BIG DATA ANALYTICS

Introduction to Spark

Spark: *Outline*

- **What is Apache Spark**
  - History of Spark
  - The Present and Future of Spark
  - Running Spark
- **Introduction to Spark**
  - Spark's Basic Architecture
    - Spark Applications
  - Spark's Language APIs
  - Spark's APIs
  - Starting Spark
  - The Spark Session
  - Data Frames
    - Partitions
  - Transformations
    - Lazy Evaluation
  - Actions
  - Spark UI
  - An End-to-End Example
    - Data Frames and SQL

What Is Apache Spark?

What is Apache Spark?

- Apache Spark is a unified computing engine and a set of libraries for parallel data processing on computer clusters.
- Spark is the most actively developed open-source engine for this task, making it a standard tool for any developer or data scientist interested in big data.
- Spark supports multiple widely used programming languages (Python, Java, Scala, and R), includes libraries for diverse tasks ranging from SQL to streaming and machine learning, and runs anywhere from a laptop to a cluster of thousands of servers. This makes it an easy system to start with and scale-up to big data processing or incredibly large scale.

What Is Apache Spark?

What is Apache Spark?

- Spark has many parallels with MapReduce, in terms of both API and runtime.
- Spark is closely integrated with Hadoop: it can run on YARN and works with Hadoop file formats and storage backends like HDFS.
- Spark is best known for its ability to keep large working datasets in memory between *jobs*.
- This capability allows Spark to outperform the equivalent MapReduce workflow (by an order of magnitude or more in some cases), where datasets are always loaded from disk.

What Is Apache Spark?

What is Apache Spark?

- Two styles of application that benefit greatly from Spark's processing model are iterative algorithms (where a function is applied to a dataset repeatedly until an exit condition is met) and interactive analysis (where a user issues a series of ad hoc exploratory queries on a dataset).
- Even if we don't need in-memory caching, Spark is very attractive for a couple of other reasons: its DAG engine and its user experience. Unlike MapReduce, Spark's DAG engine can process arbitrary pipelines of operators and translate them into a single job for the user.
- Spark's user experience is also second to none, with a rich set of APIs for performing many common data processing tasks, such as joins.
- At the time of writing, Spark provides APIs in three languages: Scala, Java, and Python.

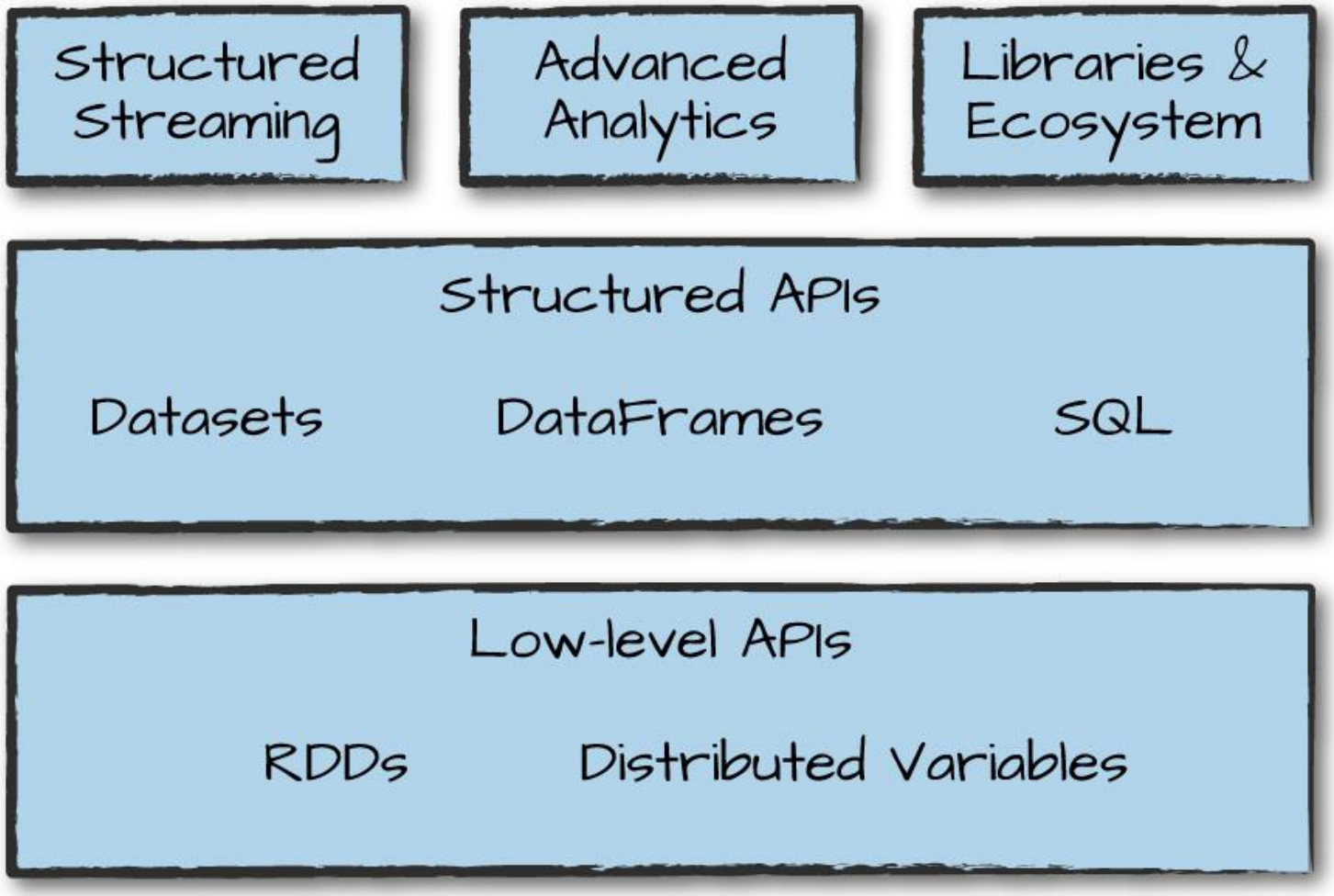
What Is Apache Spark?

What is Apache Spark?

- Spark also comes with a REPL (read-eval-print loop) for both Scala and Python, which makes it quick and easy to explore datasets.
- Spark is proving to be a good platform on which to build analytics tools, too, and the Apache Spark project includes modules for machine learning (MLlib), graph processing (GraphX), stream processing (Spark Streaming), and SQL (Spark SQL) etc.

What Is Apache Spark?

Spark's toolkit: Following figure illustrates all the components and libraries Spark offers to end-users



RDDs - Resilient Distributed Datasets

Figure. Spark's toolkit

What Is Apache Spark?

Key Concepts:

- **Unified Engine:** Spark provides a single platform for batch processing, real-time processing, and more.
- **Libraries:** Includes Spark SQL, Spark Streaming, MLlib for machine learning, and GraphX for graph processing.
- **Supports:** SQL queries, machine learning, real-time analytics, and graph computations.

What Is Apache Spark?

The Spark Ecosystem:

- **Spark Core:** The foundation that provides parallel processing.
- **Spark SQL:** Used for structured data and allows running SQL queries.
- **Spark Streaming:** Enables real-time data processing.
- **MLlib:** Machine learning library built for large-scale learning.
- **GraphX:** Used for graph-based data processing.

Example:

Spark SQL: If a company has a large amount of structured data, such as customer records stored in a database, Spark SQL allows them to query and analyze the data quickly.

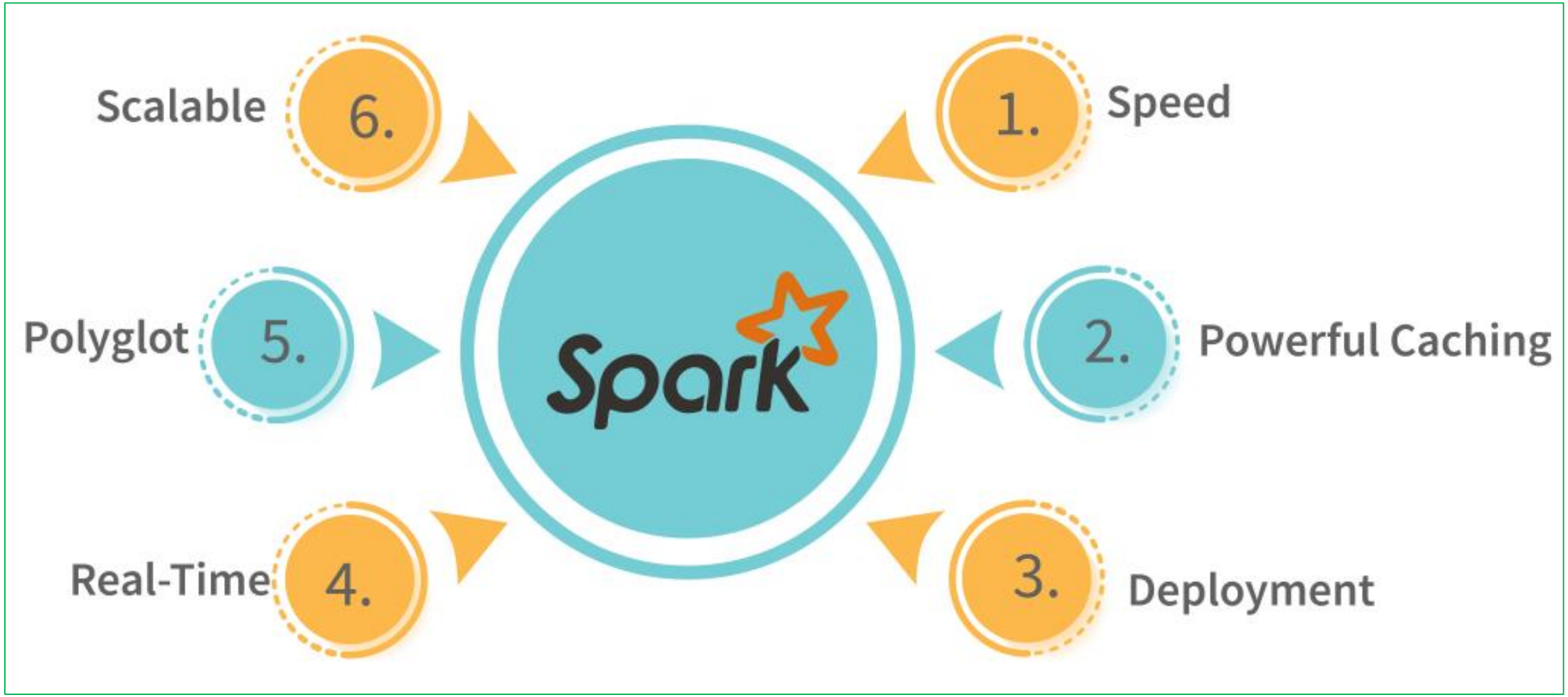
What Is Apache Spark?

Apache Spark Features:

- Apache Spark, a popular cluster computing framework, was created in order to accelerate data processing applications.
- Spark, which enables applications to run faster by utilising in-memory cluster computing, is a popular open-source framework.
- A cluster is a collection of nodes that communicate with each other and share data.
- Because of implicit data parallelism and fault tolerance, Spark may be applied to a wide range of sequential and interactive processing demands.

What Is Apache Spark?

Apache Spark Features:



What Is Apache Spark?

Apache Spark Features:

- **Speed:** Spark performs up to 100 times faster than MapReduce for processing large amounts of data. It is also able to divide the data into chunks in a controlled way.
- **Powerful Caching:** Powerful caching and disk persistence capabilities are offered by a simple programming layer.
- **Deployment:** Mesos, Hadoop via YARN, or Spark's own cluster manager can all be used to deploy it.
- **Real-Time:** Because of its in-memory processing, it offers real-time computation and low latency.
- **Polyglot:** In addition to Java, Scala, Python, and R, Spark also supports all four of these languages. We can write Spark code in any one of these languages. Spark also provides a command-line interface in Scala and Python.

What Is Apache Spark?

Why is Apache Spark Popular?

- **Speed:** In-memory data processing helps achieve faster computations compared to disk-based systems.
- **Ease of Use:** APIs available for different languages make it accessible to a broad range of users.
- **Unified:** It combines batch and real-time processing in a single framework.

Example:

Spark can handle both real-time data streaming from Twitter and perform batch analysis of historical data at the same time, unlike traditional systems that require different tools for each task.

What Is Apache Spark?

Apache Spark Philosophy:

- **Unified Engine:** Handles different workloads (batch, real-time, SQL) under one system.
- **Composability:** Multiple operations can be applied in sequence (chaining).
- **Performance:** Optimized for both storage and computation.

What Is Apache Spark?

Context of Big Data:

Why is Spark Important?

- The growth in data volumes requires distributed systems.
- The era of multi-core processors requires parallelized programming models.

Example:

- A company like Netflix, processing billions of logs every day for user behavior analysis, requires systems like Spark that can handle such large-scale data efficiently and in parallel.

What Is Apache Spark?

A Brief History of Spark:

- Apache Spark started in 2009 at UC Berkeley as a research project led by Matei Zaharia and others at the AMPlab.
- Spark was developed to address the inefficiencies of Hadoop MapReduce, which required separate jobs for each data pass, making tasks like machine learning slow.
- The team built Spark to provide an API based on functional programming, allowing for more efficient, in-memory data sharing across steps in a computation.
- Initially, Spark supported batch processing, but soon expanded to interactive data science and SQL queries through the development of Shark.
- Spark evolved rapidly with the addition of libraries like MLlib, Spark Streaming, and GraphX, making it a versatile platform for big data applications.
- In 2013, Spark became part of the Apache Software Foundation, and Databricks was founded to further develop the project.
- Key milestones include Spark 1.0 in 2014 and Spark 2.0 in 2016, with continued development of powerful APIs like Spark SQL, DataFrames, and Structured Streaming.

What Is Apache Spark?

A Brief History of Spark:

Example:

- Spark was initially designed to support machine learning algorithms like clustering and recommendation systems, which need iterative processing - something that MapReduce couldn't efficiently handle.

What Is Apache Spark?

The Present and Future of Spark:

- Spark continues to grow in popularity and use cases.
- New technologies like Structured Streaming (2016) are pushing the boundaries of data processing.
- Widely used by companies like Uber and Netflix for streaming and machine learning.
- NASA, CERN, and MIT's Broad Institute leverage Spark for scientific data analysis.
- Spark is essential for big data analysis and continues to develop rapidly.
- A critical tool for data scientists and engineers solving large-scale data challenges.

What Is Apache Spark?

Running Spark:

- Spark supports Python, Java, Scala, R, and SQL.
- It requires Java (JVM); for Python or R users, install the relevant interpreters.

1. Install Spark Locally:

- Download and Install:
  - Visit the official Spark download page and select "Pre-built for Hadoop 2.7+."
  - Ensure Java and Python (if needed) are installed.
- Extract and Run:
  - Extract the downloaded TAR file and run Spark from the command line.
  - Example:

```
cd ~/Downloads
tar -xf spark-2.2.0-bin-hadoop2.7.tgz
cd spark-2.2.0-bin-hadoop2.7
```

What Is Apache Spark?

Running Spark:

2. Launching Spark's Interactive Consoles:

- We can start an interactive shell in Spark for several different programming languages like Python, Scala, SQL and etc.

1. Launching the Python console:

- We'll need Python 2 or 3 installed in order to launch the Python console.
- From Spark's home directory, run the following code:

```
./bin/pyspark
```

- After we've done that, type "spark" and press Enter. We'll see the SparkSession object printed.

What Is Apache Spark?

Running Spark:

2. Launching Spark's Interactive Consoles: cont'd.

2. Launching the Scala console:

- To launch the Scala console, we will need to run the following command:

```
./bin/spark-shell
```

- After we've done that, type "spark" and press Enter. As in Python, we'll see the SparkSession object.

What Is Apache Spark?

Running Spark:

2. Launching Spark's Interactive Consoles: cont'd.

3. Launching the SQL console:

- To launch the SQL console, we will need to run the following command:

```
./bin/spark-sql
```

What Is Apache Spark?

Running Spark:

3. Running Spark in the Cloud:

Run Spark in the Cloud with Databricks:

- Databricks Community Edition:
  - Offers a free cloud environment with an interactive notebook experience for learning Spark.
- Supports Multiple Languages:
  - Use Scala, Python, SQL, or R in a web-based interface to run and visualize results.

Spark: *Outline*

- **What is Apache Spark**
  - History of Spark
  - The Present and Future of Spark
  - Running Spark
- **Introduction to Spark**
  - Spark's Basic Architecture
    - Spark Applications
  - Spark's Language APIs
  - Spark's APIs
  - Starting Spark
  - The Spark Session
  - Data Frames
    - Partitions
  - Transformations
    - Lazy Evaluation
  - Actions
  - Spark UI
  - An End-to-End Example
    - Data Frames and SQL

Introduction to Spark

Spark's Basic Architecture:

Single Machine Limitations

- Single machines are ideal for tasks like watching movies or working with spreadsheets.
- They struggle with data processing due to limited power and resources.
- Large computations on huge datasets can be slow or unfeasible.

Clusters as a Solution

- A cluster pools resources from multiple machines.
- Cumulative resources act as if they are from a single, more powerful computer.
- However, simply having multiple machines isn't enough-we need a way to manage them.

Introduction to Spark

Spark's Basic Architecture:

Role of Spark

- Spark is a framework that coordinates tasks across a cluster of computers.
- It ensures efficient use of all the resources for computations on data.
- Tasks are distributed across different machines and processed in parallel.

Cluster Managers

- A cluster manager like Spark's standalone manager, YARN, or Mesos controls the cluster.
- Spark applications are submitted to the cluster manager.
- The cluster manager allocates resources to these applications for task execution.

Introduction to Spark

Spark's Basic Architecture:

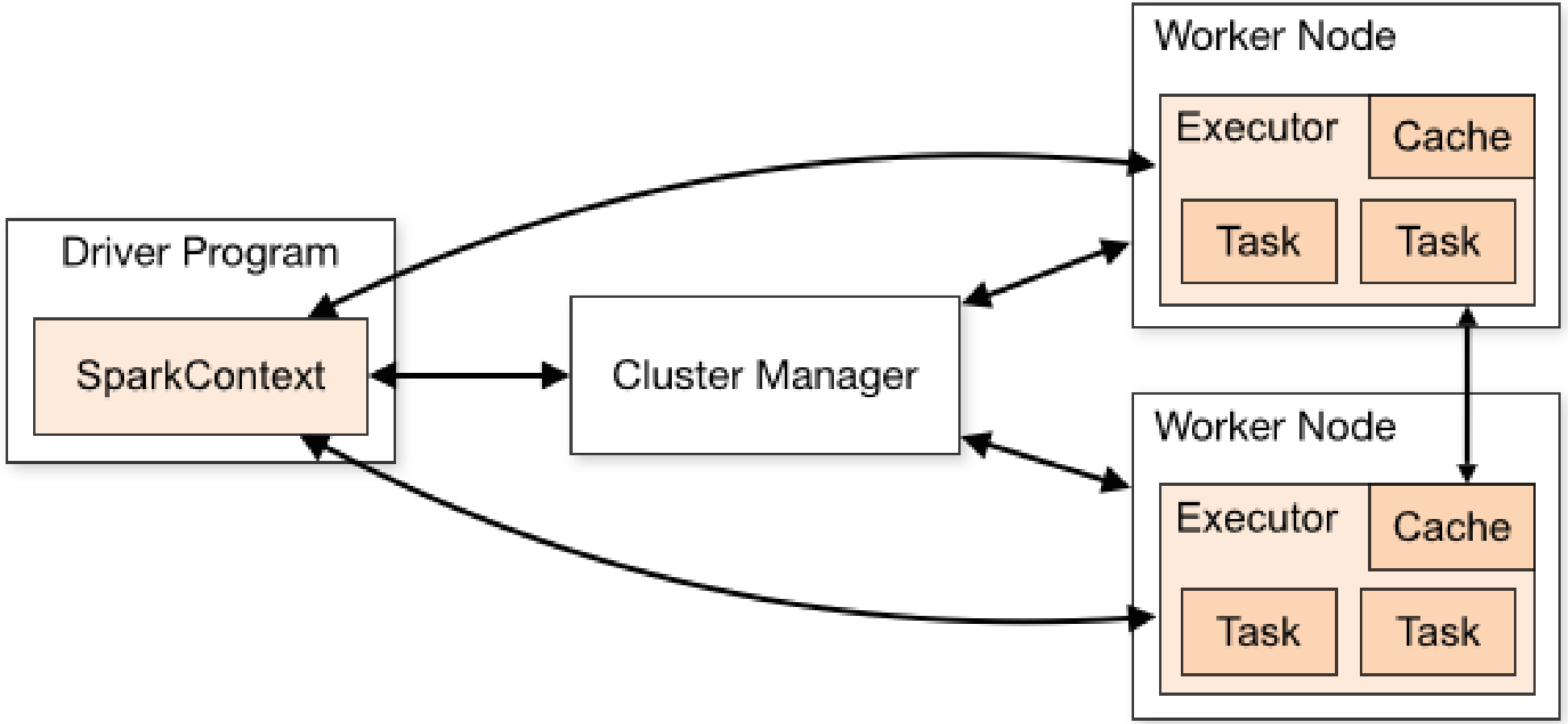


Figure. Apache Spark architecture

Introduction to Spark

Spark's Basic Architecture:

Components

- **Driver:** Controls the application, distributes work, and coordinates across the cluster.
- **Executor:** Performs the work on different nodes.
- **Cluster Manager:** Manages resources in the cluster.

Introduction to Spark

Spark's Basic Architecture: Spark Applications

Driver Process and Executor Processes Overview

1. Spark Application Structure

The following figure demonstrates how the cluster manager controls physical machines and allocates resources to Spark Applications

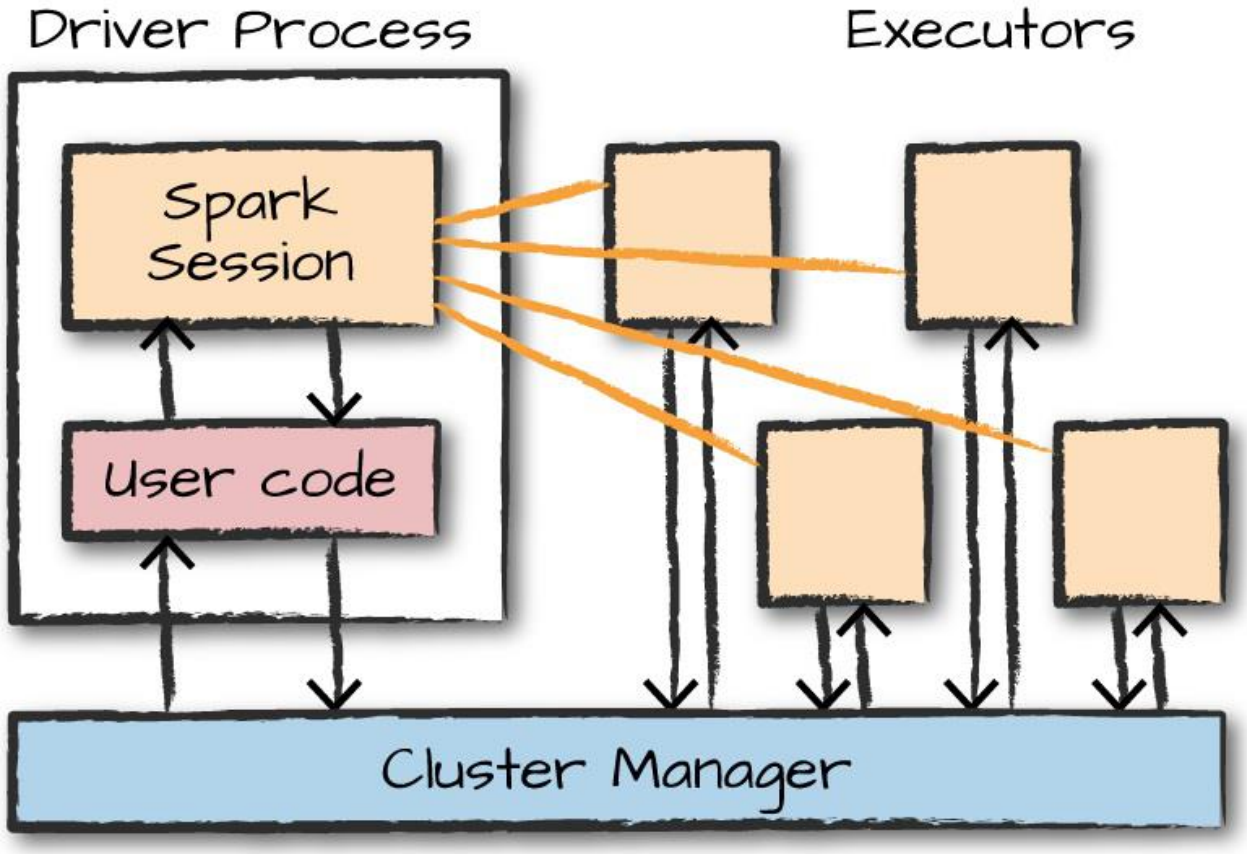


Figure: Spark Application Architecture

Introduction to Spark

Spark's Basic Architecture: Spark Applications

Driver Process and Executor Processes Overview

1. Spark Application Structure cont'd.

- A Spark Application consists of a driver process and multiple executor processes.
- The driver runs the main() function and is essential for managing the Spark Application.
- Executors perform the actual computation and report back to the driver.

Introduction to Spark

Spark's Basic Architecture: Spark Applications

Driver Process and Executor Processes Overview

2. Role of the Driver Process

- Maintains information about the Spark Application.
- Responds to user input and analyzes the program.
- Distributes and schedules tasks across the executors.

3. Role of Executors

- Executors carry out tasks assigned by the driver.
- They execute the code and report computation results back to the driver.

Introduction to Spark

Spark's Basic Architecture: Spark Applications

Driver Process and Executor Processes Overview

4. Cluster Manager and Resources

- The cluster manager allocates resources to Spark Applications.
- Can be Spark's standalone manager, YARN, or Mesos.
- Multiple Spark Applications can run on the same cluster simultaneously.

5. Key Points about Spark Applications

- The driver manages execution and coordinates tasks across executors.
- Executors continuously run Spark code and handle tasks.
- The driver can interact with Spark through different language APIs.

Introduction to Spark

Spark's Basic Architecture: Spark Applications

Driver Process and Executor Processes Overview

Spark Application Example:

- A data scientist submits a Spark job to analyze website logs.
- The driver distributes the work of reading, cleaning, and aggregating the logs to different executors.

Introduction to Spark

Spark's Language APIs:

Spark provides APIs for different programming languages, all built on the same core concepts.

These APIs translate our code into Spark commands that run on a cluster, ensuring similar performance across languages when using Structured APIs.

- Scala: Spark's default language, as it's written in Scala.
- Java: Fully supported, allowing Spark code to be written in Java.
- Python: Nearly all Scala features are available, with translations into Python.
- SQL: Supports ANSI SQL 2003, making Spark accessible to non-programmers.
- R: Supports SparkR (in Spark core) and sparklyr (community package).
- DataFrames: Distributed table-like structures, a core concept for working with structured data.

Introduction to Spark

Spark's Language APIs:

The following Figure presents a simple illustration of the relationship between the SparkSession and Spark's Language API

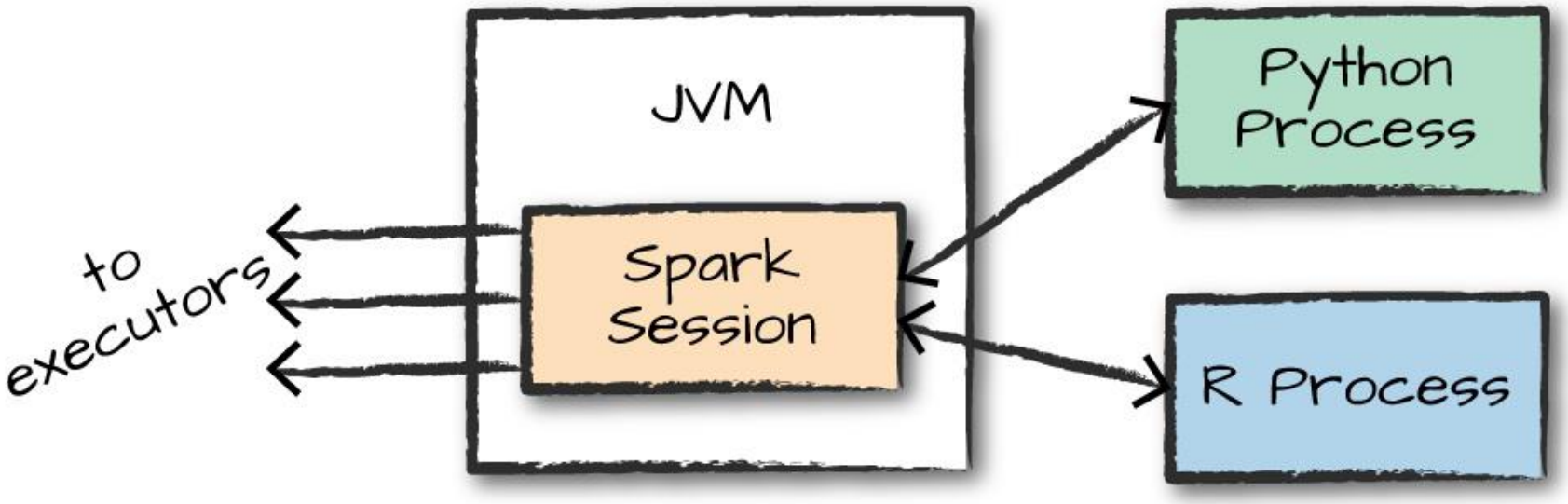


Figure. The relationship between the SparkSession and Spark's Language API

In all languages, you interact with a SparkSession object, the entry point to running Spark code. When using Python or R, we write code in those languages, and Spark translates it to run on JVMs, managing the complexity for us.

Introduction to Spark

Spark's Language APIs:

Example:

A Python user can use PySpark to write a job that reads and analyzes data from a CSV file in the cloud:

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()
df = spark.read.csv("data.csv", header=True)
df.show()
```

Introduction to Spark

Spark's APIs:

- Although we can drive Spark from a variety of languages, what it makes available in those languages is worth mentioning.
- Spark has two fundamental sets of APIs:
  - the low-level "unstructured" APIs, and
  - the higher-level structured APIs.

Introduction to Spark

Spark's APIs:

Low-Level "Unstructured" APIs:

- These APIs provide detailed control over Spark's core functionalities but require a more in-depth understanding of its internals.
- They are often used for fine-tuned optimizations or when we need functionality that the higher-level APIs do not expose.
- For example, we can directly manipulate RDDs (Resilient Distributed Datasets), which are low-level data structures in Spark.
- This allows for custom partitioning, caching, and iteration over elements, which are not directly available through structured APIs.

Introduction to Spark

Spark's APIs:

Low-Level "Unstructured" APIs: Example

```
# Using RDDs to filter and count even numbers
from pyspark import SparkContext
sc = SparkContext()
data = sc.parallelize([1, 2, 3, 4, 5, 6])
filtered = data.filter(lambda x: x % 2 == 0)
print(filtered.count())
# Outputs: 3
```

Introduction to Spark

Spark's APIs:

High-Level "Structured" APIs:

- These are more user-friendly and abstract the complexities of distributed computing.
- They are suitable for most data manipulation tasks, especially when working with structured or semi-structured data.
- The Structured APIs include DataFrames and Datasets, which provide a higher level of abstraction and are similar to working with databases.
- These APIs support various data sources and can optimize queries automatically.

Introduction to Spark

Spark's APIs:

High-Level "Structured" APIs: Example

```
# Using DataFrame to count occurrences of a name
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('Example').getOrCreate()
data = spark.createDataFrame([("Alice", 1), ("Bob", 2),
   ("Alice",3)], ["Name", "Id"])
counts = data.groupBy("Name").count()
counts.show()
# Outputs:
# +-----+-----+
# | Name|count|
# +-----+-----+
# |Alice|    2|
# |  Bob|    1|
# +-----+-----+
```

Introduction to Spark

Starting Spark:

- When we actually go about writing our Spark Application, we are going to need a way to send user commands and data to it.
- We do that by first creating a SparkSession.
- When we start Spark in this interactive mode, we implicitly create a SparkSession that manages the Spark Application.
- When we start it through a standalone application, we must create the SparkSession object ourselves in our application code.

Introduction to Spark

Starting Spark:

- To do this, we will start Spark's local mode.
- This means running `./bin/spark-shell` to access the Scala console to start an interactive session.
- We can also start the Python console by using `./bin/pyspark`. This starts an interactive Spark Application.
- There is also a process for submitting standalone applications to Spark called spark-submit, whereby we can submit a precompiled application to Spark.

Introduction to Spark

The SparkSession:

- It's the main control point for a Spark Application.
- Manages how Spark executes user tasks across the cluster.
- One SparkSession per Spark Application.
- In Scala and Python, the variable is available as `spark` when we start the console.
- In Scala, we should see something like the following:

```
res0: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@...
```

- In Python we'll see something like this:

```
<pyspark.sql.session.SparkSession at 0x7efda4c1ccd0>
```

Introduction to Spark

The SparkSession: Example

Perform the simple task of creating a range of numbers. This range of numbers is just like a named column in a spreadsheet:

```
// in Scala
val myRange = spark.range(1000).toDF("number")
# in Python
myRange = spark.range(1000).toDF("number")
```

Outcome:

- Creates a DataFrame with 1,000 rows (values from 0 to 999).
- Represents a distributed collection across the cluster.

Introduction to Spark

DataFrames in Spark:

- **DataFrame:** Distributed collection of data organized into named columns.
- **Advantages:**
  - Enables large-scale data processing using SQL queries.
  - Works across multiple languages (Python, Scala, R).

Example:

DataFrames are similar to a table in a relational database, but they can scale horizontally across multiple machines.

Introduction to Spark

DataFrames in Spark:

The following Figure illustrates the fundamental difference: a spreadsheet sits on one computer in one specific location, whereas a Spark DataFrame can span thousands of computers.

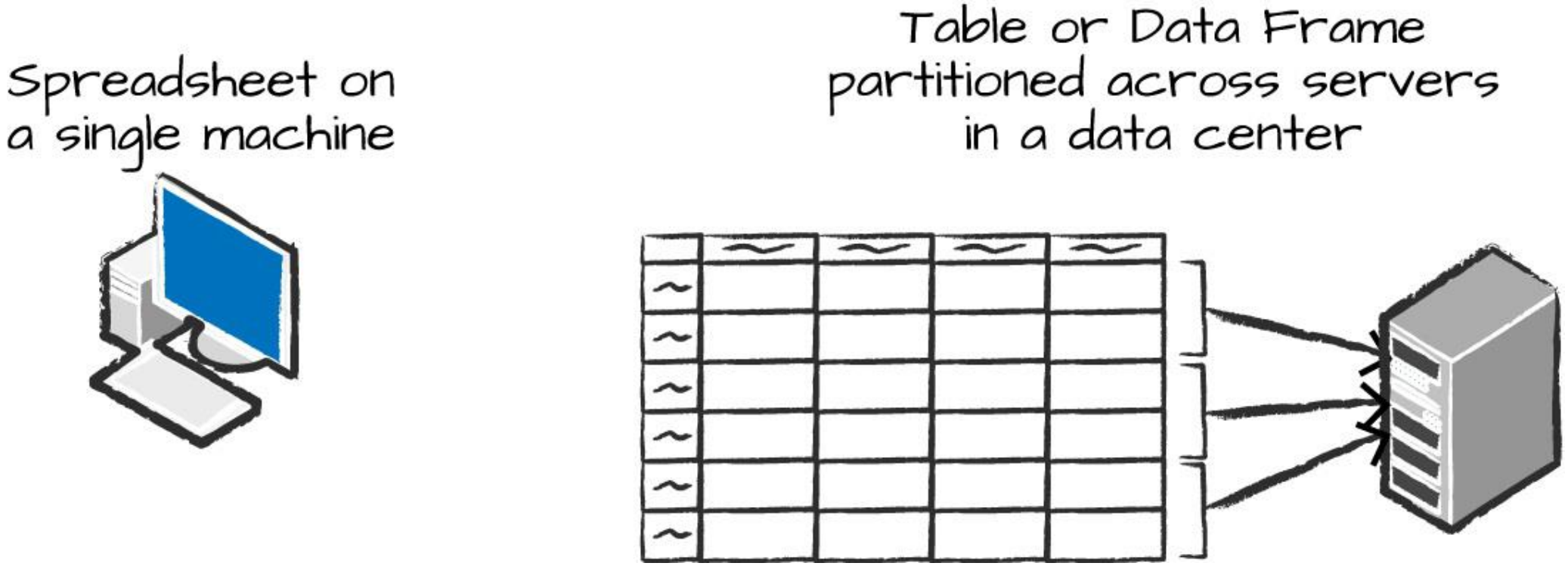


Figure. Distributed versus single-machine analysis

Introduction to Spark

DataFrames in Spark:

Note:

Spark has several core abstractions: Datasets, DataFrames, SQL Tables, and Resilient Distributed Datasets (RDDs).

These different abstractions all represent distributed collections of data.

The easiest and most efficient are DataFrames, which are available in all languages.

Introduction to Spark

DataFrames in Spark: Partitions

Partitions in Spark

What is a Partition?

- A chunk of data that resides on one physical machine in the cluster.
- Determines how data is distributed and processed in parallel.

Key Points:

- More Partitions = More Parallelism:
  - One partition = One parallel task, regardless of the number of executors.
  - Many partitions with one executor = Limited parallelism due to one computation resource.
- DataFrame Handling:
  - Partitions are managed automatically by Spark.
  - Focus on high-level data transformations.
  - Spark handles partitioning and execution.

Lower-Level Control:

- Advanced users can manipulate partitions through the RDD interface.

Introduction to Spark

Spark Transformations:

- Immutable Data Structures:
  - In Spark, data structures cannot be changed once created.
  - Instead, we modify data by applying transformations.
- What Are Transformations?
  - Transformations are instructions to modify DataFrames.
  - They don't produce immediate results but define what operations to perform.
- Example: Finding even numbers in a DataFrame:
  - Scala: `val divisBy2 = myRange.where("number % 2 = 0")`
  - Python: `divisBy2 = myRange.where("number % 2 = 0")`
  - Note: No output is shown until an action is called.

Introduction to Spark

Spark Transformations:

- Types of Transformations:
  - Narrow Transformations:

    Each input partition affects only one output partition (e.g., filtering data).
  - Wide Transformations:

    Input partitions affect multiple output partitions (e.g., shuffling data across the cluster).

Introduction to Spark

Spark Transformations:

- Narrow vs Wide Transformations:
  - Narrow: Operations are done in-memory (e.g., multiple filters).
  - Wide: Requires disk storage (e.g., shuffling data).

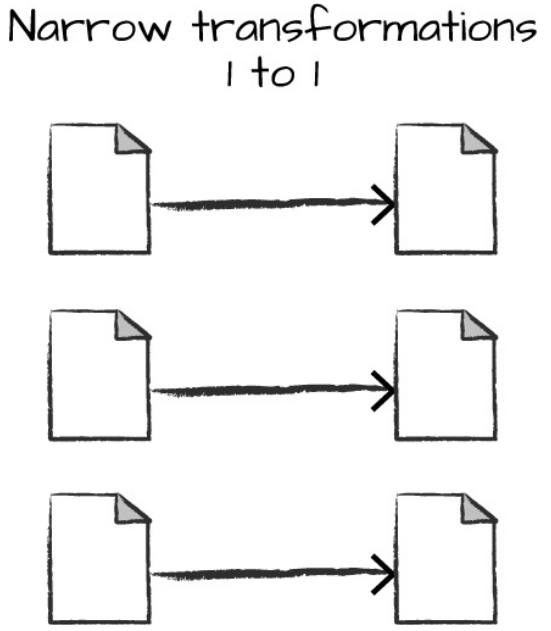


Figure 1. A narrow dependency

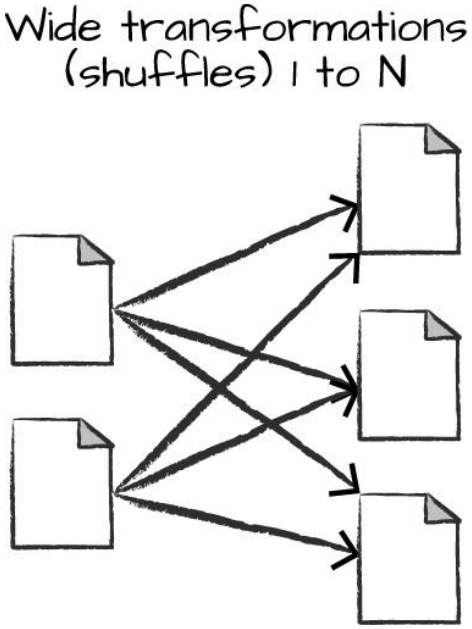


Figure 2. A wide dependency

Introduction to Spark

Spark Transformations: Lazy Evaluation

- Lazily evaluate operations such as `map`, `filter`, `groupBy`.
- These are not executed immediately; Spark waits until an action is called.
  - Concept: Spark delays execution until the last moment.
  - Benefit: Builds an optimized plan for data processing.
  - Example: Predicate Pushdown-Spark optimizes filters to access only the required data efficiently.

Concept: Spark builds a Directed Acyclic Graph (DAG) of transformations and only executes when an action is triggered.

Example:

If we chain multiple transformations, Spark optimizes the entire pipeline before running the job, which results in faster execution.

Introduction to Spark

Spark Actions:

- What Are Actions?

  Actions trigger the execution of transformations and compute results.

- Simple Example:
  - Count Action:

    ```
    divisBy2.count()
    ```

    - ❖ Counts the total number of records in the DataFrame.
    - ❖ Example Output: 500

Introduction to Spark

Spark Actions:

- Types of Actions:

1. View Data:

   Displays data in the console.

2. Collect Data:

   Gathers data into native objects in the programming language.

3. Write Data:

   Outputs data to external sources.

Introduction to Spark

Spark Actions:

- How Actions Work:
  - Actions start a Spark job that processes transformations and generates results.
  - The Spark UI can be used to monitor and inspect these jobs.

Actions: Trigger the actual computation, such as `count()`, `collect()`, `save()`, `show()`.

Example:

- Display the result with `evenNumbers.show()`.

Introduction to Spark

Spark UI:

- What Is Spark UI?

  A web interface to monitor and manage Spark jobs.

- Accessing Spark UI:
  - Local Mode: `http://localhost:4040`
  - Cluster Mode: Available on port 4040 of the driver node.
- Features:
  - Job Progress: View the state of our Spark jobs.
  - Environment Details: Check the environment and cluster status.
  - Tuning and Debugging: Useful for performance tuning and troubleshooting.

Introduction to Spark

Spark UI:

- Example:

  The UI shows details such as stages and tasks executed for a Spark job.

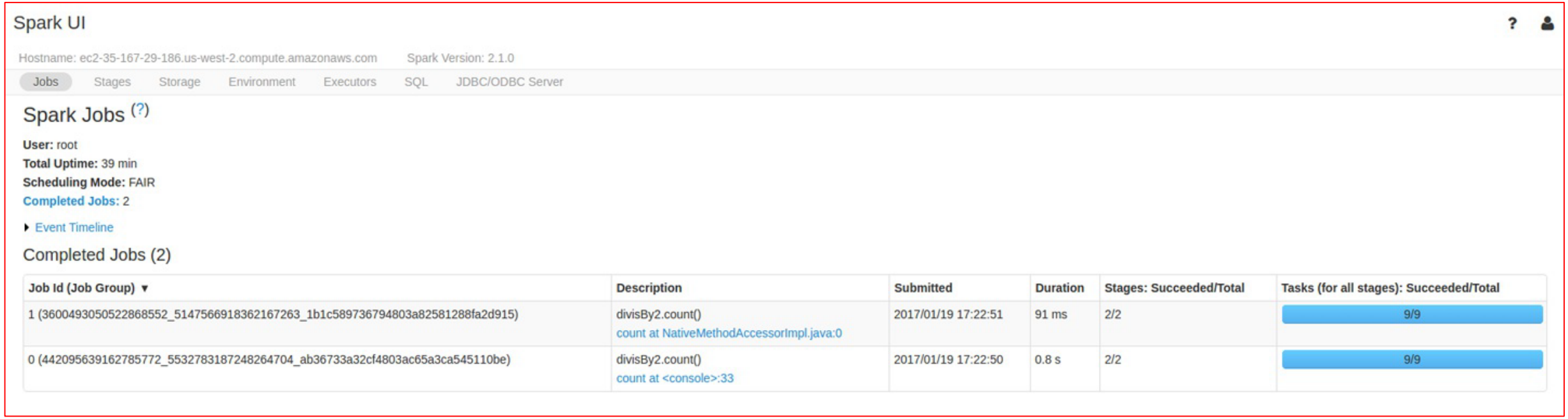


Figure. The Spark UI

Introduction to Spark

Analyzing Flight Data with Spark: An End-to-End Example

1. Reading the Data (CSV Format)

We use Spark to analyze flight data from CSV files, containing columns such as `DEST_COUNTRY_NAME`, `ORIGIN_COUNTRY_NAME`, and `count`.

Spark reads the file, infers the schema, and treats the first row as the header.

```
// python example
flightData2015 = spark.read \
    .option("inferSchema", "true") \
    .option("header", "true") \
    .csv("/data/flight-data/csv/2015-summary.csv")
```

- Schema Inference: Spark guesses data types based on file content.
- Lazy Evaluation: The data isn't read until an action like `take()` or `show()` is performed.

Introduction to Spark

Analyzing Flight Data with Spark: An End-to-End Example

2. Transformations (Sorting & Grouping Data)

Transformations in Spark are lazy, meaning they don't run until an action is triggered. Let's sort the data by the `count` column:

```
// python example
flightData2015.sort("count").explain()
```

- Explain Plan: Shows the steps Spark will take to execute this query.
- Spark waits to execute the sort until an action is performed.

Introduction to Spark

Analyzing Flight Data with Spark: An End-to-End Example

3. Performing Actions

Actions force Spark to execute transformations and return results.

For example, to view the top 2 rows after sorting:

```
// python example
flightData2015.sort("count").take(2)
```

- Example Output:

```
// SQL Code
[Row(DEST_COUNTRY_NAME='United States', ORIGIN_COUNTRY_NAME='Singapore',
count=1),
Row(DEST_COUNTRY_NAME='Moldova', ORIGIN_COUNTRY_NAME='United States',
count=1)]
```

Introduction to Spark

Analyzing Flight Data with Spark: An End-to-End Example

4. SQL Integration with Spark

We can register a DataFrame as a SQL view and perform SQL queries on it:

```
// python example
flightData2015.createOrReplaceTempView("flight_data_2015")
sqlWay = spark.sql("""
    SELECT DEST_COUNTRY_NAME, sum(count) as destination_total
    FROM flight_data_2015
    GROUP BY DEST_COUNTRY_NAME
    ORDER BY destination_total DESC
    LIMIT 5
""")
sqlWay.show()
```

Introduction to Spark

Analyzing Flight Data with Spark: An End-to-End Example

4. SQL Integration with Spark

- Example Output:

```
+-----------------+-----------------+
|DEST_COUNTRY_NAME|destination_total|
+-----------------+-----------------+
|    United States|           411352|
|           Canada|             8399|
|           Mexico|             7140|
|  United Kingdom|             2025|
|           Japan|              1548|
+-----------------+-----------------+
```

Analyzing Flight Data with Spark: An End-to-End Example

5. DataFrame Transformations

We can achieve the same result using DataFrame transformations:

```
// python example
from pyspark.sql.functions import desc

flightData2015.groupBy("DEST_COUNTRY_NAME") \
    .sum("count") \
    .withColumnRenamed("sum(count)", "destination_total") \
    .sort(desc("destination_total")) \
    .limit(5) \
    .show()
```

- Same Output: The result is the same as the SQL query output.

Introduction to Spark

Analyzing Flight Data with Spark: An End-to-End Example

6. Performance Optimization (Shuffle Partitions)

To improve performance, we can adjust Spark's shuffle partitions.

By default, Spark uses 200 partitions for shuffles. We can reduce this number to optimize performance.

```
// python example

spark.conf.set("spark.sql.shuffle.partitions", "5")
```

7. Monitoring Performance

We can monitor the job's progress and execution plan via the Spark UI (usually available at port 4040). This interface provides insights into both the logical and physical execution plans of your Spark jobs.

Introduction to Spark

Analyzing Flight Data with Spark: An End-to-End Example

Summary

- Spark allows seamless data processing through transformations and actions.
- You can work with data using SQL queries or DataFrame APIs, and both compile to the same underlying execution plan.
- By adjusting configurations such as shuffle partitions, you can optimize the performance of your Spark jobs.