Replication Management
: State-Machine Approach

Chapter 7
Chapter 7 : Contents

- Introduction : State-Machines
- Fault Tolerance
- Fault-Tolerant State-Machine
  - Stability test using Logical Clock
  - Synchronized Real-Time Clock
  - Replica-generated Identifiers
- Faulty clients
- Reconfiguration
Introduction

• Replicating services
  – fault-tolerance and throughput

• Replication Management
  – synchronizing requests, operations and their order
  – State-machine approach
    • no centralized control
  – Primary-backup approach
State-Machines

• A state machine consists of
  – state variable
  – commands: state transition (atomic)
  – commands implementation
    • procedure call (shared data)
      – concurrent execution
    • simple server process that awaits
    • collection of interrupt handlers
Fault-Tolerance

• Failure
  – Byzantine failures
  – Fail-stop failures

• Availability of a system
  – t-fault tolerant
    • a system satisfies its spec provided that no more than t of components become faulty during some interval
    • Minimum number of servers to ensure t-fault tolerance in question
Fault-Tolerant State Machines

- A set of replicated state machines
  - output determined by the majority of all replicas
  - Only Fail-stop failure
    - $t + 1$ replicas for $t$-FT
  - Byzantine failure with msg authentication
    - $2t + 1$ needed (majority rules)
  - Malicious Byzantine failure
    - $3t + 1$ (with Byzantine agreement protocol)
Fault-Tolerant State Machines

- Key Goal: Replica Coordination
  - All replicas receive and process the same sequence of requests

- Two requirements
  - Agreement
    - Every non-faulty state-machine replica receives every requests
  - Order
    - processes the requests it receives in the same relative order
Fault-Tolerant State Machines

• Weaker conditions
  – Agreement
    • read-only requests & fail-stop : agreement guaranteed with only 1 replica
  – Order
    • commutable requests can be mixed up
Agreement

• “transmitter” to ensure agreement
  – IC1 : All non-faulty processors agree on the same value
  – IC2 : If the transmitter is non-faulty, then all non-faulty processors use its value as the one on which they agree
  – Byzantine Agreement / Reliable Broadcast protocol
Order

• To satisfy Order requirement
  – conceptually “unique id” for each request
  – replicas process requests in their id order
• stable request
  – A request is stable if “no request with lower id will be delivered” is guaranteed
• Order Implementation made simple :
  – A replica processes the stable request with smallest unique identifier
Stability Test using Logical Clock

• A logical clock
  – a mapping $T$ from events to the integers
  – $e \rightarrow e'$ then $T(e) < T(e')$
  – Each process $p$ maintain a timestamp
    LC1: $T_p$ is incremented after each event at $p$
    LC2: Upon receipt of a message with timestamp $t$, process $p$ resets $T_p$:
      $$ T_p := \max(T_p, t) + 1 $$
Logical Clock

• Logical Clock Example
Logical Clock

• Assumptions
  – FIFO channel
  – Failure detection

• Stability test under failstop assumption
  – A request is stable at $sm_i$ if a request with larger timestamp has been received by $sm_i$ from every correct clients
  – no request with smaller id will be received
Synchronized Real-Time Clock

• Assumptions
  – no two request is generated at the same clock tick
  – degree of clock synchronization is better than minimum message delivery time
• Use real-time clock as $T_p(e)$
Stability test using Synchronized RTC

• Stability condition
  – A request $r$ is stable at $sm_i$ when local clock is $t$ if $uid(r) < t - \Delta$
Replica-Generated Identifiers

• 2 Phase to determine identifier
  – P1 : each replica propose $cuid(r)$
  – P2 : one c-uid is selected for that request

• 2 way to determine
  – agreement protocol between replicas
  – client decides and notify servers
Replica-generated Identifiers

• Uid constraints
  – UID1: $cuid(sm_i, r) \leq uid(r)$
  – UID2: If a request $r'$ is seen by replica $sm_i$, $uid(r) < cuid(sm_i, r')$

• Stability test
  – $r$ is stable if there is no request $r'$ which is
    1) seen by $sm_i$ and
    2) not accepted by $sm_i$ and
    3) $cuid(sm_i, r') \leq uid(r)$
Replica-generated Identifiers

- A simple implementation (Failstop)
  - Each replica maintains
    - $SEEN_i$: the largest $cuid(sm_i, r)$ so far
    - $ACCEPT_i$: the largest $uid(r)$ assigned so far
  - Upon receipt of a request $r$,
    - $cuid(sm_i, r) := \max(\lceil SEEN_i \rceil, \lceil ACCEPT_i \rceil) + 1 + i$
    - $uid(r) := \max(cuid(sm_i, r))$
Replica-generated Identifiers

: An Example Scenario

• Let example requests
  – seen by $sm_1 : r_1, r_2$ ; by $sm_2 : r_2, r_1$

\[
\begin{align*}
\text{sm}_1 & \\
\text{SEEN1} = 2 & \quad \text{cuid}(\text{sm}_1, r_1) = 2 \\
\text{SEEN1} = 4 & \quad \text{cuid}(\text{sm}_1, r_2) = 4 \\
\text{ACCEPT1} = 6 & \quad \text{uid}(r_1) = 6 \\
& \quad \text{uid}(r_2) = 4 \\
\end{align*}
\]

\[
\begin{align*}
\text{sm}_2 & \\
\text{SEEN2} = 3 & \quad \text{cuid}(\text{sm}_1, r_2) = 3 \\
\text{SEEN2} = 6 & \quad \text{cuid}(\text{sm}_1, r_1) = 6 \\
\text{ACCEPT2} = 6 & \quad \text{uid}(r_1) = 6 \\
& \quad \text{uid}(r_2) = 4
\end{align*}
\]
Faulty Devices

• Faulty devices outside the system
  – replicate device and voter

• Faulty devices inside the system
  – the client itself combine the outputs of replicas
    • collect t+1 outputs from t-FT state machines
Faulty Clients

• t-FT state machine is not sufficient as a t-FT system
  – What if a client makes malicious requests?
• Replicating the client
  – consensus on request contents
• Defensive programming
Using time to Make requests

• Not receiving a request can be considered as a request
• schedule default request / vote on timeout
Reconfiguration

- Reconfiguration
  - configurator dynamically reconfigures system
    - Client set C, server replicas S, output devices O

- Re-integration of servers
  - state and pending requests forwarded to $sm_{new}$
  - relay incoming requests to $sm_{new}$ for a while: until when?
Integration Protocol

• with Failstop + Logical Clocks
  – send every state variables and pending requests to $sm_{new}$
  – relay every subsequent request $r$ such that $uid(r) < uid(r_c)$ where $r_c$ is the first request received directly from clients to $sm_{new}$

• With Failstop + Real-time Clocks
  – continue relaying during next $\Delta$ interval
Stability Revised

• Stability Test during Restart
  – A request $r$ received directly from a client $c$ by a restarting replica $sm_{new}$ is stable only after the last request from $c$ relayed by another processor has been received by $sm_{new}$