

Byzantine Fault Tolerance in Distributed Systems

Prof. William A. Maniatty

maniatty@cs.albany.edu

<http://www.cs.albany.edu/~maniatty/teaching/os/>

Some Related Issues

Distributed Systems

- Time/Event ordering
- Synchronization
- Distributed Consensus (Voting)
- Security
 - ▷ *Cryptography*
 - ▷ *Byzantine Generals Problem*
 - ▷ *Intrusion Detection*

Mobile Computing

- Tolerating Disconnection
- Wireless and Ad Hoc Networking
- Power Management
- Security (link layer)

User Interface Design, aka Human Computer Interaction (HCI)

Embedded and Real Time Systems

Introduction to Fault Tolerance

Failures - Cause a machine to give the wrong result for some inputs

- Persistent or Intermittent
- Node Failure vs. Communication Failure
- Security intrusions can be modeled as failures

A formal model of a distributed system

- Modeled as a graph $G = (V, E)$
 - ▷ $|V| = N$, *i.e. there are N nodes.*
 - ▷ $|E| \leq \frac{N^2 - N}{2}$, *where E is the number of communication channels (links).*

A *fault tolerant* system can continue to operate properly in the presence of a reasonable number of failures.

- *Fail Stop* - Failed nodes/links shut down
- *Byzantine* - Failed links/nodes give incorrect values
- Note: undetected faults cannot be tolerated

Fault Tolerance in Distributed Systems

By definition distributed systems don't have a centralized controller

Thus distributed solution methods require reaching consensus (voting)

Distributed systems can be characterized as:.

- *Asynchronous* - Makes no assumption about timing, no time outs.
- *Synchronous* - Permits time outs

Fault Tolerance in Asynchronous Systems

Fisher, et al. proved [5] Cannot be guaranteed even under ideal conditions

- Fail stop model.
- Only one failure in N nodes

Why?

- Remember no timing assumptions allowed in Asynchronous Model
- Hence can't time out
- During a long wait for a message or is the node/link just really slow?
- However, G. Bracha and S. Toueg [1] demonstrated that probabilistic consensus is possible
 - ▷ *the probability of indefinite delay can be made negligible (have probability 0).*

Asynchronous systems are of a more theoretical interest.

- Probabilistic consensus is possible
 - ▷ *the probability of indefinite delay can be made negligible (have probability 0).*

- Adding failure detectors (so that you know if a node or link is dead) can help.
- Relaxing asynchrony (by allowing atomic operations) helps.

Byzantine Fault Tolerance in Synchronous Systems

Lamport et al. [6] defined the Byzantine Generals Problem (BGP) as:

- Consider a city under siege by N divisions of the Byzantine Army
- Each division has a General.
 - ▷ *There is one commanding general.*
 - ▷ *The commander has $N - 1$ lieutenant generals*
- Generals communicate by messengers
- Have to agree on a common strategy (or globally fail)
- What if some generals are traitors? Our goals are:
 - ▷ *All loyal generals should agree on the same strategy*
 - ▷ *A small number of traitors should not be able to trick the loyal generals into using a bad strategy.*

BGP Formalized

One possibility the commander is traitor.

This gives rise to Lamport et al's formalization using Interactive Consistency Conditions

- IC1) All loyal lieutenants obey the same order
- IC2) If the commander is loyal, all loyal lieutenants obey the order he sends.

A question

Consider a case where there is 1 traitor and 3 generals, can we guarantee a correct outcome?

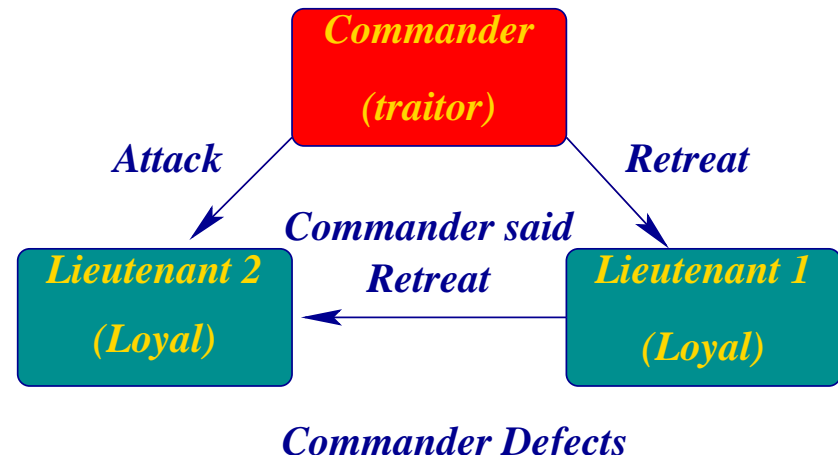
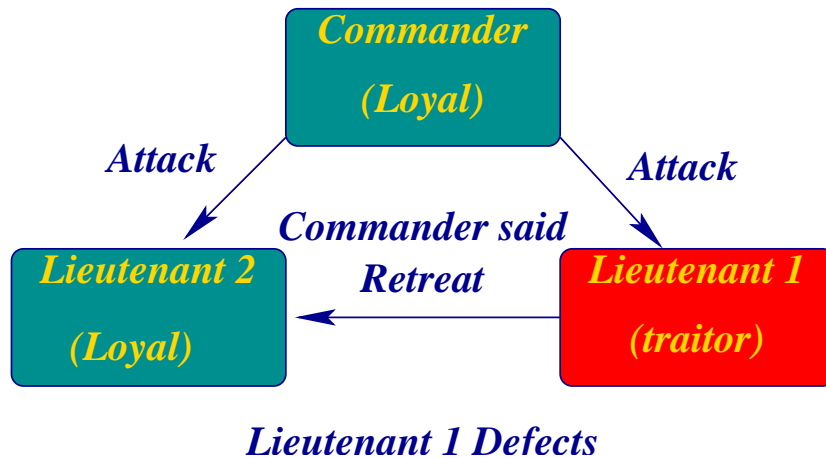
- (HINT) Lieutenants can relay the commander's order.

An Answer

Given: 1 traitor and 3 generals.

To Prove: A correct outcome is not guaranteed

The idea: Prove One Lieutenant Gets Conflicting Reports And Doesn't know what to do



Answer Details

In both cases Loyal Lieutenant 1 receives:

- Attack order directly from Commander
- Retreat order directly from Lieutenant 2

Case 1: Lieutenant 2 defects

- IC2) implies Lieutenant 1 should attack
- Suggests a (faulty) rule: Listen only to the commander

Case 2: Commander defects

- If Lieutenant 1 obeys commander he must attack
- If Lieutenant 2 obeys commander he must retreat
- But this violates IC1)

▷ *Thus, lieutenants need to listen to each other to detect a traitorous commander*

Generalizing the Result

What if we have $N > 3$ generals and $m < N$ traitors?

To distinguish this from the 3 general Byzantine General Problem we call these generals *Albanian Generals*.

In general if $N < 3m + 1$, there is no solution

- Suppose $N = 3m$
- Without loss of generality we can model this by partitioning the Albanians
 - 2 Byzantine Lieutenants, each representing m Albanian Lieutenants
 - 1 Byzantine Commander, representing 1 Albanian commander and $m - 1$ Albanian Lieutenants
- But this representation is exactly the unsolvable Byzantine Generals Problem

Approach to Conflicting Messages

So what should a node do if it gets conflicting messages?

Explode in a fiery cataclysm of doom? No...

Each node picks a ‘representative’ message value using a voting method.

- Majority
- Median value
- Mean value (for continuous values)

Picking a voting method depends on application and message type

Approximate Agreement in the BGP 1 of 2

If we have N generals and $m \geq \frac{N}{3}$ approximate agreement is impossible.

Consider a scenario with 3 Generals and one traitor where they

- Have synchronized clocks
- All loyal lieutenants must attack within 10 minutes of each other

This gives rise to modified versions of IC1) and IC2)

- IC1)' All loyal lieutenants must attack within 10 minutes of each other
- IC2)' If the commander is loyal, all loyal lieutenants must attack within 10 minutes of the time given in his order.

Approximate Agreement in the BGP 2 of 2

The commander sends a message with a time

- 1:00 means attack at 1:00
- 2:00 means retreat

Lamport Suggests Each Lieutenant does the following:

- Step 1) If the commander's message is
 - ▷ (a) *1:10 or earlier, attack*
 - ▷ (b) *1:50 or later, retreat*
 - ▷ (c) *Otherwise do step 2*
- Step 2) Ask other lieutenant what they decided
 - ▷ *If the other lieutenant decided, do the same action*
 - ▷ *Otherwise retreat*

It can be shown that this approach fails if the commander is a traitor.

Oral Message BGP

Oral messages use a reliable channel where:

- Every sent message is correctly delivered
- The receiver of a message knows who sent it
- The absence of a message can be detected

Lamport et al. developed an Oral Message Algorithm $OM(m)$, where

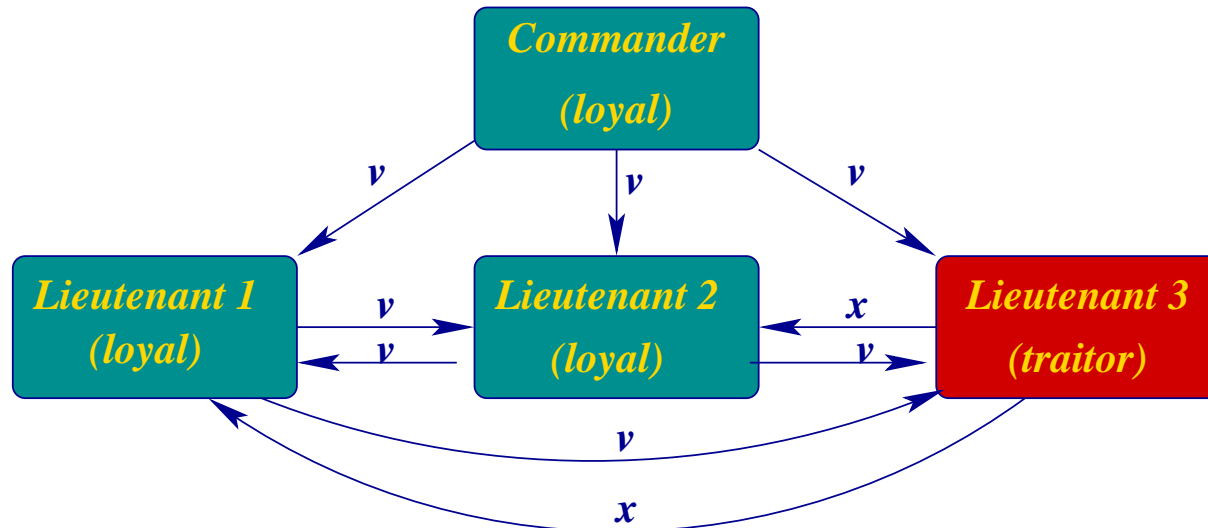
- There are N generals with
 - ▷ *1 Commander*
 - ▷ *$N - 1$ Lieutenants*
 - ▷ *m of the generals are loyal*
- Each pair of generals has a channel for oral messages
- Can't have too many traitors, requires $N \geq 3m + 1$
- Use a function to obtain representative value $\text{majority}(v_1, v_2, \dots, v_{N-1})$
 - ▷ *Can use simple majority, median for ordered sets or average for continuous values*

The Oral Message tolerating m traitors, $OM(m)$ algorithm

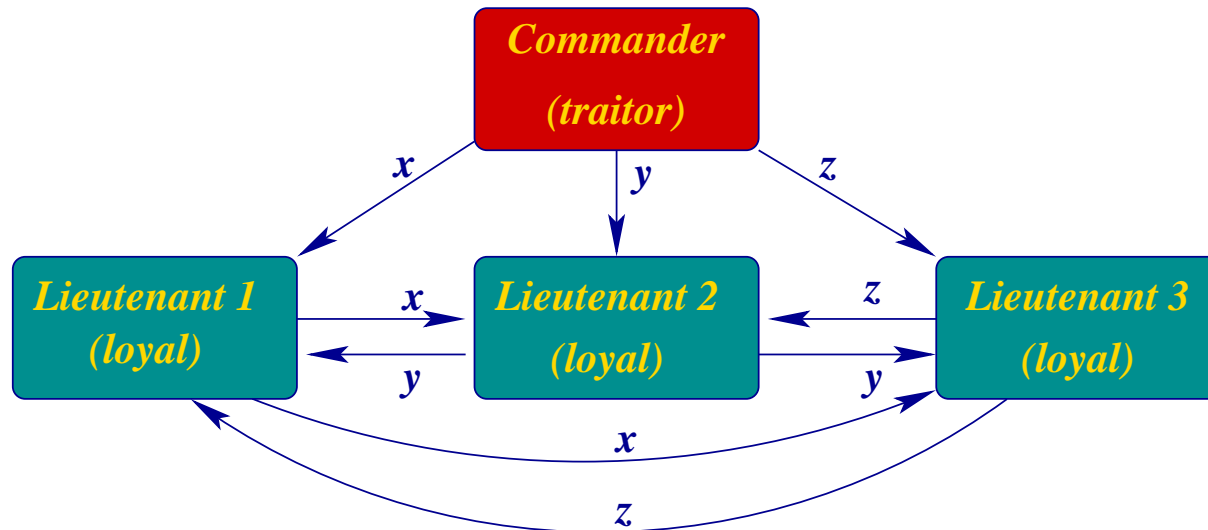
1. $OM(0)$ ($m = 0$ case, i.e. there are no traitors)
 - (a) The commander sends his value to every lieutenant
 - (b) Each lieutenant receiving a command uses the value received, if a message does not arrive, uses the value RETREAT

2. $OM(m)$ ($m > 0$ case, i.e. there are m traitors')
 - (a) The commander sends his value to every lieutenant
 - (b) For each Lieutenant $i, 1 \leq i \leq N$ let v_i be the value i receives from the commander or RETREAT if no such value was received.
In the next stage, Lieutenant i will act as a commander of the remaining $n - 2$ Lieutenants in $OM(m - 1)$ with order v_i .
 - (c) For each node i , let $j \neq i, 1 \leq j \leq N$, be some other Lieutenant. Let v_j be the value j sends to i in Step 2b (using $OM(m - 1)$) or else retreat if he receives no such value.
Lieutenant i uses $\text{majority}(v_1, v_2, \dots, v_{N-1})$.

Examples of OM(1) for $N = 4$



Lieutenant 3 Defects



Commander Defects

Remarks on Correctness of $OM(m)$

Theorem: Algorithm $OM(m)$ satisfies IC1 and IC2 if there are no more than m traitors and at least $3m$ generals (i.e. $n > 3m$).

Proof by induction on m .

- Base Case: $m = 0$ means there are no traitors, so $OM(0)$ satisfies IC1 and IC2.
- Induction Step: Show that theorem holds for $OM(m)$ case if the theorem holds for $OM(m - 1)$ where $m > 0$.
- Case 1: The Commander is loyal.
 - ▷ *Lemma: For any m and k , $OM(0)$ satisfies IC2 if there are at least $2k + m$ generals and no more than k traitors.*
 - ▷ *If $k = m$ then $OM(m)$ satisfies IC2 and since the commander is loyal IC1 holds.*
- Case 2: The commander is a traitor.
 - ▷ *Then there are at most $m - 1$ traitorous lieutenants and 1 traitorous commander.*
 - ▷ *From our hypothesis are $n - 1 > 3m - 1$ lieutenants, and $m - 1$ traitors. $OM(m - 1)$ on the lieutenants obeys our constraint since $n - 1 > 3m - 1 > 3(m - 1)$.*

Some Cost Measures in Distributed/Parallel Algorithms

Common measures of parallel algorithm resource efficiency are:

- Run Time - when the last processor finishes
- Number of rounds (for algorithms that synchronize on iterations).
- Number of messages transmitted
- Operations performed by a single processor
- Work = Operations per processor \times num processors.
- Memory needed (per node or global memory required).

Remarks on Cost/Complexity of OM(m)

- Time: The algorithm runs for $m + 1$ rounds.
 - Work per round is proportional to the number of messages
- Message Count: $O(N^{(m+1)})$.
 - ▷ Round 1: Commander sends $N - 1$ messages
 - ▷ Round 2: $N - 1$ lieutenants act as commanders for $N - 2$ of their peers for a total of $(N - 1)(N - 2)$ messages.
 - ▷ By induction Round k , $1 \leq k \leq m + 1$ requires

$$\prod_{i=1}^k (N - i) = (N - 1)(N - 2) \dots (N - k) \quad (1)$$

- So the total number of messages is:

$$\text{Number of Messages} = \sum_{i=1}^{m+1} \prod_{j=1}^i (N - j) = O(N^{(m+1)}) \quad (2)$$

Concluding Remarks and Alternatives

Number of rounds is inherently $m + 1$ for this class of problem

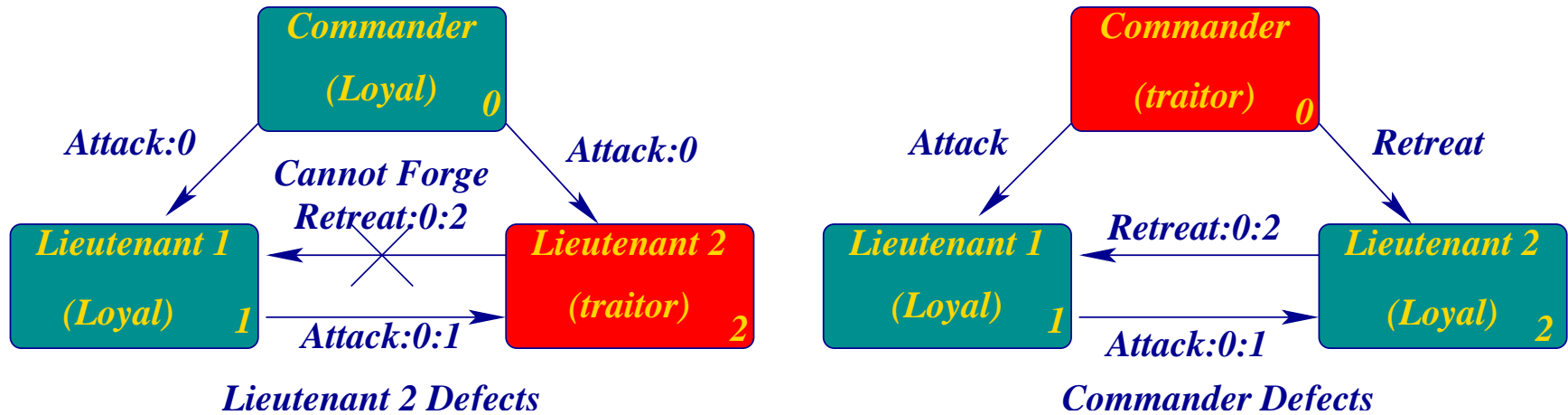
Even if the faults happen to be fail stop instead of Byzantine

Message count is large, since generals must check for altered messages

- If faults are fail stop, the message count can be reduced to (I think to $O(mN^2)$ but I'm not sure).
- Lamport et al [6] developed a *written message protocol* (assumes Byzantine Faults)
 - ▷ *The messages exchanged have tamper resistant signatures appended*
 - ▷ *Forging signatures is hard (correctly guessing has negligible probability)*
 - ▷ *Readers of messages can use the signature to detect tampering.*
 - ▷ *Increases message size*
 - ▷ *For N generals tolerates up to $m < \frac{N}{3}$ traitors.*
 - ▷ *Still takes $O(m + 1)$ rounds and $O(N^{(m+1)})$ total messages.*
 - ▷ *Can append signatures to message*
 - ▷ *In 3 general case, can now detect 1 traitor.*

- ▷ *Dolev and Strong [4] were able to reduce the number of messages to $O(N^2)$ messages by avoiding retransmitting messages that were already sent.*

Signed Messages Allow Byzantine Agreement with $N = 3m$ Generals



Castro and Liskov's Work

Castro and Liskov [2, 3] developed a Byzantine Fault Tolerance Approach.

System View

- Synchronous Distributed nodes with network connections
- Network is “unreliable”
 - ▷ *Delete messages*
 - ▷ *Corrupt Messages*
 - ▷ *Delay messages*
 - ▷ *Deliver messages out of order*
- System has n server nodes
 - ▷ *Each node implements a replica of the server.*
 - ▷ *Clients issue requests to the replicated service and block pending a reply.*

System View Continued

- Distributed server emulates a single node server
 - ▷ *Server is modeled as a deterministic state machine*
 - ▷ *One server keeps a master copy called the primary*
 - ▷ *All other servers are backups*
 - ▷ *Each nonfaulty backup node is a replica of the primary*
 - ▷ *Replicas may have different implementations of the same server.*
 - ▷ *Goal is to provide reliability and availability.*
- System Properties - Focus is on server nodes
 - ▷ *Faulty nodes may behave arbitrarily (Byzantine)*
 - ▷ *Nodes fail independently*
 - ▷ *Could use n different operating systems/server implementations*
 - ▷ *Assumes correct configuration (e.g. give each machine its own root password).*

Cryptography used in Castro and Liskov's Approach

Consider messages m from node i

- $\langle m \rangle \sigma_i$ denotes a i 's *Message Authentication Code* of m that ensures the integrity of origin
 - ▷ *Using Public Key Signatures*
- $D(M)$ is a *Message Digest* of m that ensures data integrity.
 - ▷ *Using cryptographic hash functions*
 - ▷ *Designed to ensure if $m \neq m'$ then the probability of $D(M) = D(M')$ is negligible.*

Castro and Liskov's Properties of Adversary

Adversary can coordinate faulty nodes/links

- Like zombies

Adversary can delay correctly functioning nodes/links

- Delay receipt of messages (but not indefinitely)
- Delay correct nodes (but not indefinitely).

Adversary has limited computational resources

- It is highly unlikely that they can subvert the cryptographic primitives.
- Adversaries cannot undetectably forge messages or alter their content with high probability.

Correctness Properties

These terms are used to describe correctness properties of concurrent and distributed systems.

A concurrent computation is *linearizable* if it is equivalent to a correct serial execution.

Optimally Resilient: Works for $n \leq 3f + 1$, where f is the number of faulty replicas.

Does NOT offer fault tolerant privacy/confidentiality.

Ensures Safety and Liveness (discussed later).

Safety

Informally, *safety* means that nothing “bad” ever happens during execution.

- If there are fewer than $\lfloor \frac{n-1}{3} \rfloor$ faulty servers then, the server execution is linearizable.
 - ▷ *The distributed system behaves like a centralized sequential system.*
 - ▷ *Faulty clients cannot break the server’s invariants.*
 - ▷ *The distributed system will faithfully serve any client request.*
 - ▷ *However, a misbehaving client could send a command to corrupt shared data.*
 - ▷ *The algorithm in [2] does not handle this.*
 - ▷ *Note: The sequential server would do the same.*
 - ▷ *Castro and Liskov suggest access control, and the distributed server would faithfully implement that.*

Liveness

Informally, *liveness* means that something “good” eventually happens during execution.

- Liveness relies on synchrony.
- Clients are guaranteed to receive replies to their requests if no more than $\lfloor \frac{n-1}{3} \rfloor$ faulty replicas.
- That $delay(t)$ grows slower than t where:
 - ▷ $delay(t)$ is the time between a message being sent and arriving at its destination.
 - ▷ t is the current time.

Views, Choosing a Primary and Evolving State

The algorithm is a form of state machine replication

- Each state machine replica maintains the service state and implements the service operations.

Let R be the set of replicas with f faulty replicas tolerated

- Each replica has a number assigned to it in $\{0, 1, \dots, |R| - 1\}$.
- Then $|R| \geq 3f + 1$.
 - ▷ *The number and size of messages is minimized when $|R| = 3f + 1$.*

The replicas go through a succession of configurations, called *views*

- Views can be numbered consecutively using a *viewstamp* $0, 1, \dots$

Let p be the number of the primary and the other $|R| - 1$ replicas are the backups

- Replicas take turns being the primary, $p = v \bmod |R|$.
- If the primary fails, view changes are carried out

Overview of the Algorithm

Follows treatment in [2]

1. A client sends a request to invoke a service operation to the primary
2. The primary multicasts the request to the backups
3. Each Replica executes the request and sends a reply to the client
4. The client waits for $f + 1$ replies from different replicas with the same result; this is the result of the operation.

This imposes requirements on replicas

- Replicas must be deterministic
 - ▶ *When in a given a state and given an input there must be a unique next state.*
- All replicas start in the same initial state

Given the above requirements the algorithm satisfies the safety property.

- It guarantees that all non-faulty replicas agree on a total order for the execution of requests despite failures.

Client Server Request/Reply Messages Sent

- Client c issues a request for operation o at time t by sending $\langle \text{REQUEST}, o, t, c \rangle \sigma_c$ to the primary.
 - ▷ *The timestamps used for t are totally ordered (e.g. the system clock) and ensure that the request message is processed exactly once.*
- The primary atomically multicasts this request to all backups (to be described later).
- The i th server replica replies to client c with a message $\langle \text{REPLY}, v, t, c, i, r \rangle \sigma_i$ where:
 - ▷ *v is the current view (viewstamp)*
 - ▷ *r is the result*
 - ▷ *σ_i is the i 's digital signature*
- The client c waits for $f + 1$ replies with valid digital signatures from replicas with the same t and r values before accepting r as the result.
- The client may time out, in which case it broadcasts its request to all replicas.
 - ▷ *If a replica has already processed the request, it retransmits r .*
 - ▷ *Otherwise if the replica is not the primary, it relays the request to the primary.*

Establishing Replica Consensus

Each replica maintains the following internal state

- State of the service
- Message logs all accepted messages
- The current view number

When the primary p receives a message, m it begins the 3-phase protocol:

- Pre-prepare
- Prepare
- Commit

Pre-prepare Step of 3-phase server consensus protocol (1 of 2)

- Pre-prepare is initiated by primary p in response to message m
 - ▷ p assigns a sequence number, n , to the request
 - ▷ multicasts a pre-prepare message with m piggybacked to all the backups
 - ▷ appends the message to its log.
- The message has the form $\langle \text{PRE} - \text{PREPARE}, v, n, d \rangle_{\sigma_p, m}$, where:
 - ▷ v indicates the view in which the message is being sent
 - ▷ m is the client's request message
 - ▷ d is m 's digest (cryptographic hash)..
- Pre-prepare messages are used as a proof that the request was assigned sequence number n in view v in view changes
 - ▷ so Requests are not included in the pre-prepare messages to keep them small.

Pre-prepare Step of 3-phase server consensus protocol (2 of 2)

- A backup accepts a pre-prepare message provided:
 - ▷ *the signatures in the request and the pre-prepare message are correct and d is the digest for m ;*
 - ▷ *it is in view v ;*
 - ▷ *it has not accepted a pre-prepare message for view v and sequence number n containing a different digest;*
 - ▷ *the sequence number n in the pre-prepare message is between a low water mark, h , and a high water mark, H .*
 - ▷ *Prevents a faulty client from picking a large n and exhausting the number space.*

Prepare Step of 3-phase server consensus protocol

- If backup i accepts the $\langle \text{PRE} - \text{PREPARE}, v, n, d \rangle_{\sigma_p, m}$ message, it enters the prepare phase by:
 - ▷ *multicasting a $\langle \text{PREPARE}, v, n, d, i \rangle_{\sigma_i}$ message to all other replicas*
 - ▷ *adding both messages to its log*

The *prepared* predicate

- Want to show that non-faulty replicas agree on transaction order.
- Define the predicate $prepared(m, v, n, i)$ to be true iff replica i has inserted in its log:
 - ▷ *the request m ,*
 - ▷ *a pre-prepare for m in view v with sequence number n and*
 - ▷ *$2f$ prepares from different backups that match the pre-prepare*
 - ▷ *Replicas verify this by checking that they have the same view, sequence number and digest.*

Total Ordering of Transactions Guaranteed

- The pre-prepare and prepare phases guarantee that non-faulty replicas agree on a total order for the requests in a view.
 - ▷ *They ensure the invariant: if $\text{prepared}(m, v, n, i)$ is true then $\text{prepared}(m', v, n, j)$ is false for any non-faulty replica j (including $i = j$) and any m' such that $D(m') \neq D(m)$.*
 - ▷ *Holds true because:*
 - ▷ *$\text{prepared}(m, v, n, i)$ and $|R| = 3f + 1$ imply that at least $f + 1$ non-faulty replicas have sent a pre-prepare or prepare for m in view v with sequence number n .*
 - ▷ *Thus, for $\text{prepared}(m', v, n, j)$ to be true at least one of these replicas needs to have sent two conflicting prepares (or pre-prepares if it is the primary for v ,*
 - ▷ *But we are given replica j is not faulty.*
 - ▷ *As per our digest assumption, the probability of digests matching for different messages is negligible.*

The Commit Phase (1 of 2)

- Replica i multicasts a $\langle \text{COMMIT}, v, n, D(m), i \rangle \sigma_i$ to the other replicas when $\text{prepared}(m, v, n, i)$ becomes true.
 - ▷ *This starts the commit phase.*
- Replicas accept commit messages and insert them in their log provided:
 - ▷ *They are properly signed,*
 - ▷ *The view number in the message matches the replica's current view,*
 - ▷ *The sequence number is between h and H .*
- Define the following predicates and notation:
 - ▷ *$\text{committed}(m, v, n)$ is true iff for all i in a set of $f + 1$ non-faulty replicas $\text{prepared}(m, v, n, i)$ holds.*
 - ▷ *A commit matches a pre-prepare if they have the same view, sequence number, and digest*
 - ▷ *$\text{committed} - \text{local}(m, v, n, i)$ is true iff $\text{prepared}(m, v, n, i)$ is true and i has accepted $2f + 1$ commits (possibly including its own) from different replicas that match the pre-prepare for m .*

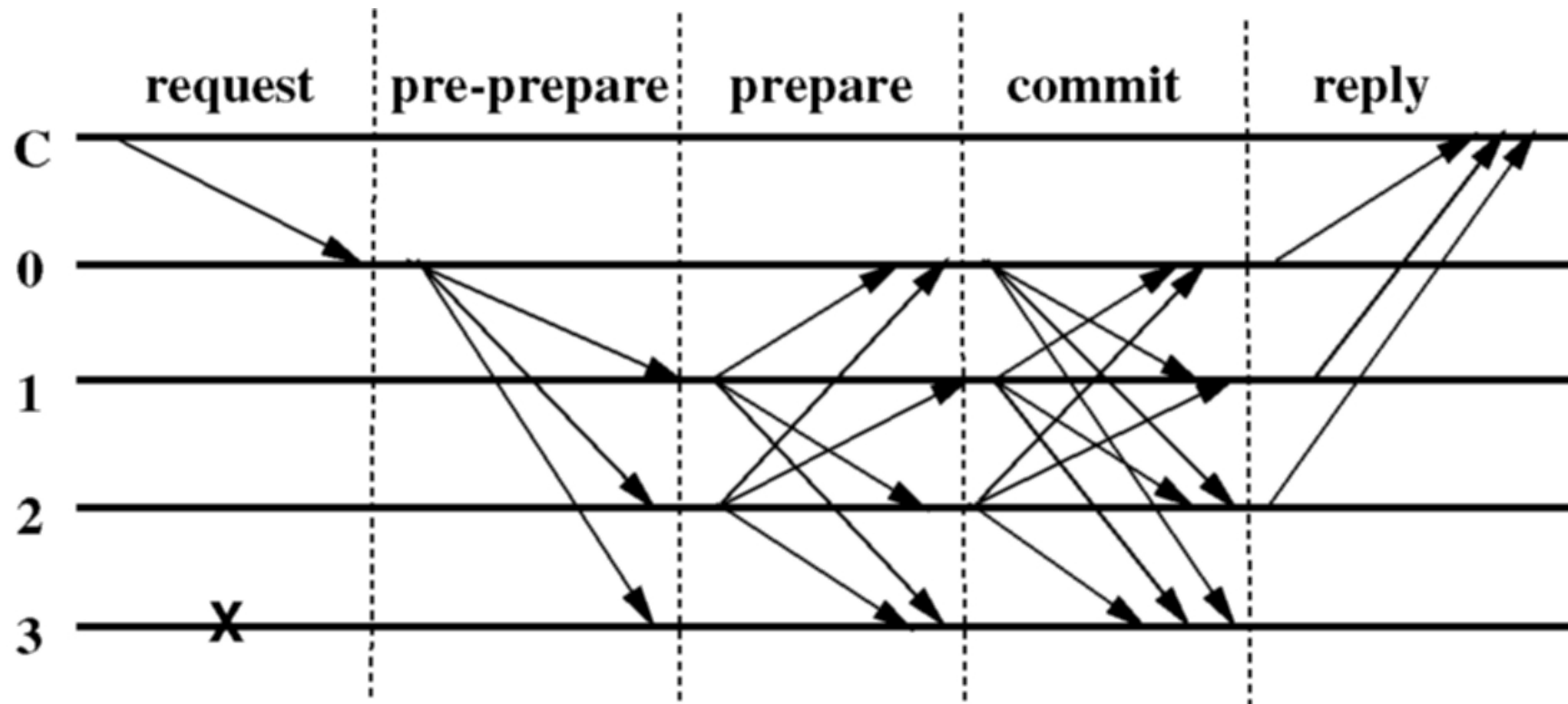
The Commit Phase (2 of 2)

- Each replica i executes the operation requested by m after:
 - ▷ *committed* – $local(m, v, n, i)$ is true and
 - ▷ *i 's state reflects the sequential execution of all requests with lower sequence numbers.*
- After executing the requested operation, replicas send a reply to the client.
 - ▷ *Replicas discard requests with timestamps older than the last reply they sent to ensure exactly once semantics.*
 - ▷ *To reduce the amount of data sent the client designates on replica to send the result in its reply, the others send just the digests.*

Properties of the Commit Phase

- The commit phase ensures the invariant: if $committed - local(m, v, n, i)$ is true for some non-faulty i then $committed(m, v, n)$ is true
- Commit invariant and view-change protocol ensure that:
 - ▷ *Non-faulty replicas agree on sequence numbers, even if they commit in different views.*
 - ▷ *Any request that commits locally at a non-faulty replica will commit at $f + 1$ or more non-faulty replicas eventually.*
- Execution after $committed - local(m, v, n, i)$:
 - ▷ *Reflects sequential execution of requests with lower sequence numbers.*
 - ▷ *Ensures all non-faulty replicas execute requests in the same order --- provides safety*

Examples of normal operation for $f = 1, |R| = 4$



Garbage Collection

When is it safe to discard old state (messages)?

- When the node knows it is safe and can prove it to all other nodes.
- Uses distributed checkpointing.
- Proof generation is expensive.
- Manages the jumping advancement of the window $H = h + k$.
- Notation:
 - ▷ *states produced by the execution of requests are checkpoints*
 - ▷ *A checkpoint with a proof is a stable checkpoint.*
- Generate expensive proofs periodically to amortize out the cost.

Proofs of Checkpoint Correctness

How is a proof of a checkpoint's correctness generated?

- When a replica i produces a checkpoint, it multicasts a message $\langle \text{CHECKPOINT}, n, d, i \rangle \sigma_i$ to the other replicas, where:
 - ▷ n is the sequence number of the last request whose execution is reflected in the state and
 - ▷ d is the digest of the state
- Each replica collects checkpoint messages in its log until it has $2f + 1$ of them for sequence number n with the same digest d signed by different replicas.
 - ▷ including possibly its own such message
- These $2f + 1$ messages are the proof of correctness for the checkpoint.

View Change Protocol Overview/Background

View change ensures liveness by allowing progress when the primary fails.

- View changes are triggered by timeouts at backups
 - ▷ *Prevents indefinite waiting*
 - ▷ *A backup is waiting for a request if*
 - ▷ *the backup received a valid request and*
 - ▷ *it has not executed the request.*
- Timer Management
 - ▷ *A backup starts a timer when it receives a request and the timer is not already running.*
 - ▷ *It stops the timer when it is no longer waiting to execute the request, but restarts it if at that point it is waiting to execute some other request.*

View Change Protocol (1 of)

- If the timer of backup i expires in view v , the backup starts a view change to move the system to view $v + 1$.
 - ▷ *It stops accepting messages other than checkpoint, view-change, and new-view messages and*
 - ▷ *multicasts a $\langle \text{VIEW} - \text{CHANGE}, v + 1, n, C, P, i \rangle_{\sigma_i}$ message to all replicas, where:*
 - ▷ *n is the sequence number of the last stable checkpoint s known to i ,*
 - ▷ *C is a set of $2f + 1$ valid checkpoint messages proving the correctness of s , and*
 - ▷ *P is a set containing a set P_m for each request m that prepared at i with a sequence number higher than n .*
 - ▷ *Each set P_m contains:*
 - ▷ *a valid pre-prepare message (without the corresponding client message) and*
 - ▷ *$2f$ matching, valid prepare messages signed by different backups with the same view, sequence number, and digest of m .*

View Change Protocol (2 of)

- ▶ When the primary p of view $v + 1$ receives $2f$ valid view-change messages for view $v + 1$ from other replicas it multicasts a $\langle \text{NEW} - \text{VIEW}, v + 1, V, O \rangle_{\sigma_p}$ message to all other replicas, where:
 - ▶ V is a set containing the valid view-change messages received by the primary plus the view-change message for $v + 1$ the primary sent (or would have sent), and
 - ▶ O is a set of pre-prepare messages (without the piggybacked request). O is computed as follows:
 1. The primary determines the sequence number $\text{min} - s$ of the latest stable checkpoint in V and the highest sequence number $\text{max} - s$ in a prepare message in V .
 2. The primary creates a new pre-prepare message for view $v + 1$ for each n , where $\text{min} - s \leq n \leq \text{max} - s$. There are two cases:
 - (a) There is some set(s) in the P component of some view-change message in V with sequence number n
 - ▶ the primary creates a new message $\langle \text{PRE} - \text{PREPARE}, v + 1, n, d \rangle_{\sigma_p}$, where: d is the request digest in the pre-prepare message for sequence number n with the highest view number in V .
 - (b) otherwise there is no such set.
 - ▶ it creates a new pre-prepare message $\langle \text{PRE} - \text{PREPARE}, v + 1, n, d^{\text{null}} \rangle_{\sigma_p}$, where d^{null} is the digest of a special null request
 - ▶ a null request when is processed as a no-op.

View Change Protocol (3 of)

- ▶ Next the primary appends the messages in O to its log.
 - ▶ If $min - s$ is greater than the sequence number of its latest stable checkpoint, the primary also inserts the proof of stability for the checkpoint with sequence number $min - s$ in its log, and discards information older than the stable checkpoint.
- ▶ Then it enters view $v + 1$: at this point it is able to accept messages for view $v + 1$.
- ▶ A backup accepts a new-view message for view $v + 1$ if
 - ▶ it is signed properly,
 - ▶ the view-change messages it contains are valid for view $v + 1$, and
 - ▶ the set O is correct
 - ▶ it verifies the correctness of O by a computation similar to the way the primary created O .
- ▶ Then it
 - ▶ adds the new information to its log as described for the primary,
 - ▶ multicasts a prepare for each message in O to all the other replicas,
 - ▶ adds these prepares to its log, and enters view $v + 1$.
- ▶ Now it resumes Normal Case operation (runs the consensus protocol)

- ▷ *If a replica is missing a request message or a stable checkpoint, it will need to get a copy of the state from another replica.*
 - ▷ *The replica can be selected if the checkpoint messages are certified for correctness.*
 - ▷ *Copy on write can be used so that only what got different gets sent.*

Review and Conclusions

Review some Background material

- Lampson's paper on your own.
- Byzantine Generals Problem/Distributed Fault Tolerance
- Castro and Liskov's Model for Distributed Implementation of Servers

Conclusions on Fault Tolerance

- Byzantine Generals Problem is a very strong result
- However, reaching consensus is expensive
- Especially for large systems
- Or systems with expensive data communication
- But some applications need it

Bibliography

References

- [1] G. Bracha and S. Toueg. Asynchronous consensus and broadcast protocols. *J. ACM*, pages 824--840, October 1985.
- [2] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Third Symposium on Operating Systems Design and Implementation (OSDI)*, New Orleans, Louisiana, February 1999. USENIX Association, Co-sponsored by IEEE TCOS and ACM SIGOPS.
- [3] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398--461, November 2002.
- [4] D. Dolev and H. R. Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12:656--666, 1983.

- [5] M. J. Fisher, N. A. Lynch, and M. S Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374--382, April 1985.
- [6] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382--401, July 1982. Republished in *Advances in Ultra-Dependable Distributed Systems*, 1995, N. Suri, C. J. Walter, and M. M. Hugue (Eds.).