

Software replication techniques

Carlo Marchetti, PhD
carlo.marchetti [AT] dis.uniroma1.it

Overview

- ✦ What is availability?
- ✦ What consistency properties for replicated objects?
- ✦ How to enforce “strong” consistency?
 - Passive replication
 - Active Replication
- ✦ Sum up and comparison of replication techniques
- ✦ Software architectures for implementing sw replication

Availability

✧ Dependability =

Availability (output is produced)

+

Reliability (output is correct)

✧ Availability = $\text{Uptime} / (\text{Uptime} + \text{Downtime})$

Availability, failures, replication

✧ Let p be the probability with which an object O can stop working (crash failure model)

✧ Then:

– Availability of O : $1-p$

✧ If the service provided by O , i.e., S_o , can be provided by n replicas of O then:

– Availability of S_o : $1-p^n$

Replication, consistency

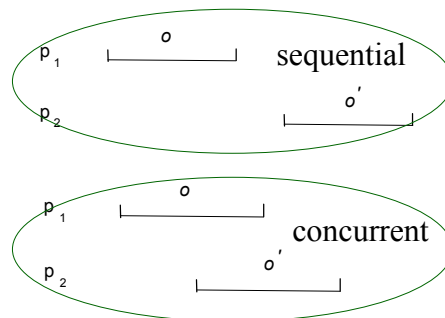
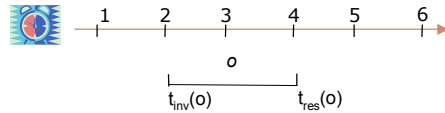
- ✚ Increasing the number of service providers increases the probability of producing a reply
- ✚ What about reply correctness?
 - Replicated uncoordinated objects return independent replies which might well hinder consistency
 - Site reservation service
 - Bank accounts...
- ✚ Need of stating a consistency criteria for our replicated objects

Consistency criteria

- ✚ There are several, e.g.:
 - Eventual consistency
 - E.g. DNS, some P2P systems
 - Sequential consistency
 - Replicated objects
 - Linearizability
 - Sequential + real time constraints on operations
- ✚ In this lecture, we focus on linearizability and sequential consistency (strong consistency criteria)
- ✚ These criteria permits to client to “forget” about replication
 - Replication transparency

Operation order

- ☛ Operations take time to complete
- ☛ They can overlap if executed by concurrent sequential processes
- ☛ Two operations are said
 - Concurrent if they overlap
 - Sequential if they don't
 - $O < O'$



Linearizability

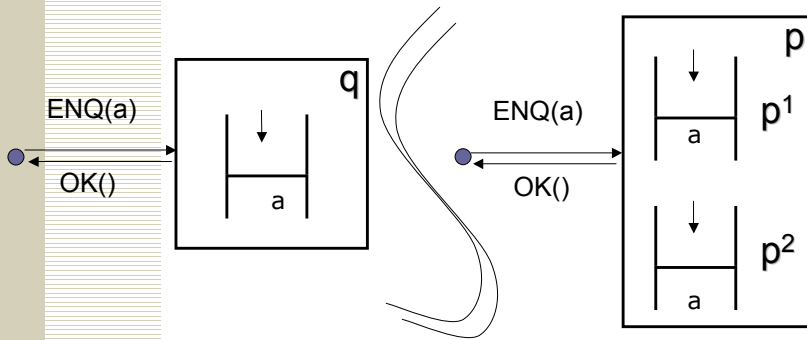
- ☛ Given
 - an object O with operation specification O_s
 - a set of sequential processes SeP implementing So
 - a set of (sequential client processes invoking) O 's operations on members of SeP
- ☛ The history of the executions of SeP members is linearizable iff there exist at least a sequence of O 's operations S , s.t.
 - It contains all and only the operations executed in SeP
 - If $o < o'$ in SeP then $o < o'$ in S
 - S satisfy O_s

Fifo queue example

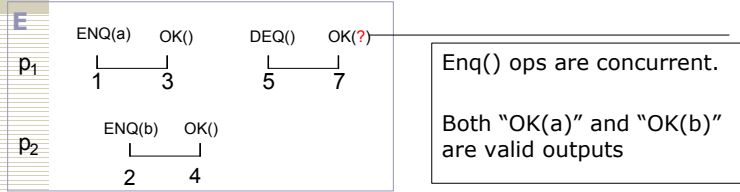
Assume a FIFO queue q is implemented by two distinct processes

Semantics world

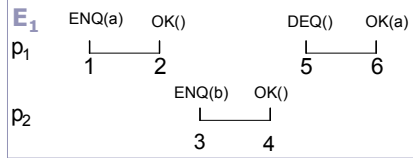
Implementation world



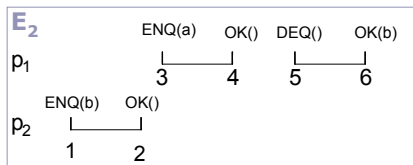
Fifo queue example



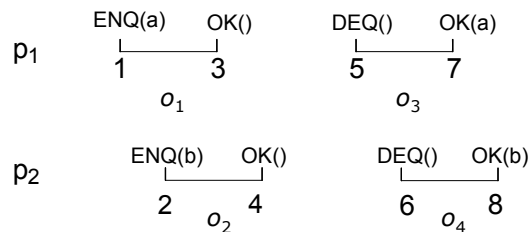
If p₁ returns "a" then
S=E1



Otherwise (p₁ returns "b")
S=E2

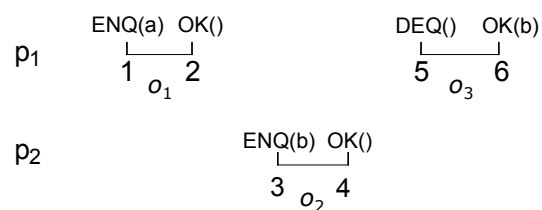


Fifo queue example



- ✦ If we assume q starts empty in E , then there exists $S = \sigma_2 \sigma_1 \sigma_4 \sigma_3$ s.t. linearizability is satisfied by E
- ✦ What if $O_4 = OK(a)$?

Fifo queue example



- ✦ Is E linearizable?
- ✦ ...
- ✦ Why?
- ✦ ...

Implementing linearizability

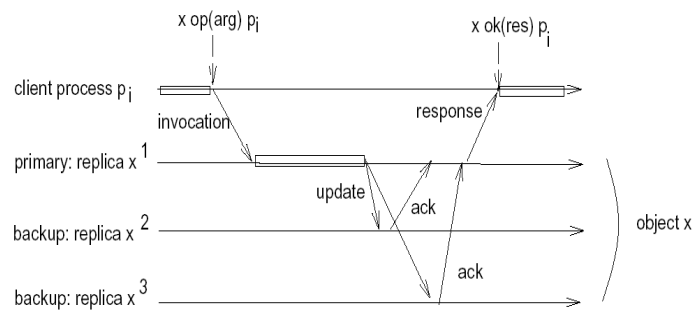
- ✦ Sufficient conditions for implementing linearizable executions of a replicated object implemented by n sequential processes called replicas are:
 - **Uniform Atomicity**: if a replica completes operation O then eventually all correct replicas complete O
 - **Uniform Order**: if two replicas both execute two operation O and O' , then they execute O and O' in the same order
- ✦ Note that
 - real times are referred to executions on replicas and not on clients (operations can be reordered wrt to client invocations real time order)
 - Uniform properties \leftrightarrow non-malicious fault models...

SW replication techniques

- ✦ Two main techniques
 - Active replication
 - Deterministic replicas, failure masking
 - Passive replication (or primary-backup)
 - Nondeterministic replicas, no failure masking
- ✦ System model: asynchronous distributed system
 - Processes: clients and replicas (non overlapping)
 - Channels: quasi-reliable (reliable delivery, no creation, no duplication)
 - Crash fault model

Passive replication

- ✦ The primary replica manages all the request processing
- ✦ Backup replicas follow the primary state



Replica update

- ✦ Consistent replication is “*fighting possible inconsistencies due to nondeterminism*”, therefore
 - The primary replica executes operations and updates backups with:
 - **invid**: unique request identifier
 - **res**: response to be sent to client (retransmissions)
 - **state-update**: state to be set by backups
 - Primary waits for all backup acks before proceeding

Failure scenarios

🚩 Let's note

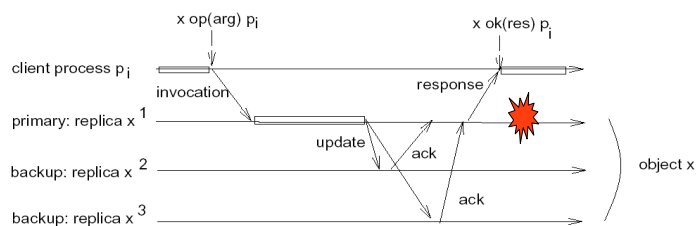
- faulty backups are not a problem

🚩 Let's assume

- in case of a primary failure, another (unique) primary is eventually elected among backups

- 🚩 In the following slides, by analysing primary failure scenarios, we identify the main inconsistency sources in order to spot the mechanisms permitting to avoid them

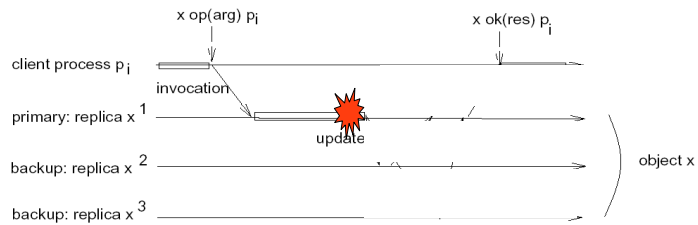
Primary failure 1



- 🚩 The failure occurs after the client received the reply

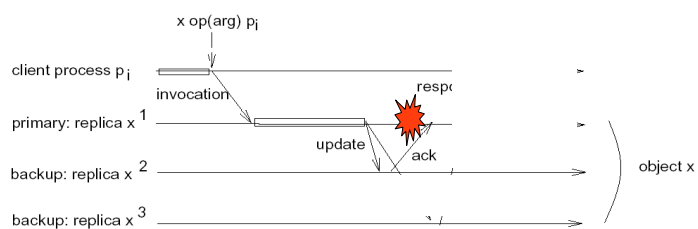
- No problem: a new primary will be soon elected (cfr. assumptions)

Primary failure 2



- ✘ If the primary fails before updating any backup, no reply is sent back to clients
 - No problem, again

Primary failure 3



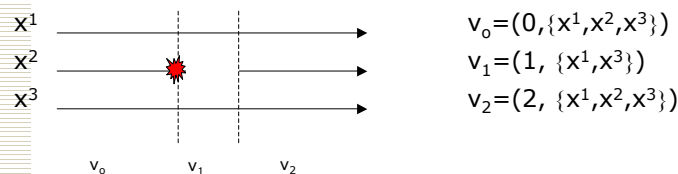
- ✘ The failure occurs before receiving all acks by bcks.
 - Problem: some bcks updated, some others not
 - Update atomicity violation

Enforcing order and atomicity

- ✦ As long there's at most a primary in the replica group, order is guaranteed:
 - Need: *enforcing a consistent view of replica group composition by all alive replicas*
- ✦ As long as backups receive the same set of updates from primaries, responses returned to clients are consistent
 - Need: *enforcing atomicity of update delivery among backups*
- ✦ Both needs can be solved using a group communication primitive named *view synchronous multicast*

VSM: definition

- ✦ View: tuple containing an integer (Vid) and an ordered list of processes (that “belong” to the view)



- ✦ Views are agreed upon among processes according to a specification implemented by a “group membership service”
 - A process is said to “install” a view upon delivering it

VSM: definition

Some membership service properties:

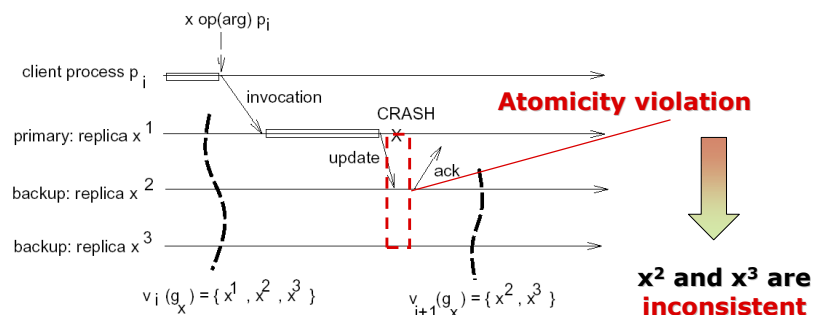
- If $p \in g_x$ crashes, a new view v has to be defined and p not belongs to v
- Let $v_i(g_x)$ be a view s.t. $p \in v_i(g_x)$, then either p installs $v_i(g_x)$ or $\exists k > 0$ t.c. $p \notin v_{i+k}(g_x)$.
- $\forall p, q \in g_x$, if p and q both install $v_i(g_x)$ and $v_j(g_x)$ and $i \neq j$, then p and q install them in the same order

NOTE: processes can be removed by views even if not crashed (unreliable failure detection)

VSM: definition

We just introduced views and their properties wrt processes (the group membership service)

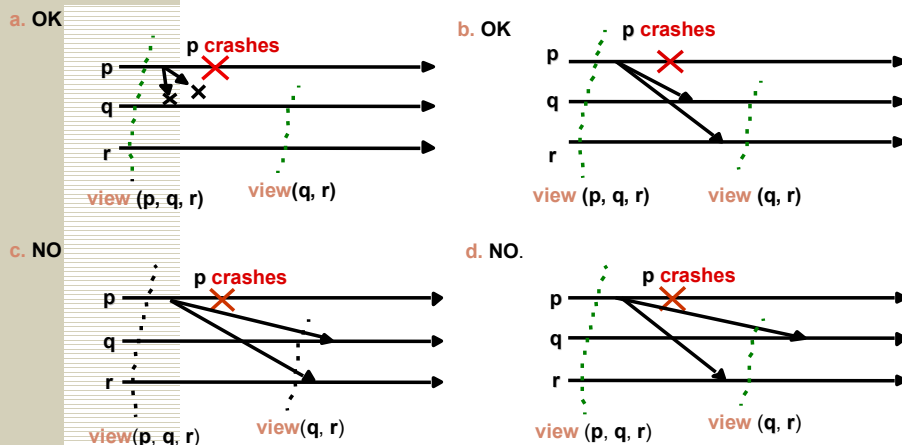
This is not sufficient for enforcing consistency



VSM: definition

- ✦ VSM is a multicast primitive based on a group membership service
- ✦ The properties of VSM basically relate message delivery to the views in which messages are delivered
- ✦ This permits to state atomicity of delivery within a view, formally: “if $\exists p \in v_i$ that delivers message m in v_i and installs v_{i+1} , then $\forall q \in v_i$ that delivers v_{i+1} , q delivers m before delivering v_{i+1} ”

VSM: definition



- ✦ C is not strictly a problem for passive replication. Why?

VSM and passive replication

✦ GM+VSM “solves” passive replication

- Primary can be elected using views and the order of members in a view
 - Order is then guaranteed by the primary
- The multicast primitive permits to update either all or none backups in a view
 - Update atomicity is guaranteed by VSM

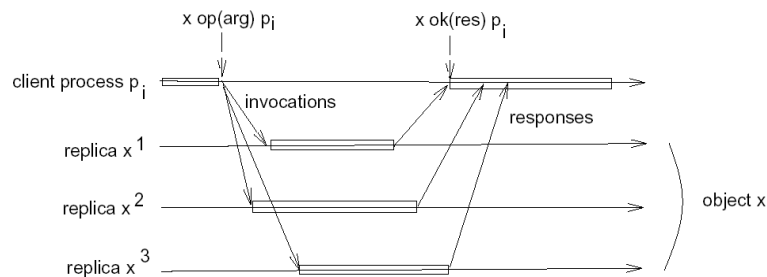
✦ Problem: we have not dealt with liveness conditions

Active replication

✦ Basics

- All replicas are equal (no primary)
- Only deterministic replicas
 - The reply only depends on the sequence of executions
- When a client issues a request,
 - All correct replicas receive the request and return a reply to the client
 - Client only wait for the first reply

Active replication

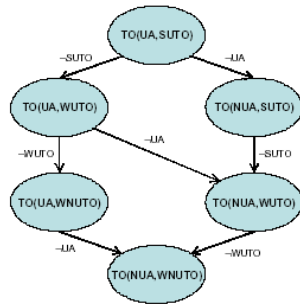


- ✦ All correct replicas execute requests and return replies to clients

Consistency

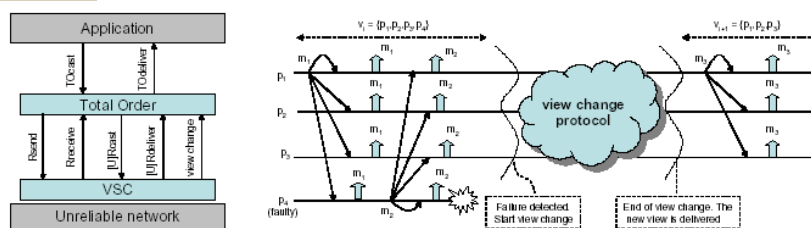
- ✦ In order to get consistency there must hold:
 - **Uniform atomicity**: if a replica executes a request req , then eventually all correct requests execute req
 - **Uniform order**: if two replicas execute the same two requests, they execute these requests in the same order
- ✦ Channels are quasi reliable and FIFO
 - No consistency can be guaranteed without building a proper communication layer

Total order multicast



- Many specifications and implementations available
- What spec to permits to implement linearizable executions?
- ...

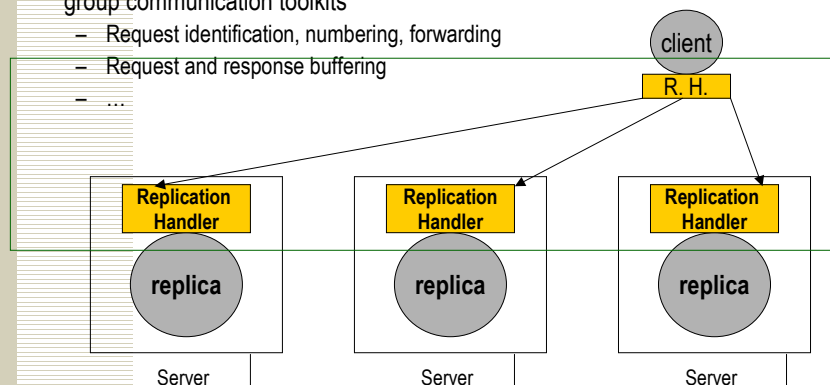
TOM implementation



- Membership and view synchronous delivery are not strictly necessary but very useful
- Clients and client requests are omitted in figure

Sw architecture for replication

- ✦ Typical two-tier sw replication architecture for client/server applications
- ✦ Client use retransmission and dynaming name resolution mechanisms
- ✦ Replication handlers implement consistency mechanisms in replicas using group communication toolkits
 - Request identification, numbering, forwarding
 - Request and response buffering
 - ...



Some g.c. toolkits

- ✦ Jgroups (www.javagroups.com)
 - Transport protocols: UDP (IP Multicast), TCP, JMS
 - Reliable unicast and multicast message transmission.
 - Failure detection and group membership
 - Ordering protocols: Atomic (all-or-none message delivery), Fifo, Causal, Total Order (sequencer or token based)
 - USED to implement Jboss Cluster
- ✦ Appia (<http://appia.di.fc.ul.pt>)
- ✦ Ensemble toolkit (<http://www.cs.technion.ac.il/dsl/projects/Ensemble/>)
- ✦ ...

