Chapter 4
SPARK

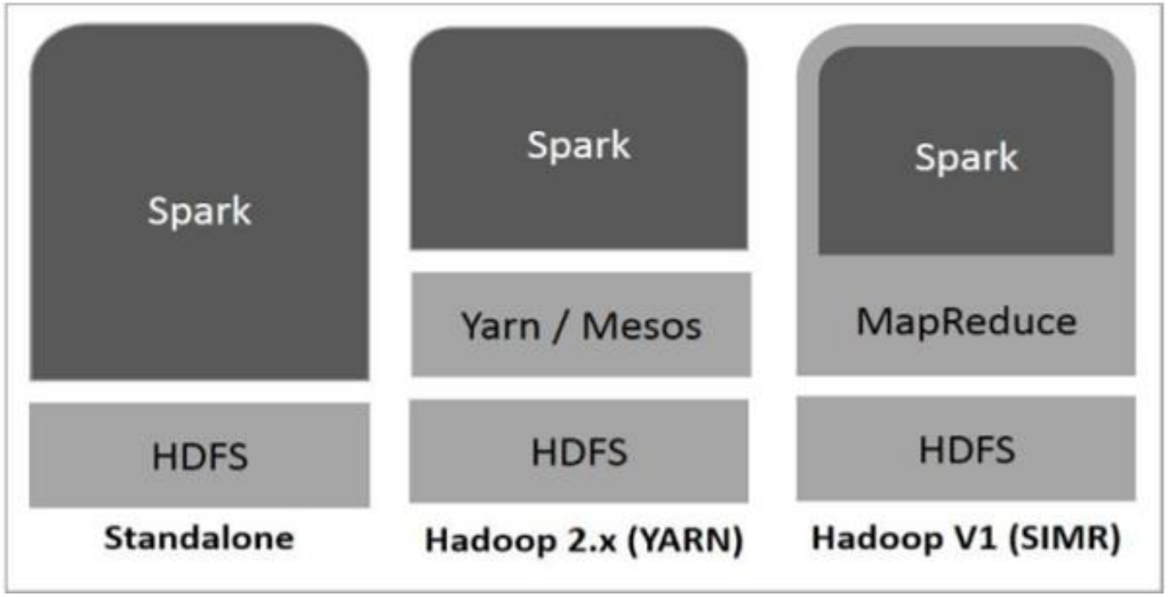
Dr. Nilesh M. Patil
Associate Professor
Computer Engineering
SVKM’s DJSCE, Mumbai

Features of Spark

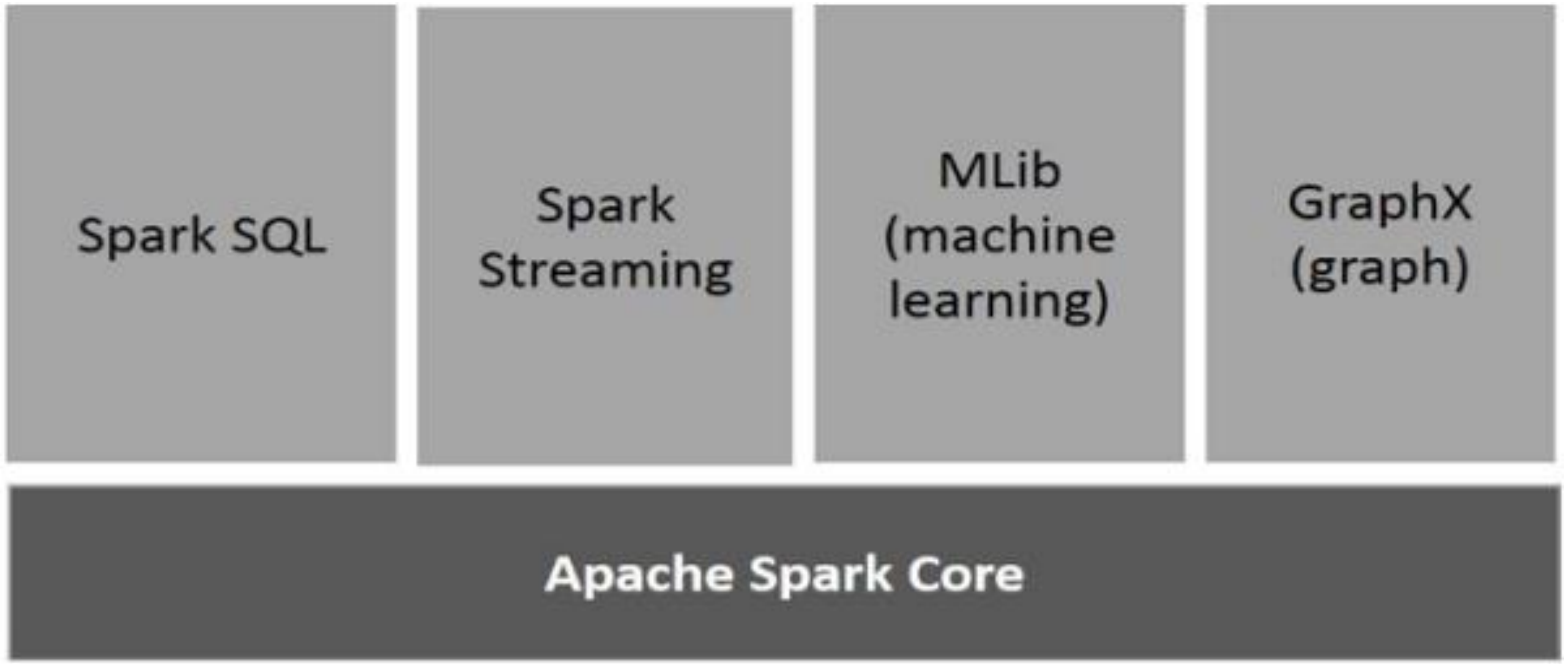
- Apache Spark is an open-source, distributed processing system used for big data workloads.
- It utilizes in-memory caching, and optimized query execution for fast analytic queries against data of any size.
- Spark helps to run an application in Hadoop cluster, up to 100 times faster in memory, and 10 times faster when running on disk. This is possible by reducing the number of read/write operations to disk.
- Spark not only supports ‘Map’ and ‘Reduce’.
- It provides development APIs in Java, Scala, Python and R, and supports code reuse across multiple workloads like batch processing, interactive queries, real-time analytics, machine learning, and graph processing.

Spark Built on Hadoop

- The following diagram shows three ways of how Spark can be built with Hadoop components.
- **Standalone** − Spark Standalone deployment means Spark occupies the place on top of HDFS (Hadoop Distributed File System) and space is allocated for HDFS, explicitly. Here, Spark and MapReduce will run side by side to cover all spark jobs on cluster.
- **Hadoop Yarn** − Hadoop Yarn deployment means, simply, spark runs on Yarn without any pre-installation or root access required. It helps to integrate Spark into Hadoop ecosystem or Hadoop stack. It allows other components to run on top of stack.
- **Apache Mesos** : Apache Mesos (Mesosphere's Cluster Operating System) is a cluster management technology that can be used with Apache Spark to manage resources in a distributed environment. Mesos provides a layer of abstraction between the cluster's resources and the distributed applications running on top of it, enabling multiple frameworks to run concurrently and share the same set of cluster resources.
- **Spark in MapReduce (SIMR)** − Spark in MapReduce is used to launch spark job in addition to standalone deployment. With SIMR, user can start Spark and uses its shell without any administrative access.



Components of Spark (1/6)



Components of Spark (2/6)

Spark Core

- Spark Core is the foundation of the platform.
- It is responsible for memory management, fault recovery, scheduling, distributing & monitoring jobs, and interacting with storage systems.
- Spark Core is exposed through an application programming interface (APIs) built for Java, Scala, Python and R.
- These APIs hide the complexity of distributed processing behind simple, high-level operators.

Components of Spark (3/6)

MLlib (Machine Learning)

- Spark includes MLlib, a library of algorithms to do machine learning on data at scale.
- Machine Learning models can be trained by data scientists with R or Python on any Hadoop data source, saved using MLlib, and imported into a Java or Scala-based pipeline.
- Spark was designed for fast, interactive computation that runs in memory, enabling machine learning to run quickly.
- The algorithms include the ability to do classification, regression, clustering, collaborative filtering, and pattern mining.

Components of Spark (4/6)

Spark Streaming (Real-time)

- Spark Streaming is a real-time solution that leverages Spark Core’s fast scheduling capability to do streaming analytics.
- It ingests data in mini-batches, and enables analytics on that data with the same application code written for batch analytics.
- This improves developer productivity, because they can use the same code for batch processing, and for real-time streaming applications.
- Spark Streaming supports data from Twitter, Kafka, Flume, HDFS, and ZeroMQ, and many others found from the Spark Packages ecosystem.

Components of Spark (5/6)

Spark SQL (Interactive Queries)

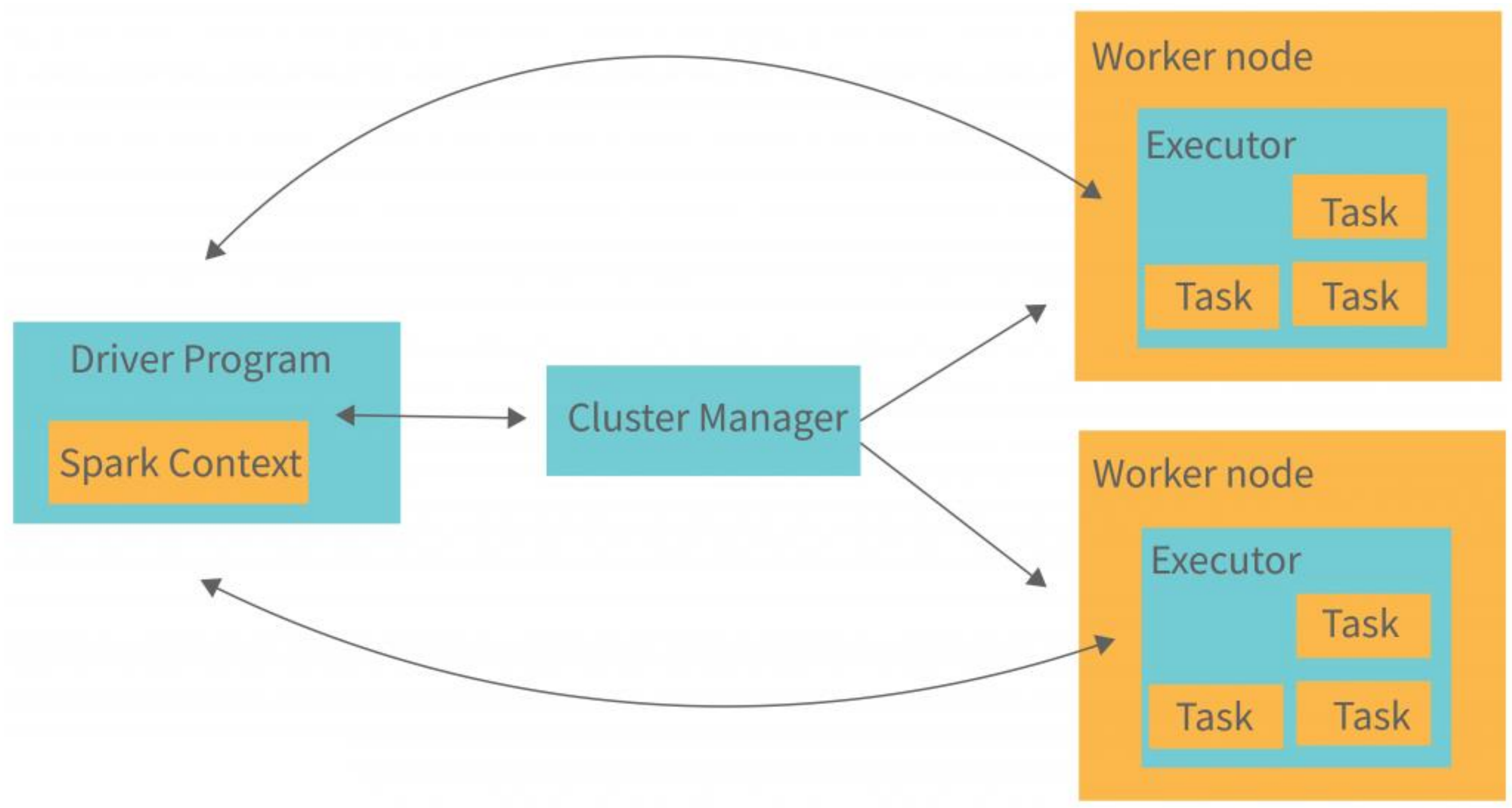
- Spark SQL is a distributed query engine that provides low-latency, interactive queries up to 100x faster than MapReduce.
- It includes a cost-based optimizer, columnar storage, and code generation for fast queries, while scaling to thousands of nodes.
- Business analysts can use standard SQL or the Hive Query Language for querying data.
- Developers can use APIs, available in Scala, Java, Python, and R.
- It supports various data sources out-of-the-box including JDBC, ODBC, JSON, HDFS, Hive, ORC, and Parquet.
- Other popular stores—Amazon Redshift, Amazon S3, Couchbase, Cassandra, MongoDB, Salesforce.com, Elasticsearch, and many others can be found from the Spark Packages ecosystem.

Components of Spark (6/6)

GraphX (Graph Processing)

- Spark GraphX is a distributed graph processing framework built on top of Spark.
- GraphX provides ETL, exploratory analysis, and iterative graph computation to enable users to interactively build, and transform a graph data structure at scale.
- It comes with a highly flexible API, and a selection of distributed Graph algorithms.

Architecture of Spark (1/2)



Architecture of Spark (2/2)

- When the **Driver Program** in the Apache Spark architecture executes, it calls the real program of an application and creates a SparkContext. SparkContext contains all of the basic functions. The Spark Driver includes several other components, including a DAG Scheduler, Task Scheduler, Backend Scheduler, and Block Manager, all of which are responsible for translating user-written code into jobs that are actually executed on the cluster.
- The **Cluster Manager** manages the execution of various jobs in the cluster. Spark Driver works in conjunction with the Cluster Manager to control the execution of various other jobs. The cluster Manager does the task of allocating resources for the job. Once the job has been broken down into smaller jobs, which are then distributed to worker nodes, SparkDriver will control the execution. Many **worker nodes** can be used to process an RDD created in the SparkContext, and the results can also be cached.
- The **Spark Context** receives task information from the Cluster Manager and enqueues it on worker nodes.
- The **Executor** is in charge of carrying out these duties. The lifespan of executors is the same as that of the Spark Application. We can increase the number of workers if we want to improve the performance of the system. In this way, we can divide jobs into more coherent parts.

Apache Hadoop Vs Apache Spark

Category for Comparison	**Hadoop**	**Spark**
Performance	Slower performance, uses disks for storage and depends on disk read and write speed.	Fast in-memory performance with reduced disk reading and writing operations.
Cost	An open-source platform, less expensive to run. Uses affordable consumer hardware. Easier to find trained Hadoop professionals.	An open-source platform, but relies on memory for computation, which considerably increases running costs.
Data Processing	Best for batch processing. Uses MapReduce to split a large dataset across a cluster for parallel analysis.	Suitable for iterative and live-stream data analysis. Works with RDDs and DAGs to run operations.
Fault Tolerance	A highly fault-tolerant system. Replicates the data across the nodes and uses them in case of an issue.	Tracks RDD block creation process, and then it can rebuild a dataset when a partition fails. Spark can also use a DAG to rebuild data across nodes.
Scalability	Easily scalable by adding nodes and disks for storage. Supports tens of thousands of nodes without a known limit.	A bit more challenging to scale because it relies on RAM for computations. Supports thousands of nodes in a cluster.
Security	Extremely secure. Supports LDAP, ACLs, Kerberos, SLAs, etc.	Not secure. By default, the security is turned off. Relies on integration with Hadoop to achieve the necessary security level.
Ease of Use and Language Support	More difficult to use with less supported languages. Uses Java or Python for MapReduce apps.	More user friendly. Allows interactive shell mode. APIs can be written in Java, Scala, R, Python, Spark SQL.
Machine Learning	Slower than Spark. Data fragments can be too large and create bottlenecks. Mahout is the main library.	Much faster with in-memory processing. Uses MLlib for computations.
Scheduling and Resource Management	Uses external solutions. YARN is the most common option for resource management. Oozie is available for workflow scheduling.	Has built-in tools for resource allocation, scheduling, and monitoring.

RDD in Spark

- RDD is a core abstraction in Spark, which stands for **Resilient Distributed Dataset**.
- It enables partition of large data into smaller data that fits each machine, so that computation can be done parallelly on multiple machines.
- Moreover, RDDs automatically recover from node failures to ensure the storage resilience.
- It is an immutable fault-tolerant, distributed collection of objects that can be operated on in parallel.
- An RDD can contain any type of object and is created by loading an external dataset or distributing a collection from the driver program.
- RDDs support two types of operations:
- **Transformations** are operations (such as map, filter, join, union, and so on) that are performed on an RDD and which yield a new RDD containing the result.
- **Actions** are operations (such as reduce, count, first, and so on) that return a value after running a computation on an RDD.

Features of RDD

Here are some features of RDD in Spark:

- **Resilience**: RDDs track data lineage information to recover lost data, automatically on failure. It is also called fault tolerance.
- **Distributed**: Data present in an RDD resides on multiple nodes. It is distributed across different nodes of a cluster.
- **Lazy evaluation**: Data does not get loaded in an RDD even if you define it. Transformations are actually computed when you call action, such as count or collect, or save the output to a file system.
- **Immutability**: Data stored in an RDD is in the read-only mode—you cannot edit the data which is present in the RDD. But, you can create new RDDs by performing transformations on the existing RDDs.
- **In-memory computation**: An RDD stores any immediate data that is generated in the memory (RAM) than on the disk so that it provides faster access.
- **Partitioning**: Partitions can be done on any existing RDD to create logical parts that are mutable. You can achieve this by applying transformations to the existing partitions.

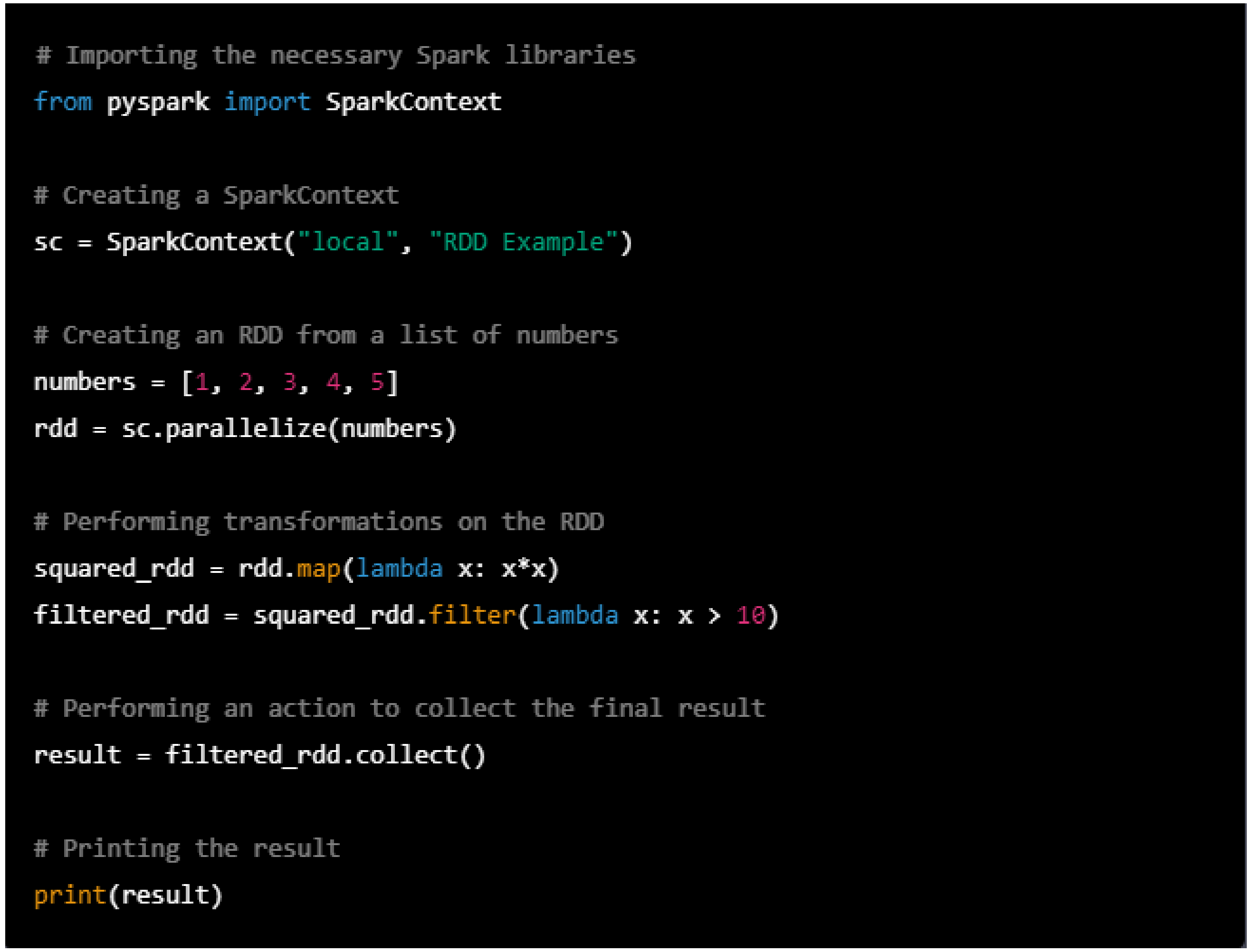
Advantages of RDD

- RDD aids in increasing the execution speed of Spark.
- RDDs are the basic unit of parallelism and hence help in achieving the consistency of data.
- RDDs help in performing and saving the actions separately
- They are persistent as they can be used repeatedly.

Limitation of RDD

- There is no input optimization available in RDDs
- One of the biggest limitations of RDDs is that the execution process does not start instantly.
- No changes can be made in RDD once it is created.
- RDD lacks enough storage memory.
- The run-time type safety is absent in RDDs.

RDD Example



```
# Importing the necessary Spark libraries
from pyspark import SparkContext

# Creating a SparkContext
sc = SparkContext("local", "RDD Example")

# Creating an RDD from a list of numbers
numbers = [1, 2, 3, 4, 5]
rdd = sc.parallelize(numbers)

# Performing transformations on the RDD
squared_rdd = rdd.map(lambda x: x*x)
filtered_rdd = squared_rdd.filter(lambda x: x > 10)

# Performing an action to collect the final result
result = filtered_rdd.collect()

# Printing the result
print(result)
```

- In this example, we first create a SparkContext with the name "RDD Example".
- Then, we create an RDD from a list of numbers using the parallelize() method.
- We then perform two transformations on the RDD: map() to square each element, and filter() to keep only the elements that are greater than 10.
- Finally, we perform an action collect() to retrieve the final result of the computation, which is a list of numbers [16, 25]. The result is printed to the console.
- Note that RDDs are lazy-evaluated, meaning that the transformations are not executed until an action is performed. In this example, collect() is the action that triggers the computation of the transformations.

Spark SQL

- Spark SQL is a Spark component that supports querying data either via SQL or via the Hive Query Language.
- It originated as the Apache Hive port to run on top of Spark (in place of MapReduce) and is now integrated with the Spark stack.
- Spark SQL introduces SchemaRDD, a new data abstraction that provides support for structured and semi-structured data.

Features of Spark SQL

- **Easy to Integrate**: One can mix SQL queries with Spark programs easily. Structured data can be queried inside Spark programs using either SQL or a Dataframe API. Running SQL queries alongside analytic algorithms is easy because of this tight integration.
- **Compatibility with Hive**: Hive queries can be executed in Spark SQL as they are.
- **Unified Data Access**: Loading and querying data from various sources is possible.
- **Standard Connectivity**: Spark SQL can connect to Java and Oracle using JDBC (Java Database Connectivity) and ODBC (Oracle Database Connectivity) APIs.
- **Performance and Scalability**: To make queries agile, alongside computing hundreds of nodes using the Spark engine, Spark SQL incorporates a code generator, a cost-based optimizer, and columnar storage. This provides complete mid-query fault tolerance.

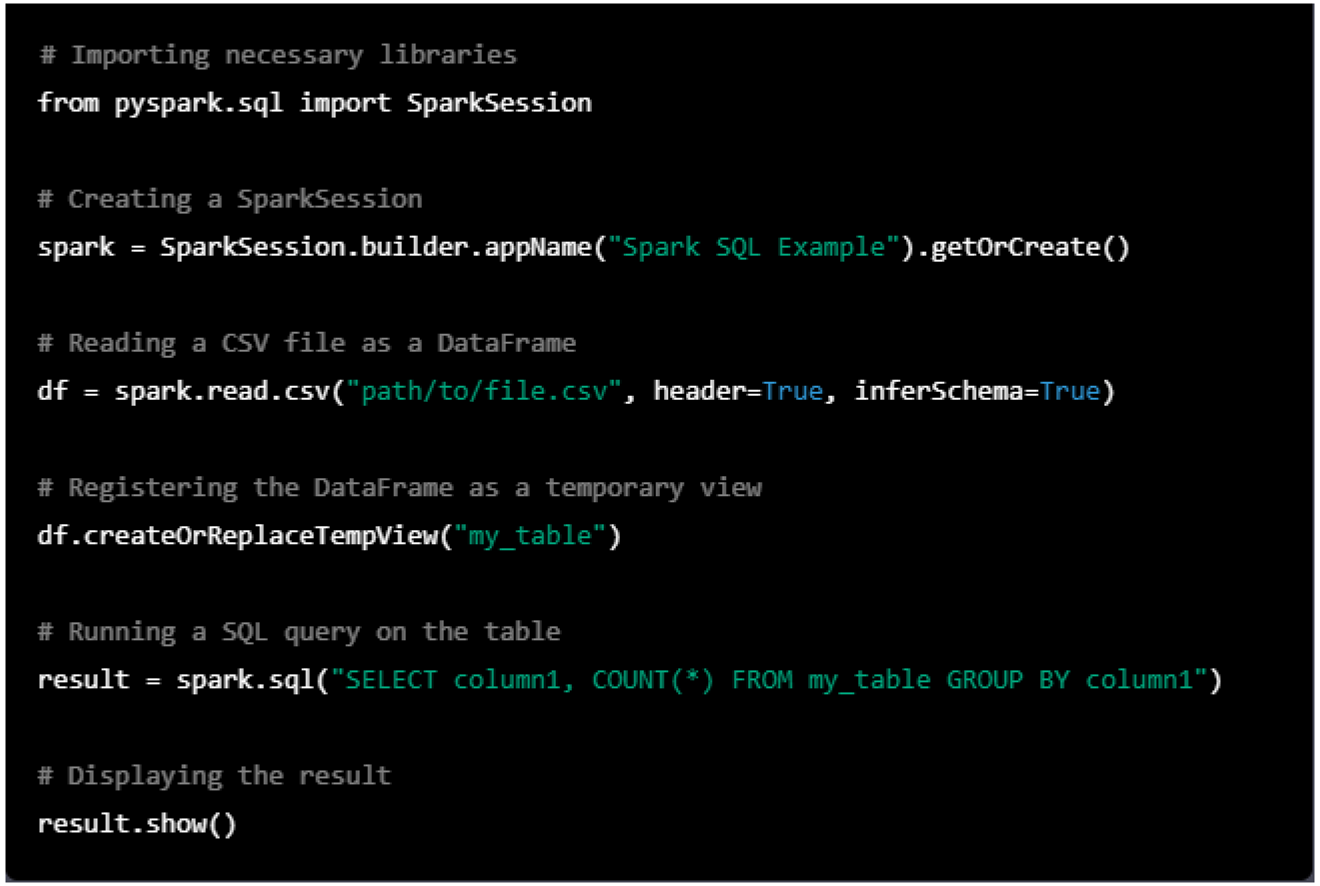
Advantages of Spark SQL

- It helps in easy data querying. The SQL queries are mixed with Spark programs for querying structured data as a distributed dataset (RDD). Also, the SQL queries are run with analytic algorithms using Spark SQL’s integration property.
- Another important advantage of Spark SQL is that the loading and querying can be done for data from different sources. Hence, the data access is unified.
- It offers standard connectivity as Spark SQL can be connected through JDBC or ODBC.
- It can be used for faster processing of Hive tables.
- Another important offering of Spark SQL is that it can run unmodified Hive queries on existing warehouses as it allows easy compatibility with existing Hive data and queries.

Disadvantages of Spark SQL

- Creating or reading tables containing union fields is not possible with Spark SQL.
- It does not convey if there is any error in situations where the varchar is oversized.
- It does not support Hive transactions.
- It also does not support the Char type (fixed-length strings). Hence, reading or creating a table with such fields is not possible.

Spark SQL Example



```
# Importing necessary libraries
from pyspark.sql import SparkSession

# Creating a SparkSession
spark = SparkSession.builder.appName("Spark SQL Example").getOrCreate()

# Reading a CSV file as a DataFrame
df = spark.read.csv("path/to/file.csv", header=True, inferSchema=True)

# Registering the DataFrame as a temporary view
df.createOrReplaceTempView("my_table")

# Running a SQL query on the table
result = spark.sql("SELECT column1, COUNT(*) FROM my_table GROUP BY column1")

# Displaying the result
result.show()
```

- In this example, we first create a SparkSession with the name "Spark SQL Example".
- Then, we read a CSV file as a DataFrame using the read.csv() method.
- We then register the DataFrame as a temporary view using the createOrReplaceTempView() method, which allows us to query it using SQL.
- We then run a SQL query on the table, which counts the number of occurrences of each distinct value in column1.
- The result of the query is stored in a DataFrame called result.
- Finally, we display the result using the show() method.

Schedulers in Spark

- In Spark, the DAG (Directed Acyclic Graph) scheduler and the task scheduler are two important components of the execution engine that work together to execute the user's application code.
- **DAG Scheduler**

1. The DAG scheduler is responsible for generating a DAG of stages based on the user's code and optimizing it for efficient execution.
2. The stages represent a set of tasks that can be executed in parallel.
3. The DAG scheduler divides the computation into smaller stages based on the dependencies between RDDs (Resilient Distributed Datasets) and operators in the code.
4. The DAG scheduler then submits the stages to the task scheduler for execution.

- **Task Scheduler**

1. The task scheduler is responsible for assigning tasks to workers in the cluster.
2. It receives the stages from the DAG scheduler and breaks them down into smaller tasks that can be executed in parallel.
3. The task scheduler then assigns these tasks to workers based on the data locality of the tasks and the availability of resources in the cluster.
4. It also monitors the progress of the tasks and handles any failures that may occur.

- Together, the DAG scheduler and task scheduler ensure that the user's code is executed efficiently and reliably on the Spark cluster.
- They work in tandem to maximize the parallelism of the computation and minimize the data movement across the network.
- This helps to achieve high performance and scalability when processing large amounts of data.

Shared Variables in Spark

- Shared variables are the variables that are required to be used by many functions & methods in parallel.
- Spark provides two special type of shared variables – Broadcast Variables and Accumulators.
- **Broadcast variables**:

1. Used to cache a value in memory on all nodes. Here only one instance of this read-only variable is shared between all computations throughout the cluster.
2. Spark sends the broadcast variable to each node concerned by the related task. After that, each node caches it locally in serialized form.
3. Now before executing each of the planned tasks instead of getting values from the driver system retrieves them locally from the cache.
4. Broadcast variables are: Immutable (Unchangeable), Distributed i.e., broadcasted to the cluster and Fit in memory.

- **Accumulators**:

1. As its name suggests Accumulators main role is to accumulate values. The accumulator is variables that are used to implement counters and sums. Spark provides accumulators of numeric type only.
2. The user can create named or unnamed accumulators.
3. Unlike Broadcast Variables, accumulators are writable. However, written values can be only read in driver program. It is why accumulators work pretty well as data aggregators.