

Directed Works TD 1: Hadoop Systems & MapReduce Paradigm (Solution)

Solution to Exercise 1

1. Transfer Time Calculation:

- **Total Size in Bits:** $50 \text{ TB} = 50 \times 10^{12} \text{ Bytes} = 400 \times 10^{12} \text{ bits} = 400,000 \text{ Gb (Gigabits)}$
- **Transfer Speed:** $1 \text{ Gbps} = 10^9 \text{ bits/second}$
- **Transfer Time in Seconds:** $\text{Time} = \frac{400,000 \text{ Gb}}{1 \text{ Gbps}} = 400,000 \text{ seconds}$
- **Conversion to Hours and Days:**

$$\text{Hours} = \frac{400,000}{3,600} \approx \mathbf{111.11 \text{ h}}$$

$$\text{Days} = \frac{111.11}{24} \approx \mathbf{4.63 \text{ days}}$$

- *Takeaway:* Transferring raw data across the network takes over **4.6 days** before any analysis can even begin.

2. Single-Machine Hardware Bottlenecks:

- **Disk I/O:** Sequential read speeds on a single hard drive (~100–200 MB/s) require dozens of hours to read 50 TB.
- **RAM & CPU Limits:** A single machine cannot load or process petabyte-scale indices into memory, causing severe thrashing.
- **Single Point of Failure (SPOF):** If the single server hardware fails, the entire application and dataset become unavailable.
- **Cost non-linearity:** Upgrading to enterprise-grade mainframes with proprietary SAN storage grows exponentially expensive compared to scaling horizontally with commodity hardware.

3. Data Locality Solution:

- In traditional systems (*Data* → *Code*), massive data volumes are pulled across network switches to a central compute engine.
- Hadoop reverses this by applying **Data Locality** (*Code* → *Data*): the small compiled JAR/Python code (kilobytes in size) is sent over the network to the specific

DataNodes holding the 128 MB blocks locally. Computation runs in parallel locally, avoiding network overhead

Solution to Exercise 2

1. Storage Calculations:

- **a) Number of Blocks:** $1 \text{ TB} = 1,024 \text{ GB} = 1,048,576 \text{ MB}$
 $\text{Blocks} = \frac{1,048,576 \text{ MB}}{128 \text{ MB/block}} = \mathbf{8,192 \text{ blocs}}$
- **b) Total Storage with Replication:** Total Physical Space = 1 TB × 3 (Replication Factor) = 3 T (302G)

2. Cluster Topology Matrix:

Component	Node Type	Primary Responsibility
NameNode	Master	Stores file namespace metadata & block mapping
DataNode	Worker	Stores actual data blocks & executes local reads/writes
ResourceManager (YARN)	Master	Allocates cluster compute resources to applications
NodeManager (YARN)	Worker	Manages execution containers on individual worker nodes

3. Fault Tolerance Mechanism Trace:

- **Heartbeat Loss:** The NameNode regularly receives periodic Heartbeats (every 3 seconds) and Block Reports from all active DataNodes.
- **Failure Detection:** If DataNode #4 fails to send a Heartbeat for a set timeout window (e.g., 10 minutes), the NameNode marks it as *dead*.
- **Replication Check:** The NameNode inspects its metadata catalog and detects that Block #512 now has only 2 active replicas instead of the configured.
- **Automated Replication:** The NameNode commands an active DataNode holding a valid surviving copy of Block #512 to copy it over the rack switch to another healthy DataNode, restoring the replication factor to 3 without interrupting running jobs.

Solution to Exercise 3

1. Map Phase Output:

- **Function Signature:** `map(Line_Offset, Line_Text) -> List(HTTP_Status_Code, 1`
- **Mapper 1 Output (Split 1):**
 - ("200", 1)
 - ("404", 1)
 - ("500", 1)
- **Mapper 2 Output (Split 2):**
 - ("403", 1)
 - ("404", 1)

- ("404", 1)

2. **Shuffle & Sort Phase Output:**

- The framework automatically collects intermediate outputs, sorts them by key, and routes identical keys to the same reducer node:
 - Key "200" → 2
 - Key "403" → 2
 - Key "404" → 2
 - Key "500" → 2

3. **Reduce Phase Output:**

- **Function Signature:** reduce(HTTP_Status_Code, List(Counts)) -> (HTTP_Status_Code, Total_Sum)
- **Final HDFS Output File:**
 - ("200", 1)
 - ("403", 1)
 - ("404", 3)
 - ("500", 1)

4. **Combiner Optimization Concept:**

- A **Combiner** acts as a local "mini-reducer" on the mapper node prior to network transfer.
- For Mapper 2, instead of sending ("404", 1) and ("404", 1) separately across the network during Shuffle, the Combiner aggregates them locally into ("404", 2), reducing network bandwidth consumption during the Shuffle & Sort phase.