Consistency examples

COS 418: Distributed Systems
Precept 5

Themis Melissaris
Plan

• Midterm poll

• Consistency examples
Fill out this poll:
http://tinyurl.com/zdeq4lr
Linearizability
Strong consistency = linearizability

- Linearizability (Herlihy and Wang 1991)
  1. All servers execute all ops in some identical sequential order
  2. Global ordering preserves each client’s own local ordering
  3. Global ordering preserves real-time guarantee
     - All ops receive global time-stamp using a sync’d clock
     - If \( \text{ts}_{op1}(x) < \text{ts}_{op2}(y) \), \( \text{OP}1(x) \) precedes \( \text{OP}2(y) \) in sequence
   - Once write completes, all later reads (by wall-clock start time) should return value of that write or value of later write.
   - Once read returns particular value, all later reads should return that value or value of later write.
Sequential Consistency
Is that valid?

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

No. Why?

Ordering of the Write operations needs to be preserved
Causal Consistency
### Examples

<table>
<thead>
<tr>
<th>Operations</th>
<th>Concurrent?</th>
</tr>
</thead>
<tbody>
<tr>
<td>a, b</td>
<td></td>
</tr>
<tr>
<td>b, f</td>
<td></td>
</tr>
<tr>
<td>c, f</td>
<td></td>
</tr>
<tr>
<td>e, f</td>
<td></td>
</tr>
<tr>
<td>e, g</td>
<td></td>
</tr>
<tr>
<td>a, c</td>
<td></td>
</tr>
<tr>
<td>a, e</td>
<td></td>
</tr>
</tbody>
</table>

![Diagram](image)

Physical time ↓
Example

<table>
<thead>
<tr>
<th>Operations</th>
<th>Concurrent?</th>
</tr>
</thead>
<tbody>
<tr>
<td>a, b</td>
<td>N</td>
</tr>
<tr>
<td>b, f</td>
<td></td>
</tr>
<tr>
<td>c, f</td>
<td></td>
</tr>
<tr>
<td>e, f</td>
<td></td>
</tr>
<tr>
<td>e, g</td>
<td></td>
</tr>
<tr>
<td>a, c</td>
<td></td>
</tr>
<tr>
<td>a, e</td>
<td></td>
</tr>
</tbody>
</table>

Physical time ↓
## Example

<table>
<thead>
<tr>
<th>Operations</th>
<th>Concurrent?</th>
</tr>
</thead>
<tbody>
<tr>
<td>a, b</td>
<td>N</td>
</tr>
<tr>
<td>b, f</td>
<td>Y</td>
</tr>
<tr>
<td>c, f</td>
<td></td>
</tr>
<tr>
<td>e, f</td>
<td></td>
</tr>
<tr>
<td>e, g</td>
<td></td>
</tr>
<tr>
<td>a, c</td>
<td></td>
</tr>
<tr>
<td>a, e</td>
<td></td>
</tr>
</tbody>
</table>

Physical time ↓

![Diagram](image.png)
Example

<table>
<thead>
<tr>
<th>Operations</th>
<th>Concurrent?</th>
</tr>
</thead>
<tbody>
<tr>
<td>a, b</td>
<td>N</td>
</tr>
<tr>
<td>b, f</td>
<td>Y</td>
</tr>
<tr>
<td>c, f</td>
<td>Y</td>
</tr>
<tr>
<td>e, f</td>
<td></td>
</tr>
<tr>
<td>e, g</td>
<td></td>
</tr>
<tr>
<td>a, c</td>
<td></td>
</tr>
<tr>
<td>a, e</td>
<td></td>
</tr>
</tbody>
</table>

Physical time ↓

Concurrent? N: No, Y: Yes
## Example

<table>
<thead>
<tr>
<th>Operations</th>
<th>Concurrent?</th>
</tr>
</thead>
<tbody>
<tr>
<td>a, b</td>
<td>N</td>
</tr>
<tr>
<td>b, f</td>
<td>Y</td>
</tr>
<tr>
<td>c, f</td>
<td>Y</td>
</tr>
<tr>
<td>e, f</td>
<td>Y</td>
</tr>
<tr>
<td>e, g</td>
<td></td>
</tr>
<tr>
<td>a, c</td>
<td></td>
</tr>
<tr>
<td>a, e</td>
<td></td>
</tr>
</tbody>
</table>

Physical time ↓

- P1
  - a
  - b

- P2
  - c
  - d
  - e

- P3
  - f
  - g
## Example

<table>
<thead>
<tr>
<th>Operations</th>
<th>Concurrent?</th>
</tr>
</thead>
<tbody>
<tr>
<td>a, b</td>
<td>N</td>
</tr>
<tr>
<td>b, f</td>
<td>Y</td>
</tr>
<tr>
<td>c, f</td>
<td>Y</td>
</tr>
<tr>
<td>e, f</td>
<td>Y</td>
</tr>
<tr>
<td>e, g</td>
<td>N</td>
</tr>
<tr>
<td>a, c</td>
<td></td>
</tr>
<tr>
<td>a, e</td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

- **P1**: Operations: `a`, `b` (Concurrent: N)
- **P2**: Operations: `b`, `f` (Concurrent: Y), `c`, `f` (Concurrent: Y), `e`, `f` (Concurrent: Y)
- **P3**: Operations: `f` (Concurrent: Y), `g` (Concurrent: N)

Physical time ↓
### Example

<table>
<thead>
<tr>
<th>Operations</th>
<th>Concurrent?</th>
</tr>
</thead>
<tbody>
<tr>
<td>a, b</td>
<td>N</td>
</tr>
<tr>
<td>b, f</td>
<td>Y</td>
</tr>
<tr>
<td>c, f</td>
<td>Y</td>
</tr>
<tr>
<td>e, f</td>
<td>Y</td>
</tr>
<tr>
<td>e, g</td>
<td>N</td>
</tr>
<tr>
<td>a, c</td>
<td>Y</td>
</tr>
<tr>
<td>a, e</td>
<td></td>
</tr>
</tbody>
</table>

Physical time ↓

![Diagram with nodes and arrows](image)

- a, b: N
- b, f: Y
- c, f: Y
- e, f: Y
- e, g: N
- a, c: Y
- a, e: 

- P1
  - a
  - b

- P2
  - c
  - d
  - e

- P3
  - f
  - g
### Example

<table>
<thead>
<tr>
<th>Operations</th>
<th>Concurrent?</th>
</tr>
</thead>
<tbody>
<tr>
<td>a, b</td>
<td>N</td>
</tr>
<tr>
<td>b, f</td>
<td>Y</td>
</tr>
<tr>
<td>c, f</td>
<td>Y</td>
</tr>
<tr>
<td>e, f</td>
<td>Y</td>
</tr>
<tr>
<td>e, g</td>
<td>N</td>
</tr>
<tr>
<td>a, c</td>
<td>Y</td>
</tr>
<tr>
<td>a, e</td>
<td>N</td>
</tr>
</tbody>
</table>

Diagram:

- Physical time ↓
- Concurrent? N, Y

Nodes:
- P1
  - a
  - b
d
- P2
  - e
  - c
d
  - f
- P3
g

Arrows:
- a → b
- b → c
- e → d
- f → g
Is that valid?

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

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

Valid under causal consistency

**Why?** \( W(x)b \) and \( W(x)c \) are concurrent

So all processes don’t (need to) see them in same order

P3 and P4 read the values ‘a’ and ‘b’ in order as potentially causally related. No ‘causality’ for ‘c’.
What causal dependencies do we have?

Direction of time ------------------->

P1 : \( w_1[x] = 1 \)

P2 : \( w_2[x] = 2 \)

P3 : \( r_3[x] = 2 \)

P4 : \( r_4[x] = 1 \)
What causal dependencies do we have?

Direction of time  

P1 : w1[x] = 1  
P2 : w2[x] = 2  
P3 : r3[x] = 2  
P4 : r4[x] = 1

Execution causally consistent:
• r3[x] is causally dependent on w2[x]
• r4[x] is causally dependent on w1[x]
Designing a stock market

• What kind of consistency would you use to implement an electronic stock market?
Example: Designing a stock market

• What kind of consistency would you use to implement an electronic stock market?

• Causal Consistency:
  – Reactions to changes in stock values should be consistent.
  – Changes in stocks that are independent can be seen in different orders.
Further examples
• In the following example, simplified read, write and reduced system complexity (no replicas).
• Assignment as write, print as read.
• Intended to demonstrate how linearizability, sequential consistency are violated.
• More realistic view, as above.
Example

Client 1

\[ X_1 = X_1 + 1 \]

\[ Y_1 = Y_1 + 1 \]

Client 2

\[ A = X_2 \]

\[ B = Y_2 \]

If (A > B)

print(A)

else …
Example: Linearizable

Client 1

\[ X_1 = X_1 + 1 \]
\[ Y_1 = Y_1 + 1 \]

Client 2

\[ A = X_2 \]
\[ B = Y_2 \]

If \( A > B \)

print(A)

else ...

Example: Not Linearizable, sequentially consistent

Client 1

\[ X_1 = X_1 + 1 \]

\[ Y_1 = Y_1 + 1 \]

Client 2

\[ A = X_2 \]

\[ B = Y_2 \]

If \( A > B \)

print(A)

else …
Example: Not Linearizable nor sequentially consistent

Client 1

\[ X_1 = X_1 + 1 \]

\[ Y_1 = Y_1 + 1 \]

Client 2

\[ A = X_2 \]

\[ B = Y_2 \]

If \( A > B \) print(A) else …
Eventual Consistency
Eventual consistency

- **Eventual consistency**
  - Writes are *eventually* applied in total order
  - Reads might not see most recent writes in total order

- Why do people like eventual consistency?
  - Fast read/write of local copy (no primary, no Paxos)
  - Disconnected operation
Replication with eventual consistency

The diagram illustrates that although replicas are always available to read, some replicas may be inconsistent with the latest write on the originating node, at a particular moment in time. In the diagram, Node A is the originating node and nodes B and C are the replicas.
Bayou

Write log

Version Vector

N0

0:0
1:0
2:0

N1

0:0
1:0
2:0

N2

0:0
1:0
2:0
Bayou propagation

N0
- Write log
- 1:0 W(x)
- 2:0 W(y)
- 3:0 W(z)

Version Vector
0:3
1:0
2:0

N1
- 1:1 W(x)
0:0
1:1
2:0

N2
- 0:0
1:0
2:0
Bayou propagation

Version Vector

N0

Write log

1:0 W(x)
2:0 W(y)
3:0 W(z)

0:3
1:0
2:0

N1

1:0 W(x)
1:1 W(x)
2:0 W(y)
3:0 W(z)

0:3
1:4
2:0

1:1 W(x)

N2

0:0
1:0
2:0
Bayou propagation

Which portion of The log is stable?
Bayou propagation

Write log

Version Vector

N0
1:0 W(x)
1:1 W(x)
2:0 W(y)
3:0 W(z)

0:4
1:4
2:0

N1
1:0 W(x)
1:1 W(x)
2:0 W(y)
3:0 W(z)

0:3
1:4
2:0

N2
1:0 W(x)
1:1 W(x)
2:0 W(y)
3:0 W(z)

0:3
1:4
2:5
Bayou propagation

Write log

Version Vector

N0

1:0 W(x)
1:1 W(x)
2:0 W(y)
3:0 W(z)

N1

1:0 W(x)
1:1 W(x)
2:0 W(y)
3:0 W(z)

0:3
1:4
2:5

N2

1:0 W(x)
1:1 W(x)
2:0 W(y)
3:0 W(z)

0:4
1:4
2:5
Bayou uses a primary to commit a total order

• Why is it important to make log stable?
  – Stable writes can be committed
  – Stable portion of the log can be truncated

• Problem: If any node is offline, the stable portion of all logs stops growing

• Bayou’s solution:
  – A designated primary defines a total commit order
  – Primary assigns CSNs (commit-seq-no)
  – Any write with a known CSN is stable
  – All stable writes are ordered before tentative writes
Bayou propagation

Designated Primary

Version Vector

Write log

N0

1:1:0 W(x)
2:2:0 W(y)
3:3:0 W(z)

0:3 1:0 2:0

N1

∞:1:1 W(x)

N2

0:0 1:1 2:0

0:0 1:0 2:0

Tentative Write

∞:1:1 W(x)
Bayou propagation

Write log

Version Vector

N0
Designated Primary

1:1:0 W(x)
2:2:0 W(y)
3:3:0 W(z)
4:1:1 W(x)

0:4
1:1
2:0

N1

∞:1:1 W(x)
0:0
1:1
2:0

N2

∞:1:1 W(x)
0:0
1:0
2:0

1:1:0 W(x)
2:2:0 W(y)
3:3:0 W(z)
4:1:1 W(x)

0:4
1:1
2:0
Assignment 3
Due November 21

Monday’s topic
Concurrency Control, Locking and Recovery