Big Data Warehouses and Analytics: About Scalability and Realtime

Pedro Furtado

Sixth European Business Intelligence & Big Data Summer School (eBISS 2016)

Roadmap

1. Generic Data Organizations and Architectures
2. Spark
3. Lambda Architecture and Realtime
4. Analytics and Data Mining with Spark

What types of apps are there? NOSQL?FRAMEWORKS?

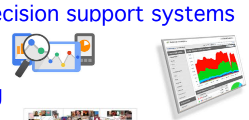
BigData Stores

Huge data
Scale
Latency < 1 secs

Transactional/operational systems

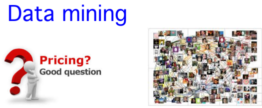
Always Available
ACID
Latency < 100 msecs

Analytic/Decision support systems



Scale complex relational ops
Latency < 5 secs
Elastic

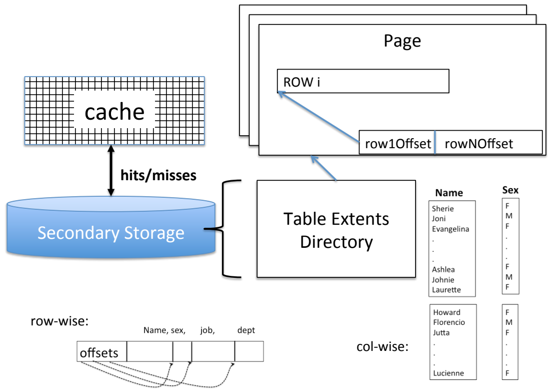
Data mining



Procedural flexibility
Scalable, elastic

Pedro Furtado 2015

Relational DBs



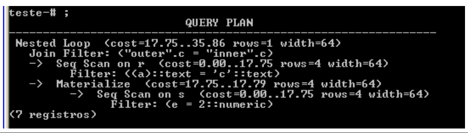
RDBMS Query Plans and Optimization

Whatever you may have (key-val, doc, hadoop),

can you match RDBMS optimization capabilities?

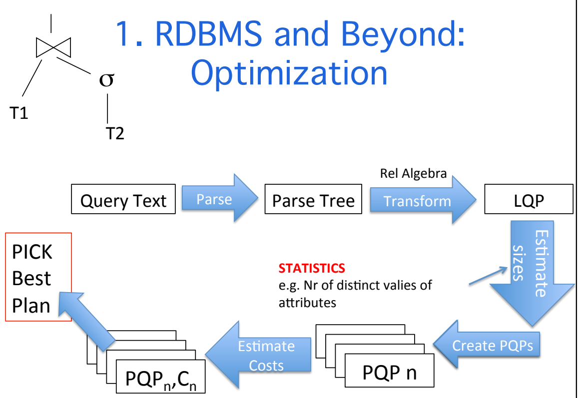
Explain Select B,D From R join S on C

Where R.A = 'c' and S.E = 2
and R.C=S.C



```
teste-# ;
                         QUERY PLAN
-----------------------------------------------------------------
 Nested Loop  (cost=17.75..35.86 rows=1 width=64)
   Join Filter: ("outer".c = "inner".c)
   ->  Seq Scan on r  (cost=0.00..17.75 rows=4 width=64)
         Filter: ((a)::text = 'c'::text)
   ->  Materialize  (cost=17.75..17.79 rows=4 width=64)
         ->  Seq Scan on s  (cost=0.00..17.75 rows=4 width=64)
               Filter: (e = 2::numeric)
(7 registros)
```

1. RDBMS and Beyond: Optimization



Critique of Relational DBs

Fallacy:

That it is mostly disk-based

Most Severe Limitation:

Lots of housekeeping overheads:

RDBMS processing profile (time spent)

12% useful work

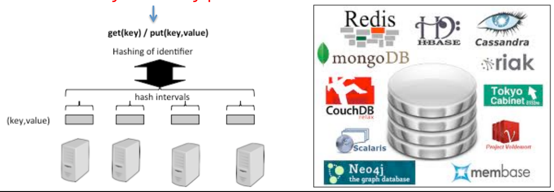
20% logging
18% locking
10% latching
29% buffer management
11% index management

"OLTP thorugh the Looking Glass, and what we found there, S. Harizopoulos, D. Abadi, S. Madden and M. Stonebraker, in ACM SIGMOD 2008"

No-SQL Data Stores

Key-value data stores, Document stores, etc

Do not oblige fixed schemas
Many allow data to be directly in text or xml files
There is frequently no complex loading, compression, opt.
No Joins and no other complex relational algebra operations
Mostly PUT, GET, FILTER AND SCAN
Are usually massively parallelizable



Key-value Stores

Most basic key-value organization:
Just use filesystem files to keep the data

Text format: keyAttribute, BL-TEXT
234332234,
["date":"23/6/2016",
"title":"Angry Ronaldo throws microphone to lake",
"text":"lore ipsum lore lore...."];

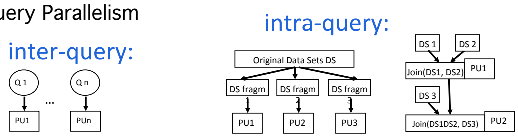
Binary format:
234332234, BinObj:#124332

Parallel Query Processing

Architectural Parallelism

shared-memory (e.g. cores, processors), threading
shared-nothing (nodes with PU, MEM, DISK)
shared-disks (multiple nodes access disks)

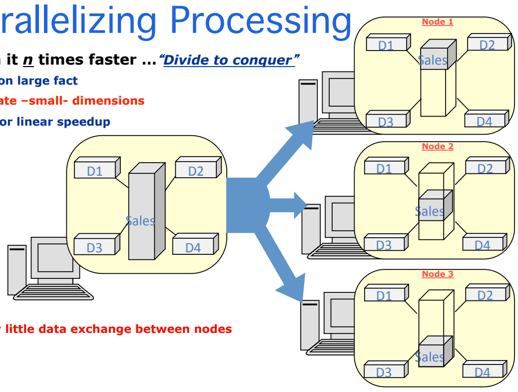
Query Parallelism



Parallelizing Processing

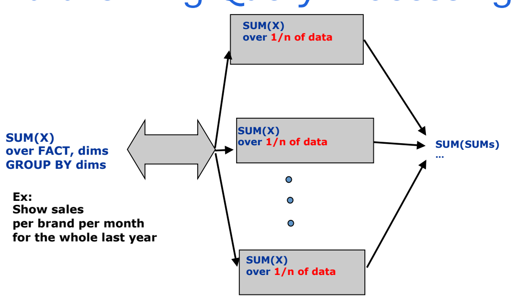
to run it *n* times faster ... ***"Divide to conquer"***

- **Partition large fact**
- **Replicate –small- dimensions**
- **Hope for linear speedup**



- **Very little data exchange between nodes**

Parallelizing Query Processing



Operations Parallelism - Join

Conceptually, every Join is:

Foreach tuple from A
  Foreach tuple from B
  if (A.a==B.b)
  result=result+ A.a I B.b

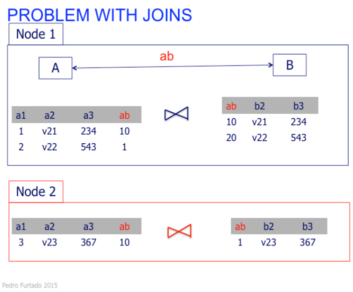
WITHOUT re-SHUFFLE:

Result should be:

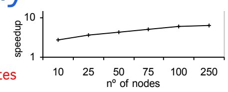
A(1,...),10 ,B(v21,...)
A(2,...),1 ,B(v23,...)
A(3,...),10 ,B(v21,...)

Result is:

A(1,...),10 ,B(v21,...)

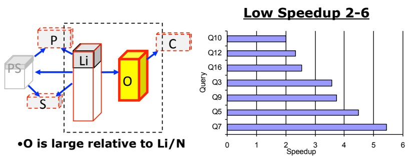


Avoid Join Problem => partition one only



Can we place the data initially so that joins are fastes

to avoid data shipment on-the-fly on every query



Join Processing with Repartitioning

Partitioned Join (NOT ALWAYS POSSIBLE)

each node must have same values of join attribute

Repartition Join -> lots of overhead, SHUFFLE

nodes do not have same values of join attribute
assign join attribute hash-intervals to each node
exchange ON-THE-FLY rows between nodes for same hash
NOW JOIN

Broadcast Join -> lots of overhead, SHUFFLE

nodes do not have same values of join attribute
broadcast ON-THE-FLY full content of one of data sets to all nodes

Pedro Furtado 2015

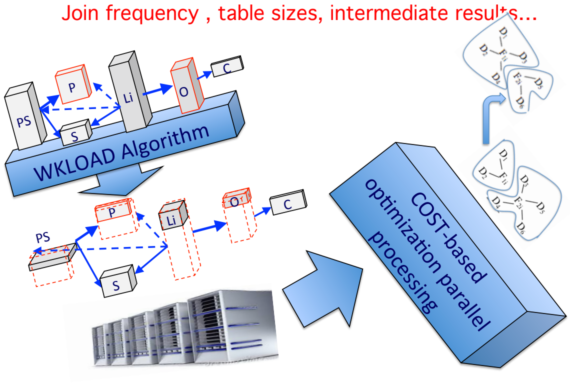
You MUST PARTITION
every big table
at PLACEMENT time...

HOW TO PARTITION?

Workload-based Partitioning

WKLOAD-BASED PARTITIONING

Join frequency , table sizes, intermediate results...



Proposed Solution Algorithm

"Very Small" Dimensions

Replicate = BROADCAST, COPY TO ALL NODES ONCE

Non-small "pure" Dimensions

Hash-Partition by PRIMARY KEY (JOINS)

Large relations / Facts

Find key that minimizes repartitioning costs
Hash-partition by it

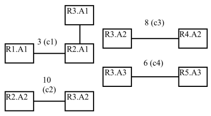
Pedro Furtado: Hash-based Placement and Processing for Efficient Node Partitioned Query-Intensive Databases. ICPADS 2004: 127-134

Pedro Furtado: Experimental evidence on partitioning in parallel data warehouses. DOLAP 2004: 23-30

WK-LOAD-BASED PARTITIONING

Relations, Query Joins, Query Intermediate Result Sizes,…

Minimize the number and cost of repartitioning;

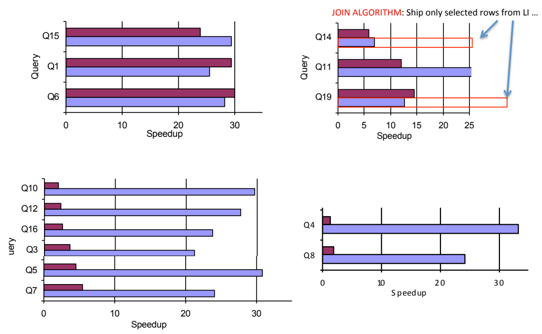


From past workload: LINK weight = Frequency of occurrence X size of join

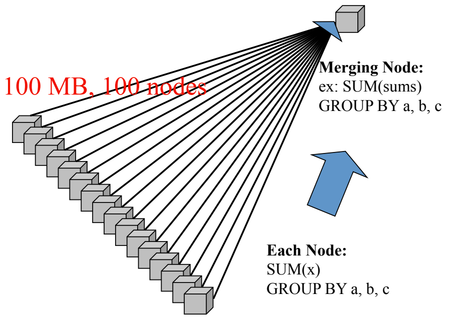
Step y – For each relation R
Pick the link with highest weight containing the relation
The relation R is to be partitioned by the links' equi-join attribute;

Pedro Furtado: Hash-based Placement and Processing for Efficient Node Partitioned Query-Intensive Databases. ICPADS 2004: 127-134

Basic VS WKLOAD-based Partitioning 25 nodes



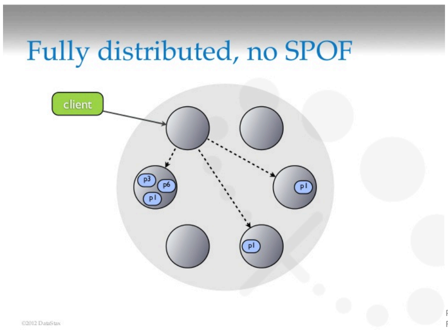
Merging bottleneck over aggregation



Massive NOSQL Parallelism

e.g. Cassandra in the perspective of Datastax

Fully distributed, no SPOF



Pedro Furtado 2015

Cassandra in the perspective of Datastax

Partitioning

Primary key determines placement*

jim	age: 36	car: camaro	gender: M
carol	age: 37	car: subaru	gender: F
johnny	age:12	gender: M	
suzy	age:10	gender: F	

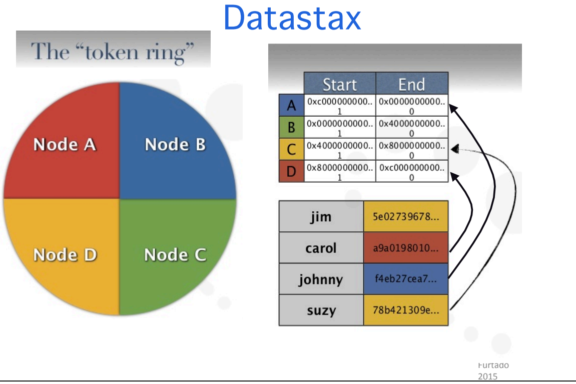
Pedro Furtado 2015

Cassandra in the perspective of Datastax

PK	MD5 Hash
jim	5e02739678...
carol	a9a0198010...
johnny	f4eb27cea7...
suzy	78b421309e...

MD5 hash operation yields a 128-bit number for keys of any size.

Cassandra in the perspective of Datastax



Furtado 2015

Modern Scalable Platforms not only DATA STORE, also process

What is HDFS?

What is Hadoop?

What is Map-Reduce?

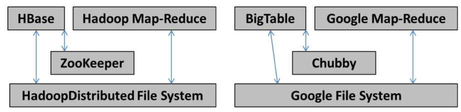
What is Hbase?

HBASE:

- *hbase > create 'product'.'characteristics'*
- *hbase > put 'product', 'tvsetphym4567','characteristics:size','100'*

To view the contents of 'product':

- *hbase > scan 'product'*



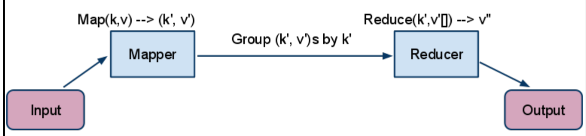
Pedro Furtado 2015

Map-Reduce

Way to parallelize computation

1. THINK OF AN ALGORITHM
2. PLACE IT UNDER M/R MODEL

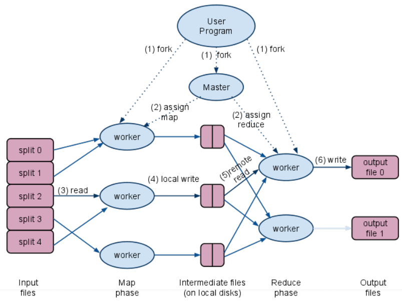
Input= FLAT file organized as (Key, value)



Jerry Zhao, Jelena Pjesivac-Grbovic, MapReduce, The Programming Model and Practice, In Sigmetrics/Performance 2009, Tutorials, June 19th 2009 MapReduce

Pedro Furtado 2015

Modern Scalability Platforms (hadoop + MR)



Jerry Zhao, Jelena Pjesivac-Grbovic, MapReduce, The Programming Model and Practice, In Sigmetrics/Performance 2009, Tutorials, June 19th 2009 MapReduce

Pedro Furtado 2015

Map and Reduce

Map: input=set of ((key,)value)

- for each line or item of (value)
  - do whatever
  - get new k, value from it
  - emit(k, value)

Reduce

- for each pair(k, value)
  - merge somehow
  - get new k, value from it
  - emit(k, value)

Hadoop and Map-Reduce: specify algs

Word Count = count the number of times each word appears in a document

Word Count Solution

Google

```
//Pseudo-code for "word counting"
map(String key, String value):
  // key: document name,
  // value: document contents
    for each word w in value:
      EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
    int word_count = 0;
    for each v in values:
      word_count += ParseInt(v);
    Emit(key, AsString(word_count));
```

No types, just strings*

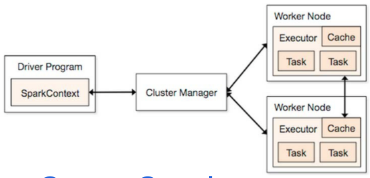
Jerry Zhao, Jelena Pjesivac-Grbovic, MapReduce, The Programming Model and Practice, In Sigmetrics/Performance 2009, Tutorials, June 19th 2009 MapReduce

Spark

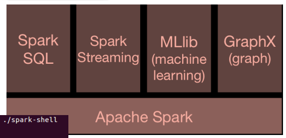
Pedro Furtado

Sixth European Business Intelligence & Big Data Summer School (eBISS 2016)

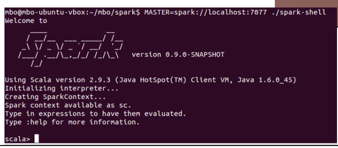
Spark Arch



Some Spark Components



Spark Shell



> bin/spark-shell

The Platform and Architecture

Hadoop

- Distributed data infrastructure: It distributes massive data collections across multiple nodes within a cluster of commodity servers;
- It also indexes and keeps track of that data, enabling big-data processing and analytics;
- Hadoop = Hadoop Distributed File System + MapReduce + …

Spark

- Data-processing tool that operates on those distributed data collections; it doesn't do distributed storage.
- Spark = Data processing framework, needs a file management system, preferably distributed for scalability (e.g. HDFS or cloud)

http://www.infoworld.com/article/3014440/big-data/five-things-you-need-to-know-about-hadoop-v-apache-spark.html
Katherine Noyes, IDG News Service | Dec 11, 2015

Scala (according to wikipedia)

Scala is a programming language for general software applications.

Scala has full support for functional programming and a very strong static type system.

Designed to be concise,[8] many of Scala's design decisions were inspired by criticism of the shortcomings of Java.

Resilient Distributed Dataset RDD

"Resilient Distributed Datasets (RDDs) are a distributed memory abstraction that lets programmers perform in-memory computations on large clusters in a fault-tolerant manner.

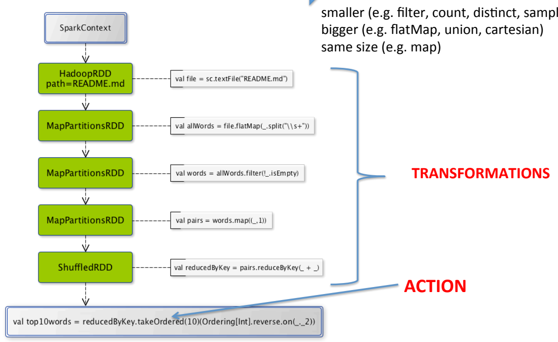
transformations - lazy operations that return another RDD.
actions - operations that trigger computation and return values.

Represents an immutable, partitioned collection of elements that can be operated on in parallel

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (NSDI'12). USENIX Association, Berkeley, CA, USA, 2-2.

RDD → TRANSFORMATION → RDD or RDDs

smaller (e.g. filter, count, distinct, sample),
bigger (e.g. flatMap, union, cartesian)
same size (e.g. map)



https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-rdd-transformations.html

Resilient Distributed Dataset RDD

In-Memory

data inside RDD is stored in memory as much (size) and long (time) as possible.

Immutable or Read-Only

does not change once created and can only be transformed using transformations to new RDDs.

Lazy evaluated

data inside RDD is not available or transformed until an action is executed that triggers the execution.

Cacheable

you can hold all the data in a persistent "storage" like memory (default and the most preferred) or disk (the least preferred due to access speed).

Parallel

process data in parallel.

Typed

values in a RDD have types, e.g. RDD[Long] or RDD[(Int, String)].

Partitioned,

the data inside a RDD is partitioned (split into partitions) and then distributed across nodes in a cluster (one partition per JVM that may or may not correspond to a single node).

List of Transformations

Transformation	Meaning
map(*func*)	Return a new distributed dataset formed by passing each element of the source through a functio
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 retu
sample(*withReplacement*, *fraction*, *seed*)	Sample a fraction *fraction* of the data, with or without replacement, using a given random number
union(*otherDataset*)	Return a new dataset that contains the union of the elements in the source dataset and the argum
intersection(*otherDataset*)	Return a new RDD that contains the intersection of elements in the source dataset and the argum
distinct([*numTasks*]))	Return a new dataset that contains the distinct elements of the source dataset.
reduceByKey(*func*, [*numTasks*])	When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each
aggregateByKey(*zeroValue*)(*seqOp*, *combOp*, [*numTasks*])	When called on a dataset of (K, V) pairs, returns a dataset of (K, U) pairs where the values for each
sortByKey([*ascending*], [*numTasks*])	When called on a dataset of (K, V) pairs where K implements Ordered, returns a dataset of (K, V) p
join(*otherDataset*, [*numTasks*])	When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with all pair
cogroup(*otherDataset*, [*numTasks*])	When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (Iterable<V>, Iterable<W
cartesian(*otherDataset*)	When called on datasets of types T and U, returns a dataset of (T, U) pairs (all pairs of elements).
pipe(*command*, *[envVars]*)	Pipe each partition of the RDD through a shell command, e.g. a Perl or bash script. RDD elements
coalesce(*numPartitions*)	Decrease the number of partitions in the RDD to numPartitions. Useful for running operations mor
repartition(*numPartitions*)	Reshuffle the data in the RDD randomly to create either more or fewer partitions and balance it ac

List of Actions

Action	Meaning
reduce(*func*)	Aggregate the elements of the dataset using a function *func* (which takes two arguments and returns one). The function shoul
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 th
count()	Return the number of elements in the dataset.
first()	Return the first element of the dataset (similar to take(1)).
take(*n*)	Return an array with the first *n* elements of the dataset.
takeSample(*withReplacement*,*num*, [*seed*])	Return an array with a random sample of *num* elements of the dataset, with or without replacement, optionally pre-specifying
takeOrdered(*n*, *[ordering]*)	Return the first *n* elements of the RDD using either their natural order or a custom comparator.
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 othe
saveAsSequenceFile(*path*) (Java and Scala)	Write the elements of the dataset as a Hadoop SequenceFile in a given path in the local filesystem, HDFS or any other Hadoo
saveAsObjectFile(*path*) (Java and Scala)	Write the elements of the dataset in a simple format using Java serialization, which can then be loaded using `SparkContext.ob`
countByKey()	Only available on RDDs of type (K, V). Returns a hashmap of (K, Int) pairs with the count of each key.
foreach(*func*)	Run a function func on each element of the dataset. This is usually done for side effects such as updating anAccumulator or i Note: modifying variables other than Accumulators outside of the foreach() may result in undefined behavior. See Understandi

RDD Persistence

Each node stores its partitions in memory for re-use next

Caching is a key tool for iterative algorithms and fast interactive use.

cache() -> stores in-memory StorageLevel.MEMORY_ONLY

Persisted RDD storage level:

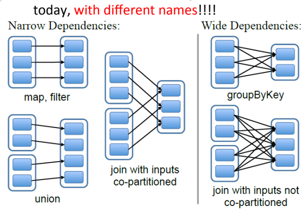
On disk,
In-memory but as serialized Java object (to save space)
Replicated across nodes
Off-heap in Tachyon (reduces garbage collection overhead and allows executors to be smaller and to share a pool of memory)

persist()-> MEMORY_ONLY, MEMORY_AND_DISK,
MEMORY_ONLY_SER (more space efficient)
MEMORY_AND_DISK_SER, DISK_ONLY,
OFF_HEAP

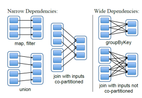
Some details on operations

Join : Joining two RDDs may lead to either two narrow dependencies (if they are both hash/range partitioned with the same partitioner), two wide dependencies, or a mix (if one parent has a partitioner and one does not). In either case, the output RDD has a partitioner (either one inherited from the parents or a default hash partitioner).

Remember that we already talked about this today, with different names!!!!



Shuffle



The under-the-hood rehashing operations

Involves disk I/O, data serialization, and network I/O

Internally, outputs from individual map tasks are kept in memory until they can't fit

When data does not fit in memory Spark will spill these tables to disk (disk I/O, serialization, garbage collection).

Shuffle operations can consume significant amounts of heap memory

Shuffle also generates a large number of intermediate files on disk.

Shuffle behavior can be configured

Broadcast Variables

Give every node a copy of a dataset

Spark uses efficient broadcast algorithms to reduce communication costs.

Spark already automatically broadcasts the common data needed by tasks within each stage.

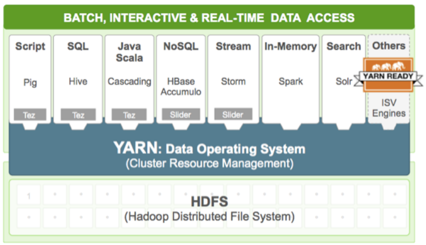
Explicitly creating broadcast variables is only useful when tasks across multiple stages need the same data

val broadcastVar = sc.broadcast(Array(1, 2, 3))

Lookup tables and SMALL tables that are joined frequently

HADOOP YARN: Cluster Resource Manager

Yet Another Resource Negotiator: 2nd-gen Hadoop, YARN is now a large-scale, distributed operating system for big data applications -> manages the resources, with multiple request submitters and so on



Spark: Examples

Easy to Code (Scala, Python, Java), easy to use

Word count in Spark's Python API

```
text_file = spark.textFile("hdfs://...")
text_file.flatMap(lambda line: line.split())
    .map(lambda word: (word, 1))
    .reduceByKey(lambda a, b: a+b)
```

Count number of lines with word 'Spark' Scala

```
scala> val    textFile = sc.textFile("README.md")
scala> textFile.filter(line => line.contains("Spark")).count()
```

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

pySpark: Processing Data and Using Mlib

Read the file, transforming the lines into float fields

```
inp_file= sc.textFile("car-data/car-milage.csv")
car_rdd= inp_file.map(lambda : [float(x) for x in line.split(',')] )
car_rdd.count()
33
car_rdd.first()
```

Calculate statistics for the minimum, maximum and average of each attribute

```
from pyspark.mllib.stat import Statistics

car_rddA =car_rdd.map(lambda x: [ array(y) for y in x ])

summary = Statistics.colStats(car_rddA)
print( str(summary) )
Count :30
Max  : 36.5 | 500.0 | 223.0 | 366.0 |   9.0 |   4.3 |   4.0 |   5.0 | 231.0 |  79.8 | 5430.0 |   1.0 |
Min  : 11.2 |  85.3 |  70.0 |  81.0 |   8.0 |   2.5 |   1.0 |   3.0 | 155.7 |  61.8 | 1905.0 |   0.0 |
Mean : 20.0 | 286.0 | 137.0 | 217.9 |   8.3 |   3.1 |   2.6 |   3.3 | 192.3 |  71.4 | 3625.8 |   0.7 |

for x in summary.min():  print("%4.4f " % x)
for x in summary.mean():      print("%4.4f " % x)
for x in summary.max(): print("%4.4f " % x)
```

From: Fast Data Processing with Spark, 2nd Edition, Holden Karau, Packt publishing, April 2015

https://github.com/xsankar/fdps-vii

pySpark

Use corr function from pyspark.mllib.stat statistics to study the correlation between vehicle power (hp -horse power) and vehicle weight (weight). Get the correlation based on two alternative methods, "pearson" and "spearman". What do you conclude about the degree and form of correlation?

```
hp= car_rdd.map(lambda x: x[2])
weight= car_rdd.map(lambda x: x[10])
weight.foreach(print)
    print('%2.3f' % Statistics.corr(hp, weight, method="pearson") )
    print('%2.3f' % Statistics.corr(hp, weight, method="spearman") )
```

Outputs: 0.888 e 0.874

https://github.com/xsankar/fdps-vii

From: Fast Data Processing with Spark, 2nd Edition, Holden Karau, Packt publishing, April 2015

pySpark

Create code to analyze the speeches of 5 US presidents to determine the most relevant issues and challenges at the time each of them was president.

The datasets are the speeches of George Washington (GW) , Abe Lincoln (AL) , Franklin Delano Roosevelt (FDR), Kennedy (JFK) , Bill Clinton (BC) , GW Bush (GB) & Barack Obama (BO)

POA (Plan of Action)

- Read the 7 data sets to 7 RDDS
- Create the word vectors
- Transform into frequency-of-words vector
- Remove common stop words
- Inspect the keywords to get top n for each president
- Calculate the differences between top wording of two presidents

https://github.com/xsankar/fdps-vii

From: Fast Data Processing with Spark, 2nd Edition, Holden Karau, Packt publishing, April 2015

Presidents Wordings (1)

Get the vector of words designated as President BO (Barrack Obama). Then show the number of words and the frequency of each word.

```
from operator import add
lines= sc.textFile("2009-2014-BO.txt")
lines.foreach(print)

word_count_bo = lines.flatMap(lambda x: x.split(' ')).\
    map(lambda x: (x.lower().rstrip().lstrip().rstrip(',').rstrip('.'),1)).\
    reduceByKey(add)

word_count_bo.count()
 4835
word_count_bo.foreach(print)
```

```
('terrorists', 10)
('liberal;', 1)
('sense', 15)
('rich', 3)
('firmly', 2)
('back', 53)
('prudent', 1)
('strained', 1)
('preserve', 2)
('realities', 1)
('control', 9)
```

https://github.com/xsankar/fdps-vii

From: Fast Data Processing with Spark, 2nd Edition, Holden Karau, Packt publishing, April 2015

Presidents Wordings (2)

Remove common words.

```
common_words = ["us","has","all", "they", "from", "who","what","on","by","more","as","not","their","can",
  "new","it","but","be","are","--","i","have","this","will","for","with","is","that","in",
  "our","we","a","of","to","and","the","that's","or","make","do","you","at","it\'s","than",
  "if","know","last","about","no","just","now","an","because","<p>we","why","we\'ll","how",
  "two","also","every","come","we've","year","over","get","take","one","them","we\'re","need",
  "want","when","like","most","-","been","first","where","so","these","they\'re",
  "good","would","there","should","-->","<!--","up","i\'m","his","their","which","may","were","such","some",
  "those","was","here","she","he","its","her","his","don\'t","i\'ve","what\'s","didn\'t",
  "shouldn\'t","(applause.)","let\'s","doesn\'t"]
```

Show ordered by the most frequent words in speech.

```
word_count_bo_1 = word_count_bo.\
        sortBy(lambda x: x[1],ascending=False)

word_count_bo_1.take(5)
word_count_bo_clean = word_count_bo_1.filter(lambda x: x[0] not in common_words)
word_count_bo_clean.take(10)
```

```
('jobs', 148), ('people', 144), ('american', 133), ('america', 131), ('years', 116),
('work', 108), ('americans', 105), ('time', 89), ('energy', 87)
```

https://github.com/xsankar/fdps-vii

From: Fast Data Processing with Spark, 2nd Edition, Holden Karau, Packt publishing, April 2015

Presidents Wordings (2)

Detect the top differences between speech of BO and GWB. Do this in two ways:

1. show the top 5 words of each,
2. compute and print the difference between those speeches.

```
word_count_bo_clean.take(10)
word_count_gwb_clean.take(10)

for x in
    word_count_bo_clean.subtractByKey(word_count_gwb_clean).sortBy(lambda x: x[1],ascending=False).take(15):
        print(x)
```

```
('banks', 22)
('industry', 20)
('trillion', 17)
('middle-class', 13)
('wage', 12)
('class', 12)
...
```

https://github.com/xsankar/fdps-vii

From: Fast Data Processing with Spark, 2nd Edition, Holden Karau, Packt publishing, April 2015

Spark RDDs, Data Frames and DataSets

RDDs, DataFrames and Datasets in Apache Spark
Brian Clapper, in NE Scala 2016

Interfaces to SparkSQL and Spark

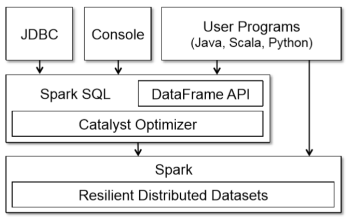


Figure 5. Interfaces to SparkSQL and interaction with Spark [5]

DataFrame API

RDDs, DataFrames and Datasets in Apache Spark
Brian Clapper, in NE Scala 2016

Higher-level
Closer to WHAT instead of HOW
Builds a query plan underneath
you can ALSO SPECIFY it with SQL

You do not care where the data comes from, it can be read from a JSON file, DBMS, whatever!!

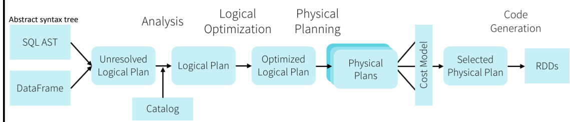
Transforming the RDD into a DF
project(page,numRequests)
You can read a dataframe from any input file format
Then do a group by, a sum, a limit, and show 100 records (action)

```
val df =
 parsedRDD.toDF("project", "page", "numRequests")

df.filter($"project" === "en").
  groupBy($"page").
  agg(sum($"numRequests").as("count")).
  limit(100).
  show(100)
```

OR

```
df.registerTempTable("edits")

sqlContext.sql("SELECT page, sum(numRequests)
      AS count FROM edits
      WHERE project = 'en'
      GROUP BY page LIMIT
.show(100)
```

Optimization (catalyzer)



From dataBricks

The query plan is optimized just like in a DBMS

It produces a logical plan, then an optimized logical plan, then physical plans
Runs through a cost model, selects the best physical plan based on cost

RDDs, DataFrames and Datasets in Apache Spark
Brian Clapper, in NE Scala 2016

Problem of DFs

No Type Safety, types are "lost"

```
scala> :type df.collect()
Array[org.apache.spark.sql.Row]
```

Rows are defined as: Row extends Serializable

Mapping it back to something useful is ugly and error-prone:

```
df.collect().map { row =>
  val project = row(0).asInstanceOf[String]   // Yuck.
  val numRequests = row(1).asInstanceOf[Long] // Yuck.
}
```

RDDs, DataFrames and Datasets in Apache Spark
Brian Clapper, in NE Scala 2016

Datasets

It has a type...

- read a JSON and say you want to see it as Person
- names must match

```
// Read a DataFrame from a JSON file
val df = sqlContext.read.json("people.json")

// Convert the data to a domain object.
case class Person(name: String, age: Long)
val ds: Dataset[Person] = df.as[Person]
//
```

CONCLUSION: DataFrame is a DataSet with a "useless" datatype ROW

RDDs, DataFrames and Datasets in Apache Spark
Brian Clapper, in NE Scala 2016

Datasets VS DataFrames and RDDs

RDDs:

```
val lines = sc.textFile("hdfs://path/to/some/ebook.txt")
val words = lines.flatMap(_.split("""\s+""")).filter(_.nonEmpty)
val counts = words.groupBy(_.toLowerCase).map { case (w, all) => (w, all.size) }
```

DataSets:

- Easier to understand, look at blue code

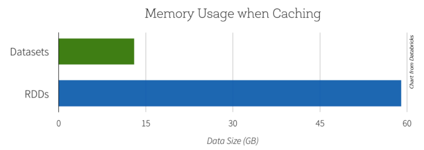
```
val lines = sqlContext.read.text("hdfs://path/to/some/ebook.txt").as[String]
val words = lines.flatMap(_.split("""\s+""")).filter(_.nonEmpty)
val counts = words.groupByKey(_.toLowerCase).count()
```

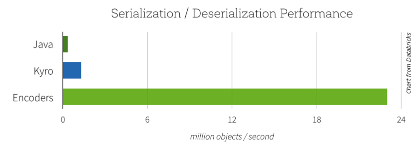
- More efficient organization = Tungsten
  - Instead of Java objects, serialized objects on the JVM heap (RDDs)
  - Off-heap memory, compact columnar-based storage, operated fast
  - ENCODERS generated code on-the-fly to serialize/des and operate on serialized

RDDs, DataFrames and Datasets in Apache Spark
Brian Clapper, in NE Scala 2016

Datasets – Memory and Performance

Spark has to serialize data... a lot
- send data across the network,
- e.g. groub by key=SHUFFLE= SERIALIZE
- dump to disk => serialize + deserialize later





Speed

Caching is important
- if you use something more than once, cache it!

Use broadcast variables liberally
- Only serialized once
- Available to all executors
- Extremely fast lookups

Distributed memory maps well parallelized
- Nr of partitions should be much larger than nr of nodes
- Workload-based partitioning would be great here!

Spark Scalability Experiments with TPCH

Decision Support Data Analysis in Spark using TPC-H Benchmark, in Sistemas Gestão Dados, 2016
R. Rei, P. Furtado FCTUC Coimbra, Portugal

TPC-H on Spark/Hive

Create a HiveContext using the SparkContext as input:

val hiveContext = new org.apache.spark.sql.hive.HiveContext(sc)

SQL computations in Spark can be done by applying transformations and actions on DataFrames.

The HiveContext object is used to create DataFrames via loading data stored in hive tables. The method .sql next is a TRANSFORMATION (lazy evaluation, not run immediately)

```
val DF_query6 = hiveContext.sql("""
    select sum(l_extendedprice * l_discount) as revenue
    from lineitem
    where l_shipdate >= '1993-01-01' and l_shipdate < '1994-01-01'
    and l_discount between 0.06 - 0.01 and 0.06 + 0.01 and l_quantity < 25
    """);
```

Lazy evaluation using ACTION show on DataFrame:

scala> DF_query6.show | revenue| +------------------+ |8.203611004383996E8|

Processing FlowChart

Show ACTION

Spark passes the query in method "sql" to optimizer

Optimizer turns query into Spark code

The optimizer = Catalyst

- compiles the operations applied to DataFrame
- Applies optimizations
- Creates a physical plan

Computation is performed by the Executors residing in Workers processes in slaves.

Experimental Setup

Decision Support Data Analysis in Spark using TPC-H Benchmark, in Sistemas Gestão Dados, 2016
R. Rei, P. Furtado FCTUC Coimbra, Portugal

1 node:

12 *cores*, 32GB de RAM, 70GB disk

3 nodes:

each with 12 *cores*, *32*GB de RAM and 70GB disk

OS Ubuntu 14.04 LTS versão 64 *bits.*

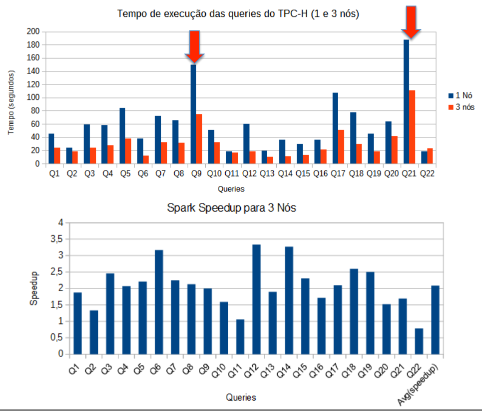
Hadoop versão 2.7.1, *Hive* versão 1.2.1, o *Spark* versão 1.6.1

MySQL as *Metastore* of *Hive.*

TPC-H data by *DBGEN* with *scale factor* 10GB.

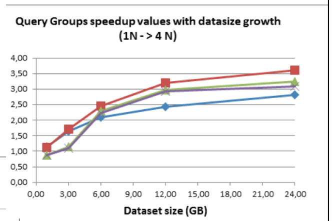
Stored as *Hive tables in HDFS.*

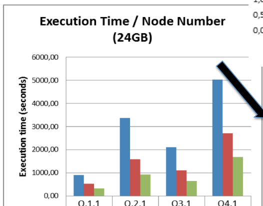
10GB



Hive on bigger data

(Rafael, Furtado, Bernardino, 2014)





	Q.1.1	Q.2.1	Q3.1	Q4.1
1N	904,78	3372,68	2117,66	5039,17
2N	527,39	1581,88	1113,06	2718,82
4N	318,12	921,13	650,27	1682,77

Nr. Of nodes

- Intel core i7 920 2.67 GHz
- 40 GB of RAM
- 4 SATA disks of 465 GB, 7,500 RPM.
- All nodes created in HyperV 2012 virtual platform, in a host with Microsoft Windows Operating System version 8.1 64 bit,
- data node = 2 virtual processors,
  - 8 GB RAM
  - 125 GB disk
  - CentOS 6.5 64-bit operating system

Pedro Furtado
2016

Streaming, Lambda Architecture

Pedro Furtado

Sixth European Business Intelligence & Big Data Summer School (eBISS 2016)

Lambda Architecture, Streaming

- Analytics in realtime

e.g. recommendation engine which

recommends in realtime

recomputes recommendation model very often to reflect changes

e.g. pricing model

changes in context, competition, sales, reflected ASAP

- Requires operating on streams of data

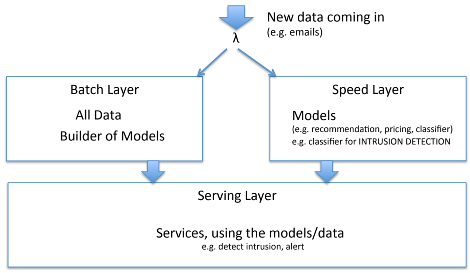
rebuilding the model

deploy model that runs quickly

scale, tolerate faults

Lambda Architecture

Allows us to handle streaming data efficiently



Lambda Architecture

Batch layer allows you to always go back to correct state, because you have everything always

Since you have all data in the batch layer, you can

create new models

create an updated model

correct a model

incorporate new things in a model

incorporate new data to rebuild the model

Batch layer + streaming layer allows you to answer all types of queries, forget about OLTP + OLAP duality form the point of view of using the data

Lambda Architecture with SPARK

Apache Spark

you have Spark (batch layer) and Spark Streaming (streaming data)

spark streaming is integrated into spark

same programming and model as spark, same code runs, completely integrated

can use any input data (streams, logs, files, from ports, subscribe to kafka)

has rich machine learning library

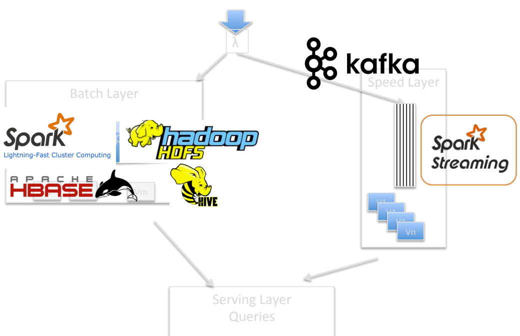
has a lot other capabilities

is totally scalable

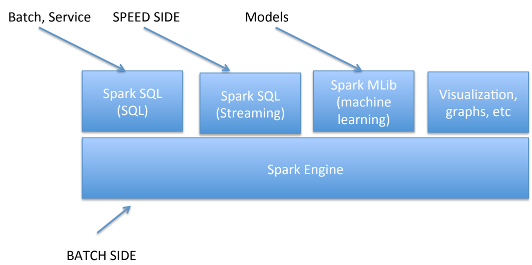
Apache Kafka

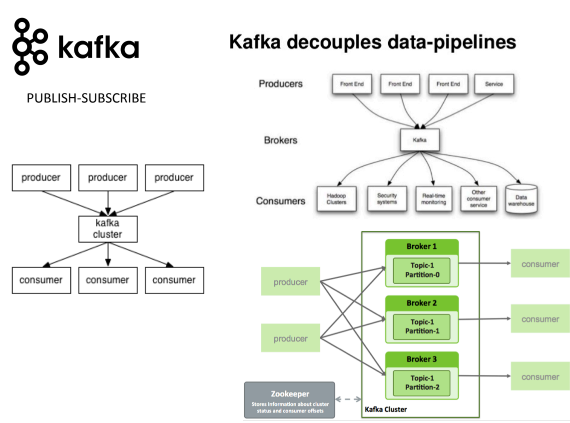
pub/sub messaging system

Lambda Architecture



You can build Lambda with spark alone!!





Kafka example - Producer

```
Call:
...
 Producer.main(args);
 ...
```

```
bootstrap.servers=localhost:9092
acks=all
...
block.on.buffer.full=true
```

topic   message

```
producer = new KafkaProducer(properties);

While(whatever)
        producer.send(new ProducerRecord("fast-messages", "This is a dummy message"));
   ...
   } finally {
        producer.close();
   }
  ...
```

https://github.com/mapr-demos/kafka-sample-programs/blob/1.0/src/main/java/com/mapr/examples/Producer.java

Kafka example - Consumer

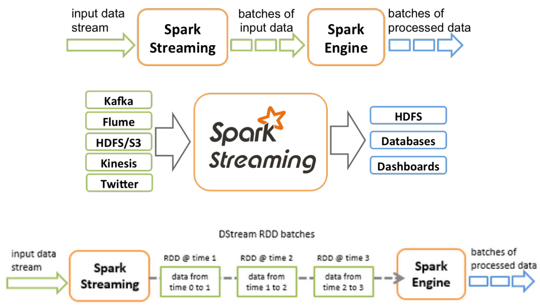
```
bootstrap.servers=localhost:9092
group.id=test
enable.auto.commit=true
...
max.partition.fetch.bytes=2097152
```

topic 1   topic 2

```
consumer = new KafkaConsumer(properties);

consumer.subscribe(Arrays.asList("fast-messages", "summary-markers"));

...
   while (true) {
      ConsumerRecords records = consumer.poll(200);
      ...
   }
...
```

https://github.com/mapr-demos/kafka-sample-programs/blob/1.0/src/main/java/com/mapr/examples/Producer.java

Spark Streaming



Discretized Streams (Dstreams)

Basic abstraction provided by Spark Streaming.

Represents a continuous stream of data, either the input data stream received from source, or the processed data stream generated by transforming the input stream.

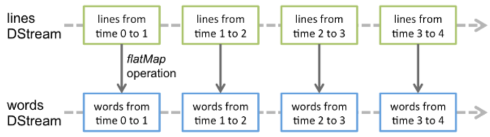
Internally, a DStream is represented by a continuous series of RDDs, which is Spark's abstraction of an immutable, distributed dataset

Spark Streaming

How many twitter twits contain the word Spark on every 5 seconds:

```
TwitterUtils.createStream(...)
   .filter(_.getText.contains("Spark"))
   .countByWindow(Seconds(5))
```

Wordcount…



Spark Streaming – Wordcount app

Create a DStream that represents streaming data from a TCP source

(val ssc = new StreamingContext(conf, **Seconds(1)**))

SCALA:

```
// Create a DStream that will connect to hostname:port, like localhost:9999
val lines = ssc.socketTextStream("localhost", 9999)
```

PYTHON:

```
# Create a DStream that will connect to hostname:port, like localhost:9999
lines = ssc.socketTextStream("localhost", 9999)
```

Spark Streaming – Wordcount app

Split the lines by space characters into words

Each line will be split into multiple words and the stream of words is represented as the words Dstream

```
// Split each line into words
val words = lines.flatMap(_.split(" "))

// Count each word in each batch
val pairs = words.map(word => (word, 1))
val wordCounts = pairs.reduceByKey(_ + _)

// Print the first ten elements of each RDD generated in this DStream to the console
wordCounts.print()
```

Spark Streaming – Wordcount app

To start the processing after all the transformations have been setup, call

```
ssc.start()            // Start the computation
ssc.awaitTermination()  // Wait for the computation to terminate
```

(val ssc = new StreamingContext(conf, **Seconds(1)**))

Spark Streaming – Wordcount app

Need to feed port with phrases…

run Netcat (a small utility found in most Unix-like systems) as a data server

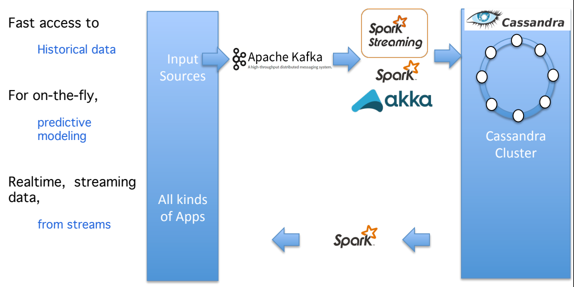
$ nc -lk 9999

in a different terminal, you can start the example

```
# TERMINAL 1:
# Running Netcat

$ nc -lk 9999

hello world



...
```

```
# TERMINAL 2: RUNNING NetworkWordCount

$ ./bin/run-example streaming.NetworkWordCount localhost 9999
...
-------------------------------------------
Time: 1357008430000 ms
-------------------------------------------
(hello,1)
(world,1)
...
```

Fast + Streaming + Historical
CAN ALL BE PROGRAMMED IN SCALA



Note: Akka is a toolkit and runtime for building highly concurrent, distributed, and resilient message-driven applications on the JVM.

Qualities needed

- Scalability
- Partition
- Replicate for fault Tolerance
- Share-Nothing
- Asynchronous Message Passing
- Parallelism
- Isolation
- Data Locality
- Location independence

Spark
- fast, distributed,
- scalable, FT,
- low latencies,
- complex analytics

Kafka
- high-throughput,
- distributed messaging
- supports massive # consumers
- partition on cluster
- auto-recovery node fail

Cassandra
- massively scalable
- database
- high-performance
- masterless

PERSIST IN Casandra Tables

```
CREATE TABLE raw_weather_data (
  wsid text,          // Composite of Air Force Datsav3 station number and NCDC WBAN number
  year int,           // Year collected
  month int,          // Month collected
  day int,            // Day collected
  hour int,           // Hour collected
  temperature double,     // Air temperature (degrees Celsius)
  dewpoint double,        // Dew point temperature (degrees Celsius)
  pressure double,        // Sea level pressure (hectopascals)
  wind_direction int,     // Wind direction in degrees. 0-359
  wind_speed double,      // Wind speed (meters per second)
  sky_condition int,      // Total cloud cover (coded, see format documentation)
  sky_condition_text text, // Non-coded sky conditions
  one_hour_precip double,  // One-hour accumulated liquid precipitation (millimeters)
  six_hour_precip double,  // Six-hour accumulated liquid precipitation (millimeters)
  PRIMARY KEY ((wsid), year, month, day, hour)
) WITH CLUSTERING ORDER BY (year DESC, month DESC, day DESC, hour DESC);
```

Basic Spark + Cassandra

Retrieve from commits table... select and where is pushed to Cassandra for efficiency

Type= CassandraRDD[CassandraRow]

```
val rdd=sc.cassandraTable("github","commits")
      .select("user","count","year","month")
      .where("commits >= ? and year = ?", 1000, 2015)
```

Now retrieve from commits_aggregate into a streaming rdd:

Convert CassandraRow into MonthlyCommits case class

```
val rdd=scc.cassandraTable[MonthlyCommits]("github","commits_aggregate")
      .where("user >= ? and project_name = ? and year=?", "Helena",
                              "spark-cassandra-connector", 2015)
```

Kafka Streaming Word Count

As soon as data starts coming into kafka topics, computation starts:

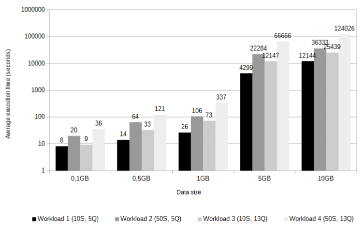
```
sparkConf.set("spark.cassandra.connection.host","10.20.3.45")
val streamingContext = new StreamContext(conf, Seconds(30))

KafkaUtils. createStream[String, String, StringDecoder, StringDecoder]
              (streamingContext, kafkaParams,
                          topicMap, StorageLevel.MEMORY_ONLY)
.map(_._2)
.countByValue()
.saveToCassandra("my_keyspace","wordcount")
```

REALTIME DATA WAREHOUSING AND ANALYSIS

Pedro Furtado
2016

Under Load...

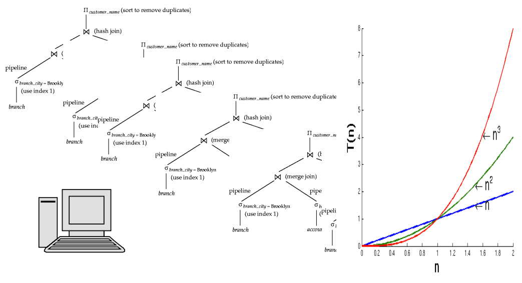
A database engine should not STALL while processing a "normal" query.... Should it?



Pedro Furtado 2016

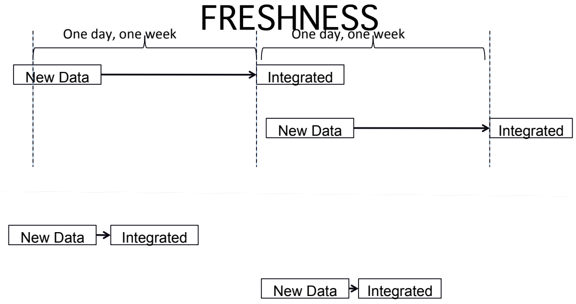
What if there are 1000 queries, even on smaller data?

One-query-at-a-time model is a problem



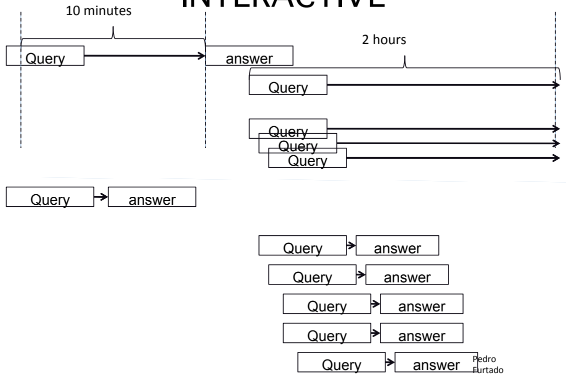
Pedro Furtado 2016

From traditional to realtime DW: FRESHNESS



Pedro Furtado 2016

From traditional to realtime DW: INTERACTIVE



Pedro Furtado 2016

(near)Realtime DWs

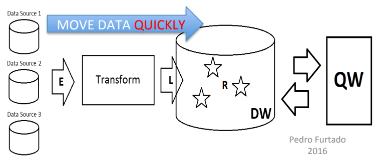
What if we need/want fresh data?

- Incorporate events into analysis quickly
- (e.g. Detect fraud or react quickly to events)

Why is it difficult in traditional DWs?

Which factors influence those near-realtime capabilities?

Are there solutions to make existing DW near-realtime?



Pedro Furtado 2016

Experimental Setup

Oracle DBMS

SSB star schema benchmark + REALTIME LOADING

The Query Workload had 13 queries

Two computers used in tests

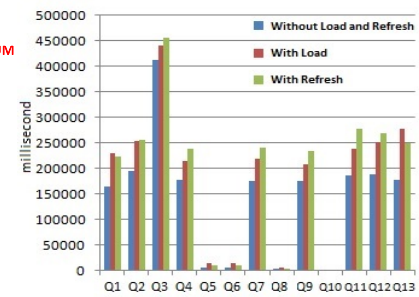
- Intel(R) Core(TM) 2 Quad 2.5GHz with 4GB of RAM
- Intel(R) Core(TM) i5 3.40GHz with 16GB of RAM

Each test (15 runs, average 10) (<+-5%)

Pedro Furtado 2015

Test 1: effect of simultaneous loading on query performance

LOW-MEDIUM EFFECT

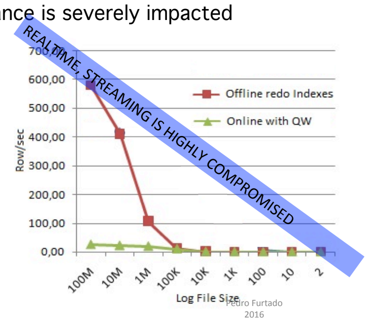


Pedro Furtado 2015

Test 2: Online loading vs Offline loading

Loading performance is severely impacted

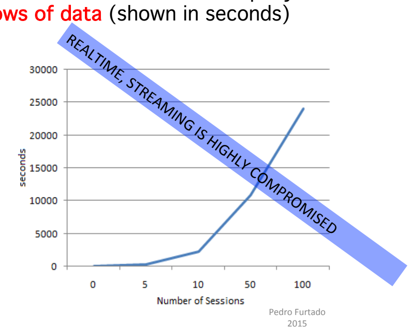
high EFFECT!!!



Test 3: Effect on loading of increse in nr of query sessions

Effect of number of simultaneous query sessions on loading 10 rows of data (shown in seconds)

HUGE EFFECT

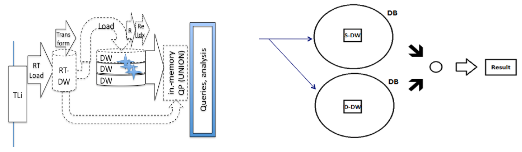


RealTime-Data Warehouse

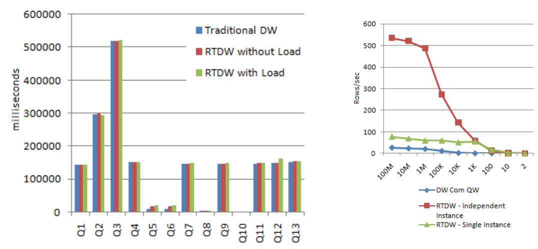
Totally separate DW from RT Component

Merge DW and RT query results on-the-fly

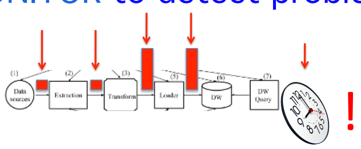
Still need to load RT data into DW periodically, offline



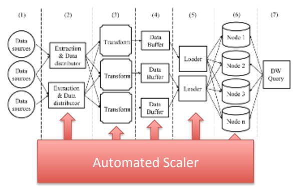
RT-DW Realtime results



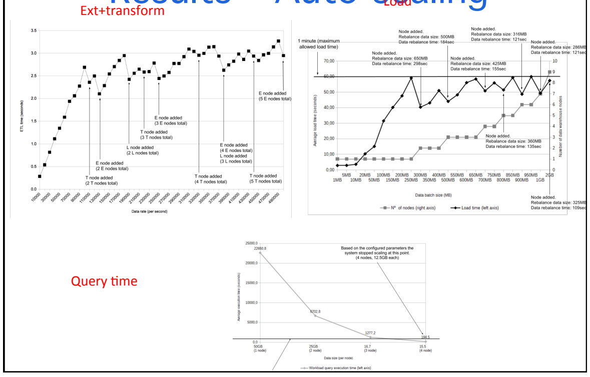
MONITOR to detect problems:



SCALE automatically:



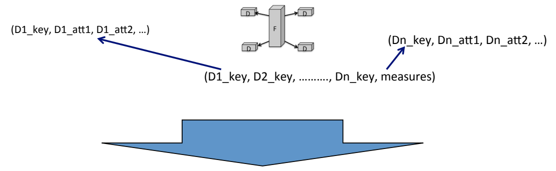
Results – Auto-scaling



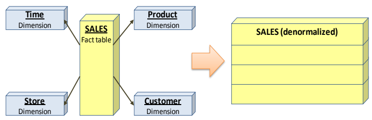
Traditional Parallel DBMS seem complex, and they can fail to run FAST…

De-normalize
TOTALLY Parallelize

[1] João Pedro Costa, J. Cecílio, P. Martins, and P. Furtado, "ONE: A Predictable and Scalable DW Model," DaWak'11



(D1_att1, D1_att2, ………, Dn_attx, measures)



Pedro Furtado 2015

Immutable, append-only

The ONE repository is completely IMMUTABLE, append-only

There are no deletes or updates

(except for input typo undos)

ONE is just a huge denormalized log of data

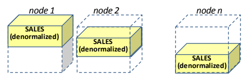
ONE's data is always correct

Querying and other data management are automatically rewritten to work on ONE

Parallel ONE (ONE-P) [2]

Simpler deployment

- ONE relation fully partitioned among nodes
- Partition size fitting node's capabilities
- No repartitioning is required



[2] João Pedro Costa, José Cecílio, Pedro Martins, Pedro Furtado: "A predictable storage model for scalable parallel DW", IDEAS 2011

@ João Costa 106

I WANT *x* (m)seconds GUARANTEE, ALWAYS!!!!

Now you can finally say:

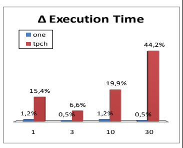
I want any query to take at most X

And the system will get you X… if you have machines for that

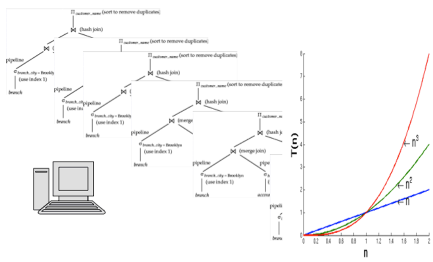
P-ONE does just that,

because execution time is totally predictable

$$\frac{t_S \times (ts_S + ts_R)}{block_{size}} \times IO$$

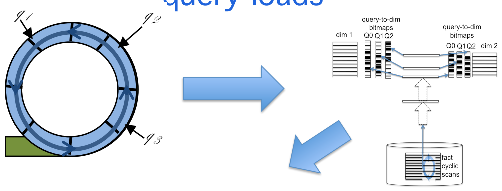


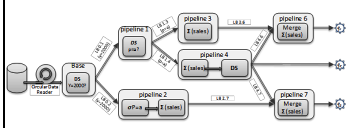
SPIN: But what about dealing with 1000 Concurrent Queries ?



@ João Costa

SPIN: Providing Scalable concurrent query loads





Combine similar predicates

Share intermediate results

q_1 = SUM ($\sigma_{p=a}$(sales))
q_2 = SUM ($\sigma_{y=2000}$(sales))
q_3 = SUM ($\sigma_{y=2000 \wedge p=a}$(sales))

Analytics

Pedro Furtado

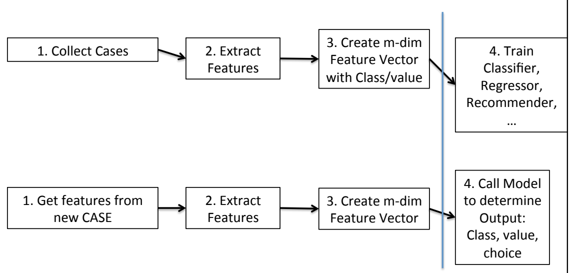
Sixth European Business Intelligence & Big Data Summer School (eBISS 2016)

Ex: Dataset Diabetes and Obesity

Adult, 57, bad food quality, 3days/week 30 mins exercise… is he in risk of obesity or diabetes?

person	age	age class	sex	region	income	food quality	exercise	IMC	IMC Class	blood pressure	cardiovascular disease	diabetes
14	49	Adult2	F	W	3rd	Low	VeryUnhealthy	41	MorbidlyObese	HU	N	**Y**
15	61	Adult3	M	SE	Highest	Low	VeryUnhealthy	38	Obese	HU	N	**N**
23	53	Adult2	M	W	4th	Low	VeryUnhealthy	27	Overweight	NU	N	**Y**
24	85	VeryOld	M	N	3rd	Low	VeryUnhealthy	34	Obese	NC	Y	**N**
37	52	Adult2	M	SW	4th	Low	NotVeryHealthy	28	Overweight	NU	N	**Y**
38	66	Old	M	NW	Lowest	Low	NotVeryHealthy	28	Overweight	NU	N	**N**
51	70	Old	M	SW	4th	Low	NotVeryHealthy	36	Obese	HU	N	**Y**
52	73	Old	M	C	3rd	High	VeryHealthy	22.35	Normal	NU	N	**N**
55	85	VeryOld	M	W	Highest	Low	NotVeryHealthy	36	Obese	HC	Y	**Y**
56	66	Old	M	NW	Lowest	High	QuiteHealthy	19.27	Normal	NU	N	**N**
147	57	Adult3	F	N	Lowest	Low	NotVeryHealthy	43	MorbidlyObese	HU	N	**Y**
148	36	Adult1	F	NW	Highest	High	VeryHealthy	22.4	Normal	NU	N	**N**
150	84	VeryOld	M	C	3rd	Low	VeryUnhealthy	26	Overweight	NU	Y	**Y**
151	17	Teen	M	E	Lowest	High	VeryHealthy	17.37	Underweight	NU	N	**N**
198	16	Teen	M	N	4th	High	VeryHealthy	19.75	Normal	NU	N	**Y**
199	22	Teen	F	SW	Lowest	High	VeryHealthy	21.29	Normal	NU	N	**N**
205	66	Old	M	N	Lowest	High	NotVeryHealthy	23.63	Normal	NU	N	**Y**

Data mining, m-dim Datasets



Massive Processing

Divide and Conquer on every step

Parallel Feature Extraction;

Algorithms execute over parts of the m-dim dataset;

Matrix Computations can be partitioned over nodes

…

Titanic (binary classification)

We are given details of Titanic passengers (e.g. name, gender, fare, cabin)
We are told whether the person survived the Titanic disaster.
Build a Model that can predict if any passenger is likely to survive.

ATTRIBUTES:
0 pclass,
1 survived,
2 l.name,
3.f.name,
4 sex,
5 age,
6 sibsp,
7 parch,
8 ticket,

ATTRIBUTES:
9 fare,
10 cabin,
11 embarked,
12 boat,
13 body,
14 home.dest

https://github.com/xsankar/fdps-vii

From: Fast Data Processing with Spark, 2nd Edition, Holden Karau, Packt publishing, April 2015

Tintanic: load and analyze data

```
$ head train.csv
PassengerId,Survived,Pclass,Name,Sex,Age,SibSp,Parch,Ticket,Fare,Cabin,Embarked
1,0,3,"Braund, Mr. Owen Harris",male,22,1,0,A/5 21171,7.25,,S
2,1,1,"Cumings, Mrs. John Bradley (Florence Briggs Thayer)",female,38,1,0,PC 17599,71.2833,C85,C
3,1,3,"Heikkinen, Miss. Laina",female,26,0,0,STON/O2. 3101282,7.925,,S
4,1,1,"Futrelle, Mrs. Jacques Heath (Lily May Peel)",female,35,1,0,113803,53.1,C123,S
5,0,3,"Allen, Mr. William Henry",male,35,0,0,373450,8.05,,S
6,0,3,"Moran, Mr. James",male,,0,0,330877,8.4583,,Q
7,0,1,"McCarthy, Mr. Timothy J",male,54,0,0,17463,51.8625,E46,S
8,0,3,"Palsson, Master. Gosta Leonard",male,2,3,1,349909,21.075,,S
9,1,3,"Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)",female,27,0,2,347742,11.1333,,S
```

From: Fast Data Processing with Spark, 2nd Edition, Holden Karau, Packt publishing, April 2015

Titanic Parse Passenger Data

To LabeledPoint

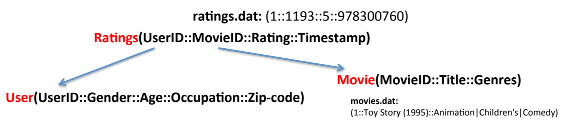
LabeledPoint(survived,Vectors.dense(pclass,sex,age,sibsp,parch,fare))

```
def main(args: Array[String]): Unit = {

  val sc = new SparkContext("local","SampleApps")

  val dataFile= sc.textFile("/Users/data/titanic3_01.csv")

  val titanicRDDLP = dataFile.map(_.trim).
            filter( _.length > 1).
            map(line => parsePassengerDataToLP(line))

  titanicRDDLP.foreach(println)

  println(titanicRDDLP.first().label)
  println(titanicRDDLP.first().features)

  val categoricalFeaturesInfo = Map[Int, Int]()

  val mdlTree = DecisionTree.trainClassifier(titanicRDDLP,
     2,                       // numClasses
     categoricalFeaturesInfo, // features are continuous
     "gini",                  // impurity
     5,                       // Maxdepth
    32)                       //maxBins

  println(mdlTree)

val predictions =
      mdlTree.predict(titanicRDDLP.map(x=>x.features))

val labelsAndPreds =
      titanicRDDLP.map(x=>x.label).zip(predictions)

val mse = labelsAndPreds.map( vp =>
      math.pow( (vp._1 - vp._2),2 ) ).
      reduce(_+_) / labelsAndPreds.count()

println("Mean Squared Error = " + "%6f".format(mse))

labelsAndPreds.foreach(println)

}
```

https://github.com/xsankar/fdps-vii/blob/master/code/MLlib02.scala

From: Fast Data Processing with Spark, 2nd Edition, Holden Karau, Packt publishing, April 2015

Recommendation Example



From: Fast Data Processing with Spark, 2nd Edition, Holden Karau, Packt publishing, April 2015

Load the files

```
movies_file = sc.textFile("movies.dat")
movies_rdd = movies_file.map(lambda line: line.split('::'))
movies_rdd.count()
movies_rdd.first()

ratings_file = sc.textFile("ratings.dat")
ratings_rdd = ratings_file.map(lambda line: line.split('::'))
ratings_rdd.count()
ratings_rdd.first()

ratings_file = sc.textFile("users.dat")
ratings_rdd = ratings_file.map(lambda line: line.split('::'))
ratings_rdd.count()
ratings_rdd.first()
```

From: Fast Data Processing with Spark, 2nd Edition, Holden Karau, Packt publishing, April 2015

Transform fields

Create a function to transform columns user_id, movie_id, rating and timestamp into int. int, float and int respectively.
Create an RDD ratings_rdd_01 with those values

```
def parse_ratings(x):
      user_id = int(x[0])
      movie_id = int(x[1])
      rating = float(x[2])
      timestamp = int(x[3])/10
return [user_id, movie_id, rating, timestamp]


ratings_rdd_01 = ratings_rdd.map(lambda x: parse_ratings(x))
ratings_rdd_01.count()
ratings_rdd_01.first()
```

From: Fast Data Processing with Spark, 2nd Edition, Holden Karau, Packt publishing, April 2015

Create DS train, test, validation

Divide the dataset into three datasets : training , validation and test , as follows:
- timestamp % 10 <6 -> training ;
- timestamp 10 % > = 6 to < 8 -> validation ;
- timestamp % 10 > = 8 - > test ;

Count the number of records of each of the resulting datasets.

```
training = ratings_rdd_01.filter(lambda x: (x[3] % 10) < 6)
validation = ratings_rdd_01.filter(lambda x: (x[3] % 10) >= 6 and (x[3] % 10) < 8)
test = ratings_rdd_01.filter(lambda x: (x[3] % 10) >= 8)

numTraining = training.count()
numValidation = validation.count()
numTest = test.count()
print "Training: %d, validation: %d, test: %d"
```

From: Fast Data Processing with Spark, 2nd Edition, Holden Karau, Packt publishing, April 2015

Train model – Alternating Least Squares

Train the model using 20 iterations, with variable rank = 10.

```
from pyspark.mllib.recommendation import ALS

rank = 10
numIterations = 20
# Build the model on training data (userID, movie, rating)
# rank is the number of latent factors in the model.
# iterations is the number of iterations to run.
train_data = training.map(lambda p: (p[0], p[1], p[2]))
model = ALS.train(train_data, rank, numIterations)
```

From: Fast Data Processing with Spark, 2nd Edition, Holden Karau, Packt publishing, April 2015

Evaluate model

Evaluate the model in the training data . Show the first recommendation, the first line of testdata and the first real rating (test).

```
# Evaluate the model on test data (userID, movie)
testdata = test.map(lambda p: (p[0], p[1]))
predictions = model.predictAll(testdata).map(lambda r: ((r[0], r[1]), r[2]))
predictions.count()

predictions.first()
testdata.first()
test.first()
```

From: Fast Data Processing with Spark, 2nd Edition, Holden Karau, Packt publishing, April 2015

Practical Cases: Synset

Practical Case 1: Synset "dish" of Imagenet

Use Machine Learning algorithms for annotation and classification of images

- Multi-class classifier (decision trees, naïve bayes, rand forests)
- Dishes Synset contains 313 classes of recipes
- We had to decrease the nr of classes in practical testbed (10 to 50)
- Later we will do it for all classes

We added a customized SVM algorithm (two classes 1/all)

We optimized pre-processing + analyzed partitioning scalability

Compared accuracy and runtime

Future work (very much looking forward to it)

- Deep Learning approach

M. Freitas, J. Alves, P. Furtado, Classificação de imagens do synset "dish" do ImageNet usando Apache Spark, in Sistemas de Gestão de Dados, U. Coimbra, 2016 UC-DEI TechReport1010

Proposed Roadmap

Pre-processing

- Reduce images + toGrayScale (python cv2 binding to OpenCV)
- Transfer all images to HDFS \Synsets\~~\sushi, etc

Feature extraction

- SIFT features-> keypoints->bagofwords->k-means->pooling
- Using OpenCV+pySpark
- All intermediate files in Parquet format

Creation of train and test datasets

- 10 to 15% of images = test

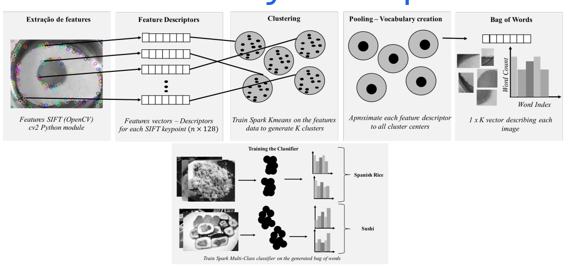
Train Spark classifier OFF-THE-SHELF

- Naïve Bayes, Logistic Regression, Decision Trees, Random Forests, SVMs -> there is no multi-class SVM, therefore created a hier. one

Validate Spark Classifier using test dataset

M. Freitas, J. Alves, P. Furtado, Classificação de imagens do synset "dish" do ImageNet usando Apache Spark, in Sistemas de Gestão de Dados, U. Coimbra, 2016 UC-DEI TechReport1010

Summary of Steps



M. Freitas, J. Alves, P. Furtado, Classificação de imagens do synset "dish" do ImageNet usando Apache Spark, in Sistemas de Gestão de Dados, U. Coimbra, 2016 UC-DEI TechReport1010

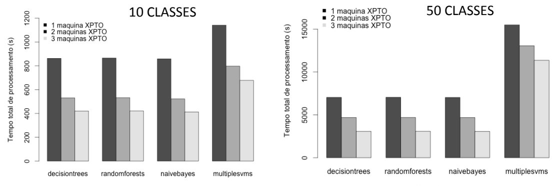
Results SS

Load time to HDFS:

Nr Classes	nr images	MB	load time (secs)
10	6524	875	252.1
50	29326	3500	492.61

BELOW: Maximum speed-ups of 1.62 for 2 machines, 2.30 for 3 machines.

Note: one of the machines is simultaneously master and slave



M. Freitas, J. Alves, P. Furtado, Classificação de imagens do synset "dish" do ImageNet usando Apache Spark, in Sistemas de Gestão de Dados, U. Coimbra, 2016 UC-DEI TechReport1010

Some References

Furtado P., "Node-Partitioned Data Warehouses: Experimental Evidence and Improvements",
DOLAP 2004.
Journal of Database Management, Ideas Group, April-June 2006.

Furtado P., Model and Procedure for Performance and Availability-wise Parallel Warehouses. In Distributed and Parallel Databases: Volume 25, Issue 1 (2009), Page 71, Springer-Verlag.

Furtado P., A Survey on Parallel and Distributed Data Warehouses. In International Journal of Data Warehousing and Mining, Ed.-in-Chief D. Taniar, (Editor) IGI Publishing – Vol 5, N. 2, April-June 2009.

João Costa, Pedro Martins, José Cecilio, Pedro Furtado, "Providing Timely Results with an Elastic Parallel DW", in The 20th International Symposium on Methodologies for Intelligent Systems, 4-7 December, 2012, Macau.

Joao Pedro Costa, Pedro Furtado: SPIN, in Dawak 2013.
Nickerson Ferreira, Pedro Furtado: RTDW, in IDEAS 2013.

Expect to publish book by September 2016 (Amazon):
BigData: Concepts, analytics and programming
Spark, Spark Streaming, Realtime, NoSQL and Integration
Pedro Furtado

http://eden.dei.uc.pt/~pnf/

- Thank you all!
  Pedro Furtado
  pnf@dei.uc.pt

Pedro Furtado 2015

The END...questions?

pnf@dei.uc.pt

Take-away:
- Concepts are more useful/relevant than systems

Pedro Furtado 2015