Module 6

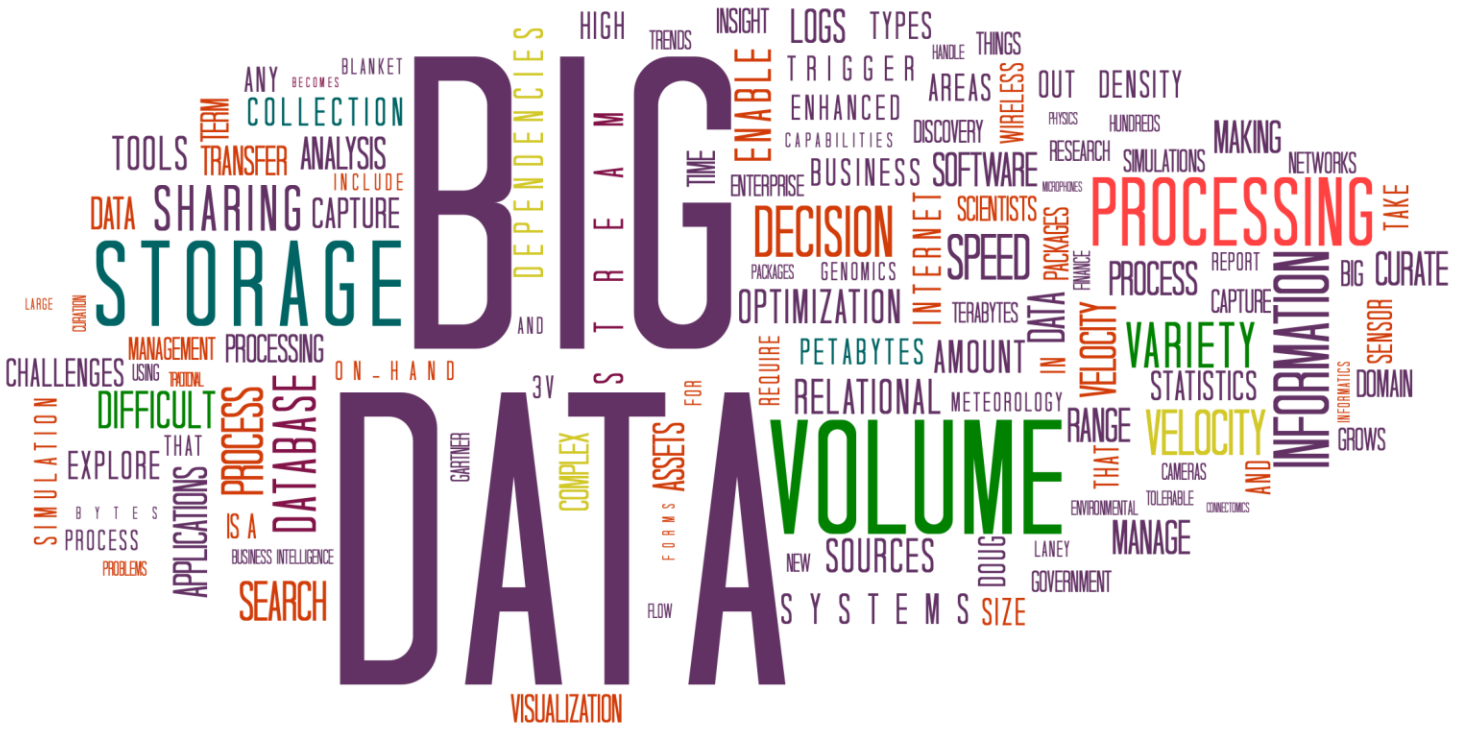
Procesamiento de datos escalable: Desarrollo de aplicaciones en entornos Big Data con Hadoop y Spark

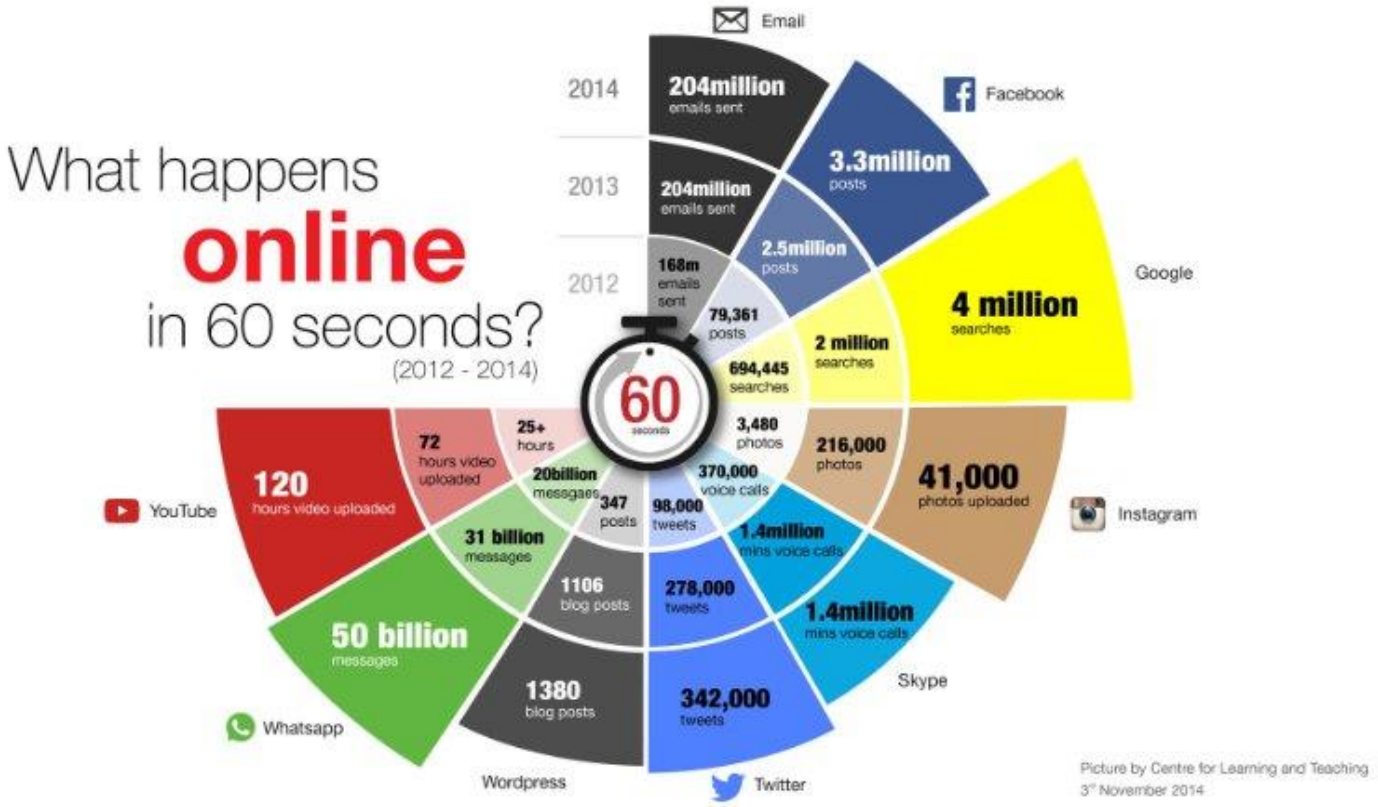
Antonio J. Nebro (ajnebro@uma.es) – University of Málaga
Cristóbal Barba (cbarba@uma.es) – University of Málaga

Table of contents

Introduction to scalable data processing

- What is Big Data?





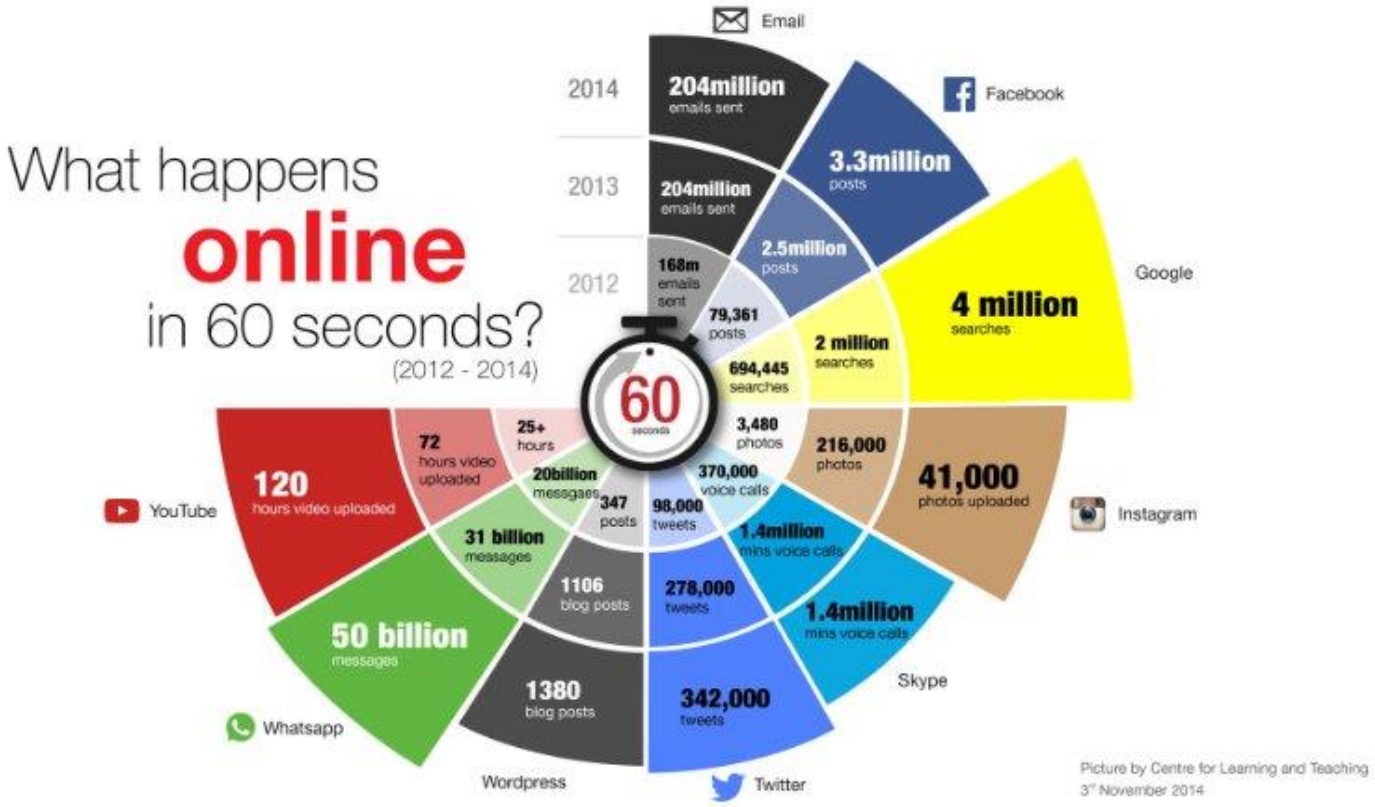
Nowadays, huge amounts of information are generated (**Volume**)

Applications are required to process and analyse this data:

- Hardware: **Scalable distributed systems**, from one node to thousands.
- Software: **Powerful, easy-to-use parallel and distributed processing systems**

Example application:

- Determine trending topics on Twitter or Facebook by analysing the last hour's posts.



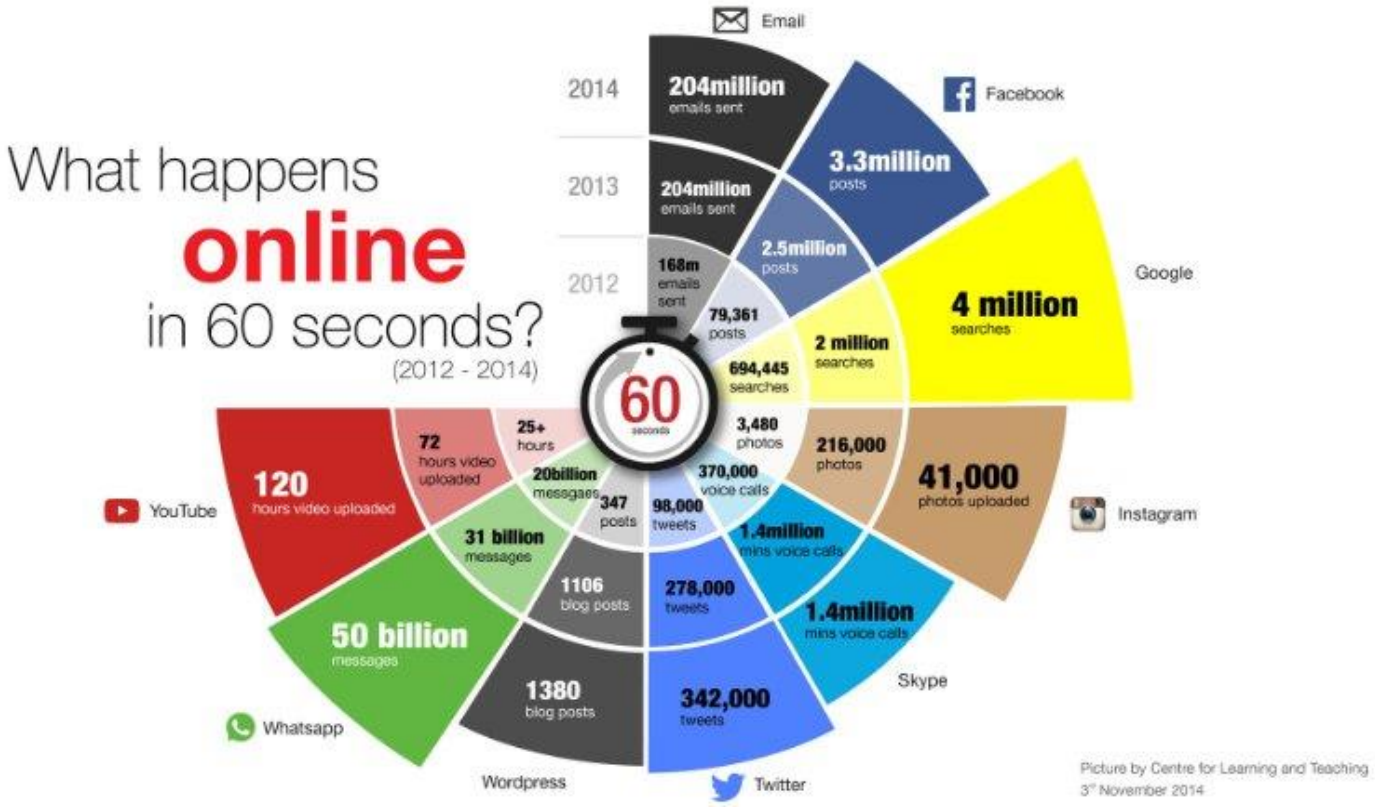
Data are produced very quickly (**Velocity**).

Some examples:

- Social networking
- Video cameras
- Sensors

Applications are required to process and analyse this data:

- Must be processed on the fly (streaming).
- Not feasible to store in databases for later analysis
  - Lack of storage capacity
  - Inability to analyze data in real time



Data may come from different sources (**Variety**).

Requirements for applications that process and analyse this data:

- Need to integrate data from different sources.

Example application:

- Predicting the level of traffic on a street for the next hour.
- Possible data sources
  - Historical traffic time data
  - Weather forecasting
  - Video cameras showing the flow of vehicles in real time
  - Websites (information from local councils on traffic closures)

Then, wh


Volume
Size of Data
Velocity
The Speed at which Data is Generated
Variety
Different type of Data
Veracity
Data Accuracy
Value
Useful Data
BigData
Validity
Data quality, Governace, Moster Data Management on Massive
Variability
Dynamic, Evolving Behavior in Data Source
Venue
Distributed Heterogeneous Data from Multiple Platforms
Vocabulary
Data Models, Semantics that describes data Structure
Vagueness
Confusion over Meaning of BigData and Tools used

The Spark engine for large scale data analitycs

- Apache Spark (https://spark.apache.org/)

What is Apache Spark™?

Apache Spark™ is a multi-language engine for executing data engineering, data science, and machine learning on single-node machines or clusters.

Key features

Batch/streaming data

Unify the processing of your data in batches and real-time streaming, using your preferred language: Python, SQL, Scala, Java or R.

SQL analytics

Execute fast, distributed ANSI SQL queries for dashboarding and ad-hoc reporting. Runs faster than most data warehouses.

Data science at scale

Perform Exploratory Data Analysis (EDA) on petabyte-scale data without having to resort to downsampling

Machine learning

Train machine learning algorithms on a laptop and use the same code to scale to fault-tolerant clusters of thousands of machines.

- It provides high-level APIs in Java, Scala, Python, and R
- It can access diverse data sources
  - HDFS, Cassandra, Hbase, S3

Spark versions

- https://en.wikipedia.org/wiki/Apache_Spark

Version	Original release date	Latest version	Release date
0.5	2012-06-12	0.5.1	2012-10-07
0.6	2012-10-14	0.6.2	2013-02-07
0.7	2013-02-27	0.7.3	2013-07-16
0.8	2013-09-25	0.8.1	2013-12-19
0.9	2014-02-02	0.9.2	2014-07-23
1.0	2014-05-26	1.0.2	2014-08-05
1.1	2014-09-11	1.1.1	2014-11-26
1.2	2014-12-18	1.2.2	2015-04-17
1.3	2015-03-13	1.3.1	2015-04-17
1.4	2015-06-11	1.4.1	2015-07-15
1.5	2015-09-09	1.5.2	2015-11-09
1.6	2016-01-04	1.6.3	2016-11-07
2.0	2016-07-26	2.0.2	2016-11-14
2.1	2016-12-28	2.1.3	2018-06-26
2.2	2017-07-11	2.2.3	2019-01-11
2.3	2018-02-28	2.3.4	2019-09-09
2.4 LTS	2018-11-02	2.4.8	2021-05-17[38]
3.0	2020-06-18	3.0.3	2021-06-01[39]
3.1	2021-03-02	3.1.3	2022-02-18[40]
3.2	2021-10-13	3.2.1	2022-01-26
3.3	2022-06-16	3.3.0	2022-06-16

Legend: Old version | Older version, still maintained | **Latest version** | Latest preview version

Spark and scalable processing

- Goals of parallelism
  - To run programs faster
  - To run bigger programs
    - More CPUs, main memory, secondary memory ls of parallel processing
- Scalable processing:
  - A same program should work on a single computer and on a cluster of thousands of nodes
- Stuff about parallelism learned in the Computer Science Degree
  - Background: critical sections, deadlocks, livelocks
  - Low level solutions: semaphores, shared memory, message passing
  - These approaches are very complex

Spark and scalable processing

- Spark has a high level parallel programing model
- A same Spark program can run
  - In a single multi-core processor
  - In a local cluster (stand-alone deploy mode)
  - Apache Mesos (https://mesos.apache.org)
  - Kubernetes
  - Hadoop Yarn

Spark requirements

- Development tools (Java)
  - Java JDK 19+
  - Spark: https://spark.apache.org/downloads.html
  - IDE de desarrollo: Eclipse, IntelliJ Idea
  - Maven (dependency management): http://maven.apache.org/download.cgi?Preferred=ftp://mirror.reverse.net/pub/apache/

Spark requirements

ANACONDA.

- Development tools (Python)
  - Python interpreter (https://www.anaconda.com/download)
  - Spark: https://spark.apache.org/downloads.html
  - Development environment: PyCharm Community https://www.jetbrains.com/pycharm/download/
  - PySpark: “pip install pyspark”

APACHE Spark™

Spark programming

- Example: add numbers

```python
from pyspark import SparkContext, SparkConf


def main() -> None:
    """
    Python program that uses Apache Spark to sum a list of numbers
    """
    spark_conf = SparkConf()
    spark_context = SparkContext(conf=spark_conf)

    logger = spark_context._jvm.org.apache.log4j
    logger.LogManager.getLogger("org").setLevel(logger.Level.WARN)

    data = [1, 2, 3, 4, 5]
    distributed_data = spark_context.parallelize(data)

    sum = distributed_data.reduce(lambda s1, s2: s1 + s2)

    print("The sum is " + str(sum))


if __name__ == '__main__':
    main()

```

```java
package org.masterbigdata.spark;

import ...

no usages
public class AddNumbersLambda {
  public static void main(String[] args) {
    // Step 1: create a SparkConf object
    SparkConf sparkConf = new SparkConf().setAppName("Add numbers") ;

    // Step 2: create a Java Spark Context
    JavaSparkContext sparkContext = new JavaSparkContext(sparkConf) ;

    // Step 3: initialize an array of integers
    Integer[] numbers = new Integer[]{1,2,3,4,5,6,7,8} ;

    // Step 4: create a list of integers
    List<Integer> integerList = Arrays.asList(numbers) ;

    // Step 5: create a JavaRDD
    JavaRDD<Integer> distributedList = sparkContext.parallelize(integerList) ;

    // Step 6: sum the numbers
    int sum = distributedList.reduce((integer, integer2) -> integer + integer2) ;

    // Step 6: print the sum
    System.out.println("The sum is: " + sum) ;

    // Step 7: stop the spark context
    sparkContext.stop() ;
  }
}
```

Spark programming

- Example: add numbers
  - Running the Python code

```
python — -bash — 80×11
pdi-120-161:python ajnebro$ spark-submit spark/rdd/AddNumbers.py
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
18/02/26 13:35:52 INFO SparkContext: Running Spark version 2.2.0
18/02/26 13:35:53 WARN NativeCodeLoader: Unable to load native-hadoop library fo
r your platform... using builtin-java classes where applicable
18/02/26 13:35:53 INFO SparkContext: Submitted application: AddNumbers.py
18/02/26 13:35:53 INFO SecurityManager: Changing view acls to: ajnebro
18/02/26 13:35:53 INFO SecurityManager: Changing modify acls to: ajnebro
18/02/26 13:35:53 INFO SecurityManager: Changing view acls groups to:
18/02/26 13:35:53 INFO SecurityManager: Changing modify acls groups to:
18/02/26 13:35:53 INFO SecurityManager: SecurityManager: authentication disabled
```

```
python — -bash — 80×11
18/02/26 13:35:54 INFO BlockManagerMaster: Registering BlockManager BlockManager
Id(driver, 192.168.120.161, 59437, None)
18/02/26 13:35:54 INFO BlockManagerMasterEndpoint: Registering block manager 192
.168.120.161:59437 with 366.3 MB RAM, BlockManagerId(driver, 192.168.120.161, 59
437, None)
18/02/26 13:35:54 INFO BlockManagerMaster: Registered BlockManager BlockManagerI
d(driver, 192.168.120.161, 59437, None)
18/02/26 13:35:54 INFO BlockManager: Initialized BlockManager: BlockManagerId(dr
iver, 192.168.120.161, 59437, None)
The sum is 15
pdi-120-161:python ajnebro$ spark-submit spark/rdd/AddNumbers.py
```

Spark programming

- Spark provides two APIs:
  - RDD
    - Resilient Distributed Dataset (list of elements)
    - Collections of elements that can be processed in parallel
    - Fault tolerant
  - Dataframe
    - Table (matrix cols x rows)
    - They also also fault tolerant and can be processed in parallel

Spark programming

- Which API to use?
  - RDDs were the first Spark API
    - Low level, easy to understand
  - Dataframes are becoming the main API
    - High level, more complex

- The trend Is to towards the dataframes API
  - The new machine learning and streaming engines are based on dataframes

Spark applications: RDD API

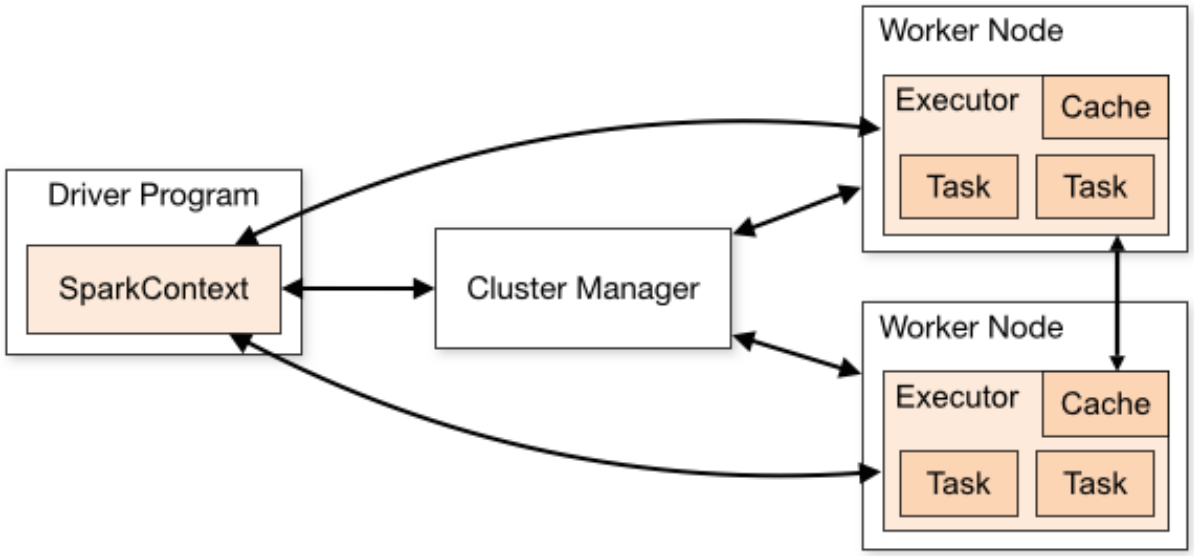
- RDD: Resilient Distributed Dataset
  - Can be created from lists and from data stored in files

```
// Step 4: create a list of integers
List<Integer> integerList = List.of(1,2,3,4,5,6,7,8) ;

// Step 5: create a JavaRDD
JavaRDD<Integer> distributedList = sparkContext
        .parallelize(integerList) ;
```

```
data = [1, 2, 3, 4, 5, 6, 7, 8]
distributed_data = spark_context.parallelize(data)
```

Spark architecture




```
List<Integer> integerList = List.of(1,2,3,4,5,6,7,8) ;
```

```
data = [1, 2, 3, 4, 5, 6, 7, 8]
```

Stored and processed in the drive node

```
JavaRDD<Integer> distributedList = sparkContext
        .parallelize(integerList) ;
```

```
distributed_data = spark_context.parallelize(data)
```

Stored and processed in the worker nodes

Spark applications: RDD API

- Operations with RDDs: transformations
  - Create an RDD from other RDDs

Transformation	Meaning
map(func)	Return a new distributed dataset formed by passing each element of the source through a function func.
filter(func)	Return a new dataset formed by selecting those elements of the source on which func returns true.
flatMap(func)	Similar to map, but each input item can be mapped to 0 or more output items (so func should return a Seq rather than a single item).
union(otherDataset)	Return a new dataset that contains the union of the elements in the source dataset and the argument.

Spark applications: RDD API

- Operations with RDDs: actions
  - Return a value after processing a set of RRDs

Action	Meaning
reduce(*func*)	Aggregate the elements of the dataset using a function *func* (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.
collect()	Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
count()	Return the number of elements in the dataset.
first()	Return the first element of the dataset.
saveAsTextFile(*path*)	Write the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system.

Spark applications: RDD API

- Example: sum numbers stored in files (RDD, Java)

```java
package org.masterinformatica.spark;

import ...

/**
 * Created by ajnebro on 13/4/16.
 */
public class AddNumbers2Lambda {
  public static void main(String[] args) {
    // Step 1: create a SparkConf object
    SparkConf conf = new SparkConf().setAppName("Add numbers") ;

    // Step 2: create a Java Spark Context
    JavaSparkContext context = new JavaSparkContext(conf) ;

    JavaRDD<String> lines = context.textFile(args[0]) ;

    JavaRDD<Integer> numbers = lines.map(s -> valueOf(s)) ;

    long initTime = System.currentTimeMillis() ;
    // Step 6: sum the numbers
    long sum = numbers.reduce((integer, integer2) -> integer + integer2) ;

    // Step 6: print the sum
    long computingTime = System.currentTimeMillis() - initTime ;

    System.out.println("Computing time: " + computingTime);
    System.out.println("Sum: " + sum);

    // Step 7: stop the spark context
    context.stop() ;
  }
}
```

Spark applications: RDD API

- Example: sum numbers stored in files (RDD, Python)

```python
import sys
import time

from pyspark import SparkConf, SparkContext


def main(file_name: str) -> None:
    spark_conf = SparkConf()
    spark_context = SparkContext(conf=spark_conf)

    logger = spark_context._jvm.org.apache.log4j
    logger.LogManager.getLogger("org").setLevel(logger.Level.WARN)

    start_computing_time = time.time()

    sum = spark_context \
        .textFile(file_name) \
        .map(lambda line: int(line)) \
        .reduce(lambda x, y: x + y)

    total_computing_time = time.time() - start_computing_time

    print("Sum: ", sum)
    print("Computing time: ", str(total_computing_time))

    spark_context.stop()


if __name__ == "__main__":
    """
    Python program that uses Apache Spark to sum a list of numbers stored in files
    """

    if len(sys.argv) != 2:
        print("Usage: spark-submit AddNumbersFromFilesWithTime.py <file>", file=sys.stderr)
        exit(-1)

    main(sys.argv[1])

```

Spark applications: RDD API

- Example: sum the number of lines of a text file containing the characters a and b (RDD, Python)

```python
import sys

from pyspark import SparkConf, SparkContext

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: spark-submit CountCharacters <file>", file=sys.stderr)
        exit(-1)

    spark_conf = SparkConf()
    spark_context = SparkContext(conf=spark_conf)

    logger = spark_context._jvm.org.apache.log4j
    logger.LogManager.getLogger("org").setLevel(logger.Level.WARN)

    lines = spark_context \
        .textFile(sys.argv[1])\
        .cache()

    number_of_as = lines\
        .filter(lambda line: "a" in line)\
        .count()

    number_of_bs = lines\
        .filter(lambda line: "b" in line)\
        .count()

    print("Number of 'a's: " + str(number_of_as))
    print("Number of 'b's: " + str(number_of_bs))

    spark_context.stop()
```

Spark applications: RDD API

- Example: Word count (RDD, Java)

```java
public class WordCountFromFiles {
  static Logger log = Logger.getLogger(WordCountFromFiles.class.getName());

  public static void main(String[] args) {
    Logger.getLogger("org").setLevel(Level.OFF) ;

    // STEP 1: create a SparkConf object
    if (args.length < 1) {
      log.fatal("Syntax Error: there must be one argument (a file name or a directory)")  ;
      throw new RuntimeException();
    }

    // STEP 2: create a SparkConf object
    SparkConf sparkConf = new SparkConf().setAppName("Spark Word count") ;

    // STEP 3: create a Java Spark context
    JavaSparkContext sparkContext = new JavaSparkContext(sparkConf) ;

    // STEP 4: read lines of files
    JavaRDD<String> lines = sparkContext.textFile(args[0]);

    // STEP 5: split the lines into words
    JavaRDD<String> words = lines.flatMap(s -> Arrays.asList(s.split(" ")).iterator()) ;

    // STEP 6: map operation to create pairs <word, 1> per every word
    JavaPairRDD<String, Integer> pairs = words
        .mapToPair(word -> new Tuple2<>(word, 1)) ;

    // STEP 6: reduce operation that sum the values of all the pairs having the same key (word)
    //         generating a pair <key, sum>
    JavaPairRDD<String, Integer> groupedPairs = pairs
            .reduceByKey((integer, integer2) -> integer + integer2) ;

    // STEP 7: map operation to get an RDD of pairs <sum, key>. We need this step because Spark
    //         Spark provides a sortByKey() funcion (see next step) but not a sortByValue()
    JavaPairRDD<Integer, String> reversePairs = groupedPairs
        .mapToPair(pair -> new Tuple2<>(pair._2(), pair._1)) ;

    // STEP 8: sort the results by key ant take the first 20 elements
    List<Tuple2<Integer, String>> output = reversePairs
        .sortByKey(false)
        .take(20) ;

    // STEP 9: print the results
    for (Tuple2<?, ?> tuple : output) {
      System.out.println(tuple._1() + ": " + tuple._2()) ;
    }

    // STEP 19: stop the spark context
    sparkContext.stop();
  }
```

Spark applications: RDD API

- Example: Word count, compact version (RDD, Java)

```java
public class WordCountFromFilesCompactVersion {
  static Logger log = Logger.getLogger(WordCountFromFilesCompactVersion.class.getName());

  public static void main(String[] args) {
    Logger.getLogger("org").setLevel(Level.OFF) ;

    // STEP 1: create a SparkConf object
    if (args.length < 1) {
      log.fatal("Syntax Error: there must be one argument (a file name or a directory)")  ;
      throw new RuntimeException();
    }

    // STEP 2: create a SparkConf object
    SparkConf sparkConf = new SparkConf().setAppName("Spark Word count") ;

    // STEP 3: create a Java Spark context
    JavaSparkContext sparkContext = new JavaSparkContext(sparkConf) ;

    List<Tuple2<Integer, String>> output = sparkContext
        .textFile(args[0])
        .flatMap(s -> Arrays.asList(s.split(" ")).iterator())
        .mapToPair(word -> new Tuple2<>(word, 1))
        .reduceByKey((integer, integer2) -> integer + integer2)
        .mapToPair(pair -> new Tuple2<>(pair._2(), pair._1))
        .sortByKey(false)
        .take(20) ;

    // STEP 9: print the results
    for (Tuple2<?, ?> tuple : output) {
      System.out.println(tuple._1() + ": " + tuple._2()) ;
    }

    // STEP 19: stop the spark context
    sparkContext.stop();
  }
}
```

Spark applications: RDD API

- Example: Word count (RDD, Python)

```python
import sys

from pyspark import SparkConf, SparkContext

def split_line(line: str):
    return line.split(' ')

def generate_pair(word: str):
    return (word, 1)

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: spark-submit WordCountVerbose.py <file>", file=sys.stderr)
        exit(-1)

    spark_conf = SparkConf()
    spark_context = SparkContext(conf=spark_conf)

    logger = spark_context._jvm.org.apache.log4j
    logger.LogManager.getLogger("org").setLevel(logger.Level.WARN)

    lines = spark_context.textFile(sys.argv[1])
    words = lines.flatMap(lambda line: split_line(line))
    pairs = words.map(lambda word: generate_pair(word))
    reduced_pairs = pairs.reduceByKey(lambda a, b: a + b)

    output = reduced_pairs.map(lambda pair : (pair[1], pair[0]))\
        .sortByKey(False)\
        .take(20)

    for (count, word) in output:
        print("%i: %s" % (count, word))

    spark_context.stop()
```

Spark applications: RDD API

- Example: Word count, compact version (RDD, Python)

```python
import sys

from pyspark import SparkConf, SparkContext

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: spark-submit wordcount <file>", file=sys.stderr)
        exit(-1)

    spark_conf = SparkConf()
    spark_context = SparkContext(conf=spark_conf)

    logger = spark_context._jvm.org.apache.log4j
    logger.LogManager.getLogger("org").setLevel(logger.Level.WARN)

    output = spark_context \
        .textFile(sys.argv[1]) \
        .flatMap(lambda line: line.split(' ')) \
        .map(lambda word: (word, 1)) \
        .reduceByKey(lambda a, b: a + b) \
        .sortByKey() \
        .collect()

    for (word, count) in output:
        print("%s: %i" % (word, count))

    spark_context.stop()
```

Spark applications: Dataframe API

- Definitions:
  - Dataset: collection of rows
  - Dataframe: dataset where all the columns have a name
- As RDDs, Spark dataframes are
  - Immutable
  - Distributed
  - Manipulated with lazy operations

id	ident	type	name	latitude_deg	longitude_deg	elevation_ft	continent	iso_country	iso_region	municipality	scheduled_service	gps_code	iata_code	local_code	h
6523	00A	heliport	Total Rf Heliport	40.07080078125	-74.93360137939453	11	NA	US	US-PA	Bensalem	no	00A	null	00A	
6524	00AK	small_airport	Lowell Field	59.94919968	-151.695999146	450	NA	US	US-AK	Anchor Point	no	00AK	null	00AK	
6525	00AL	small_airport	Epps Airpark	34.864799449951172	-86.77030181884766	820	NA	US	US-AL	Harvest	no	00AL	null	00AL	
6526	00AR	heliport	Newport Hospital ...	35.608699798583984	-91.25489807128906	237	NA	US	US-AR	Newport	no	00AR	null	00AR	
6527	00AZ	small_airport	Cordes Airport	34.305599212646484	-112.16500091552734	3810	NA	US	US-AZ	Cordes	no	00AZ	null	00AZ	
6528	00CA	small_airport	Goldstone /Gts/ A...	35.350498199499995	-116.888000488	3038	NA	US	US-CA	Barstow	no	00CA	null	00CA	
6529	00CO	small_airport	Cass Field	40.62220001220703	-104.34400177001953	4830	NA	US	US-CO	Briggsdale	no	00CO	null	00CO	
6531	00FA	small_airport	Grass Patch Airport	28.645500183105469	-82.21900177001953	53	NA	US	US-FL	Bushnell	no	00FA	null	00FA	
6532	00FD	heliport	Ringhaver Heliport	28.846599578857422	-82.34539794921875	25	NA	US	US-FL	Riverview	no	00FD	null	00FD	
6533	00FL	small_airport	River Oak Airport	27.230899810791016	-80.96920013427734	35	NA	US	US-FL	Okeechobee	no	00FL	null	00FL	
6534	00GA	small_airport	Lt World Airport	33.767501831054688	-84.06829833984375	700	NA	US	US-GA	Lithonia	no	00GA	null	00GA	
6535	00GE	heliport	Caffrey Heliport	33.884201049804688	-84.73390197753906	957	NA	US	US-GA	Hiram	no	00GE	null	00GE	
6536	00HI	heliport	Kaupulehu Heliport	19.832500457763672	-155.98199462890625	43	NA	US	US-HI	Kailua/Kona	no	00HI	null	00HI	
6537	00ID	small_airport	Delta Shores Airport	48.145301818847656	-116.21399688720703	2064	NA	US	US-ID	Clark Fork	no	00ID	null	00ID	
6538	00II	heliport	Bailey Generation...	41.644500732421875	-87.122802734375	600	NA	US	US-IN	Chesterton	no	00II	null	00II	
6539	00IL	small_airport	Hammer Airport	41.97840118408203	-89.5604019165039	840	NA	US	US-IL	Polo	no	00IL	null	00IL	
6540	00IN	heliport	St Mary Medical C...	41.51139831542969	-87.2605972290039	634	NA	US	US-IN	Hobart	no	00IN	null	00IN	
6541	00IS	small_airport	Hayenga's Cant Fi...	40.025600433349609	-89.1229019165039	820	NA	US	US-IL	Kings	no	00IS	null	00IS	
6542	00KS	small_airport	Hayden Farm Airport	38.72779846191406	-94.93049621582031	1100	NA	US	US-KS	Gardner	no	00KS	null	00KS	
6543	00KY	small_airport	Robbins Roost Air...	37.4094009399414	-84.61969757080078	1265	NA	US	US-KY	Stanford	no	00KY	null	00KY	

http://spark.apache.org/docs/latest/sql-getting-started.html

Spark applications: Dataframe API

- Example: Reading a CSV fileDataset: collection of rows

```
from pyspark.sql import SparkSession

"""
Data file source: https://data.sfgov.org/Culture-and-Recreation/Film-Locations-in-San-Francisco/yitu-d5am
"""

def main() -> None:

    spark_session = SparkSession \
        .builder \
        .getOrCreate()

    logger = spark_session._jvm.org.apache.log4j
    logger.LogManager.getLogger("org").setLevel(logger.Level.WARN)

    data_frame = spark_session\
        .read\
        .format("csv")\
        .option("header", "true")\
        .load("data/Film_Locations_in_San_Francisco.csv")

    data_frame.printSchema()
    data_frame.show()


if __name__ == '__main__':
    main()
```

Spark applications: Dataframe API

- A dataframe can be created
  - From an existing RDD
  - From a data file (CSV, JSON, text, etc.)

- The scheme of a dataframe
  - Can be inferred automatically
  - But can be defined explicitly
    - To reduce computer overhead
    - Mandatory to process data in streaming

Spark applications: Dataframe API

- Example: creating a dataframe from an existing RDD

```python
from pyspark.sql import SparkSession, Row

def main() -> None:

    spark_session = SparkSession \
        .builder \
        .getOrCreate()

    list_of_pairs = [('Luis', 23), ('Ana', 24), ('Jose', 20), ('Carlos', 26), ('Maria', 23)]

    rdd_of_pairs = spark_session.sparkContext.parallelize(list_of_pairs)

    students = rdd_of_pairs.map(lambda pair: Row(name=pair[0], age=int(pair[1])))
    students_data_frame = spark_session.createDataFrame(students)

    students_data_frame.printSchema()

    students_data_frame.show()

if __name__ == '__main__':
    main()
```

exampleRDD

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
root
 |-- age: long (nullable = true)
 |-- name: string (nullable = true)

+---+------+
|age|  name|
+---+------+
| 23|  Luis|
| 24|   Ana|
| 20|  Jose|
| 26|Carlos|
| 23| Maria|
+---+------+
```

Spark applications: Dataframe API

- Example: creating a dataframe from a data file

```python
from pyspark.sql import SparkSession

def main() -> None:
    spark_session = SparkSession \
        .builder \
        .getOrCreate()

    logger = spark_session._jvm.org.apache.log4j
    logger.LogManager.getLogger("org").setLevel(logger.Level.WARN)

    data_frame = spark_session\
        .read\
        .format("csv")\
        .options(inferschema = "true")\
        .load("data/numbers.txt")

    data_frame.printSchema()
    data_frame.show()


if __name__ == '__main__':
    main()
```

```python
from pyspark.sql import SparkSession

def main() -> None:

    spark_session = SparkSession \
        .builder \
        .getOrCreate()

    logger = spark_session._jvm.org.apache.log4j
    logger.LogManager.getLogger("org").setLevel(logger.Level.WARN)

    data_frame = spark_session\
        .read\
        .json("data/primer-dataset.json")

    data_frame.printSchema()

if __name__ == '__main__':
    main()
```

Spark applications: Dataframe API

- Example: creating a dataframe from a data file
  - Explicit schema definition

```
from pyspark.sql.types import StructType, StructField, StringType, DoubleType, IntegerType

from pyspark.sql import SparkSession


def main() -> None:
    spark_session = SparkSession \
        .builder \
        .getOrCreate()

    logger = spark_session._jvm.org.apache.log4j
    logger.LogManager.getLogger("org").setLevel(logger.Level.WARN)

    fields = [StructField("id", StringType(), True),
              StructField("ident", StringType(), True),
              StructField("type", StringType(), True),
              StructField("name", StringType(), True),
              StructField("latitude_deg", DoubleType(), True),
              StructField("longitude_deg", DoubleType(), True),
              StructField("elevation_ft", IntegerType(), True),
              StructField("continent", StringType(), True),
              StructField("iso_country", StringType(), True),
              StructField("iso_region", StringType(), True),
              StructField("municipality", StringType(), True),
              StructField("scheduled_service", StringType(), True),
              StructField("gps_code", StringType(), True),
              StructField("iata_code", StringType(), True),
              StructField("local_code", StringType(), True),
              StructField("home_link", StringType(), True),
              StructField("wikipedia_link", StringType(), True),
              StructField("keywords", StringType(), True)]

    schema = StructType(fields)

    data_frame = spark_session \
        .read \
        .format("csv") \
        .schema(schema)\
        .load("data/airports.csv")

    data_frame.printSchema()
    data_frame.show()

if __name__ == '__main__':
    main()
```

Spark applications: Dataframe API

- Basic dataframe operations

Operación	Funcionalidad
show	Impresión por pantalla
printSchema	Impresión del esquema por pantalla
select	Selecciona una columna
filter	Filtra filas según una condición
head(n)	Devuelve las n primeras filas
count()	Cuenta el número de filas

http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrame

Spark applications: Dataframe API

- It is possible to make SQL queries on dataframes

```python
from pyspark.sql import SparkSession

def main() -> None:

    spark_session = SparkSession \
        .builder \
        .getOrCreate()

    logger = spark_session._jvm.org.apache.log4j
    logger.LogManager.getLogger("org").setLevel(logger.Level.WARN)

    data_frame = spark_session\
        .read\
        .json("data/primer-dataset.json")

    data_frame.createOrReplaceTempView("restaurants")

    sql_data_frame = spark_session.sql("SELECT * FROM restaurants")
    sql_data_frame.show()


if __name__ == '__main__':
    main()
```