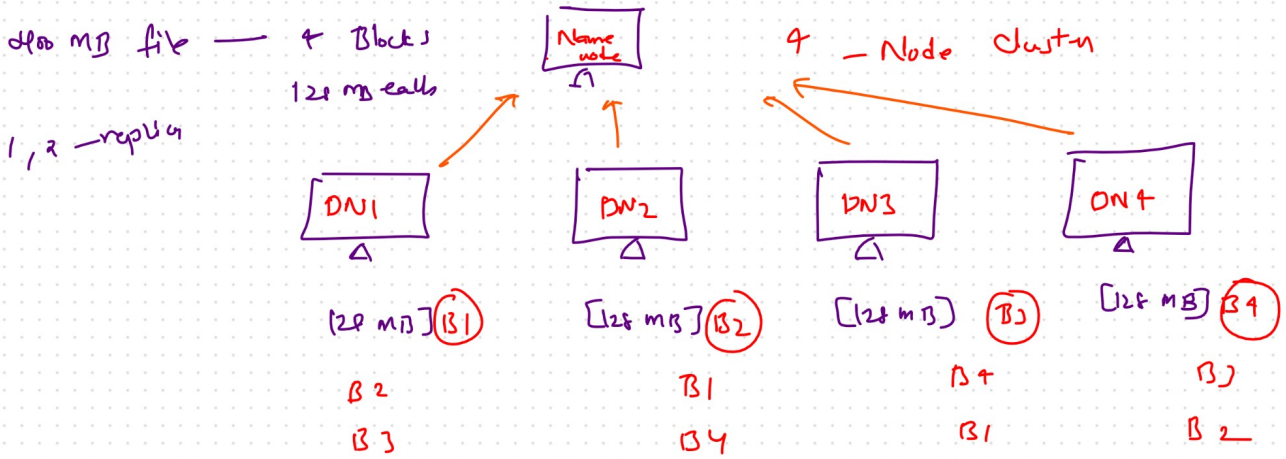
Map Reduce

— Cluster processing

Multi-threading Vs Distributed processing

Multi-threading : parallel processing within a single machine

Map-Reduce : Distributed processing across multiple machines (cluster nodes)



Node Configuration

CPU Resources (4 cores)

RAM

HDD

* Core Processing Rule: (Fundamental principle)

→ One CPU Core can process one mapper job on one block of data

* remember Import concept for understanding Map Reduce Processing

Example 1 : 4 - node cluster, each with 4 cores = 16 total cores

file : 400 MB → 4 Blocks of 128 MB each

yarn

DN1	DN2	DN3	DN4
Core 1 → Block 1	Core 1 → Block 2	Core 1 → Block 3	Core 1 → Block 4
Core 2 ← Idle	Core 2 – Idle	Core 2 – Idle	Core 2 – Idle
Core 3 – Idle	Core 3 – Idle	Core 3 – Idle	Core 3 – Idle
Core 4 – Idle	Core 4 – Idle	Core 4 – Idle	Core 4 – Idle

Ex 2 : Same 16 - core cluster

File : Large file with (≈ 4.1 GB) = 32 blocks

Processing : Can only process 16 blocks at a time

→ Batch 1 : Process 16 blocks simultaneously (using all 16 cores)

↓

→ Batch 2 : process remaining 16 blocks

Result : Processing happens in batches, takes more time but doesn't fail

Block size Impact on parallelism

Block size vs Parallelism

(1) 128 MB Block size [standard]

1 GB file / 128 MB = 8 blocks = 8 parallel processes

2) Smaller Blocksize : 64 MB

1GB File / 64MB = 16 blocks = 16 parallel processes

✓ Better parallelism

✗ More metadata overhead

3) Very small Block size : 4MB

1GB File / 4MB = 256 Blocks

✗ Too many batches needed

✗ High metadata overhead

✗ processing overhead

16 Cores

1 Batch — 16 Blocks

2 Batch — 16 Blocks

⋮

(16 Batches — 16 Blocks

256 Blocks)

4) Larger Block size : 512 MB

1GB / 512 MB = 2 Blocks = 2 parallel process

✗ Reduced parallelism

✗ underutilized cores

Block size selection:

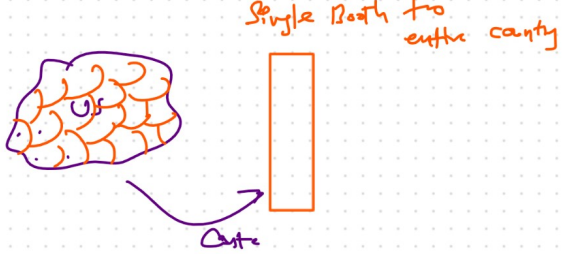
128 MB → well-researched & experimentally proven optimal size

Map Reduce

def: MapReduce is a framework for distributed data processing in Hadoop ecosystem. , helps on the processing aspect of big data operations

Key concept : Code Locality

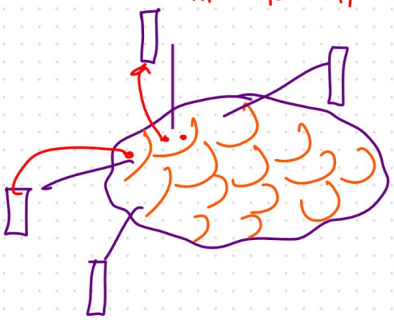
eg: Election → polling System



Example : Software Distribution

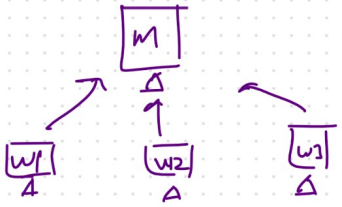
Inefficient Approach → users buy laptops to G.C Company headquarters for s/w installation

In efficient approach

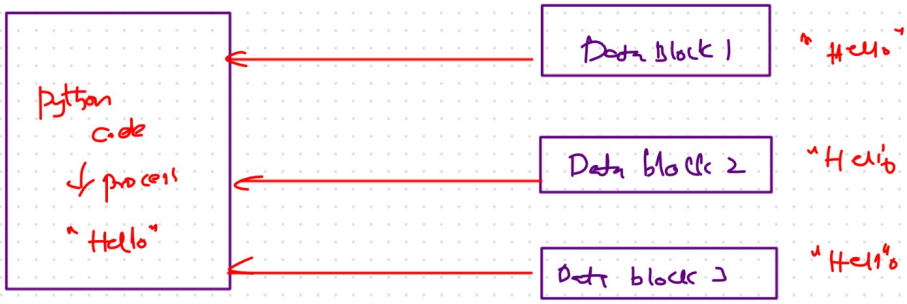


Efficient Approach : Download Software to user's device

Traditional vs Distributed Processing



Master Machine — Worker Nodes



Issues with Traditional Approach :

(1) **Data movement :** Larger Datasets travelling to processing unit (master)

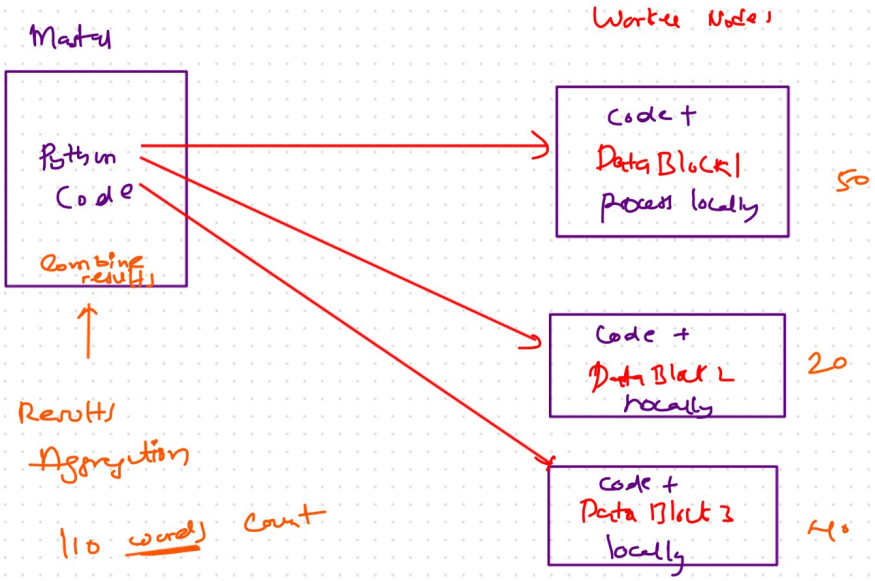
2) Single Point Processing : only master machines resources utilized

3) No Data locality : Data stored locally on workers but processed remotely

4) No local Processing : Worker node CPU/RAM resources wasted

5) Scalability Issues : Bottle neck at master node

Distributed Processing Solution :



Advantages of Distributed Approach

1) Code to Data Nodes : lightweight code (KB/MB) travels instead of heavy data files (GB)

2) Data locality : Data processed where it resides

3) Local processing : All worker nodes resources utilized

4) Parallel processing : multiple machines process simultaneously

5) Scalability : Performance improves with more machines