CS 555: DISTRIBUTED SYSTEMS
[REPLICATION & CONSISTENCY]

Shrideep Pallickara
Computer Science
Colorado State University

November 7, 2017

Frequently asked questions from the previous class survey

- Numerical deviations dependent on ordering deviations?
- Consistency:
  - Who defines it?
  - Does this force the use of a global clock?
- Are there performance bounds on a consistency model, given the strictness of guarantees?

Topics covered in this lecture

- Consistent Ordering of Operations
  - Sequential consistency
  - Causal consistency
  - Client-centric consistency models

CONSISTENT ORDERING OF OPERATIONS

- Operations of processes depicted along time axis
- Write by a process $P_i$ to data item $x$ with value $a$
  - $W_i(x,a)$
- Read by a process $P_i$ of data item $x$ that returns the value $b$
  - $R_i(x,b)$
- All items are initially NIL

Consistent ordering of operations

- Class of models from concurrent programming
  - We will look at
    - Sequential consistency
    - Causal consistency

Sequential consistency: Notations

November 7, 2017
CS555: Distributed Systems [Fall 2017]
Dept. Of Computer Science, Colorado State University
Sequential consistency

- Defined by Lamport
  - Context: Shared memory in multiprocessor setting
  - When processes run concurrently
    - Any valid interleaving of read/write is acceptable
    - But all processes must see the same interleaving

Sequential consistency example

- P1: \( \text{W(x)} \)
- P2: \( \text{R(x)_N I L} \), \( \text{R(x)} \)

Time to propagate update of \( x \) to P2 is acceptable

Sequential consistency: Example

- P1: \( \text{W(x)} \)
- P2: \( \text{W(x)} \)
- P3: \( \text{R(x)} \), \( \text{R(x)} \)
- P4: \( \text{R(x)} \), \( \text{R(x)} \)

Write operation of P2 appears to be before P1
This is acceptable

Sequential Consistency: Another example

- Process 1
  - \( x = 1 \)
  - \( \text{print}(y,z) \)
- Process 2
  - \( y = 1 \)
  - \( \text{print}(x,z) \)
- Process 3
  - \( z = 1 \)
  - \( \text{print}(x,y) \)

Multiple interleaved sequences are possible

- With 6 statements there are
  - 6! possibilities = 720
  - Some of these violate program order
- 120 (5!) sequences begin with \( x=1 \)
  - Half \( \text{print}(x,z) \) before \( y=1 \)
  - Half \( \text{print}(x,y) \) before \( z=1 \)
  - Only 15 or 30 are valid
- Similarly, there are 30 that start with \( y=1, z=1 \)
  - Total of 90 valid execution sequences
Different, but valid interleaving of the statements

<table>
<thead>
<tr>
<th>Statement</th>
<th>Print</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>print(x,y)</td>
<td>001011</td>
<td>Signature</td>
</tr>
<tr>
<td>y = 1</td>
<td>101011</td>
<td></td>
</tr>
<tr>
<td>print(x,z)</td>
<td>110101</td>
<td></td>
</tr>
<tr>
<td>z = 1</td>
<td>111111</td>
<td>Signature</td>
</tr>
<tr>
<td>print(x,y)</td>
<td>001011</td>
<td>Signature</td>
</tr>
<tr>
<td>print(y,z)</td>
<td>101011</td>
<td>Signature</td>
</tr>
<tr>
<td>print(x,z)</td>
<td>110101</td>
<td>Signature</td>
</tr>
</tbody>
</table>

Contract between processes and shared data store

- Processes must accept all valid results
- Must work if any of them occurs

Invalid sequences in signature patterns

- 0000000
  - Print statements ran before assignments
  - Violates program order
- 001001
  - (00) y and z were 0 when P1 did its printing
  - (01) P2 ran after P1 started, but before P3 started
  - (01) P3 must complete before P1 starts
  - Not possible!

Causal consistency

- Weakens sequential consistency
- Makes distinction between events that are causally related
  - If event b caused/is-influenced by event a
  - Everyone must see a before b
- Operations not causally related: concurrent

Causal consistency (Example 1)

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

Note: This is NOT ALLOWED in sequential consistency
**Causal consistency example:**

**Example 2**

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

Writes W1(x)a and W2(x)b are causally related
Process must see them in the same order

**Example 3**

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

Writes W1(x)a and W2(x)b are concurrent writes
Process can see them in different orders

**Concurrency using synchronization operations**

- Operations bracketed by
  - ENTER_CS
  - LEAVE_CS
  - CS: Critical Section
- Semantics enforced using shared synchronization variables

**GROUPING OPERATIONS**

- Each synchronization variable has an owner
- Owner may repeatedly enter or exit critical section
- Process that does not own a synchronization variable
  - Must own it before it can enter critical section
  - Acquire by sending a message to the owner

**Critical sections and synchronization variables**

- Acquire cannot complete until all guarded shared data is up to date
- Before updating a shared item
  - Enter critical section in exclusive mode
- If a process enters a critical region in non-exclusive mode
  - Fetch recent copies of the shared guarded data from owner
Entry consistency example

- **Time**
  - P1: Acq(Lx) W(x) Acq(Ly) W(y) Rel(Lx) Rel(Ly)
  - P2: Acq(Lx) R(x) R(y) NIL
  - P3: Acq(Ly) R(y)

P2 does an acquire for x, but not y: MAY read NIL

---

Applications have different requirements about:

- Concurrency
- Consistency

---

Often only one or a few processes can perform updates:

- How fast should these be propagated to processes that only read?
  - DNS: Different domains managed by naming authority
    - Owner of that domain
    - Write-write conflicts never occur
    - Read-write conflicts may occur
    - But it is still OK to do lazy updates

---

The DNS and Web page examples can be viewed as large (distributed) databases

- That tolerate a high degree of inconsistency
- If no updates take place for a long time
  - All replicas gradually become consistent
  - Eventual consistency
The caveat for eventual consistency

- Works fine as long as clients access the same replica
- Problems when you access different replicas within a short interval

An example of a mobile user accessing different replicas

Client-centric consistency models

- Provides guarantees for a single client accessing the store
- No guarantees for concurrent accesses of store by different clients

Client-centric consistency models

- Monotonic read
- Monotonic write
- Read-your-writes
- Writes-follow-read

Notations for client-centric consistency

- Version of data item $x$ at local copy $L_i$ at time $t_i$:
  - $x[f_i]$
- $x[f_i]$ is the result of a series of operations at $L_i$ since initialization:
  - This set of operations: $WS(x[f_i])$
  - Operation at $L_i$ at $t_i$, and at $L_j$ at time $t_j$
  - $WS(x[f_i], x[j,f_i])$
Monotonic read consistency

- If a process reads a value of x, any successive read on x by that process returns either:
  - Same value
  - More recent value
- If process sees a value of x at time t:
  - It never sees an older version

A mailbox example of monotonic read consistency

- Each user’s mailbox is replicated & distributed
- Lazy/on-demand updates
  - When copies need data for consistency the updates are propagated
- User reads mail in San Francisco … goes to NYC
- Monotonic consistency
  - Messages in mailbox in SF are also there in NYC

Monotonic Read Consistency:
Operations by a single process P

```
L1: W(x1) R(x1) R(x2)
L2: W(x1) R(x1) R(x2)
```

Operations at L1 have been propagated to L2

```
L1: W(x2) R(x2)
L2: W(x2) R(x2)
```

Operations at L1 have NOT been propagated to L2

Representing client-centric consistency

- Time is along horizontal axis
- Different copies of a replica on the vertical axis
- Operations are carried out by a single process

Monotonic Writes

- Write operation on data item x is completed
  - Before any successive write operation on x by the same process
- Copy on which write is performed
  - Reflects effect of a previous write
  - Irrespective of where it was initiated

Monotonic writes

- When each write completely overwrites x
  - Getting things up to date is easier
  - In most cases we perform partial updates; for e.g. x could be software library
  - We update functions etc. to get to the next version
  - If an update is performed to library
    - All preceding updates must first be performed
Monotonic Write Consistency:

Operations by a single process P

- Effect of a write operation on data item x:
  - Seen by successive reads on x by the same process
  - Write operation is always completed before a successive read operation
  - By same process
  - No matter where operations are performed

Read your writes

- User reads an article A
- Reacts by posting article B
- Write follows reads consistency
  - B will be posted to a copy of the newsgroup
  - Only after A has been written

Read-your-Writes Consistency:

Operations by a single process P

- Write operation by process on data item x:
  - Following a previous read on x by the same process
  - Will take place on the same (or more recent) value of x
  - Write operation on item x will be performed on a copy that is up to date
  - With value (most) recently read by process

Write Follow Reads

- Web designer creates a web page
- Tries to view it
- But browser/proxy has cached the older version
- With a read-your-write consistent browser
  - Cache is invalidated when page is updated
- Other example: Updating passwords

Example of inability to enforce read-your-write consistency

- Web designer creates a web page
- Tries to view it
- But browser/proxy has cached the older version
- With a read-your-write consistent browser
  - Cache is invalidated when page is updated
- Other example: Updating passwords
Writes-Follow-Reads Consistency:
Operations by a single process $P$

$L1: W5(x_1) \rightarrow R(x_1) \rightarrow W(x_1)$

Previous operation at $L1$ has been propagated to $L2$

$L2: W5(x_2) \rightarrow R(x_2) \rightarrow W(x_2)$

Operation at $L1$ has NOT been propagated to $L2$

The contents of this slide-set are based on the following references