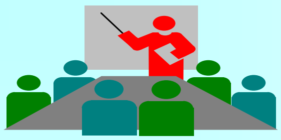
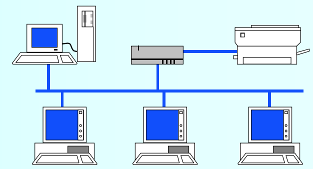




Distributed Systems
600.437
Distributed Operating Systems

Department of Computer Science
The Johns Hopkins University

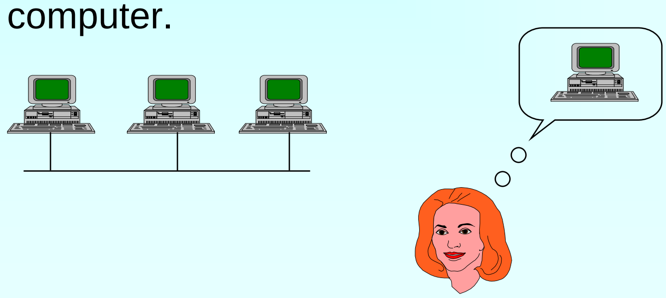
Distributed Operating Systems

Lecture 13

Distributed Operating Systems, Andrew Tanenbaum.
In Search of Clusters, Gregory F. Pfister.

A Distributed Operating System

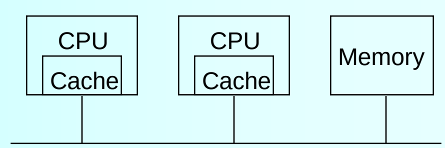
An operating system which manages a collection of independent computers and makes them appear to the users of the system as a single computer.



Hardware Architectures

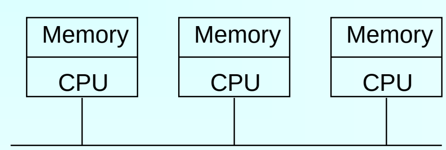
- Multiprocessors
  - Tightly coupled.
  - Shared memory.

Parallel architecture



- Multicomputers.
  - Loosely coupled.
  - Private memory.
  - Autonomous.

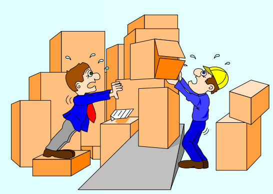
Distributed architecture



Software Architectures

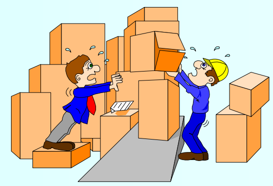
- Multiprocessor OS
  - Looks like a virtual uniprocessor, contains only one copy of the operating system, communication via shared memory, single run-queue
- Network OS
  - Does not look like a virtual uniprocessor, contains n copies of the operating system, communication via shared files, n run-queues
- Distributed OS
  - Looks like a virtual uniprocessor (more or less), contains n copies of the operating system, communication via messages, n run-queues

Design Issues



- Transparency
  - Location transparency
    - Location of processes
    - Location of cpu's and other devices
    - Location of files
  - Replication transparency (of files)
  - Concurrency transparency (the user should not notice the existence of other users).
  - Parallelism transparency (the user should write a serial program, and the compiler and the OS will handle the rest) - this is not doable today

Design Issues (cont.)



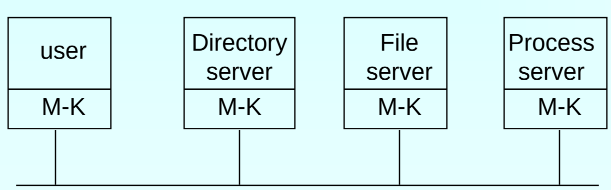
- Performance
  - Throughput / response time
  - Load balancing (static/dynamic)
  - Communication is slow compared to processing speed : Fine grain / coarse grain
- Scalability
- Reliability
- Flexibility (Micro-kernel architecture).

Micro Kernel

Provides:

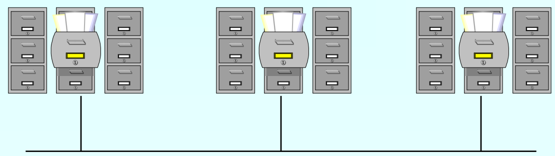
- Inter-process communication mechanisms
- Low level I/O
- Some memory management
- Process management and scheduling

All the rest is done in user level.



Distributed File Systems

- File and directory naming
- Semantics of file sharing
- Implementation considerations
  - Caching
  - Update protocols
  - Replication



File and Directory Naming

- Machine + path naming /machine/path.
  - one name space, but not transparent.
- Mounting remote file systems onto the local file hierarchy.
  - The view of the file system may be different at each computer.
- A single name space that looks the same on all machines.
  - Full naming transparency.

File Sharing Semantics

- One-copy semantics (unix semantics).
  - Updates are written to the single copy and are available immediately.
- Serializability.
  - Transaction semantics (locking files - share for read and exclusive for write).
- Session semantics.
  - Copy the file on open, work on local copy, and copy back on close.

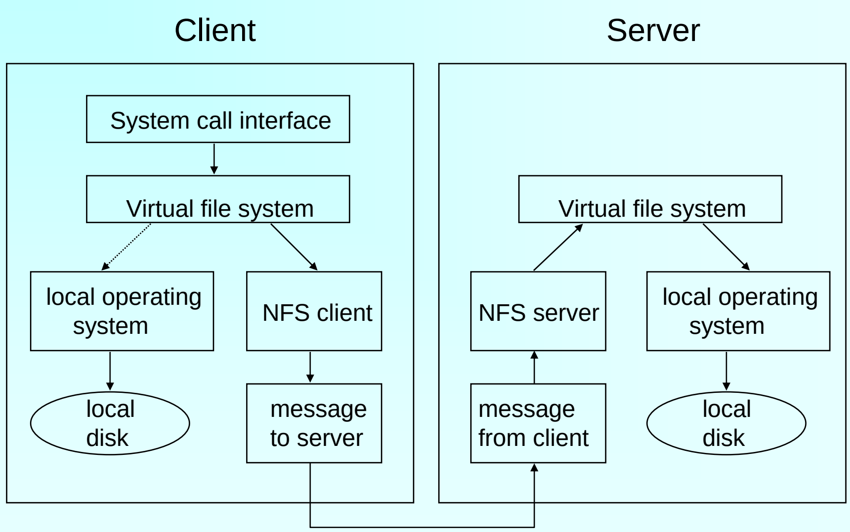
Sun-NFS

- Supports heterogeneous systems.
- Architecture
  - Server exports one or more directory trees for access by remote clients.
  - Clients access exported directory trees by mounting them to the client local tree.
  - Diskless clients can mount exported directory to their root directory.
  - Auto-mount (on the first access).

Sun-NFS (cont.)

- Protocols
  - Mounting protocol
  - Directory and file accessing protocol
    - Stateless
    - No open / close messages.
    - Each read / write message contain the full path and file description position.
- Semantics
  - Not entirely Unix since there is no way to lock files

Sun-NFS Structure

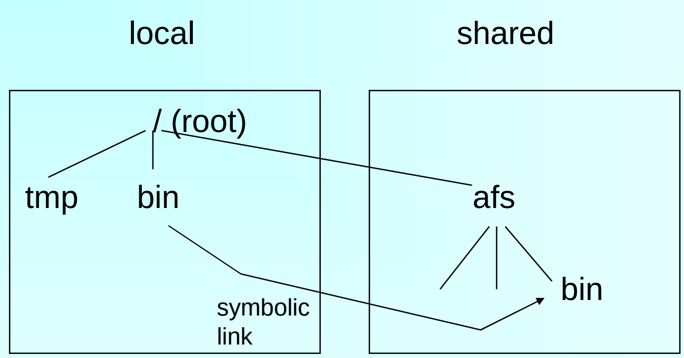


Andrew File System

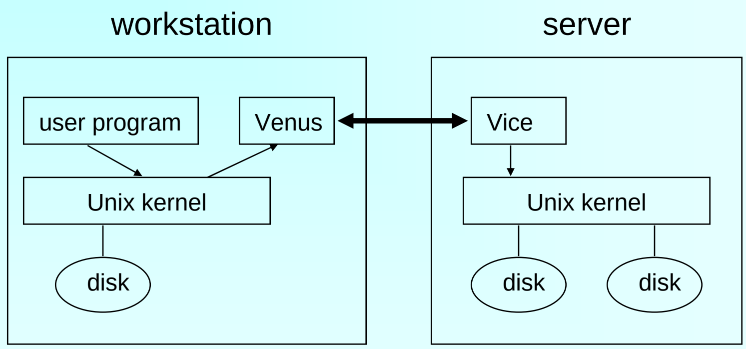
- Supports information sharing on a large scale (thousands of workstations).
- Uses a session semantics.
- The entire file is copied to the local machine (Venus) from the server (Vice) when open. If the file is changed - it is copied to the server when closed.

- The method works because in practice most files are changed by one person only (non-database).
- Measurements show that only 0.4% of changed files were updated by more than one user during one week.

Andrew File System (cont)



AFS Structure



AFS File Validation

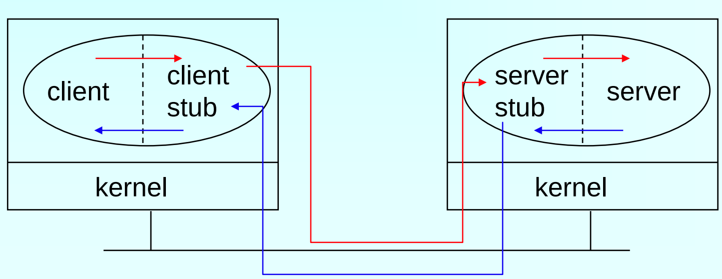
- Older AFS versions:
  - On open: the Venus access the vice to see if its copy of the file is still valid. This causes a substantial delay even if the copy is valid.
  - The Vice is stateless
- Newer AFS versions:
  - The Vice maintains lists of valid copies. If a file is modified the Vice invalidates other copies.
  - On open: if the Venus has a valid copy it can open it immediately.
  - If Venus crashes it has to invalidate its version or check their validity.

The Coda File System

- Descendant of AFS that is substantially more resilient to server and network failures.
- Support for “mobile” users.
- Directories are replicated in several servers (Vice)
- When the Venus is disconnected, it uses local versions of files. When Venus reconnects, it reintegrates using optimistic update scheme.

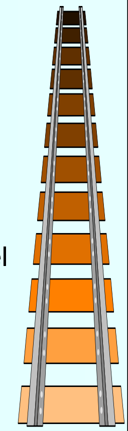
Remote Procedure Call

A convenient way to construct a client-server connection without explicitly writing send/ receive type programs (helps maintain transparency).



Remote Procedure Call (cont.)

- Client procedure **calls** the client stub in a normal way
- Client stub **builds** a message and **traps** to the kernel
- Kernel **sends** the message to remote kernel
- Remote kernel **gives** the message to server stub
- Server stub **unpacks** parameters and **calls** the server
- Server **computes** results and **returns** it to server stub
- Server stub **packs** results in a message and **traps** to kernel
- Remote kernel **sends** message to client kernel
- Client kernel **gives** message to client stub
- Client stub **unpacks** results and **returns** to client

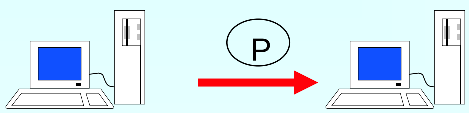


RPC NG: DCOM & CORBA

- Object models allow services and functionality to be called from distinct processes
- DCOM/COM+(Win2000) and CORBA IIOP extend this to allow calling services and objects on different machines
- More OS features (authentication,resource management,process creation,…) are being moved to distributed objects.

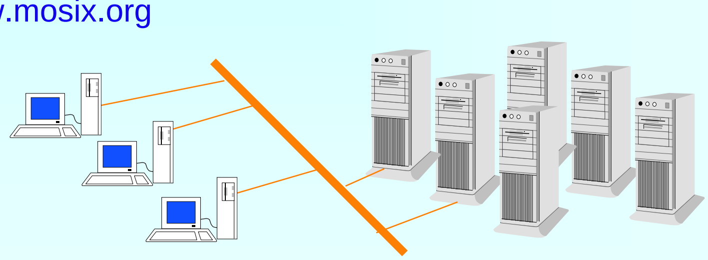
Process Migration

- Load Balancing
  - Static load balancing - CPU is determined at process creation.
  - Dynamic load balancing - processes dynamically migrate to other computers to balance the CPU (or memory) load.
- Migration architecture
  - One image system
  - point of entrance dependent system (the deputy concept)

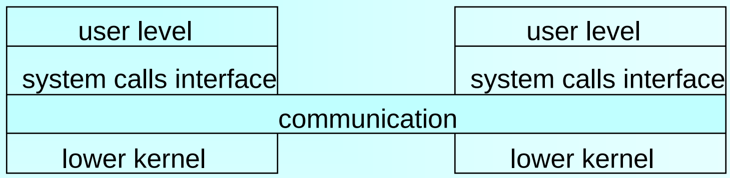


A Mosix Cluster

- Mosix (from Hebrew U): Kernel level enhancement to Linux that provides dynamic load balancing in a network of workstations.
- Dozens of PC computers connected by local area network (Fast-Ethernet and up).
- Any process can migrate anywhere anytime.
- www.mosix.org



An Architecture for Migration

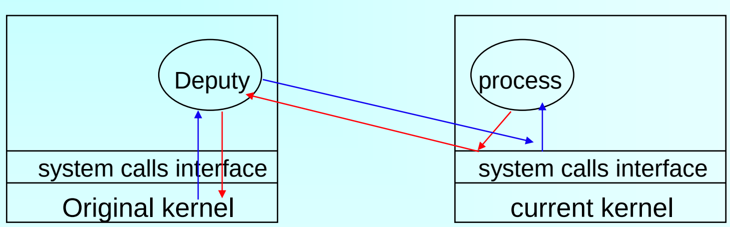


Architecture that fits one system image.
Needs location transparent file system.

(Mosix previous versions)

Architecture for Migration (cont.)

The process migrated from the original computer to the current computer and now is initiating a system call.



- Architecture that fits entrance dependent system.
- Easier to implement based on current Unix.

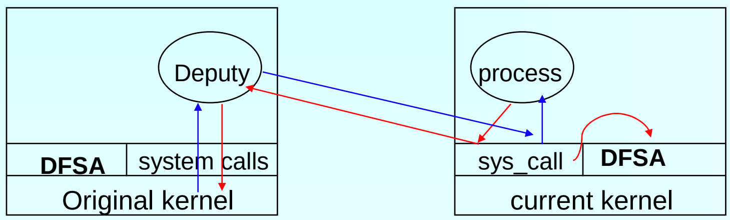
(Mosix current versions)

Mosix: File Access

Regularly, each file access must go back to deputy…

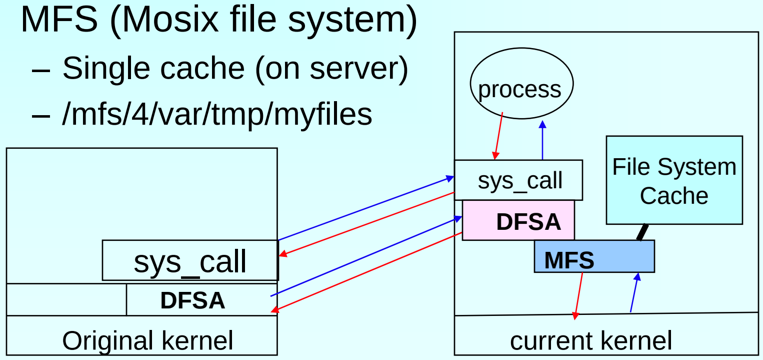
➔ Very Slow for I/O apps.

Solution-- Allow processes to access a distributed file system through the current kernel.

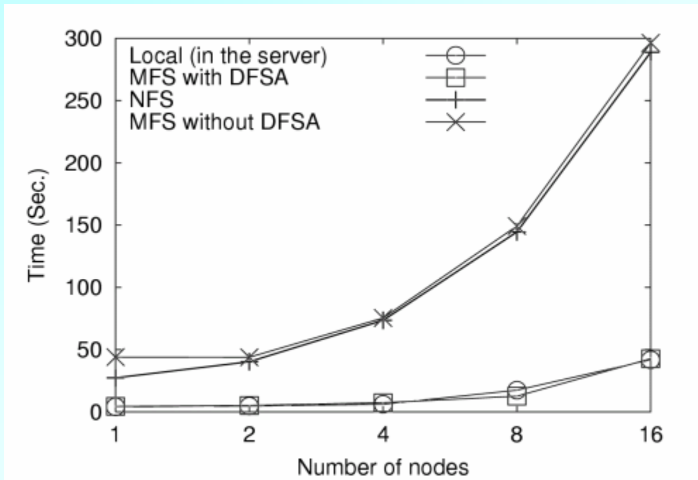


Mosix: File Access

- DFSA (direct file system access)
  - Requirements (cache coherent, monotonic timestamps, files not deleted until all nodes finished)
  - Bring the process to the files.
- MFS (Mosix file system)
  - Single cache (on server)
  - /mfs/4/var/tmp/myfiles



Mosix: DFSA/MFS Performance



Other Considerations for Migration

- Not only CPU load!!!
- Memory.
- I/O - where is the physical device?
- Communication - which processes communicate with which other processes ?

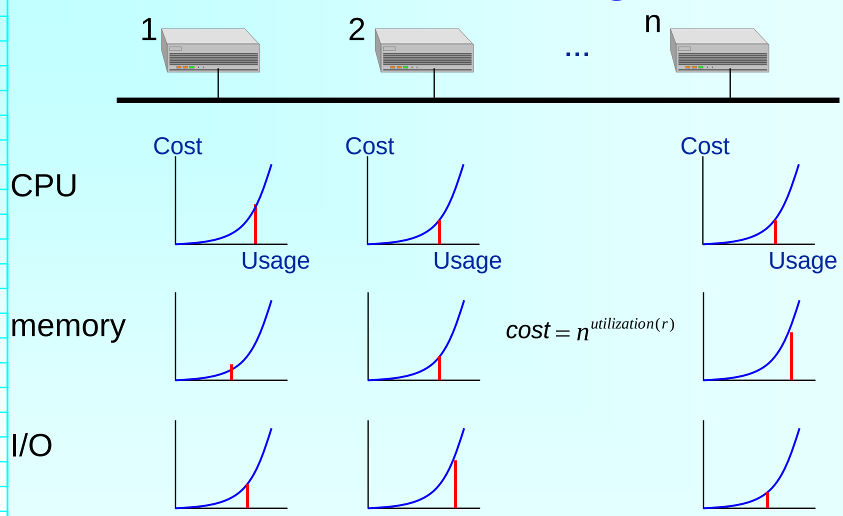
Work at the CNDS lab in this area

…

Resource Management of DOS

- A new *online* job assignment policy based on economic principles, competitive analysis.
- Guarantees near-optimal global lower-bound performance.
- Converts usage of *heterogeneous* resources (CPU, memory, IO) into a single, homogeneous cost using a **specific** cost function.
- Assigns/migrates a job to the machine on which it incurs the lowest cost.

DOS Resource Management



$$Machine\_cost = \sum n^{utilization(r_i)}$$

What did we do?

We looked at two metacomputing systems: PVM and Mosix.

- PVM - Parallel Virtual Machine
  - Static Assignment of jobs to machines.
  - Default - round robin assignment policy.
  - Widely used.
- Mosix
  - Dynamic job migration.
  - Main objective is load balancing, with some ad-hoc heuristics for memory depletion.

Enhanced PVM and Enhanced Mosix

- Enhanced PVM is similar to PVM. The decision where to statically place a new job is made according to the cost benefit framework.
- Enhanced Mosix is similar to Mosix. Job migration decisions are made according to the cost-benefit framework.

This framework currently takes into account:

- CPU load
- Memory utilization

We know how to add I/O and this is in the works.
We have no clue how to add IPC for the general case.

Evaluation Methodology

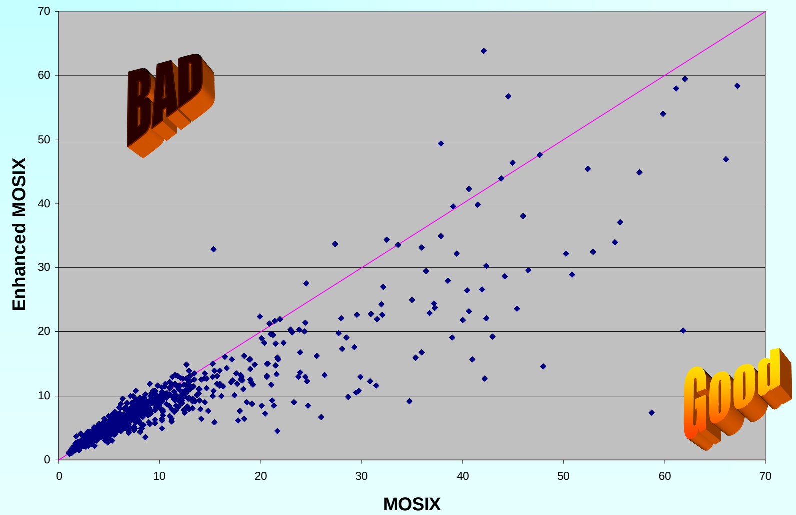
- Cluster:

Machine Type	# of these Machines	Processing Speed	Installed Memory
Pentium Pro	3	200 MHz.	64 MB of RAM
Pentium	2	133 MHz.	32 MB of RAM
Laptop w/ Ethernet	1	90 MHz.	24 MB of RAM.

- Simulation:
  - 3,000 identical executions (scenarios) per policy, each represents 10,000 - 20,000 sec.
- Validation (real life executions):
  - 50 identical executions per policy (similar to the simulations).

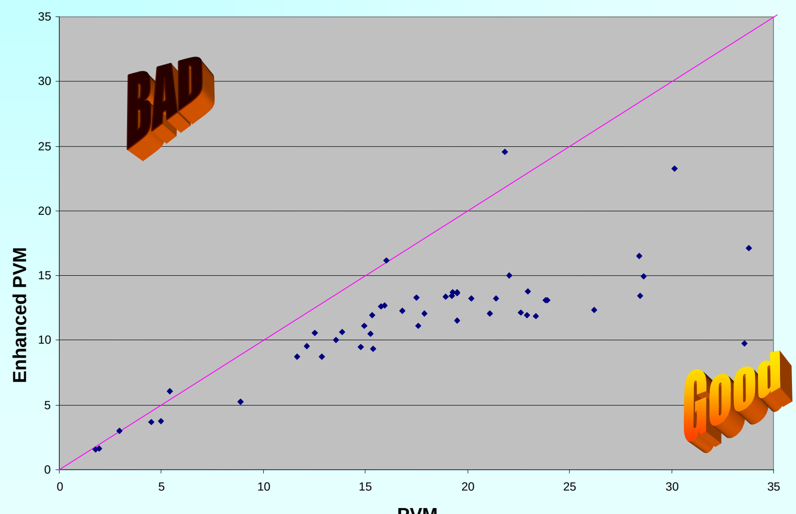
Simulation: Mosix / EMosix

Average Slowdown (lower is better)



Real Life: PVM / EPVM

Average Slowdown (lower is better)



Interesting: Mosix / EPVM

Average Slowdown (lower is better)