Code and Process Migration

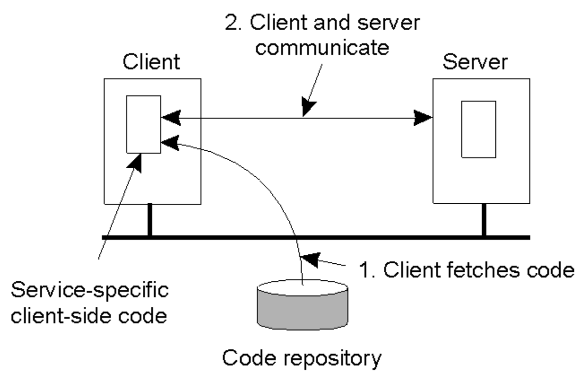
- Motivation
- How does migration occur?
- Resource migration
- Agent-based system
- Details of process migration

Motivation

- Key reasons: performance and flexibility
- Process migration (aka *strong mobility*)
  – Improved system-wide performance – better utilization of system-wide resources
  – Examples: Condor, DQS
- Code migration (aka *weak mobility)*
  – Shipment of server code to client – filling forms (reduce communication, no need to pre-link stubs with client)
  – Ship parts of client application to server instead of data from server to client (e.g., databases)
  – Improve parallelism – agent-based web searches

Motivation

- Flexibility
  - Dynamic configuration of distributed system
  - Clients don't need preinstalled software – download on demand



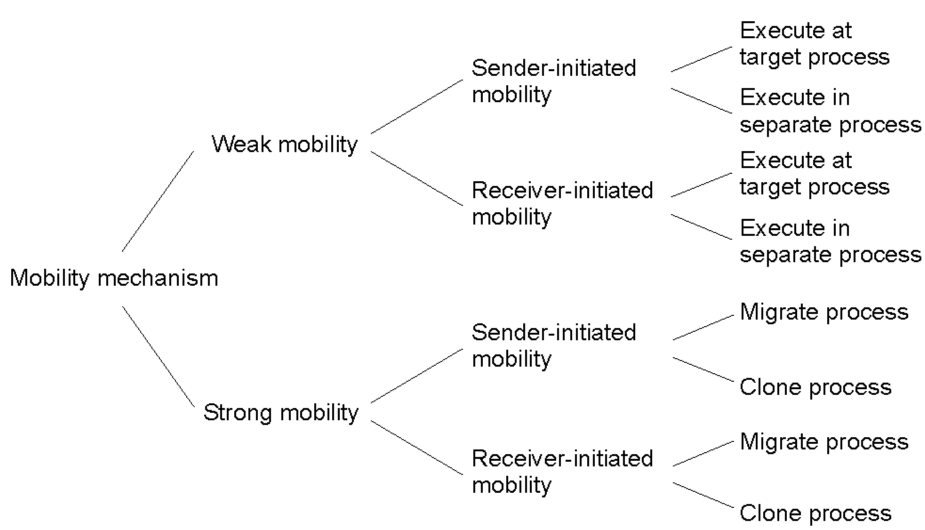
Migration models

- Process = code seg + resource seg + execution seg
- Weak versus strong mobility
  - Weak => transferred program starts from initial state
- Sender-initiated versus receiver-initiated
  - Sender-initiated (code is with sender)
    - Client sending a query to database server
    - Client should be pre-registered
  - Receiver-initiated
    - Java applets
    - Receiver can be anonymous

Who executes migrated entity?

- Code migration:
  - Execute in a separate process
  - [Applets] Execute in target process
- Process migration
  - Remote cloning
  - Migrate the process

Models for Code Migration



Do Resources Migrate?

- Depends on resource to process binding
  - By identifier: specific web site, ftp server
  - By value: Java libraries
  - By type: printers, local devices
- Depends on type of "attachments"
  - Unattached to any node: data files
  - Fastened resources (can be moved only at high cost)
    - Database, web sites
  - Fixed resources
    - Local devices, communication end points

Resource Migration Actions

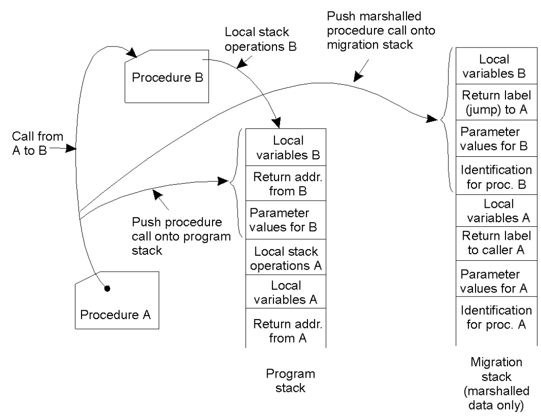
Resource-to machine binding

Process-to-resource binding		Unattached	Fastened	Fixed
	By identifier	MV (or GR)	GR (or MV)	GR
	By value	CP (or MV, GR)	GR (or CP)	GR
	By type	RB (or GR, CP)	RB (or GR, CP)	RB (or GR)

- Actions to be taken with respect to the references to local resources when migrating code to another machine.
- GR: establish global system-wide reference
- MV: move the resources
- CP: copy the resource
- RB: rebind process to locally available resource

Migration in Heterogeneous Systems

- Systems can be heterogeneous (different architecture, OS)
  - Support only weak mobility: recompile code, no run time information
  - Strong mobility: recompile code segment, transfer execution segment [migration stack]
  - Virtual machines - interpret source (scripts) or intermediate code [Java]



Machine Migration

- Rather than migrating code or process, migrate an "entire machine" (OS + all processes)
  - Feasible if virtual machines are used
  - Entire VM is migrated
    - Can handle small differences in architecture (Intel-AMD)
- Live VM Migration: migrate while executing
  - Assume shared disk (no need to migrate disk state)
  - Iteratively copy memory pages (memory state)
    - Subsequent rounds: send only pages dirtied in prior round
    - Final round: Pause and switch to new machine

Case Study: BOINC

- Internet scale operating system
  - Harness compute cycles of thousands of PCs on the Internet
  - PCs owned by different individuals
  - Donate CPU cycles/storage when not in use (pool resouces)
  - Contact coordinator for work
  - Coodinator: partition large parallel app into small tasks
  - Assign compute/storage tasks to PCs
- Examples: Seti@home, P2P backups

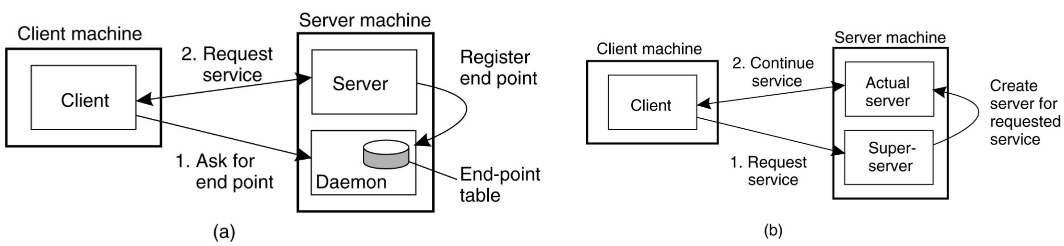
Case study: Condor

- Condor: use idle cycles on workstations in a LAN
- Used to run large batch jobs, long simulations
- Idle machines contact condor for work
- Condor assigns a waiting job
- User returns to workstation => suspend job, migrate
- Flexible job scheduling policies

Case Study: Amazon EC2

- Cloud computing platform
  - Users rent servers by the hour
  - Can also rent storage
  - Uses virtual machines
- New user request for a EC2 server
  - Central coordinator allocates physical server
  - Create a new VM, copy user-specified image to machine
    - User gets root-level access to the machine (via ssh)
  - Can allocate new serveror terminate as needed
- Distributed scheduling on a cluster of servers for rent

Server Design Issues

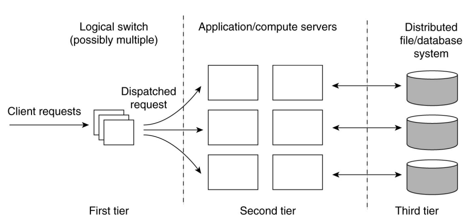


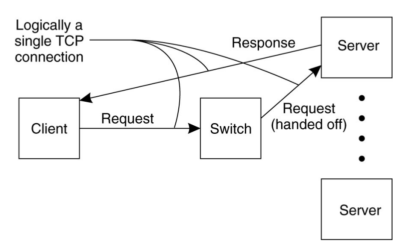
- Server Design
  - Iterative versus concurrent
- How to locate an end-point (port #)?
  - Well known port #
  - Directory service (port mapper in Unix)
  - Super server (inetd in Unix)

Stateful or Stateless?

- Stateful server
  - Maintain state of connected clients
  - Sessions in web servers
- Stateless server
  - No state for clients
- Soft state
  - Maintain state for a limited time; discarding state does not impact correctness

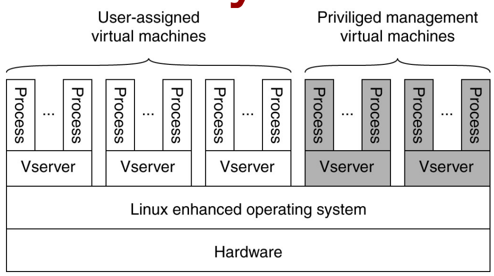
Server Clusters





- Web applications use tiered architecture
  - Each tier may be optionally replicated; uses a dispatcher
  - Use TCP splicing or handoffs

Case Study: PlanetLab



- Distributed cluster across universities
  – Used for experimental research by students and faculty in networking and distributed systems
- Uses a virtualized architecture
  – Linux Vservers
  – Node manager per machine
  – Obtain a “slice” for an experiment: slice creation service

Server Architecture

- Sequential
  – Serve one request at a time
  – Can service multiple requests by employing events and asynchronous communication
- Concurrent
  – Server spawns a process or thread to service each request
  – Can also use a pre-spawned pool of threads/processes (apache)
- Thus servers could be
  – Pure-sequential, event-based, thread-based, process-based
- Discussion: which architecture is most efficient?

Scalability

- *Question:* How can you scale the server capacity?
- Buy bigger machine!
- Replicate
- Distribute data and/or algorithms
- Ship code instead of data
- Cache