Advanced Operating Systems

20MCAT172

(Elective 1)

Module III

Syllabus

- **Distributed Resource Management**:- Mechanisms for building Distributed File Systems – Design Issues – Distributed Shared Memory – Algorithms for Implementing Distributed Shared Memory – Issues in Load Distributing – Components of Load Distributing Algorithms – Sender - Initiated Algorithm – Receiver - Initiated Algorithms

(Mukesh Singhal and Niranjan G. Shivaratri, “***Advanced Concepts in Operating Systems*** – Distributed, Database, and Multiprocessor Operating Systems”, Tata McGraw-Hill, 2001.)

Distributed Resource Management

- The Resource Management in Distributed Environment is concerned with a system in which the main aim is to make sure that a user/client can access the remote resources with as much ease as it can access the local resources.

- A distributed file system is a resource management component of a distributed operating system.

- It implements a common file system that can be shared by all the autonomous computers in the system.

- The basis of Resource Management in the distributed system is resource sharing.

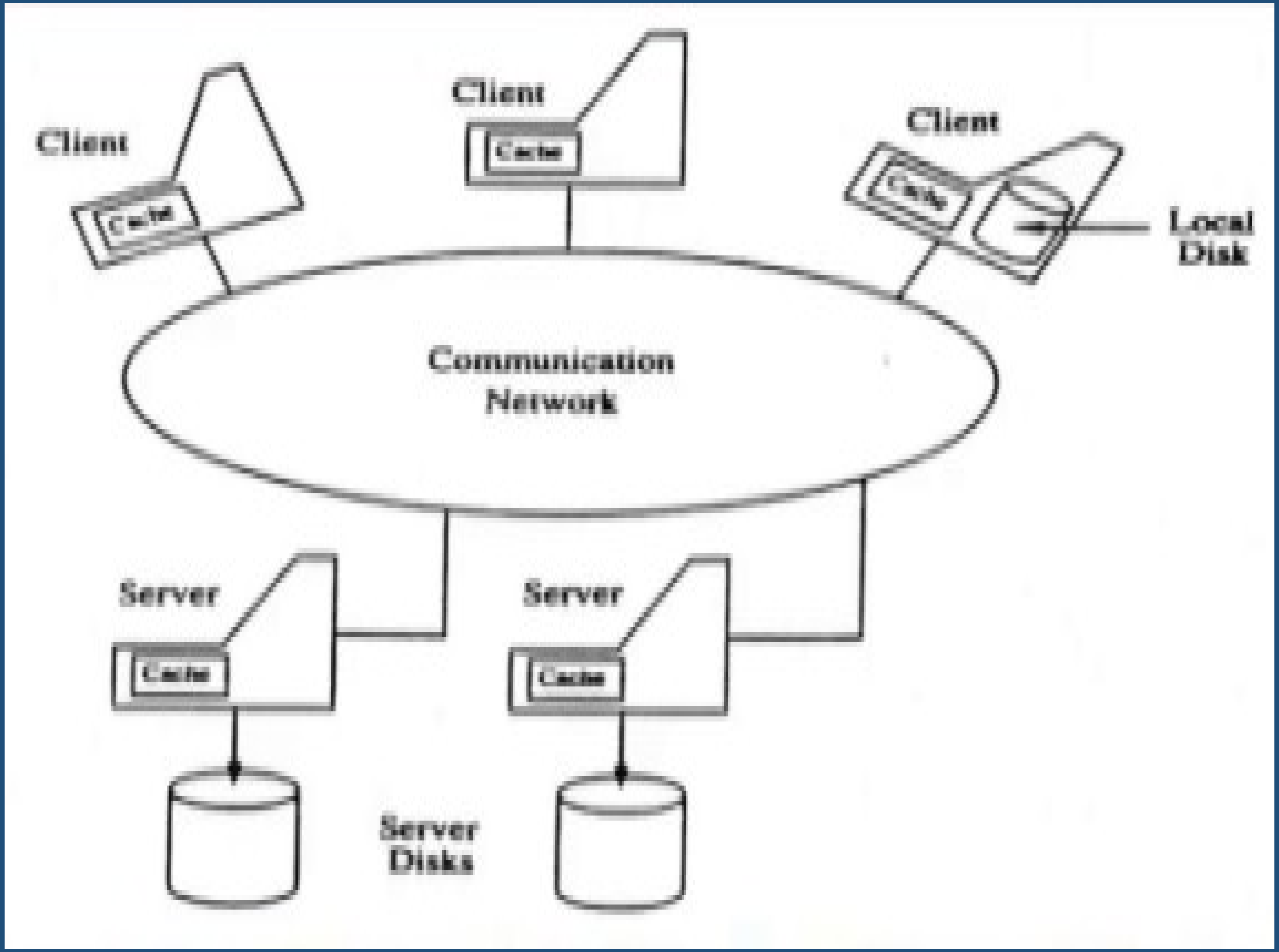
Distributed Resource Management

- **Two** important goals of a distributed file systems are:

1. **Network Transparency**
   - Users should be able to access files over a network as easily as if the files were stored locally.
   - Users should not have to know the physical location of a file to access it.

2. **High Availability**
   - Files should be easily and quickly accessible, irrespective of their physical location.
   - System failures or regularly scheduled activities such as backups or maintenance should not result in the unavailability of files.

Architecture of a Distributed File System



Mechanisms For Building Distributed File Systems

Mechanisms For Building Distributed File Systems

- The basic mechanisms underlying the majority of the distributed file systems operating today are:

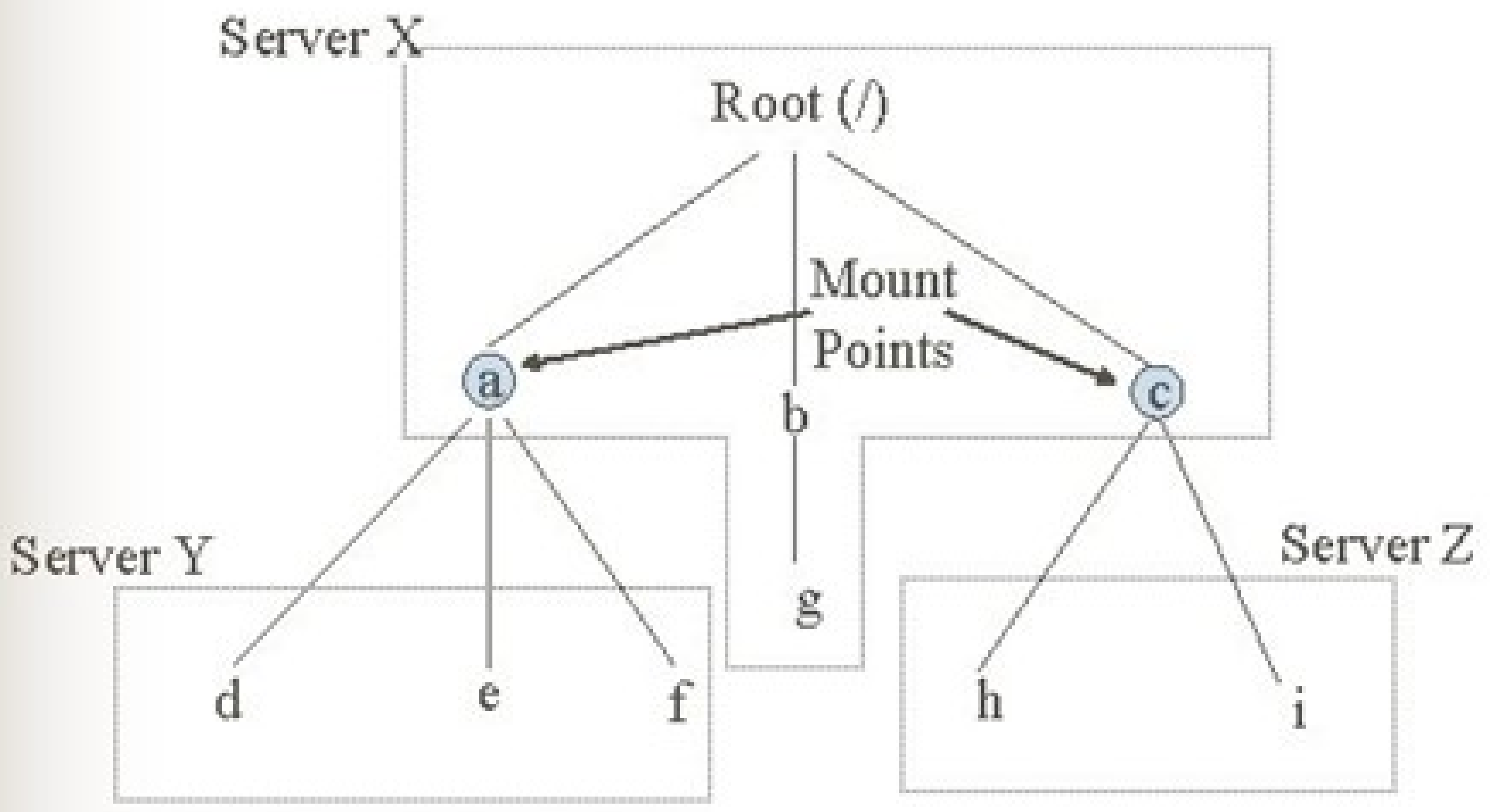
1. **Mounting**
2. **Caching**
3. **Hints**
4. **Bulk Data Transfer**
5. **Encryption**

Mechanisms For Building Distributed File Systems

1. Mounting

- This mechanism provides the binding together of different **filename spaces** to form a single hierarchically structured name space.
- It is UNIX specific and most of existing DFS are based on UNIX.
- A filename space can be bounded to or mounted at an internal node or a leaf node of a namespace tree.
- A node onto which a name space is mounted is called **mount point**.
- The kernel maintains a **mount table**, which maps mount points to appropriate storage devices.

Name Space Hierarchy



Mechanisms For Building Distributed File Systems

- **Uses of Mounting in DFS**
  - File systems maintained by remote servers are mounted at clients so that each client have information regarding file servers.
  - Two approaches are used to maintain mount information.
    - ***Approach 1:*** Mount information is maintained at clients that is each client has to individually mount every required file system. When files are moved to a different server then mount information must be updated in mount table of every client.
    - ***Approach 2:*** Mount information is maintained at servers. If files are moved to a different servers, then mount information need only be updated at servers.

Mechanisms For Building Distributed File Systems

2. Caching

- This mechanism is used in DFS to <u>reduce delays</u> in accessing of data.
- In file caching, a copy of data stored at remote file server is brought to client when referenced by client
- Subsequent access of data is performed locally at client, thus reducing access delays due to network latency.
- Data can be cached in main memory or on the local disk of the clients.
- Data is cached in main memory at servers **to reduce disk access latency**.

<u>Need of Caching in DFS:</u>

- File system performance gets improved accessing remote disks is much slower than accessing local memory or local disks. It also reduces the frequency of access to file servers and the communication network, so scalability gets increased.

Mechanisms For Building Distributed File Systems

3. Hints

- Caching results in the cache consistency problem when multiple clients cache and modify shared data.
- This problem can be avoided by great level of co-operation between file servers and clients which is very expansive.
- Alternative method is that is cached data are not expected to be completely accurate.
- Only those class of applications which can recover after discovering that cached data are invalid can use this approach.
- **Example:** After the name of file or directory is mapped to physical object, the address of object can be stored as hint in the cache. If the address is incorrect that is fails to map the object, the cached address is deleted form the cache and file server consult the same server to obtain the

Mechanisms For Building Distributed File Systems

4. Bulk Data Transfer

- In this mechanism, multiple consecutive data blocks are transferred from server to client.
- This reduces file access overhead by obtaining multiple number of blocks with a single seek, by formatting and transmitting multiple number of large packets in single context switch and by reducing the number of acknowledgement that need to be sent.
- This mechanism is used as many files are accessed in their entirety.

Mechanisms For Building Distributed File Systems

5. Encryption

- This mechanism is used for security in Distributed systems.
- The method was developed by Needham Schrodkar is used in DFS security.
- In this scheme, two entities which want to communicate establish **a key for conversation** with help of authentication server.
- The conversation key is determined by the **authentication server, but is never sent in plain text to either of the entities.**

Design Issues

Design Issues

- The various issues that must be addressed in the design and implementation of distributed file systems are:

  1. Naming and Name Resolution
  2. Caches on Disk or Main Memory
  3. Writing Policy
  4. Cache Consistency
  5. Availability
  6. Scalability
  7. Semantics

Design Issues

1. **Naming and Name Resolution**

- **Name** refers to an object such as file or a directory.
- **Name Resolution** refers to the process of mapping a name to an object that is physical storage.
- **Name space** is collection of names.
- **Names** can be assigned to files in distributed file system in three ways:

a) Concatenate the host name to the names of files that are stored on that host.

b) Mount remote directories onto local directories.

c) Maintain a single global directory where all the files in the system belong to single namespace.

Design Issues

a) Concatenate the host name to the names of files that are stored on that host.

- **Advantages:**
  - This approach guarantees that a file name is unique system wide.
  - Name resolution is simple as file can be located easily.
- **Limitations:**
  - It conflicts with the goal of network transparency.
  - Moving a file from one host to another requires changes in filename and the application accessing that file that is naming scheme is not location independent.

b) Mount remote directories onto local directories.

- Mounting a remote directory require that host of directory to be known only once.
- Once a remote directory is mounted, its files can be referred in location transparent way.
- This approach resolve file name without consulting any host.

Design Issues

c) Maintain a single global directory where all the files in the system belong to single namespace.

- The main **limitation** of this scheme is that it is limited to one computing facility or to a few co-operating computing facilities.
- This scheme is not used generally.

Design Issues

2. Caches on Disk or Main Memory

- Caching refers to storage of data either into the main memory or onto disk space after its first reference by client machine.

Advantages of having cache in main memory:

- Diskless workstations can also take advantage of caching.
- Accessing a cache in main memory is much faster than accessing a cache on local disk.
- The server cache is in the main memory at the server, a single design for a caching mechanism is used for clients and servers.

Design Issues

Limitations:

- Large files cannot be cached completely so caching done block oriented which is more complex.
- It competes with virtual memory system for physical memory space, so a scheme to deal with memory contention cache and virtual memory system is necessary.
- Thus, more complex cache manager and memory management is required.

Advantages of having cache on a local disk:

- Large files can be cached without affecting performance.
- Virtual memory management is simple.
- Portable workstation can be incorporated in distributed system.

Design Issues

3. Writing Policy:

- This policy decides when a modified cache block at client should be transferred to the server. Following policies are used:

  a) Write Through: All writes required by clients applications are also carried out at servers immediately. The main **advantage** is reliability that is when client crash, little information is lost. This scheme cannot take advantage of caching.

-

  b) Delayed Writing Policy: It delays the writing at the server that is modifications due to write are reflected at server after some delay. This scheme can take advantage of caching. The main limitation is less reliability that is when client crash, large amount of information can be lost.

  c) Another scheme delays the updating of files at the server until the file is closed at the client. When average period for which files are open is short then this policy is equivalent to write through and when period is large then this policy is equivalent to delayed writing policy.

Design Issues

4.Cache Consistency:

When multiple clients want to modify or access the same data, then cache consistency problem arises.

•Two schemes are used to guarantee that data returned to the client is valid.

a) Server initiated approach:

- Server inform the cache manager whenever data in client caches becomes valid.
- Cache manager at clients then retrieve the new data or invalidate the blocks containing old data in   cache in cache.
- Server has maintain reliable records on what data blocks are cached by which cache managers.

- Co operation between servers and cache manager is required.

b) Client-initiated approach: It is the responsibility of cache manager at the clients to validate data with server. This approach does not take benefit of caching as the cache manager consult the server for validation of cached block each time.

Design Issues

5. Availability:

- It is one of the important issue is design of Distributed file system.
- Server failure or communication network can affect the availability of files.
- Replication: The primary mechanism used for enhancing availability of files is replication. In this mechanism, many copies or replicas of files are maintained at different servers.

Limitations:

- Extra storage space is required to store replicas.
- Extra overhead is required in maintained all replicas up to date.

Following situations cause inconsistency among replicas:

- Replica is not updated due to failure of server storing the replica
- All the file servers storing the replicas of file are not reachable from all clients due to network partition and replicas of file in different partition are updated differently.

Design Issues

6. Scalability:

- The design of DFS should be such that new systems can be easily introduced without affecting it.
- Generally, client-server organisation is used to define DFS structure.
- Caching is used in this organisation to improve performance.
- Server initiated cache invalidation is used to maintain cache consistency.
- In this approach, server maintain a record based information regarding all the clients sharing file stored on it. This information represents server state.
- As the system grows both the size of server state and load due to invalidations increases on server.

Design Issues

6. Scalability:

- Following schemes can be used to reduce server state and server load:

a) Exploit knowledge about usage of files that is it is found that most commonly used and shared files are accessed in read only mode. So, there is no need to check the validity of these files of maintain the list of clients at servers for validation purpose.

b) Generally, data required by a client is found in another client's cache so a client can obtain required data from another client rather than server.

- Structure of server process play an important role.
- If server is designed with single process, then many clients have to wait for a long time whenever a disk input/ output is initiated. This can be avoided if separate process is assigned to each client.

Design Issues

7. Semantics:

- The semantic of a file system represent the affects of accesses on file.
- The basic semantic is that a read operation will return the data (stored) due to latest write operation.
- The semantic can be guaranteed in two ways:
  - All read and writes from various clients will have to go through the server.
  - Sharing will have to be disallowed either by server or by the use of locks by application.

  - In first way, the server become bottleneck and in second way, the file is not available for certain clients.

Distributed Shared Memory

Distributed Shared Memory

- **Distributed shared memory(DSM)** system is a resource management component of distributed operating system that implements shared memory model in distributed system which have no physically shared memory.
- The shared memory model provides a virtual address space which is shared by all nodes in a distributed system.
- The central issues in implementing DSM are:
    - how to keep track of location of remote data.
    - how to overcome communication overheads and delays involved in execution of communication protocols in system for accessing remote data.
    - how to make shared data concurrently accessible at several nodes to improve performance.

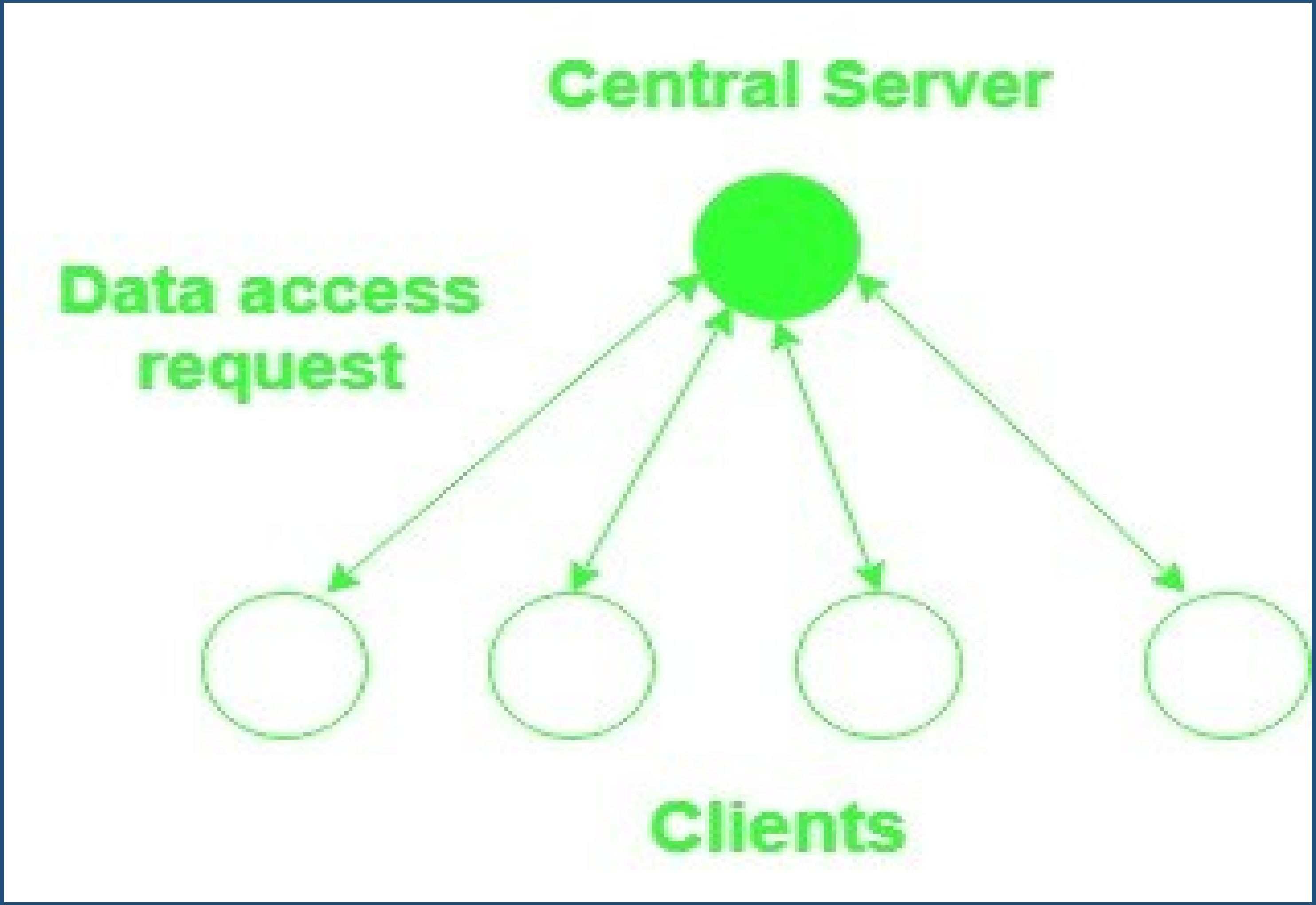
Distributed Shared Memory

- Based on these challenges there are algorithms designed to implement distributed shared memory.
- There are four algorithms –
    - **The Central Server Algorithm**
    - **The Migration Algorithm**
    - **The Read Replication Algorithm**
    - **The Full Replication Algorithm**

The Central Server Algorithm

- All shared data is **maintained by the central server**.
- Other nodes of the distributed system **request for reading and writing data** to the server which serves the request and updates or provides access to the data along with **acknowledgment messages**.
- It services the read request from other nodes or clients by returning the data items to them.
- It updates the data on write requests by clients and returns acknowledgment messages.
- These acknowledgment messages are used to provide the status of the data request is served by the server.
- When the data is sent to the calling function, it acknowledges a number that shows the access sequence of the data to maintain concurrency.
- Duplicate write requests can be detected by associating **sequence numbers** with write request.
- And time-out is returned in case of failure.
- For larger distributed systems, there can be more than one server. In this case, the servers are located using their address or using mapping functions.

The Central Server Algorithm



The Migration Algorithm

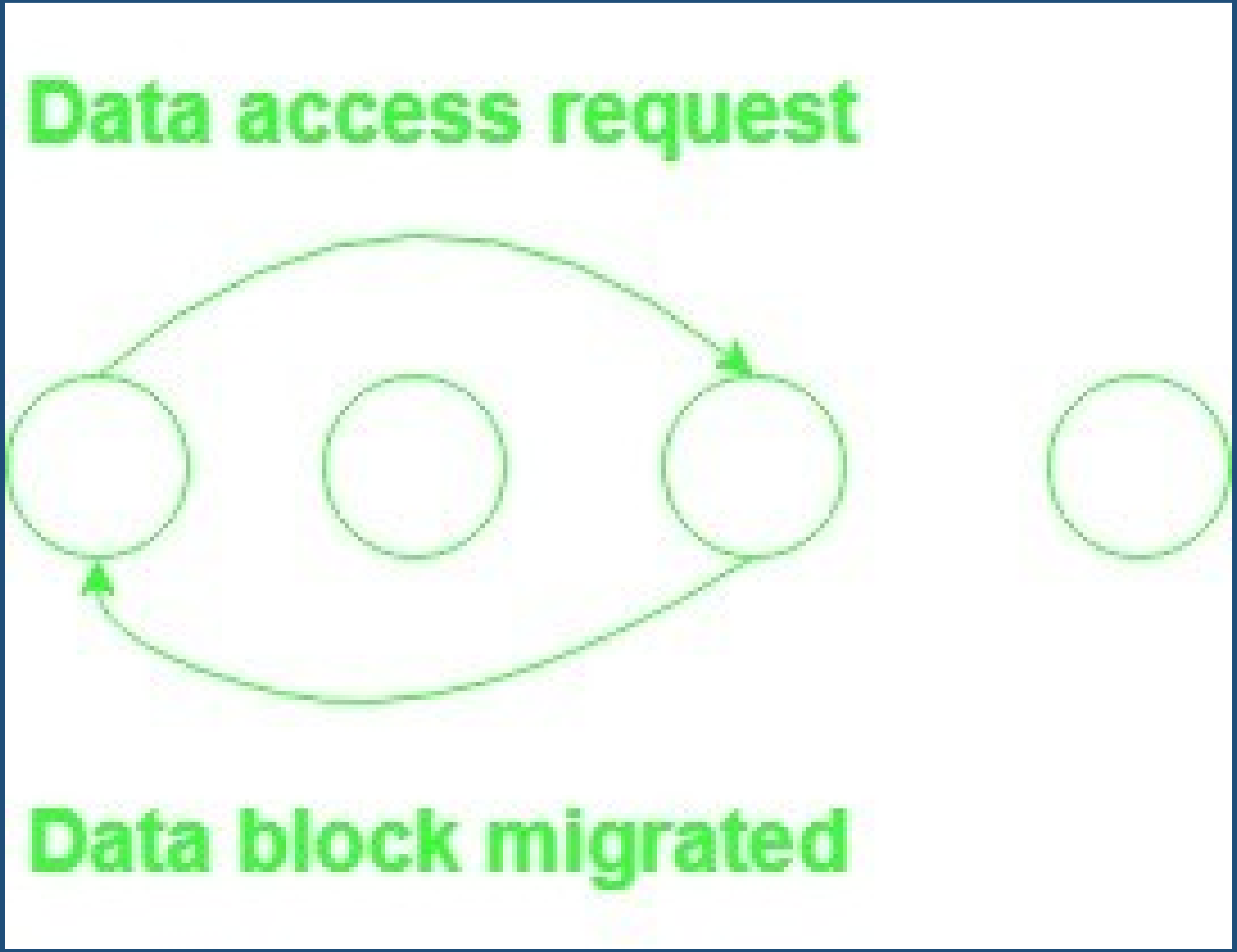
- As the name suggest the migration algorithm does the work of migration of data elements.
- Instead of using a central server serving each request, the **block containing the data requested by a system is migrated** to it for further access and processing.
- It migrates the data on request.
- This algorithm though is good if when a system accesses the same block of data multiple times and the **ability to integrate virtual memory** concept, has some shortcomings that are needed to be addressed.
- Only one node is able to access the shared data element at a time and the whole block is migrated to that node.

The Migration Algorithm

- The whole page or block containing the data item migrates instead of an individual item requested.
- This algorithm is more **prone to thrashing** due to the migration of data items upon request by the node.
- Thrashing: Where pages frequently migrate between nodes while servicing only a few requests.
- To reduce thrashing, the Mirage system uses a **tunable parameter** that determines the duration for which a node can possess a shared data item.
- This allow a node to make a number of accesses to the page before it is migrated to another node.

The Migration Algorithm

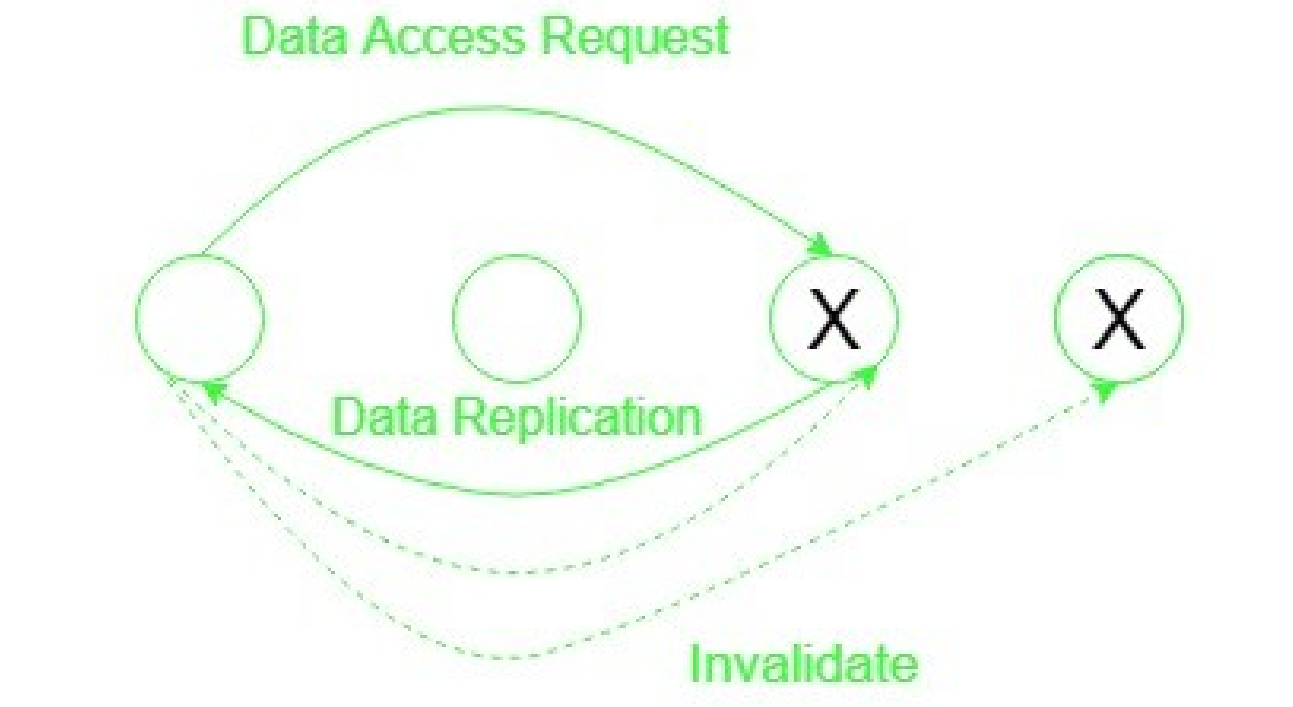
•



The Read Replication Algorithm

- In the read replication algorithm, the data block that is to be accessed is replicated and only reading is allowed in all the copies.
- If a write operation is to be done, then all read access is put on halt till all the copies are updated.
- Overall system performance is improved as concurrent access is allowed.
- But write operation is expensive due to the requirement of updating all blocks that are shared to maintain concurrency.
- All copies of data element are to be tracked to maintain consistency.

The Read Replication Algorithm



Write operation in Read Replication algorithm

The Full Replication Algorithm

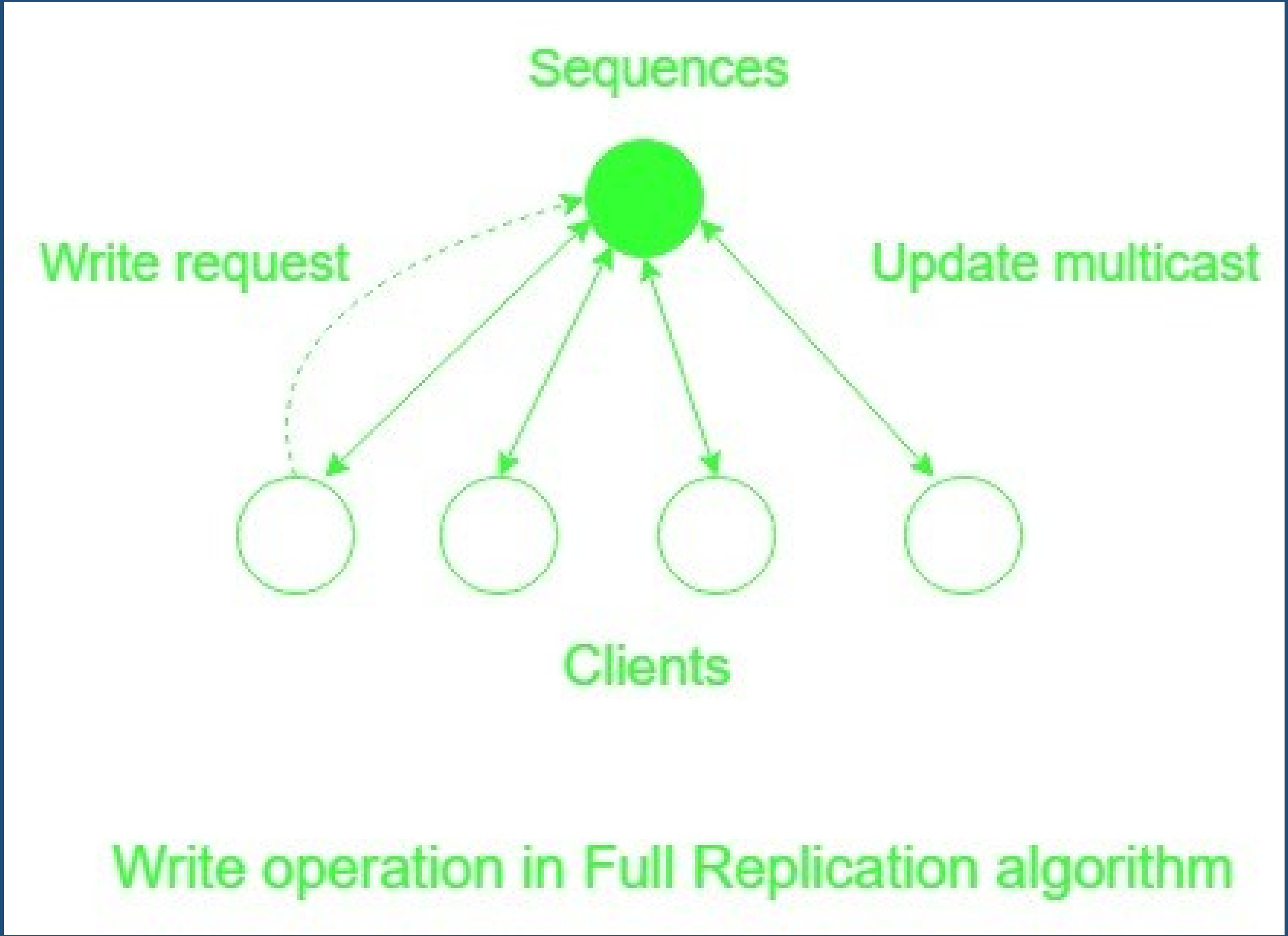
- An extension to read the replication algorithm allowing the nodes to perform both read and write operation on the shared block of concurrently.

- But this access of nodes is controlled to maintain its consistency.

- To maintain consistency of data on concurrent access of all nodes sequence is maintained and after every modification that is made in the data a multicast with modifications is reflected all the data copies.

The Full Replication Algorithm

•



Write operation in Full Replication algorithm

Issues in Load Distributing

Distributed Scheduling Issues in Load Distributing

- Scheduling refers to the execution of non-interactive processes or tasks at designated times and places around a network of computer.
- **Distributed scheduling refers to the chaining of different jobs into a coordinated workflow that spans several computers.**
- For example: - you schedule a processing job on computer1 and computer2, and when these are finished you need to schedule a job on computer3, this is distributed scheduling.
- Distributed Systems offer a tremendous processing capacity.
- **Good resource allocation schemes** are needed to take full advantage of distributed systems.
- ***A distributed scheduler is a resource management component of a distributed operating system.***

Motivation:

- Locally distributed system consists of a collection of **autonomous computers**, connected by a LAN.
- **In a locally distributed system, there is a good possibility that several computers are heavily loaded while others are ideal or lightly loaded.**
- If we can move jobs around, the overall performance of the system can be **maximized.**
- A **distributed scheduler** is a resources management component of a distributed operating system that focuses on judiciously and transparently redistributing the load of the system among the computers to maximize the overall performance.

A Distributes System Without Load Distributing

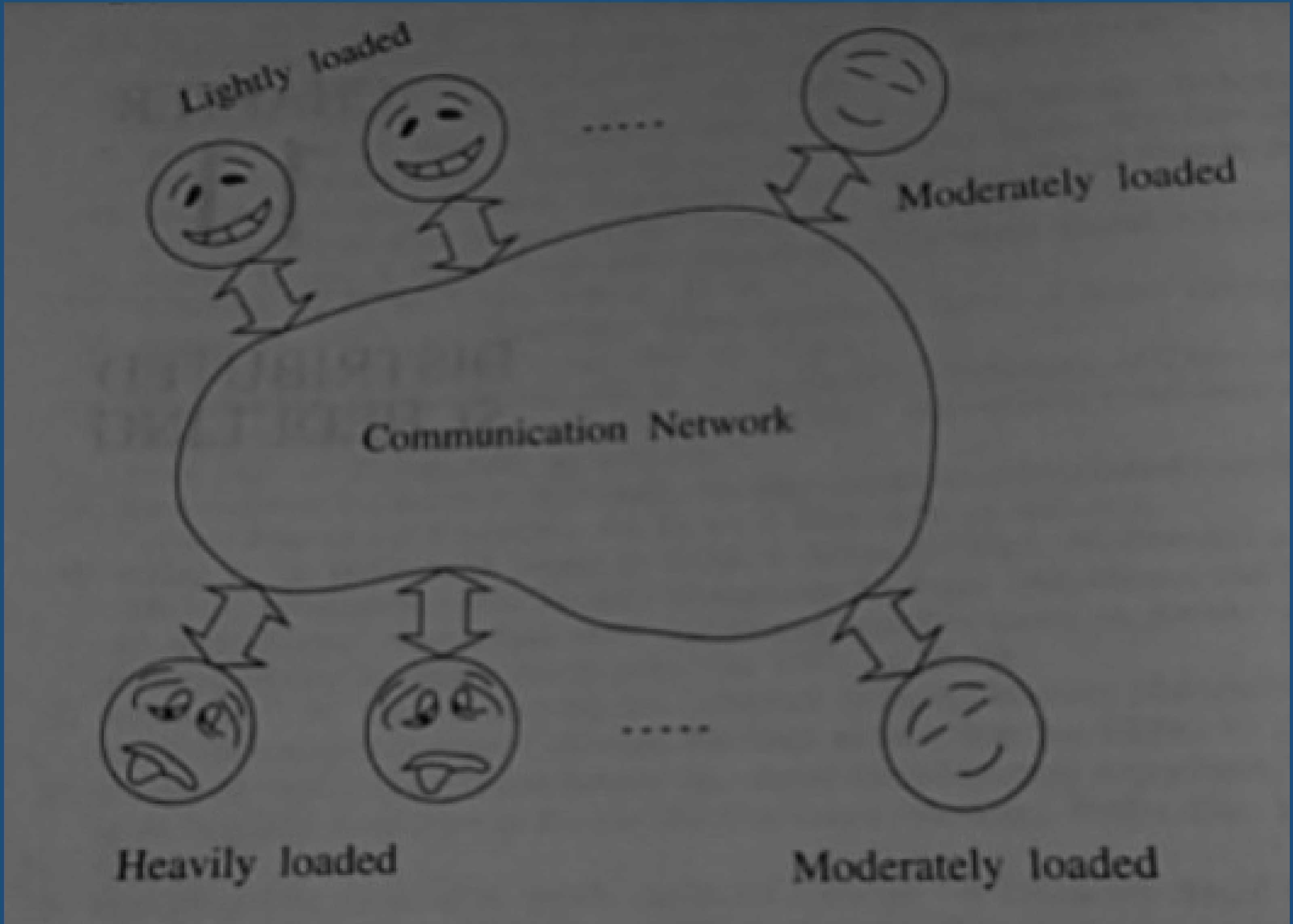


FIGURE 11.1
A distributed system without load distributing (adapted from [32]).

Issues in Load Distributing

The Various Issues are:

1. Load
2. Classification of Algorithms
3. Load Sharing vs. Load Balancing
4. Preemptive vs. Nonpreemptive Transfers

1. Load

- Load on a system/node can correspond to the queue length of tasks/ processes that need to be processed.
- Distributing load: transfer tasks/processes among nodes.
- If a task transfer (from another node) takes a long time, the node may accept more tasks during the transfer time.
- Causes the node to be highly loaded. Affects performance.
- Solution: artificially increment the queue length when a task is accepted for transfer from remote node (to account for the proposed increased in load).
- Task transfer can fail? : use timeouts.

2. Classification of Load Distributing Algorithms

Static load distribution algorithms: Decisions are hard-coded into an algorithm with a priori knowledge of system.

Dynamic load distribution algorithms: use system state information such as task queue length, processor utilization and the system state information (the loads at node).

Adaptive load distribution algorithms: are a special class of dynamic load distributing algorithms in that they adapt their activities by dynamically changing the parameters of the algorithm to suit the changing system state.

- This algorithm adapt the approach based on system state.(e.g.) Dynamic distribution algorithms collect load information from nodes.
- Load information collection itself can add load on the system as messages need to be exchanged.
- Adaptive distribution algorithms may stop collecting state information at high loads.

3. Load Balancing vs. Load sharing

Load balancing: Equalize load on the participating nodes.

- Transfer tasks even if a node is not heavily loaded so that queue length on all Nodes are approximately equal.
- More number of tasks transfers, might degrade performance.

Load sharing: Reduce burden of an overloaded node.

- Transfer tasks only when the queue length exceeds a certain threshold.
- Less number of task transfers.
- Anticipatory task transfer: Transfer from overloaded nodes to ones that are likely to become idle/highly loaded
- More like load balancing, but may be less number of transfers.

4. Preemptive vs. Nonpreemptive Transfers

- **Preemptive task** transfers involves the transfer of a task that is partially executed.
  - This transfer is an expensive operation as the collection of a task's state can be difficult.
  - Typically a task state consists of a virtual memory image, a process control block, unread I/O buffers and messages, file pointers, timers that have been set etc.
- **Nonpreemptive task** transfers involve the transfer of tasks that have not begun execution and hence do not require the transfer of the task's state.
  - Also referred as task placements.

Components of Load Distributing Algorithms

Components of Load Distributing Algorithms

- A **load distributing algorithm** has **four** components:-
    - **Transfer Policy**
    - **Selection Policy**
    - **Location Policy**
    - **Information policy**

1. Transfer Policy

- This policy determines whether the node is in a suitable state to share the load.
- Most of the transfer policies are threshold based policies.
- If a load at a node exceeds a threshold value T, then the node is overloaded and act as a sender.
- If the load at the node falls below a threshold T, then the load is under loaded and acts as a receiver.

2. Selection Policy

- This policy selects a task for transfer, once the transfer policy decides that the node is a sender.
- **Simplest approach** is to select the newly originated task at the node which has made this node as the sender for task transfer, it will also be a cheap operation as it will be a non preemptive task transfer.
- The **basic criterion** which is to be satisfied during selection of a task is that the **overhead incurred in task transfer must be less** than the reduction in response time of the task and long-lived task satisfies this condition.
- Another factor which is to be considered during selection of tasks is that the task must have **fewer location dependent system calls** since such calls are to be executed on the same machine where the task has been originated.

3. Location Policy

- This policy is to find suitable nodes to share load.
- Simplest approach to find such node is **Polling.**
- In polling, one node calls another node to determine whether it is in a state of load sharing.
- Nodes can be called either serially or parallel.
- A node can be selected for polling randomly or based on the information collected during the previous polls, or on a nearest neighbor basis.
- An alternative to polling is to distribute a query to all nodes to find the available nodes.

4. Information Policy

- This policy is responsible for determining when the system state of other nodes in the system should be **collected**, from where it is to be collected from and what information is to be collected.
- It is of three types:
  - ***Demand Driven***
  - ***Periodic***
  - ***State Change Driven Policy***

3. Information Policy...

Demand Driven:

- In this policy, the node starts correcting the state of other nodes when it becomes a sender or receiver. This policy is basically a dynamic policy. This policy can be sender initiated, receiver initiated or symmetrically initiated. In sender initiated, a sender looks for receiver to give their load, In receiver initiated, receiver searches for sender and take their load, and symmetrically initiated is the combination of both where the collection of the system state is started whenever there is need of extra processing power or extra work.

Periodic:

- In this policy, the nodes exchange their system information periodically and on the basis of this information, task transfer is made. This policy does not adapt its activities according to system state change. For example if the system is already overloaded then exchanging the system state information periodically will further worsen the situation.

3. Information Policy...

State Change Driven Policy:

- In this policy, a node disseminates its state information to other nodes whenever its state changes by certain degree. This policy differs from demand driven policy as in this case, the nodes disseminate their state information rather than collecting the state information of other nodes. Under centralized approach, a node disseminates its information t centralized collection point whereas in case of decentralized approach, a node disseminate to peers.

Load Distributing Algorithms

Load Distributing Algorithms

- **Sender-Initiated:**
  - distribution initiated by an overloaded node.
- **Receiver-Initiated:**
  - distribution initiated by lightly loaded nodes.
- **Symmetric:**
  - initiated by both senders and receivers.
- **Adaptive:**
  - sensitive to state of the system.

Sender - Initiated

- **Transfer Policy:**
    - Use thresholds.
    - Sender if queue length exceeds T.
    - Receiver if accepting a task will not make queue length exceed T.
- **Selection Policy:** Only newly arrived tasks.
- **Location Policy:**
    - Random: Use no remote state information.
    - Task transferred to a node at random.
        - No need for state collection. Unnecessary task transfers (processor thrashing) may occur.
    - Threshold: poll a node to find out if it is a receiver. Receiver must accept the task irrespective of when it (task) actually arrives.
    - PollLimit, ie., the number of polls, can be used to reduce overhead.
    - Shortest: Poll a set of nodes.
    - Select the receiver with shortest task queue length.
- **Information Policy:**
    - demand-driven.

Sender - Initiated

- **Transfer Policy:**
  - Use thresholds.
  - Sender if queue length exceeds T.
  - Receiver if accepting a task will not make queue length exceed T.
- **Selection Policy:** Only newly arrived tasks.
- **Location Policy:**
  - Random: Use no remote state information.
  - Task transferred to a node at random.
    - No need for state collection. Unnecessary task transfers (processor thrashing) may occur.
  - Threshold: poll a node to find out if it is a receiver. Receiver must accept the task irrespective of when it (task) actually arrives.
  - PollLimit, ie., the number of polls, can be used to reduce overhead.
  - Shortest: Poll a set of nodes.
  - Select the receiver with shortest task queue length.

Receiver - Initiated

- **Transfer Policy:**
  - uses thresholds.
  - Queue lengths below T identifies receivers and those above T identifies senders.
- **Selection Policy:** as before.
- **Location Policy:** Polling.
  - A random node is polled to check if a task transfer would place its queue length below a threshold.
  - If not, the polled node transfers a task.
  - Otherwise, poll another node till a static PollLimit is reached.
  - If all polls fail, wait until another task is completed before starting polling operation.
- **Information policy:** demand-driven.

Receiver - Initiated

- **Drawback:**
  - Polling initiated by receiver implies that it is difficult to find senders with new tasks.
  - Reason: systems try to schedule tasks as and when they arrive.
  - Effect: receiver-initiated approach might result in preemptive transfers.
  - Hence transfer costs are more.
  - Sender-initiated: transfer costs are low as new jobs are transferred and so no need for transferring task states.

Thank You