Q1. (C2 – 5 marks)
Explain the working of the **MapReduce model**. Identify its stages and advantages over traditional processing in distributed systems.

Working of MapReduce :

MapReduce works in two main steps:

1. **Map Step**
   - Breaks big data into smaller pieces
   - Each piece is processed separately (in parallel)
   - Outputs data as key-value pairs
2. **Reduce Step**
   - Gathers all key-value pairs with the same key
   - Combines or summarizes them into final results

Stages in MapReduce

1. **Splitting** – Input data is divided into chunks.
2. **Mapping** – Map function applied to each chunk.
3. **Shuffling** – Intermediate data grouped by key.
4. **Sorting** – Grouped keys are sorted for reduction.
5. **Reducing** – Final output is generated by Reduce function.
6. **Output** – Results written to storage (e.g., HDFS).

Advantages over Traditional Systems:

- Easy to write and understand
- Automatically handles parallelism and failures
- Scales to thousands of machines
- Efficient data processing using **data locality**

Q2. (C2 – 5 marks) Differentiate between **Distance Vector Routing (DVR)** and **Link State Routing (LSR)**. Explain how **Bellman-Ford** and **Dijkstra's algorithms** apply to these with basic **pseudocode**.

🔁 Distance Vector Routing (DVR)

- Routers share info **with neighbors only**.
- Uses **Bellman-Ford algorithm**.
- Routers know distance to other routers, **not full path**.
- Slower updates, can face issues like **count-to-infinity**.

🌐 Link State Routing (LSR)

- Routers share info **with all routers**.
- Uses **Dijkstra's algorithm**.
- Each router builds **full map** of the network.
- Faster and more accurate routing.

📘 Bellman-Ford (used in DVR) – Finds shortest path by relaxing edges:

text Copy Edit

```
Repeat (V - 1) times:
  For each edge (u, v):
    if distance[u] + weight < distance[v]:
      update distance[v]
```

📗 Dijkstra's (used in LSR) – Finds shortest path using nearest unvisited node:

text Copy Edit

```
Start from source:
  Pick node with smallest distance
  Update distances of its neighbors
  Repeat until all nodes are visited
```

Q3. (C2 – 5 marks) Compare shared memory and **distributed memory** architectures. Provide advantages, disadvantages, and suitable use-cases.

1. Shared Memory Architecture

Aspect	Description
Memory Access	All processors share the same physical memory
Communication	Through **shared variables/memory**
Speed	Fast communication

Aspect	Description
Complexity	Easier programming, harder synchronization

✅ *Advantages:*

- Simple to program and debug
- Faster data access and sharing

❌ *Disadvantages:*

- Hard to scale (limited number of processors)
- Risk of memory conflicts (need locks)

📌 *Use-cases:*

- Multicore processors
- Single-node parallel processing

🌐 2. Distributed Memory Architecture

Aspect	Description
Memory Access	Each processor has its **own private memory**
Communication	Via **message passing (e.g., MPI)**
Speed	Slower communication due to network delay
Complexity	More complex programming

✅ *Advantages:*

- Highly scalable
- No memory conflicts

❌ *Disadvantages:*

- Complex code (need explicit communication)

- Slower communication

📌 *Use-cases:*

- Supercomputers, cloud clusters
- Large-scale data processing (e.g., Hadoop, MPI)

Q4. (C3 – 5 marks) Explain how **barrier synchronization** works in MPI. Provide an example where synchronization is necessary.

◆ What is Barrier Synchronization?

In MPI, **barrier synchronization** means **all processes must stop and wait** at a certain point (the "barrier") until **every process reaches it**. After that, they all move forward **together**.

MPI provides this using:

```
MPI_Barrier(MPI_COMM_WORLD);
```

◆ Why It's Needed? (Example)

Imagine every process is loading data.
If one process starts using the data **before others finish loading**, it can cause errors.

So we use `MPI_Barrier()` to make sure:

- **All processes finish loading data**
- **Then start computing together**

◆ Simple Example:

```
// All processes load data
// Then wait at the barrier
MPI_Barrier(MPI_COMM_WORLD);
// All start computation together
```

Q5. (C2 – 5 marks) Name the **taxonomy** that categorizes computers into four types. Discuss the types briefly.

◆ Name of the Taxonomy:

Flynn's Taxonomy

◆ Four Types of Computers in Flynn's Taxonomy:

1. **SISD (Single Instruction, Single Data):**
   - One processor executes **one instruction** on **one data** at a time.
   - Example: Traditional single-core computer.
2. **SIMD (Single Instruction, Multiple Data):**
   - One instruction is applied to **multiple data elements** at the same time.
   - Example: Graphics Processing Units (GPUs).
3. **MISD (Multiple Instruction, Single Data):**
   - Many instructions operate on the **same data**.
   - Rare in practice; mostly used for **fault-tolerant systems**.
4. **MIMD (Multiple Instruction, Multiple Data):**
   - Many processors execute **different instructions** on **different data**.
   - Example: Multi-core processors, parallel systems.

Q6. (C2 – 5 marks) Derive the formula for **scalability** that affects speedup in parallel processing and explain with a numeric example.

✓ Step-by-Step Derivation:

Let:

- **T_serial** = time taken by the program with 1 processor
- **P** = fraction of the program that can be parallelized
- **(1 − P)** = serial portion (cannot be parallelized)
- **N** = number of processors

1. **Total Execution Time with Parallelism:**

Saved memory full ⓘ

$$T_{\text{parallel}} = T_{\text{serial}} \times \left[(1-P) + \frac{P}{N}\right]$$

---

2. **Speedup Formula:**

Speedup S is how much faster the program runs:

$$S = \frac{T_{\text{serial}}}{T_{\text{parallel}}}$$

Now substitute the parallel time from step 1:

$$S = \frac{T_{\text{serial}}}{T_{\text{serial}} \times \left[(1-P) + \frac{P}{N}\right]}$$

Cancel out T_{serial}:

$$S = \frac{1}{(1-P) + \frac{P}{N}}$$

📌 **Final Formula:**

$$\boxed{S = \frac{1}{(1-P) + \frac{P}{N}}}$$

This is Amdahl's Law — a key formula for analyzi scalability in parallel processing.

🔢 **Example:**

Let:

- $P = 0.9$ (90% of the code is parallel)
- $N = 10$ processors

$$S = \frac{1}{(1-0.9) + \frac{0.9}{10}} = \frac{1}{0.1 + 0.09} = \frac{1}{0.19} \approx 5.26$$

Result:

With 10 processors and 90% parallel code, you get around **5.26× speedup**.
Even with more processors, the 10% serial part limits the maximum speedup.