DIGITAL INNOVATION ONE

Aceleração Global Dev

Processando grandes conjuntos de dados de forma paralela e distribuída com Spark

Ivan Pereira Falcão
Expert Data/Cloud Engineer

DIGITAL INNOVATION ONE

Objetivos da Aula

1. Conhecendo o Spark

2. Instalação e Execução

3. SparkSQL

DIGITAL INNOVATION ONE

Requisitos Básicos

- ✓ Conhecimentos básicos de Shellscript
- ✓ Conhecimentos básicos de Linux
- ✓ Conhecimentos básicos de linguagens de programação

DIGITAL
INNOVATION
ONE

Parte 1: Conhecendo o Spark

Processando grandes conjuntos de dados de forma paralela e distribuída com Spark

DIGITAL INNOVATION ONE

Conhecendo o Spark

- Segundo o próprio site do Spark:
  "Apache Spark is a unified analytics engine for large-scale data processing."

- Podemos dizer que o Spark é um framework analítico distribuído, capaz de realizar diversas operações de maneira extremamente rápida.

DIGITAL INNOVATION ONE

Conhecendo o Spark

- Mantido pela fundação Apache (https://www.apache.org/);
- Distribuído de maneira comercial pela Databricks;

DIGITAL INNOVATION ONE

Conhecendo o Spark

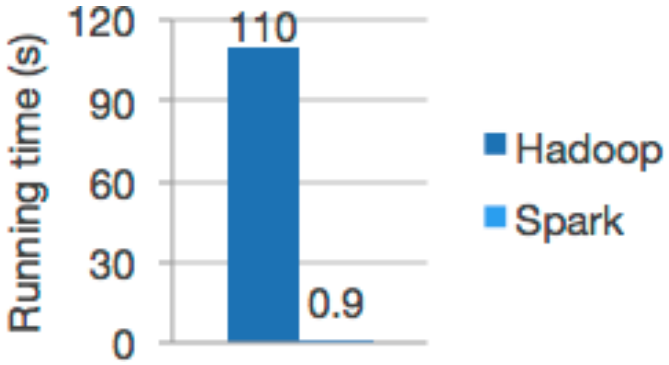
- O Spark é conhecido como um framework in-memory, sendo assim, RAM é fundamental para o bom funcionamento;

- Spark permite que utilizemos as mais diversas linguagens:
  Scala
  Java
  Python
  R
  SQL

Conhecendo o Spark

- Por se tratar de um framework distribuído de big data, somos capazes de trabalhar com quantidades enormes de dados.

- A nível de comparação o Spark consegue ser até 100x mais rápido que o mapreduce tradicional:



DIGITAL INNOVATION ONE

Conhecendo o Spark

- Em 2014, o Spark ganhou o 2014 Gray Sort Benchmark (categoria de 100TB). Em outras palavras, o Spark foi capaz de ordenar 100TB de dados em 23 minutos;

- O recorde anterior era do Hadoop Mapreduce, ordenando o mesmo arquivo em 72 minutos;

DIGITAL INNOVATION ONE

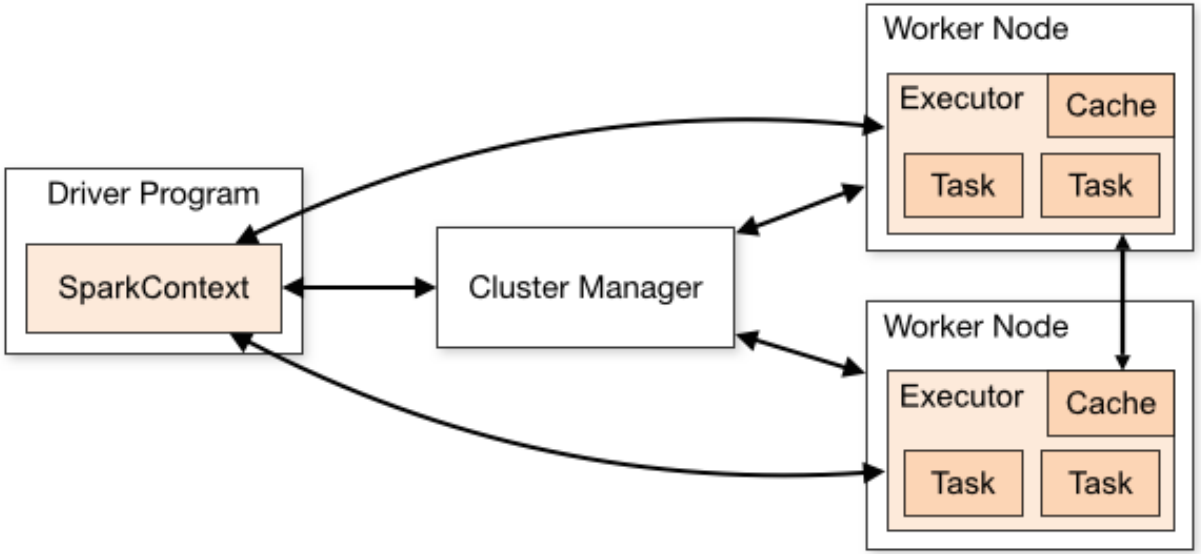
Conhecendo o Spark

- A título de comparação, o Spark utilizou 206 máquinas (EC2 i2.8xlarge), enquanto o cluster Hadoop Mapreduce tinha 2600 nós;
- Resumindo, o Spark realizou a mesma operação 3x mais rápido e com 10x menos recursos;
- Toda operação foi feita em disco, sem fazer uso do cache em memória do Spark.

DIGITAL INNOVATION ONE

Arquitetura

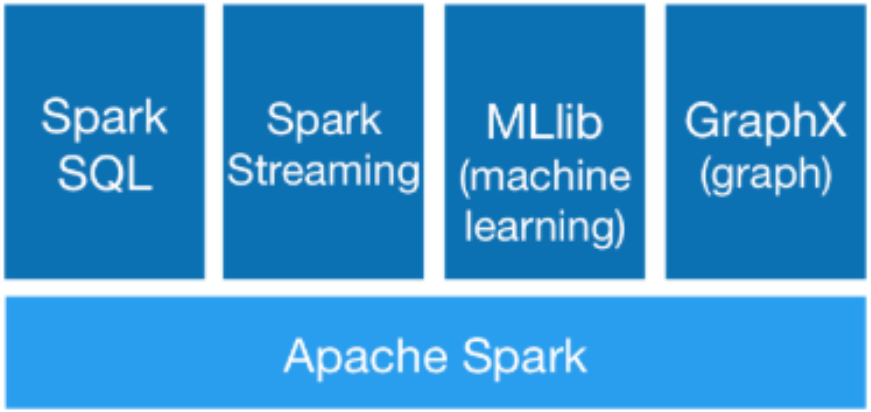
- Entendendo a arquitetura do Spark



DIGITAL INNOVATION ONE

Arquitetura

Podemos dividir o Spark em cinco bibliotecas principais:



DIGITAL INNOVATION ONE

Arquitetura

- Podemos dividir nossos processamentos em:
  - Single node: O Spark rodando em uma única máquina (1);
  - Cluster mode: O Spark distribuirá a carga entre diversas máquinas. Nesse modo dependemos de gerenciadores de recursos:
    - YARN;
    - Mesos;
    - EC2;
    - Kubernetes.

(1) Não confunda o rodar em um único computador com a não distribuição do trabalho; o Spark ainda criará diversos executores dentro de uma mesma máquina, otimizando o processamento para a mesma (recomendo a leitura: https://databricks.com/blog/2018/05/03/benchmarking-apache-spark-on-a-single-node-machine.html);

DIGITAL INNOVATION ONE

Parte 2: Instalação e execução

Processando grandes conjuntos de dados de forma paralela e distribuída com Spark

Instalação

Para instalar o Spark:

- Baixar a versão desejada no site: http://spark.apache.org/downloads.html (trabalharemos com a 2.4.7);
- Descompactar o arquivo (tar -zxvf spark-2.4.7-bin-hadoop2.7.tgz);
- Mover para o local desejado;

Instalação

Para instalar o Spark:

- Configurar a variável de ambiente SPARK_HOME, para a pasta onde se encontra o Spark;

- Para facilitar, configurar no PATH do sistema a pasta bin, dentro do diretório do Spark:

  - No caso do Linux, adicionar ao arquivo /etc/bash.bashrc (Debian) ou /etc/bashrc (RHEL):
    - export SPARK_HOME="/opt/spark-2.3.1-bin-hadoop2.7"
    - PATH="$PATH:$SPARK_HOME/bin"
    - export PYSPARK_PYTHON="python3"

DIGITAL INNOVATION ONE

Instalação

Para instalar o Spark no windows:

- Em geral, utilizamos o Spark em sistemas Linux, mas nada nos impede de utiliza-lo em sistemas Windows. Para tanto:
  - https://medium.com/@dvainrub/how-to-install-apache-spark-2-x-in-your-pc-e2047246ffc3

DIGITAL INNOVATION ONE

Acessando o Spark

O Spark pode ser utilizado de várias maneiras:

- spark-shell;
- pyspark-shell;
- sql-shell;
- R-shell;
- Jupyter Notebooks (Python);
- Zeppelin Notebooks (Python e Scala);
- spark-submit;

DIGITAL INNOVATION ONE

Acessando o Spark

Spark shell:

- O Spark shell é um shell interativo, no qual podemos executar rotinas de spark **em scala** de maneira dinâmica;
- Ele pode ser acessado utilizando o comand spark-shell, no terminal;
- Vale lembrar que o Spark shell criará automaticamente o SparkContext (sc) e o SparkSession (spark);

Acessando o Spark

Spark shell:

- Podemos testar o spark-shell, utilizando os seguintes comandos:
  - Baixar https://raw.githubusercontent.com/fivethirtyeight/data/master/avengers/avengers.csv
  - val insurance = spark.read.format("csv").option("sep", ",").option("header", "true").load("file:///home/everis/avengers.csv")
  - insurance.show()

DIGITAL INNOVATION ONE

Acessando o Spark

Pyspark shell:

- O Pyspark shell também é um shell interativo, porém executamos rotinas de spark **em python**;
- Ele pode ser acessado utilizando o comando pyspark, no terminal;
- Vale lembrar que o Pyspark shell criará automaticamente o SparkContext (sc) e o SparkSession (spark);

DIGITAL INNOVATION ONE

Acessando o Spark

Pyspark shell:

- Podemos testar o pyspark, utilizando os seguintes comandos:

```
insurance = spark.read.format("csv").option("sep", ",").option("header", "true").load("file:///home/everis/avengers.csv")

insurance.show()
```

DIGITAL INNOVATION ONE

Acessando o Spark

Spark SQL:

- O Spark sql é outro shell interativo, no qual podemos executar rotinas de spark **em SQL** de maneira dinâmica;
- Ele pode ser acessado utilizando o comand spark-sql, no terminal;
- Nesse shell, não utilizamos de maneira direta o Spark Context ou Spark Session.

DIGITAL INNOVATION ONE

Acessando o Spark

Spark shell:

- Podemos testar o sql-shell, utilizando os seguintes comandos:

SELECT * FROM csv.`file:///home/everis/avengers.csv`

DIGITAL INNOVATION ONE

Acessando o Spark

Spark R shell:

- O Spark R shell também é um shell interativo, porém executamos rotinas de spark **em R**;
- Ele pode ser acessado utilizando o comando sparkR, no terminal;
- O Spark R depende do R estar instalado no sistema;
- Vale lembrar que o Spark R shell criará automaticamente o SparkContext (sc) e o SparkSession (spark);

DIGITAL INNOVATION ONE

Acessando o Spark

Jupyter notebook:

- O Jupyter notebook é um famoso ambiente interativo para desenvolvimento em python;
- Ele pode ser integrado com o Spark para executar comandos como o pyspark;
- Para instala-lo utilizar o comando pip3 install jupyter.

DIGITAL INNOVATION ONE

Acessando o Spark

Zeppelin Notebook:

- O Zeppelin notebook é um ambiente interativo semelhante ao jupyter; capaz de trabalhar com Scala, Python e Spark de maneira nativa;

- Para o funcionamento juntamente com o Spark é necessária a configuração da variável de ambiente SPARK_HOME.

DIGITAL INNOVATION ONE

RDDs

- O RDD ou Resilient Distributed Dataset é a principal abstração do Spark;
- Ele nada mais é do que uma coleção de elementos particionados entre os diversos nós de um cluster;

DIGITAL INNOVATION ONE

RDDs

- RDDs contém algumas características principais:
  - Ele, como o próprio nome diz, é resiliente a falhas, ou seja, caso haja algum erro durante o processo, ele é capaz de se recuperar e continuar a atividade;
  - RDDs são estruturados para serem naturalmente distribuídos, sendo capazes de existir entre diversos nós de um cluster;
  - Eles são imutáveis. Um RDD gera outro RDD, jamais ele poderá ser modificado. Seu conteúdo poderá ser transformado, resultando em outro RDD.

DIGITAL INNOVATION ONE

Spark Context

- Spark Context:
  - O primeiro passo de um programa Spark é a criação de um contexto;
  - O contexto Spark é o ponto de entrada do programa. É ele o responsável por fazer a comunicação entre o programa e o ambiente;
  - Quando o contexto é criado todos os objetos Spark criados ficam associados a este contexto;
  - Várias propriedades do spark são configuradas diretamente no Spark Context;

Spark Context

- Criando Spark Context:

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkConf

val conf = new SparkConf().setAppName("meu aplicativo spark")
val sc = new SparkContext(conf)
```

DIGITAL INNOVATION ONE

Parte 3: SparkSQL

Processando grandes conjuntos de dados de forma paralela e distribuída com Spark

DIGITAL INNOVATION ONE

Dataframes e Datasets

O SparkSQL consiste em um dos módulos do Spark, sendo ele uma abstração acima do core do Spark. Abaixo algumas características do módulo:

- Trabalha exclusivamente com objetos conhecidos como Dataframes e Datasets. Estes são executados acima do Spark Core (RDDs). Em outras palavras, Dataframes e Datasets nada mais são do que abstrações de tabelas dentro do Spark;

DIGITAL INNOVATION ONE

Dataframes e Datasets

- Apesar dos Dataframes terem uma linguagem própria para manipulação dos dados, podemos trabalhar quase que exclusivamente com SQL (ANSI 2003);

- O Spark nos permite trabalhar com diversas fontes de dados (arquivos em HDFS, tabelas Hive, tabelas HBase, bancos de dados relacionais, etc.) de maneira unificada. Dessa forma, podemos cruzar diversas informações de maneira extremamente simples;

DIGITAL INNOVATION ONE

Spark Session

- Spark Session:
  - O Spark Session é o ponto central do módulo de dataframes;
  - Internamente o Spark Session tem um Spark Context associado;
  - Nas versões anteriores (<2.0) tínhamos a figura do SQLContext e do HiveContext; O Spark Session unificou esses dois componentes;
  - Várias configurações podem ser aplicadas ao Spark Session;

Spark Session

- Criando Spark Session:

```
import org.apache.spark.sql.SparkSession

val spark = SparkSession
  .builder()
  .appName("Spark SQL")
  .config("configuracao", "valor da configuracao")
  .getOrCreate()
```

Spark Session

Em versões antigas do Spark (<2.0):
val sparkConf = new SparkConf()

val sc = new SparkContext(sparkConf).set("spark.some.config.option", "some-value")
val sqlContext = new org.apache.spark.sql.SQLContext(sc)

DIGITAL INNOVATION ONE

Documentação e outras linguagens

- Como dissemos anteriormente, um programa para Spark pode ser desenvolvido nas mais diversas linguagens. Caso queira maiores detalhes sobre cada uma das sintaxes:
  - https://spark.apache.org/docs/2.4.7/
  - https://spark.apache.org/docs/2.3.1/rdd-programming-guide.html
  - https://spark.apache.org/docs/2.3.1/sql-programming-guide.html

SparkSQL

- Podemos carregar uma variedade de dados utilizando o Spark SQL:

```
val dfJson = spark.read.json("file:///home/spark/Downloads/people.json")

val dfParquet = spark.read.format("json").load("file:///home/spark/Downloads/
people.parquet")
```

SparkSQL

- Podemos carregar uma variedade de dados utilizando o Spark SQL:

```
val peopleDFCsv = spark.read.format("csv").option("sep", ",").option("header",
"true").load("file:///home/spark/Downloads/FL_insurance_sample.csv")

val jdbcDF = spark.read
  .format("jdbc")
  .option("url", "jdbc:postgresql:dbserver")
  .option("dbtable", "schema.tablename")
  .option("user", "username")
  .option("password", "password")
  .option("driver", "com.driver.MyDriver")
  .load()
```

SparkSQL

- Temos várias operações associadas aos Dataframes:

```
df.printSchema()

df.show(50,false)

df.select("field1", "field2").show()

df.select($"field1", $"field2"+1).show()

df.filter($"age" > 21).show()

df.groupBy("age").count().show()
```

DIGITAL INNOVATION ONE

SparkSQL

- Temos várias operações associadas aos Dataframes:

```
df.withColumn("new_column_name",
col("old_column_name")).show()

df.withColumn("new_column_name",
col("old_column_name").cast("long")).show()

df.avg("age").show()

df.sum("sales").show()

df.max("age").show()
```

DIGITAL INNOVATION ONE

SparkSQL

- Agora um pequeno exercício:
  - Vamos baixar o arquivo https://raw.githubusercontent.com/shankarmsy/practice_Pandas/master/FL_insurance_sample.csv
  - Obtenham a média do campo eq_site_limit, agrupado por construction

SparkSQL

- Solução:

```
import org.apache.spark.sql.SparkSession

val spark = SparkSession.builder().appName("Spark SQL basic example").
getOrCreate()

val peopleDFCsv = spark.read.format("csv").option("sep",
",").option("header","true").load("file:///home/everis/FL_insurance_sample.csv")

peopleDFCsv.withColumn("eq_site_limit",
col("eq_site_limit").cast("long")).groupBy("construction",
"county").avg("eq_site_limit").show()
```

DIGITAL INNOVATION ONE

Pera.... Mas não era SQL?

DIGITAL INNOVATION ONE

SparkSQL

- Sim! Podemos trabalhar quase que exclusivamente com SQL, para tanto, precisamos cadastrar nosso dataframe:

```
df.createTempView("people")
df.createOrReplaceTempView("people")
df.createGlobalTempView("people")
df.createOrReplaceGlobalTempView("people")
```

- A partir daí conseguimos realizar operações de SQL:

```
spark.sql("SELECT * FROM people").show()
spark.sql("SELECT * FROM global_temp.people").show()
```

DIGITAL INNOVATION ONE

SparkSQL

- Sempre que executamos uma query via sparkSQL, obtemos como retorno outro dataframe:

```
val newDF = spark.sql("SELECT * FROM global_temp.people")
```

- A partir daí conseguimos realizar operações de SQL:

```
newDF.show()
```

- Também podemos carregar um arquivo diretamente via SQL:

```
spark.sql("SELECT * FROM
csv.`file:///home/spark/Downloads/FL_insurance_sample.csv`").
show()
```

SparkSQL

- Outra função interessante é o create temporary view do SQL. Ele permite que criemos uma view temporária sem depender de um dataframe:

```
spark.sql("CREATE TEMPORARY VIEW nova_tabela as (SELECT * FROM people) ")

spark.sql("CREATE GLOBAL TEMPORARY VIEW temp_view AS (SELECT * FROM people) ")
```

SparkSQL

- Agora precisamos gravar nossos dados:

```
val people = spark.sql("SELECT * FROM people")

people.write.format("csv")
.option("sep", ",")
.option("header", "true")
.save("file:///path/output_folder")

people.coalesce(1).write.format("csv")
.option("sep", ";")
.option("header", "true")
.save("file:///path/output_folder")
```

SparkSQL

- E se quisermos os valores particionados?

```
people.write.partitionBy("field_name_1", "field_name_2").format("csv")
.option("sep", ";")
.option("header", "true")
.save("file:///path/output_folder")

people.write.bucketBy(42,
"name").sortBy("age").saveAsTable("people_bucketed")
```

SparkSQL

- E para gravarmos os dados em um banco?

```
people.write
.format("jdbc")
.option("url", "jdbc:postgresql:dbserver")
.option("dbtable", "schema.tablename")
.option("user", "username")
.option("password", "password")
.save()
```

SparkSQL

- Avro depende de um pouco mais de esforço:

```
import com.databricks.spark.avro._

spark.conf.set("spark.sql.avro.compression.codec", "deflate")
spark.conf.set("spark.sql.avro.deflate.level", "5")

val df = spark.read.avro("/tmp/example.avro")

df.write.avro("/tmp/output")
```

Save Mode

- Um ponto importante é o SaveMode:

```
people.write. format("csv")
.option("sep", ";")
.option("header", "true")
.mode(SaveMode.Overwrite).
.save("file:///path/output_folder")
```

DIGITAL INNOVATION ONE

Save Mode

- Tipos de SaveMode:
  - Append
  - ErrorIfExists
  - Ignore
  - Overwrite

UDFs

- UDFs ou User Defined Functions são funções que são definidas pelo usuário e podem ser utilizadas para realizar transformações nos dados:

```
spark.udf.register("minhaUDF", (s: String) => s.length())

spark.sql("SELECT minhaUDF(field_1),* FROM table").show()
```

DIGITAL INNOVATION ONE

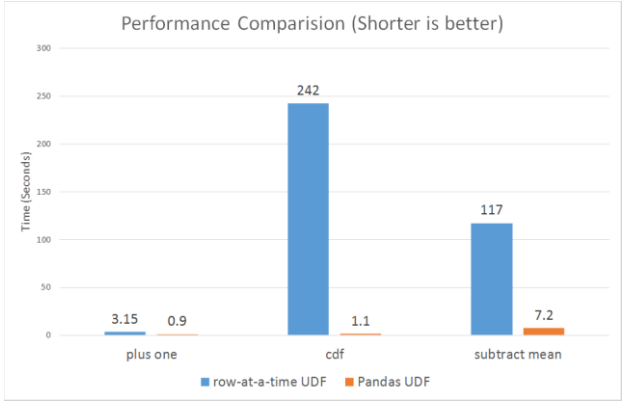
UDFs

- UDAFs ou User Defined Aggregation Functions são semelhantes as UDFs, porém são responsáveis por realizar funções de agregação:
  - Elas são divididas em dois tipos:
    - Untyped: http://spark.apache.org/docs/latest/sql-programming-guide.html#untyped-user-defined-aggregate-functions
    - Typed: http://spark.apache.org/docs/latest/sql-programming-guide.html#type-safe-user-defined-aggregate-functions

DIGITAL INNOVATION ONE

UDFs

- Pandas UDFs foram inseridas na versão 2.3 do Spark;
- São UDFs otimizadas para python, utilizando as capacidades do Apache Arrow;
- Elas melhoram muito a velocidade de execução de UDFs em python;



UDFs

- Utilizando Pandas UDFs:

```
from pyspark.sql.functions import col, pandas_udf
from pyspark.sql.types import LongType

def multiply_func(a, b):
    return a * b

multiply = pandas_udf(multiply_func, returnType=LongType())

df.select(multiply(col("x"), col("x"))).show()
```

DIGITAL INNOVATION ONE

UDFs

- Importante!
  - UDFs, UDAFs e Pandas UDFs são naturalmente mais pesadas e impactarão na performance do seu processo;
  - Sempre que possível execute operações diretamente com comandos de Dataframe ou código SQL nativo:

https://spark.apache.org/docs/2.4.7/api/sql/index.html

Spark e Hive

- Podemos acessar diretamente o Hive via spark:

```
import java.io.File
import org.apache.spark.sql.{Row, SaveMode, SparkSession}
import spark.implicits._
import spark.sql

val warehouseLocation = new File("spark-warehouse").getAbsolutePath

val spark = SparkSession
  .builder()
  .appName("Spark Hive")
  .config("spark.sql.warehouse.dir", warehouseLocation)
  .enableHiveSupport()
  .getOrCreate()
```

DIGITAL INNOVATION ONE

Spark e Hive

- Podemos acessar diretamente o Hive via spark:

```
spark.sql("CREATE TABLE IF NOT EXISTS src (key INT, value STRING)
USING hive")
spark.sql("LOAD DATA LOCAL INPATH
'examples/src/main/resources/kv1.txt' INTO TABLE src")

spark.sql("SELECT * FROM src").show()


val df = spark.table("src")
df.write.mode(SaveMode.Overwrite).saveAsTable("hive_records")
```

DIGITAL INNOVATION ONE

Persist e Cache

- A seguir veremos alguns pontos importantes para melhorar a performance dos nossos programas:
  - cache e persist: Em geral o Spark pode precisar refazer uma determinada transformação várias vezes a cada ação. Para otimizar nossos programas podemos usar o conceito de persist (ou cache), comando que armazenará os dados na memória para ser reutilizado;
  - Pode melhorar muito a performance do seu projeto, mas deve ser usada com parcimônia, dependendo dos recursos do cluster;
  - No próximo slide veremos os possíveis storage levels para persistência;

Persist e Cache

Exemplo:
val dfJson = spark.read.json("
file:///examples/src/main/resources/people.json")

dataframe.persist(StorageLevel.MEMORY_AND_DISK)
dataframe.unpersist()
dataframe.cache()

Persist e Cache

Storage Level	Meaning
MEMORY_ONLY	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly each time they're needed. This is the default level.
MEMORY_AND_DISK	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, store the partitions that don't fit on disk, and read them from there when they're needed.
MEMORY_ONLY_SER (Java and Scala)	Store RDD as *serialized* Java objects (one byte array per partition). This is generally more space-efficient than deserialized objects, especially when using a fast serializer, but more CPU-intensive to read.
MEMORY_AND_DISK_SER (Java and Scala)	Similar to MEMORY_ONLY_SER, but spill partitions that don't fit in memory to disk instead of recomputing them on the fly each time they're needed.
DISK_ONLY	Store the RDD partitions only on disk.
MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc.	Same as the levels above, but replicate each partition on two cluster nodes.
OFF_HEAP (experimental)	Similar to MEMORY_ONLY_SER, but store the data in off-heap memory. This requires off-heap memory to be enabled.

DIGITAL INNOVATION ONE

Spark Submit

Spark submit:

- É a maneira mais comum de executarmos um sistema em Spark;
- Permite a configuração de diversos parâmetros do Spark;
- Pode trabalhar tanto com códigos python (.py, .whl, .zip) quanto com pacotes .jar (e mais recentemente .R);

DIGITAL INNOVATION ONE

Spark Submit

Spark submit:

- Basicamente é chamado utilizando o seguinte comando:

```
spark-submit \
  --class <main-class> \
  --master <master-url> \
  --deploy-mode <deploy-mode> \
  --conf <key>=<value> \
  ... # other options
  <application-jar> \
  [application-arguments]
```

Spark Submit

```
# Run application locally on 8 cores
./bin/spark-submit \
  --class
org.apache.spark.examples.SparkPi \
  --master local[8] \
  /path/to/examples.jar \
  100
```

```
# Run on a Spark standalone cluster
in client deploy mode
./bin/spark-submit \
  --class
org.apache.spark.examples.SparkPi \
  --master spark://207.184.161.138:7077
\
  --executor-memory 20G \
  --total-executor-cores 100 \
  /path/to/examples.jar \
  1000
```

Spark Submit

```
# Run on a Spark standalone cluster in cluster deploy mode with supervise
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master spark://207.184.161.138:7077 \
  --deploy-mode cluster \
  --supervise \
  --executor-memory 20G \
  --total-executor-cores 100 \
  /path/to/examples.jar \
  1000
```

```
# Run on a YARN cluster
export HADOOP_CONF_DIR=XXX
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master yarn \
  --deploy-mode cluster \  # can be client for client mode
  --executor-memory 20G \
  --num-executors 50 \
  /path/to/examples.jar \
  1000
```

Spark Submit

```
spark-submit \
    --class=com.everis.tricorder.TricorderRun \
    --properties-file spark.conf\
    --files log4j.properties,[Outros arquivos] \
    --conf "spark.executor.extraJavaOptions=-
Dlog4j.configuration=file://log4j.properties" \
    --conf "spark.driver.extraJavaOptions=-
Dlog4j.configuration=file://log4j.properties" \
    ./target/app.jar \
    arg01 “val_arg01" \
    arg02 “val_arg02"
```

Spark Submit

spark.master yarn
spark.app.name "tricorder"
spark.yarn.queue Desenvolvimento
spark.dynamicAllocation.enabled true
spark.dynamicAllocation.initialExecutors 1
spark.dynamicAllocation.minExecutors 5
spark.dynamicAllocation.maxExecutors 10
spark.shuffle.service.enabled true
spark.executor.cores 6
spark.driver.cores 8
spark.executor.memory 10G

Spark Submit

```
#spark.yarn.executor.memoryOverhead 1000
spark.driver.memory 20G
#spark.yarn.driver.memoryOverhead 1000
#spark.ui.port 4142
spark.ui.enabled false
spark.shuffle.compress true
spark.driver.maxResultSize 5000m
spark.default.parallelism 20000
spark.executor.heartbeatInterval 10s
spark.dynamicAllocation.sustainedSchedulerBacklogTimeout 1s
spark.dynamicAllocation.cachedExecutorIdleTimeout 120s
```

Spark Submit

spark.dynamicAllocation.executorIdleTimeout 60s
spark.sql.broadcastTimeout 36000
spark.network.timeout 600s
spark.serializer org.apache.spark.serializer.KryoSerializer
spark.sql.shuffle.partitions 20000
spark.hadoop.hive.exec.dynamic.partition true
spark.hadoop.hive.exec.dynamic.partition.mode nonstrict

Spark Submit

```
# Define the root logger with appender X
log4j.rootLogger = INFO,stdout
log4j.logger.com.everis = DEBUG,stdout

# Direct log messages to stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd
HH:mm:ss} %-5p %c{1}:%L - %m%n
```

DIGITAL
INNOVATION
ONE

Spark Submit

Spark submit:

- Lista com todas as configurações do spark-submit:
  - https://spark.apache.org/docs/latest/configuration.html

DIGITAL
INNOVATION
ONE

Dúvidas?

Processando grandes conjuntos de dados de forma paralela e distribuída com Spark