Operating Systems

Processor allocation in distributed systems

Lecturer:
William Fornaciari
Politecnico di Milano
fornacia@elet.polimi.it
www.elet.polimi.it/~fornacia

SUMMARY

- Processor Allocation
- Migration Strategies
- Co-scheduling

Introduction

- The system consists in a set of (fully connected) processors onto which a set of processes have to be allocated
- Possible allocation strategies
  - Non-migratory
    - P remains on the same machine until it terminates
  - Migratory
    - Move (migrate) an active (already started) process from one machine to another (target) machine
    - It requires transferring a sufficient amount of the state of a process from one machine to another
    - Better load balancing, more complex, significant impact on systems design

Overall Design issues

- Long term goals
  - Maximization of CPU utilization
  - Minimization of mean response time
  - Minimization of response ratio, i.e. the amount of time it takes to run a process on a machine divided on the value obtained executing it on an unloaded benchmark processor
- Load sharing
  - move processes from heavily loaded to lightly loaded systems
  - load can be balanced to improve overall performance

Overall Design issues

- Communications performance
  - processes that interact intensively can be moved to the same node to reduce communications cost
  - may be better to move process to where the data reside when the data is large
- Availability
  - long-running process may need to move because the machine it is running on, will be down
- Utilizing special capabilities
  - process can take advantage of unique hardware or software capabilities which are in a particular system

Allocation algorithms: taxonomy

- Deterministic vs heuristic alg.
  - Difficult prediction of process behavior and requirements
- Centralized vs distributed alg.
  - Centralized is less robust, better tuned, with system bottleneck
- Optimal vs sub-optimal alg.
  - Due to cost, typ Heuristic, distributed, sub-optimal
- Local vs global alg. (transfer policy)
  - Which information are used to decide
- Sender-initiate vs receiver initiated alg. (location)
  - Where to send the process, it cannot be local

Implementation problems

- Determination of the CPU load of all the machines
- Overhead of the algorithm
- Implementation complexity
  - Maybe…simple is better
- Stability
  - Different algorithms runs simultaneously from one other, with not up-to-date and possibly consistent information

PROCESS MIGRATION: BASIC IDEA

- Transfer of sufficient amount of the state of a process from one machine to another
- The process executes on the target machine
- Must destroy the process on the source system and create it on the target system
- Process control block and any links must be moved

PROCESS MIGRATION: MOTIVATION

- Load sharing
  - Move processes from heavily loaded to lightly load systems
  - Load can be balanced to improve overall performance
- Communications performance
  - Processes that interact intensively can be moved to the same node to reduce communications cost
  - May be better to move process to where the data reside when the data is large
- Availability
  - Long-running process may need to move because the machine it is running on will be down
- Utilizing special capabilities
  - Process can take advantage of unique hardware or software capabilities

WHO BEGINS THE MIGRATION?

- Operating system
  - when goal is load balancing
- Process
  - when goal is to reach a particular resource

Classes of algorithms

- Deterministic Graph-based
- Centralized
- Hierarchical
- Distributed Heuristic
- Bidding

Deterministic Graph-based

- Assumptions
  - Knowledge of CPU and memory requirements for each process and of a matrix with the average traffic for each processor pair (**difficult)**
- Model: The system is a weigthed graph
  - Nodes = processes, arcs msg flow
- Problem
  - Assignment of multiple proc. to the CPUs so that net traffic is minimized, while fulflilling processes constr.
  - Find a graph partition in disjoined sub-graphs subject to contraints, i.e. discovering “clusters”

Centralized (UP-DOWN)

- Does not requires *a priori* knowledge, is heuristic
- A coordinator maintains a global **usage table (**of Workstations), that is updated sending msg
- The allocation decisions are based on the infos of the usage table, on-the-fly, not only based on workload balancing
- When a process is created, the owner ws can move it on other machine cumulating *penality points* that are summed to the owner table entry (UP)
- If no pending requests, table values decrese (DOWN)

Centralized (UP-DOWN)

- Entry values
  - + : the corresp. workstation is a big user of resorces
  - - : the ws needs resources
  - 0 : neutral
- Heuristic
  - When a processor becomes free, the pending request whose owner has the lowest score wins
- Consequence: fairly capacity allocation
  - a user occupying no processor, rising a request peding for a long time, always beat someone using many processors
- Good in many cases, but do not scale well to large systems

Hierarchical

- Try to remove the bottlenek of centralized
- The processors (workers and managers) are hierarchically organized, indipendently of the physical organization of network
  - For each group of workers, a manager traks who is busy or idle and distributes the works
  - Possibility of several levels of managers, but only one reports to the upper level: reduced communication
  - To prevent vulnerability, the hierarchy should not be a pyramid, the top level contains >1 managers

Hierarchical

- If a manager receives a request and is in shortage of resources, it pass the request upwards to the tree
- The number of processor R associated to a manager must be properly chosen
  - R large: too many processors allocated, waste of computational capacity
  - R small: overhead due to upwards passing of requests
- Potential problems
  - Random multiple requests happens at various stages of the algorithms, giving out-of-date estimates of workers capability, race conditions, deadlocks

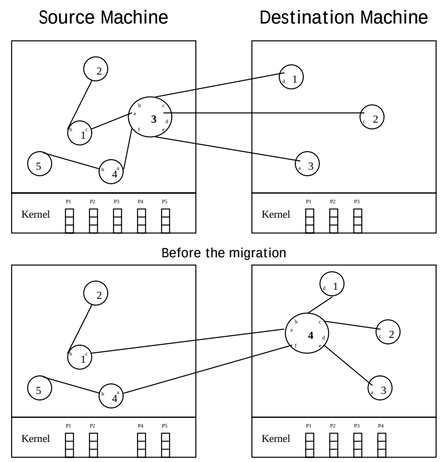
Distributed Heuristic

- Studied mainly by Eager (see next)
- A process is created, and the machine on which it is created send a probe msg to a randomly chosen machine, asking if its load is under a given threshold
  - If yes: the process is sent there
  - If no: another machine is chosen for probe
- It exists a upper bound N to the probes, then the process will run on the originating machine

Bidding

- Computer system actors
  - Buyers, Sellers of services, Prices set by supply/demand
  - The price of a processor depends on memory size, hardware, espected response time, ...
- Processes must buy CPU time, Processors provides their services to the highest bidder
- A process starts a child
  - Determine the set of processors offering the needed services and chose the cheapest, fastest, or best price/performance
  - Prices/offers are periodically updated

AN EXAMPLE OF MIGRATION (1)



After the migration

AN EXAMPLE OF MIGRATION (2)

- The process 3 migrates from the source machine to the destination machine
- All the IDs of the links (a,b,c,d...) of the processes don't change
- The operating system has to move the process control block and it has to modify the mapping of the links
- The process migration has to be invisible either to the migrated process and its partners
  - Problems: addressing space and open files

MIGRATION STRATEGIES (1)

- Eager (all): Transfer entire address space
  - no trace of process is left behind
  - if address space is large and if the process does not need most of it, then this approach may be unnecessarily expensive
  - the initial cost of the migration can be some minutes
- Precopy: Process continues to execute on the source node while the address space is copied
  - pages modified on the source during precopy operation have to be copied a second time
  - reduces the time that a process is frozen and cannot execute during migration

MIGRATION STRATEGIES (2)

- Eager (dirty): Transfer only that portion of the address space that is in main memory
  - any additional blocks of the virtual address space are transferred on demand
  - the source machine is involved throughout the life of the process (e.g., remote memory paging)
  - good if process is temporarily going to another machine
- Copy-on-reference: Pages are only brought over on reference
  - variation of eager (dirty)
  - has lowest initial cost of process migration

MIGRATION STRATEGIES (3)

- Flushing: Pages are cleared from main memory by flushing dirty pages to disk
  - later use copy-on-reference strategy
  - relieves the source of holding any pages of the migrated process in main memory, making it available for other processes

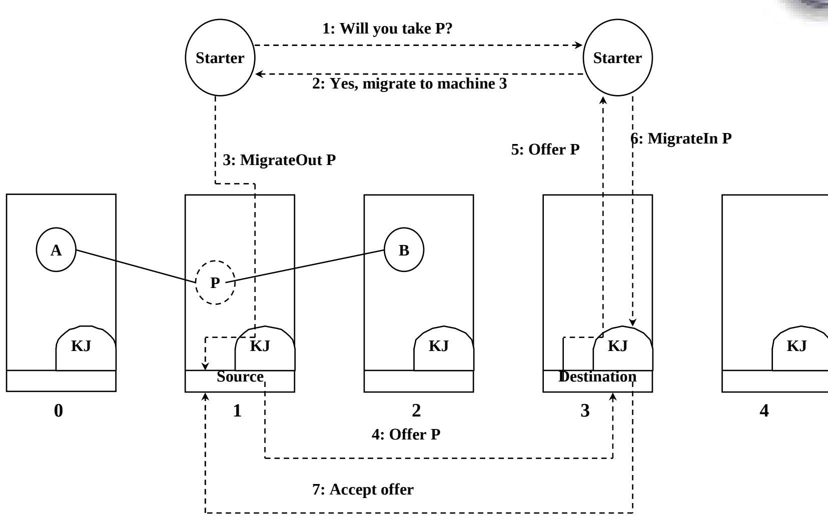
CHOICE OF A STRATEGY

- If a process uses a little address space of the source machine memory (e.g, it moves temporarily on another machine and then return back) we will choose
  - Eager (dirty)
  - Copy on reference
  - Flushing
- Otherwise, when sooner or later the entire address space will be referenced, we will choose
  - Eager (all)
  - Precopy

NEGOTIATION OF PROCESS MIGRATION

- Migration policy is responsibility of Starter utility
- Each machine has a Starter Utility which controls the process migration
- Starter utility is also responsible for long-term scheduling and memory allocation
- Decision to migrate must be reached jointly by two Starter processes (one on the source and one on the destination)

NEGOTIATION OF PROCESS MIGRATION



NEGOTIATION OF PROCESS MIGRATION

1 The Starter Utility of the source system wants the process P to migrate to a destination system. So it sends a message to the Starter Utility of the destination system
2 The destination Starter Utility answers a positive acknowledgment, if it ready
3 The source Starter Utility sends a message of migration to its kernel job (KJ), which converts msg among remote processes into system calls
4 The source kernel offers to send the process P to the destination (the offer includes statisticson P like age and communication overhead)

NEGOTIATION OF PROCESS MIGRATION

5 The destination kernel moves the offer to its Starter Utility. If it is in shortage of resources, it can reject
6 The Starter Utility uses a MIGRATE_IN call
7 The destination accepts the offer, allocating resources so to avoind stalls or other problems

- Source machine (1) must send messagges toward those containing other processes communicating with P, to update links

Eviction

- System evict a process that has been migrated to it
- If a workstation is idle, process may have been migrated to it
  - Once the workstation is active, it may be necessary to evict the migrated processes to provide adequate response time

EXAMPLE: AIX (IBM Unix) - Actions

- When a process decide to migrate (automigration)
  - it select the target machine
  - send a msg of remote tasking (part of the process image and data concerning open files)
- The server kernel of the receiver machine fork off a child with the received information
- The new process recalls data, environment, arguments and information on the stack, necessary to complete the migration
- The original process if informed of the succeded migration, it sends a ACK to the new process and suicide

Co-scheduling

- In many cases processes operate in “group”
- Scheduling can consider interprocess communication to ensure that all the member of a group run at the same time
- The algorithm uses a *conceptual* matrix
  - Column: process table for one processor
  - Row: set of processes that in a given time slot are running
- Goal: each processor uses a RR scheduling with all processes in slot o, then RR for those in slot 1, …
- A broadcast message can be used to tell each processor when perfoming switching, to synchronize timeslicing

Co-scheduling

- Putting members of a process group in the same slot takes advantage of N-fold parallelism, to maximize communication throughput
- This scheduling techniques can be combined with hierarchical model of process managements, in some systems
- Other variation breaks the matrix into rows and concatenates consecutive slots belonging to different processors

