Distributed Software Development

Introduction

Chris Brooks

Department of Computer Science
University of San Francisco
1-0: Class structure

- 1st half: fundamentals and principles:
  - Mondays: lab on a current piece of technology.
  - Wednesdays: lecture on fundamental problems and principles of distributed systems.

- Work:
  - Weekly labs to turn in.
  - Midterm
1-1: Class structure

- 2nd half: Case studies, examples, and applications
  - More lecture, guest speakers, discussion

- Work:
  - Two projects
    - P2P client
    - Project of your choosing.
  - Final
1-2: Course Policy

- Class participation is very important.
  - Attendance is required
  - Active participation is encouraged (and rewarded!)

- Texts:
  - We will use a number of the O’Reilly texts available through Safari.
  - I will also provide you with handouts covering lecture topics.
  - I will expect you to do the reading before class.

- Languages:
  - For the labs, you must use an ’appropriate’ language.
  - For the projects, you may use whatever language you want.
1-3: What is a distributed system?

(Couloris) “A distributed system is one in which hardware or software components communicate or coordinate their actions only by passing messages.”

This covers everything from a parallel computer to the Internet.
1-4: What is a distributed system?

- So how is this class different from Pacheco’s or Miller’s?
- Different set of challenges:
  - Heterogeneity
  - Openness
  - Scalability
  - Failure models
  - Degree of parallelism
1-5: Issues in distributed systems

The definition Couloris provides leads to the following problems that must be considered:

- Concurrency - work is happening on multiple computers simultaneously and must be coordinated.
- No global clock.
- Independent failures, both network and computer.

We’ll spend a lot of time discussing these problems.
To illustrate some of the issues involved in distributed computing, we’ll look at a well-known distributed system: DNS.

DNS stands for domain system.

This is the system that maps *symbolic* hostnames (such as stargate.cs.usfca.edu) to IP addresses (such as 138.202.171.14).

Symbolic names are much easier for humans to work with.

Computers, on the other hand, do better with IP addresses.

How can we look up the correct IP address for a hostname?
From a web browser’s perspective, resolving a hostname looks like this:
1. The hostname is extracted from the URL.
2. The browser sends a query to a DNS server.
3. The server eventually returns a reply, which contains the corresponding IP address.
4. The browser then opens a TCP connection to that IP address.

From the client side, this looks pretty simple.
Along with translating hostname/IP address pairs, DNS can do the following:

- **Host aliasing** - for example, `nexus.cs.usfca.edu` is also `www.cs.usfca.edu`
  - `nexus.cs` is the *canonical hostname*
- **Mail server aliasing** - the CS mail server is `nexus.cs.usfca.edu`, yet I can send email to `brooks@cs.usfca.edu` successfully.
- **Load distribution**. We can map multiple IP addresses to a single hostname.
  - The DNS server will return all IP addresses, but permute the order.
One way to build a DNS server would be the following:

get a fast Internet connection and set up a single high-speed computer that will do all DNS resolutions via a huge database.

What are some problems with this approach?
1-10: A naive solution

- Single point of failure
- Not scalable
- Not “close to” all clients - unacceptable delay.
- Difficult to maintain.
1-11: How DNS actually works

- DNS is a distributed, hierarchical database.
- Large number of servers worldwide.
- No database contains all DNS entries.
- DNS servers are divided into three classes:
  - Root servers
  - Top-level domain servers
  - Authoritative servers.
Returning to our from-the-client-perspective:
1. The client contacts a root server.
2. This returns the addresses of TLD servers
3. The root server contacts a TLD server (for example, for the .com domain)
4. This returns the address of an authoritative server at cs.usfca.edu (for example).
5. The client then queries this server to find out the IP address of nexus.cs.usfca.edu
Often, the authoritative DNS server will not have the addresses for all hosts in an organization.

Instead, local DNS servers will manage the names for subsections of the network.

This is typically the program that acts as a proxy for the client.

DNS servers also typically cache much of this information so as to avoid resending requests for common lookups.
What issues does this particular design of DNS solve?
What issues does this particular design of DNS solve?

- **Heterogeneity**: It will work with computers using different operating systems, languages, hardware, network interfaces or platforms.
What issues does this particular design of DNS solve?

- Openness - Can new additions, changes, or improvements be made to the system?
- Are the system and its interfaces publically described?
- DNS (and most internet systems and protocols) are publically described in a set of RFCs.
What issues does this particular design of DNS solve?

- Openness - Can new additions, changes, or improvements be made to the system?

Open systems can typically be constructed from heterogeneous components, as long as the vendors/implementors conform to a published standard.

This makes it easier to extend a service or add new services on top of existing ones.

- For example, TCP was added on top of IP.
What issues does this particular design of DNS solve?

- Redundancy - In DNS, every address is stored in at least two servers. If one server fails, others can be queried.
What issues does this particular design of DNS solve?

- Concurrency: Many requests can happen simultaneously.
- No bottleneck waiting for queries to resolve.
- The resource (lookup tables) is distributed across a large number of hosts.
- The system is able to operate consistently in a concurrent environment.
  - This is easier said than done.
1-20: DNS as a distributed system

6 What issues does this particular design of DNS solve?
   △ Transparency - The separation of components is hidden from the user.
   △ No need for a user to know about root or TLD servers.
   △ As in OO design, transparency makes development easier for clients.
What issues does this particular design of DNS solve?

- Scalability - As we add machines, resources, or users to the system, how quickly does performance degrade?
- How do costs change as more resources or machines are added?
- Will resources ever run out?
- Are there bottlenecks in the system?

DNS provides a scalable way to do hostname resolution.
What issues does this particular design of DNS solve?

- Handling failure. This includes both detecting and recovering from failure.
- This might also include proceeding in the face of failures that are suspected, but can’t be detected.

DNS is able to recover from or tolerate failure in the sense that, if one local DNS server fails, hosts on other parts of the Internet can still be resolved.