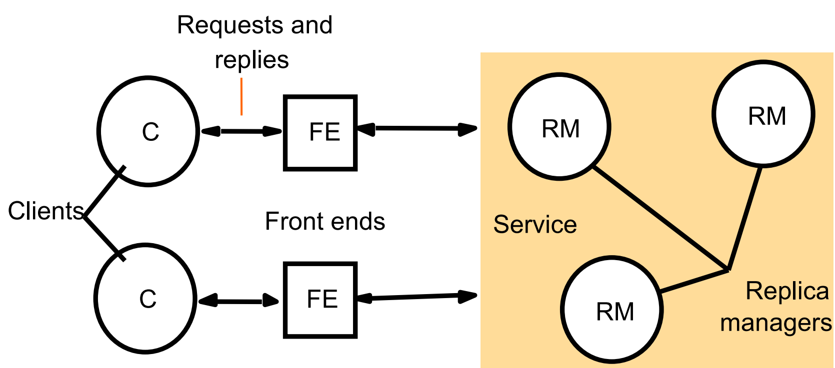
Replication and Consistency in distributed systems (cont'd)

Distributed Software Systems

A basic architectural model for the management of replicated data

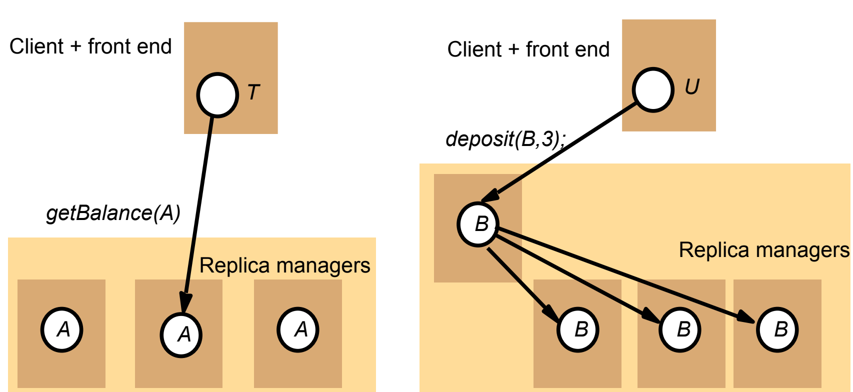


System model

Five phases in performing a request

- Front end issues the request
  - Either sent to a single replica or multicast to all replica mgrs.
- Coordination
  - Replica managers coordinate in preparation for the execution of the request, I.e. agree if request is to be performed and the ordering of the request relative to others
    - FIFO ordering, Causal ordering, Total ordering
- Execution
  - Perhaps tentative
- Agreement
  - Reach consensus on effect of the request, e.g. agree to commit or abort in a transactional system
- Response

Transactions on replicated data



One copy serializability

- Replicated transactional service
  - Each replica manager provides concurrency control and recovery of its own data items in the same way as it would for non-replicated data
- Effects of transactions performed by various clients on replicated data items are the same as if they had been performed one at a time on a single data item
- Additional complications: failures, network partitions
  - Failures should be serialized wrt transactions, i.e. any failure observed by a transaction must appear to have happened before a transaction started

Replication Schemes

- Primary Copy
- Read one – Write All
  - Cannot handle network partitions
- Schemes that can handle network partitions
  - Available copies with validation
  - Quorum consensus
  - Virtual Partition

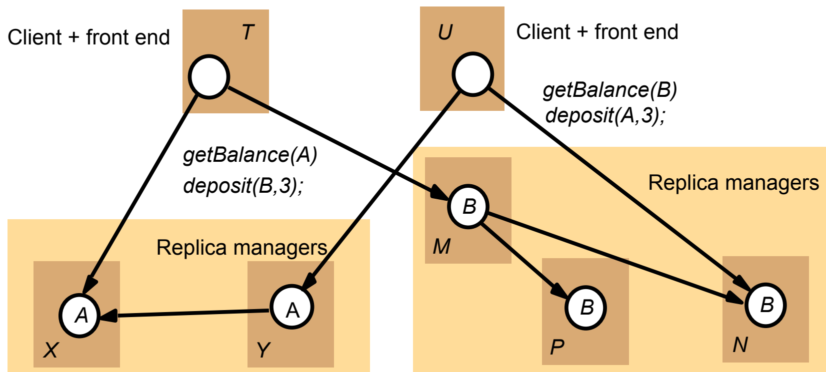
Replication Schemes cont'd

- Read-one write-all
  - Each write operation sets a write lock at each replica manager
  - Each read sets a read lock at one replica manager
- Two phase commit
  - Two-level nested transaction
    - Coordinator -> Workers
    - If either coordinator or worker is a replica manager, it has to communicate with replica managers
- Primary copy replication
  - ALL client requests are directed to a single primary server

Available copies replication

- Can handle some replica managers are unavailable because they have failed or communication failure
- Reads can be performed by any available replica manager but writes must be performed by all available replica managers
- Normal case is like read one/write all
  - *As long as the set of available replica managers does not change during a transaction*

Available copies



Available copies replication

- Failure case
  - One copy serializability requires that failures and recovery be serialized wrt transactions
  - This is not achieved when different transactions make conflicting failure observations
  - Example shows local concurrency control not enough
  - Additional concurrency control procedure (called *local validation*) has to be performed to ensure correctness
- Available copies with local validation assumes no network partition - i.e. functioning replica managers can communicate with one another

Local validation - example

- Assume X fails just after T has performed GetBalance and N fails just after U has performed GetBalance
- Assume X and N fail before T & U have performed their Deposit operations
  - T's Deposit will be performed at M & P while U's Deposit will be performed at Y
  - Concurrency control on A at X does not prevent U from updating A at Y; similarly concurrency control on B at N does not prevent Y from updating B at M & P
  - Local concurrency control not enough!

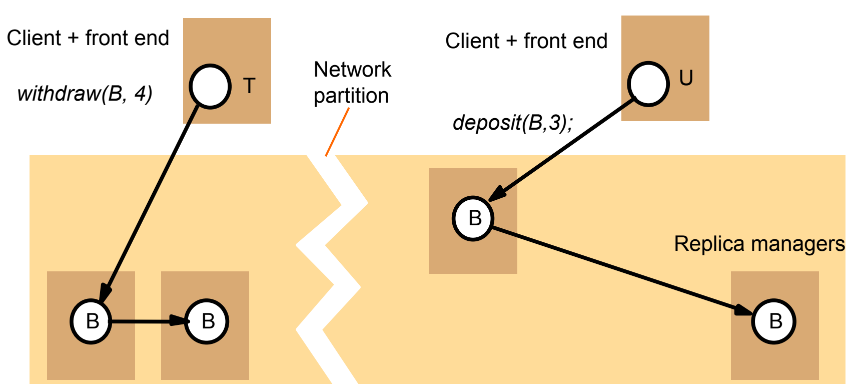
Local validation cont'd

- T has read from an item at X, so X's failure must be *after* T.
- T observes the failure of N, so N's failure must be *before* T
  - N fails -> T reads A at X; T writes B at M & P -> T commits -> X fails
  - Similarly, we can argue:
    X fails -> U reads B at N; U writes A at Y -> U commits -> N fails

Local validation cont'd

- Local validation ensures such incompatible sequences cannot both occur
- Before a transaction commits it checks for failures (and recoveries) of replica managers of data items it has accessed
- In example, if T validates before U, T would check that N is still unavailable and X,M, P are available. If so, it can commit
- U's validation would fail because N has already failed.

Network partition



Handling Network Partitions

- Network partitions separate replica managers into two or more subgroups, in such a way that the members of a subgroup can communicate with one another but members of different subgroups cannot communicate
- Optimistic approaches
  - Available copies with validation
- Pessimistic approaches
  - Quorum consensus

Available Copies With Validation

- Available copies algorithm applied within each partition
  - Maintains availability for Read operations
- When partition is repaired, possibly conflicting transactions in separate partitions are validated
  - The effects of a committed transaction that is now aborted on validation will have to be undone
    - Only feasible for applications where such compensating actions can be taken

Available copies with validation cont'd

- Validation
  - Version vectors (Write-Write conflicts)
  - Precedence graphs (each partition maintains a log of data items affected by the Read and Write operations of transactions
  - Log used to construct precedence graph whose nodes are transactions and whose edges represent conflicts between Read and Write operations
    - No cycles in graph corresponding to each partition
  - If there are cycles in graph, validation fails

Quorum consensus

- A quorum is a subgroup of replica managers whose size gives it the right to carry out operations
- Majority voting one instance of a quorum consensus scheme
  - R + W > total number of votes in group
  - W > half the total votes
  - Ensures that each read quorum intersects a write quorum, and two write quora will intersect
- Each replica has a version number that is used to detect if the replica is up to date.

Gifford's quorum consensus examples

		Example 1	*Example 2*	*Example 3*
Latency	Replica 1	75	75	75
(milliseconds)	Replica 2	65	100	750
	Replica 3	65	750	750
Voting	Replica 1	1	2	1
configuration	Replica 2	0	1	1
	Replica 3	0	1	1
Quorum	*R*	1	2	1
sizes	*W*	1	3	3

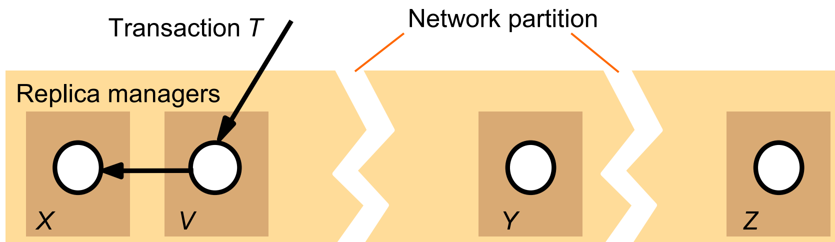
Derived performance of file suite:

Read	Latency	65	75	75
	Blocking probability	0.01	0.0002	0.000001
Write	Latency	75	100	750
	Blocking probability	0.01	0.0101	0.03

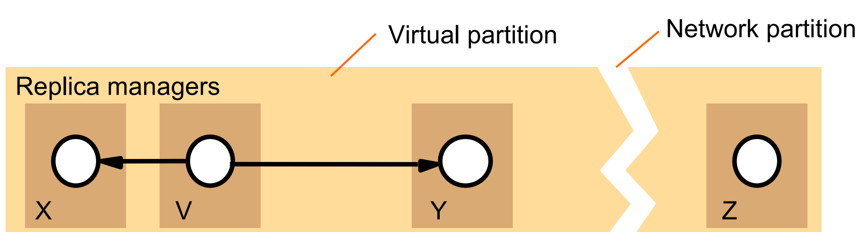
Virtual Partitions scheme

- Combines available copies and quorum consensus
- Virtual partition = set of replica managers that have a read and write quorum
- If a virtual partition can be formed, available copies is used
  - Improves performance of Reads
- If a failure occurs, and virtual partition changes during a transaction, it is aborted
- Have to ensure virtual partitions do not overlap

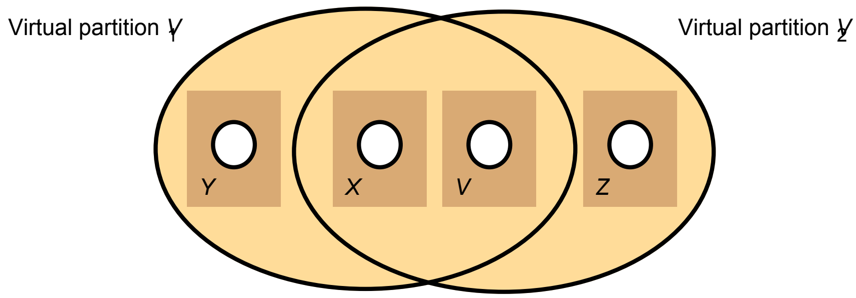
Two network partitions



Virtual partition



Two overlapping virtual partitions



Creating a virtual partition

Phase 1:

- The initiator sends a *Join* request to each potential member. The argument of *Join* is a proposed logical timestamp for the new virtual partition.
- When a replica manager receives a *Join* request, it compares the proposed logical timestamp with that of its current virtual partition.
  - If the proposed logical timestamp is greater it agrees to join and replies *Yes*;
  - If it is less, it refuses to join and replies *No*.

Phase 2:

- If the initiator has received sufficient *Yes* replies to have read and write quora, it may complete the creation of the new virtual partition by sending a *Confirmation* message to the sites that agreed to join. The creation timestamp and list of actual members are sent as arguments.
- Replica managers receiving the *Confirmation* message join the new virtual partition and record its creation timestamp and list of actual members.

CAP Conjecture

- Is it possible to achieve consistency, availability, and partition tolerance?

These slides are borrowed from lectures by Prof. Ion Stoica & Scott Shenker (UC, Berkeley)

CAP conjecture attributed to Prof. Eric Brewer (UC Berkeley)

Recent theoretical results by Prof. Nancy Lynch et al (MIT) prove the conjecture

A Clash of Cultures

- Classic distributed systems: focused on ACID semantics
  - A: Atomic
  - C: Consistent
  - I: Isolated
  - D: Durable
- Modern Internet systems: focused on BASE
  - Basically Available
  - Soft-state (or scalable)
  - Eventually consistent

ACID vs BASE

ACID

- Strong consistency for transactions highest priority
- Availability less important
- Pessimistic
- Rigorous analysis
- Complex mechanisms

BASE

- Availability and scaling highest priorities
- Weak consistency
- Optimistic
- Best effort
- Simple and fast

Why the Divide?

- What goals might you want from a shared-data system?
  - C, A, P
- **Strong Consistency**: all clients see the same view, even in the presence of updates
- **High Availability**: all clients can find some replica of the data, even in the presence of failures
- **Partition-tolerance**: the system properties hold even when the system is partitioned

CAP Conjecture (Brewer)

- You can only have two out of these three properties
- The choice of which feature to discard determines the nature of your system

Consistency and Availability

- Comment:
  - Providing transactional semantics requires all nodes to be in contact with each other
- Examples:
  - Single-site and clustered databases
  - Other cluster-based designs
- Typical Features:
  - Two-phase commit
  - Cache invalidation protocols
  - Classic DS style

Consistency and Partition-Tolerance

- Comment:
  - If one is willing to tolerate system-wide blocking, then can provide consistency even when there are temporary partitions
- Examples:
  - Distributed databases
  - Distributed locking
  - Quorum (majority) protocols
- Typical Features:
  - Pessimistic locking
  - Minority partitions unavailable
  - Also common DS style
    - Voting vs primary replicas

Partition-Tolerance and Availability

- Comment:
  - Once consistency is sacrificed, life is easy….
- Examples:
  - DNS
  - Web caches
  - Coda
  - Bayou
- Typical Features:
  - TTLs and lease cache management
  - Optimistic updating with conflict resolution

Techniques

- Expiration-based caching: AP
- Quorum/majority algorithms: PC
- Two-phase commit: AC