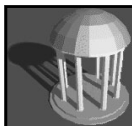


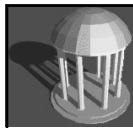
COMP 190 -- Enterprise Systems

Replicated Data

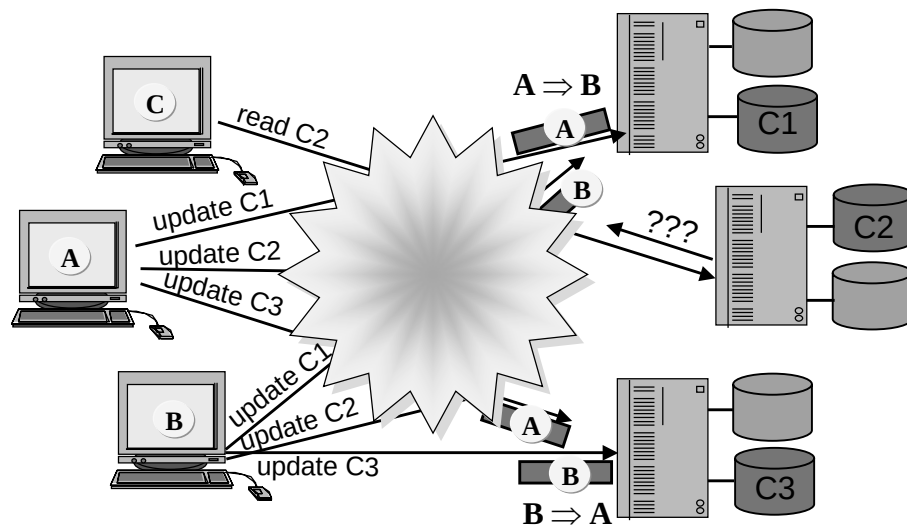


Why Replicate Data?

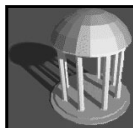
- ◆ High Availability
 - ◆ System Pairs
 - ◆ Continue operations in case of network partition
- ◆ Performance
 - ◆ Multiple locations to minimize network latency
 - ◆ Multiple locations for parallel access



Updates to Replicated Data: Concurrent Updates & Reads



3

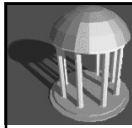


Do Concurrent Updates and Reads to Replicas Really Matter?

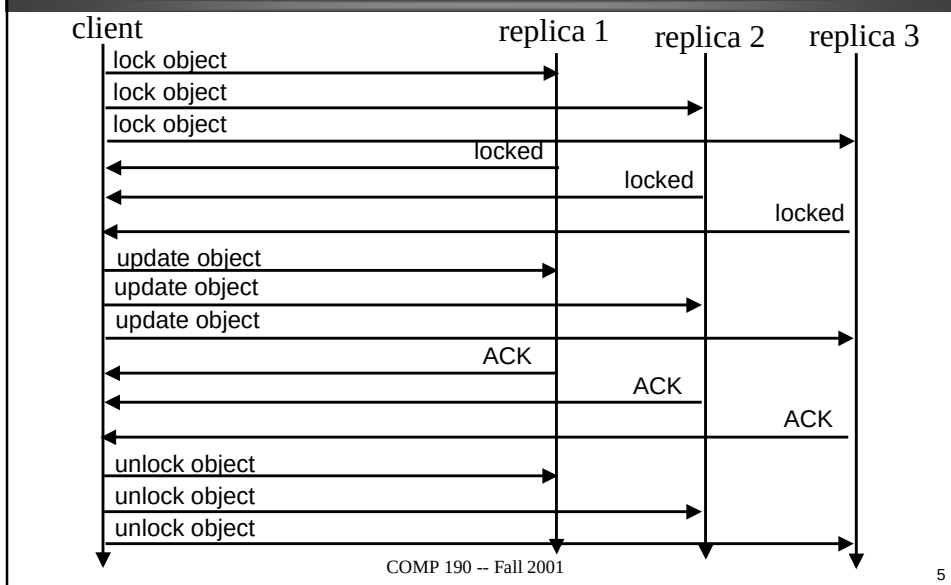
- ◆ It Depends On the Application!
 - ◆ For financial records (your bank account!), they certainly do matter
 - ◆ For maintaining critical system state between members of a system-pair, they certainly do matter
 - ◆ For ordinary files, there may be a range of choices
 - e.g., NFS vs Sprite vs Andrew
- ◆ As long as there are NO failures, protocols to implement even strong constraints on concurrent operations are straightforward (e.g., distributed locking).

COMP 190 -- Fall 2001

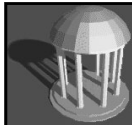
4



Consistent Replica Update



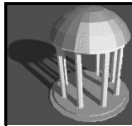
5



Strict Consistency: “One Copy” Semantics

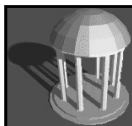
- ◆ Informally, strict consistency for replicated data means:
 - ◆ there exists **exactly one** current value at **all** replicas and that value is returned for a read of **any** replica
 - ◆ the current value “makes sense” (does not violate any application semantics defined for the data)
 - normally means **all** updates must be performed and **in the same order** at all replicas
- ◆ Logically, the ensemble of replicas yields the same results for concurrent updates and reads as a single copy.

6



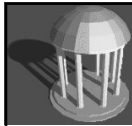
Access to Replicated Data: Issues

- ◆ What availability is required?
- ◆ What performance is required?
- ◆ What correctness and consistency properties are required?
- ◆ What are the types and frequencies of failures?
- ◆ Tradeoff:
 - ◆ Tighter correctness and consistency requirements imply lower performance and/or availability

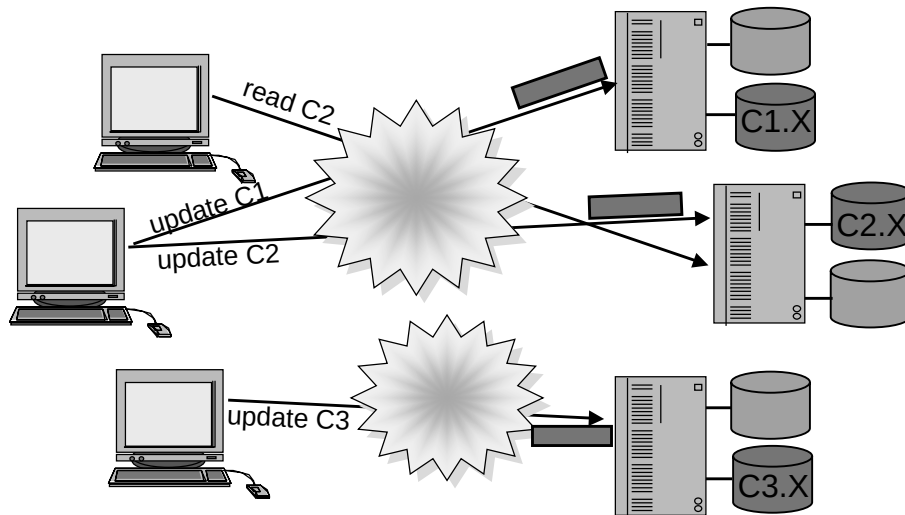


Availability (and Performance)

- ◆ If a client can read ANY replica, availability is dramatically improved.
 - ◆ If a client can read any “nearby” replica, performance is also improved.
 - ◆ “Read-Any-Update-All” replication model
- ◆ In some applications (e.g., mobile users), the ability to update at ANY replica (server) is also important for availability (and performance).
 - ◆ “Read-Any-Update-Any” replication model

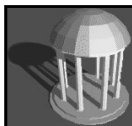


What if the network is partitioned?

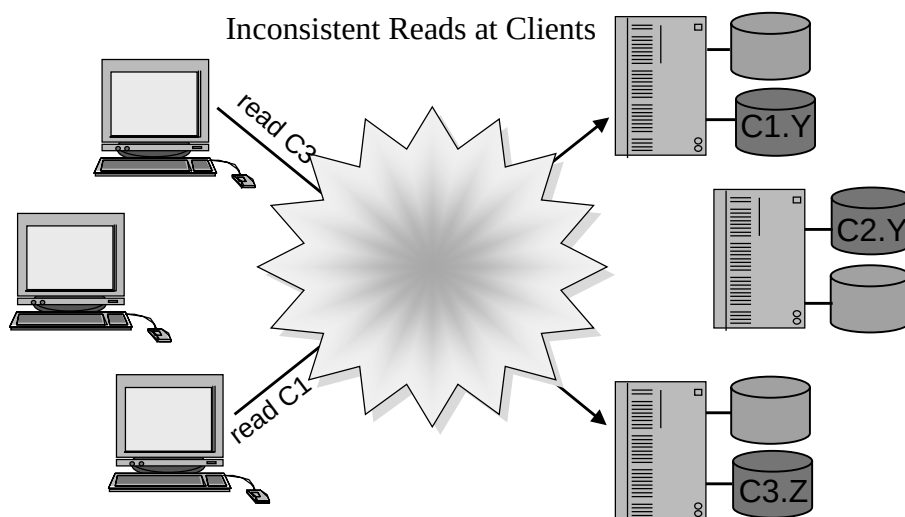


COMP 190 -- Fall 2001

9

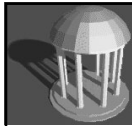


What if the partition is repaired?



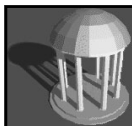
COMP 190 -- Fall 2001

10



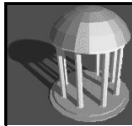
What If Some Replica(s) Cannot Be Reached?

- ◆ Nature and extent of failure cannot be determined reliably by any arbitrary node.
 - ◆ Can identify “live” sites
 - e.g., those that respond to some form of “ping”
 - ◆ Cannot determine the state of non-responsive sites
 - Crashed?
 - Network failure?
 - Live but completely isolated?
 - Live and in another partition with other live sites?
 - Slow process and/or congested link (beyond ping timeout) -- mistaken for failure?



Strict Consistency vs Network Partitions

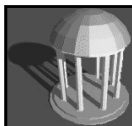
- ◆ During partitioned operations, two aspects of consistency must be maintained
 - ◆ Each partition must maintain consistency among replicas at all accessible sites in the partition
 - Easy -- use algorithms that provide one-copy semantics when all components can communicate
 - ◆ Each partition must ensure that actions taken in that partition do not violate the application’s semantics so replicas are consistent across all partitions
 - Hard -- must avoid (or detect and correct) missing updates, conflicting updates, or incorrect ordering of updates



Strict Consistency – Pessimistic Strategies

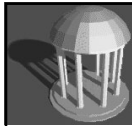
◆ A *Pessimistic* Strategy --

- ◆ Prevents any possible consistency problems
- ◆ Suspends reads and updates in some partitions
 - Reduces availability to clients in those partitions
 - Increases latency for updates in normal (non-partitioned) operations
- ◆ Guarantees one-copy semantics for all allowed reads and updates



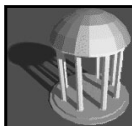
Quorum-based Replicas

- ◆ Each replica (site) is assigned a weighting factor (number of “votes”).
- ◆ To read or update, the requestor must collect a sufficient number of “votes” from *reachable* replicas to form the quorum defined for that operation.
- ◆ Let:
 - ◆ r = votes needed for a read quorum
 - ◆ u = votes needed for an update quorum
 - ◆ v = total number of votes assigned to all replicas



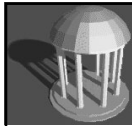
Quorum-based Replicas

- ◆ Constraints: Every pair of quorums ($\{\text{read}, \text{update}\}$ or $\{\text{update}, \text{update}\}$) are guaranteed to have at least one common copy, so each quorum has at least one up-to-date copy.
 - ◆ (1) $r + u > v$
 - ◆ (2) $u > v/2$
- ◆ Constraint (1) guarantees that a replica cannot be read in one partition and updated in another.
- ◆ Constraint (2) guarantees that a replica cannot be updated in more than one partition
- ◆ Each replica is assigned a version number that is incremented each time it is updated.



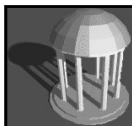
Quorum Algorithm -- Read

- ◆ Send the read request to all reachable replicas which should respond with the data, its current version number, and their assigned votes.
- ◆ Compute the sum of the returned votes
 - ◆ If $\text{sum} < r$, the read fails
 - ◆ If $\text{sum} \geq r$, accept the data from any replica with the highest version number found



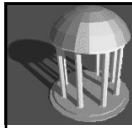
Quorum Algorithm -- Update

- ◆ Send the update request to all reachable replicas which should respond with their copy of the data, its current version, and their assigned votes.
- ◆ Compute the sum of returned votes *from replicas with the highest version number found*:
 - ◆ If $\text{sum} \geq u$ ↓
Must ensure that either all replicas in the quorum are updated or none are (make this a transaction).
 - update all replicas needed to make a quorum (which increments their version numbers)
 - propagate the update to all other available replicas with the same (highest) version number
 - completely replace other available but out-of-date replicas with an updated copy.

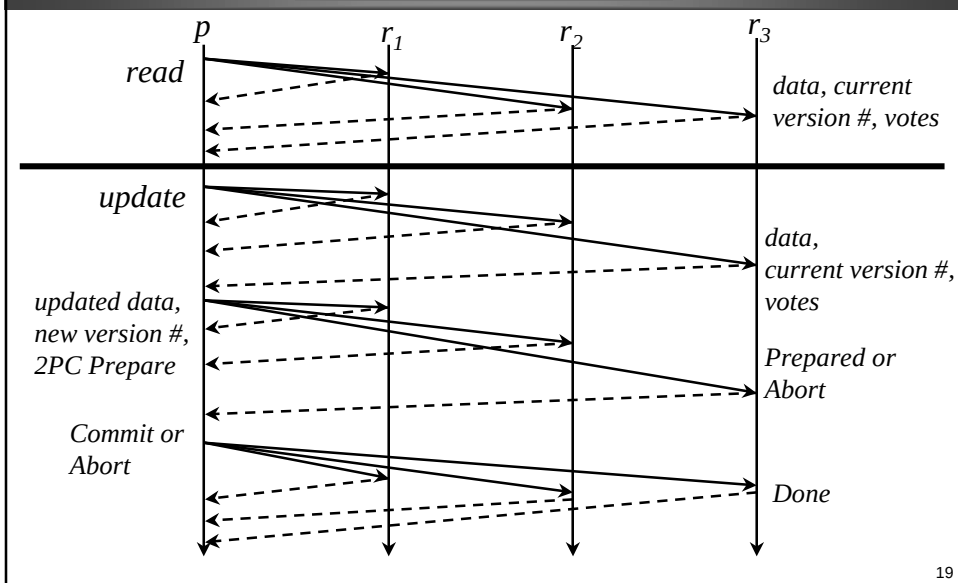


Quorum Algorithm -- Update (cont.)

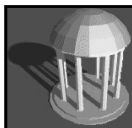
- ◆ If $\text{sum} < u$,
 - Compute the sum of votes returned from reachable replicas, *this time ignoring the reported version numbers*.
 - If $\text{sum} < u$, the update is prohibited and fails
 - If $\text{sum} \geq u$,
 - Replace available but out-of-date replicas with a replica having the highest version number found.
 - Restart the update operation



Quorum Read/Write with $R=U=3$

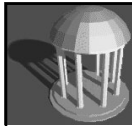


19



Notes on Quorum Algorithms

- ◆ Assigning differing weights (votes) to each replica and requiring different quorum sizes produces a family of replica strategies:
 - ◆ Each replica has 1 vote with $r=1$ and $u=v$ is the “read-one, update-all” strategy (updates not allowed anywhere if there is a partition).
 - ◆ Each replica has 1 vote with $r=u=\text{floor}(v/2) + 1$ is the “majority consensus” strategy (reads and updates can take place only in a partition with a majority of the replicas).

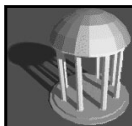


Relaxing Strict Consistency -- *Optimistic Strategies*

- ◆ Allow read and/or update access during partitions but *detect and correct* when connectivity restored.
- ◆ Maintain a version number at each replica that is incremented each time the replica is updated.
- ◆ Extend the version number to a *version vector* maintained at each replica to summarize the *update history at all replicas*.
- ◆ Used to detect conflicting update histories at different replicas.

COMP 190 -- Fall 2001

21

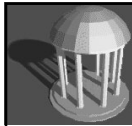


Weak Consistency

- ◆ Many applications require only approximate consistency among replicas
 - ◆ personal information management (e.g., address books, calendars, to-do lists, etc.)
 - ◆ asynchronous collaboration tools (e.g., workflow, Lotus Notes, etc.)
- ◆ For performance and availability, the “read-any, update-any” model is most appropriate, especially in “weakly connected” environments.

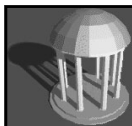
COMP 190 -- Fall 2001

22



Weak Consistency

- ◆ System guarantees that all replicas reach *eventual consistency* (number and order of updates)
 - ◆ all servers *eventually* receive all updates
 - ◆ updates are *eventually* applied in the same, well-defined order at all servers
 - ◆ two servers *holding the same set of updates* will produce the same result for a query from a client
 - it is quite likely that any two servers will not hold the same set of updates at an arbitrary point in time
 - applications must be tolerant of this transient inconsistency that can be observed by multiple queries



Update Propagation

- ◆ Updates are propagated by an “*epidemic, anti-entropy algorithm*” to sites that may not have received them.
 - ◆ epidemic implies “spreading rapidly by contact”
 - ◆ anti-entropy implies “bringing order out of chaos”
- ◆ Updates propagate by *pair-wise contact* between sites (one-way, sender to receiver).
- ◆ Sites can use any of a variety of randomized and/or deterministic methods to choose the next target of “infection” and method (push, pull) of propagation.
- ◆ Propagation time is $O(\log_2 N)$ where N is the number of sites

