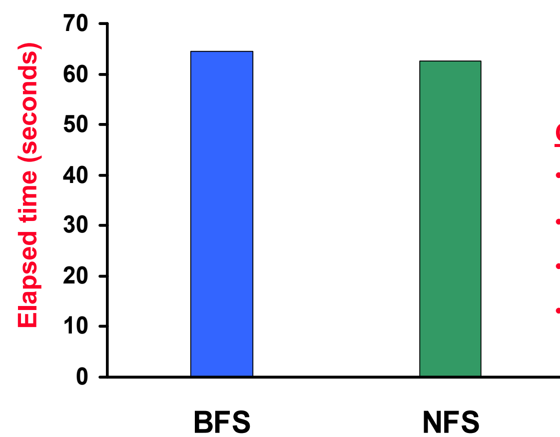


Configuration

- 1 client, 4 replicas
- Alpha 21064, 133 MHz
- Ethernet 10 Mbit/s

- **BFS-nr is exactly like BFS but without replicatio**
- **30 times worse with digital signatures**

BFS is Practical



Configuration

- 1 client, 4 replicas
- Alpha 21064, 133 MHz
- Ethernet 10 Mbit/s
- Andrew benchmark

- **NFS is the Digital Unix NFS V2 implementation**

Is this a good paper?

- What were the authors' goals?
- What about the evaluation/metrics?
- Did they convince you that this was a good system/approach?
- Were there any red-flags?
- What mistakes did they make?
- Does the system/approach meet the "Test of Time" challenge?
- How would you review this paper today?