

BIG DATA



Disciplina: Introdução ao Spark

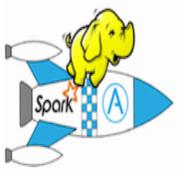
Tema da Aula: Spark Core

Coordenação:

Prof. Dr. Adolpho Walter
Pimazzi Canton

Profa. Dra. Alessandra de
Ávila Montini

Prof. Samuel Otero Schmidt



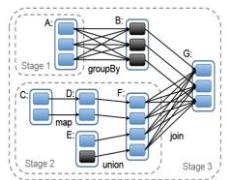
Hadoop e Spark

Spark



Cases com Spark

Spark Cluster

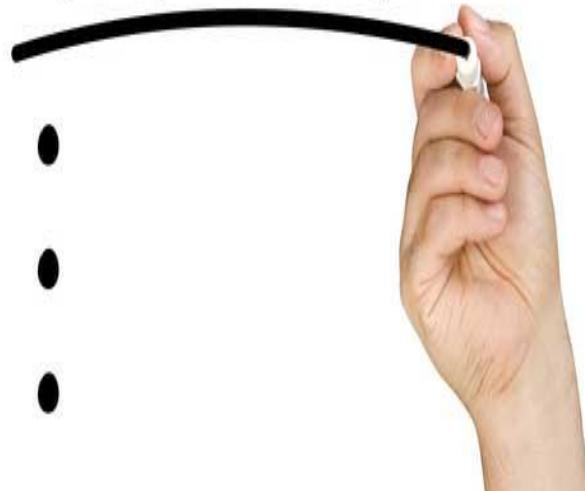


Spark Jobs, Estágios e Tarefas

Spark
Streaming



AGENDA



Hadoop e Spark

Big Data: presente e futuro

*Protocolo /
Linguagem/
Framework:*

*Não Inteligente
(Dumb Data)*

*Dados Inteligentes
(Smart Data)*

*Dados muito Inteligentes
(Very Smart Data)*



Formato:

*Tecnologia/
Conceito:*

*Fornecedor/
Player:*

Fonte: MapR (2014)

Big Data: presente e futuro

Protocolo / Linguagem/ Framework:	SCSI	FS / NFS / CFIS	Big Data	SQL	SQL Analítico
Tipo de dado:	Não Inteligente (Dumb Data)	Não-estruturado	Semi-estruturado	Dados Inteligentes (Smart Data)	Dados muito Inteligentes (Very Smart Data)
Formato:	Bloco	Arquivo	Registros (Records) em arquivos	Registros tabulados com acesso aleatório	Tabelas (Modelo Relacional)
Tecnologia/ Conceito:	- HDD - Flash	- Abrir/Ler/ Escrever/Fechar	- SQL-on-Hadoop MapReduce / Hive / Pig / Spark	- NoSQL HBase, MongoDB, Cassandra	- Banco de Dados (OLTP)
Fornecedor/ Player:	- EMC - Seagate	- NetApp - EMC (Isilon)	- Apache - Cloudera, Hortonworks - MapR - DataBricks	- Apache - MongoDB - Cloudera - Hortonworks - DataStax	- Oracle - Sybase - MySQL
					- Teradata - Netezza - Vertica

Fonte: MapR (2014)

Sistemas distribuídos: Problemas

Programação em sistemas distribuídos é complexa:

- Requer sincronização dos dados;
- Largura de banda é finita;
- É difícil lidar com falhas parciais no sistema.

Fonte: Sawant et al (2013)

Dado é o gargalo

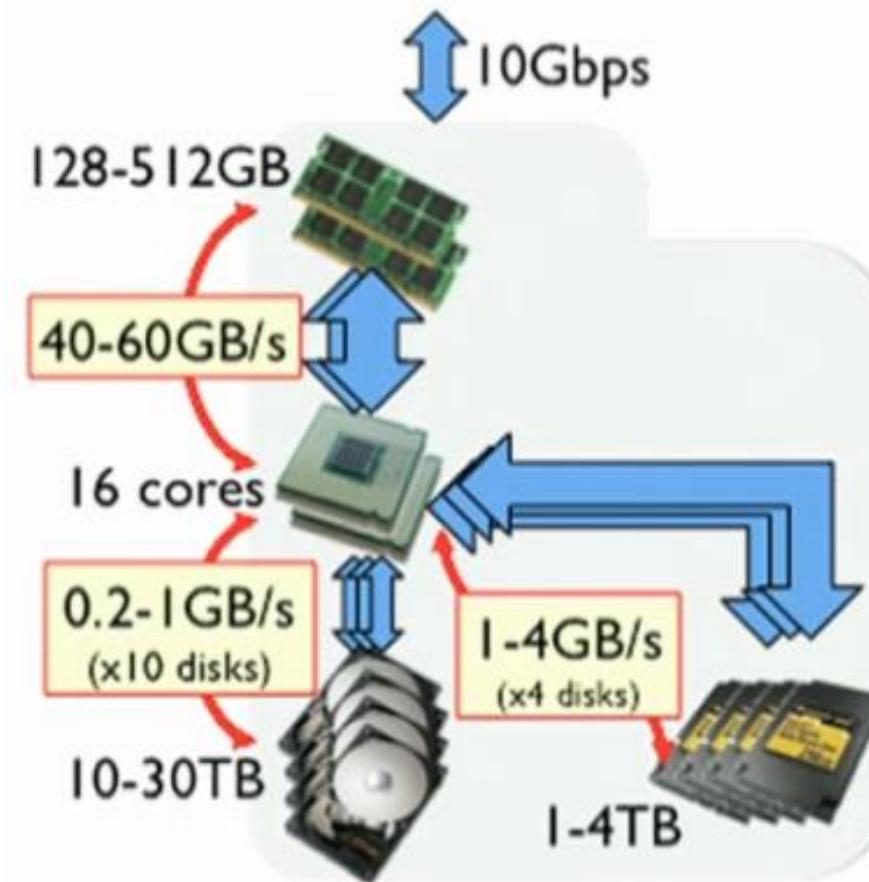
Pegar o dado e levar para o processador se tornou um gargalo.

- A capacidade de armazenamento dos discos cresceu massivamente. No entanto, a velocidade de acesso aos dados não acompanhou esse crescimento.
 - » - Um disco comum em 1990 podia armazenar 1,370 MBs de dados e tinha uma velocidade de transferência de 4.4 MB/s. Era possível ler todo o conteúdo do disco em aproximadamente 5 minutos.
 - » - Após 20 anos, discos de 1 TB são comuns, mas a transferência de dados aumentou apenas para 100MB/s. Para ler todo o conteúdo do disco leva mais de 2 horas e 30 min.
 - » - Solução: para reduzir o tempo de acesso aos dados, é necessário realizar o acesso à vários discos de uma vez. Distribuindo 1 TB em 100 discos, serão armazenados 10 GBs em cada disco, o acesso a todo o dado armazenado leva aproximadamente 2 minutos.

Fonte: White (2012)

Por que processamento em memória?

Diferença entre de desempenho de memória e disco é grande.



Fonte: <http://ampcamp.berkeley.edu/>

Processamento em MapReduce versus Spark

- Hadoop introduziu uma nova forma radical baseada em dois conceitos chave:
 - Distribuir os dados quando são armazenados;
 - Executar o processamento onde o dado está.
- Spark introduz uma nova forma levando isso a outro patamar:
 - Os dados são distribuídos em memória
- Conceito de MapReduce:

```
cat      | grep  | sort  | uniq  > arquivo  
entrada | map   | shuffle | reduce > saída
```

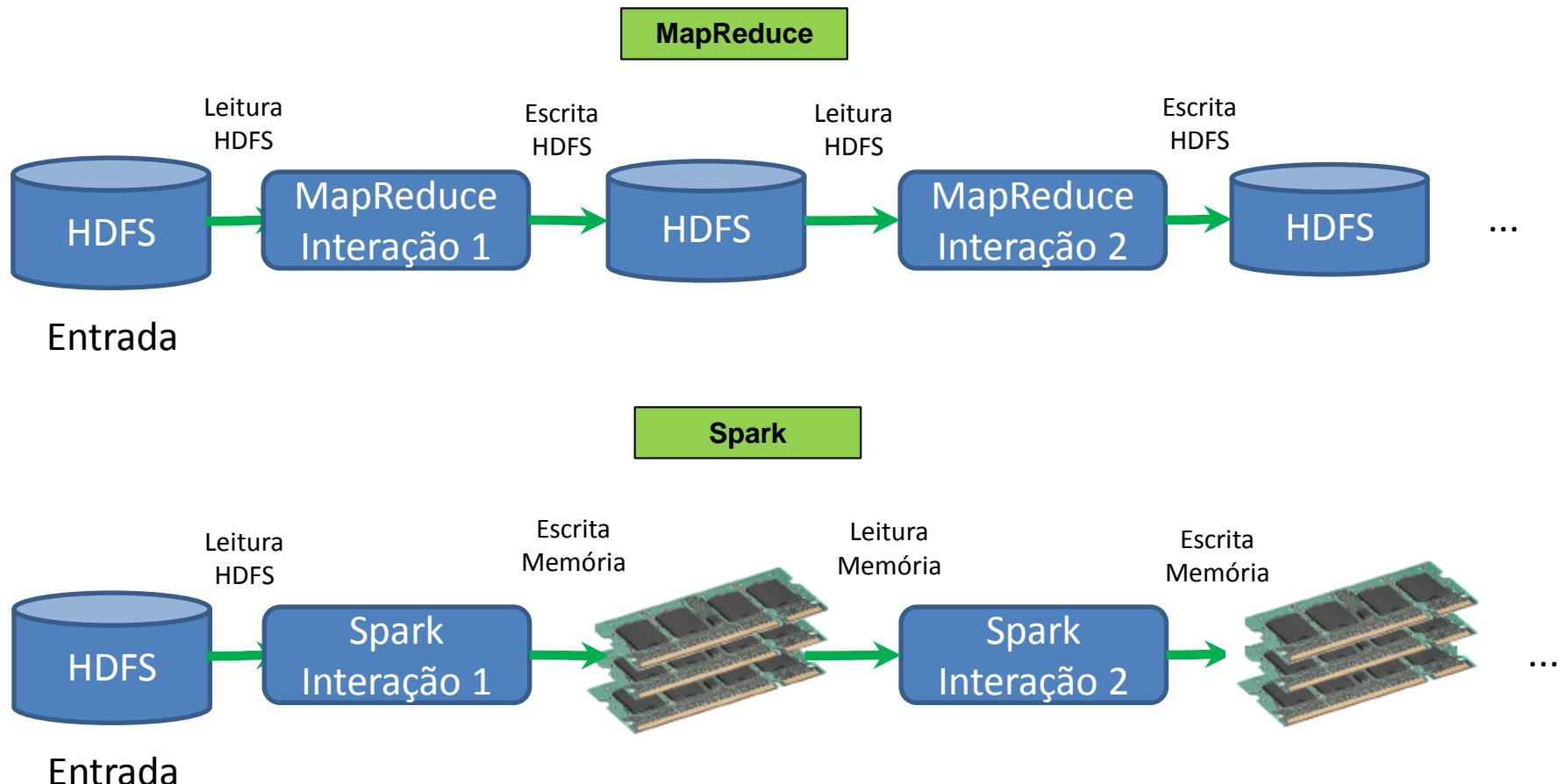
O Map

Map(key1, valor) -> lista <key2,value2>

Reduce

Reduce (key2, lista <valor2>) -> lista <value3>

Processamento em MapReduce versus Spark



Fonte: ZAHARIA et al (2015)

Spark

O Que é o Apache Spark?

- Apache Spark é uma engine rápida para processamento em larga escala desenvolvida em linguagem Scala.
- Originalmente desenvolvida no AMPLab na UC Berkeley
 - Começou como um processo de pesquisa em 2009.
 - Em 2010 o seu código foi aberto e em 2013 foi transferido para ASF.
- Atualmente é um projeto top-level.
- Projeto de Código Aberto na Fundação Apache:
 - Commiters da Cloudera, Yahoo, Databricks, UC Berkeley, Intel, Groupon,...
 - Um dos mais ativos e um dos projetos com maior crescimento da Fundação Apache
- É possível fazer aplicações Spark
 - Em Python, Scala ou Java
 - Suporte a R e Dataframes.

Fonte: ZAHARIA et al (2015)

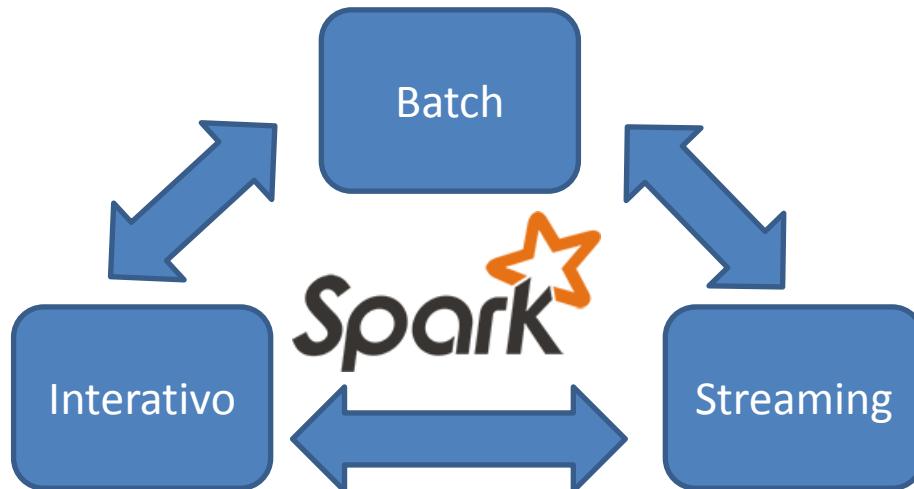
Vantagens do Spark

- Programação de alto nível
 - Programadores podem focar na lógica, e não no encanamento.
- Computação em Cluster
 - Processos são distribuídos ao redor do cluster.
 - Gerenciado por um único “master”.
 - Escalável e tolerante a falhas.
- Dados em memória
 - Cache configurável para iteração eficiente.

Fonte: *Shapira et al (2015)*

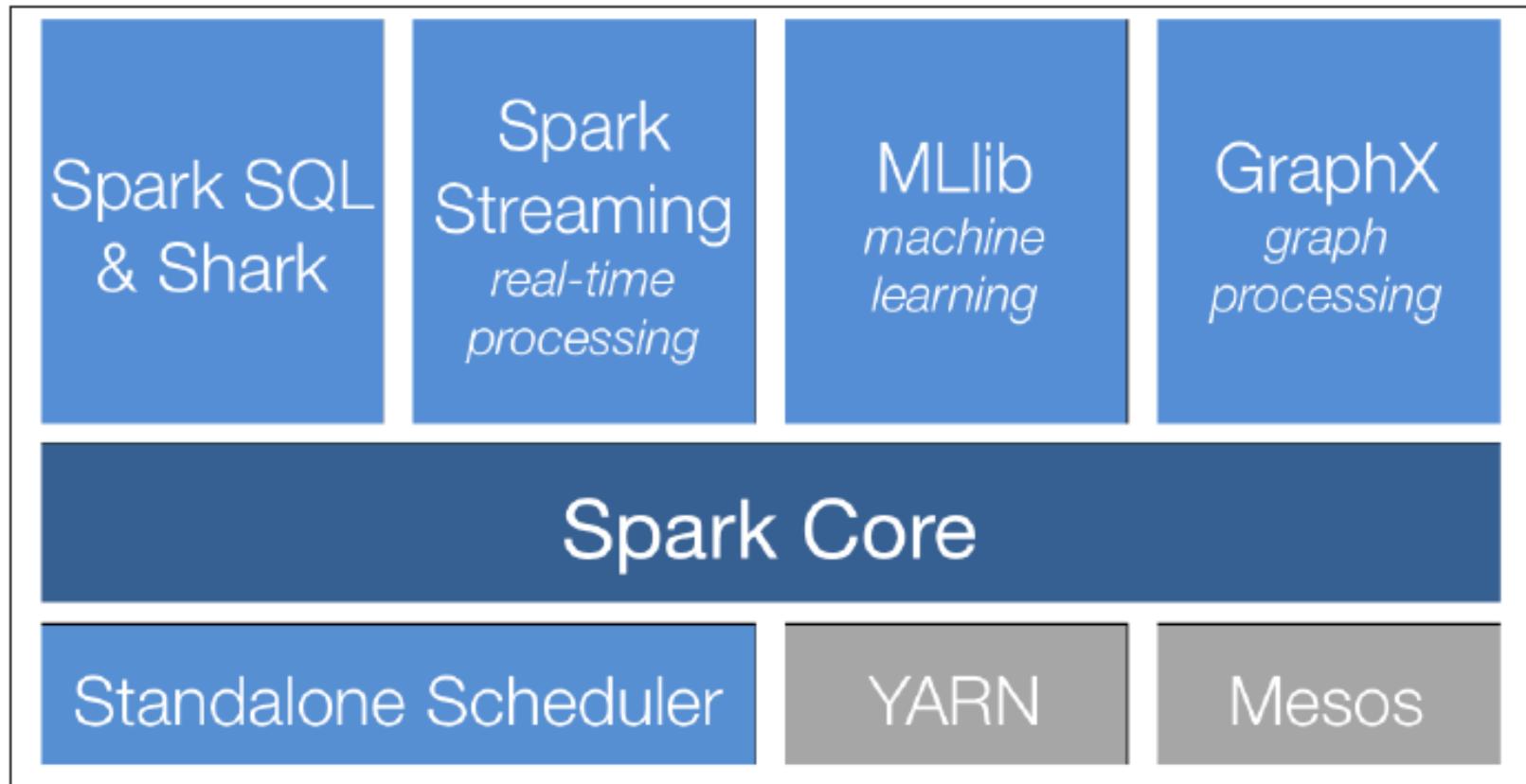
Tipos de workload para o Spark

- Suportar query interativa e processamento streaming.
 - Em memória, tolerante a falha, baixa latência.
 - Fácil de combinar batch, streaming e processamento interativo.
 - Modo fácil de desenvolver algoritmos sofisticados.



Fonte: Stoica (2013)

Spark Framework



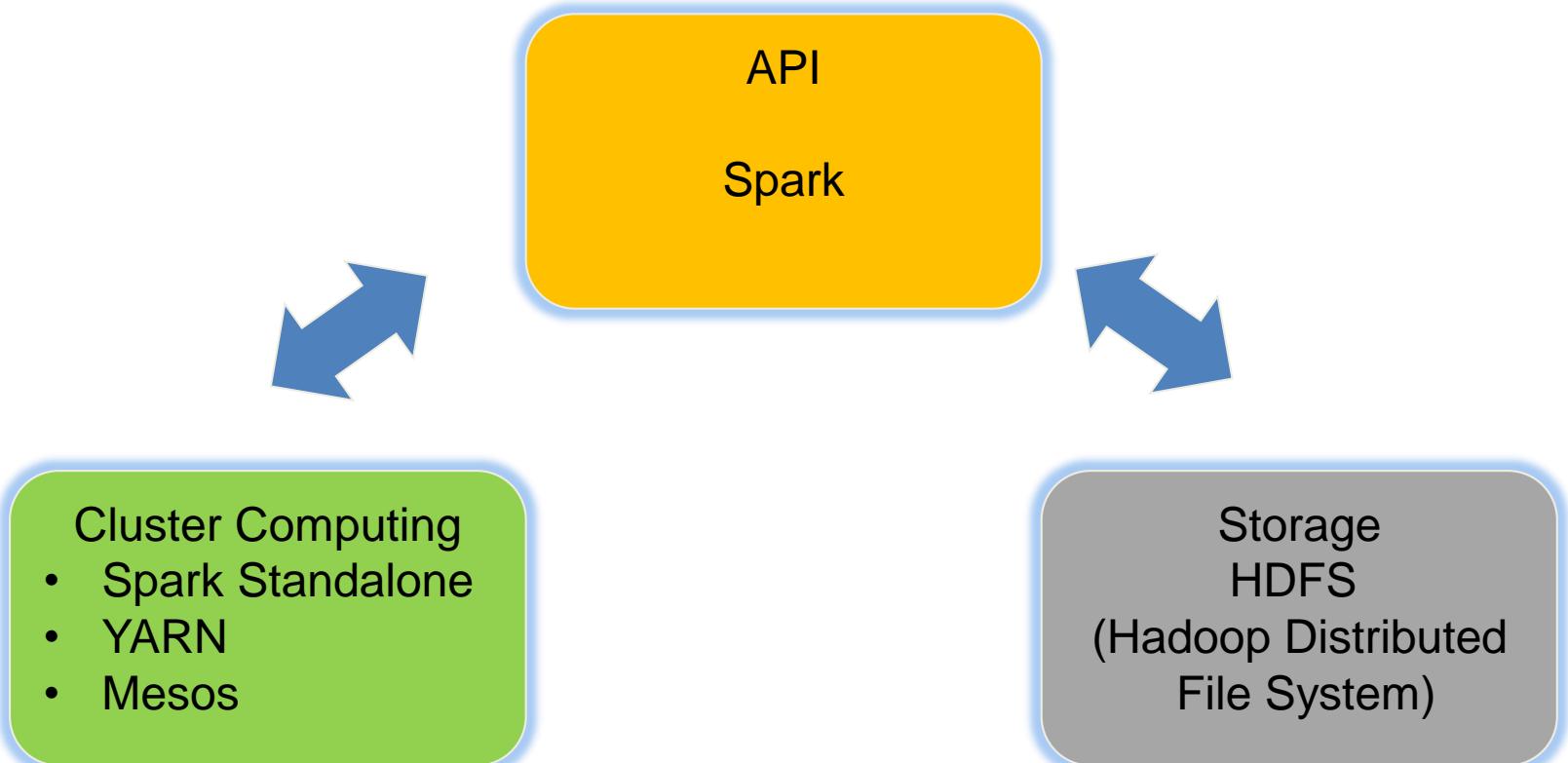
Fonte: ZAHARIA et al (2015)

Componentes do Spark

- **Spark Core:** Possui as funções básicas, incluindo os componentes para agendamento de tarefas (task scheduling), gerenciamento de memória, recuperação de falha, interação com sistemas de armazenamento (HDFS, Hbase, etc), dentre outros. É o home para API que define os RDDs (representa uma coleção de itens distribuídos entre os nodes).
- **Spark SQL:** É o componente que suporta a interação com o Spark por meio de linguagem SQL e HiveQL. Basicamente converte query SQL em operações no Spark. É uma evolução do projeto Shark iniciado pela UC Berkeley.
- **Spark Streaming:** É o componente que permite processar stream de dados em real time. Exemplos de stream de dados: log de web servers, mensagens em uma fila. Tem o mesmo nível de tolerância a falha e escalabilidade que o Spark Core.
- **MLib:** É uma biblioteca que contem as funções e algoritmos de Machine Learning, incluindo regressão, cluster, collaborative filtering, dentre outros.
- **GraphX:** É uma biblioteca que prove uma API para manipular gráficos em paralelo. Muito utilizado para análise de rede social (Social Network Analysis).
- **Cluster Managers:** O Spark pode utilizar diversos cluster manager, como YARN, Mesos ou Standalone Scheduler se você tiver apenas Spark no seu cluster.

Fonte: ZAHARIA et al (2015)

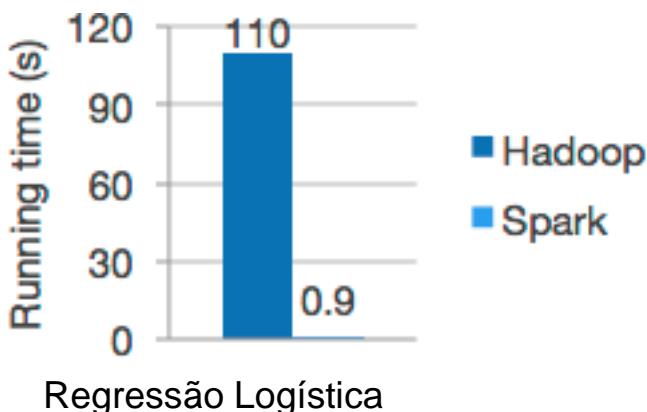
Processamento Distribuído com o Spark



Fonte: ZAHARIA et al (2015)

Spark x Hadoop MapReduce

- Spark leva o conceito do MapReduce ao próximo nível
 - API de alto nível: desenvolvimento mais rápido, mais fácil
 - Latência baixa: processamento próximo de tempo real
 - Armazenamento in-memory: até 100x de melhoria de performance



```
sc.textFile(file) \
  .flatMap(lambda s: s.split()) \
  .map(lambda w: (w,1)) \
  .reduceByKey(lambda v1,v2: v1+v2) \
  .saveAsTextFile(output)
```



```
public class WordCount {
  public static void main(String[] args) throws
    Job job = new Job();
    job.setJarByClass(WordCount.class);
    job.setJobName("Word Count");
    FileInputFormat.setInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    job.setMapperClass(WordMapper.class);
    job.setReducerClass(SumReducer.class);
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(IntWritable.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    boolean success = job.waitForCompletion(true);
    System.exit(success ? 0 : 1);
}

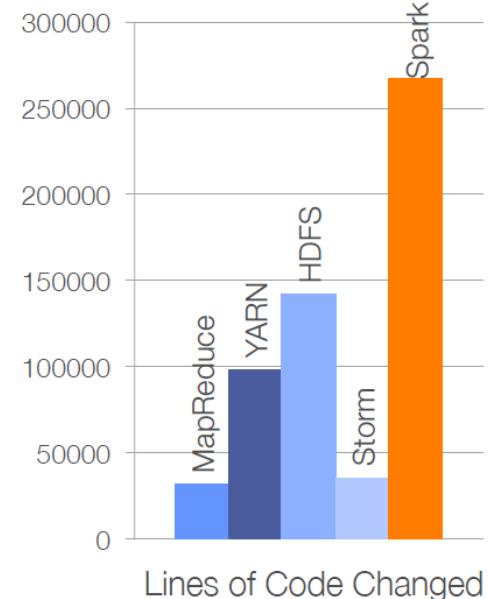
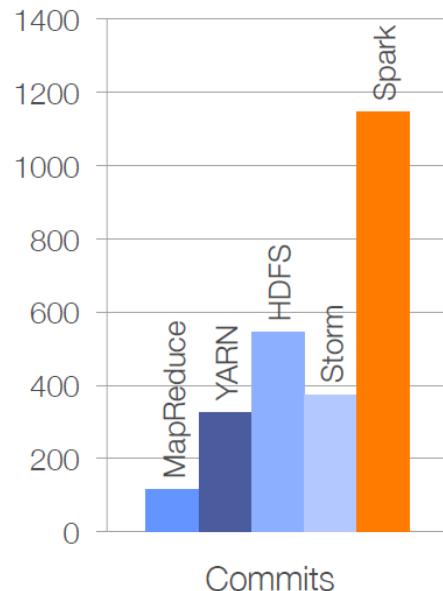
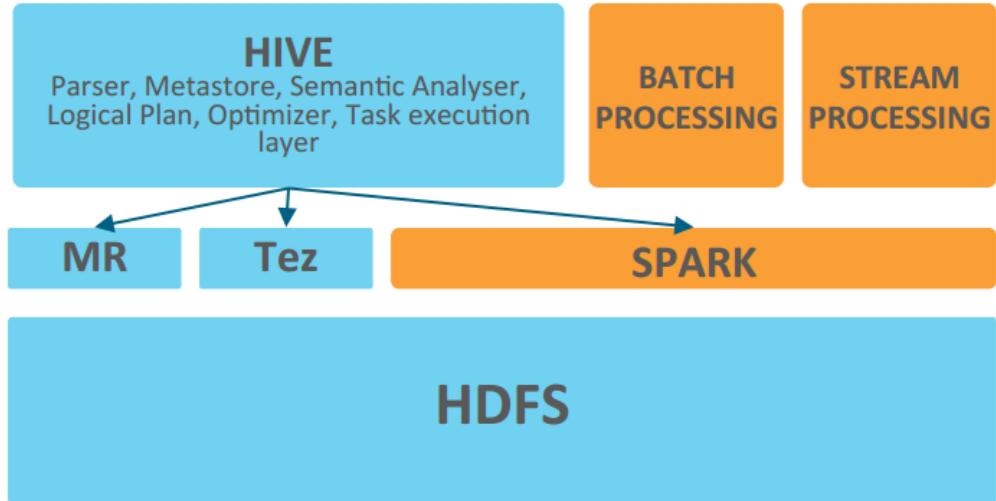
public class WordMapper extends Mapper<LongWritable, Text, Text,
IntWritable> {
  public void map(LongWritable key, Text value,
  Context context) throws IOException, InterruptedException {
    String line = value.toString();
    for (String word : line.split("\\W+")) {
      if (word.length() > 0)
        context.write(new Text(word), new IntWritable(1));
    }
  }
}

public class SumReducer extends Reducer<Text, IntWritable, Text,
IntWritable> {
  public void reduce(Text key, Iterable<IntWritable>
  values, Context context) throws IOException, InterruptedException {
    int wordCount = 0;
    for (IntWritable value : values) {
      wordCount += value.get();
    }
    context.write(key, new IntWritable(wordCount));
  }
}
```



Fonte: ZAHARIA et al (2015)

Spark e Hadoop – Integração e Apache



Fonte: ZAHARIA et al (2010) e Patrick (2014)

Comparação entre projetos Projeto na comunidade Apache - 2014

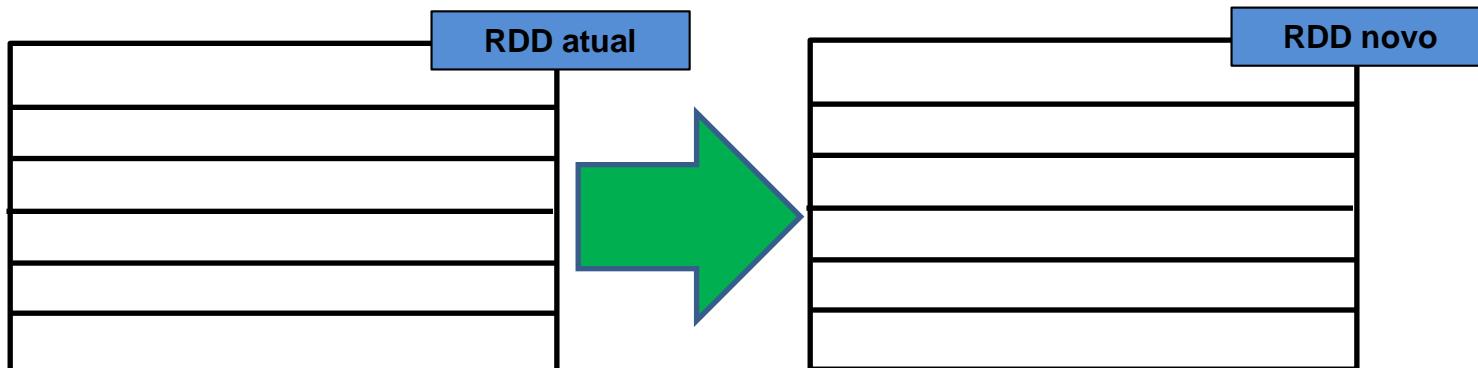
RDD (Resilient Distributed Dataset)

- **Resilient:** se os dados em memória são perdidos, eles podem ser recriados.
- **Distributed:** armazenado em memória através do cluster.
- **Dataset:** dados iniciais podem vir de um arquivo ou serem criados por meio de um programa.
- RDDs são a unidade fundamental do Spark. É uma coleção de objetos distribuídos. Eles são imutáveis.
- Existem três formas de criar um RDD:
 - De um arquivo ou conjunto de arquivos;
 - De dados na memória;
 - De outro RDD.

Fonte: ZAHARIA et al (2015)

Operações em RDD - Transformação

- Há dois tipos de operações RDD:
 - 1) **Transformação**: operações que retornam um novo RDD, utilizando Lazy Evaluation (Spark não irá executar até haver uma ação).
 - **Função map**: Cria um novo RDD realizando um determinada função para cada registro.
 - **Função filter**: Cria um novo RDD incluindo ou excluindo cada registro do RDD baseada em uma função boolean.
 - **Outras funções em um RDD**: distinct, sample.
 - **Outras funções em pares de RDD**: union, intersection, subtract, cartesian, combineByKey(), groupByKey(), join, etc.

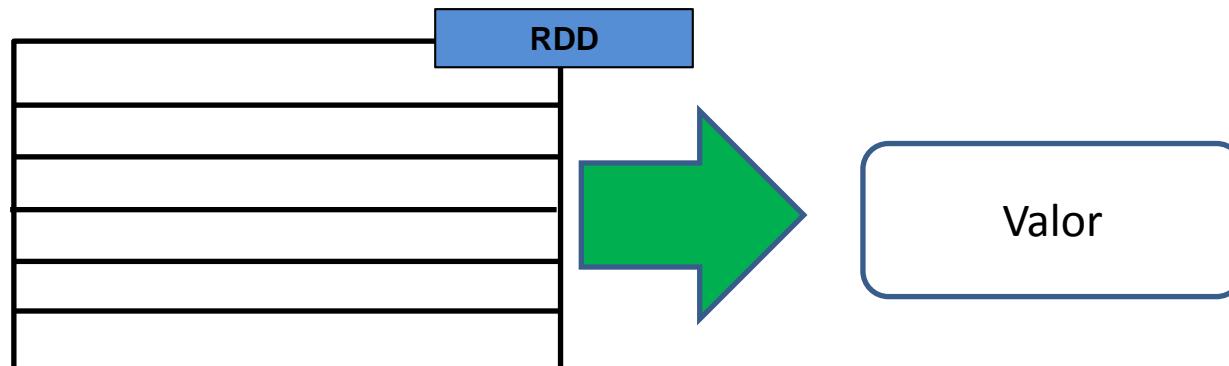


Fonte: ZAHARIA et al (2015)

Operações em RDD - Ação

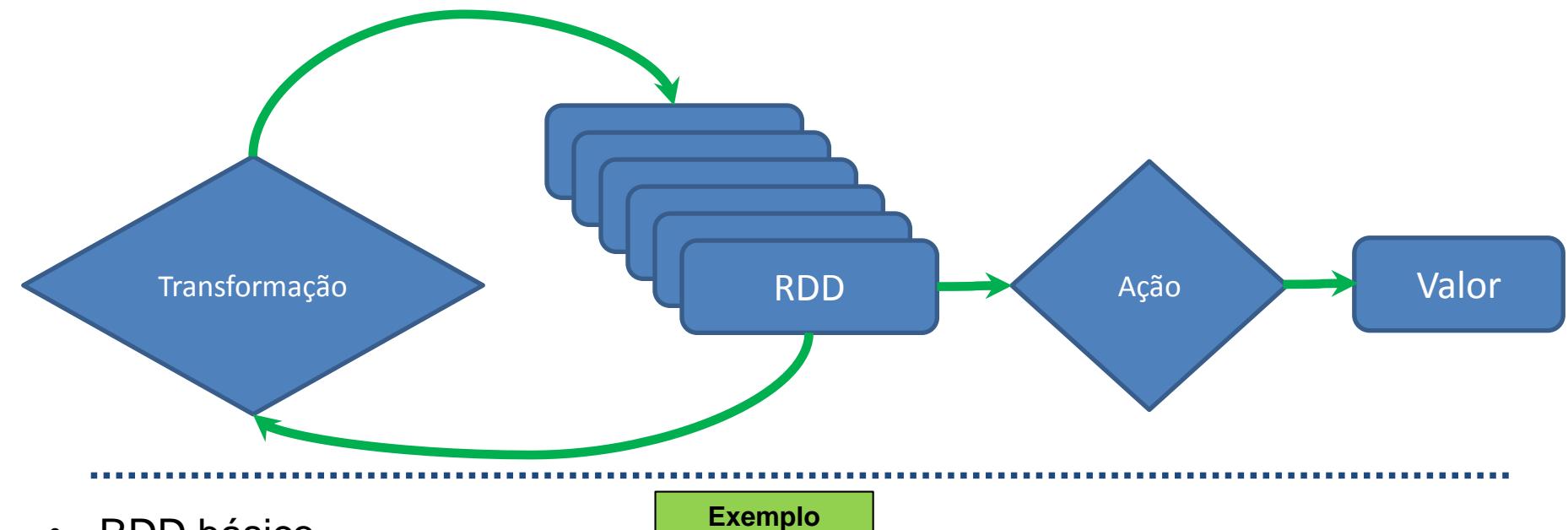
2) Ação: Uma ação que retorna um valor (resultado) para o driver ou escreve no storage. A ação mais comum utiliza reduce que realiza operações envolvendo dois elementos do mesmo tipo e retorna um novo elemento do mesmo tipo.

- **Collect ()**: Retorna todos os elementos de um RDD. Grava todo o dataset na memória. Não deve ser utilizado em datasets com tamanho muito grande que possa exceder a memória.
- **Count ()**: Retorna o número de elementos de um RDD.
- **Top(num)**: Retorna os top elementos.
- **Reduce(func)**: combina os elementos do RDD juntos em paralelo (ex: soma).



Fonte: ZAHARIA et al (2015)

Operações em RDD – Fluxo de processamento



- RDD básico
logs=
sc.textFile("file:/home/training/webserver.log")
- Transformação RDD
hora23=logs.filter(lambda x: "2014:23:00" in x)
- Ação RDD
hora23.take(2)

Exemplo

Fonte: ZAHARIA et al (2015)

Níveis Persistência (Persistence) no RDD

Comando para salvar o dado na memória apenas: `meurdd.cache()`. Para as operações de persistência, execute, exemplo:

```
from pyspark import StorageLevel
persist(StorageLevel.DISK_ONLY)
```

Level	Space Used	CPU time	In memory	On Disk	Nodes with data	Comments
MEMORY_ONLY	High	Low	Y	N	1	
MEMORY_ONLY_2	High	Low	Y	N	2	
MEMORY_ONLY_SER	Low	High	Y	N	1	
MEMORY_ONLY_SER_2	Low	High	Y	N	2	
MEMORY_AND_DISK	High	Medium	Some	Some	1	Spills to disk if there is too much data to fit in memory.
MEMORY_AND_DISK_2	High	Medium	Some	Some	2	Spills to disk if there is too much data to fit in memory.
MEMORY_AND_DISK_SER	Low	High	Some	Some	1	Spills to disk if there is too much data to fit in memory.
MEMORY_AND_DISK_SER_2	Low	High	Some	Some	2	Spills to disk if there is too much data to fit in memory.
DISK_ONLY	Low	High	N	Y	1	
DISK_ONLY_2	Low	High	N	Y	2	

Fonte: ZAHARIA et al (2015) e ZAHARIA et al (2010)

Spark Shell

- Spark Shell é um shell REPL (Read/Evaluate/Print Loop).
 - Tem a versão em Scala ou Python
 - Scala: Digite spark-shell na VM
 - Python: pyspark

Fonte: ZAHARIA et al (2015)

Spark Context e exemplos de códigos

- Toda aplicação Spark requer um Spark Context
 - Ponto de entrada da Spark API.
- O Spark Shell instancia um contexto e o habilita com o nome sc (por convenção).
 - Digite “sc.appName” no shell para ver o nome da sua aplicação
- Exemplo de código utilizando Python:
 - `>>> linhas= sc.textFile("README.md") # cria o RDD chamado linhas.`
 - `>>> linhas.count() # conta o número de linhasno RDD.`
 - `127`
 - `>>> linhas.first() # exibe a primeira linha do RDD`
 - `u'# Apache Spark'`
- Exemplo de código utilizando Scala:
 - `scala> val linhas = sc.textFile("README.md")`
 - `linhas: spark.RDD[String] = MappedRDD[...]`
 - `scala> linhas.count()`
 - `res0: Long = 127`
 - `scala> linhas.first()`
 - `res1: String = # Apache Spark`

Fonte: ZAHARIA et al (2015)

Exemplo de criação de RDDs a partir de um arquivo.

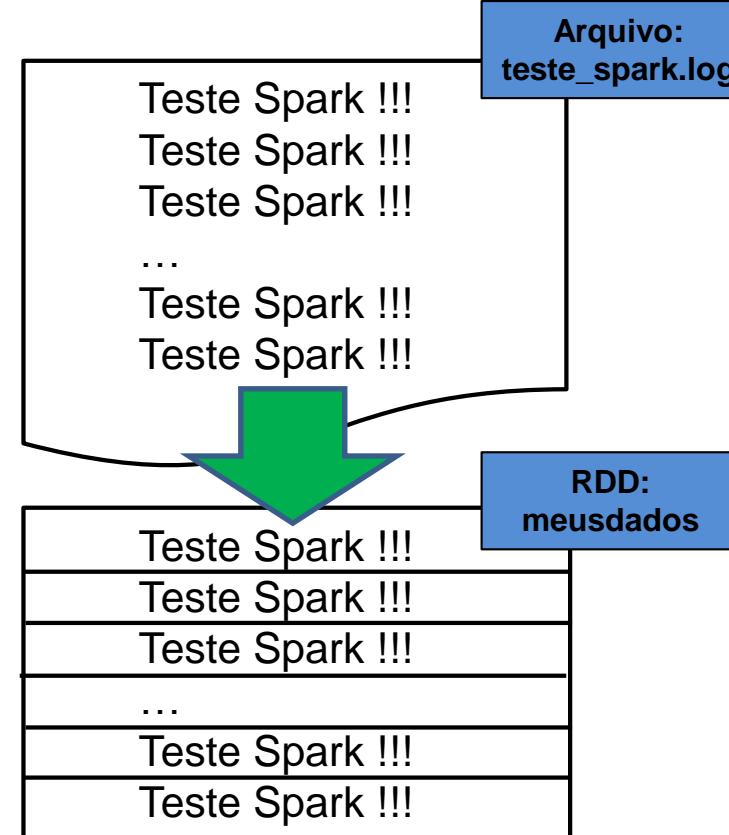
- Utilizando o `SparkContext.textFile` será atribuído a um RDD (com o nome `meusdados`) o arquivo (`/home/training/teste_spark.log`). Cada linha do arquivo será um registro separado no RDD:

```
In [5]: meusdados = sc.textFile("file:/home/training/teste_spark.log")
15/09/14 07:31:45 INFO MemoryStore: ensureFreeSpace(178235) called with
curMem=712940, maxMem=309225062
15/09/14 07:31:45 INFO MemoryStore: Block broadcast_4 stored as values
to memory (estimated size 174.1 KB, free 294.1 MB)
```

pyspark

```
In [6]: meusdados.count()
15/09/14 07:37:36 INFO FileInputFormat: Total input paths to process :
1
15/09/14 07:37:36 INFO SparkContext: Starting job: count at <ipython-in
put-6-c6cd42a4366d>:1
...
15/09/14 07:37:39 INFO SparkContext: Job finished: count at <ipython-in
put-6-c6cd42a4366d>:1, took 2.279943771 s
Out[6]: 30
```

pyspark



```
Teste Spark !!!
Teste Spark !!!
Teste Spark !!!
...
Teste Spark !!!
Teste Spark !!!
```

Arquivo:
teste_spark.log

```
Teste Spark !!!
Teste Spark !!!
Teste Spark !!!
...
Teste Spark !!!
Teste Spark !!!
```

RDD:
meusdados

Exercício 1 e 2 - Spark

Abra o arquivo de exercícios de Spark.

Cases com Spark

Casos de Uso do Spark

- ETL
- Text Mining
- Index Building
- Criação de Grafo e Análise
- Reconhecimento de padrões
- Filtro colaborativo
- Modelos preditivos
- Análise de Sentimento
- Assessment de Risco
- Real-Time Analytics

Fonte: ZAHARIA et al (2015)

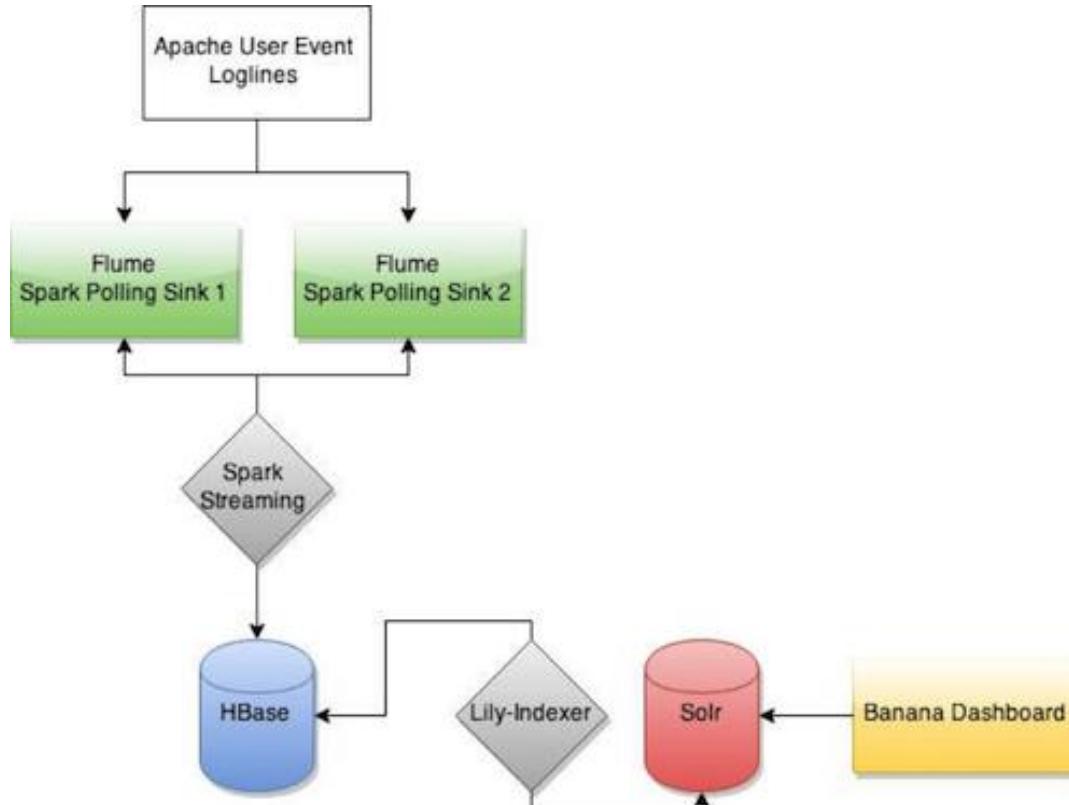
Case 1 – – Contexto

- Fundada em 1995.
- Site online com informações sobre automóveis.
- No seu site há preço para veículos usados e novos, lista de inventários, revisões de testes drive e dicas e avaliações de diversos aspectos para quem comprar ou possui um carro.
- Possui uma calculadora que estima o verdadeiro valor do mercado, informando quanto, na média, os compradores estão pagando pelos veículos novos.
- Eles querem melhorar a monitoração de tráfego de acesso ao site.
- Atualmente, a atualização da monitoração é realizada de hora em hora com um atraso de uma hora. Os dados não são apresentados em um dashboard e é necessário realizar um processo manual para visualizar os dados.
- Muitas empresas de carros realizam anúncios durante a transmissão de TV de jogos do Super Bowl. A quantidade de visualizações do site chega ao pico de **5 mil visualizações por minuto**.
- A empresa quer monitorar as seguintes métricas:
 - Contagem de Page View e Contagem de Visitantes Únicos por modelo de carro (ex: Fusion) ou pela marca do fabricante (ex: Ford).

Fonte: Kestelyn (2014)

Case 1 – – Aplicação

- Utiliza Spark Streaming para construir DashBoard em Near Real-Time.
- As métricas calculadas devem ser acumulativas, gerando relatórios por minuto, a cada 5 minutos, a cada hora e posteriormente um relatório, das 24 horas, ao final do dia.



Fonte: Kestelyn (2014)

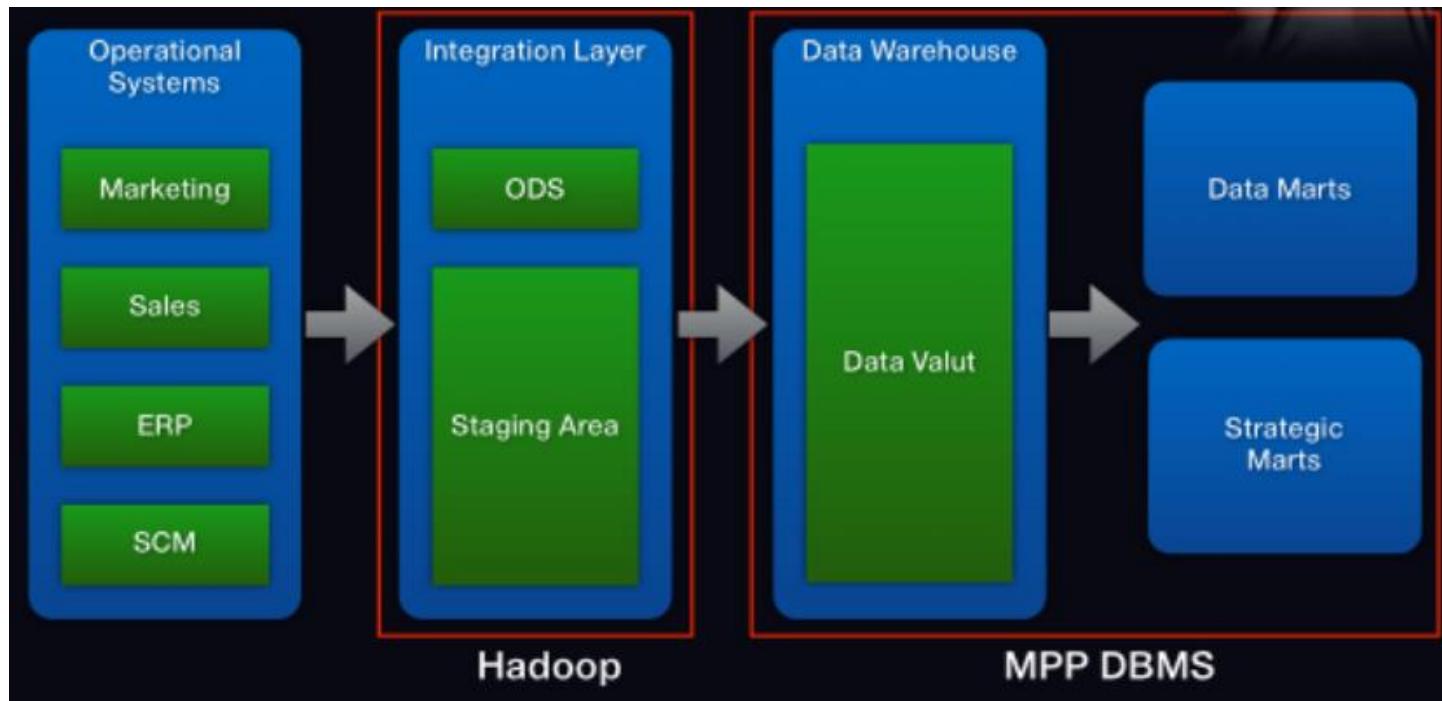
Case 2 – – Contexto

- Foi fundada em 1984.
- É um operadora de telecomunicações na Coreia de Sul.
- Demanda: 40 TBs/dia e 15 PTs armazenados até final de 2014.
- Possui mais de 10 clusters, o maior possui 500 nós e um total de 900 nós no total.
- Utiliza vários MPP Databases comerciais.
- Começaram a iniciativa de substituir o MPP Database por Spark e Hadoop.

Fonte: Lee (2014)

Case 2 – – Abordagem antiga

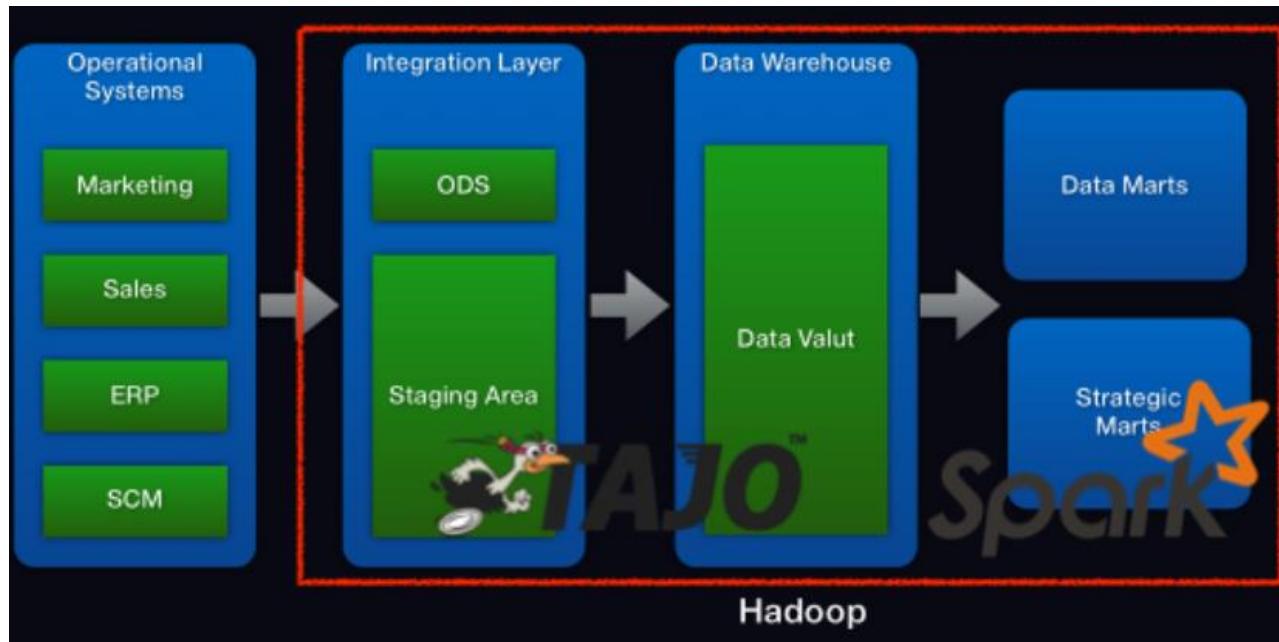
- Abordagem antiga tinha que carregar muitos dados no MPP Database.



Fonte: Lee (2014)

Case 2 – SK telecom – Nova Abordagem

- Nova abordagem suporta um ETL de alta velocidade.
- Real time queries para os web clientes.
- Requisitos do sistemas:
 - Baixa latência para ad-hoc query (< 2sec)
 - Suporte SQL ANSI (sem necessidade de Insert/Update/Delete)
 - Suporte de usuários concorrentes (10 usuários por segundo)
 - Alta disponibilidade.
- Substituindo o DW pelo Shark no Spark (SparkSQL).



Fonte: Lee (2014)

Spark em Cluster

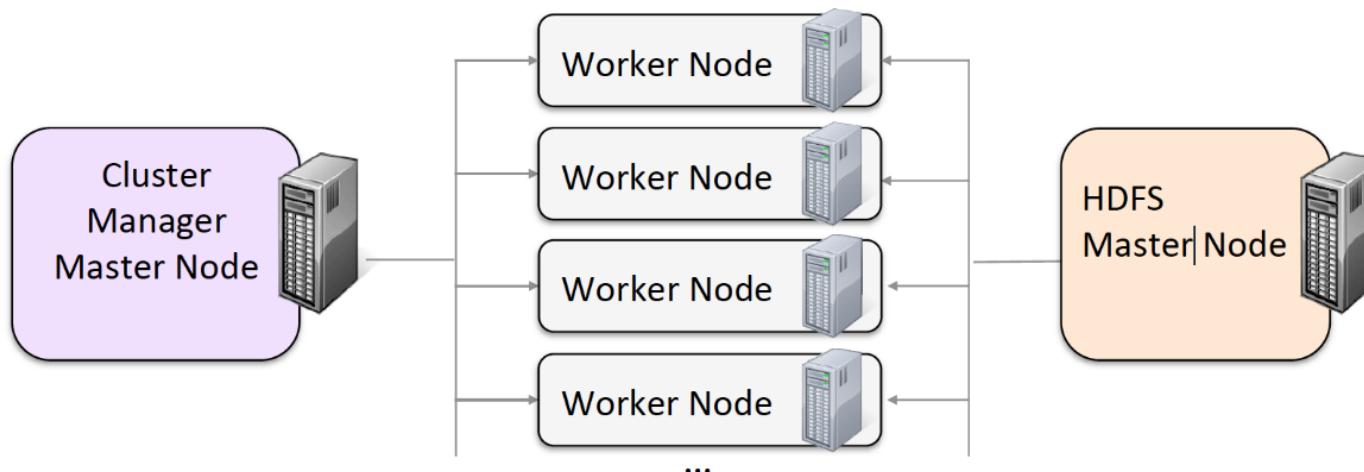
Rodando Spark em um Cluster

- O Spark pode rodar:
 - Localmente
 - Sem processamento distribuído
 - Localmente com várias threads worker
 - Num cluster
 - Spark Standalone
 - YARN
 - Mesos

Fonte: ZAHARIA et al (2015) e (2010)

Rodando Spark em um Cluster

- **Node é um computador do cluster**
 - Nodes Master gerenciam a distribuição de trabalho para os worker nodes.
- **Daemon é um programa rodando em um node**
 - Cada programa desempenha uma função diferente no cluster.



Fonte: ZAHARIA et al (2015) e (2010)

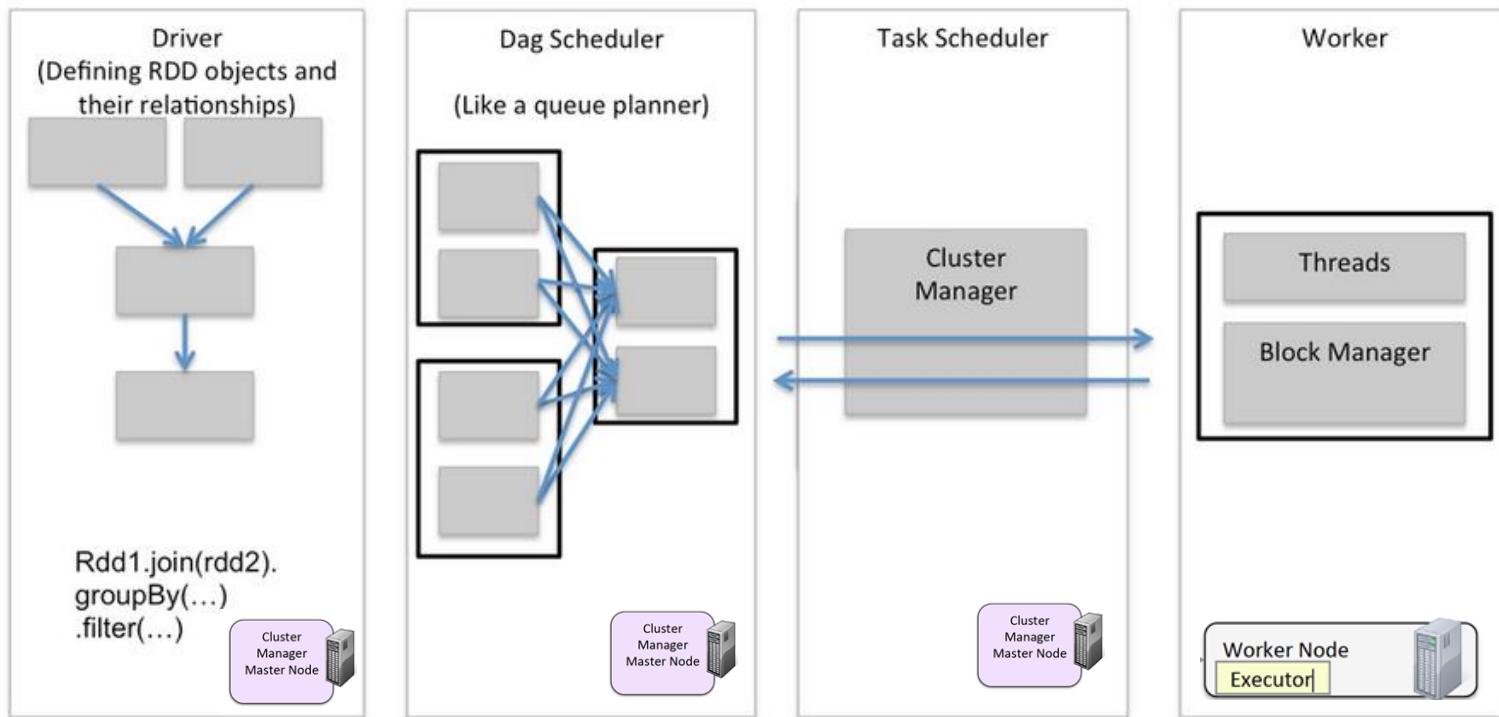
Por que rodar em um cluster?

- Rodar num cluster permite que você tenha as vantagens do processamento distribuído
 - Habilidade de processar largos volumes de dados
 - Tolerância a erro e escalabilidade
- O modo local é útil para desenvolver e testar
- Em produção, o Spark sempre roda em clusters

Fonte: ZAHARIA et al (2015) e (2010)

O Spark Driver

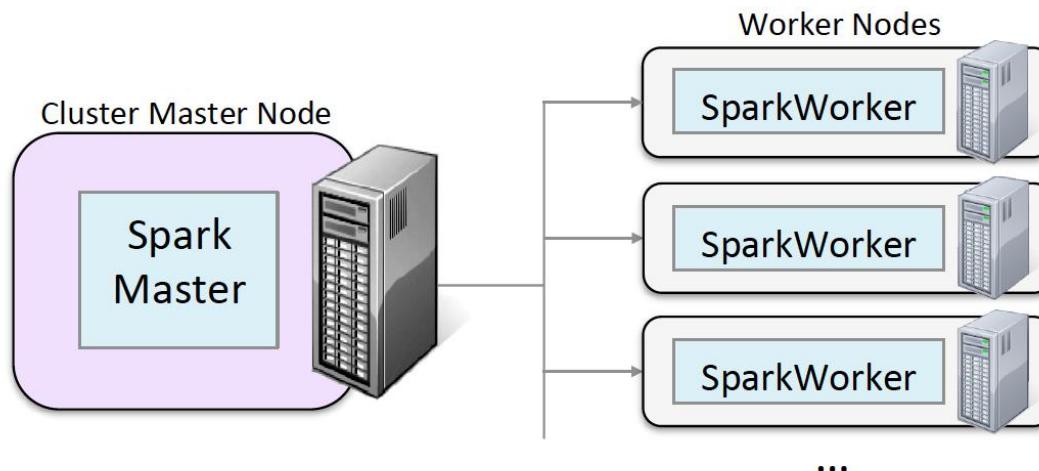
- **Spark Driver**
 - É o programa principal
 - Pode ser o Spark Shell ou uma aplicação.
 - Cria um Spark Context configurado para o cluster.
 - Comunica-se com o Cluster Manager para distribuir tarefas aos executores.



Fonte: ZAHARIA et al (2015) e (2010); Shapira (2015)

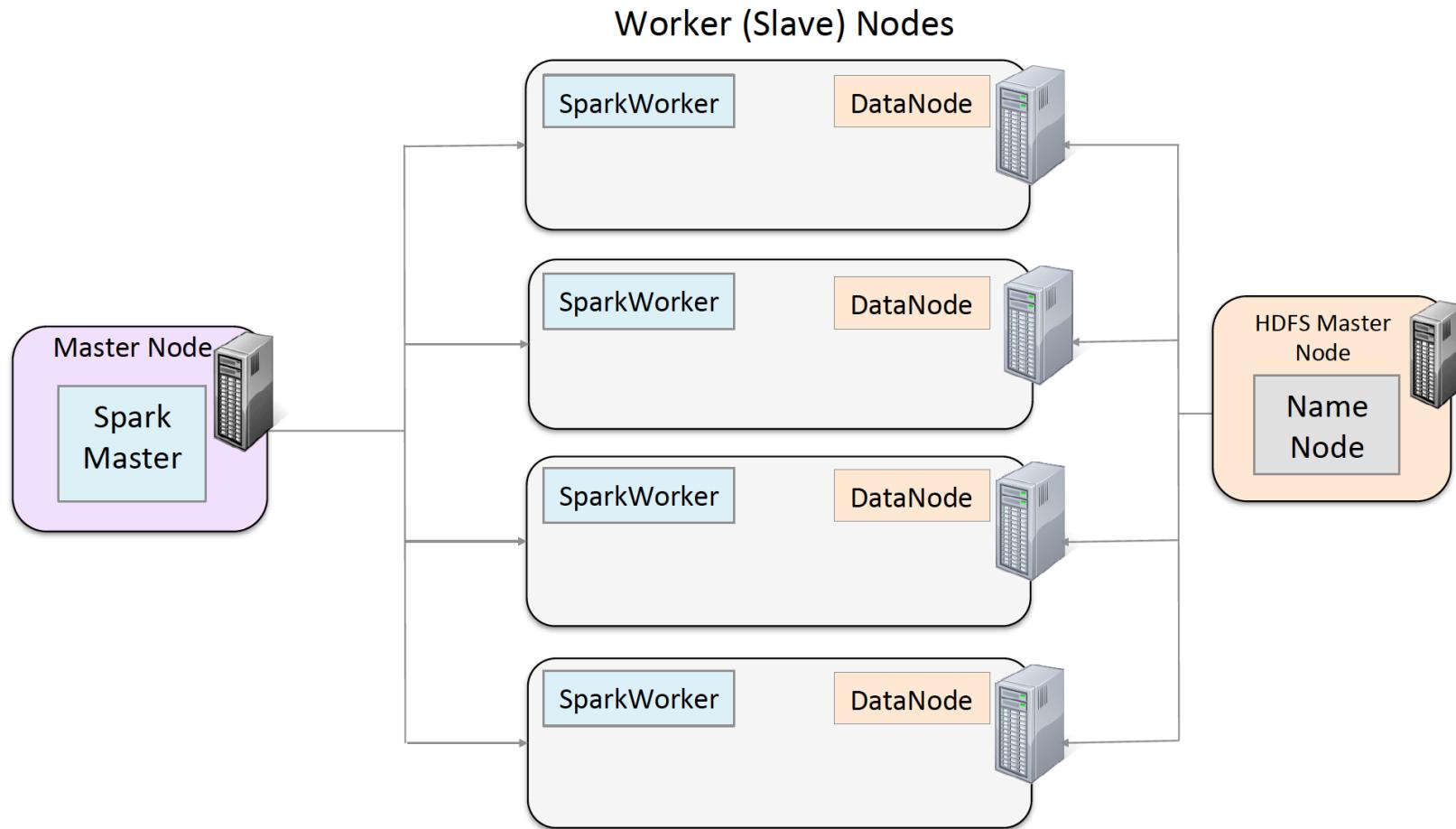
Spark Standalone Daemons

- Os daemons são:
 - Spark master
 - Um por cluster.
 - Gerencia aplicações, distribui tarefas individuais para os Spark Workers.
 - Spark worker
 - Um por worker node.
 - Começa e monitora os executores para aplicações.

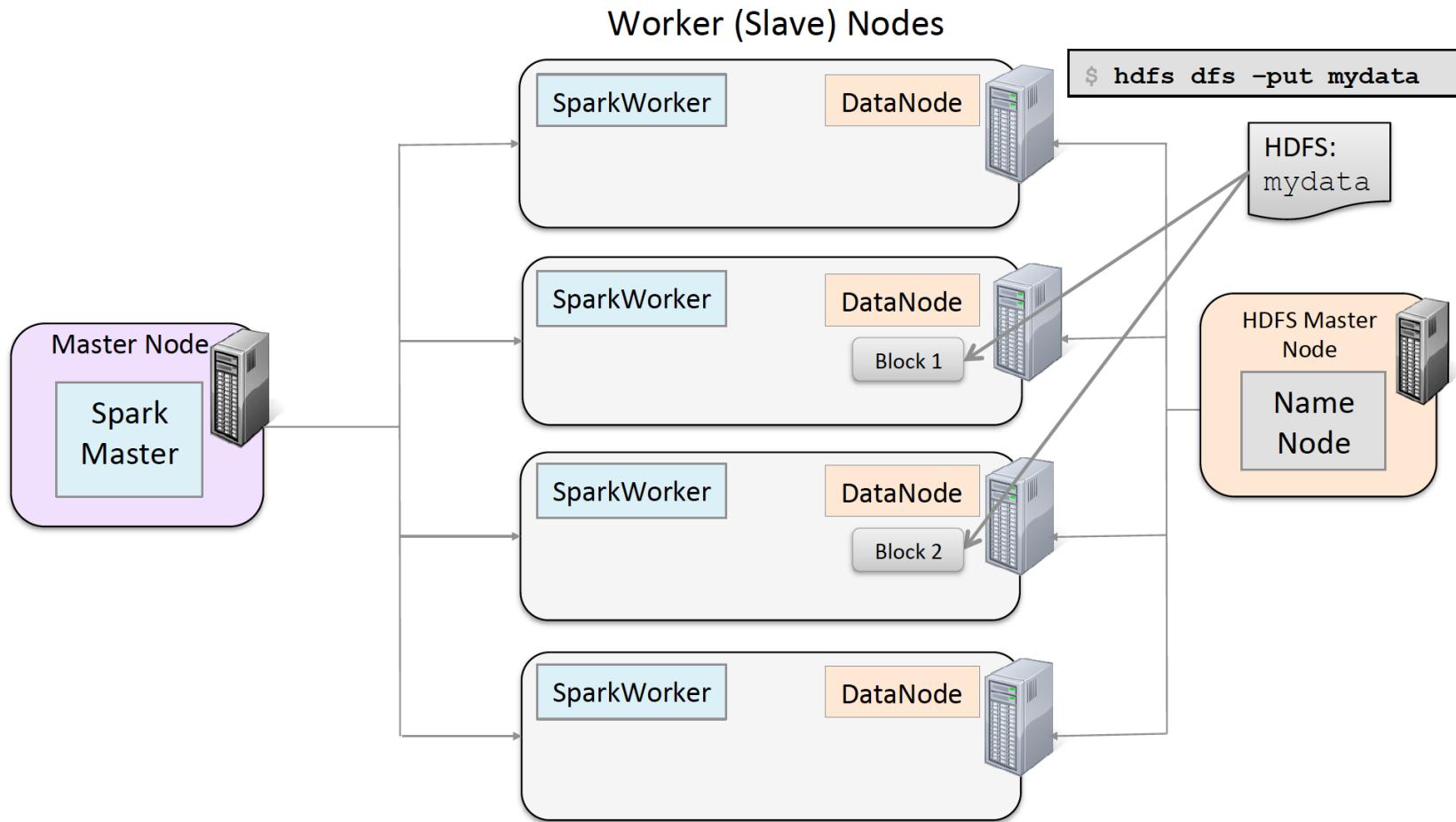


Fonte: ZAHARIA et al (2015) e (2010)

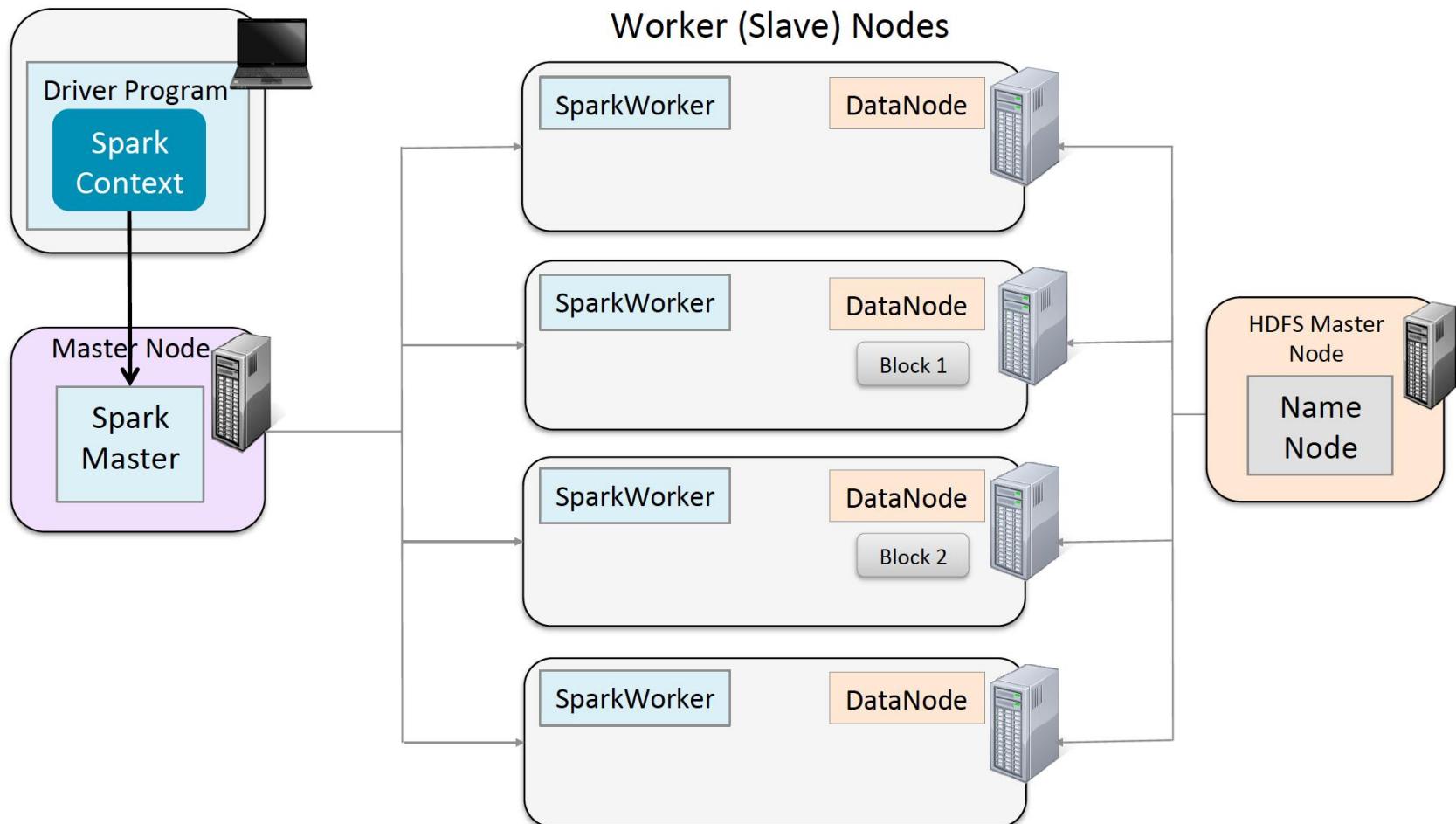
Rodando Spark num Cluster Standalone (1)



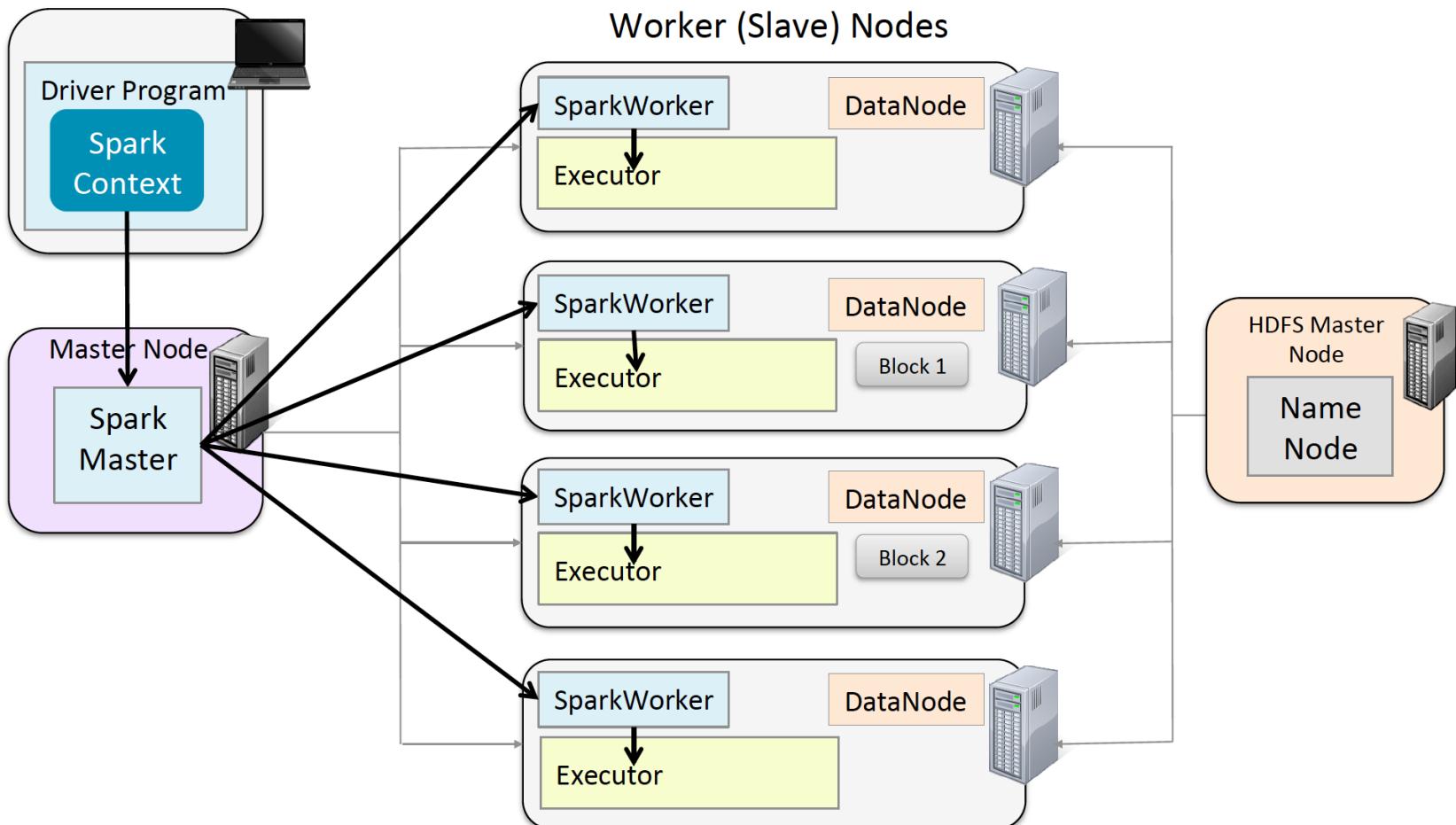
Rodando Spark num Cluster Standalone (2)



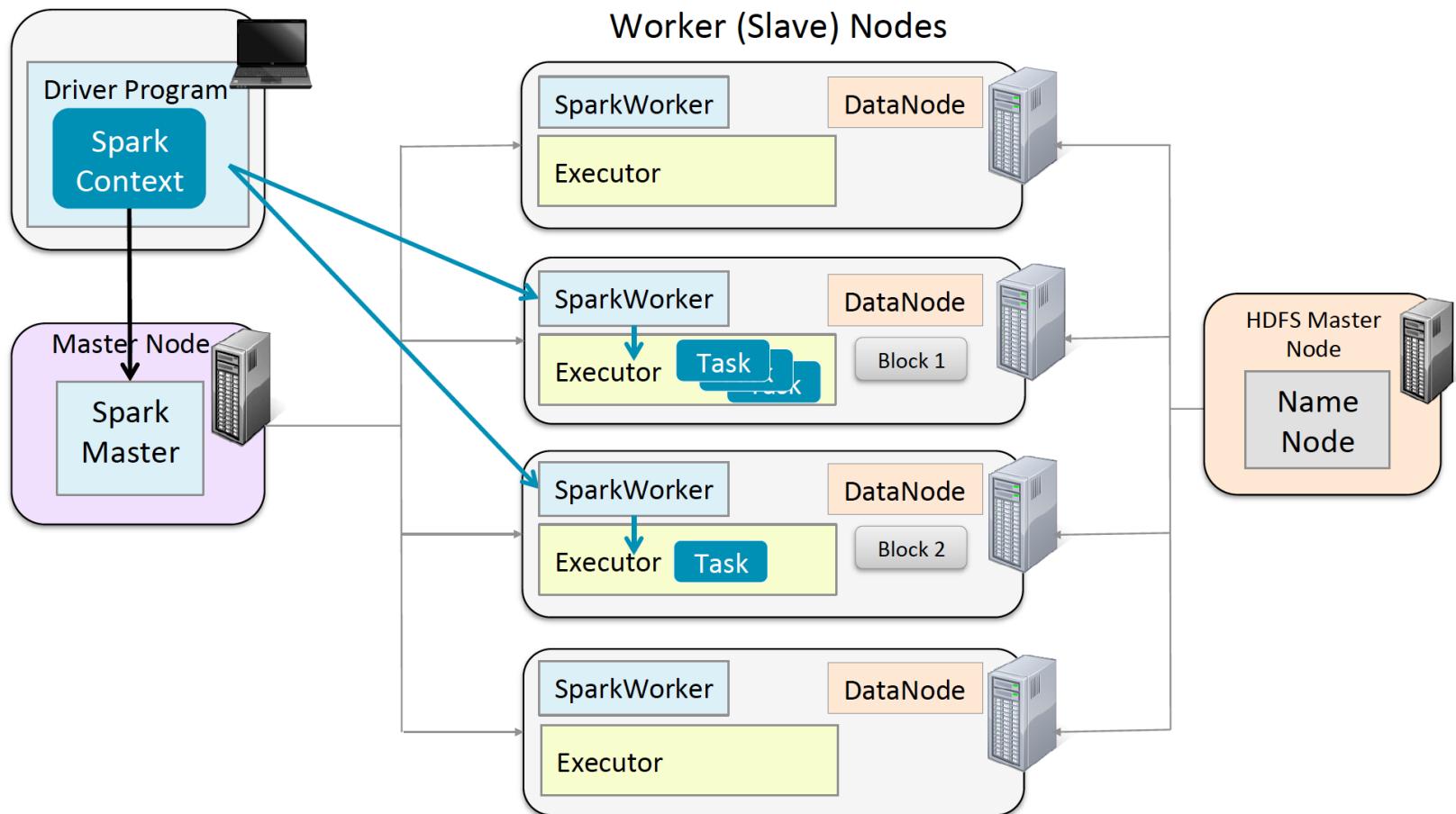
Rodando Spark num Cluster Standalone (3)



Rodando Spark num Cluster Standalone (4)



Rodando Spark num Cluster Standalone (5)



Cluster Resource Managers Suportados

- **Spark Standalone**
 - Incluso no Spark.
 - Fácil de instalar e rodar.
 - Configuração e escalabilidade limitadas.
 - Útil para testar em ambientes de desenvolvimento, ou sistemas pequenos.
- **Hadoop Yarn**
 - Mais comum em ambientes de produção.
 - Permite compartilhar recursos com outras aplicações (MapReduce, Impala etc).
- **Apache Mesos**
 - Primeira engine suportada pelo Spark.
 - Agora cada vez menos utilizada em detrimento do YARN.

Fonte: ZAHARIA et al (2015) e (2010)

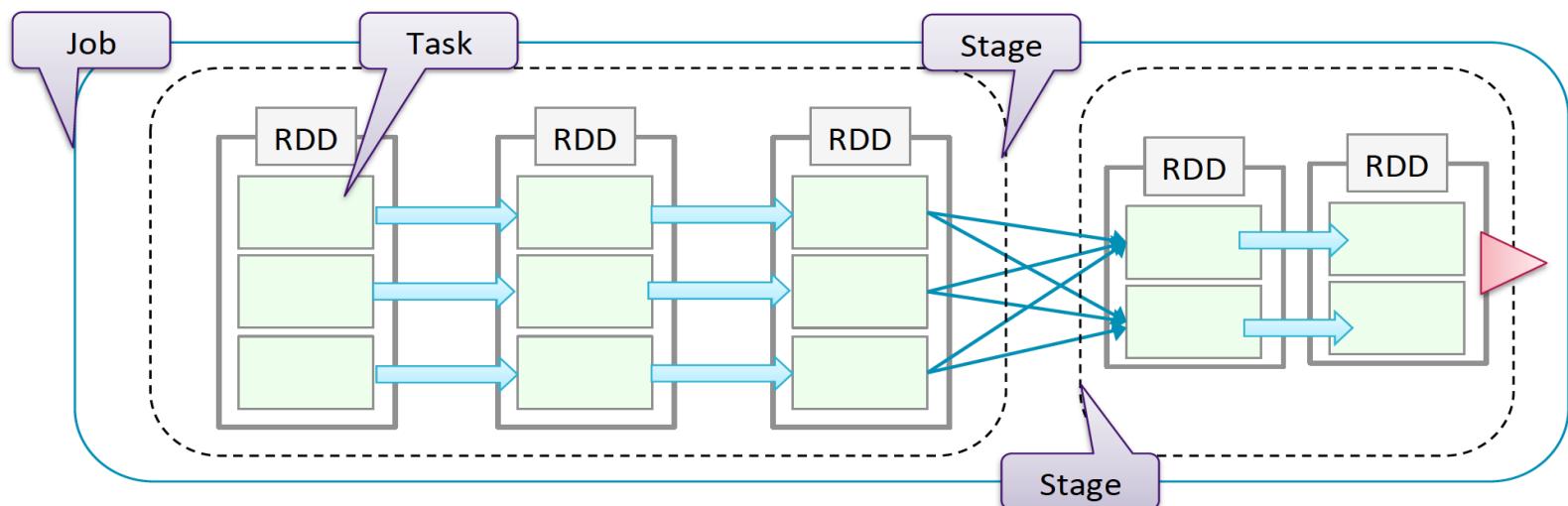
Exercício 3 - Spark

Abra o arquivo de exercícios de Spark.

Spark Jobs, Estágios e Tarefas

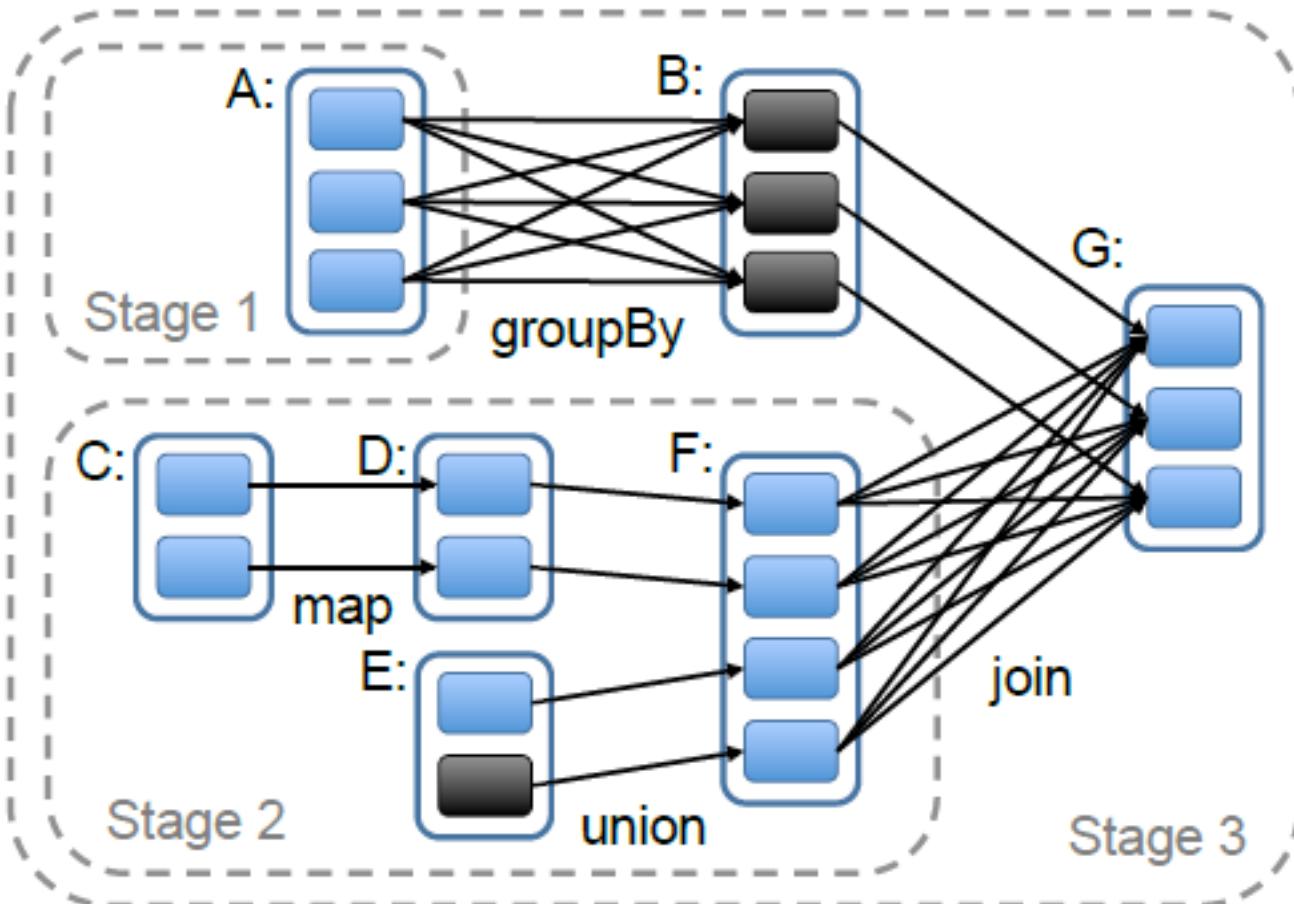
Terminologia: Applications, Jobs, Stages e Tasks

- Um aplicativo Spark que você roda é uma *Application* (*aplicação*):
 - Cada application pode ter que rodar n Jobs.
 - **Job**: será iniciado a partir de uma action (ação). Cada Job pode ter n *Stages*.
 - **Stage**: são conjuntos de *Tasks* que podem ser rodadas em paralelo. Cada **Stage** pode ter n Tasks.
 - Entre Stages sempre acontece a fase *Shuffle*.
 - **Task**: é uma unidade individual de trabalho enviada a um executor.



Fonte: ZAHARIA et al (2015) e (2010)

Applications, Jobs, Stages e Tasks



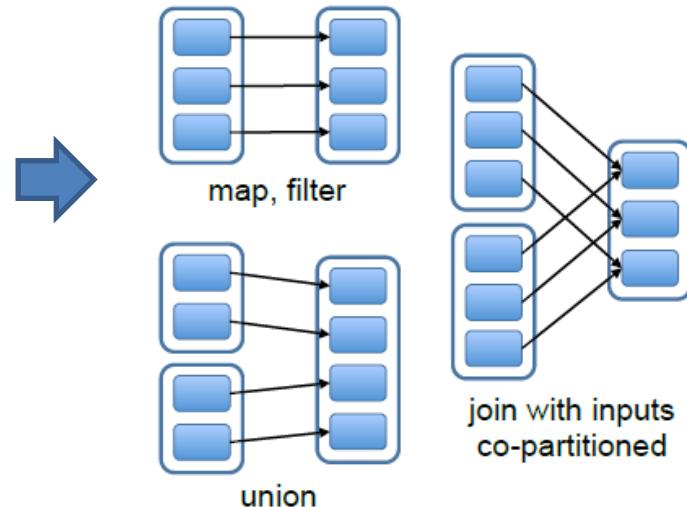
Fonte: ZAHARIA et al (2015) e (2010)

Applications, Jobs, Stages e Tasks

- Spark constrói uma **DAG** (Grafo Acíclico Direcionado) das dependências dos RDDs

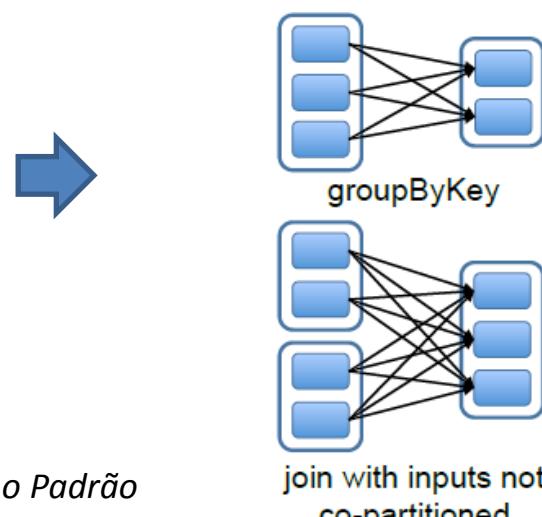
- **Operações Narrow**

- Apenas um child depende do RDD.
 - Não é necessário shuffle entre os nodes.
 - Pode ser realizada em um único estágio.
 - Ex: map, filter, union.



- **Operações Wide**

- Múltiplos filhos dependem do RDD.
 - Define novos estágios.
 - Ex: reduceByKey, join, groupByKey.
Veja estágios em <http://localhost:4040>



- **Você controla o número de partições do Spark!**

```
data = [("a", 3), ("b", 4), ("a", 1)]
```

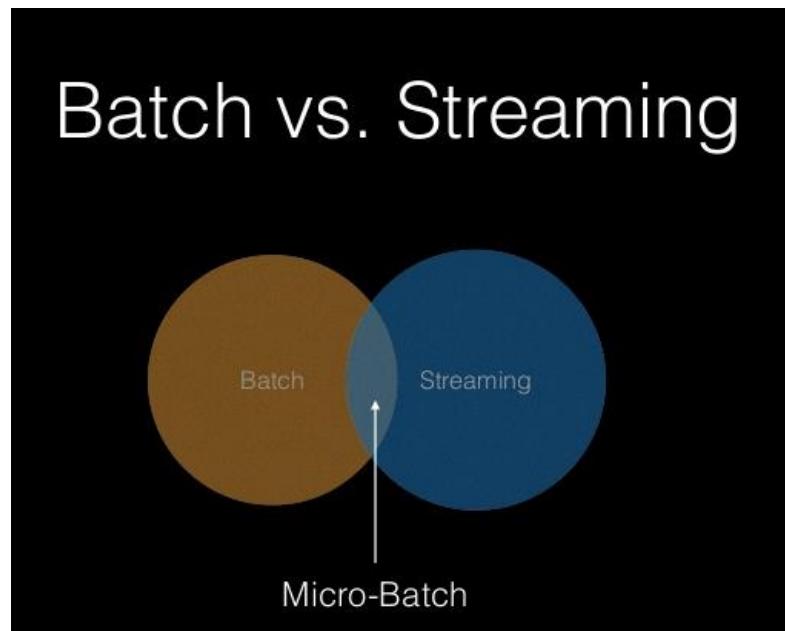
```
sc.parallelize(data).reduceByKey(lambda x, y: x + y) # Paralelismo Padrão
```

```
sc.parallelize(data).reduceByKey(lambda x, y: x + y, 10) # Paralelismo controlado
```

Spark Streaming

MICRO-BATCHING VERSUS STREAMING

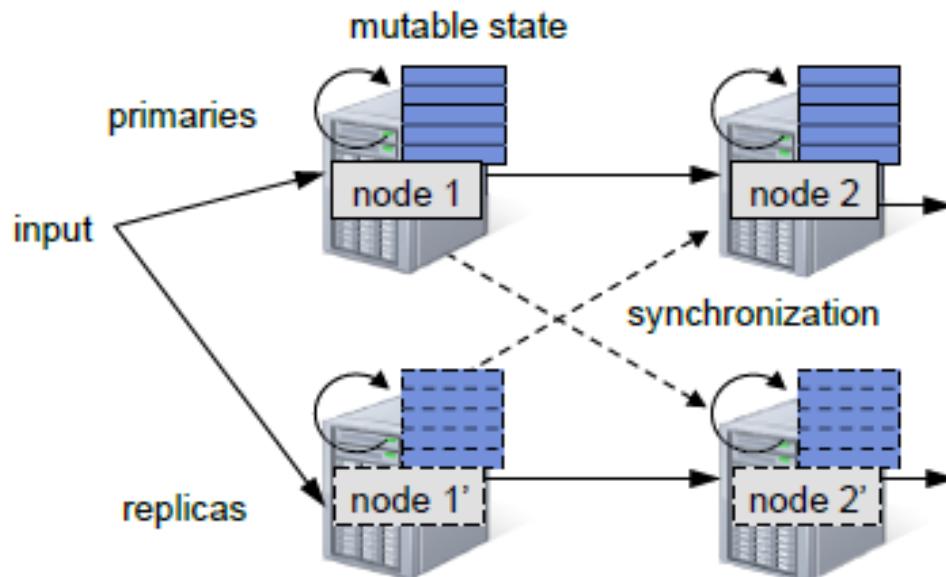
- Streaming é um processamento real time (<500 ms) que permite que as informações sejam processadas mais de uma vez. Exemplos de ferramentas: Storm e Flume.
- Micro-batching é a ferramenta que prove o processamento de eventos em uma transação batch, permitindo confiabilidade e processamento de informações uma única vez. Exemplos de ferramentas: Spark Streaming e Trident.



Shapira et al (2015)

Abordagem tradicional no sistemas de Streaming

- Um registro por vez, cada servidor possui um estado mutável.
- Para cada novo registro, atualiza o estado e envia um novo registro.
- Tolerância a falha é atingida por meio de replicação, utilizando protocolo de sincronização.



ZAHARIA et al (2015) e Shapira et al (2015)

O que é Spark Streaming?

- **Definição:** Divide um stream de dados em um processamento batch em segundos. O processamento de cada batch se torna um RDD.
- **Objetivo:** Objetivo prover processamento streaming com um desenvolvimento simples, escalável, consistente e tolerante a falhas.
- **Motivação:** Muitas aplicações precisam de processar dados em escala com baixa latência (poucos segundos): Estatística de site, detecção de intruso, filtro de spam. Ferramenta tolerante a falha e que trabalhe com eficiência.

ZAHARIA *et al* (2015) e Shapira *et al* (2015)

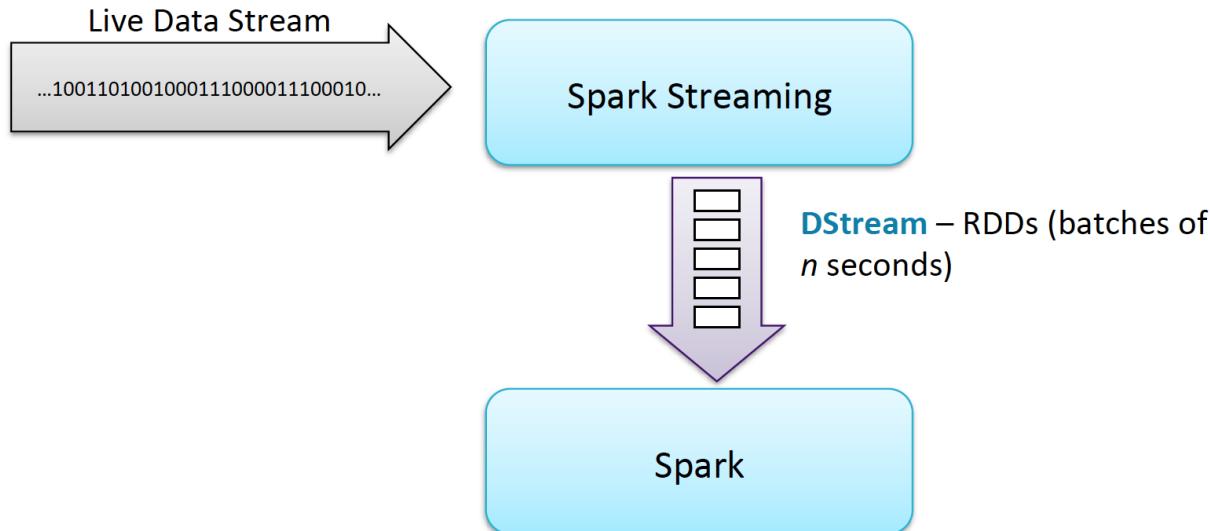
Spark Streaming

- **Spark Streaming provê analytics real-time dos dados**
- **É uma extensão do Spark Core**
- **Suporta Scala ou Java**
 - Versões mais recentes também suportam Python
- **Usado em aplicações que precisam processar grandes volumes de dados em real-time**
 - Monitoramento de websites
 - Detecção de Fraude
 - Monetização de Anúncios
 - Telemetria de veículos
 - Sensores
 - Internet das Coisas
 - “Infinite Data”



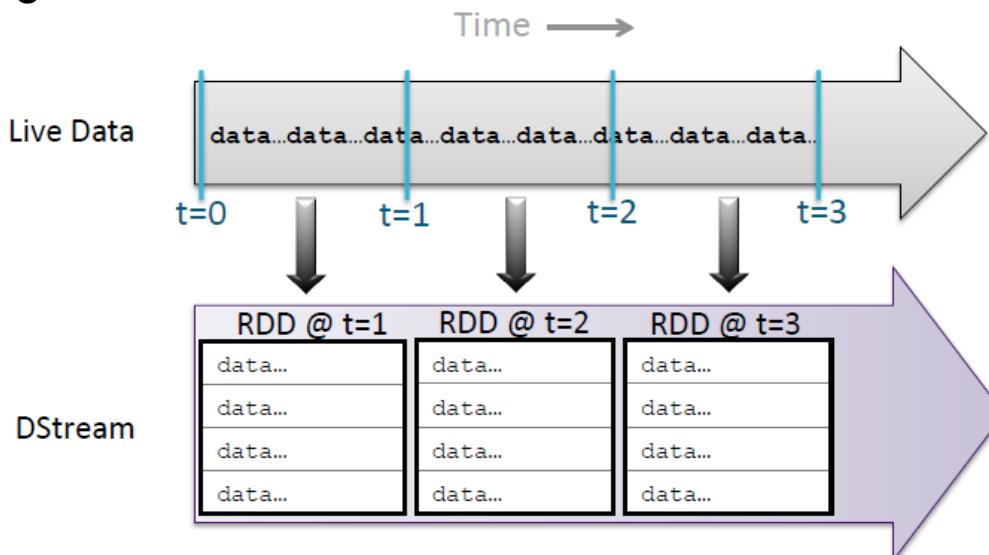
Visão Geral do Spark Streaming

- Divide o dado em lotes de n segundos, minutos etc.
- Processa cada lote no Spark como um RDD.
- Retorna resultados de operações em RDD em lote.



Nova abordagem do Spark Streaming: DStreams (Discretized Stream)

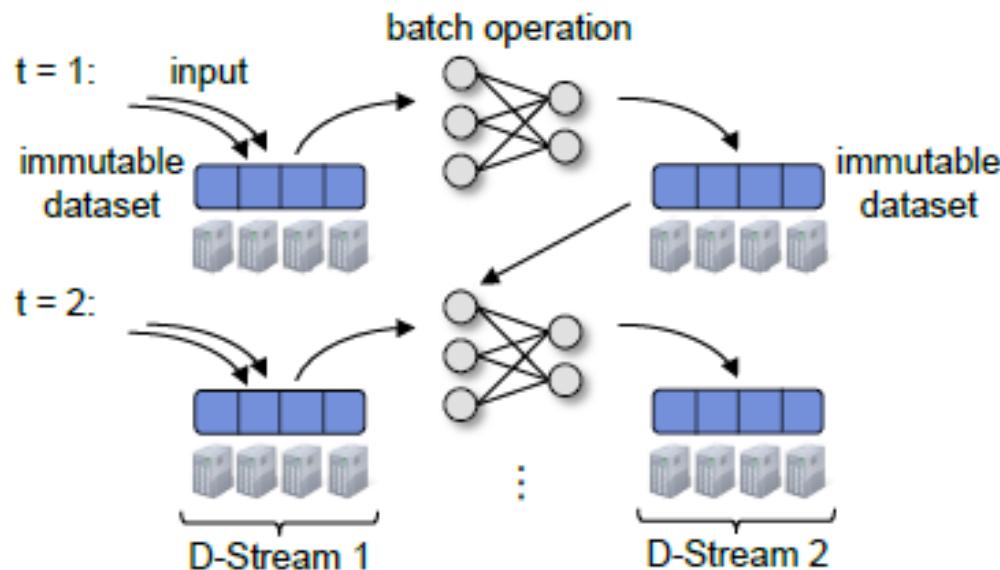
- **Definição:** É uma sequencia de RDDs que representam uma stream. As origens para o DStreams podem ser rede (sockets), ou outros componentes como Flume, Kafka, ZeroMQ, Twitter, ou arquivos no HDFS.
- **Objetivo:** Executar processamento streaming é uma série de Jobs muito pequenos e determinísticos. Exemplo: processar uma stream de tweets em batches de 1 segundo.



ZAHARIA et al (2015) e Shapira et al (2015)

DStreams (Discretized Stream)

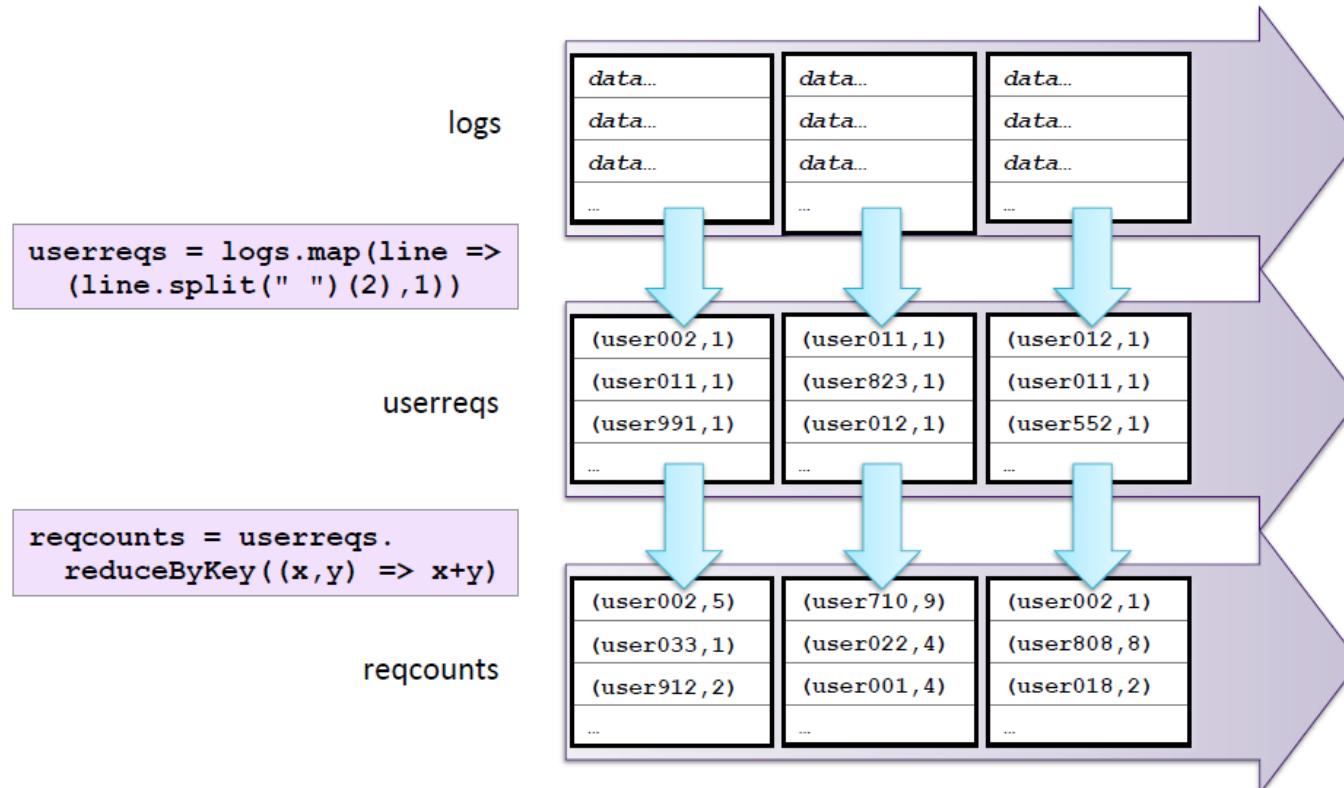
- É uma sequencia de RDDs que representam uma stream. As origens para o DStreams podem ser rede (sockets), ou outros componentes como Flume, Kafka, ZeroMQ, Twitter, ou arquivos no HDFS



ZAHARIA et al (2015) e Shapira et al (2015)

Operadores Dstreams - Transformação

- Transformação: cria novos streams baseado nos que já existem. É possível realizar transformações nos DStreams, como map, flatmap, filter, reducedByKey, joinByKey.

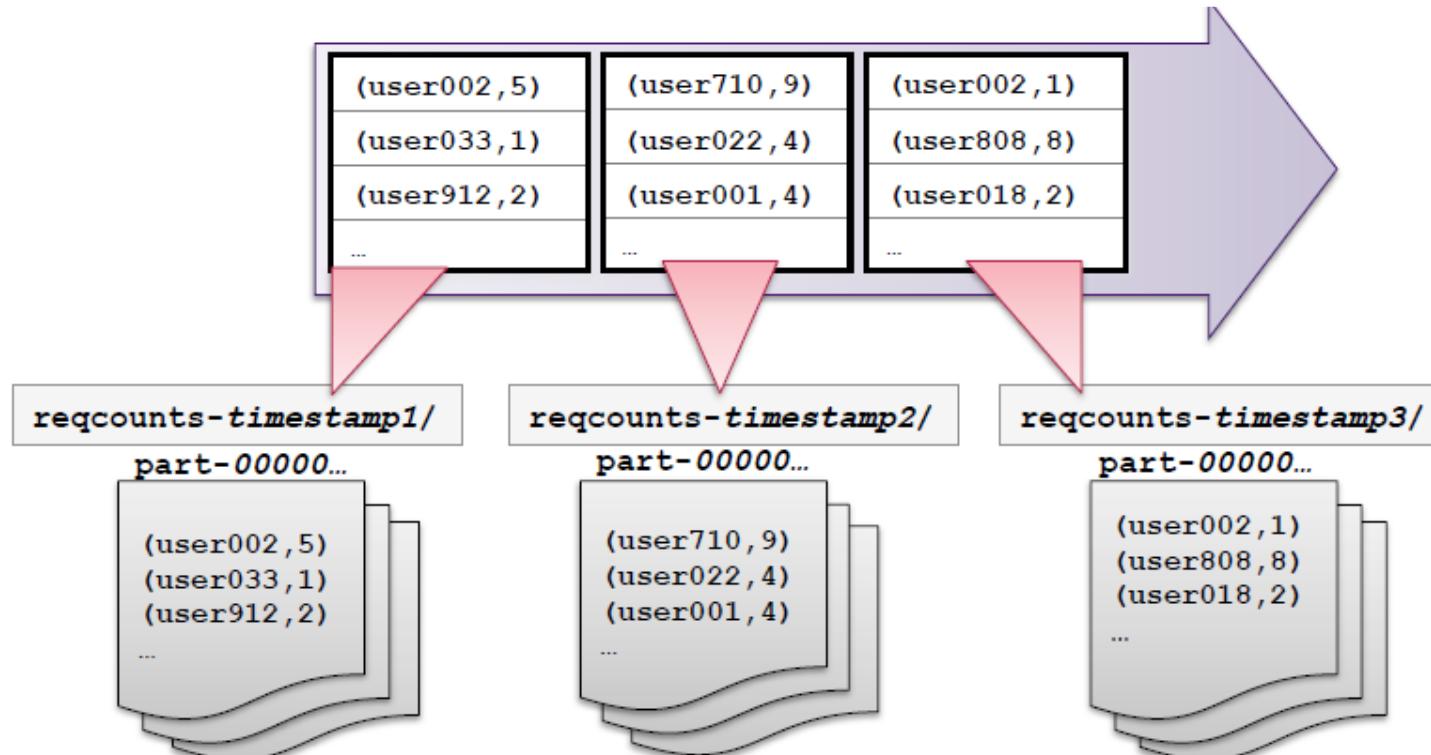


ZAHARIA et al (2015) e Shapira et al (2015)

Operadores Dstreams – Salvando os dados

- Operadores de saída: Salva os resultados em um disco externo.

```
userreqs.saveAsTextFiles (".../outdir/reqcounts")
```



ZAHARIA *et al* (2015) e Shapira *et al* (2015)

Exercício 4 - Spark

Abra o arquivo de exercícios de Spark.

Prof. MSc. Samuel Otero Schmidt



www.linkedin.com/in/samuel-otero-schmidt-16358a98



[@schmidt_samuel](https://twitter.com/schmidt_samuel)



Schmidt_Samuel / Samuel Otero Schmidt



schmidt-samuel@usp.br

Referências Bibliográficas

Patrick Wendell. The Future of Apache Spark, Spark Summit, 2014. Disponível em: [e spark-summit.org/wp-content/uploads/2014/07/Future-of-Spark-Patrick-Wendell.pdf](http://spark-summit.org/wp-content/uploads/2014/07/Future-of-Spark-Patrick-Wendell.pdf)

ZAHARIA, Matei et al. Spark: cluster computing with working sets. In:**Proceedings of the 2nd USENIX conference on Hot topics in cloud computing**. 2010. p. 10-10.

ZAHARIA, Matei et al. Learning Spark: Lightning-fast Data Analysis, O'Reilly Media, 2015.

<http://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html>

Ryza , Sandy; Uri Laserson; Sean Owen; Josh Wills. Advanced Analytics with Apache Spark: The Book, O'Reilly Media, 2015.

MapR. The Future of Hadoop *is Right Now*, 2014. Disponível em: <https://www.mapr.com/wwgd>

Kestelyn, J. How Edmunds.com Used Spark Streaming to Build a Near Real-Time Dashboard, 2014. Disponível em: <http://blog.cloudera.com/blog/2015/03/how-edmunds-com-used-spark-streaming-to-build-a-near-real-time-dashboard/>

Lee, Jungryong. Big Telco, Bigger real-time Demands: Moving Towards Real-time analytics - Jungryong Lee (SK Telecom), 2014. Disponível em: <https://spark-summit.org/2014/talk/big-telco-bigger-real-time-demands-moving-towards-real-time-analytics>

Shapira, Gwen, Jonathan Seidman, Ted Malaska, Mark Grover, Hadoop Application Architectures, O'Reilly Media, Inc., 2015.