Today

- Distributed Computing
- Distributed Scheduling
  - Multiprocessors
  - Clusters
  - Wide-area systems

Distributed Computing

- Processes executing on distributed computing resources
  - Typically group of processes executing together to accomplish common task
- Main issues:
  - How to manage distributed computing resources efficiently (e.g.: high utilization, throughput, etc.)?
  - What if processes are related, part of a single application?
  - How to move processes around?

Distributed Computing Environments

- Multiprocessor systems:
  - Tightly-coupled, may have shared memory
  - Small SMPs vs. large MPPs (Parallel computing)
- Cluster computing:
  - Independent machines connected over LAN
  - Typically homogeneous
- Wide-area computing:
  - Sets of machines connected over WAN
  - Heterogeneous, large latencies
  - E.g.: Grids, BOINC
**Distributed Scheduling**
- Question: Given a set of CPUs (nodes) and a set of processes, where to execute these processes?
- Depends on computing platform
  - Multiprocessors
  - Clusters
  - Wide-area systems

**Multiprocessor Scheduling**
- Uniprocessor: Which process to run next?
- Multiprocessor: Which process to run next *on which CPU?
- First consider symmetric multiprocessors (SMP)
  - Small # homogeneous CPUs, shared memory
- Decision can depend on:
  - Cache affinity of a process
  - Load on CPUs

**Global Scheduler**
- Single run queue
  - Idle CPU picks the next process from the queue
  - Could use priority, FCFS, SJF, etc.
- Problem?
  - CPUs need to coordinate to access run queue
  - Large synchronization overhead if many CPUs

**Affinity Scheduling**
- Processes have cache affinity with recent CPUs
  - Would like to reschedule on same CPU
- Types of affinity:
  - Soft: Likely to run on same CPU but not guaranteed
  - Hard: Guaranteed to run on a set of CPUs (Pinning)
- What could be the impact on overall system utilization?
Load Balancing

- Two-level scheduling
  - Each CPU performs scheduling on private queue
  - Balance load occasionally
- When to load balance?
  - Push migration: Periodically balance load
  - Pull migration: Idle CPU pulls load from other CPUs
- Example: Linux scheduler

Parallel Computing

- Massively Parallel Processors (MPPs)
  - 100s-1000s of CPUs connected via fast interconnect bus
  - May have NUMA architecture, distributed memory
  - E.g.: supercomputers
- Parallel jobs:
  - Consist of large number of processes
  - Processes typically work in sync, may communicate
  - Processes submitted in batches, scheduled on the system

Space Sharing

- Used for groups of processes
  - Need to run together
- Partition CPUs into sets
  - Each set runs one process group
  - Non-preemptible until all processes finish
- Job scheduling:
  - FIFO: Jobs run in arrival order
  - Priority: High-priority jobs assigned CPUs first
- What about utilization?

Backfilling

- Allow low priority jobs to get ahead if:
  - They can fit
  - Do not impact the start time of high priority jobs
Time Sharing
- Allow each CPU to run multiple processes
- Use local scheduler on each CPU
  - E.g.: Round-robin
- Benefits?

Gang Scheduling
- Extends time-sharing
- Problem: Processes communicating heavily
  - Avoid delay in communication
  - Ensure processes from same group (gang) are scheduled together
  - Need synchronization of scheduling quanta

Cluster Computing
- Multi-computer system:
  - Fast network (LAN)
  - Homogeneous machines
- Issues:
  - No shared memory
  - Processes cannot move easily
  - Each node has a local scheduler
- Question: How to allocate processes across nodes?

Centralized Processor Allocation
- Given: Set of jobs with multiple communicating processes, k machines
- Graph Theoretic Model:
  - Process = vertex, communication = edge
  - Communication amount = edge weight
- Goal: Minimize the amount of inter-machine communication
- Approach: Minimal k-partition that meets individual constraints
Distributed Processor Allocation

- Processes arrive continuously
  - Started on one of the machines
- Approach 1: Keep each process on original machine, use local scheduler
  - Problem?
- Load imbalance: Probability that at least one processor is idle while a job is waiting
  - How does this relate to avg. load in the system?

Load Balancing

- Allocate/move processes to balance load across machines
- Design issues:
  - Centralized vs. distributed?
  - How to measure load?
  - Pre-emptive vs. non-preemptive?
  - Who initiates the allocation/migration?

Centralized Load Balancing

- Central monitor:
  - Measure loads
  - Assigns processes to nodes
- Advantages?
- Disadvantages?

Distributed Load Balancing

- Each node makes its own decisions
- Transfer policy: When to transfer a process?
  - E.g.: Threshold-based
- Selection policy: Which process to transfer?
  - New, low migration cost, long execution time
- Location policy: Where to send the process?
  - Random, nearest, least-loaded
- Information policy: Who sends the information, to whom, and when?
  - On-demand, periodic, state-change-driven
Initiating Process Migration
- Sender-initiated: Overloaded node initiates process migration
- Receiver-initiated: Underloaded node looks for work
- Hybrid: Each node can act as both sender and receiver
  - Maintain two thresholds (high and low load)

Bidding-based Process Allocation
- Economic model
  - Nodes advertise resources with prices
  - Processes bid on resources
  - Highest bidders win resources
- Questions:
  - Who organizes bids and advertisements?
  - How to determine prices?

Matchmaking
- Resource providers, resource consumers
  - Providers specify their resource constraints. E.g.: OS, CPU speed, memory, etc.
  - Consumers specify their resource requirements. E.g.: need Windows XP, 1GHz CPU, 512M RAM, etc.
- Specifications provided in a specification language (e.g.: XML)
- Matchmaker matches requirements to constraints
  - E.g.: Condor

Wide-Area Distributed Scheduling
- Features:
  - Nodes may be heterogeneous
  - Longer latencies
  - Node failures
  - No central control
- Questions:
  - Resource discovery: How to locate resources?
  - Process allocation: How to allocate processes?
  - Incentive: Why do decentralized nodes participate?
Grid Scheduling

- Grids
  - Interconnected, geographically distributed compute clusters and supercomputers
- Resource Discovery:
  - Some kind of matchmaking for initial allocation
- Process Allocation:
  - May use load-balancing algorithms
- Incentive:
  - Centrally managed
  - Cooperating administrators

Donation-based Scheduling

- Nodes donate resources
  - Example: BOINC, SETI@home
  - Loosely coupled, decentralized system
- Resource discovery
  - Nodes register with a central server
- Process allocation
  - Pull-based: Node asks for task, is assigned a task
  - Simple heuristics
- Incentive: Credit for performing computation

Cycle-Sharing

- Peer-to-Peer system of compute nodes
  - Nodes participate by donating CPU cycles
  - No central server
- Resource Discovery/Process Allocation
  - Random path
  - Expanding Ring
  - Advertisement-based
  - Rendezvous point
- What is the incentive here?