15-446 Distributed Systems
Spring 2009

L-12 CDNs

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

Why Not ACID+BASE?

- What goals might you want from a system?
  - C, A, P – but you can only have 2

- 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

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.
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.

Monotonic Writes

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.

Read Your Writes

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.

Writes Follow Reads

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.
Motivating Scenario: Shared Calendar

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

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”

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!

Rm1

Rm2

No conflict
Meeting Room Scheduler

13

Meeting Room Scheduler

- Conflict detection

14

Meeting Room Scheduler

- Automated resolution

15

Meeting Room Scheduler

16
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:
  - \(<\text{Commit-stamp}, \text{Time-stamp}, \text{Server-ID of accepting server}>\>
- Updates are either committed or tentative
  - Commit-stamps increase monotonically
  - Tentative updates have commit-stamp = \(\infty\)

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

**Version Vectors**

- P: \([0,0,0]\)
- A: \([9,0,0]\)
- B: \([0,0,0]\)

All Servers Write Independently

**Identifier** (commit-stamp, time-stamp, server-ID)

- P: \(<inf,1,P>\), \(<inf,4,P>\), \(<inf,8,P>\)
- A: \(<inf,2,A>\), \(<inf,3,A>\), \(<inf,10,A>\)
- B: \(<inf,1,B>\), \(<inf,5,B>\), \(<inf,9,B>\)

Conflict Detection

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

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
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
**P Commits More Writes**

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

- 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>
  - [8,10,9]

**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

**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

**Today’s Lecture**

- CDNs
HTTP Caching

- Clients often cache documents
  - Challenge: update of documents
  - If-Modified-Since requests to check
    - HTTP 0.9/1.0 used just date
    - HTTP 1.1 has an opaque "entity tag" (could be a file signature, etc.) as well
- When/how often should the original be checked for changes?
  - Check every time?
  - Check each session? Day? Etc?
  - Use Expires header
    - If no Expires, often use Last-Modified as estimate

Example Cache Check Request

GET / HTTP/1.1
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
If-Modified-Since: Mon, 29 Jan 2001 17:54:18 GMT
If-None-Match: "7a11f-10ed-3a75ae4a"
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
Host: www.intel-iris.net
Connection: Keep-Alive

Example Cache Check Response

HTTP/1.1 304 Not Modified
Date: Tue, 27 Mar 2001 03:50:51 GMT
Server: Apache/1.3.14 (Unix) (Red-Hat/Linux) mod_ssl/2.7.1 OpenSSL/0.9.5a DAV/1.0.2 PHP/4.0.1pl2 mod_perl/1.24
Connection: Keep-Alive
Keep-Alive: timeout=15, max=100
ETag: "7a11f-10ed-3a75ae4a"

Content Distribution Networks (CDNs)

- The content providers are the CDN customers.
- Content replication CDN company installs hundreds of CDN servers throughout Internet
  - Close to users
- CDN replicates its customers’ content in CDN servers. When provider updates content, CDN updates servers
Server Selection

- Service is replicated in many places in network.
- How do direct clients to a particular server?
  - As part of routing ➔ anycast, cluster load balancing.
  - As part of application ➔ HTTP redirect.
  - As part of naming ➔ DNS.
- Which server?
  - Lowest load ➔ to balance load on servers.
  - Best performance ➔ to improve client performance.
  - Based on Geography? RTT? Throughput? Load?
  - Any alive node ➔ to provide fault tolerance.

Routing Based

- Anycast
  - Give service a single IP address.
  - Each node implementing service advertises route to address.
  - Packets get routed routed from client to “closest” service node.
  - Closest is defined by routing metrics.
  - May not mirror performance/application needs.
  - What about the stability of routes?

Application Based

- HTTP support simple way to indicate that Web page has moved.
- Server gets Get request from client.
  - Decides which server is best suited for particular client and object.
  - Returns HTTP redirect to that server.
- Can make informed application specific decision.
- May introduce additional overhead ➔ multiple connection setup, name lookups, etc.
- While good solution in general HTTP Redirect has some design flaws – especially with current browsers.

Naming Based

- Client does name lookup for service.
- Name server chooses appropriate server address.
  - A-record returned is “best” one for the client.
- What information can name server base decision on?
  - Server load/location ➔ must be collected.
  - Information in the name lookup request.
  - Name service client ➔ typically the local name server for client.
How Akamai Works

- Clients fetch html document from primary server
  - E.g. fetch index.html from cnn.com
- URLs for replicated content are replaced in html
  - E.g. `<img src="http://cnn.com/af/x.gif">` replaced with `<img src="http://a73.g.akamaitech.net/7/23/cnn.com/af/x.gif">`
- Client is forced to resolve aXYZ.g.akamaitech.net hostname

How Akamai Works

- How is content replicated?
- Akamai only replicates static content (*)
- Modified name contains original file name
- Akamai server is asked for content
  - First checks local cache
  - If not in cache, requests file from primary server and caches file
* (At least, the version we’re talking about today. Akamai actually lets sites write code that can run on Akamai’s servers, but that’s a pretty different beast)

Root server gives NS record for akamai.net
- Akamai.net name server returns NS record for g.akamaitech.net
  - Name server chosen to be in region of client’s name server
  - TTL is large
- G.akamaitech.net nameserver chooses server in region
  - Should try to chose server that has file in cache - How to choose?
  - Uses aXYZ name and hash
  - TTL is small → why?

Simple Hashing

- Given document XYZ, we need to choose a server to use
- Suppose we use modulo
- Number servers from 1...n
  - Place document XYZ on server (XYZ mod n)
  - What happens when a servers fails? n → n-1
    - Same if different people have different measures of n
  - Why might this be bad?
Simple Hashing

- Given document XYZ, we need to choose a server to use
- Suppose we use modulo
  - Number servers from 1...n
    - Place document XYZ on server (XYZ mod n)
    - What happens when a server fails? n \rightarrow n-1
    - Same if different people have different measures of n
    - Why might this be bad?

Consistent Hash

- "view" = subset of all hash buckets that are visible
- Desired features
  - Balanced – in any one view, load is equal across buckets
  - Smoothness – little impact on hash bucket contents when buckets are added/removed
  - Spread – small set of hash buckets that may hold an object regardless of views
  - Load – across all views # of objects assigned to hash bucket is small

Consistent Hash – Example

- Construction
  - Assign each of C hash buckets to random points on mod 2^n circle, where, hash key size = n.
  - Map object to random position on circle
    - Hash of object = closest clockwise bucket
  - Smoothness \rightarrow addition of bucket does not cause movement between existing buckets
  - Spread & Load \rightarrow small set of buckets that lie near object
  - Balance \rightarrow no bucket is responsible for large number of objects

How Akamai Works

End-user \rightarrow cnn.com (content provider) DNS root server Akamai server

- Get index.html
- Get /cnn.com/foo.jpg
- Akamai high-level DNS server
- Akamai low-level DNS server
- Nearby matching Akamai server
Akamai – Subsequent Requests

cnn.com (content provider) ← DNS root server ← Akamai high-level DNS server ← Akamai low-level DNS server ← Nearby matching Akamai server ← Get foo.jpg

Important Lessons

- Akamai CDN illustrate range of ideas:
  - BASE (not ACID design)
  - Weak consistency
  - Naming of objects → location translation
  - Consistent hashing

- Why are these the right design choices for this application?