PySpark - Data Processing in Python on top of Apache Spark

Peter Hoffmann

Twitter: @peterhoffmann

github.com/blue-yonder

blue yonder

Tue 16:45 Barria 1 Holger Peters	**USING SCIKIT-LEARN'S INTERFACE FOR TURNING SPAGHETTI DATA SCIENCE INTO MAINTAINABLE SOFTWARE**
Wed 11:00 A2 Sebastian Neubauer	**A PYTHONIC APPROACH TO CONTINUOUS DELIVERY**
Wed 12:30 Barria1 Stephan Erb	**RELEASE MANAGEMENT WITH DEVPI**
Wed 15:45 Google Patrick Mühlbauer	**BUILDING NICE COMMAND LINE INTERFACES – A LOOK BEYOND THE STDLIB**
Wed 16:45 Google Peter Hoffmann	**PYSPARK – DATA PROCESSING IN PYTHON ON TOP OF APACHE SPARK**
Thu 12:30 Python Moritz Gronbach	**WHAT'S THE FUZZ ALL ABOUT? RANDOMIZED DATA GENERATION FOR ROBUST UNIT TESTING**
Thu 16:45 Barria1 Christian Trebing	**BUILDING A MULTI-PURPOSE PLATFORM FOR BULK DATA USING SQLALCHEMY**
Fri 11:45 Barria2 Florian Wilhelm	**"IT'S ABOUT TIME TO TAKE YOUR MEDICATION!" OR HOW TO WRITE A FRIENDLY REMINDER BOT**
Fri 14:30 A2 Phillip Mack	**PYTHON IN THE WORLD OF RETAIL AND MAIL ORDER**

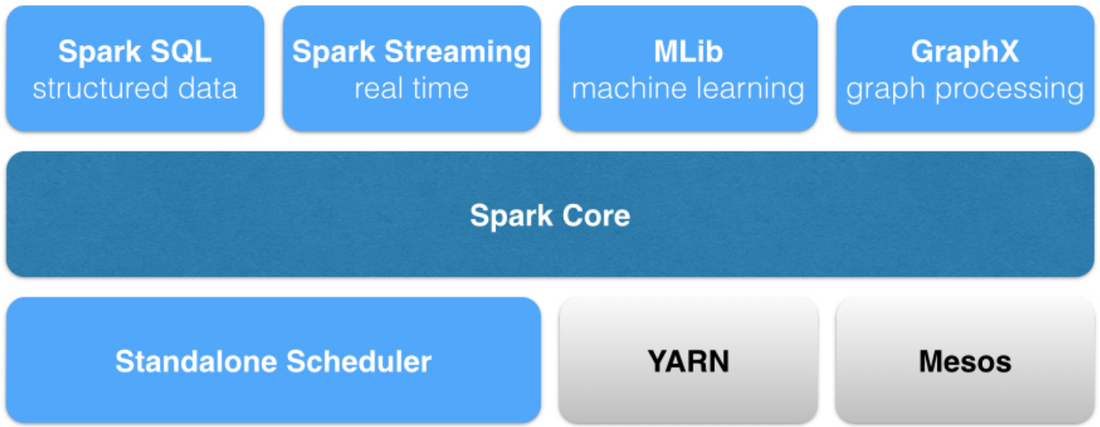
blue yonder

Spark Overview

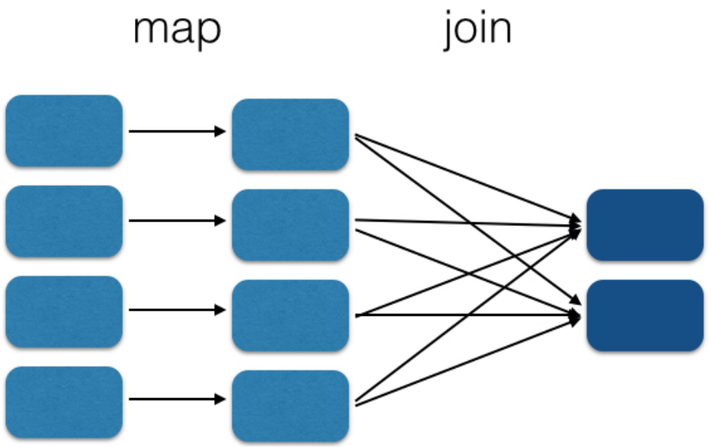
Spark is a **distributed general purpose cluster engine** with APIs in Scala, Java, R and Python and has libraries for streaming, graph processing and machine learning.

Spark offers a functional programming API to manipulate **Resilient Distrubuted Datasets (RDDs)**.

Spark Core is a computational engine responsible for scheduling, distribution and monitoring applications which consist of many **computational task** across many worker machines on a complutation cluster.



Resilient Distributed Datasets



RDDs reperesent a **logical plan** to compute a dataset.

RDDs are fault-toloerant, in that the system can revocer lost data using the **lineage graph** of RDDs (by rerunning operations on the input data to rebuild missing partitions).

RDDs offer two types of operations:

- **Transformations** construct a new RDD from one or more previous ones
- **Actions** compute a result based on an RDD and either return it to the driver program
  or save it to an external storage

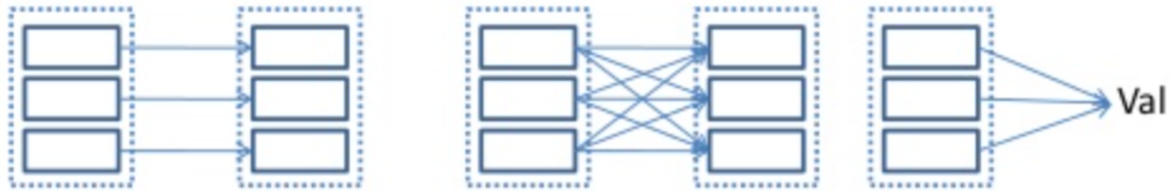
RDD Lineage Graph

Transformations are Operations on RDDs that return a new RDD (like Map/Reduce/Filter).

Many transformations are element-wise, that is that they work on an alement at a time, but this is not true for all operations.

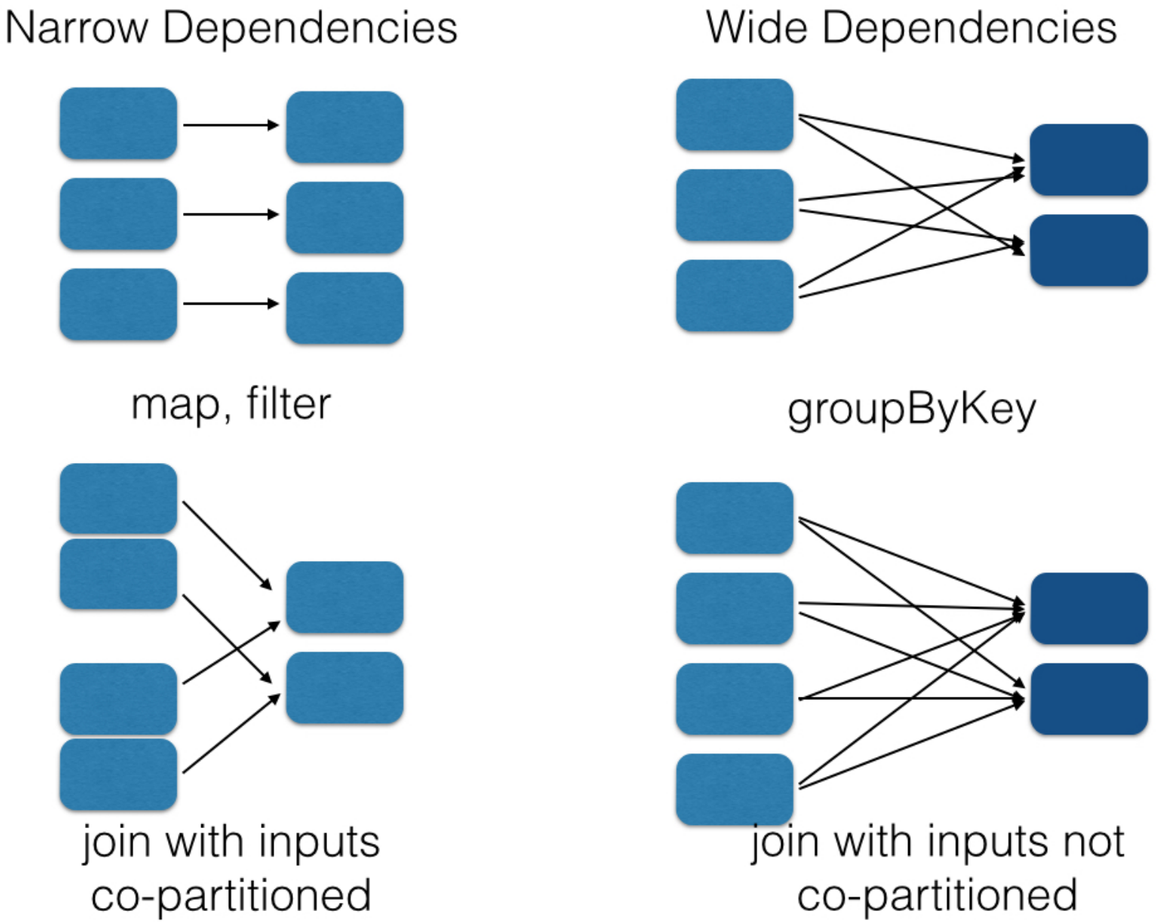
Spark internally records meta-data **RDD Lineage Graph** on which operations have been requested. Think of an RDD as an instruction on how to compute our result through transformations.

Actions compute a result based on the data and return it to the driver prgramm.



Transformations

- map, flatMap
- mapPartitions, mapPartitionsWithIndex
- filter
- sample
- union
- intersection
- distinct
- groupByKey, reduceByKey
- aggregateByKey, sortByKey
- join (inner, outer, leftouter, rightouter, semijoin)



Spark Concepts

RDD as common interface

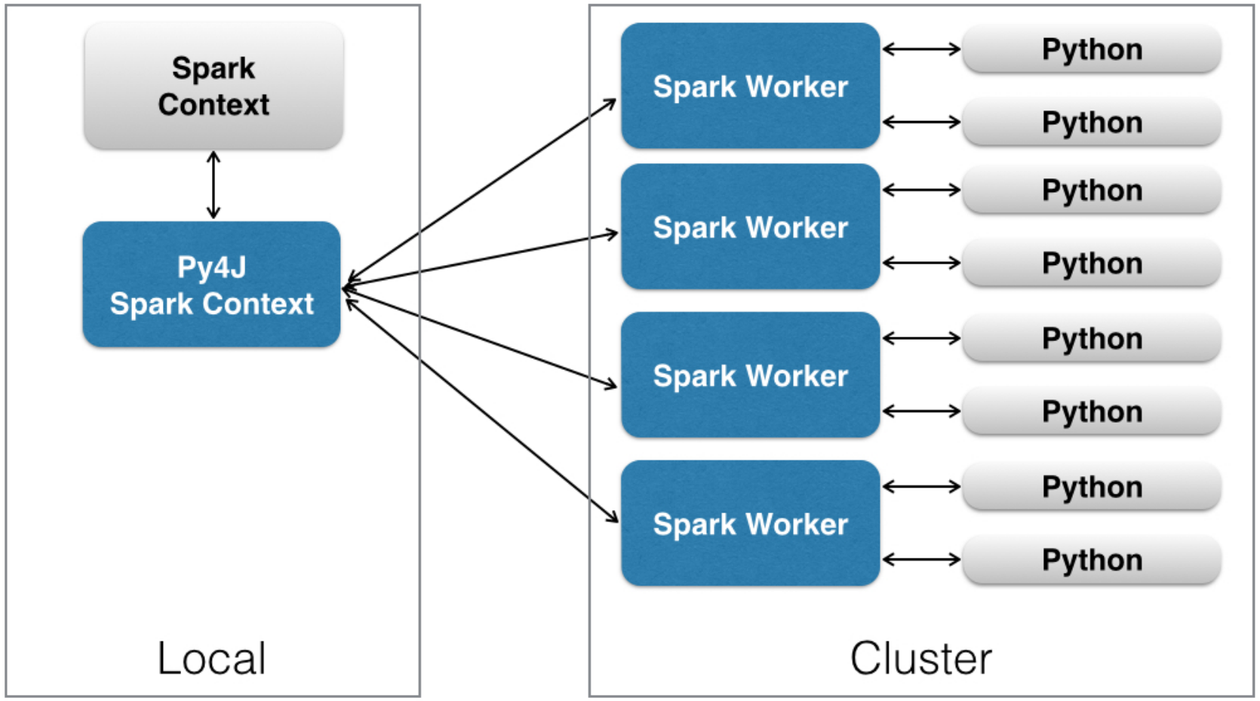
- set of **partitions**, atomic pieces of the dataset
- set of **dependencies** on parent RDD
- a funtion to compute dataset based on its parents
- metadata about the **partitioning schema** and the **data placement**.
- when possible calculation is done with respect to **data locality**
- data shuffle only when necessary

What ist PySpark

The Spark Python API (PySpark) exposes the Spark programming model to Python.

```
text_file = sc.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" ")) \
                  .map(lambda word: (word, 1)) \
                  .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```

Spark, Scala, the JVM & Python



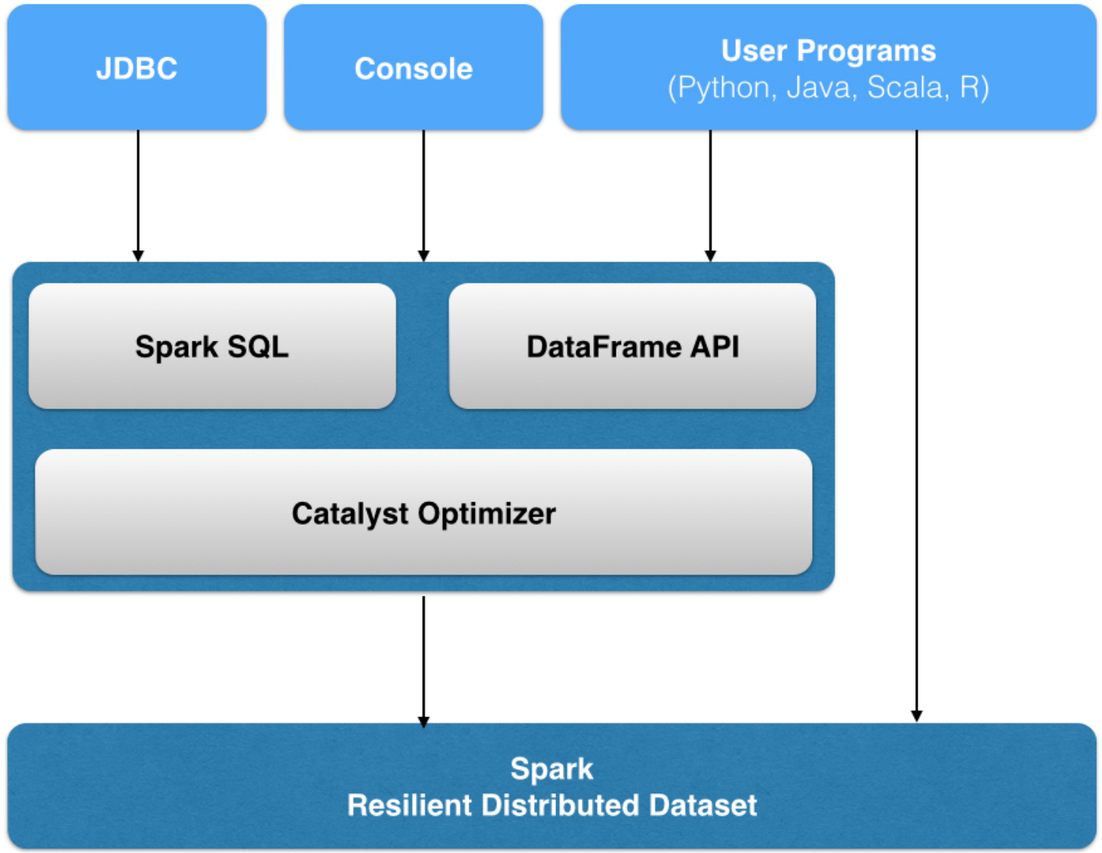
Relational Data Processing in Spark

Spark SQL is a part of Apache Spark that extends the funcional programming API with relational processing, **declarative queries** and optimized storage.

It provieds a programming abstraction called **DataFrames** and can also act as a distributed SQL query engine.

Tight integration between relational and procedual processing through a declarative DataFrame API. It includes catalyst, a highly extensible optimizer.

The DataFrame API can perform **relational operations** on external data soueces and Spark's built-in distributed collections.



blue yonder

DataFrame API

DataFrames are a distributed **collection of rows** gropued into named columns **with a schema**. High level api for common data processing tasks:

- project, filter, aggregation, join, metadata, sampling and user defined functions

As with RDDs, DataFrames are **lazy** in that each DataFrame object represents a **logical plan** to compute a dataset. It is not computed until an output operation is called.

DataFrame

A DataFrame is equivalent to a relational table in SparkSQL and can be created using vairous funcitons in the **SQLContext**

Once created it can be manipulated using the various **domain-specific-language** functions defined in DataFrame and Column.

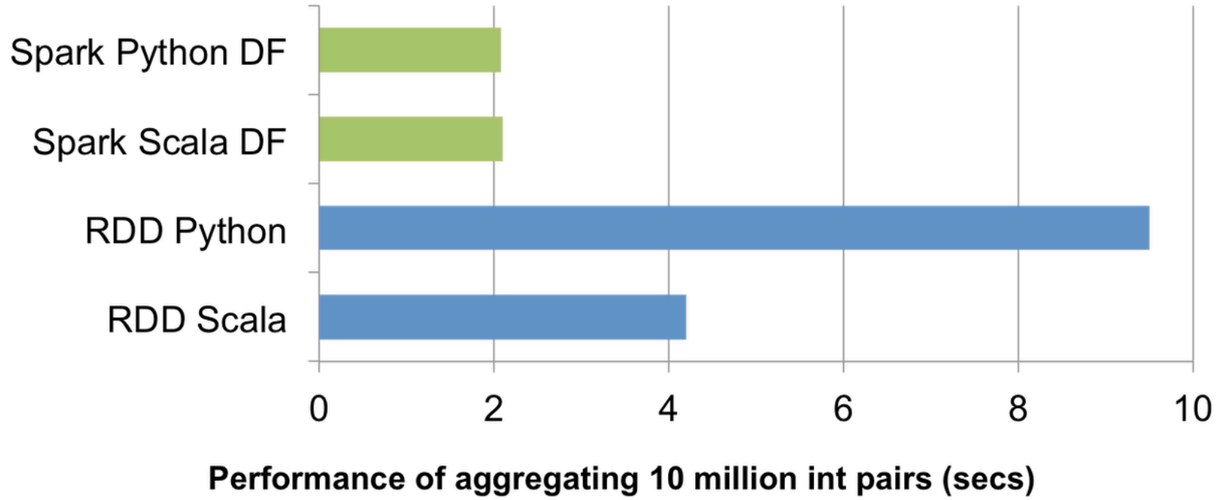
```
df = ctx.jsonFile("people.json")
df.filter(df.age >21).select(df.name, df.age +1)
ctx.sql("select name, age +1 from people where age > 21")
```

Catalyst

Catalyst is a **query optimization framework** embedded in Scala. Catalyst takes advantage of Scala's powerful language features such as **pattern matching** and runtime metaprogramming to allow developers to concisely specify complex relational optimizations

SQL Queries as well as queries specified through the declarative DataFrame API both go throug the same Query Optimizer which generates **JVM Bytecode**.

```
ctx.sql("select count(*) as anz from employees where gender = 'M'")
employees.where(employees.gender == "M").count()
```



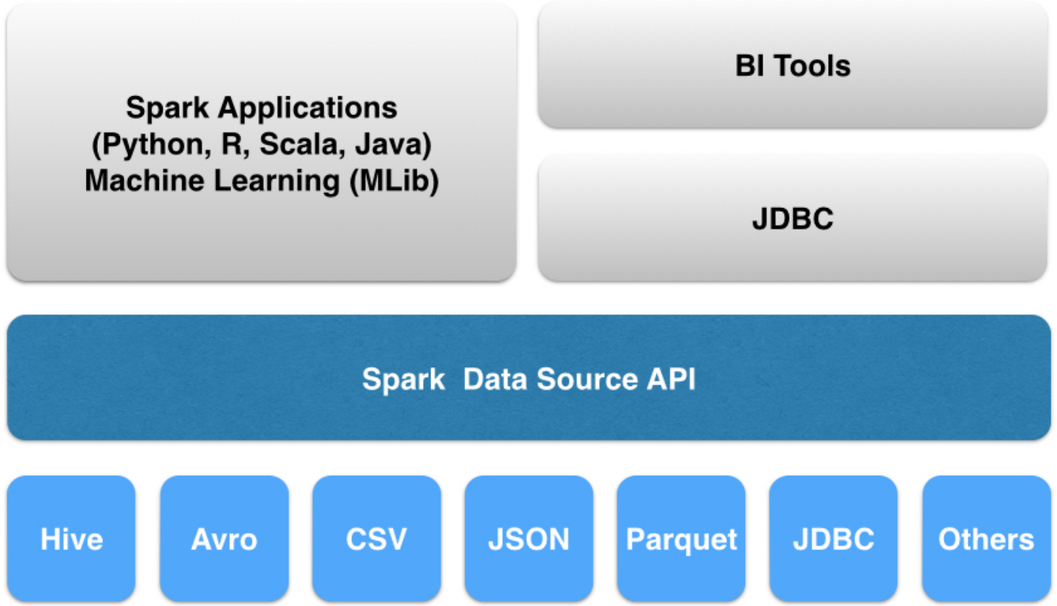
blue yonder

Data Source API

Spark can run in **Hadoop clusters** and access any Hadoop data source, RDDs on HDFS has a partition for each block for the file and knows on which machine each file is.

A DataFrame can be operated on as normal RDDs and can also be registered as a **temporary table** than they can be used in the sql context to query the data.

DataFrames can be accessed through Spark via an JDBC Driver.



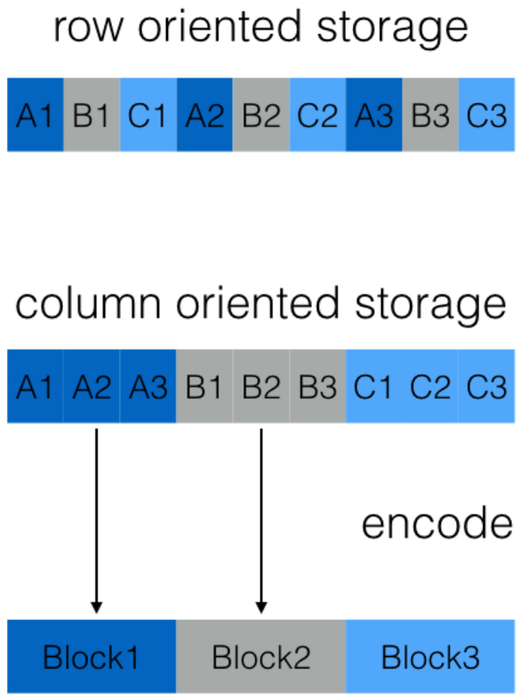
Data Input - Parquet

Parquet is a **columnar format** that is supported by many other data processing systems. Spark SQL provides support for both reading and writing Parquet files that automatically preserves the schema of the original data.

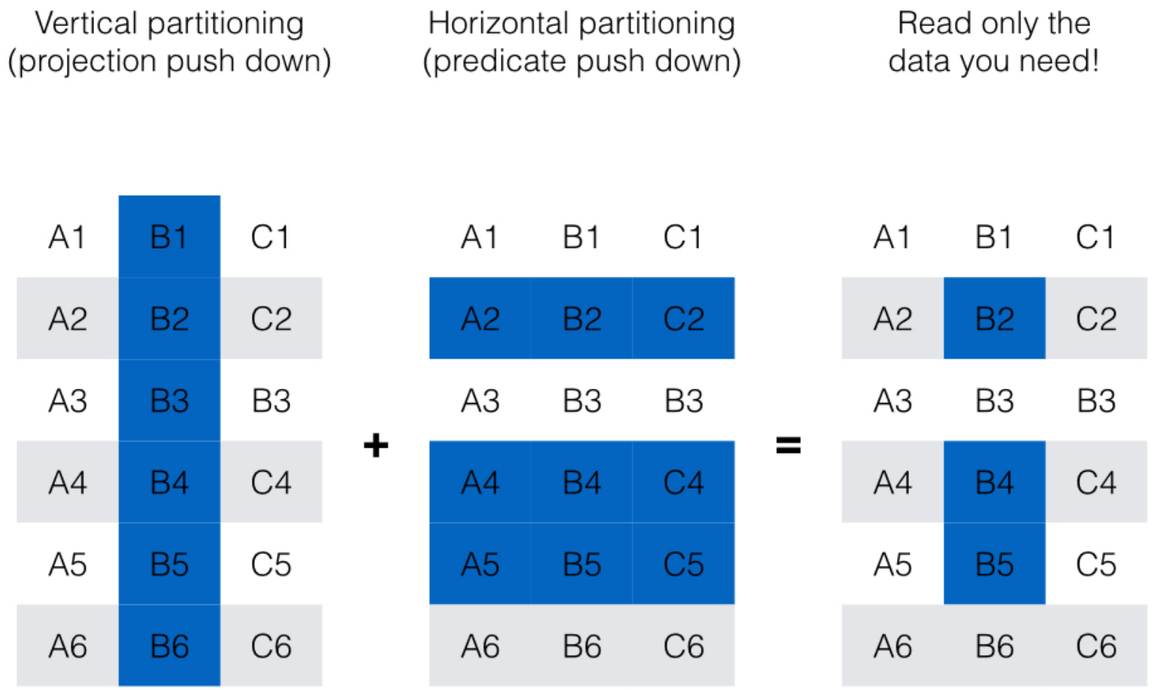
Parquet supports HDFS storage.

```
employees.saveAsParquetFile("people.parquet")

pf = sqlContext.parquetFile("people.parquet")
pf.registerTempTable("parquetFile")

long_timers = sqlContext.sql("SELECT name FROM parquetFile WHERE emp_no < 10050")
```

A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	B3



Projection & Predicate push down



Supported Data Types

- **Numeric Types** e.g. ByteType, IntegerType, FloatType
- **StringType**: Represents character string values
- **ByteType**: Represents byte sequence values
- **Datetime Type**: e.g TimestampType and DateType
- **ComplexTypes**
  - *ArrayType*: a sequence of items with the same type
  - *MapType*: a set of key-value pairs
  - *StructType*: Represents avalues with the structure described by a sequence of StructFields
  - *StructField*: Represents a field in a StructType

Schema Inference

The schema of a DataFrame can be **inferenced** from the data source. This works with typed input data like Avro, Parquet or JSON Files.

```
>>> l = [dict(name="Peter", id=1), dict(name="Felix", id=2)]
>>> df = sqlContext.createDataFrame(l)
>>> df.schema
... StructType(List(StructField(id, LongType, true),
                     StructField(name, StringType, true)))
```

Programmatically Specifying the Schema

For data sources without a schema definition you can programmatically specify the schema

```
employees_schema = StructType([
            StructField('emp_no',       IntegerType()),
            StructField('name',         StringType()),
            StructField('age',          IntegerType()),
            StructField('hire_date',    DateType()),
          ])


df = sqlContext.load(source="com.databricks.spark.csv", header="true",
                     path = filename, schema=employees_schema)
```

blueyonder

Important Classes of SparkSQL an DataFrames

- **SQLContext** Main entry point for DataFrame and SQL functionality
- **DataFrame** a distributed collection of data grouped into named columns
- **Column** a column expression in a DataFrame
- **Row** a row of data in a DataFrame
- **GroupedData** Agrregation methods, returned by DataFrame.groupBy()
- **types** List of data types available

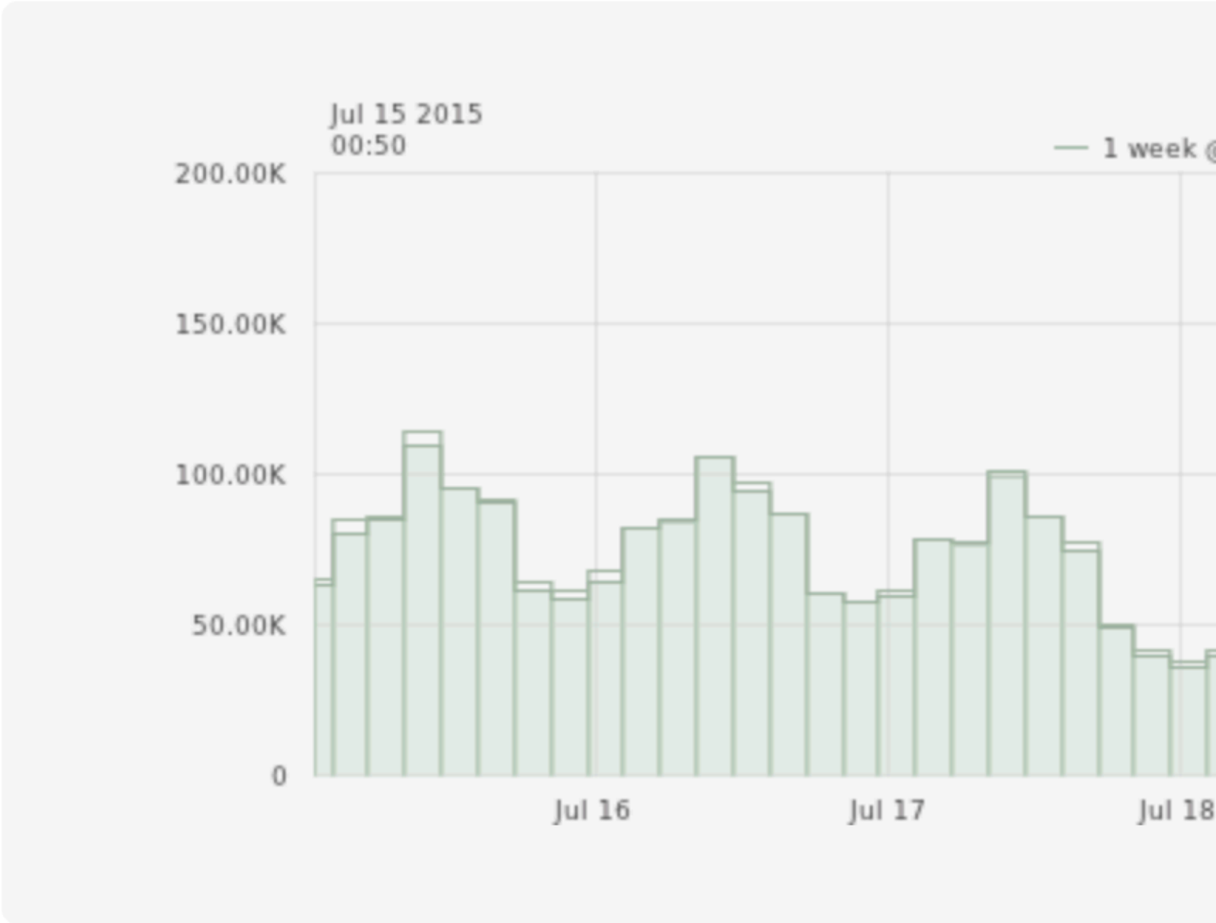
DataFrame Example

```
# Select everybody, but increment the age by 1
df.select(df['name'], df['age'] + 1).show()
## name    (age + 1)
## Michael null
## Andy    31
## Justin  20

# Select people older than 21
df.filter(df['age'] > 21).show()
## age name
## 30  Andy

# Count people by age
df.groupBy("age").count().show()
```

Demo GitHubArchive

GitHub Archive is a project to **record** the public GitHub timeline, **archive it**, and **make it easily accessible** for further analysis

- https://www.githubarchive.org
- **27GB** of JSON Data
- **70,183,530** events



Summary

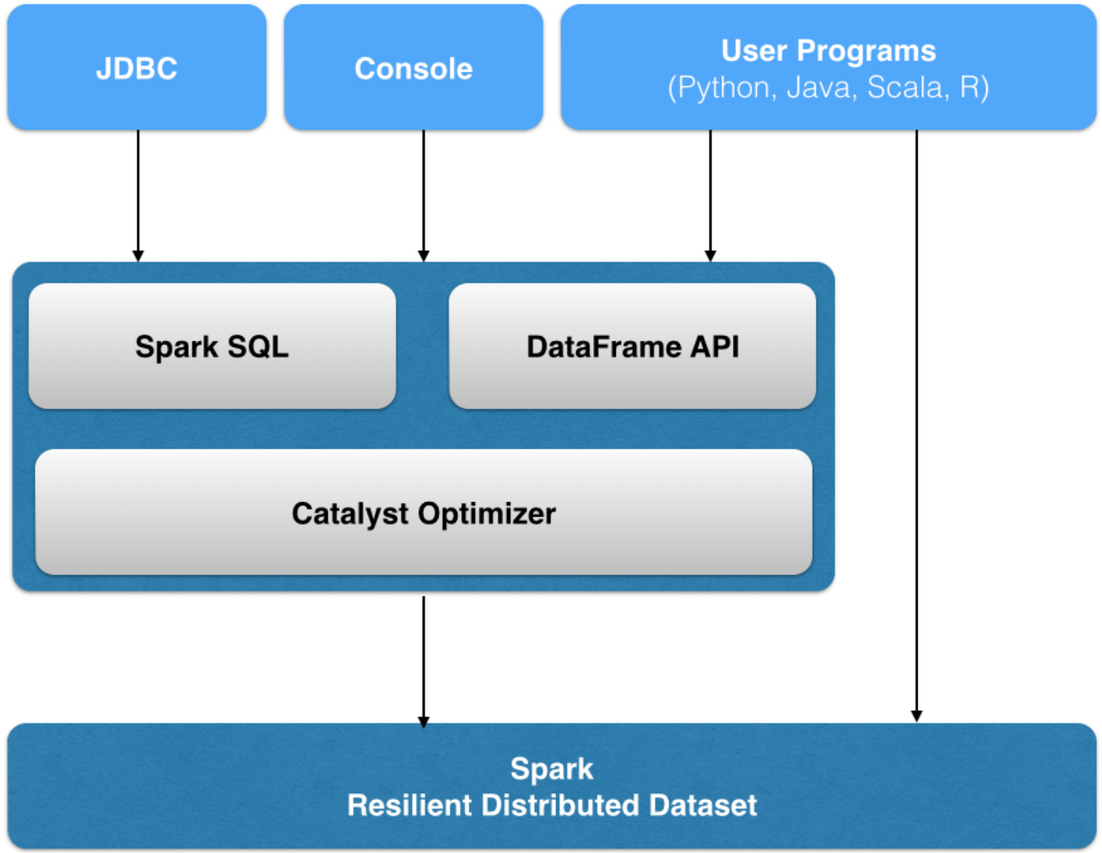
Spark implements a distributed general purpose cluster computation engine.

PySpark exposes the Spark Programming Model to Python.

Resilient Distributed Datasets represent a logical plan to compute a dataset.

DataFrames are a distributed collection of rows grouped into named columns with a schema.

DataFrame API allows maniplulation of DataFrames through a declarative domain specific language.




We love Big Data.
You love statistics.
Let’s get together >>
www.blue-yonder.com
blueyonder