CS 550:
Advanced Operating Systems

Consistency
Part 2

Ioan Raicu
Computer Science Department
Illinois Institute of Technology

CS 550
Advanced Operating Systems
March 8th, 2011

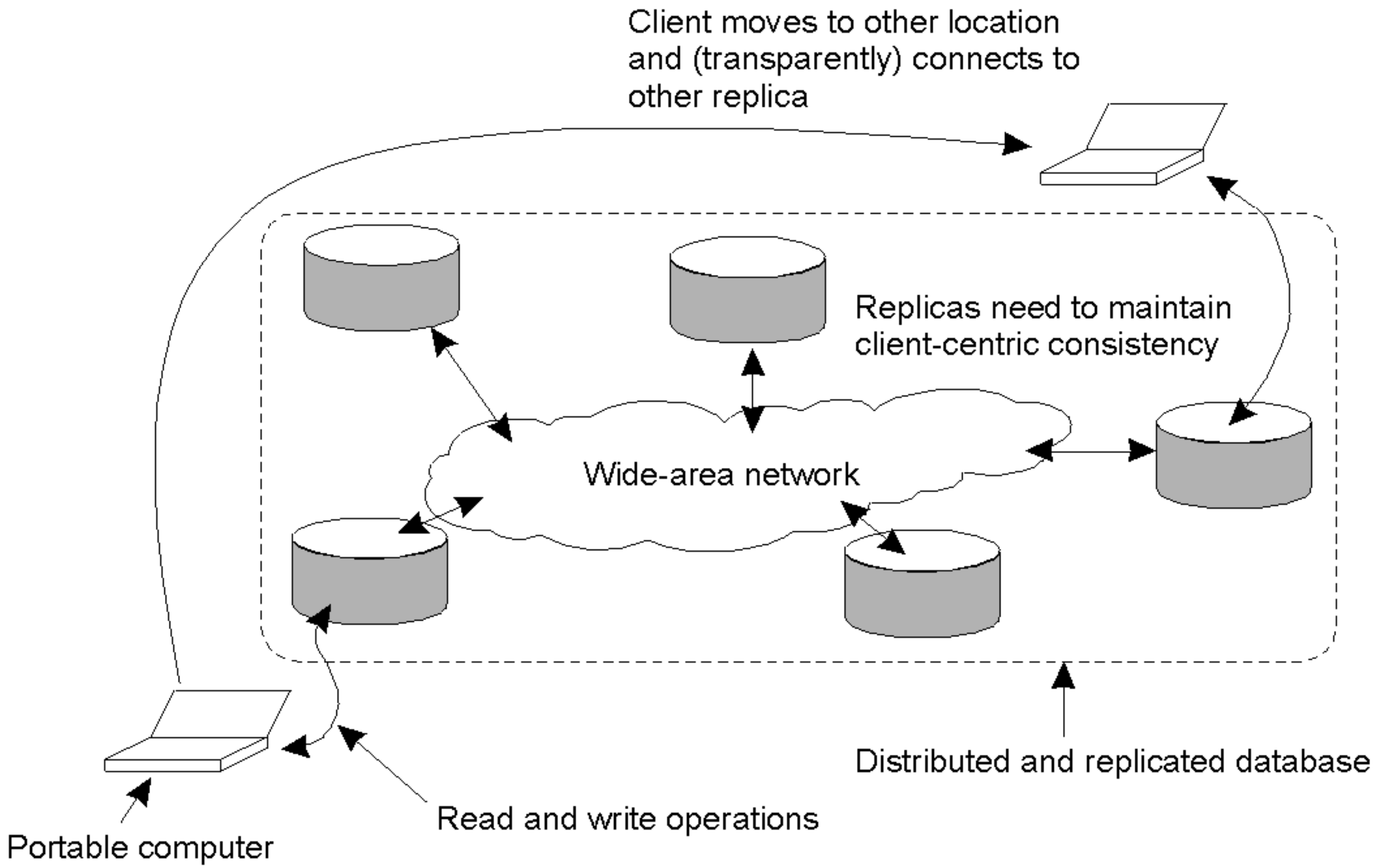
Eventual Consistency

- Many systems: one or few processes perform updates
  - How frequently should these updates be made available to other read-only processes?
- Examples:
  - DNS:
    - Single naming authority per domain
    - Only naming authority allowed updates (no write-write conflicts)
    - How should read-write conflicts (consistency) be addressed?
  - NIS:
    - User information database in Unix systems
    - Only sys-admins update database, users only read data
    - Only user updates are changes to password

Eventual Consistency

- Assume a replicated database with few updaters and many readers
- *Eventual consistency*:
  - Definition: in absence of updates, all replicas converge towards identical copies
  - Only requirement: an update should eventually propagate to all replicas
  - Cheap to implement: no or infrequent write-write conflicts
  - Things work fine as long as user accesses same replica
  - What if they don't?

Eventual Consistency



Client-centric Consistency Models

- Assume read operations by a single process *P* at two *different* local copies of the same data store, four different consistency semantics:
  - *Monotonic reads: once read, subsequent reads on that data item return the same or more recent value*
  - *Monotonic writes: a write must be propagated to all replicas before a successive write by the same process*
  - *Read your writes*: read(x) always returns write(x) by that process
  - *Writes follow reads*: write(x) following read(x) will take place on the same or more recent version of x

Epidemic Protocols

- Bayou: weakly connected replicas
  - Useful in mobile computing (mobile laptops)
  - Useful in wide area distributed databases (weak connectivity)
- Based on theory of epidemics
  - Upon an update, try to “infect” other replicas as quickly as possible
  - Pair-wise exchange of updates (*like pair-wise spreading of a disease)*
  - Terminology:
    - Infective store: store with an update that is willing to spread
    - Susceptible store: store that is not yet updated
    - Removed store: store that is not willing or able to spread its updates

Spreading an Epidemic

- Anti-entropy
  - Server P picks a server Q at random and exchanges updates
  - Three different possibilities: pull, push, or both
  - Claim: A pure push-based approach does not help spread updates quickly (Why?)

- Rumor spreading (aka *gossiping*)
  - Upon receiving an update, P tried to push to Q
  - If Q already received the update, stop spreading with probability of 1/k
  - Con?

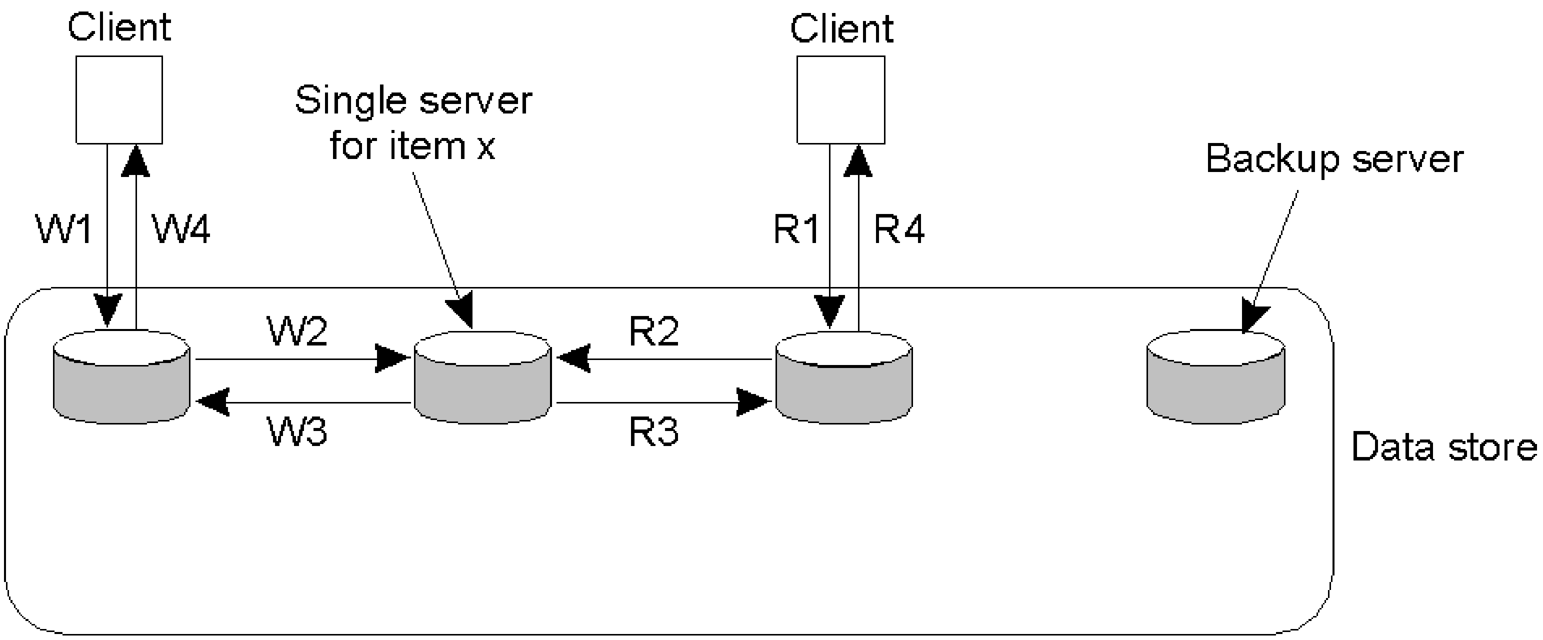
Removing Data

- Deletion of data items is hard in epidemic protocols
- Example: server deletes data item *x*
  - No state information is preserved
    - Can't distinguish between a deleted copy and no copy!

Implementation Issues

- Two techniques to implement consistency models
  - Primary-based protocols
    - Assume a primary replica for each data item
    - Primary is responsible for coordinating all writes
  - Replicated write protocols
    - No primary is assumed for a data item
    - Writes can take place at any replica

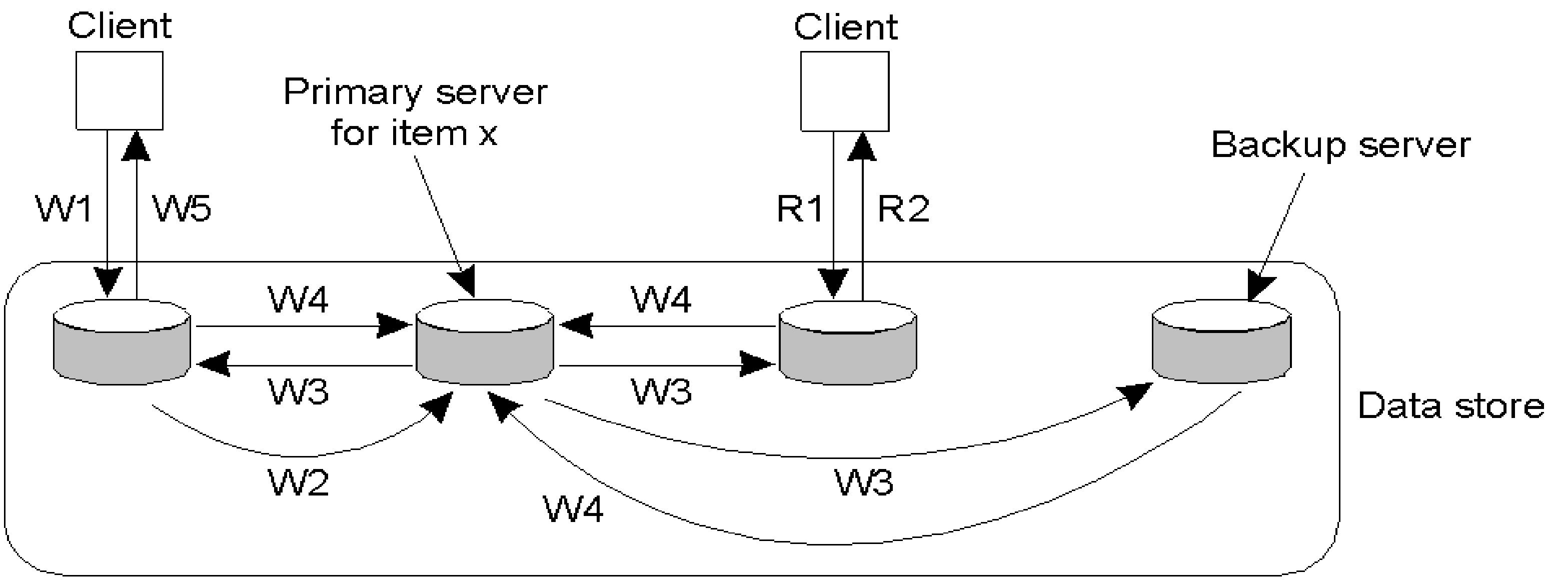
Remote-Write Protocols



W1. Write request
W2. Forward request to server for x
W3. Acknowledge write completed
W4. Acknowledge write completed

R1. Read request
R2. Forward request to server for x
R3. Return response
R4. Return response

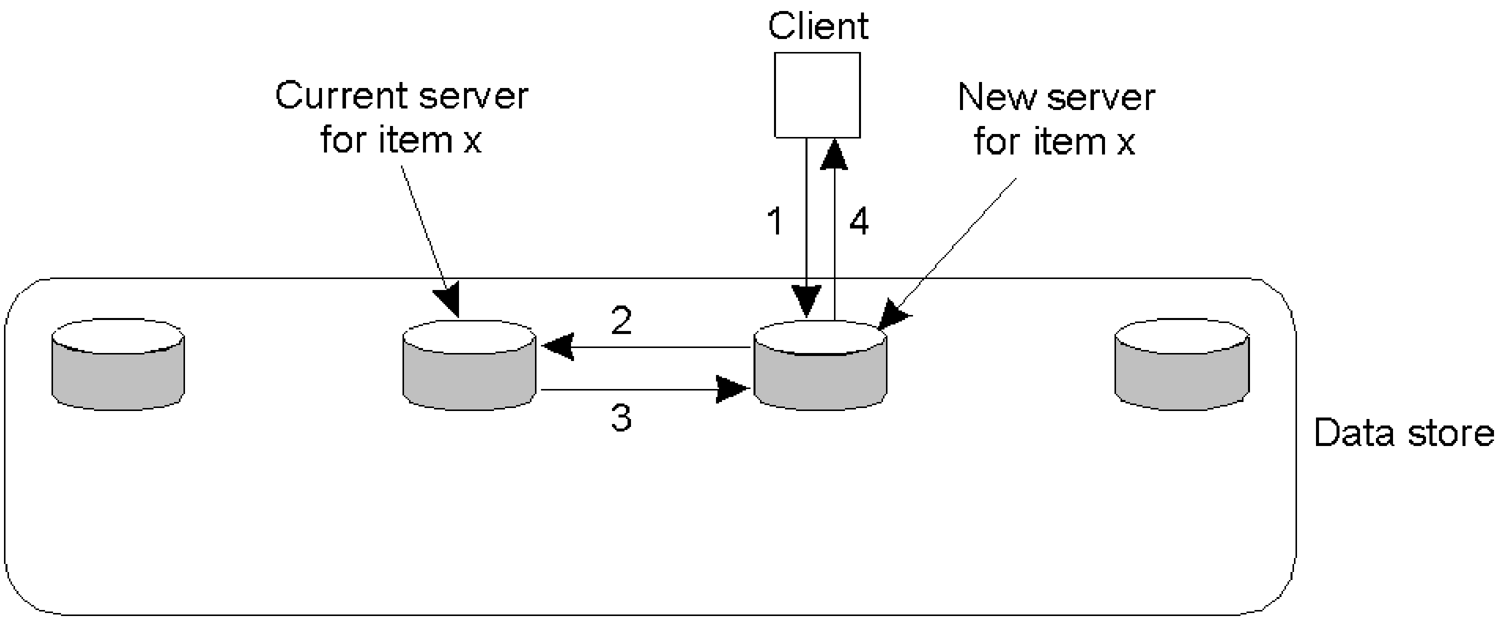
Remote-Write Protocols (2)



W1. Write request
W2. Forward request to primary
W3. Tell backups to update
W4. Acknowledge update
W5. Acknowledge write completed

R1. Read request
R2. Response to read

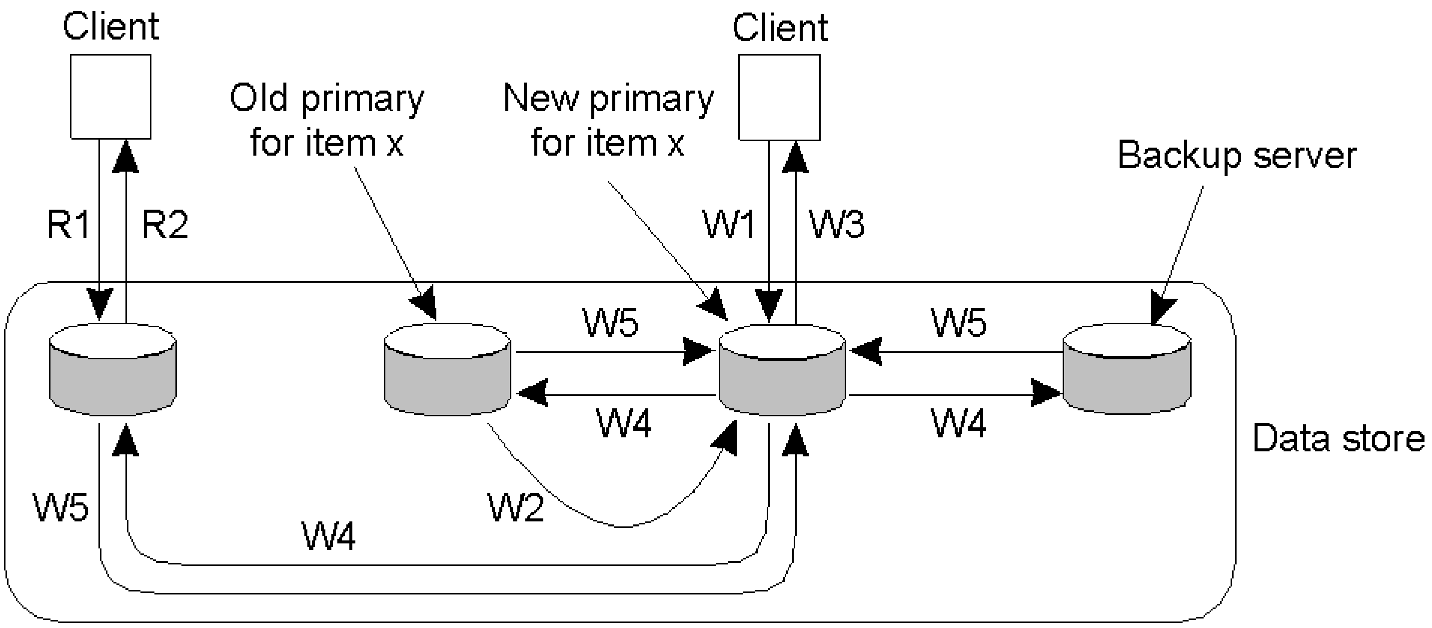
Local-Write Protocols (1)



1. Read or write request
2. Forward request to current server for x
3. Move item x to client's server
4. Return result of operation on client's server

- Limitation: ?

Local-Write Protocols (2)



W1. Write request
W2. Move item x to new primary
W3. Acknowledge write completed
W4. Tell backups to update
W5. Acknowledge update

R1. Read request
R2. Response to read

Replicated-write Protocols

- Relax the assumption of one primary
  - No primary, any replica is allowed to update
  - Consistency is more complex to achieve
- Quorum-based protocols
  - Use voting to request/acquire permissions from replicas
  - Example:
    - Consider a file replicated on N servers
    - Update: contact N/2+1 replicas and get them to agree to do the update (with a version number for the file)
    - Read: contact N/2+1 replicas and obtain the version number

Cache-coherent Protocols

- Mostly used for shared-memory systems
  - Based on hardware support (snooping or broadcast) or software-based solutions
- Two major design issues:
  - Coherence detection strategy
    - Determines when inconsistency are actually detected
  - Coherence enforcement strategy
    - Determines how caches are kept consistency with the copies stored at servers

Final Thoughts

- Replication and caching improve performance in distributed systems
- Consistency of replicated data is crucial
- Many consistency semantics (models) possible
  - Need to pick appropriate model depending on the application
  - Example: web caching: weak consistency is OK since humans are tolerant to stale information (can reload browser)
  - Implementation overheads and complexity grows if stronger guarantees are desired

Summary

- Replication
- Consistency models
- Replica placement
- Distribution protocols
- Client-centric models
- Eventual consistency and Epidemic protocols
- Implementation issues (consistency protocols)
  - Primary-based
  - Replicated-write
  - Cache-coherence
- Readings:
  - AST chpt 7

Questions

?