Apache Spark

Vinit Shah

- Open-source unified analytics engine for large-scale data processing.
- Provides an interface for programming entire clusters with implicit data parallelism and fault tolerance.
- Started as a research project at the UC Berkeley AMPLab in 2009, and was open sourced in early 2010.

Motivation

MapReduce

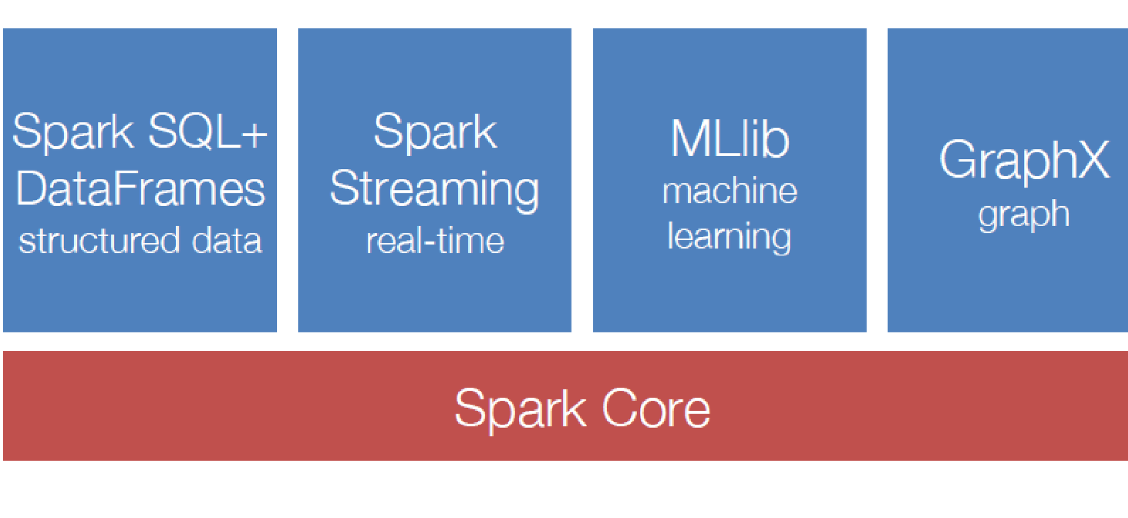
- Scalable processing engine of the Hadoop ecosystem
- Disk-based data processing framework (HDFS files)
- Data is reloaded from disk with every query → Costly I/O
- Persists intermediate results to disk
- Costly I/O → Not appropriate for iterative or stream processing workloads<
- Best for ETL like workloads (batch processing)

Spark

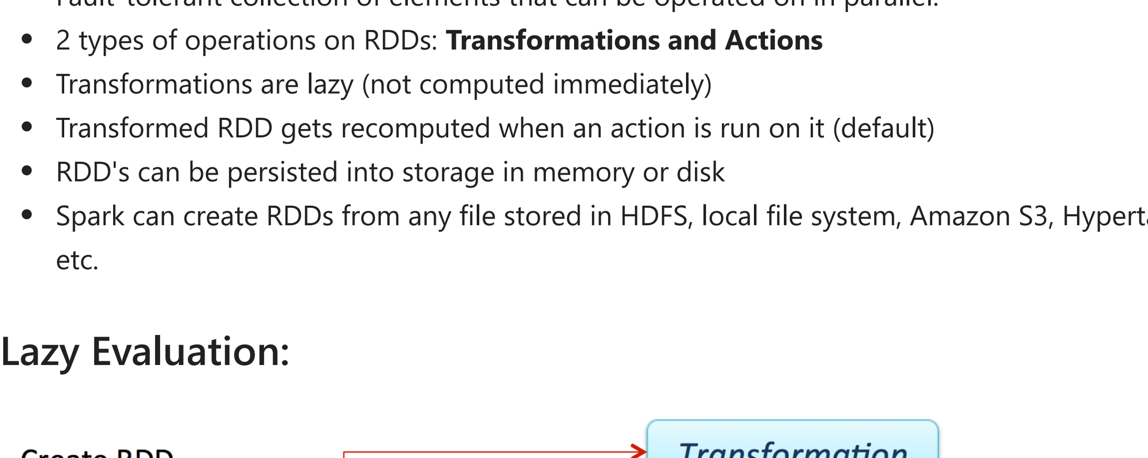
- Memory based data processing framework
- Avoids costly I/O by keeping intermediate results in memory
- Leverages distributed memory and remembers operations applied to dataset
- Data locality based computation → High Performance
- Best for both iterative (or stream processing) and batch workloads

Spark Architecture

- Driver program: The process running the main() function of the application and creating the SparkContext
- Cluster manager: An external service for acquiring resources on the cluster (e.g. standalone manager, Mesos, YARN)
- Worker node: Any node that can run application code in the cluster
- Executor: A process launched for an application on a worker node, that runs tasks and keeps data in memory or disk storage across them.
- Task: A unit of work that will be sent to one executor



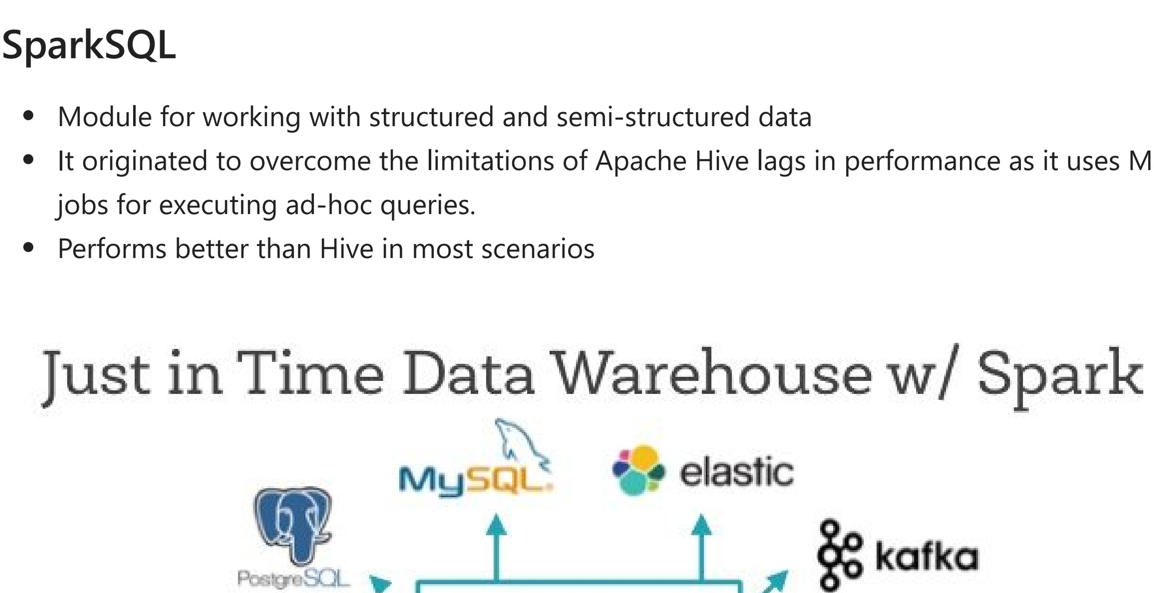
Spark Ecosystem



Spark RDD

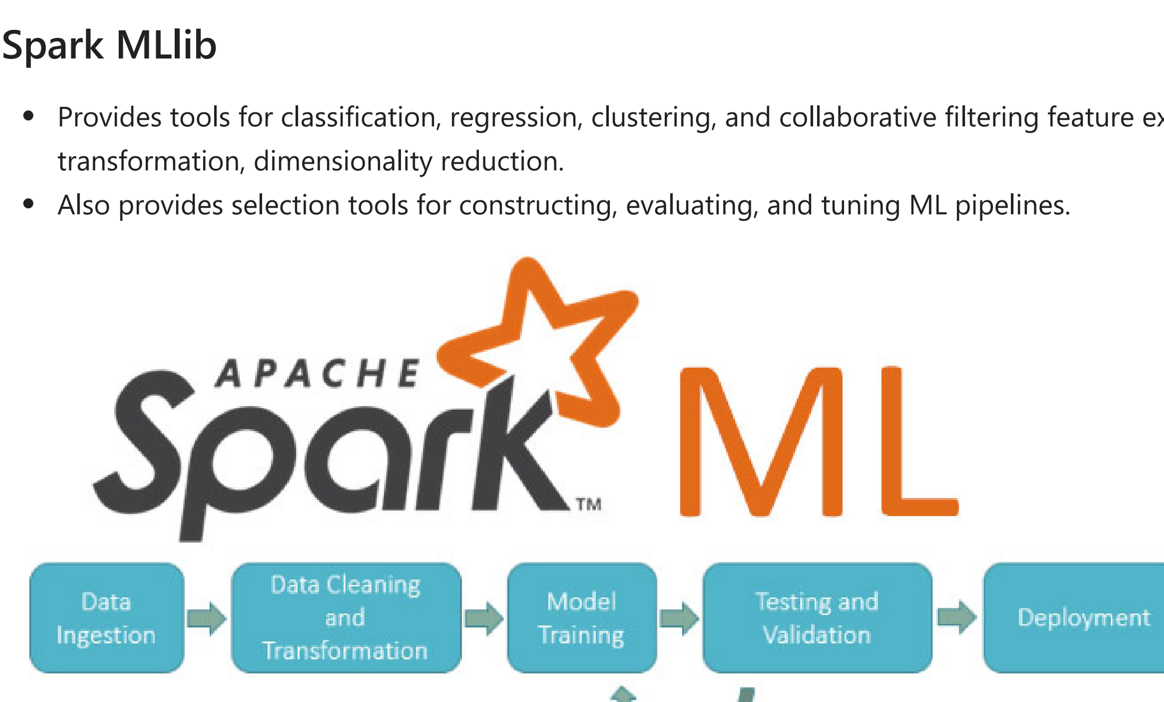
- Resilient Distributed Datasets (RDD) are the primary abstraction in Spark.
- Fault-tolerant collection of elements that can be operated on in parallel.
- 2 types of operations on RDDs: **Transformations and Actions**
- Transformations are lazy (not computed immediately)
- Transformed RDD gets recomputed when an action is run on it (default)
- RDD's can be persisted into storage in memory or disk
- Spark can create RDDs from any file stored in HDFS, local file system, Amazon S3, Hypertable, HBase, etc.

Lazy Evaluation:



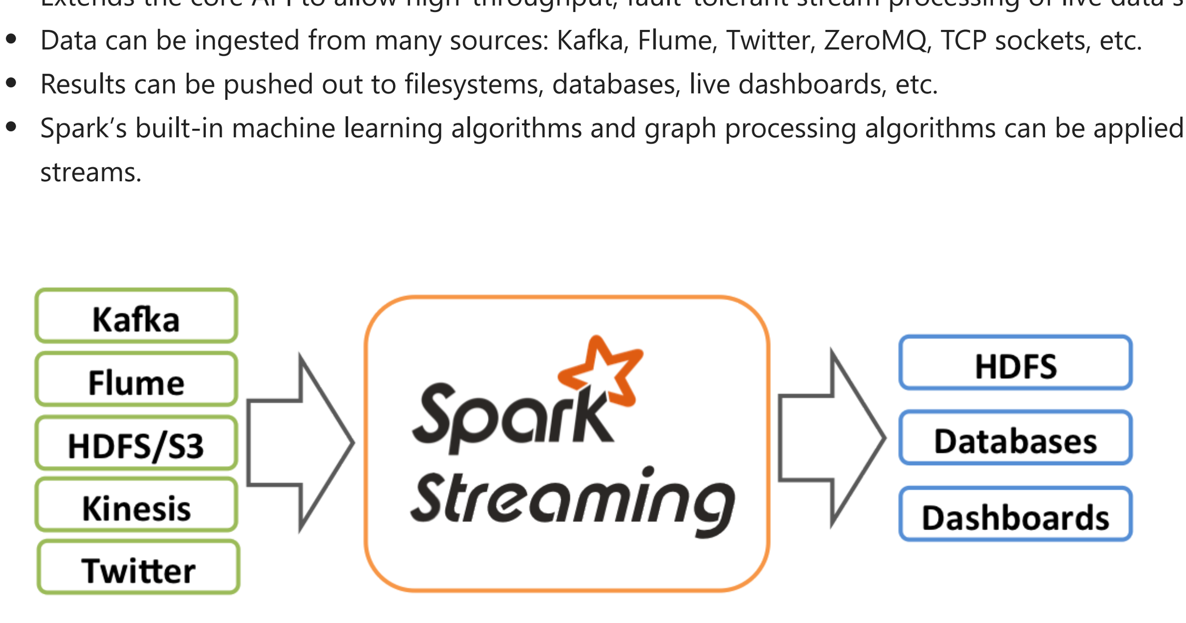
SparkSQL

- Module for working with structured and semi-structured data
- It originated to overcome the limitations of Apache Hive lags in performance as it uses MapReduce jobs for executing ad-hoc queries.
- Performs better than Hive in most scenarios



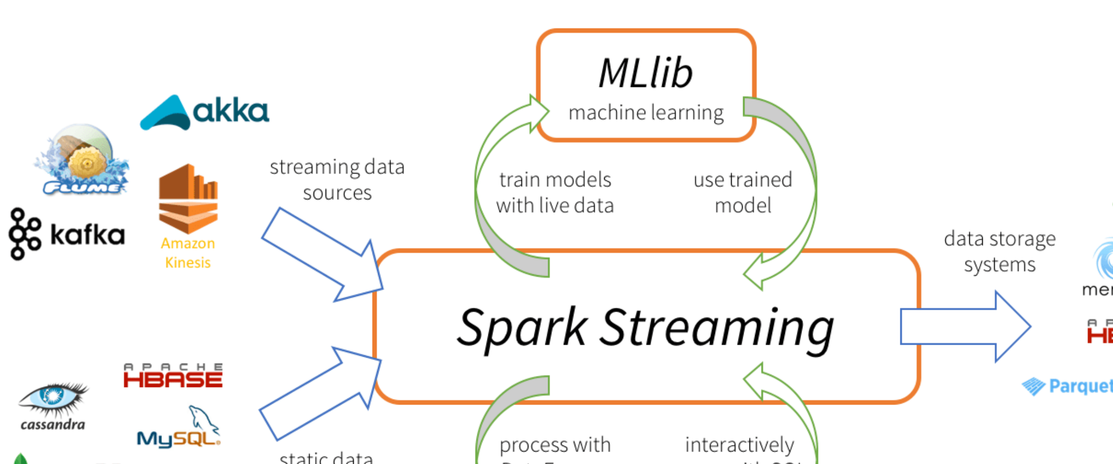
Spark MLlib

- Provides tools for classification, regression, clustering, and collaborative filtering feature extraction, transformation, dimensionality reduction.
- Also provides selection tools for constructing, evaluating, and tuning ML pipelines.



Spark Streaming

- Extends the core API to allow high-throughput, fault-tolerant stream processing of live data streams.
- Data can be ingested from many sources: Kafka, Flume, Twitter, ZeroMQ, TCP sockets, etc.
- Results can be pushed out to filesystems, databases, live dashboards, etc.
- Spark's built-in machine learning algorithms and graph processing algorithms can be applied to data streams.



Use Case:

In [1]:

```python
import findspark
findspark.init()
findspark.find()
```

Out[1]: 'C:\\Code\\Hadoop-Setup\\Spark-3.0'

In [2]:

```python
import pyspark
```

Spark RDD

In [7]:

```python
from pyspark import SparkContext

# Create Spark Context
sc = SparkContext("local", "WordCount App")

# Open text file
text_file = sc.textFile("data/word_count.txt")

# Transformations
word_count = text_file \
            .flatMap(lambda line: line.split(" ")) \
            .map(lambda word: (word, 1)) \
            .reduceByKey(lambda a, b: a + b)

# Action
result = word_count.collect()

# Result
for item in result:
    print(item)

# Save result as text file
# result.saveAsTextFile("output/word_count")

# End the Spark context
sc.stop()
```

```
('Lorem', 2)
('Ipsum', 2)
('is', 1)
('simply', 1)
('dummy', 2)
('text', 2)
('of', 2)
('the', 3)
('printing', 1)
('and', 2)
('typesetting', 1)
('industry.', 1)
('has', 1)
('been', 1)
("industry's", 1)
('standard', 1)
('ever', 1)
('since', 1)
('1500s,', 1)
('when', 1)
('an', 1)
('unknown', 1)
('printer', 1)
('took', 1)
('a', 2)
('galley', 1)
('type', 2)
('scrambled', 1)
('it', 1)
('to', 1)
('make', 1)
('specimen', 1)
('book.', 1)
```

Spark SQL

In [8]:

```python
from pyspark.sql import SparkSession

# Create Spark Session
spark = SparkSession.builder.appName("Titanic App").getOrCreate()

# Read CSV file
df = spark.read \
        .option("delimiter", ",") \
        .option("inferSchema", "true") \
        .option("header", "true") \
        .csv("data/titanic.csv")

# Drop the row if all values are null
df.na.drop("all")

df.show()
```

```
+-----------+--------+------+--------------------+------+----+-----+-----+------------
----+-------+-----+--------+
|PassengerId|Survived|Pclass|                Name|   Sex| Age|SibSp|Parch|          Ti
cket|   Fare|Cabin|Embarked|
+-----------+--------+------+--------------------+------+----+-----+-----+------------
----+-------+-----+--------+
|          1|       0|     3|Braund, Mr. Owen ...|  male|22.0|    1|    0|       A/5 2
1171|   7.25| null|       S|
|          2|       1|     1|Cumings, Mrs. Joh...|female|38.0|    1|    0|        PC 1
7599|71.2833|  C85|       C|
|          3|       1|     3|Heikkinen, Miss. ...|female|26.0|    0|    0|STON/O2. 310
1282|  7.925| null|       S|
|          4|       1|     1|Futrelle, Mrs. Ja...|female|35.0|    1|    0|          11
3803|   53.1| C123|       S|
|          5|       0|     3|Allen, Mr. Willia...|  male|35.0|    0|    0|          37
3450|   8.05| null|       S|
|          6|       0|     3|    Moran, Mr. James|  male|null|    0|    0|          33
0877| 8.4583| null|       Q|
|          7|       0|     1|McCarthy, Mr. Tim...|  male|54.0|    0|    0|           1
7463|51.8625|  E46|       S|
|          8|       0|     3|Palsson, Master. ...|  male| 2.0|    3|    1|          34
9909| 21.075| null|       S|
|          9|       1|     3|Johnson, Mrs. Osc...|female|27.0|    0|    2|          34
7742|11.1333| null|       S|
|         10|       1|     2|Nasser, Mrs. Nich...|female|14.0|    1|    0|          23
7736|30.0708| null|       C|
|         11|       1|     3|Sandstrom, Miss. ...|female| 4.0|    1|    1|         PP 
9549|   16.7|   G6|       S|
|         12|       1|     1|Bonnell, Miss. El...|female|58.0|    0|    0|          11
3783|  26.55| C103|       S|
|         13|       0|     3|Saundercock, Mr. ...|  male|20.0|    0|    0|       A/5. 
2151|   8.05| null|       S|
|         14|       0|     3|Andersson, Mr. An...|  male|39.0|    1|    5|          34
7082| 31.275| null|       S|
|         15|       0|     3|Vestrom, Miss. Hu...|female|14.0|    0|    0|          35
0406| 7.8542| null|       S|
|         16|       1|     2|Hewlett, Mrs. (Ma...|female|55.0|    0|    0|          24
8706|   16.0| null|       S|
|         17|       0|     3|Rice, Master. Eugene|  male| 2.0|    4|    1|          38
2652| 29.125| null|       Q|
|         18|       1|     2|Williams, Mr. Cha...|  male|null|    0|    0|          24
4373|   13.0| null|       S|
|         19|       0|     3|Vander Planke, Mr...|female|31.0|    1|    0|          34
5763|   18.0| null|       S|
|         20|       1|     3|Masselmani, Mrs. ...|female|null|    0|    0|            
2649|  7.225| null|       C|
+-----------+--------+------+--------------------+------+----+-----+-----+------------
----+-------+-----+--------+
only showing top 20 rows
```

In [9]:

```python
from pyspark.sql import functions as F

# Select Max value from Fare column
df.select(F.max("Fare")).show(truncate=False)
```

```
+---------+
|max(Fare)|
+---------+
|512.3292 |
+---------+
```

In [11]:

```python
# Create a Temp View for the dataset
df.createOrReplaceTempView("titanic")

# Query the table
spark.sql("SELECT Age, Sex, Fare FROM titanic WHERE Survived = 1").show()
```

```
+----+------+--------+
| Age|   Sex|    Fare|
+----+------+--------+
|38.0|female| 71.2833|
|26.0|female|   7.925|
|35.0|female|    53.1|
|27.0|female| 11.1333|
|14.0|female| 30.0708|
| 4.0|female|    16.7|
|58.0|female|   26.55|
|55.0|female|    16.0|
|null|  male|    13.0|
|null|female|   7.225|
|34.0|  male|    13.0|
|15.0|female|  8.0292|
|28.0|  male|    35.5|
|38.0|female| 31.3875|
|null|female|  7.8792|
|null|female|146.5208|
|null|female|    7.75|
|null|  male|  7.2292|
|14.0|female| 11.2417|
| 3.0|female| 41.5792|
+----+------+--------+
only showing top 20 rows
```

In [12]:

```python
# Partition the data by Survived column and export the data in parquet file format
df.write.partitionBy("Survived").parquet("output/parquet")

# Partition the data by Survived column and export the data to a Hive table
df.write.partitionBy("Survived").format("hive").saveAsTable("titanic_partition_table")

# Stop the Spark Session
spark.stop()
```