Naming and weak consistency

COS 518: *Advanced Computer Systems*
Lecture 2

Mike Freedman

Naming and system components

- How to design interface between components?
- Many interactions involve naming things
  - Naming objects that caller asks callee to manipulate
  - Naming caller and callee together

Potential Name Syntax

- Human readable?
  - If users interact with the names
- Fixed length?
  - If equipment processes at high speed
- Large name space?
  - If many nodes need unique names
- Hierarchical names?
  - If the system is very large and/or federated
- Self-certifying?
  - If preventing "spoofing" is important

Properties of Naming

- Enabling sharing in applications
  - Multiple components or users can name a shared object.
  - Without names, client-server interface pass entire object by value
- Retrieval
  - Accessing same object later on, just by remembering name
- Indirection mechanism
  - Component A knows about name N
  - Interposition: can change what N refers to without changing A
- Hiding
  - Hides impl. details, don't know where google.com located
  - For security purposes, might only access resource if know name (e.g., dropbox or Google docs URL –> knowledge gives access)

High-level view of naming

- Set of possible names
- Set of possible values that names map to
- Lookup algorithm that translates name to value
  - Global (context-free) or local names?
  - Who supplies context?

5

Hierarchical Assignment Processes

- **Host names:** www.cs.princeton.edu
  - Mnemonic, variable-length, appreciated *by humans*
  - Hierarchical, based on organizations
  - Domain: registrar for each top-level domain (eg, .edu)
  - Host name: local administrator assigns to each host

6

Hierarchical Assignment Processes

- **IP addresses:** 128.112.7.156
  - Numerical 32-bit address appreciated *by routers*
  - Hierarchical, based on organizations and topology
  - Prefixes: ICANN, regional Internet registries, and ISPs
  - Hosts: static configuration, or dynamic using DHCP

7

Hierarchical Assignment Processes

- **MAC addresses:** 00-15-C5-49-04-A9
  - Numerical 48-bit address appreciated *by adapters*
  - Non-hierarchical, unrelated to network topology
  - Blocks: assigned to vendors by the IEEE
  - Adapters: assigned by the vendor from its block

8

Case Study: Domain Name System (DNS)

Computer science concepts underlying DNS

- Indirection: names in place of addresses
- Hierarchy: in names, addresses, and servers
- Caching: of mappings from names to/from addresses

Strawman Solution #1: Local File

- Original name to address mapping
  - Flat namespace
  - /etc/hosts
  - SRI kept main copy
  - Downloaded regularly
- Count of hosts was increasing: moving from a machine per domain to machine per user
  - Many more downloads
  - Many more updates

Strawman Solution #2: Central Server

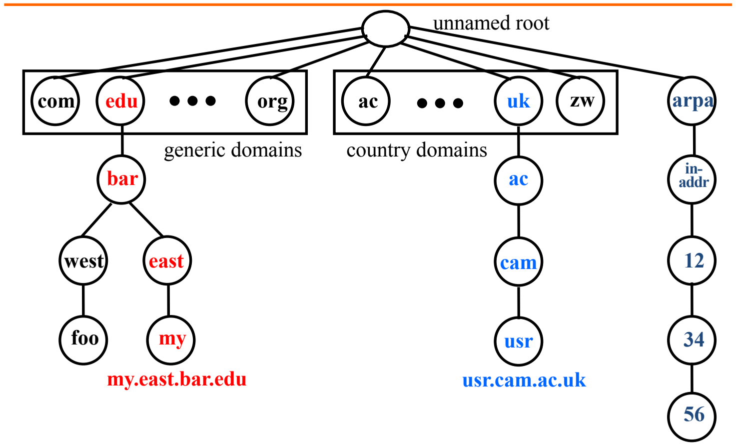
- Central server
  - One place where all mappings are stored
  - All queries go to the central server
- Many practical problems
  - Single point of failure
  - High traffic volume
  - Distant centralized database
  - Single point of update
  - Does not scale

Need a distributed, hierarchical collection of servers

Domain Name System (DNS)

- Properties of DNS
  - Hierarchical name space divided into zones
  - Distributed over a collection of DNS servers
- Hierarchy of DNS servers
  - Root servers
  - Top-level domain (TLD) servers
  - Authoritative DNS servers
- Performing the translations
  - Local DNS servers and client resolvers

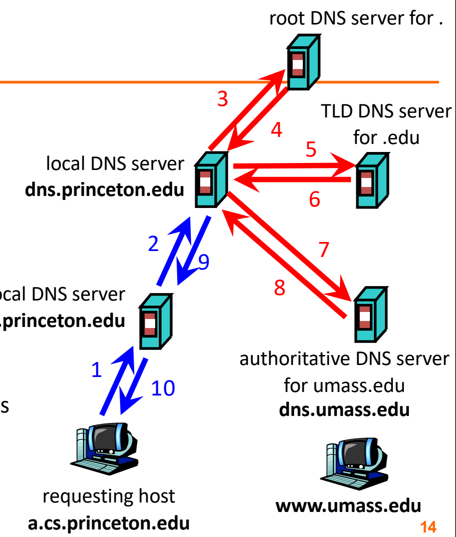
Distributed Hierarchical Database



DNS Queries

a.cs.princeton.edu wants IP address for **www.umass.edu**

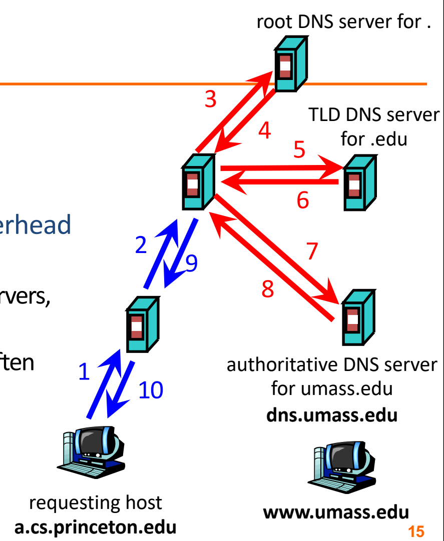
Recursive vs. Iterative Queries



14

DNS Queries

- DNS query latency:
  - e.g., 1 second
- Caching to reduce overhead and delay
  - Small # of top-level servers, that change rarely
  - Popular sites visited often
- Where to cache?
  - Local DNS server
  - Browser



15

Reliability

- DNS servers are replicated
  - Name service available if at least one replica is up
  - Queries can be load balanced between replicas
- UDP used for queries
  - Need reliability: must implement this on top of UDP
- Try alternate servers on timeout
  - Exponential backoff when retrying same server
- Same identifier for all queries
  - Don't care which server responds

16

DNS Cache Consistency

- Goal: Ensuring cached data is up to date
- DNS design considerations
  - Cached data is "read only"
  - Explicit invalidation would be expensive
    - Server would need to keep track of all resolvers caching
- Avoiding stale information
  - Responses include a "time to live" (TTL) field
  - Delete the cached entry after TTL expires
- Perform negative caching (for dead links, misspellings)
  - So failures quick and don't overload gTLD servers

17

Intro to fault tolerant + consistency

18

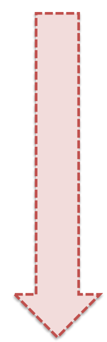
What is fault tolerance?

- Building reliable systems from unreliable components
- Three basic steps
  1. **Detecting errors**: discovering presence of an error in a data value or control signal
  2. **Containing errors**: limiting how far errors propagate
  3. **Masking errors**: designing mechanisms to ensure system operates correctly despite error (+ possibly correct error)

19

Why is fault tolerance hard?

Failures Propagate



- Say one bit in a DRAM fails…
- …it flips a bit in a memory address the kernel is writing to…
- …causes big memory error elsewhere, or a kernel panic…
- …program is running one of many distributed file system storage servers…
- …a client can't read from FS, so it hangs

20

So what to do?

1. **Do nothing**: silently return the failure
2. **Fail fast**: detect the failure and report at interface
   - Ethernet station jams medium on detecting collision
3. **Fail safe**: transform incorrect behavior or values into acceptable ones
   - Failed traffic light controller switches to blinking-red
4. **Mask the failure**: operate despite failure
   - Retry op for transient errors, use error-correcting code for bit flips, replicate data in multiple places

Masking failures

- We mask failures on **one server** via
  - Atomic operations
  - Logging and recovery
- In a distributed system with **multiple servers**, we might replicate some or all servers
- But if you give a mouse some replicated servers
  - She's going to need to figure out how to keep the state of the servers consistent (immediately? eventually?)

Safety and liveness

Reasoning about fault tolerance

- This is hard!
  - How do we design fault-tolerant systems?
  - How do we know if we're successful?
- Often use "properties" that hold true for every possible execution
- We focus on **safety** and **liveness** properties

Safety

- "Bad things" don't happen
  - No stopped or deadlocked states
  - No error states
- E.g., **mutual exclusion:**
  - Two processes can't be in critical section at same time

Liveness

- "Good things" happen
  - ...eventually
- Examples
  - **Starvation freedom:** process 1 can eventually enter a critical section as long as process 2 terminates
  - **Eventual consistency:** if a value in an application doesn't change, two servers will eventually agree on its value

Often a tradeoff

- "Good" and "bad" are application-specific
- Safety is very important in banking transactions
- Liveness is very important in social networking sites

Eventual Consistency

Eventual consistency

- Def'n: If no new updates to the object, eventually all accesses will return the last updated value
- Common: git, iPhone sync, Dropbox, Amazon Dynamo
- Why do people like eventual consistency?
  - Fast read/write of local copy (no primary, no Paxos)
  - Disconnected operation
- Challenges
  - How do you discover other writes?
  - How do you resolve conflicting writes?

29

Two prevailing styles of discovery

- Gossip pull ("anti-entropy")
  - A asks B for something it is trying to "find"
  - Commonly used for management replicated data
    - Resolve differences between DBs by comparing digests
- Gossip push ("rumor mongering"):
  - A tells B something B doesn't know
  - Gossip for multicasting
    - Keep sending for bounded period of time: *O (log n)*
  - Also used to compute aggregates
    - Max, min, avg easy. Sum and count more difficult.
- Push-pull gossip
  - Combines both : O(n log log n) msgs to spread in O(log n) time

Monday's readings

- Everybody:
  - E2E Arguments in System Design
- Signup:
  - Amazon's Dynamo
  - Yahoo!'s PNUTS

31