Distributed ML with Apache Spark

...

By @satwikkansal

About me

- Former Software Developer @ Myntra
- Co-founder @ Frobyt
- Author of “What the f*ck Python”, “Hands on RL with Tensorflow”
- GSoC ‘17 student, GCI ‘17 mentor, GSoC ‘18 mentor

Distributed Machine Learning with Apache Spark

Goal of the session

10 - Is world's leading expert on the idea.

9 - Can ask expert questions and generate new information/data on the idea.

8 - Can answer expert questions and reconcile contradictory thoughts about the idea.

7 - Can answer any layman's question and forms independent thoughts on the idea.

Spark

6 - Can answer any layman's question and forms intelligent opinions on the idea.

5 - Knows about the idea, and can discern inaccurate statements about the idea.

4 - Knows about the idea, and can explain what's been learned in one's own words.

3 - Heard of the idea, and recites what others have said about it.

2 - Heard of the idea, but doesn't know anything about it.

Spark

1 - Never heard of the idea.

What do we mean by **Distributed** mean here?

Distributed: Multiple devices a.k.a cluster of devices working in coordination to carry out a computation.

ML: Learning from data and generating inferences. Supervised / Unsupervised.

Okay, but what is **Apache Spark**?

Spark: General-purpose cluster-computing framework for processing big data.

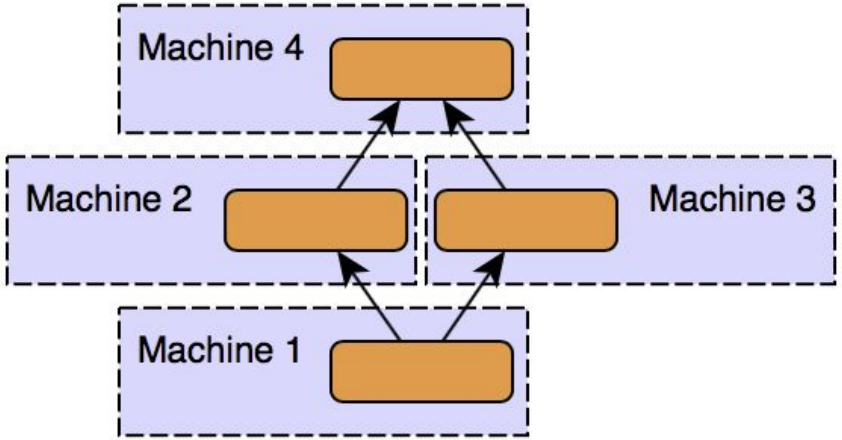
If I know numpy / tensorflow / pytorch already. Why use Spark for ML?

Why use Spark for ML?

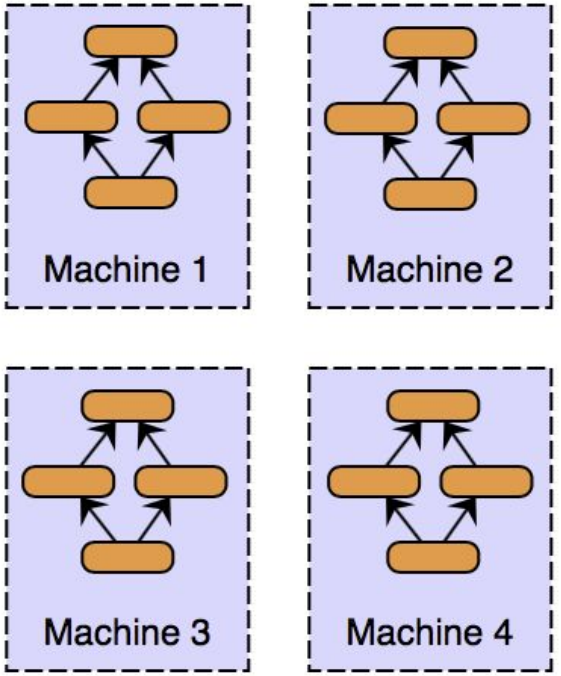
- To process large volumes of data.
- Parallelize parts of computation that are independent to each-other.
- Scale-on-demand while keeping your old hardware still useful.

Different approaches to parallelism

Model Parallelism



Data Parallelism



Thinking in terms of “data-parallelism”

Map-Reduce paradigm

What is MapReduce programming paradigm

- Programming model based on “split-apply-combine” strategy.
- Map operation maps the data to zero or more key-value pairs.
- Reduce function takes in those key-value groups and performs aggregation to get the result.
- MapReduce execution framework takes care of handling the data, running the map and reduce operations in a highly optimized and parallelized manner on the cluster, resulting in a scalable solution.

Map-Reduce Example

40 random cards from a deck of cards

Objective: Compute sum of numeric cards for each suite.

How would,

- A single person do it?
- A group of 4 people do it?

Single-person style (4o cards)

4 people style (**Split** into 10 cards each)

Apply the same process (Map)

Person A (10 cards)

Spades	7
Diamonds	5
Clubs	8
Hearts	10

Person B (10 cards)

Spades	11
Diamonds	3
Clubs	4
Hearts	9

Person C (10 cards)

Spades	5
Diamonds	15
Clubs	6
Hearts	4

Person D (10 cards)

Spades	2
Diamonds	8
Clubs	6
Hearts	12

Sort and shuffle the results

Person A (All spades)

Spades (A)	7
Spades (B)	11
Spades (C)	5
Spades (D)	2

...
Person B (All diamonds)
...

...
Person C (All clubs)
...

...
Person D (All Hearts)
...

Combine (Reduce)

Person A: Sum of spades is 25

Person B: Sum of diamonds is 34

Person C: Sum of clubs is 24

Person D: Sum of hearts is 35

A MapReduce framework implicitly exploits data-parallelism by partitioning the data and assigning concurrent processing tasks to the cluster nodes that can run independently.

The term "**independently**" is important here, because if there's a dependency between the tasks, i.e., tasks need to wait for other tasks to finish, the efficiency gets reduced.

Note: Spark is more than just Map-Reduce.

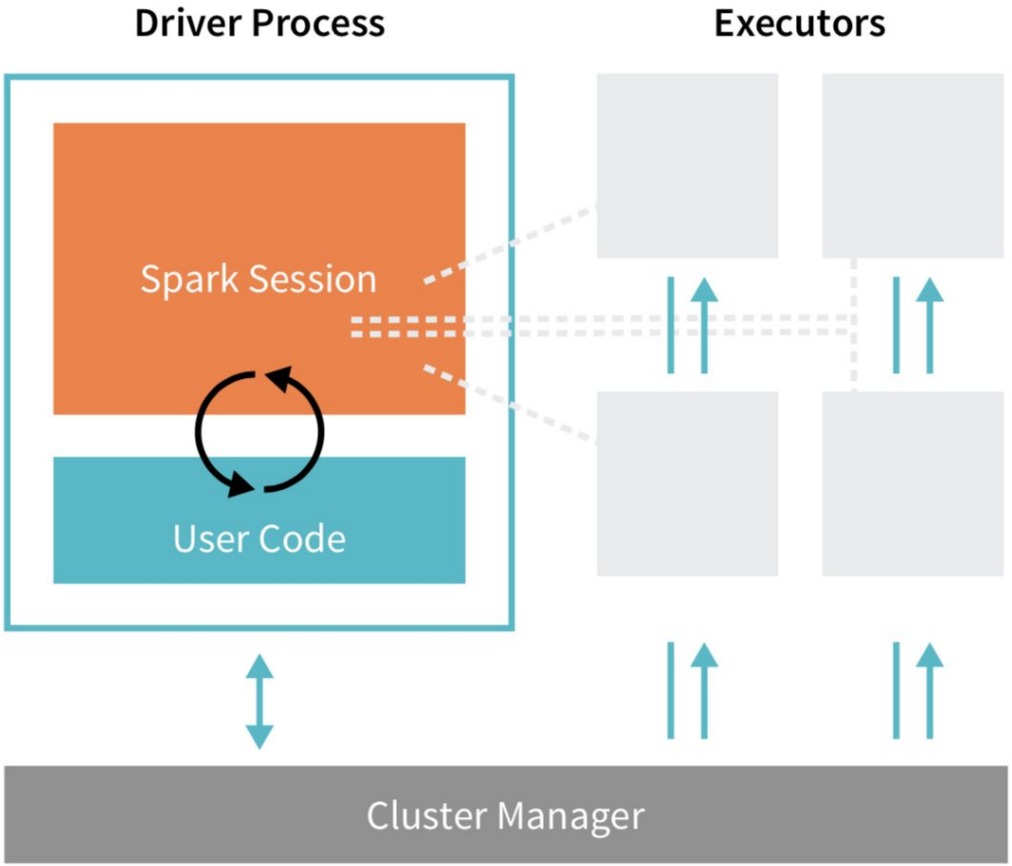
For History Fanatics

How Spark came into picture?

- Google pioneered MapReduce for running queries on its massive datasets of web pages.
- Apache Hadoop is currently the most popular MapReduce framework, which was started as an open source project to implement the ideas discussed in the MapReduce paper published by Google.
- MapReduce kind of proved that cluster of machines working together is both faster and cheaper than a single really powerful machine.
- As time passed, multiple limitations of in Hadoop became apparent, and efforts were made to get around those limitations. The Spark project was launched as one such effort and is maintained by Apache.

Spark performs better (than Hadoop) for iterative tasks where data can easily fit into the memory; that's why it's a good choice for performing machine learning.

Spark Architecture


Driver Process
Executors
Spark Session
User Code
Cluster Manager

The Driver process

Spark has a master-slave (driver-worker) architecture. The driver is the central-coordinator.

It connects to the Cluster manager for managing worker nodes in which executors run. Driver and each of the executors run in their Java processes.

Local and distributed modes

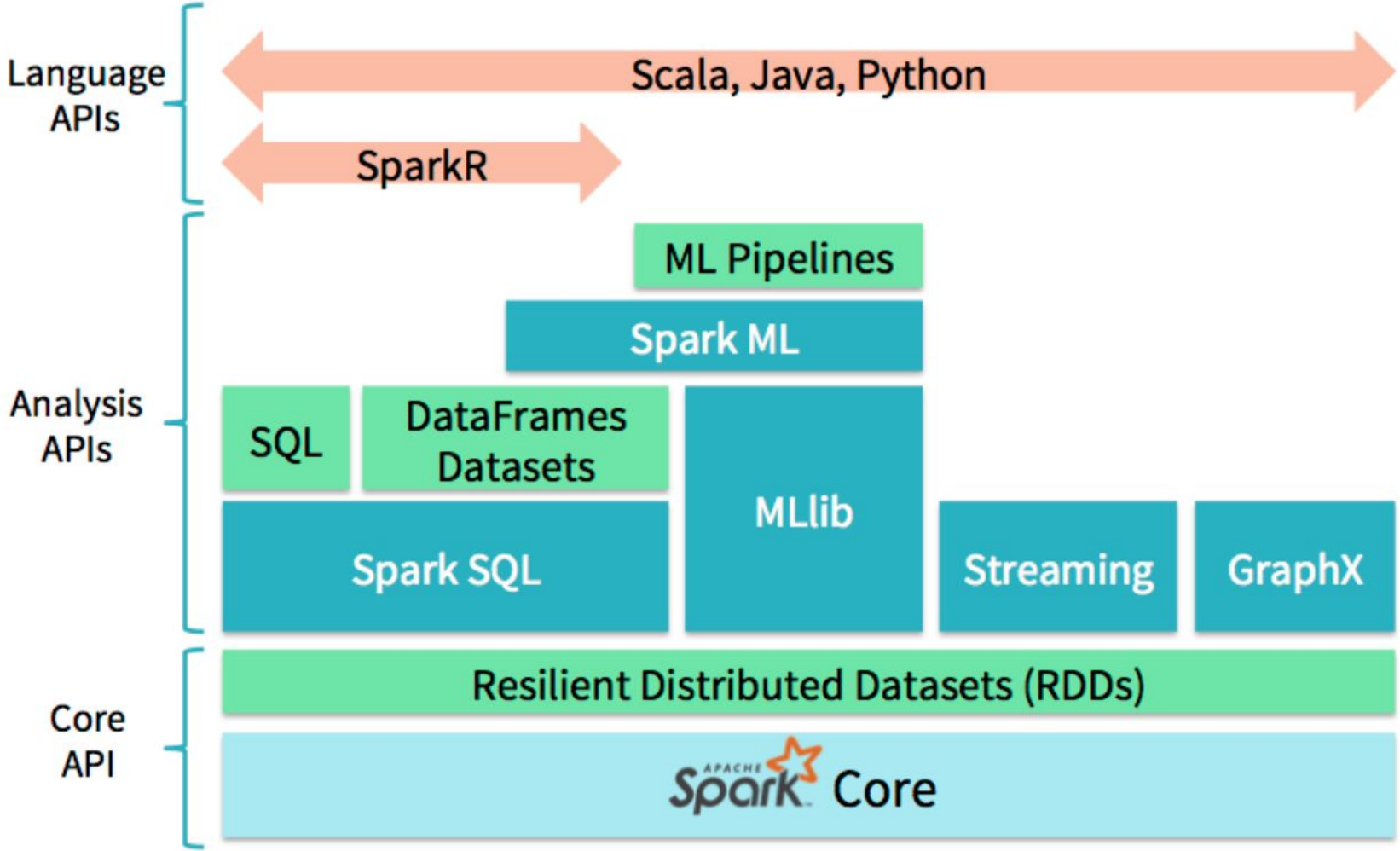
Spark local

- Local mode is suitable for testing and prototyping.
- Spark local is a multi-threaded non-distributed runtime environment.
- The degree of parallelism is determined by the number of threads specified at the time of instantiating the Spark Application.
- Doesn’t require a cluster manager.
- The drivers and the executors are spawned inside the same JVM in Spark local.

Cluster manager

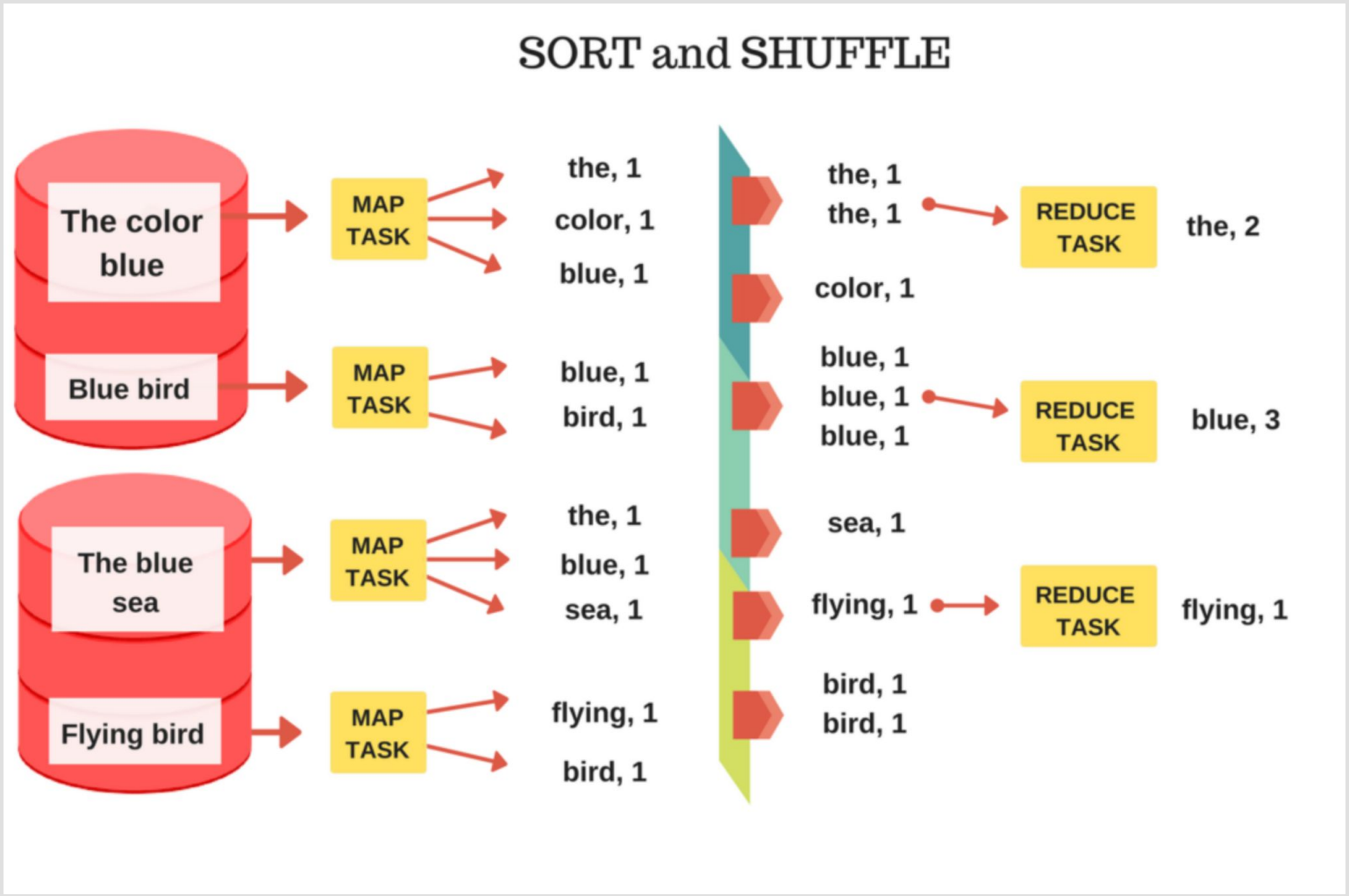
- The distributed mode, on the other hand, requires a cluster manager
- A cluster manager is responsible for spawning executor processes on the workers.
- The workers communicate the resource availability (CPU and memory) to the cluster manager, and the cluster manager conveys this information to the driver whenever requested.
- Spark has a built-in cluster manager called Spark Standalone, and we can also plug other cluster managers like Hadoop YARN and Apache Mesos.

Layers of abstraction of Spark APIs


Language APIs
Scala, Java, Python
SparkR
Analysis APIs
ML Pipelines
Spark ML
SQL
DataFrames Datasets
Spark SQL
MLlib
Streaming
GraphX
Core API
Resilient Distributed Datasets (RDDs)
Spark Core

Hello PySpark! (PySpark hello world)

The cliched word count example.


SORT and SHUFFLE
The color blue
Blue bird
The blue sea
Flying bird
MAP TASK
MAP TASK
MAP TASK
MAP TASK
the, 1
color, 1
blue, 1
blue, 1
bird, 1
the, 1
blue, 1
sea, 1
flying, 1
bird, 1
the, 1
the, 1
color, 1
blue, 1
blue, 1
blue, 1
sea, 1
flying, 1
bird, 1
bird, 1
REDUCE TASK
REDUCE TASK
REDUCE TASK
the, 2
blue, 3
flying, 1

Some important PySpark constructs

RDDs

Resilient Distributed Databases

Partitioned: An RDD is divided into multiple partitions. Different executors can concurrently perform computations on different partitions of RDD.

Distributed: An RDD resides on multiple nodes across the cluster. A node need not have all the partitions of RDD.

Resilient: If one of the nodes containing the partitions fails, Spark has mechanisms to recover the lost data.

I**mmutable**: An RDD is a read-only object and cannot be modified. Transformation operation on RDDs generates a newer RDD.

Operations in Spark

Transformations and Actions

Transformation operations in PySpark

A transformation operation returns a new RDD.

Some commonly used transform operations in PySpark are:

- `map, flatMap, mapValues`
- `filter`
- `distinct`
- `union, intersection, subtract, cartesian`
- `reduce, reduceByKey`

Actions in PySpark

An action returns a non-RDD object.

Some commonly used actions in PySpark are:

- `collect`
- `count, countByValue`
- `take, takeOrdered`
- `top`

Execution in Spark

Execution plans

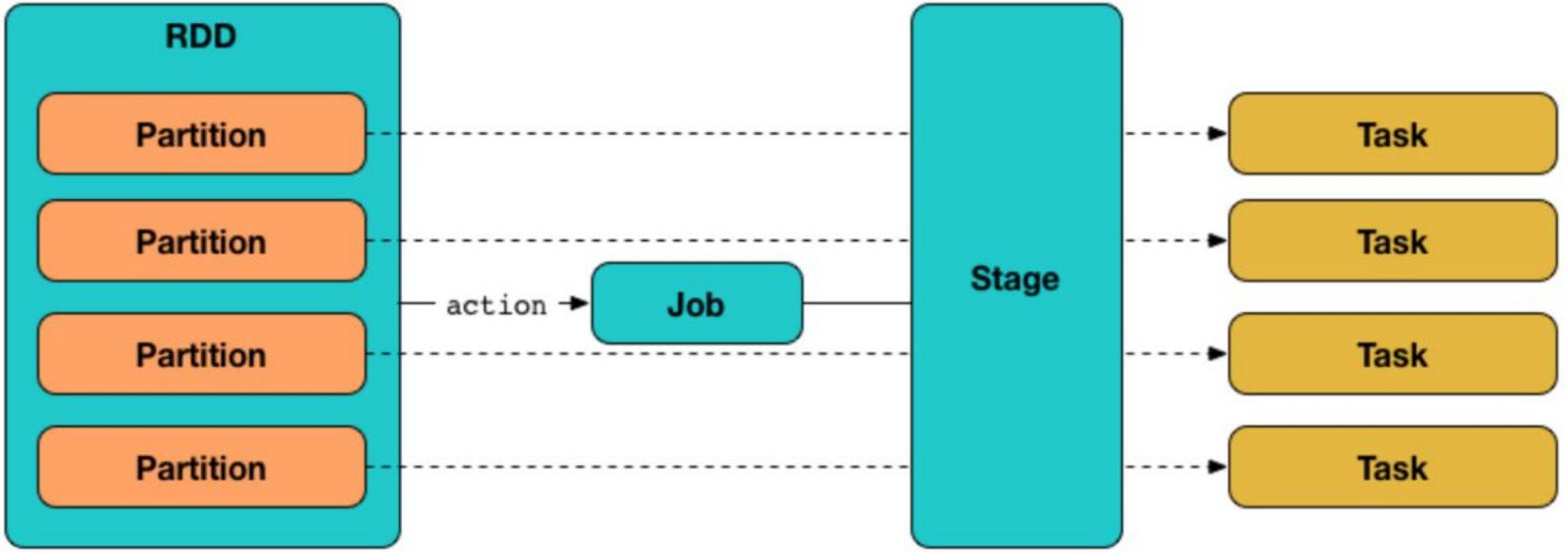
- Spark records the sequence of operations applied to RDD in the form of Direct Acyclic Graph (called RDD lineage).
- Two important components of SparkContext are the "DAGScheduler" and "TaskScheduler".
- The DAGScheduler transforms the logical execution plan (RDD lineage) to physical execution plan (TaskSets), one that's actually executed on workers.
- TaskScheduler is responsible for distributing these tasks to the executors through cluster manager.

Jobs, stages and tasks

- A Job in spark is the logical unit of execution, which is submitted to the DAGScheduler to compute the result of an action.
- A Stage is a physical unit of execution, it is a step in the physical execution plan. A stage consists of a set of parallel tasks (for different RDD partitions and executors).
- The Tasks are the smallest physical units of execution responsible for performing computation on a single partition of RDD.

Lazy execution

- The transformation operations are executed lazily in Spark, i.e., the job is not submitted to the DAGScheduler for generating theTaskSets until an action is encountered in the RDD lineage.
- Lazy execution is the reason why printing the RDD in our example above just prints the object information and not the data (somewhat similar to generators in python). When the collect action is invoked, the operations are executed, and we get our data.

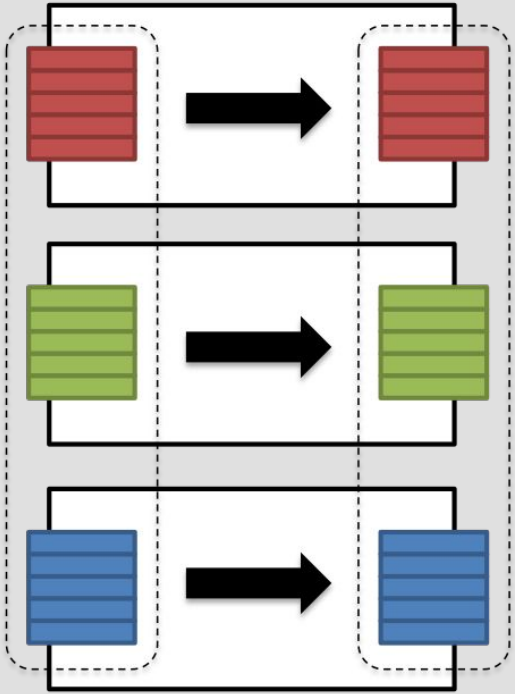

RDD
Partition
Partition
Partition
Partition
action
Job
Stage
Task
Task
Task
Task

Narrow and wide transformations.

- Narrow transformations are the result of operations like `map` and `filter` which depend on data from a single partition only.
- An output RDD for these operations has partitions with records that originate from a single partition in the parent RDD.
- Wide transformations are the result of operations like `reduceByKey`. The data required to compute the records in a single partition may reside in many partitions of the parent RDD.
- All of the tuples with the same key must end up in the same partition, and are processed by the same task.
- To satisfy these operations, Spark must execute RDD shuffle, which transfers data across cluster and results in a new stage with a new set of partitions.

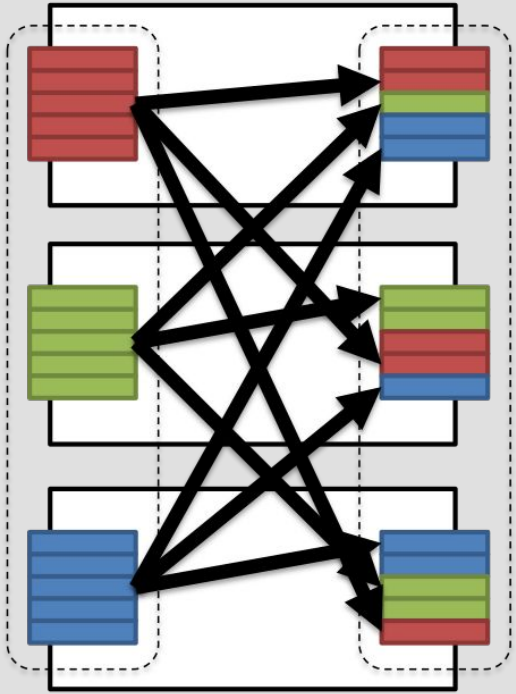
Narrow transformation

- Input and output stays in same partition
- No data movement is needed

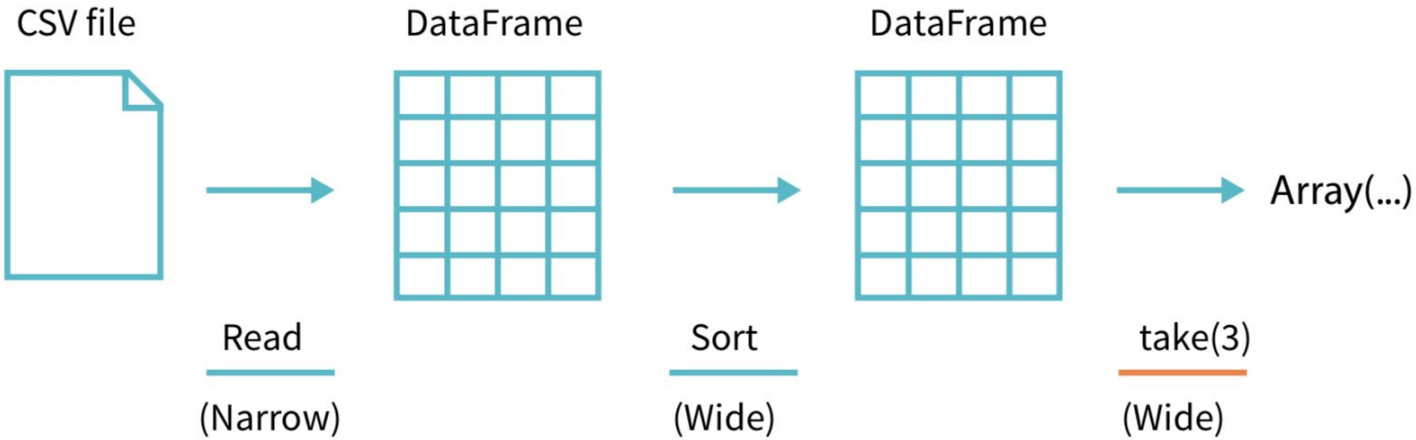


Wide transformation

- Input from other partitions are required
- Data shuffling is needed before processing

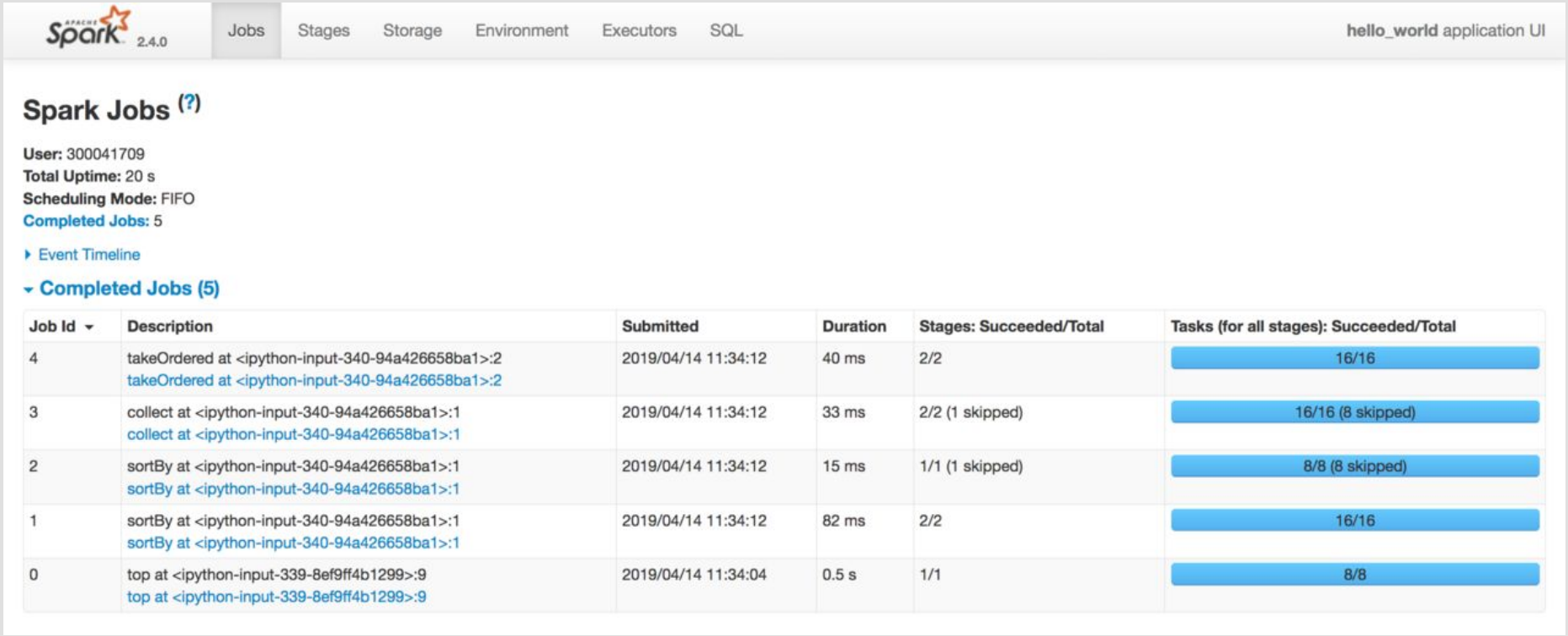


Narrow and wide transformations.



Putting everything together into perspective...

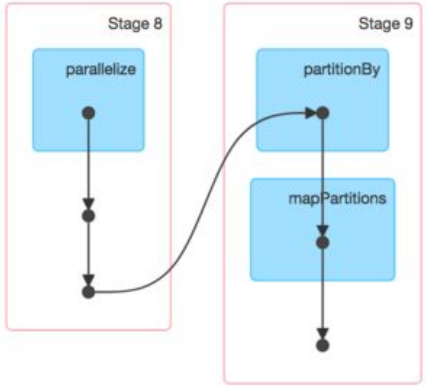
Visualizing things with Spark dashboard


Spark 2.4.0
Jobs
Stages
Storage
Environment
Executors
SQL
hello_world application UI
Spark Jobs (?)
User: 300041709
Total Uptime: 20 s
Scheduling Mode: FIFO
Completed Jobs: 5
Event Timeline
Completed Jobs (5)
Job Id
Description
Submitted
Duration
Stages: Succeeded/Total
Tasks (for all stages): Succeeded/Total
4
takeOrdered at <ipython-input-340-94a426658ba1>:2
takeOrdered at <ipython-input-340-94a426658ba1>:2
2019/04/14 11:34:12
40 ms
2/2
16/16
3
collect at <ipython-input-340-94a426658ba1>:1
collect at <ipython-input-340-94a426658ba1>:1
2019/04/14 11:34:12
33 ms
2/2 (1 skipped)
16/16 (8 skipped)
2
sortBy at <ipython-input-340-94a426658ba1>:1
sortBy at <ipython-input-340-94a426658ba1>:1
2019/04/14 11:34:12
15 ms
1/1 (1 skipped)
8/8 (8 skipped)
1
sortBy at <ipython-input-340-94a426658ba1>:1
sortBy at <ipython-input-340-94a426658ba1>:1
2019/04/14 11:34:12
82 ms
2/2
16/16
0
top at <ipython-input-339-8ef9ff4b1299>:9
top at <ipython-input-339-8ef9ff4b1299>:9
2019/04/14 11:34:04
0.5 s
1/1
8/8

Details for Job 4

Status: SUCCEEDED
Completed Stages: 2

▸ Event Timeline
▾ DAG Visualization



▾ Completed Stages (2)

Stage Id ▾	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
9	takeOrdered at <ipython-input-340-94a426658ba1>:2	+details	2019/04/14 11:34:12	13 ms	8/8			2028.0 B	
8	reduceByKey at <ipython-input-340-94a426658ba1>:2	+details	2019/04/14 11:34:12	24 ms	8/8				2028.0 B

Details for Stage 8 (Attempt 0)

Total Time Across All Tasks: 0.1 s
Locality Level Summary: Process local: 8
Shuffle Write: 2028.0 B / 26

- DAG Visualization
- Show Additional Metrics
- Event Timeline

Summary Metrics for 8 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	10 ms	15 ms	18 ms	21 ms	22 ms
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms
Shuffle Write Size / Records	0.0 B / 0	0.0 B / 0	458.0 B / 6	471.0 B / 6	499.0 B / 6

Aggregated Metrics by Executor

Executor ID ▲	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks	Shuffle Write Size / Records	Blacklisted
driver	192.168.31.102:57487	0.1 s	8	0	0	8	2028.0 B / 26	false

Tasks (8)

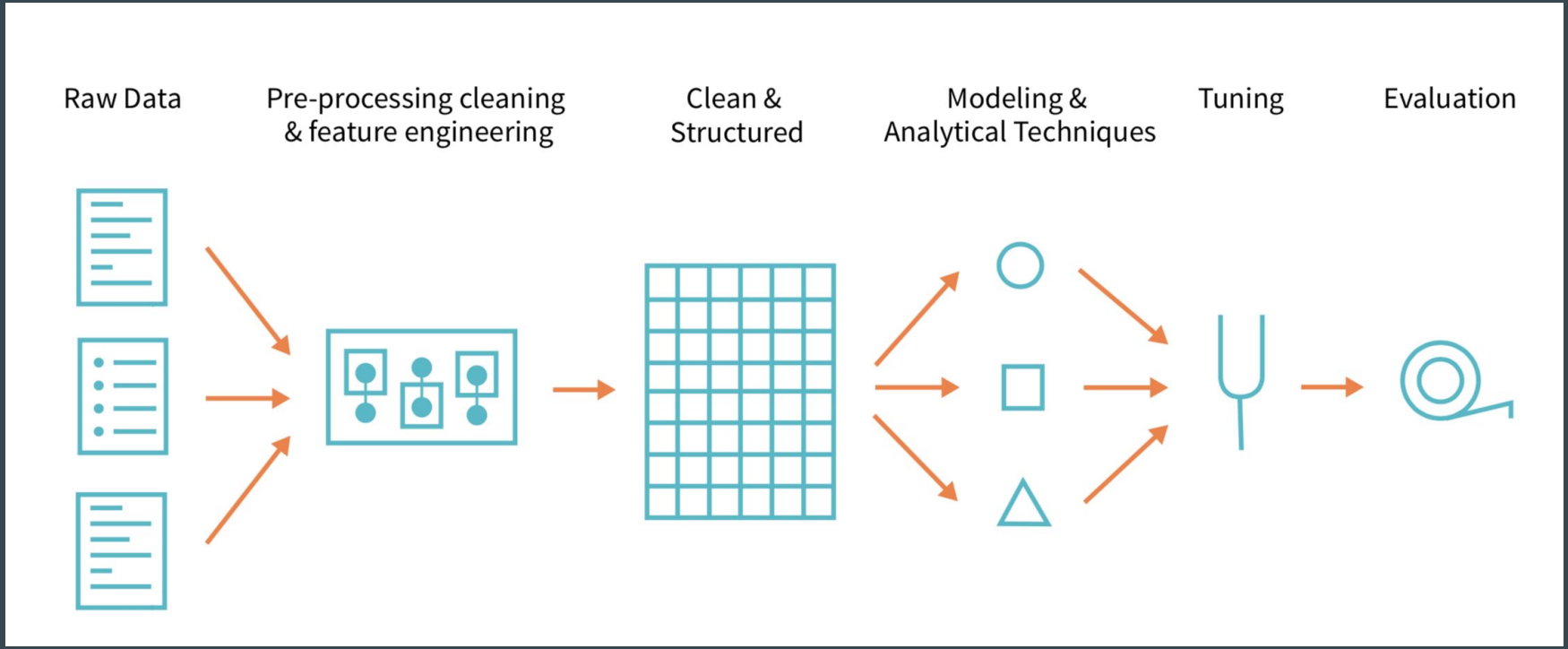
Index ▲	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	GC Time	Write Time	Shuffle Write Size / Records	Errors
0	48	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2019/04/14 11:34:12	14 ms			0.0 B / 0	
1	49	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2019/04/14 11:34:12	20 ms		5 ms	458.0 B / 6	
2	50	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2019/04/14 11:34:12	13 ms			0.0 B / 0	
3	51	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2019/04/14 11:34:12	24 ms		9 ms	458.0 B / 6	
4	52	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2019/04/14 11:34:12	22 ms		4 ms	471.0 B / 6	
5	53	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2019/04/14 11:34:12	15 ms			0.0 B / 0	
6	54	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2019/04/14 11:34:12	20 ms		5 ms	499.0 B / 6	
7	55	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2019/04/14 11:34:12	18 ms		2 ms	142.0 B / 2	


Scheduler Delay
Task Deserialization Time
Shuffle Read Time
Executor Computing Time
Shuffle Write Time
Result Serialization Time
Getting Result Time
driver / localhost
630
635
640
645
650
11:34:12

Let’s implement linear regression in PySpark

Typical ML process

1. Gathering and collecting relevant data
2. Data cleaning, and inspection (EDA)
3. Optional data processing and train-test split
4. Train test split and generating some candidate models.
5. Evaluating and comparing models
6. Using the model to make predictions, recommendations, detect anomalies or solve more general business challenges.


Raw Data
Pre-processing cleaning & feature engineering
Clean & Structured
Modeling & Analytical Techniques
Tuning
Evaluation

Spark in production: Beyond local environment and linear regression

Use Structured APIs (Dataframes and Datasets)

SparkML and MLlib

SparkML and MLlib

- Both packages leverage different core data structures.
- SparkML maintains an interface for use with Spark DataFrames.
- SparkML also maintains a high level interface for building machine learning pipelines that help standardize the way in which you perform the above steps.
- MLlib maintains interfaces for Spark's Low-Level, RDD APIs. o use its features, we've to work with data models like Vector, LabeledPoint, and Matrix.
- MLlib has now been pushed to maintenance mode.

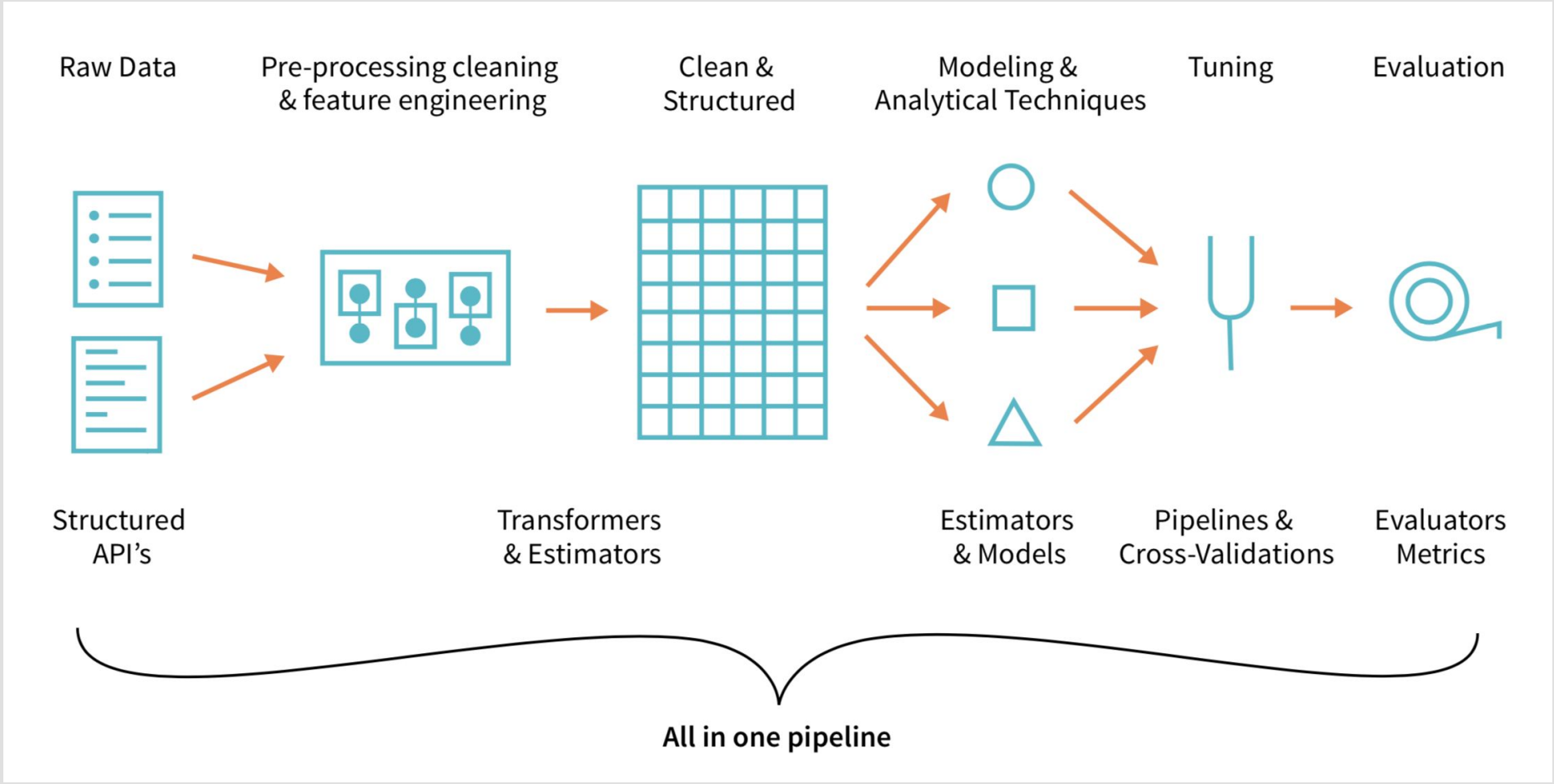
High level SparkML concepts

- **Transformers**: Pre-processing the raw data and feature generation (manipulate columns)
- **Estimators**: Models that we'll be training to make predictions
- **Evaluators**: Mechanisms to evaluate estimators based on some criteria.
- **Stage**: Single pass of data through transformers / estimators / evaluators.
- **Pipeline**: Sequential specification of stages, similar to scikit-learn's pipeline concept.

Some nice-to-know things

For most algorithms,

- The training data has a single feature column called “features”, and label column called “label”.
- Accepted data-structure for features is “Vector of Doubles” (could be sparse or dense).
- You can use R-formula for specifying transformations declaratively.
- Fitting an ML model (calling **fit** method) is always eagerly performed.
- To make predictions, we call the **transform** method on the fitted model.


Raw Data
Pre-processing cleaning & feature engineering
Clean & Structured
Modeling & Analytical Techniques
Tuning
Evaluation
Structured API's
Transformers & Estimators
Estimators & Models
Pipelines & Cross-Validations
Evaluators Metrics
All in one pipeline

Structured streaming

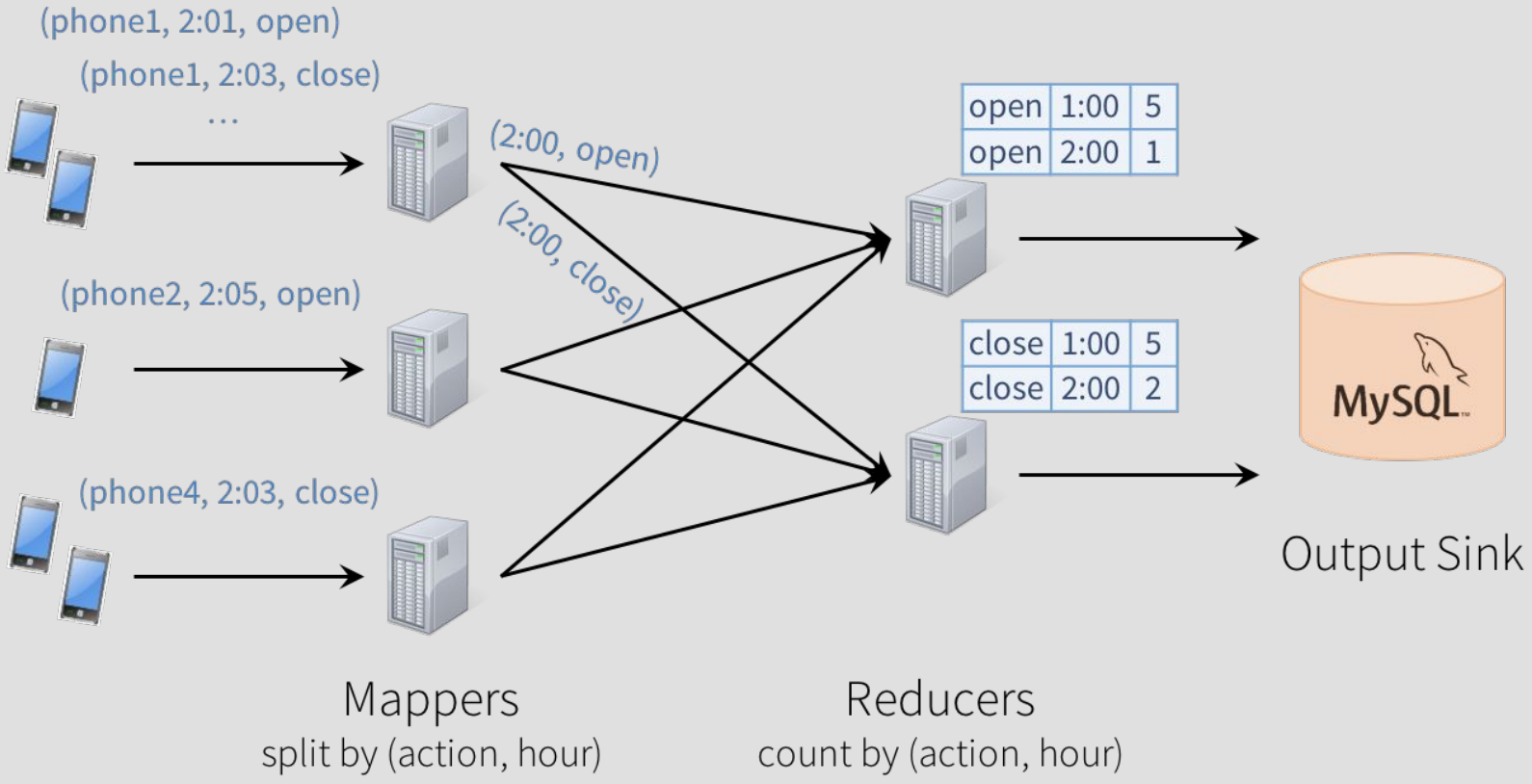
Structured Streaming in Spark

High-level API for building continuous applications, Spark 2.x onwards

How it works?

- You can load stream from “replayable sources” (Kafka, Kinesis, filestream, etc) into Datasets and Dataframes.
- You can then describe the batch-job like query, and spark will take care of running them incrementally incrementally, ensuring “Consistency”, “Fault tolerance”, and “Order”.
- The outputs of the query are “transactionally” written to sink (File sinks, in-memory sinks, databases)

Use cases: ETL pipelines, Online machine learning


(phone1, 2:01, open)
(phone1, 2:03, close)
...
(phone2, 2:05, open)
(phone4, 2:03, close)
(2:00, open)
(2:00, close)
open 1:00 5
open 2:00 1
close 1:00 5
close 2:00 2
MySQL
Output Sink
Mappers
split by (action, hour)
Reducers
count by (action, hour)

Setting up cluster

DIY

- Procure multiple machines
- Add master and slave DNS lookup entries in `/etc/hosts` file
- Configure ssh to slaves on master node
- Install java and Spark binary
- Add master and slave node configurations (IPs, Java home, etc)
- Start cluster from master
- Use Spark Master UI and Spark Application UI

Source:
https://medium.com/ymedialabs-innovation/apache-spark-on-a-multi-node-cluster-b75967c8cb2b

Submitting jobs through spark-submit

- CLI utility that allows you to submit applications to a spark cluster to run.
- You can either initiate execution, request status of Spark applications, or kill them.

Detailed explanation with examples:
https://spark.apache.org/docs/latest/submitting-applications.html

Setting up Spark cluster on cloud

Cloud options for Spark

Platform-as-a-service options

- Amazon Web Services Elastic MapReduce
- Azure HDinsight
- Google Dataproc

Recent benchmarking shows similar performance. Features to look for,

- Easy deployments
- On demand Elasticity
- Decoupled compute and storage
- Pay per go pricing

Common deployment patterns

- Train offline -> Store results in db -> Use them. (Recommender systems)
- Train offline -> persist model to disk -> Load into spark application and serve. (High latency, load balancing to be handled manually)
- Train online -> Use it online (Structured streaming)

Alternatives: Apache Mahout, Apache Flink, Dask

Beyond 7

10 - Is world's leading expert on the idea.

9 - Can ask expert questions and generate new information/data on the idea.

8 - Can answer expert questions and reconcile contradictory thoughts about the idea.

7 - Can answer any layman's question and forms independent thoughts on the idea.

Spark

6 - Can answer any layman's question and forms intelligent opinions on the idea.

5 - Knows about the idea, and can discern inaccurate statements about the idea.

4 - Knows about the idea, and can explain what's been learned in one's own words.

3 - Heard of the idea, and recites what others have said about it.

2 - Heard of the idea, but doesn't know anything about it.

Spark

1 - Never heard of the idea.

Getting beyond 7

- https://pages.databricks.com/definitive-guide-spark.html (Spark: The Definitive Guide by Databricks). PS: Image courtesy.
- Know the standard library well!
- https://jaceklaskowski.gitbooks.io/mastering-apache-spark (The Internals of Apache Spark gitbook by Jacek Laskowski)
- Spark-summit talks
- Lastly, just do stuff with Spark with lots of data!
- Community mailing list for questions, contributions, feature requests and spark help in general.

Thank you!

Doubts, queries, feedbacks, opportunities? Feel free to reach out ...

Handle: @satwikkansal

https://satwikkansal.xyz