Replication

CMPS 4760/6760: Distributed Systems
Overview

- Replications (18.1)
- View-synchronous group communication (18.2)
- Fault-tolerance service (18.3)
Examples of Replication

- Caching in browsers
- Caching in web proxy servers
- DNS caching
- Cloud storage: dropbox,
- ...
Performance enhancement

- Techniques: data caching, load balancing
- Benefit: reduce latency and network traffic
- Overhead: keep changing data up-to-date
Increased availability

- Availability: portion of time for which a service is accessible with reasonable response time, should be close to 100%

- Factors that reduce availability
  - Delay: slow links; delay due to pessimistic concurrency control (e.g., locking)
  - Server failures
  - Network partitions and disconnected operation (e.g., due to mobility)

- Ex: An object is replicated at $n$ servers. If each of them fail independently with a probability $p$, the availability is $1 - p^n$
Fault Tolerance

- Highly available data ≠ strictly correct data
- Fault tolerance: guarantee strictly correct behavior despite of failures
  - Data freshness
  - Timeliness
- Replicating data and functionality helps
  - The system need to coordinate its components precisely to maintain correctness in the face of failures
Requirements

- Replication **Transparency**
  - Clients should not have to be aware of multiple physical copies on the server side

- Replication **Consistency**
  - All clients see single consistent copy of data, in spite of replication
  - For transactions, guarantee ACID
A basic architectural model

- Recoverable operations
- RM is a state machine
  - Atomicity
  - Deterministic

- Read-only requests
- Update requests
Overview

- Replications (18.1)
- View-synchronous group communication (18.2)
- Fault-tolerance service (18.3)
Group Membership Management

- Dynamic membership is important for replica management
  - Users may add or withdraw a replica manager, or a replica manager may crash
- Full group membership service maintain group **views**
  - i.e., lists of the current group members, identified by their unique process ids
  - The list is ordered, e.g., according to when the members joined the group
Example. Current view $v_0(g) = \{0, 1, 2, 3\}$. Let 1, 2 leave and 4 join the group concurrently. This view change can be serialized in many ways:

- $\{0,1,2,3\}, \{0,1,3\} \{0,3,4\}$, OR
- $\{0,1,2,3\}, \{0,2,3\}, \{0,3\}, \{0,3,4\}$, OR
- $\{0,1,2,3\}, \{0,3\}, \{0,3,4\}$

To make sure that every member observe these changes in the same order, changes in the view should be sent via totally ordered multicast.
View delivery - Requirements

- **Order:** if a process $p$ delivers view $v(g)$ and then view $v'(g)$, then no other processes delivers $v'(g)$ before $v(g)$

- **Integrity:** if process $p$ delivers view $v(g)$, then $p \in v(g)$

- **Non-triviality:**
  - (a) if process $q$ joins a group and is or becomes indefinitely reachable from $p \neq q$, then eventually $q$ is always in the views that $p$ delivers.
  - (b) if the group partitions and remains partitioned, then eventually the views delivered in any one partition will exclude any processes in another partition.
View-synchronous group communication

*Rule:* With respect to each message, all correct processes have the same view.

$m$ sent in view $V \Rightarrow m$ received in view $V$

This is also known as *virtual synchrony*
View-synchronous group communication

- **Agreement**: If a correct process $k$ delivers a message $m$ in $v_i(g)$ before delivering the next view $v_{i+1}(g)$, then every correct process $j \in v_i(g) \cap v_{i+1}(g)$ must deliver $m$ before delivering $v_{i+1}(g)$.

- **Integrity**: If a process $j$ delivers a view $v_i(g)$, then $j \in v_i(g)$

- **Validity**: If a correct process $k$ delivers a message $m$ in view $v_i(g)$ and another process $j \in v_i(g)$ does not deliver $m$, then the next view $v_{i+1}(g)$ delivered by $k$ must exclude $j$. 
a (allowed).

p crashes

view (p, q, r) view (q, r)

b (allowed).

p crashes

view (p, q, r) view (q, r)

c (disallowed).

p crashes

view (p, q, r) view (q, r)

d (disallowed).

p crashes

view (p, q, r) view (q, r)
Overview

- Replications (18.1)
- View-synchronous group communication (18.2)
- Fault-tolerance service (18.3)
Passive (primary-backup) model for fault tolerance

Request: the front end issues the request containing a unique id to primary RM

Coordination: primary takes each request atomically, in the order in which it receives it

Execution: primary executes the request and stores the response

Agreement: if the request is an update, primary sends the updated state, response, and unique id to all backups using view synchronous communication

Response: the primary responds to the front end
Passive (primary-backup) model for fault tolerance

- Implements linearizability, i.e., there is an interleaving of operations that is
  - *valid*,
  - *per-process order-preserving*, and
  - *real-time order-preserving*
- If the primary crashes, a single backup becomes a new primary
- survives up to $f$ crashes with $f + 1$ RMs
- does not tolerate Byzantine failures
Active replication

**Request:** the front end multicast the request to RMs using **totally ordered and reliable** multicast

**Coordination:** the request is delivered to every correct RM in the same order

**Execution:** every RM executes the request

**Agreement:** N/A

**Response:** each RM sends its response to the front end. The front end wait for (a subset of) replies, depending on application
Active replication

- Achieves **sequential consistency**, i.e., there is an interleaving of operations that is
  - *valid*,
  - *per-process order-preserving*
- Works even with Byzantine failures (assuming the front end may fail by crashing at worst)
- Can use other types of ordering, depending on application: FIFO, causal