

Distributed Systems

27. Process Migration & Allocation

Paul Krzyzanowski

pxk@cs.rutgers.edu

Processor allocation

- Easy with multiprocessor systems
 - Every processor has access to the same memory and resources.
 - All processors pick a job from a common run queue.
 - Process can be restarted on any available processor.

- Much more complex with multicomputer systems
 - No shared memory (usually)
 - Little or no shared state
(file name space, open files, signals, ...)
 - Network latency

Allocation or migration?

- Migratory or nonmigratory processes?
- Most environments are nonmigratory:
 - System decides where a process is born
 - User decides where a process is born
- Migratory processes:
 - Move a process between machines during its lifetime
 - Can achieve better system-wide utilization of resources

Need transparency

- Process must see the same environment on different computers
 - Same set of system calls & shared libraries
- Non-migratory processes:
 - File system name space
 - *stdin, stdout, stderr*
- Migratory processes:
 - File system name space
 - Open file descriptors (including *stdin, stdout, stderr*)
 - Signals
 - Shared-memory segments
 - Network connections (e.g. TCP sockets)
 - Semaphores, message queues
 - Synchronized clocks

Migration strategies

- Move entire state to remote system

Columbia Zap

- Pod (PrOcess Domain)
 - Thin virtualization layer to decouple dependencies from OS
 - Collection of processes with a virtualized view of OS
 - Checkpoint/restart
 - Can be suspended to storage or moved to machines
 - Each pod contains a virtual namespace (e.g., PIDs)
- Suspend a Pod
 - Stop processes in pod
 - Save virtualization mappings
 - Save all process state (memory, cpu registers, open file handles)
- Relies on:
 - Network file servers for storing apps & data
 - Dynamic DNS servers for network address virtualization

Migration strategies

- Move entire state to remote system
- **Keep state on original system**
 - Use RPC for system calls

Berkeley Sprite OS

- Kernel-level RPC support
- Remember the “home” system for a process
- If the process migrates:
 - If a system call requires access to kernel state (file operations, signals, etc.) an RPC is made to the home system

Migration strategies

- Move entire state to remote system
- Keep state on original system
 - Use RPC for system calls
- Move a virtual machine/environment

Xen Live Migration

- VMM provides most of transparency
- Focus on minimizing downtime during migration
 - “Pre-copy” approach: pages of memory are copied to the destination host without stopping the execution of the VM
 - Pages that are modified after the copy get copied again
- Assume environment is a cluster in a data center – not a wire-area network

Migration strategies

- Move entire state to remote system
- Keep state on original system
 - Use RPC for system calls
- Move a virtual machine/environment
- **Ignore state**

Constructing process migration algorithms

- Deterministic vs. heuristic
- centralized, hierarchical or distributed
- optimal vs. suboptimal
- local or global information
- location policy

Up-down algorithm

- Goal: provide a fair share of available compute power
 - do not allow the user to monopolize the environment
- Centralized coordinator maintains usage table
- System creates process
 - decides if local system is too congested for local execution
 - sends request to central manager, asking for a process
- Centralized coordinator keeps points per workstation
 - +points for running jobs on other machines
 - -points if you have unsatisfied requests pending
 - If your points > 0 : you are a net user of processing resources
- Coordinator takes request from workstation with lowest score

Hierarchical algorithm

- Removes central coordinator to provide greater scalability
- Each group of “workers” (processors) gets a “manager” (coordinator responsible for process allocation to its workers)
- Manager keeps track of workers available for work
- If a manager does not have enough workers (CPU cycles), it then passes the request to its manager (up the hierarchy)

Distributed algorithms

- **Sender initiated distributed heuristic**

- If a system needs help in running jobs:
 - pick machine at random
 - send it a message: *Can you run my job?*
 - if it cannot, repeat (give up after n tries)
- Algorithm has been shown to behave well and be stable
- Problem: network load increases as system load increases

- **Receiver initiated distributed heuristic**

- If a system is not loaded:
 - pick machine at random
 - send it a message: *I have free cycles*
 - if it cannot, repeat (sleep for a while after n tries and try again)
- Heavy network load with idle systems but no extra load during critical (loaded) times

Migrating a Virtual Machine

- Checkpoint an **entire operating system**
- Restart it on another system

- Does the checkpointed image contain a filesystem?
 - Easy if all file access is network or to a migrated file system
 - Painful if file access goes through the host OS to the host file system.

- Migrate machine address
 - Most likely lose TCP state.

The End