



# GERÊNCIA DE INFRAESTRUTURA PARA BIG DATA

---

Tiago Coelho Ferreto – Aula 06

Pós-Graduação em  
Ciência de Dados e Inteligência Artificial

# *Ementa da disciplina*

Introdução à arquitetura para Big Data Analytics. Visão geral sobre Infraestrutura de armazenamento de dados para Big Data. Visão geral sobre Infraestrutura de computação e de rede para Big Data. Tópicos sobre virtualização e computação em nuvem. Plataformas de Big Data na nuvem: HDFS, Hadoop e MapReduce. Estudos de caso com Spark.

# Professores

## **MARCOS TAKESHI**

Professor Convidado

Especialista em Big Data na Semantix, que atua em diversos projetos de empresas do setor financeiro, telecom, varejo e saúde. Realiza análises de arquiteturas, infraestruturas, ambientes, sistemas e ferramentas big data, visando o correto funcionamento e performance. Formado em engenharia eletrônica pela Escola de Engenharia Mauá, pós-graduado em Administração de Empresas pela FGV-SP, MBA em Big Data na FIAP, e empreendedorismo no Babson College.

## **TIAGO COELHO FERRETO**

Professor PUCRS

É professor adjunto da Pontifícia Universidade Católica do Rio Grande do Sul. Possui Doutorado em Ciência da Computação pela PUCRS (2010) com Doutorado sanduíche na Technische Universität Berlin, Alemanha (2007-2008). Tem experiência na área de Ciência da Computação, com ênfase em Redes de Computadores, atuando principalmente nos seguintes temas: computação em nuvem, grades computacionais, virtualização, processamento de alto desempenho e gerência de infraestrutura de TI.

# Encontros e resumo da disciplina

## AULA 1

Para ser um profissional de Data Science é necessário ter paciência e construir um bom Network.

Empresas tem grande interesse em processar os dados e deles extrair informação com a finalidade de monetizar.

É bom estar no meio de pessoas que saibam mais do que você, sempre você tem que estar no meio de pessoas melhores.

**MARCOS TAKESHI**  
Professor Convidado

## AULA 2

O Spark possibilita a obtenção de resultados imediatos.

É importante você saber e conseguir atuar em mais de uma frente.

Certificações podem mostrar que você tem conhecimento do assunto.

**MARCOS TAKESHI**  
Professor Convidado

## AULA 3

Nos últimos anos a gente tem, a cada ano, um novo software auxiliando no processamento de grandes volumes de dados.

Além de armazenar e processar, eu tenho que conseguir extrair valor.

O Hadoop como a principal ferramenta para trabalhar com grandes volumes de dados.

**TIAGO COELHO FERRETO**  
Professor PUCRS

## AULA 4

A redundância garante a persistência da informação.

O HDFS é a principal fonte de dados de entrada e saída do Hadoop.

Como utilizar as aplicações Sqoop e Flume.

**TIAGO COELHO FERRETO**  
Professor PUCRS

## AULA 5

MapReduce uma solução de escalonamento e capacidade de processamento.

Hadoop Streaming como implementação de funções Map e Reduce em linguagens diferentes de Java.

O Pig como linguagem alternativa para programar MapReduce.

**TIAGO COELHO FERRETO**  
Professor PUCRS

## AULA 6

O Hive trabalha com a linguagem SQL com interações através de linhas de comando em formato shell.

O Spark tem como benefícios uma melhor performance, extensibilidade e melhor suporte para outros cenários.

O componente principal do Spark é o RDD (Resilient Distributed Dataset).

**TIAGO COELHO FERRETO**  
Professor PUCRS

**Ciências de Dados e Inteligência Artificial**

# Gerência de Infraestrutura para Big Data

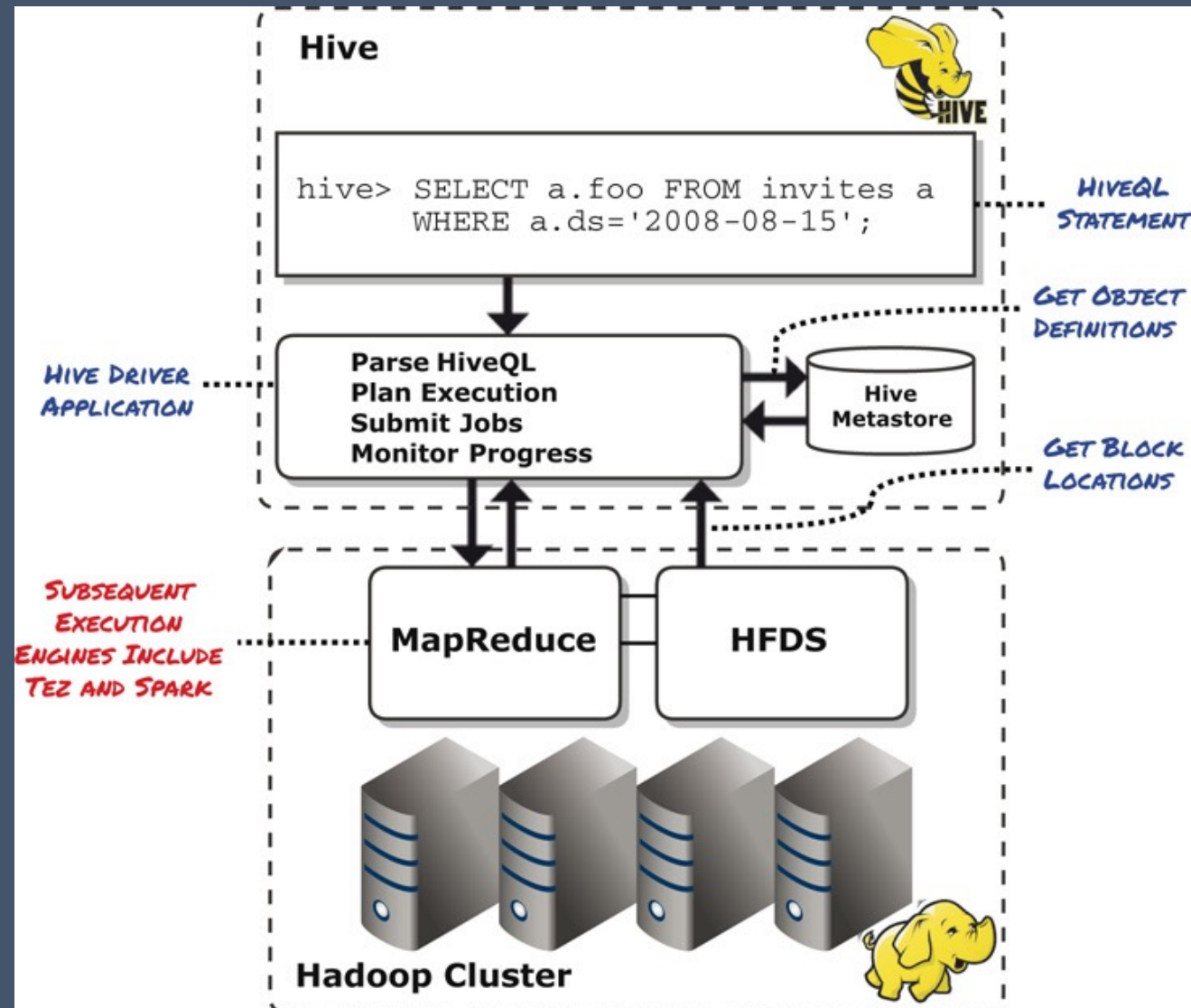
Hive

Prof. Tiago Ferreto

*tiago.ferreto@pucrs.br*

# Hive - Introdução

- Iniciado em 2010 no Facebook
- Objetivo: fornecer uma interface de alto nível para Hadoop MapReduce usando SQL
  - **HiveQL (Hive Query Language)** - implementa um subconjunto de SQL-92 com extensões
- Processo
  - O Hive analisa a consulta em HiveQL e gera operações Java MR
  - MR são enviados ao cluster Hadoop
  - O Hive monitora e retorna os resultados ao cliente (ou grava no HDFS)

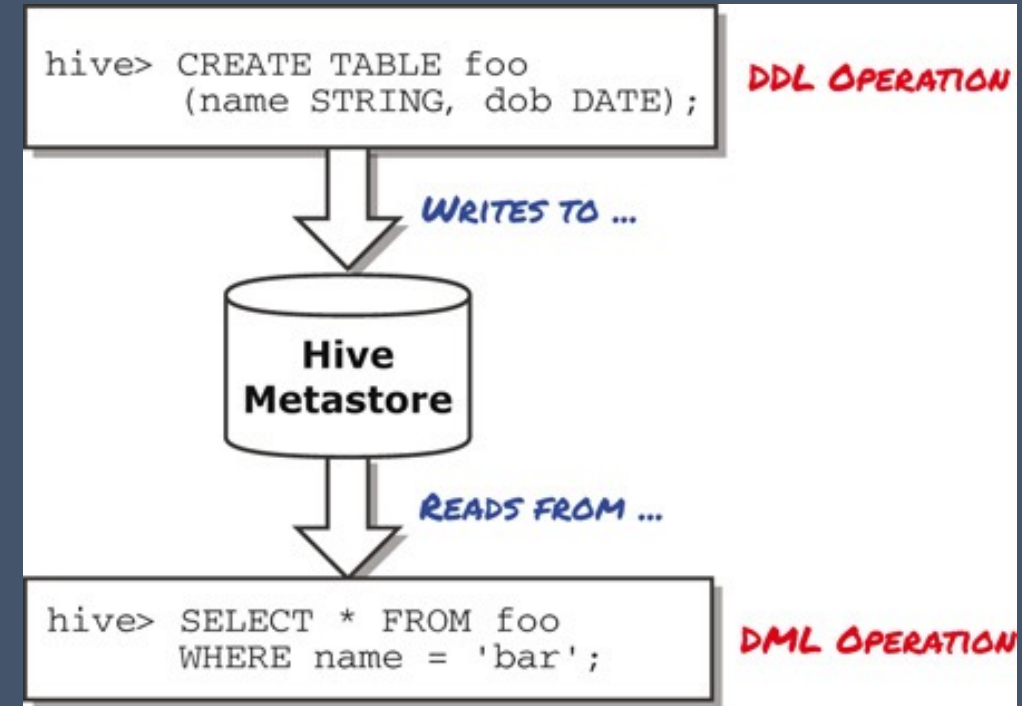


# Objetos Hive

- Hive implementa uma **abstração tabular para objetos no HDFS**
  - Os diretórios e arquivos são representados como tabelas com colunas predefinidas com tipos de dados
  - As tabelas são criadas usando comandos SQL DDL (*Data Definition Language*) e acessadas usando instruções SQL DML (*Data Manipulation Language*)
- Diferenças entre o Hive e uma plataforma de banco de dados convencional
  - UPDATE é implementado como uma transformação de granulação grossa
    - Como o HDFS é um sistema de arquivos imutável, modificações nas tabelas não são permitidas
  - ~~Sem transações, sem rollbacks, sem nível de isolamento de transação real~~
    - Adicionado em Hive 0,14 (transações ACID)
  - ~~Sem integridade referencial declarativa (DRI), sem chaves primárias, sem chaves estrangeiras~~
    - Incluído no Hive 2.1
  - Dados formatados incorretamente são representados para o cliente como NULL

# Hive Metastore

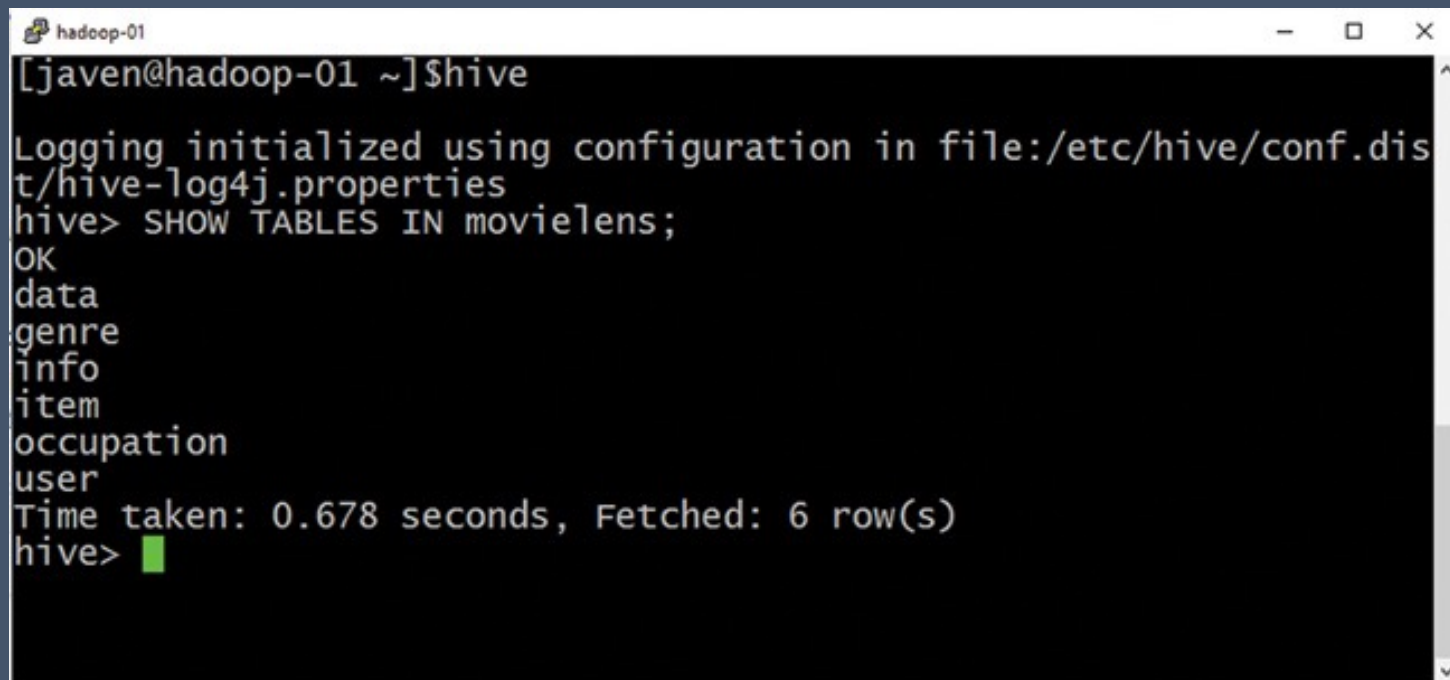
- Mapeamento de tabelas para caminhos no HDFS, colunas e definições são mantidas no Hive Metastore
  - Hive Metastore é um banco de dados relacional acessado pelo cliente Hive
- As definições de objeto incluem formatos de E/S de arquivos (tabelas) e funções de serialização/desserialização para definir como os registros e campos são extraídos
- Hive Metastore é geralmente implementado como um banco de dados Derby integrado
  - Também pode ser implementado como um banco de dados local ou remoto (MySQL, PostgreSQL, etc)
- HCatalog - subprojeto Hive
  - Estende objetos criados no Hive para outros projetos (por exemplo, Apache Pig)





# Hive CLI

- Aceita e analisa comandos de entrada HiveQL
- Método comum para realizar consultas ad hoc
- Usado quando o cliente Hive ou aplicação é instalada na máquina local, incluindo a conexão com o Metastore

A screenshot of a terminal window titled 'hadoop-01'. The prompt is '[javen@hadoop-01 ~]\$'. The user enters 'hive'. The terminal shows the Hive CLI interface with the prompt 'hive>'. The user enters 'SHOW TABLES IN movielens;'. The output lists six tables: 'data', 'genre', 'info', 'item', 'occupation', and 'user'. Below the list, it says 'Time taken: 0.678 seconds, Fetched: 6 row(s)'. The prompt 'hive>' is followed by a green cursor.

```
hadoop-01
[javen@hadoop-01 ~]$hive
Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j.properties
hive> SHOW TABLES IN movielens;
OK
data
genre
info
item
occupation
user
Time taken: 0.678 seconds, Fetched: 6 row(s)
hive> █
```

# HiveServer2

- Abordagem cliente / servidor para usar Hive
- Usado em implementações em grande escala
  - Os detalhes da conexão com o Metastore são mantidos no servidor e o acesso ao cluster é controlado
- Pode atuar como uma aplicação multi-sessão para vários clientes
- Fornece uma interface JDBC para ser acessada por clientes externos (por exemplo, Beeline)

# Hive Batch Mode

- O Hive também oferece suporte à execução não interativa ou em lote
- Exemplo

```
# run all statements in a file
$ hive -f MyHiveQuery.hql
# run an individual statement
$ hive -e "SELECT * FROM mytable"
```

# Banco de dados e tabelas no Hive

- Os objetos Hive consistem basicamente em bancos de dados e tabelas
- Bancos de dados - usados para organização, autorização e gerenciamento de namespace
  - O banco de dados padrão é denominado "*default*"
- Tabelas - existem em um banco de dados Hive
- O contexto do banco de dados pode ser alterado usando a instrução USE

```
hive> USE bikeshare;
```

- Objetos podem ser referenciados usando o identificador completo do objeto (banco de dados e nome da tabela)

```
hive> SELECT * FROM bikeshare.trips;
```

# Autorização Hive

- O Hive oferece suporte para autorização básica
  - Determina os níveis de acesso a objetos em uma instância do Hive
  - Não habilitado por padrão
  - Configurado no arquivo hive-site.xml

```
<property>
  <name>hive.security.authorization.enabled</name>
  <value>true</value>
</property>
<property>
  <name>hive.security.authorization.createtable.owner.grants</name>
  <value>ALL</value>
</property>
```

- Privilégios de nível de objeto são atribuídos usando a instrução GRANT
  - Operações: SELECT, CREATE, ALTER, DROP, ALL, ...
  - Escopo: GROUP, ROLE, USER
  - Revogando privilégios: instrução REVOKE
  - Exemplos:

```
hive> GRANT SELECT ON TABLE trips TO USER user1;
hive> REVOKE SELECT ON TABLE trips FROM USER user1;
```

# Objetos Hive

- O Hive é compatível com os tipos de dados primitivos mais comuns e também com tipos de dados complexos (struct, map, array)

Datatype	Category	Description
TINYINT	Numeric	1-byte signed integer (−128 to 127)
SMALLINT	Numeric	2-byte signed integer (−32,768 to 32,767)
INT	Numeric	4-byte signed integer
BIGINT	Numeric	8-byte signed integer
FLOAT	Numeric	4-byte single precision floating point
DOUBLE	Numeric	8-byte double precision floating point
DECIMAL (p, s)	Numeric	User definable precision and scale
TIMESTAMP	Date/Time	Unix timestamp
DATE	Date/Time	Date (formatted as YYYY-MM-DD)
STRING	String	Character sequence (variable length)
VARCHAR	String	Character sequence (variable length)
CHAR (n)	String	Character sequence (fixed length)
BOOLEAN	Misc	True or False
BINARY	Misc	Raw binary data

# Instrução CREATE TABLE

- Cria uma tabela gerenciada (Managed table) dentro do Hive Metastore
  - Ciclo de vida dos dados é gerenciado totalmente pelo Hive (remoção da tabela remove os dados do HDFS)

```
hive> CREATE TABLE stations (  
> station_id INT,  
> name STRING,  
> lat DOUBLE,  
> long DOUBLE,  
> dockcount INT,  
> landmark STRING,  
> installation STRING  
> )  
> ROW FORMAT DELIMITED  
> FIELDS TERMINATED BY ','  
> STORED AS TEXTFILE;
```

```
hive> DESCRIBE stations;  
OK  
station_id          int  
name                 string  
lat                  double  
long                 double  
dockcount            int  
landmark             string  
installation         string  
Time taken: 0.035 seconds, Fetched: 7 row(s)
```

# Instrução CREATE EXTERNAL TABLE

- Cria somente a estrutura da tabela (esquema) no Hive Metastore. Os dados são mantidos no diretório original do HDFS.
  - A remoção de uma tabela não remove os dados do HDFS
  - Útil quando os dados são usados também fora do Hive

```
hive> CREATE EXTERNAL TABLE stations (  
> station_id INT,  
> name STRING,  
> lat DOUBLE,  
> long DOUBLE,  
> dockcount INT,  
> landmark STRING,  
> installation STRING  
> )  
> ROW FORMAT DELIMITED  
> FIELDS TERMINATED BY ','  
> STORED AS TEXTFILE  
> LOCATION 'hdfs:///user/hadoop/bikeshare/stations';
```



# Formatos de entrada/saída e SerDes no Hive

- O Hive usa um InputFormat e um SerDe (Serializer/Deserializer) para determinar como ler arquivos de entrada e extrair registros para processamento
  - Especificado usando a diretiva STORED AS na instrução CREATE TABLE
  - Exemplo: STORED AS TEXTFILE corresponde ao TextInputFormat
- Outros InputFormats: SEQUENCEFILE, ORC, PARQUET, AVRO
- SerDe padrão usado por Hive - LazySimpleSerDe
  - O arquivo é delimitado por um caractere (imprimível ou não) indicado na cláusula FIELDS TERMINATED BY (definido como Ctrl-A se não estiver presente)
- Outros SerDe (especificado usando a diretiva ROW FORMAT SERDE)
  - RegexSerDe - esquema para dados de largura fixa
- OutputFormat padrão → HiveIgnoreKeyTextOutputFormat (usado para gravar dados em arquivos de texto no Hadoop)
- Outros OutputFormats: ORC, Parquet, Avro, SequenceFiles, ...

# Carregando dados no Hive

- Abordagens para carregar dados em tabelas
  - Copiar o(s) arquivo(s) para o diretório HDFS do objeto de tabela
  - Usar o comando LOAD DATA INPATH do Hive para dados existentes no HDFS, que usa uma função de movimentação HDFS subjacente
  - Usar o comando LOAD DATA LOCAL INPATH do Hive para dados que NÃO EXISTEM no HDFS (usa a função HDFS put)
- Exemplos

```
hive> LOAD DATA INPATH '/bikeshare/stations' INTO TABLE stations;
```

- Use OVERWRITE para substituir o conteúdo existente

```
hive> LOAD DATA INPATH '/bikeshare/stations'  
> OVERWRITE INTO TABLE stations;
```

- Os dados podem ser carregados durante o processo de criação da tabela

```
hive> CREATE TABLE stations_copy AS  
> SELECT * FROM stations;
```

# Analizando dados com Hive

- Hive Query Language (HiveQL) é baseada na especificação SQL-92, com algumas funções adicionais específicas do Hive
- As instruções HiveQL podem abranger várias linhas e são encerradas por um ponto e vírgula
- Comentários de linha única são suportados usando o hífen duplo (--)
- A semântica SQL típica, como listas de colunas e cláusulas WHERE, são totalmente suportadas no Hive
- Exemplo

```
-- this is a comment
hive> SELECT name, lat, long
> FROM stations
> WHERE landmark = 'San Jose';
```

# Funções integradas do Hive

- O Hive inclui várias funções integradas para realizar operações (matemáticas, manipulação de string e data, etc)
  - Exemplos: ROUND, CEIL, FLOOR, RAND, SUBSTRING, LOWER, UPPER, RTRIM, LTRIM, CONCAT, TO\_DATE, DAY, MONTH, YEAR
- Muitas funções são idênticas às disponíveis nos dialetos SQL nos RDBMS mais populares
- Se uma função estiver faltando, UDFs adicionais (funções definidas pelo usuário) podem ser escritas (em Java)
- Exemplos

```
hive> SELECT LOWER(name) FROM stations;
```

- DESCRIBE pode ser usado para mostrar ajuda e uso para funções

```
hive> DESCRIBE FUNCTION RAND;
```

```
OK
```

```
RAND([seed]) - Returns a pseudorandom number between 0 and 1
```

# Agrupamento e agregação de dados

```
hive> SELECT landmark, COUNT(*) FROM stations
> GROUP BY landmark;
OK
Mountain View      7
Palo Alto          5
Redwood City       7
San Francisco      35
San Jose           16
Time taken: 19.443 seconds, Fetched: 5 row(s)
```

# Joins no Hive

- O Hive é compatível com todos os tipos de joins comuns (INNER JOIN - padrão, LEFT OUTER JOIN, RIGHT OUTER JOIN e FULL OUTER JOIN)

```
hive> SELECT t.trip_id,  
> t.duration, t.start_date, s.name  
> FROM stations s  
> JOIN trips t ON s.station_id = t.start_terminal;
```

# Saída de dados com Hive

- O Hive oferece suporte a vários métodos para persistir a saída de uma consulta no sistema de arquivos local ou HDFS
  - INSERT OVERWRITE → envia os resultados da consulta para outra tabela Hive (um diretório no HDFS), substituindo o conteúdo existente
  - INSERT INTO TABLE → envia os resultados da consulta para outra tabela Hive, acrescentando a saída ao conteúdo da tabela (diretório) existente
  - INSERT OVERWRITE DIRECTORY → salva os resultados em um diretório no HDFS que pode ou não ser atribuído a uma tabela Hive
  - INSERT OVERWRITE LOCAL DIRECTORY → salva os resultados em um diretório local, não em HDFS

```
hive> INSERT INTO TABLE trips_counts
> SELECT start_terminal , start_station, COUNT(*) AS count
> FROM trips
> GROUP BY start_terminal, start_station
> ORDER BY count
> DESC LIMIT 10;
```

# Tipos de dados complexos no Hive

- O Hive oferece suporte a tipos de dados complexos ou aninhados
  - Comum em XML, JSON e outras fontes de dados semiestruturados
- **ARRAY**
  - Armazena uma lista ordenada de elementos do mesmo tipo de dados
  - Usa a cláusula **COLLECTION ITEMS TERMINATED BY** para determinar o delimitador usado para ler elementos na coleção

```
hive> CREATE TABLE customers (  
> id INT,  
> fname STRING,  
> lname STRING,  
> email STRING,  
> orderids ARRAY<INT>)  
> ROW FORMAT DELIMITED  
> FIELDS TERMINATED BY '|'   
> COLLECTION ITEMS TERMINATED BY ','  
> ;
```

```
hive> SELECT orderids[0] FROM customers;  
1
```



# Tipos de dados complexos no Hive

- STRUCT

- Consiste em campos nomeados que podem ser do mesmo tipo ou de tipos de dados diferentes
- Usa '<' e '>' para associar tipos de dados a campos

```
hive> CREATE TABLE customers (  
    > id INT,  
    > fname STRING,  
    > lname STRING,  
    > email STRING,  
    > orderids ARRAY<INT>,  
    > email_preferences STRUCT<opcomms:boolean, promos:boolean>  
    > ROW FORMAT DELIMITED  
    > FIELDS TERMINATED BY '|'   
    > COLLECTION ITEMS TERMINATED BY ','  
    > ;  
hive> SELECT email_preferences.opcomms FROM customers;  
TRUE
```

# Tipos de dados complexos no Hive

- MAP
  - Representa pares de chave-valor

```
hive> CREATE TABLE customers (  
  > id INT,  
  > fname STRING,  
  > lname STRING,  
  > email STRING,  
  > orderids ARRAY<INT>,  
  > email_preferences STRUCT<opcomms:boolean, promos:boolean>,  
  > address_map MAP<STRING, STRING>  
  > ROW FORMAT DELIMITED  
  > FIELDS TERMINATED BY '|'   
  > COLLECTION ITEMS TERMINATED BY ','  
  > MAP KEYS TERMINATED BY ':';  
hive> SELECT address_map['city'] FROM customers;  
Hayward
```

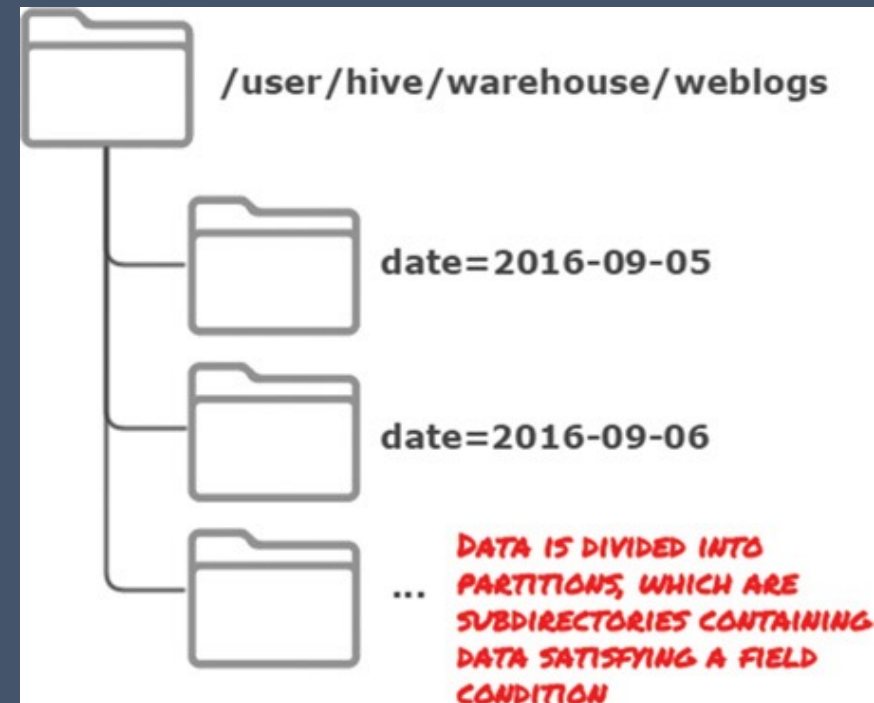
# Gerenciamento e otimização de consulta no Hive

- O Hive requer otimização no design das consultas para melhorar o desempenho
  - Ao contrário de uma plataforma de banco de dados relacional tradicional, o Hive não tem índices e estatísticas para otimizar consultas e acesso a dados
    - OBS. A indexação do Hive foi adicionada na versão 0.7 e removida desde a versão 3.0
      - Alternativas: materialized view ou uso de formatos de arquivo colunares (Parquet, ORC)
- Um fator importante para a otimização é a quantidade de dados lidos por uma consulta específica
- O Hive oferece duas abordagens principais para limitar ou restringir a quantidade de dados que uma consulta precisa ler
  - Partitions
  - Buckets

# Partições no Hive

- Usado para dividir dados em subdiretórios com base em uma ou mais condições (normalmente usado em cláusulas WHERE)
- Usado normalmente para agrupamento de data de granulação grossa
- Exemplo

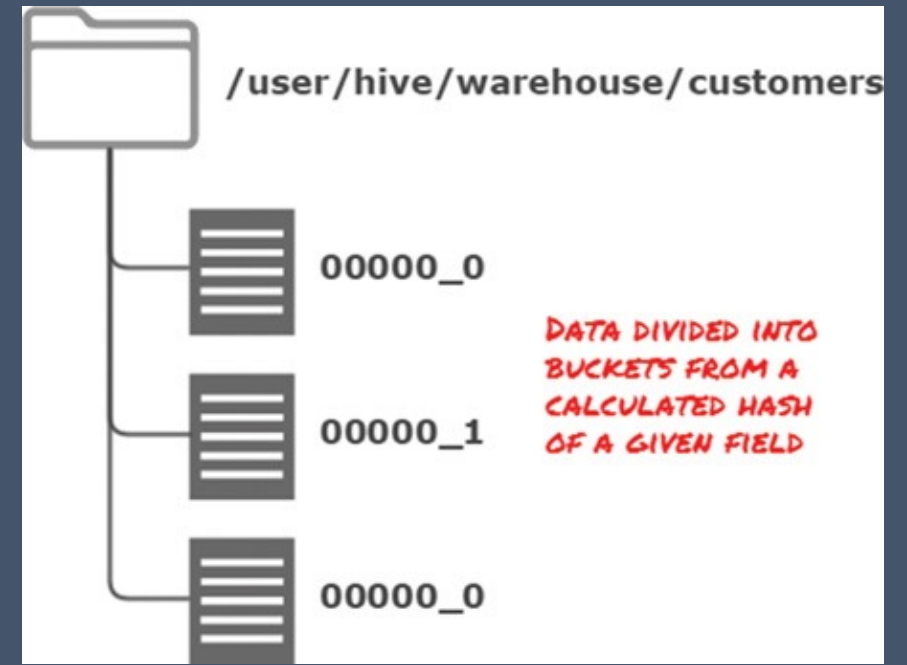
```
hive> CREATE TABLE weblogs (  
  > ip STRING,  
  > time_local STRING,  
  > method STRING,  
  > uri STRING,  
  > protocol STRING,  
  > status STRING,  
  > bytes_sent STRING,  
  > referer STRING,  
  > useragent STRING)  
  > PARTITIONED BY (date STRING);
```



# Buckets no Hive

- Usado para agrupar itens com base em um hash

```
hive> CREATE TABLE customers
> (cust_id INT,
> fname STRING,
> lname STRING,
> email STRING
> )
> CLUSTERED BY (cust_id) INTO 3 BUCKETS;
```



# Instrução EXPLAIN

- Usado para apresentar o plano de execução do otimizador de consulta Hive para uma consulta específica

```
hive> EXPLAIN SELECT method, COUNT(*) FROM weblogs GROUP BY method;  
STAGE DEPENDENCIES:  
Stage-1 is a root stage  
Stage-0 depends on stages: Stage-1  
STAGE PLANS:  
Stage: Stage-1  
Map Reduce  
Map Operator Tree:  
TableScan  
alias: weblogs  
...
```

**Ciências de Dados e Inteligência Artificial**

# Gerência de Infraestrutura para Big Data

Hive

Prof. Tiago Ferreto

*tiago.ferreto@pucrs.br*

# Ciências de Dados e Inteligência Artificial

## Gerência de Infraestrutura para Big Data



Prof. Tiago Ferreto  
*tiago.ferreto@pucrs.br*



# Introdução ao Spark

- Iniciado em 2009 - Berkeley RAD Lab (Universidade da Califórnia)
- Criado como uma **alternativa ao MapReduce** no Hadoop
  - MapReduce é inadequado para consultas interativas ou em tempo real, aplicações de baixa latência
    - MR persiste os dados intermediários no disco entre as fases de processamento Map e Reduce
  - Benefícios do Spark
    - melhor performance
    - extensibilidade
    - melhor suporte para outros cenários (acesso SQL, processamento de dados de streaming, processamento de grafos e NoSQL, aprendizado de máquina, etc.)
- Projeto ASF - <http://spark.apache.org/>
  - Vários contribuidores: Facebook, Yahoo!, Intel, Netflix, Databricks, etc

# Características do Spark

- Escrito em Scala (construído sobre a JVM e Java runtime)
  - Multiplataforma (compatível com Windows e Linux)
- Permite que os desenvolvedores criem rotinas complexas de processamento de dados multi-estágios
  - Fornece uma API de alto nível e uma estrutura tolerante a falhas
- O Spark implementa uma estrutura distribuída e tolerante a falhas na memória chamada **RDD (Resilient Distributed Dataset)**
  - Maximiza o uso de memória nas máquinas → melhora o desempenho em ordens de magnitude

# Aplicações típicas do Spark

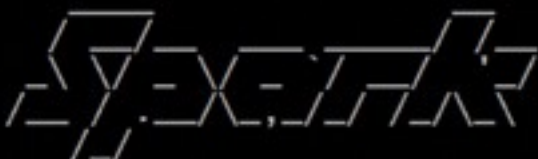
- Operações de extração-transformação-carregamento (ETL)
- Análise preditiva e aprendizado de máquina
- Operações de acesso a dados (como consultas SQL)
- Mineração e processamento de texto
- Processamento de eventos em tempo real
- Processamento de grafos
- Reconhecimento de padrões
- Mecanismos de recomendação

# Interfaces de programação

- Fornece suporte nativo para
  - Scala
  - Python
  - Java
  - SQL
  - R
  - E outros (Clojure, Julia, etc)

# Interação com Spark

- Spark fornece shells interativos em Python (**PySpark**), Scala, R e SQL

```
root@mycluster:~  
[root@mycluster ~]# pyspark  
Python 2.6.6 (r266:84292, Jul 23 2015, 15:22:56)  
[GCC 4.4.7 20120313 (Red Hat 4.4.7-11)] on linux2  
Type "help", "copyright", "credits" or "license" for more information.  
15/11/17 00:16:08 WARN NativeCodeLoader: Unable to load native-hadoop library for  
your platform... using builtin-java classes where applicable  
Welcome to  
 version 1.3.1  
Using Python version 2.6.6 (r266:84292, Jul 23 2015 15:22:56)  
SparkContext available as sc, HiveContext available as sqlContext.  
>>> █
```

# Uso não-interativo

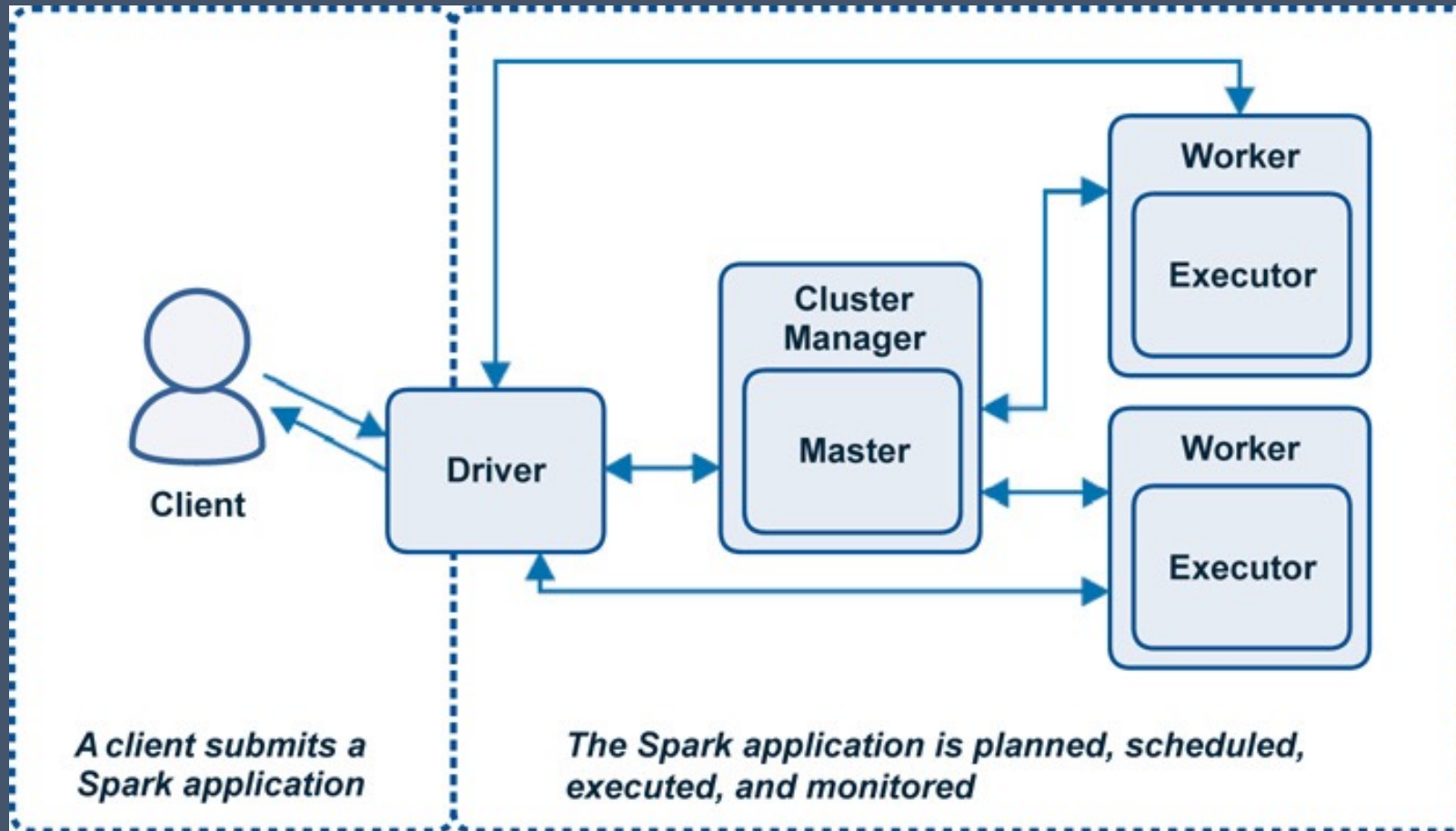
- O Spark fornece o comando `spark-submit` para executar aplicações não-interativas

```
$SPARK_HOME/bin/spark-submit \  
--class org.apache.spark.examples.SparkPi \  
--master yarn-cluster \  
--num-executors 4 \  
--driver-memory 10g \  
--executor-memory 10g \  
--executor-cores 1 \  
lib/spark-examples*.jar 10
```

# Tipos de entrada/saída

- Suporta vários sistemas de origem/destino
  - HDFS
  - Sistemas de arquivos locais ou de rede
  - Armazenamento de objetos, como Amazon S3 ou Ceph
  - Sistemas de banco de dados relacional
  - Bancos de dados NoSQL, incluindo Apache Cassandra, HBase e outros
  - Sistemas de mensagens, como Kafka

# Arquitetura Spark





# Gerenciadores de cluster suportados

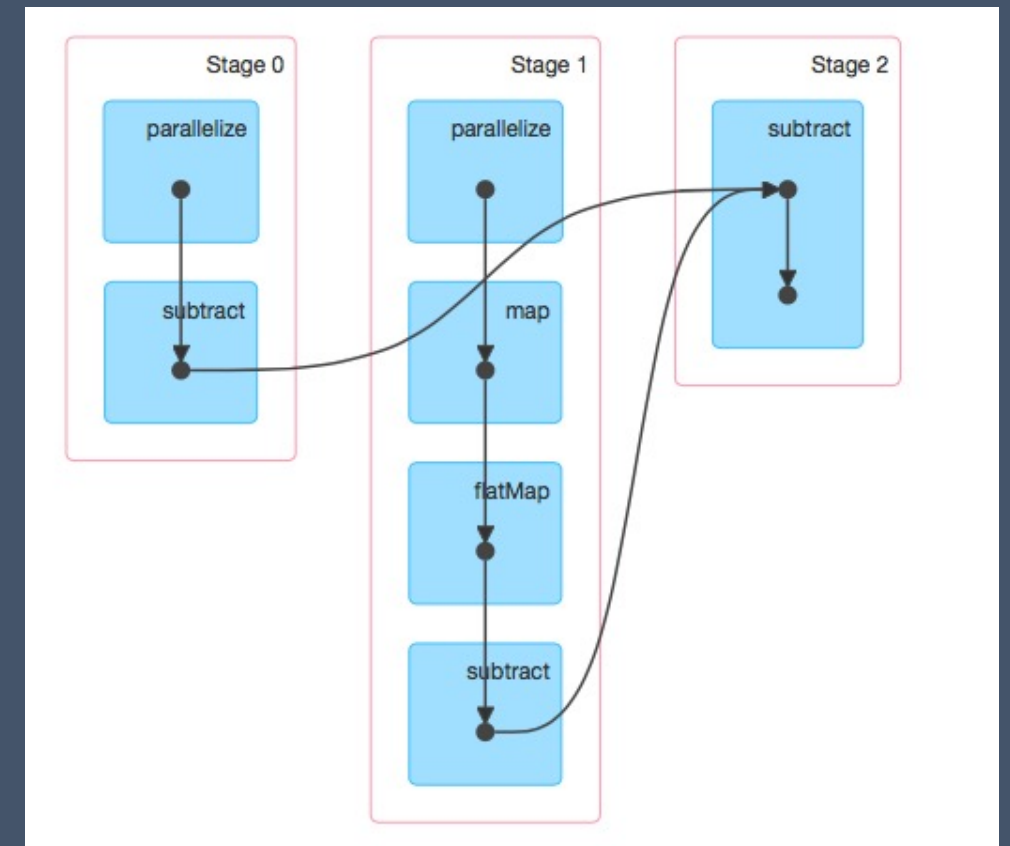
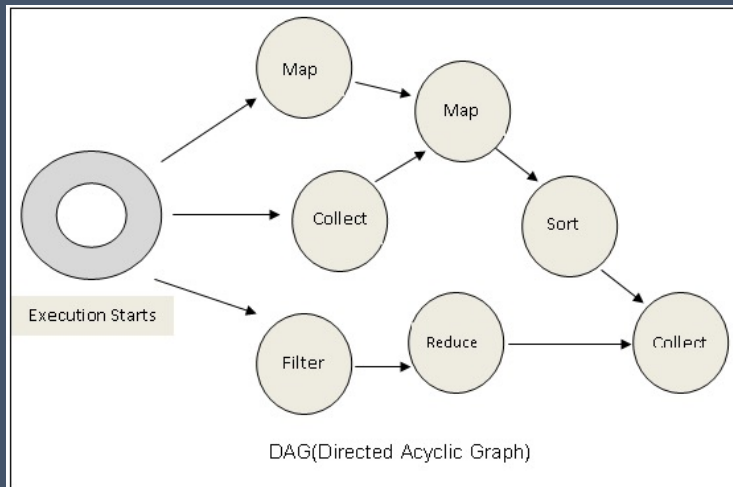
- **Standalone** – gerenciador de cluster simples incluído no Spark que facilita a configuração de um cluster
- **Apache Mesos** - gerenciador de cluster geral que também pode executar Hadoop MapReduce
- **Hadoop YARN** - gerenciador de recursos no Hadoop
- **Kubernetes** - sistema de código aberto para automatizar a implantação, escalonamento e gerenciamento de aplicativos em contêineres

# Spark Driver

- Representado pelo processo usado pelos clientes para submeter aplicações
- Gerencia o **ciclo de vida de aplicações Spark**
- Cria um **SparkContext**
  - Instância de aplicação que representa a conexão entre o mestre e os executores do Spark
  - É criado no início de uma aplicação Spark e usado até o encerramento do programa
- **Planeja e coordena a execução** do programa Spark
  - Fases de planejamento e escalonamento

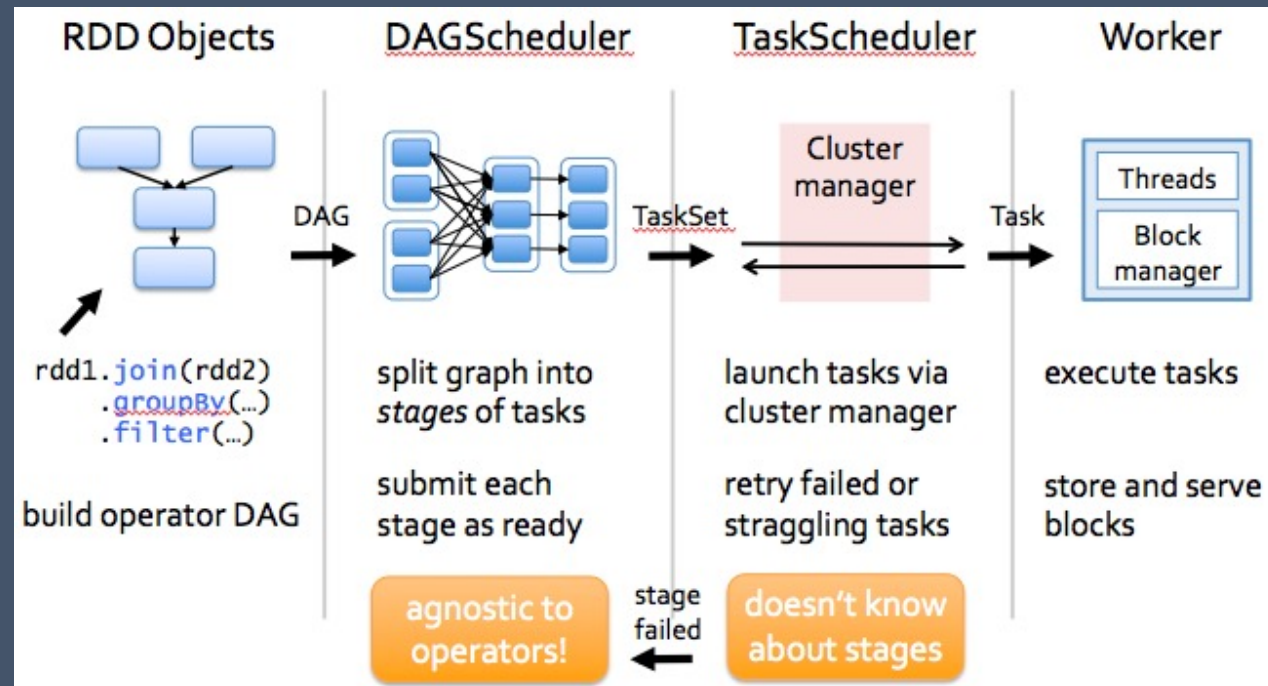
# Spark Driver - Planejamento

- Cria um **DAG (Directed Acyclic Graph)** com base nas transformações solicitadas (operações de manipulação de dados) e ações (solicitações de saída)
  - DAG consiste em **tarefas** e **estágios**
    - **Tarefa** → menor unidade de trabalho agendável
    - **Estágio** → conjunto de tarefas que podem ser executadas juntas
    - Os estágios são dependentes entre si



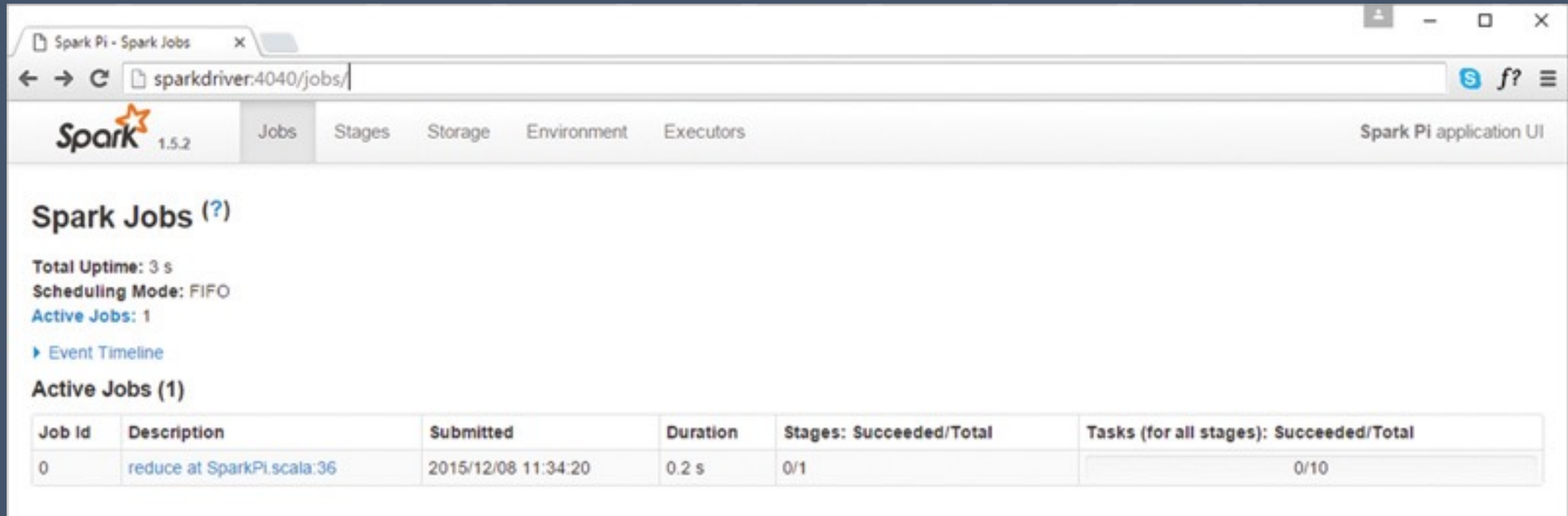
# Spark Driver - Escalonamento

- Coordena a execução de estágios e tarefas
  - Acompanha os recursos disponíveis para executar tarefas
  - Agenda tarefas perto dos dados
  - Coordena a localização e a movimentação de dados entre os estágios



# Spark Driver

- Responsável por retornar resultados de uma aplicação
  - Pode ser código de retorno ou dados
- Fornece uma Web UI na porta 4040



The screenshot displays the Spark Pi application UI in a web browser. The browser's address bar shows the URL `sparkdriver:4040/jobs/`. The UI features a navigation bar with tabs for **Jobs**, **Stages**, **Storage**, **Environment**, and **Executors**. The **Jobs** tab is selected, showing the **Spark Jobs (?)** section. This section includes summary statistics: **Total Uptime: 3 s**, **Scheduling Mode: FIFO**, and **Active Jobs: 1**. A link for **Event Timeline** is also present. Below these statistics, the **Active Jobs (1)** section contains a table with the following data:

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	<a href="#">reduce at SparkPi.scala:36</a>	2015/12/08 11:34:20	0.2 s	0/1	0/10

# Spark - Executores

- Consiste em processos nos quais as tarefas de um Spark DAG são executadas
- Os executores reservam **recursos de CPU e memória** em nós de um cluster Spark
- Eles são **dedicados a uma aplicação Spark específica** e encerrados quando a aplicação é concluída
  - Um executor Spark pode executar centenas ou milhares de tarefas em um programa Spark.
- Os executores do Spark são **hospedados em JVMs**
- Os executores **armazenam dados de saída de tarefas na memória ou em disco**
  - Executores só estão cientes das tarefas atribuídas a eles
  - O driver é responsável por compreender o conjunto completo de tarefas e suas respectivas dependências que compõem uma aplicação

# RDD – Resilient Distributed Dataset

- Conjuntos de dados em um aplicativo Spark
  - Desde o carregamento dos dados até o seu resultado final
- Suporta vários tipos de elementos: inteiros, ponto flutuante, strings, listas, hashes, objetos aninhados, objetos Java e Scala serializados, etc.
- **Normalmente armazenado na memória** (mas também pode ser persistido no disco)

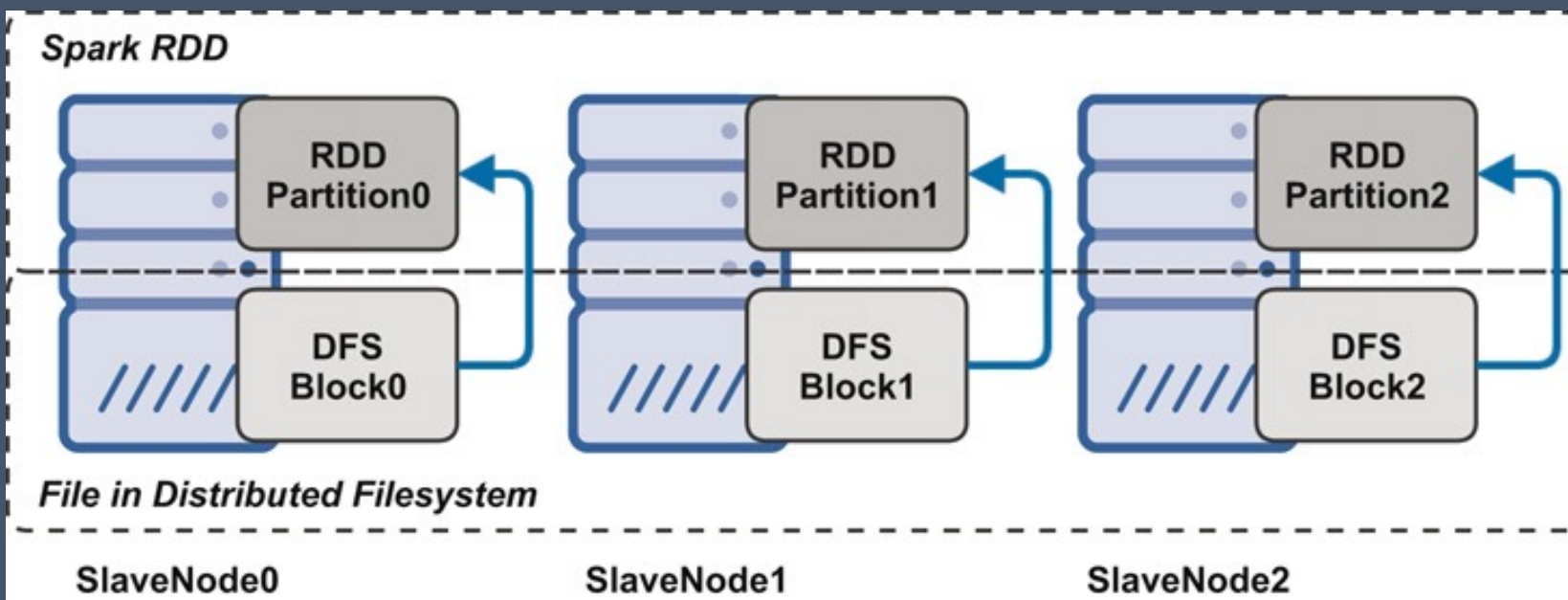
# Características do RDD

- **Resiliente** - pode ser reconstruído se um nó for perdido → Spark sabe como gerar cada RDD (RDDs NÃO SÃO REPLICADOS como blocos HDFS)
- **Distribuído** - os dados em RDDs são divididos em uma ou mais partições e são distribuídos como coleções de objetos na memória entre nós do cluster
- **Conjunto de dados** - RDDs consistem em registros, que são coleções de dados identificáveis dentro de um conjunto de dados
- **Sem compartilhamento** - os RDDs são particionados de forma que cada partição contenha um conjunto único de registros e possa ser operado de forma independente
- **Imutabilidade** - RDDs não podem ser atualizados após serem instanciados e preenchidos com dados → novos RDDs são criados por meio de transformações



# Localidade de dados com RDDs

- O Spark lê dados em um RDD dos nós próximos a ele
- Como o Spark geralmente acessa dados particionados distribuídos (do HDFS), ele cria partições para conter blocos subjacentes do sistema de arquivos distribuído
- Os RDDs também podem ser carregados de outras fontes de dados (de bancos de dados relacionais até objetos Python / Scala simples)

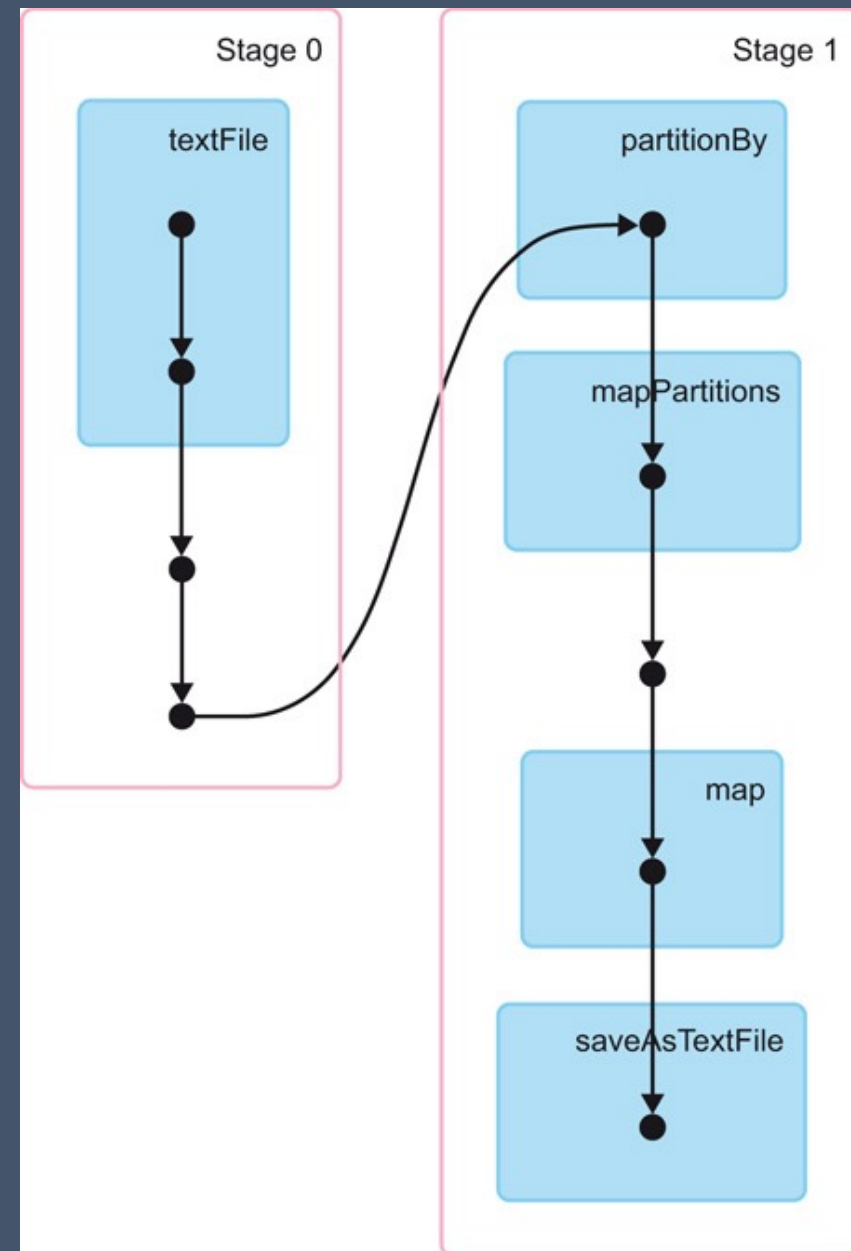


# Persistência e reutilização de RDD

- Normalmente RDDs são objetos transitórios que existem apenas enquanto são necessários
- Impacto no desempenho quando um RDD é necessário para mais de uma ação
- **Spark fornece métodos de API para persistir, armazenar em cache e verificar RDDs**

# RDD Lineage

- O Spark mantém o controle de cada linhagem  
→ **sequência de operações que resultou no RDD**
- Cada operação RDD recalcula toda a linhagem por padrão, a menos que a persistência RDD seja solicitada



# Tolerância a falhas com RDDs

- Em caso de qualquer falha, qualquer RDD pode ser reconstruído usando registros Spark da linhagem RDD
- Uma vez que os RDDs são distribuídos, eles podem tolerar e se recuperar da falha de qualquer nó do cluster

# Transformações e ações no Spark

- **Transformações** - operações realizadas contra RDDs resultando em novos RDDs

- Transformações comuns: funções de mapa e filtro
- Exemplo

```
originalrdd = sc.parallelize([0, 1, 2, 3, 4, 5, 6, 7, 8])  
newrdd = originalrdd.filter(lambda x: x % 2)
```

- **Ações** - operações que produzem saída de um RDD ou salvam o conteúdo de um RDD em um sistema de arquivos (local, HDFS, S3 ou outro)

- Exemplo

```
newrdd.collect() # will return [1, 3, 5, 7]
```

# Lazy Evaluation (Avaliação Preguiçosa)

- **O processamento é adiado até que uma ação seja chamada** (ou seja, quando uma saída é necessária)
- Várias transformações em RDDs podem ser realizadas antes que qualquer processamento seja iniciado
- Cada instrução é analisada sintaticamente até que uma ação seja solicitada (por exemplo, *count ()* ou *saveAsTextFile ()*)
- Um DAG é criado junto com planos de execução lógica e física, que são orquestrados e gerenciados pelo driver
- Essa abordagem permite reduzir estágios de processamento e minimizar a quantidade de dados transferidos entre os executores do Spark

# DataFrames

- Maior nível de abstração para acessar RDDs
- Semelhante a dataframes em Pandas e R
- Aplica um esquema de dados semelhante a uma tabela em um RDD
- Pode ser criado a partir de
  - RDD existente, arquivo JSON, arquivo de texto, tabela Hive, tabela RDBMS, etc.
- Criação de um DataFrame a partir de um RDD existente
  - *SparkSession.createDataFrame (dados, esquema = Nenhum, samplingRatio = Nenhum)*
  - Exemplo

```
myrdd = sc.parallelize([('Jeff', 48), ('Kellie', 45)])  
spark.createDataFrame(myrdd).collect()  
# returns:  
# [Row(_1=u'Jeff', _2=48), Row(_1=u'Kellie', _2=45)]
```

# Extensões Spark

- SparkSQL - fornece abstração semelhante a SQL para Spark
- Spark Streaming - permite o processamento de fluxos de dados
- SparkR - mecanismo de execução com a linguagem R
- MLlib - biblioteca de aprendizado de máquina integrada ao Spark
- GraphX - processamento de grafos com Spark

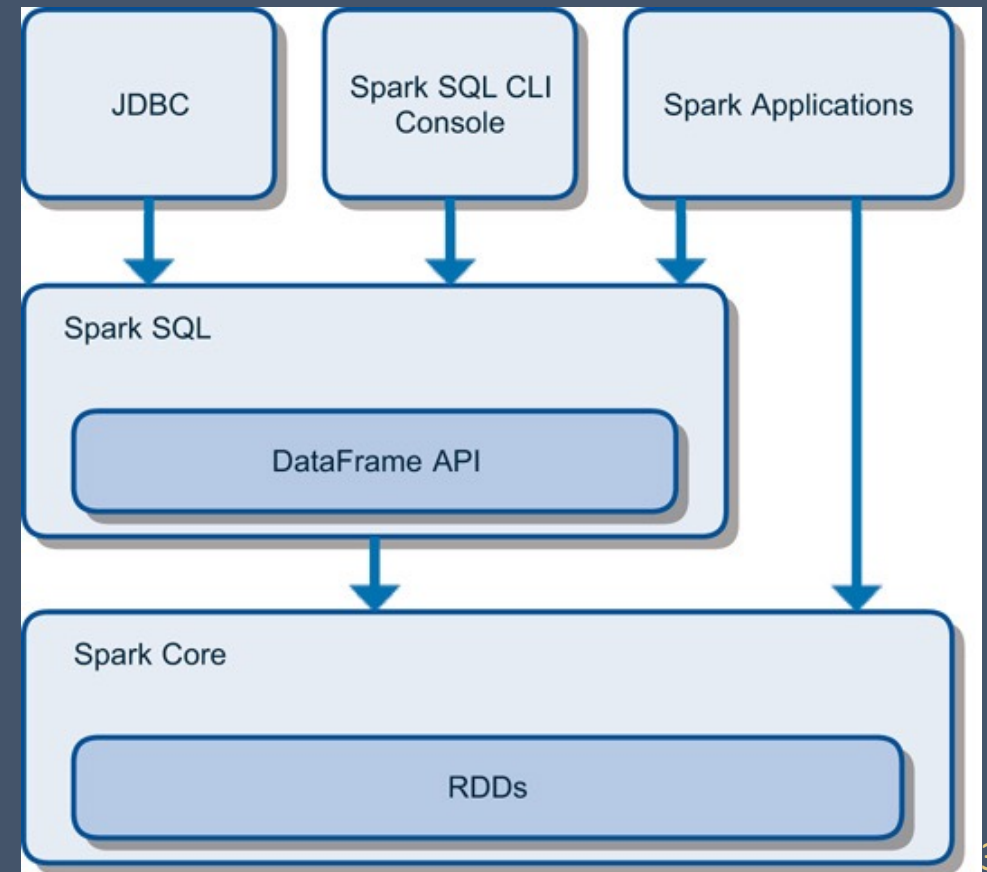


# SparkSQL

- Fornece **abstração semelhante a SQL** para a API principal do Spark
- DataFrame - extensão para RDD implementando um esquema de armazenamento colunar na memória otimizado para acesso SQL
- Exemplo

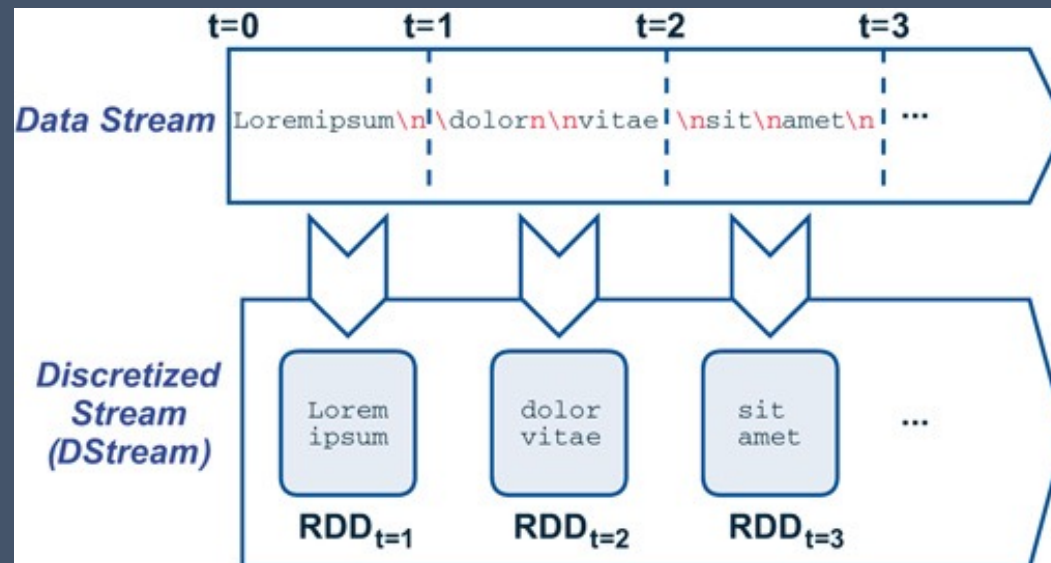
```
sql_cmd = """SELECT name, lat, long
FROM stations
WHERE landmark = 'San Jose'"""
df = sqlContext.sql(sql_cmd)
df.show()
```

```
+-----+-----+-----+
|           name|        lat |        long|
+-----+-----+-----+
|  Adobe on Almaden|37.331415|   -121.8932|
| San Pedro Square|37.336721|-121.894074|
| Paseo de San Antonio|37.333798|-121.886943|
| San Salvador at 1st|37.330165|-121.885831|
|      Japantown|37.348742|-121.894715|
|           ...   |        ... |        ...   |
+-----+-----+-----+
```



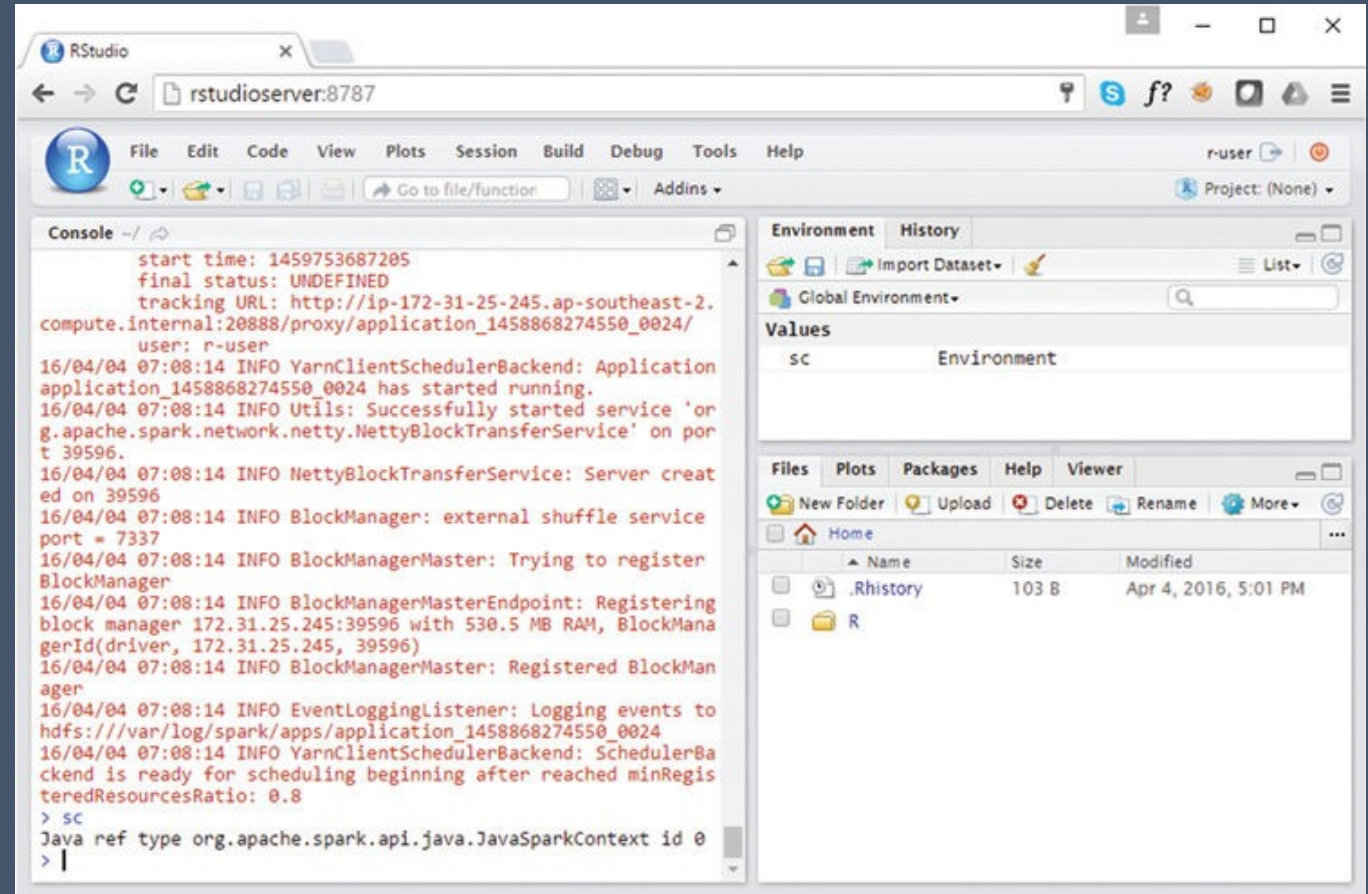
# Spark Streaming

- Apresenta objetos e funções projetadas para processar fluxos de dados
- **DStream (fluxo discretizado)** - abstração RDD composta de fluxos de dados agrupados em RDDs com base em intervalos de tempo
- As transformações do Spark podem ser aplicadas a Dstreams usando funções a cada RDD no Dstream
- Suporta operações de processamento de fluxo (operações de estado e janela deslizando)
  - Exemplo: operações em “janelas” de dados (hora, dia, etc)



# SparkR

- Spark engine para R
  - R → linguagem de programação e ambiente para computação estatística, análise visual e modelagem preditiva
- Fornece acesso ao Spark e operações em dataframes distribuídos através do ambiente R
- Os dataframes R podem ser usados para operações distribuídas com Spark
- Disponível através do shell SparkR ou RStudio



# MLlib

- Biblioteca integrada do Spark para **aprendizado de máquina**
- API construída sobre o modelo RDD
- Inclui algoritmos de aprendizado de máquina comuns e utilitários para preparação de dados, extração de recursos, treinamento de modelo e teste

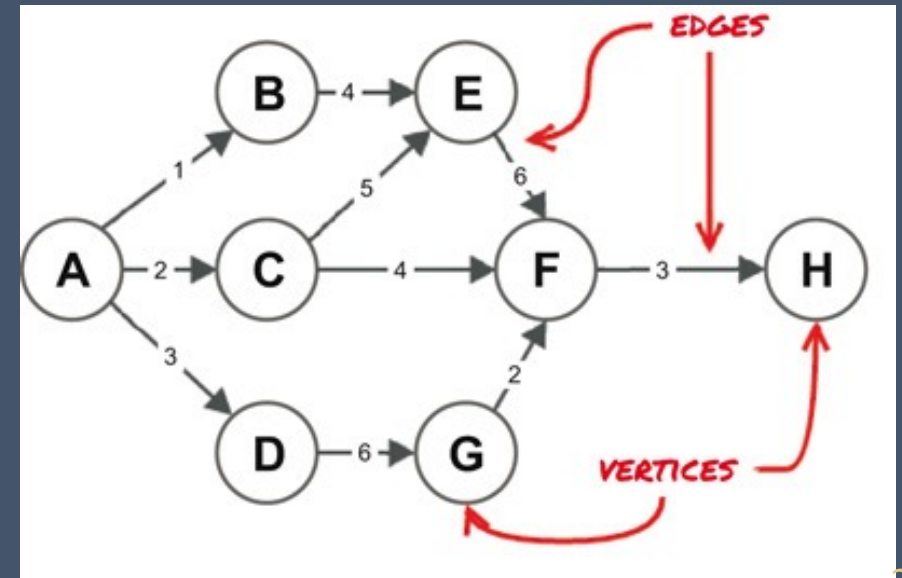
```
from pyspark.mllib.tree import DecisionTree
model = DecisionTree.trainClassifier(data=data,
                                     numClasses=2,
                                     categoricalFeaturesInfo={0: 3})

...
print(model.toDebugString())
If (feature 2 <= 80.0)
    If (feature 1 <= 65.0)
        If (feature 1 <= 64.0)
            Predict: 1.0
        Else (feature 1 > 64.0)
            Predict: 0.0
        Else (feature 1 > 65.0)
            Predict: 1.0
    Else (feature 2 > 80.0)
        If (feature 0 in {0.0})
            Predict: 0.0
        Else (feature 0 not in {0.0})
            If (feature 1 <= 71.0)
                If (feature 1 <= 70.0)
                    Predict: 1.0
                Else (feature 1 > 70.0)
                    Predict: 0.0
            Else (feature 1 > 71.0)
                Predict: 1.0
```

# GraphX

- Processamento de grafos usando Spark
  - Permite modelar relacionamentos entre diferentes entidades ou itens de dados discretos
- Fornece um conjunto completo de abstrações, transformações e algoritmos para processamento de grafos
- **GraphFrames** - abstração RDD para processamento de grafo
- Exemplo usando o algoritmo PageRank
  - usado em motores de busca na web

```
graph = GraphRDD(vertices, edges)
ranks = graph.pageRank(0.0001).vertices
```



# Ciências de Dados e Inteligência Artificial

## Gerência de Infraestrutura para Big Data



Prof. Tiago Ferreto  
*tiago.ferreto@pucrs.br*

**PUCRS** online  **UOL** edtech\_