Byzantine fault tolerance
Srivatsan ravi
Byzantine Generals and fault tolerance

- Overview and definition
- Naive solutions
- Solution with oral messages
- Solution with signed messages
- Communication paths
- Practical considerations
Motivation

- Coping with failures in distributed systems
- Failed component sends conflicting information
- No apriori assumption on behavior of faulty components
- Need for agreement in the presence of faults
Problem definition

- Each division of the Byzantine army is directed by its own General (computer components)
- There are n Generals some of whom are traitors
- Communicate with each other by messengers
- Unanimous agreement to ATTACK
Byzantine Generals and fault tolerance

- Overview and definition
- *Naive solutions*
  - Solution with oral messages
  - Solution with signed messages
  - Communication paths
- Practical considerations
Naive solution

- $G(i)$ sends $v(i)$ to all other $G$
- All $G$ combine their information $v(1), v(2), \ldots, v(n)$ the same way
- Majority ($v(1) \ldots v(n)$) agree on ATTACK, else RETREAT
- Ignore minority traitors
Naive solution does not work!!

- Traitors may send different values to different G
- Loyal G may get conflicting values from traitors
- Any two loyal generals must use the same value of v(i) to decide on the same plan of action
- *Reduce the problem to General sending his orders to (n-1) other Lieutenants*
Consistency

- Interactive consistency1: All loyal lieutenants obey the same order
- Interactive consistency2: If G is loyal, then each L(i) obeys the order i.e IC1 implies IC2
3-General impossibility

- 3 Generals, one traitor among them
- Two messages: Attack or Retreat
- IMPOSSIBLE to satisfy both IC1 and IC2

Fig. 1. Lieutenant 2 a traitor.

Fig. 2. The commander a traitor.
n-General Impossibility

- Theorem 1: No solution with fewer than $3n+1$ generals can cope with $n$ traitors
- Proof: By contradiction, assume there is a solution for a group of $3n$ or fewer and use it to construct a 3-G solution to BGP that works with one traitor
- But this is impossible. Hence the initial assumption was wrong
- Q.E.D!
Byzantine Generals and fault tolerance

- Overview and definition
- Naive solutions
- **Solution with oral messages**
- Solution with signed messages
- Communication paths
- Practical considerations
Solution with Oral messages

- Sending of content is entirely under the control of sender
- Each message sent is delivered correctly (A1)
- Receiver of message knows who sent it (A2)
- Absence of message can be detected (A3)
OM(m=0)

- Commander sends his value to every lieutenant.
- Each lieutenant (L) uses the value received from commander, or RETREAT if no value is received.

Algorithm OM(m), m>0

- Commander sends his value to every lieutenant.
- 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 not equals 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), v(2), ..., v(n-1)).
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.
Complexity

- 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
- OM(m) invokes n-1 OM(m-1)
- O(n^m) – algorithm grows exponentially to number of failures-Expensive!
Byzantine Generals and fault tolerance

- Overview and definition
- Naive solutions
- Solution with oral messages
- Solution with signed messages
- Communication paths
- Practical considerations
Solution II: Signed messages

- 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
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.

Use choice(V) as the desired action.
Example-Signed messages

Fig. 5. Algorithm SM(1); the commander a traitor.
Byzantine Generals and fault tolerance

- Overview and definition
- Naive solutions
- Solution with oral messages
- Solution with signed messages
- Communication paths
- Practical considerations
Missing communication paths

- Nature of processor graph
- Strong connectivity: The graph be 3m-regular
- Strong connectivity is impractical
- Weakest connectivity requirement: Subgraph formed by the loyal generals be connected
What does it take for majority voting to work?

- Non-faulty processors to produce same outputs (IC1)
- If input unit (commander) is non-faulty, all non-faulty processes use the value it provides as input (IC2)

A1 – Every message sent by non-faulty process is delivered correctly.

- Failure of communication line cannot be distinguished from failure of nodes.
- OK because we still are tolerating m failures.

A2 – Processor can determine origin of message

- Use of signatures (A4)
A3 – Absence of a message can be detected.
  - Timeouts
  - Synchronized clocks
A4 – Unforgeable signatures.
  - Anyone can verify Sig
Concluding thoughts

- BGP solutions are expensive (communication overheads and signatures)
- Use of redundancy and voting to achieve reliability.
- What if >1/3 nodes (processors) are faulty?
- 3m+1 replicas for m failures. Is that expensive?
- Tradeoffs between reliability and performance
- Nature of processor graph
Practical Byzantine fault tolerance

- Miguel Castro and Barbara Liskov
Byzantine research

- 1980- Reaching agreement in presence of faults- Lamport, Pease, Shostak
- 1982- Byzantine Generals problem- Lamport et.al
- 1983- Byzantine Generals strike again- Dolev
- 1983- Randomized Byzantine generals- Rabin
- 1988- ViewStamped replication- Liskov et.al
Byzantine research-1990's

- 1992-Optimal async Byzantine agreement-Canneti
- 1998-SecureRing protocol for group comm-Kihlstrom et.al
- 1998-Byzantine quorum systems-Dahlia et.al
- 1999-Practical Byzantine fault tolerance-Liskov and Castro!
The problem

- Provide a reliable answer to a computation even in the presence of Byzantine faults.
- A client would like to
  - Transmit a request
  - Wait for $k$ replies
  - Conclude that the answer is a true answer
Failures of previous algorithms

- Theoretically feasible but inefficient in practice
- Assumes synchrony – known bounds of message delays and process speeds
- Synchrony assumption for correctness
- Can we do better?
The Model

- Networks are unreliable
  - Can delay, reorder, drop, retransmit
- Some fraction of nodes are unreliable
  - May behave in any way, and need not follow the protocol.
- Nodes can verify the authenticity of messages
- *Message delay does not grow exponentially*
FLP impossibility?

- Strong adversary can delay correct nodes in order to cause most damage to the replicated service
- Assume adversary cannot delay nodes indefinitely
- Rely on synchrony to provide liveness
- Does not rely on synchrony to guarantee safety
- Otherwise it could be used to implement consensus in async setting
- NOT POSSIBLE!
Protocol overview

- Form of Lamport & Schneider state machine replication
- Service modelled as a state machine replicated across nodes
- Replicas maintains service state
- Cryptography to detect message corruption
Views

- Replicas move through a succession of configurations called views
- View = Primary + Backup nodes
- Primary = v mod n
  - N is number of nodes
  - V is the view number
Nodes

- Maintain a state
  - Log
  - View number
  - state

- Can perform a set of operations
  - Need not be simple read/write
  - Must be deterministic

- Well behaved nodes must:
  - start at the same state
  - Execute requests in the same order
Replica requirements

- Deterministic replicas
- All replicas start in same state
- Safety: Agree on total order
- Primary picks ordering
- Backups ensure primary behaves correct
- Trigger view changes
Why doesn’t traditional RSM work with Byzantine nodes?

- Cannot rely on the primary to assign seq-no
  - Malicious primary can assign the same seq-no to different requests!
- Cannot use Paxos for view change
  - Paxos uses a majority accept-quorum to tolerate $f$ benign faults out of $2f+1$ nodes
  - Bad node tells different things to different quorums!
Basic Algorithm

- Three-phase protocol to multicast requests to replicas
  - Pre-prepare and prepare order within views
  - Prepare and commit order across views
- Messages are authenticated
- Replicas remember messages - maintain log
Normal Case operation

- Primary receives request and starts a 3-phase protocol
- Pre-prepare: Accept requests only if valid
- Prepare: Multicasts prepare messages
- Wait for $2f+1$ replicas to agree
- Commit: Commit if $2f+1$ agree to commit
1. A client sends a request to invoke a service operation to the primary

\[ \langle \text{REQUEST}, o, t, c, s \rangle_{o, t, c} \]

- o = requested operation
- t = timestamp
- c = client
- s = signature
2. The primary multicasts the request to the backups (three-phase protocol)
3. Replicas execute the request and send a reply to the client.

\[
\{\text{REPLY}, v, t, c, i, r\}_{\sigma_i}
\]

- \(o\) = requested operation
- \(v\) = view
- \(t\) = timestamp
- \(c\) = client
- \(i\) = replica
- \(r\) = result
- \(\sigma_i\) = signature
4. The client waits for \( f+1 \) replies from different replicas with the same result; this is the result of the operation.
Improvements in this algorithm

- Does not rely on synchrony of safety
- Magnitude order improvement
- Efficient authentication using message authentication codes (MAC)
- Public key cryptography
- Handling malicious primary
Byzantine-Fault-tolerant File System

- BFS is implemented using replication library
- Replicas
- User-level relay processes mediate communication between the standard NFS client and the replicas.
- Relay receives NFS requests, invokes procedure of replication library and sends the result back to NFS client.
- The performance of BFS is only 3% worse than the standard NFS implementation.
Conclusion

- Able to tolerate Byzantine failures
- Works in an asynchronous system like Internet
- Better lower bounds than previous known algorithms
- No assumptions on synchrony for safety
- Can we reduce the number of replicas used?
- Fault tolerant privacy-faulty replica may leak information to attacker
- Zyzyva: Speculative Byzantine Fault Tolerance-SOSP07