Clients and Servers as Interacting Processes

while (true) {
  RcvRequest();
  ... 
  SendReply();
  ... 
}

while (true) {
  SendRequest();
  RcvReply();
  ... 
  RcvRequest();
  ... 
  SendReply();
  ...
}

Stream Example: HTTP 1.0

iaddr = InetAddress.getByName("www.duke.edu");
s = new ServerSocket(80);
clientconn = s->accept();
s = new Socket(iaddr, 80);
connect
accept
request
(s->read*); (800)
reply
response = (s->read*);
(s->close());
httpd
Web browser

Are URLs transparent?

other players:
DNS
TCP
MIME
HTML
thousands of helpers

cmd = (s->read*);

Are URLs transparent?

...???...

*write*(response);

...???...

s->close();

*write*("GET <URL>");

*write*(response);

Remote Procedure Call

Language-integrated request/response communication is often called remote procedure call (RPC).

- Generally accepted as a paradigm for building network services structured as a set of exported operations. Operations of a server are “just like” procedures in a module.
- Client thread(s) issue a request and block awaiting the reply. Synchronous operation is “just like” a local procedure call. Reply message arrival is “just like” a return from the call.
- With a little syntactic sugar, the network communication can be hidden from the calling code.
Mechanics of RPC

RPC communication is supported by an RPC layer in the network protocol stack.

- Client thread issues a request by calling a stub procedure, possibly generated by compiling an interface description.
- Stub formats and sends request, and blocks for reply. Arguments/results represented in a network standard format (e.g., external data representation or XDR).
- Awaken waiting thread when reply arrives. An xid (“transaction” ID) in each request/reply message pairs requests with replies.
- Stub unpacks reply and returns to caller.

From Servers to Services

Are Web servers and RPC servers scalable? Available?

A single server process can only use one machine.
Upgrading the machine causes interruption of service.
If the process or machine fails, the service is no longer reachable.

We improve scalability and availability by replicating the functional components of the service.
(May need to replicate data as well, but save that for later.)
- View the service as made up of a collection of servers.
- Pick a convenient server: if it fails, find another (fail-over).
Using Clusters for Scalable Services

Clusters are a common vehicle for improving scalability and availability at a single service site in the network.

Are network services the Killer App for clusters?

• incremental scalability
  just wheel in another box...
• excellent price/performance
  high-end PCs are commodities: high-volume, low margins
• fault-tolerance
  “simply a matter of software”
• high-speed cluster interconnects (SANs) are on the market
  cluster nodes can coordinate to serve requests w/ low latency

[Fox/Brewer]: SNS, TACC, and All That

[Fox/Brewer97] proposes a cluster-based generic software infrastructure for scalable network services (“SNS”).

• TransSend: scalable, active proxy middleware for the Web
  think of it as a dial-up ISP in a box, in use at Berkeley
  distills/transforms pages based on user request profiles
• Inktomi/HotBot search engine
  core technology for Inktomi Inc., today a $120M company
  “bringing parallel computing technology to the Internet”

Potential services are based on Transformation, Aggregation, Caching, and Customization (TACC)
TACC [Fox/Brewer]

Vision: deliver “the content you want” by viewing HTML content as a dynamic, mutable medium.

1. Transform Internet content according to:
   - network and client limitations
     e.g., on-the-fly compression/distillation [ASPLOS96], packaging Web pages for PalmPilots, etc.
   - user requests kept in a profile database

2. Aggregate content from different back-end services.

3. Cache content to reduce cost/latency of delivery.

4. Customize (see Transform)

TranSend Structure

[adapted from Armando Fox (through http://ninja.cs.berkeley.edu/pubs)]
SNS/TACC Philosophy

1. Specify services by plugging generic programs into the TACC framework, and compose them as needed.
   - sort of like CGI with pipes
   - run by long-lived worker processes that serve request queues
   - allows multiple languages, etc.

2. Worker processes in the TACC framework are loosely coordinated, independent, and stateless.
   - ACID vs. BASE
     - serve independent requests from multiple users
     - narrow view of a “service”: one-shot readonly requests, and stale data is OK

3. Handle bursts with designated overflow pool of machines.

SNS/TACC Functional Issues

1. What about fault-tolerance?
   - Service restrictions allow simple, low-cost mechanisms.
     - Primary/backup process replication is not necessary with BASE model and stateless workers.
   - Uses a process-peer approach to restart failed processes.
     - Processes monitor each other’s health and restart if necessary.
     - Workers and manager find each other with “beacons” on well-known ports.

2. Load balancing?
   - Manager gathers load info and distributes to front-ends.
   - How are incoming requests distributed to front-ends?
Server Load Balancing

A basic problem for multi-server services: how do we evenly distribute incoming requests among the specific servers?

- It must be fast and reliable.
  A common front end can be a bottleneck and/or point of failure.
- It should be transparent to the client....
  ...unless we can easily download a smart stub into the client.
- The “balancing act” should be fine-grained.
  Each session request from a given client should be routed independently to an appropriate server.
- Across a WAN, it should consider network topology.
  Selected server should be close as well as lightly loaded.

The Server Selection Problem

Which network site?
Which server?
“Contact the weather service.”

better solutions
- DNS round robin [Brisco, RFC 1794]
- IBM eNetwork Dispatcher
- Cisco LocalDirector
- Alteon “layer 4 server switching”
- WebOS “smart clients” etc. [Vahdat97]

not-so-great solutions
- static client binding
- manual selection
- HTTP forwarding
Network Address Translation

Smart router or switch:
1. recognizes connect request (TCP SYN)
2. selects specific server (d)
3. replaces x with d in connect request packet
4. remembers session \((C,p1),(d,p2)\)
5. for incoming packets from \((C,p1)\) for \((x,p2)\)
   replace virtual IP address x with d
   forward to d
6. for outgoing packets from \((d,p2)\) for \((C,p1)\)
   replace d with x
   forward to C

Examples
Cisco LocalDirector
Alteon ACEdirector

Network Dispatching

IBM’s eNetwork Dispatcher exploits asymmetry in the traffic stream to reduce cost by avoiding translations of outgoing traffic.

Dispatcher node:
1. recognizes connect request (TCP SYN)
2. selects specific server (d)
3. replaces x with d in connect request packet
4. remembers session \((C,p1),(d,p2)\)
5. for incoming packets from \((C,p1)\) for \((x,p2)\)
   replace virtual IP address x with d
   forward to d
6. for outgoing packets from \((d,p2)\) for \((C,p1)\)
   replace d with x
   forward to C
Issues for Server Load Balancing

- monitoring scheme and server selection algorithm
- accommodating redundancy
- support for differentiated quality-of-service (QoS) among different traffic streams
- filtering and security (e.g., integrated firewall)
- failover, preserving session guarantees
- what is the effect of HTTP 1.1?
- combining local and wide-area approaches
- client-based solutions