Scalability and Performance Studies

---

1. Definition

Scalability refers to the **ability of a parallel or distributed system to effectively handle increased workload** by adding more resources (processors, memory, or nodes).

Performance studies involve **analyzing and measuring system efficiency, speedup, and resource utilization** to understand how well a system or algorithm performs as workload or resources change.

- **Goal:** Ensure **efficient utilization of hardware** while solving larger problems.

---

2. Types of Scalability

Type	Description	Example
Strong Scalability	System solves **same problem faster** as processors increase	Summing 1 million numbers using 1, 2, 4, 8 processors; execution time decreases
Weak Scalability	System solves **proportionally larger problem** with more processors while keeping execution time roughly constant	Processing 1M numbers on 1 processor, 2M on 2 processors, etc.
Cost-Optimal Scalability	Adding resources results in **proportional performance improvement** without extra overhead	Doubling processors nearly halves execution time

---

3. Performance Metrics

3. Performance Metrics

1. **Speedup (S)**
   - Ratio of serial execution time T_s to parallel execution time T_p
   - $$S = \frac{T_s}{T_p}$$
   - Ideal speedup: $S = p$ (number of processors)
2. **Efficiency (E)**
   - How effectively processors are used
   - $$E = \frac{S}{p} = \frac{T_s}{p \cdot T_p}$$
   - $0 < E \leq 1$; closer to 1 is better

1. **Throughput**
   - Number of tasks completed per unit time
2. **Latency**
   - Time taken to complete a single task
3. **Scalability Factor**
   - Measures how performance improves with added resources

---

4. Illustrative Example

Problem: Sum an array of 16 numbers

Processors	Execution Time (T_p)	Speedup S = T_s/T_p	Efficiency E = S/p
1	16 ms	1	1
2	9 ms	1.78	0.89
4	5 ms	3.2	0.8
8	3 ms	5.33	0.67

- Observation: Efficiency decreases as processors increase due to **communication and synchronization overhead**.

---

5. Behavior in Parallel and Distributed Systems

Feature	Parallel Systems	Distributed Systems
Scaling Limitations	Memory bus contention, core limits	Network latency, bandwidth, node failure
Communication Overhead	Low; shared memory	Higher; message passing required
Performance Bottlenecks	Critical sections, load imbalance	Network congestion, load imbalance, synchronization delays
Load Balancing	Threads share tasks evenly	Tasks must be distributed across nodes carefully
Use Cases	Multithreaded algorithms, HPC within single machine	Cloud computing, distributed simulations, large data processing

---

6. Key Points

- **Scalability** ensures algorithms can **handle larger problems or resources efficiently**.
- **Speedup and efficiency** are key metrics to measure performance.
- **Overhead** from communication and synchronization reduces efficiency in distributed systems.
- **Performance studies** guide algorithm design, resource allocation, and system tuning.

Scheduling in Parallel and Distributed Systems

---

1. Definition

Scheduling is the process of **assigning tasks or processes to processors or nodes** in a parallel or distributed system to **optimize performance, resource utilization, and execution time**.

- Goal: Minimize **completion time (makespan)** and **communication overhead**, and **balance load** across processors.
- Essential in systems where tasks have **dependencies, varying execution times, or communication requirements**.

---

2. Key Concepts

Concept	Description
Task	Unit of computation to be scheduled
Processor / Node	Hardware unit that executes tasks
Makespan	Total time to complete all tasks
Load Balancing	Even distribution of tasks among processors
Preemptive Scheduling	Tasks can be interrupted and resumed later
Non-Preemptive Scheduling	Tasks run to completion once started

---

3. Types of Scheduling

A. Static Scheduling

- **Definition:** Assignment of tasks is **fixed before execution**.
- **Example:** Assigning 4 matrix blocks to 4 processors in advance.
- **Pros:** Simple, low runtime overhead
- **Cons:** Inflexible to dynamic workloads or delays

B. Dynamic Scheduling

- **Definition:** Tasks are assigned **at runtime** based on processor availability.
- **Example:** Work-stealing in OpenMP: idle threads take tasks from busy threads.
- **Pros:** Adaptable to varying workloads
- **Cons:** Higher runtime overhead, requires monitoring

C. List Scheduling

- **Definition:** Tasks are sorted by priority and assigned to available processors.
- **Example:** DAG-based tasks with dependency-aware priorities

D. Task Graph / DAG Scheduling

- **Definition:** Scheduling based on **task dependencies** represented in a DAG.
- **Example:** Node A → Node B → Node C; scheduler ensures dependencies respected

E. Gang Scheduling

- **Definition:** A group of related tasks (e.g., threads of a parallel job) are scheduled **simultaneously on different processors**.
- **Use:** Time-shared parallel systems

---

4. Illustrative Example

Problem: Execute 4 tasks T1–T4 on 2 processors

DAG with dependencies:

- T1 → T3
- T2 → T4

Static Scheduling:

- P1: T1 → T3
- P2: T2 → T4

Dynamic Scheduling:

- P1 starts T1, P2 starts T2
- When T1 finishes, P1 takes T4 if ready, etc.

Goal: Minimize makespan while respecting dependencies

---

5. Behavior in Parallel and Distributed Systems

Feature	Parallel (Shared Memory)	Distributed (Cluster/Nodes)
Task Assignment	Threads scheduled by OS or runtime (OpenMP)	Explicit by scheduler or master node (MPI, Spark)
Communication Overhead	Minimal; shared memory	Significant; depends on network
Synchronization	Barrier or critical section	Message passing to coordinate task completion
Load Balancing	OS/runtime handles small differences	Scheduler must consider network delays, task size
Scalability	Limited to cores per machine	Can scale across hundreds/thousands of nodes
Use Cases	Multithreaded simulations, HPC	Distributed databases, cloud computing, big data processing

---

6. Key Points

- **Scheduling ensures efficient utilization of processors and minimizes completion time**.
- **Static scheduling:** simple, good for predictable workloads
- **Dynamic scheduling:** adaptable, handles irregular or unpredictable workloads
- Scheduling is critical in **both parallel and distributed systems**, especially for **task dependencies and communication costs**.

Storage Systems in Parallel and Distributed Computing

---

1. Definition

A **Storage System** is the **hardware and software infrastructure used to store, manage, and retrieve data** in a parallel or distributed system.

- Ensures **data availability, reliability, and high performance**.
- Key in **parallel systems** for fast shared access and in **distributed systems** for scalable and fault-tolerant storage.

---

2. Types of Storage Systems

A. Shared Memory Storage

- **Definition:** Single memory space accessible by all processors.
- **Used in:** Multi-core CPUs, SMP systems.
- **Example:** RAM shared among threads in OpenMP.
- **Behavior:** Fast access, low latency, requires synchronization for data consistency.

B. Distributed Storage

- **Definition:** Each node has **local memory or disk**, and data is shared via **network communication**.
- **Used in:** HPC clusters, cloud storage, grid computing.
- **Example:** HDFS (Hadoop Distributed File System), Ceph storage.
- **Behavior:** Scalable, fault-tolerant, communication overhead exists.

C. Cache Memory

- **Definition:** Small, fast memory close to the processor to reduce access time.
- **Used in:** Multi-level caches (L1, L2, L3) in CPUs.
- **Behavior:** Improves performance; consistency must be maintained in parallel access.

D. Disk Storage

- **Definition:** Persistent storage for large datasets.
- **Used in:** Local disks or networked storage (NAS, SAN).
- **Behavior:** High capacity, slower than memory; I/O optimization needed.

E. Hybrid Storage

- **Definition:** Combination of fast local memory and distributed persistent storage.
- **Used in:** Big data frameworks and HPC clusters.
- **Example:** Spark uses in-memory RDDs with HDFS for persistent storage.

---

3. Illustrative Example

Problem: Parallel matrix multiplication storing partial results

- **Shared Memory:** Threads write results directly to shared array → fast, requires locks/atomic operations.
- **Distributed Memory:** Each node computes partial matrix → sends results to master → aggregated → network communication overhead.
- **Cache Usage:** Partial results kept in CPU cache for faster access → reduces main memory access.
- **Disk Storage:** Final result stored on disk/HDFS for persistence.

---

4. Behavior in Parallel and Distributed Systems

Feature	Parallel Systems (Shared Memory)	Distributed Systems (Cluster/Nodes)
Memory Access	Direct via shared memory	Local memory; data communicated explicitly
Latency	Very low	Higher due to network
Consistency	Maintained via locks or atomic operations	Requires replication protocols (e.g., Paxos, Raft)

Fault Tolerance	Low; single memory failure affects all	High; replication and redundancy mitigate node failures
Scalability	Limited by memory bus and core count	Highly scalable across multiple nodes
Use Cases	Multithreaded applications, simulations	Big data processing, cloud storage, distributed databases

---

5. Key Points

- **Shared memory storage:** fast, suitable for tightly-coupled parallel systems
- **Distributed storage:** scalable, fault-tolerant, suitable for clusters and cloud
- **Cache & hybrid storage:** improve performance by combining fast local memory with persistent storage
- **Storage design** affects **performance, consistency, and scalability** of parallel/distributed applications