Zurich University
of Applied Sciences

Information Engineering 2

Introduction to Spark

Prof. Dr. Kurt Stockinger

Semesterplan

Zurich University of Applied Sciences

SW	Datum	Vorlesungsthema	Praktikum
1	23.02.2022	Data Warehousing Einführung	Praktikum 1: KNIME Tutorial
2	02.03.2022	Dimensionale Datenmodellierung 1	Praktikum 1: KNIME Tutorial (Vertiefung)
3	09.03.2022	Dimensionale Datenmodellierung 2	Praktikum 2: Datenmodellierung
4	16.03.2022	Datenqualität und Data Matching	Praktikum 3: Star-Schema, Bonus: Praktikum 4: Slowly Changing Dimensions
5	23.03.2022	Big Data Einführung	DWH Projekt - Teil 1
6	30.03.2022	Spark - Data Frames	DWH Projekt - Teil 2 (Abgabe: 4.4.2022 23:59:59)
7	06.04.2022	Data Storage: Hadoop Distributed File System & Parquet	Praktikum 1: Data Frames
8	13.04.2022	Query Optimization	Praktikum 2: Data Storage
9	20.04.2022	Spark Best Practices & Applications	Praktikum 3: Query Optimization & Performance Analysis
10	27.04.2022	Machine Learning mit Spark 1	Praktikum 3: Query Optimization & Performance Analysis (Vertiefung)
11	04.05.2022	Machine Learning mit Spark 2 + Q&A	Praktikum 4: Machine Learning (Regression)
12	11.05.2022	NoSQL Systems	Big Data Projekt - Teil 1
13	18.05.2022	Keine Vorlesung (Arbeit am Projekt)	Big Data Projekt - Teil 2
14	25.05.2022	Keine Vorlesung (Arbeit am Projekt)	Big Data Projekt - Teil 3 (Abgabe: 30.5.2022 23:59:59)

Educational Objectives for Today

- Know the main concepts of Apache Spark
- Understand DataFrames and major algorithms
- Implement various Spark examples

Zurich University
of Applied Sciences

Literature

- edX-Courses:

Introduction to Big Data with Apache Spark

Learn how to apply data science techniques using parallel programming in Apache Spark to explore big (and small) data.

Berkeley
UNIVERSITY OF CALIFORNIA

Scalable Machine Learning

Learn the underlying principles required to develop scalable machine learning pipelines and gain hands-on experience using Apache Spark.

Berkeley
UNIVERSITY OF CALIFORNIA

- Papers & Books:

Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica
University of California, Berkeley

NSDI 2012

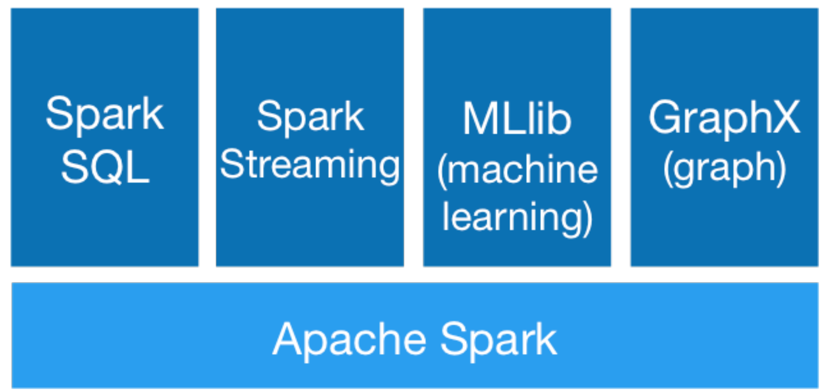
Scaling Spark in the Real World: Performance and Usability

Michael Armbrust, Tathagata Das, Aaron Davidson, Ali Ghodsi, Andrew Or, Josh Rosen, Ion Stoica, Patrick Wendell, Reynold Xin, Matei Zaharia[†]
Databricks Inc. [†]MIT CSAIL

VLDB 2015

What is Apache Spark?

- General purpose cluster computing system
- Originally developed at UC Berkeley, now one of the largest Apache projects
- Typically faster than Hadoop due to main-memory processing
- High-level APIs in Java, Scala, Python and R
- Functionality for:
  - Map/Reduce
  - SQL processing
  - Real-time stream processing
  - Machine learning
  - Graph processing

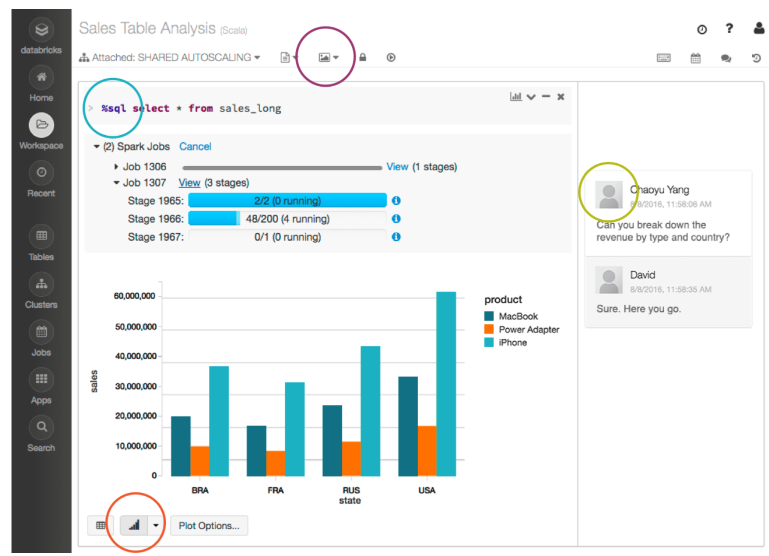


What is Databricks ?

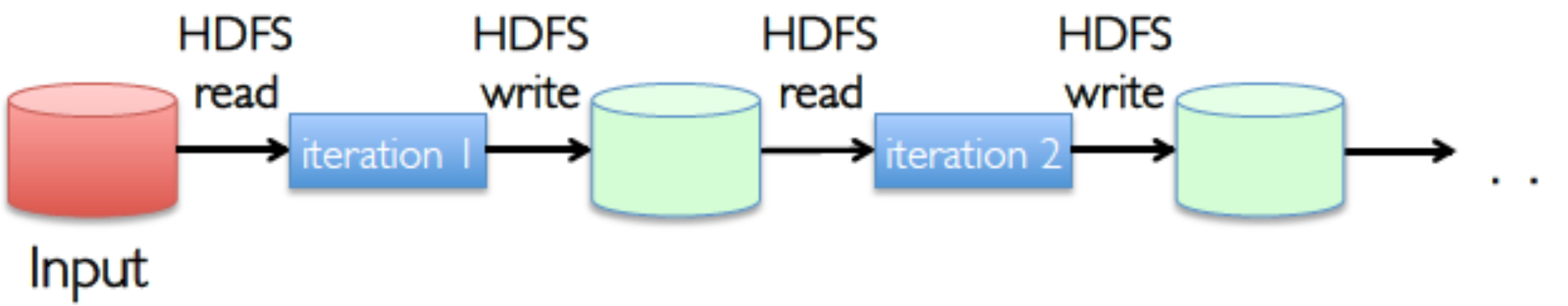
- Berkeley Spin-Off company, founded mainly by the Apache Spark creators
- Employing sizable part of main Spark open-source contributors
- Offering Cloud-based hosted Spark notebook: "Databricks Cloud"
    - Adds a GUI to Apache Spark and automates cluster management
    - Completely web-based
    - Backed by Amazon EC2
    - **Free educative offer: *Databricks community***
    - Mature, up-to-date solution

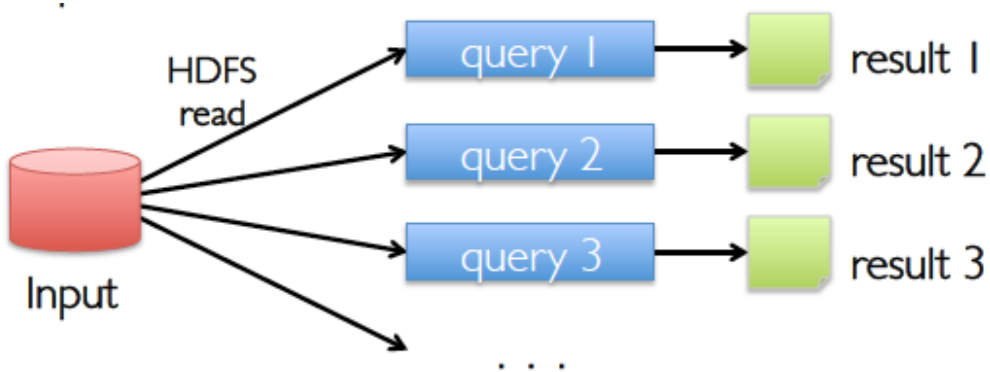
But:

As a US cloud offer, not suitable for anybody in a regulated area or concerned with data privacy
We will use *Databricks community* for the lab sessions



Iterative Processing with Hadoop or RDBMS





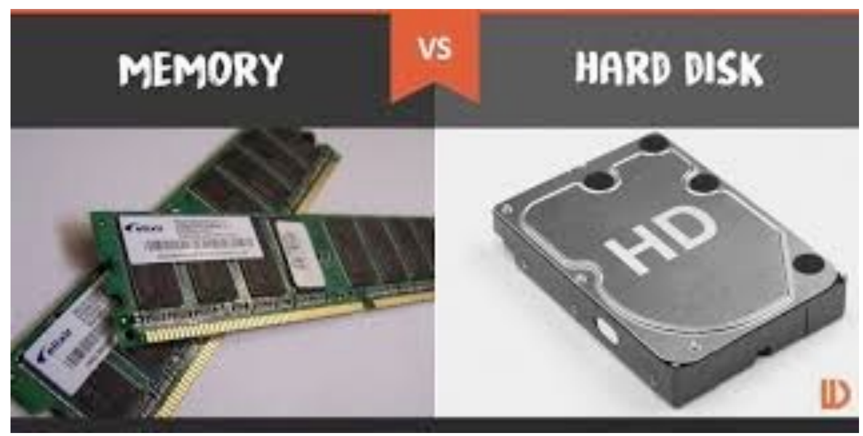
What is wrong with this approach?

HDFS… Hadoop Distributed File System

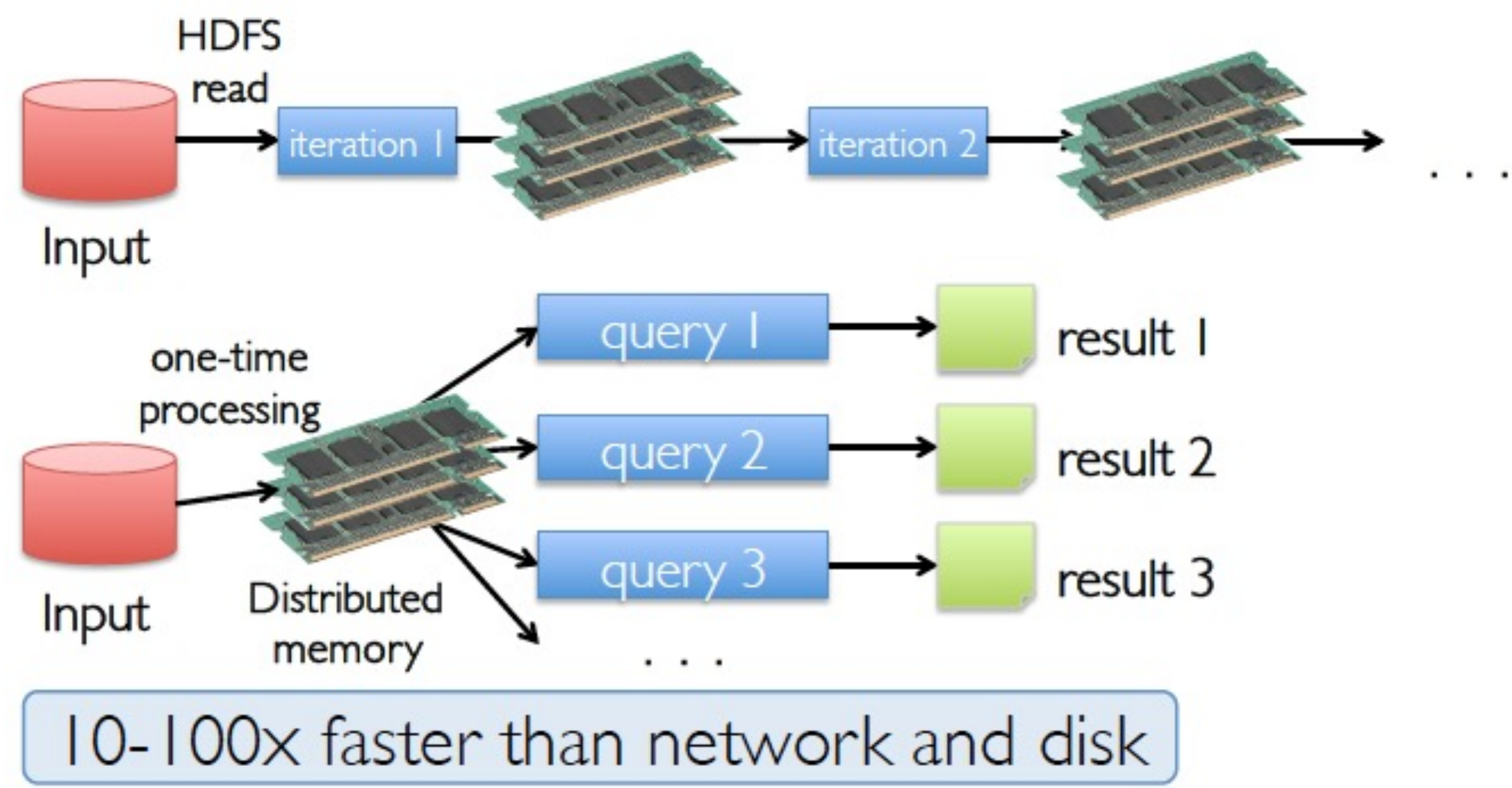
Throughput of Main Memory vs. Disk

- Typical throuput of disk: ~ 100 MB/sec
- Typical throuput of main memory: 50 GB/sec

=> Main memory is ~ 500 times faster than disk

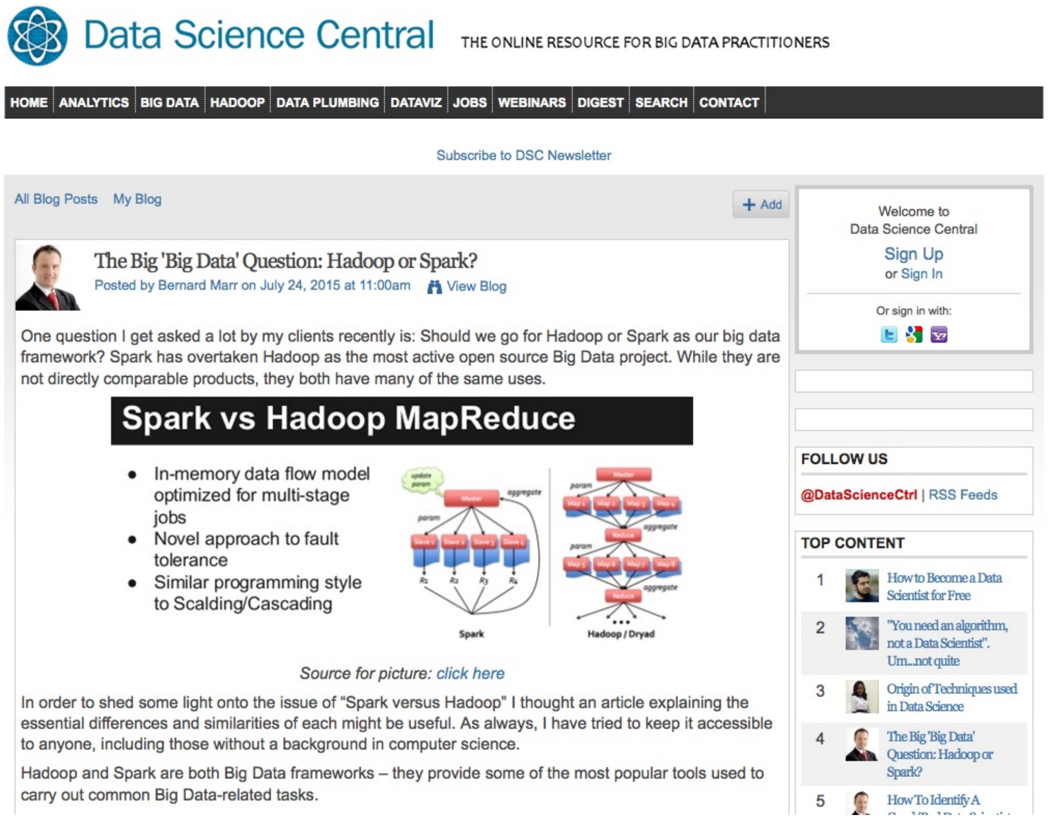


Apache Spark Approach: In-Memory Data Sharing



(from Matei Zaharia 2012, UC Berkeley)

Spark vs. Hadoop #1



Data Science Central THE ONLINE RESOURCE FOR BIG DATA PRACTITIONERS

HOME | ANALYTICS | BIG DATA | HADOOP | DATA PLUMBING | DATAVIZ | JOBS | WEBINARS | DIGEST | SEARCH | CONTACT

Subscribe to DSC Newsletter

All Blog Posts My Blog + Add

The Big 'Big Data' Question: Hadoop or Spark?

Posted by Bernard Marr on July 24, 2015 at 11:00am View Blog

One question I get asked a lot by my clients recently is: Should we go for Hadoop or Spark as our big data framework? Spark has overtaken Hadoop as the most active open source Big Data project. While they are not directly comparable products, they both have many of the same uses.

Source for picture: click here

In order to shed some light onto the issue of "Spark versus Hadoop" I thought an article explaining the essential differences and similarities of each might be useful. As always, I have tried to keep it accessible to anyone, including those without a background in computer science.

Hadoop and Spark are both Big Data frameworks – they provide some of the most popular tools used to carry out common Big Data-related tasks.

Welcome to
Data Science Central
Sign Up
or Sign In

Or sign in with:

FOLLOW US

@DataScienceCtrl | RSS Feeds

TOP CONTENT

1 How to Become a Data Scientist for Free

2 "You need an algorithm, not a Data Scientist". Um...not quite

3 Origin of Techniques used in Data Science

4 The Big 'Big Data' Question: Hadoop or Spark?

5 How To Identify A

http://www.datasciencecentral.com/profiles/blogs/the-big-big-data-question-hadoop-or-spark

Spark vs. Hadoop #2

	Hadoop Map Reduce	Spark
Storage	Disk only	In-memory or on disk
Operations	Map and Reduce	Map, Reduce, Join, Sample, etc…
Execution model	Batch	Batch, interactive, streaming
Programming environments	Java	Scala, Java, R, and Python

(from Ameet Talwalkar, UCLA, 2015)

Spark Usage

Who: Typically industries with large amounts of data, rapid growth and accepting a certain margin for errors:

- Advertising
- Telco
- Retail
- Research

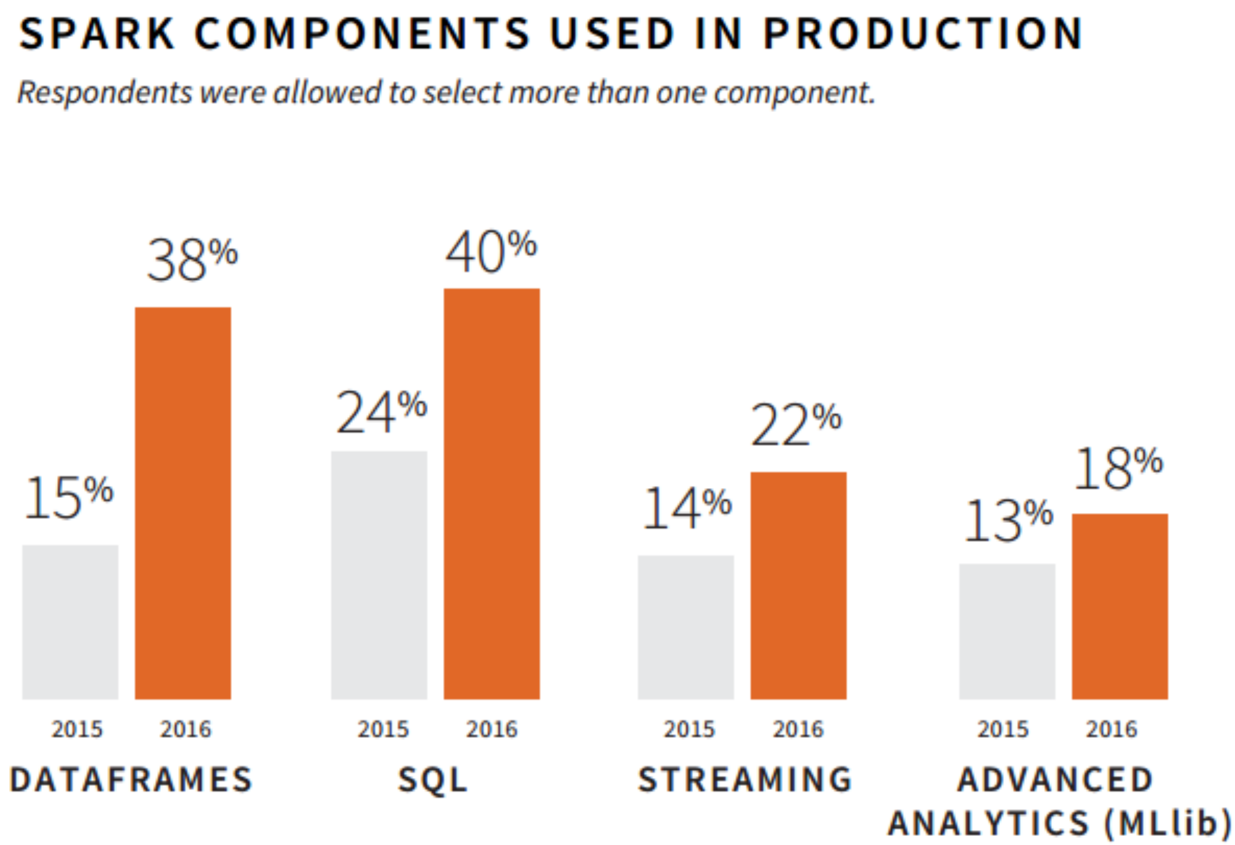
+ Potentially all Hadoop users, as Spark is part of every major Hadoop distribution

What for:

- Business Intelligence: 68%
- DWH – Ingestion, Processing: 52%
- Streaming/RealTime: 45%
- Recommendation Engines: 40%
- Log Processing: 37%

DataBricks Survey 2016 -http://go.databricks.com/2016-spark-survey

Components used in Production - 2016



Source: https://databricks.com/blog/2016/09/27/spark-survey-2016-released.html

Spark integrated with Advanced Analytics

Commercial Advanced Analytics Solutions started using Spark:

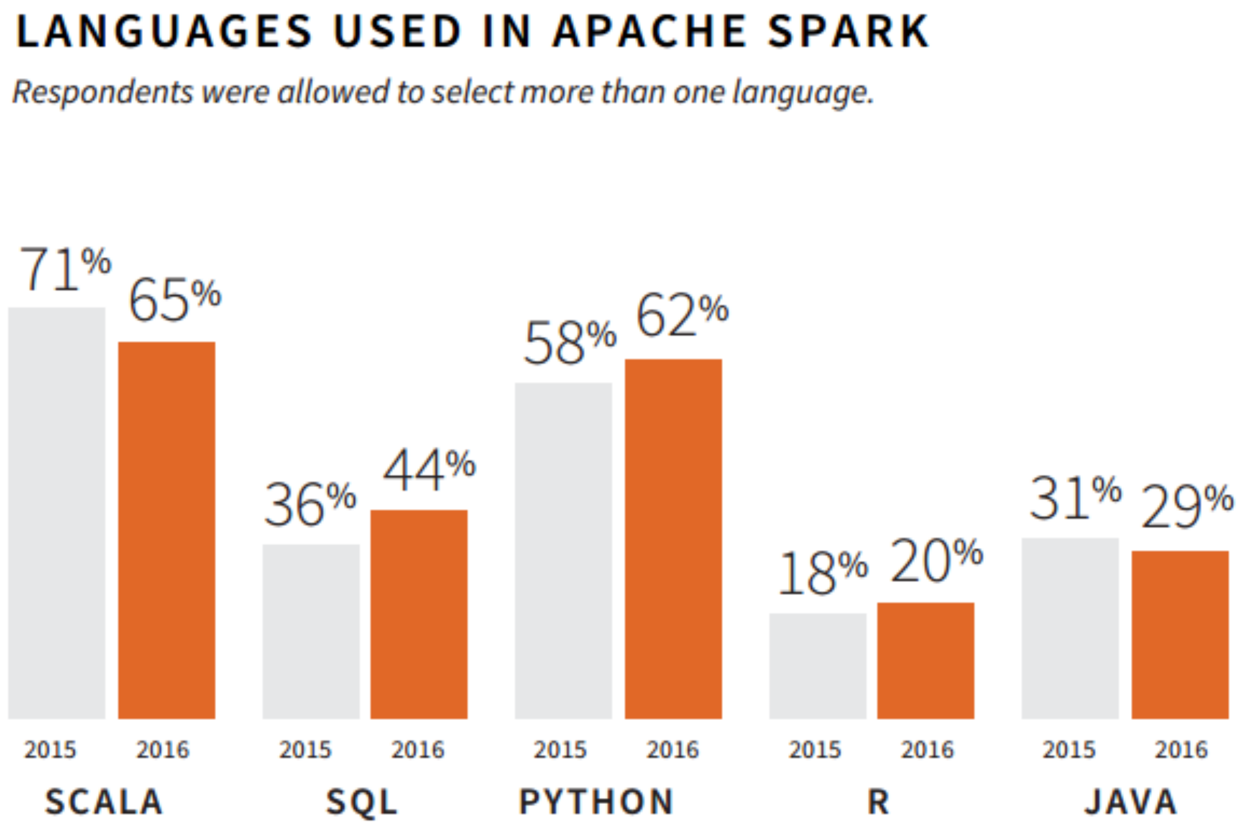
- Data movement interface for proprietary engine
- Cluster-execution engine for platform's proprietary analytics code
- Interface for integration of custom Spark code into workflows
- Repackaged and integrated as a whole in proprietary solution/UI

Explicitly permitted by Apache Licence

Source: Gartner *"Magic Quadrant"* for Advanced Analytics Platforms, 2016

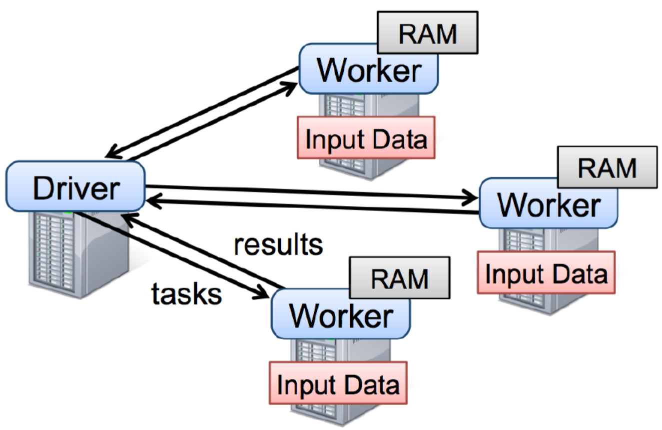
Source: https://thomaswdinsmore.com/2017/02/14/spark-is-the-future-of-analytics/

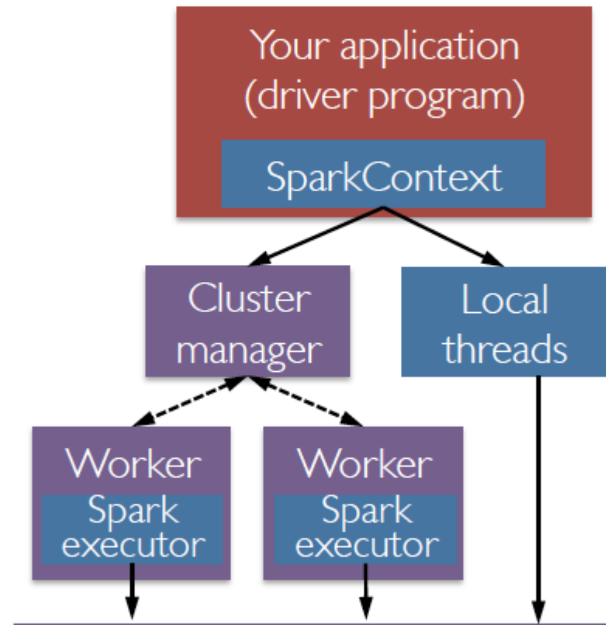
Languages used in Production



Spark Runtime

- Driver:
  - Application developer writes driver program
  - Driver connects to cluster of workers
- Workers:
  - Read data blocks from distributed file system
  - Process and store data partitions in RAM across operations





Local storage, HDFS, Amazon S3,…

Zurich University
of Applied Sciences

Usage of Spark

- Interactive analysis with Apache shell (pyspark)
  - Python programming interface to Spark
  - Parallel runtime
  - pyspark

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.4.0
      /_/

Using Python version 2.7.5 (default, Mar  9 2014 22:15:05)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
```

- Running self-contained applications (spark-submit)
  - spark-submit MyApplication.py
- Via Jupyter Notebook on Databricks

Spark and SQL Context

- Spark program creates a SparkContext object:
  - SparkContext tells Spark how and where to access a cluster
  - pySpark and Databricks CE automatically create SparkContext
  - iPython and programs must create a new SparkContext

- Afterwards a sqlContext object is created

- Use sqlContext to create DataFrames

DataFrames behave like Tables

- DataFrames are the primary abstraction in Spark ("almost" like tables):
  - The are immutable (can't be changed!)
  - Track lineage information to efficiently recompute lost data
  - Enable operations on collection of elements in parallel

- DataFrames can be constructed:
  - By parallelizing existing Python collections (lists)
  - By transforming an existing Spark or pandas DataFrame
  - From files in HDFS or any other storage system

DataFrame Example

- Each row of a DataFrame is a Row object
- Fields in a Row can be accessed like attributes

```
>>> row = Row(name="Alice", age=11)
>>> row
Row(age=11, name='Alice')
>>> row['name'], row['age']
('Alice', 11)
>>> row.name, row.age
('Alice', 11)
```

Creating DataFrames

Create DataFrames from Python collections (lists)

```
>>> data = [('Alice', 1), ('Bob', 2)]
>>> data
[('Alice', 1), ('Bob', 2)]
>>> df = sqlContext.createDataFrame(data)
[Row(_1=u'Alice', _2=1), Row(_1=u'Bob', _2=2)]
>>> sqlContext.createDataFrame(data, ['name', 'age'])
[Row(name=u'Alice', age=1), Row(name=u'Bob', age=2)]
```

No computation occurs with `sqlContext.createDataFrame()`

- Spark only records how to create the DataFrame

When to Use Data Frames?

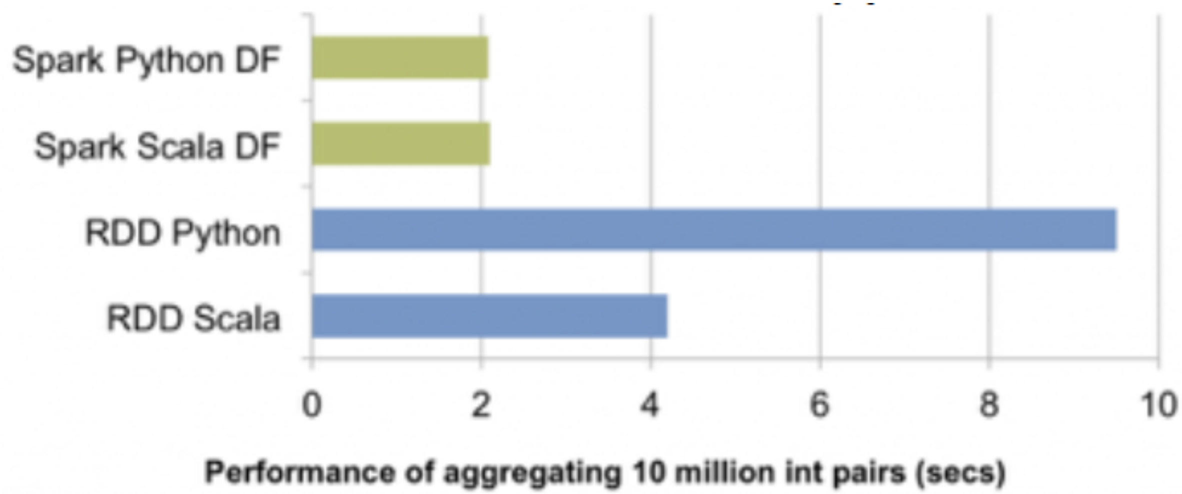
- Spark has three different major abstractions:
  - RDD (resilient distributed data set):
    - low level: e.g. map reduce functionality
  - Data Frame:
    - high level (sits on top of RDD): e.g. SQL functionality
  - Dataset:
    - Strongly typed JVM object

Language	Main Abstraction
Scala	Dataset[T] & DataFrame (alias for Dataset[Row])
Java	Dataset[T]
Python*	DataFrame
R*	DataFrame

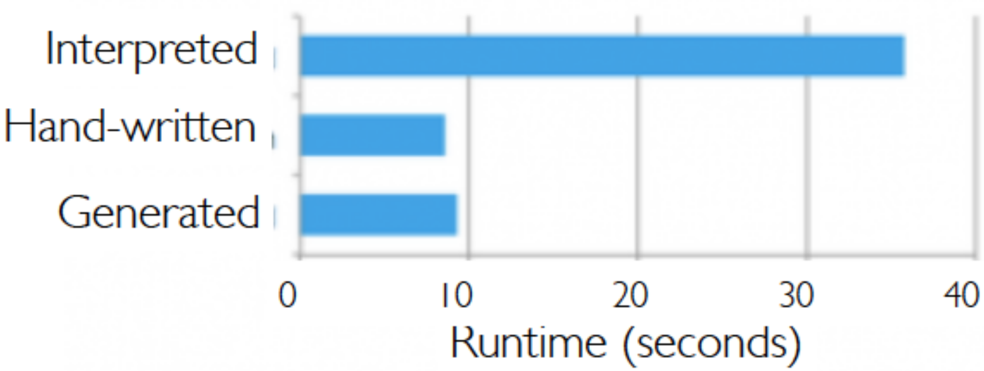
- Benefits of Data Frames:
  - High-level transformations and actions
  - Typed data (structured or semi-structured)
  - Performance optimizations:
    - Catalyst Optimization Engine (query optimization)
    - Project Tungsten (optimized off-heap memory management)

Performance of DataFrames vs. RDDs

Faster than RDDs



Benefits from Catalyst optimizer



Data Frame Example #1

Create a DataFrame from a Python collection

- Create Python object:

```
>>> data = [('Alice',1),('Bob',2)]
```

- Show content

```
>>> data

[('Alice', 1), ('Bob', 2)]
```

- Create data frame:

```
>>> df = sqlContext.createDataFrame(data)
```

- Show content

```
>>> df
DataFrame[_1: string, _2: bigint]
```

Data Frame Examples #2

- Show stucture

>>> df.show()

```
+-----+---+
|   _1| _2|
+-----+---+
|Alice|  1|
|  Bob|  2|
+-----+---+
```

- Register DataFrame with explicit column names

>>> df1 = sqlContext.createDataFrame(data, ['name','age'])

>>> df1.show()

```
+-----+---+
| name|age|
+-----+---+
|Alice|  1|
|  Bob|  2|
+-----+---+
```

Data Frame Examples #3

- Create temporary table:

```
>>>df1.registerTempTable("t1")
```

- Execute SQL statement:

```
>>> res = sqlContext.sql("select * from t1")
```

- Show results:

```
>>> res.show()
+-----+---+
| name|age|
+-----+---+
|Alice|  1|
|  Bob|  2|
+-----+---+
```

Data Frame Examples #4

- More complex SQL query

```
>>> res = sqlContext.sql("select * from t1 where age=2")
```

- Show result

```
>>> res.show()
+----+---+
|name|age|
+----+---+
| Bob|  2|
+----+---+
```

Data Frame Examples #5

Create a DataFrame from a text file

Name,Age,City
Luana,5,Zurich
Peter,45,Genf
Laura,24,Genf
Hans,79,Zurich
Sarah,38,Zurich

- Load data:

```
>>> people = sqlContext.read.format("com.databricks.spark.csv")\
  .option("header","true")\
  .option("inferSchema", "true")\
  .load(”data_people.txt")
```

Data Frame Examples #6

- Print schema:

```
>>> people.printSchema()
root
 |-- Name: string (nullable = true)
 |-- Age: double (nullable = true)
 |-- City: string (nullable = true)
```

- Show content:

```
>>> people.show()
+-----+----+-------+
| Name| Age|   City|
+-----+----+-------+
|Luana| 5.0| Zurich|
|Peter|45.0|   Genf|
|Laura|24.0|   Genf|
| Hans|79.0| Zurich|
|Sarah|38.0| Zurich|
+-----+----+-------+
```

Data Frame Examples #7

- Register temp table:

```
>>> people.registerTempTable("peopleT")
```

- Queries: Find people in Zurich

```
>>> res2 = sqlContext.sql("select * from peopleT
                           where City='Zurich'")

+-----+----+-------+
| Name| Age|   City|
+-----+----+-------+
|Luana| 5.0| Zurich|
| Hans|79.0| Zurich|
|Sarah|38.0| Zurich|
+-----+----+-------+
```

Useful Transformations

Transformation	Description
`filter(func)`	returns a new **`DataFrame`** formed by selecting those rows of the source on which *func* returns true
`where(func)`	**`where`** is an alias for **`filter`**
`distinct()`	return a new **`DataFrame`** that contains the distinct rows of the source **`DataFrame`**
`orderBy(*cols, **kw)`	returns a new **`DataFrame`** sorted by the specified *column(s)* and in the sort order specified by *kw*
`sort(*cols, **kw)`	Like **`orderBy`**, **`sort`** returns a new **`DataFrame`** sorted by the specified *column(s)* and in the sort order specified by *kw*
`explode(col)`	returns a new row for each element in the given array or map

`func` is a Python named function or **`lambda`** function

Data Frame API #1

- Directly using DataFrame-API

```
>>>people.select("Name", "City").show()
```

```
+-----+-------+
| Name|   City|
+-----+-------+
|Luana| Zurich|
|Peter|   Genf|
|Laura|   Genf|
| Hans| Zurich|
|Sarah| Zurich|
+-----+-------+
```

Data Frame API #2

```
>>> people.filter(people['Age'] > 10).show()

+-----+----+-------+
| Name| Age|   City|
+-----+----+-------+
|Peter|45.0|   Genf|
|Laura|24.0|   Genf|
| Hans|79.0| Zurich|
|Sarah|38.0| Zurich|
+-----+----+-------+
```

Some Useful Actions

show(*n*, *truncate*)	prints the first *n* rows of the **DataFrame**
take(*n*)	returns the first *n* rows as a list of **Row**
collect()	return all the records as a list of **Row** WARNING: make sure will fit in driver program
count()+	returns the number of rows in this **DataFrame**
describe(**cols*)	Exploratory Data Analysis function that computes statistics (count, mean, stddev, min, max) for numeric columns – if no columns are given, this function computes statistics for all numerical columns

Data Frame API

- Count number of people

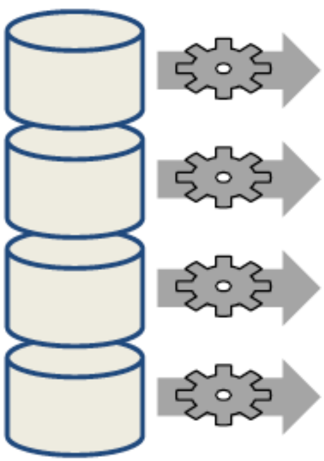
```
>>> people.count()

5
```

- Show first 3

```
>>>people.take(3)

[Row(Name=u'Luana', Age=5.0, City=u' Zurich'),
 Row(Name=u'Peter', Age=45.0, City=u' Genf'),
 Row(Name=u'Laura', Age=24.0, City=u' Genf')]
```

Lazy Evaluation

```
distFile = sqlContext.read.text ("...")
```



Loads text file and returns a DataFrame with a single string column named "value"

Each line in text file is a row

Lazy evaluation means no execution happens now

Working with JSON-Files

- Assume we have the following JSON-file called “people.json”

```
{"name":"Michael"}
{"name":"Andy", "age":30}
{"name":"Justin", "age":19}
```

- Create a DataFrame based on the JSON-file:

```
df = spark.read.json(“people.json")
```

- Displays the content of the DataFrame to stdout

```
df.show()
+----+-------+
| age|   name|
+----+-------+
|null|Michael|
|  30|   Andy|
|  19| Justin|
+----+-------+
```

Saving DataFrame to File

- Save names into a binary (parquet) file

```
>>> df = spark.read.json(“people.json")
>>> df1 = df.select("name")
>>> df1.write.save("/tmp/people_names")
```

Output:

/tmp/people_names/
  _SUCCESS
  part-r-00000-9cad4663-0001-4069-a61b-4422b472c6e1.snappy.parquet

PAR1^U^@^UF^UJ,^U^F^U^@^U^F^U^H^\^X^GMichael^X^DAndy^V^@^@^@^@#<88>^B^@^@^@^C^G
^G^@^@^@Michael^D^@^@^@Andy^F^@^@^@Justin^U^B^Y,H^Lspark_schema^U^B^@^U^L%^B^X^Dname%^@^@^V^F^Y^\^Y^\^\^U^L^
Y5^@^F^H^Y^X^Dname^U^B^V^F^V<8E>^A^V<92>^A<^X^GMichael^X^DAndy^V^@^@^@^@^V<8E>^A^V^F^@^Y^\^X)org.apache.spark.sql
.parquet.row.metadata^XZ{"type":"struct","fields":[{"name":"name","type":"string","nullable":true,"metadata":{}}]}^@^X;parquet-mr (build
32c46643845ea8a705c35d4ec8fc654cc8ff816d)
^@(^A^@^@PAR1

Saving DataFrame to JSON

```
>>> df1.write.save("/tmp/people_names1", format="json")
```

Output:

```
/tmp/people_names1/
  _SUCCESS
  part-r-00000-9ce54d20-d299-4ad1-970a-499439d2056c.json
```

```
{"name":"Michael"}
{"name":"Andy"}
{"name":"Justin"}
```

Saving DataFrame to CSV

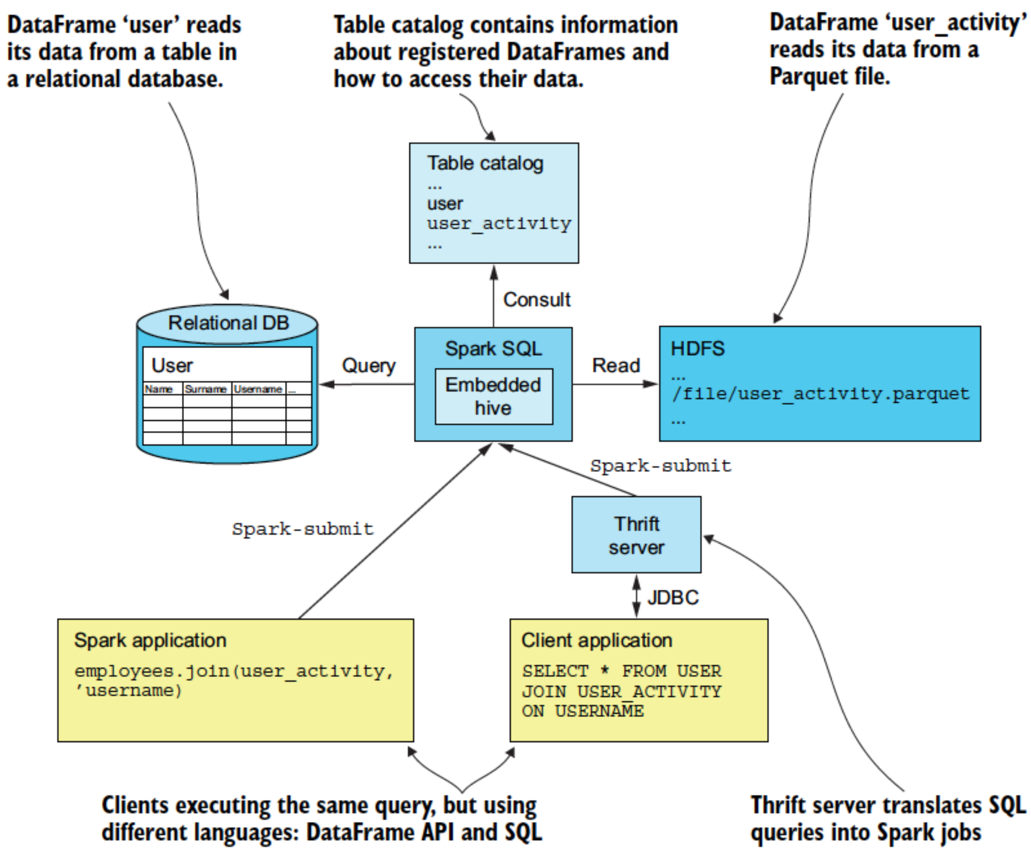
```
>>> df1.write.save("/tmp/people_names2", format="csv")
```

Output:

```
/tmp/people_names2/
  _SUCCESS
  part-r-00000-014c0d2f-4b9d-4bf7-9e3a-361922ae41f5.csv
```

Michael
Andy
Justin

Using Spark to Access different Data Sources



Access to Spark from 3rd Party Applications

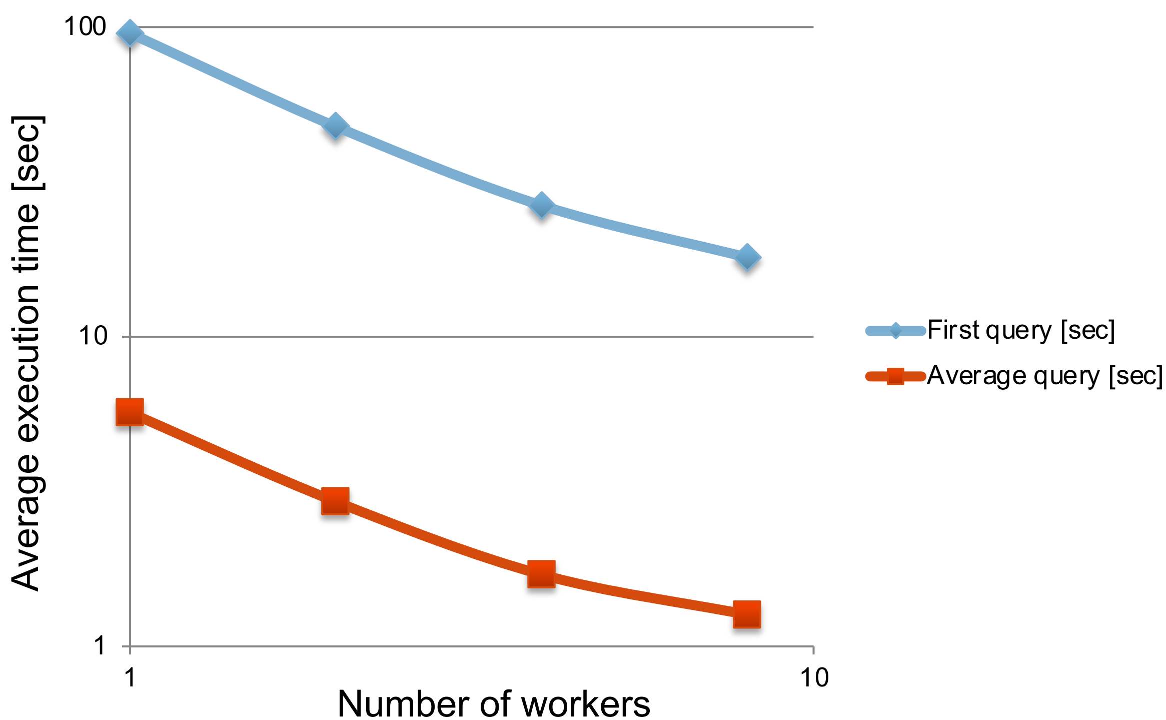
- Applications can use standard JDBC or ODBC protocols to connect to Spark
- Use SQL to query data from registered DataFrames tables

Performance of SQL Range Queries on Amazon Web Services (AWS Cloud) #1

Range queries on 1 GB text file with 10 columns

```
from pyspark.sql import SparkSession
import time

# creat SparkSession
spark = SparkSession.builder.appName("PythonSQL").config("spark.some.config.option","some-value").getOrCreate()

#create DataFrame
df = spark.read.csv("tabularFile1GB.txt")


# cached SQL
df.cache()
print "\n Cache DataFrame df \n"

for i in range(numIter):
        print i
        start = time.time()
        df1 = spark.sql("select count(*) from t1 where _c3 < 50")
        df1.count()
        stop = time.time()
        print "spark.sql - count: ", df1.show()
        print "Time: ", (stop-start), " seconds."
```

Performance of SQL Range Queries on Amazon Web Services (AWS Cloud) #2



Almost linear scalability up to 8 worker nodes

Conclusions

- Spark significantly simplifies parallel computing
- Easier programming model than MapReduce
- Parallel SQL on big data