CSCI 339
Distributed Systems

Lecture 7
Naming, Directory, and Discovery Services
Feb 23, 2023
Administrative Details

• Web server should be submitted
  • Make sure you add/commit writeup and code

• Project 2 will be officially posted soon
  • Draft is available now
  • Will send Google form about partners soon
  • Easier than Project 1 wrt to code (it’s not C)
  • More emphasis on design and evaluation
Lessons Learned

• Messy code leads to annoying bugs
• Cleaning up code sometimes leads to more bugs
  • Try not to wait to clean code up until the end!
  • Do it incrementally
• Spend time planning overall design before coding
• Debugging distributed systems is hard
• For a server, strive for high availability…avoid crashing
Last Time

• Discussed challenges associated with Internet services
  • “Lessons from Giant-Scale Services”
• Learned about DNS
Today’s Outline

• Discuss challenges associated with:
  • Naming services - white pages
  • Directory services - yellow pages
  • Discovery services - “dynamic” yellow pages
• Briefly discuss DHTs (distributed hash tables)
Case Study: Domain Name System
Motivation

- 1982: A single hosts.txt file stored and distributed (via FTP) from a central site to all computers
  - Flat-naming scheme based on simple table lookup
  - Maintained by one man (Jon Postel)
- By 1988 there were ~1400 hosts in hosts.txt
Domain Name System

- Hierarchical namespace
- Control delegated in hierarchical fashion (i.e., a tree)
- Designed to be extensible, robust, scalable
- DNS database is distributed (through combination of replication and data partitioning) across logical network of servers
- Translate human understandable names to machine understandable names
- Caching along the way to improve performance
Hierarchical Design

 rooted domain

.com

.edu

.mil

.org

.ucsd

.amherst

.williams

.harvard

.mit

.ca

.uk

.com

.covering

.org

.mil

.edu

.ucsd

.amherst

.williams

.harvard

.mit

.ca

.uk

.com
Query Processing

- Clients (i.e., your computers) query local name server
  - Local name server can provide authoritative or cached answers
- Many name servers support both recursive and iterative queries
  - Recursive queries - servers recursively find answer and return it to resolver (host who originally submitted query)
  - Iterative queries - each server responds to resolver with information about next server to ask
  - (more on next slide)
- If response is not cached locally, locate server **lowest** in the hierarchy with entry in local DB
  - In the worst case, contact root (.)
  - Cache locally with TTL
  - Note that queries never move “up” the tree
Recursive DNS Lookup Example

(Note that the servers could also just route the request among themselves rather than going back to the local nameserver each time.)
Iterative DNS Lookup Example

Local nameserver only tells client what it already knows.
Domains and Zones

- **Domains**
  - Entire branches in tree

- **Zones**
  - Portions of a domain (anchored at domain nodes)
  - Any contiguous set of nodes in the tree
  - Provide local autonomy
  - Can be grown to arbitrary size
  - Each zone should provide redundant name servers
Zones and Domains

edu Zone

.org

.mil

.com

.uk

.ca

.ucsd

.amherst

.williams

.harvard

.mit

.edu

.edu Domain

.zone and Domain

.root

.www

.cs

.oit

.limia

.lohani

.sysnet
Caching

• Caching (and replication) are used extensively to improve performance

• Name servers cache lookups for some period of time
  • Time to keep data in cache determined by TTL value
  • (TTL = time to live)
  • Design decisions:
    • Low TTL -> more accurate results, more lookups
    • High TTL -> less accurate results (stale data), less lookups

• Caching can significantly reduce lookup time and traffic, but only if the right design choices are made

• Cache poisoning
1988 DNS Status

• ~20k hosts available through DNS
• 30 top level domain names
• 7 Root servers
  • Support 1 query per second, driven by tuning of parameters
• Query breakdown
  • All info (25-40%)
  • Hostname to address (30-40%)
  • Address to hostname (10-15%)
  • Mail MX record (<10%)
Discussion

- Where does security fit into DNS?
- What tradeoffs are associated with caching DNS lookups?
- Who controls DNS root servers?
- Who should control them?
- International domain names?
DNS Attacks

- https://defintel.com/blog/index.php/2017/05/these-6-dns-attacks-threaten-your-business.html
- Distributed DoS attack (DDos)
- Distributed Reflection DoS attack
- Cache poisoning
- TCP SYN floods
- DNS tunneling
- DNS hijacking/redirection
  - https://www.wired.com/story/what-is-dns-hijacking/
DNS Reflection/Amplification
DDOS Attack

https://www.youtube.com/watch?v=xTKjHWkDwP0

DNS reflection and amplification

DNS Reflection Flood

Attacker/ Bot

The Attacker spoofs the target host IP

Source IP: 1.2.3.4

Open DNS Resolvers

Target Host

Destination IP: 1.2.3.4

DNS TCP SYN flood DDOS Attack

https://defintel.com/blog/index.php/2017/05/these-6-dns-attacks-threaten-your-business.html
Other Naming Services

- Directory services
  - Name services allow users to find attributes for a given name
  - But how do you find names for a given attribute?
- “Yellow pages services”
- Example:
  - Find all computers running Mac OS X in this building
  - Find closest printer
- Popular directory services
  - Microsoft’s Active Directory Services, X.500, LDAP
- There are also special directory services called discovery services
- More complicated than name services
- How can we maintain/build directory and discovery services?
Directory Service

• Formal definition
  • A service that stores collections of bindings between names and attributes
  • Looks up entries that match attribute-based specifications

• Popular examples
  • X.500
  • LDAP (Lightweight Directory Access Protocol)
X.500

- Application level service in OSI set of standards (1988?)
- Data stored in X.500 servers is organized into a tree structure
- Name tree is called the Directory Information Tree (DIT)
- Entire directory structure is called Directory Information Base (DIB)
- Servers are called Directory Service Agents (DSA)
- Clients are Directory User Agents (DUA)
- X.500 is almost like DNS for “people”
• DIB entry consists of a set of attributes
• Each attribute has a type and one or more values
• Name of DIB entry is determined by selecting distinguished attributes called Distinguished Names (DN)

• Accessing the directory:
  • Read - specify name (similar to domain name) and desired attributes, DSA navigates DIT and returns requested information
  • Search - specify base name and filter expression, DSA returns DNs for all entries below base name for which filters evaluate to true
X.500

• Updating the DIB
  • DSA interface supports adding, deleting, modifying entries in DIB
  • Expected that DIB is partitioned and replicated, but X.500 standard does not address implementation issues directly

• Issues with X.500
  • Very heavy-weight! Complex and difficult to implement
  • Uses upper layers of network stack
  • Check out Wikipedia…
LDAP to the Rescue

- X.500 is too complex for many applications
- *Lightweight* Directory Access Protocol (LDAP) is based on X.500, but is simplified
- Runs over TCP/IP
- LDAP, unlike X.500, is widely used in Internet applications
- LDAP directory service consists of a number of records made up of (attribute, value) pairs
**LDAP**

- Sample LDAP namespace:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Abbr.</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Country</td>
<td>C</td>
<td>US</td>
</tr>
<tr>
<td>Locality</td>
<td>L</td>
<td>Massachusetts</td>
</tr>
<tr>
<td>Organization</td>
<td>O</td>
<td>Williams College</td>
</tr>
<tr>
<td>OrganizationalUnit</td>
<td>OU</td>
<td>Comp. Sci.</td>
</tr>
<tr>
<td>CommonName</td>
<td>CN</td>
<td>Main server</td>
</tr>
<tr>
<td>Mail_Servers</td>
<td>--</td>
<td>137.164.8.4, 137.165.8.5</td>
</tr>
<tr>
<td>WWW_Server</td>
<td>--</td>
<td>137.165.8.6</td>
</tr>
</tbody>
</table>
LDAP

- Directory entries are again called directory information base (DIB)
- Records are uniquely named so that they can be looked up
- Unique name is derived from sequence of naming attributes
- Use of globally unique names obtained by listing naming attributes in sequence leads to a hierarchy (as in DNS) called directory information tree (DIT)
LDAP

• DIT is partitioned and distributed across several servers (DSAs)
• Each DSA behaves like name server in DNS
• Key difference between LDAP and DNS are the facilities for **searching** through a DIB
• For example, perhaps we want to know all main servers at Williams College
  • answer = search("&(C=US)(O=Williams College)(OU = *)(CN=Main server)")
• These queries are not possible in DNS
• Searches like this can be expensive to complete
Problem

• DNS is a relatively simple hierarchical name service that does not provide “yellow pages” style searching mechanisms

• X.500 and LDAP are hierarchical directory services that provide advanced searching though it can be expensive (have to visit many leaves in the tree)

• Can we avoid expensive searching?

• What data structure provides efficient searching/lookups?
Decentralized Solutions

• How about distributed/decentralized attribute-based naming systems?
• The goal is to provide efficient searching by avoiding an exhaustive (expensive) search
• We can use distributed hash tables (DHTs) to avoid expensive searches and provide efficient lookups
• Very popular in the late 2000s
Distributed Hash Tables

• Hash tables map keys (attributes) to values to provide simple and efficient lookups without searching

• DHTs are essentially hash tables that are partitioned and spread across several nodes in a distributed system

• DHTs tend to be scalable and fault tolerant
**Discovery services**

- **Resource discovery** services provide a similar functionality as directory services for dynamic attributes
  - For example, CPU load on all machines, free memory available, etc.
- In general, *discovery services* are used for more “spontaneous” contexts
- DHTs work well for this too!
- Case study: SWORD performs *resource discovery* using a DHT (…well, at least it *used* to use a DHT)
SWORD Challenges

• Support for dynamic attributes
  • Find machines with CPU load < .5
• Need an expressive query language
• Need a way to express “utility”
Resource monitors collect information about resources

(ii) Logical Database and Query Processor

Populate database with measurements

Candidate nodes

(iii) Matcher and Optimizer

Optimal groups and total penalty accrued
Challenges

• Conceptually, SWORD has three distinct architectural components
• But in reality, the same set of nodes perform all three tasks!

• So how can we use a DHT to make searching for attributes more efficient?
Hash tables are efficient because they provide simple lookups for key-value mappings.

Thus we need to create keys that represent dynamic resource descriptions (i.e., load=2.0).

What hash function should we use?
  - We also want to partition as evenly as possible…

When generating a key, use \( n \) bits to describe attribute name, and \( m \) bits to describe value.
  - Also save a few random bits to help with load balancing.

For each attribute, the possible range of values is partitioned, and each server gets a specific subrange.
SWORD Key Generation

free_mem = 13870396 (KB)

attribute bits

value bits

random bits

key = 0x 3 1 0 0 0 D 3 A 5 3 C 5 E 8 F 9 7 9 C 3
Query Processing

• To process a query for all nodes with load between 0 and 1.0, just visit all servers responsible for the associated keys

• No need to search entire set of nodes!
  • This reduces the search space and improves efficiency
Multi-Attribute Queries

• So….what if you want to search for multiple attributes?
  • Load < 1.0 AND Mem > 500MB

• How do you efficiently lookup keys?
Implementation Tradeoffs

Updates
Queries
Query Replies
Implementation Tradeoffs

- Do we really need to be completely decentralized??