Distributed Systems (ICE 601)

Replication & Consistency - Part 1

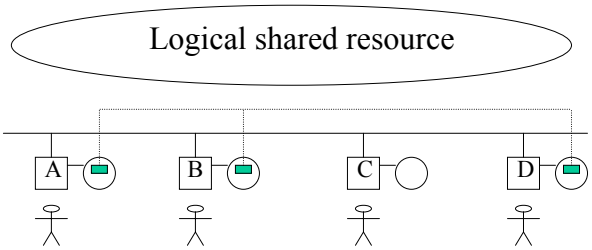
Dongman Lee
ICU

Class Overview

- Introduction
- Replication Model
- Request Ordering
- Consistency Models
- Consistency Protocols
- Case study
  - Lazy replication
  - ISIS
  - Transactions with Replicated Data

Why Replication?

- Purpose
  - increase availability, dependability and/or performance without knowledge of replica visibility
- Replication transparency
  - hiding the replication of state in a system
    - active vs. primary/stand-by replicas
    - generic functions: active and passive replication mechanisms

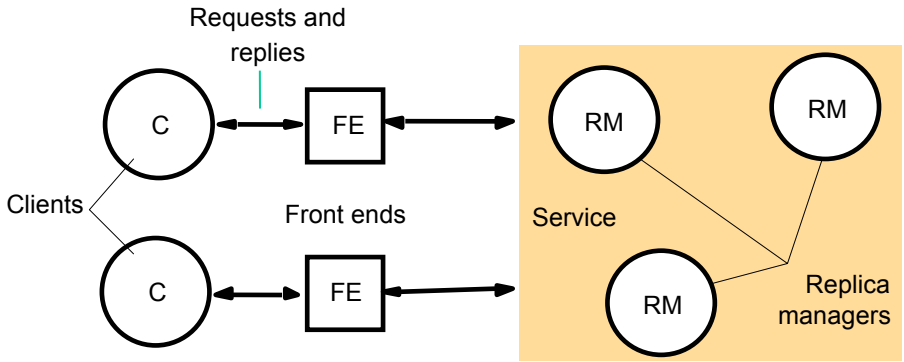


Replication Model

- Replication model spectrum
  - consistency
    - totally synchronous model
      - complete synchronization among replicas
    - asynchronous model
      - asynchronous update of replicas - that is, allow temporal inconsistency among replicas
    - most replication models are somewhere between these two models
  - purpose
    - performance improvement
      - reduction of delay by caching or replicating a server near clients
    - availability
      - make the service accessible (close to 100%) in the presence of process and network failures (partition and disconnection)
    - fault tolerance
      - guarantee strictly correct behavior despite of failures (byzantine and crash)

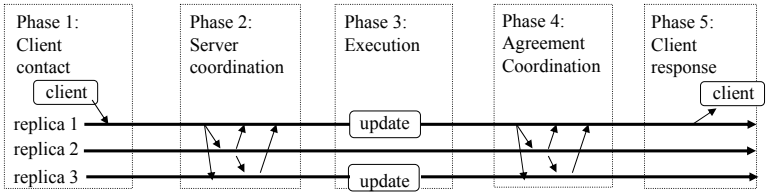
Replication Model (cont.)

- Basic architectural model for replicated data management



Replication System Model [Weismann]

- Replication protocol model
  - Request phase
    - active replication
    - passive replication
  - Server coordination
    - message ordering: FIFO, causal, total
  - Execution
  - Agreement coordination
    - necessary in database while ordering guarantee is enough for distributed systems
  - Client response
    - synchronous vs. lazy or asynchronous



Replication System Model (cont.)

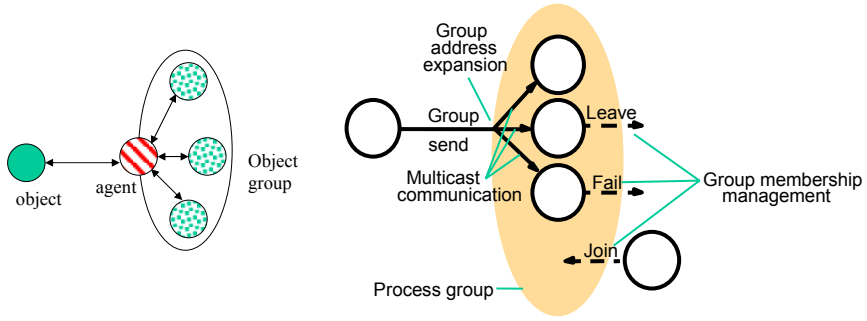
- Replication model
  - Active replication
    - deterministic execution
    - request sent to replicas using atomic totally ordered multicast
    - no need of agreement
  - Passive replication
    - non-deterministic execution
    - view synchronization
    - no need of server coordination
  - Semi-active replication
    - non-deterministic execution
    - request sent to replicas using atomic totally ordered multicast
    - leader informs followers of its choice using view synchronization
  - Semi-passive replication
    - same as passive without view synchronization
    - allow for aggressive time-outs values and suspecting crashed processes without incurring too high cost for incorrect failure suspicions

Group System Model

- Definition
  - a set of one or more objects, joined through a common interface, acting as a single unit for purposes of naming and function
- Group types
  - replicate
    - replicated members for highly availability and/or reliability
      - primary/stand-by
      - modular redundant
  - partition
    - members executing a common job in a divided manner
      - e.g. highly parallel array processing
  - aggregate
    - non-replicated members sharing the provision of the service defined by group's interface
      - e.g. group conferencing

Group Communication Model

- Group communication =
  membership service + multicast with ordering support



Object group interaction model

Membership Service

- Membership service
  - interface for group membership changes
  - failure detection
  - membership change notification
  - group address expansion
- View delivery
  - view: a list of the currently active and connected members in a group
  - basic requirements for view delivery (view notification)
    - order
      - If a process p delivers view $v(g)$ and the $v'(g)$, then no other process $q \neq p$ delivers $v'(g)$ before $\underline{v}(g)$
    - integrity
      - If process p delivers view $v(g)$ then $p \in v(g)$
    - Non-triviality
      - If process q joins a group and is or becomes indefinitely reachable from process $p \neq q$, then eventually q is always in the views that p delivers

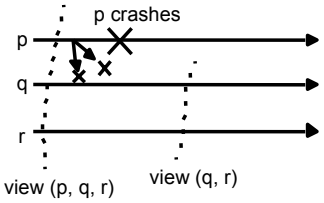
Membership Service (cont.)

- View-synchronous group communication
  - Guarantees provided by view-synchronous group communication
    - agreement
      - correct processes deliver the same set of messages in any given view
    - integrity
      - if a process *p* delivers message *m*, then it will not deliver *m* again
    - validity
      - correct processes always deliver the messages that they send

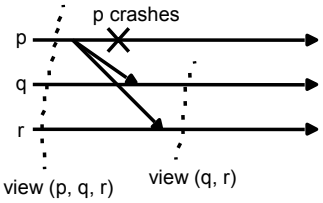
Membership Service (cont.)

- View-synchronous group communication (cont.)

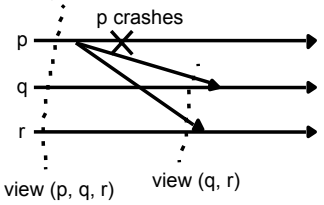
a (allowed).



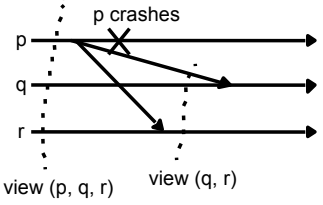
b (allowed).



c (disallowed).



d (disallowed).



Request Ordering

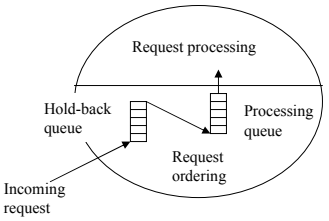
- Why ordering is concerned?
  - concurrent execution of update requests at replicas may result in inconsistency among replicated data
  - ⇨ serial equivalence of update requests is required
    - expense of ordering should also be considered
- Ordering requirements
  - total ordering
    - requests are processed in the same order at all replicas
  - causal ordering
    - causally related requests are only ordered at all replicas
  - sync ordering
    - requests are ordered in sync before or after a certain request at all replicas

Request Ordering (cont.)

- Request handling at replicas
  - every request is *held-back* until ordering constraints can be met
  - request is defined to be *stable* at a replica once no request from a client and bearing a lower unique identifier can be subsequently delivered to replica; that is, all prior requests have been processed
- Request ordering implementation
  - group communication
    - ISIS
  - exchanging gossip messages among replicas
    - lazy replication

Request Ordering (cont.)

- Properties for request ordering
  - safety
    - no message will be delivered out of order from hold-back queue to processing queue
    - once a message has been in the processing queue, no prior request should not be in there
  - liveness
    - no message should wait indefinitely in hold-back queue



Request Ordering (cont.)

- Total ordering implementation
  - requires a mechanism to uniquely sequence each request, which enables sequential ordering among messages
  - unique id generation
    - sequencer approach
      - request id is generated by a designated process, sequencer
      - every request is sent to the sequencer which assigns a unique id being incremented monotonically and forwards the request to replicas
      - sequencer may become performance bottleneck and point of failure
    - data update protocol approach
      - token holder sends a request with a temporary id to all replicas
      - each replica (site i) replies with a new id of max(temp id, id) + 1 + i/N; token holder selects largest id among proposed id from all replicas and uses it as the agreed id
      - token holder notifies all replicas of the final id; replica readjusts the message's position at hold-back queue

Request Ordering (cont.)

- Causal ordering implementation
  - requires a mechanism to enable causally related requests to be ordered
  - vector timestamp approach
    - all replicas p_i initializes VT_i (vector time) to zeros
    - when p_i generates a new event, it increments $VT_i[I]$ by 1; it attaches the value $vt = VT_i$ on outgoing messages
    - when p_j handles a request with timestamp vt, it updates it vector clock such as Vt_j = merge (Vt_j, vt)