Apache Spark

Start at 9:05

→ • Agenda → • What is Spark
• Distributed Computing
• Pyspark

⇒ • Typical Data Size →

→ Few mb → 1-100 mb → 10,000 Row

→ 3-5 GB → 1M-30M Rows

→ → Amazon Order Table → 100M Row

→ Constraint

→ Time → Disk I/O Performance

→ Memory Constraint.

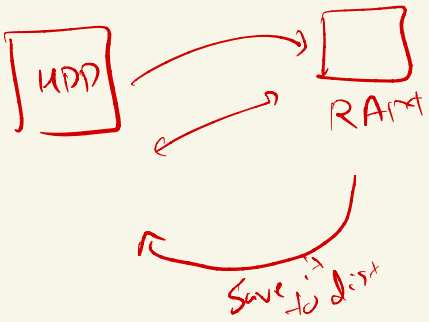
→

⇒ RAM → 16 GB

HD → 1 TB

10 GB File → Store it on → Hard Disk →

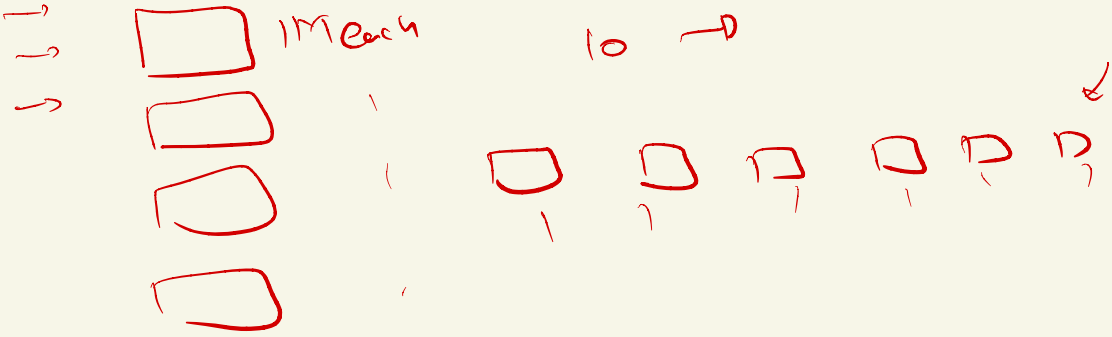
When we process →



→ 10 M rows → Fullname → FN + LN

→ 1 TB RAM >
100 TB HDD

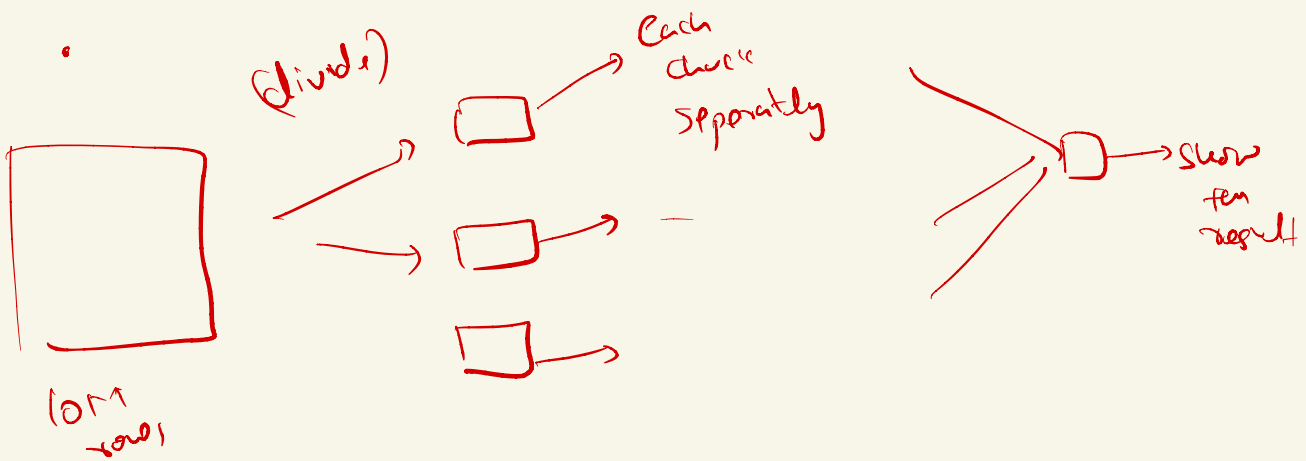
→ applied line by line | row by row



→ Lack of parallelism →
→ Python (pandas) → only a single

→ Big Data → Unstructured data.

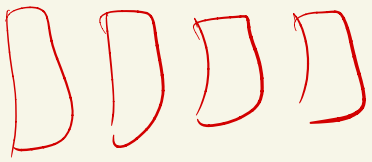
→ • Parallel Computation



→ Spark → . Very easy to use • mature
° Fast

→ open source, distributed Computing framework

→ . 100x → Traditional system (Pandas)
° Spark → Connect to almost all data source.
↳ almost all major language → . Java
. Python

Clustered Computing	Parallel Computing	Distributed Computing
Pooling resources of multiple machine →	→ Task can be run at the same time.	→ Clustered + Parallel + Done over a network.
→ Horizontal Scaling	Full name = FN + LN	
	→ 	

→ RDD → Resilient Distributed Dataset

↳ How spark store the data

→ Pyspark → built on top of Spark framework

- Handle much more data than pandas
- ML Task + Python Task

→ . Steeper learning Curve

- Setting up Spark is very painful
- Pandas is sufficient for most small cases.

Dask → even 5x Pandas

Hadoop → Storage framework

⇒ Other framework

→ Mapreduce

→ Disk based processing

99% i/o

Slow

Spark

→ In memory processing

→ Fast

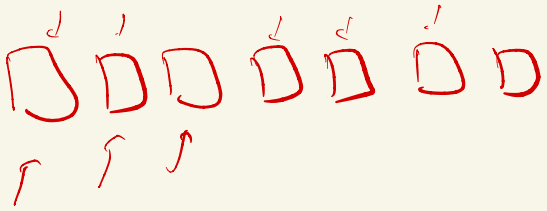
→ Spark → Store the data in distribution

→ • Multiple storage computer

• Duplication → (3x) →

- S3
- your own server
- HDFS

10 TB ⇒ 1



→ 10 TB → 10 machines →

→ 1 TB data to each machine → normal

→ • duplication replication

→  3 TB data to each machine

→ Spark Backend
↳ ① Distributed Computing
② Make replication of Data

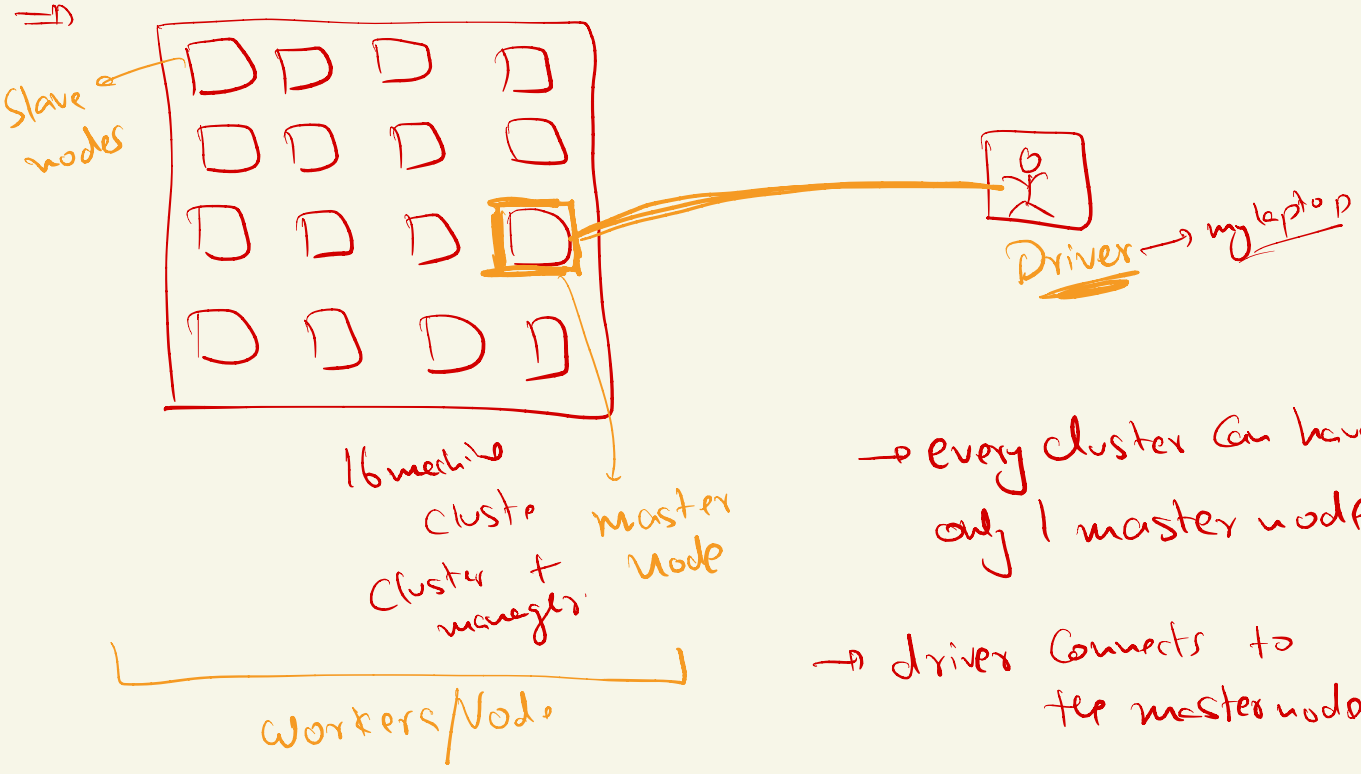
⇒ • 30 years of data →

→ We will only require small part of this data on our local machine

→ • all your Computing → on network / cloud cluster

Results get send to local machine.

⇒



→ every cluster can have only 1 master node

→ driver connects to the master node

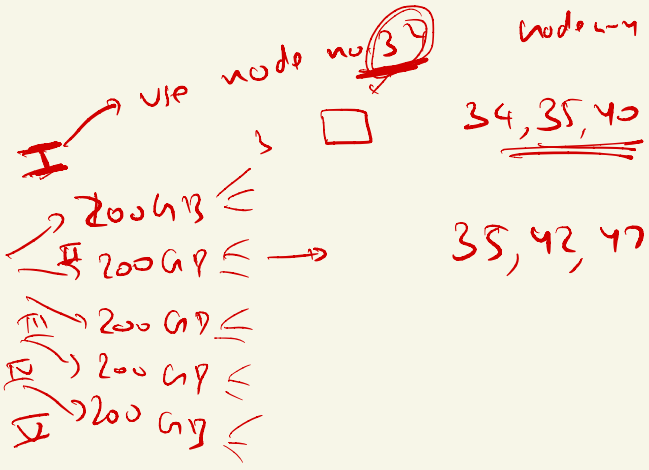
→ master node distribute the work to other workers via execution plan

→ • Data Splitting → divide into multiple chunks.
→ ↳ which column
• How many parts

• Task Distribution →

• In-memory Processing →

→ 1 TB data → Sort

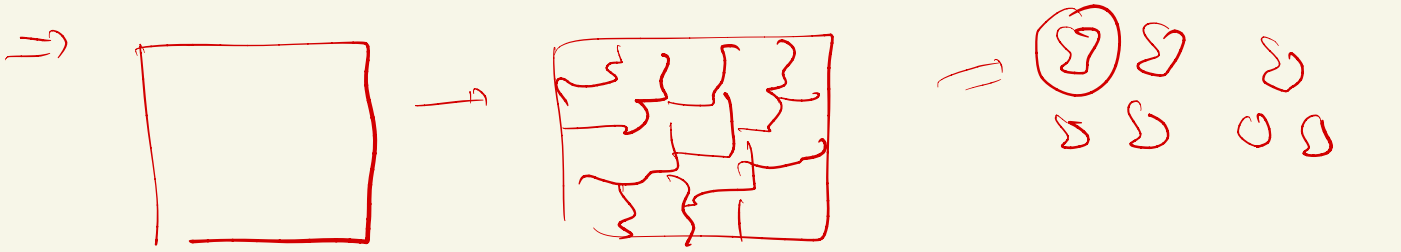


* RDD → • immutable
↳ Cannot be modified

Resilient Distributed Dataset

df (1M) → 100 K, 100 K, 100 K, 100 K, 100 K

→ Aggregation → Merging data from each worker node. Process



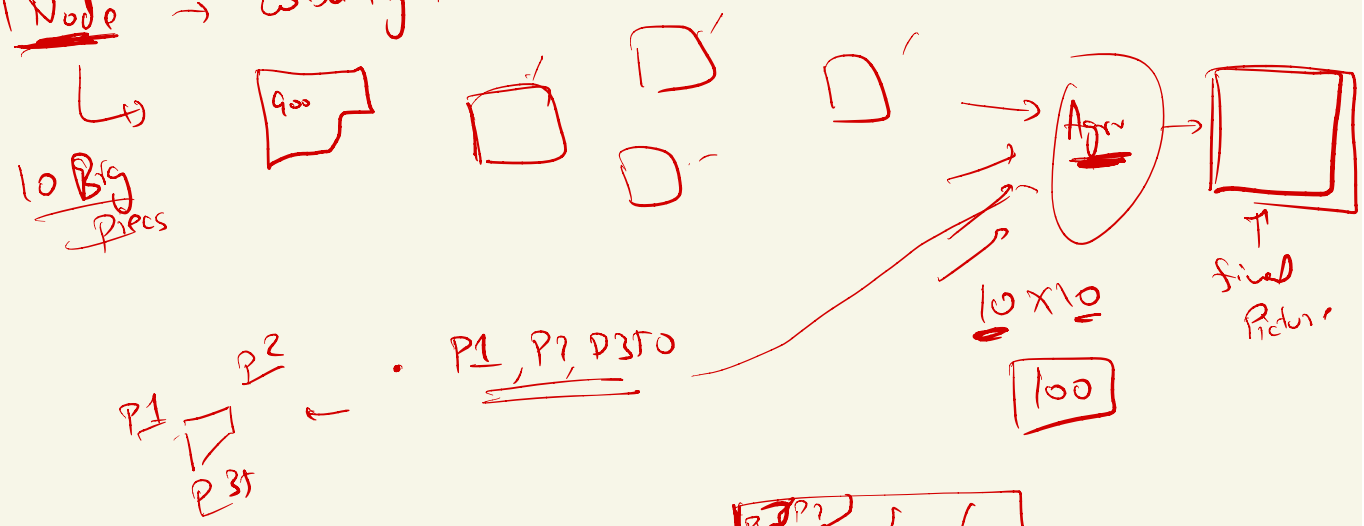
Create the original

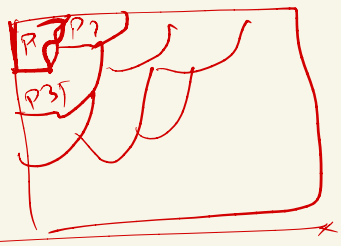
→ 100k pieces to each

10 nodes

1 Node → will try to solve the puzzle in it

10 Big pieces





⇒ Partitioning → • Improving Performance
• Consistent data across node
• Help in faults tolerance
• Efficient Storage

100 TB → 40 TB, 5 TB, 5 TB

→ Flipkart → daywise sales

⇒ data for each day → 2nd → Index → find avg of this data

⇒ BB Sales Com

→ 28th Jun

29th Jun

100 K → 1 M orders →

Doubts

We used sagemaker's xgboost and trained it using a different instance to reduce cost. But for models like KNN, DBSCAN, algomerative clustering, for which if you need the prediction, you need to retrain the whole model again, how does it work?

⇒ Xgboost → Saves → Separate instance → final model → Shut the instance

→ Instance you running for sagemaker → increase size → just as normal code

→ • → Sagemaker instance → Pythonic operations Basic EDA

→ Training instance →

→ deployment → Model 1 → 100k visits/day →

2 → 10k visits/day →

3 → 100 visits/day →

	day to day Use	Interview
• Git	✓	
• Flask	✓	—
• ML System design	—	✓
ML Flow	✓	
Spark	—	✓
Docker	—	—