Consistency and Replication

- Primary Objectives of Replication – reliability and performance (scaling)
- Main Issue – consistency of replicated data

Topics

- Data centric consistency models
- Client-centric consistency models
- Replica management
Consistency Models

- A contract between processes and data store - if processes obey certain rules the store promises to work correctly

- **Data-Centric Consistency Models**
  - A read operation on a data item should return a value based on the results of the last write operation on that data

- **Client-Centric Consistency Models**
  - Do not consider the fact that data may be shared by several users
  - Concentrate on the consistency that an individual client should be offered
Data-Centric Consistency Models

A simple organization of a logical data store, physically distributed and replicated across multiple processes.
Continuous Consistency - Deviation

- Deviation in numerical values between replicas
  - Absolute numerical deviation
  - Relative numerical deviation
- Deviation in staleness between replicas
  - Old data can be tolerated as long as it is not too old
- Deviation in ordering of update operations
  - Different ordering of updates are allowed as long as the differences remain bounded
Continuous Consistency – Conit

- The notion of a Conit (CONsistency un1T) – a unit over which consistency is to be measured
- Example – assume that two replicas may differ in no more than one outstanding update
- When each data items have been updated in the first replica, the second one needs to be updated as shown in (a)
Choosing the appropriate granularity for a commit

(b) No update propagation is needed (yet)
Continuous Consistency – contd.

- **Appropriate granularity for a conit**
  - If a conit represents a lot of data, then upgrades are aggregated for all the data in the conit – bring replicas sooner in an inconsistent state
  - If the conits are small, the replicas may still be considered up to date (provided the data items contained in a conit are used completely independently) – overhead required for the management of large number of conits
Consistent Ordering of Operations

Complimentary to continuous consistency – when tentative updates at replicas need to be committed, replicas will require to agree on a global ordering on those updates

- Strict consistency
- Sequential consistency
- Causal consistency
- Entry consistency
Strict Consistency

- Any read on a data item x returns a value corresponding to the result of the most recent write on x, independent of:
  - How soon after the change the reads are done
  - Which processes are reading and where they are located
  - How quickly the next write is done

- All writes are passed instantaneously to all processes and an absolute global time order is maintained

Behavior of two processes, operating on the same data item.

(a) A strictly consistent store  
(b) A store that is not strictly consistent
Sequential Consistency

- Difficult to maintain a global clock – slightly weaker than strict consistency
- The result of any execution is the same as if the (read and write) operations by all processes on the data store were executed in some sequential order and the operations of each individual process appear in this sequence in the order specified by its program – all processes see the same interleaving of operations. Nothing is said about time – no reference to the most recent write.

\begin{center}
\begin{tabular}{c|c}
P1: & W(x)a \\
\hline
P2: & W(x)b \\
\hline
P3: & R(x)b \quad \text{R(x)a} \\
\hline
P4: & R(x)b \quad \text{R(x)a} \\
\end{tabular}
\end{center}

\begin{center}
\begin{tabular}{c|c}
P1: & W(x)a \\
\hline
P2: & W(x)b \\
\hline
P3: & R(x)b \quad \text{R(x)a} \\
\hline
P4: & \text{R(x)a} \quad \text{R(x)b} \\
\end{tabular}
\end{center}

(a) A sequentially consistent data store (all processes see the same interleaving) 
(b) A data store that is not sequentially consistent (P3 and P4 see different interleaving)
Sequential Consistency – Interleaving Example

<table>
<thead>
<tr>
<th>Process P1</th>
<th>Process P2</th>
<th>Process P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 1;</td>
<td>y = 1;</td>
<td>z = 1;</td>
</tr>
<tr>
<td>print (y, z);</td>
<td>print (x, z);</td>
<td>print (x, y);</td>
</tr>
<tr>
<td>y = 1;</td>
<td>z = 1;</td>
<td></td>
</tr>
<tr>
<td>print (x, z);</td>
<td>print (x, y);</td>
<td>print (x, z);</td>
</tr>
<tr>
<td>z = 1;</td>
<td>x = 1;</td>
<td></td>
</tr>
<tr>
<td>print (x, y);</td>
<td>print (x, y);</td>
<td>print (y, z);</td>
</tr>
</tbody>
</table>

Three concurrently executing processes

Four valid execution sequences for the processes of the previous slide

(a) x = 1; print (y, z); y = 1; print (x, z); z = 1; print (x, y); Prints: 001011

(b) x = 1; y = 1; print (x, z); print (y, z); print (x, y); Prints: 101011

(c) y = 1; z = 1; print (x, y); print (x, z); print (y, z); Prints: 111111

(d) Process P1 Process P2 Process P3

x = 1; y = 1; z = 1; print (x, y); print (x, z); print (y, z); print (x, y); Prints: 110111
Causal Consistency

- Weaken sequential consistency by making a distinction between events that are potentially causally related and those that are not.
- Necessary condition:
  - Writes that are potentially causally related must be seen by all processes in the same order. Concurrent writes may be seen in a different order on different machines.

<table>
<thead>
<tr>
<th></th>
<th>P1: W(x)a</th>
<th>W(x)c</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2:</td>
<td>R(x)a</td>
<td>W(x)b</td>
</tr>
<tr>
<td>P3:</td>
<td>R(x)a</td>
<td>R(x)c</td>
</tr>
<tr>
<td>P4:</td>
<td>R(x)a</td>
<td>R(x)b</td>
</tr>
</tbody>
</table>

- This sequence is allowed with a causally-consistent store, but not with sequentially or strictly consistent store - W2(x)b and W1(x)c are concurrent - not required that all processes see them in the same order.
Causal Consistency – another example

a) A violation of a causally-consistent store – W2(x)b potentially depending on W1(x)a because b may be a result of a computation involving the value read by R2(x)a. The two writes are causally related – all processes must see them in the same order.

b) A correct sequence of events in a causally-consistent store – the read operation is removed, so W1(x)a and W2(x)b are now concurrent writes – does not require to be globally ordered.
Entry Consistency

- An acquire access of a synchronization variable is not allowed to perform until all updates to the guarded shared data have been performed with.
- No other process may hold the synchronization variable, not even in nonexclusive mode.
- After an exclusive mode access to a synchronization variable has been performed, any other process’s next non-exclusive mode access to that synchronization variable may not be performed until it has performed with respect to that variable’s owner.

<table>
<thead>
<tr>
<th>P1: Acq(Lx) W(x)a Acq(Ly) W(y)b Rel(Lx) Rel(Ly)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2: Acq(Lx) R(x)a R(y)NIL</td>
</tr>
<tr>
<td>P3: Acq(Ly) R(y)b</td>
</tr>
</tbody>
</table>

A valid event sequence for entry consistency (associates lock with each data item instead of entire shared data)
## Summary of Consistent Ordering of Operations

<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>Sequential</strong></td>
<td>All processes see all shared accesses in the same order (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>Entry</strong></td>
<td>Shared data pertaining to a critical region are made consistent when a critical region is entered</td>
</tr>
</tbody>
</table>
Summary

- Data Centric Consistency Models
  - Continuous consistency
  - Consistent ordering of operations
    - Strict consistency
    - Sequential consistency
    - Causal consistency
    - Entry consistency