Data Replication

Why: Scalability

- ***Distribute requests across replicas***
  - Task: load-balancing
  - Web-Server Example:
    - Service and static web-pages replicated
  - load distribution mechanisms:
    - Stateless: round robin, random…
    - Stateful:
      - send next request to the least loaded server most of the time
      - Cluster request types to exploit cache at web-server
- Add new replicas when load increases
  - Task: provisioning

Why: Fast Local Access

- Fast local access in WAN settings
  - Install replicas close to clients
    - Access closest replica

Why: failure handling

- ***Fault-tolerance***
  - If failure occurs "failover" to other replica
    - Usual correctness model: Strong Consistency
    - physical replicas appear as a single logical replica
      - Paxos and Co.
- ***Availability***
  - Service available to client as long as it can access one replica
    - Usual correctness models: Weak/Strong Consistency
    - Availability despite network partitions: Weak Consistency
  - Important for WAN and LAN environments
- Terms not always used clearly

CAP Theorem

- **C**onsistency – **A**vailability – Network **P**artitions

In the Presence of Network Partitions, it is impossible to achieve both (strong) consistency and availability

- Network Partition important in WANs

Data Consistency

- From user perspective there is one logical copy of each data item
- Users submit operations against logical copies
- these operations must be translated into operations against one, some, or all physical copies
- Many existing approaches follow a ROWA(A) approach:
  - Read-one-write-all-(available)
  - Update has to be (eventually) executed at all replicas to keep them consistent
  - Read can be performed at one replica
- More and more: quorums for writes
  - Try all, but a majority good enough (Paxos)
  - Read correctness not clear

Data Consistency

- Strong consistency
  - All available copies of an object have the same value at the end of the execution of an update request
  - Clients always read latest versions of data
  - High overhead
  - Tricky if crashes and network partitions
  - Paxos or 2PC
- All kinds of different levels of weaker consistency
  - Temporal divergence allowed
  - Clients might read stale / wrong data
- Eventual consistency
  - if update activity ceases, then all copies of a data item converge eventually to the same value
  - See homework 1

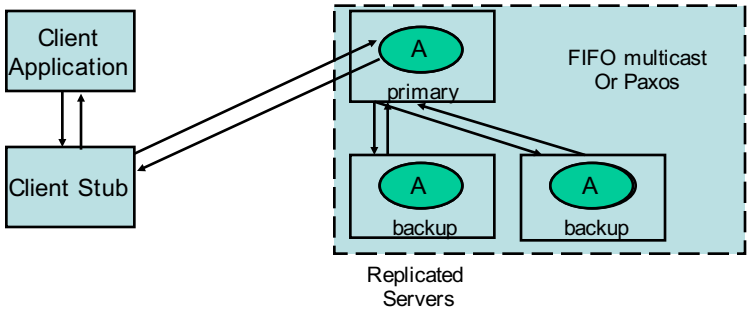
Three Communities

- Distributed Systems Community:
  - Object Replication for Fault-tolerance
    - Paxos/Total-Order Coordination
- Database Community:
  - Replication for Fault-tolerance
    - 2PC
  - Replication for Fast Local Access
    - Weak Consistency
  - Replication for Scalability
    - Strong and Weak Consistency Solutions
- Replication in Storage Systems
    - Availability / Fault-Tolerance

Object Replication / State Machine Replication: Fault-tolerance

- A large body of research
- Implemented in distributed computing environments / data stores
- Two realizations
  - Group communication system
    - Handles total order multicast
    - Reliability (uniform)
    - Group maintenance, failure detection, etc.
  - Paxos
- Correctness
  - Replicated System should behave as non-replicated system that has no failures
  - Each request has exactly one "successful" execution
  - Client receives exactly one response (failure transparency)
  - **strong data consistency**: data copies are consistent at the end of request execution
- passive (primary backup) replication vs. active replication

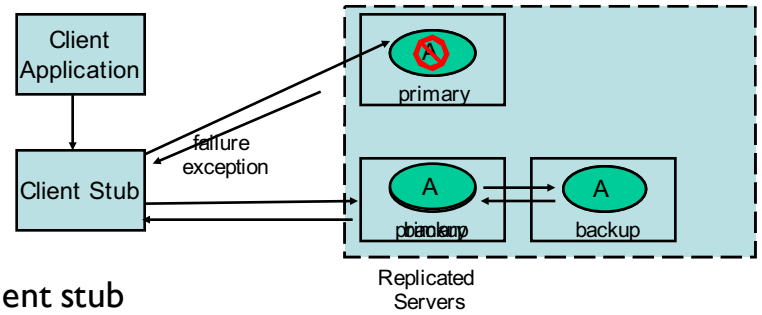
Passive Replication



1. The client sends the request to the primary..
2. The primary executes the request.
3. The primary runs an agreement protocol with the backups
   1. Paxos
   2. Uniform reliable FIFO
4. The primary sends the answer to the client.

Passive Replication Failures

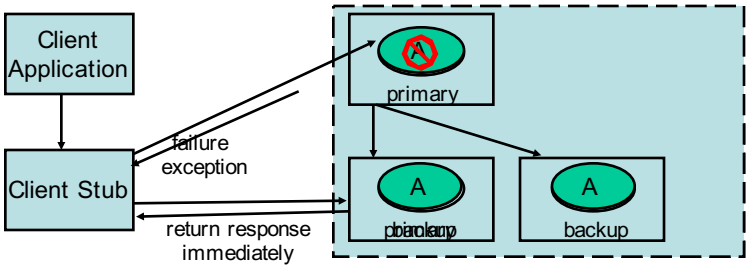
- Before Update Propagation
  - reexecute on new primary



- Client stub
  - Detect failure
  - Know new primary
  - reexecute on new primary

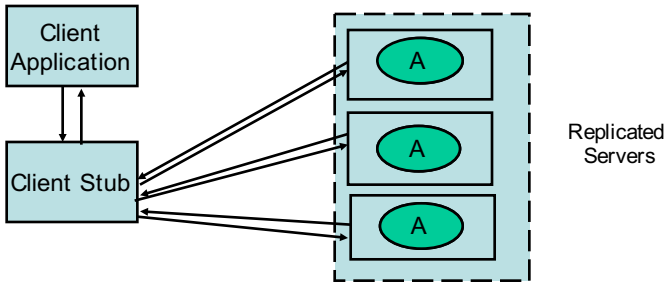
Passive Replication Failures

- After Update Propagation: return result immediately



- GCS / Paxos wrapper guarantees
  - all or none of the backups have state changes
  - all have same view of who is primary/backup
- Avoiding wrong reexecution
  - request must have unique ids
  - primary must send response with state changes
  - backups must keep responses

Active Replication



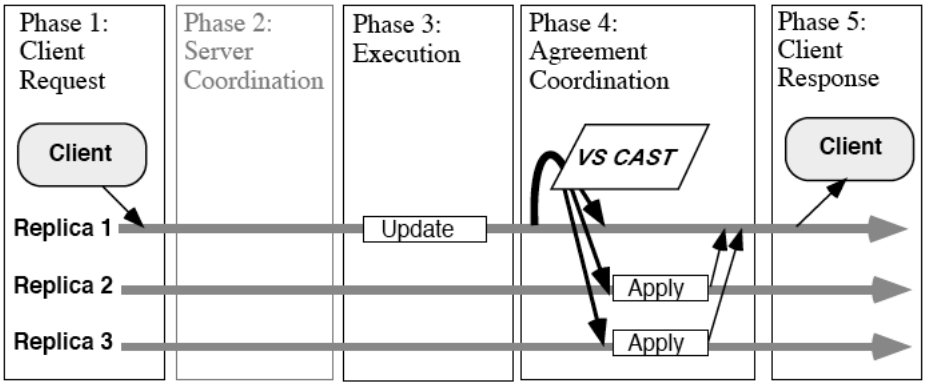
- Client multicasts request to all
- All execute
- Crash Failures: return first response
- Byzantine Failure: take majority response

- Total order multicast

Active vs. Passive Replication

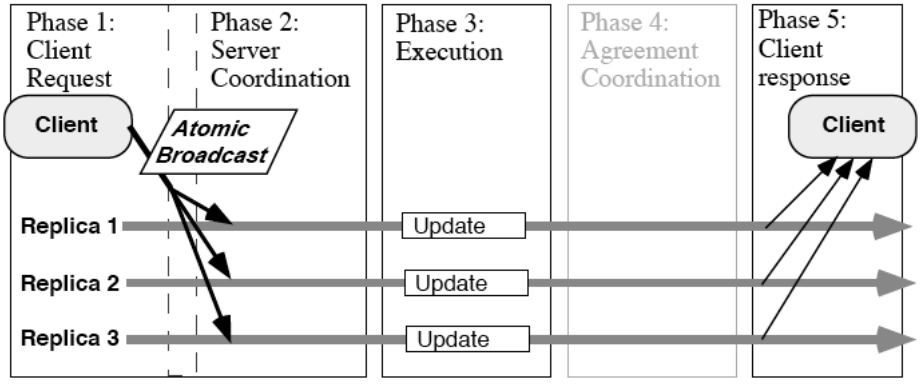
- Determinism
- Execution during normal processing
  - Communication Overhead
  - CPU overhead
  - Complexity
- Termination protocol
- Failure types
- Write / read

Passive Replication



1. The client sends the request to the primary.
2. There is no initial coordination.
3. The primary executes the request.
4. The primary coordinates with the other replicas by sending the update information to the backups.
5. The primary sends the answer to the client.

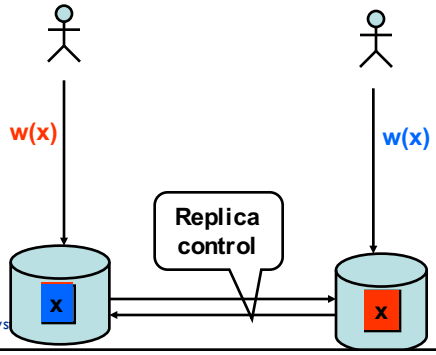
Active Replication



1. The client multicasts request to the servers with total order
2. **S**erver coordination is given by the total order property
3. All replicas execute the request in the order they are delivered.
4. No coordination necessary (Assumption: determinism)
   - All replicas produce the same result
5. All replicas send result to the client; client waits for the first answer

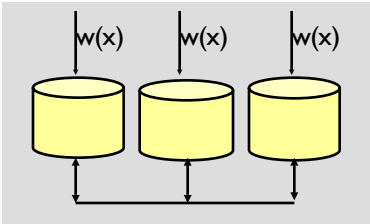
Replication for Performance/Availability: Database and Data Storage Community

- Keep copies consistent: **replica control**
- Different scope than solutions developed for object replication for fault-tolerance
- Transactional / per object
- Distinguish between Reads and Writes

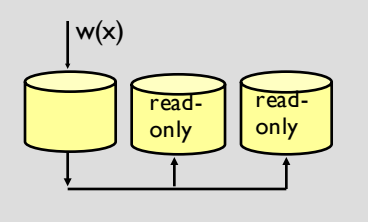


Where can updates be submitted?

- *Update Anywhere:*
  - Update transactions can be submitted to any ONE site
  - Site forwards updates to other sites



- *Primary Copy:*
  - Update transactions can only execute at the primary copy (master)
  - Primary forwards updates to secondaries

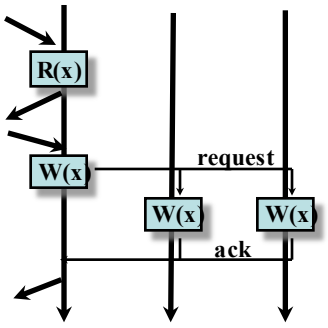


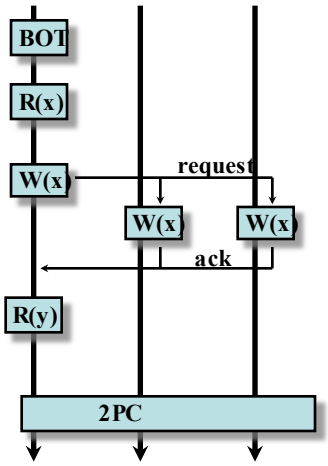
Primary vs. Update anywhere / Update everywhere / multi-master

- Less coordination necessary / optimizations are easier
- Transparency and flexibility loss:
  - Clients must know primary to submit update transactions
  - Have to distinguish update from read-only transactions
- In WAN: primary only close to subset of clients
- Primary is single point of failure and potential bottleneck
- Multiple primaries
  - Each object can have other primary

When to propagate

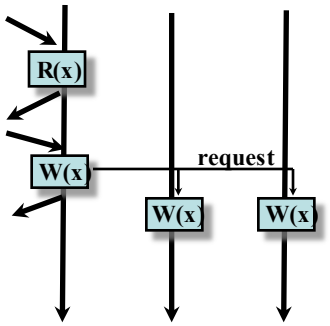
- *Eager:*
  - within the boundaries of the request/transaction
  - Transactions terminate usually with 2PC

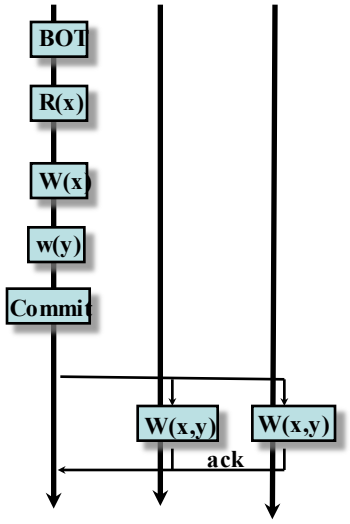




When to propagate

- *Lazy:*
  - After terminating request execution / transaction



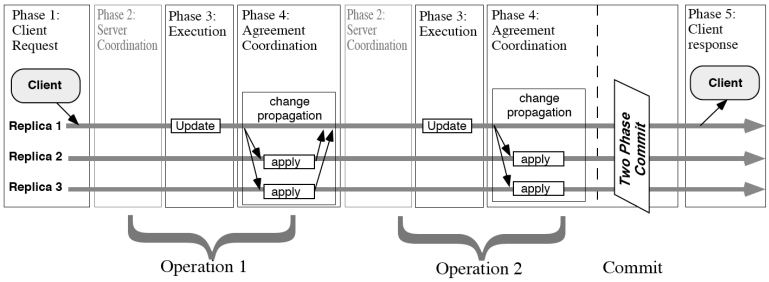


Lazy vs. Eager

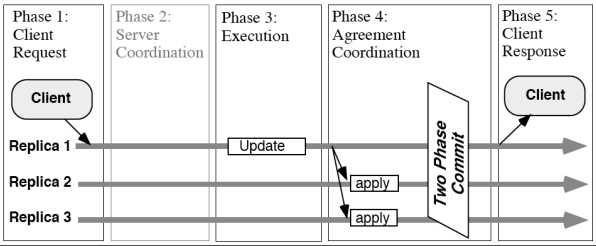
- Lazy primary copy: stale reads
- Lazy update anywhere: inconsistencies and reconciliation
- No communication within transaction response time
- Possible transaction loss in case of crash
- Optimizations for update propagation possible

Eager Primary Copy

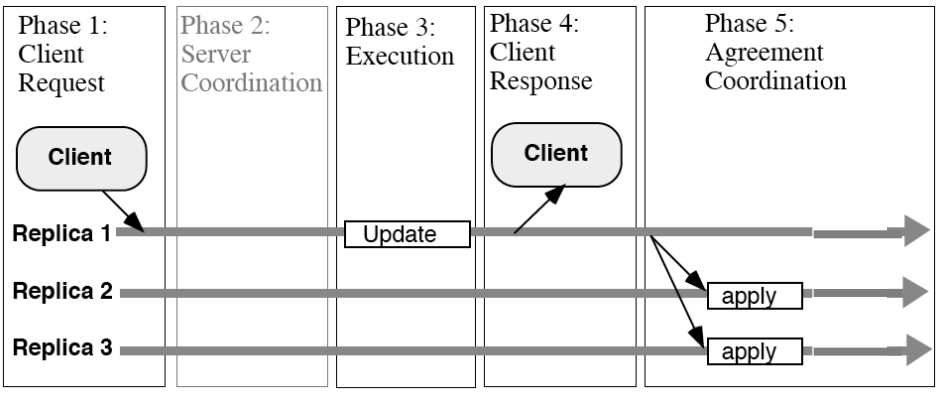
- Update propagation after each update



- Update propagation at end of transaction



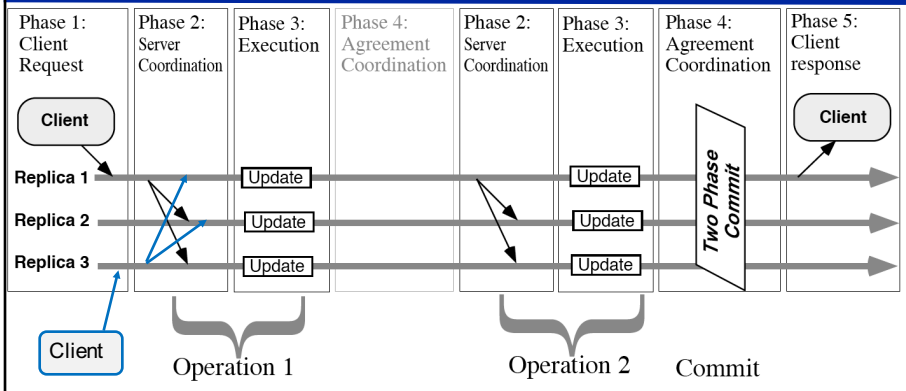
Lazy Primary Copy



Lazy / Primary Copy

- Primary Copy:
  - Upon read: read locally and return to user
  - Upon write: write locally and return to user
  - Upon commit/abort: terminate locally
  - Sometime after commit: multicast changed objects in a single message to other sites (in FIFO)
- Secondary copy:
  - Upon read: read locally
  - Upon message from primary copy: install all changes (FIFO)
  - Upon write from client: refuse (writing clients must submit to primary copy)
  - Upon commit/abort request (only for read-only txn): local commit
  - Note: transaction might write local data that is NOT replicated or for which the site is the primary copy
- Only local deadlocks
- Note: existing systems allow different objects to have different primary copies
  - A transaction that wants to write X (primary copy is site S1) and Y (primary copy on site S2) is usually disallowed

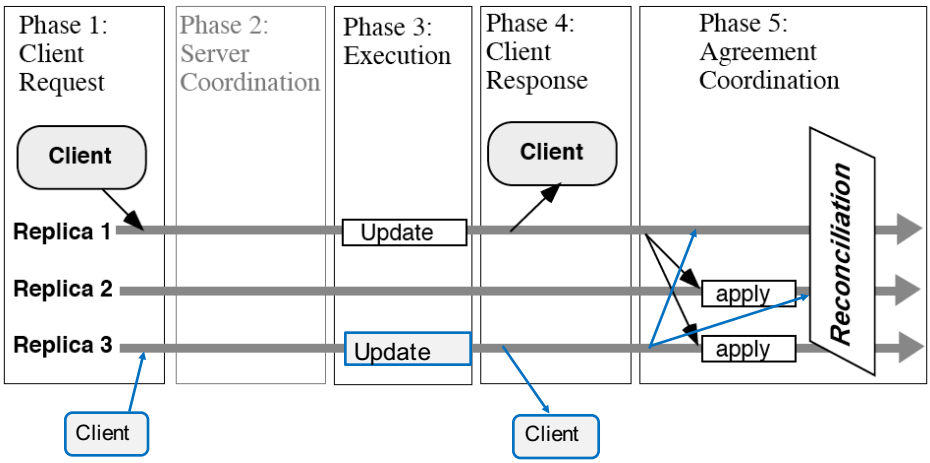
Eager / Update Anywhere



Eager / Update Anywhere with distributed locking

- Upon read: request local read lock and read locally and return value to user
- Upon write from client: request local write lock and write locally, multicast write request to other sites.
- Upon write from other site: request local write lock, write locally, and send ok back to initiating site
- Upon receiving ok from all other sites, return ok to the user
- Upon commit request: run 2PC to ensure that all have really installed the changes.
- Upon abort: abort and inform other sites about abort
- Deadlocks might occur.

Lazy Update Anywhere



Lazy / Update Anywhere

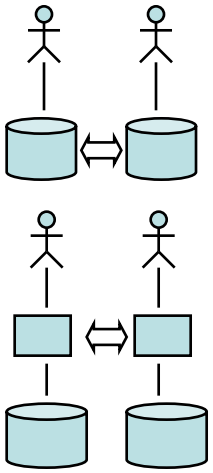
- Only after commit/ok to user multicast changes to other replicas
- Conflict detection
- Conflict resolution: eventual consistency
  - for numeric types (or types with comparison):
    - average:
    - minimum/maximum:
    - additive:
  - discard new value, overwrite old value
  - Site priority
  - value priority
  - earliest/latest timestamp

Combinations

- Primary copy / eager:
  - Similar to passive replication ➔
  - Widely used for fault-tolerance
    - e.g. DB2 high availability solution
- Update anywhere / eager:
  - Databases: Many middleware-based systems
  - NoSQL data stores: Often concept of quorum
- Primary copy / lazy:
  - Used for fault-tolerance with weaker guarantees
  - Used for systems with different sets of update / read workload (e.g., data warehousing
- Update anywhere / lazy:
  - For availability
  - For disconnected operations
  - For fast local access
  - Assume low conflict rate

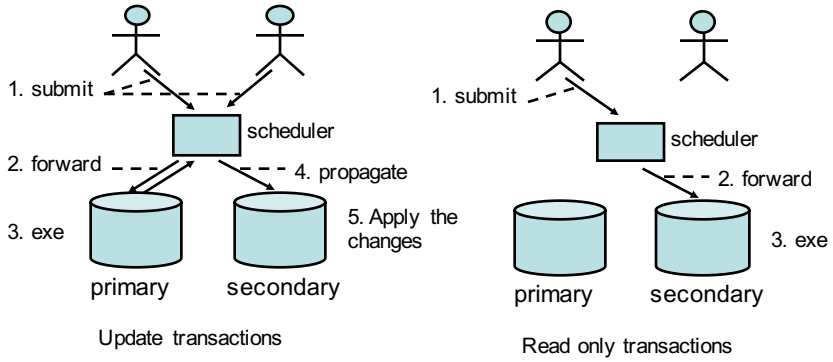
Architectures

- Kernel-based approach
- Middleware-based approach
  - Advantages
    - Modular
    - Do not need access to data store code
    - Reusability
  - Disadvantages
    - No access to concurrency control information in the data store



Middleware Primary Copy Lazy DB Replication

- (e.g. Ganymed)



Eager Protocols and Failures

- So far: read-one-write-all protocols (ROWA)
- Site failures:
  - Read-one-write-all-AVAILABLE (ROWAA)
- Quorum protocols

Quorums

- Failure Model
  - Network partitions are possible, message loss is possible
- Quorum: set of replicas
  - Each read operation must contact a read quorum of replicas
  - Each write operation must contact a write quorum of replicas
  - Guarantee: any two write-quorums and combination of read/write quorum intersect

Simple Majority

- Read quorum: any n/2 replicas
- Write quorum: any n/2 + 1 replicas