Distributed Systems

Day 9: Replication [Part 1]
• Does your client know about all of FB’s servers?
  • Security issues?
  • Performance issues?

• How do clients send all requests to replicas?
  • Multicast?
  • Special protocol?

Partition data into shards, maps shards to server with consistent hashing

Maintain multiple copies for fault tolerance and to reduce latency

Clients send requests to all replicas

Servers use vector Clocks
Control ordering

Clocks allow “same” ordering across servers

Partition data into shards, maps shards to server with consistent hashing
Replica Transparency:

- Clients interact with a Front End server (FE)
- FE controls replica to the servers
  - FE hides replica complexity
  - FE is owned by FB → knows all FB servers
  - FE is stateless → FE crash does not lead to data loss
Partition data into shards, maps shards to server with consistent hashing

Maintain multiple copies for fault tolerance and to reduce latency

Servers use vector Clocks
Control ordering

Clocks allow “same” ordering across servers

Clients send requests to all replicas
Replication (Server Replication)

• Maintain several copies of data because
  • Recovery from failure
  • Lower latency: put data closer to client

• Issues with replication
  • Consistency: should all data be identical?
  • Availability: client have access to at least one data?
  • Performance: quick reads/writes to data?
Do We Always Need Total Ordering?

• For some applications: yes!!!
  • My back account

• For others: “causal ordering”
  • Dependency between events are preserved
  • $A \rightarrow B$: on all servers $A$ is processed before $B$.

• How do I capture and enforce causal ordering???
Data Consistency Challenges With Replication

• Ideal consistency: All replicas behave as one server
  • Linearizability → total order according to real time

• Recall, Linearizability is almost impossible
  • Total order is costly to achieve

• Practical consistency: all replicas have identical ordering on subset of events
  • Casual ordering?
  • Fifo ordering?
Partition data into shards, maps shards to server with consistent hashing

• Does your client know about all of FB’s servers?
  • Security issues?
  • Performance issues?

• How do clients send all requests to replicas?
  • Multicast?
  • Special protocol?
Approaches to Replication

### Active Replication
- FIFO ordering
- Tolerates byzantine failures

### Passive Replication
- Total ordering
- Protocols: Zookeeper, Paxos, Chubby

### Lazy Replication
- Causal ordering
- Protocols: Gossip, DynamoDB, CassandraDB, VoldemortDB, MongoDB
System Assumptions
Assumptions!

- Each program is a state machine
  - Deterministic
  - Given initial state + sequence of events
    - Terminates at same state

- Replicated State Machine (RSM)

- Implications of RSM
  - Each server can independently process events
  - AND reach same conclusion
    - ONLY if events are total ordered
Total Ordering

- If totally ordered what is the final state?

- If FIFO ordered what are potential final states?
FIFO Ordering 1

- If totally ordered what is the final state?
- If FIFO ordered what are potential final states?
FIFO Ordering 2

- If totally ordered what is the final state?
- If FIFO ordered what are potential final states?
Assumptions!

• Each program is a state machine
  • Deterministic
  • Given initial state + sequence of events
    • Terminates at same state

• Replicated State Machine (RSM)

• Implications of RSM
  • Each server can independently process events
  • AND reach same conclusion
    • ONLY if events are total ordered

Given identical initial states, applying updates in same sequence results in same final state
Server Recovery with RSM

- On node failure
  - Start new replica
  - Feed it sequence of events

Interesting questions:
- How to detect failures?
- How to determine sequence of events?
Server Recovery with RSM

- On node failure
  - Start new replica
  - Feed it sequence of events

- Interesting questions:
  - How to detect failures?
  - How to determine sequence of events?

To recover from N failures, you need N+1 replicas. There is always at least one live node.

A manager process uses heartbeats to track node liveness. The manager can get log of events from a working node to bootstrap a new replica.
Active Replication
Ordered Reliable Multicast

1. FE gets ID from local sequencer
2. Sends (event, ID) to all servers
3. On receipt:
   - Server sends to all server + FE
   - Sends the same msg (event, ID)
   - Then processes it with RSM
4. FE waits for response
   - FE receives response from all servers
   - FE knows msg was delivered

Requires $N \times N$ messages
Implementing FIFO Ordering

- Local sequencer: Process locally maintain ID
  - ID is monotonic

- Client Pseudocode
  - Increment ID
  - Assign ID to message
  - Send message to all servers

- Server Pseudocode
  - Maintain a queue for each client
  - Order message by ID -- process in order

This is FIFO Order

Client maintain different IDs: local and monotonic
Active Replication

- Active Replication Steps:
  - Request
    - Client $\to$ FE to Replicas
  - Ordering: ordered reliable multicast
  - Execute $\to$ each server executes
    - Happens have total ordered multicast is completed
    - No Need for Agreement!!!
  - Response
    - Replicas $\to$ FE
    - FE-$\to$ Client (after replicas respond)
Passive Replication
Terminology and Assumptions

• Servers are divided into two groups
  • Leader: only one at a time
  • Followers: all non-leader servers

• FE only communicates with Leader

• Leader provides role of global sequencer
  • Leader orders events
  • Leader is single point of failure
    • Will discuss recovery in Raft!!!
Passive Replication

- **Steps:**
  - Request
    - Client → FE to Leader
  - Ordering: Leader provides total ordering
    - Only processes messages one at a time
  - Execute → only leader executes
  - Agreement → leader informs follower of response
    - All replicas agree on value
  - Response
    - Leader → FE → Client
Active Versus Passive Replication
Replication

Active Replication

- Local Sequencer
- FE
- Server A
- Server B
- Server C

Reliable Multicast: Lots of Messages

Passive Replication

- FE
- Server A (leader)
- Server B (follower)
- Server C (follower)

Single point of failure

Only leader executes: Saves CPU
Replication

• Passive Replication Steps:
  • Request
    • Client → FE to Leader
  • Ordering: Leader provides total ordering
    • Only processes messages one at a time
  • Execute → only leader executes
  • Agreement → leader informs follower of response
    • All replicas agree on value
  • Response
    • Leader response AFTER Agreement
    • Leader → FE → Client

• Active Replication Steps:
  • Request
    • Client → FE to Replicas
  • Ordering: ordered reliable multicast
  • Execute → each server executes
    • Happens have total ordered multicast is completed
    • No Need for Agreement!!!
  • Response
    • Replicas → FE
    • FE-> Client (after replicas respond)

Single point of failure
Requires N*N messages
Requires N* CPU processing
Must wait for All followers
Must wait for All Servers