Frequently asked questions from the previous class survey
Topics covered in this lecture

- Replication
- Consistency Models
- Data centric consistency model
  - Continuous consistency models
  - Sequential consistency
What we will look at in our discussions

- Replication
- Consistency
  - Models
  - Client models
  - Protocols
- Eventual Consistency
- Brewer’s CAP Theorem

Why are these inter-related topics important?

- Performance
- Correctness

- Failure to account for interactions between these issues?
  - Poor performance
  - Inaccurate results

The holy grail of demonstrable incompetency in systems development!
Rationale for replication

- Reliability
- Availability
- Performance
Rationale for replication: Reliability

- Replication as a safeguard against **failures**
- Protection against data **corruptions**
- File System example:
  - 3 copies
  - If one fails, process can choose from the other two
  - Read/write performed on each copy
    - At least 2 of the reads must **concur**
    - Protects against a failing write

Rationale for replication: Increased Availability

- Users require services to be **highly available**
  - Proportion of time when service is accessible with reasonable response times should be close to 100%
- Factors relevant to high-availability
  - Delays due to pessimistic concurrency control
  - Server failures
  - Network partitions and disconnected operations
Replication maintains availability despite server failures

- Data is replicated at failure independent servers
- Client software should be able to access data at an alternative server if default server fails

Replication maintains availability despite server failures

- If each of the $n$ servers has an independent probability $p$ of failing or becoming unreachable
- The availability of an object stored at each of these servers?
  - $1 - \text{probability(all servers fail or are unreachable)}$
  - $1 - p^n$
Replication maintains availability despite server failures: Example

- There is a 5% probability of independent server failures?
- There are two servers
  - Availability is $1 - p^n$
  - $1 - (0.05)^2 = 1 - 0.0025 = 99.75%$

Rationale for replication: Performance

- Ability to scale with **numbers**
  - Processes access data managed by a server
  - Replicate server; distribute work

- Ability to scale with **geographical area**
  - Place copy of data in *proximity* of processes using it
  - Time to access service decreases
    - Perceived performance improves
But replication exacts a price …

- A client may perceive better performance but ...
  - More network bandwidth needed
  - To keep replicas in sync

- Consistency problems
  - When a copy is modified, it becomes different
  - Modifications have to be made on all copies

Replication Costs: When and how modifications must be made to copies

- Fetching a page from a remote Web server
  - OBJECTIVE: Improving access times

- Web browsers locally cache a web page
  - If user requests the same page
    - Returned from cache
    - User is happy with the load times
  - What if user always wants the latest copy?
Simple solutions to the stale copy problem

1. Don’t cache web page
   - If there is no nearby replica, performance is poor
   - Also, what if the page does not change that often?

2. Let server invalidate/update caches
   - Server must track all caches
   - Degrades server performance

Replication as a scaling technique

- Placing data copies close to processes
  - Improves access times
  - Distributes work

- Potential problems …
Replication for scaling:
Network bandwidth

- Process \( P \) accesses a replica \( N \) times per second
- Replica is itself updated \( M \) times per second
- If \( N \ll M \) ?
  - Several updated versions of replica never accessed
  - Network traffic to install those versions: wasted!
  - Perhaps installing a replica was not a good idea?

Replication for scaling:
Consistency issues

- Consistency might itself be subject to scaling problems
- Collection of copies is consistent when all copies are the same
  - Read on any copy returns the same result
  - Updates propagated to all copies before the next operation?
    - Tight consistency
Consistency issues in replication

- Update performed at all copies as an **atomic operation**
  - Transaction

- Implementing atomicity with large number of replicas is difficult
  - May be dispersed on a WAN
  - Operations *cannot* complete quickly

Other things that replicas need to agree on ...

- Replicas must agree on **when** operation must be performed locally

- Replicas need to decide on **ordering**
  - Lamport timestamps
  - Coordinator assigned order
The Replication Dilemma

- Alleviating scalability issues
  - Replication and caching: Improves performance

- Keeping copies consistent?
  - Requires *global* synchronization
  - *Costly* in terms of performance
    - Time
    - Network bandwidth

DATA CENTRIC CONSISTENCY MODELS
Data centric consistency models

- Consistency is in the context of read/write operation on distributed, shared data
  - Memory
  - Database
  - File systems

- The broader term data store is more commonly used

Distributed data store

![Diagram of distributed data store with processes and local copies]
Consistency model

- **Contract** between processes and the data store

- If processes agree to obey certain **rules**
  - Data store works correctly

Consistency that we intuitively expect

- Process performing a *read* on a data item
  - Expects value to show results of *last write* operation on that item

- Without a global clock?
  - Difficult to define *which* write was the last one
We thus need to provide other definitions ...

consistency models

- Each model restricts values that a read operation on a data item can return
- Models with the greatest restrictions
  - Easiest to use
- Models with minor restrictions
  - Difficult to use
- Easy-to-use models do not perform as well as difficult ones

Loosening of consistency

- Needed for efficiency and performance
- No general rules however
  - Tolerance depends on the application
Continuous consistency

- **Three axes** for defining inconsistencies
  - Deviations between replicas in terms of
    - Numerical values
    - Staleness between replicas
    - Ordering of update operations
  - Deviations form **continuous consistency** ranges
Example of using continuous consistency models: Stock prices

- Two copies of a stock should not deviate by more than 2 cents.  
  - **Absolute numerical deviation**

- Two copies do not deviate by more than 0.5%
  
  - **Relative numerical deviation**

- If stock goes up and one replica is updated
  
  - If change *does not* violate specified deviations?
  
    - Replicas are considered consistent

Numerical and Staleness deviations

- Numerical deviation can also be expressed in terms of number of updates
  
    - Applied at a replica, but not seen by other replicas

- **Staleness** deviations
  
    - Last time a replica was updated
    
    - Replica can provide *old data* as long as it is *not too old*
      
      - Weather reports
Ordering of updates may also be allowed to be different

- Within a certain **bound**
- Updates applied *tentatively* at local copy
  - Need **global agreement** with all replicas
- Before an update becomes **permanent**
  - Might be rolled back
  - Applied in a different order

**CONSISTENCY UNIT (CONIT)**
Consistency Unit: conit

- Specifies *unit* over which consistency is to be measured

- Examples
  - Record representing a stock
  - Weather report

Looking at the conit a little closer:
Example with 2 replicas

- Each replica maintains a 2D vector clock
- Operation carried out by replica *i* at (its) logical time \( t : <t, i> \)
- Example conit contains data items \( x \) and \( y \)
Tracking consistency deviations:
Conit items $x$ and $y$ are initialized to 0

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$&lt;5, B&gt;$</td>
<td>$x = x+2$ [ $x = 2$ ]</td>
</tr>
<tr>
<td>$&lt;8, A&gt;$</td>
<td>$y = y+2$ [ $y = 2$ ]</td>
</tr>
<tr>
<td>$&lt;12, A&gt;$</td>
<td>$y = y+1$ [ $y = 3$ ]</td>
</tr>
<tr>
<td>$&lt;14, A&gt;$</td>
<td>$x = yx2$ [ $x = 6$ ]</td>
</tr>
</tbody>
</table>

Conit: $x=6$, $y=3$

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$&lt;5, B&gt;$</td>
<td>$x = x+2$ [ $x = 2$ ]</td>
</tr>
<tr>
<td>$&lt;10, B&gt;$</td>
<td>$y = y+5$ [ $y = 5$ ]</td>
</tr>
</tbody>
</table>

Tentative Operation:

Vector Clocks at each replica

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$&lt;5, B&gt;$</td>
<td>$x = x+2$ [ $x = 2$ ]</td>
</tr>
<tr>
<td>$&lt;8, A&gt;$</td>
<td>$y = y+2$ [ $y = 2$ ]</td>
</tr>
<tr>
<td>$&lt;12, A&gt;$</td>
<td>$y = y+1$ [ $y = 3$ ]</td>
</tr>
<tr>
<td>$&lt;14, A&gt;$</td>
<td>$x = yx2$ [ $x = 6$ ]</td>
</tr>
</tbody>
</table>

Vector clock A = (15, 5)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$&lt;5, B&gt;$</td>
<td>$x = x+2$ [ $x = 2$ ]</td>
</tr>
<tr>
<td>$&lt;10, B&gt;$</td>
<td>$y = y+5$ [ $y = 5$ ]</td>
</tr>
</tbody>
</table>

Vector clock B = (0, 11)
Order deviations are the number of tentative operations at each replica

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;5, B&gt;</td>
<td>x = x+2</td>
</tr>
<tr>
<td>&lt;8, A&gt;</td>
<td>y = y+2</td>
</tr>
<tr>
<td>&lt;12, A&gt;</td>
<td>y = y+1</td>
</tr>
<tr>
<td>&lt;14, A&gt;</td>
<td>x = yx2</td>
</tr>
<tr>
<td><strong>Order Deviation = 3</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;5, B&gt;</td>
<td>x = x+2</td>
</tr>
<tr>
<td>&lt;10, B&gt;</td>
<td>y = y+5</td>
</tr>
<tr>
<td><strong>Order Deviation = 2</strong></td>
<td></td>
</tr>
</tbody>
</table>

Numerical deviations in our example

- Numerical deviation here is the number of *unseen updates* from the other replica
- Weight of this deviation at replica A is the maximum *difference* between
  - Committed values of conit at A
  - Result from operations at B not seen by A
Quantifying the numerical deviations at each replica

Replica A

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;5, B&gt;</td>
<td>x = x+2 [x = 2]</td>
</tr>
<tr>
<td>&lt;8, A&gt;</td>
<td>y = y+2 [y = 2]</td>
</tr>
<tr>
<td>&lt;12, A&gt;</td>
<td>y = y+1 [y = 3]</td>
</tr>
<tr>
<td>&lt;14, A&gt;</td>
<td>x = xy2 [x = 6]</td>
</tr>
</tbody>
</table>

Unseen Updates = 1
Weight = Max[diff(2,2), diff(0,5)]
= 5

Replica B

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;5, B&gt;</td>
<td>x = x+2 [x = 2]</td>
</tr>
<tr>
<td>&lt;10, B&gt;</td>
<td>y = y+5 [y = 5]</td>
</tr>
</tbody>
</table>

Note: B’s committed value is (0,0)

Unseen updates = 3
Weight = Max[diff(0,6), diff(0,3)]
= 6

Tradeoffs between fine grained and coarse grained conits

- If conit represents a lot of data
  - Updates aggregated for all data in conit
  - Replicas become inconsistent sooner

- If conit is smaller
  - Fewer updates needed
  - Total number of conits to be managed goes up
Before we put conits to practical use two things need to happen

- **Protocols** to enforce consistency
- Developers **specify** consistency requirements
  - Difficult!

Conits are declared alongside updates

AffectsConit(ConitQ, 1, 1)
append message m to queue Q

- Appending message m to queue Q belongs to a conit named ConitQ
Conits are declared alongside reads

```
DependsOnConit(ConitQ, 4, 0, 60)
```

- Numerical deviation: 4
  - At most 4 unseen updates at other replicas
- Ordering deviation: 0
  - No tentative local updates
- Staleness deviation: 60 seconds
  - Check Q for staleness periodically
Consistent ordering of operations

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

Sequential consistency: Notations

- 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
Two processes operating on the same data item

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

Time to propagate update of x to P2 is acceptable 😊

---

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

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

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

Sequential consistency: Example

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

P3 concludes final value is a
P4 concludes final value is b
Unacceptable
Sequential Consistency:
Another example

<table>
<thead>
<tr>
<th>Process 1</th>
<th>Process 2</th>
<th>Process 3</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>
</tbody>
</table>

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 print(x,z) before y=1
    - Half print(x,y) before z=1
    - Only ¼ 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

**Signature** is the concatenation of the outputs of P1, P2 and P3

<table>
<thead>
<tr>
<th>$x = 1$</th>
<th>$x = 1$</th>
<th>$y = 1$</th>
<th>$y = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>print($y, z$)</td>
<td>print($z, x$)</td>
<td>print($z, x$)</td>
<td>print($x, z$)</td>
</tr>
<tr>
<td>$y = 1$</td>
<td>$y = 1$</td>
<td>$z = 1$</td>
<td>$z = 1$</td>
</tr>
<tr>
<td>print($x, z$)</td>
<td>print($y, z$)</td>
<td>print($x, y$)</td>
<td>print($x, z$)</td>
</tr>
<tr>
<td>$z = 1$</td>
<td>$z = 1$</td>
<td>$x = 1$</td>
<td>$x = 1$</td>
</tr>
<tr>
<td>print($x, y$)</td>
<td>print($x, y$)</td>
<td>print($y, z$)</td>
<td>print($x, y$)</td>
</tr>
</tbody>
</table>

Prints: 001011  Prints: 101011  Prints: 010111  Prints: 111111
Signature: 001011  Signature: 101011  Signature: 110101  Signature: 111111

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

- **000000?**
  - Print statements ran before assignments
  - Violates program order

- **001001?**
  - \{00\} y and z were 0 when P1 did its printing
  - P1 executes its statements before P2 and P3 start
  - \{10\} P2 ran after P1 started, but before P3 started
  - \{01\} P3 must complete before P1 starts
  - Not possible!

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