Practical Byzantine Fault Tolerance

CS 249: Distributed Computing
Team 2: Rashmeet Khanuja, Anusha Vijay, Steven Yen

Introduction

- As reliance on internet information and services grow, so does the consequence of malicious attacks
- Increase in bugs as systems scale
- These attacks and errors end up causing faulty nodes, which exhibit arbitrary behaviour.
- A unique state machine replication protocol in an asynchronous model that survives Byzantine Faults
- Ways to optimize the algorithm so it can work in real world systems.

Why previous algorithms were not “Practical”

- Previous algorithm
  - Demonstrated techniques with theoretical feasibility , i.e they were too inefficient to be implemented for practical systems
  - Assumed synchronous behaviour i.e relied on known upper bounds of message delays and processing speed
- Why this did not work?
  - A simple DOS would change a non faulty node to a faulty node by just delaying the process in the node its communication
  - Excluding it from the replica group

What’s “Practical” about this?

- Describes first State machine replication protocol which survives Byzantine faults in **asynchronous networks.**
- Optimized to perform well when implemented in real world systems
  - Improved response times : One message RT to execute read-only and 2 for read-write
  - Safety and liveness for at most (n-1)/2 of n faulty nodes.

System Model

- Asynchronous distributed system, nodes are interconnected by the network
- Faulty nodes have arbitrary behaviour
- Independent Node failures
- Cryptographic techniques to prevent spoofing,replays, detect corrupt messages
- Public Key signatures, authentication code
    - All replica know each other’s public keys to verify signature
- Adversary is bound by
    - computation capabilities
    - Cannot indefinitely delay non faulty nodes

Service Properties

- Implement any deterministic replicated services on n nodes
- Each replicated node has a state and any deterministic operations
- Provides safety and liveness for at most [n-1]/3 faulty nodes
  - Safety: Rep Services satisfies Linearizability
  - Liveliness: Clients eventually receive a reply to their request
- Access Control is used to limit the damage of faulty clients
- Optimal resiliency is 3f+1 beyond which leads to performance degradation
- The algorithm does not address the problem of fault tolerant privacy: a faulty replica may leak information to an attacker.

The Algorithm

- Form of Statement Machine replication algorithm
- Service is modeled as a state machine
- The state machine is replicated across different nodes of the system, and maintains maintains the service state and implements the service operations.
- R => set of replicas, each replica is identified by using an integer in {0,..., |R|-1}
    - Assumption => |R| = 3f + 1 ; where f is the maximum number of replicas that may be faulty

The Algorithm

- The replicas move through a succession of views.
    - A view is a configuration, where one replica is the primary and the others are backups.
        - Primary replica ‘p’ is such that p = v mod|R| ; where v is the view number
    - Views are numbered consecutively
    - When the primary replica fails, view changes are carried out

The Algorithm

- Requirements imposed on replicas:
    - Must be deterministic (i.e., the execution of an operation in a given state and with a given set of arguments must always produce the same result)
    - They must start in the same state

The Algorithm - Client

- The client
  1. A client c requests the execution of state machine operation o by sending a <REQUEST, o, t, c> to the current primary
  2. The primary atomically multicasts the request to all the backups (the other replicas)
  3. Replica processes the request and sends the reply to the request in the form <REPLY,v,t,c,i,r> directly to the client, where v = current view number, t = timestamp of the request, i = replica number, r = result of running the requested operation
  4. After receiving f+1 replies (where f = max number of faulty replicas) with valid signatures from different replicas with the same r and t values, the client will accept the result r.

Phase 1: Pre-Prepare Phase

- Each state ‘o’ of replica includes
  - State of service,message log and id<int> of current view
- Pre-prepare with prepare phase orders request sent in the same view
- Primary assigns sequence no n (h<n<H) to request m. Multicasts i to all backup. $\langle\langle \text{PRE-PREPARE}, v, n, d\rangle_{\sigma_p}, m\rangle$,
  - Primary logs are updated with sending pre-prepared event
- Backup accepts a pre-prepared message if
  - Signature and d in request and PP message are correct
  - V is local
  - Correct sequence no
  - No pp message for v,n with different d
- Once accepted, enters “PREPARED” phase both messages are logged
  -

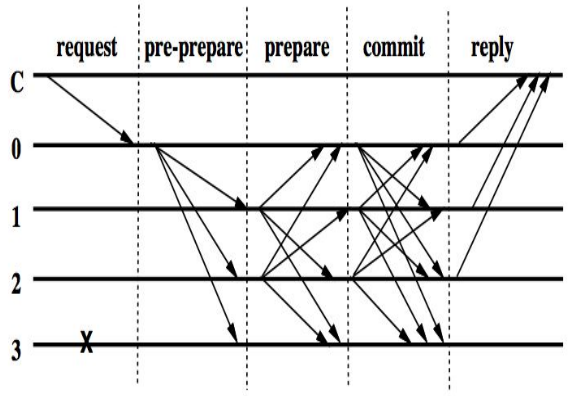

Figure 1: Normal Case Operation

Phase 2: Prepare Phase

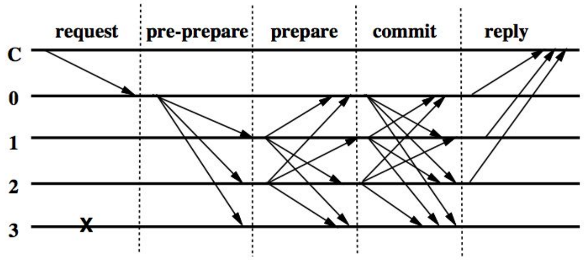

Figure 1: Normal Case Operation

- Multicasts $\langle \text{PREPARE}, v, n, d, i \rangle_{\sigma_i}$ ll other replicas (0,2,3)
- Accepted and added to log after verifying signature, v and n
- Prepared= True IFF
  - A request
  - A pre-prepared with v, n
  - At least 2f prepares from DIFFERENT backups for each pre-prepare (same v,n and d)
- **Ensures non-faulty replicas agree on the total order of request within the view**

Garbage Collection

- To meet the safety condition, messages need to be in each replica's log until it knows the request has been served by f+1 non-faulty nodes
    - Need to prove this to others in view changes
- Replicas need proof of correctness of state
- Proofs of correctness (expensive affairs, done periodically)
    - When a replica produces a checkpoint, it multicasts a message to other replicas including the sequence number of the last request whose execution is reflected in the state and a digest
    - Each replica collects checkpoint messages in its log until it has 2f + 1 of them for sequence number with the same digest signed by different replicas
    - Stable Checkpoints - checkpoint with a proof
- Checkpoint protocol - used to advance the low and high water marks (which limit what messages will be accepted)
    - Low watermark 'h' - sequence number of the last stable checkpoint
    - High watermark 'H' = h + k where k is big enough so that replicas do not stall waiting for a checkpoint to become stable

Implementation: Replication Library

- Replication Library: used as basis for replicated services
- **Client side:**
  - Consists of *invoke* procedure:
    - Argument: input buffer containing request to invoke state machine operation
    - Uses a protocol that causes requested operation to be executed on replicas
    - Select correct reply among replies from individuals replicas
    - Returns a pointer to a buffer containing the result of the operation
- **Server side:**
  - Replication code calls procedures that part of the application the server must implement:
    - Execute requests (*execute*): execute procedure in input buffer, put result in output buffer
    - Maintain checkpoints of service state (*make_checkpoint, delete_checkpoint)*
    - Obtain digests of a checkpoint (*get_digest*)
    - Obtain missing info (*get_checkpoint*, *set_checkpoint*)

Implementation: BFS

- Byzantine-Fault-tolerant File System (BFS): NFS using the replication library
- Kept the regular NFS client and server in the kernel
- Application processes interact with mounted file system using the NFS client in the kernel
- User level relay processes mediate the communication between NFS client and replicas:
    - It receives NFS requests
    - Calls *invoke*
    - Sends the result back to NFS client
- Ensure replicas start in same initial state & deterministic

Conclusion

- A new state-machine replication algorithm that is able to tolerate Byzantine faults and can be used in practice has been described
  - The first to work correctly in an asynchronous system
- Also described BFS, a Byzantine-fault tolerant implementation of NFS
  - The performance of BFS is only 3% worse than that of the standard NFS implementation
- What contributes to the good performance - replacing public-key signatures by vectors of message authentication codes, reducing the size and number of messages, and the incremental checkpoint-management techniques.
- The algo cannot mask a software error that occurs at all replicas
  - Can mask errors that occur independently at different replicas, including nondeterministic software errors, which are the most problematic and persistent errors since they are the hardest to detect

Areas of improvement

- Reducing the amount of resources required to implement the algorithm
- Number of replicas can be reduced by using f replicas as witnesses that are involved in the protocol only when some full replica fails
- It might also be possible to reduce the number of copies of the state to f + 1

Thank You