

# OUTLINE

- Introduction
- A System Performance Model
- Static Process scheduling with communication
- Dynamic load sharing and balancing
- Distributed process implementation
- Distributed file systems (DFS):
  - Transparencies and characteristics of DFS
  - DFS design and implementation
  - Transaction service and concurrency control
  - Data and file replication
- Case Studies:
  - Sun network file systems
  - General parallel system and window's file systems
  - Andrew and coda file systems

# DISTRIBUTED PROCESS SCHEDULING

- The primary objective of *scheduling* is to enhance overall system performance metrics such as process completion time and processor utilization.
- The existence of multiple processing nodes in distributed systems present a challenging problem for scheduling processes onto processors and vice versa.

## A SYSTEM PERFORMANCE MODEL

Partitioning a task into multiple processes for execution can result in a speedup of the total task completion time.

The *speedup* factor  $S$  is a function.

$$S = F(\text{Algorithm, System, Schedule})$$

$S$  can be written as:

$$S = \frac{OSPT}{CPT} = \frac{OSPT}{OCPT_{ideal}} \times \frac{OCPT_{ideal}}{CPT} = S_i \times S_d$$

where

- $OSPT$  = optimal sequential processing time

## SYSTEM MODEL CONT...

- $CPT$  = concurrent processing time
- $OCPT_{ideal}$  = optimal concurrent processing time
- $S_i$  = the ideal speedup
- $S_d$  = the degradation of the system due to actual implementation compared to an ideal system

$S_i$  can be rewritten as:

$$S_i = \frac{RC}{n \cdot RP}$$

where

$$RP = \frac{1}{OSPT} \sum_{i=1}^m P_i$$

and

$$RC = \frac{1}{OCPT_{ideal} \times n} \sum_{i=1}^m P_i$$

and  $n$  is the number of processors. The term  $\sum_{i=1}^m P_i$  is the total computation of the concurrent algorithm where  $m$  is the number of tasks in the algorithm.  $S_d$  can be rewritten as:

$$S_d = \frac{1}{1 + \rho}$$

where

$$\rho = \frac{CPT - OCPT_{ideal}}{OCPT_{ideal}}$$

## SYSTEM MODEL CONT...

- *RP is Relative Processing*: how much loss of speedup is due to the substitution of the best sequential algorithm by an algorithm better adapted for concurrent implementation.
- *RC is the Relative Concurrency* which measures how far from optimal the usage of the n-processor is. It reflects how well adapted the given problem and its algorithm are to the ideal n-processor system. The final expression for *speedup S* is:

$$S = (RC/RP) \times (1/(1 + \rho)) \times n$$

- The term  $\rho$  is called *efficiency loss*. It is a function of scheduling and the system architecture. It would be decomposed into two independent terms:  $\rho = \rho_{\text{sched}} + \rho_{\text{syst}}$ , but this is not easy to do since scheduling and the architecture are interdependent. The best possible schedule on a given system hides the communication overhead (overlapping with other computations).

# SYSTEM MODEL CONT...

- The unified speedup model integrates three major components:
  1. **Algorithm development**
  2. **System architecture**
  3. **Scheduling policy**
- With the objective of minimizing the total completion time (makespan) of a set of interacting processes.
- If processes are not constrained by precedence relations and are free to be redistributed or moved around among processors in the system, performance can be further improved by sharing the workload.
- **Statically - load sharing**
- **Dynamically - load balancing**

# WORKLOAD DISTRIBUTION

- Performance can be further improved by *load distribution and sharing*.
- ***Load Sharing: Static Workload Distribution***
  - I. Dispatch process to the idle processor statically on arrival.
  - II. Corresponding to processor- pool model.
- ***Load Balancing: Dynamic Workload Distribution***
  1. Migrate process dynamically from heavily loaded processor to lightly loaded processors.
  2. Corresponding to migration workstation model.

# FIRST WE MUST KNOW: WHAT IS LOAD BALANCING AND LOAD SHARING??

## ***Load Balancing:***

The aim of *load balancing* products is to create a *distributed* network where requests are evenly split among various servers.

## ***Load Sharing:***

Meanwhile, *load sharing* entails sending a portion of the traffic to one server and another portion elsewhere.

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

Low cost but high gain

Extensibility and incremental growth

Higher throughput, reliability

Load balancing reduces the job idle time

Small jobs do not suffer from long starvation

Maximum utilization of resources

Response time becomes shorter



# STATIC LOAD BALANCING ALGORITHM

- In static algorithm the processes are assigned to the processors at the compile time according to the performance of the nodes.
- Once the processes are assigned, no change or reassignment is possible at the run time.
- Number of jobs in each node is fixed in static load balancing algorithm.** Static algorithms do not collect any information about the nodes.
- The assignment of jobs is done to the processing nodes on the basis of the following factors: *incoming time, extent of resource needed, mean execution time and inter-process communications.*
- Since these factors should be measured before the assignment, this is why static load balance is also called **probabilistic algorithm**.

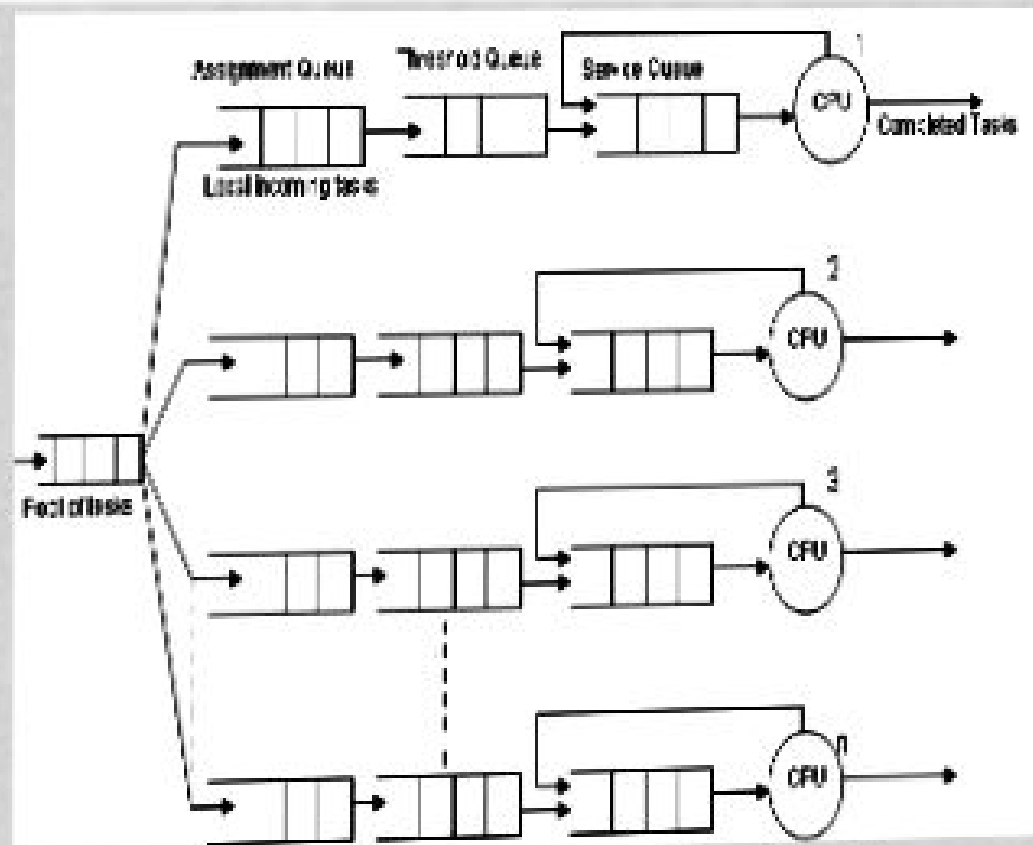


Fig: Model of Processing Node

## SUB CLASSES OF SLBA

The static load balancing algorithms can be divided into two sub classes:

1. Optimal static load balancing.
2. Sub optimal static load balancing.

### *Optimal Static Load Balancing Algorithms:*

- If all the information and resources related to a system are known Optimal static load balancing can be done.
- It is possible to increase throughput of a system and to maximize the use of the resources by optimal load balancing algorithm.

## SUB CLASSES OF SLBA CONT...

### ***Sub-Optimal Static Load Balancing Algorithm:***

- Sub-optimal load balancing algorithm will be mandatory for some applications ***when optimal solution is not found.***
- The thumb-rule and heuristics methods are important for sub-optimal algorithm.

## DYNAMIC LOAD BALANCING (DLB)

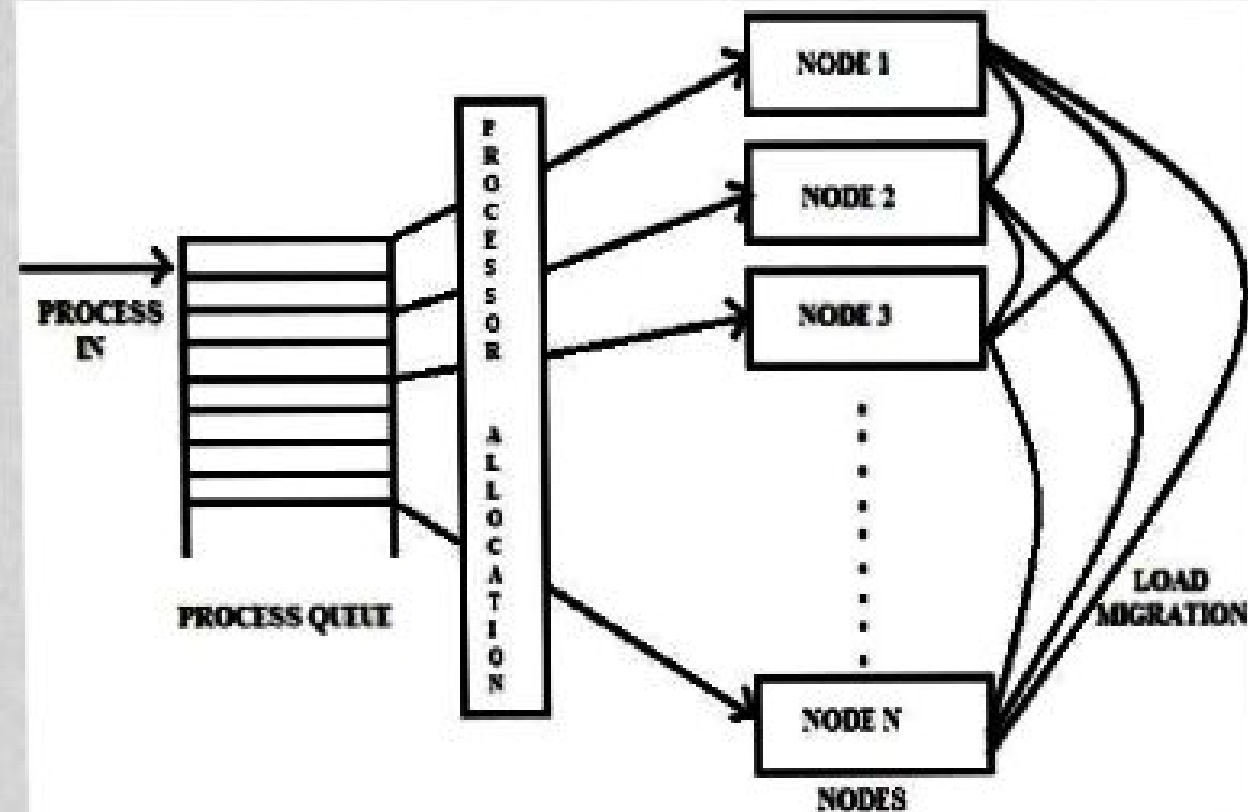
- In dynamic load balancing algorithm assignment of jobs is done at the runtime.
- In DLB jobs are reassigned at the runtime depending upon the situation that is the load will be transferred from heavily loaded nodes to the lightly loaded nodes.
- In this case communication overheads occur and become more when number of processors increase.
- In dynamic load balancing no decision is taken until the process gets execution.

# DYNAMIC LOAD BALANCING (DLB) CONT...

- This strategy collects the information about the system state and about the job information.

- As more information is collected by an algorithm in a short time, potentially the algorithm can make better decision.

- Dynamic load balancing is mostly considered in heterogeneous system because it consists of nodes with different speeds, different communication link speeds, different memory sizes, and variable external loads due to the multiple.



**Fig: Job Migration in Dynamic Load Balancing Strategy**

# COMPARISON OF SLB AND DLB

S.NO.	Factor	Load Balancing	
		SLB Algorithm	DLB Algorithm
1	Nature	Work load is assigned at compile time	Work load is assigned at run time
2	Overhead involved	Little overhead due to IPC	Greater overhead due to process redistribution
3	Resource utilization	Lesser utilization	Greater utilization
4	Processor thrashing	No thrashing	Considerable thrashing
5	State woggling	No woggling	Considerable woggling
6	Predictability	Easy to predict	Difficult to predict
7	Adaptability	Less adaptive	More adaptive
8	Reliability	Less	More
9	Response time	Short	Longer
10	Stability	More	Less
11	Complexity	Less	More
12	Cost	Less	More

# QUEUEING THEORY

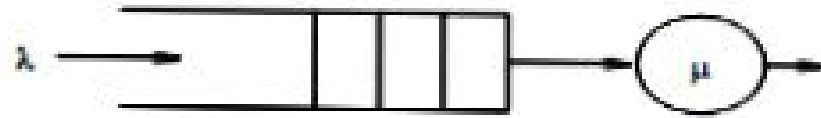
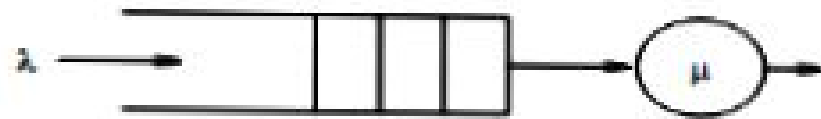
Queueing theory deals with problems which involve queueing (or waiting). Typical examples might be:

- banks/supermarkets - waiting for service
- computers - waiting for a response
- failure situations - waiting for a failure to occur e.g. in a piece of machinery
- public transport - waiting for a train or a bus

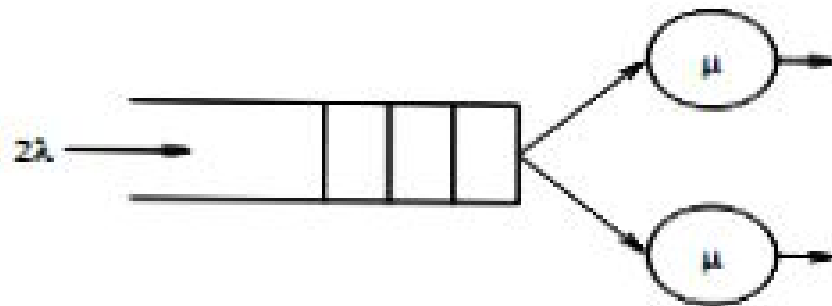
Performance of system described as queueing models can be computed using queueing theory. An  $x/y/z$  queue is one where:

- *X: Arrival process*
- *Y: Service time distribution*
- *Z: Number of servers*
- *$\lambda$ : arrival rate       $\mu$ : service rate*
- *$\gamma$ : Migration rate which depend on the channel bandwidth, migration protocol, context and state information of the process being transferred*

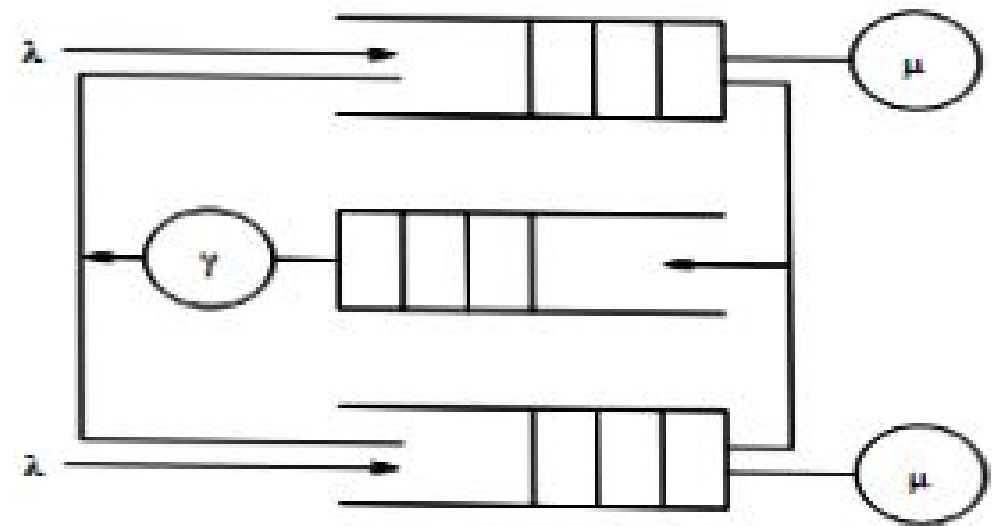
# QUEUEING CONT...



(a) M / M / 1 isolated workstations



(b) M / M / 2 processor pool model



(c) Migration workstation model

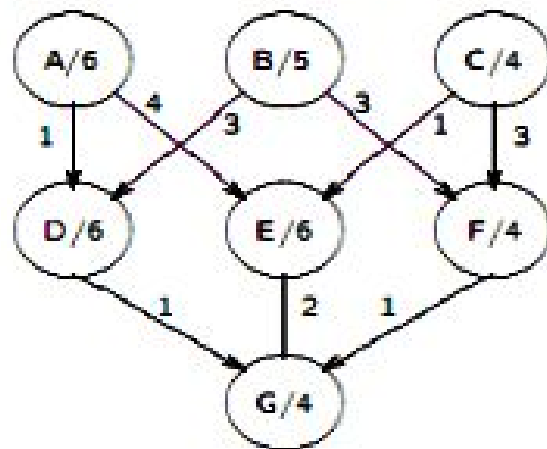


# STATIC PROCESS SCHEDULING

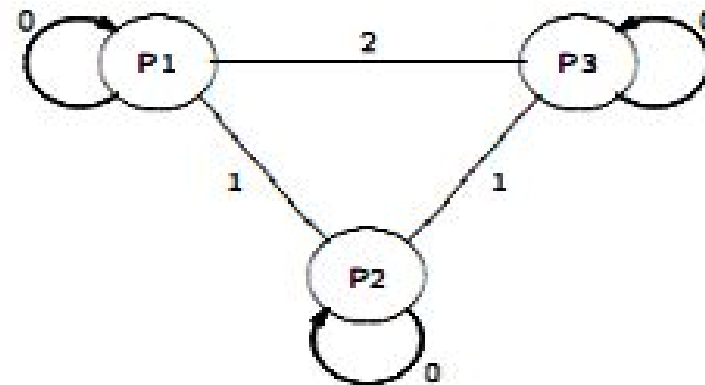
- Scheduling a set of partially ordered tasks on a non-preemptive multiprocessor system of identical processors to minimize the overall finishing time (makespan).
- Except for some very restricted cases scheduling to optimize makespan is NP-complete.
- A good heuristic distributed scheduling algorithm is one that can best balance and overlap computation and communication.
- In *static scheduling*, the mapping of processes to processors is determined before the execution of the processes. Once a process is started, it stays at the processor until completion.

# PRECEDENCE PROCESS MODEL

- Program is represented by a *directed acyclic graph (DAG)*
- Computational model
- Primary objective of task scheduling is to achieve maximal concurrency for task execution within a prog



(a) Precedence process model



(b) Communication system model

# PRECEDENCE PROCESS MODEL CONT...

- Finding the minimum makespan is NP-complete, so we will rely on heuristic algorithms for finding good mapping of the process model to the system model.
  - *For precedence process graphs*, the notion of critical path is useful - the longest execution path in the DAG, which is the lower bound of the makespan. Simple heuristic: map all tasks in a critical path onto a single processor.
1. *List Scheduling (LS) strategy*: No processor remains idle if there are some tasks available that it could process (without considering communication overhead).
  2. *Extended List Scheduling (ELS) strategy*: Allocating tasks to processors according to LS and adding communication delays, communication overhead.
  3. *Earliest Task First (ETF)*: The earliest schedulable task is scheduled first (calculation includes communication overhead).

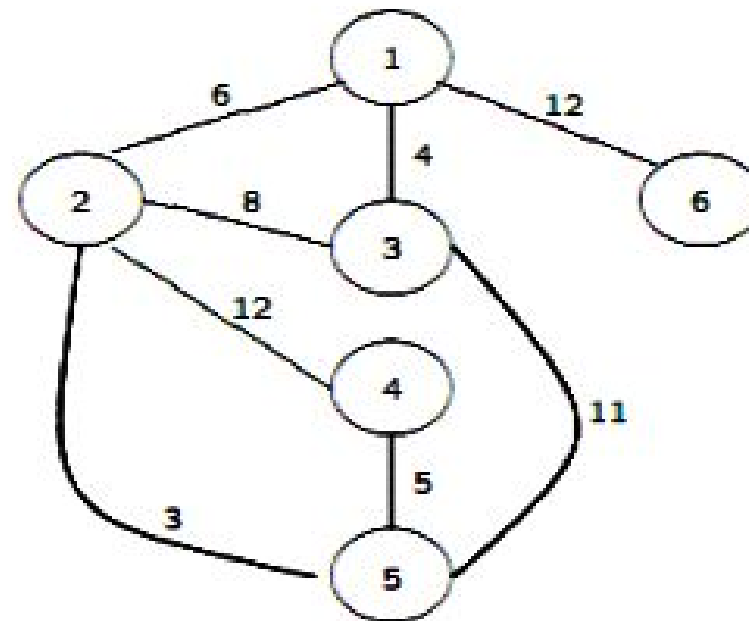
## PRECEDENCE PROCESS MODEL CONT...

- *Process scheduling* for many system applications has a perspective very different from precedence model – applications may be created independently, processes do not have explicit completion time and precedence constraints.
- Primary objectives of process scheduling are to maximize resource utilization and to minimize inter process communication.
- Communication process model is an undirected graph  $G$  with node and *edge sets*  $V$  and  $E$ , where nodes represent processes and the weight on an edge is

# PRECEDENCE PROCESS MODEL CONT...

Process	Cost on A	Cost on B
1	5	10
2	2	infinity
3	4	4
4	6	3
5	5	2
6	infinity	4

(a) Computation set



(b) Communication cost

# DYNAMIC LOAD SHARING AND LOAD BALANCING

Objective of scheduling: utilization of the system (has direct bearing on throughput and completion time) and fairness to the user processes (difficult to define).

- If we can designate a controller process that maintains the information about the queue size of each processor:
- Fairness in terms of equal workload on each processor (join the shortest queue) - migration workstation model (use of load sharing and load balancing, perhaps load redistribution i.e. process migration)
- Fairness in terms of user's share of computation resources (allocate processor to a waiting process at a user site that has the least share of the processor pool) - processor pool model
- *Solutions without a centralized controller:* sender- and receiver-initiated algorithms.

# CONT...

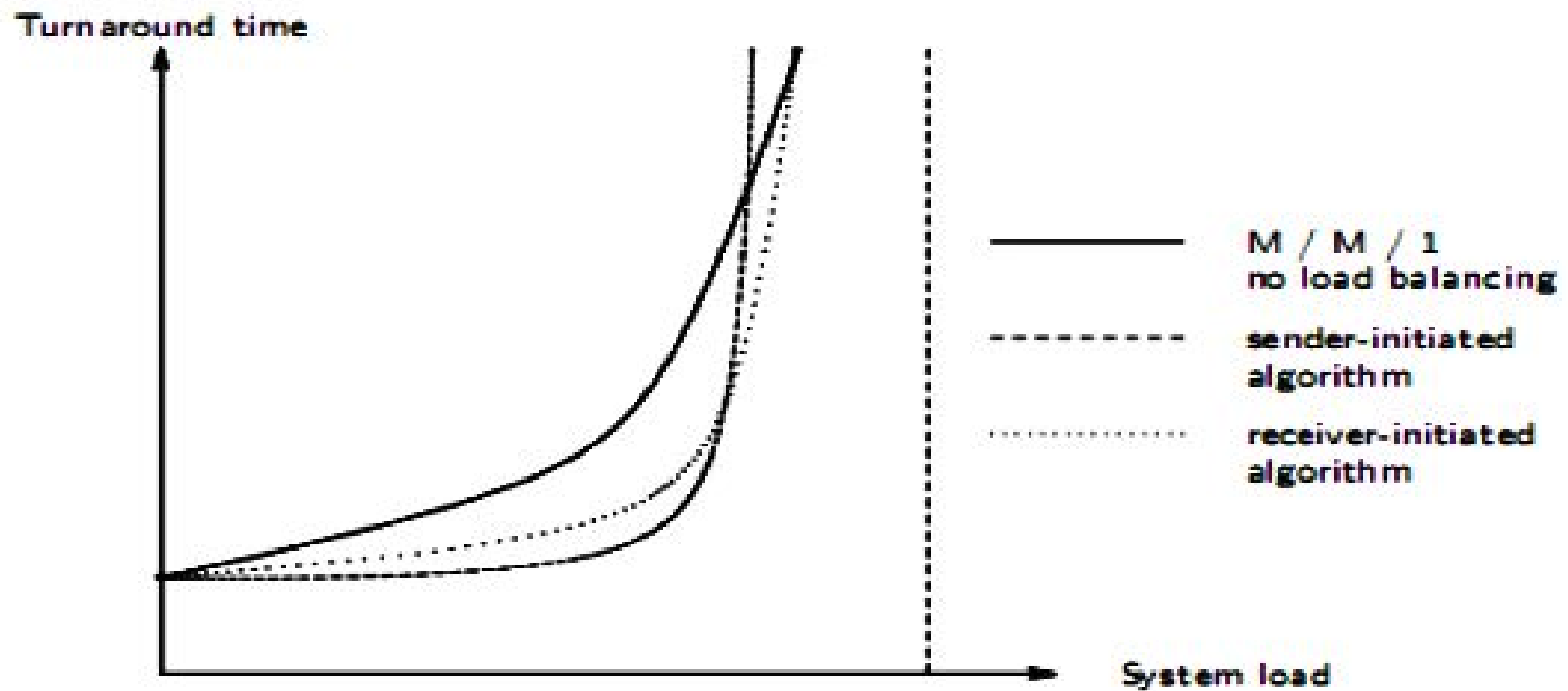
## **Sender-initiated algorithms:**

- push model
- includes probing strategy for finding a node with the smallest queue length (perhaps multicast)
- performs well on a lightly loaded system

## **Receiver-initiated algorithms:**

- pull model
  - probing strategy can also be used
  - more stable
  - perform on average better
- 
- Combinations of both algorithms are possible: choice based on the estimated system load information or reaching threshold values of the processing nodes queue.

# PERFORMANCE COMPARISON



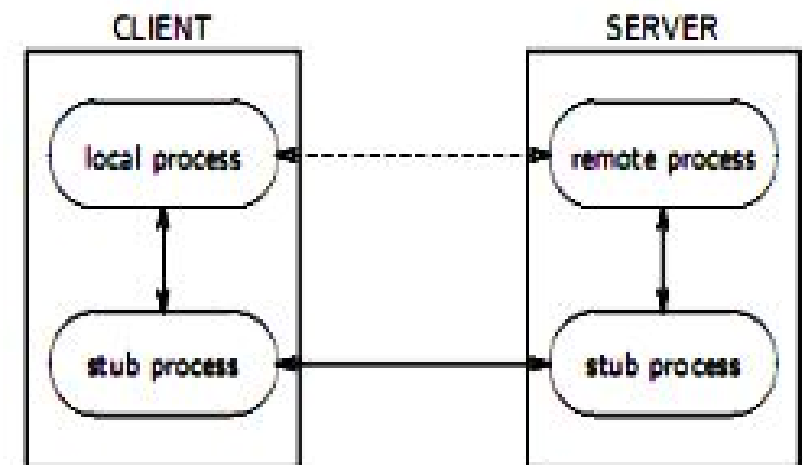
Performance comparison of dynamic load-sharing algorithms



# DISTRIBUTED PROCESS IMPLEMENTATION

Three significant application scenarios:

- **Remote service:** The message is interpreted as a request for a known service at the remote site (constrained only to services that are supported at the remote host)
  1. Remote procedure calls at the language level
  2. Remote commands at the operating system level
  3. Interpretive messages at the application level



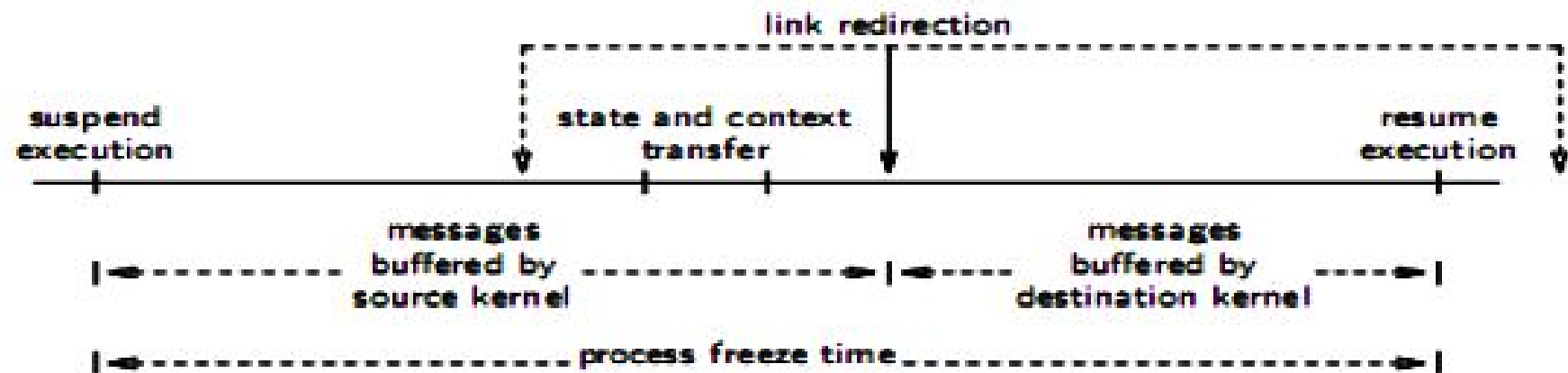
Logical model of local and remote processes

# DISTRIBUTED PROCESS IMPLEMENTATION CONT...

- **Remote execution:** The messages contain a program to be executed at the remote site; implementation issues:
  - load sharing algorithms (sender-initiated, registered hosts, broker...)
  - location independence of all IPC mechanisms including signals
  - system heterogeneity (object code, data representation)
  - protection and security
- **Process migration:** The messages represent a process being migrated to the remote site for continuing execution (extension of load-sharing by allowing a remote execution to be preempted)

# DISTRIBUTED PROCESS IMPLEMENTATION CONT...

State information of a process in distributed systems consists of two parts: *computation state* (similar to conventional context switching) and *communication state* (status of the process communication links and messages in transit). The transfer of the communication state is performed by link redirection and message forwarding.



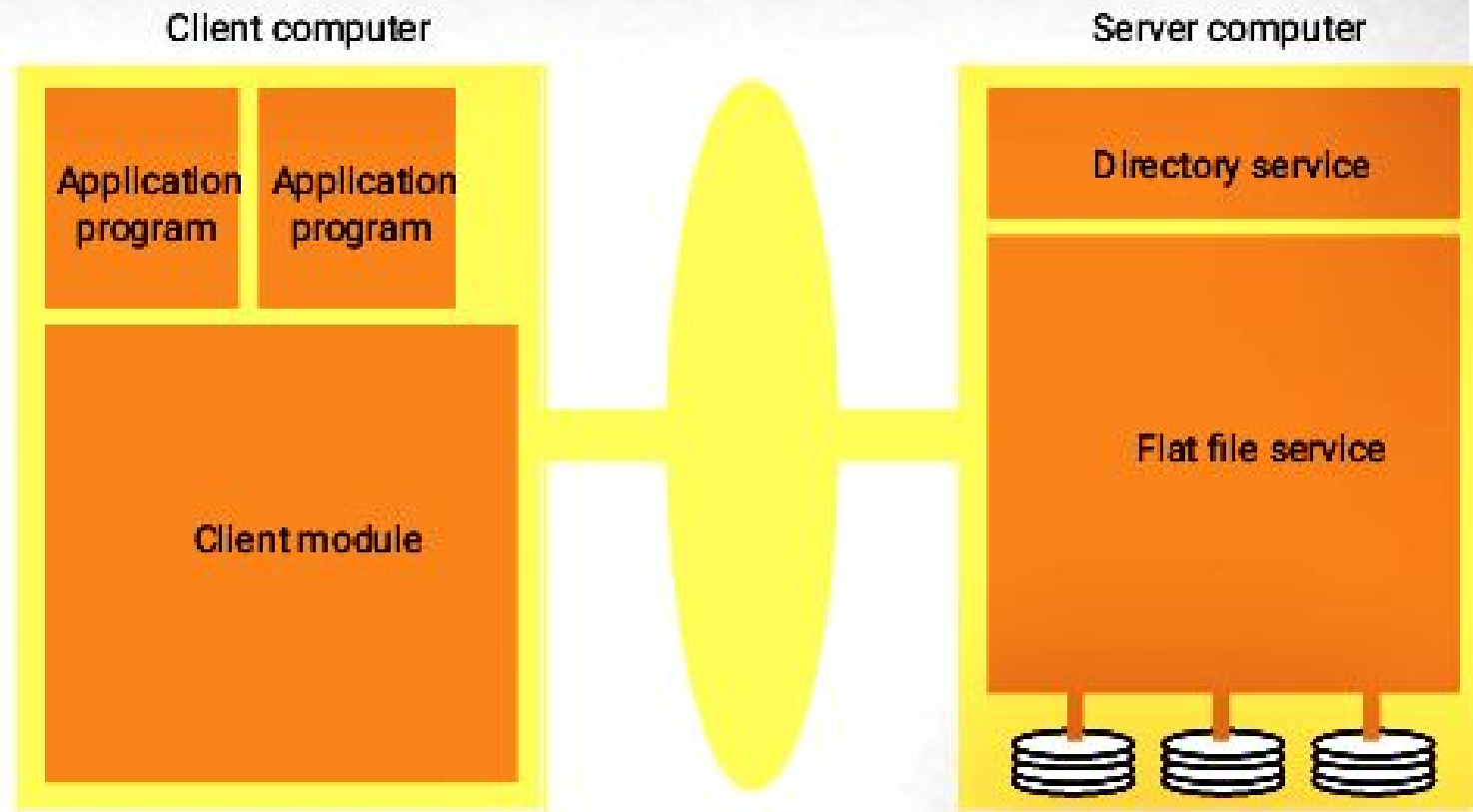
Link redirection and message forwarding

# DISTRIBUTED FILE SYSTEMS (DFS)

- A Distributed File System (DFS) is a file system that supports sharing of files and resources in the form of persistent storage over a network.
- First file servers were developed in the 1970s.
- First widely used distributed file system was Sun's Network File System (NFS) introduced in 1985.
- Examples of distributed file systems: Andrew File System (CMU), CODA (CMU), Google File System(Google)

# FILE SERVICE ARCHITECTURE

- An architecture that offers a clear separation of the main concerns in providing access to files is obtained by structuring the file service as three components:
  - A flat file service
  - A directory service
  - A client module.



# FILE SERVICE ARCHITECTURE

- The Client module implements exported interfaces by flat file and directory services on server side.
- Responsibilities of various modules can be defined as follows:
  - Flat file service:
    - ❖ Concerned with the implementation of operations on the contents of file. Unique File Identifiers (UFIDs) are used to refer to files in all requests for flat file service operations. UFIDs are long sequences of bits chosen so that each file has a unique among all of the files in a distributed system.
  - Directory service:
    - ❖ Provides mapping between text names for the files and their UFIDs. Clients may obtain the UFID of a file by quoting its text name to directory service. Directory service supports functions needed generate directories, to add new files to directories.

# FILE SERVICE ARCHITECTURE

## Client module:

- ❖ It runs on each computer and provides integrated service (flat file and directory) as a single API to application programs. For example, in UNIX hosts, a client module emulates the full set of Unix file operations.
- ❖ It holds information about the network locations of flat-file and directory server processes; and achieve better performance through implementation of a cache of recently used file blocks at the client.

# FILE OPERATIONS

<b>-Read(<i>FileId</i>, <i>i</i>, <i>n</i>) -&gt; <i>Data</i></b>	If $1 \leq i \leq \text{Length}(\text{File})$ : Reads a sequence of up to <i>n</i> items
<b>-throws Bad Position</b>	From a file starting at item <i>i</i> and returns it in <i>Data</i> .
<b>-Write(<i>FileId</i>, <i>i</i>, <i>Data</i>)</b>	If $1 \leq i \leq \text{Length}(\text{File}) + 1$ : Write a sequence of <i>Data</i> to a
<b>-throws BadPosition</b>	File, starting at item <i>i</i> , extending the file if necessary.
<b>-Create() -&gt; <i>FileId</i></b>	Creates a new file of length 0 and delivers a UFID for it.
<b>-Delete(<i>FileId</i>)</b>	Removes the file from the file store.
<b>-GetAttributes(<i>FileId</i>) -&gt; <i>Attr</i></b>	Returns the file attributes for the file.
<b>-SetAttributes(<i>FileId</i>, <i>Attr</i>)</b>	Sets the file attributes (only those attributes that are not shaded.)



# CHARACTERISTICS OF FILE SYSTEMS

- File systems are responsible for the organization, storage, retrieval, naming, sharing and protection of files.
- Files contains both data and attributes
- File systems are designed to store and manage large numbers of files, with facilities for creating, naming and deleting files

# FILE ATTRIBUTE RECORD STRUCTURE

File length
Creation timestamp
Read timestamp
Write timestamp
Attribute timestamp
Reference count
Owner
File type
Access control list

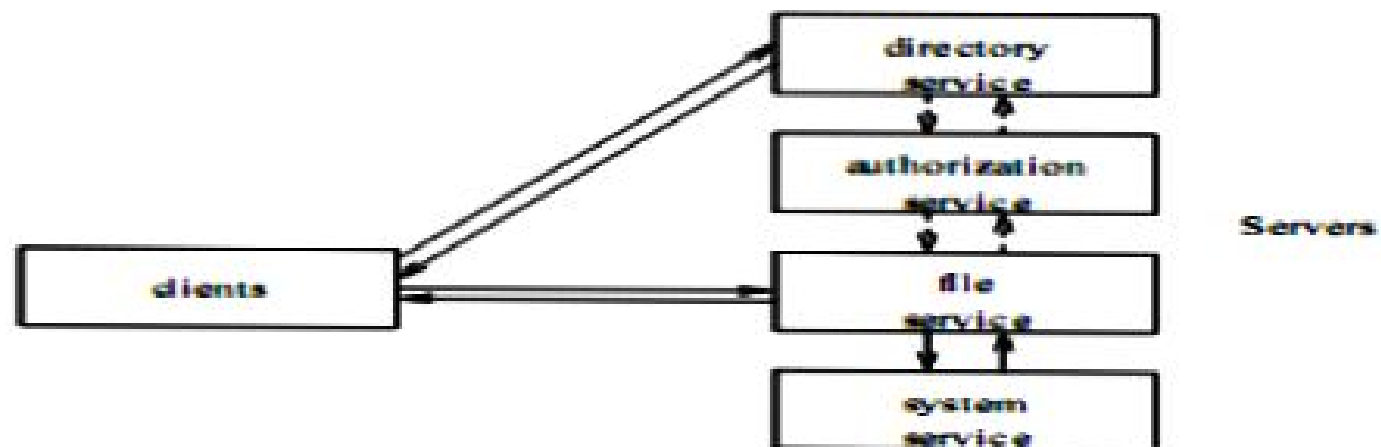
# DESIGN & IMPLEMENTATION

- It is one of the two important components (process and file) in any distributed computation.
- It is a good example for illustrating the concept of transparency and client/server model.
- File sharing and data replication present many interesting research problems.
- File: sequential, direct, indexed, indexed-sequential
- File System: flat, hierarchical

## Structure of and access to a file system

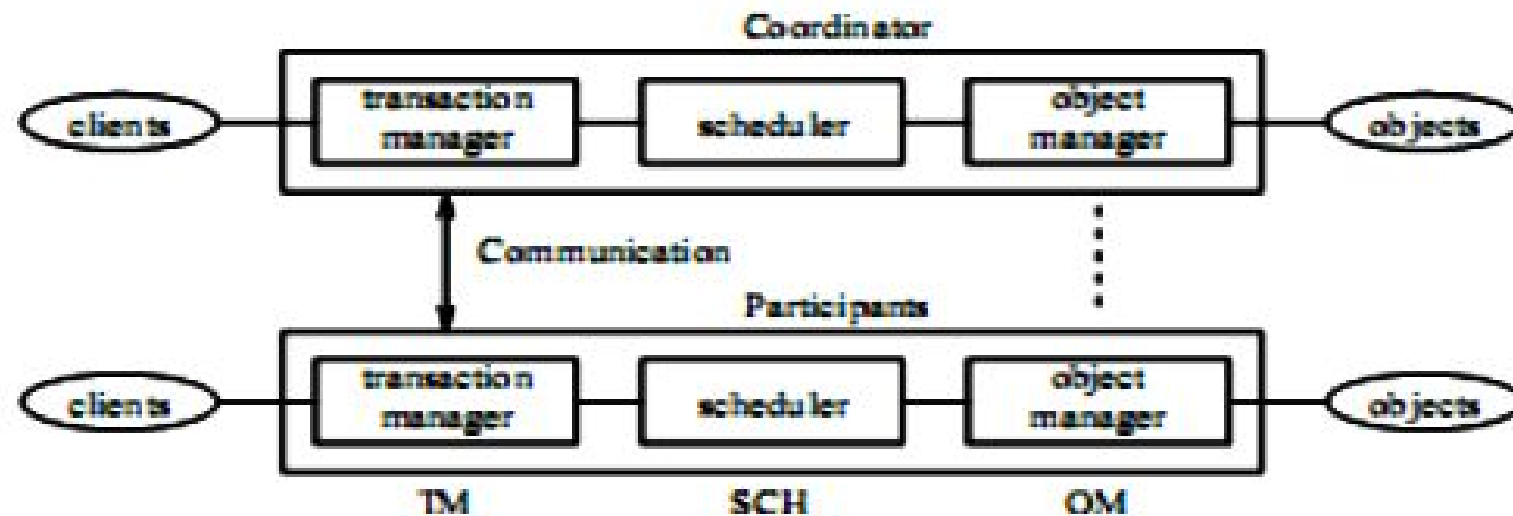
directory service		name resolution, add and deletion of files
authorization service		capability and/or access control list
file service	transaction	concurrency and replication management
	basic	read/write files and get/set attributes
system service		device, cache, and block management

### File Service Interactions



# TRANSACTION AND CONCURRENCY CONTROL

- The conventional definition of a transaction ACID properties:
  - **Atomicity** – either all tasks in a transaction are performed, or none of them are;
  - **Consistency** – data is in a consistent state when the transaction begins, and when it ends;
  - **Isolation** – all operations in a transaction are isolated from operations outside the transaction;
  - **Durability** – upon successful completion, the result of the transaction will persist.
- Transaction processing system (TPS)
  - Transaction manager (TM)
  - Scheduler (SCH)
  - Object manager (OM)
- Atomicity
  - All or none: TM, two-phase commit
  - Indivisible (serializable): SCH, concurrency control protocols
- Atomic update: OM, replica management



Several concurrency control mechanisms are available for maintaining consistency of data items such as: turn-taking, serialization, transactional locking mechanism, and operational transformation. Lock mechanisms, as a widely used method for concurrency control in transaction models, provide enough isolation on modified data items (Exclusive lock) to ensure there is no access to any of these data

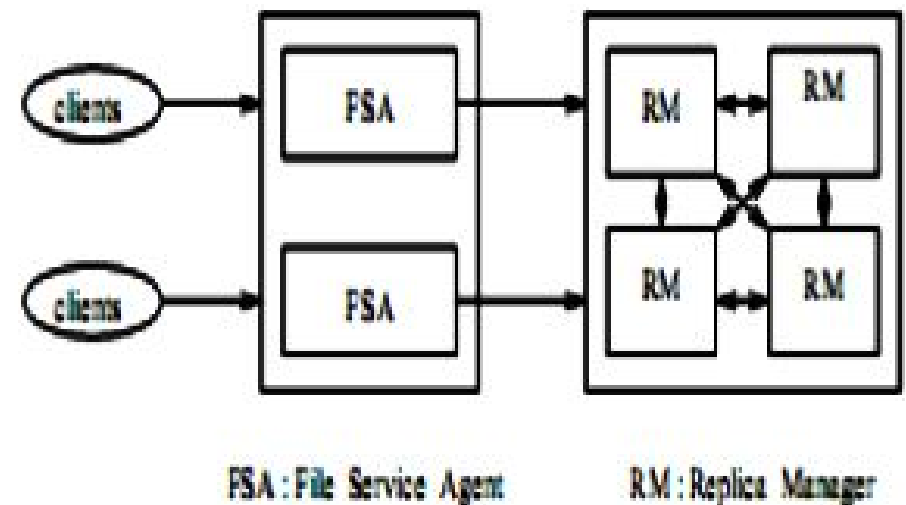
# DATA AND FILE REPLICATION

**Data replication** is a service responsible for copying/duplicating **files** and folders from a primary storage (production) to a secondary storage (**replica**).

**File level replication** is asynchronous, so files schedule/time interval changes, are collected and replicated to the secondary are first written to the primary storage and then, based on a defined storage unit.

Because data is copied with latency, primary storage failure can cause data loss if most recent modifications were not transferred to the secondary storage.

**Architecture of a replica manager:**



## Read operations

- Read-one-primary
- Read-one
- Read-quorum

## Write operation

- Write-one-primary
- Write-all
- Write-all-available
- Write-quorum
- Write-gossip

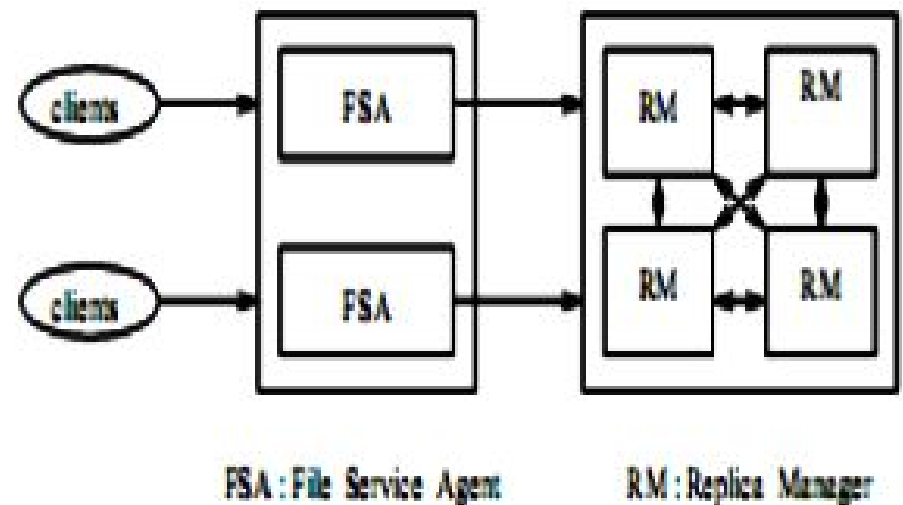
# DATA AND FILE REPLICATION

**Data replication** is a service responsible for copying/duplicating **files** and folders from a primary storage (production) to a secondary storage (**replica**).

**File level replication** is asynchronous, so files schedule/time interval changes, are collected and replicated to the secondary are first written to the primary storage and then, based on a defined storage unit.

Because data is copied with latency, primary storage failure can cause data loss if most recent modifications were not transferred to the secondary storage.

**Architecture of a replica manager:**



## Read operations

- Read-one-primary
- Read-one
- Read-quorum

## Write operation

- Write-one-primary
- Write-all
- Write-all-available
- Write-quorum
- Write-gossip



## CASE STUDIES

**Data replication** is a service responsible for copying/duplicating files and folders from a primary storage (production) to a secondary storage (**replica**).

***File level replication*** is asynchronous, so files schedule/time interval changes, are collected and replicated to the secondary are first written to the primary storage and then, based on a defined storage unit.

Because data is copied with latency, primary storage failure can cause data loss if most recent modifications were not transferred to the secondary storage.

**Architecture of a replica manager:**