Consistency and Replication

Some slides are from Prof. Jalal Y. Kawash at Univ. of Calgary
Reasons for Replication

• Reliability/Availability:
  – Mask failures
  – Mask corrupted data

• Performance:
  – Scalability (size and geographical)

• Examples:
  – Web caching
  – Horizontal server distribution
  – Object distribution
Example – Object Replication

(a) A distributed system for replication-aware distributed objects.
(b) A distributed system responsible for replica management
Cost of Replication

- Replicas must be kept consistent

Dilemma:

1. Replicate data for better performance
2. Modification on one copy triggers modifications on all other replicas
3. Propagating each modification to each replica can degrade performance
Consistency Issues – Access/Update Ratio

User accesses to the page

Updates to the Web page

Lost Updates

Time
Consistency Model

- *When and how* the modifications are made = *consistency model:*
  - Weak versus strong consistency model
Consistency Models (cont.)

The general organization of a logical data store, physically distributed and replicated across multiple processes.
Consistency Models (cont)

- A process performs a read operation on a data item, expects the operation to return a value that shows the result of the last write operation on that data.
- No global clock ⇒ difficult to define the last write operation.
- Consistency models provide other definitions.
- Different consistency models have different restrictions on the values that a read operation can return.
# Summary of Consistency Models

<table>
<thead>
<tr>
<th>Consistency</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strict</td>
<td>Absolute time ordering of all shared accesses matters.</td>
</tr>
<tr>
<td>Linearizability</td>
<td>All processes must see all shared accesses in the same order. Accesses are furthermore ordered according to a (nonunique) global timestamp</td>
</tr>
<tr>
<td>Sequential</td>
<td>All processes see all shared accesses in the same order. Accesses are not ordered in time</td>
</tr>
<tr>
<td>Causal</td>
<td>All processes see causally-related shared accesses in the same order.</td>
</tr>
<tr>
<td>FIFO</td>
<td>All processes see writes from each other in the order they were used. Writes from different processes may not always be seen in that order</td>
</tr>
</tbody>
</table>

(a)

<table>
<thead>
<tr>
<th>Consistency</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weak</td>
<td>Shared data can be counted on to be consistent only after a synchronization is done</td>
</tr>
<tr>
<td>Release</td>
<td>Shared data are made consistent when a critical region is exited</td>
</tr>
<tr>
<td>Entry</td>
<td>Shared data pertaining to a critical region are made consistent when a critical region is entered.</td>
</tr>
</tbody>
</table>

(b)

a) Consistency models not using synchronization operations.
b) Models with synchronization operations.
Distribution Models

• Replica Placement:
  – *Where* is a replica placed?
  – *When* is a replica created?
  – *Who* creates the replica?

• How do we distribute updates between replicas?
  – Update propagation
  – Epidemic protocols
Types of Replicas

The logical organization of different kinds of copies of a data store into three concentric rings.
Permanent Replicas

- Initial set of replicas
  - Other replicas can be created from them
  - Small and static set
- Example: Web site horizontal distribution
  1. Replicate Web site on a limited number of machines on a LAN
     - Distribute request in round-robin
  2. Replicate Web site on a limited number of machines on a WAN (*mirroring*)
     - Clients choose which sites to talk to
Server-Initiated Replicas (1)

- Dynamically created at the request of the owner of the DS
- Example: push-caches
  - Owners: web owners for CNN, Yahoo
  - Web hosting servers can dynamically create replicas close to the demanding client
- Need dynamic policy to create and delete replicas
- One is to keep track of Web page hits
  - Keep a counter and access-origin list for each page
Server-Initiated Replicas (2)

Counting access requests from different clients.
Client-Initiated Replicas

- These are caches
  - Temporary storage (expire fast)
- Managed by clients
- Cache hit: return data from cache
- Cache miss: load copy from original server
- Kept on the client machine, or on the same LAN
- Multi-level caches
Multi-Level Caches
Revisit: Iterative versus Recursive Resolution

The comparison between recursive and iterative name resolution with respect to caching.
Design Issues for Update Propagation

- A client sends an update to a replica.
- Based on the consistency model supported, the update is then propagated to all other replicas at its proper time.

1. Propagate state or operation
2. Pull or Push protocols
3. Unicast or multicast propagation
State versus Operation Propagation (1)

1. Propagate a notification of update
   - Invalidate protocols
     - When data item x is changed at a replica, it is invalidated at other replicas
     - An attempt to read the item causes an “item-fault” triggering updating the local copy before the read can complete
   - Uses little network bandwidth
   - When is good to use this distribution protocol?
     - When read-to-write ratio is low or high?
     - Good when read-to-write ratio is Low
State versus Operation Propagation (2)

When read-to-write ratio is high:

2. Transfer modified data
   - Uses high network bandwidth

3. Propagate the update operation
   - Each replica must have a process capable of performing the update
   - Uses very low network bandwidth
Pull versus Push

- **Push**: updates are propagated to other replicas without solicitation
  - Typically, from permanent to server-initiated replicas
  - Used to achieve a high degree of consistency
- **Pull**: A replica asks another for an update
  - Typically, from client-initiated replica
  - Inconsistent cache results in longer response time
A comparison between push-based and pull-based protocols in the case of multiple client, single server systems.

<table>
<thead>
<tr>
<th>Issue</th>
<th>Push-based</th>
<th>Pull-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>State of server</td>
<td>List of client replicas and caches</td>
<td>None</td>
</tr>
<tr>
<td>Messages sent</td>
<td>Update (and possibly fetch update later)</td>
<td>Poll and update</td>
</tr>
<tr>
<td>Response time at client</td>
<td>Immediate (or fetch-update time)</td>
<td>Fetch-update time</td>
</tr>
</tbody>
</table>
Unicast versus Multicast

- **Unicast**: a replica sends separate n-1 updates to every other replica

- **Multicast**: a replica sends one message to multiple receivers
  - Network takes care of multicasting
  - Can be cheaper
  - Suites push-based protocols
Consistency Protocols

• Actual implementation of consistency models

• Whether primary copy exists or not
  – Primary-based protocols
  – Replicated-write protocols
Primary-Based Remote Write Protocols

• Non-replicated
  – Single copy of each data item
  – Both read and write at the (remote) primary server
Primary-based remote-write protocol with a fixed server to which all read and write operations are forwarded.
Primary-Based Remote-Write Protocols (cont.)

• Replicated: primary-backup remote write protocol
  – Primary copy and backups for each data item
  – Read from local copy
  – Write to the (remote) primary server
  – Update backups
    • Blocking vs. non-blocking update
The principle of primary-backup protocol.
Primary-Based Local Write Protocols

- Non-replicated
  - Single copy of each data item
  - Move primary copy to local server and write to it

- Replicated: Primary-backup local write protocol
  - Primary copy and backups for each data item
  - Read from local copy
  - Move primary copy to local server and write to it
  - Update backups
Primary-based local-write protocols (cont.)

Primary-based local-write protocol in which a single copy is migrated between processes.

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
Primary-based local-write protocols (cont.)

Primary-backup protocol in which the primary migrates to the process wanting to perform an update.

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
• Which consistency protocol does DNS (Domain Name System) follow?
No Primary Copy — Replicated-Write Protocols

- Active replication
- Quorum-based protocol
Ordering Guarantees

• Anti-entropy, Gossip & Unreliable multicast
  – Messages sent from different processes may be delivered in different orders at different sites

• Totally-ordered multicast

• Causally-ordered multicast

• The sequencer approach
  – All requests must be sent to a sequencer, where they are given an identifier
  – The sequencer assigns consecutive increasing identifiers as it receives requests
  – Requests arriving at sites are held back until they are next in sequence
Active Replication

The problem of replicated invocations.
Active Replication (cont.)

Forwarding an invocation request from a replicated object.
Returning a reply to a replicated object.
Network Partitions
Well-known Solution: Quorum-Based Protocols

• Idea?
  – Use Majority
    • Write
    • Read

• Read
  – Retrieve number of replicas in read quorum
  – Select the one with the latest version.
  – Perform a read on it

• Write
  – Retrieve number of replicas in write quorum.
  – Find the latest version and increment it.
  – Perform a write on the entire write quorum.
Quorum-Based Protocols

- **N**: Total #Replicas
- **N_R**: #Replicas in Read Quorum
- **N_W**: #Replicas in Write Quorum
- **Constraints:**
  1. $N_R + N_W > N$
  2. $N_W > N/2$
Quorum-Based Protocols

Three examples of the voting algorithm for $N = 12$ replicas
(a) A correct choice of read and write set
(b) A choice that may lead to write-write conflicts
(c) A correct choice, known as ROWA (read one, write all)