A Practical Introduction to BIG DATA

Alvaro del Castillo San Félix (alvarodelcastillo@gmail.com)

21st June, esLibre 2019, Granada

https://github.com/aylabs/bigdata-practical-intro

v1.2

GitHub:
https://github.com/acs/

LinkedIn:
https://www.linkedin.com/in/acslinkedin/

Twitter:
https://twitter.com/acstw

Email:
alvaro.delcastillo@gmail.com

/me: Alvaro del Castillo

Open Source believer and coder!

Cofounder of Barrapunto and Bitergia

Working in different roles in Tech companies

Now in **Paradigma Digital** as Software Architect filling the BBVA Data Lake in the Transcende project. **We are hiring!**

In the process of surfing the Machine Learning wave

Summary

https://www.pexels.com/photo/notebook-1226398/

Data

Big

Practical

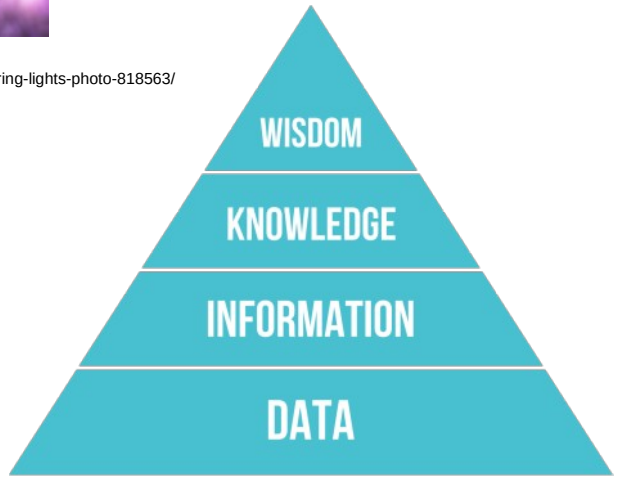
Open Source

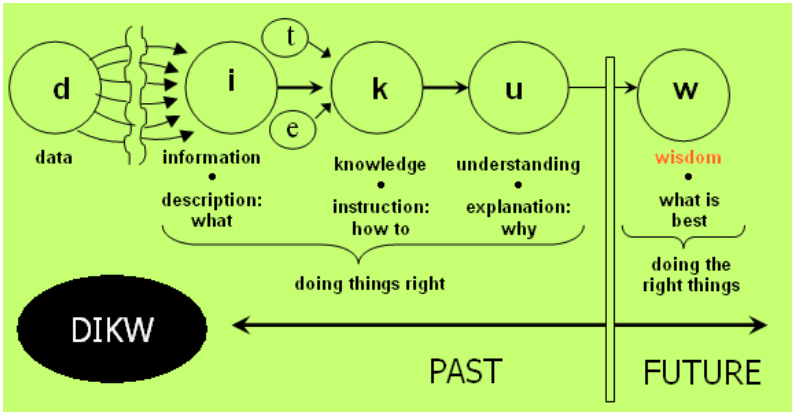
https://www.pexels.com/photo/person-holding-string-lights-photo-818563/

Data

https://en.wikipedia.org/wiki/DIKW_pyramid

"Typically information is defined in terms of data, knowledge in terms of information, and wisdom in terms of knowledge"





By Longlivetheux - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=37705247

https://www.pexels.com/photo/bandwidth-close-up-computer-connection-1148820/

Big

The 3 Vs of Big Data:

- **Volume**: Scalability (up to the infinite)
- **Velocity**: Performance
- **Variety**: Flexibility

2 more Vs:

- **Veracity**: data quality
- **Value**: from data to business

Extra:

- Clouds of **cheap** commodity computers

Big Data Architectures

Batch

Streaming

Lambda Architecture: Batch + Streaming in a single platform

https://www.pexels.com/photo/golden-gate-bridge-san-francisco-1591382/

https://www.pexels.com/photo/woman-in-blue-dress-walking-on-concrete-staircase-leading-to-buildings-929168/

Batch Data Processing

Data Collection

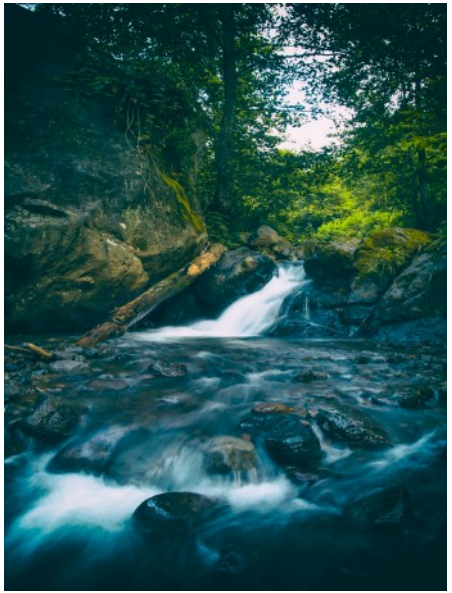
From upstream to staging
From staging to raw

Data Modelling

Automatic data modelling using inference
Manual modelling

Data Transformation

From raw to master



https://www.pexels.com/photo/time-lapse-photography-of-waterfall-2108374/

Stream Data Processing

A data stream is an **ordered sequence of instances** (packets)

The instances arrives in a **continuous flow**

Instead of collection, **near real-time** processing of the stream (stream mining)

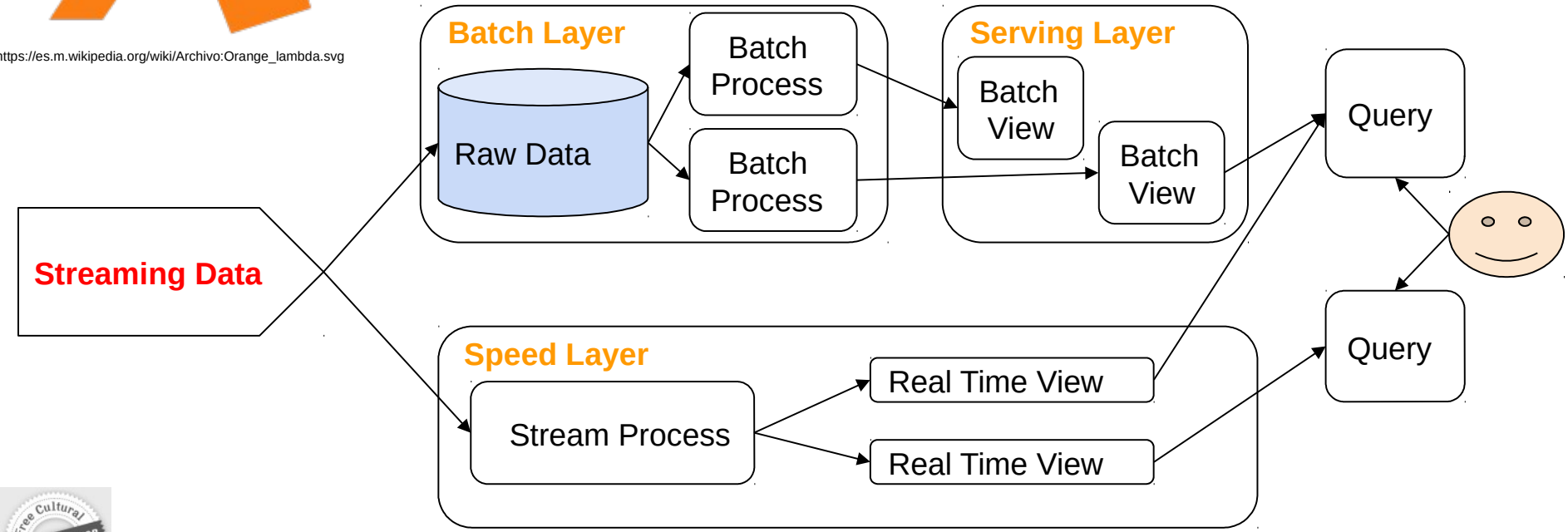
The data is also modelled and processed, and optionally, it can be stored

Twitter, Netflix, Spotify, TCP/UDP ... some samples of data streaming

https://es.m.wikipedia.org/wiki/Archivo:Orange_lambda.svg

Lambda Architecture: Batch + Stream

http://lambda-architecture.net/



https://www.pexels.com/photo/beach-coast-island-landscape-348520/

Spark and Lambda relationship

RDDs, Datasets and Dataframes (Batch and Stream)

Structured Streaming (Stream)

https://www.pexels.com/photo/person-holding-pumpkin-beside-woman-1374545/

Practical

Spark as the reference platform for building Big Data platforms:

- Data collection
- Data modelling
- Data transformations

Distributed data processing (scalability and performance) based on **data partitioning**

https://www.pexels.com/photo/flying-hot-air-balloon-above-snow-covered-mountain-1740103/

Open Source

Apache Hadoop “umbrella”:

- Apache HDFS
- Apache Spark
- Many others

Apache Spark Basics

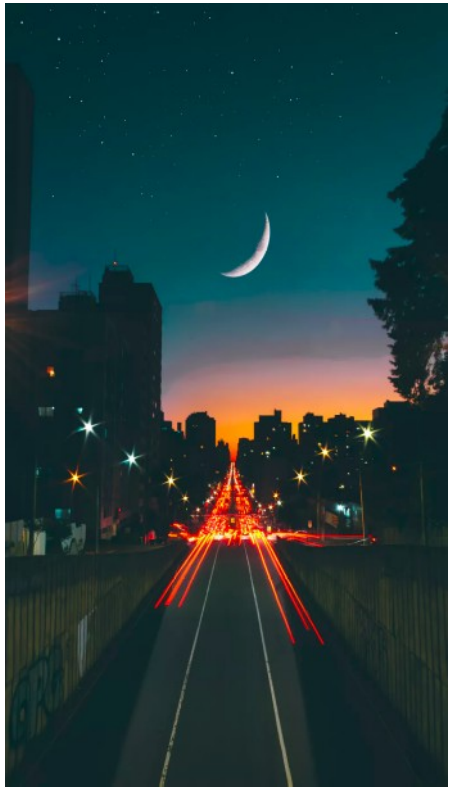
https://spark.apache.org/

Unified analytics engine for **large-scale data processing**

Data processing done in **memory** (fast!) and **distributed** (scalable)

Easy to use API (Scala, Python, Java and R) hiding the distributed complexity (in most cases).

Extra batteries: SQL, Streaming, Machine Learning, Graphs



https://www.pexels.com/photo/night-sky-over-city-road-1775302/

Research Paper: RDDs in the core

https://pages.databricks.com/rs/094-YMS-629/images/nsdi_spark.pdf

«**R**esilient **D**istributed **D**atasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing»

The **immutable** data to be processed in converted to RDDs with strong typing and distributed to the cluster for its processing in large clusters in a fault-tolerant manner.

Friendly for programmers with a simple but powerful functional API

https://www.pexels.com/photo/sears-tower-usa-1722183/

Spark Cluster Types

Standalone: default one used in one node deployments. Your local dev is executed with the same cluster code than in production.

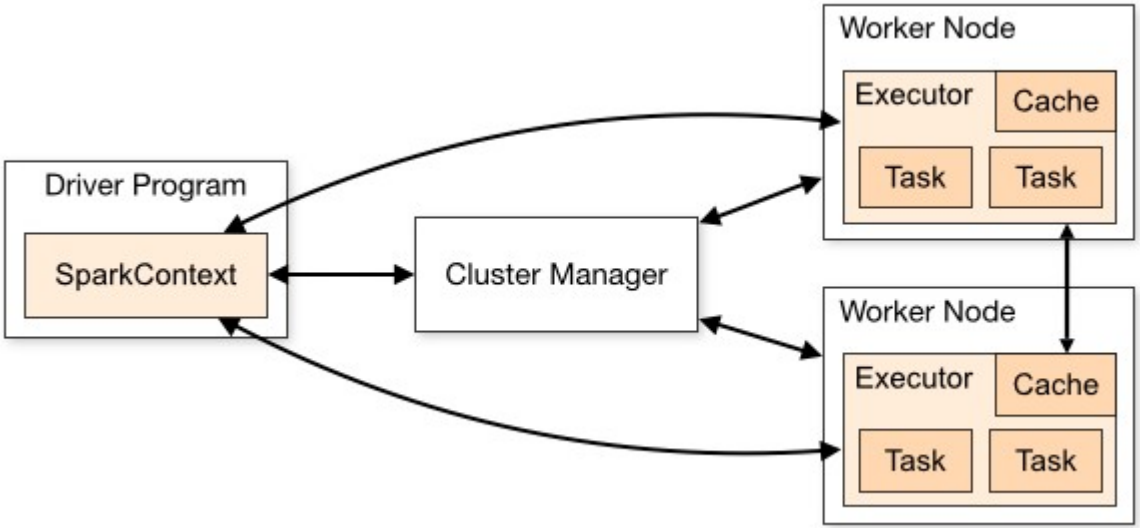
Apache Mesos

Apache Hadoop Yarn

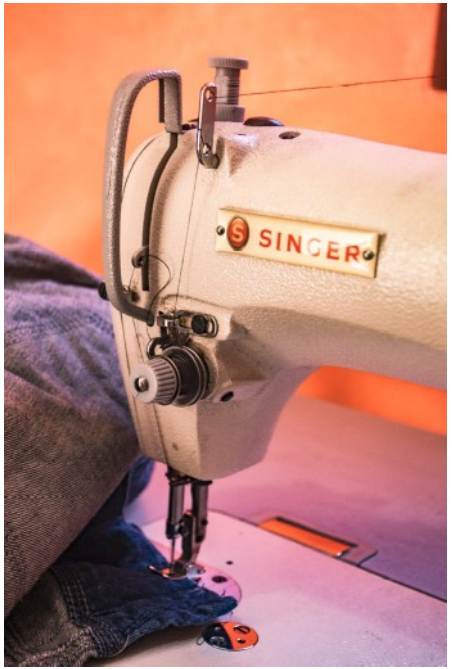
Kubernetes: the hotter one

https://www.pexels.com/photo/person-holding-surfboard-standing-on-rock-1749580

Execution of Sparks programs



Playing with Apache Spark



https://www.pexels.com/photo/white-singer-sewing-machine-783590/

Let’s try to do it all together:

Download Apache Spark (you need Java to execute it):

https://spark.apache.org/downloads.html (DO IT!)

Start Apache Spark Shell

Follow the practical session

https://www.pexels.com/photo/multicolored-smoke-bomb-digital-wallpaper-1471748/

Transformations and actions

The way Spark models the data processing

Transformations don't execute anything: **Laziness**

Transformations are chained (**DAG**) and optimized (Catalyst) before their execution

DAG: Direct Acyclic Graph describing the processing to be done on the data

Actions fire the current DAG and execute the transformations in the cluster

https://www.pexels.com/photo/mountains-nature-arrow-guide-66100/

The used API: SparkSQL (Dataset API)

https://spark.apache.org/docs/latest/sql-programming-guide.html

Spark SQL is a Spark module for structured data processing: extra optimization based on the knowledge of the data

Build on top of RDDs:

Dataset is a RDD with an schema (structure of data)

DataFrame is a Dataset organized into named columns (**Rows**, like a SQL table)

https://www.pexels.com/photo/sparkler-new-year-s-eve-sylvester-sparks-38196/

Starting Apache Spark Shell

https://spark.apache.org/docs/latest/quick-start.html

SparkSession is the entry point to programming Spark with the Dataset and DataFrame API. Be sure to use Java 1.8.

```
acastillo@acastillo:~/devel/spark/spark-2.4.3-bin-hadoop2.7$ JAVA_HOME=/usr/lib/jvm/java-8-openjdk-
amd64/ bin/pyspark
SparkSession available as 'spark'. (the shell is the driver)
>>> spark.sparkContext.appName
u'PySparkShell'
>>> textFile = spark.read.text("README.md")
>>> textFile
DataFrame[value: string]
>>> textFile.count()
105
>>> textFile.show(3, False)
+------------------------------------------------------------------------------+
|value                                                                         |
+------------------------------------------------------------------------------+
|# Apache Spark                                                                |
|                                                                              |
|Spark is a fast and general cluster computing system for Big Data. It provides|
+------------------------------------------------------------------------------+
```

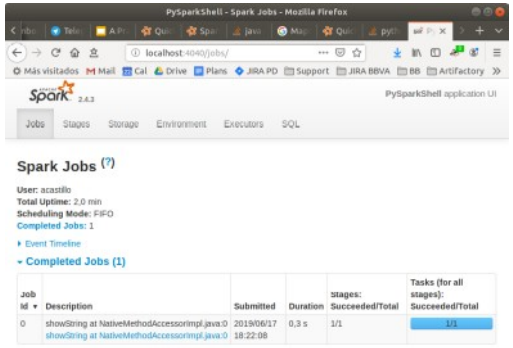
https://www.pexels.com/photo/woman-walking-in-beach-509127/

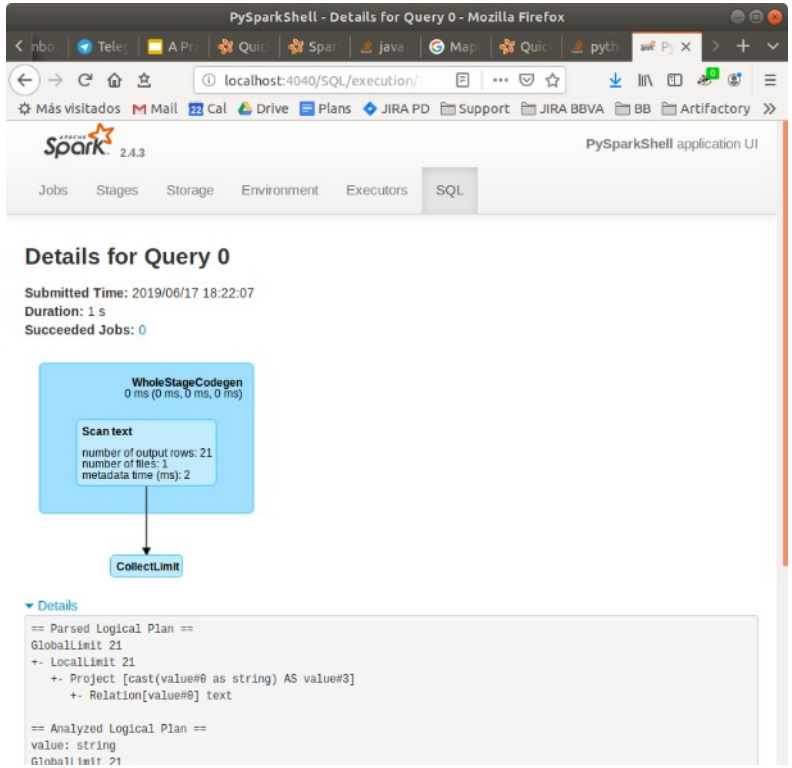
Partitions and transformations

```
>>> textFile.rdd.getNumPartitions()
1 one worker used
>>> textFile.repartition(8).rdd.getNumPartitions()
8 eight workers used
>>> textFile.filter(textFile.value.contains("Spark")).count()
20
acastillo@acastillo:~/devel/spark/spark-2.4.3-bin-hadoop2.7$ cat
README.md  | grep Spark | wc -l
20
```

The Spark Web console

https://www.pexels.com/photo/selective-focus-photo-of-magnifying-glass-1194775/





Jobs, Stages and Tasks

https://www.pexels.com/photo/multicolored-smoke-bomb-digital-wallpaper-1471748/

A job is created and executed once an action is executed.

The job is divided in tasks. **Each task works with a partition**. Tasks are distributed in the executors available.

Tasks are grouped in stages: there is no shuffle between the tasks inside the same stage.

https://www.pexels.com/photo/achievement-adult-agreement-arms-1243521/

Sending an App to Spark Cluster

```
acastillo@acastillo:~/devel/spark/spark-2.4.3-bin-hadoop2.7$ cat simple.py

"""SimpleApp.py"""
from pyspark.sql import SparkSession

logFile = "/home/acastillo/devel/spark/spark-2.4.3-bin-hadoop2.7/README.md"
spark = SparkSession.builder.appName("SimpleApp").getOrCreate()
logData = spark.read.text(logFile).cache()

numAs = logData.filter(logData.value.contains('a')).count()
numBs = logData.filter(logData.value.contains('b')).count()

print("Lines with a: %i, lines with b: %i" % (numAs, numBs))

spark.stop()

acastillo@acastillo:~/devel/spark/spark-2.4.3-bin-hadoop2.7$ JAVA_HOME=/usr/lib/jvm/java-8-
openjdk-amd64 bin/spark-submit simple.py
…
Lines with a: 62, lines with b: 31
…
```

https://www.pexels.com/photo/battle-black-and-white-board-game-challenge-358562/

Main transformations

A narrow transformation does not need **shuffle** (partitions exchange between executors); wide yes.

Narrow: map, filter, flatMap, sample, union

Wide: join, distinct, intersection, groupByKey, reduceByKey, sort, partitionBy, repartition, coalesce, dropDuplicates

https://www.pexels.com/photo/action-adult-athlete-blur-213775/

Main actions

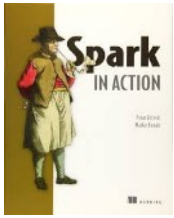
Count, collect, reduce, lookup, save, head, show, foreach

Be careful: the results of the actions could return all data to the driver (memory and disk usage risks).

https://www.pexels.com/photo/top-view-of-library-with-red-stairs-1261180/

Using GitHub archive data

```
acastillo@acastillo:~/devel/spark/spark-2.4.3-bin-hadoop2.7$ wget http://data.githubarchive.org/2019-05-31-
21.json.gz
acastillo@acastillo:~/devel/spark/spark-2.4.3-bin-hadoop2.7$ gunzip 2019-05-31-21.json.gz
acastillo@acastillo:~/devel/spark/spark-2.4.3-bin-hadoop2.7$ JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/
bin/pyspark
>>> events = spark.read.json("2019-05-31-21.json")
>>> events.count()
63826
>>> events.printSchema()
root
 |-- actor: struct (nullable = true)
 |      |-- avatar_url: string (nullable = true)
 |      |-- display_login: string (nullable = true)
…
>>> events.select("repo.name").show(20, False)
+--------------------------------------+
|name                                  |
+--------------------------------------+
|kedacore/keda                         |
|OSSIA/score                           |
|Unity-Technologies/ml-agents          |
…
>>> events.filter("repo.name = 'eslibre/charlas'").select("actor.display_login", "created_at","type",
"payload.pull_request.html_url").show(truncate=False)
+-------------+--------------------+---------------+----------------------------------------+
|display_login|created_at          |type           |html_url                                |
+-------------+--------------------+---------------+----------------------------------------+
|dfurmans     |2019-05-31T21:37:33Z|PullRequestEvent|https://github.com/eslibre/charlas/pull/43|
+-------------+--------------------+---------------+----------------------------------------+
```



https://www.pexels.com/photo/top-view-of-library-with-red-stairs-1261180/

Using GitHub archive data

```
acastillo@acastillo:~/devel/spark/spark-2.4.3-bin-hadoop2.7/github$ wget http://data.githubarchive.org/2019-05-31-
{0..23}.json.gz
acastillo@acastillo:~/devel/spark/spark-2.4.3-bin-hadoop2.7/github$ du -sh .
5,7G
acastillo@acastillo:~/devel/spark/spark-2.4.3-bin-hadoop2.7$ JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/
bin/pyspark
>>> events = spark.read.json("github/2019-05-31-*.json")
>>> events.count()
1767137
>>> from pyspark.sql.functions import desc
>>> events.groupby("repo").count().select("repo.name","count").sort(desc("count")).show(10, False)
+-----------------------------------+-----+
|name                               |count|
+-----------------------------------+-----+
|Parkboyoung11/Kirito               |9197 |
|willcbaker-ext/subt                |3587 |
…
>>> events.registerTempTable("events")
>>> spark.sql("select repo.name, count(*) as count from events group by repo order by count desc limit 10").show()
+--------------------+-----+
|                name|count|
+--------------------+-----+
|Parkboyoung11/Kirito| 9197|
| willcbaker-ext/subt| 3587|
…
```

Volume: 2/12/2011 -> 6/19/2019: 3049 days * 5.7GB = 17 TB
Variety: more than 20 types of events
Velocity: 2M events/day, 1388 events/s

https://www.pexels.com/photo/man-person-street-shoes-2882/

Main pitfalls

Out of Memory errors in workers (in shuffle operations mainly)

Out of Memory errors in the driver, executing program in the driver

Bad partitioning of the data

Implementing algorithms not suitable for data partitioning (recursive ones)

https://www.pexels.com/photo/activity-blueprint-building-building-site-583393/

A real use case using Spark

BBVA Transcendence:

http://www.expansion.com/empresas/banca/2019/01/03/5c2d1480468aeb73778b45db.html

https://www.pexels.com/photo/black-and-white-connected-hands-love-265702/

Credits

https://spark.apache.org/

https://www.pexels.com/

https://www.paradigmadigital.com/

https://www.bbva.com/

https://github.com/aylabs/bigdata-practical-intro