Concurrency control, Fault tolerance

Concurrency Control and Fault Tolerance

---

1. Concurrency Control

Definition:

Concurrency control is a set of techniques used in **parallel and distributed systems** to ensure that **multiple processes or threads can execute simultaneously without conflicting** when accessing shared resources (like memory, files, or databases).

- **Goal:** Maintain **data consistency, integrity, and correctness** while allowing parallel execution.

Mechanisms / Techniques:

1. **Locks / Mutexes:** Prevent multiple processes from accessing a resource simultaneously.
2. **Semaphores:** Signal-based mechanism to control access to resources.
3. **Monitors:** High-level synchronization construct to manage shared resources.
4. **Transactional Control (in databases):** Ensure **ACID properties** (Atomicity, Consistency, Isolation, Durability) in distributed transactions.

Example:

- **Parallel computing:** Two threads updating the same counter.
  - Without concurrency control: Counter may be incremented incorrectly.
  - With a **mutex lock**: Only one thread updates the counter at a time → correct result.
- **Distributed system:** Multiple users updating a bank account simultaneously.
  - Use **locking or transactions** to prevent inconsistent balances.

Behavior in Parallel & Distributed Systems:

Feature	Parallel Systems	Distributed Systems
Resource Type	Shared memory	Local memory + networked resources
Synchronization	Locks, semaphores, barriers	Distributed locks, transactions, consensus
Conflict Handling	Local	Network-aware, may need quorum or version control
Performance Impact	Can block threads, reduce speedup	Can increase latency due to network coordination

---

2. Fault Tolerance

Definition:

Fault tolerance is the ability of a system to **continue operating correctly even when one or more components fail**. This is crucial in **parallel and distributed computing**, where multiple processors or nodes increase the chance of failure.

- **Goal:** Ensure **reliability, availability, and correctness** despite hardware/software failures.

Techniques / Mechanisms:

1. **Redundancy:** Duplicate hardware/software components to take over in case of failure.
2. **Checkpointing / Rollback:** Periodically save system state so tasks can restart from the last checkpoint.
3. **Replication:** Maintain multiple copies of data across nodes for reliability.
4. **Consensus Algorithms:** Use algorithms like **Paxos or Raft** in distributed systems to ensure agreement despite failures.

Example:

- **Parallel computing:** If one processor fails during matrix computation, **checkpointing** allows the computation to restart from the last saved state instead of beginning from scratch.
- **Distributed system:** In a cloud storage system, if a node storing user data crashes, **replica nodes** provide uninterrupted access.

Behavior in Parallel & Distributed Systems:

Feature	Parallel Systems	Distributed Systems
Failure Type	Processor/core failure	Node, network, or software failure
Recovery Method	Checkpointing, rollback	Replication, consensus, failover
Impact on Performance	Moderate (local recovery)	Higher (network coordination and data recovery)
Fault Tolerance Level	Low to Medium	Medium to High
Use Cases	HPC clusters, simulations	Cloud computing, distributed databases, web services

---

✅ **Key Points:**

- **Concurrency control** ensures **correctness and consistency** when multiple tasks access shared resources.
- **Fault tolerance** ensures **system reliability and availability** despite failures.
- Both are **essential in parallel and distributed computing**, but methods differ depending on **shared memory vs distributed nodes**.