# Apache Hadoop com Apache Spark

## Diferencias

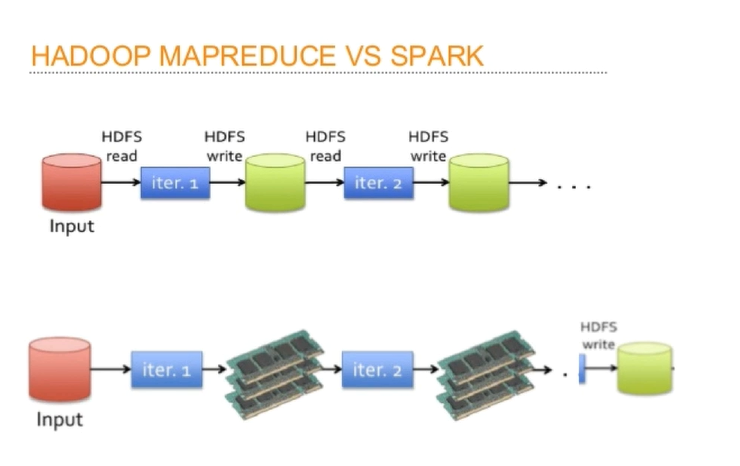
Podemos comparar o Apache Spark com o Map reduce. Mas o Apache Saprk não possui um sistema de armazenamento, podendo usar por exemplo o HDFS.

1. Custo: ambos são open sources mas para montar o cluster precisamos de maquinas e professionais capacitados, etc.
2. Velocidade: Comparando o spark com o hadoop map reduce, nesta comparação o Spark é mais veloz dependendo do grande volumem de dados o mapreduce pode ser uma boa alternativa.
3. Versatilidade: O Spark é mais versátil porque suporta outras linguagens, os dois suportam linguagem java, mais o Spark aceita linguagem python, R e Scala.
4. Competências: Tanto Spark como Hadoop requere professionais qualificados.

Embora alguns departamentos de TI, podem se sentir comprometidos em escolher entre Spark e Hadoop, o fato é que muitas empresas escolheram os dois por ser tecnologias complementares. Mas é totalmente possível usar um sem o outro.

## Diagrama que mostra a diferencia entre Hadoop Mapreduce e Spark

Basicamente o grande diferencial é que o Spark faz as gravações intermediarias em memória sendo mais rápido mesmo assim com um volumem de dados grande as vezes precisa gravar em disco. A escolha de qual usar vai depender do grande volumem de dados, tipo de processamento e as configurações do cluster.



## Hadoop Spark juntos

Imagina que queremos processamento de dados em tempo real, Spark faz isso o Hadoop não.

Imagina que quer trabalhar com agrupamento de dados o Spark faz isso facilmente. O spark é considerado o futuro do processamento distribuído no ecossistema Hadoop.

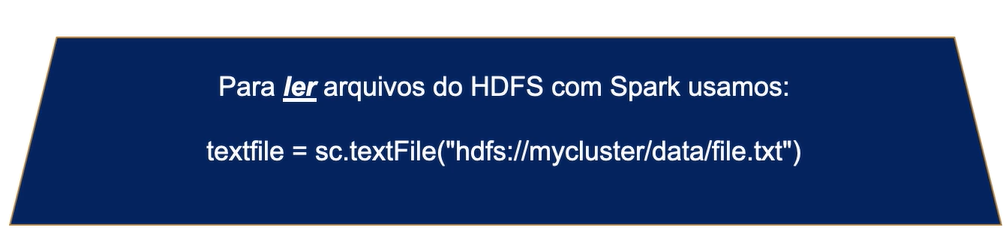
Com o Spark podemos realizar:

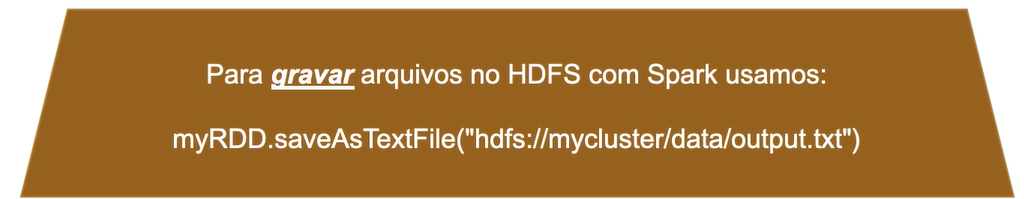
* Operações de ETL
* Análises preditivas e Machine Learning
* Operações de acessos de Dados com sql.
* Text Mining: Mineração de textos
* Processamento de eventos em tempo real (Apache Spark Streamng)
* Aplicações gráficas
* Reconhecimento de Padrões em grandes conjuntos de dados.
* Sistemas de Recomendação.

O Spark e normalmente utilizado com HDFS, mas outros sistemas de arquivo ou sistemas de armazenamento podem ser usados como

1. Sistemas arquivos local ou de rede (NFS)
2. Amazon S3 (arquivo distribuído em nuvem da amazon)
3. RDBMS
4. Nosql (Apache Cassandra, HSDB, etc)
5. Sistemas de mensagens Kafka.

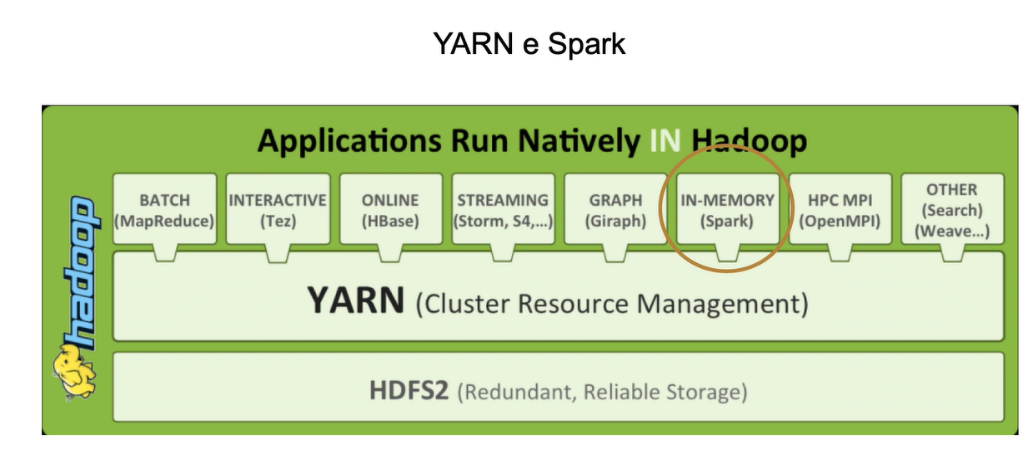
## Como Spark Funciona sobre HDFS





## Instalação do Spark

* Standalone: o Spark e executado sem um agente de tarefas, ele vai usar o agente interno e não o yarn por exemplo.
* Yarn(Hadoop)
* Mesos



## Porque usar Spark para construir meu processo, se pode fazê-lo em Python ou R?

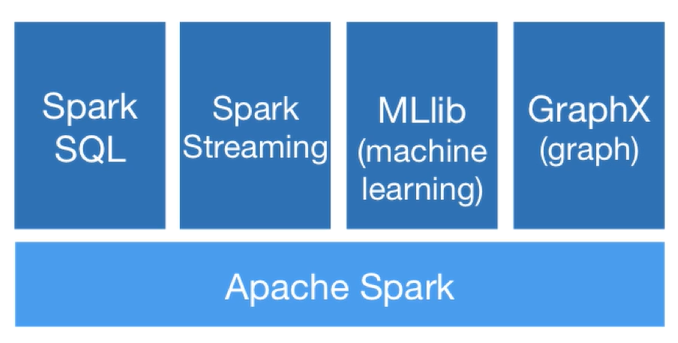
1. Porque Spark suporta um grande volumem de dados,
2. Porque você quer usar as APIs prontas do Spark de SQL ou Streaming, por exemplo.

## O que é uma iteração?

É repetir uma determinada operação em machine learning precisamos que os dados passem várias e várias vezes (varias iterações), o spark e ideal para isso porque trabalha os dados em memoria.

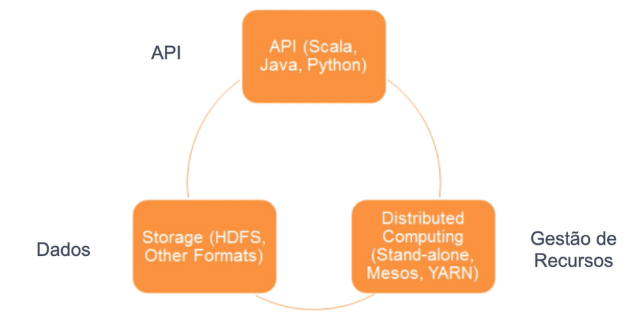
## Esquema do Spark

Essas são as APIs mais importante do framework Spark. Spark streaming para ser gerados em tempo real, exemplo pegar os dados em twitter, Mllib é uma biblioteca para Machine Learning com algoritmos prontos para fazer executados de maneira distribuída, GraphX para trabalhar grafos, existem outros conectores como Spark R , e um conector para trabalhar com o Cassandra



## Arquitectura de Spark

Em uma aplicação de análises de dados vai usar um API, um gestor do cluster e um lugar onde armazenar e ler os dados

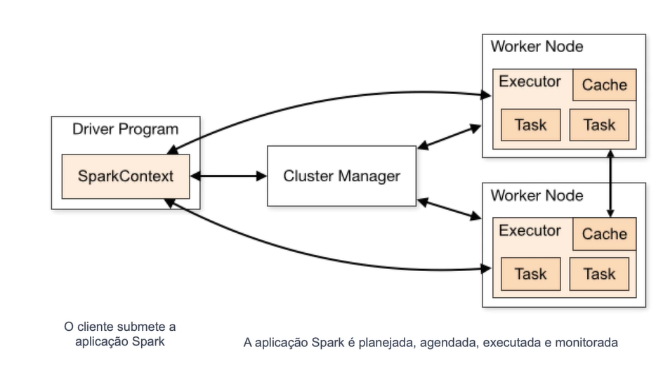


### Processamento de uma aplicação Spark

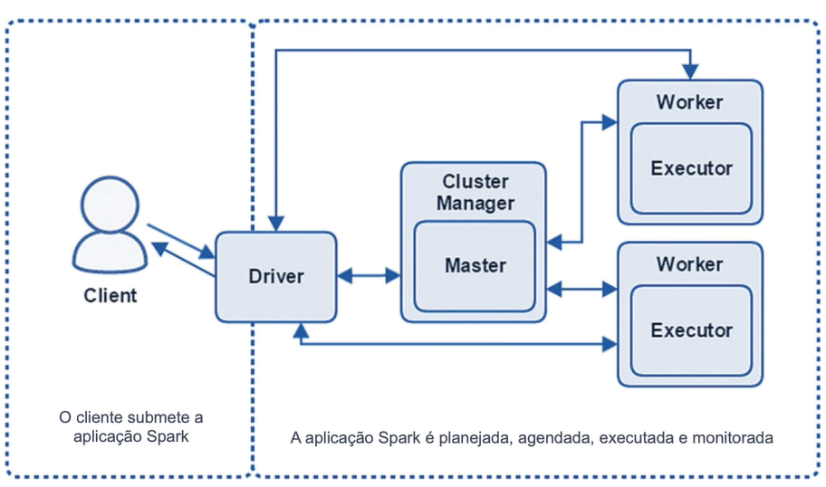
Driver Program imagina que você esta trabalhando com Pyspark, no momento da execução do comando Pyspark , você abre um Driver Program, que é a forma que você vai se conectar com seu cluster Spark. Quando você abre o driver você está criando um contexto sc (spark Context), é uma área de trabalho para acionar o cluster manager, em no caso do hadoop vai ser o Yarn.

Quando o cluster manager pede a execução ele vai enviar o pedido ao Worker Manager (uma máquina que compõe um cluster Spark). Dentro do worker node temos o executor que executa as tarefas elas podem ser executadas de maneira simultânea. Dependendo do tipo de aplicação o posso falar que queiro gravar os dados em Cache, uma vez que o Worker Node término seu trabalho ele pode responder para o cluster manager ou ´para o Spark Context.

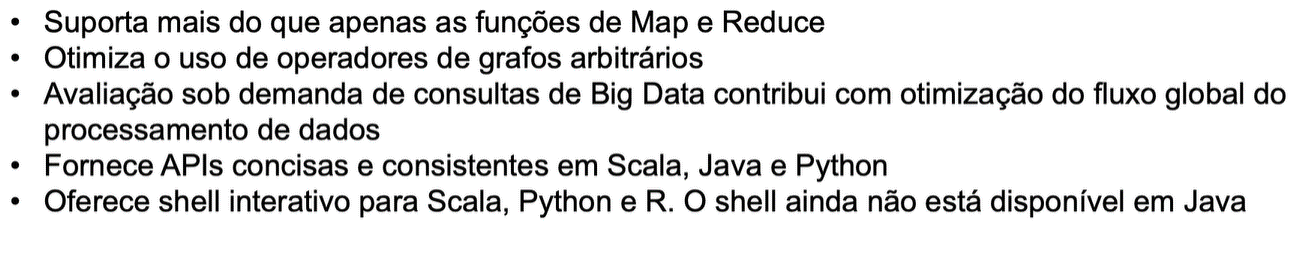
Diagrama

Descrição gerada automaticamente

#Anatomia de uma aplicação Spark



## Outras características do Spark



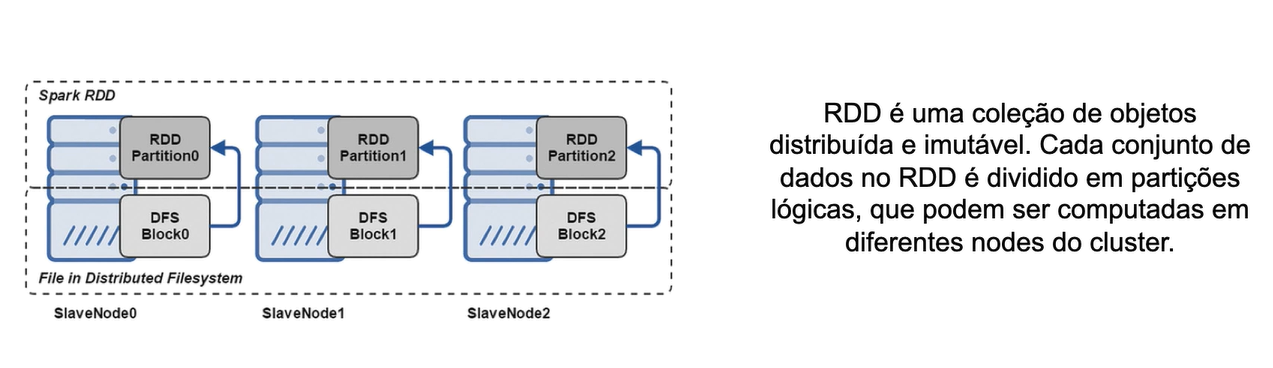
# RDDs e Dataframes

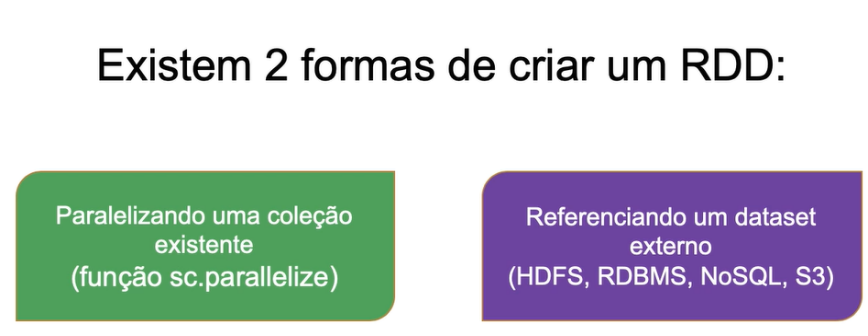
O apache Spark é uma ferramenta de processamento de dados.

Para isso precisamos armazenar isso em uma estruturas: RDD(Resilent Distributed Datasets) e Dataframes. Em resumo com RDDs eu trabalho dados não estruturados e com dataframes eu trabalho dados estruturados ou semiestruturados.

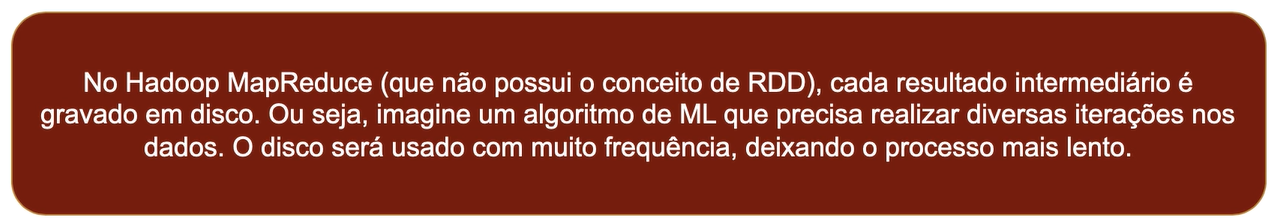
**## RDD:** ele é uma coleção de objetos distribuída e imutável. Ele é o conceito central do framework Spark.

Que significa imutavel? Quando você cria um RDD não pode alterar ele, cada vez que você quer transformar o RDD deve criar um RDD. Não pode ser alterado. Ele é tolerante a falhas e criando em forma paralela.

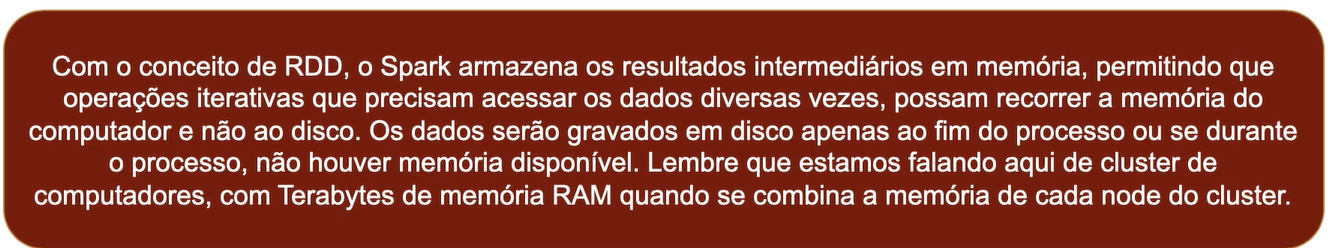




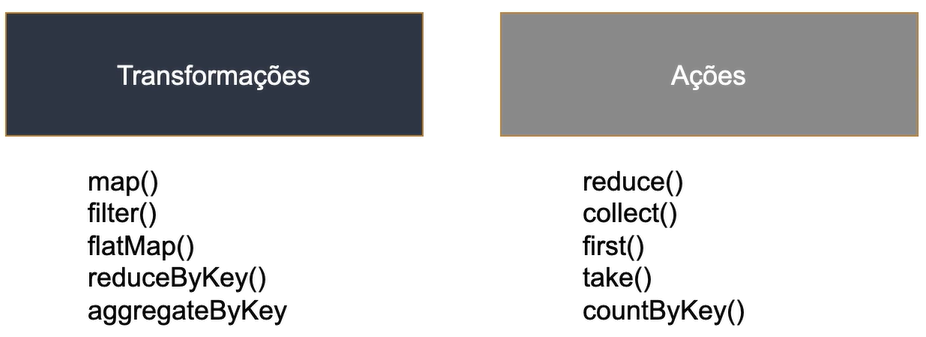
### No Hadoop Mapreduce



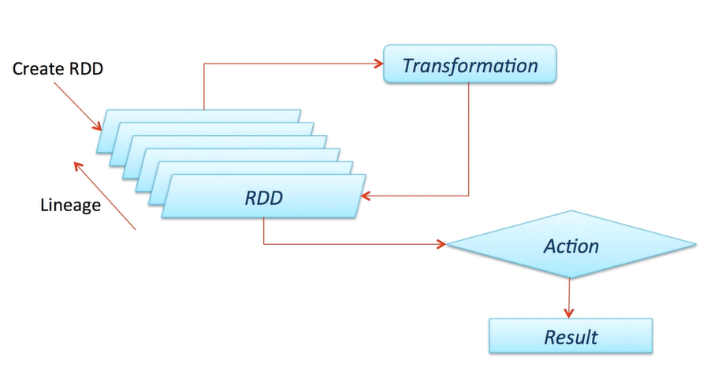
### Spark



Os RDDs podem ser particionados e persistidos em memória e disco. Ele suporta dois tipos de operações: transformações e as ações. A transformação ele não altera o RDD , ele cria um novo RDD ate realizar uma ação. Esse processo é chamado Lazy Evaluation. Com isso podemos utilizar dos funciones cache() e persistant(), que nos permitem fazer a persistência de um rdd em memoria ou disco.

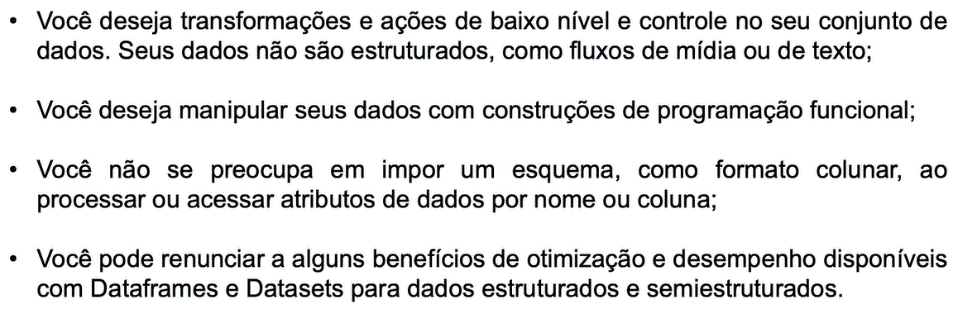


### Lazy Evaluation

Diagrama

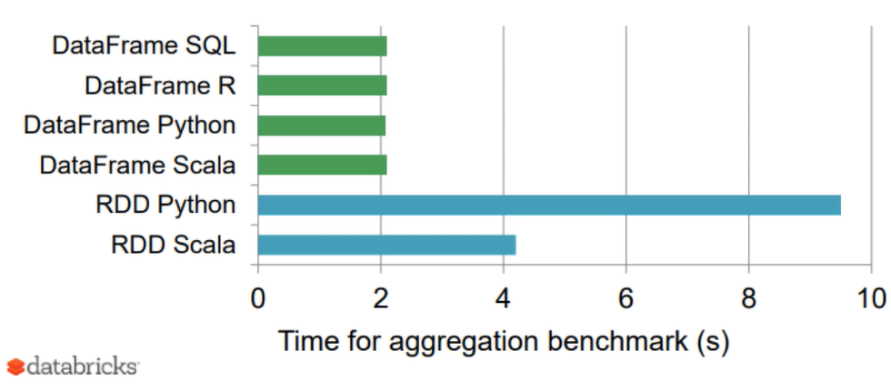
Descrição gerada automaticamente

### Quando usamos RDDs



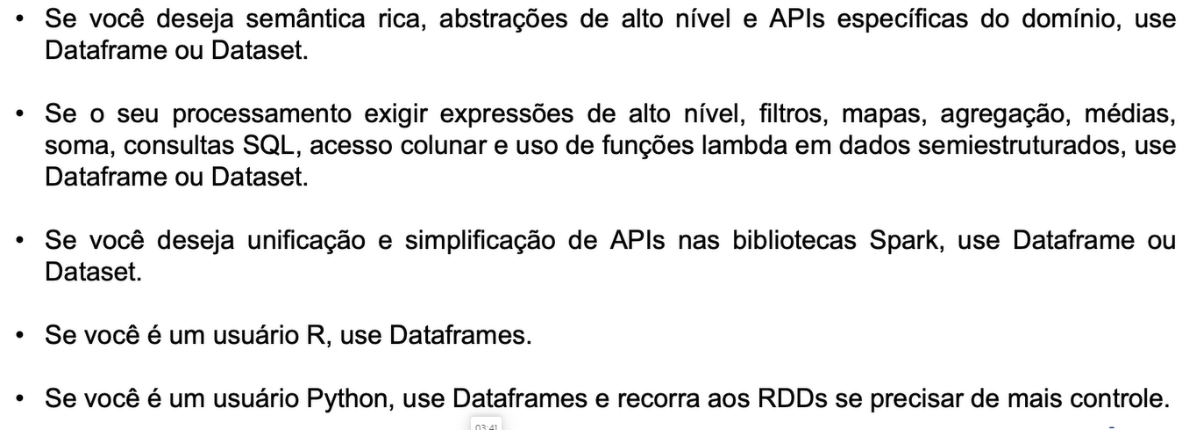
**## Dataframes:**

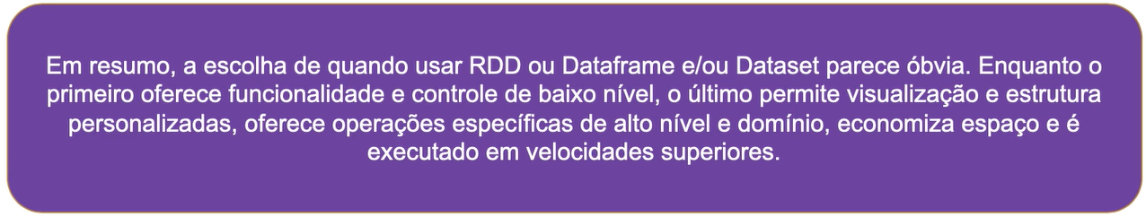
Os dataframes são mais rápidos que o RDD, olha o tempo em segundos na seguinte tabela

****

1. Como um RDD os dataframes são uma construção distribuída imutável de dados.
2. Ao contrário do RDD, os dataframes são organizados em colunas nomeadas, como uma tabela de um banco de dados relacional. Por isso que ele é mais útil para dados estruturados.
3. Projetados para facilitar ainda mais o processamento de grandes conjuntos de dados, o dataframe permite que os desenvolvedores imponham uma estrutura em uma coleção distribuída de dados, permitindo abstração de nível superior.

### Quando usamos Dataframes?





## Testando Spark Standalone

### Executar Spark Pi

```sh

spark-submit --class org.apache.spark.examples.SparkPi --master local $SPARK\_HOME/examples/jars/spark-examples\_2.11-2.4.3.jar

```

### Projeto

Vamos pegar um arquivo de textos para contabilizar de maneira analítica a quantidade de palavras desse arquivo de texto. Isso pode ser usado depois para uma análise de sentimento. Para este exemplo vamos pegar um texto qualquer e gravar em input.txt

```sh

cd ~

gedit input.txt

```

Agora com o pyspark vamos construir uma aplicação analítica usando Mapreduce em no sublimetext,

```sh

subl app.py

```

Em no editor de texto app.py, como vamos usar o spark-summit , ele vai encaminhar para o apache spark por isso preciso criar o Spark contexto. No Jupiter notebook ou spark shell o Spark cria automaticamente para você. O SparkConf, vai criar a configuração nome da aplicação, servidor que vou usar etc. SparkContext vai criar o contexto que é um driver.

```sh

import sys

from pyspark import SparkContext, SparkConf

if \_\_name\_\_ == “\_\_main\_\_”:

# Criar Spark Context

conf = SparkConf().setAppName("Conta Palavras").set("master", "local")

sc = |SparkContext(conf = conf)

#Carregar arquivos. Textfile cria um RDD, onde posso fazer transformações(gera outro #RDD) e ações

palavras = sc.textFile("/home/hadoop/input.txt").flatMap(lambda line: line.split(" "))

#Contagem - Mapreduce

contagem = palavras.map(lambda palavra: (palavra, 1)).reduceByKey( lambda a,b: a+b )

#Salvar o resultado

contagem.saveAsTextFile("/home/hadoop/saida")

```

* Para executar app.py

```sh

spark-submit app.py

cd saida

```

## Como fazer o acesso remoto.

Vai Spark arquives e procura a versão 2.4.4

<https://archive.apache.org/dist/spark/spark-2.4.4/spark-2.4.4-bin-hadoop2.7.tgz>

No Terminal:

```sh

cd Downloads

wget <https://archive.apache.org/dist/spark/spark-2.4.4/spark-2.4.4-bin-hadoop2.7.tgz>

cd /opt/

ls -la

sudo mv /opt/spark /opt/spark-2.4.3

cd ~

cd Downloads

tar -xvf spark-2.4.4-bin-hadoop2.7.tgz

sudo mv spark-2.4.4-bin-hadoop2.7 /opt/spark

cd /opt/

cd ~

```

Verificando as variáveis de ambiente:

```sh

gedit .bashrc

```

## Configurando o cluster Spark

Para verificar a direção ip da máquina virtual ifconfig

```sh

cd /opt/spark/conf

cp spark-env.sh.template spark-env.sh

cp slaves.template slaves

ls -la

gedit spark-env.sh

```

No arquivo agregar spark-env.sh:

```sh

export SPARK\_LOCAL\_IP=192.168.0.81

export SPARK\_MASTER\_HOST=192.168.0.81

```

No arquivo slaves, tirar o localhost e colocar ip da máquina:

```sh

192.168.0.81

```

### Inicializar o cluster de Spark (Master e Worker)

```sh

cd ~

cd /opt/spark/sbin/

./start-all.sh

jps

```

**./start-slave.sh spark://** localhost:7077

sudo ufw allow from 192.168.0.18

### No Firefox a porta do spark é 8080

```sh

192.168.0.81:8080

```

### Para ver os logs vai /opt/spark/logs/

Vai observar dois arquivos um para a inicialização do master e outro para o worker:

```sh

cd /opt/spark/logs/

gedit spark-hadoop-org.apache.spark.deploy.master.Master-1-dataserver.out

```

## Configurar a máquina cliente

Lembra você tem que usar a mesma versão que você usa no servidor

Em Windows remover as seguintes variáveis

```sh

PYSPARK\_DRIVER\_PYTHON jupyter

PYSPARK\_DRIVER\_PYTHON\_OPTS notebook

PYSPARK\_PYTHON python

```

Mudar:

```sh

SPARK\_HOME C:\opt\spark

C:\opt\spark\bin

```

Terminal para conectar remotamente, na sua máquina:

```sh

C:\Users\paola>spark-shell --master spark://192.168.0.81:7077

```

Criar

```sh

if \_\_name\_\_ == "\_\_main\_\_":

# Criar Spark Context

conf = SparkConf().setAppName("Conta Palavras").setMaster("spark://192.168.0.81:7077")

sc = SparkContext(conf = conf)

#Carregar arquivos. Textfile cria um RDD, onde posso fazer transformaÃ§Ãµes(gera outro RDD)

palavras = sc.textFile("input.txt").flatMap(lambda line: line.split(" "))

#Contagem - Mapreduce

contagem = palavras.map(lambda palavra: (palavra, 1)).reduceByKey( lambda a,b: a+b )

#Salvar o resultado

contagem.saveAsTextFile("saida")

```

Executar Na máquina:

spark-submit app.py

No Vitrtual Machine você vai ver onde você salvou a saída um diretório \_temporary, dentro dele um diretório 0, e dentro dele vai ver que vai ter uma serie de diretórios um para cada execução de tarefa.