

### 1. Scenario Explanation

A research university runs a climate simulation generating **10 TB/day** on **500 nodes**.

Key challenges:

- Huge data volume → storage bottleneck
- Simultaneous data generation → I/O contention
- Need for parallel access → analytics efficiency

A **Parallel File System (PFS)** is ideal because it supports **high-speed concurrent data access**.

---

### 2. Parallel I/O (Input/Output)

Parallel I/O is the **simultaneous reading/writing of data** by multiple nodes or processors.

**Key Idea:**

Instead of one node handling I/O, **many nodes perform I/O together**.

**Benefits:**

- Reduces I/O bottleneck
  - Improves throughput
  - Supports large-scale applications
- 

### 3. Need for Parallel I/O

Traditional I/O is **sequential**, causing:

- Slow performance
- Poor scalability

Parallel I/O solves this by:

- Allowing **multiple nodes to access storage simultaneously**
  - Increasing system efficiency in:
    - Scientific simulations
    - Big data analytics
-

## 4. Parallel File Systems (PFS)

A Parallel File System distributes data across multiple storage devices and allows **concurrent access**.

**Examples:**

- Lustre
  - IBM Spectrum Scale (GPFS)
  - BeeGFS
- 

## 5. Components of Parallel File System

### 1. Client Nodes

- Run applications
- Send read/write requests

### 2. Metadata Server

- Stores file information (location, permissions)
- Handles operations like open, close

### 3. Storage Servers

- Store actual data
- Data is **striped across multiple servers**

### 4. Network

- High-speed interconnect (e.g., InfiniBand)
  - Enables fast communication
- 

## 6. Working Example

A **1 GB file** is split across 4 servers:

- Process 1 → Server 1
- Process 2 → Server 2
- Process 3 → Server 3
- Process 4 → Server 4

 All read simultaneously → **4× faster performance**

---

## 7. Advantages of Parallel File Systems

- High performance
  - Scalability
  - Concurrent access
  - Suitable for HPC applications
- 

## 8. Striping

Striping divides data into **blocks (stripes)** and distributes them across disks.

### Working:

- File → split into chunks
- Each chunk stored on different disk

### Benefits:

- Parallel access
  - Increased throughput
  - Reduced latency
- 

## 9. Data Partitioning

Data is divided so each node works on its own part.

### Types:

- Range partitioning
- Hash partitioning
- Round-robin

➡ Reduces contention and improves efficiency

---

## 10. Parallel Access Methods

Libraries simplify parallel I/O:

- MPI-IO
- HDF5
- NetCDF

- ➔ Provide easy interface for developers
- 

## 11. Data Replication and Caching

- **Replication:** Copies data on multiple nodes → reliability
  - **Caching:** Stores frequently used data in memory → faster access
- 

## 12. Striping Details

### a) Data Division

- Data split into fixed-size blocks

### b) Distribution

- Stored across disks (round-robin or hashing)

### c) Parallel Access

- Multiple disks read/write simultaneously

### d) Load Balancing

- Even distribution prevents bottlenecks

### e) Fault Tolerance

- RAID techniques add reliability

### f) Stripe Size

- Large → better for sequential I/O
  - Small → better for random I/O
- 

## 13. Conclusion

Parallel I/O and Parallel File Systems are essential for handling **large-scale data-intensive applications** like climate simulations.

They:

- Eliminate I/O bottlenecks
  - Improve scalability
  - Enable efficient parallel processing
-

---

## Parallel Computing Concepts

### 1 Parallel Algorithm Models

Parallel algorithm models provide a framework to design and analyze parallel programs.

#### Purpose:

- Efficient data partitioning
- Proper task mapping
- Reduced communication overhead

### 1. Parallel Algorithm Models

Parallel algorithm models define how tasks are divided and executed simultaneously.

#### Types of Parallel Models:

##### 1. Data-Parallel Model

In this model, the same operation is applied to different parts of data simultaneously.

*Example:* Adding numbers in an array at the same time.

##### 2. Task Graph Model

Tasks are represented as nodes in a graph, and dependencies are shown by edges.

*Example:* One task starts only after another finishes.

##### 3. Work Pool Model

All tasks are placed in a pool, and processors pick tasks when they are free.

*Example:* Workers taking jobs from a queue.

##### 4. Master-Slave Model

One processor (master) assigns tasks to other processors (slaves).

*Example:* Teacher assigning work to students.

##### 5. Pipeline Model

Tasks are divided into stages, and each stage works in parallel.

*Example:* Assembly line in a factory.

##### 6. Hybrid Model

Combination of two or more models to improve performance.

##### 7. Bulk Synchronous Parallel (BSP)

Execution occurs in three phases: computation, communication, and synchronization.

---

## 2. Parallel Architecture

Parallel architecture is the design of a computer system that uses multiple processors to execute tasks simultaneously for faster performance.

---

### 3. Parallel Processors / Cores

Parallel processors or cores are multiple processing units that perform computations at the same time.

*Example:* A quad-core processor can execute four tasks simultaneously.

---

### 4. Memory Architecture

Memory architecture defines how memory is organized in a parallel system.

**Types:**

**(a) Shared Memory:**

All processors access a common memory space. It is easy to use but may cause conflicts.

**(b) Distributed Memory:**

Each processor has its own memory and communicates through messages.

---

### 5. Interconnection Network

An interconnection network connects processors and allows communication between them.

*Examples:* Bus, switches, network links.

---

### 6. Parallel Programming Models

These models define how parallel programs are written and executed.

*Examples:*

- Thread-based model (shared memory)
  - Message passing model (distributed memory)
  - GPU programming (CUDA/OpenCL)
- 

### 7. Task Scheduling and Load Balancing

**Task Scheduling:** Assigns tasks to processors.

**Load Balancing:** Ensures equal distribution of work among processors to avoid idle time.

---

### 8. Parallel Algorithms

Parallel algorithms are designed to solve problems using multiple processors simultaneously to reduce execution time.

*Examples:* Parallel sorting, matrix multiplication.

---

## 9. Types of Parallelism

1. **Data parallelism:** Same operation on multiple data units
  2. **Task parallelism:** Different tasks run concurrently
  3. **Pipeline parallelism:** Sequential stages with overlapping execution
- 

## 10. Parallel Computing Platforms

These are systems that support parallel execution.

*Examples:*

- Multi-core processors
  - GPUs
  - Clusters
  - Cloud computing systems
- 

## 11. Conclusion

Parallel computing improves performance by dividing tasks into smaller parts and executing them simultaneously using multiple processors. It is widely used in modern computing systems for faster and efficient processing.

---