Issues in Transactions and Concurrency Control (Review)

- Centralized transactions
  - Concurrency control
    - Locking algorithms
      - Static locking
      - Two-phase locking (2PL)
      - Strict two-phase locking (strict 2PL)
    - Optimistic concurrency control
    - Timestamp ordering
  - Handling deadlock for locking algorithms
    - Deadlock detection
    - Deadlock prevention
      - Lock timeouts
      - Transaction timestamps
- Distributed transactions
  - Simple distributed vs. nested
  - Atomic commit protocols
    - One-phase
    - Two-phase

Distributed Transactions

- A *distributed transaction* invokes operations in several different servers
  - Simple distributed transaction
    - Client makes requests to more than one server
    - Each server carries out the client's requests without involvement by others
  - Nested distributed transaction
    - Client makes requests to more than one server
    - Some of those servers make requests of yet other servers to carry out the client's request, and some of those servers may…
    - Example:
      - Client A tells server M to transfer $4 from account A to C, and $3 from B to D
      - A is at server X, B is at server Y, and C and D are at server Z
      - M tells server X to withdraw $4 from A
      - M tells server Y to withdraw $3 from B
      - M tells server Z to deposit $4 into C, and $3 into D

Atomic Commit Protocols

- Distributed transactions are still required to be completed atomically
- First server involved in the distributed transaction becomes the coordinator
  - Coordinator is responsible for committing or aborting the transaction
  - All transactions involved know the identity of the coordinator
- One-phase atomic commit protocol
  - Client has requested that operations be performed at more than one server
  - Transaction ends when client requests that it be committed or aborted
  - Coordinator tells all the servers in the transaction to commit / abort, and keeps repeating that request until all of them acknowledge that they have carried it out

Atomic Commit Protocols (cont.)

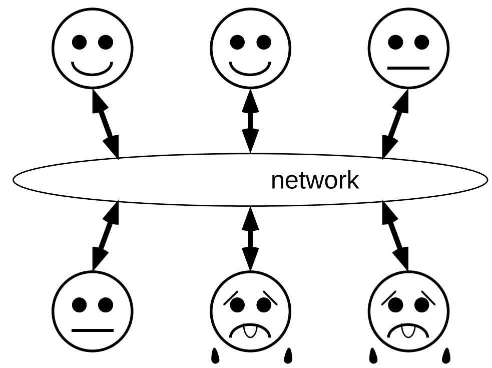
- Two-phase atomic commit protocol
  - Allows any server to abort its part of the transaction; atomicity then requires the entire transaction to be aborted
  - Phase 1: (voting phase)
    - Coordinator asks each worker if it can commit its transaction
    - Worker replies to coordinator; if its answer is *no*, the worker immediately aborts
  - Phase 2: (completion phase)
    - Coordinator collects the votes (including its own)
      - If there are no failures, and all votes are *yes*, the coordinator sends a *commit* request to each worker
      - Otherwise, the coordinator sends an *abort* request to all workers that voted *yes*
    - Workers that voted *yes* wait for a *commit* or *abort* message, act accordingly, and in the case of *commit* send a have_*committed* message afterwards

Distributed Scheduling

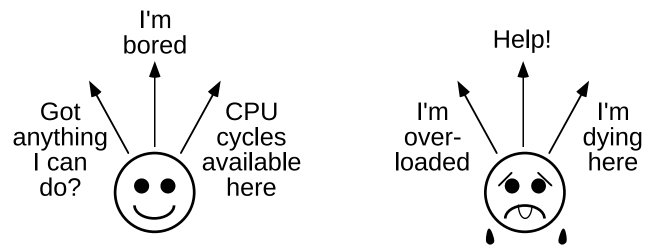
- Scheduling in a centralized system:
  - Resource = CPU
  - Consumer = process
  - Scheduling = assign each process to some period of time on the CPU
- Scheduling in a distributed system:
  - Resource = processor / workstation
  - Consumer = computation task
  - Scheduling = assign each computation task to some processor
- Goal: distribute tasks to the set of processors so as to optimize some cost function (e.g., response time, utilization)
  - *Load distribution* — deciding which tasks to move from one processor to another, and when to move them

Motivation for Load Distribution

- Have this situation:



- Want to allow this:



Example Load Distribution Algorithm

- All processors constantly monitor their load — the number of active processes
- When a processor's load goes above some particular threshold, it becomes a "sender"
- The new process that caused it to become a sender is selected for transfer
- The sender polls the other processes, one by one, until it finds a "receiver" — a process with a load below some particular threshold
- The selected process is frozen, transferred (migrated) from the sender to the receiver, and restarted there

Advantages of Load Distribution

- Reduce response time for processes
  - Move to lightly loaded node
- Speed up individual jobs
  - Go to faster node
  - Split up process across multiple nodes
- Gain higher throughput
  - Balance system load
  - Mix I/O & CPU bound processes
- Utilize resources effectively
  - Move to node where resources reside
- Reduce network traffic
  - Cluster related processes on same node

Desirable Features of a Good Load Distribution Method

- No *a priori* knowledge about processes
- Dynamic in nature — change with system load, allow process migration
- Quick decision-making capability
- Balanced system performance and overhead — don't reduce system performance collecting state information
- Stability — don't migrate processes so often that no work gets done (better definition later)
- Scalability — works on both small and large networks
- Fault tolerance — recover if one or more processors crashes

Load Distribution vs. Process Migration

- *Load distribution* — deciding which tasks to move from one processor to another, and when to move them
  - Selection of process to migrate
  - Selection of destination node
- *Process migration* is the relocation of a process from its current location (*source node*) to another node (*destination node*)
  - Preemptive — after process starts
  - Non-preemptive — before process starts
  - Mechanics of process migration:
    - Freeze process on source node, restart it on destination node
    - Transfer address space of process
    - Forward messages sent to old processor
    - Support communication with migrated processes after move to new processor

Desirable Features of a Good Process Migration Method

- Transparency
  - Access to all objects from everywhere
  - Location-independent system calls
- Minimal interference
  - Minimize freeze time (stopped execution while process is being transferred)
- Minimal residual dependencies
  - Migrated process should not depend in any way on source node
    - Adds to load on source node
    - Failure of source node could affect it
- Efficiency
  - Keep inefficiency to a minimum
    - Time to select process and destination
    - Time required to migrate a process
    - Cost of remote execution afterwards

Process Migration Mechanisms

- Freezing and restarting a process
  - Only an issue for preemptive transfers
  - Immediate blocking
    - If not executing a system call
    - If executing a sys call, but sleeping and interruptable
  - Delayed blocking
    - If executing a system call, but sleeping at a non-interruptable priority — must delay until system call is complete
  - Wait for completion of fast I/O operations, don't wait for completion of slow I/O
  - Keep track of files, switch to local files if possible
  - Keep same process ID after migration

Process Migration Mechanisms (cont.)

- Transferring the address space
  - Entire process state: registers, scheduling info, memory tables, I/O states, process ID, file info, etc.
    - Must stop execution during transfer
  - Address space: code, data, stack, heap
    - Transfer can take a long time!
    - Can continue execution during transfer
  - Total freeze
    - Stop execution during addr. space transfer
    - Possible long suspension in execution
  - Pre-transfer
    - Continue execution during address space transfer, then freeze process and transfer remaining modified pages
    - Small freeze time = little interruption
  - Transfer on reference
    - Leave address space on source node, only transfer pages when and if they are referenced

Process Migration Mechanisms (cont.)

- Message-forwarding
  - 3 types of messages to forward
    1. Messages received at source node after execution has stopped there, but before execution has started on destination
    2. Messages received at source node after execution has started on destination
    3. Messages sent to process later
  - Resending the message
    - Return or drop type 1 & 2 messages, hope sender will resend to new location
  - Origin site mechanism
    - Messages are sent to original source site, which forwards them as necessary
  - Link traversal mechanism
    - Type 1 messages are part of migration
    - Type 2 & 3 messages follow a link (forwarding address) left behing

Process Migration in Heterogeneous Systems

- Must translate data
  - Big endian, little endian (bytes & words)
  - ASCII, EBCDIC, etc.
  - External data representation
    - Use standard representation for transfer
  - Sinha describes various techniques for migrating the exponent and mantissa of floating point numbers
    - However, many systems now use the IEEE floating point format, for consistency
    - Single precision = 32 bits (1 sign, 8 exponent, 23 mantissa)
    - Double precision = 64 bits (1 sign, 10 exponent, 53 mantissa)
    - For details, see my Computer Organization lecture on the subject
  - Also have to handle signed-infinity and signed-zero, if those values are supported by one or both of the nodes