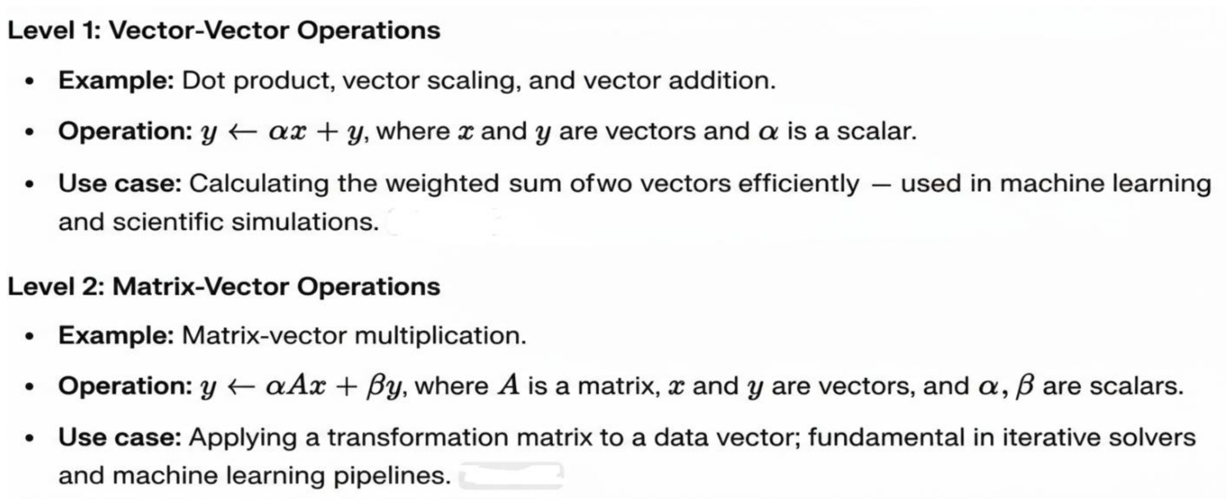
Unit 4
Cluster Computing

BLAS

BLAS (Basic Linear Algebra Subprograms) are a set of foundational routines for performing basic vector and matrix operations, widely used in high-performance computing (HPC).

When we talk about **BLAS in the context of cluster computing**, it involves using BLAS-like routines across multiple nodes of a distributed system to perform high-performance linear algebra computations.

BLAS routines are typically highly optimized for various hardware architectures, taking advantage of processor features such as parallelism, pipelining, and efficient memory access. As a result, using BLAS can significantly speed up computations compared to naïve implementations, especially for large-scale problems.

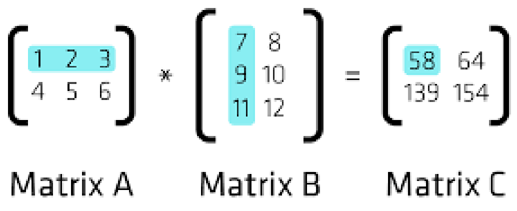


Level 1: Vector-Vector Operations

- **Example:** Dot product, vector scaling, and vector addition.
- **Operation:** $y \leftarrow \alpha x + y$, where x and y are vectors and α is a scalar.
- **Use case:** Calculating the weighted sum ofwo vectors efficiently — used in machine learning and scientific simulations.

Level 2: Matrix-Vector Operations

- **Example:** Matrix-vector multiplication.
- **Operation:** $y \leftarrow \alpha Ax + \beta y$, where A is a matrix, x and y are vectors, and α, β are scalars.
- **Use case:** Applying a transformation matrix to a data vector; fundamental in iterative solvers and machine learning pipelines.


$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} * \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \\ 139 & 154 \end{bmatrix}$
Matrix A
Matrix B
Matrix C

Level 3: Matrix-Matrix Operations

- Example: Matrix-matrix multiplication.
- Operation: $C \leftarrow \alpha AB + \beta C$ with A,B,C being matrices and α,β scalars.
- Use Case: Used for neural network computations, scientific simulations, and physics modeling with massive speed improvements compared to simple loop-based multiplications.

LAPACK

LAPACK (Linear Algebra PACKage) routines are highly efficient computational subroutines designed to solve large-scale systems of linear equations and related problems, leveraging matrix factorizations for scalability and speed.

LAPACK Solves Large Systems Efficiently:

1. *Matrix Factorization as Core Technique:* LAPACK routines solve Ax=b, where A is a large matrix and b a vector (or multiple right-hand sides), by first factorizing A. Common factorizations used include:
    - **LU Factorization:** Decomposes a matrix in lower and upper triangular matrices (L and U) so the system can be solved via forward and backward substitution.
    - **Cholesky Factorization:** For symmetric, positive definite matrices, decomposes A into reducing computational complexity.
    - **QR Factorization:** Useful for least squares problems and rank-deficient systems.

Driver Routines: LAPACK provides driver routines that encapsulate the entire solve process, including factorization and solution steps, optimized for different matrix properties:

Simple Drivers: Automatically factorize and solve for general matrices.

Expert Drivers: Allow detailed control and error handling, improving numerical stability and precision in challenging cases

Use of BLAS: LAPACK is built on BLAS for matrix operations, which ensures high efficiency by optimizing low-level vector and matrix computations at hardware level.

LAPACK is Suitable for Large Systems because of the following factor:

Numerical Stability: Factorization methods combined with iterative refinement improve accuracy.

Memory Efficiency: LAPACK exploits matrix structure and blocking techniques.

Hardware Optimization: Leveraging BLAS ensures optimized performance on different CPU/GPU architectures.

Scalability: Designed to handle very large linear systems that appear in scientific computing, engineering simulations, and data analysis.

Importance of BLAS and LAPACK in Scientific Computing:

Scientific computing workflows heavily rely on matrix and vector operations for modeling physical phenomena, simulating biological processes, and analyzing large data sets. BLAS offers highly optimized building blocks for these operations (e.g., matrix multiplication, vector addition), while LAPACK extends BLAS capabilities to solve systems of linear equations and eigenvalue problems efficiently.

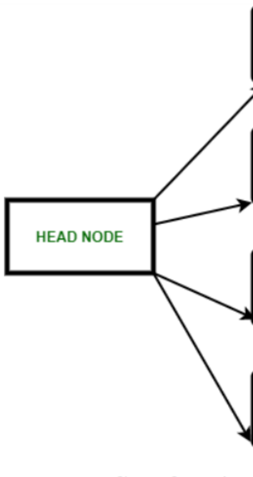
Example Applications

Weather Modeling: Weather models solve large partial differential equations that describe atmospheric dynamics, involving massive matrices representing state variables. BLAS accelerates basic operations like matrix multiplications within these models, while LAPACK routines efficiently solve the resulting large linear systems and eigenproblems, enabling more frequent and precise simulations to predict climate change and extreme weather.

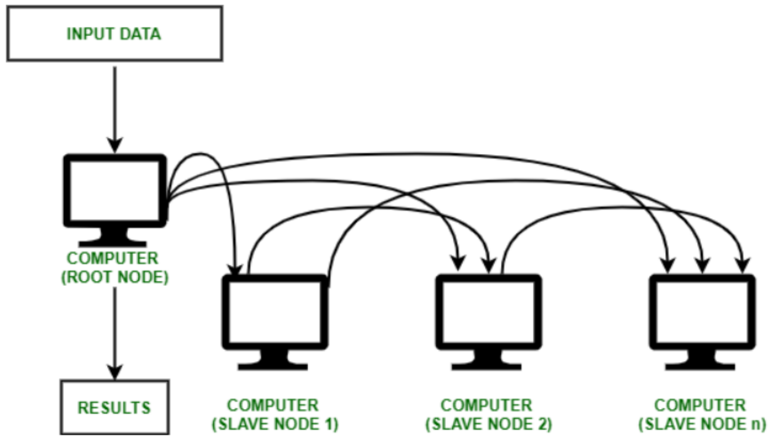
Genomics Research: Massive datasets from genome sequencing require linear algebra for data normalization, statistical modeling, and clustering. BLAS expedites core computations on genetic matrices, whereas LAPACK manages complex system solving related to gene interactions and drug response prediction, facilitating breakthroughs in personalized medicine.

Cluster Computing

Cluster computing is a collection of tightly or loosely connected computers that work together so that they act as a single entity. The connected computers execute operations all together thus creating the idea of a single system. The clusters are generally connected through fast local area networks (LANs)



Cluster Computing



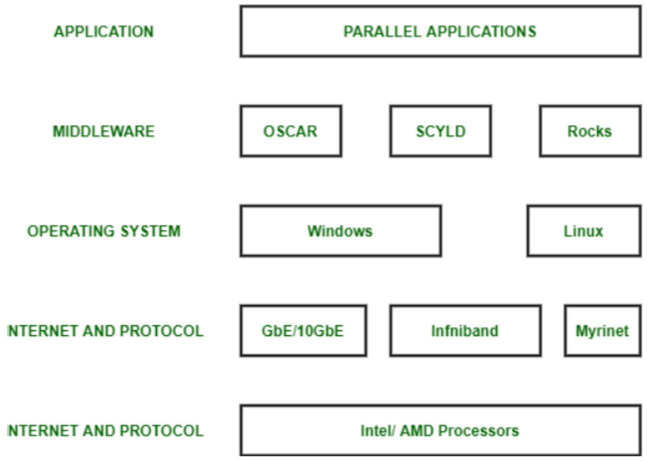
A Simple Cluster Computing Layout

Why is Cluster Computing important?

1. Cluster computing gives a relatively inexpensive, unconventional to the large server or mainframe computer solutions.
2. It resolves the demand for content criticality and process services in a faster way.
3. Many organizations and IT companies are implementing cluster computing to augment their scalability, availability, processing speed and resource management at economic prices.
4. It ensures that computational power is always available.
5. It provides a single general strategy for the implementation and application of parallel high-performance systems independent of certain hardware vendors and their product decisions.

Components of a Cluster Computer :

1. Cluster Nodes
2. Cluster Operating System
3. The switch or node interconnect
4. Network switching hardware



Cluster Components

Classification of Cluster :

1. Open Cluster :
IPs are needed by every node and those are accessed only through the internet or web. This type of cluster causes enhanced security concerns.

2. Close Cluster :
The nodes are hidden behind the gateway node, and they provide increased protection. They need fewer IP addresses and are good for computational tasks.

Key Factors in Clustering Architectures: The critical design factors in clustering architectures are redundancy, scalability, load balancing, monitoring and automated recovery, and robust networking and storage. These factors ensure high availability, optimal performance, and fault tolerance.

Scalability: Ability of the cluster to grow (add more nodes) without significant loss in performance. Essential for workloads that grow with data size (e.g., weather modeling, genomics)

Performance: How efficiently the cluster processes tasks (throughput, latency). It is determined by Node hardware, Network speed

Fault Tolerance and High Availability: Ability of the cluster to continue functioning when one or more nodes fail. Implemented via: Redundant nodes, health monitoring.

Load Balancing: Even distribution of workload across all nodes. Prevents idle nodes and bottlenecks. It can be Static or Predefined distribution and Dynamic which is adjusted during runtime based on load.

Network Architecture: How nodes are connected and communicate. Can be Star, Mesh, Ring, Torus, Hypercube.

Advantages of Cluster Computing :

1. High Performance:
The systems offer better and enhanced performance than that of mainframe computer networks.

2. Easy to manage:
Cluster Computing is manageable and easy to implement.

3. Scalable:
Resources can be added to the clusters accordingly.

4. Expandability:
Computer clusters can be expanded easily by adding additional computers to the network. Cluster computing is capable of combining several additional resources or the networks to the existing computer system.

5. Availability:
The other nodes will be active when one node gets failed and will function as a proxy for the failed node. This makes sure for enhanced availability.

6. Flexibility:
It can be upgraded to the superior specification or additional nodes can be added.

Disadvantages of Cluster Computing:

1. High cost :

It is not so much cost-effective due to its high hardware and its design.

2. Problem in finding fault:

It is difficult to find which component has a fault.

3. More space is needed:

Infrastructure may increase as more servers are needed to manage and monitor.

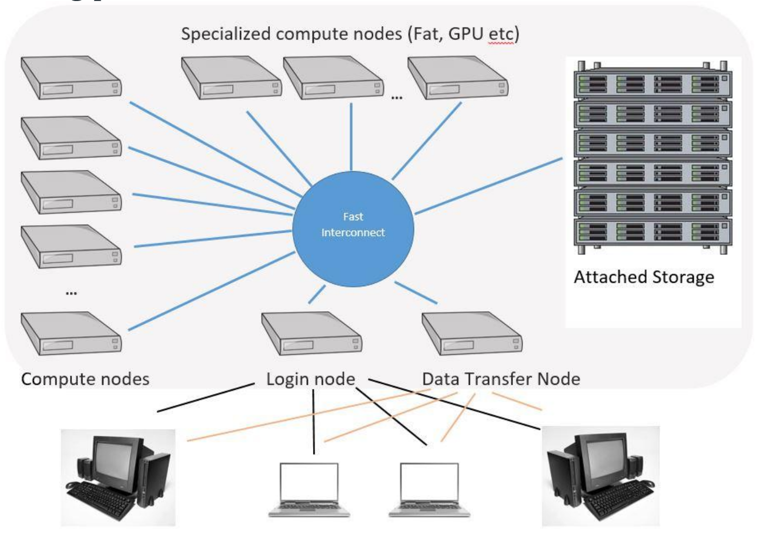
Applications of Cluster Computing:

- Various complex computational problems can be solved.
- It can be used in the applications of aerodynamics, astrophysics and in data mining.
- Weather forecasting.
- Image Rendering.
- Various e-commerce applications.
- Earthquake Simulation.
- Petroleum reservoir simulation.

Types of Cluster Computing

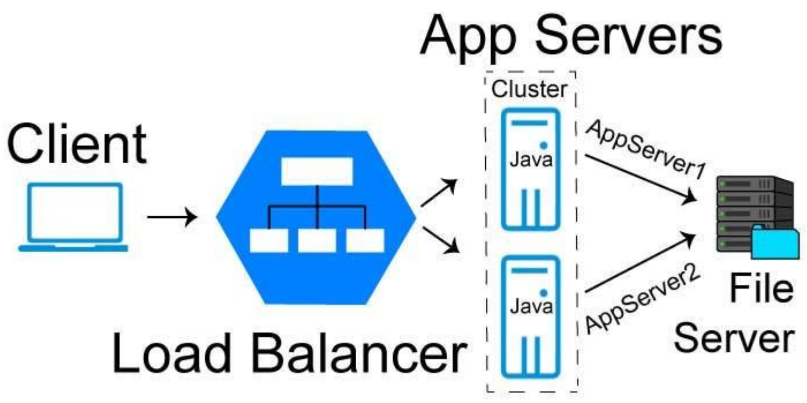
- **High performance (HP) clusters:**

HP clusters use computer clusters and supercomputers to solve advance computational problems. They are used to performing functions that need nodes to communicate as they perform their jobs. They are designed to take benefit of the parallel processing power of several nodes.



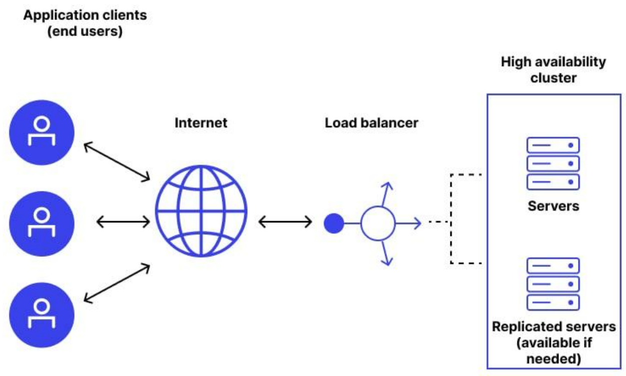
- **Load-balancing clusters:**

Incoming requests are distributed for resources among several nodes running similar programs or having similar content. This prevents any single node from receiving a disproportionate amount of task. This type of distribution is generally used in a web-hosting environment.



- **High Availability (HA) Clusters:**

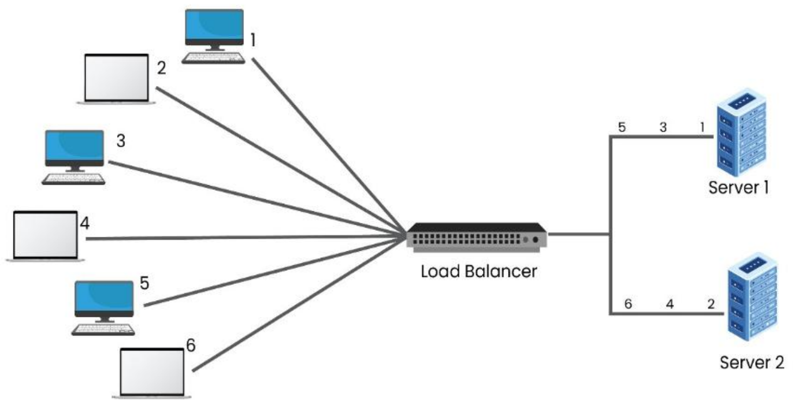
HA clusters are designed to maintain redundant nodes that can act as backup systems in case any failure occurs. Consistent computing services like business activities, complicated databases, customer services like e-websites and network file distribution are provided. They are designed to give uninterrupted data availability to the customers.


Application clients (end users)
Internet
Load balancer
High availability cluster
Servers
Replicated servers (available if needed)

Cluster Computing Load Balancing Algorithms

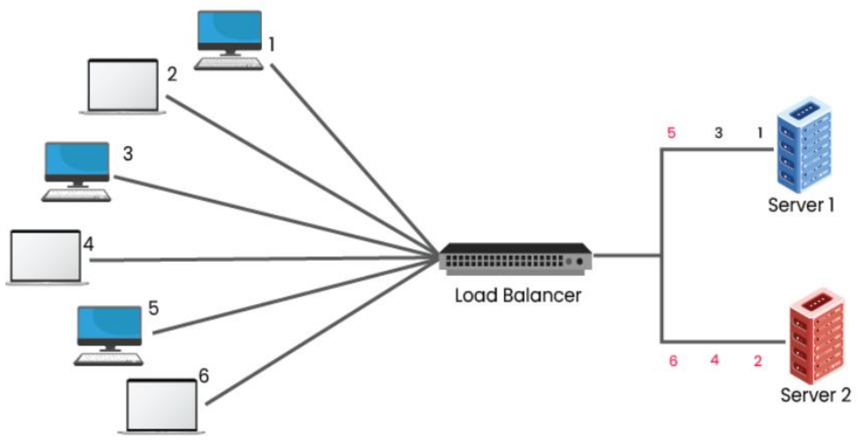
Round Robin Algorithm:

In the below diagram, there are two servers awaiting requests behind the load balancer. Upon the arrival of the first request, the load balancer will forward the request to Server 1 and after getting the second request, it is allotted to Server 2. As there are only two servers in the network, the next request will again be forwarded to Server 1—this way, the requests are sent to both the servers in a cyclic format.



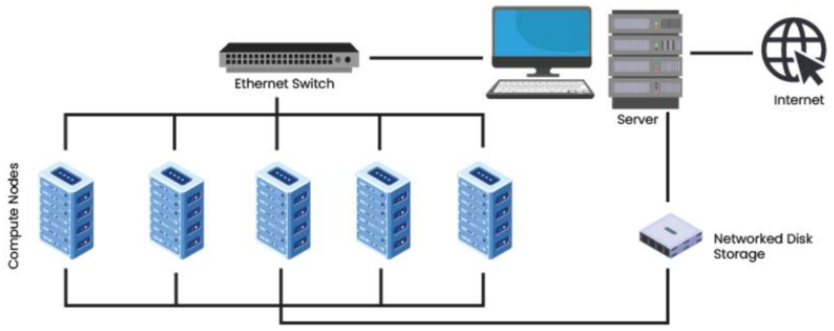
Least Connections:

The Least Connections Algorithm considers the number of connections that each server currently has. Whenever there is a connection request, the load balancer determines which of the servers has the least number of connections and then the new connection will be assigned to that server.



Beowulf Clusters:

One of the most remarkable technology advancements today has been the growth of Personal Computers' computing performance. The Beowulf Cluster was created to achieve the rising high-performance power in certain scientific areas for developing a powerful and affordable Cloud computing system. The constant evolution of performance, processor, and collaboration between PCs and workstations resulted in decreased network processors' reduced costs for influencing the improved high-performance clusters. A Beowulf cluster is a computer cluster of normally identical, commodity-grade computers networked into a small local area network with libraries and programs installed that allow processing to be shared among them. The result is a high-performance parallel computing cluster from inexpensive personal computer hardware.



Aspect	High-Availability (HA) Cluster	Load-Balancing Cluster
Primary Objective	To ensure continuous service and minimize downtime during node or system failures.	To distribute workload evenly among multiple nodes to improve performance and responsiveness.
Architecture Design	Usually designed with redundant nodes (active–passive or active–active configuration). If one node fails, another automatically takes over.	Designed with multiple active nodes working simultaneously. A load balancer distributes incoming requests across nodes.
Control Mechanism	Uses heartbeat signals and failover mechanisms for fault detection and recovery.	Uses load-balancing algorithms (e.g., Round Robin, Least Connections) to manage resource utilization.
Focus Area	Reliability and fault tolerance — ensures services remain available even after a failure.	Performance and scalability — ensures system can handle many users efficiently.

Typical Configuration	May include shared storage and replication to keep data synchronized across nodes.	Nodes may operate independently, each handling separate portions of requests or sessions.
Example Use Cases	Banking systems, healthcare monitoring, online transaction processing (OLTP).	Web servers, cloud-based services, content delivery networks (CDNs).
Failure Handling	If a node fails, another node automatically takes over (failover) to maintain service continuity.	If a node fails, the load balancer redirects traffic to other active nodes.
Performance Goal	Maintain uptime and availability.	Achieve speed, scalability, and efficient resource utilization.

Peer-to-Peer and Master-Slave Clustering Models

•

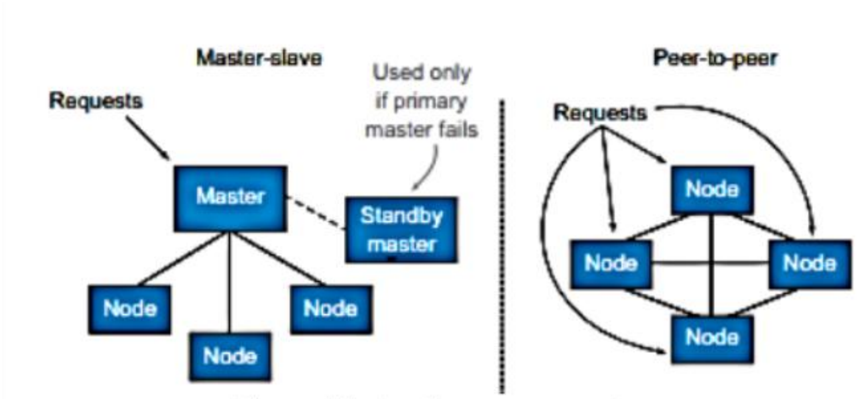


Figure: Master-slave versus peer-to-peer

Aspect	Master-Slave Clustering	Peer-to-Peer Clustering
Architecture	Centralized: One master node controls multiple slave nodes	Decentralized: All nodes are equal peers communicating directly
Control	Master handles all write operations and coordination	All peers can handle both reads and writes
Fault Tolerance	Master is a single point of failure	No single point of failure; higher fault tolerance
Scalability	Limited due to master bottleneck	Highly scalable as all peers share load
Consistency	Strong consistency since master serializes writes	Eventual consistency due to asynchronous peer updates
Complexity	Simpler to implement and manage	More complex due to conflict resolution and synchronization
Use cases	Suitable for read-heavy systems where writes are controlled	Suitable for distributed and highly available systems

Examples

1. University Cluster:

Usually follows a Master-Slave Model, where a master node schedules and distributes academic or research computing tasks to slave nodes. This model supports controlled job execution and workload balance. For example, high-performance computing clusters in universities use this architecture to manage parallel processing tasks efficiently.

2. E-Commerce Cluster:

Typically uses a Peer-to-Peer or Multi-Master Model to ensure high availability, scalability, and fault tolerance necessary for handling concurrent transactions, shopping sessions, and inventory updates across multiple distributed nodes. For example, large e-commerce platforms like Amazon leverage distributed (peer-to-peer) architectures to maintain continuous service despite node failures.

Role of network bandwidth and latency in clustering performance:

Network bandwidth and latency are fundamental to clustering performance in distributed systems. Higher bandwidth allows for faster data exchange between cluster nodes, enabling quicker synchronization, aggregation, and overall processing tasks; limited bandwidth can bottleneck large data transfers, slow down convergence, and reduce throughput in clustering workloads. Network latency, expressed as the delay in data transmission, can increase response times for database operations, model training, and cluster state updates. High latency leads to longer job completion times and can significantly restrict the efficiency of distributed clustering, especially in time-sensitive tasks or geographically dispersed clusters.

Impact of Bandwidth and Latency

Sufficient bandwidth minimizes communication delay, improves scalability, and supports more simultaneous connections within clusters.

High latency between nodes forces longer coordination times, raises data transfer delays during synchronization, and degrades system responsiveness, impacting large-scale or real-time clustering.

Extreme tail latency can compound delays, cause retransmissions, and disrupt AI model training and inference phases.

Both bandwidth and latency must be optimized for cluster routing, load balancing, and distributed job computation for best results.

Node Heterogeneity and Task Scheduling

Node heterogeneity refers to clusters consisting of nodes with varying computing capabilities, memory, storage, and network speeds. Task scheduling mechanisms must account for these differences to maximize cluster performance.

Schedulers that ignore node heterogeneity risk placing heavy workloads on underpowered nodes, resulting in stragglers and increased job completion times.

Modern scheduling algorithms assess node capabilities, dynamically assign tasks based on resource strengths, and avoid overloading slow or less capable nodes.

Optimization techniques such as speculative execution or priority-based mapping move tasks away from straggling nodes, leveraging faster nodes to maintain workflow efficiency.

Accurate profiling of nodes (CPU, GPU, memory, network speed) is essential to prevent load imbalance and ensure fault tolerance in heterogeneous clusters.

Consider a Hadoop or CPU-GPU cluster:

If the scheduler uniformly distributes jobs, slower nodes may cause whole-job delays.

Advanced schedulers profile node performance, sending compute-intensive tasks to high-end servers and lighter tasks to less powerful nodes.

By matching job requirements with node capacities, the cluster can reduce overall execution times by up to 40% compared to generic task assignment.

Mission-Critical and Business-Critical Applications

Mission-critical applications are essential to immediate business operations; failure causes the business or process to stop completely.

Examples: Payment processing for e-commerce, online banking, hospital patient monitoring, air traffic control, emergency dispatch.

Industry	Mission-Critical Examples	Downtime Impact
Banking	Transaction processing, fraud detection, account access	Immediate loss of service, financial/legal risk
Healthcare	Patient monitoring, EMR/EHR, care coordination	Danger to health/life, compliance breach

Business-critical applications are important for everyday operations but do not cause total operational halt if disrupted; failure may cause financial or reputational losses, but the core business can typically continue for some time.

Examples: HR management software, scheduling systems, CRM platforms, financial reporting, messaging services.

System Type	Downtime Tolerance	Examples
Mission-Critical	Near-zero; seconds/min	Payment gateways, hospital monitoring
Business-Critical	Minutes/hours	Payroll, CRM, sales analytics

Fault Detection and Fault Masking

Fault Detection:

Identifying that a component (node, process, link) has failed or is behaving incorrectly.

These are essential for high availability, reliability, and fault tolerance in HPC clusters, distributed databases, real-time systems, and mission/business-critical applications.

Examples in HPC:

Node failure: Detecting a node failure (using log monitoring or machine learning) and masking it by rescheduling jobs to other available nodes.

Memory errors: Detecting memory errors (using error detection codes) and masking them by re-reading the data from another location.

Fault Detection

In cluster/distributed systems, **early detection** of faults is critical for:

- Ensuring high availability

- Triggering failovers
- Maintaining consistency
- Preventing system-wide crashes

Fault Detection Algorithms:

Heartbeat Mechanism

Timeout-Based Detection

Failure Suspectors (Crash Detection with Assumptions)

Consensus-Based Detection

Watchdog Timers

Check pointing

Checkpointing is the process of saving the state of a running application or system at regular intervals so that it can resume from the last saved state after a failure, instead of starting from scratch.

It helps to:

- Minimize computation loss
- Allow recovery after failures
- Essential in long-running or mission-critical computations

Types of Checkpointing:

- Full Checkpointing: Saves complete memory, files, states
- Incremental: Saves only changes since the last checkpoint
- Coordinated: All processes in a distributed system checkpoint simultaneously to maintain consistency
- Uncoordinated: Processes checkpoint independently (can lead to "domino effect" – rollback of many processes)
- Application-level: Handled by app logic (more optimized)
- System-level: Transparent to application (e.g., CRIU in Linux)

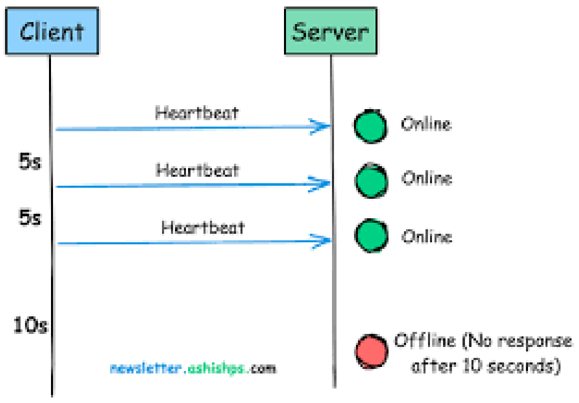
Heartbeat Mechanism

A heartbeat is a periodic signal/message sent from one system component to another to indicate that it is alive and functioning.

It helps to:

- Monitor liveness of nodes, services, or applications
- Trigger failure detection if heartbeat is not received within timeout
- Used in clusters for failover, leader election, and service discovery

```
Every 5 seconds → Node A sends a heartbeat to Node B.
If Node B misses 3 consecutive heartbeats → Node A is marked as failed.
```



Absence of Heartbeat Leading to Failure

Heartbeats are small, periodic signals sent by processes or nodes to indicate that they are alive and functioning.

If a monitoring process does not receive a heartbeat within a pre-defined timeout interval, it suspects failure and can mark the node as suspect or failed.

This absence may be caused by hardware faults, software bugs, network congestion, or misconfigurations.

Failure to detect absence promptly can result in prolonged outages, degraded performance, or split-brain scenarios where inconsistent states exist in the cluster.

Heartbeat Signals in Banking Clusters: Used for continuous monitoring of server health, application nodes, and services such as transaction processing, funds transfer, or fraud detection.

Each node sends a periodic heartbeat signal to a central monitoring service or peers to confirm it is alive and functioning.

If heartbeat signals are missing or delayed beyond a threshold, the system marks the node as failed or unreachable.

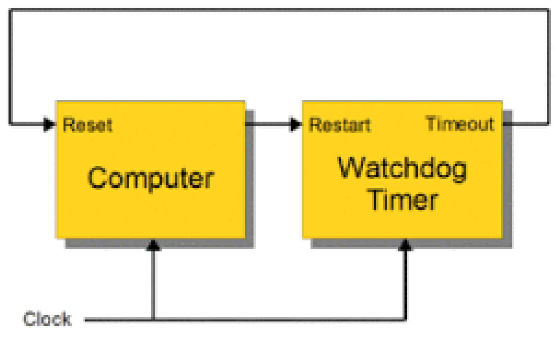
This triggers failover mechanisms: redirecting transactions to backup nodes, initiating recovery processes, or alerting operators to minimize downtime and prevent transaction loss/fraud.

Watchdog Timers

A watchdog timer is a hardware or software timer that resets or takes corrective action if the monitored process fails to respond or behave properly within a certain time window.

It helps to:

- Detect hangs or deadlocks
- Auto-restart or reset unresponsive components
- Ensure system reliability in embedded and real-time systems



Significance of Watchdog Timers

Watchdog timers are essential hardware or software components for fault detection and recovery in systems ranging from embedded controllers to distributed applications.

A watchdog timer expects regular signals (“kicks” or “refreshes”) from the system; absence of this signal within the set interval triggers corrective action, such as resetting the system, issuing an alert, or switching to backup logic.

Watchdogs help ensure critical tasks are performed within deadlines and rapidly recover from errors, minimizing downtime and improving fault tolerance.

Time-windowed watchdogs allow for granular fault detection, spotting both missed and early refreshes, enabling the system to log errors, self-heal minor issues, and trigger resets for persistent faults.

In safety-critical and mission-critical applications, watchdog timers enhance reliability and help maintain safe operational states even during faults, reducing the risk posed by hardware failures, software bugs, or unexpected system hangs.

Watchdog Timers in Banking Applications

Embedded in critical banking components such as ATM controllers, core banking software modules, and network gateways.

Expect periodic “kick” or reset signals from application processes or hardware modules to confirm normal operation.

On missed signals, the watchdog timer triggers corrective action, such as system reset, process restart, or fail-safe mode activation.

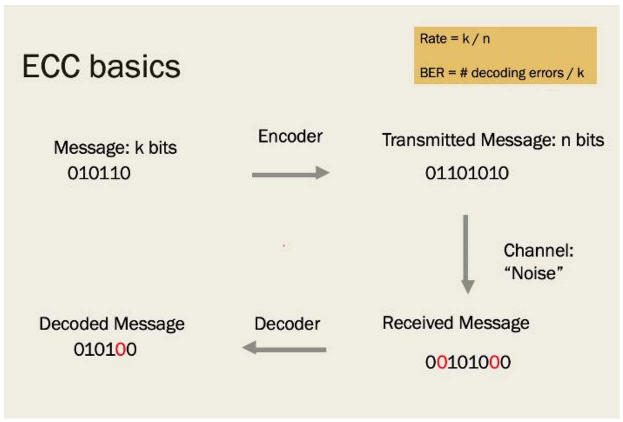
This mechanism ensures recovery from software hangs, hardware anomalies, or unexpected crashes that could disrupt financial transactions or customer services.

For instance, in ATM machines, watchdog timers prevent frozen or stalled transaction processes that would otherwise lead to customer service issues or cash dispensing errors.

Error-Correcting Codes (ECC)

- ECC, or Error Correction Code, is a technique used to detect and correct errors in data, particularly in memory and data transmission.

- It adds extra bits to the original data, creating redundancy that allows the system to identify and fix errors that might occur due to noise, interference, or other factors.
- This ensures data integrity and reliability, especially in critical systems like servers and storage devices.



Fault Masking:

Fault masking algorithms are designed to hide the presence of faults from the system's users or other components by ensuring continuous correct operation despite failures. They achieve this through redundancy, error detection, and correction mechanisms, so faults do not propagate or cause visible errors.

fault masking algorithms:

Triple Modular Redundancy (TMR)

Utilizes three identical modules performing the same task simultaneously with a voter mechanism choosing the correct output by majority voting. This masks any single module failure effectively.

N-Modular Redundancy

An extension of TMR where N = 2m + 1, masking up to m concurrent failures via voting on replicates output.

Error Detection and Correction Codes

Techniques like parity bits, Hamming codes, and cyclic redundancy checks (CRC) add redundancy at the data level to detect and correct bit errors, masking faults in data transmission or storage.

Consensus Algorithms (e.g., Paxos, Raft)

Achieve agreement among distributed nodes despite faults, masking failures related to node crashes or message loss by continuing operation as long as a majority agree.

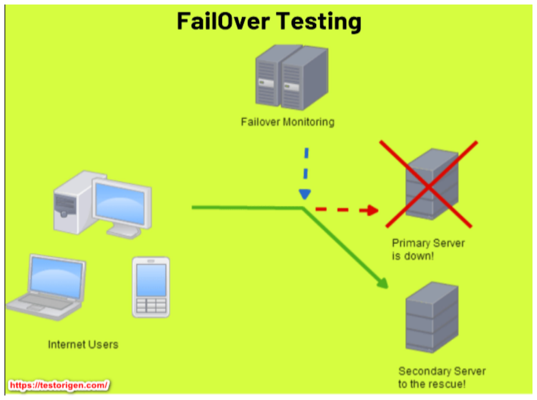
Bulkhead Pattern

Isolation technique to mask failure by segregating system components so failure in one part doesn't affect others, effectively masking faults across system modules.

Failover

Failover is the mechanism of transferring workload and data processing from a primary system to a secondary system due to a failure or planned outage.

It provides immediate redundancy and ensures that critical services remain available despite system issues.



Examples:

Switching from a primary database server to a replicated database server.

Initiating a virtual machine (VM) failover to a backup VM in a different location.

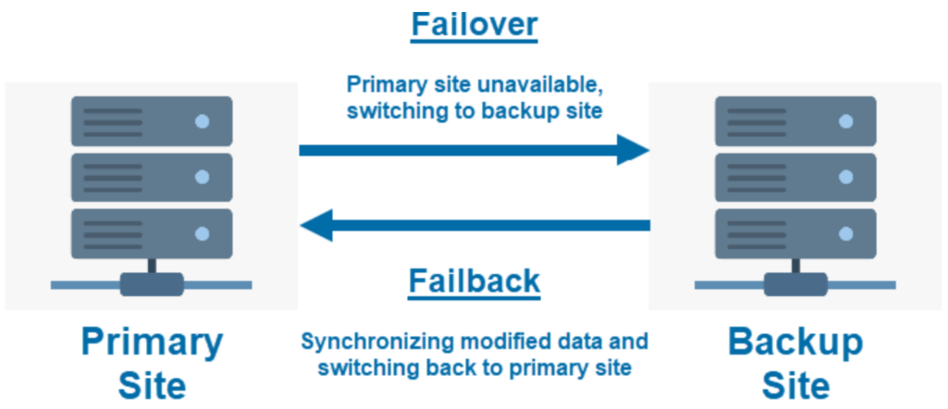
Phases: Failover typically involves:

1. Detecting the failure
2. Initiating the switch to the backup system
3. Redirecting traffic.

Failback

Failback is the process of returning operations from the secondary system back to the original, now restored, primary system.

Once the primary system is deemed operational, failback ensures that the system returns to its normal, preferred state.


Failover
Primary site unavailable, switching to backup site
Failback
Synchronizing modified data and switching back to primary site
Primary Site
Backup Site

Examples:

Replicating any changes made on the backup VM back to the original VM.

Redirecting website traffic back to the original, now repaired, server.

Phases:

1. Failback involves ensuring the primary system is functioning correctly

2. Synchronizing data between the systems
3. Redirecting traffic back to the primary.

Importance in Disaster Recovery

Business Continuity: Both failover and failback are critical components of a robust disaster recovery plan, ensuring that businesses can continue operating even when faced with unexpected outages.

Reduced Downtime: By enabling rapid switching to backup systems, failover minimizes downtime and its associated costs.

Data Protection: Failover and failback processes often involve data replication, ensuring data consistency and availability across systems.

In essence, failover and failback work in tandem to provide fault tolerance and business continuity, allowing organizations to recover from system failures and maintain operational efficiency.