NAMING, DNS, AND CHORD

George Porter
Feb 4, 6, and 8, 2019

ATTRIBUTION

• These slides are released under an Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0) Creative Commons license
• These slides incorporate material from:
  • Michael Freedman and Kyle Jamieson, Princeton University (also under a CC BY-NC-SA 3.0 Creative Commons license)
  • Andrew Moore, Univ. of Cambridge
  • The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, 2nd ed., by Barroso, Clidaras, and Hölzle
ANNOUNCEMENTS
Project 1 due today

Gradescope invitation code: 97EGV3

Optional reading: “The datacenter as a computer” linked off the course page

Outline
• Terminology: Parallelism vs Concurrency
• Processes, threads, and OS-level mechanisms
• Datacenters
DATACENTERS ARE NOT EXACTLY NEW...

EDSAC, 1949

“ROWS” OF SERVERS IN A DATACENTER
“RACKS” MAKING UP ONE ROW

- 20-40 “pizza box” servers per rack
- Each rack has a “top of rack” network switch that connects it to the rest of the datacenter network

A SINGLE RACK
CONNECTING RACKS TOGETHER

• “Aggregation” and “Core” network switches provide connectivity between racks

BROCADE REFERENCE DESIGN
CISCO REFERENCE DESIGN

DATACENTER PERFORMANCE

- Ideal: Homogeneous performance
  - Uniform bandwidth/latency between all servers
- Reality (typical): Heterogeneous performance
  - Two servers in the same rack
    - Very high bandwidth/very low latency
  - Two servers in same row (not same rack)
    - Medium bandwidth / medium latency
  - Two servers in different rows
    - Low bandwidth / high latency
EXTREME MODULARITY

- Containers filled with a 2 or 4 rows of servers
- Many containers

EFFECT OF THE NETWORK ON PERFORMANCE

One Server
- DRAM: 16 GB, 100 ns, 20 GB/s
- Disk: 2 TB, 10 ms, 200 MB/s
- Flash: 128 GB, 100 ns, 1 GB/s

Local Rack (80 servers)
- DRAM: 1 TB, 300 ns, 100 MB/s
- Disk: 160 TB, 11 ms, 100 MB/s
- Flash: 20 TB, 400 ns, 100 MB/s

Cluster (30 racks)
- DRAM: 30 TB, 500 ns, 10 MB/s
- Disk: 4.80 PB, 12 ms, 10 MB/s
- Flash: 600 TB, 600 ns, 10 MB/s
Outline

- Naming overview
- Flat naming
- Routing over flat names via Chord
- Hierarchical naming and DNS
NAMING OVERVIEW

• Access point: way of contacting resource in a networked system
• Address: name of an access point
  • Can change over time
• Names
  • Flat
  • Structured

• Flat:
  • “Opaque” identifier, no indication as to location
  • Examples: phone numbers (858-8325 vs 858-1220)
  • Ethernet: 03:1a:0f:f1:de:91

• Structured:
  • Location encoded in the address
  • 9500 Gilman Drive, La Jolla, CA
  • 206.109.3.12

NAMING COMPONENTS

• www.cs.ucsd.edu
• /home/aturing/docs/paper.pdf
• 03:A8:BF:01:00:2C
• 206.109.21.7
• aturing@cs.ucsd.edu
• aturing
NAMING HIERARCHY FOR SCALE

- Host name: www.ucsd.edu
  - Domain: registrar for each top-level domain (e.g., .edu)
  - Host name: local administrator assigns to each host

- IP addresses: 128.54.70.238
  - Prefixes: ICANN, regional Internet registries, and ISPs
  - Hosts: static configuration, or dynamic using DHCP

- MAC addresses: 58:B0:35:F2:3C:D9
  - OIDs: assigned to vendors by the IEEE
  - Adapters: assigned by the vendor from its block

MAPPING BETWEEN IDENTIFIERS

- Domain Name System (DNS)
  - Given a host name, provide the IP address
  - Given an IP address, provide the host name

- Address Resolution Protocol (ARP)
  - Given an IP address, provide the MAC address
  - To enable communication within the Local Area Network

- Dynamic Host Configuration Protocol (DHCP)
  - Automates host boot-up process
  - Given a MAC address, assign a unique IP address
  - ... and tell host other stuff about the Local Area Network
Outline

• Naming overview
• Flat naming
• Routing over flat names via Chord
• Hierarchical naming and DNS

DNS HOSTNAME VERSUS IP ADDRESS

• **DNS host name** (e.g. www.cs.ucsd.edu)
  • Mnemonic name appreciated by humans
  • Variable length, full alphabet of characters
  • Provides little (if any) information about location

• **IP address** (e.g. 128.112.136.35)
  • Numerical address appreciated by routers
  • Fixed length, decimal number
  • Hierarchical address space, related to host location
MANY USES OF DNS

- Hostname to IP address translation
- IP address to hostname translation (reverse lookup)
- Host name aliasing: other DNS names for a host
  - Alias host names point to canonical hostname
- Email: Lookup domain’s mail server by domain name

ORIGINAL DESIGN OF DNS

- Per-host file named /etc/hosts (1982)
  - Flat namespace: each line = IP address & DNS name
  - SRI (Menlo Park, California) kept the master copy
  - Everyone else downloads regularly
- But, a single server doesn’t scale
  - Traffic implosion (lookups and updates)
  - Single point of failure
- Need a distributed, hierarchical collection of servers
DNS: GOALS AND NON-GOALS

- A wide-area **distributed database**
- Goals:
  - **Scalability**; decentralized maintenance
  - **Robustness**
  - Global scope
    - Names mean the same thing everywhere
  - Distributed updates/queries
  - Good **performance**
- But don’t need strong consistency properties

DOMAIN NAME SYSTEM (DNS)

- Hierarchical name space divided into contiguous sections called **zones**
  - Zones are distributed over a collection of DNS servers
- Hierarchy of DNS servers:
  - **Root** servers (identity hardwired into other servers)
  - **Top-level domain (TLD)** servers
  - **Authoritative** DNS servers
- Performing the translations:
  - **Local DNS servers** located near clients
  - **Resolver** software running on clients
DNS IS HIERARCHICAL

- Hierarchy of namespace matches hierarchy of servers
- Set of nameservers answers queries for names within zone
- Nameservers store names and links to other servers in tree

TLDs: com. gov. edu.

Root


cs.ucsd.edu.

ANNOUNCEMENTS

2\textsuperscript{nd} chance to submit Project 1 w/o penalty $\rightarrow$ Thu @ 5pm

No submissions at all after that time

Project 2 goes out Thursday at 5

Gradescope invitation code: \texttt{97EGV3}
DNS ROOT NAMESERVERS

• 13 root servers. *Does this scale?*

• Each server is really a *cluster of servers* (some geographically distributed), replicated via *IP anycast*
TLD AND AUTHORITATIVE SERVERS

- ftp://ftp.internic.net/domain/named.root
- Top-level domain (TLD) servers
  - Responsible for com, org, net, edu, etc, and all top-level country domains: uk, fr, ca, jp
  - Network Solutions maintains servers for com TLD
  - Educause non-profit for edu TLD
- Authoritative DNS servers
  - An organization’s DNS servers, providing authoritative information for that organization
  - May be maintained by organization itself, or ISP

LOCAL NAME SERVERS

- Do not strictly belong to hierarchy
- Each ISP (or company, or university) has one
  - Also called default or caching name server
- When host makes DNS query, query is sent to its local DNS server
  - Acts as proxy, forwards query into hierarchy
  - Does work for the client
DNS RESOURCE RECORDS

- DNS is a distributed database storing resource records
- Resource record includes: (name, type, value, time-to-live)

  **Type = A (address)**
  - name = hostname
  - value is IP address

  **Type = CNAME**
  - name = alias for some “canonical” (real) name
  - value is canonical name

  **Type = NS (name server)**
  - name = domain (e.g. princeton.edu)
  - value is hostname of authoritative name server for this domain

  **Type = MX (mail exchange)**
  - name = domain
  - value is name of mail server for that domain

DNS IN OPERATION

- Most queries and responses are UDP datagrams
- Two types of queries:

  **Recursive**: Nameserver responds with answer or error

  ![Recursive query diagram]

  Client: `www.ucsd.edu?`

  Answer: `www.ucsd.edu A 132.239.180.101`

  **Iterative**: Nameserver may respond with a referral

  ![Iterative query diagram]

  Client: `www.ucsd.edu?`

  Referral: `.edu NS a.edu-servers.net.`
ITERATIVE DNS LOOKUP

1. [nl,vu,cs,ftp]
2. #[nl], [vu,cs,ftp]
3. [vu,cs,ftp]
4. #[vu], [cs,ftp]
5. [cs,ftp]
6. #[cs], [ftp]
7. [ftp]
8. #[ftp]

Recursive DNS Lookup

1. [nl,vu,cs,ftp]
8. #[nl,vu,cs,ftp]
7. #[vu,cs,ftp]
6. #[cs,ftp]
5. #[ftp]

Nodes are managed by the same server.
**RECURSIVE LOOKUP STATE**

**Recursive name resolution of \([nl, vu, cs, ftp]\)**

<table>
<thead>
<tr>
<th>Server for node</th>
<th>Should resolve</th>
<th>Looks up</th>
<th>Passes to child</th>
<th>Receives and caches</th>
<th>Returns to requester</th>
</tr>
</thead>
<tbody>
<tr>
<td>cs</td>
<td>([ftp])</td>
<td>([ftp])</td>
<td>—</td>
<td>—</td>
<td>([ftp])</td>
</tr>
<tr>
<td>vu</td>
<td>([cs, ftp])</td>
<td>([cs])</td>
<td>([ftp])</td>
<td>([ftp])</td>
<td>([cs])</td>
</tr>
<tr>
<td>(nl)</td>
<td>([vu, cs, ftp])</td>
<td>([vu])</td>
<td>([cs, ftp])</td>
<td>([cs])</td>
<td>([vu, cs])</td>
</tr>
<tr>
<td>root</td>
<td>([nl, vu, cs, ftp])</td>
<td>([nl])</td>
<td>([vu, cs, ftp])</td>
<td>([vu])</td>
<td>([nl])</td>
</tr>
</tbody>
</table>

**DNS CACHING**

- Performing all these queries takes time
  - And all this before actual communication takes place
- Caching can greatly reduce overhead
  - The top-level servers very rarely change
    - Popular sites visited often
  - Local DNS server often has the information cached
- How DNS caching works
  - All DNS servers **cache responses to queries**
  - Responses include a time-to-live (TTL) field
    - Server deletes cached entry after TTL expires
JULIA EVAN’S GUIDE TO DIG

dig makes DNS queries!
$ dig google.com
google.com 208 IN A
IP address is 172.217.13.111

dig TYPE domain.com
This lets you choose which DNS record to query for:
types to try: SERV, default
MX, TXT, AAAA, A

dig @ 8.8.8.8 domain
Google DNS server
dig @ server lets you pick which DNS server to query! Useful to check if your system DNS is misbehaving.

dig +trace domain
traces how your domain gets resolved, starting at the root nameservers

dig -x 172.217.13.111
makes a reverse DNS query - find which domain resolves to an IP!

dig +short domain
Usually dig prints lots of output! With +short it just prints the IP address value of the DNS record

Outline

- Naming overview
- Flat naming
- Routing over flat names via Chord
BROADCAST: ADDRESS RESOLUTION PROTOCOL (ARP)

• IP forwarding tables: one entry per network not host
• Thus, routes designed to get packets to proper network
• Network needs to take over from there to get to proper host
• Address resolution protocol (ARP) translates IP addresses to link-level addresses (e.g., 48-bit Ethernet addr)
• Broadcast request over network for IP→link-level mapping
• Maintain local cache (with timeout)

ARP OVERVIEW

Broadcast: Anyone know the Ethernet address for 152.3.145.240?

Reply: Yes, I’m at 08-00-2b-18-bc-65
BROADCAST TO HANDLE AUTOCONFIGURATION

- Host doesn’t have an IP address yet
  - So, host doesn’t know what source address to use

- Host doesn’t know who to ask for an IP address
  - So, host doesn’t know what destination address to use

- Solution: shout to discover a server who can help
  - Install a special server on the LAN to answer distress calls

DHCP

- Broadcast-based LAN protocol algorithm
  - Host broadcasts “DHCP discover” on LAN (e.g. Ethernet broadcast)
  - DHCP server responds with “DHCP offer” message
  - Host requests IP address: “DHCP request” message
  - DHCP server sends address: “DHCP ack” message w/IP address

- Easy to have fewer addresses than hosts (e.g. UCSD wireless) and to renumber network (use new addresses)

- What if host goes away (how to get address back?)
  - Address is a “lease” not a “grant”, has a timeout
  - Host may have different IP addresses at different times?
FORWARDING POINTERS

• Simple idea: leave behind pointer

---

HOME/FOREIGN AGENT FORWARDING

Figure 1: Operation of Mobile IPv4
Generalizing forwarding pointers?

- Can we automate the following of forwarding pointers?
  - Scalable
    - What do we mean by scalable?
  - Fault-tolerant
  - Performant?

Outline

- Naming overview
- Flat naming
- Routing over flat names via Chord
- Hierarchical naming and DNS
FLAT NAMING AND PEER-TO-PEER (P2P) NETWORKS

- A distributed system architecture:
  - No centralized control
  - Nodes are roughly symmetric in function
  - Large number of unreliable nodes

P2P ADOPTION

- Successful adoption in some niche areas –

1. Client-to-client (legal, illegal) file sharing
   - Popular data but owning organization has no money

2. Digital currency: no natural single owner (Bitcoin)

3. Voice/video telephony: user to user anyway
   - Issues: Privacy and control
**EXAMPLE: CLASSIC BITTORRENT**

1. User clicks on download link
   - Gets *torrent* file with content hash, IP addr of *tracker*
2. User’s BitTorrent (BT) client talks to tracker
   - Tracker tells it *list of peers* who have file
3. User’s BT client downloads file from one or more peers
4. User’s BT client tells tracker it has a copy now, too
5. User’s BT client serves the file to others for a while

*Provides huge download bandwidth, without expensive server or network links*

**FLAT NAME LOOKUP PROBLEM**

Publisher (N₄)

Client

Internet

N₁ → N₂ → N₃ → N₅ → N₆

put(“Star Wars.mov”, [content])

get(“Star Wars.mov”)
CENTRALIZED LOOKUP (NAPSTER)

SetLoc("Star Wars.mov", IP address of N_4)  

Publisher (N_4)  
key="Star Wars.mov", value=[content]  

Simple, but $O(N)$ state and a single point of failure

FLOODED QUERIES (ORIGINAL GNUTELLA)

Lookup("Star Wars.mov")

Publishers (N_4)  
key="Star Wars.mov", value=[content]  

Robust, but $O(N = \text{number of peers})$ messages per lookup
Can we make it robust, reasonable state, reasonable number of hops?

SYSTEMATIC FLAT NAME LOOKUPS VIA DHTS

- **Local hash table:**
  
  \[
  \text{key} = \text{Hash(name)}
  \]
  
  \[
  \text{put(key, value)}
  \]
  
  \[
  \text{get(key) } \rightarrow \text{ value}
  \]

- **Service:** Constant-time insertion and lookup

  *How can I do (roughly) this across millions of hosts on the Internet?*

  *Distributed Hash Table (DHT)*
WHAT IS A DHT (AND WHY)?

• Distributed Hash Table:
  key = hash(data)
  lookup(key) → IP addr (Chord lookup service)
  send-RPC(IP address, put, key, data)
  send-RPC(IP address, get, key) → data

• Partitioning data in truly large-scale distributed systems
  • Tuples in a global database engine
  • Data blocks in a global file system
  • Files in a P2P file-sharing system

WHY MIGHT DHT DESIGN BE HARD?

• Decentralized: no central authority

• Scalable: low network traffic overhead

• Efficient: find items quickly (latency)

• Dynamic: nodes fail, new nodes join
CHORD LOOKUP ALGORITHM PROPERTIES

- **Interface**: `lookup(key) → IP address`

- **Efficient**: $O(\log N)$ messages per lookup
  - $N$ is the total number of servers

- **Scalable**: $O(\log N)$ state per node

- **Robust**: survives massive failures

CHORD IDENTIFIERS

- **Key identifier** = `SHA-1(key)`

- **Node identifier** = `SHA-1(IP address)`

- SHA-1 distributes both uniformly

- **How does Chord partition data?**
  - *i.e.*, map key IDs to node IDs
CONSISTENT HASHING [KARGER ‘97]

Key is stored at its successor: node with next-higher ID

CHORD: SUCCESSOR POINTERS
BASIC LOOKUP

SIMPLE LOOKUP ALGORITHM

**Lookup**(key-id)

succ $\leftarrow$ my successor

if my-id < succ < key-id // next hop
    call Lookup(key-id) on succ

else // done
    return succ

- **Correctness** depends only on **successors**
IMPROVING PERFORMANCE

• **Problem:** Forwarding through successor is slow

• Data structure is a linked list: $O(n)$

• **Idea:** Can we make it more like a binary search?
  • Need to be able to halve distance at each step

CHORD INTUITION

• Skip Lists (Pugh, 1989)

• Consider a linked list:

```
head 1 2 3 4 5 6 7 8 9 10 NIL
```

• Lookup time: $O(n)$
CHORD INTUITION

• Skip Lists (Pugh, 1989)
• Consider a linked list:

Add 2nd row of pointers spaced further apart
• Still O(n), but more efficient
• Use 2nd row to get as close as possible without going over
• Then last row to get to the desired element

CHORD INTUITION

• Skip Lists (Pugh, 1989)
• Consider a linked list:

Add log(N) rows
• Get as close as possible on top row, then drop down a row, then drop down another row, until the bottom row
• O(log N) lookup time
AN ASIDE: IS LOG(N) FAST OR SLOW?

• For a million nodes, it’s 20 hops

• If each hop takes 50 milliseconds, lookups take a second

• If each hop has 10% chance of failure, it’s a couple of timeouts

• So in practice log(n) is better than O(n) but not great
JOINING: LINKED LIST INSERT

1. Lookup(36)

JOIN (2)

2. N36 sets its own successor pointer
JOIN (3)

3. Copy keys 26..36 from N40 to N36

N25

N36  K30

N40  K30  K38

NOTIFY MESSAGES MAINTAIN PREDECESSORS

notify N25

N25

N36

N40

notify N36
STABILIZE MESSAGE FIXES SUCCESSOR

JOINING: SUMMARY

- Predecessor pointer allows link to new node
- Update finger pointers in the background
- Correct successors produce correct lookups
WHAT DHTS GOT RIGHT

• Consistent hashing
  • Elegant way to divide a workload across machines
  • Very useful in clusters: actively used today in Amazon Dynamo and other systems

• Replication for high availability, efficient recovery after node failure

• Incremental scalability: “add nodes, capacity increases”

• Self-management: minimal configuration

• Unique trait: no single server to shut down/monitor