


Characteristics of Distributed Systems


- No common physical clock
- No common shared memory
- Geographical separation
- Autonomy & Heterogeneity

Connects processes via a communication network.

Motivation :

- Inherently distributed computation 
- Resource sharing
- Access to remote resources
- Increased performance/cost ratio
- Reliability
- Scalability
- Modularity
- Incremental expandability.

Challenges : (System Perspective)

- Communication mechanism - RPC, ROI, ...?
- Design of distributed programs, thread/process management etc.
- Universal  nomenclature for easy identification of resources & processes.
- Synchronization
- Data storage & access
- Consistency & replication
- Fault-tolerance
- Security & transparency
- Scalable & modular algorithm design.
- API building

Challenges : (Algorithm Perspective)

- Maintaining global state & time
- Sync. & coordination mechanisms.
- Group comm. & ordered msg. delivery
- Monitoring events & predicates
- Distributed program design & verification tools
- Debugging distributed programs.
- Shared memory abstraction
- Reliable & fault-tolerant systems
- Load balancing.
- Real-time scheduling
- Performance analysis

Applications :

- Mobile systems
- P2P computing
- Sensor networks
- Ubiquitous & pervasive computing
- Pub/Sub, CDN
- Distributed agents.
- Distributed data mining
- Grid computing

1.4.2 Flynn's taxonomy

Flynn [14] identified four processing modes, based on whether the processors execute the same or different instruction streams at the same time, and whether or not the processors processed the same (identical) data at the same time. It is instructive to examine this classification to understand the range of options used for configuring systems:

- **Single instruction stream, single data stream (SISD)**

This mode corresponds to the conventional processing in the von Neumann paradigm with a single CPU, and a single memory unit connected by a system bus.

- **Single instruction stream, multiple data stream (SIMD)**

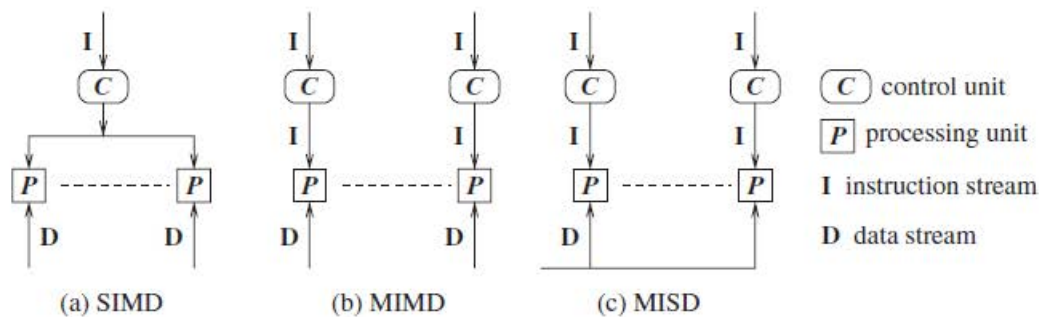
This mode corresponds to the processing by multiple homogenous processors which execute in lock-step on different data items. Applications that involve operations on large arrays and matrices, such as scientific applications, can best exploit systems that provide the SIMD mode of operation because the data sets can be partitioned easily.

Several of the earliest parallel computers, such as Illiac-IV, MPP, CM2, and MasPar MP-1 were SIMD machines. Vector processors, array processors' and systolic arrays also belong to the SIMD class of processing. Recent SIMD architectures include co-processing units such as the MMX units in Intel processors (e.g., Pentium with the streaming SIMD extensions (SSE) options) and DSP chips such as the Sharc [22].

- **Multiple instruction stream, single data stream (MISD)**

This mode corresponds to the execution of different operations in parallel on the same data. This is a specialized mode of operation with limited but niche applications, e.g., visualization.

Figure 1.6 Flynn's taxonomy of SIMD, MIMD, and MISD architectures for multiprocessor/multicomputer systems.



- **Multiple instruction stream, multiple data stream (MIMD)**

In this mode, the various processors execute different code on different data. This is the mode of operation in distributed systems as well as in the vast majority of parallel systems. There is no common clock among the system processors. Sun Ultra servers, multicomputer PCs, and IBM SP machines are examples of machines that execute in MIMD mode.

SIMD, MISD, and MIMD architectures are illustrated in Figure 1.6. MIMD architectures are most general and allow much flexibility in partitioning code and data to be processed among the processors. MIMD architectures also include the classically understood mode of execution in distributed systems.

| Aspect | Message-Passing Systems | Shared Memory Systems |
|------------------------|--|--|
| Communication Model | Processes communicate via messages. | Processes share a common memory space. |
| Communication Overhead | Higher overhead due to message passing. | Lower communication overhead as data is shared directly. |
| Synchronization | Often requires explicit synchronization mechanisms like locks or semaphores. | Implicit synchronization as processes access shared data. |
| Scalability | Typically scales well for a large number of processes. | May face scalability challenges when many processes access shared memory simultaneously. |
| Example | MPI (Message Passing Interface) used in high-performance computing. | OpenMP for shared memory parallelism in multi-core processors. |

| Aspect | Synchronous Executions | Asynchronous Executions |
|--------------------------|--|---|
| Timing Control | Processes proceed in lockstep, waiting for each other to complete specific steps before moving forward. | Processes operate independently without waiting for others, allowing for potentially faster execution. |
| Coordination | Requires strict coordination and synchronization among processes, often using barriers or synchronization primitives. | Processes coordinate through message passing or callbacks, reducing the need for strict synchronization. |
| Complexity | Simpler to reason about as steps are executed predictably and sequentially. | More complex due to potential race conditions and non-deterministic behavior. |
| Latency Tolerance | Less tolerant of latency, as processes must wait for each other, which can lead to performance bottlenecks. | More tolerant of latency, as processes can continue working without waiting, potentially improving overall system responsiveness. |
| Example | A traditional classroom lecture, where students wait for the instructor to finish one topic before moving to the next. | A web server handling multiple client requests simultaneously, with each request processed independently. |