

# Replication in Distributed Systems

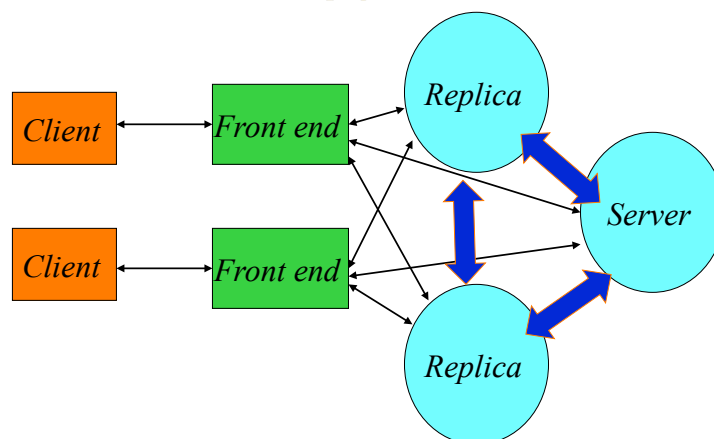
INF 5040 autumn 2011

lecturer: Roman Vitenberg

INF5040, Roman Vitenberg

1

## Replication architecture



INF5040, Roman Vitenberg

2

## Why replication I?

- Better performance
  - Multiple servers offer the same service – parallel processing of client requests
  - Geographical distribution
    - Creating copies of data/objects closer to the clients leads to smaller network delay and possibly reduced network traffic

## Why replication II?

- Better availability (continuous operation despite failures of individual components)
  - For many services it is important that availability with acceptable response time approaches 100%, despite that ...
    - Server processes may fail
    - Parts of the network may fail
    - Data may get corrupted
- Example: 5% chance of a server failure within a given period - two independent servers give 99.75% availability

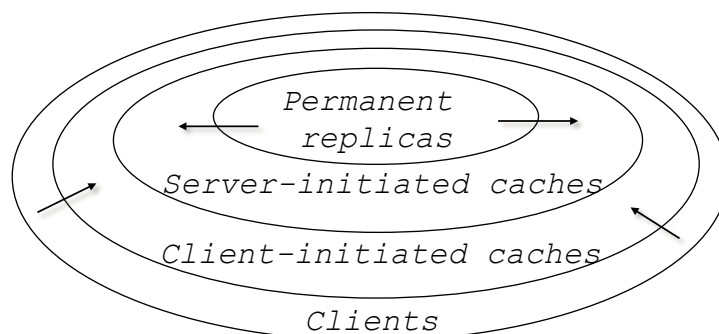
## Challenges of replication

- Complex mechanisms
  - Placement of replicas (and search for them)
  - Propagation of data (e.g., updates) among the replicas
  - Consistency maintenance
  - Monitoring and failover mechanisms
- These protocols also consume bandwidth
- Some of this complexity is exposed to the clients
  - Impossible to achieve complete replication transparency

INF5040, Roman Vitenberg

5

## Placement of replicas



INF5040, Roman Vitenberg

6

## Placement of replicas

- Permanent replicas
  - Clusters of servers
  - Geographically dispersed web mirrors (Akamai)
- Server-initiated caches
  - Placement of hosting servers
  - Placement of caches
    - Flash crowds in the Web
- Client-initiated caches
  - Enterprise proxies or web browser caches

INF5040, Roman Vitenberg

7

## Propagation of updates among the replicas

- Push-based propagation
  - A replica pushes the update to the others
  - May push the new data or parameters of the update operation
- Pull-based propagation
  - A replica requests another replica to send the newest data it has

Issue	Push-based	Pull-based
State at the server	List of client caches	A server to pull data
Messages sent	Update	Poll and update
Freshness of replicas	Eager propagation	By demand

INF5040, Roman Vitenberg

8

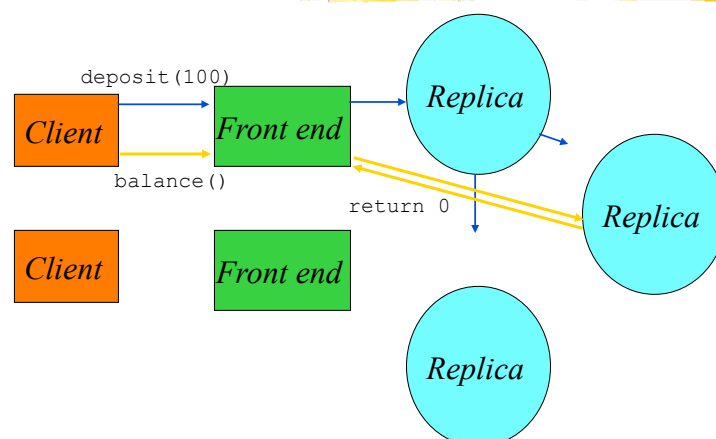
## Propagation of updates among the replicas

- Pushing data vs pushing updates
  - Pushing updates reduces traffic
  - Requires more processing power on each replica
  - Requires deterministic operations
- Hybrid push-pull approaches
  - Lease-based propagation
  - Pushing invalidations
    - A replica that performs the update notifies other replicas
    - A replica informed that a newer version is available will fetch the new version at a later point

INF5040, Roman Vitenberg

9

## Lack of consistency



INF5040, Roman Vitenberg

11

## Lack of consistency

### *Client 1*

deposit<sub>B</sub>(x, 100)

deposit<sub>A</sub>(y, 100)

### *Client 2*

balance<sub>A</sub>(y) → 100

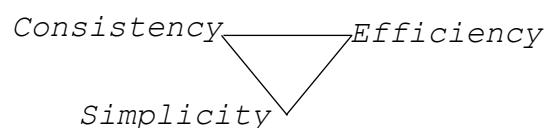
balance<sub>A</sub>(x) → 0

INF5040, Roman Vitenberg

12

## Consistency

- A contract between the client developer and a provider of the replicated service
  - The provider guarantees that the data will be updated according to some consistency criteria
  - The application developer will need to devise applications with these criteria in mind
- “Ideal consistency”: system behavior is indistinguishable from a non-replicated system
- The consistency-efficiency-simplicity triangle



INF5040, Roman Vitenberg

13

## Sequential consistency

- A system consists of a number of servers and a number of objects replicated on those servers
- Objects have well-defined interfaces
- An execution consists of events
  - Each event is an invocation of an operation on one of the replicas at one of the servers (with input and output values)
  - For each object, it is defined whether a sequence of ops makes sense (i.e., fulfills the specification of a single object copy)
- Sequential consistency: for each possible global history produced by system execution there should exist a linearization that fulfills the specification of each object

## Example revisited

### **Client 1**

deposit<sub>B</sub>(x, 100)

deposit<sub>A</sub>(y, 100)

### **Client 2**

balance<sub>A</sub>(y) → 100

balance<sub>A</sub>(x) → 0

*This is not sequentially consistent, because there is no corresponding sequential execution of a non-replicated system*

## More examples

C1	C2	C1			
Deposit(a, 50)	Balance(a) = empty Balance(a) = 50	Deposit(a, 50) Balance(a) = empty			
C1	C2	C1	C2	C3	C4
Deposit(a,50)	Balance(a) = 50 Balance(a) = empty	Bal(b)=30 Bal(a)=em	Bal(a)=50 Bal(b)=em	Dep(a,50)	Dep(b,30)

INF5040, Roman Vitenberg

17

## Active replication (replicated state machine)

- The idea:
  - Every replica sees exactly the same set of messages in the same order and will process them in that order
- Benefits:
  - Every server is able to respond to client queries with updated data
  - Immediate fail-over
- Limitations:
  - Waste of resources, since all replicas are doing the same
  - Update propagation only, which requires determinism
- Different implementation levels
  - Machine instruction level (or VM), e.g., Tandem™
  - Logical state (software-based active replication)

INF5040, Roman Vitenberg

18



## Passive replication (primary-backup replication)

- One server plays a special primary role
  - Performs all the updates
  - May propagate them to backup replicas eagerly or lazily
  - Maintains the most updated state
- Backup servers may take off the load of processing client requests but only if stale results are ok
- Implementable without deterministic operations
- Typically easier to implement than active replication
- Less network traffic during the normal operation but longer recovery with possible data loss
- Several sub-schemes (cold backup, warm backup, hot standby)

INF5040, Roman Vitenberg

19

## Primary-backup replication (cold backup)

- Only the primary is active
- Periodically checkpoints its state to backup storage
  - Stable storage or shared storage (SAN)
- When the primary fails, the backup is initiated, it loads the state from storage, and takes over
  - Slow recovery
    - Need to start the backup (run applications, obtain resources, etc.)
    - Either the backup replays the last actions from a log file, or it may miss the last updates since the most recent checkpoint
  - Most resource-efficient
- It is possible to have several backups to survive multiple failures

INF5040, Roman Vitenberg

20

## Primary-backup replication (other than cold backup)

- Warm backup
  - In this case, the backup is (at least) partially alive, so the recovery phase is faster
    - But typically still requires some replaying of last transactions, or losing the last few updates
- Hot standby (leader/follower)
  - The backup is also up, and is constantly updated about the state of the primary
- Local-write scheme
  - The primary migrates between the servers
  - Commonly used in mobile systems

INF5040, Roman Vitenberg

21

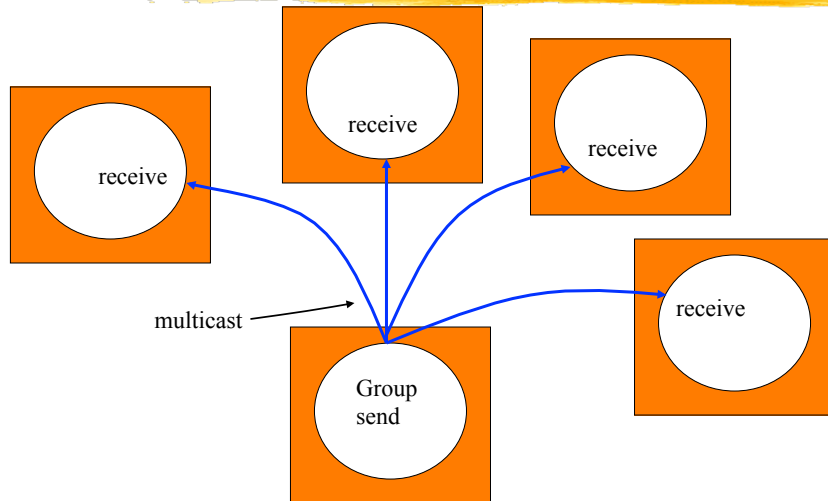
## Quorum-based replication

- Typically used with data-replacing updates
  - Such updates can be performed on old non-updated replicas
- An update is performed on a majority of replicas
- A query is sent to a majority of replicas
  - Replies include both versions and values
  - A client picks a reply with the highest version
  - The replica that sent such a reply is guaranteed to be the most updated one
- The scheme can be generalized
  - Write quorum  $Sw = \{Sw_1, \dots, Sw_n\}$ ,  $Sw_i$  is a set of replicas
  - $\forall i, j, Sw_i \cap Sw_j \neq \emptyset$
  - A client picks  $i$  and performs the update on all replicas in  $Sw_i$
  - Read quorum  $Sr = \{Sr_1, \dots, Sr_n\}$ ,  $\forall i, j, Sr_i \cap Sw_j \neq \emptyset$

INF5040, Roman Vitenberg

22

## From multicast to reliable group communication



INF5040, Roman Vitenberg

23

## From multicast to reliable group communication

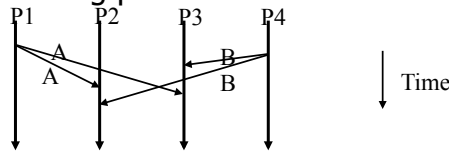
- Group membership service
  - Dynamic maintenance of groups
  - Failure detection
  - Distributes information about changes in the membership
  - Address expansion – an address for multicast to the entire group
- Reliable delivery
  - Acknowledgement of message reception
  - Message retransmissions
- Stability detection
  - Learning when all members of the group have received the message

INF5040, Roman Vitenberg

24

## What is still missing for active replication support?

- Replicas should receive the same events in the same order
- Problem: synchronization between membership changes and message delivery
  - P1 receives m before it learns about a membership change
  - P2 receives m after it learns about a membership change
- Message ordering problem:



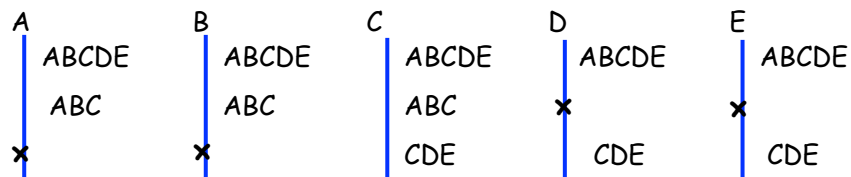
- Group communication provides **view synchrony** & **ordered delivery**

INF5040, Roman Vitenberg

25

## View synchrony

- **View**: epoch of system evolution between two consecutive changes of membership
- The evolution of the system can be seen as a global sequence of views
- Illusion of a static system in each view



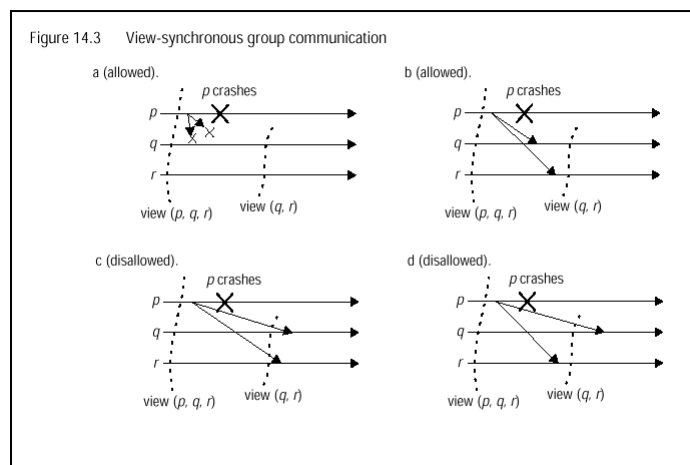
INF5040, Roman Vitenberg

26

# View synchrony

- Synchronization:
  - Processes deliver views and messages in the same sequence of events
  - If two different processes deliver  $m$ , they do it in the same view
- Delivering the same set of messages:
  - If the process  $p$  delivers  $m$  in  $v(g)$  and later delivers  $v(g')$ , then every process  $q$  that delivers both  $v(g)$  and  $v(g')$  delivers  $m$  in  $v(g)$
  - This implies retransmitting missing messages
  - If  $p$  delivers  $m$  in  $v(g)$ , and a process  $q$  does not deliver  $m$  in  $v(g)$ , the next view  $p$  delivers will not include  $q$

# Illustration (from the book)



## Ordered message delivery

- Some rule (binary relation) that establishes that two messages  $m_1$  &  $m_2$  sent in the system are ordered:  $m_1 < m_2$ 
  - Standard relation properties
- Two variants of ordered message delivery
  - Unreliable ordered delivery: if a process delivers  $m_1$  and  $m_2$ , it should deliver  $m_2$  after  $m_1$
  - Reliable ordered delivery: if a process delivers  $m_2$ , it should have already delivered  $m_1$ 
    - Delay message delivery until earlier messages arrive
    - To implement, one may need a lot of space for message buffering

INF5040, Roman Vitenberg

29

## Commonly used orderings

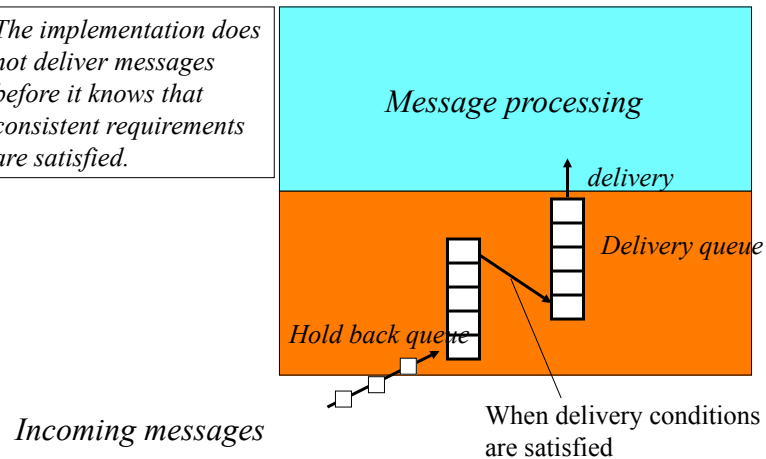
- FIFO
- Causal: two messages are ordered if related by the happen-before relation
  - Many applications require message delivery in an order that preserves cause and effect
    - Publish/subscribe (netnews), email, control systems, root cause determination
- Total: all messages will be received in the same order by all the processes in the group
  - Useful towards implementing the state machine abstraction

INF5040, Roman Vitenberg

30

# Implementing ordered delivery

*The implementation does not deliver messages before it knows that consistent requirements are satisfied.*



INF5040, Roman Vitenberg

31