What is a “distributed system”?

• Very broad definition
  – loosely-coupled to tightly-coupled
• Nearly all systems today are distributed in some way
  – they use email
  – they access files over a network
  – they access printers over a network
  – they're backed up over a network
  – they share other physical or logical resources
  – they use the web
  – they cooperate with other people on other machines

Distributed systems are now a requirement

• Economics dictate that we buy small computers
• Everyone needs to communicate
• We need to share physical devices (printers) as well as information (files, etc.)
• Many applications are by their nature distributed (bank teller machines, airline reservations, ticket purchasing)
• To solve the largest problems, we will need to get large collections of small machines to cooperate together (parallel programming)

Loosely-coupled systems

• Earliest systems used simple explicit network programs
  – FTP (rcp): file transfer program
  – telnet (rlogin/rsh): remote login program
  – mail (SMTP)
• Each system was a completely autonomous independent system, connected to others on the network

Closely-coupled systems

• A distributed system becomes more “closely-coupled” as it
  – appears more uniform in nature
  – runs a “single” operating system
  – has a single security domain
  – shares all logical resources (e.g., files)
  – shares all physical resources (CPUs, memory, disks, printers, etc.)
• In the limit, a distributed system looks to the user as if it were a centralized timesharing system, except that it’s constructed out of a distributed collection of hardware and software components

• Even today, most distributed systems are loosely-coupled
  – each CPU runs an independent autonomous OS
  – computers don’t really trust each other
  – some resources are shared, but most are not
  – the system may look differently from different hosts
  – typically, communication times are long
Tightly-coupled systems

- A “tightly-coupled” system usually refers to a multiprocessor
  - runs a single copy of the OS with a single job queue
  - has a single address space
  - usually has a single bus or backplane to which all processors and memories are connected
  - has very low communication latency
  - processors communicate through shared memory

Grapevine distributed mail service

- Xerox PARC, 1980
  - cf. Microsoft Outlook/Exchange today!!!!!
- Goals
  - cannot rely on integrity of client
  - once the system accepts mail, it will be delivered
  - no single Grapevine computer failure will make the system unavailable to any client either for sending or for receiving mail
- Components
  - GrapevineUser package on each client workstation
  - Registration Servers
  - Message Servers
- Implementation: Remote Procedure Call

Grapevine: Functional diagram

Grapevine: Sending a message

- User prepares message using mail client
- Mail client contacts GrapevineUser package on same workstation to actually send message
- GrapevineUser package
  - contacts any Registration Server to get a list of Message Servers
  - contacts any Message Server to transmit message
    - presents source and destination userids, and source password, for authentication
      - Message Server uses any Registration Server to authenticate
    - sends message body to Message Server
      - Message Server places it in stable storage and acknowledges receipt

Grapevine: Transport and buffering

- For each recipient of the message, Message Server contacts any Registration Server to obtain list of Message Servers holding mail for that recipient
- Sends a copy of the message to one of those Message Servers for that recipient

Grapevine: Retrieving mail

- User uses mail client to contact GrapevineUser package on same workstation to retrieve mail
- GrapevineUser package
  - contacts any Registration Server to get a list of each Message Server holding mail for the user ("inbox site")
  - contacts each of these Message Servers to retrieve mail
    - presents user credentials
    - Message Server uses any Registration Server to authenticate
    - acknowledges receipt of messages so that the server can delete them from its storage
Grapevine: Scalability

- Can add more Registration Servers
- Can add more Message Servers
- Only thing that didn’t scale was handling of distribution lists
  - the accepting Message Server was responsible for expanding the list (recursively if necessary) and delivering to an appropriate Message Server for each recipient
  - some distribution lists contained essentially the entire user community
- Jeff Dean (Google) told us they don’t even think about more than two decimal orders of magnitude
  - fundamental design decisions will need to change
  - advances in technology will make it possible

Some issues in distributed systems

- Transparency (how visible is the distribution)
- Security
- Reliability
- Performance
- Scalability
- Programming models
- Communication models