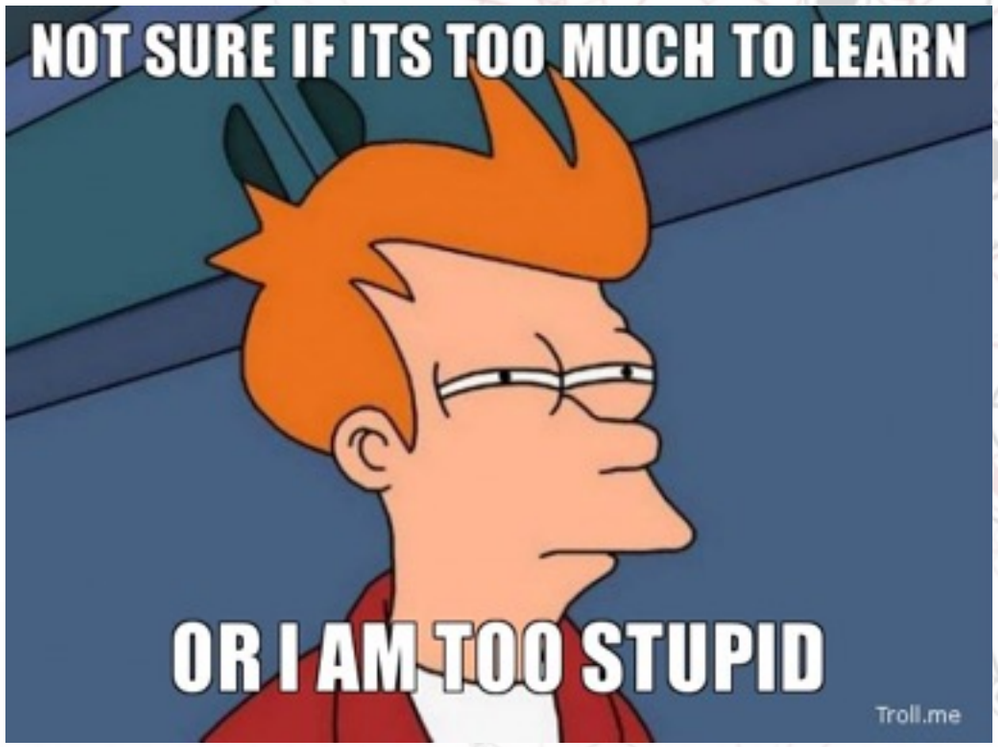
BigData

Iniciando meu estudo com Grandes Dados

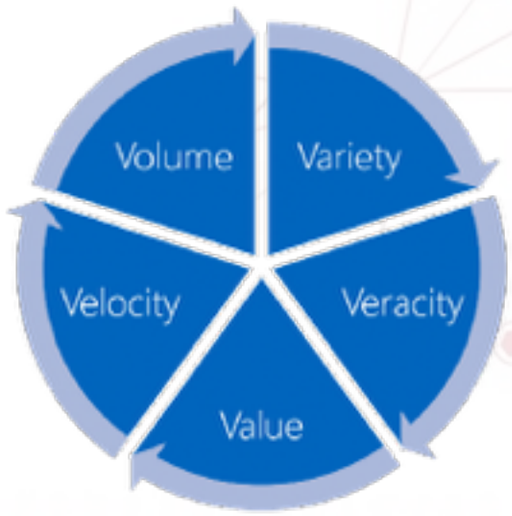
email: susana.bouchardet@gmail.com

Algumas palavras que escutei…
Dessas qual é a mais importante?
Muita coisa nova pra mim.



Porque me senti assim?
Muitas Palavras novas, não conhecia nada O_O
“é de comer ou de passar no cabelo"

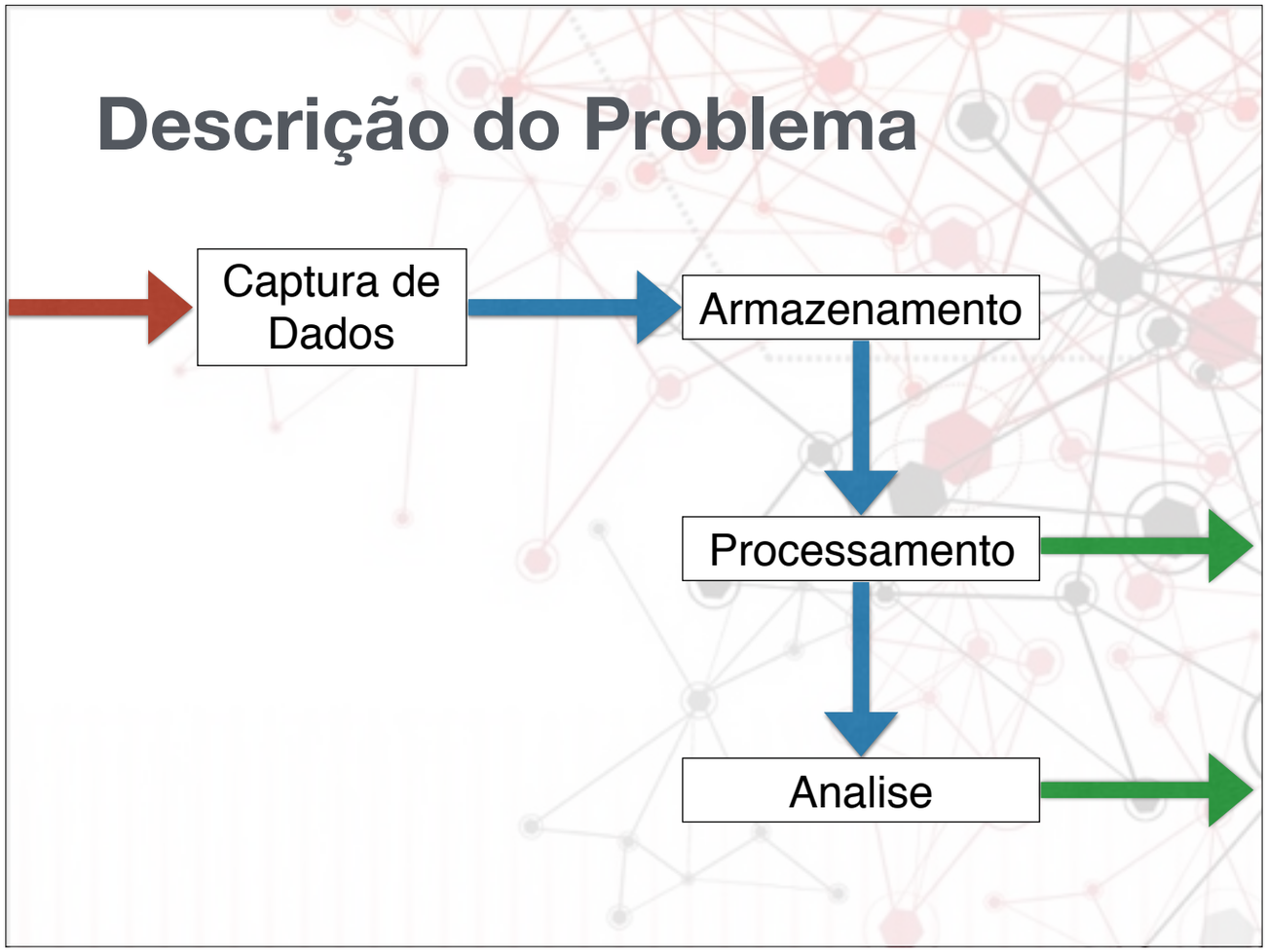
BigData 5V



- **Volume**: Muitos dados, desafio de armazenamento.
- **Variedade**: diferentes formatos de dados estruturados e/ou não.
- **Veracidade**: Acurácia e autenticidade.
- **Valor**: Retorno desses dados para o negócio.
- **Velocidade**: Tempo entre dado gerado e analisado.

falar dos 5Vs
Variedade: hot Data, warm data, cold data

Descrição do Problema



Captura: Como pegar os dados de maneira eficiente?
 - pra onde eu mando eles antes de salvar de fato? (Kafka)
Armazenamento: Salvar esses dados mais “crus”, na maneiro como eles vieram. Raw Data
Processamento: Processar os dados para serem úteis para analises ou alimentar outros serviços (Recomendação).
Analise: Transformar os dados em informações.

GFS

- Publicado em 2003
- **Revolucionário**
- arquivos em **chunks** de 64mb
- cada chunk esta no mínimo em 3 máquina

Google File Sistem
Se algum maquina se perde, a triplicação dos chuncks faz com que ela possa ser reconstruindo a partir de pedacinhos das outras

MapReduce

- Publicado em 2004
- modelo para processar **grandes volumes de dados**.
- Técnica muito usada em linguagens funcionais

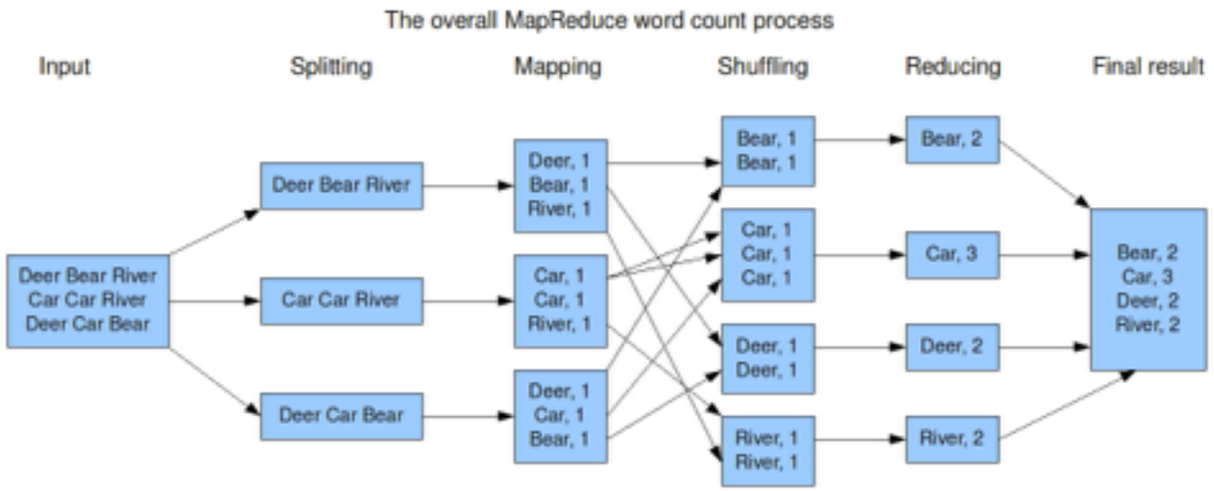
MapReduce

MapReduce

```
1  map(String input_key, String input_value):
2      // input_key: document name
3      // input_value: document contents
4      for each word w in input_value:
5          EmitIntermediate(w, "1");
6
7  reduce(String output_key, Iterator intermediate_values):
8      // output_key: a word
9      // output_values: a list of counts
10     int result = 0;
11     for each v in intermediate_values:
12         result += ParseInt(v);
13     Emit(AsString(result));
```

MapReduce é um Famework feito pelo Google.
Facilitar o processamento de grandes volumes de dados já que trabalha em paralelo.
Tem Basicamente as funções Map e Reduce.

MapReduce

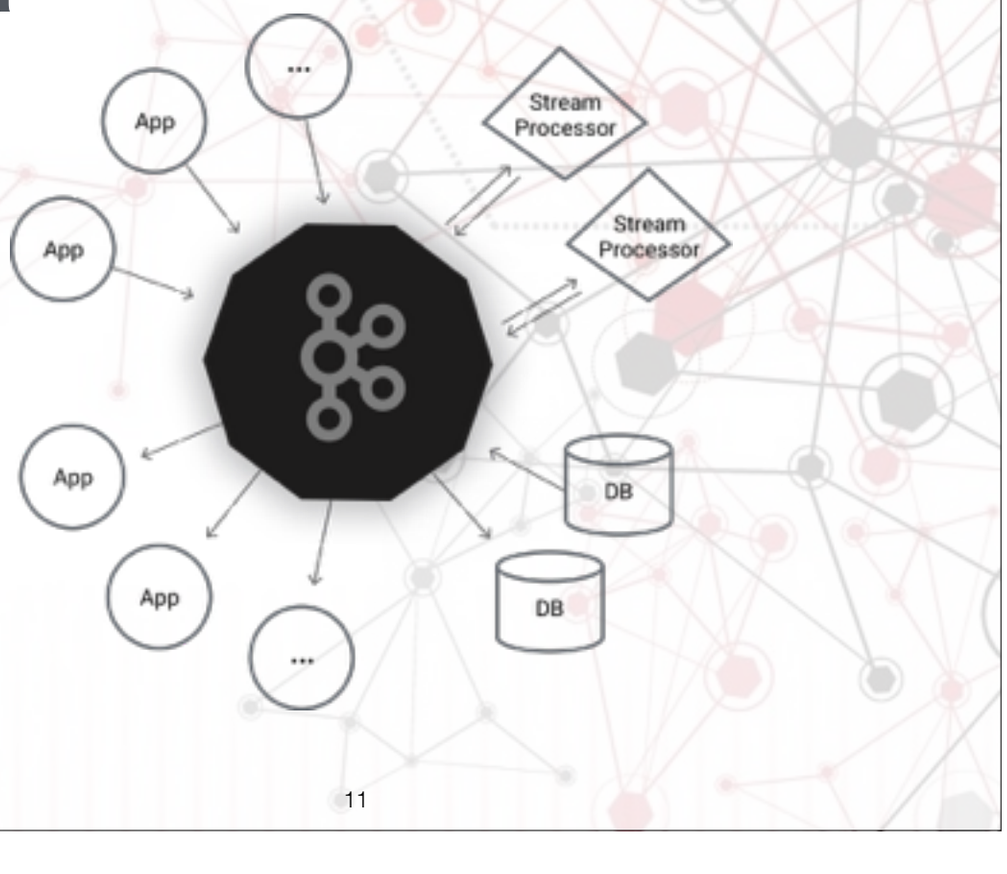

The overall MapReduce word count process
Input
Splitting
Mapping
Shuffling
Reducing
Final result
Deer Bear River
Car Car River
Deer Car Bear
Deer Bear River
Car Car River
Deer Car Bear
Deer, 1
Bear, 1
River, 1
Car, 1
Car, 1
River, 1
Deer, 1
Car, 1
Bear, 1
Bear, 1
Bear, 1
Car, 1
Car, 1
Car, 1
Deer, 1
Deer, 1
River, 1
River, 1
Bear, 2
Car, 3
Deer, 2
River, 2
Bear, 2
Car, 3
Deer, 2
River, 2

Hadoop

- Nasceu no Yahoo em 2006
- HDFS
- HadoopMapReduce
- HadoopYarn
- Bancos de dados e novos Frameworks surgiram baseados nele.

é uma plataforma em Java voltada para computação distribuida.
Beleza. Mas mais uma pergunta.

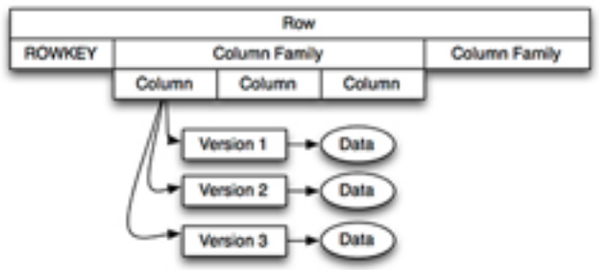
Kafka



Tópico <->Fila
Você cria e consome mensagens dos seus topicos.
espécie de Barramento. ActiveMQ

HBase

- Foi criado para ser usado com Hadoop
- Banco Distribuido
- Distribuído em **Master > RegionServers > Regions**
- Orientado por colunas

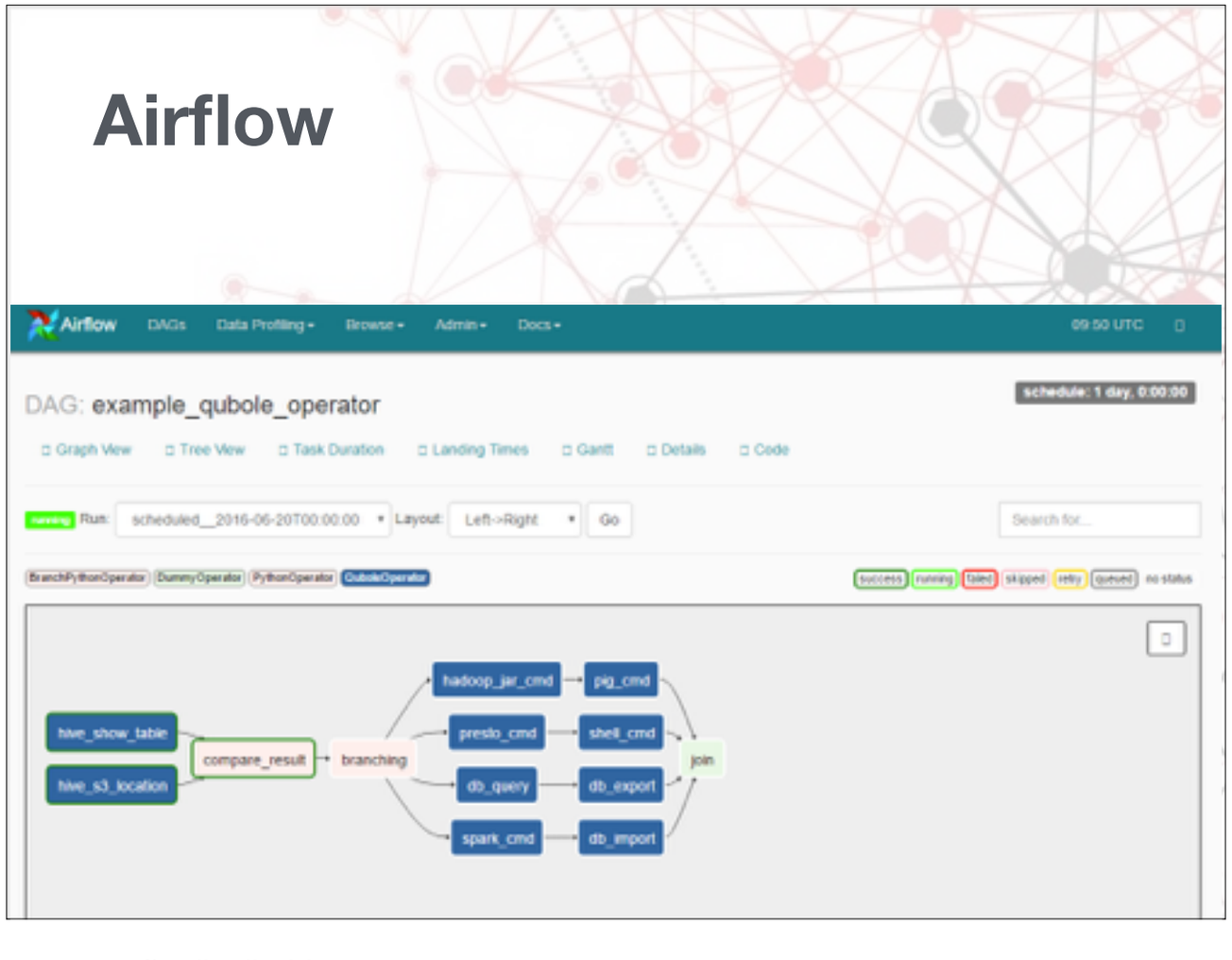


arquivo **REGIONSERVERS** que listas os servers e o primeiro é o master.

Airflow

- Criado em 2014 pelo AirBnB
- Conceit de DAG: sequencia de tarefas a serem executadas em ordem.
- Agendor de DAG's
- Divide a DAG em tasks

é uma plataforma em Java voltada para computação distribuida.
Beleza. Mas mais uma pergunta.



é uma plataforma em Java voltada para computação distribuida.
Beleza. Mas mais uma pergunta.

Será que a Globo.com precisa mesmo?

Exemplos dos dados capturados.
Nos dois últimos falar um pouco sobre a área de recomendação

Números

3 Bilhões de eventos diários

2 Milhões de conexões **simultâneas**

50 Milhões de usuários únicos por mês

100 Mil novos conteúdos por mês

+20 algoritmos diferentes

100 Mil recomendações por minuto

Exemplos dos dados capturados.
Nos dois últimos falar um pouco sobre a área de recomendação

Pra que tudo isso?

SPARK.
DAQUELAS PALAVRAS a mais importante pra mim era o spark

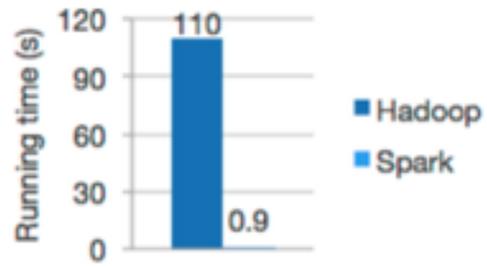
Pra que tudo isso?

- ➤Predição
- ➤Personalização
- ➤Saber mais sobre o público
- ➤Distribuição mais inteligente de publicidade

SPARK.
DAQUELAS PALAVRAS a mais importante pra mim era o spark

Spark

- Nasceu em 2014
- Foco em **velocidade**, facilidade de uso e análises sofisticadas



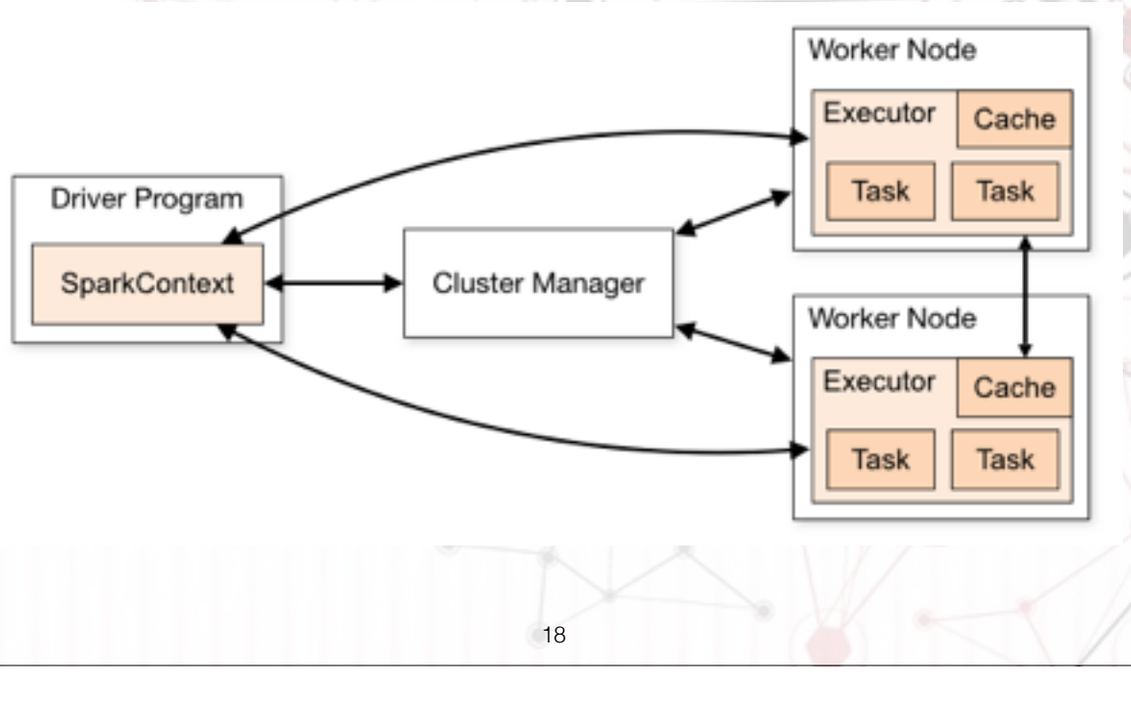
Logistic regression in Hadoop and Spark

- Aplicações até **100 vezes** mais rápido em **memória** e até **10 vezes** mais rápido em **disco**

Framework para processamento de dados em larga escala

Você programa mas ele se preocupa de como as coisas vão ser executadas.

Spark

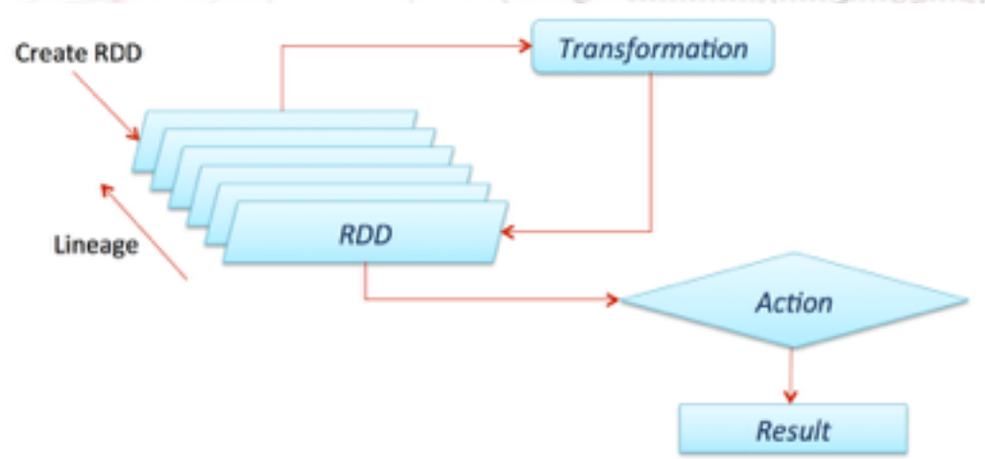


SparkConf: Quanto de Recurso eu quero
o problema do exagero do recurso pensando no pior caso.
SparkContext: Recebe o SparkConf
SQLContex: A maneira de abstrair a manipulação do dado.

O Spark permite que os programadores desenvolvem pipelines compostos por várias etapas complexas usando grafos direcionais acíclicos.
A estrutura que permite esse tipo de pipeline é o RDD

Spark

➤**RDD** (**R**esilient **D**istributed **D**atasets)

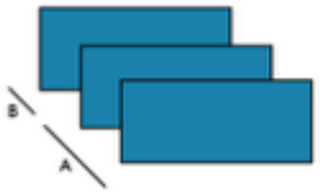


Essa estrutura permite operações iterativas sobre o Dataset.
Se uma query for feita repetidas vezes sobre a mesma estrutura, esse dado pode ser mantido em memória para melhorar o tempo de execução.
Temos também o **Dataframe**, que é bem semelhante. Entretanto o Dataram da mais liberdade do Spark otimizar a execução a Query.

Spark

- RDD

COLLECT

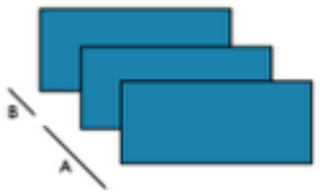


```
11  x = sc.parallelize([1,2,3],2)
12  y = x.collect()
13
14  print x.glom().collect()
15  print y
16
17  sc.stop()
```

Spark

- RDD

COLLECT



```
11  x = sc.parallelize([1,2,3],2)
12  y = x.collect()
13
14  print x.glom().collect()
15  print y
16
17  sc.stop()
```

```
x: [[1],[2,3]]
y:[1,2,3]
```

Spark

➤RDD

GETNUMPARTITIONS


```
11    x = sc.parallelize([1,2,3],2)
12    y = x.getNumPartitions()
13
14    print x.glom().collect()
15    print y
16
```

Spark

➤RDD

GETNUMPARTITIONS

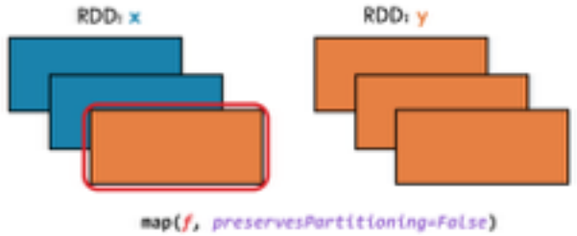

```
11    x = sc.parallelize([1,2,3],2)
12    y = x.getNumPartitions()
13
14    print x.glom().collect()
15    print y
16
```

```
x: [[1],[2,3]]
y: 2
```

Spark

- RDD

MAP



Spark

➤RDD

```
x = sc.parallelize(['a','b','c'])
y = x.map(lambda z:(z,1))

print x.collect()
print y.collect()
```

Spark

➤RDD

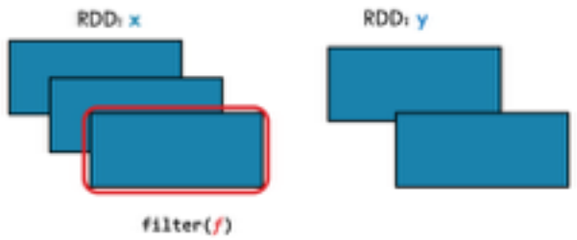
```
x = sc.parallelize(['a','b','c'])
y = x.map(lambda z:(z,1))

print x.collect()
print y.collect()
```

```
x: ['a','b','c']
y: [('a',1),
    ('b',1),
    ('c',1)]
```

Spark

➤RDD

FILTER

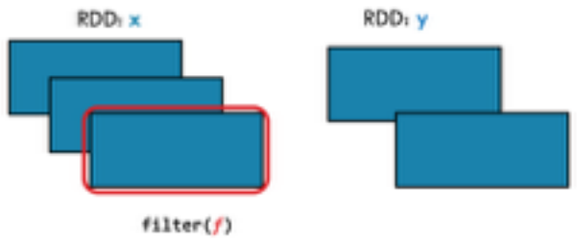



```
11    x = sc.parallelize([1,2,3])
12    y = x.filter(lambda x: x > 1)
13
14    print x.collect()
15    print y.collect()
16
```

Spark

➤RDD

FILTER



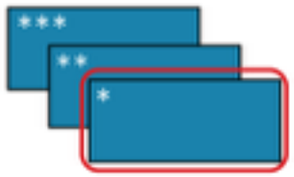

```
11  x = sc.parallelize([1,2,3])
12  y = x.filter(lambda x: x > 1)
13
14  print x.collect()
15  print y.collect()
16
```

```
x: [1,2,3]
y: [2,3]
```

Spark

➤RDD

REDUCE

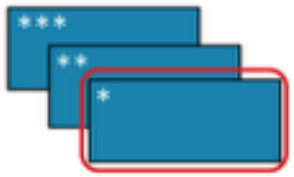



```
11    x = sc.parallelize([1,2,3,4])
12    y = x.reduce(lambda x,y: x + y)
13
14    print x.glom().collect()
15    print y
```

Spark

- RDD

REDUCE




```
11    x = sc.parallelize([1,2,3,4])
12    y = x.reduce(lambda x,y: x + y)
13
14    print x.glom().collect()
15    print y
```

```
x: [[1],[2],
    [3],[4]]
y: 10
```

Spark

- RDD

> REDUCE

```
x = sc.parallelize([1,2,3],2)
y = x.reduce(lambda x,y: x - y)
```

==

```
x = sc.parallelize([1,2,3],3)
y = x.reduce(lambda x,y: x - y)
```

?

Spark

- RDD

REDUCE

```
x = sc.parallelize([1,2,3],2)
y = x.reduce(lambda x,y: x - y)
```

==

```
x = sc.parallelize([1,2,3],3)
y = x.reduce(lambda x,y: x - y)
```

?

y: 2

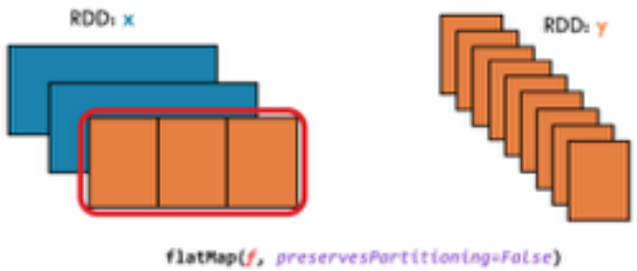
y: -4

Porque?

Spark

➤RDD

FLATMAP




```
flatMap(f, preservesPartitioning=False)
```

Spark

➤RDD

```
11    x = sc.parallelize([1,2,3])
12    y = x.flatMap(lambda z:(z,z**2,42))
13
14    print x.collect()
15    print y.collect()
16
```

Spark

- RDD

```
11    x = sc.parallelize([1,2,3])
12    y = x.flatMap(lambda z:(z,z**2,42))
13
14    print x.collect()
15    print y.collect()
16
```

```
x: [1,2,3]
y: [1,1,42,
    2,4,42,
    3,9,42]
```

Spark

- RDD

REDUCEBYKEY

```
11  x = sc.parallelize(['a','a','b','b'])
12  y = x.map(lambda z:(z,1)).reduceByKey(lambda x,y:x+y)
13
14  print x.collect()
15  print y.collect()
16
```

Spark

- RDD

REDUCEBYKEY

```
11  x = sc.parallelize(['a','a','b','b'])
12  y = x.map(lambda z:(z,1)).reduceByKey(lambda x,y:x+y)
13
14  print x.collect()
15  print y.collect()
16
```

```
x: ['a','a','b','b']
y: [('a',2),
    ('b',2)]
```

Spark

➤RDD

GROUPBYKEY

```
11  x = sc.parallelize(['a','a','b','b']).map(lambda x:(x,1))
12  y = x.groupByKey().map(lambda x:(x[0],list(x[1])))
13
14  print x.collect()
15  print y.collect()
```


GroupByKey != ReduceByKey. Por q?

Spark

➤RDD

GROUPBYKEY

```
11  x = sc.parallelize([('a',1),('a',1),('b',1),('b',1)]).map(lambda x:(x,1))
12  y = x.groupByKey().map(lambda x:(x[0],list(x[1])))
13
14  print x.collect()
15  print y.collect()
```

```
x: [('a',1),
    ('a',1),
    ('b',1),
    ('b',1)]
y: [(‘a’,[1,1]),
    (‘b’,[1,1])]
```


GroupByKey != ReduceByKey. Por q?

Spark

Exemplo

```python
import time
import pandas as pd
words_list = []

start_time= time.time()
with open('really_big_text.txt') as f:
    for i in f.readlines():
        words = i.split()
        words_list = words_list + map(lambda x:(x,1),words)

df = pd.DataFrame(words_list,columns=['word','count'])\
                    .groupby('word').sum()\
                    .sort('count',ascending=False)


print "TIME: %s"%(time.time()-start_time)
print df

```

..ython_sudeste (zsh)

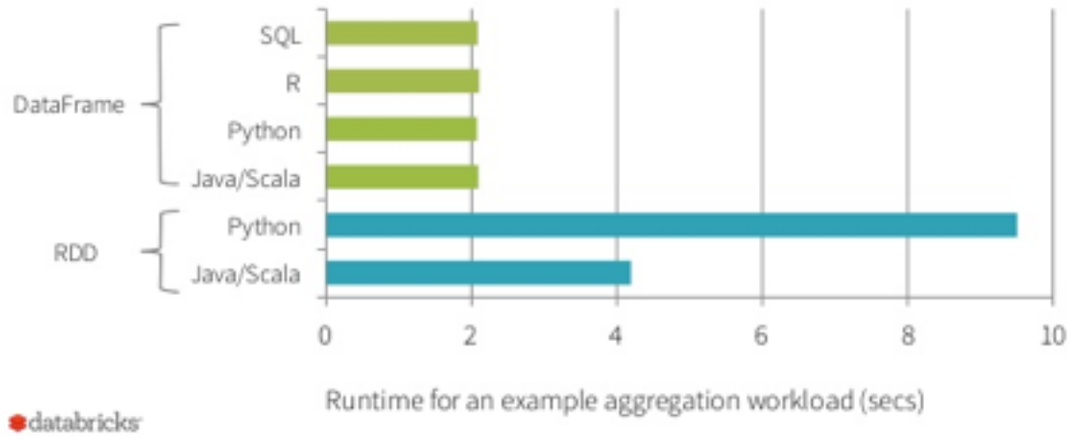
```
(.env) python_sudeste % python only_python.py
only_python.py:11: FutureWarning: sort(columns=...
  df = pd.DataFrame(words_list,columns=['word','co
TIME: 1506.24237919
              count
word
the           72271
and           46446
of            40346
to            16539
that          15553
in            14504
he            11377
shall         10872
```

```python
import time
from pyspark.sql import SQLContext
from pyspark import SparkConf, SparkContext

conf = (SparkConf()
         .setAppName("MyFirstApp")
         .set("spark.executor.memory", "12g"))

sc = SparkContext(conf = conf)
sqlContext = SQLContext(sc)

start_time = time.time()

text = sqlContext.read.text('really_big_text.txt').rdd
text_words = text.flatMap(lambda x: x['value'].split())
text_words.cache()

count_words = text_words\
                  .map(lambda x: (x,1))\
                  .reduceByKey(lambda x,y:x+y)\
                  .sortBy(lambda x:x[1],False)\
                  .toDF(schema=['word','count'])

print "TIME: %s"%(time.time()-start_time)
print "Top 100"
count_words.show(10)
print "Total : %s"%len(text_words.collect())
sc.stop()

```

```
(.env) python_sudeste % ./submit_job.sh
TIME: 10.1741509438
Top 100
+-----+-----+
| word|count|
+-----+-----+
|  the|72271|
|  and|46446|
|   of|40346|
|   to|16539|
| that|15553|
|   in|14504|
|   he|11377|
|shall|10872|
| unto|10056|
|  his|10017|
+-----+-----+
only showing top 10 rows

Total : 948432
```


Mas oque eu use diariamente?

Benefit of Logical Plan: Performance Parity Across Languages



databricks

OBRIGADA!


susana bouchardet


@s_bouchardet