How Spark Works Internally:
RDD and DAG

Dano Lee

RDD

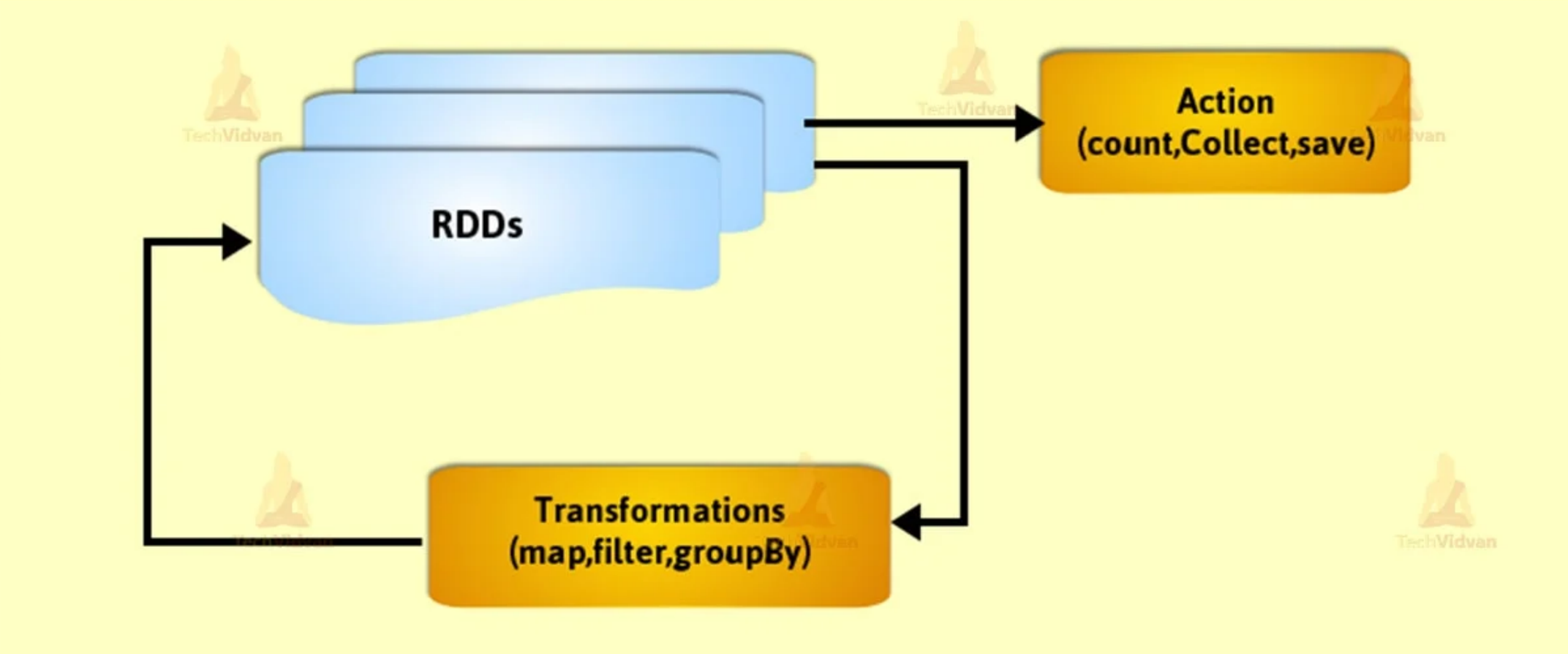
RDD

- **Resilient Distributed Dataset**
  - It is the **fundamental unit of data** in Apache Spark. RDDs are distributed a collection of elements across cluster nodes.
  - One of the Important parameters is **RDD supports parallel operations**.
  - Spark RDDs are **immutable in nature**. We can not make any changes though it can generate by transforming existing RDD.

Creating RDD

- We can create RDD in Spark by several ways.
  1. Parallelized collections
  2. External datasets
  3. Existing RDDs

Transformation vs Action



source: techVidvan

Lazy Evaluation

- Lazy evaluation in Spark means that **transformations** on RDDs are not executed immediately.
- Instead, Spark keeps track of the operations applied to each RDD (its **lineage**) and waits until an **action** is called to execute them.
- This allows Spark to **optimize the execution plan and minimize unnecessary computations**.

Transformation vs Action

- Transformations
  - Transformations **produce new RDDs from the data in existing ones** using operations
  - operations that are lazily evaluated and return a new RDD, DataFrame, or Dataset.
  - Examples include **map, filter, flatMap, groupBy**, etc.
  - Transformations are not executed immediately; they are only executed when an action is called.
- Actions
  - An action **does not create a new RDD**, but returns the result of a computation on the data set
  - operations that **trigger the evaluation of transformations** and return a result to the driver program or write data to external storage.
  - Examples include **collect, count, saveAsTextFile, reduce,** etc.

Narrow & Wide Transformations

- **Narrow Transformations**
  - transformations where each input partition **contributes to only one output partition**.
  - Examples include **map, filter, and flatMap**.
  - Narrow transformations can be **computed in parallel without shuffling data across partitions.**
- **Wide Transformations**
  - transformations where each input partition may **contribute to multiple output partitions**.
  - Examples include **groupBy, reduceByKey, and join**.
  - Wide transformations require **shuffling data across partitions and may involve data movement between nodes.**

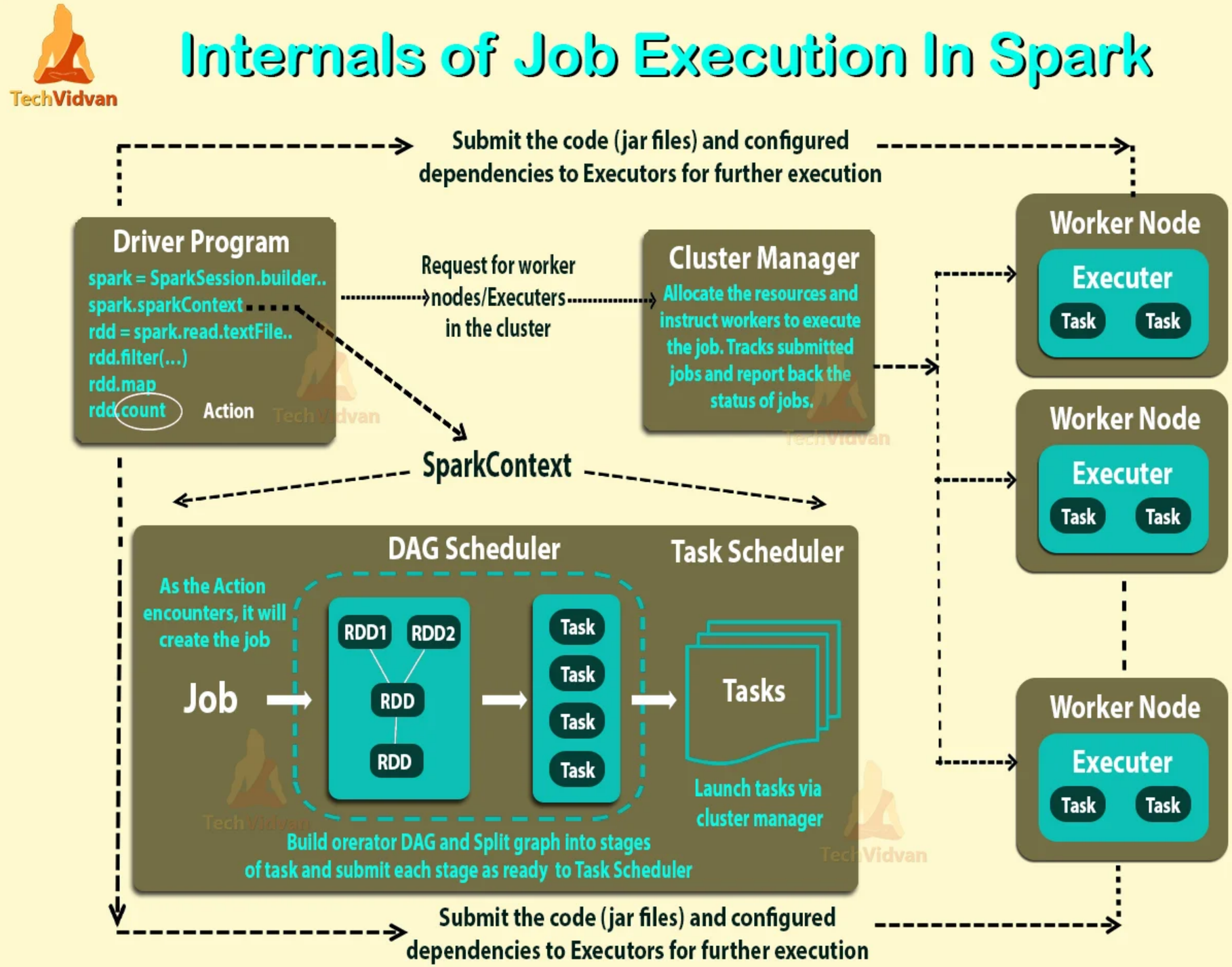
Partition

- A partition in Spark is a **smaller, logical division of data that allows computations to be performed in parallel**.
- Each partition is **processed by a single task in a single executor**.
- Spark **automatically determines the number of partitions** based on the size of the data and the available resources.

Shuffle in Spark

- Shuffling is **the process of distributing data across the cluster workers** in order to process it in parallel.
- It happens generally
  - when data is not evenly distributed, (data skewing)
  - when data should be arranged in a specific way to be processed
    - **when data needs to be aggregated or joined across partitions**, requiring data to be moved between nodes in the cluster.
  - when there is not enough memory on a single node to store all the required data for processing.
- Exchange in DAG represents Shuffling, i.e., physical data movement on the cluster among several nodes.

How Spark Works Internally


TechVidvan
Internals of Job Execution In Spark
Submit the code (jar files) and configured dependencies to Executors for further execution
Driver Program
spark = SparkSession.builder..
spark.sparkContext
rdd = spark.read.textFile..
rdd.filter(...)
rdd.map
rdd.count
Action
Request for worker nodes/Executers in the cluster
Cluster Manager
Allocate the resources and instruct workers to execute the job. Tracks submitted jobs and report back the status of jobs.
Worker Node
Executer
Task
Task
Worker Node
Executer
Task
Task
Worker Node
Executer
Task
Task
SparkContext
DAG Scheduler
Task Scheduler
As the Action encounters, it will create the job
Job
RDD1
RDD2
RDD
RDD
Task
Task
Task
Task
Tasks
Launch tasks via cluster manager
Build orerator DAG and Split graph into stages of task and submit each stage as ready to Task Scheduler
Submit the code (jar files) and configured dependencies to Executors for further execution

source

Executing Spark Jobs (1)

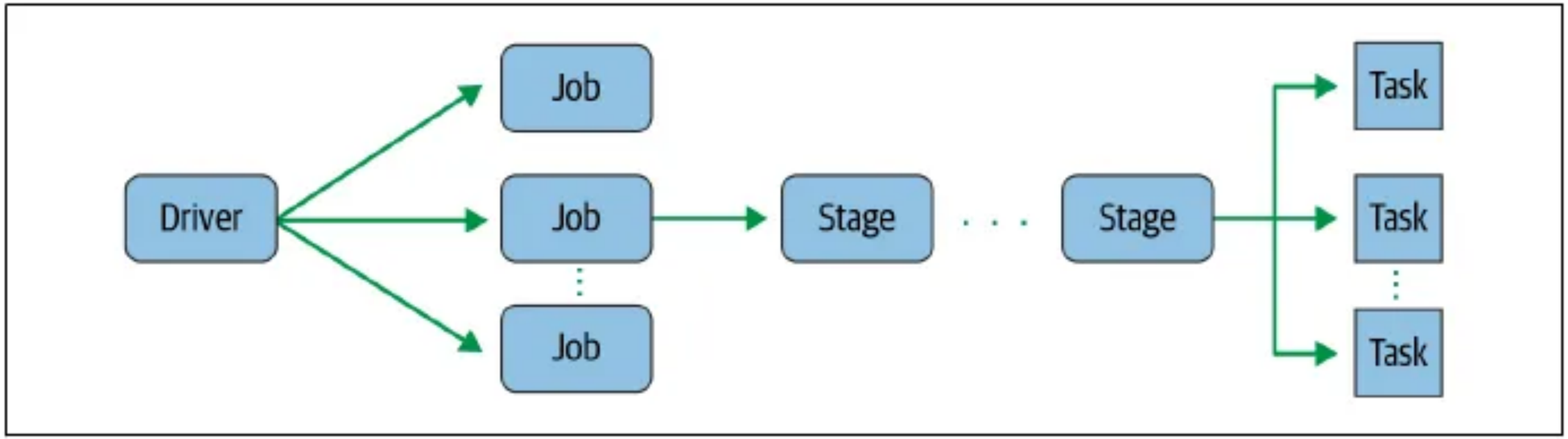
- Once you do a Spark submit, a driver program is launched and this requests for resources to the cluster manager and at the same time the main program of the user function of the user processing program is initiated by the driver program.
- Based on that, the execution logic is processed and parallelly Spark context is also created. Using the Spark context, the different transformations and actions are processed. So, **till the time the action is not encountered, all the transformations will go into the Spark context in the form of DAG that will create RDD lineage.**
- **Once the action is called job is created**. Job is the collection of different task stages. **Once these tasks are created, they are launched by the cluster manager on the worker nodes and this is done with the help of a class called task scheduler.**

source

Executing Spark Jobs (2)

- **The conversion of RDD lineage into tasks is done by the DAG scheduler**. Here DAG is created based on the different transformations in the program and once the action is called these are split into different stages of tasks and submitted to the task scheduler as tasks become ready.
- Then these are launched on the different executors in the worker node through the cluster manager. The entire resource allocation and the tracking of the jobs and tasks are performed by the cluster manager.
- As soon as you do a Spark submit, **your user program and other configuration mentioned are copied onto all the available nodes in the cluster**. So that the program becomes **the local read on all the worker nodes**. Hence, the parallel executors running on the different worker nodes do not have to do any kind of network routing.

source



source

Job

- A parallel computation consisting of multiple tasks that **gets spawned in response to a Spark action** (e.g., save(), collect()).
- During interactive sessions with Spark shells, the driver converts your Spark application into one or more Spark jobs.
- It then transforms each job into a DAG. This, in essence, is Spark's execution plan, where each node within a DAG could be a single or multiple Spark stages.
- So a spark job is **created whenever an action is called.**

Stage and Task

- **Stage** — Each job gets divided into **smaller sets of tasks called stages that depend on each other**. As part of the DAG nodes, stages are created **based on what operations can be performed serially or in parallel**. Not all Spark operations can happen in a single stage, so they may be divided into multiple stages. Often stages are delineated on the operator's computation boundaries, where they dictate data transfer among Spark executors.
- **Task** — A single unit of work or execution that will be sent to a Spark executor. Each stage is comprised of Spark tasks (a unit of execution), which are fed across each Spark executor, **each task maps to a single partition of data.**

Tasks and Stages in Spark (1)

- Task and Stage Formation:
  - Tasks: Tasks in Spark are units of work that are executed on partitions of data.
  - Stages: Stages are groups of tasks that can be executed together, **typically as a result of a narrow transformation (like map or filter) that does not require shuffling data across the cluster**.
- **Boundary of Stages:**
  - A stage boundary is **typically defined by operations that require data to be shuffled across the cluster**.
  - Examples include operations like **reduceByKey, groupByKey, join, or any operation that causes a data exchange (shuffle) between different partitions of data.**
  - When such an operation is encountered in a Spark job, it marks the end of the current stage and the beginning of a new stage.

Tasks and Stages in Spark (2)

- Data Transfer During Stage Execution:
  - Within a stage, tasks operate on **data that is already partitioned across the cluster.** Tasks within the same stage can **operate independently on their respective partitions without needing to exchange data**.
  - When a stage completes and a shuffle boundary is encountered, **Spark reorganizes data across the cluster according to the requirements of the subsequent stage**.
  - Data transfer among Spark executors primarily occurs during these shuffle operations. **Executors exchange data partitions to ensure that each executor has the data it needs for the next stage of computation**.

Tasks and Stages in Spark (3)

- Impact on Performance:
  - Efficient stage management is crucial for Spark job performance. **Minimizing the number of stages and optimizing data partitioning** can reduce the overhead associated with data shuffling and improve overall execution time.
  - Understanding stage boundaries helps in optimizing Spark jobs by reducing unnecessary data movements and improving resource utilization.
- Example Scenario:
  - Consider a simple Spark job that reads data, performs a map operation (forming one stage), then performs a reduceByKey operation (forming another stage due to shuffle). Between these stages, data will be shuffled across the cluster based on the keys used in the reduceByKey operation.

```
from pyspark.sql import SparkSession

sc = SparkContext(conf=conf)

rdd = sc.textFile("path_to_your_data.csv")

rdd_filtered = rdd.filter(lambda x: x[2] > 0)

rdd = rdd.map(parse_row).filter(lambda x: x is not None)

total_trips = rdd_filtered.count()
```

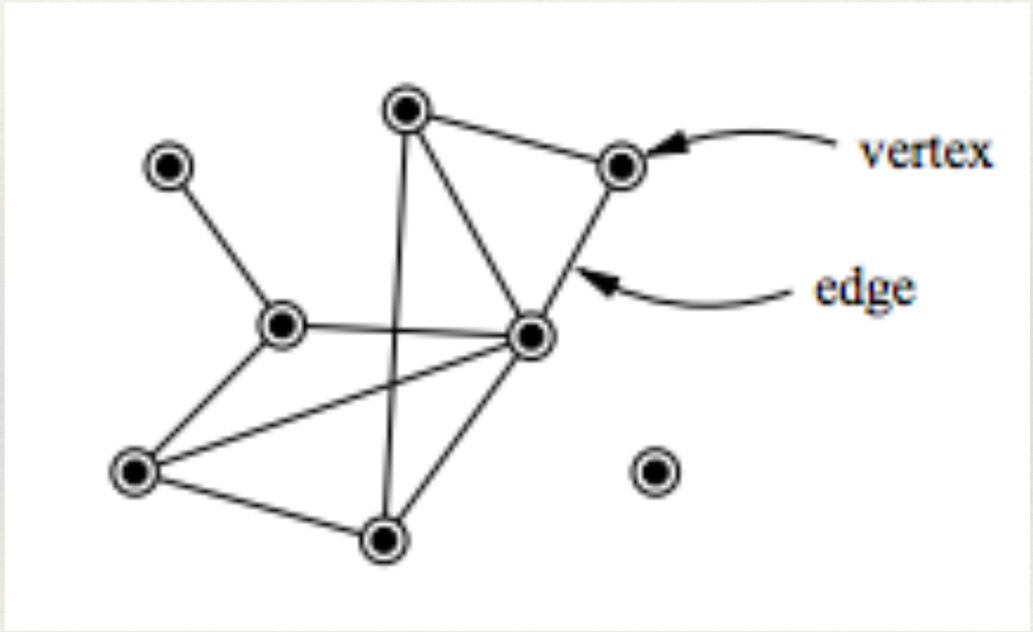
```
from pyspark.sql import SparkSession

sc = SparkContext(conf=conf)

rdd = sc.textFile("path_to_your_data.csv")

rdd_filtered = rdd.filter(lambda x: x[2] > 0)

rdd = rdd.map(parse_row).filter(lambda x: x is not None)

total_trips = rdd_filtered.count()

all_records_count = rdd.count()
```

DAG

- **Directed Acyclic Graph**
  - Directed – here means that each edge has a defined direction, that represents **a unidirectional flow from one vertex to another**. This creates a sequence i.e. each node is in linkage from earlier to later in the appropriate sequence.
  - Acyclic – here means that **there are no loops or cycles in the graph**, so that for any given vertex, if we follow an edge that connects that vertex to another, there is no path in the graph to get back to that initial vertex. Once a transformation takes place it cannot returns to its earlier position.
  - Graph – From graph theory, it is **a combination of vertices and edges**. Those pattern of connections together in a sequence is the graph.

Graph - Vertices and Edges

- The order of the activities is represented by a graph, which is visually presented as a set of circles, each **circle represent an activity, connected by lines, which represent the execution flow from one activity to another**. Circles are known as a **vertices** and connecting lines are known as **edges**.



source

RDD vs DataFrames

RDD vs DataFrames

- While an RDD is the basic data structure of Spark, it is a lower-level API that requires a more verbose syntax and lacks the optimizations provided by higher-level data structures.
- Spark shifted toward a more user-friendly and optimized API with the introduction of DataFrames—higher-level abstractions built on top of RDDs. The data in a DataFrame is organized into named columns, structuring it more like the data in a relational database.
- DataFrame operations also benefit from **Catalyst**, Spark SQL's optimized execution engine, which can increase computational efficiency, potentially improving performance.
- Transformations and actions can be run on DataFrames the way they are in RDDs.

RDD vs DataFrames

- Because of their higher-level API and optimizations, DataFrames are typically easier to use and offer better performance; however, due to their lower-level nature,
- RDDs can still be useful for defining custom operations, as well as debugging complex data processing tasks. RDDs offer more granular control over partitioning and memory usage.
- When dealing with raw, unstructured data, such as text streams, binary files, or custom formats, RDDs can be more flexible, allowing for custom parsing and manipulation in the absence of a predefined structure.

PROJECT 주제

- 현대 자동차에서 신규 모델이 출시되었을 때 SNS (유튜브, 인스타그램, 커뮤니티 등)에서 해당 모델에 대한 소비자들의 반응을 모니터링해서 특정 이슈에 대한 대화가 얼마나 이루어지고 있는지를 모니터링하고 알림을 제공하는 Data Product

Honor Code

- 어떤 경우에도 동료를 존중하고, 함부로 대하지 않는다.
- 혼자가 아닌 함께 성장하려고 노력한다.
- 지식을 적극적으로 나누고, 서로를 돕는다.

PROJECT 진행 방법

- 같은 주제로 시작하지만 해석은 다 다를 수 있겠죠?
- 매주 전체 리뷰해서 배울 것들 배우고 나눌 것들 나누고 계속 이어서 진행
  - Data Insight를 얻으면 같은 주제로 따로 만들 수도
  - 모두 필요한 기능인 경우 공통 모듈을 만들 수도
  - 합치면 강력해지면 나눠서 작업하고 합칠 수도

AWS 사용

- 팀당 AWS 계정 1개 제공
  - AWS IAM (Identity and Access Management) 설정
- 인스턴스 셋업은 Dano의 승인 하에만 진행
  - 아키텍쳐 안 통과되면 Dano에게 리뷰 요청 => 승인되면 작업

W4M1 & W4M2

by Sunday 24:00

Spark Standalone Mode

- To install Spark Standalone mode, you simply place a compiled version of Spark on each node on the cluster. You can obtain pre-built versions of Spark with each release or build it yourself.

Running Spark on YARN

- Running Spark on YARN requires a binary distribution of Spark which is built with YARN support.
- YARN handles the distribution of Spark binaries to worker nodes when it allocates containers for executors. This means the essential Spark libraries and dependencies are shipped with the job, making a local installation technically unnecessary for the job execution.
- 2 distribution types
  - One is pre-built with a certain version of Apache Hadoop; this Spark distribution contains built-in Hadoop runtime, so we call it with-hadoop Spark distribution.
  - The other one is pre-built with user-provided Hadoop; since this Spark distribution doesn't contain a built-in Hadoop runtime, it's smaller, but users have to provide a Hadoop installation separately. We call this variant no-hadoop Spark distribution.

Running Spark on YARN

- For with-hadoop Spark distribution, since it contains a built-in Hadoop runtime already, by default, when a job is submitted to Hadoop Yarn cluster, to prevent jar conflict, it will not populate Yarn's classpath into Spark. To override this behavior, you can set spark.yarn.populateHadoopClasspath=true.
- For no-hadoop Spark distribution, Spark will populate Yarn's classpath by default in order to get Hadoop runtime.
- For with-hadoop Spark distribution, if your application depends on certain library that is only available in the cluster, you can try to populate the Yarn classpath by setting the property mentioned above. If you run into jar conflict issue by doing so, you will need to turn it off and include this library in your application jar.