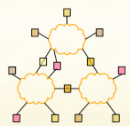
15-446 Distributed Systems Spring 2009



L-10 Consistency

1

Important Lessons

- Replication → good for performance/ reliability
  - Key challenge → keeping replicas up-to-date
- Wide range of consistency models
  - Will see more next lecture
  - Range of correctness properties
- Most obvious choice (sequential consistency) can be expensive to implement
  - Multicast, primary, quorum

2

Today's Lecture

- ACID vs. BASE – philosophy
- Client-centric consistency models
- Eventual consistency
- Bayou

3

Two Views of Distributed Systems

- **Optimist**: A distributed system is a collection of independent computers that appears to its users as a single coherent system
- **Pessimist**: “You know you have one when the crash of a computer you’ve never heard of stops you from getting any work done.” (Lamport)

4

Recurring Theme

- Academics like:
  - Clean abstractions
  - Strong semantics
  - Things that prove they are smart
- Users like:
  - Systems that work (most of the time)
  - Systems that scale
  - Consistency *per se* isn't important
- Eric Brewer had the following observations

5

A Clash of Cultures

- Classic distributed systems: focused on ACID semantics (transaction semantics)
  - Atomicity: either the operation (e.g., write) is performed on all replicas or is not performed on any of them
  - Consistency: after each operation all replicas reach the same state
  - Isolation: no operation (e.g., read) can see the data from another operation (e.g., write) in an intermediate state
  - Durability: once a write has been successful, that write will persist indefinitely
- Modern Internet systems: focused on BASE
  - Basically Available
  - Soft-state (or scalable)
  - Eventually consistent

6

ACID vs BASE

ACID

- Strong consistency for transactions highest priority
- Availability less important
- Pessimistic
- Rigorous analysis
- Complex mechanisms

BASE

- Availability and scaling highest priorities
- Weak consistency
- Optimistic
- Best effort
- Simple and fast

7

Why Not ACID+BASE?

- What goals might you want from a system?
  - C, A, P
- **Strong Consistency**: all clients see the same view, even in the presence of updates
- **High Availability**: all clients can find some replica of the data, even in the presence of failures
- **Partition-tolerance**: the system properties hold even when the system is partitioned

8

CAP Theorem [Brewer]

- You can only have two out of these three properties
- The choice of which feature to discard determines the nature of your system

9

Consistency and Availability

- Comment:
  - Providing transactional semantics requires all functioning nodes to be in contact with each other (no partition)
- Examples:
  - Single-site and clustered databases
  - Other cluster-based designs
- Typical Features:
  - Two-phase commit
  - Cache invalidation protocols
  - Classic DS style

10

Partition-Tolerance and Availability

- Comment:
  - Once consistency is sacrificed, life is easy….
- Examples:
  - DNS
  - Web caches
  - Practical distributed systems for mobile environments: Coda, Bayou
- Typical Features:
  - Optimistic updating with conflict resolution
  - This is the "Internet design style"
  - TTLs and lease cache management

11

Voting with their Clicks

- In terms of large-scale systems, the world has voted with their clicks:
  - Consistency less important than availability and partition-tolerance

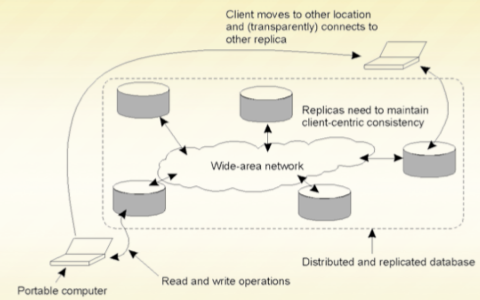
12

Today's Lecture

- ACID vs. BASE – philosophy
- Client-centric consistency models
- Eventual consistency
- Bayou

13

Client-centric Consistency Models



- A mobile user may access different replicas of a distributed database at different times. This type of behavior implies the need for a view of consistency that provides guarantees for single client regarding accesses to the data store.

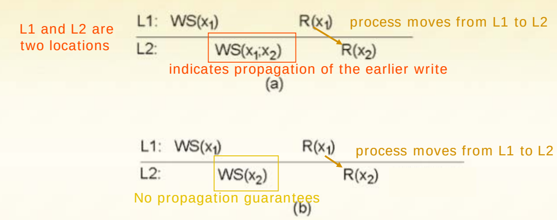
14

Session Guarantees

- When client move around and connects to different replicas, strange things can happen
  - Updates you just made are missing
  - Database goes back in time
- Responsibility of "session manager", not servers
- Two sets:
  - Read-set: set of writes that are relevant to session reads
  - Write-set: set of writes performed in session
- Update dependencies captured in read sets and write sets
- Four different client-central consistency models
  - Monotonic reads
  - Monotonic writes
  - Read your writes
  - Writes follow reads

15

Monotonic Reads



A data store provides monotonic read consistency if when a process reads the value of a data item x, any successive read operations on x by that process will always return the same value or a more recent value.

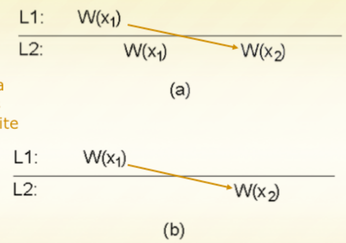
Example error: successive access to email have 'disappearing messages'

a) A monotonic-read consistent data store
b) A data store that does not provide monotonic reads.

16

Monotonic Writes

In both examples, process performs a write at L1, moves and performs a write at L2



A write operation by a process on a data item x is completed before any successive write operation on x by the same process. Implies a copy must be up to date before performing a write on it.

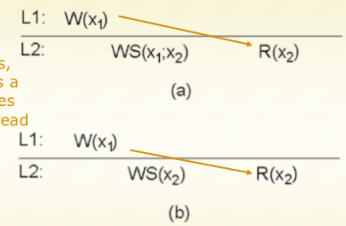
Example error: Library updated in wrong order.

a) A monotonic-write consistent data store.
b) A data store that does not provide monotonic-write consistency.

17

Read Your Writes

In both examples, process performs a write at L1, moves and performs a read at L2



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.

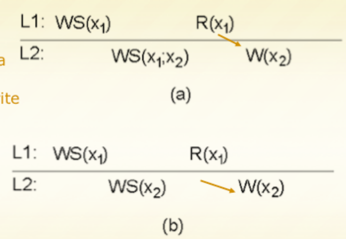
Example error: deleted email messages re-appear.

a) A data store that provides read-your-writes consistency.
b) A data store that does not.

18

Writes Follow Reads

In both examples, process performs a read at L1, moves and performs a write at L2



A write operation by a process on a 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.

Example error: Newsgroup displays responses to articles before original article has propagated there

a) A writes-follow-reads consistent data store
b) A data store that does not provide writes-follow-reads consistency

19

Today's Lecture

- ACID vs. BASE – philosophy
- Client-centric consistency models
- Eventual consistency
- Bayou

20

Many Kinds of Consistency

- Strict: updates happen instantly everywhere
  - A read has to return the result of the latest write which occurred on that data item
  - Assume instantaneous propagation; not realistic
- Linearizable: updates appear to happen instantaneously at some point in time
  - Like "Sequential" but operations are ordered using a global clock
  - Primarily used for formal verification of concurrent programs
- Sequential: all updates occur in the same order everywhere
  - Every client sees the writes in the same order
    - Order of writes from the same client is preserved
    - Order of writes from different clients may not be preserved
  - Equivalent to Atomicity + Consistency + Isolation
- Eventual consistency: if all updating stops then eventually all replicas will converge to the identical values

21

Eventual Consistency

- There are replica situations where updates (writes) are rare and where a fair amount of inconsistency can be tolerated.
  - DNS – names rarely changed, removed, or added and changes/additions/removals done by single authority
  - Web page update – pages typically have a single owner and are updated infrequently.
- If no updates occur for a while, all replicas should gradually become consistent.
- May be a problem with mobile user who access different replicas (which may be inconsistent with each other).

22

Why (not) eventual consistency?

- Support disconnected operations
  - Better to read a stale value than nothing
  - Better to save writes somewhere than nothing
- Potentially anomalous application behavior
  - Stale reads and conflicting writes…

23

Implementing Eventual Consistency

Can be implemented with two steps:

1. All writes eventually propagate to all replicas

2. Writes, when they arrive, are written to a log and applied in the same order at all replicas
   - Easily done with timestamps and "undo-ing" optimistic writes

24

Update Propagation

- Rumor or epidemic stage:
  - Attempt to spread an update quickly
  - Willing to tolerate incompletely coverage in return for reduced traffic overhead
- Correcting omissions:
  - Making sure that replicas that weren't updated during the rumor stage get the update

25

Anti-Entropy

- Every so often, two servers compare complete datasets
- Use various techniques to make this cheap
- If any data item is discovered to not have been fully replicated, it is considered a new rumor and spread again

26

Today's Lecture

- ACID vs. BASE – philosophy
- Client-centric consistency models
- Eventual consistency
- Bayou

27

System Assumptions

- Early days: nodes always on when not crashed
  - Bandwidth always plentiful (often LANs)
  - Never needed to work on a disconnected node
  - Nodes never moved
  - Protocols were "chatty"
- Now: nodes detach then reconnect elsewhere
  - Even when attached, bandwidth is variable
  - Reconnection elsewhere means often talking to different replica
  - Work done on detached nodes

28

Disconnected Operation

- Challenge to old paradigm
  - Standard techniques disallowed any operations while disconnected
  - Or disallowed operations by others
- But eventual consistency not enough
  - Reconnecting to another replica could result in strange results
    - E. g., not seeing your own recent writes
  - Merely letting latest write prevail may not be appropriate
  - No detection of read-dependencies
- What do we do?

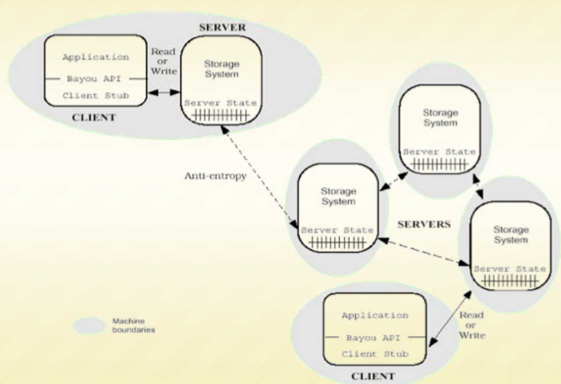
29

Bayou

- System developed at PARC in the mid-90's
- First coherent attempt to fully address the problem of disconnected operation
- Several different components

30

Bayou Architecture



31

Motivating Scenario: Shared Calendar

- Calendar updates made by several people
  - e.g., meeting room scheduling, or exec+admin
- Want to allow updates offline
- But conflicts can't be prevented
- Two possibilities:
  - Disallow offline updates?
  - Conflict resolution?

32

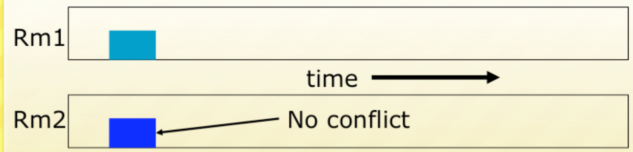
Conflict Resolution

- Replication **not** transparent to application
  - Only the application knows how to resolve conflicts
  - Application can do record-level conflict detection, not just file-level conflict detection
  - Calendar example: record-level, and easy resolution
- Split of responsibility:
  - Replication system: propagates updates
  - Application: resolves conflict
- Optimistic application of writes requires that writes be "undo-able"

33

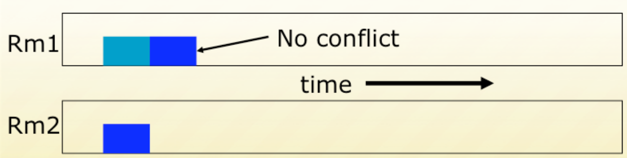
Meeting room scheduler

- Reserve same room at same time: conflict
- Reserve different rooms at same time: no conflict
- Reserve same room at different times: no conflict
- Only the application would know this!



34

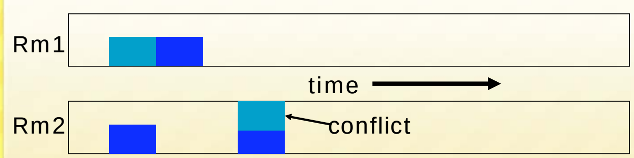
Meeting Room Scheduler



35

Meeting Room Scheduler

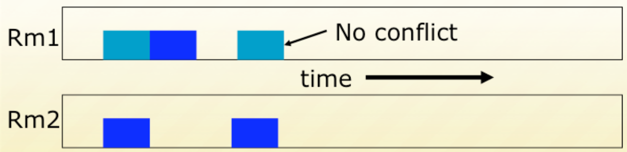
- Conflict detection



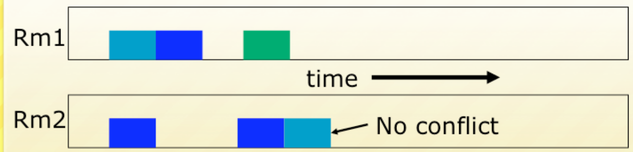
36

Meeting Room Scheduler

- Automated resolution



Meeting Room Scheduler



Other Resolution Strategies

- Classes take priority over meetings
- Faculty reservations are bumped by admin reservations
- Move meetings to bigger room, if available
- Point:
  - Conflicts are detected at very fine granularity
  - Resolution can be policy-driven

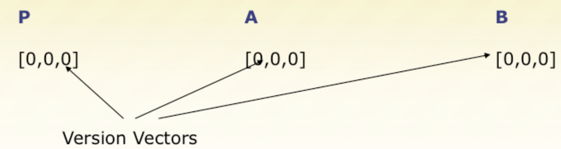
Updates

- Client sends update to a server
- Identified by a triple:
  - <Commit-stamp, Time-stamp, Server-ID of accepting server>
- Updates are either committed or tentative
  - Commit-stamps increase monotonically
  - Tentative updates have commit-stamp = inf

Anti-Entropy Exchange

- Each server keeps a vector timestamp
- When two servers connect, exchanging the version vectors allows them to identify the missing updates
- These updates are exchanged in the order of the logs, so that if the connection is dropped the crucial monotonicity property still holds
  - If a server X has an update accepted by server Y, server X has all previous updates accepted by that server

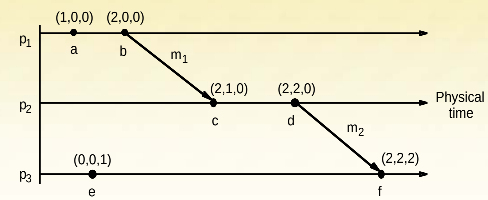
Example with Three Servers



Vector Clocks

- Vector clocks overcome the shortcoming of Lamport logical clocks
  - *L*(e) < *L*(*e*') does not imply *e* happened before *e*'
- Vector timestamps are used to timestamp local events
- They are applied in schemes for replication of data

Vector Clocks



- How to ensure causality?
- Two rules for delaying message processing:
  1. VC must indicate that this is next message from source
  2. VC must indicate that you have all the other messages that "caused" this message

All Servers Write Independently

P	A	B
<inf,1,P> <inf,4,P> <inf,8,P>	<inf,2,A> <inf,3,A> <inf,10,A>	<inf,1,B> <inf,5,B> <inf,9,B>
[8,0,0]	[0,10,0]	[0,0,9]

45

Bayou Writes

- Identifier (commit-stamp, time-stamp, server-ID)
- Nominal value
- Write dependencies
- Merge procedure

46

Conflict Detection

- Write specifies the data the write depends on:
  - Set X=8 if Y=5 and Z=3
  - Set Cal(11:00-12:00)=dentist if Cal(11:00-12:00) is null
- These write dependencies are crucial in eliminating unnecessary conflicts
  - If file-level detection was used, all updates would conflict with each other

47

Conflict Resolution

- Specified by merge procedure (mergeproc)
- When conflict is detected, mergeproc is called
  - Move appointments to open spot on calendar
  - Move meetings to open room

48

P and A Do Anti-Entropy Exchange

P	A	B
<inf,1,P>	<inf,1,P>	<inf,1,B>
<inf,2,A>	<inf,2,A>	<inf,5,B>
<inf,3,A>	<inf,3,A>	<inf,9,B>
<inf,4,P>	<inf,4,P>	
<inf,8,P>	<inf,8,P>	[0,0,9]
<inf,10,A>	<inf,10,A>	
[8,10,0]	[8,10,0]	
<inf,1,P>	<inf,2,A>	
<inf,4,P>	<inf,3,A>	
<inf,8,P>	<inf,10,A>	
[8,0,0]	[0,10,0]	

49

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

50

P Commits Some Early Writes

P	A	B
<1,1,P>	<inf,1,P>	<inf,1,B>
<2,2,A>	<inf,2,A>	<inf,5,B>
<3,3,A>	<inf,3,A>	<inf,9,B>
<inf,4,P>	<inf,4,P>	
<inf,8,P>	<inf,8,P>	[0,0,9]
<inf,10,A>	<inf,10,A>	
[8,10,0]	[8,10,0]	
<inf,1,P>		
<inf,2,A>		
<inf,3,A>		
<inf,4,P>		
<inf,8,P>		
<inf,10,A>		
[8,10,0]		

51

P and B Do Anti-Entropy Exchange

P	A	B
<1,1,P>	<inf,1,P>	<1,1,P>
<2,2,A>	<inf,2,A>	<2,2,A>
<3,3,A>	<inf,3,A>	<3,3,A>
<inf,1,B>	<inf,4,P>	<inf,1,B>
<inf,4,P>	<inf,8,P>	<inf,4,P>
<inf,5,B>	<inf,10,A>	<inf,5,B>
<inf,8,P>		<inf,8,P>
<inf,9,B>	[8,10,0]	<inf,9,B>
<inf,10,A>		<inf,10,A>
[8,10,9]		[8,10,9]
<1,1,P>		<inf,1,B>
<2,2,A>		<inf,5,B>
<3,3,A>		<inf,9,B>
<inf,4,P>		
<inf,8,P>		[0,0,9]
<inf,10,A>		

52

P Commits More Writes

P		P
<1,1,P>		<1,1,P>
<2,2,A>		<2,2,A>
<3,3,A>		<3,3,A>
<inf,1,B>	→	<4,1,B>
<inf,4,P>		<5,4,P>
<inf,5,B>		<6,5,B>
<inf,8,P>		<7,8,P>
<inf,9,B>		<inf,9,B>
<inf,10,A>		<inf,10,A>
[8,10,9]		[8,10,9]

53

Bayou Summary

- Simple gossip based design
- Key difference → exploits knowledge of application semantics
  - To identify conflicts
  - To handle merges
- Greater complexity for the programmer
  - Might be useful in ubicomp context

54

Important Lessons

- ACID vs. BASE
  - Understand the tradeoffs you are making
  - ACID makes things better for programmer/system designed
  - BASE often preferred by users
- Client-centric consistency
  - Different guarantees than data-centric
- Eventual consistency
  - BASE-like design → better performance/availability
  - Must design system to tolerate
  - Bayou a good example of making tolerance explicit

55