9 Replication: Availability and Consistency

- Motivation for replication
- Multicasting updates to a group of replicas
- Total Ordering
- Causal Ordering
- Techniques for ordering protocols
- ISIS CBCAST

9.1 What is Replication?

- Multiple copies of dynamic state stored on multiple machines eg Copies of files stored on different machines, name servers storing name address mappings
- Caching can be seen as a form of replication.

9.1.1 Why is Replication used?

Performance enhancement
- Single Server acts as a bottleneck - if we can balance load amongst multiple servers, get apparent performance gain
- If clients are geographically distributed, we can site servers near clients and reduce communication costs

Availability
- If a machine fails, then we can still provide a service
- Probability of total failure reduced such as all data being lost, since data replicated across multiple machines
- If probability of failure is $pr(fail)$ for a given machine in n machines, then probability of loss of service is $1 - pr(fail)^n$
- eg, if mean time between failure for 3 machines is 5 days, repair time is four hours, then assuming independence of failure, $pr(fail) = \frac{4}{5 \times 24} = 0.03$.
  Availability $= 1 - 0.03^3 = 99.996\%$

Fault Tolerance Even in the presence of failure, the service will continue to give the correct service
- Stronger than availability, since can provide real-time guarantees (with extra work!)
- Can protect against arbitrary failure where machines feed wrong information (Byzantine Failure)

9.2 Issues in Replication

A collection of replicas should behave as if state was stored at one single site

- When accessed by client, view should be consistent
- Replication should be transparent - client unaware that servers are replicated

If we are providing a replica service, replica can be passive or active.

9.2.1 Passive Replication

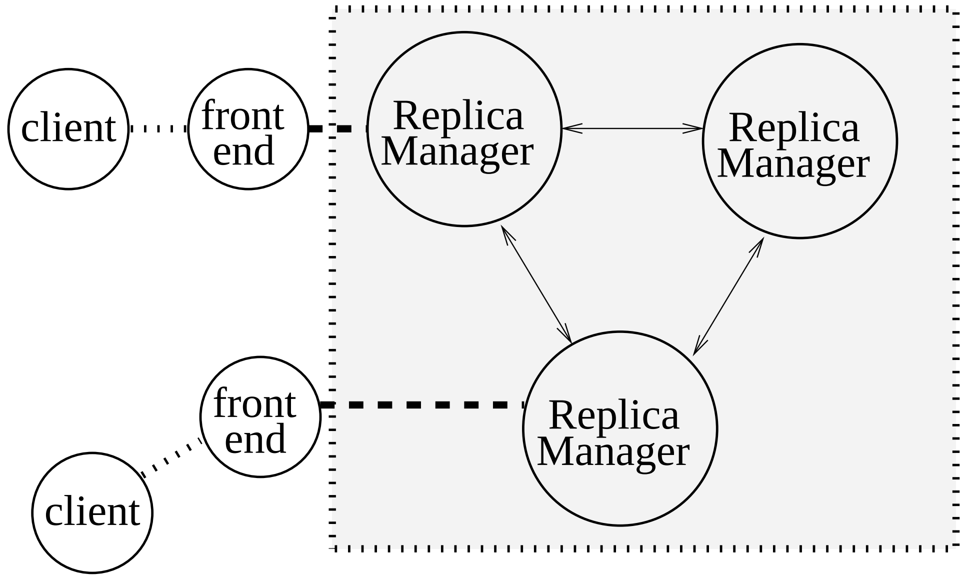
Passive replicas are standbys, to maintain service on failure. No performance improvement.

Standbys must monitor and copy state of active server
Provide availability in simple manner.
Used for highly available systems eg space applications

9.3 Consistency

- Clients can modify resource on any of the replicas.
- What happens if another client requests resource before replica has informed others of modification, as in cache consistency in distributed file systems?
- Answer depends upon application...

9.3.1 Example Distributed Bulletin Board System (BBS)



- Users read and submit articles through Front End.
- Articles replicated across a number of servers
- Front Ends can connect to any server
- Servers propagate articles between themselves so that all servers hold copies of all articles.
- User membership of a given bbs is tightly controlled.

Questions on BBS:

- How should messages be passed between replicas?
- Should order of presentation of articles to clients be the same across all replicas? Are weaker ordering semantics possible?
- When a client leaves bbs group, can they see articles submitted after they have left? Is this desireable?
- What should happen when replicas are temporarily partitioned?

9.4 Updating Server state

Clients read and update state at any of the replicated servers eg submit messages in bbs. To maintain consistency, three things are important

Multicast communication Messages delivered to all servers in the group replicating data

Ordering of messages Updates occur in the same "order" at each server

Failure recovery When servers or the network fails, and comes back, the replicas must be able to regain consistency. Done through Voting and Transactions (later in course)

9.5 Multicast and Process Groups

A Process Group: a collection of processes that co-operate towards a common goal.

Multicast communication: One message is sent to the members of a process group

Idea: Instead of knowing address of process, just need to know an address representing the service. Lower levels take care of routing messages.

Useful for:

Replicated Services One update message goes to all replicas, which perform identical operations. Reduces communication costs.

Locating objects in distributed services Request for object goes to all processes implementing service, but only process holding object replies.

9.5.1 Group Services

Maintenance of group information is a complex function of the name service (for tightly managed groups)

Create Group Create a group identifier that is globally unique.

Join Group Join a group. Requires joining process information to be disseminated to message routing function. May require authentication and notification of existing members.

Leave Group Remove a process from a group. May require authentication, may occur as a result of failure or partition. Need to notify message routing function, may notify other members.

Member List Supply the list of processes within a group. Needed for reliable message delivery, may require authentication.

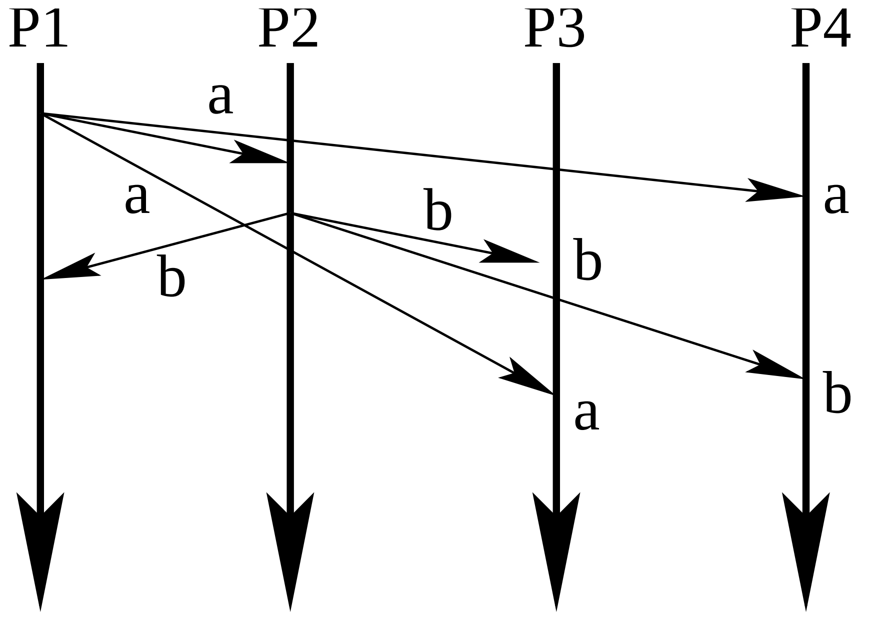
9.6 Message Ordering

If two processes multicast to a group, the messages may be arbitrarily ordered at any member of the group.

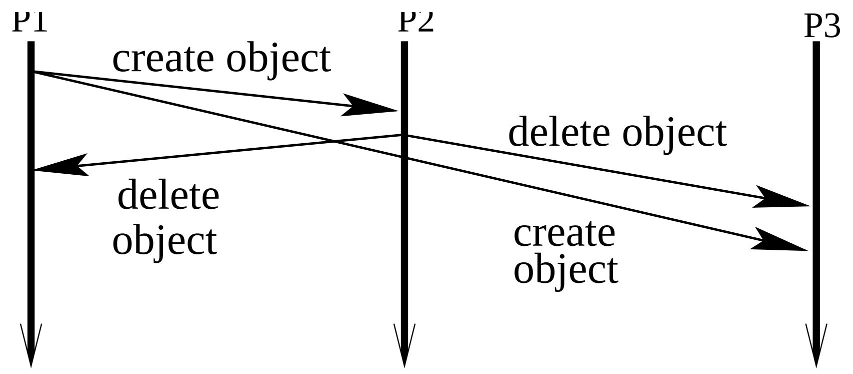
Process P1 multicasts message a to a group comprising processes P1, P2, P3 and P4.

Process P2 multicasts message b to the same group

The order of arrival of a and b at members of the group can be different.



9.6.1 Ordering example



- Order of operations may be important - delete object, create object.
- If delete object arrives before create object, then operation not completed

9.6.2 Ordering Definitions

Various definitions of order with increasing complexity in multicasting protocol

FIFO Ordering Messages from one process are processed at all group members in same order

Causal Ordering All events which preceded the message transmission at a process precede message reception at other processes. Events are message receptions and transmissions.

Total Ordering Messages are processed at each group member in the same order.

Sync Ordering For a sync ordered message, either an event occured before message reception at all processes, or after message. Other events may be causally or totally ordered.

9.6.3 FIFO ordering

Achieved by process adding a sequence number to each message.

Group member orders incoming messages with respect to sequence number.

Applicable when each process state is separate, or operations don't modify state, just add incremental updates or read.

9.6.4 Total Ordering

When several messages are sent to a group, all members of the group receive the messages in the same order.

Two techniques for implementation:

Sequencer Elect a special sequencing node. All messages are sent to sequencer, who then sends messages onto replicas. FIFO ordering from sequencer guarantees total ordering. Suffers from single point of failure (recoverable by election) and bottleneck.

Holdback Queue Received messages are not passed to the application immediately, but are held in a *holdback queue* until the ordering constraints are met.

Sequence Number Negotiation Sender negotiates a largest sequence number with all replicas.

1. Replicas store largest final sequence number yet seen F_{max}, and largest proposed sequence number P_{max}
2. Sender sends all replicas message with temporary ID.
3. Each Replica i replies with suggested sequence number of $max(F_{max}, P_{max}) + 1$. Suggested sequence number provisionally assigned to message and message placed in holdback queue (ordered with *smallest* sequence number at front)
4. Sending site chooses largest sequence number and notifies replicas of final sequence number. Replicas replace provisional sequence number with final sequence number.
5. When item at front of queue has an agreed final sequence number, deliver the message.

9.6.5 Causal Ordering

"Cause" means "since we don't know application, messages might have causal ordering"

a and b are events, generally sending and receiving of messages.

We define the causal relation, $a \rightarrow b$, if

1. if a and b are events at the same process, $a \rightarrow b$ implies a happened before b
2. if a is a message sent by process P1 and b is the arrival of the same message at P2, then $a \rightarrow b$ is true

In bulletin board, an article titled "re: Multicast Routing" in repsonse to an article called "Multicast Routing" should always come after, even though may be received before the initial article.

9.6.6 CBCAST - Causal ordering in ISIS

ISIS is a real commercial distributed system, based on process groups.

Causal ordering for multicast within a group is based around *Vector Timestamps*

The vector VT has an identifier entry for each member of the group, typically an integer.

Vector timestamps have one operation defined

$merge(u, v)[k] = max(u[k], v[k])$, for $k = 1..n$

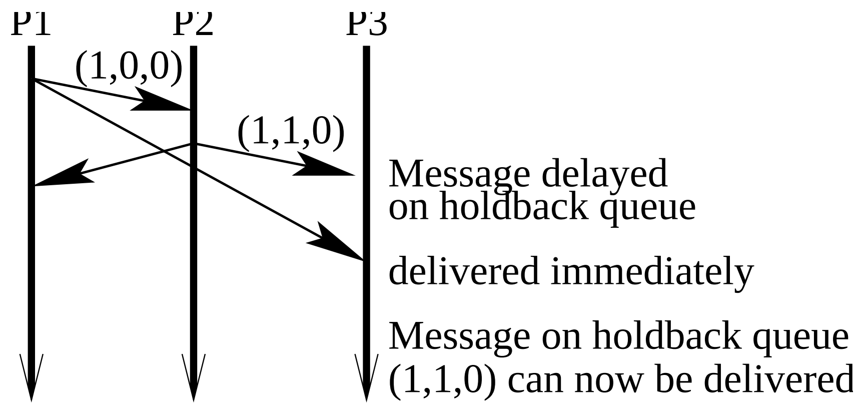
Incoming messages are placed on a holdback queue, until all messages which causally precede the message have been delivered.

CBCAST Implementation

1. All processes p_i initialise the vector to zero
2. When p_i multicasts a new message, it first increments $VT_i[i]$ by 1; it piggy-backs $vt = VT_i$ on the message
3. Messages are delivered to the application in process P_j when
   - The message is the next in sequence from p_i i.e. $vt[i] = VT_j[i] + 1$
   - All causally prior messages that have been delivered to p_i must have been delivered to p_j, i.e. $VT_j[k] \geq vt[k]$ for $k \neq i$.
4. When a message bearing a timestamp vt is delivered to p_j, p_j's timestamp is updated as $VT_j = merge(vt, VT_j)$

In words

- Incoming vector timestamp is compared to current timestamp.
- If conditions for delivery to process not met, then message placed on holdback queue.
- When an incoming message is delivered, the timestamp is updated by the merge.
- Examine all messages in the holdback queue to see if they can be delivered.
- CBCAST requires reliable delivery.


P1
P2
P3
(1,0,0)
(1,1,0)
Message delayed on holdback queue
delivered immediately
Message on holdback queue (1,1,0) can now be delivered

Causal Example

Group View Changes

- When group membership changes, what set of messages should be delivered to members of changed group?
- What happens to undelivered messages of failed members?
- What messages should new member get?
- ISIS solves by sending a *sync ordered* message announcing that the *group view* has changed. Messages thus belong to a particular group view.
- Use coordinator to decide which messages belong to which view.

9.7 Summary

- Replication of services and state increase availability
- Replication increases performance
- Replication increases Fault tolerance
- To maintain consistency, multicast updates to all replicas
- Use sequence numbers to maintain FIFO ordering
- Use Vector Timestamps to maintain Causal Ordering
- Use elected sequencers or identifier negotiation to maintain total ordering