Machine Learning – Apache Spark

Week 14 – Part 1 – Big Data Processing using Apache Spark

CS 457 - L1 Data Science

Zeehasham Rasheed

Outline

- What is Apache Spark?
- Where Big Data Comes From?
- The Structure Spectrum
- Apache Spark and DataFrames
- Transformations and Actions

Objectives

- Understand Apache Spark's history and development
- Understand the conceptual model: **<u>DataFrames</u>**
- Transformations, data analytics and visualization using
  - **<u>pySpark</u> and <u>SparkSQL</u>**

Prerequisites

Basic programming skills and experience

Some experience with Python or R

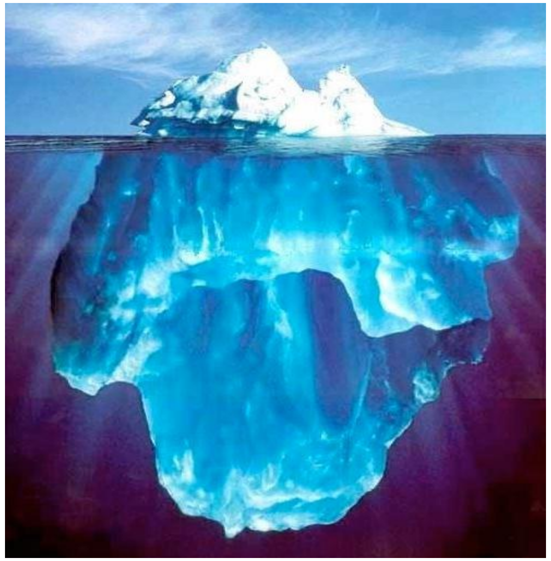
What is Apache Spark?

Scalable, efficient framework for analyzing Big Data

Where Does Big Data Come From?

It's all happening online – could record every:

» Click
» Ad impression
» Billing event
» Fast Forward, pause,…
» Server request
» Transaction
» Network message
» Fault
» …



Where Does Big Data Come From?

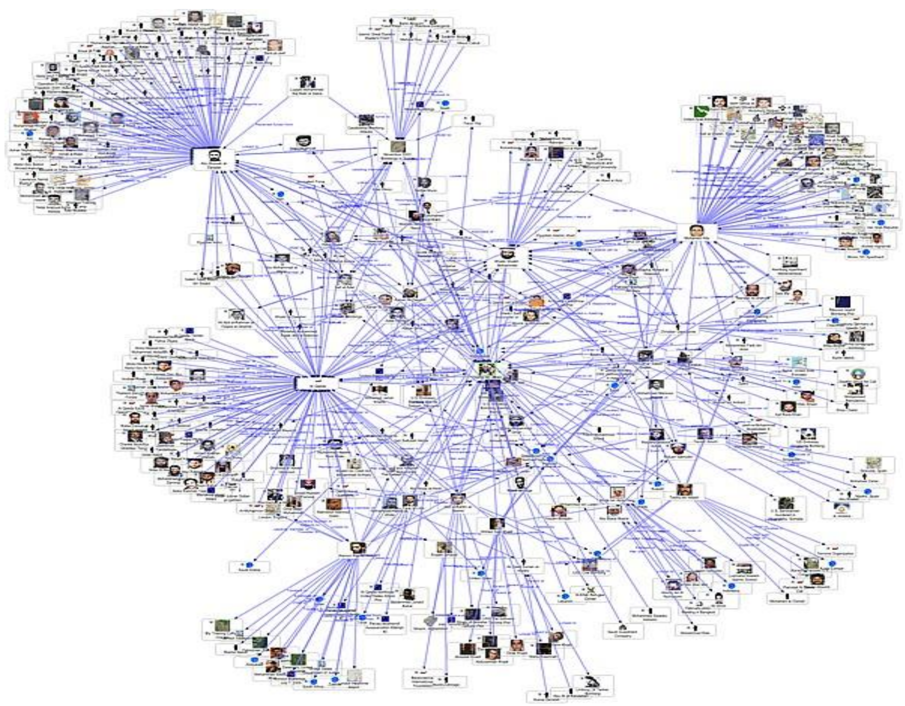
User Generated Content (Web & Mobile)

- » Facebook
- » Instagram
- » Yelp
- » TripAdvisor
- » Twitter
- » YouTube
- » …

Graph Data

Lots of interesting data has a graph structure:

- Social networks
- Telecommunication Networks
- Computer Networks
- Road networks
- Collaborations/Relationships
- …



Some of these graphs can get quite large

(e.g., Facebook user graph)

Log Files – Apache Web Server Log

ix,esc,ca2,07.ix.netcom.com , , [01/Aug/1995:00:00:09 ,0400] "GET /images/launch, logo.gif HTTP/1.0" 200 1713

uplherc.upl.com , , [01/Aug/1995:00:00:10 ,0400] "GET /images/WORLD,logosmall.gif HTTP/ 1.0" 304 0

slppp6.intermind.net , , [01/Aug/1995:00:00:10 ,0400] "GET /history/skylab/skylab.html HTTP/1.0" 200 1687

piweba4y.prodigy.com , , [01/Aug/1995:00:00:10 ,0400] "GET /images/launchmedium.gif HTTP/1.0" 200 11853

tampico.usc.edu , , [14/Aug/1995:22:57:13 ,0400] "GET /welcome.html HTTP/1.0" 200 790

uplherc.upl.com , , [01/Aug/1995:00:00:07 ,0400] "GET / HTTP/1.0" 304 0

uplherc.upl.com , , [01/Aug/1995:00:00:08 ,0400] "GET /images/ksclogo,medium.gif HTTP/ 1.0" 304 0

uplherc.upl.com , , [01/Aug/1995:00:00:08 ,0400] "GET /images/MOSAIC,logosmall.gif HTTP/1.0" 304 0

uplherc.upl.com , , [01/Aug/1995:00:00:08 ,0400] "GET /images/USA,logosmall.gif HTTP/ 1.0" 304 0

Machine Syslog File

```
dhcp,47,129:CS100_1> syslog ,w 10
Feb 3 15:18:11 dhcp,47,129 Evernote[1140] <Warning>: ,[EDAMAccounting read:]: unexpected field ID 23 with type 8. Skipping.
Feb 3 15:18:11 dhcp,47,129 Evernote[1140] <Warning>: ,[EDAMUser read:]: unexpected field ID 17 with type 12.
    Skipping.
Feb 3 15:18:11 dhcp,47,129 Evernote[1140] <Warning>: , [EDAMAuthenticationResult read:]: unexpected field ID 6 with type
11. Skipping.
Feb 3 15:18:11 dhcp,47,129 Evernote[1140] <Warning>: , [EDAMAuthenticationResult read:]: unexpected field ID 7 with type
11. Skipping.
Feb 3 15:18:11 dhcp,47,129 Evernote[1140] <Warning>: ,[EDAMAccounting read:]: unexpected field ID 19 with type 8. Skipping.
Feb 3 15:18:11 dhcp,47,129 Evernote[1140] <Warning>: ,[EDAMAccounting read:]: unexpected field ID 23 with type 8. Skipping.
Feb 3 15:18:11 dhcp,47,129 Evernote[1140] <Warning>: ,[EDAMUser read:]: unexpected field ID 17 with type 12.
    Skipping.
Feb 3 15:18:11 dhcp,47,129 Evernote[1140] <Warning>: ,[EDAMSyncState read:]: unexpected field ID 5 with type 10. Skipping.
Feb 3 15:18:49 dhcp,47,129 com.apple.mtmd[47] <Notice>: low priority thinning
needed for volume Macintosh HD(/) with 18.9 <= 20.0 pct free space
```

Internet of Things: RFID tags

California FasTrak Electronic Toll Collection transponder

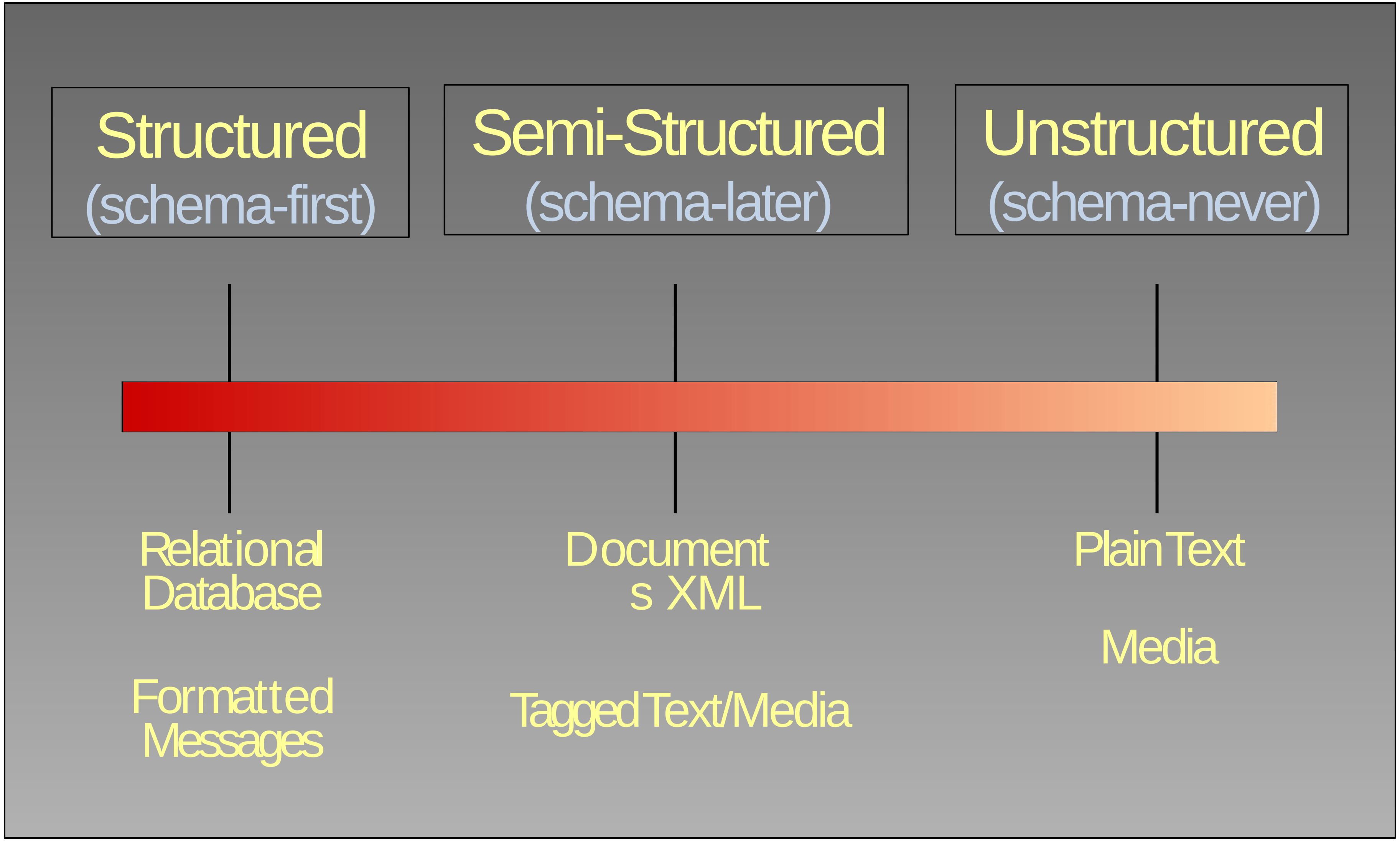
Used to pay tolls

Collected data also used for traffic reporting

http://www.511.org/

http://en.wikipedia.org/wiki/FasTrak

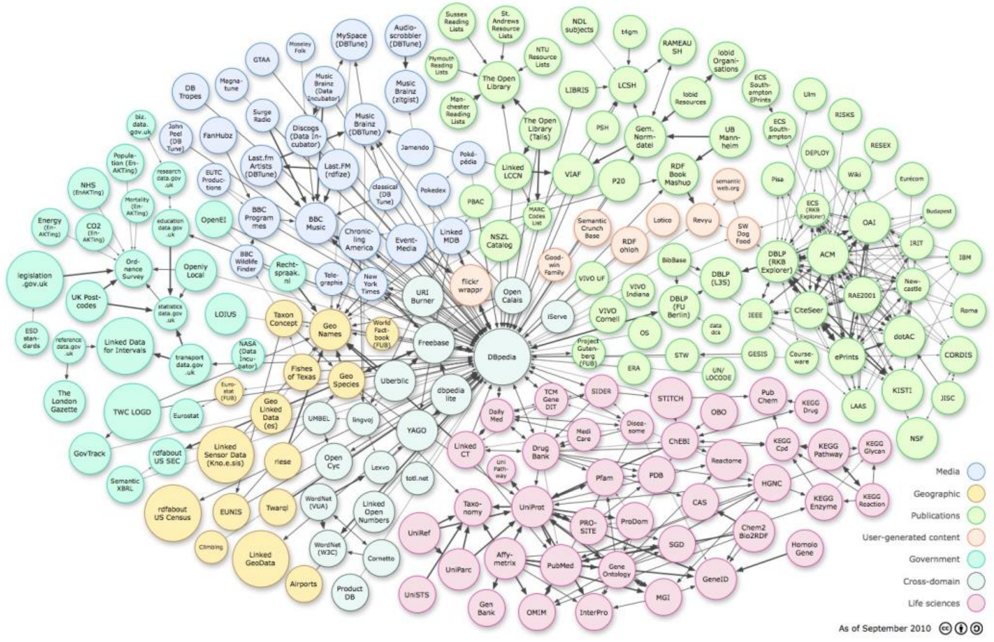
The Structure Spectrum



Whither Structured Data?

Conventional Wisdom:

» Only 20% of data is structured



Decreasing due to:

» Consumer applications
» Enterprise search
» Media applications

http://upload.wikimedia.org/wikipedia/commons/2/23/Lod-datasets_2010-09-22_colored.png

Unstructured Data

Only one column with string or binary type Examples:

» Facebook post

» Instagram image

» Youtube video

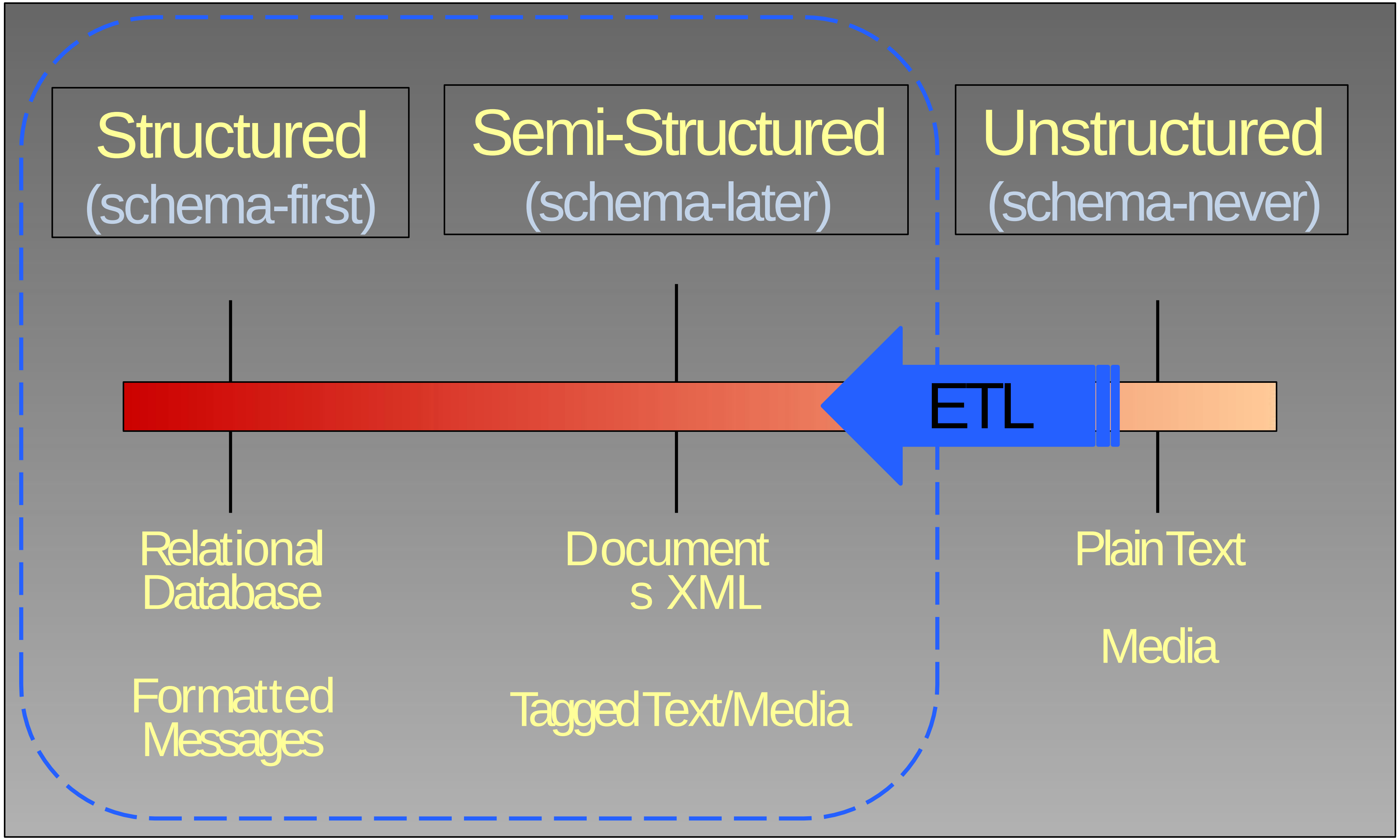
» Blog post

» News article

» User Generated Content

» …

The Structure Spectrum



Extract-Transform-Load (ETL)

- Imposes structure on unstructured data

The Big Data Problem

Data growing faster than computation speeds

Growing data sources

» Web, mobile, scientific, …

Storage getting cheaper

» Size doubling every 18 months

But, stalling CPU speeds and storage bottlenecks

Traditional Tools

» Unix shell commands (grep, awk, sed), pandas, R

(All run on a single machine!)

Big Data Examples

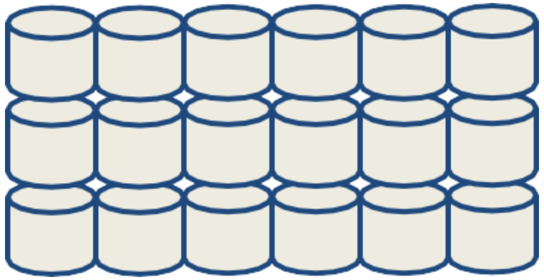
- Facebook's daily logs: 60 TB
- 1,000 genomes project: 200 TB
- Google web index: 10+ PB

- Cost of 1 TB of disk: ~$35
- Time to read 1 TB from disk: 3 hours (100 MB/s)
-

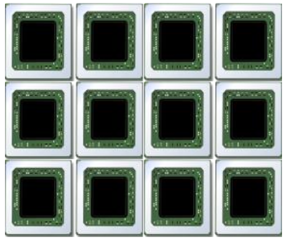
The Big Data Problem

One machine can not process or even store all the data!

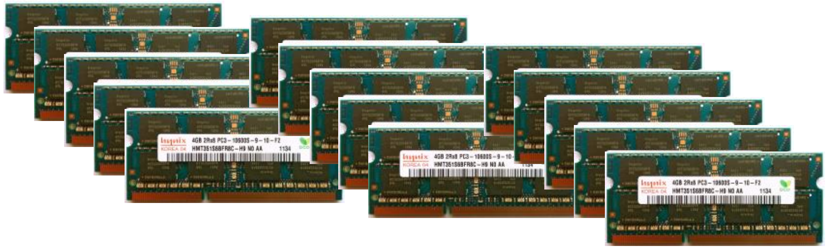
Solution is to distribute data over cluster of machines



Lots of hard drives



… and CPUs



… and memory!

End of Part 1

Machine Learning – Apache Spark

Week 14 – Part 2 – Apache Spark Framework

CS 457 - L1 Data Science

Zeehasham Rasheed

The Spark Computing Framework

Provides programming abstraction and parallel runtime to hide complexities of fault-tolerance and slow machines

Apache Spark Components

Spark SQL

Spark Streaming

MLlib & ML (machine learning)

GraphX (graph)

Apache Spark

Real World Spark Analysis Use Cases

- Big Data Genomics using ADAM API
- Conviva optimizing Internet video stream delivery
- Data processing for wearables and Internet of Things
- Personalized Yahoo! news pages
- Analytics for Yahoo! advertising
- Capital One product recommendations

Python Spark (pySpark)

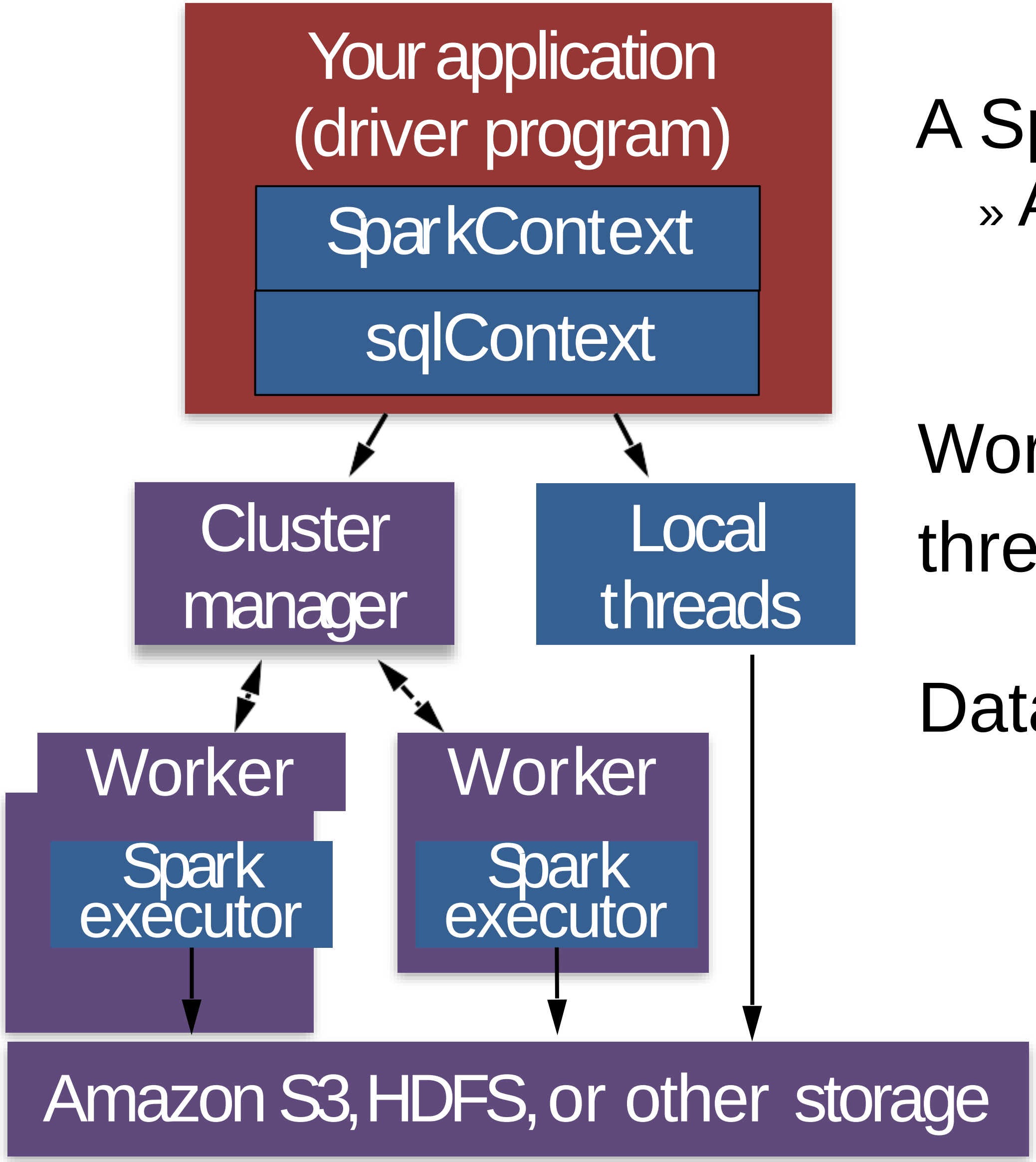
"Here's an operation, run it on all of the data"

DataFrames are the key concept

We can use Python programming interface to Spark (**pySpark**)

pySpark provides an easy-to-use programming abstraction and parallel runtime:

Spark Driver and Workers



A Spark program is two programs:

» A driver program and a workers program

Worker programs run on cluster nodes or in local threads

DataFrames are distributed across workers

Spark and SQL Contexts

A Spark program first creates a SparkContext object

» SparkContext tells Spark how and where to access a cluster

» pySpark shell, Databricks CE automatically create SparkContext

» Jupyter, iPython programs must create a new SparkContext

Data Frames

The primary abstraction in Spark

- Immutable once constructed
- Track lineage information to efficiently recompute lost data
- Enable operations on collection of elements in parallel

You construct DataFrames

- by *parallelizing* existing Python collections (lists)
- by *transforming* an existing Spark or pandas DFs
- from *files* in HDFS or any other storage system

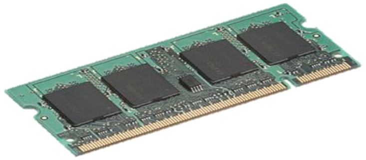
Distributed Memory

Big Data

Word	Index	Count
I	0	1
am	2	1
Sam	5	1
I	9	1
am	11	1
Sam	14	1

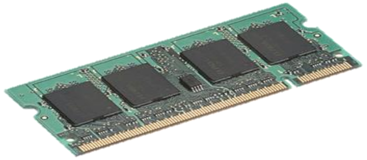
I	0	1
am	2	1

Partition 1



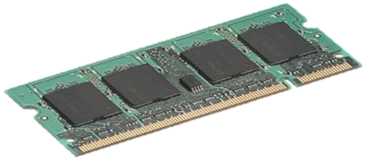
Sam	5	1
I	9	1

Partition 2



am	11	1
Sam	14	1

Partition 3



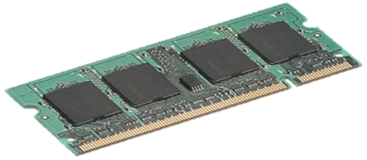
Spark DataFrames

Big Data

Word	Index	Count
I	0	1
am	2	1
Sam	5	1
I	9	1
am	11	1
Sam	14	1

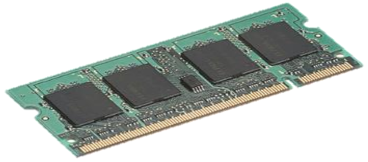
DataFrame

Partition 1



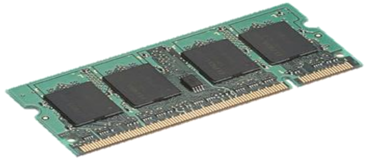
I	0	1
am	2	1

Partition 2



Sam	5	1
I	9	1

Partition 3



am	11	1
Sam	14	1

Data Frames

Each row of a DataFrame is a Row object

The 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)
```

DataFrames

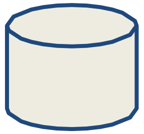
Two types of operations: *transformations* and *actions*

Transformations are **lazy** (*not computed immediately*)

Transformed DF is executed when action runs on it

Persist (cache) DFs in memory or disk

Working with DataFrames

Create a DataFrame from a data source:  <list>

Apply *transformations* to a DataFrame: filter select

Apply *actions* to a DataFrame: show count

show action causes **createDataFrame**, **filter**, and **select** transforms to be executed

Creating DataFrames

The entry point into all relational functionality in Spark is the **SQLContext** class, or one of its decedents. To create a basic SQLContext, all you need is a **SparkContext**

SCALA

```scala
val sc: SparkContext // An existing SparkContext.
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
```

JAVA

```java
JavaSparkContext sc = ...; // An existing JavaSparkContext.
SQLContext sqlContext = new org.apache.spark.sql.SQLContext(sc);
```

Python

```python
from pyspark.sql import SQLContext

sqlContext = SQLContext(sc)
```

R

```r
sqlContext <- sparkRSQL.init(sc)
```

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

Creating DataFrames

Easy to create pySpark DataFrames from pandas (and R) DataFrames

```
# Create a Spark DataFrame from Pandas
>>> spark_df = sqlContext.createDataFrame(pandas_df)
```

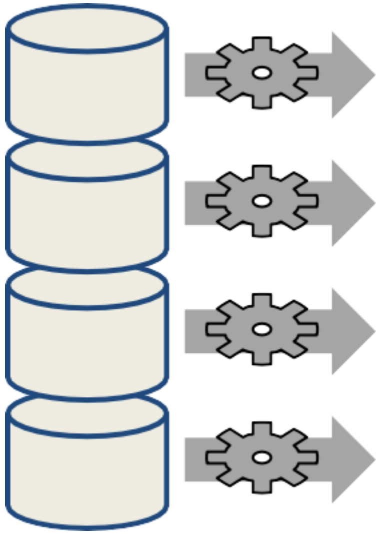
Creating DataFrames

- From HDFS, text files, JSON files, Apache Parquet, Hypertable, Amazon S3, Apache Hbase, SequenceFiles, any other Hadoop Input Format

```
>>> df = sqlContext.read.text("README.txt")
>>> df.collect()
[Row(value=u'hello'), Row(value=u'this')]
```

Creating a DataFrame from a File

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

End of Part 2

Machine Learning – Apache Spark

Week 14 – Part 3 – Spark (Transformation Operations)

CS 457 - L1 Data Science

Zeehasham Rasheed

Spark Transformations

Create new DataFrame from an existing one

Use *lazy evaluation*: results not computed right away – Spark remembers set of transformations applied to base DataFrame

- » Spark uses *Catalyst* to optimize the required calculations
- » Spark recovers from failures and slow workers

Think of this as a recipe for creating result

Column Transformations

The apply method creates a DataFrame from one column:

```
>>> ageCol  =  df.age
```

Column Transformations

The apply method creates a DataFrame from one column:

```
>>> ageCol  =  df.age
```

You can select one or more columns from a DataFrame:

```
>>> df.select('*')
```

* selects all the columns

Column Transformations

The apply method creates a DataFrame from one column:

```
>>> ageCol = df.age
```

You can select one or more columns from a DataFrame:

```
>>> df.select('*')
```

* selects all the columns

```
>>> df.select('name’, 'age')
```

* selects the name and age columns

Column Transformations

The apply method creates a DataFrame from one column:

```
>>> ageCol = df.age
```

You can select one or more columns from a DataFrame:

```
>>> df.select('*')
```

  * selects all the columns

```
>>> df.select('name', 'age')
```

  * selects the name and age columns

```
>>> df.select(df.name,
                    (df.age + 10).alias('age'))
```

  * selects the name and age columns,
    increments the values in the age column by 10, and renames (alias) the age +10 column as age

More Column Transformations

The drop method returns a new DataFrame that drops the specified column:

```
>>> df.drop(df.age)
[Row(name=u'Alice'), Row(name=u'Bob')]
```

Review: Python lambda Functions

- Small anonymous functions (not bound to a name)

```
lambda a, b: a + b
```

  » returns the sum of its two arguments

- Can use lambda functions wherever function objects are required
- Restricted to a single expression

User Defined Function Transformations

Transform a DataFrame using a User Defined Function

```
>>>     from pyspark.sql.types import IntegerType
>>>     slen =     udf(lambda s: len(s), IntegerType())
>>>     df.select(slen(df.name).alias('slen'))
```

* Creates a DataFrame of [Row(slen=5), Row(slen=3)]

Remember the input
[Row(name=u'Alice'), Row(name=u'Bob')]

UDF takes named or lambda function and the return type of the function

Other 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

Using Transformations (I)

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

Using Transformations (I)

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

>>> from pyspark.sql.types import IntegerType
>>> doubled = udf(lambda s: s * 2, IntegerType())
>>> df2 = df.select(df.name, doubled(df.age).alias('age'))
[Row(name=u'Alice', age=2), Row(name=u'Bob', age=4)]
```

* selects the name and age columns, applies the UDF to age column
and aliases resulting column to age

Using Transformations (I)

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

>>> from pyspark.sql.types import IntegerType
>>> doubled = udf(lambda s: s * 2, IntegerType())
>>> df2 = df.select(df.name, doubled(df.age).alias('age'))
[Row(name=u'Alice', age=2), Row(name=u'Bob', age=4)]
```

* selects the name and age columns, applies the UDF to age column and aliases resulting column to age

```
>>> df3 = df2.filter(df2.age > 3)
[Row(name=u'Bob', age=4)]
```

* only keeps rows with age column greater than 3

Using Transformations (II)

```
>>> data2 = [('Alice', 1), ('Bob', 2), ('Bob', 2)]

>>> df = sqlContext.createDataFrame(data2, ['name', 'age'])
[Row(name=u'Alice', age=1), Row(name=u'Bob', age=2),
     Row(name=u'Bob', age=2)]

>>> df2 = df.distinct()
[Row(name=u'Alice',age=1), Row(name=u'Bob', age=2)]
```

* only keeps rows that are distinct

Using Transformations (II)

```
>>> data2 = [('Alice', 1), ('Bob', 2), ('Bob', 2)]
>>> df = sqlContext.createDataFrame(data2, ['name', 'age'])
[Row(name=u'Alice', age=1), Row(name=u'Bob', age=2),
    Row(name=u'Bob', age=2)]

>>> df2 = df.distinct()
[Row(name=u'Alice', age=1), Row(name=u'Bob', age=2)]
```

* only keeps rows that are distinct

```
>>> df3 = df2.sort("age", ascending=False)
[Row(name=u'Bob', age=2), Row(name=u'Alice', age=1)]
```

* sort ascending on the age column

Using Transformations (III)

```
>>> data3 = [Row(a=1, intlist=[1,2,3])]
>>> df4 = sqlContext.createDataFrame(data3)
[Row(a=1, intlist=[1,2,3])]

>>> df4.select(explode(df4.intlist).alias("anInt"))
[Row(anInt=1), Row(anInt=2), Row(anInt=3)]
```

* turn each element of the intlist column into a Row, alias the resulting column to an Int, and select only that column

Grouped Data Transformations

groupBy(*cols*)

groups the DataFrame using the specified columns, so we can run aggregation on them

GroupedData Function	Description
agg(*exprs*)	Compute aggregates (avg, max, min, sum, or count) and returns the result as a DataFrame
count()	counts the number of records for each group
avg(*args*)	computes average values for numeric columns for each group

Using Grouped Data (I)

```
>>> data = [('Alice',1,6), ('Bob',2,8), ('Alice',3,9), ('Bob',4,7)]
>>> df = sqlContext.createDataFrame(data, ['name', 'age', 'grade'])
>>> df1 = df.groupBy(df.name)
>>> df1.agg({"*": "count"}).collect()
[Row(name=u'Alice', count(1)=2), Row(name=u'Bob', count(1)=2)]
```

Using GroupedData (I)

```
>>> data = [('Alice',1,6), ('Bob',2,8), ('Alice',3,9), ('Bob',4,7)]
>>> df = sqlContext.createDataFrame(data, ['name', 'age', 'grade'])
>>> df1 = df.groupBy(df.name)
>>> df1.agg({"*": "count"}).collect()
[Row(name=u'Alice', count(1)=2), Row(name=u'Bob', count(1)=2)]

>>> df.groupBy(df.name).count()
[Row(name=u'Alice', count=2), Row(name=u'Bob', count=2)]
```

Using Grouped Data (II)

```
>>> data = [('Alice',1,6), ('Bob',2,8), ('Alice',3,9), ('Bob',4,7)]
>>> df = sqlContext.createDataFrame(data, ['name', 'age', 'grade'])
>>> df.groupBy().avg().collect()
[Row(avg(age)=2.5, avg(grade)=7.5)]
```

Using Grouped Data (II)

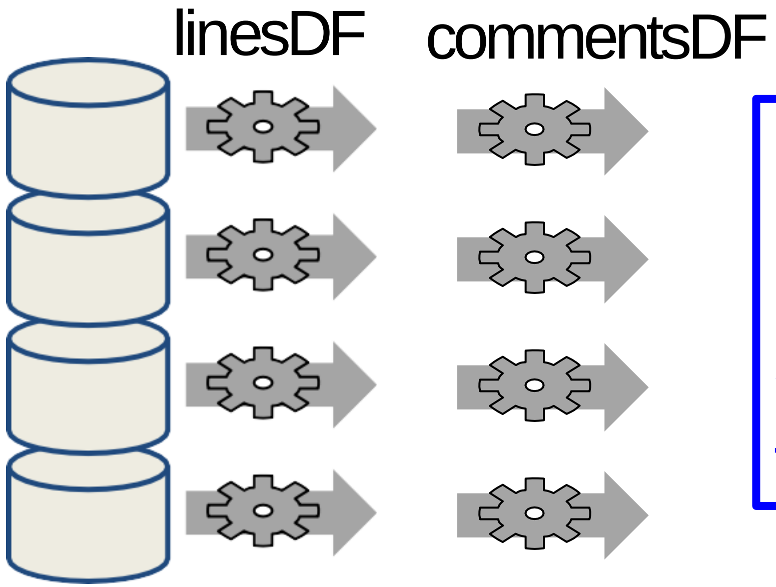
```
>>> data = [('Alice',1,6), ('Bob',2,8), ('Alice',3,9), ('Bob',4,7)]
>>> df = sqlContext.createDataFrame(data, ['name', 'age', 'grade'])
>>> df.groupBy().avg().collect()
[Row(avg(age)=2.5, avg(grade)=7.5)]

>>> df.groupBy('name').avg('age', 'grade').collect()
[Row(name=u'Alice', avg(age)=2.0, avg(grade)=7.5),
     Row(name=u'Bob', avg(age)=3.0, avg(grade)=7.5)]
```

Transforming a DataFrame (Lazy)

```
linesDF = sqlContext.read.text('...')

commentsDF = linesDF.filter(<condition>)
```



Lazy evaluation means nothing executes – Spark saves recipe for transforming source

End of Part 3

Machine Learning – Apache Spark

Week 14 – Part 4 – Spark (Actions Operations)

CS 457 - L1 Data Science

Zeehasham Rasheed

Spark Actions

- Cause Spark to execute recipe to transform source Mechanism for getting results out of Spark

Some Useful Actions

Action	Description
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

+count for DataFrames is an action, while for GroupedData it is a transformation

Getting Data Out of DataFrames (I)

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

Getting Data Out of DataFrames (I)

- >>> df = sqlContext.createDataFrame(data, ['name', 'age'])
- >>> df.collect()

[Row(name=u'Alice', age=1), Row(name=u'Bob', age=2)]

- >>> df.show()

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

>>> df.**count**()

2

Getting Data Out of DataFrames (II)

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

Getting Data Out of DataFrames (II)

- >>> df = sqlContext.createDataFrame(data, ['name', 'age'])
- >>> df.take(1)

[Row(name=u'Alice', age=1)]

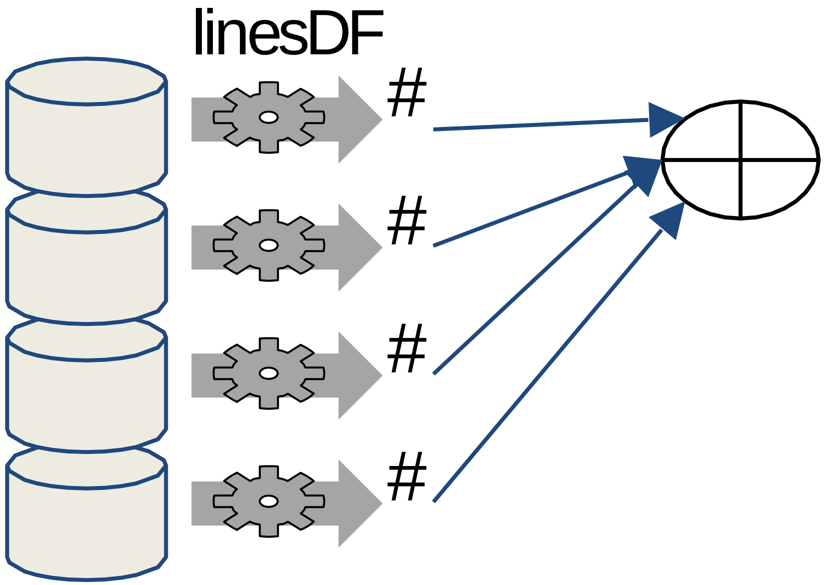
- >>> df.describe()

```
+,,,,,,,+,,,,,,,,,,,,,,,,,,+
|summary|        age|
+,,,,,,,+,,,,,,,,,,,,,,,,,,+
|    count|        2|
|      mean|    1.5|
|  stddev   |    0.7071067811865476|
|        min|                    1|
|        max|                    2|
+,,,,,,,,+,,,,,,,,,,,,,,,,,,,+
```

Spark Programming Model

```
linesDF = sqlContext.read.text('...')

print linesDF.count()
```

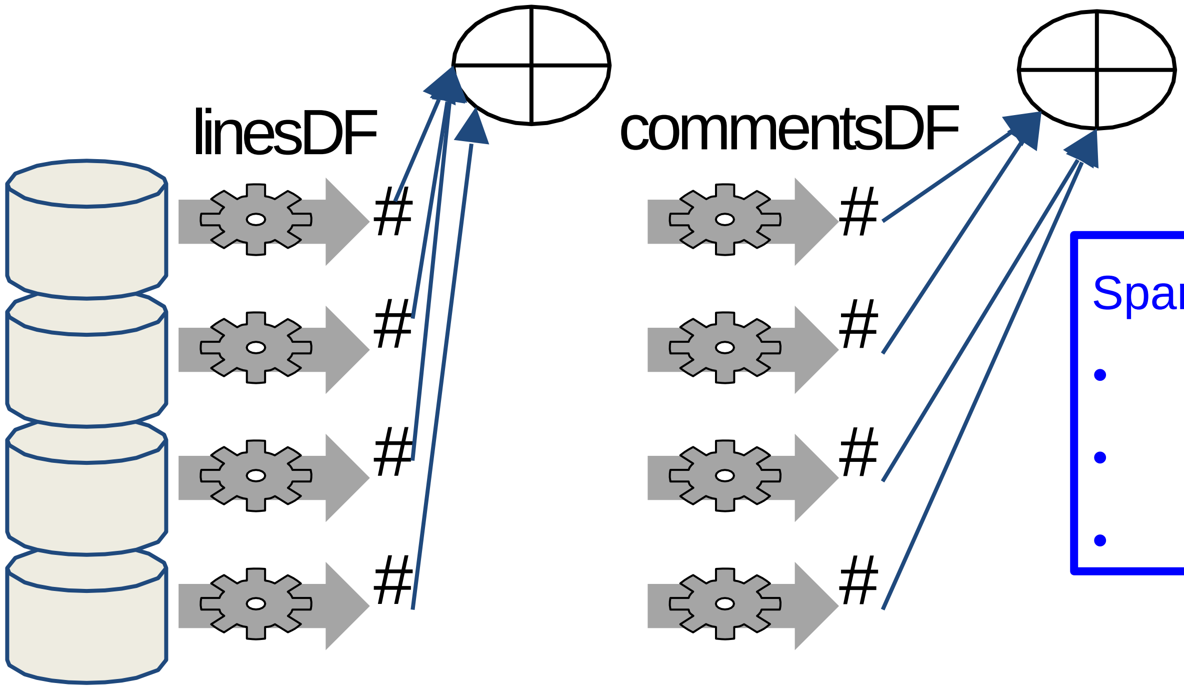


count() causes Spark to:

- read data
- sum within partitions
- combine sums in driver

Spark Programming Model

```
linesDF = sqlContext.read.text('...')
commentsDF = linesDF.filter(<condition>)
print linesDF.count(), commentsDF.count()
```

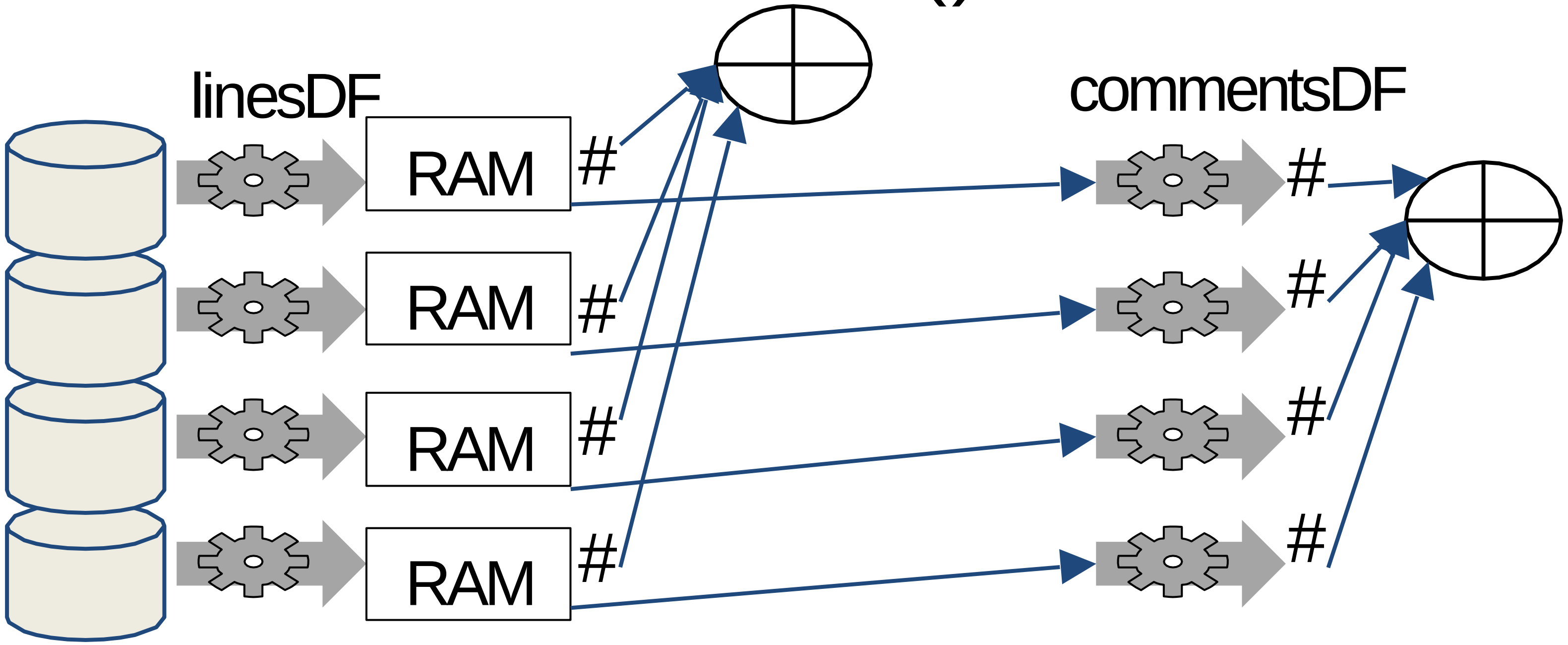


Spark recomputes linesDF:

- read data (again)
- sum within partitions
- combine sums in driver

Caching DataFrames

```
linesDF = sqlContext.read.text('...')
linesDF.cache() # save,don't recompute!
commentsDF = linesDF.filter(<condition>)
Print linesDF.count(),commentsDF.count()
```



Spark Program Lifecycle

1. Create DataFrames from external data or <u>createDataFrame</u> from a collection in driver program
2. Lazily <u>transform</u> them into new DataFrames
3. **cache()** some DataFrames for reuse (help in reducing the cost of recovery)
4. Perform actions to execute parallel computation and produce results

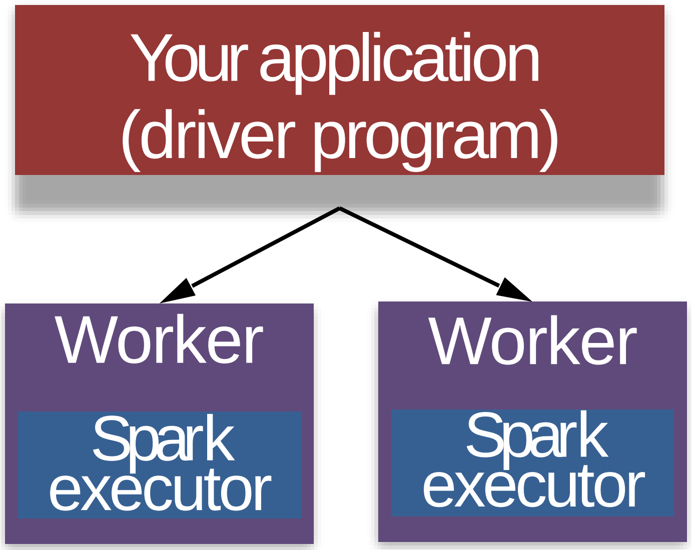
Local or Distributed?

Where does code run?

- » Locally, in the driver
- » Distributed at the executors
- » Both at the driver and the executors

Very important question:

- » Executors run in parallel
- » Executors have much more memory



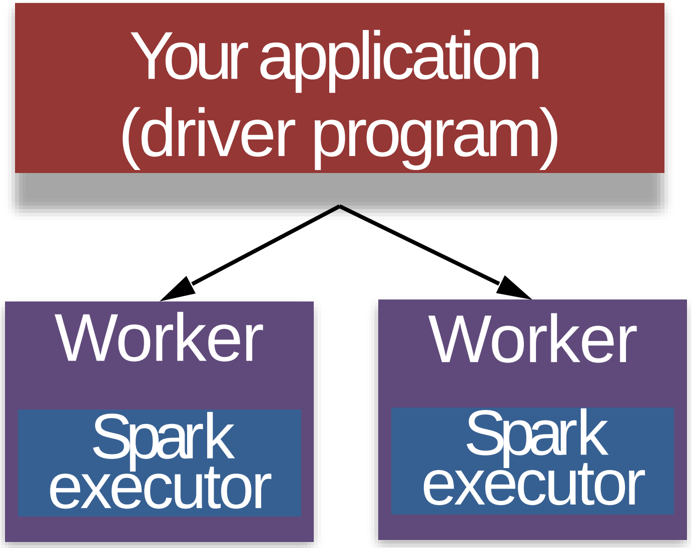
Where Code Runs

Most Python code runs in driver

» Except for code passed to transformations

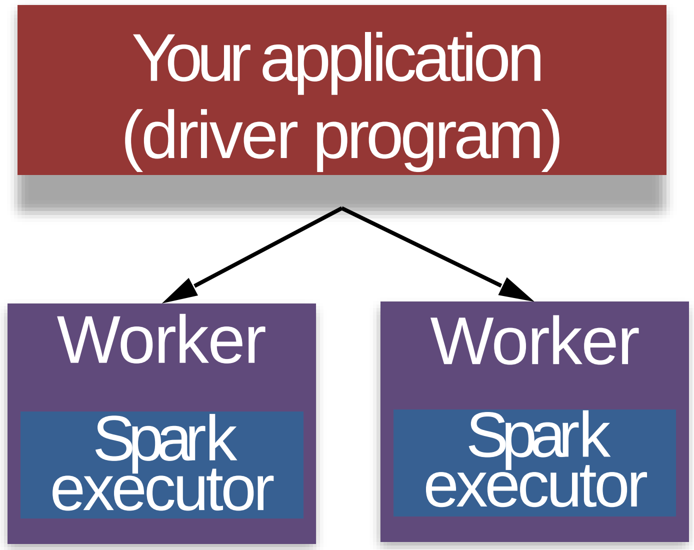
Transformations run at executors

Actions run at executors and driver



Examples

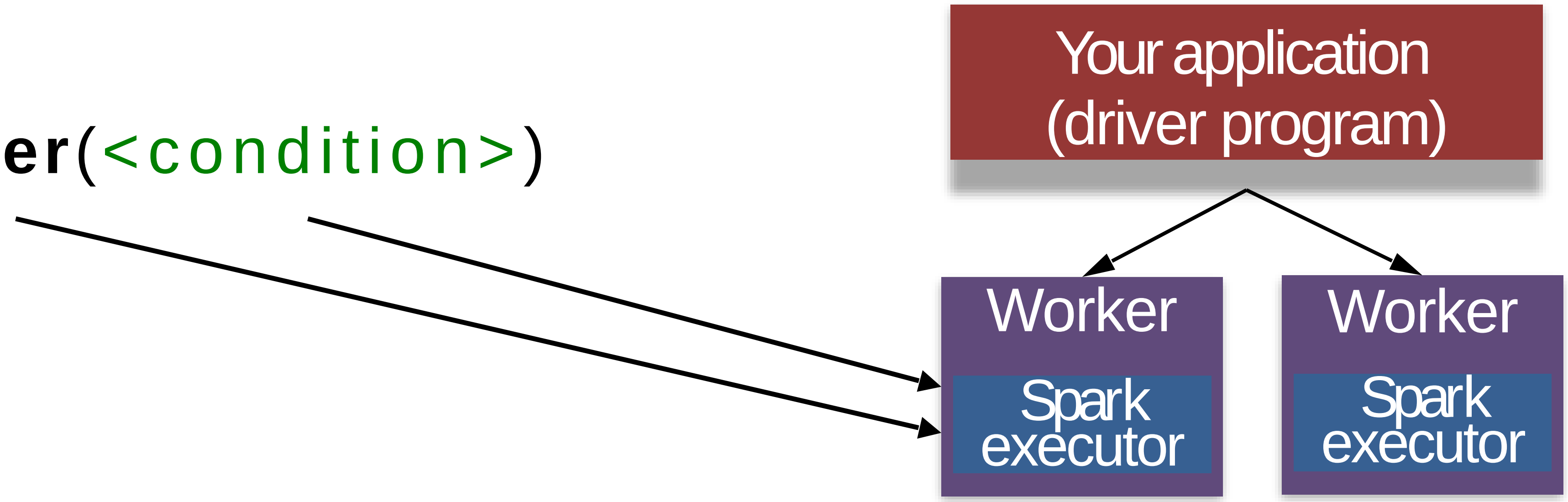
```
>>> a = a + 1
```



Examples

```
>>> a = a + 1

>>> linesDF.filter(<condition>)
```

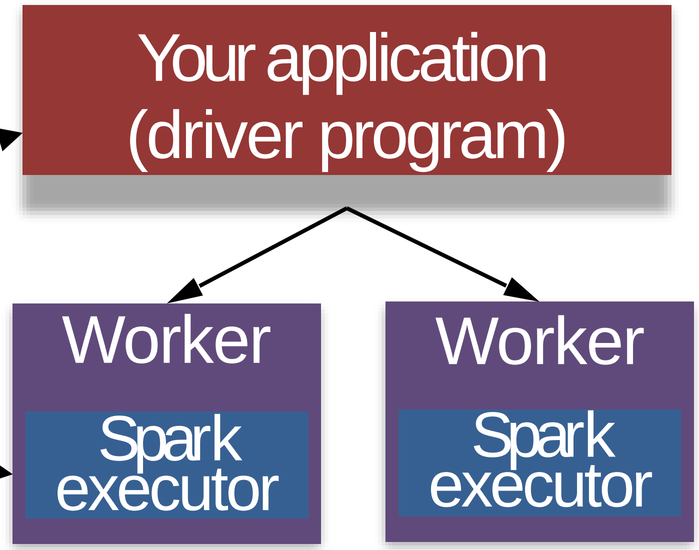


Examples

```
>>> a = a + 1

>>> linesDF.filter(<condition>)

>>> commentsDF.count()
```



How Not to Write Code

Let's say you want to combine two DataFrames: aDF, bDF

You remember that df.collect()

returns a list of Row, and in Python you can combine two lists with +

A naïve implementation would be:

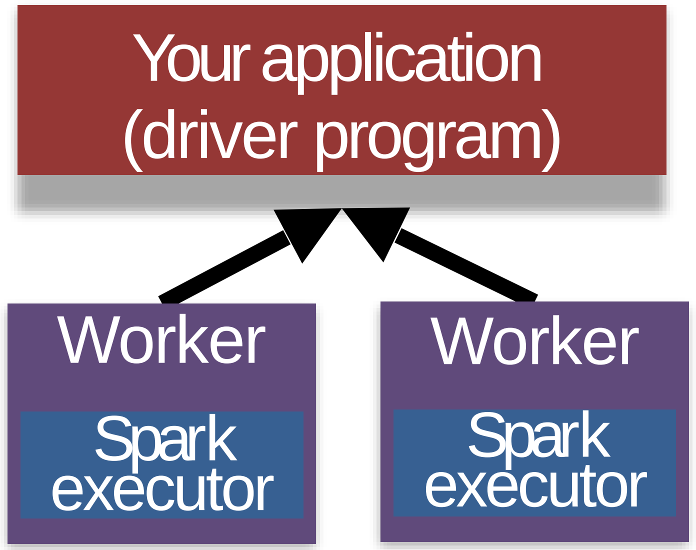
```
>>>  a  =  aDF.collect()
>>>  b  =  bDF.collect()
>>> cDF = sqlContext.createDataFrame(a + b)
```

Where does this code run?

Combine two DataFrames: aDF, bDF

```
>>> a = aDF.collect()
>>> b = bDF.collect()
```

* all distributed data for a and b is sent to driver



What if a and/or b is very large?

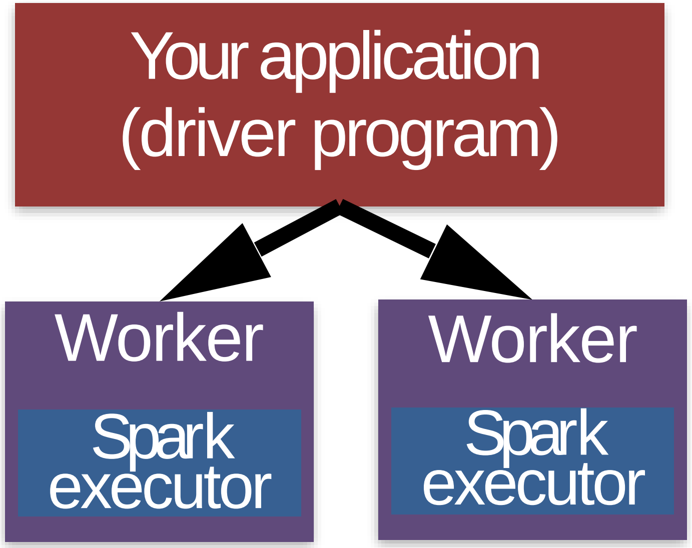
» Driver could run out of memory: Out Of Memory error (OOM)

» Also, takes a long time to send the data to the driver

combine two DataFrames: aDF, bDF

>>> cDF = **sqlContext.createDataFrame**(a + b)

* all data for cDF is sent to the executors



What if the list a + b is very large?

» Driver could run out of memory:

Out Of Memory error (OOM)

» Also, takes a long time to send the

data to executors

The Best Implementation

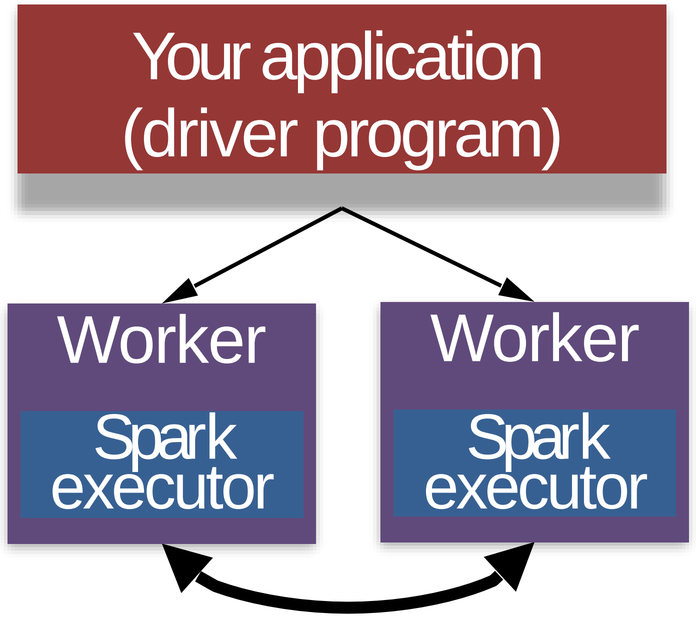
\>>> cDF = aDF.[unionAll](bDF)

Use the DataFrame reference API!

» unionAll(): “Return a new DataFrame containing union of rows in this frame and another frame”

» Runs <u>completely</u> at executors:

- Very scalable and efficient



Some Programming Best Practices

Use Spark Transformations and Actions wherever possible

» Search DataFrame reference API

Never use collect() in production, instead use take(*n*)

cache() DataFrames that you reuse a lot

Apache Spark References

http://spark.apache.org/docs/latest/programming-guide.html
http://spark.apache.org/docs/latest/api/python/index.html
http://spark.apache.org/docs/latest/api/python/pyspark.sql.html

SparkML

- **spark.ml** is a new package introduced in Spark 1.2
  - Aims to provide a uniform set of high-level APIs that help users create and tune practical machine learning pipelines.

End of Part 4