CS 194: Lecture 10

Bayou, Brewer, and Byzantine

1

Agenda

- Review of Bayou
- Channeling Eric Brewer (CAP theorem)
- A peek at fault tolerance

2

Review of Bayou

With examples!

3

Why Bayou?

- Eventual consistency: strongest scalable consistency model
- But not strong enough for mobile clients
  - Accessing different replicas can lead to strange results
  - Application-independent conflict detection misses some conflicts and creates others falsely
- Bayou was designed to move beyond eventual consistency
  - Session guarantees
  - Application-specific conflict detection and resolution

4

Bayou System Assumptions

- Variable degrees of connectivity:
  - Connected, disconnected, and weakly connected
- Variable end-node capabilities:
  - Workstations, laptops, PDAs, etc.
- Availability crucial

5

Resulting Design Choices

- Variable connectivity ⇒ Flexible update propagation
  - Incremental progress, pairwise communication
- Variable end-nodes ⇒ Flexible notion of clients and servers
  - Some nodes keep state (servers), some don't (clients)
  - Laptops could have both, PDAs probably just clients
- Availability crucial ⇒ Must allow disconnected operation
  - Conflicts inevitable
  - Use application-specific conflict detection and resolution

6

Components of Design

- Update propagation
- Conflict detection
- Conflict resolution
- Session guarantees

7

Updates

- 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
- Primary server does all commits: (why?)
  - It sets the commit-stamp
  - Commit-stamp different from time-stamp

8

Update Log

- Update log in order:
  - Committed updates (in commit-stamp order)
  - Tentative updates (in time-stamp order)
- Can truncate committed updates, and only keep db state
  - Why?
- Clients can request two views: (or other app-specific views)
  - Committed view
  - Tentative view

9

Tentative vs Committed Views

- Committed view:
  - Updates will never be reordered
  - But may be substantially out-of-date
- Tentative view:
  - Much more current
  - But updates might be reordered
- Tradeoff is application-dependent:
  - Calendars: avoid tentative commitments, but don't count on them
  - Weather: being current more important than permanence

10

Anti-Entropy Exchange

- Each server keeps a version vector:
  - R.V[X] is the latest timestamp from server X that server R has seen
- 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

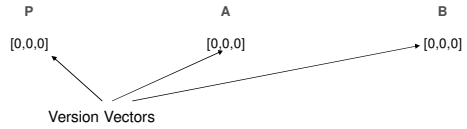
11

Requirements for Eventual Consistency

- Universal propagation: anti-entropy
- Globally agreed ordering: commit-stamps
- Determinism: writes do not involve information not contained in the log (no time-of-day, process-ID, etc.)

12

Example with Three Servers



13

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]

14

P and A Do Anti-Entropy Exchange

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

15

P Commits Some Early Writes

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

16

P and B Do Anti-Entropy Exchange

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

17

P Commits More Writes

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

18

Bayou Writes

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

19

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

20

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

21

Session Guarantees

- Ensured by client, not by distribution mechanism
- Needed to ensure user sees sensible results
- To implement, client records:
  - All writes during that session (write-set)
  - The writes relevant to each read read-set)
    - Must be supplied by server
    - Can be approximated by version vector

22

The Four Session Guarantees

Guarantee	State updated	State checked
Read your writes	Write	Read
Monotonic reads	Read	Read
Writes follow reads	Read	Write
Monotonic writes	Write	Write

23

Example

- Return to example with servers P, A, and B
- Client attaches to server P with vector [8,3,5]
- Client reads, with read-set {P6,A1,A2,B5}
- Client writes, with timestamp P9
- Client then detaches and reattaches to another server
- For which of these vectors can client read or write?

24

What Reads/Writes are Allowed?

Read-set {P6,A1,A2,B5}, Write-set P9

- [7,1,6] Read Your Writes: No
  Monotonic Reads: No
  Writes Following Reads: No
  Monotonic Writes: No
  No R, No W
- [7,4,6] Read Your Writes: No
  Monotonic Reads: Yes
  Writes Following Reads: Yes
  Monotonic Writes: No
  No R, No W

25

What Reads/Writes are Allowed?

Read-set {P6,A1,A2,B5}, Write-set P9

- [9,3,4] Read Your Writes: Yes
  Monotonic Reads: No
  Writes Following Reads: No
  Monotonic Writes: Yes
  No R, No W
- [10,3,8] Read Your Writes: Yes
  Monotonic Reads: Yes
  Writes Following Reads: Yes
  Monotonic Writes: Yes
  R, W

26

Channeling Eric Brewer

Slightly more hair, much less wisdom

27

A Clash of Cultures

- Classic distributed systems: focused on ACID semantics
  - A: Atomic
  - C: Consistent
  - I: Isolated
  - D: Durable
- Modern Internet systems: focused on BASE
  - Basically Available
  - Soft-state (or scalable)
  - Eventually consistent

28

ACID vs BASE

ACID	BASE
Strong consistency for transactions highest priority	Availability and scaling highest priorities
Availability less important	Weak consistency
Pessimistic	Optimistic
Rigorous analysis	Best effort
Complex mechanisms	Simple and fast

29

Why the Divide?

- What goals might you want from a shared-date 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

30

CAP Conjecture (later theorem)

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

31

Consistency and Availability

- Comment:
  - Providing transactional semantics requires all nodes to be in contact with each other
- Examples:
  - Single-site and clustered databases
  - Other cluster-based designs
- Typical Features:
  - Two-phase commit
  - Cache invalidation protocols
  - Classic DS style

32

Consistency and Partition-Tolerance

- Comment:
  - If one is willing to tolerate system-wide blocking, then can provide consistency even when there are temporary partitions
- Examples:
  - Distributed databases
  - Distributed locking
  - Quorum (majority) protocols
- Typical Features:
  - Pessimistic locking
  - Minority partitions unavailable
  - Also common DS style
    - Voting vs primary replicas

33

Partition-Tolerance and Availability

- Comment:
  - Once consistency is sacrificed, life is easy….
- Examples:
  - DNS
  - Web caches
  - Coda
  - Bayou
- Typical Features:
  - TTLs and lease cache management
  - Optimistic updating with conflict resolution
  - This is the “Internet design style”

34

Techniques

- Expiration-based caching: AP
- Quorum/majority algorithms: PC
- Two-phase commit: AC

35

Byzantine

36

Failures

- So far, have assume nodes are either up or down
- But nodes are far more interesting than that!

37

Failure Models

Type of failure	Description
Crash failure	A server halts, but is working correctly until it halts
Omission failure *Receive omission* *Send omission*	A server fails to respond to incoming requests A server fails to receive incoming messages A server fails to send messages
Timing failure	A server's response lies outside the specified time interval
Response failure *Value failure* *State transition failure*	The server's response is incorrect The value of the response is wrong The server deviates from the correct flow of control
Arbitrary failure	A server may produce arbitrary responses at arbitrary times

38

Previous Algorithms

- Only cope with crash-failure
- What happens if some other failure occurs?
- Bayou as an example:
  - If server lies about updates, algorithm gets hopelessly confused
- Generally, most other distributed protocols fail when faced with anything other than crash failures
- Next: how to deal with a wider variety of failures

39

Same Dichotomy Exists

- Classic Distributed Systems:
  - Byzantine Algorithms
  - Two-phase Commit
- Internet style:
  - Checkable or “self-verifying” protocols
  - Very new field in Internet research
  - You now know as much as we do about it…..

40