CSC 539: Operating Systems Structure and Design

Spring 2006

Distributed systems

- networked vs. distributed operating systems
- network structure: LAN vs. WAN
- Distributed File Systems (DFS's)
  - naming, remote file access, caching, stateless vs. stateful
  - example: AFS
- distributed coordination
  - distributed mutual exclusion: centralized vs. fully distributed approaches

1

Distributed systems

a *distributed system* is collection of loosely coupled processors interconnected by a communications network

- processors are variously called *nodes, computers, machines, hosts*
- location of the processor is called the *site*

reasons for distributed systems

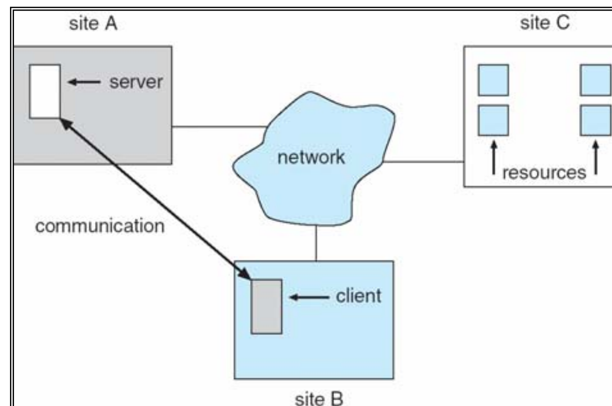
- resource sharing
  - sharing and printing files at remote sites
  - processing information in a distributed database
  - using remote specialized hardware devices
- computation speedup
  - if a computation can be partitioned, can distribute and run concurrently
  - can also move jobs from overloaded processors to less busy ones (*load sharing*)
- reliability
  - detect and recover from site failure, transfer its functionality, reintegrate failed site
- communication
  - more cost effective to have many cheap machines, allow message passing

2

Client-server model

a common approach is to have

- servers: hosts that provide/manage resources (e.g., database access, printer)
- clients: hosts that request resources or services from the servers



3

Network-oriented operating systems

there are 2 general categories of network-oriented OS's

- network operating systems
- distributed operating systems

network operating systems

- users are aware of the multiplicity of machines
- access to resources of various machines is done explicitly by:
  - ✓ remote logging into the appropriate remote machine (telnet, ssh)
  - ✓ remote desktop (Microsoft Windows)
  - ✓ transferring data from remote machines to local machines, via the File Transfer Protocol (FTP) mechanism

distributed operating systems

- users are not aware of multiplicity of machines
- access to remote resources similar is to access to local resources

4

Distributed operating systems (cont.)

when a user want to access/process remote data, can either do

- *data migration:* transfer the required data to the user's machine
  - if any modifications are made, must transfer changes back when done
  - (transfers can be done as entire files, or smaller blocks)
- *computation migration:* transfer the computation, rather than the data
  - especially useful if remote site contains a large, complex database
  - requires some sort of message passing system, e.g., RPC's

a logical extension of computation migration is *process migration*

- execute an entire process, or parts of it, at different sites
- possible reasons:
  - load balancing – distribute processes across network to even the workload
  - computation speedup – subprocesses can run concurrently on different sites
  - hardware preference – process execution may require specialized processor
  - software preference – required software may be available at only a particular site
  - data access – run process remotely, rather than transfer all data locally

5

The Web as a distributed-computing environment

data migration

- files are stored on across the Internet on servers
- when a client requests a page via a browser (using HTTP), file is downloaded

computation migration

- client can trigger the execution of server-side programs using CGI, servlets, ASP, server-side includes, PHP, …

process migration

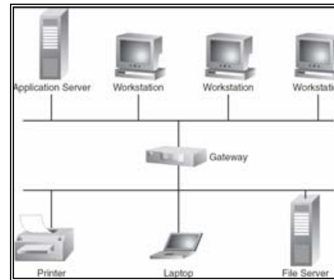
- JavaScript programs and Java applets are downloaded, executed on the client

6

Network structures: LAN vs. WAN

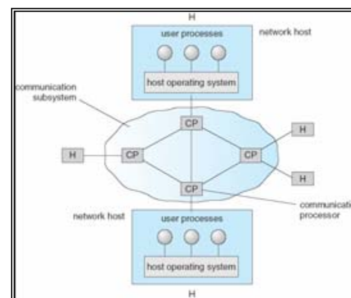
local-area network (LAN) – designed to cover small geographical area.

- emerged in the early 1970's as an alternative to large mainframes
  - more effective to have many cheap machines, connected to share resources
- common configurations: multiaccess bus, ring, star network
- broadcast is fast and cheap (10-1000 Mb/sec)



wide-area network (WAN) – links geographically separated sites

- emerged in the late 1960's (ARPANet)
  - provide point-to-point connections over long-haul lines, resource sharing
- broadcast usually requires multiple messages, reliability can be an issue
  - (speed ≈ 1.544 – 45 Mb/sec)



Distributed file systems (DFS)

a DFS manages set of dispersed storage devices

- provides services (read/write/access/remove files) to clients
- ideally, the DFS should provide transparency to the user
  - i.e., not distinguish between local and remote files

common DFS's

- Network File System (NFS), Sun Microsystems
- Andrew File System (AFS), CMU → OpenAFS, IBM
- Distributed File System (Dfs), Microsoft
- Apple File Protocol (AFP), Apple

in a conventional file system, performance is based on

service time for request = seek time + rotational latency

in a distributed file system, remote access has additional overhead

service time for request = (send request to remote server +)
seek time + rotational latency
(+ transfer time back to client)

Naming

naming is the mapping between logical and physical objects

- *location transparency:* file name should not reveal its physical storage location
- *location independence:* file name should not need to be changed when its physical storage location changes *(supported by AFS, few others)*

3 main approaches to naming schemes

1. files named by combination of their host name and local name; guarantees a unique systemwide name
   e.g., URL in HTTP
2. attach remote directories to local directories, giving the appearance of a coherent directory tree
   only previously mounted remote directories can be accessed transparently
   e.g., NSF attaches remote directories to local directories, giving virtual hierarchy
3. total integration of the component file systems
   a single global name structure spans all the files in the system
   difficult to manage in practice

9

Remote file access

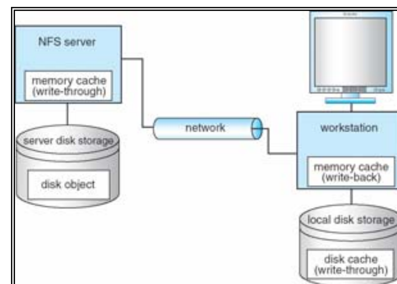
a *remote-service mechanism* defines requests to the server, server access, and transfer back to the client

- commonly implemented via remote procedure calls (RPC's)

to improve performance, caching is commonly used

- can think of as a *network virtual memory*
- cache can be stored on disk (more reliable), or main memory (faster)
- NFS does not provide for disk caching
  recent Solaris implementations added a disk caching option, *cachefs*

  cache data is stored both in main memory and on a local disk



10

Cache consistency

as with any form of caching, maintaining consistency is an issue

- *write-through:* can write data through to disk as soon as placed on any cache
  reliable, but poor performance
- *delayed-write:* modifications written to the cache, written through to the server later
  good performance; but poor reliability (if disk crashes, all is lost)
- *variations:*
  scan cache at regular intervals and flush blocks that have been modified
  write data back to the server when the file is closed

with caching, many remote accesses can be handled by local cache

- reduces server load and network traffic; enhances potential for scalability
- (requires client machines to have local disks or large main memories)

without it, every remote access involves network traffic & server processing

caching is superior in access patterns with infrequent writes

- with frequent writes, substantial overhead is incurred to overcome cache-consistency problem

11

Stateful vs. stateless file service

2 approaches for storing server-side info on remote accesses

stateful file service

- server tracks each file being accessed by each client, e.g.,
  1. client opens a file
  2. server fetches information about the file from its disk, stores it in its memory, and gives the client a connection identifier unique to the client and the open file
  3. identifier is used for subsequent accesses until the session ends
  4. server must reclaim the main-memory space used by clients who are no longer active
- improved performance: keeps client info in main memory, fewer disk accesses
- examples: AFS, NFS V4

stateless file service

- server provides requested blocks, no other knowledge maintained
  avoids state information by making each request self-contained
  each request identifies the file and position in the file
- no need to establish and terminate a connection by open and close operations
- example: NFS V3

12

Tradeoffs between stateful & stateless service

failure recovery

- a stateful server loses all its volatile state in a crash
  - restore state by recovery protocol based on a dialog with clients, or abort operations that were underway when the crash occurred
  - server needs to be aware of client failures in order to reclaim space allocated to record the state of crashed client processes
- with a stateless server, the effects of server failure are almost unnoticeable
  - a restored server can respond to a self-contained request without any difficulty

penalties for using the robust stateless service:

- longer request messages
- slower request processing
- additional constraints imposed on DFS design

note: some environments require stateful service

- UNIX use of file descriptors and implicit offsets is inherently stateful

13

An example: AFS

Andrew File System (AFS)

- distributed computing environment developed at CMU
- purchased by IBM and released as Transarc DFS
- now open sourced as OpenAFS

- most feature-rich nonexperimental DFS
- highly scalable: can span 5,000 workstations

clients are presented with a partitioned space of file names: a *local name space* and a *shared name space*

- dedicated servers present the shared name space to the clients as an homogeneous, identical, and location transparent file hierarchy
- the local name space is the root file system of a workstation, from which the shared name space descends
- servers collectively are responsible for the storage and management of the shared name space
  - *access lists protect directories, UNIX bits protect files*

14

AFS: whole file caching

fundamental architectural principle is the caching of entire files

- opening a file causes it to be cached, in its entirety, on the local disk
- subsequent reads/writes are done to local copy (so fast!)

- cache consistency is ensured using *write-on-close* and the *callback* mechanism
  - ✓ when a client caches a file/directory, the server updates its state info
    we say the client has a callback on that file
  - ✓ the server notifies that client if any other client wishes to write to the file
    we say that the server has removed the client's callback
  - ✓ a client can only open the cached copy if it has a callback
    otherwise, subsequent opens must get the updated version from the server

- the LRU replacement algorithm is used to keep the cache size bounded

15

Distributed coordination

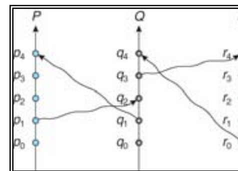
Chapters 6 & 7 described mechanisms for processes to coordinate actions, share resources, and avoid deadlocks

- these same issues arise in a distributed system, but coordination is more difficult

simple example: determining event ordering is difficult

- since distributed systems have separate clocks which may not be synchronized, can't simply go by a timestamp to know which event came first

- to maintain a partial ordering, define a *happened-before* relation (denoted by $\rightarrow$)
  - ✓ If A and B are events in the same process, and A was executed before B, then $A \rightarrow B$
  - ✓ If A is the event of sending a message by one process and B is the event of receiving that message by another process, then $A \rightarrow B$
  - ✓ If $A \rightarrow B$ and B $\rightarrow C$ then $A \rightarrow C$



- in practice, each computer maintains its own logical clock and timestamps events
  - ✓ on same processor P_j, $A \rightarrow B$ implies $\text{timestamp}_j(A) < \text{timestamp}_j(B)$
  - ✓ if P_j receives a message from P_k about event C and $\text{timestamp}_k(C) < \text{clock}_j$, then update $\text{clock}_j = \text{timestamp}_k(C) + 1$

16

Distributed Mutual Exclusion (DME)

can solve the Critical Section problem using

- a *centralized* approach
  1. one of the processes in the system is chosen as coordinator
  2. a process wanting to enter its critical section sends a **request** to the coordinator
  3. the coordinator decides which process can enter the critical section next, and sends that process a **reply** message
  4. when the process receives a reply message, it enters its critical section
  5. after exiting its critical section, the process sends a **release** message to the coordinator and proceeds with its execution

- a *fully distributed* approach
  1. when P_i wants to enter its critical section, it generates a new timestamp, *TS*, and sends the message **request (P_i, TS)** to all other processes in the system
  2. when each process receives a request message, it will **reply** if it doesn't want its critical section or if its time stamp is greater than TS (otherwise, it defers)
  3. when P_i receives a reply message from all other processes in the system, it can enter its critical section
  4. after exiting its critical section, P_i sends **release** messages to all its deferred requests

17

Centralized vs. fully distributed DME

both approaches ensure mutual exclusion, avoid starvation & deadlock

fully distributed requires less message passing

- the number of messages per critical-section entry is the theoretical minimum

$$2 \times (n - 1) \qquad \text{assuming n processes}$$

drawbacks of fully distributed

- the processes need to know the identity of all other processes in the system, which makes the dynamic addition and removal of processes more complex
- if one of the processes fails, then the entire scheme collapses (can be dealt with by continuously monitoring the state of all the processes in the system)
- processes that have not entered their critical section must pause frequently to assure other processes that they intend to enter the critical section

- fully distributed is therefore suited for small, stable sets of cooperating processes

18