Big Data

Lecture 1. Introduction.

4th V does not only apply to Big Data (opinion)   Same with 5th → added by business people to maintain hype.

big data $\not\Leftrightarrow$ volume.   velocity (streaming) ⇒ big data.

- Shared nothing: communication, consistency issues. (Spark)

Map/Reduce strategy → difficult!

- Lambda architecture: (λ):   IoT example with trucks.

↳ divide data in batch (persistent; inmutable) → master data   it precoumptes queries
speed (real time)
↳ serving layer.

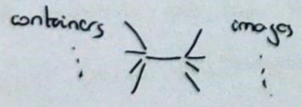
Lecture 2. Intro to CS/Cloud.

Lecture 3. Dockerize model.

- Docker (SAAP): container management.   vs.   Virtual Machines (good isolation // higher overload)
  ↳ has everything an app needs to run.
  ↓
  slim
  easy ship, pack, run
  easy CI/CD
  scability

- CLI: intermediario en las couminaciones al servidor.

- docker: HOST



docker run -p (-d) 8080:8080 -v $\underbrace{\text{volume-path}}_{\text{your disk}}$ ~~#~~: $\underbrace{\text{mount}}_{\text{inside container.}}$ jupyter/datascience-notebook

- Kubernetes: manages clusters of hosts (against docker images not having enough server)
  ↑ usable by anyone if app is dockerized.

Lecture 4. NoSQL : Not only SQL.

↳ scalable, fast, flexible, easy...

schemas (vs. SQL: rigid)

• Disadvantadge: no standarization rules (each DB has its own query language)

bad with relational Data

...

• Types: 1) Key value: big hash map (distributed) (Key, value) pairs. (Amazon shopping basket)

2) document-based: stores data as JSON objects. MongoDB. Elastic Search → indexes to categorize similar data. ↗ Wikipedia

3) column-based: BigTable, Kassandra. SQL-likewise, difference in how it is stored. → we store rows → each one is HashMap (each can have its own columns)

Hash-Map of Hash-Map.

4) graph-based: model relationships between entities

DATA-MODELING. generally not main DB.

↳ Fraud detection.

SQL queries can crash with really large joins → MongoDB.

Check equivalencies Mongo - SQL.

Lecture 5. Hadoop & S3

• Hadoop Distributed File System (HDFS) + MapReduce (Computation)

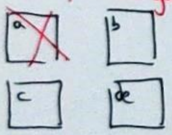
storage

do not your data to your local (huge!) ⇒ take your code to the server.

- blocks: storage units (files are divided in blocks).

data distribution

↗ if one goes down = corrupted data.



REPLICATE EACH BLOCK THRICE

↳ name node shows where each block is. (up to million of files)

↳ use this for processing.

• S3 : object storage (id, data, metadata) only to dump data there

not splitted. unlimited.

↓

Simple Storage Service

durable

two types of data: frequent or not accesses. → storage class.

↓ Standard

↓ Standard-IA, One Zone-IA (for backups)

↓ Intelligent Tiering (unknown, adaptative)

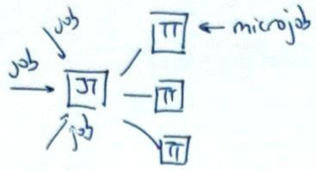
Glacier: logs and archives.

- Hadoop Processing:

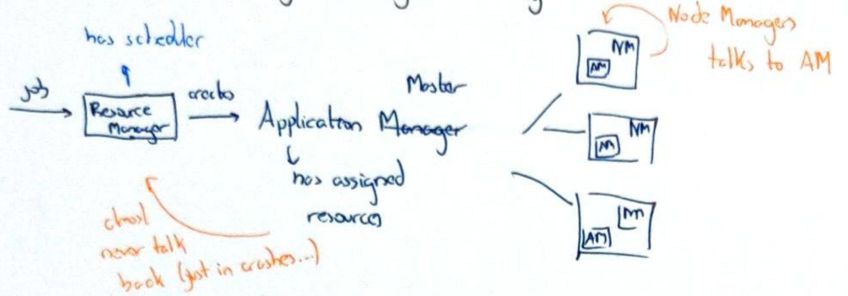
- MapReduce: $\underbrace{\text{distribute by Keys after}}_{\text{reduce}}$ $\underbrace{\text{dividing into machines.}}_{\text{map}}$

input splits → map → shuffle → reduce

Job/TasK-tracker for status managements.



- Hadoop 2: Yarn for resource management + job scheduling.


has scheduler
job
Resource Manager
creates
Application Master Manager
has assigned resources
almost never talk back (just in crashes...)
NM
AM
NM
AM
NM
AM
Node Managers talks to AM

NM ≈ TT

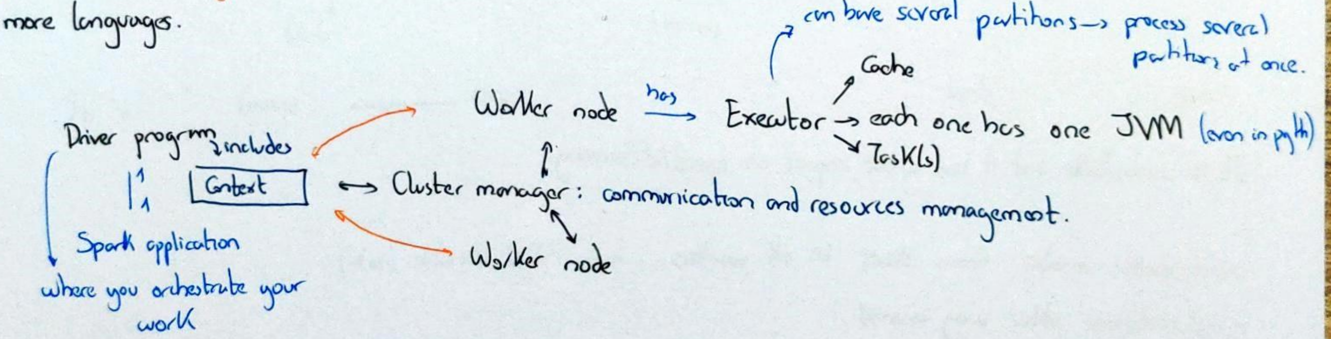
AM ≈ JT (for one job only though)

Spark uses the same architecture.

L6. SparK Fundamentals

- Not a replacement of Hadoop or BD storage: big data processing and analysis (instead of MapReduce)
  - Automatically distributes the work (fast)
  - Interactive exploration and analytics.
  - Easier code in more languages.

- Architecture:


Driver program
includes
Context
Spark application
where you orchestrate your work
Worker node
has
Executor
can have several partitions → process several partitions at once.
Cache
each one has one JVM (even in python)
TasK(s)
Cluster manager: communication and resources management.
Worker node

obs: lazily ≡ created only when needed.

- RDD: inmutable collection that is partitioned and distributed among workers. → many types.
- Lineage: tree of transformations. → you can recreate a piece if lost and not all. 〉 DAGs. no cycle graph.

transformations act on RDDs (data) / action: you want other Kind of output count() (if you call twice an action, it will do everything again)

lazy until you call an

filter, coalsce (change number of packages)

~~(cost)~~ ← save the RDD for reuse!

(cache())

- pair RDD: "dictionaries"

no loops in RDD (items are distributed)

reduce for operations with two items.

L8. Spark SQL

- Datasets and dataframes

  ↓ Typed: string? int?   → untyped.

- SparkSession are the new entrypoint (instead of SparkContext)

  ↳ better for SQL context.

  - DataFrames can be created from RDDs, hive, data sources

    ↳ RDDs in the background.

- UDF in SQL: User defined functions.

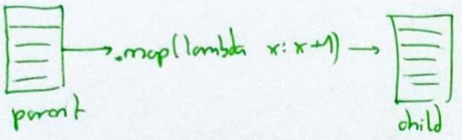
L9. Spark Advanced

- Optimization: spark pipeline runs as much code as possible in a same stage.

  new stages have to be created when data needs to move across stages (shuffling)

Spark is immutable ⇒ each transformation ≡ a new RDD

- Narrow dependency (fast) each partition of parent RDD is used at most by one partition of child RDD



  Wide " slow " " " " " " used ______ several " s of ...

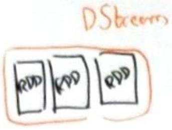
It is controllable and it has great impact on speed/efficiency.

- broadcast variables: from driver to all workers. (non RDD variables (data))
- accumulators: other way around.

L10.

- Window SQL Function: group in a window a set of rows.
  - Spark SQL: window functions: 1) Ranking functions
    2) Analytic functions
    3)

- Streaming processing: spark streaming, flink, apache storm.

  ↳ microbatching: divide strem in batches. with DStreams. (Data Stream) ou can access to specific RDDs: 

we need