INTRODUCTION TO DISTRIBUTED SYSTEMS

INTRODUCTION

The process of computation was started from working on a single processor. This uni-processor computing can be termed as **centralized computing**. As the demand for the increased processing capability grew high, multiprocessor systems came to existence. The advent of multiprocessor systems, led to the development of distributed systems with high degree of scalability and resource sharing. The modern day parallel computing is a subset of distributed computing

> ***A distributed system is a collection of independent computers, interconnected via a network, capable of collaborating on a task. Distributed computing is computing performed in a distributed system.***

A distributed system is a collection of independent entities that cooperate to solve a problem that cannot be individually solved. Distributed computing is widely used due to advancements in machines; faster and cheaper networks. In distributed systems, the entire network will be viewed as a computer. The multiple systems connected to the network will appear as a single system to the user. Thus the distributed systems hide the complexity of the underlying architecture to the user. Distributed computing is a special version of parallel computing where the processors are in different computers and tasks are distributed to computers over a network.

The definition of distributed systems deals with two aspects that:

Deals with hardware: The machines linked in a distributed system are autonomous.

Deals with software: A distributed system gives an impression to the users that they are dealing with a single system.

Features of Distributed Systems:

No common physical clock - This is an important assumption because it introduces the element of "distribution" in the system and gives rise to the inherent asynchrony amongst the processors.

No shared memory - A key feature that requires message-passing for communication. This feature implies the absence of the common physical clock.

Geographical separation – The geographically wider apart that the processors are, the more representative is the system of a distributed system**.**

Autonomy and heterogeneity – Here the processors are "loosely coupled" in that they have different speeds and each can be running a different operating system.

Issues in distributed systems

Heterogeneity

Openness

Security

Scalability

Failure handling

Concurrency

Transparency

Quality of service

QOS parameters

The distributed systems must offer the following QOS:

- Performance
- Reliability
- Availability
- Security

Differences between centralized and distributed systems

Centralized Systems	Distributed Systems
In Centralized Systems, several jobs are done on a particular central processing unit(CPU)	In Distributed Systems, jobs are distributed among several processors. The Processor are interconnected by a computer network
They have shared memory and shared variables.	They have no global state (i.e.) no shared memory and no shared variables.
Clocking is present.	No global clock.

RELATION TO COMPUTER SYSTEM COMPONENTS

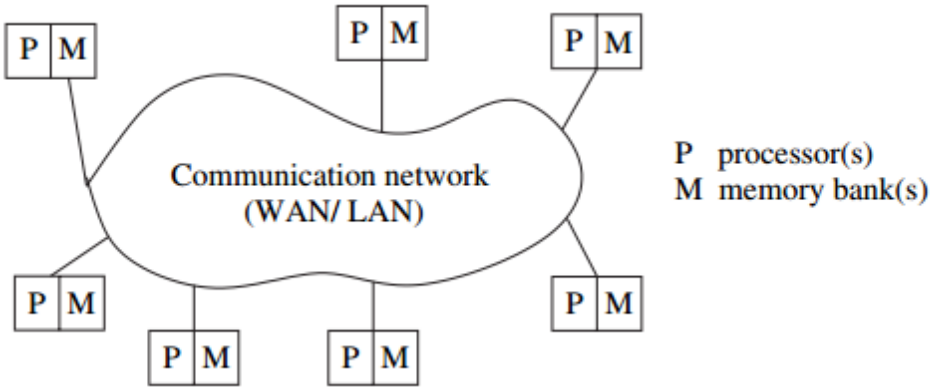

Fig : Example of a Distributed System

As shown in Fig., Each computer has a memory-processing unit and the computers are connected by a communication network. Each system connected to the distributed networks hosts distributed software which is a middleware technology. This drives the Distributed System (DS) at the same time preserves the heterogeneity of the DS. The term **computation or run** in a distributed system is the execution of processes to achieve a common goal.

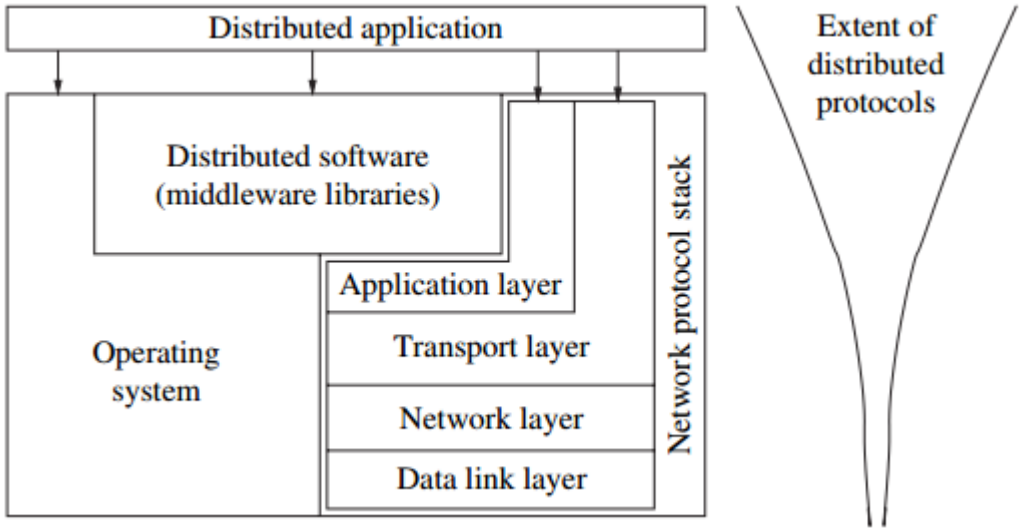

Fig : Interaction of layers of network

The interaction of the layers of the network with the operating system and middleware is shown in Fig. The middleware contains important library functions for facilitating the operations of DS.

The distributed system uses a layered architecture to break down the complexity of system design. The middleware is the distributed software that drives the distributed system, while providing transparency of heterogeneity at the platform level

Examples of middleware: Object Management Group's (OMG), Common Object Request Broker Architecture (CORBA) [36], Remote Procedure Call (RPC), Message Passing Interface (MPI)

MOTIVATION

The following are the keypoints that acts as a driving force behind DS:

Inherently distributed computations: DS can process the computations at geographically remote locations.

Resource sharing: The hardware, databases, special libraries can be shared between systems without owning a dedicated copy or a replica. This is cost effective and reliable.

Access to geographically remote data and resources: As mentioned previously, computations may happen at remote locations. Resources such as centralized servers can also be accessed from distant locations.

Enhanced reliability: DS provides enhanced reliability, since they run on multiple copies of resources. The distribution of resources at distant locations makes them less susceptible for faults.

The term reliability comprises of:

1. **Availability:** the resource/ service provided by the resource should be accessible at all times
2. **Integrity:** the value/state of the resource should be correct and consistent.
3. **Fault-Tolerance:** the ability to recover from system failures

Increased performance/cost ratio: The resource sharing and remote access features of DS naturally increase the performance / cost ratio.

Scalable: The number of systems operating in a distributed environment can be increased as the demand increases.

1.4 RELATION TO PARALLEL SYSTEMS

> ***Parallel Processing Systems divide the program into multiple segments and process them simultaneously.***

The main objective of parallel systems is to improve the processing speed.
They are sometimes known as **multiprocessor or multi computers or tightly coupled systems.**

1.4.1 Characteristics of parallel systems

A parallel system may be broadly classified as belonging to one of three types:

1. A *multiprocessor system*
2. A *multicomputer parallel system*
3. *Array processors*

1. A *multiprocessor system*

A *multiprocessor system* is a parallel system in which the multiple processors have *direct access to shared memory* which forms a common address space. The architecture is shown in Figure 1.3(a). Such processors usually do not have a common clock.

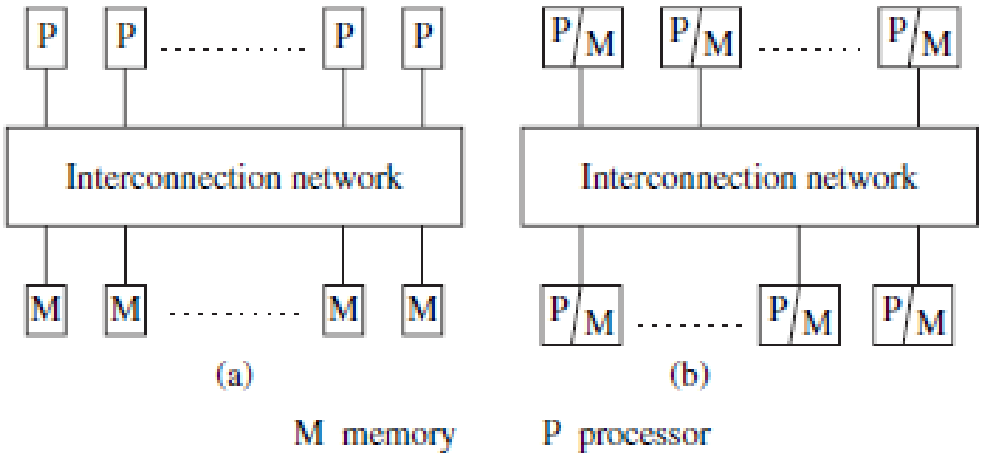

Figure 1.3 Two standard architectures for parallel systems. (a) Uniform memory access (UMA) multiprocessor system. (b) Non-uniform memory access (NUMA) multiprocessor.

Standard Architectures for parallel systems

i) Uniform Memory Access (UMA)

- Single memory controller is used
- Here, all the processors share the physical memory in a centralized manner with equal access time to all the memory words.
- Slower than NUMA

ii) Non-uniform Memory Access (NUMA)

- In NUMA multiprocessor model, the access time varies with the location of the memory word.
- Here, the shared memory is physically distributed among all the processors, called local memories.
- NUMA systems also share CPUs and the address space, but each processor has a local memory, visible to all other processors.
- In NUMA systems access to local memory blocks is quicker than access to remote memory blocks.

- Different memory controllers are used
- Faster than UMA

Types of Multiprocessors systems

- When all the processors have equal access to all the peripheral devices, the system is called a **symmetric multiprocessor.**
- When only one or a few processors can access the peripheral devices, the system is called an **asymmetric multiprocessor.**

Multi Stage Interconnection Networks

- Composed of Processing elements, memory elements and switching elements.
- Switching elements are used to connect PEs with MEs
- Switching elements themselves are usually connected to each other in stages

- Figure 1.4 shows two popular interconnection networks – the Omega network and the Butterfly network, each of which is a multi-stage network formed of 2×2 switching elements.
- Each 2×2 switch allows data on either of the two input wires to be switched to the upper or the lower output wire.
- In a single step, however, only one data unit can be sent on an output wire.
- So if the data from both the input wires is to be routed to the same output wire in a single step, there is a collision.

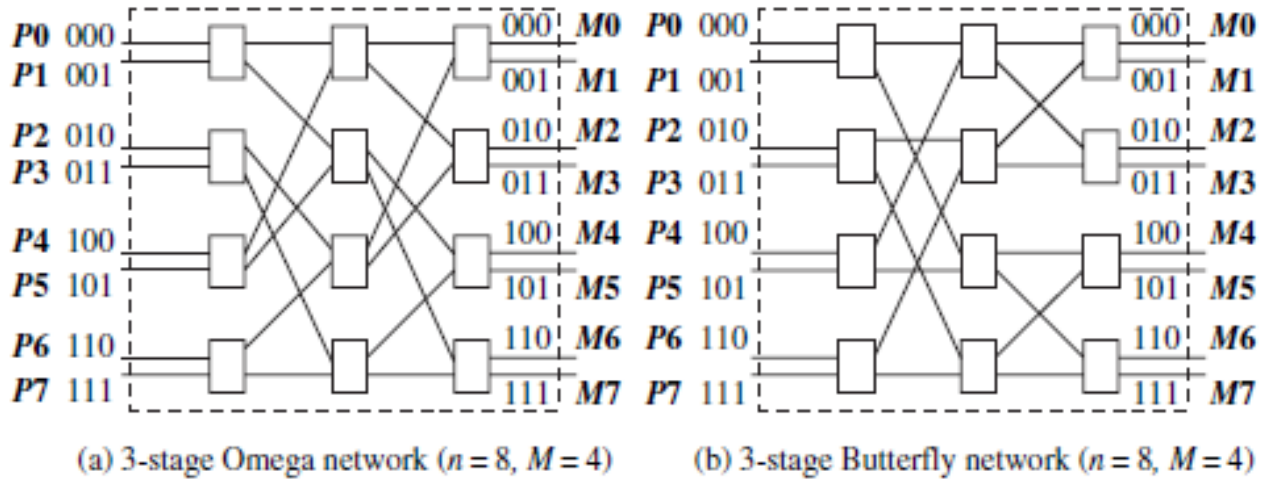

Figure 1.4 Interconnection networks for shared memory multiprocessor systems

Omega interconnection function

The Omega network which connects n processors to n memory units has n/2log2 n switching elements of size 2×2 arranged in log2 n stages.

Between each pair of adjacent stages of the Omega network, a link exists between output i of a stage and the input j to the next stage according to the following perfect shuffle pattern which is a left-rotation operation

The Iterative generation function is

$$j = \begin{cases} 2i, & \text{for } 0 \le i \le n/2 - 1, \\ 2i + 1 - n, & \text{for } n/2 \le i \le n - 1. \end{cases}$$

- The routing function from input line i to output line j considers only j and the stage number s, where s ∈ [0, log n – 1].

- In a stage s switch, if the s + 1th most significant bit of j is 0, the data is routed to the upper output wire, otherwise it is routed to the lower output wire.

Butterfly network

A butterfly network links multiple computers into a high-speed network. For a butterfly network with n processor nodes, there need to be n (log n + 1) switching nodes. The generation of the interconnection pattern between a pair of adjacent stages depends not only on n but also on the stage numbers.In a stage (s) switch, if the s + 1th MSB of j is 0, the data is routed to the upper output wire, otherwise it is routed to the lower output wire.

2. A *multicomputer parallel system*

It is a parallel system in which the multiple processors *do not have direct access to shared memory*. The memory of the multiple processors may or may not form a common address space. Such computers usually do not have a common clock. The architecture is shown in Figure 1.3(b).

Torus or 2D Mesh Topology

A k X k mesh will contain k^2processor with maximum path length as 2*(k/2 -1). Every unit in the torus topology is identified using a unique label, with dimensions distinguished as bit positions.

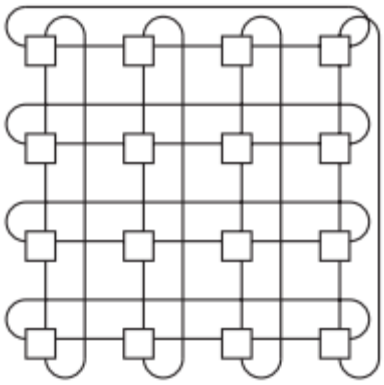

Fig 1.5: 2-D Mesh

Hypercube

The path between any two nodes in 4-D hypercube is found by Hamming distance. Routing is done in hop to hop fashion with each adjacent node differing by one bit label. This topology has good congestion control and fault tolerant mechanism.

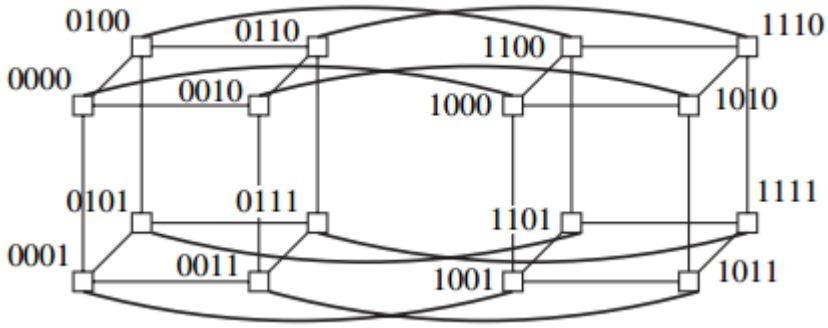

Fig 1.6: 4-D Hypercube

3. Array Processors

They are a class of processors that executes one instruction at a time in an array or table of data at the same time rather than on single data elements on a common clock.

They are also known as vector processors. An array processor implement the instruction set where each instruction is executed on all data items associated and then move on the other instruction. Array elements are incapable of operating autonomously, and must be driven by the control unit.

1.4.2 Flynn's Taxonomy

Flynn's taxonomy is a specific classification of parallel computer architectures that are based on the number of concurrent instruction (single or multiple) and data streams (single or multiple) available in the architecture.

Flynn's taxonomy based on the number of instruction streams and data streams are the following:

1. (SISD) single instruction, single data
2. (MISD) multiple instruction, single data
3. (SIMD) single instruction, multiple data
4. (MIMD) multiple instruction, multiple data

1. SISD (Single Instruction, Single Data stream)

- Single Instruction, Single Data (SISD) refers to an Instruction Set Architecture in which a single processor (one CPU) executes exactly one instruction stream at a time.
- It also fetches or stores one item of data at a time to operate on data stored in a single memory unit.
- Most of the CPU design is based on the von Neumann architecture and the follow SISD.
- The SISD model is a non-pipelined architecture with general-purpose registers, Program Counter (PC), the Instruction Register (IR), Memory Address Registers (MAR) and Memory Data Registers (MDR).

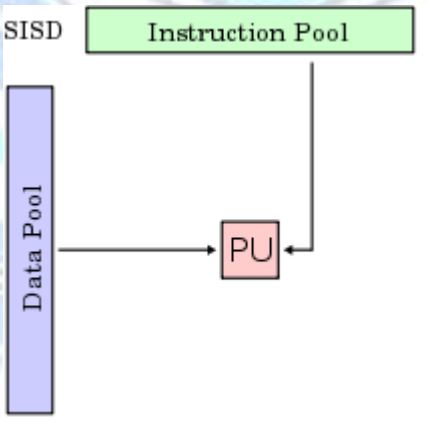

Fig 1.7: Single Instruction, Single Data Stream

SIMD (Single Instruction, Multiple Data streams)

- Single Instruction, Multiple Data (SIMD) is an Instruction Set Architecture that have a single control unit (CU) and more than one processing unit (PU) that operates like a von Neumann machine by executing a single instruction stream over PUs, handled through the CU. The CU generates the control signals for all of the PUs and by which executes the same operation on different data streams.
- The SIMD architecture is capable of achieving data level parallelism.

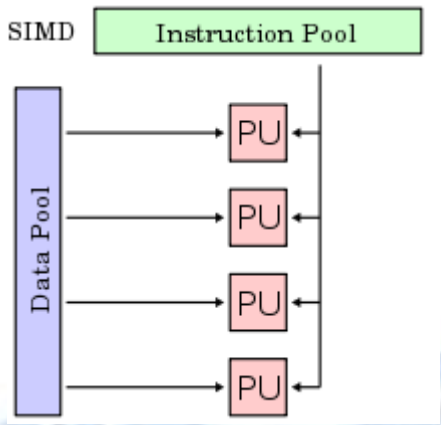

Fig 1.8:Single Instruction, Multiple Data streams

MISD (Multiple Instruction, Single Data stream)

- Multiple Instruction, Single Data (MISD) is an Instruction Set Architecture for parallel computing where many functional units perform different operations by executing different instructions on the same data set.
- This type of architecture is common mainly in the fault-tolerant computers executing the same instructions redundantly in order to detect and mask errors.

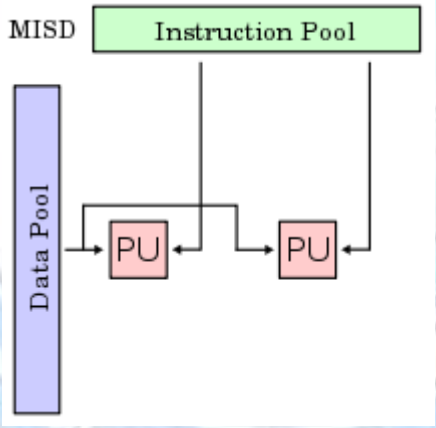

Fig 1.9: Multiple Instruction, Single Data stream

MIMD (Multiple Instruction, Multiple Data streams)

- Multiple Instruction stream, Multiple Data stream (MIMD) is an Instruction Set Architecture for parallel computing that is typical of the computers with multiprocessors.
- Using the MIMD, each processor in a multiprocessor system can execute asynchronously different set of the instructions independently on the different set of data units.
- The MIMD based computer systems can used the shared memory in a memory pool or work using distributed memory across heterogeneous network computers in a distributed environment.
- The MIMD architectures is primarily used in a number of application areas such as computer-aided design/computer-aided manufacturing, simulation, modelling, communication switches etc.

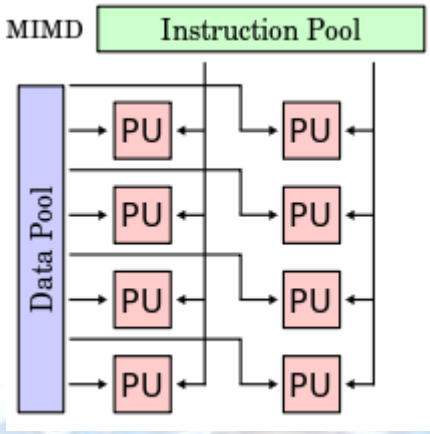

Fig 1.10:Multiple Instruction, Multiple Data streams

	Single	**Multiple**
Single	**SISD** Von Neumann Single computer	**MISD** May be pipelined computers
Multiple	**SIMD** Vector processors Fine grained data Parallel computers	**MIMD** Multi computers Multiprocessors

1.2.4 Coupling, parallelism, concurrency, and granularity

Coupling

Coupling is the strength of interconnection between two software modules: the higher the strength of interconnection, the higher the coupling.

> The degree of coupling among a set of modules, whether hardware or software, is measured in terms of the interdependency and binding among the modules.

The multiprocessor systems are classified into two types based on coupling:

1. Loosely coupled systems
2. Tightly coupled systems

Tightly Coupled systems:

- Tightly coupled multiprocessor systems contain multiple CPUs that are connected at the bus level with both local as well as central shared memory.
- Tightly coupled systems perform better, due to faster access to memory and intercommunication and are physically smaller and use less power. They are economically costlier.

Loosely Coupled systems:

- Loosely coupled multiprocessors consist of distributed memory where each processor has its own memory and IO channels.
- The processors communicate with each other via message passing or interconnection switching.
- Loosely coupled systems are less costly than tightly coupled systems, but are physically bigger and have a low performance compared to tightly coupled systems.

Parallelism or speedup of a program on specific system

- Problems are broken down into instructions and are solved concurrently as each resource which has been applied to work is working at the same time.

- This is a measure of the relative speedup of a specific program, on a given machine. The speedup depends on the number of processors and the mapping.

Parallelism within a parallel/distributed program

- This is an aggregate measure of the percentage of time that all the processors are executing CPU instructions productively.

Concurrency

- The *parallelism/ concurrency* in a parallel/distributed program can be measured by the ratio of the number of local operations to the total number of operations, including the communication or shared memory access operations.

Granularity

Granularity or grain size is a measure of the amount of work or computation that is performed by that task.

- Granularity is also the communication overhead between multiple processors or processing elements.

Parallelism can be classified into three categories based on work distribution among the parallel tasks:

1. **Fine-grained 2. Coarse-grained parallelism**. 3. **Medium-grained**

Classes of OS of Multiprocessing systems:

- **Network Operating Systems**
- **Distributed Operating systems**
- **Multiprocessor Operating Systems**