15-440 Distributed Systems

Naming and Hashing
Overview

• Spanner
• DNS
• DNS today
• Naming with hashes
What is Spanner?

- Distributed multiversion database
  - General-purpose transactions (ACID)
  - SQL query language
  - Schematized tables
  - Semi-relational data model
Example: Social Network

OSDI 2012
Read Transactions

• Generate a page of friends’ recent posts
  – Consistent view of friend list and their posts

Why consistency matters
1. Remove untrustworthy person X as friend
2. Post P: “My government is repressive...”
Single Machine

Block writes

Friend1 post
Friend2 post
...
Friend999 post
Friend1000 post

User posts
Friend lists

Generate my page
Multiple Machines

Block writes

Friend1 post
Friend2 post
...
Friend999 post
Friend1000 post

User posts
Friend lists

Generate my page

OSDI 2012
Multiple Datacenters

Friend1 post
US

Friend2 post
Spain
...

Friend999 post
Brazil

Friend1000 post
Russia

Generate my page

OSDI 2012
Key aspect of differentiating Spanner – using globally meaningful timestamps for distributed transactions in achieving external consistency.
### Version Management

- **Transactions that write use strict 2PL**
  - Each transaction $T$ is assigned a timestamp $s$
  - Data written by $T$ is timestamped with $s$

<table>
<thead>
<tr>
<th>Time</th>
<th>&lt;8</th>
<th>8</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>My friends</td>
<td>[X]</td>
<td>[]</td>
<td></td>
</tr>
<tr>
<td>My posts</td>
<td></td>
<td></td>
<td>[P]</td>
</tr>
<tr>
<td>X’s friends</td>
<td>[me]</td>
<td>[]</td>
<td></td>
</tr>
</tbody>
</table>
Synchronizing Snapshots

Global wall-clock time

==

External Consistency:
Commit order respects global wall-time order

==

Timestamp order respects global wall-time order
given
timestamp order == commit order
Timestamps, Global Clock

- **Strict two-phase locking for write transactions**
- **Assign timestamp while locks are held**

[Diagram showing timeline with events: Acquired locks, Pick $s = \text{now()}$, Release locks]
TrueTime

• “Global wall-clock time” with bounded uncertainty

\[ \text{TT.now()} \]

earliest \quad \text{latest}

\[ 2\varepsilon \]
Spanner TrueTime API

**TT.now()** – TTInterval: [earliest, latest]
(Why is time an interval and not a specific instant?)
**TT.after(t)** – true if t has definitely passed
**TT.before(t)** – true if t has definitely not arrived

**TT.after(t) = true**  **TT.now()**  **TT.before(t) = true**

“Global wall-clock time” with bounded uncertainty
Timestamps and TrueTime

Acquired locks

Pick \( s = \text{TT.now().latest} \)

Release locks

Wait until \( \text{TT.now().earliest} > s \)

Commit wait

average \( \varepsilon \)  

OSDI 2012
Spanner Commit Wait

Why do you need to wait for TT.now().earliest > s before releasing locks?
Spanner External Consistency

• If a transaction $T_1$ commits before another transaction $T_2$ starts, then $T_1$'s commit timestamp is smaller than $T_2$

• Similar to how we reason with wall-clock time

\[ s_1 < s_2 \]

\[ s_1 < s_2 \]
Summary

- **GFS / HDFS**
  - Data-center customized API, optimizations
  - Append focused DFS
  - Separate control (filesystem) and data (chunks)
  - Replication and locality
  - Rough consistency → apps handle rest

- **Spanner**
  - Globally consistent replicated database system
  - Implements distributed transactions
  - Lock-free reads and Paxos based writes
  - Implements external consistency using TrueTime API
  - Able to survive data center wipeouts
Overview

- Spanner
- DNS
- DNS today
- Naming with hashes
Names

- Names are associated with objects
  - Enables passing of references to objects
  - Indirection
  - Deferring decision on meaning/binding

- Examples
  - Registers → R5
  - Memory → 0xdeadbeef
  - Host names → srini.com
  - User names → sseshan
  - Email → srini@cmu.edu
  - File name → /usr/srini/foo.txt
  - URLs → http://www.srini.com/index.html
  - Ethernet → f8:e4:fb:bf:3d:a6
Name Discovery

- Well-known name
  - www.google.com, port 80…
- Broadcast
  - Advertise name \(\rightarrow\) e.g. 802.11 Beacons
- Query
  - Use google
- Broadcast query
  - Ethernet ARP
- Use another naming system
  - DNS returns IP addresses
- Introductions
  - Web page hyperlinks
- Physical rendezvous
  - Exchange info in the real world

OTHER KEY DIFFERENCES AS WELL….
Overview

- Spanner
- DNS
- DNS today
- Naming with hashes
DNS Records

RR format: \((\text{class}, \text{name}, \text{value}, \text{type}, \text{ttl})\)

- DB contains tuples called resource records (RRs)
  - Classes = Internet (IN), Chaosnet (CH), etc.
  - Each class defines value associated with type

### FOR IN class:

- **Type=A**
  - **name** is hostname
  - **value** is IP address
- **Type=NS**
  - **name** is domain (e.g. foo.com)
  - **value** is name of authoritative name server for this domain
- **Type=CNAME**
  - **name** is an alias name for some “canonical” (the real) name
  - **value** is canonical name
- **Type=MX**
  - **value** is hostname of mailserver associated with **name**
DNS Design: Zone Definitions

- Zone = contiguous section of name space
  - E.g., Complete tree, single node or subtree
- A zone has an associated set of name servers
  - Must store list of names and tree links
Typical Resolution

Client

www.cs.cmu.edu

Local DNS server

www.cs.cmu.edu
NS ns1.cmu.edu
NS ns1.cs.cmu.edu
A www=IPaddr

root & edu DNS server

ns1.cs.cmu.edu DNS server

ns1.cmu.edu DNS server
Typical Resolution

• Steps for resolving www.cmu.edu
  • Application calls gethostbyname() (RESOLVER)
  • Resolver contacts local name server (S₁)
  • S₁ queries root server (S₂) for (www.cmu.edu)
  • S₂ returns NS record for cmu.edu (S₃)
  • What about A record for S₃?
    • This is what the additional information section is for (PREFETCHING)
  • S₁ queries S₃ for www.cmu.edu
  • S₃ returns A record for www.cmu.edu
Lookup Methods

Recursive query:
• Server goes out and searches for more info (recursive)
• Only returns final answer or “not found”

Iterative query:
• Server responds with as much as it knows (iterative)
• “I don’t know this name, but ask this server”

Workload impact on choice?
• Local server typically does recursive
• Root/distant server does iterative
Workload and Caching

- Are all servers/names likely to be equally popular?
  - Why might this be a problem? How can we solve this problem?

- DNS responses are cached
  - Quick response for repeated translations
  - Other queries may reuse some parts of lookup
    - NS records for domains

- DNS negative queries are cached
  - Don’t have to repeat past mistakes
  - E.g. misspellings, search strings in resolv.conf

- Cached data periodically times out
  - Lifetime (TTL) of data controlled by owner of data
  - TTL passed with every record
Typical Resolution

Client → Local DNS server

- www.cs.cmu.edu
- ns1.cs.cmu.edu
- ns1.cmu.edu

- root & edu DNS server
- www.cs.cmu.edu
- NS ns1.cmu.edu
- NS ns1.cs.cmu.edu
- A www=IPaddr

www.cs.cmu.edu
Subsequent Lookup Example

Client \rightarrow Local DNS server

ftp.cs.cmu.edu

root & edu DNS server

cmu.edu DNS server

ftp=IPaddr

cs.cmu.edu DNS server
Tracing Hierarchy (1)

- Dig Program
  - Use flags to find name server (NS)
  - Disable recursion so that operates one step at a time

```
unix> dig +norecurse @a.root-servers.net NS four.cmcl.cs.cmu.edu

;; ADDITIONAL SECTION:
a.edu-servers.net  172800 IN  A   192.5.6.30
b.edu-servers.net. 172800 IN  A   192.26.92.30
c.edu-servers.net. 172800 IN  A   192.31.80.30
d.edu-servers.net. 172800 IN  A   192.35.51.30
e.edu-servers.net. 172800 IN  A   192.42.93.30
f.edu-servers.net. 172800 IN  AAAA 2001:503:cc2c::2:36
g.edu-servers.net. 172800 IN  A   192.41.162.30
l.edu-servers.net. 172800 IN  A   192.41.162.30
```

- All .edu names handled by set of servers
Prefetching

- Name servers can add additional data to response
- Typically used for prefetching
  - CNAME/MX/NS typically point to another host name
  - Responses include address of host referred to in “additional section”
• 3 servers handle CMU names

```
unix> dig +norecurse @g.edu-servers.net NS four.cmcl.cs.cmu.edu

;; AUTHORITY SECTION:
cmu.edu.       172800 IN  NS  ny-server-03.net.cmu.edu.
cmu.edu.       172800 IN  NS  nsauth1.net.cmu.edu.
cmu.edu.       172800 IN  NS  nsauth2.net.cmu.edu.
```
Tracing Hierarchy (3 & 4)

• 3 servers handle CMU CS names

```bash
unix> dig +norecurse @nsauth1.net.cmu.edu NS four.cmcl.cs.cmu.edu

;; AUTHORITY SECTION:
cs.cmu.edu.     600     IN      NS      AC-DDNS-2.NET.cs.cmu.edu.
cs.cmu.edu.     600     IN      NS      AC-DDNS-1.NET.cs.cmu.edu.
cs.cmu.edu.     600     IN      NS      AC-DDNS-3.NET.cs.cmu.edu.
```

• Server within CS is “start of authority” (SOA) for this name

```bash
unix> dig +norecurse @AC-DDNS-2.NET.cs.cmu.edu NS four.cmcl.cs.cmu.edu

;; AUTHORITY SECTION:
cs.cmu.edu.     300     IN      SOA      PLANISPHERE.FAC.cs.cmu.edu.
```
Tracing Hierarchy (3 & 4)

- Any will return the A record

```
unix> dig +norecurse @AC-DDNS-2.NET.cs.cmu.edu
four.cmcl.cs.cmu.edu

;; ANSWER SECTION:FOUR.CMCL.CS.CMU.EDU. 3600 IN A
128.2.209.189
```
DNS Hack #1

- Can return multiple A records → what does this mean?

- Load Balance
  - Server sends out multiple A records
  - Order of these records changes per-client
Server Balancing Example

- DNS Tricks

unix1> dig www.yahoo.com

;; ANSWER SECTION:
atsv2-fp.wg1.b.yahoo.com. 49 IN A 98.139.180.149
atsv2-fp.wg1.b.yahoo.com. 49 IN A 98.138.252.30
atsv2-fp.wg1.b.yahoo.com. 49 IN A 98.139.183.24
atsv2-fp.wg1.b.yahoo.com. 49 IN A 98.138.253.109

unix2> dig www.yahoo.com

;; ANSWER SECTION:
atsv2-fp.wg1.b.yahoo.com. 30 IN A 98.139.180.149
atsv2-fp.wg1.b.yahoo.com. 30 IN A 98.138.253.109
atsv2-fp.wg1.b.yahoo.com. 30 IN A 98.138.252.30
atsv2-fp.wg1.b.yahoo.com. 30 IN A 98.139.183.24
Overview

- Spanner
- DNS
- DNS today
- Naming with hashes
Protecting the Root Nameservers

Attack On Internet Called Largest Ever
By David McGuire and Brian Krebs
washingtonpost.com Staff Writers
Tuesday, October 22, 2002; 5:40 PM

The heart of the Internet sustained its largest and most sophisticated attack ever, starting late Monday, according to officials at key online backbone organizations.

Around 5:00 p.m. EDT on Monday, a "distributed denial of service" (DDOS) attack struck the 13 "root servers" that provide the primary roadmap for almost all Internet communications. Despite the scale of the attack, which lasted about an hour, Internet users worldwide were largely unaffected, experts said.

Defense Mechanisms

• Redundancy: 13 root nameservers
• IP Anycast for root DNS servers {c,f,i,j,k}.root-servers.net
  • RFC 3258
  • Most physical nameservers lie outside of the US
## Defense: Replication and Caching

<table>
<thead>
<tr>
<th>Letter</th>
<th>Old name</th>
<th>Operator</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>ns.internic.net</td>
<td>VeriSign</td>
<td>Dulles, Virginia, USA</td>
</tr>
<tr>
<td>B</td>
<td>ns1.isi.edu</td>
<td>ISI</td>
<td>Marina Del Rey, California, USA</td>
</tr>
<tr>
<td>C</td>
<td>c.psi.net</td>
<td>Cogent Communications</td>
<td>distributed using anycast</td>
</tr>
<tr>
<td>D</td>
<td>terp.umd.edu</td>
<td>University of Maryland</td>
<td>College Park, Maryland, USA</td>
</tr>
<tr>
<td>E</td>
<td>ns.nasa.gov</td>
<td>NASA</td>
<td>Mountain View, California, USA</td>
</tr>
<tr>
<td>F</td>
<td>ns.isc.org</td>
<td>ISC</td>
<td>distributed using anycast</td>
</tr>
<tr>
<td>G</td>
<td>ns.nic.ddn.mil</td>
<td>U.S. DoD NIC</td>
<td>Columbus, Ohio, USA</td>
</tr>
<tr>
<td>H</td>
<td>aos.arl.army.mil</td>
<td>U.S. Army Research Lab</td>
<td>Aberdeen Proving Ground, Maryland, USA</td>
</tr>
<tr>
<td>I</td>
<td>nic.nordu.net</td>
<td>Autonomica</td>
<td>distributed using anycast</td>
</tr>
<tr>
<td>J</td>
<td></td>
<td>VeriSign</td>
<td>distributed using anycast</td>
</tr>
<tr>
<td>K</td>
<td></td>
<td>RIPE NCC</td>
<td>distributed using anycast</td>
</tr>
<tr>
<td>L</td>
<td></td>
<td>ICANN</td>
<td>Los Angeles, California, USA</td>
</tr>
<tr>
<td>M</td>
<td></td>
<td>WIDE Project</td>
<td>distributed using anycast</td>
</tr>
</tbody>
</table>

source: wikipedia
What Happened on Oct 21st 2016?

• DDoS attack on Dyn
• Dyn provides core Internet services for Twitter, SoundCloud, Spotify, Reddit and a host of other sites
• Why didn’t DNS defense mechanisms work in this case?
• Let’s take a look at the DNS records…
What was the source of attack?

- Mirai botnet
  - Used in 620Gbps attack last month

- Source: bad IoT devices, e.g.,
  - White-labeled DVR and IP camera electronics
  - username: root and password: xc3511
  - password is hardcoded into the device firmware
Attack Waves

- DNS lookups are routed to the nearest data center
- First wave
  - On three Dyn data centers – Chicago, Washington, D.C., and New York
- Second wave,
  - Hit 20 Dyn data centers around the world.
  - Required extensive planning.
  - Since DNS request go to the closest DNS server, the attacker had to plan a successful attack for each of the 20 data centers with enough bots in each region to be able to take down the local Dyn services
Solutions?

- Dyn customers
  - Going to backup DNS providers
  - Signing up with an alternative today after the attacks, as PayPal did

- Lowering their time-to-life settings on their DNS servers
  - Redirect traffic faster to another DNS service that is still available
Root Zone

- Generic Top Level Domains (gTLD) = .com, .net, .org, etc...
- Country Code Top Level Domain (ccTLD) = .us, .ca, .fi, .uk, etc...
- Root server ({a-m}.root-servers.net) also used to cover gTLD domains
  - Load on root servers was growing quickly!
  - Moving .com, .net, .org off root servers was clearly necessary to reduce load → done Aug 2000
gTLDs

- **Unsponsored**
  - .com, .edu, .gov, .mil, .net, .org
  - .biz → businesses
  - .info → general info
  - .name → individuals

- **Sponsored (controlled by a particular association)**
  - .aero → air-transport industry
  - .cat → catalan related
  - .coop → business cooperatives
  - .jobs → job announcements
  - .museum → museums
  - .pro → accountants, lawyers, and physicians
  - .travel → travel industry

- **Starting up**
  - .mobi → mobile phone targeted domains
  - .post → postal
  - .tel → telephone related

- **Proposed**
  - .asia, .cym, .geo, .kid, .mail, .sco, .web, .xxx
New Registrars

• Network Solutions (NSI) used to handle all registrations, root servers, etc…
  • Clearly not the democratic (Internet) way
  • Large number of registrars that can create new domains → However NSI still handles A root server
Do you trust the TLD operators?

- Wildcard DNS record for all .com and .net domain names not yet registered by others
  - September 15 – October 4, 2003
  - February 2004: Verisign sues ICANN
- Redirection for these domain names to Verisign web portal (SiteFinder)
- What services might this break?
DNS (Summary)

- Motivations → large distributed database
  - Scalability
  - Independent update
  - Robustness
- Hierarchical database structure
  - Zones
  - How is a lookup done
- Caching/prefetching and TTLs
- Reverse name lookup
- What are the steps to creating your own domain?
DNS Dist Sys Lessons

- Availability and reliability
- Caching and consistency
- Federated design
- Security
- Scalability
- Coordination/standardization
Overview

- Spanner
- DNS
- DNS today
- Naming with hashes
Two uses of hashing that are becoming wildly popular in distributed systems:

- Content-based naming
- Consistent Hashing of various forms
Example systems that use them

- BitTorrent & many other modern p2p systems use content-based naming
- Content distribution networks such as Akamai use consistent hashing to place content on servers
- Amazon, Linkedin, etc., all have built very large-scale key-value storage systems (databases--) using consistent hashing
Problem: Dividing items onto storage servers

• Option 1: Static partition (items a-c go there, d-f go there, ...)
  • If you used the server name, what if “cowpatties.com” had 1000000 pages, but “zebras.com” had only 10? → Load imbalance
  • Could fill up the bins as they arrive → Requires tracking the location of every object at the front-end.
Hashing

- Let nodes be numbered 1..m
- Client uses a **good** hash function to map a URL to 1..m
- Say hash (url) = x, so, client fetches content from node x
- No duplication – not being fault tolerant.
- Any other problems?
  - What happens if a node goes down?
  - What happens if a node comes back up?
  - What if different nodes have different views?
Option 2: Conventional Hashing

- bucket = hash(item) / num_buckets
- Sweet! Now the server we use is a deterministic function of the item, e.g., sha1(URL) → 160 bit ID / 20 → a server ID
- But what happens if we want to add or remove a server?
Option 2: Conventional Hashing

- 90 documents, node 1..9, node 10 which was dead is alive again
- Simple assignment:
  - ID/9 (for 9 servers) vs. ID/10 (for 10 servers)
- % of documents in the wrong node?
  - 10, 19-20, 28-30, 37-40, 46-50, 55-60, 64-70, 73-80, 82-90
  - *Disruption coefficient* = \( \frac{1}{2} \) 😞
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 much 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
Hashing 2: For naming

- Many file systems split files into blocks and store each block on a disk.
- Several levels of naming:
  - Pathname to list of blocks
  - Block #s are addresses where you can find the data stored therein. (But in practice, they’re logical block #s – the disk can change the location at which it stores a particular block... so they’re actually more like names and need a lookup to location :)
Another problem to solve...

- Imagine you’re creating a backup server
- It stores the full data for 1000 CMU users’ laptops
- Each user has a 100GB disk.
- That’s 100TB and lots of $$$
- How can we reduce the storage requirements?
“Deduplication”

• A common goal in big archival storage systems. Those 1000 users probably have a lot of data in common -- the OS, copies of binaries, maybe even the same music or movies.

• How can we detect those duplicates and coalesce them?

• One way: Content-based naming, also called content-addressable foo (storage, memory, networks, etc.)

• A fancy name for...
Name items by their hash

- Imagine that your filesystem had a layer of indirection:
  - pathname → hash(data)
  - hash(data) → list of blocks

- For example:
  - `/src/foo.c` -> 0xfff32f2fa11d00f0
  - 0xfff32f2fa11d00f0 -> [5623, 5624, 5625, 8993]

- If there were two identical copies of `foo.c` on disk ... We’d only have to store it once!
  - Name of second copy can be different
A second example

- Several p2p systems operate something like:
  - Search for “national anthem”, find a particular file name (starspangled.mp3).
  - Identify the files by the hash of their content (0x2fab4f001...)
  - Request to download a file whose hash matches the one you want
  - Advantage? You can verify what you got, even if you got it from an untrusted source (like some dude on a p2p network)
P2P-enabled Applications: Self-Certifying Names

- Name = Hash(pubkey, salt)

- Value = <pubkey, salt, data, signature>
  - can verify name related to pubkey and pubkey signed data

- Can receive data from caches, peers or other 3rd parties without worry
Desirable Properties of Hashes

- **Compression**: Maps a variable-length input to a fixed-length output.
- **Ease of computation**: A relative metric...
- **Pre-image resistance**: For all outputs, computationally infeasible to find input that produces output.
- **2nd pre-image resistance**: For all inputs, computationally infeasible to find second input that produces same output as a given input.
- **Collision resistance**: For all outputs, computationally infeasible to find two inputs that produce the same output.
Hash functions

• Given a universe of possible objects $U$, map $N$ objects from $U$ to an $M$-bit hash.

• Typically, $|U| \gg \gg 2^M$.
  • This means that there can be collisions: Multiple objects map to the same $M$-bit representation.

• Likelihood of collision depends on hash function, $M$, and $N$.
  • Birthday paradox $\rightarrow$ roughly 50% collision with $2^{M/2}$ objects for a well designed hash function
Longevity

• “Computationally infeasible” means different things in 1970 and 2012.
  • Moore’s law
  • Some day, maybe, perhaps, sorta, kinda: Quantum computing.

• Hash functions are not an exact science yet.
  • They get broken by advances in crypto.
Real hash functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Introduced</th>
<th>Weakened</th>
<th>Broken</th>
<th>Lifetime</th>
<th>Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD4</td>
<td>1990</td>
<td>1991</td>
<td>1995</td>
<td>1-5y</td>
<td>MD5</td>
</tr>
<tr>
<td>MD5</td>
<td>1992</td>
<td>1994</td>
<td>2004</td>
<td>8-10y</td>
<td>SHA-1</td>
</tr>
<tr>
<td>MD2</td>
<td>1992</td>
<td>1995</td>
<td>abandoned</td>
<td>3y</td>
<td>SHA-1</td>
</tr>
<tr>
<td>RIPEMD</td>
<td>1992</td>
<td>1997</td>
<td>2004</td>
<td>5-12y</td>
<td>RIPEMD-160</td>
</tr>
<tr>
<td>HAVAL-128</td>
<td>1992</td>
<td>-</td>
<td>2004</td>
<td>12y</td>
<td>SHA-1</td>
</tr>
<tr>
<td>SHA-0</td>
<td>1993</td>
<td>1998</td>
<td>2004</td>
<td>5-11y</td>
<td>SHA-1</td>
</tr>
<tr>
<td>SHA-1</td>
<td>1995</td>
<td>2004</td>
<td>not quite yet</td>
<td>9+</td>
<td>SHA-2 &amp; 3</td>
</tr>
<tr>
<td>SHA-2 (256, 384, 512)</td>
<td>2001</td>
<td>still good</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHA-3</td>
<td>2012</td>
<td>brand new</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Using them

• How long does the hash need to have the desired properties (preimage resistance, etc)?
  • rsync: For the duration of the sync;
  • dedup: Until a (probably major) software update;
  • store-by-hash: Until you replace the storage system

• What is the adversarial model?
  • Protecting against bit flips vs. an adversary who can try 1B hashes/second?
Hashing forms the basis for MACs - message authentication codes

- Basically, a hash function with a secret key.
- \( H(\text{key}, \text{data}) \) - can only create or verify the hash given the key.
- Very, very useful building block
Final pointer 2: Rabin Fingerprinting

File Data

Hash 1

Hash 2

Rabin Fingerprints

4 7 8

Natural Boundary

2 8

Given Value -8
Summary

- Hashes used for:
  - Splitting up work/storage in a distributed fashion
  - Naming objects with self-certifying properties

- Key applications
  - Key-value storage
  - P2P content lookup
  - Deduplication
  - MAC

- Many types of naming
  - DNS names, IP addresses, Ethernet addresses, content-based addresses
  - Make sure you understand differences
The way we outlined it results in moderate load imbalance between buckets (remember balls and bins analysis of hashing?)

To reduce imbalance, systems often represent each physical node as \( k \) different buckets, sometimes called “virtual nodes” (but really, it’s just multiple buckets).

\( \log n \) buckets gets you a very pleasing load balance - \( O(\#\text{items}/n) \) with high probability, if \( \#\text{items} \) large and uniformly distributed
Use of consistent hashing

• Consistent hashing was first widely used by Akamai CDN
• Also used in systems like Chord DHT
  • Provided key-value storage, designed to scale to millions or billions of nodes
  • Had a p2p lookup algorithm, completely decentralized, etc. Fun to read about; very influential, but not widely used outside of p2p systems.
• In practice, many more systems use consistent hashing where the people doing the lookups know the list of all storage nodes (tens to tens of thousands; not too bad) and directly determine who to contact
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
Self-certifying Names

• Use a name that helps validate the data associated with the name
  • Seems like a circular argument but...
  • Traditional name → Declaration of Independence
  • Self-certifying name → SHA1(Declaration of Independence contents)
    • SHA1 → cryptographic hash
Self-Certifying Names

- Can also create names using public key crypto
  - Name = Hash(pubkey, salt)
  - Value = <pubkey, salt, data, signature>
  - Can verify name related to pubkey and pubkey signed data

- Benefits
  - Can verify contents after receiving file
  - Can fetch file from untrustworthy sources

- Weaknesses
  - No longer human readable
Overview

• BigTable

• Spanner

• Naming Overview

• Hashing Tricks
Names

- Names are associated with objects
  - Enables passing of references to objects
  - Indirection
  - Deferring decision on meaning/binding

- Examples
  - Registers → R5
  - Memory → 0xdeadbeef
  - Host names → srini.com
  - User names → sseshan
  - Email → srini@cmu.edu
  - File name → /usr/srini/foo.txt
  - URLs → http://www.srini.com/index.html
Naming Model

• 3 key elements

1. Name space
   • Alphabet of symbols + syntax that specify names

2. Name-mapping
   • Associates each name to some value in…

3. Universe of values
   • Typically an object or another name from original name space (or another name space)

• Name-to-value mapping is called a “binding” i.e. name is bound to value
Names

• Uniqueness
  • One-to-one mapping
  • One-to-many or many-to-one (name-to-value) mappings
  • Context sensitive resolution

• Stable binding
  • Names that are never reused
  • Values that can only have one name
  • E.g. using MD5 of file contents, bank account numbers

• Reverse lookup support
Name Mapping

- Names are mapped to values within some context
  - E.g., different lookup tables for names in different settings

- Two sources for context
  - Resolver can supply default context
  - Name can specify an explicit context to use → qualified name
    - "cd /users/srini/440/midterm" vs "cd 440/midterm"
• Common problem → what context to use for names without context
• Consider email from CMU
  • To: srini, yuvraj@gmail.com
  • What happens when yuvraj replies to all?
    • What context will he email srini
• Solutions:
  • Sendmail converts all address to qualified names
    • Not in body of message
  • Provide context information in email header
    • E.g. like base element in HTML
Name Lookup Styles

- **Table lookup**
  - Simple, table per context

- **Recursive**
  - Names consist of context + name
  - E.g. path + filename, hostname + domain name
  - Context name must also be resolved
    - Need special context such as “root” built into resolver

- **Multiple lookup**
  - Try multiple contexts to resolve name ➔ search paths
Recursive Name Spaces

- A general naming graph with a single root node.
Name Discovery

- Well-known name
  - www.google.com, port 80…
- Broadcast
  - Advertise name → e.g. 802.11 Beacons
- Query
  - Use google
- Broadcast query
  - 802.11 probes
- Use another naming system
  - DNS returns IP addresses
- Introductions
  - Web page hyperlinks
- Physical rendezvous
  - Exchange info in the real world
Feedback

Issues
- Due dates
- Slide details
- Writeup clarity
- Record lecture
- More OH
- Readings

Positives
- Go/proj
- Lectures
- TAs
- Piazza (more + than -)
- Midterm review