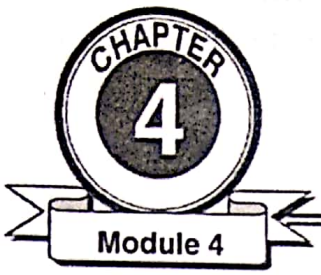


Resource and Process Management

Syllabus

Desirable Features of global Scheduling algorithm, Task assignment approach, Load balancing approach, load sharing approach, Introduction to process management, process migration, Threads, Virtualization, Clients, Servers, Code Migration.

4.1 Introduction

- In distributed system, resources are distributed on many machines. If local node of the process does not have needed resources then this process can be migrated to another node where these resources are available.
- In this chapter, resource is considered as processor of the system. Each processor forms the node of the distributed system.
- Following techniques are used for scheduling the processes of the distributed system
  - **Task assignment approach :** In this method, tasks of each process are scheduled to suitable nodes to improve performance.
  - **Load-balancing approach :** In this approach, main objective is to distribute the processes to different nodes to make workload equal among the nodes in the system.
  - **Load-sharing approach :** In this approach, main objective is to guarantee that no node will be idle while processes in system are waiting for processors.

4.2 Desirable Features of Global Scheduling Algorithm

Following are the desirable features of good global scheduling algorithm

- **No Priori Knowledge about the Processes :** Process scheduling algorithm should not consider a priori knowledge about the characteristics and resources need of the processes. It imposes extra burden on users as they should specify this information while submitting the job for execution.
- **Dynamic in Nature :** Process scheduling algorithm should always consider the current information about load of the system and should not obey any fixed static policy to assign the process to particular node. It should always consider dynamically changing load on the nodes of the system. Hence, algorithm should have flexibility to migrate a process more than once. This is necessary as after some time system load may change. Hence, it is necessary to transfer the process from previous node to new node again to adapt to the changed system load. This is possible only if the system support preemptive process t
- **Quick Decision Making Capability :** Process scheduling algorithm should always make quick decisions about regarding assigning the processes to nodes.
- **Balanced System Performance and Scheduling Overhead :** Process scheduling algorithm should not collect more global state information in order to use it for process assignment decisions.

- In distributed system, collecting global state information requires more overhead. Further, due to aging of information being collected and low scheduling frequency due to cost of collecting and processing, usefulness of information can reduce. Hence, algorithm offering near optimal performance by using minimum global state information is preferred.
- **Stability :** There may be a situation where all the nodes in the system spends their time maximum time in migrating the processes without carrying out any useful work. This migration of processes is carried out to schedule the processes for achieving the better performance. Such unproductive migration is called processor thrashing. Processor thrashing occurs if nodes are capable of scheduling its own processes locally and independently. In this case, node takes scheduling decision based on local information and does not coordinate with other nodes for the same. It may also happen that, the information at node may become old due to transmission delay between the nodes. Suppose node 1 and node 2 observes that node n3 is idle, both sends their part of the load to node 3 without coordinating with each other. In this situation, if node 3 becomes overloaded then it also starts transferring its processes on other nodes. This leads to unstable state of the system due to repetition of such cycle again and again.
- **Scalability :** Process scheduling algorithm should work for small as well as large networks. If algorithm enquires for lightly loaded node and sends processes there then it can work better for smaller networks. In large network, as many replies needs to be processes it consumes bandwidth as well as processing power. Hence, it cannot be scalable. Better to probe k out of n nodes.
- **Fault Tolerance :** Algorithm should continue its working although one or more nodes in system crashes. It should consider available nodes in decision making.

4.3 Task Assignment Approach

- This approach finds an optimal assignment policy for the tasks of the individual process. Following are the assumptions in this approach.
  - Tasks of the process are less interdependent and data transfer among them will be minimized.
  - How much computation is required for task and speed of processor is already known.
  - Cost of processing needed by each task on every node is known.
  - Intercrosses communication (IPC) cost between each pair of tasks is known and it is zero if communicating tasks are running on the same node.
  - Precedence relationship among tasks, resources need of each task, resources available on each node is already known.
- Considering above assumptions, the main goal of assigning the tasks of the process to nodes in the manner to achieve following.
  - Reducing IPC cost
  - Quick turnaround time for complete process
  - A high degree of parallelism
  - Efficient utilization of system resources.
- It is not possible to achieve all these goals simultaneously as they conflicts each other. You can minimize IPC by keeping all tasks on same node. But for efficient utilization of resources, tasks must be evenly distributed among nodes. Precedence relationship among tasks also limits their parallel execution. If x number of tasks and y number of nodes then x^y are the possible assignments of tasks to nodes. In practice, certain tasks cannot be assigned to certain nodes due to some restrictions. Hence, possible assignments can be less than x^y.

- The total cost which is optimal not necessarily will be gained by assigning equal no of processes to each node. For two processors, optimal assignment can be done in polynomial time. But for arbitrary number of processors the cost is NP hard. Heuristics algorithm found to be efficient from computation point of view but produce sub optimal assignments.

4.4 Load-Balancing Approach

- Load balancing algorithms use this approach. It assumes that, equal load should be distributed among all nodes to ensure better utilization of resources. These algorithms transfers load from heavily loaded nodes to lightly loaded nodes in transparent way to balance the load in system. This is carried out to achieve good performance relative to parameters used to measure system performance.
- From user point of view, performance metric often considered is response time. From resource point of view, performance metric often considered is total system throughput. Throughput metric involves treating all users in the system fairly with making progress as well. As a result, all load balancing algorithms focuses on maximizing the system throughput.

4.4.1 Classification of Load Balancing Algorithms

- Basically, load balancing algorithms are mainly classified into two types as **static** and **dynamic** algorithms.
- Static algorithms are classified as **deterministic** and **probabilistic**.
- Dynamic algorithms are classified as **centralized** and **distributed** algorithms.
- Further distributed algorithms are classified as **cooperative** and **non-cooperative**.
- **Static** algorithms do not consider current state of system. Whereas, **dynamic** algorithms consider it. Hence, dynamic algorithms avoid giving response to system states which degrades system performance. However, dynamic algorithms need to collect information about current state of the system and they should give response about the same. As a result, the dynamic algorithms are more complex than static algorithms.
- **Deterministic** algorithms consider properties of nodes and characteristics of processes to be assigned to them to ensure optimized assignment, for example; task assignment approach. A **probabilistic algorithm uses** information such as network topology, number of nodes and processing capability of each node etc. Hence, such information helps to improve system performance.
- In **centralized** dynamic scheduling algorithm, scheduling responsibility is carried out by single node. In **distributed** dynamic scheduling algorithm, various nodes are involved in making scheduling decision, i.e. assignment of processes to processors. In **centralized** approach, system state information is collected at single node which is centralized server node. This node takes processes assignment decision based on collected system state information. All other nodes periodically send this information to this centralized server node. This is efficient approach to take scheduling decision as single node knows load on each node and number of processes needing service. Single point of failure is the limitation of this approach. Message traffic increases towards single node, hence it may become bottleneck.
- In **distributed** dynamic scheduling algorithm various nodes are involved in making scheduling decision and hence, it avoids bottleneck of collecting state information at single node. Each local controller on each node runs concurrently with others. They takes decision in coordination with other based on systemwide objective function instead of local one.
- In **cooperative** distributed scheduling algorithms, distributed entities makes scheduling decision by cooperating with each other. Whereas, **non-cooperative** distributed scheduling algorithms do not cooperate with each other for making scheduling decisions and each one acts as autonomous entity.

4.4.2 Issues in Designing Load-Balancing Algorithms

- Following are issues related to designing load-balancing algorithms.
  1. Policy for how to calculate workload of each node.
  2. Process transfer policy to decide execution of process locally or remotely.
  3. Selection of node to transfer selected process called as location policy.
  4. Policy for how to exchange system load information among load.
  5. Policy to assign priority of execution of local and remote process on particular node.
  6. Migration limiting policy to determine number of times process can migrate from one node to other.
- If process is processed at its originating node then it is **local** process. If process is processed on node which is not its originating node then it is **remote** process. If process arrived from other node and admitted for processing then it becomes local process on this current node. Otherwise it is sent to other nodes across the network. It then is considered as remote process at destination node.

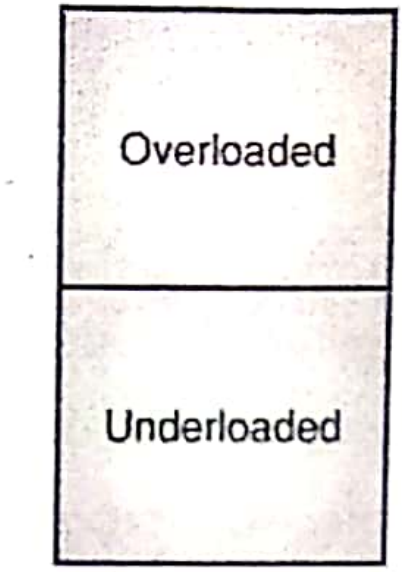
Load Estimation Policy

- Load-balancing algorithm should first estimate the workload of the particular node based on some measurable parameters. These parameters include factors which are time dependent and node-dependent. These factors are: Total number of processes present on node at the time of load estimation, resource demands of these processes, instruction mixes of these process and architecture and speed of the processors.
- As current state of the load is important, estimation of the load should be carried out in efficient manner. Total number of processes present on node is considered to be inappropriate measure for estimation of load. This is because actual load could vary depending on remaining service time for those processes. Again, how to measure remaining service time of all processes is the problem.
- Again both measures, total number of processes and total remaining service time of all processes are not suitable to measure load in modern distributed system. In modern distributed system, idle node may have several processes such as daemons, window managers. These daemons wakeup periodically and again sleeps.
- Therefore, CPU utilization is considered as best measure to estimate the load of the node. CPU utilization is number of CPU cycles actually executed per unit of real time. Timer is set up to observe CPU state time to time.

Process Transfer Policies

- In order to balance load in system, process is transferred from heavily loaded nodes to lightly loaded nodes. There should be some policy to decide whether node is heavily or lightly loaded. Most of the load-balancing algorithms use **threshold policy**.
- At each node, threshold value is used. Node accepts new process to run locally if its workload is below threshold value. Otherwise process is transferred to another lightly loaded node. Following policies are used to decide threshold value.
  - **Static policy :** Each node has predefined threshold value as per its processing capabilities. It remains fixed and does not vary with changing workload at local or remote node. No exchange of state information is needed in this policy.
  - **Dynamic policy :** In this policy, threshold value for each node is product of average workload of all the nodes and predefined constant for that node.
  - Predefined constant value c_k for node n_k is depends on processing capability of node n_k with respect to processing capability of all other nodes. Hence state information exchange between nodes is needed in this policy.

- Most of the load-balancing algorithms use **singl**
  Overloaded and underloaded. In this policy, if load
  and it accepts new local or remote processes for e
  the local processes and reject to accept remote pro



Thresh

Single-Threshold policy

(a)

- In single threshold policy, suppose load is just belo
  case, load becomes larger than threshold value as so
  remote process, node will start transferring its proce
- Therefore it is suggested that, node should transfe
  performance of its local processes. Node should
  processes do not much affect the service to local pro

- **Bidding :** In this policy, bidding process is carried out in which each node plays two roles; manager and contractor. Manager node has processes to send for execution and contractor is a node which is ready to accept processes for execution. Same node can play both the roles. Manager broadcasts request-for-bid message to all the nodes and contractor return bid with message containing memory size, processing capability, resource availability and other. Manager determines best bid upon receiving bid from contractor and sends process there. It may also happen that, contractor receives many bids from many managers and becomes overloaded. Solution to this problem is that, manager sends message to contractor before transferring the process. Contractor replies with message whether process will be accepted or rejected. In this policy, both managers and contractor have freedom to send and accept/reject processes respectively. Drawback is more communication overhead and difficult to decide best pricing policy.
- **Pairing :** In this policy, two nodes that greatly differs in their load are temporarily paired each other. Process transfer then is carried out from heavily loaded node to lightly loaded node. There may be many pairs established in the system and simultaneous transfer of processes is carried out among the pairs.

State Information Exchange Policies

- It is necessary to choose best policy to support dynamic load balancing algorithms and to reduce number of messages transmitted.
- Following policies are used to exchange state information.
- **Periodic Broadcast :** In this policy, each node broadcast its state information after elapse of every t units of time. It generates heavy network traffic and unnecessary messages get transferred although state is not changed. It has poor scalability.
- **Broadcast When State Changes :** In this method, node broadcast its state information only if its state changes. Node's state changes if it transfers its processes to other nodes or if it receives processes from other nodes. Further improvement to this method is to do not report small change in state to other nodes as nodes takes part in load balancing process if they are underloaded or overloaded. Refined method is node will broadcast its state information when it moves from normal load to underloaded or overloaded state.
- **On-Demand Exchange :** In this method, node broadcast its state information request when its state moves from normal load to either underloaded or overloaded region. Receivers of this request, replies their current state to requesting node. This method work with double-threshold policy. In order to reduce number of messages, further improvement is to avoid reply regarding current state from all the nodes. If requesting node is underloaded then only overloaded nodes can cooperate with it. If requesting node is overloaded then only underloaded nodes can cooperate with it in load-balancing process. As a result, in this improved policy requesting nodes identity is included in state information request message so that only nodes which can cooperate with requesting node sends reply. In this case, other nodes do not sends any reply to requesting nodes.
- **Exchange by polling :** This method do not use broadcast which limits the scaling. Instead, the node which needs cooperation from other nodes for balancing the load searches for suitable partner by polling other nodes one by one. Hence, exchange of state information takes place between polling node and polled node. Polling process stops either if appropriate node is located or predefined poll limit is attained.

Priority Assignment Policies

- Main issue in scheduling the local or remote processes at particular node in load balancing process is deciding the priority assignment rules. Following rules are used to assign priority to the processes at particular node.
  - **Selfish :** In this rule, local processes gets higher priority compared to remote processes.
  - **Altruistic (Unselfish) :** In this rule, remote processes are given higher priority compared to local processes.

- **Intermediate :** In this rule, priority is determined on the basis of number of local processes and number of remote processes at that particular node. This rule assigns higher priority to local processes if their total number is greater or equal to number of remote processes. If not, remote processes are assigned with higher priority.

Migration Limiting Policies

- Total number of times the process should migrate from one node to other is also important in balancing the node of system. Following two policies may be used.
- **Uncontrolled :** Process can be migrated any number of times from one node to other. This policy leads to instability.
- **Controlled :** As process migration is costlier operation, and hence, this policy limits migration count to 1. This means process should be migrated to only one node. For large size processes this count can be increased as process cannot finish execution in specified time on the same node.

4.5 Load-Sharing Approach

- In load-balancing approach, exchange of state information among nodes is required. This involves considerable overhead. Since resources are distributed, load cannot be equally distributed among all nodes. There should be a proper utilization of these resources. In large distributed system, number of processes in a node always fluctuating. Hence, temporal unbalancing of load among nodes always exists.
- Therefore, it is sufficient to keep all the nodes busy. It should ensure that, no node will remain idle while other nodes have several processes to execute. This is called as dynamic load sharing.

4.5.1 Issues in Designing Load-Sharing Algorithms

Load sharing algorithms aims at no node is idle while other nodes are overloaded. In load sharing algorithms, issues are similar to issues in load-balancing algorithms. It is simpler to decide about policies in load-sharing algorithms with compare to load-balancing algorithms.

Load Estimation Policies

- Load-sharing algorithms only ensure that no node is idle. Hence, simplest policy to estimate load is total number of processes on node.
- As in modern distributed system, several processes permanently resides on idle node, CPU utilization is considered as measure to estimate load on node

Process Transfer Policies

- Load-sharing algorithms mainly concern with two states of nodes: busy or idle. It uses single threshold policy with threshold value of 1. Node accepts process if it has no process. Node transfers the process as soon as it has more than two processes. If node becomes idle, it can be unable to accept new process immediately. This policy wastes the processing power of idle node.
- The solution to above problem is to transfer the process to the node which is expected to become idle. Therefore, some load-sharing algorithms use single threshold policy with threshold value of 2. If CPU utilization is considered as measure of load then double threshold policy should be used.

Location Policies

- In load-sharing algorithms, sender or receiver node is involved in load sharing process.

- **Sender Initiated Policy :** In this location policy, sender takes decision about transfer of process to particular node. In this policy, heavily loaded node search for lightly loaded nodes to transfer the process. When load of the node becomes greater than threshold value, it either broadcast or randomly probes the other nodes to find lightly loaded node which can accepts one or more of its processes.
- The good candidate node to accept the processes will be that node whose load would not exceed threshold value after accepting the processes for execution. In case of broadcast, sender node immediately knows about availability of suitable node to accept processes when it receives reply from all the receivers. Whereas, in random probing, probing continuous till appropriate receiver is found or number of probes reaches to static probe limit. If no suitable node is found then process is executed on its originating node. Probing method is more scalable than broadcast.
- **Receiver Initiated Policy :** In this location policy, receiver takes decision about from where to get the process. In this policy, lightly loaded node search for heavily loaded nodes to get load from there. When load of the node goes below the threshold value, it either broadcast or randomly probes the other nodes to find heavily loaded node which can transfer one or more its processes.
- The good candidate node to send its processes will be that node whose load would not reduce below threshold value after sending the processes for execution. In case of broadcast, sender node immediately knows about availability of suitable node to send processes to it when it receives reply from all the receivers. Whereas, in random probing, probing continuous till appropriate node is found from which processes can be obtained or number of probes reaches to static probe limit.
- In later case, node waits for fixed timeout period before trying again to initiate transfer. Probing method is more scalable than broadcast.
- Preemptive-process migration is costlier than non-preemptive as execution state needs to be transfer along with process. In non-preemptive process migration, process is transferred before it starts the execution on its current node. Receiver initiated process transfer is mostly preemptive. Sender initiated process transfer is either preemptive or non-preemptive.

State Information Exchange Policies

- In load-sharing approach state information exchange is carried out only when node changes the state. Because, node needs to know state of other node when it becomes overloaded or underloaded. In this case following policies are used.
- **Broadcast When State Changes :** In this policy, node broadcast state information request when its state changes. In sender initiated policy, this message broadcast is carried out when node becomes overloaded and in receiver initiated policy, this message broadcast is carried out when node becomes underloaded.
- **Poll When State Changes :** Polling is appropriate method compared to broadcast in large networks. In this policy, upon change in state, node polls other nodes one by one and exchanges its changed state information with them. Exchanging the state information continuous till appropriate node for sharing load is found or number of probes reaches to probe limit. In sender initiated policy, probing is carried by overloaded node and in receiver initiated policy, probing is carried out by underloaded node.

4.6 Introduction to Process Management

- Process management includes different policies and mechanisms to share processors among all the processes in system.

- Process management should ensure best utilization of processors by allocating them to different processes in system. Process migration deals with transfer of process from its current location to the node or processor to which it has been assigned.

4.7 Process Migration

- Process migration is transfer of process from its current node which is source node to destination node. There are two types of process migration. These are given below.
  - **Non-preemptive process migration :** In this approach, process is migrated from its current to destination node before it starts executing on current node.
  - **Preemptive process migration :** In this approach, process is migrated from its current to destination node during its course of execution on current node. It is costlier than non-preemptive process migration as process environment also needs to transfer along with process to destination node.
- Migration policy includes selection of process to be migrated and selection of destination node to which process should be migrated. This is already described in resource management. Process migration describes actual transfer of process to destination node and mechanism to transfer it.
- **Freezing time** is the time period for execution of process is stopped for transferring its information to destination node.

4.7.1 Desirable Features of Good Process Migration Mechanism

Following are desirable features of good process migration mechanism.

- **Transparency :** Following level of transparency is found.
  - **Object Access Level :** It is minimum requirement for system to support non-preemptive process migration. Access to objects like file and devices should be location independent. This allows to initiation of program on any node. To support transparency at object access level, system should support for transparent object naming and locating.
  - **System Calls and IPC :** System calls and interprocess communication should be location independent. Migrated process should not depend on its originating node. For preemptive process migration, system should support this transparency. Process should receive its messages from other processes directly on recently migrated node and not through its originating node.
- **Minimal Interference :** Migration of process should not affect the progress of process and system as a whole. This can be achieved by minimizing the freezing time of process being migrated.
- **Minimal Residual Dependency :** Migrated process should not continue to depend on its previous node once it has started its execution on its new node. Otherwise, failure of previous node will cause the process to fail. Also, it imposes load on its previous node.
- **Efficiency :** Process migration will be inefficient if it takes more time for migrating the process, more cost of locating the object and involves more cost to support remote execution once process is migrated.
- **Robustness :** Failure of other node (excluding current node on which process is running) should not affect accessibility and execution of that process.
- **Communication between Coprocesses of a Job :** If processes of the single job are distributed over several nodes for parallel execution then these processes should be able to directly interact with each other irrespective of their location.

4.7.2 Process Migration Mechanisms

Following activities are involved in process migration.

1. Freezing the process at source node and restarting its execution on its destination node.
2. Transferring the process's address space from its source node to its destination node.
3. Forwarding all the messages for process on its destination node.
4. Handling communication between cooperating processes placed on different nodes due to process migration.

Mechanism for Freezing and Restarting the Process

- In preemptive process migration, usually snapshot of the process state is taken at its source node and then this snapshot is reinstated at its destination node. First process execution is suspended at its source node. Its state information is then transferred at destination node. After this, its execution is restarted at its destination node.
- Freezing the processes at source node means its execution is suspended and all external interaction with the process is postponed.
- **Immediate and Delayed Blocking of the Process :** In preemptive process migration, execution of the process must be blocked at its source node. This blocking can be carried out immediately or after process reaches the state when it can be blocked. Hence, when to block the process depends on its current state. If process currently not executing system call then it can be immediately blocked. If process currently executing system call but is sleeping at interruptible priority waiting for occurring of kernel event then it can be immediately blocked. If process currently executing system call but is sleeping at non-interruptible priority waiting for occurring of kernel event then it cannot be blocked immediately. The mechanism of blocking can vary as per implementation.
- **Fast and Slow I/O Operations :** Allow completing all fast I/O operations such as disk I/O, before freezing the process at source node. It is not feasible to wait for slow I/O operations such as those on pipes and terminal.
- **Information about Open Files :** State information of process also contains information about files currently open by process. It includes name or identifier of the files, access modes, current position of their file pointers etc. In distributed system as network transparent execution environment is supported, this state information can be collected. Same protocol is used to access local or remote file. The UNIX based system recognizes the file by its full pathname. Operating system returns file descriptor to process once files is opened by it. It then uses file descriptor to complete I/O operations on file. In such system, pointer to file must be preserved so that migrated process can keep on accessing the file.
- **Reinstating Process on its Destination Node :** New empty process is created on destination node which is same as that allocated during process creation. Identifier of this newly allocated process can be same or different depending on implementation. If it is different, then it is changed to the original identifier in a subsequent step before process starts executing on destination node. Operations on both the processes are suspended. Hence, rest of the system cannot notice existence of two copies. When entire state of migrating process is transferred and copied in empty process, the new process is unfrozen and old is deleted. Thus process restarts execution from the state it has before being migrated. Migration mechanism should allow performing system call at destination node which was not completed due to freezing of the process at source node.

Address Space Transfer Mechanism

- Following information is transferred when process is migrated from source node to destination node.
  - **Process State :** It includes execution status i.e. content of registers, scheduling information, memory used by process, I/O states, and its rights to access the objects which is capability list, process's identifier, user and group identifier of the process, information about file opened by process.
  - **Process Address Space :** It includes code, data and stack of the program.
- Size of process state information is in few kilobytes. Whereas, process address space is of large size. Process address space can be transferred without stopping its execution on source node. However, while transferring state information, it is necessary to stop execution of the process.
- To start execution at destination node, state information is needed at destination node. Address space can be transferred before or after process starts execution at destination node.
- Address space can be transferred in following manners.

Total Freezing

- In this method, process is not allowed to continue its execution when decision of its transfer to destination is taken. The execution of the process is stopped while its address space is being transferred. This method is simple and easy to implement.
- The limitations of this method is that if process that is to be transferred is suspended for longer period of time during process of migration then timeouts may occur. Also user may observe delay if the process is interactive.

Pretransferring

- In this method, process is allowed to run on its source node until its address space has been transferred on destination node. The modified pages at source node during transfer of address space also are retransferred to the destination followed by previous transfer. This retransfer of modified pages is carried out repeatedly until number of modified pages is relatively small. These remaining modified pages are transferred after the process is frozen for transferring its state information.
- In first pretransfer operation, as first transfer of address space takes longest time, a longest time is offered accordingly to carry out modifications related to changed pages. Obviously, second transfer will take short time for the same as only modified pages during first transfer is transferred. These modifications will be lesser compared to first transfer. Subsequently, fewer pages will be transferred in next transfer to destination node till number of pages converges to zero or very few.
- These remaining pages are then transferred from source node to destination node when process is frozen. At source node, pretransfer operation gets higher priority with respect to other programs. In this method, frozen time is reduced and hence, leads to minimal interfere of migration with interactions of process to other processes and users. As same pages can be transferred repetitively due to modifications, the total time elapsed in migration can increase.

Transfer on Reference

- In this method, entire address space of process is kept on its source node and only needed address space for execution is requested from destination node during execution (just like relocated process executes at destination node) . The page is transferred from source node on its reference at destination node. This is just like demand driven copy on reference approach.
- This method is more dependent on source node and imposes burden on it. Moreover, if source node fails or reboots the process execution will fail.

Message-Forwarding Mechanism

- It should be ensured that, the migrated process should receive all its pending, messages in transmission (en-route messages) and future messages at destination node. Following types of messages needs to be forwarded at destination node of migrated process.

Type 1 : Messages which are received at source node after process's execution has been stopped and its execution has not until now been commenced at destination node.

Type 2 : Messages received at source node after process's execution started at destination node.

Type 3 : Messages expected by already migrated process after it starts its execution on destination node.

Mechanism of Resending the Message

- This mechanism is used in V-System and amoeba distributed OS to handle above all types of messages. In type 1 and 2 case, messages are simply returned to sender as they are not deliverable or simply dropped with assumption that, sender maintained the copy of them and able to retransmit it. Messages of type 3 are directly sent to the destination node of the process.
- In this method, there is no need to maintain the process state at source node. But, message forwarding mechanism of process migration operation is not transparent to the other processes communicating with the migrant process.

Original Site Mechanism

- In this method, the process identifier has its home node embedded in it and every node is responsible to maintain information about current location of all the processes created on it. Hence, current location of the process can be obtained from its home node. Any process now first forwards the message to home node which then forwards message to process's current node.
- The drawback of this method is that, it imposes continuous load on process's home node although process is migrated to other node. Also, failure of home node disturbs the message forwarding mechanism.

Link Traversal Mechanism

- In this method, message queue of the migrant process is created on its source node. All the type 1 messages are placed in this queue. After knowing that the process is established at destination node, all the messages in message queue are forwarded to the destination node.
- For Type 2 and 3 messages, a link pointing to destination node of the migrant process is maintained at source node. This link is address for forwarding processes. This link consists of two components: a system wide unique process identifier which is identifier of the originating node of the process and other is unique local identifier which is a last known location of a process. First component remains same during lifetime of the process. Whereas second component may change. A series of links, starting from node where process is created, is traversed in order to forward Type 2 and 3 messages. This traversal stops at process's current node and second component of the link is now updated when process is accessed from the node. This improves efficiency next time when process is located from the same node.
- The drawback of this method is that, failure of any node in the chain fails then process cannot be located. It has poor efficiency as many links needs to be traversed to locate the process.

Mechanism for Handling Coprocesses

There should be efficient communication between parent and child processes if they are migrated and placed at different nodes.

Not Allowing Separation of Processes

- Processes that wait for one or more of their children to complete are not allowed to migrate. It is used by UNIX based networked system. The drawback of this approach is that, it does not permit use of parallelism within job. It is due restricting the various tasks of job to distribute on different nodes.
- If parent process migrates then its children processes will be migrated along with it. It is used by V-System. The drawback of this approach is that, it involves more overhead in migrating the process if logical host comprising the address space of V-System and their associated processes is large. It means logical host comprises several host processes.

Home Node in Original Sight Concept

- In this method, process and its children are allowed to migrate independently on different nodes. All the communications between parent process and its children takes place via home node.
- Drawback is that message traffic and communication cost increases.

4.7.3 Process Migration in Heterogeneous System

- In heterogeneous environment, it is necessary to translate data format when process is migrated to destination node. CPU format at source node and at destination node can be different. If n numbers of CPUs are present in the system then n(n–1) pieces of translation software is required.
- Solution to reduce this software complexity is to use external data representation in which each processor has to convert data to and from the standard form. Hence, if 4 processors are present then only 8 conversions are required. Two conversions for each processor, where one is to convert data from CPU format to external format and other in reverse order (external format to CPU format).
- In this method, translation of floating point numbers which consist of exponent, mantissa and sign needs to be handled carefully. In handling of exponent example suppose processor A uses 8 bit, B uses 16 bits and external data representation is designed for processor A offers 12 bits. Hence, process can be migrated from A to B which involves conversion of 8 bits to 12 bits and a12 bits to 16 bits for processor B. Process requiring exponent data more than 12 bits cannot be migrated from processor B to A. Therefore, external data representation should be designed with number of bits at least as longest exponent of any processor in the system.
- Migration from processor B to processor A also not possible if exponent is less than or equal to 12 bits but greater than 8 bits. In this case, although external data representation has sufficient number of bits but processor A does not.
- Suppose processor A uses 32 bits, B uses 64 bits and external data representation uses 48 bits. In this case, handling mantissa is same as handling exponent. Only transferring process from B to A will result in computation with half precision. This may not be acceptable when accuracy of result is important. Second problem may occur due to loss of precision due to multiple migrations between set of processors. Solution is to design external data representation with longest mantissa of any processor. External data representation also should take care of signed-infinity and signed-zero.

4.7.4 Advantages of Process Migration

Following are the advantages of process migration.

- **Reducing average response time of processes :** The response time of processes increases as load on the node increases. In process migration, as processes are moved from heavily loaded node to idle or lightly loaded node, the average response time of processes reduces.

- **Speeding up individual job :** The tasks of the job can be migrated to different node. Also job can be migrated to node having faster CPU. In both cases, execution of the job speeds up.
- **Gaining higher throughput :** As all the CPUs are better utilized with proper load balancing policy, higher throughput can be achieved. By properly mixing CPU and I/O bound jobs globally, throughput can be increased.
- **Effective utilization of resources :** In distributed system, hardware and software resources are distributed on many nodes. These resources can be effectively used by migrating processes to the nodes as per their resource requirements. Moreover, if processes require special-purpose hardware then they can be migrated to node having such hardware facility.
- **Reducing network traffic :** Migrating processes closer to the resources or to the node having their required resources, reduces network traffic as resources are accessed locally. Instead of accessing huge database remotely, it is better to move the process to the node where such database is present.
- **Improving system reliability :** Critical process can be migrated to node with higher reliability in order to improve reliability.
- **Improving system security :** This can be achieved by migrating sensitive process to more secure node.

4.8 Threads

- Threads improve performance of the application. In operating system supporting thread facility, basic unit of CPU utilization is threads. A thread is a single sequence stream within a process. Threads are also called as lightweight processes as it possess some of the properties of processes. Each thread belongs to exactly one process.
- In operating system that support multithreading, process can consist of many threads. These threads run in parallel improving the application performance. Each such thread has its own CPU state and stack, but they share the address space of the process and the environment.
- Threads can share common data so they do not need to use interprocess communication. Like the processes, threads also have states like ready, executing, blocked etc. priority can be assigned to the threads just like process and highest priority thread is scheduled first.
- Each thread has its own Thread Control Block (TCB). Like process context switch occurs for the thread and register contents are saved in (TCB). As threads share the same address space and resources, synchronization is also required for the various activities of the thread.

4.8.1 Comparison between Process and Thread

- Threads are more advantageous with compare to process. Following table shows comparison between thread and process.

Sr. No.	Process	Thread
1.	Program in execution called as process. It is heavy weight process.	Thread is part of process. It is also called as light weight process
2.	Process context switch takes more time as compared to thread context switch because it needs interface of operating system.	Thread context switch takes less time as compared to process context switch because it needs only interrupt to kernel only.
3.	New Process creation takes more time as compared to new thread creation.	New thread creation takes less time as compared to new process creation.

Sr. No.	Process	Thread
4.	New Process termination takes more time as compared to new thread termination.	New thread termination takes less time as compared to new process termination.
5.	Each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
6.	In process based implementation, if one process is blocked no other server process can execute until the first process unblocked.	In multithreaded server implementation, if one thread is blocked and waiting, second thread in the same process could execute.
7.	Multiple redundant processes use more resources than multiple threaded process.	Multiple threaded processes use fewer resources than multiple redundant processes.
8.	Context switch flushes the MMU (TLB) registers as address space of process changes.	No need to flush TLB as address space remains same after context switch because threads belong to the same process.

4.8.2 Server Process

Following models can be used to construct server process. Consider the example of file server.

- **As a Single Threaded process :** In this model client requests are served sequentially. There is no parallelism. The model uses blocking system call. File server gets client's file access request from request queue. If access is permitted then server checks if disk access is needed. If no, request is served immediately and reply is sent to client process. If disk access is needed then file server sends disk access request to disk server and waits till reply arrives. After getting disk server's reply, it services client request and sends reply to client process. Server then goes back to handle next request. Hence, performance server is unacceptable.
- **As a Finite-State Machine :** It supports parallelism with non-blocking call. Event queue is maintained to store disk server's reply messages and client's requests. If server remains idle, it takes next request from event queue. If it is client's request and if access is permitted then server checks if disk access is needed. If disk access is needed then file server sends disk access request to disk server. At this point, instead of blocking, it saves current state of client's request in table and goes to next request in event queue. This message may be new client's request or reply of previous request from disk server. If it is new client's request then it is handled in above manner and if it is reply from disk server then it retrieves current state of client's request in table. It the further processes this client request with retrieved client's current state.
- **As a Group of Threads :** This model supports parallelism with blocking system calls. In this model, server comprises one dispatcher thread and several worker threads. Worker threads can be created dynamically to handle the requests or pool of threads can be created at start up time. The dispatcher thread keeps waiting for the request. if request arrives and access is permitted then it creates new thread or chooses idle thread from available pool to handover this request. Now dispatcher changes state from running to ready. Worker thread checks if disk access is needed. It first checks if request is satisfied from block cache shared by threads. If not, then it sends disk access request to disk server and blocks waiting for reply from disk server. Now scheduler may chose another thread either dispatcher or worker which are ready to run.

4.8.3 Models for Organizing Threads

Following are the three different ways to organize the threads :

- **Dispatcher-worker Model :** In this model, process consist of one dispatcher thread and several worker threads. The dispatcher thread accepts the client's request chooses one of the free worker thread to handover this request for further processing. Hence, multiple client's request can be processed by multiple worker threads in parallel.
- **Team Model :** In this model, all thread are equal level. Each thread accepts client's request and process it on its own. A specific type of thread to handle specific requests can be created. Hence, parallel execution of different client's requests can be carried out.
- **Pipeline Model :** This model is useful for the applications which are based on producer consumer model. In this model, output of the first thread is given as input to second thread for processing, the output of the second thread is given as input to third thread for processing and so on. Threads are arranged in pipeline. In this way, output of the last thread is final output of the process to which threads belongs.

4.8.4 Issues in Designing a Thread Package

Following are the issues related designing a thread package. System should supports primitives for thread related operations.

Thread Creation

- Threads can be created statically in which number of threads remains fixed till lifetime of the process. In dynamic creation of threads, initially process is started with single thread. New threads are created when needed during the execution of process. A thread may destroy itself after finishing of its job by using exit system call.
- In static approach, number of threads is decided while writing or compiling the program. In this approach, a fixed stack is allocated to the thread. In dynamic approach, stack size is specified through parameter to the system call.

Thread Termination

- A thread may destroy itself after finishing of its job by using exit system call. A kill command is used to kill thread from outside by specifying thread identifier as parameter. In many cases, threads are never killed until process terminates.

Thread Synchronization

- The portion of the code where thread may be accessing some shared variable is referred as the Critical region (CR). It is necessary to prevent multiple threads accessing same data. For this purpose ***mutex*** variable is used. Thread that wants to execute in CR performs lock operation on corresponding mutex variable. In single atomic operation, state of mutex changes from unlocked to locked state.
- If mutex variable is already in locked state then thread is blocked and waits in queue of waiting threads on the mutex variable. Otherwise thread carries out other job but continuously retries to lock the mutex variable. In multiprocessor system where threads run in parallel, two threads may carry out lock operation on same mutex variable. In this case, one thread waits and other wins. To exit the CR, thread carries out unlock operation on mutex variable.
- Condition variables are used for more general synchronization. Operations wait() and signal() are provided for conditional variable. When thread carries out wait operation on condition variable the associated mutex variable is unlocked and thread is blocked till signal operation is carried out by other thread on the condition variable. When thread carries out signal operation on condition variable the mutex variable is locked and blocked thread on condition variable starts execution.

4.8.5 Thread Scheduling

Thread packages supports following scheduling features.

- **Priority Assignment Policy :** Threads are scheduled with simple FIFO or in round-robin (RR) basis. Application programmers can assign priority to threads so that important threads should get highest priority. This priority based scheduling can be non-preemptive or preemptive.
- **Flexibility to vary quantum size dynamically :** In multiprocessor system, if there are fewer runnable threads than available processors then it is wasteful to interrupt the running thread. Hence, instead of using fixed length quantum, variable quantum is used. In this case, size of time quantum is inversely with number of threads in the system. It gives good response time.
- **Handoff Scheduling :** Currently running thread may name its successor to run. It may give up processor and request the successor to run next, bypassing the queue of runnable threads. It enhances performance if cleverly used.
- **Affinity Scheduling :** In this scheduling in multiprocessor system, a thread is scheduled on processor on which it was last run, expecting that its address space is still in that processor cache.

4.8.6 Implementing a Thread Package

Thread Package can be implemented either in user space or kernel.

Implementing Threads in User Space

User Level Threads

- In user level implementation, kernel is unaware of the thread. In this case, thread package entirely put in user space. Java language supports threading package.
- User can implement the multithreaded application in java language. Kernel treats this application as a single threaded application. In a user level implementation, all of the work of thread management is done by the thread package.
- Thread management includes creation and termination of thread, messages and data passing between the threads, scheduling thread for execution, thread synchronization and after context switch saving and restoring thread context etc.
- Creation and destroying of thread requires less time and is chip operation. It is the cost of allocating memory to set up a thread stack and deallocating the memory while destroying the thread.
- Both operations requires less time. Since threads belong to the same process, no need to flush TLB as address space remains same after context switch. User-level threads requires extremely low overhead, and can achieve high computational performance. Fig. 4.8.1 shows the user level threads.

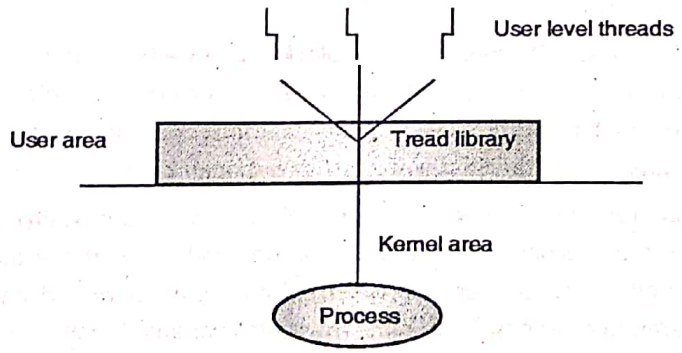


Fig. 4.8.1 : User level threads

Advantages of user level threads

- Thread switching does not require flushing of TLB and doing CPU accounting. Only value of CPU register need to be stored and reloaded again.
- User level threads are platform independent. They can execute on any operating system.
- Scheduling can be as per need of application.
- Thread management are at user level and done by thread library. so kernels burden is taken by threading package. Kernels time is saved for other activities

Disadvantages of user level threads

- If one thread is blocked on I/O, entire process gets blocked.
- The applications where after blocking of one thread other requires to run in parallel, user level threads are of no use.

Implementing Threads in Kernel

Kernel Level Threads

- In this, threads are implemented in operating system's kernel. The thread management is carried out by kernel.
- All these thread management activities are carried out in kernel space. So thread context and process context switching becomes same.
- Application can be written as multithreaded and threads of the application are supported as threads in single process.
- Kernel threads are generally requires more time to create and manage than the user threads.

Advantages of kernel level threads

- The kernel can schedule another thread of the same process even though one thread in a process is blocked. Blocking of one thread does not block the entire process.
- Kernel can simultaneously schedule multiple threads from the same process on multiple processes.

Disadvantages of kernel level threads

- Context switch requires kernel intervention.
- Kernel threads are generally requires more time to create and manage than the user threads.

Hybrid Implementation

- Considering the advantages of user level and kernel level threads, a hybrid threading model using both types of threads can be implemented.
- The Solaris operating system supports this hybrid model. In this implementation, all the thread management functions are carried out by user level thread package at user space. So operations on thread do not require kernel intervention.
- The advantage of hybrid model of the threads is that, if the applications are multithreaded then it can take advantage of multiple CPUs if they are available.
- Other threads can continue to make progress even if the kernel blocks one thread in a system function.

4.9 Virtualization

- From availability and security point of view, using separate computer to put each service is more preferable by organizations. If one server fails, other will not be affected.

- It is also beneficial in case if organizations need to use different types of operating system. However, keeping separate machines for each service is costly. Virtual machine technology can solve this problem. Although this technology seems to be modern but idea behind it is old.
- The VMM (Virtual Machine Monitor) creates the illusion of multiple (virtual) machines on the same physical hardware. A VMM is also called as a hypervisor. Using virtual machine, it is possible to run legacy applications on operating systems no longer supported or which do not work on current hardware.
- Virtual machines permits to run at the same time applications that use different operating systems. Several operating systems can run on single machine without installing them in separate partitions. Virtualization technology plays major role in cloud computing.

4.9.1 The Role of Virtualization in Distributed Systems

- Practically, every machine offers a programming interface to higher level software, as shown in Fig. 4.9.1. There are many different types of interfaces, for example, instruction set as offered by a CPU, and as other example, huge collection of application programming interfaces that are shipped with many present middleware systems.

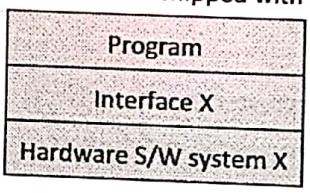


Fig. 4.9.1 : Program, Interface and System

- Virtualization makes it possible to extend or replace an existing interface so as to imitate the activities of another system, as shown in Fig. 4.9.2.

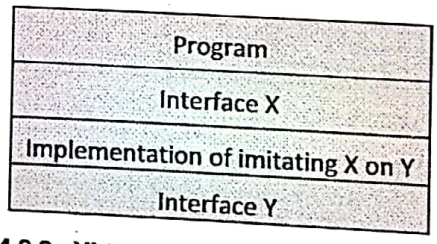


Fig. 4.9.2 : Virtualizing of system X on top of Y

- With advancement in technology, hardware and low-level system software change reasonably fast compared to applications and middleware which are at higher level. Hence, it is not possible to maintain pace of legacy software with the platforms it relies on. With virtualization, legacy interfaces can be ported to new platforms and hence, the latter can be opened for large classes of existing programs.
- In large distributed system, environment is heterogeneous. It comprises of heterogeneous collection of servers running different applications which are accessed by clients. Application should also use various resources easily and efficiently. In this case, virtualization plays major role.
- Each application can run on its own virtual machine, perhaps with the related libraries *and* operating system, which, in turn, run on a common platform. In this way, the diversity of platforms and machines can be minimized and high degree of portability and flexibility can be achieved.

4.9.2 Architectures of Virtual Machines

- Four types of interfaces are offered by any computer system. These are : An interface between the hardware and software, consisting of machine instructions that can be invoked by any program, interface between the hardware and software consists of machine instructions that can be invoked by OS only, an interface consisting of system calls and application programming interface (API).

- In first technique of virtualization, we can build a runtime system that basically offers an abstract instruction set that is to be used for executing applications. Instructions can be interpreted, but could also be emulated as is done for running Windows applications on UNIX platforms.
- In other technique of virtualization, a system that is basically implemented as a layer completely shielded the original hardware is provided and offered the complete instruction set of that same or other hardware as an interface. This interface can be offered *at the same time* to different programs. Hence, it is now possible to have multiple, and different operating systems run independently and in parallel on the same platform.

4.10 Clients

4.10.1 Networked User Interfaces

Client contact the server to access remote service. For this purpose, client machine may have separate counterpart that can contact the service over network. Alternatively, a convenient user interface can be offered to access service. Client need not have local storage and has only terminal. In the case of networked user interfaces, entire processing and storage is carried out at server.

Example : The X Window System

- It is used to control bit-mapped terminals, which include a monitor, keyboard, and a pointing device such as a mouse. It is just like component of an operating system that controls the terminal. The X kernel comprises all the terminal-specific device drivers. It is usually highly hardware dependent. The X kernel offers a relatively low-level interface for controlling the screen, but also for capturing events from the keyboard and mouse.
- This interface is made available to applications as a library called Xlib. The X kernel and the X applications may reside on the same or different machine. Particularly, X offers the X protocol, which is an application-level communication protocol by which an instance of Xlib can exchange data and events with the X kernel.
- As an example, Xlib can request to X kernel to create or kill a window, for setting colors, and defining the type of cursor to display, etc. The X kernel will react to local events such as keyboard and mouse input by sending event packets back to Xlib. Several applications can communicate simultaneously with the X kernel. Window manager application has special rights. This application can dictate the "look and feel" of the display as it appears to the user.

4.10.2 Client-Side Software for Distribution Transparency

- In client server computing, some part of processing and data level is executed at client-side as well. A special class is formed by embedded client software, such as for automatic teller machines, cash registers, barcode readers, TV set-top boxes, etc. In these cases, the user interface is a relatively small part of the client software, in contrast to the local processing and communication facilities.
- Client software also includes components for achieving distribution transparency. A client should not be aware that it is communicating with remote processes. On the contrary, distribution is often less transparent to servers for reasons of performance and correctness.
- As, at client side offers the same interfaces which are available at server, access transparency is achieved by client stub. The stub offers the same interface as available at the server. It hides the possible differences in machine architectures, as well as the actual communication.
- The location, migration, and relocation transparency is handled with different manner. Client request is always forwarded to all copy of server replica. Client-side software can transparently collect all responses and pass a single response to the client application. Failure transparency is also handled by client. Middleware carries out masking of communication failure. Client can repetitively attempt to connect the same server or other server.

4.11 Servers

4.11.1 General Design Issues

- Server takes request sent by client, processes it and sends response to client. Iterative server itself handles request and sends response back to client. Concurrent server passes incoming request to thread and immediately waits for another request. Multithreaded server is example of concurrent server.
- Server listen client request at endpoint. Client sends request to this end point called as port. These end points are globally assigned for well-known services. For example, servers that handle Internet FTP requests always listen to TCP port 21. Some services uses dynamically assigned endpoint. A time-of-day server may use an end point that is dynamically assigned to it by local OS.
- In this case, a client has to search the end point. For this purpose, a special daemon process is maintained on each server machine. The daemon keeps track of the current endpoint of each service implemented by a co-located server. The daemon itself listens to a well-known end point. A client will first contact the daemon, request the end point, and then contact the specific server.
- Another issue is how to interrupt the server while operation is going on. In this case, user suddenly exits the client application, immediately restart it, and pretend nothing happened. The server will finally tear down the old connection, thinking the client has most likely crashed. Other technique to handle communication interrupts is to design the client and server such that it is possible to send out-of-band data, which is data that is to be processed by the server before any other data from that client.
- One solution is to let the server listen to a separate control end point to which the client sends out-of-band data, while at the same time listening (with a lower priority) to the end point through which the normal data passes. Another solution is to send out-of-band data across the same connection through which the client is sending the original request.
- Other important issue is whether server should be stateless or stateful. A stateless server does not keep information on the state of its clients, and can change its own state without having to inform any client. Consider the example of file server. Client access the remote files from server. If server keeps track about each file being accessed by each client then the service is called stateful. If server is simply provides requested blocks of data to client and does not keep track about how client makes use of them then service is called stateless.

Stateful File Service

- First client must carry out open( ) operation on file prior to accessing it. Server the access information about file from disk and store in main memory.
- Server then gives unique connection identifier to client and opens the file for client. This identifier is used by client to access the file throughout the session.
- On closing of the file, or by garbage collection mechanism, the server should free the main memory space used by client when it is no longer active. The information maintained by server regarding client is used in fault tolerance. AFS uses stateful approach.

Stateless File Service

- In this case server does not keep any information in main memory about opened files. Here, each request identifies the file. There is no need to open and close the file with operations open( ) and close( ).
- Each file operation in this case is not a part of session but it is on its own separately. Closing the file at last is also a remote operation. NFS is example of stateless file service approach.

- Stateful service gives more performance. As connection identifier is used to access information maintained in main memory, disk accesses are reduced. Moreover, as server knows that file is open for sequential access, then read ahead next block improves the performance. In stateful case, if server crashes then recovery of volatile state information is complex. A dialog with client is used by recovery protocol. Server also needs to know about crash of client to free the memory space allocated to it. All the operation underway during crash should be aborted.
- In case of stateless service, the effect of server crash and recovery is invisible. Client keeps retransmitting the requests if server crash and no response from it.

4.11.2 Server Clusters

- Server cluster is collection of server machines that are connected through network. Consider these machines are connected through LANs which often offers high bandwidth. This server cluster logically organized in three tiers as shown in Fig. 4.11.1.
  - **Switch :** Client requests are routed through it.
  - **Application/Compute Servers :** These are high performance servers or application servers.
  - **Data Processing Servers :** These are file or database servers.
- The server cluster offering multiple services can have different machines running different application servers. Therefore, the switch should distinguish services. Otherwise it will not be possible to forward requests to the proper machines.

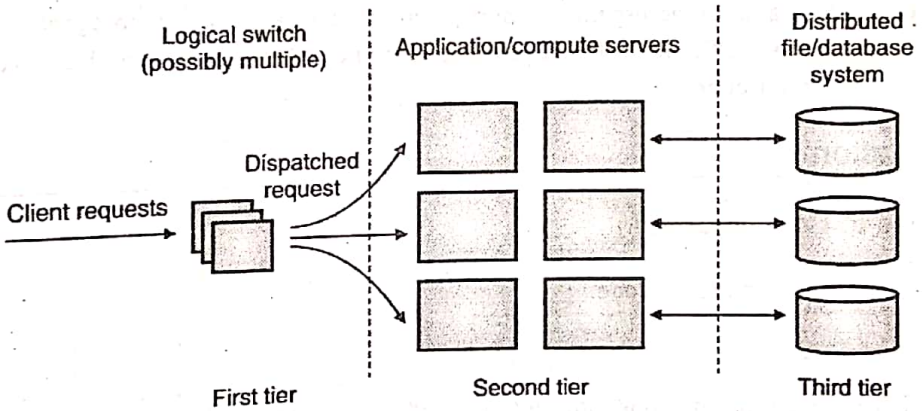


Fig. 4.11.1 : Logical organization of server cluster

- In server cluster, some of the servers may remain idle as it may be running single service. Hence, it is advantageous to temporarily migrate the services to idle machines. Client applications should always remain unaware about the internal organization of cluster.
- In order to access cluster, a TCP connection is set up over which application-level requests are sent as part of a session. A session ends by terminating the connection. If transport-layer switches are used, then switch accepts incoming TCP connection requests, and hands off such connections to one of the servers. In coming request comes to single address of switch which then forwards it to appropriate server.
- Server's ACK message to client contains IP address of switch as a source address. Switch follows the round robin policy to distribute load among servers. More advanced server selection criteria can be deployed as well.

4.11.3 Distributed Servers

- Using single switch (access point) to forward clients request to multiple servers may cause single point of failure. The cluster may become unavailable. As a result of this, several access points can be used, of which the addresses are made publicly available.
- A distributed server is dynamically varying collection of machines, with also possibly varying access points which appears to the outside world as a single powerful machine. With such a distributed server, the clients benefit from a robust, high-performing, stable server. The available networking services, particularly mobility support for IP version 6 (MIPv6) is used. In MIPv6, a mobile node normally resides in home network with its home address (HoA) which is stable.
- When mobile node changes network, it receives care-of-address in foreign network. This care-of address is reported to the node's home agent (router attached to home network) which forwards traffic to the mobile node. All applications communicate with mobile node using home address, they never see care of address.
- This concept can be used for stable address of distributed server. In this case, a single unique contact address is initially assigned to the server cluster. The contact address will be the server's life-time address to be used in all communication with the outside world. At any time, one node in the distributed server will work as an access point using that contact address, but this role can easily be taken over by another node.
- What happens is that the access point records its own address as the care-of address at the home agent associated with the distributed server. At that point, all traffic will be directed to the access point, which will then take care in distributing requests among the currently participating nodes. If the access point fails, a simple fail-over mechanism comes into place by which another access point reports a new care-of address. Home agent and access point may become bottleneck as whole traffic would flow these two machines. This situation can be avoided by using MIPv6 feature known as route optimization.

4.12 Code Migration

4.12.1 Approaches to Code Migration

Let us discuss about why to migrate the code.

Reasons for Migrating Code

- Performance is improved by moving code from heavily loaded machines to lightly loaded machine, although it is costlier operation. If server manages huge database and client application perform operations on this database that involves large quantity of data. In this case, it is better to transfer part of client application to server machine. In other case, instead of sending requests for data validation to server it is better to carry out client side validation. Hence, to transfer part of server application to client machine.
- Code migration also helps to improve performance by exploiting parallelism where there is no need to consider details of parallel programming. Consider the example of searching for information in the Web. A search query which is small mobile program (mobile agent) moves from site to site. Each copy of this mobile agent can be transferred to different sites to achieve a linear speedup compared to using just a single program instance. Code migration also helps to configure distributed system dynamically.
- In other case, no need to keep all software at client. Whenever client binds with server, it can dynamically download the required software from web server. Once work is done, all these software can be discarded. No need to preinstall

Models for Code Migration

- In first model, program is comprises of code segment, resource segment and execution segment. Code segment contains all the instructions of program. Resource segment includes references to external resources needed by the process, such as files, printers, devices, other processes, and so on. Execution state includes current execution state that contains private data, the stack, and program counter.
- In weak mobility, only code segment with initialization data is transferred. For example, Java applets always start execution from initial state. On the other hand, in strong mobility, execution segment is needed to resume execution on target node as execution of processes is stopped at source node and resumes at target node.
- Migration of code can be sender-initiated or receiver-initiated. In sender initiated migration, migration initiates from the node (source node) where the code is currently running. In receiver-initiated migration, target machine takes initiative for code migration. Receiver-initiated migration is simpler than sender-initiated migration. Some time client initiate migration and in the same way server initiates migration.
- In weak mobility, migrated code either is executed by the target process or a separate process needs to be started. For example, downloaded Java applets execute in the browser's address space. It avoids staring a separate process and hence, avoids communication at the target machine. The main drawback is that the target process needs to be protected against malicious or unintentional code executions.
- A simple solution is to let the operating system take care of that by creating a separate process to execute the migrated code. Migration by cloning is a simple way to improve distribution transparency in which process creates child processes.
- Following are the alternatives for code migration.

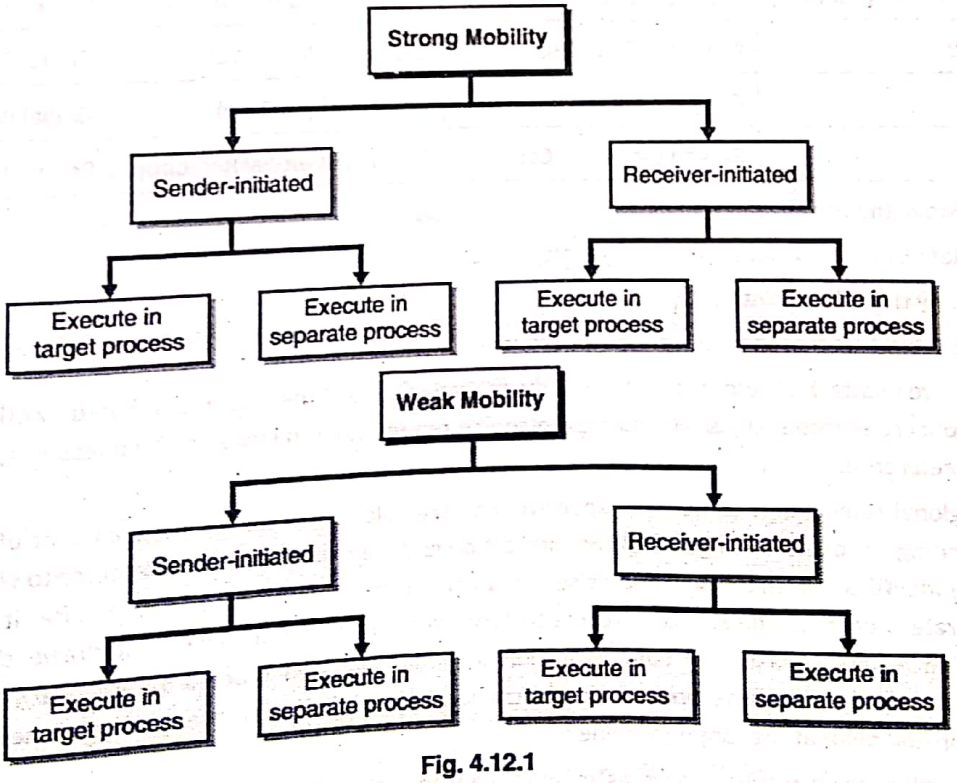


Fig. 4.12.1

4.12.2 Migration and Local Resources

- In migration of resource segment, suppose process holds a reference to a specific TCP port to communicate with remote processes. This reference is held in its resource segment. After migration to other node, process will have to give up the port and request a new one at the destination. In other cases such as file, reference which is URL will remain valid.
- Three types of **process-to-resource** bindings exist. **Binding by identifier** requires precisely the referenced resource such as URL to refer to a specific Web site or to FTP server by means of that server's Internet address. References to local communication end points are also in this category.
- In **binding by value,** only the value of a resource is needed. In this binding, execution of the process does not affect if same value is available on target node. A typical example of binding by value is libraries that are locally available, although their exact location in the local file system may differ between nodes. In **binding by type,** a process indicates it needs only a resource of a specific type. Example is: references to local devices, such as monitors, printers, and so on.
- It is also important to consider the **resource-to-machine** bindings. **Unattached resources** can be easily moved from one machine to other. The example is data file. On the other hand, it is possible to move **fastened resources** but it incurs high cost. Local databases and complete Web sites are the example of fastened resources. **Fixed resources** like local devices and communication endpoints cannot be moved.
- Considering above types of process-to-resource bindings and resource-to-machine bindings, following combination is required to consider while migrating the code.

Process to resource binding	Unattached Resources	Fastened Resources	Fixed Resources
By Identifier	Move (or Global Ref)	Global Ref (or Move)	Global Ref
By Value	Copy (or Move, Global Ref)	Global Ref (or Copy)	Global Ref
By Type	Rebind (or Move, Copy)	Rebind (or Global Ref, Copy)	Rebind (or Global Ref

  - **Move :** Move the resource.
  - **Global Ref :** Establish a global systemwide reference.
  - **Copy :** Copy the value of resource.
  - **Rebind :** Rebind process to locally available resource.
- It is best to move unattached resource along with the migrating code. If the resource is shared by other processes then establish a global reference such as URL to access resource remotely. For fastened or fixed resource, best solution is to create global reference.
- Establishing global reference is sometime expensive. For example, to access the huge amount of multimedia data. Suppose migrating a process is using a local communication end point which is a fixed resource to which the process is bound by the identifier. In this case, process set up a connection to the source machine after it has migrated and install a separate process at the source machine to forward all incoming messages. The drawback is that failure of source machine results in failure of communication with the migrated process. Alternatively, all processes that communicated with the migrating process should change their global reference, and send messages to the new communication end point at the target machine.
- In binding by value, fixed resource such as memory can be shared by establishing global reference. In this case, distributed shared memory support is needed.

- Runtime libraries are the example of binding by value with fastened resources. It can be available on the target machine, or should otherwise be copied before code migration takes place. In this case global reference is preferred if huge amount of data needs to be transferred.
- Unattached resources can be copied or moved to the new destination, if not shared by a many processes. If it is shared then, establishing a global reference is the only option. In case of bindings by type, the obvious solution is to rebind the process to a locally available resource of the same type.

4.12.3 Migration in Heterogeneous Systems

- In large distributed system, machine's architecture and platforms used are different. Hence, environment is heterogeneous. With the use of scripting languages and highly portable languages such as Java, solution to code migration in heterogeneous systems relies on virtual machine. In case of scripting languages, these virtual machines directly interpret source code. In case of Java, it interprets intermediate code generated by a compiler.
- Recently, instead of migrating only processes, migration of entire computing environment is preferred. It is possible to decouple a part from the underlying system and actually migrate it to another machine, provided proper compartmentalization is carried out. In this way, such migration supports strong mobility. In this type of migration, many complexities that arise in binding with different types of resources may be solved.

Review Questions

Q. 1 Explain in short scheduling techniques used in distributed system.

Q. 2 Explain desirable features of good global scheduling algorithm.

Q. 3 Explain in detail task assignment approach.

Q. 4 Explain and classify different load balancing algorithms.

Q. 5 Explain different issues in designing load-balancing algorithms.

Q. 6 Explain policy to estimate the load of a node in load balancing approach.

Q. 7 Explain different process transfer policies in load balancing approach.

Q. 8 Explain different approaches to locate the node in load balancing approach.

Q. 9 Explain different state information exchange policies in load balancing approach.

Q. 10 What are the different priority assignment techniques in load balancing approach? Explain.

Q. 11 Explain issues in designing load-sharing algorithms.

Q. 12 Explain different approaches to locate the node in load sharing approach.

Q. 13 Explain different state information exchange policies in load sharing approach.

Q. 14 Explain the types of process migration.

Q. 15 Explain desirable features of good process migration mechanism.

Q. 16 Explain the mechanism for freezing and restarting the process in process migration.

Q. 17 Explain the address space transfer mechanisms in process migration.

Q. 18 Explain message-forwarding mechanism in process migration.

Q. 19 Explain mechanism to handle coprocesses in process migration.

Q. 20 Explain Process Migration in Heterogeneous System.

Q. 21 What are the advantages of process migration? Explain.

Q. 22 Explain the different advantages of threads with compare to process.

Q. 23 Explain different models to construct server process.

Q. 24 Explain different models for organizing threads.

Q. 25 Explain Issues in Designing a Thread Package.

Q. 26 Explain different approaches to implement thread package.

Q. 27 Explain different techniques of thread scheduling.

Q. 28 Explain the role of virtualization in distributed systems.

Q. 29 Explain architecture of virtual machines.

Q. 30 Explain general design issues of server.

Q. 31 Explain the client-side software for distribution transparency.

Q. 32 What is server cluster? Explain in brief logical organization of server cluster.

Q. 33 Explain different approaches to code migration.

Q. 34 Explain hoe binding to local resources is handled in process migration.

Q. 35 Write short note on "code migration in heterogeneous systems".