Outline

- Other Topics in Distributed Systems
  - Consistency
  - Replication
  - Fault Tolerance

- Q&A
## Summary of Data-Centric Consistency Models

<table>
<thead>
<tr>
<th>Consistency</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Strict</strong></td>
<td>Absolute time ordering of all shared accesses matters.</td>
</tr>
<tr>
<td><strong>Linearizability</strong></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><strong>Sequential</strong></td>
<td>All processes see all shared accesses in the same order. Accesses are not ordered in time</td>
</tr>
<tr>
<td><strong>Causal</strong></td>
<td>All processes see causally-related shared accesses in the same order.</td>
</tr>
<tr>
<td><strong>FIFO</strong></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>
## Summary of Data-Centric Consistency Models

<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>
Client-Centric Consistency Models

- Eventual Consistency
- Monotonic Reads
- Monotonic Writes
- Read Your Writes
- Writes Follow Reads
- Implementation
Eventual Consistency

All Replicas Will Gradually Become Consistent

Distributed Data Stores Characterized by the Lack of Simultaneous Updates: e.g., DNS and WWW

The principle of a mobile user accessing different replicas of a distributed database.
Monotonic-Read Consistency

If a process reads the value of a data item \( x \), any successive read operation on \( x \) by that process will always return that same value or a recent value.

Write Set on \( x \) at L1 returning \( x_1 \):

\[
L1: \text{WS}(x_1) \quad R(x_1)
\]

\[
L2: \quad \text{WS}(x_1;x_2) \quad R(x_2)
\]

→ Time

\[
L1: \text{WS}(x_1) \quad R(x_1)
\]

\[
L2: \quad \text{WS}(x_2) \quad R(x_2) \quad \text{WS}(x_1;x_2)
\]

Monotonic-Read Consistent Data Store

Data Store Not Providing Monotonic Reads
Monotonic-Write Consistency

- A Write Operation by a Process on a Data Item $x$, is completed before any successive Write Operation on $x$ by the same Process.

$L_1: W(x_1)$
$L_2: W(x_1) \rightarrow \text{Time} \quad W(x_2)$
$L_1: W(x_1)$
$L_2: \quad W(x_2)$

Monotonic-Write Consistent Data Store

Previous Write Operation at L1 Has Already Been Propagated to L2

Data Store Not Providing Monotonic Writes
Read-Your-Writes Consistency

The Effect of a Write Operation by a Process on Data Item $x$ Will Always Be Seen by a Successive Read Operation on $x$ by the Same Process

$L1: W(x_1)$
$L2: WS(x_1; x_2) \quad R(x_2)$

→ Time
$L1: W(x_1)$
$L2: WS(x_2) \quad R(x_2)$
Writers-Follow-Reads Consistency

A write operation by a process on data item \( x \) following a previous read operation on \( x \) by the same process is guaranteed to take place on the same or a more recent value of \( x \) that was read.

\[
\begin{align*}
L1: & \ WS(x1) \quad R(x1) \\
L2: & \ WS(x1;x2) \quad W(x2) \quad \text{→Time} \\
L1: & \ WS(x2) \quad R(x1) \\
L2: & \ WS(x2) \quad W(x2)
\end{align*}
\]
Implementation

Assumptions

- Each write operation is assigned a globally unique identifier by the server that accepts the operation for the first time (i.e., the operation is initiated at that server)
- Read set for a client consists of the write identifiers relevant for the read operations performed by a client
- Write set consists of the writes performed by the client
Monotonic-Read Consistency

- When a client performs a read at a server, that server is handed the client’s read set to check whether all the identified writes have been taken place locally.
- If not, the read is postponed. Alternatively, the read is forwarded to a server where the writes have already taken place.

Monotonic-Write Consistency

- When a client performs a write at a server, that server is handed the client’s write set to check whether all the identified writes have been performed first and in correct order.
Read-Your-Writes Consistency

- When a client performs a read at a server, that server is handed the client’s write set to check whether all the identified writes have been taken place locally.

Writes-Follow-Reads Consistency

- When a client performs a write at a server, that server is handed the client’s read set to check whether all the identified writes have been taken place locally.
Replica Placement

- **Permanent Replicas**
  - Distribution of a Web site
    - Replication across servers
    - Mirroring
- **Server-Initiated Replicas**
- **Client-Initiated Replicas**
  - (Client) caches

Load Balancing Issue
Server-Initiated Replicas

Counting access requests from different clients.
Pull vs Push Protocols

<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>

A comparison between push-based and pull-based protocols in the case of multiple client, single server systems.
Consistency Protocols

- Protocols for Sequential Consistency
  - Classified by Whether or Not There Is a Primary Copy of data to Which All Write Operations Should Be Forwarded
    - Primary-Based Protocols
    - Replicated-Write Protocols
      - A write operation can be initiated at any replica
Remote-Write Protocols

Primary-based remote-write protocol with a fixed server to which all read and write operations are forwarded.
Remote-Write Protocols (Cont’d)

The principle of primary-backup protocol.

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

Primary-based local-write protocol in which a single copy is migrated between processes.
Local Write Protocols (Cont’d)

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
Active Replication

- An Operation Is Forwarded to Any Replica with an Associated Process Carrying Out Update Operations

  - Potential Problems
    - Operations need to be carried out in the same order everywhere
      - Totally ordered multicast mechanism
      - Sequencer, a central coordinator
    - Invocations can be replicated
Replicated Invocation Problem

The problem of replicated invocations.
Replicated Invocation Solution

(a) Forwarding an invocation request from a replicated object.

(b) Returning a reply to a replicated object.
Quorum-Based Protocol

Idea: To Require Clients to Request and Acquire the Permission of Multiple Servers before Carrying Out Any Operation

- To update a file, a client must first contact at least half the server plus one (a majority) and get them to agree to do the update.

- To read a replicated file, a client must also contact at least half the servers plus one and ask them to send the version numbers associated with the file.
Fault Tolerance

- **Dependability**
  - **Availability**
    - Ready to be used immediately
  - **Reliability**
    - Running continuously without failure
- **Safety**
  - Not leading to catastrophe in the case of temporary failure
- **Maintainability**
  - How easy a failed system can be repaired
## Failure Modes

<table>
<thead>
<tr>
<th>Type of failure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crash failure</td>
<td>A server halts, but is working correctly until it halts</td>
</tr>
<tr>
<td>Omission failure</td>
<td>A server fails to respond to incoming requests</td>
</tr>
<tr>
<td></td>
<td>A server fails to receive incoming messages</td>
</tr>
<tr>
<td></td>
<td>A server fails to send messages</td>
</tr>
<tr>
<td>Timing failure</td>
<td>A server's response lies outside the specified time interval</td>
</tr>
<tr>
<td>Response failure</td>
<td>The server's response is incorrect</td>
</tr>
<tr>
<td></td>
<td>The value of the response is wrong</td>
</tr>
<tr>
<td></td>
<td>The server deviates from the correct flow of control</td>
</tr>
<tr>
<td>Arbitrary failure</td>
<td>A server may produce arbitrary responses at arbitrary times</td>
</tr>
</tbody>
</table>

**Different types of failures.**
Failure Masking by Redundancy

A Fault Here Effectively Equals a Fault in B1

Triple modular redundancy.
Process Resilience

Flat Groups vs Hierarchical Groups

**Advantage:** No Single Point of Failure

**Disadvantage:** Complicated Decision Making

(a) Communication in a flat group.

(b) Communication in a simple hierarchical group
Agreement in Faulty Systems

- Two-Army Problem: Agreement Even between TWO Processes Is Not Possible in the Face of Unreliable Communication

- Byzantine Generals Problem
  - Whether to reach an agreement with loyal generals and traitors
The Byzantine generals problem for 3 loyal generals and 1 traitor.

a) The generals announce their troop strengths (in units of 1 kilosoldiers).

b) The vectors that each general assembles based on (a)

c) The vectors that each general receives in step 3.
Agreement in Faulty Systems (Cont’d)

A System with \( m \) Faulty Processes, Agreement Can Be Achieved Only If \( 2m+1 \) Correctly Functioning Processes Are Present, for a Total of \( 3m+1 \)

The same as in previous slide, except now with 2 loyal generals and one traitor.