Communication, Services, and Coordination

Architectures for coordination?

What assumptions can we make:
- about the network?
- about the nodes?
How do these properties affect the software and its behavior?

Services

request/response paradigm ==> client/server roles
- Remote Procedure Call (RPC)
- object invocation, e.g., Remote Method Invocation (RMI)
- HTTP (the Web)
- device protocols (e.g., SCSI)

How does the Web work?
The canonical example in your Web browser

Click here
“here” is a Uniform Resource Locator (URL)

http://www-cse.ucsd.edu
It names the location of an object (document) on a server.

In Action…

http://www-cse.ucsd.edu
- Client uses DNS to resolve name of server (www-cse.ucsd.edu)
- Establishes an HTTP connection with the server over TCP/IP
- Sends the server the name of the object (null)
- Server returns the object

HTTP in a Nutshell

HTTP supports request/response message exchanges of arbitrary length. Small number of request types: basically GET and POST, with supplements. object name, + content for POST optional query string optional request headers Responses are self-typed objects (documents) with attributes and tags. optional cookies optional response headers
HTTP began as a souped-up FTP that supports hypertext URLs.
Service builders rapidly began using it for dynamically-generated content.
Web servers morphed into Web Application Servers.
Common Gateway Interface (CGI)
Java Servlets and JavaServer Pages (JSP)
Microsoft Active Server Pages (ASP)
“Web Services”

**Review: Network Protocols**

<table>
<thead>
<tr>
<th>Layer</th>
<th>Protocols</th>
</tr>
</thead>
</table>
| L7    | Application
|       | Presentation |
| L5-7  | MIME, SSL
|       | SOAP, etc. |
| L4    | HTTP
|       | TCP |
| L3    | IPv4, IPv6
|       | UDP, TCP |
| L2    | Ether

**Assumptions About the Network**

Most of what we study in this class is at the session or presentation levels of the OSI “layer cake”.
We assume properties of the transport and network layers:
- uniform network address space (IP address, port)
- best-effort delivery of messages of arbitrary size
- reliable ordered stream communication (TCP)
- flow and congestion control

The key issue is: how to use the network to build networked applications and services with the properties we want?
In practice, many critical structuring and performance issues do not permit us to draw so clean a line...but we’ll try.

**Web Protocols**

What kind of transport protocol should the Web use?
HTTP 1.0
- One TCP connection per request
  - Complaints: inefficient, slow, burdensome…
HTTP 1.1
- One TCP connection/many requests (persistent connections)
  - Solves all problems, right? Huge amount of complexity
    - Clients, proxies, servers
How do they compare?
- Protocol differences [Krishnamurthy99], performance comparison
  - [Nielsen97], effects on servers [Manley97], overhead of TCP connections [Caceres98]

HTTPS: HTTP with authentication and encryption

**Persistent Connections**

There are three key performance reasons for persistent connections:
- connection setup overhead
- TCP slow start: just do it and get it over with
- pipelining as an alternative to multiple connections
And some new complexities resulting from their use, e.g.:
- request/response framing and pairing
- unexpected connection breakage
  - Just ask anyone from Akamai...
- large numbers of active connections
  - How long to keep connections around?

These motivations and issues manifest in HTTP, but they are fundamental for request/response messaging over TCP.
Internet Growth and Scale

The Internet

http://www.netsizer.com

How to handle all those client requests raining on your server?

Scaling Server Sites: Clustering

Goals
- server load balancing
- failure detection
- access control filtering
- priorities/QoS
- request locality
- transparent caching

What to switch/filter on?
- L3 source IP and/or VIP
- L4 (TCP) ports etc.
- L7 URLs and/or cookies
- L7 SSL session IDs

Scaling Services: Replication

Distribute service load across multiple sites.

How to select a server site for each client or request?

Is it scalable?

Scaling with Peer-to-Peer

Is (e.g.) Napster a service?

Is the peer-to-peer approach fundamentally more scalable?

More robust?

What does it assume about the clients?

Coordination

If the solution to availability and scalability is to decentralize and replicate functions and data, how do we coordinate the nodes?

- data consistency
- update propagation
- mutual exclusion
- consistent global states
- group membership
- group communication
- event ordering
- distributed consensus
- quorum consensus

Fundamental Questions

Synchronous vs. asynchronous
- Are the node clocks synchronized? Is there a bound on drift?
- How long can messages be delayed?
- How long can it take a node to respond to a message?

Failure model:
- Is message delivery reliable?
- Do failed nodes:
  - Stop forever? (fail-stop)
  - Restart in initial state?
  - Restart and recover some previous state?
  - Behave in an unpredictable fashion (byzantine)?
  - Lie about identity and/or corrupt messages from other nodes?
- How long can recovery be delayed?