Process Management

Process management deals with mechanisms and policies for sharing the processor of the system among all processes.

3 important concepts

1. **Processor allocation** deals with the process of deciding which process should be assigned to which processor.
2. **Process migration** deals with the movement of a process from its current location to the processor to which it has been assigned.
3. **Threads** deals with fine-grained parallelism for better utilization of the processing capability of the system.

Process Migration

Process migration is the relocation of a process from its current location (the source node) to another node (the destination node). The flow of execution of a migrating process is illustrated in Figure 8.1.

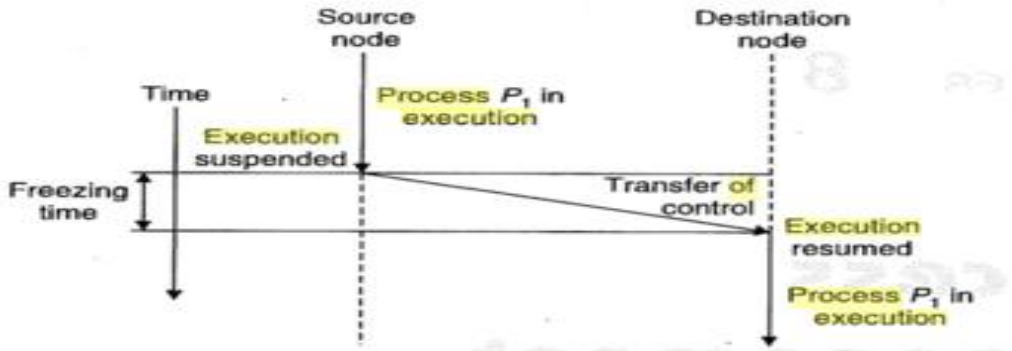


Fig. 8.1 Flow of execution of a migrating process.

A process may be migrated either before it starts executing on its source node (**non-preemptive process migration)** or during the course of its execution (**preemptive process migration)**.

Major steps in Process migration

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.

Desirable Features of a Good Process Migration

1. Transparency

a. Object access level

If a system supports transparency at the object access level

- access to objects such as files and devices can be done in a location independent manner.
- allows free initiation of programs at an arbitrary node.
- the system must provide a mechanism for transparent object naming and locating.

b. System call and interprocess communication level

A migrated process does not continue to depend upon its originating node after being migrated. It is necessary that all system calls, including interprocess communication, are location independent.

Transparency of interprocess communication is also the transparent redirection of messages during the transient state of process that recently migrated. That is, once a message sent, it should reach its receiver process without the need for resending a from the sender node is sure the receiver process moves to another node before the message is received.

2. Minimal Interference

Migration of a process should cause minimal interference to the progress of the process involved the system as a whole. One method to achieve this is by minimizing the freezing time of the process being migrated. Freezing time is defined as the time period for which the execution of the process is stopped for transferring its information to the destination node.

3. Minimal Residual Dependencies

No residual dependency should be left on the previous node. That is, 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:

- The migrated process will continue to impose a load on its previous node.
- A failure or reboot of the previous node will cause the process to fail.

4. Efficiency

The main sources of inefficiency are:

- Time required for migrating a process.
- Cost of location an object
- Cost of supporting remote execution once the process is migrated.

All these costs should be kept to minimum.

5. Robustness

It is the case where 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 Coprocesses of a job

If this parallel processing facility is supported, to reduce communication cost, it is also necessary that these coprocesses be able to directly communicate with each other irrespective of their locations.

Process Migration Mechanisms

4 Major subactivities

1. Freezing the process on its source node and restarting it on its destination node.
2. Transferring the process's address space from its source node to its destination node.
3. Forwarding messages meant for the migrant process.
4. Handling communication between cooperating processes.

1. Mechanisms for Freezing and Restarting a Process

General Issues

a. Immediate and Delayed Blocking of the Process

Before a process can be frozen, its execution must be blocked.

Typical Cases:

- If the process is not executing a system call, it can be immediately blocked from further execution.
- If the process is executing a system call, but is sleeping at an interruptible priority (a priority at which any received signal would awaken the process) waiting for a kernel event to occur, it can be immediately blocked from further extension.
- If the process is executing a system call and is sleeping at a noninterruptible priority waiting for a kernel event to occur, it cannot be blocked immediately. The process's blocking has to be delayed until the system call is complete.

b. Fast and Slow I/O Operations

The process is frozen after the completion of all fast I/O operations (disk I/O). It is not feasible to wait for slow I/O operations, such as those on a pipe or terminal, because the process must be frozen in a timely manner for the effectiveness of process migration.

c. Information about Open Files

A process's state information also consists of the information pertaining to files currently open by process. This includes such information as the names or identifiers of the files, their access modes, and the current positions of their pointer. In a distributed system that provides a network transparent execution environment, there is no problem in collecting this state information because the same protocol is used to access local as well as remote files using the system wide unique their full pathname. But in these systems it is difficult for a process in execution to obtain a file's complete path name owing to UNIX file system semantics.

d. Reinstating the Process on its Destination Node

On the destination node, an empty process state is created that is similar to that allocated during process creation .Once all the state of the migrating process has been transferred from the source node to destination node and copied into empty process state, the new copy of the process is unfrozen and the old copy is deleted. Thus the process is restarted on its destination node in whatever state it was in before being migrated.

2. Address Space Transfer Mechanisms

Process migration involves the transfer of the following information from the source node to the destination node:

- **Process's state** that includes execution status, scheduling information, information about main memory, I/O states, a list of objects to which the process has a right to access, process's identifier, process's user and group identifiers, information about the files opened by the process etc.
- **Process's address space** that includes code, data and stack of the program.

3 methods for Address space transfer

a) Total freezing
b) Pretransferring

c) Transfer on reference.

a) Total Freezing

In this method, process's execution is stopped while its address space is being transferred.

Disadvantages:

- If a process is suspended for a long time during migration, timeouts may occur.
- If the process is interactive, the delay will be noticed by the user.

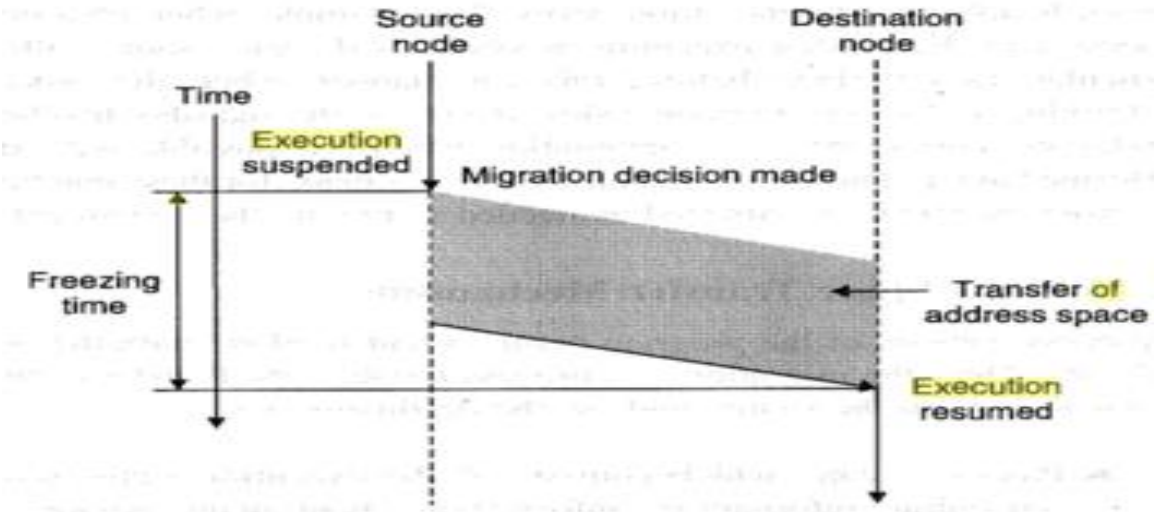


Fig. 8.2 Total freezing mechanism.

b) Pretansferring/Precopying

In this method, address space is transferred while the process is still running on the source node. Pretransferring or precopying is done as an initial transfer of the complete address space followed by repeated transfers of the pages that are modified.

In this pretransfer operation,

- first transfer moves the entire address space and takes longest time
- second transfer moves only those pages that were modified during the first transfer, thus taking less time.
- Subsequent transfer operations have to move fewer and fewer pages, finally converging to zero or very few pages, which are then transferred after the process is frozen.

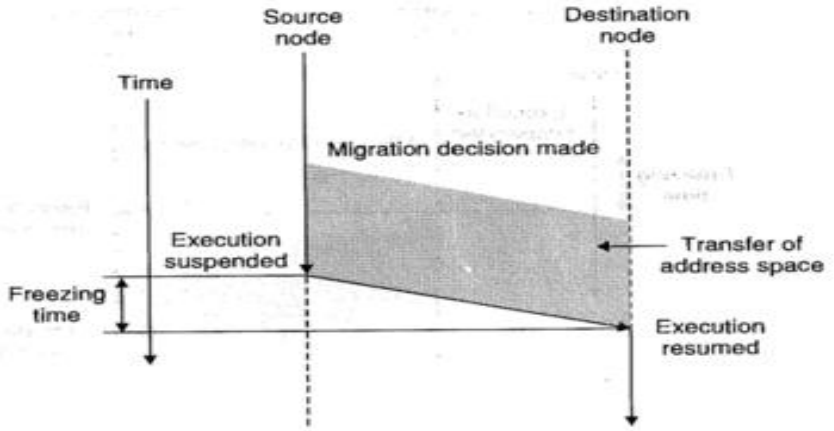
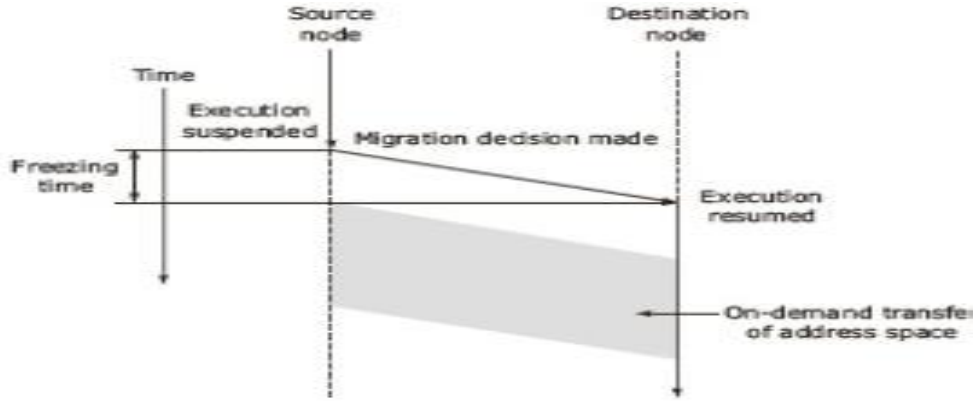


Fig. 8.3 Pretransfer mechanism.

c) Transfer on Reference

- This method is based on the assumption that processes tend to use only a relatively a small part of their address spaces while executing.
- In this **demand-driven copy-on reference approach**, a page of the migrant process's address space is transferred from source node to its destination node only when referenced.
- Switching time is very short and independent of the size of the address space.
- Not efficient in terms of cost.
- Imposes continued load on process's source node.
- Results in failure if source node fails or reboots.



3. Message-Forwarding Mechanisms

In moving a process, it must be ensured that all pending, en-route and future messages arrive at the process's new location.

Types of Messages

- **Type 1:** Messages received at the source node after the process's execution has been stopped on its source node and the process's execution has not yet been started on its destination node.
- **Type 2:** Messages received at the source node after the process's execution has started on its destination node.

- **Type 3:** Messages that are to be sent to the migrant process from any other node after it has started executing on the destination node.

Generally followed Message Forwarding Mechanisms

a. Mechanism of Resending the message

- Messages of type 1 and 2 are returned to the sender as not deliverable or are simply dropped, with the assurance that the sender of the message is storing a copy of the data and is prepared to retransmit it. Type 3 messages are sent directly to the process's destination node.
- Does not require any process state to be left behind on the process's source node.
- **Drawback**: It is non transparent to the processes interacting with the migrant process.

b. Origin Site Mechanism

- Each site is responsible for keeping information about the current locations of all processes created on it.
- Messages for a particular process are always sent to its origin site.
- The origin site then forwards the message to the process's current location.
- **Drawback**:
    - Failure of the origin site will disrupt message forwarding mechanism.
    - Continuous load on the migrant process's origin site even after the process has migrated from that node.

c. Link Traversal Mechanism

- Messages of type 1 are placed in a message queue in the source node.
- Messages of type 2 and 3 are redirected using the forwarding address called **link** which is placed in the source node pointing to the destination node of the migrant process.
- Message process address is the main part of the link that contains **2 components**
    - systemwide, unique, process identifier
    - last known location of the process
- Migrated process is located by traversing a series of links.

d. Link Update Mechanism

- Processes communicate via location-independent links which are capabilities of duplex communication channels.
- During the transfer phase, source node sends link-update messages to the kernels controlling all of the migrant process's communication partners.
- **Link Update Message**:
    - Tells the new address of each link held by the migrant process.
    - Acknowledged for synchronization purposes.
- Messages of type 1 and 2 are forwarded to the destination node by the source node.
- Type 3 messages are sent directly to the destination node.

4. Mechanisms for handling Coprocesses

It provides efficient communication between a process (parent) and its subprocesses (children).

2 Mechanisms

a) Disallowing Seperation of Coprocesses
b) Home Node or Origin Site Concept

a) Disallowing Seperation of Coprocesses

This can be achieved by the following ways:

- By disallowing the migration of processes that wait for one or more of their children to complete.
- By ensuring that when a parent process migrates, its children processes will be migrated along with it.

Drawback:

- Does not allow parallism within jobs
- Overhead is large when logical host contains several processes

b) Home Node or Origin Site Concept

This allows the complete freedom of migrating a process or its subprocesses independently and executing them on different nodes.

Drawback: Since all communications between a parent process and its children processes take place via the home node, the message traffic and the communication cost increase considerably.

Advantages of Process Migration

1. **Reducing average response time of processes:** Process migration facility is used to reduce the average response time of the processes of a heavily loaded node by migrating some of the processes that is either idle or whose processing capacity is underutilized.
2. **Speeding up individual jobs:** There are two methods to speed up individual jobs. One method is to migrate the tasks of a job to different nodes to execute them concurrently. Another method is to migrate a job to a node having a faster CPU.
3. **Gaining higher throughput:** The throughput is increased by using a suitable load balancing approach. And it also executes a mixture of I/O and CPU bound processes on a global basis to increase the throughput.
4. **Utilizing resources effectively:** In a distributed system there are various resources such as CPU's, printers, storage devices etc. Depending upon the nature of the process the resources has to be allocated to the suitable node.
5. **Reducing network traffic:** Migrating a process closer to the resources it is using is a mechanism used to reduce the network traffic. It can also migrate and cluster two or more processes which frequently communicate with each other on the same node of the system.
6. **Improving system reliability:** System reliability can be improved in two ways. One way is to migrate a critical process to a node whose reliability is higher than other nodes in the system. Another way is to migrate a copy of the critical process

to some node and execute both the original and copied processes concurrently on different nodes.

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.

Threads

In operating systems with threads facility, a process consists of an address space and one or more threads of control. Each thread of a process has its own program counter, its own register states, and its own stack. But all the threads of a process share the same address space. Hence they also share the same global variables.

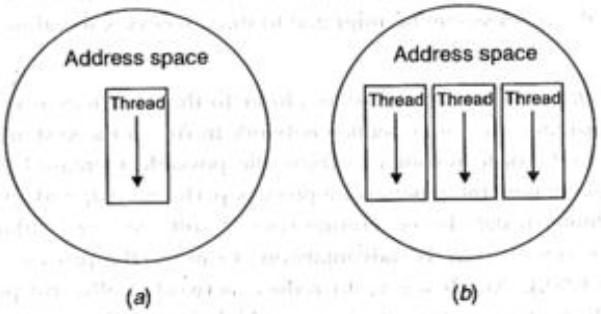


Fig. 8.6 (*a*) Single-threaded and (*b*) multithreaded processes. A single-threaded process corresponds to a process of a traditional operating system.

8.3.1 Motivations for Using Threads

1. The overheads involved in creating a new process are in general considerably greater than those of creating a new thread within a process.
2. Switching between threads sharing the same address space is considerably cheaper than switching between processes that have their own address spaces.
3. Threads allow parallelism to be combined with sequential execution and blocking system calls.
4. Resource sharing can be achieved more efficiently and naturally between threads of a process than between processes because all threads of a process share the same address space.

Models for Organizing Threads

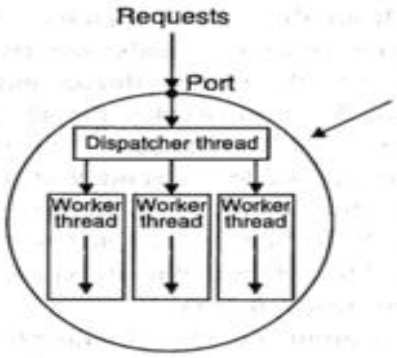
3 ways to organize the threads of a process

1. Dispatcher-workers model
2. Team model
3. Pipeline model

1. Dispatcher-workers model

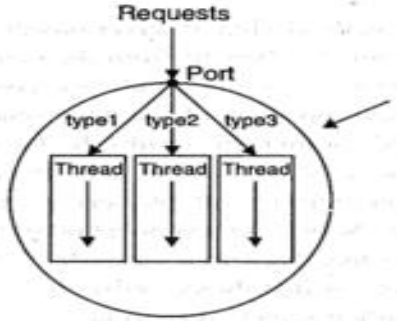
- The process consists of a single dispatcher thread and multiple worker threads.
- The **dispatcher thread** accepts requests from clients and, after examining the request, dispatches the request to one of the free worker threads for further processing of the request.

- Each **worker thread** works on a different client request. Therefore multiple client requests can be processed in parallel.



2. Team model

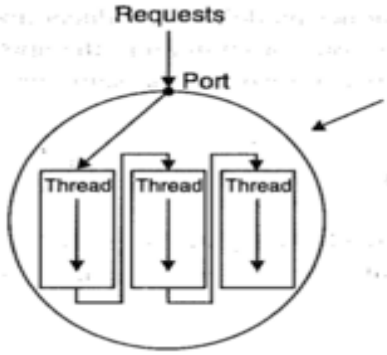
All threads behave as equals in the sense that there is no dispatcher-worker relationship for processing clients' requests. Each thread gets and processes clients' requests on its own.



3. Pipeline model

This model is based on the **producer-consumer model**, in which the output data generated by one part of the application is used as input for another part of the application.

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.



Issues in Designing a Threads Package

A system that supports threads facility must provide a set of primitives to its users for thread related operations and those primitives forms the **thread package**

Thread Creation

Threads can be created either statically or dynamically. In **static approach**, the number of threads of a process remains fixed for its entire lifetime. Fixed stack is allocated for each thread in this approach. On the other hand, in **dynamic approach** number of threads for a process keeps changing dynamically. New threads are created when needed. Here the stack size for each thread is passed as parameter in the system call.

Thread Termination

A thread may either destroy itself after it finishes its job or it can be killed from outside using kill command. In many cases threads are never terminated. In some cases, threads are never killed until the process terminates.

Thread Synchronization

Generally all threads share a common address space, some mechanism must be used to prevent multiple threads from trying to access the same data simultaneously. A segment of code in which a thread may be accessing some shared variable is called critical region. Execution of critical regions in which the same data is accessed by the threads must be mutually exclusive in time. A mutex variable is like a binary semaphore that is always in one of the two states locked or unlocked.

If the mutex variable is already locked depending on the implementation the lock operations is handled in one of the following ways.

1. The thread is blocked and entered in a queue of threads waiting on the mutex variable.
2. A status code indicating failure is returned to the thread.

When a thread finishes executing in its critical region it performs an unlock operation on the corresponding mutex variable.

A condition variable is associated with the mutex variable and reflects a Boolean state of that variable. Wait and signal are two operations normally provided for a condition variable.Condition variables are often used for cooperation between threads.

Thread Scheduling

Another important issue in the design of threads is how to schedule the threads.Some of the special features of threads scheduling are:

Priority assignment facility - In a simple scheduling algorithm, threads are scheduled on a first-in,first-out basis or round robin policy is used to timeshare the CPU cycles among the threads on a quantum-by-quantum basis.Priority based scheduling algorithm can be either preemptive or non-preemptive.Higher priority thread always preempts the lower priority thread.

Flexibility to vary quantum size dynamically - Generally in round robin scheduling algorithm, a fized quantum size is used to timeshare the CPU cycles.Fixed length quantum is not suitable in a multiprocessor system, hence

quantum length can be kept as variable.In this case, it gives good response time to short requests.

Handoff Scheduling - A handoff scheduling scheme allows a thread to name its successor if it wants to. In this case, after sending a message to another thread, the sending thread can give up the CPU and request the receiving thread be allowed to run next.

Affinity Scheduling - In this scheduling, a thread is scheduled on the CPU it last ran on in hopes that part of its address space is still in that CPU's cache.It is used in multiprocessor system.

Desirable features of good global scheduling algorithms

No a Priori knowledge about the processes – A good process scheduling algorithm should operate with absolutely no a prori knowledge about the processes to be executed.

Dynamic in nature – A good process scheduling algorithm should be able to take care of the dynamically changing load of the various nodes of the system. This feature also requires that the system support preemptive process migration facility.

Quick Decision-Making Capability – A process scheduling algorithm must make quick decisions about the assignment of processes to processors.

Balanced system performance and Scheduling overhead–Global scheduling algorithms collect global state information and use this information in making process assignment decisions.An overhead is also is involved in collecting more information about the global state of the systems.

Stablity–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.This form of fruitless migration of processes is called as processor thrashing.

Scalability–A scheduling algorithm should be capable of handling small as well as large networks.

Fault tolerance – A scheduling algorithm should not be disabled by the crash of one or more nodes of the system and hence it is said to be fault tolerant.

Fairness of Service – Fairness of service is the case where if two users simultaneously initiates equivalent processes expect to receive about the same quantity of service. Load sharing concept is introduced to facilitate this service.

Task assignment approach

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.

Assumptions made in this approach are:

- A process has already been split into pieces called tasks.
- Amount of computation required by each task and the speed of each processor are known.

- Cost of processing each task on every node of the system is known.
- Interprocess Communication(IPC) costs between every pair of tasks is known.

The main goals to be achieved in task assignment algorithms are:

- Minimization of IPC costs.
- Quick turnaround time for the complete process.
- A high degree of parallelism
- Efficient utilization of system resources is gained.

Considering an assignment problem, the intertask costs and the execution costs are given in a tabular form with tasks(t1,t2..etc) and the nodes(n1,n2,..).Infinite costs of a particular task against a particular node indicates that the task cannot be performed in that node.Serial assignment of the tasks along with the node is illustrated in a tabular form.

The problem of finding an assignment of tasks to nodes that minimizes the total execution and communication costs is analyzed using a network flow model.

Optimal assignment is found using a static assignment graph.A cutset is a graph defined with a set of edges such that when these edges are removed, the nodes of the graph is partitioned into two disjoint subsets such that the nodes in one subset are reachable from n1 and the nodes in the other are reachable from n2.

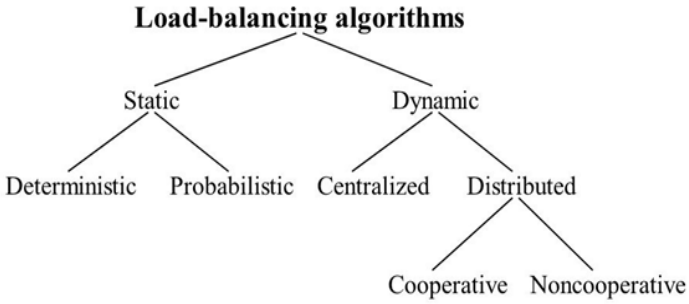
Load balancing approach in distributed systems

Scheduling algorithms using load balancing approach are known as load-balancing algorithms or load-leveling algorithms. Load balancing algorithm tries to balance the total system load by transparently transferring the workload from heavily loaded nodes to lightly loaded nodes. It ensures overall good performance of the system. Basic goal of almost all load balancing algorithms is to maximize the system throughput.

Taxonomy of Load Balancing Algorithms

Static Vs Dynamic

Static algorithms use only information about the average behavior of the system ignoring the current state of the system. On the other hand, dynamic algorithms react to the system state that changes dynamically.Static algorithms are simpler in nature but give a less performance compared to dynamic algorthims.



Deterministic Vs Probabilistic

Static load-balancing algorithms may be either deterministic or probabilistic. Deterministic algorithms use the information about the properties of the nodes and the characteristics of the process to be scheduled to deterministically allocate processes to nodes. Probabilistic load balancing algorithms uses information regarding static attributes of the system such as number of nodes, processing capability of each node, network topology etc.

Centralized Vs Distributed

In Centralized dynamic scheduling algorithm the responsibility of scheduling physically resides on a single node. In this the system state information is maintained with a centralized server node. In Distributed dynamic scheduling algorithm, the information is distributed among the various nodes of the system.

Cooperative Vs Noncooperative

In noncooperative algorithms, individual entities act as autonomous entities and make scheduling decisions independent of the actions of other entities. In cooperative algorithms, distributed entities cooperate with each other to make scheduling decisions.

Load estimation approaches

The main goal of load balancing algorithms is to balance the workload on all nodes of the system. A node's workload can be estimated based on some measurable parameters. The parameters include time-dependent and node-dependent factors such as the following :

Total number of processes on the node at the time of load estimation.

Resource demands of the processes

Instruction mixes of the processes

Architecture and speed of the node's processor

The main challenge of this method is to calculate the node's workload. One of the measures to calculate the node's workload is the sum of the remaining service time of all the processes on that node.

Following methods are used for the above purpose:

Memoryless method:This method assumes that all processes have the same expected remaining service time independent of the time used so far.

Pastrepeats: This method assumes that the remaining service of a process is equal to the time used so far by it.

Distribution method: If the distribution of service times is known, the associated processes

remaining service time is the expected remaining time conditioned by the time already used.

Central processing unit(CPU) utilization is defined as the number of CPU cycles actually executed per unit of real time.

Priority assignment policy for Load-balancing algorithms

When process migration is supported by a distributed operating system, it becomes necessary to devise a priority assignment rule for scheduling both local and remote processes at a particular node.

One of the following priority assignment rules may be used for this purpose:

Selfish : Local processes are given higher priority than remote processes.

Altruistic : Remote processes are given higher priority than local processes

Intermediate: The priority of processes depends on the number of local processes and the number of remote processes at the concerned node.

A study on the effect of the above three priority assignment policies on the overall response time performance reveals the following results:

The selfish priority assignment rule yields the worst response time performance of the three policies.

The altruistic priority assignment rule achieves the best response time performance of the three policies.

The performance of the intermediate priority assignment rule falls between the other two policies.

Migration Limiting Policies

Another important policy to be used by a distributed operating system that supports process migration is to decide about the total number of times a process should be allowed to migrate.One of the following two policies may be used for this purpose:

Uncontrolled: In this case, a remote process arriving at a node is treated just as a process originating at the node.Under this policy, a process may be migrated any number of times.

Controlled: To overcome the instability problem of the uncontrolled policy most systems treat remote processes different from local processes and use a migration count parameter tp fix a limit on the number of times that a process may migrate.

Load sharing approach.

Proper utilization of the resources of a distributed system serves to be challenge and this is met through some of the load sharing approaches. There are many important decisions to be made on load sharing approaches.

Load Estimation policies

Since load sharing algorithms simply attempt to ensure that no node is idle while processes wait for service at some other node, it is sufficient to know whether a node is busy or idle.

Process Transfer policies

Since load sharing algorithms are normally interested only in the busy or idle states of a node, most of them employ the all-or-nothing strategy. In the all-or-nothing strategy, a node that becomes idle is unable to immediately acquire new processes to execute even though processes wait for service at other nodes, resulting in a loss of available processing power in the system.

Location policies

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 the node that takes the initiative there are two different location policies. They are:

Sender-Initiated policy

In this policy, the sender node of the process decides where to send the process. Heavily loaded nodes search for lightly loaded nodes to which work has to 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 process.

Receiver-Initiated policy

In this policy, the receiver node of the process decides where to get the process. Lightly loaded nodes search for heavily loaded nodes from which work can be transferred.When a node's load falls below the threshold value it either broadcasts a message indicating its willingness to receive process or randomly sends probes to heavily loaded nodes.

LOAD SHARING APPROACH

Several researchers believe that load balancing with its implication of attempting to equalize workload on all the nodes o the system, is not an appropriate objective. This is because the overhead involved in gathering state information to achieve this objective is normally very large, especially in distributed systems having a large number of nodes. Moreover, load balancing in the sense is not achievable because the number of processes in a node is always fluctuating and the temporal unbalance among the nodes exists at every moment, even if the static (average) load is perfectly balanced for the proper utilization the resources of a distributed system, it is not required to balance the load on all the nodes. Rather, it is necessary and sufficient to prevent the nodes from being idle while some other nodes have more than two processes. Therefore this rectification is often called dynamic load sharing instead of dynamic load balancing.

Issues in Designing Load Sharing Algorithms :

Similar to the load balancing algorithms, the design of a load sharing algorithm also requires that proper decisions be made regarding load estimation policy, process transfer policy, state information exchange policy, location policy, priority assignment policy, and

with threads facility, a process having a single thread corresponds to a process of a traditional operating system. Threads are often referred to as lightweight processes and traditional processes are referred to as heavyweight processes.

Motivations for Using Threads:

The main motivations for using a multithreaded process instead of multiple single threaded processes for performing some computation activities are as follows :

1. The overheads involved in creating a new process are in general considerably greater than those of creating a new thread within a process.
2. Switching between threads sharing the same address space is considerably cheaper than switching between processes that have their own address space.
3. Threads allow parallelism to be combined with sequential execution and blocking system calls. Parallelism improves performance and blocking system calls make programming easier make programming easier.
4. Resource sharing can be achieved more efficiently and naturally between threads of a process than between processes because all threads of a process share the same address space.

These advantages are elaborated below:

The overheads involved in the creation of a new process and building its execution environment are liable to be much greater than creating a new thread within an existing process. This is mainly because when a new process is created its address space has to be created from scratch, although a part of it might be inherited from the process's parent process. However, when a new thread is created, it uses the address space of its process that need not be created from scratch. For instance, in case of a kernel supported virtual memory system, a newly created process will incur page faults as date and instructions are referenced for the first time. Moreover, hardware caches will initially contain no data values for the new process, and cache entries for the process's data will be created as the process executes. These overheads may also occur in thread creation, but they are liable to be less. This is because when the newly created thread accesses code and data that have recently been accessed by other threads within the process, it automatically takes advantage of any hardware or main memory caching that has taken place.

Threads also minimize context switching time, allowing the CPU to switch from one unit of computation to another unit of computation with minimal overhead. Due to the sharing of address space and other operating system resources among the threads of a process, the overhead involved in CPU switching among peer threads is very small as compared to CPU switching among processes having their own address spaces. This is the reason why threads are called lightweight processes.

True file service: It is concerned with the operation on individual files, such operations for accessing and modifying the data in files and for creating and deleting. To perform these primitive file operations correctly and efficiently, typical design issues of a true file service component include file accessing mechanism, file sharing semantics, file caching mechanism, file replication mechanism, concurrency control mechanism, data consistency and multiple copy update protocol, and access control mechanism. Note that the separation of the storage service from the true file

service makes it easy to combine different methods of storage and different storage media in a single file system.

Name service : IT provides a mapping between text names for files and references to files, that is, file IDs. Text names are required because, file IDs are awkward and difficult for human users to remember and use. Most file systems use directories to perform this mapping. Therefore, the name service is also known as a directory service. The directory service is responsible for performing directory related activities such as creation and deletion of directories, adding a new file to a directory deleting a file from a directory, changing the name of a file, moving a file from one directory to another, and so on.

The design and implementation of the storage service of a distributed file system is similar to that of the storage service of a centralized file system. Readers interested in the details of the storage service may refer to any good book on operating systems. Therefore, this chapter will mainly deal with the design and implementation issues of the true file service component of distributed file systems.