Unit: 4 Processes and Threads in Distributed Systems

Thread

- A program has one or more locus of execution. Each execution is called a thread of execution.
- In traditional operating systems, each process has an address space and a single thread of execution.
- It is the smallest unit of processing that can be scheduled by an operating system.
- A thread is a single sequence stream within in a process. Because threads have some of the properties of processes, they are sometimes called lightweight processes. In a process, threads allow multiple executions of streams.

Thread Structure

- Process is used to group resources together and threads are the entities scheduled for execution on the CPU.
- The thread has a program counter that keeps track of which instruction to execute next.
- It has registers, which holds its current working variables.
- It has a stack, which contains the execution history, with one frame for each procedure called but not yet returned from.
- Although a thread must execute in some process, the thread and its process are different concepts and can be treated separately.
- What threads add to the process model is to allow multiple executions to take place in the same process environment, to a large degree independent of one another.
- Having multiple threads running in parallel in one process is similar to having multiple processes running in parallel in one computer.

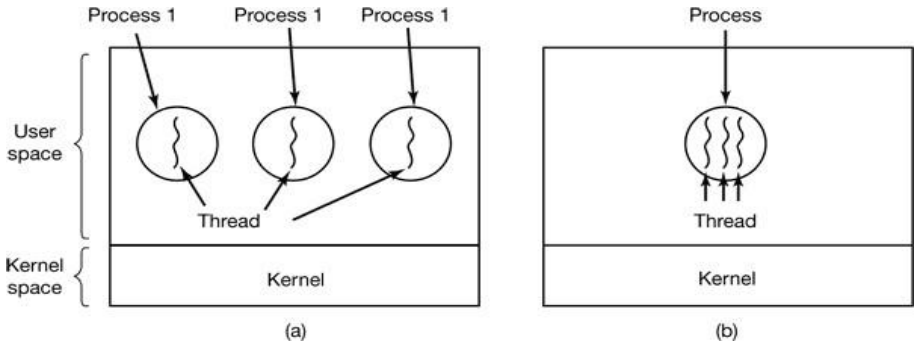


Figure: (a) Three processes each with one thread. (b) One process with three threads.

- In former case, the threads share an address space, open files, and other resources.
- In the latter case, process share physical memory, disks, printers and other resources.
- In Fig. (a), we see three traditional processes. Each process has its own address space and a single thread of control.
- In contrast, in Fig. (b), we see a single process with three threads of control.
- Although in both cases we have three threads, in Fig. (a) each of them operates in a different address space, whereas in Fig.(b) all three of them share the same address space.
- Like a traditional process (i.e., a process with only one thread), a thread can be in any one of several states: running, blocked, ready, or terminated.
- When multithreading is present, processes normally start with a single thread present. This thread has the ability to create new threads by calling a library procedure **thread_create.**

- When a thread has finished its work, it can exit by calling a library procedure **thread_exit.**
- One thread can wait for a (specific) thread to exit by calling a procedure **thread_join.** This procedure blocks the calling thread until a (specific) thread has exited.
- Another common thread call is **thread_yield**, which allows a thread to voluntarily give up the CPU to let another thread run.

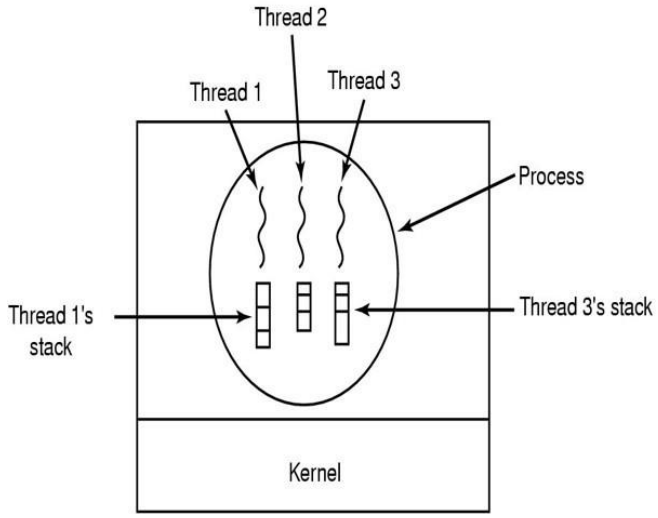


Figure: Each thread has its own stack.

Similarities and Dissimilarities between Process and Thread

Similarities between Process and Thread

- Like processes threads share CPU and only one thread is running at a time.
- Like processes threads within a process execute sequentially.
- Like processes thread can create children.
- Like a traditional process, a thread can be in any one of several states: running, blocked, ready or terminated.
- Like process threads have Program Counter, Stack, Registers and State.

Dissimilarities between Process and Thread

- Unlike processes threads are not independent of one another.
- Threads within the same process share an address space.
- Unlike processes all threads can access every address in the task.
- Unlike processes threads are design to assist one other. Note that processes might or might not assist one another because processes may be originated from different users.

Difference between User level threads and Kernel level threads

USER LEVEL THREAD	KERNEL LEVEL THREAD
User thread are implemented by users.	Kernel threads are implemented by OS.
OS doesn't recognized user level threads.	Kernel threads are recognized by OS.
Implementation of User threads is easy.	Implementation of Kernel thread is complex.
Context switch time is less.	Context switch time is more.
Context switch requires no hardware support.	Context switch requires hardware support.
If one user level thread performs blocking operation, then entire process will be blocked.	If one kernel thread performs blocking operation, then another thread within the same process can continue execution.

Example: Java thread, POSIX threads.	Example: Windows, Solaris

Thread models OR Types of Thread Usage

Dispatcher Worker model

- In this model, the process consists of a single dispatcher thread and multiple worker threads.
- The dispatcher thread accepts requests from clients and, after examining the request, dispatches the request to one of the free worker threads for further processing of the request.
- Each worker thread works on a different client request.
- Therefore, multiple client requests can be processed in parallel.

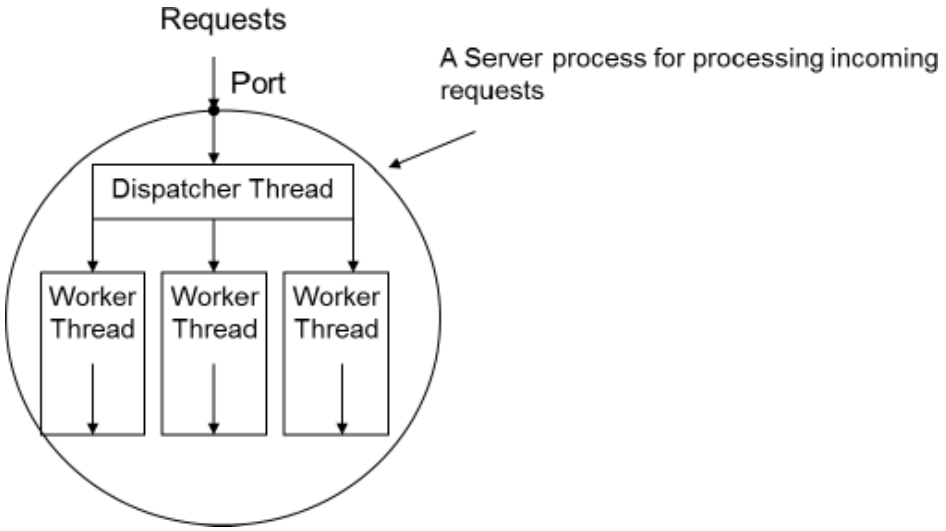


Figure: Dispatcher worker model

Team Model

- In this model, all threads behave as equal.
- Each thread gets and processes client's requests on its own.
- This model is often used for implementing specialized threads within a process.
  - Each thread of the process is specialized in servicing a specific type of request.

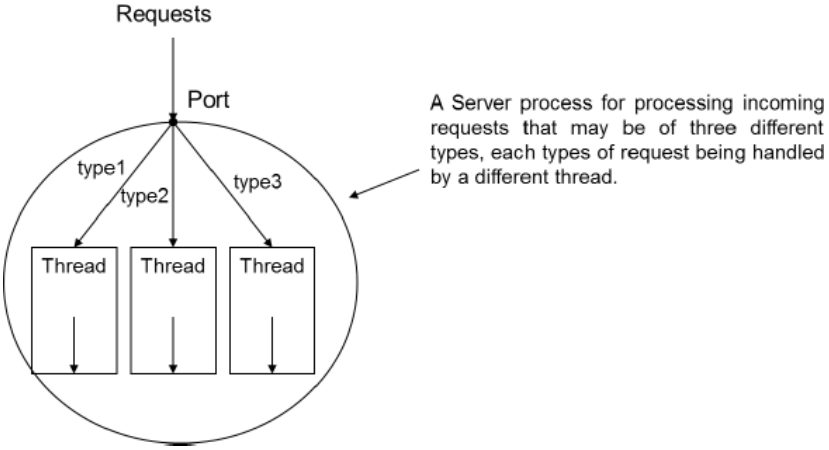


Figure: Team model

Pipeline Model

- This model is useful for applications based on the producer-consumer model.

- The output data generated by one part of the application is used as input for another part of the application.
- In this model, the threads of a process are organized as a pipeline so that the output data generated by the first thread is used for processing by the second thread, the output of the second thread is used for processing by the third thread, and so on.
- The output of the last thread in the pipeline is the final output of the process to which the threads belong.

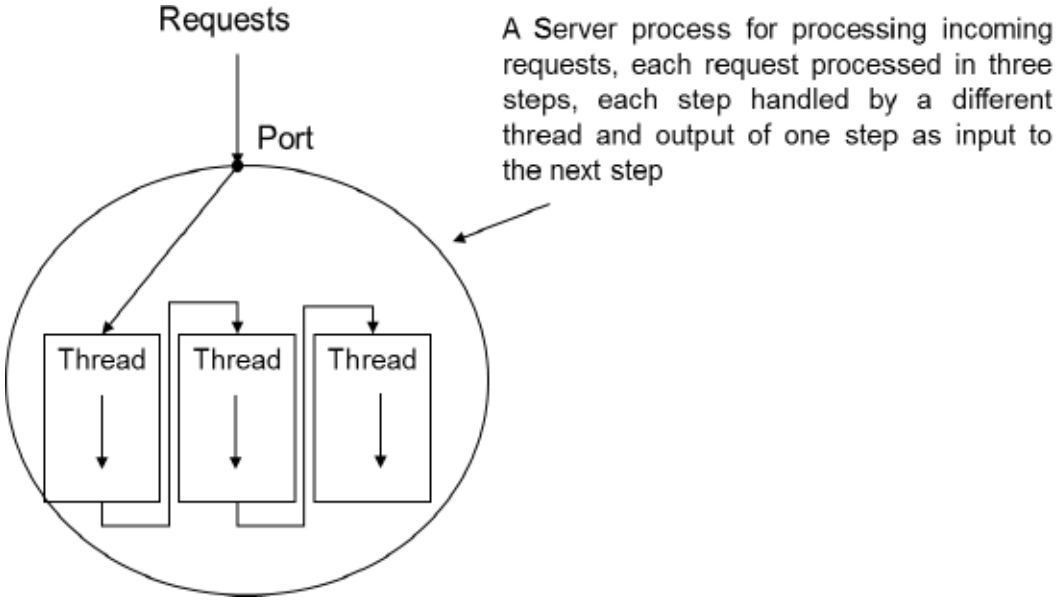


Figure: Pipeline model

Designing issues in Thread package

- A system that supports thread facility must provide a set of primitives to its users for threads - related operations.
- These primitives of the system are said to form a thread package.
- Some of the important issues in designing a thread package are:
  1. Thread Creation
  2. Thread Termination
  3. Thread Synchronization
  4. Thread Scheduling
  5. Signal Handling

Thread Creation

- Threads can be created either statically or dynamically.
  1. **Static:**
     - The number of threads of a process remains fixed for its entire lifetime.
     - Memory space is allocating to each thread.
  2. **Dynamic:**
     - The number of threads of a process keeps changing dynamically.
     - Threads are created as and when it is needed during the process life cycle.
     - It exits when task is completed.
     - Here the stack size for the threads is specified as parameter to the system call for thread creation.

Thread Termination

- Threads may terminate or never terminate until life cycle of process.
- Thread termination can be done as follows:
    - Thread destroys itself on task completion by making an EXIT call.
    - Thread is killed from outside using KILL command with thread identifier as parameter.
- In some process, all its threads are created immediately after the process start and then these threads are never killed until the process terminates.

Thread Synchronization

- Since threads belongs to same process share the same address space, thread synchronization is required to ensure that multiple threads don't access the same data simultaneously.
- For example:
    - If two threads want to increment the same global variable with in the same process.
    - One thread should exclusive access to shared variable, increment it, and then pass control to the other thread.
    - It means that only one thread can execute in critical region at any instance of time.

Thread Scheduling

- Another important issue in designing threads package is to decide an appropriate scheduling algorithm.
- Thread packages provide calls to give the users the flexibility to specify the scheduling policy to be used for their applications.
- Some of the special features for threads scheduling that may be supported by a threads package are as follows:
    - Priority assignment facility
    - Flexibility to vary quantum size dynamically
    - Handoff scheduling
    - Affinity scheduling

Signal Handling

- Signals provide software-generated interrupts and exceptions.
- Interrupts are externally generated disruptions of a thread or process.
- Exceptions are caused by the occurrence of unusual conditions during a thread's execution.
- The two main issues associated with handling signals in a multithreaded environment are as follows:
    - A signal must be handled properly no matter which thread of the process receives it.
    - Signals must be prevented from getting lost when another signal of the same type occurs in some other thread before the first one is handled by the thread in which it occurred.

Scheduling in distributed systems

- A resource manager schedules the processes in a distributed system to make use of the system resources.
- Scheduling is to optimize resource usage, response time, network congestion. It can be broadly classified into three types:

- **Task assignment approach**
    - In which each process is viewed as a collection of related tasks.

- These tasks are scheduled to suitable nodes so as to improve performance.

- **Load-balancing approach**
  - In which all the processes submitted by the users are distributed among the nodes of the system to equalize the workload among the nodes.

- **Load-sharing approach**
  - Which simply attempts to conserve the ability of the system to perform work by assuring that no node is idle while processes wait for being processed.

Desirable features of a good Global scheduling algorithm

1. **No A Priori knowledge about the Processes**

- A good process scheduling algorithm should operate with absolutely no a priori knowledge about the processes to be executed.
- Obtaining prior knowledge poses an extra burden upon the users who must specify this information while submitting their processes for execution.

2. **Dynamic in Nature**

- Process assignment decisions should be based on the current load of the system and not on some fixed static policy.
- Thus system support pre-emptive process migration facility in which a process can be migrated from one node to another during the course of its execution.

3. **Quick Decision making capability**

- A good process scheduling algorithm must make quick decisions about the assignment of processes to processors.
- For example, an algorithm that models the system by a mathematical program and solves it on line is unsuitable because it does not meet this requirement.
- Heuristic methods requiring less computational effort while providing near optimal results are therefore normally preferable to exhaustive solution methods.

4. **Balance System Performance and Scheduling Overhead**

- Several global scheduling algorithms collect global state information and use this information in making process assignment decisions.
- A common intuition is that greater amounts of information describing global system state allow more intelligent process assignment decisions to be made that have a positive effect on the system as a whole.
- In a distributed environment, however, information regarding the state of the system is typically gathered at a higher cost than in a centralized system.
- Hence algorithms that provide near optimal system performance with a minimum of global state information gathering overhead are desirable.

5. **Stability**

- A scheduling algorithm is said to be unstable if it can enter a state in which all the nodes of the system are spending all of their time migrating processes without accomplishing any useful work in an attempt to properly schedule the processes for better performance.
- This form of fruitless migration of processes is known as processor thrashing.

- Processor thrashing can occur in situations where each node of the system has the power of scheduling its own processes and scheduling decisions are based on relatively old data to transmission delay between nodes.
- For example, it may happen that node n1 and n2 both observe that node n3 is idle and then both offload a portion of their work to node n3 without being aware of the offloading decision made by the other.
- Now if node n3 becomes overloaded due to the processes received from both nodes n1 and n2, then it may again start transferring its processes to other nodes.
- This entire cycle may be repeated again and again, resulting in an unstable state.

6. Scalable

- Algorithm should be scalable and able to handle workload inquiry from any number of machines in the network.
- The N2 nature of the algorithm creates more network traffic and quickly consumes network bandwidth.
- A simple approach to make an algorithm scalable is to probe only m of N nodes for selecting a host.
- The value of m can be dynamically adjusted depending upon the value of N.

7. Fault Tolerance.

- A good scheduling algorithm should not be disabled by the crash of one or more nodes of the system.
- At any instance of time, it should continue functioning for nodes that are up at that time.

8. Fairness of Service.

- In any load balancing scheme, heavily loaded nodes will obtain all the benefits while tightly loaded nodes will suffer poor response time A fair strategy that improves response time of heavily loaded nodes without unduly affecting response time of poorly loaded node.

Processor Allocation OR Task Assignment in Distributed system

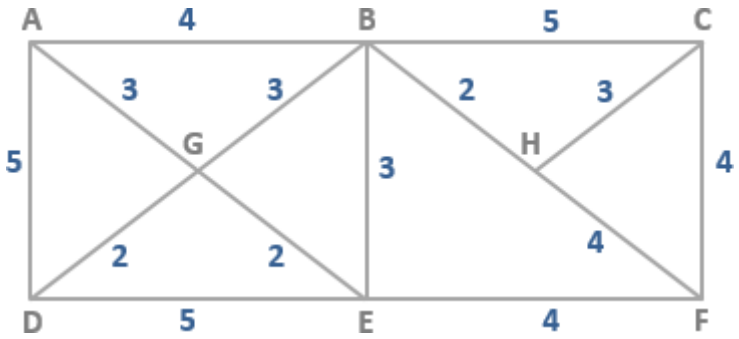
- Each process is divided into multiple tasks.
- These tasks are scheduled to suitable processor to improve performance.
- It requires characteristics of all the processes to be known in advance.
- This approach does not take into consideration the dynamically changing state of the system.
- In this approach, a process is considered to be composed of multiple tasks and the goal is to find an optimal assignment policy for the tasks of an individual process.

Process Allocation OR Task Assignment Algorithms

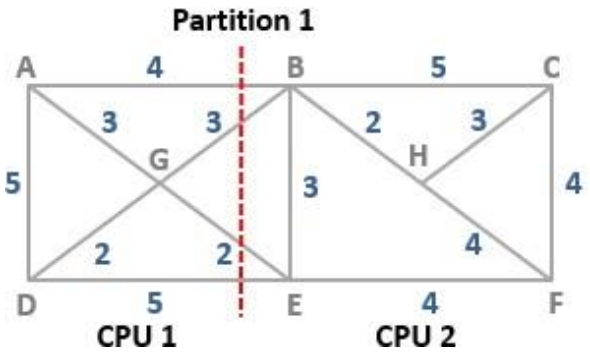
1. Graph Theoretic Deterministic Algorithm

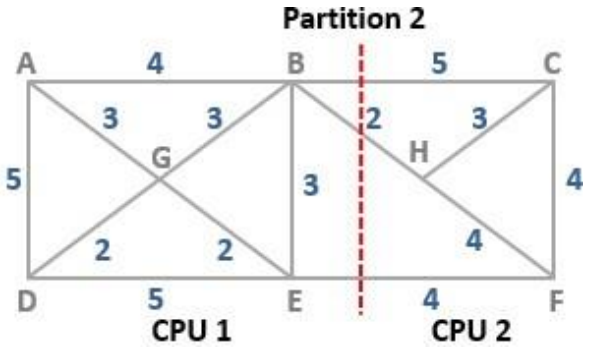
- A system with m CPUs and n processes has any of the following three cases:
- m=n: Each process is allocated to one CPU.
- M>n: Some CPUs may remain idle (free) or work on earlier allocated processes.
- M<n: There is a need to schedule processes on CPUs, and several processes may be assigned to each CPU.
- The main objective of performing CPU assignment is to:
    - Minimize IPC cost.
    - Obtain quick turnaround time.

- Achieve high degree of parallelism for efficient utilization.
- Minimize network traffic.



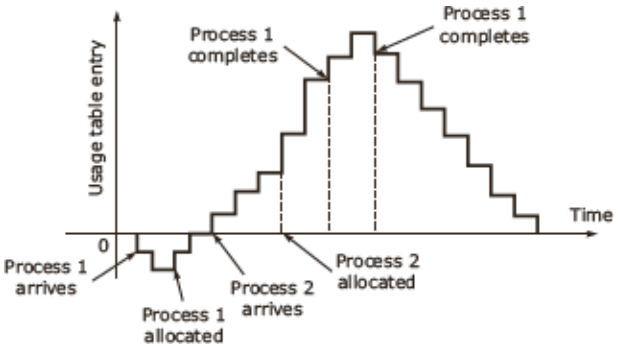
- Processes are represented as nodes A, B, C, D, E, F, G and H.
- Arcs between sub-graphs represent network traffic and their weights represent IPC costs.
- Total network traffic is the sum of the arcs intersected by the dotted cut lines.





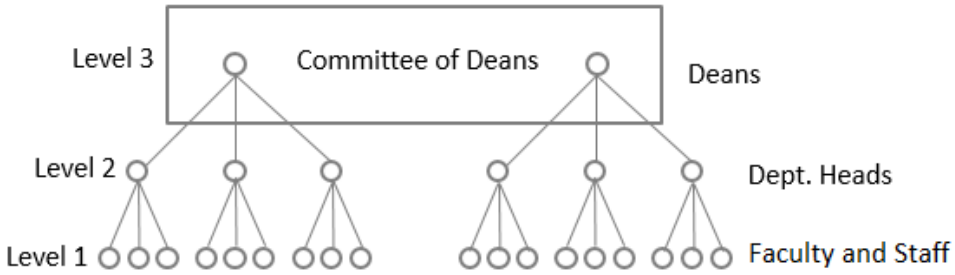
- Partition 1
  - CPU 1 runs A, D, G
  - CPU 2 runs B, E, F, H and C
  - Network traffic= 4+3+2+5=14
- Partition 2
  - CPU 1 runs processes A, D, E, G, B
  - CPU2 runs processes H, C and F
  - Network traffic= 5+2+4=11
  - Thus partition 2 communication generates less network traffic as compared to partition 1.

Centralized Heuristic Algorithm



- It is also called Top down algorithm.
- It doesn't require advance information.
- Coordinator maintains the usage table with one entry for every user (processor) and this is initially zero.
- Usage table entries can either be zero, positive, or negative.
    - Zero value indicates a neutral state.
    - Positive value implies that the machine is using system resources.
    - Negative value means that the machine needs resources.

Hierarchical Algorithm



- Process hierarchy is modelled like an organization hierarchy.
- For each group of workers, one manager machine (department head) is assigned the task of keeping track of who is busy and who is idle.
- If the system is large, there will be number of department heads, so some machines will function as "deans".
- Each processor maintains communication with one superior and few subordinates.
- When a dean or department head stops functioning (crashes), promote one of the direct subordinates of the faulty member to fill in for the head.
- The choice of which can be made by the subordinates themselves.

Load balancing approaches

- The distribution of loads to the processing elements is simply called the load balancing.
- Load balancing approaches can be classified as follows:

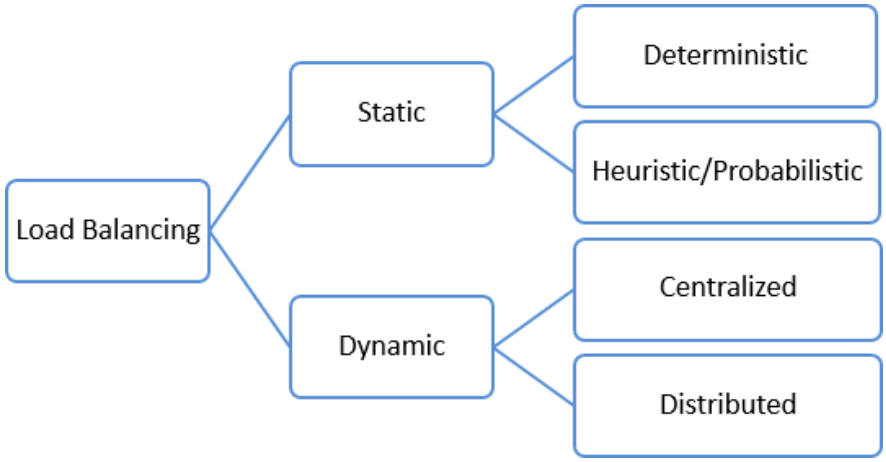


Figure: Classification of Load Balancing approaches

Static Load Balancing

- In static algorithm the processes are assigned to the processors at the compile time according to the

performance of the nodes.

- Once the processes are assigned, no change or reassignment is possible at the run time.
- Number of jobs in each node is fixed in static load balancing algorithm.
- Static algorithms do not collect any information about the nodes.
- The static load balancing algorithms can be divided into two sub classes:
    - **Optimal Static Load Balancing Algorithm**
        - If all the information and resources related to a system are known optimal static load balancing can be done.
    - **Sub optimal static load balancing Algorithm**
        - Sub-optimal load balancing algorithm will be mandatory for some applications when optimal solution is not found.

Dynamic Load Balancing

- In dynamic load balancing algorithm assignment of jobs is done at the runtime.
- In DLB jobs are reassigned at the runtime depending upon the situation.
- The load will be transferred from heavily loaded nodes to the lightly loaded nodes.
- No decision is taken until the process gets executed.
- This strategy collects the information about the system state and about the job information.
- As more information is collected, the algorithm can make better decision.

Deterministic Vs Probabilistic

- Deterministic algorithms are suitable when the process behavior is known in advance.
- If all the details like list of processes, computing requirements, file requirements and communication requirements are known prior to execution, then it is possible to make a perfect assignment.
- In the case load is unpredictable or variable from minute to minute or hour to hour, a Probabilistic/heuristic processor allocation is preferred.

Centralized Vs Distributed

CENTRALIZED LOAD BALANCING	DISTRIBUTED LOAD BALANCING
Scheduling decision is carried out at one single node called the centralized node.	Scheduling decision is carried out at different nodes called the Distributed nodes.
Every information is available at a single node.	Information is distributed at different nodes.
Centralized approach leads to a bottleneck as number of requests increases.	Distributed approach handle the multiple requests by physically distributing among various nodes.
If centralized server fails, all scheduling in the system will fails.	If distributed server fails, all scheduling will be handled by other servers.

Advantages of Load balancing

- Load balancing improves the performance of each node and hence the overall system

performance will improve.

- Load balancing reduces the job idle time.
- Low cost but high gain.
- Small jobs do not suffer from long starvation.
- Extensibility and incremental growth.
- Maximum utilization of resources.
- Higher throughput.
- Response time becomes shorter.

Issues in Designing Load Balancing Algorithms

- **Load estimation:** determines how to estimate the workload of a node in a distributed system.
- **Process transfer:** decides whether the process can be executed locally or remotely.
- **Static information exchange:** determines how the system load information can be exchanged among the nodes.
- **Location policy:** determines the selection of a destination node during process migration.
- **Priority assignment:** determines the priority of execution of a set of local and remote processes on a particular node.
- **Migration limiting policy:** determines the total number of times a process can migrate from one node to another.

Load sharing in Distributed system

- Load sharing algorithms ensure that no node is idle or heavily loaded.
- Policies for load sharing approach are the same as load balancing polices.
- They include load estimation policy, process transfer policy, location policy and state information exchange.
- They differ in location policy.

Location Policies

- The location policy decides the sender node or the receiver node of a process that is to be moved within the system for load sharing.
- Depending on the type of node that takes the initiative to globally search for a suitable node for the process, the location policies are of the following types:

1. **Sender-initiated policy:**
   - In which the sender node of the process decides where to send the process.
   - The heavily loaded nodes search for lightly loaded nodes to which work may be transferred.
   - When a node's load becomes more than the threshold value, it either broadcasts a message or randomly probes the other nodes one by one to find a lightly loaded node that can accept one or more of its processes.
   - If a suitable receiver node is not found, the node on which the process originated must execute that process.

2. **Receiver-initiated policy:**
   - In which the receiver node of the process decides from where to get the process.

- In this policy lightly loaded nodes search for heavily loaded nodes from which processes can be accepted for execution.
- When the load on a node falls below a threshold value, it broadcasts a probe message to all nodes or probes nodes one by one to search for a heavily loaded node.
- Some heavily loaded node may transfer one of its process if such a transfer does not reduce its load below normal threshold.

Process Migration

- Process migration is a specialized form of process management whereby processes are moved from one computing environment to another.
- Process migration is classified into two types:
  - Non-preemptive: Process is migrated before execution start in source node.
  - Preemptive: Process is migrated during its execution.

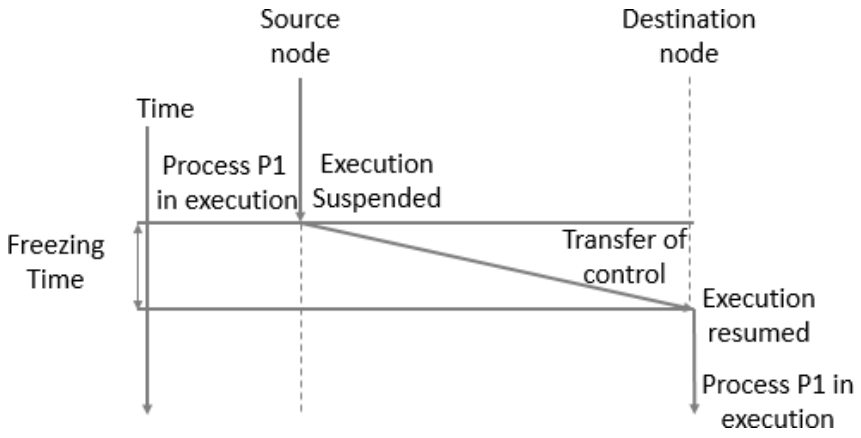


Figure: Flow of execution of a migrating process

- Process migration involves the following major steps:
  1. Selection of a process that should be migrated.
  2. Selection of the destination node to which the selected process should be migrated.
  3. Actual transfer of the selected process to the destination node.

Advantages of Process Migration

1. **Reducing average response time of processors**
   - Process migration facility is being used to reduce the average response time of the processes of a heavily loaded node by some processes on idle or underutilized nodes.

2. **Speeding up individual jobs**
   - Process migration facility may be used to speed up individual jobs in two ways.
   - The first method is to migrate the tasks of a job to the different nodes of the system and to execute them concurrently.
   - The second approach is to migrate a job to a node having a faster CPU or to a node at which it has minimum turnaround time.

3. **Gaining higher throughout**
   - In a system with process migration facility, the capabilities of the CPUs of all the nodes

can be better utilized by using a suitable load-balancing policy.
- This helps in improving the throughput of the system.

4. Utilizing resources effectively
- Process migration policy also helps in utilizing resources effectively, since any distributed system consisting of different resources such as CPU, printers, storage etc. and software (databases, files with different capabilities are optimally used.
- Depending nature of process, it can be appropriately migrated to utilize the system resource efficiently.

5. Reducing network traffic
- Migrating a process closer to the resource it is using most heavily may reduce network traffic in the system if the decreased cost of accessing its favorite resources offsets the possible increased cost of accessing its less favored ones.
- Another way to reduce network traffic by process migration is to migrate and cluster two or more processes, which frequently communicate with each other, on the same node of the system.

6. Improving system reliability
- Simply migrate a critical process to a node whose reliability is higher than other nodes in the system.
- Migrate a copy of a critical process to some other node and to execute both the original and copied processes concurrently on different nodes.
- In failure modes such as manual shutdown, which manifest themselves as gradual degradation of a node, the process of the node, for their continued execution, may be migrated to another node before the dying node completely fails.

7. Improving system security
- A sensitive process may be migrated and run on a secure node that is not directly accessible to general users, thus improving the security of that process.

Desirable Features of a Good process migration mechanism:

1. Transparency
- Transparency is an important requirement for a system that supports process migration.
- Object access level transparency
  - Object access level transparency is the minimum requirement for a system to support non preemptive process migration
  - Access to objects such as files and devices can be done in a location independent manner.
  - The object access level transparency allows, free initiation of programs at an arbitrary node.
- System call and inter process communication transparency
  - System call and inter process communication level transparency is must for a system to support preemptive process migration facility.
  - Transparency of inter process communication is also desired for the transparent redirection of messages during the transient state of process that recently migrated.

2. Minimal Interface

- Migration of a process should cause minimal interference to the progress of the process involved and to the system as a whole.
- This can be achieved by minimizing the freezing time of the process being migrated.

3. Minimal Residual Dependencies

- A migrated process should not in any way continue to depend on its previous node once it has started executing on its new node since, otherwise, the following will occur.

4. Efficiency

- It is the major issue in implementing process migration.
- Main sources of inefficiency are: time required to migrate a process, the cost of locating an object and the code of supporting remote execution once the process is migrated.

5. Robustness

- The failure of a node other than the one on which a process is currently running should not in any way affect the accessibility or execution of that process.

6. Communication between co-processes of a job

- Benefit of process migration is the parallel processing among the processes of a single job distributed over several nodes.
- Co-processes are able to directly communicate with each other irrespective of their locations.

Fault Tolerance and classification of Component faults.

- The main objective of designing a fault tolerance system is to ensure that the system operates correctly even in presence of faults.
- Distributed system made up of independent computers connected over the network, can fail due to fault in any of hardware components or software components.
- **Component faults can be classified as:**

1. Transient faults

- It can occur suddenly, disappear, and may not occur again if the operation is repeated.
- For example:
  - During heavy traffic in network, a telephone call is misroute but if call is retried, it will reach destination correctly.

2. Intermittent faults

- These faults occur often, but may not be periodic in nature.
- For example:
  - The loose connection to network switch may cause intermittent connection problems.
  - Such faults are difficult to diagnose.
  - During debugging process, it is possible that the fault may not be detected at all.

3. Permanent faults

- These faults can be easily identified and the components can be replaced.
- For example:
  - Software bug or disk head crash causes permanent faults.
  - These faults can be identified and corresponding action can be taken.

Technique to handle Redundancy

- Technique to handle fault tolerance is **Redundancy**.
- Types of Redundancy are as follows:

1. Information redundancy

- Extra bits are added to data to handle fault tolerance by detecting errors.

2. Time redundancy

- An action performed once is repeated if needed after a specific time period.
- For example, if an atomic transaction aborts, it can be executed without any side effects.
- Time redundancy is helpful when the faults are transient or intermittent.

3. Physical redundancy

- Extra equipment's are added to enable the system to tolerate faults due to loss or malfunction of some components.
- For example, extra stand by processors can be used in the system.

Real time distributed system

- It is developed by collection of computers in a network.
- Some of these are connected to external devices that produce or accept data or expect to be controlled in real time.
- Computers may be tiny microcontrollers or stand-alone machines.
- In both cases they usually have sensors for receiving signals from the devices and actuators for sending signals to them.
- The sensors and actuators may be digital or analog.

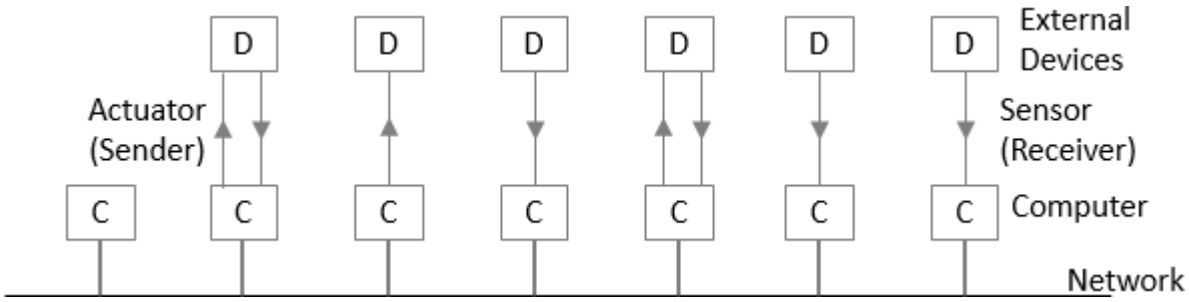


Figure: A Distributed real-time computer system

Types of Real time distributed system

- Real-time systems are generally split into two types depending on how serious their deadlines are and the consequences of missing one.
  1. **Soft real time system:** missing an occasional deadline is all right. For example, a telephone switch might be permitted to lose or misroute one call in 10^5 under overload conditions and still be within specification.
  2. **Hard real time system:** even a single missed deadline in a hard real-time system is unacceptable, as this might lead to loss of life or an environmental disaster.