

# 云计算服务端技术—Spark与Spark SQL

陈敏刚 博士

2017/4/17

上海计算机软件技术开发中心  
上海市计算机软件评测重点实验室



# 目录

---

1

**Spark简介**

2

**Spark RDD、编程模型及运行架构**

3

**Spark RDD的操作及应用实例**

4

**Spark SQL与DataFrame**

5

**Spark SQL应用实例**

# 什么是Spark

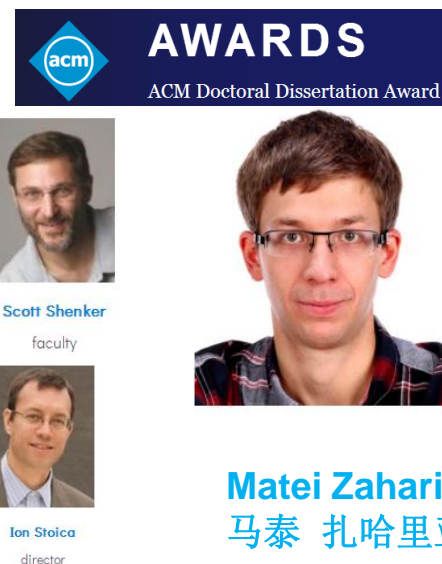


- Apache Spark是快速的、通用的大数据分析处理引擎。  
<http://spark.apache.org/> , 目前最新版本为2.1.0
- Spark扩展了MapReduce计算模型, 而且高效地支持更多计算模式(场景), 包括交互式查询( Spark SQL )、流式处理( Spark Streaming )、图数据处理( Spark GraphX )和分布式机器学习( MLlib )
- Spark的目标是将各种计算模式(场景)在统一的框架下运行, 是大数据分析处理统一的软件栈
- Spark提供了Python、Scala、Java、SQL API , 及丰富的内建的程序库



# Spark的简史

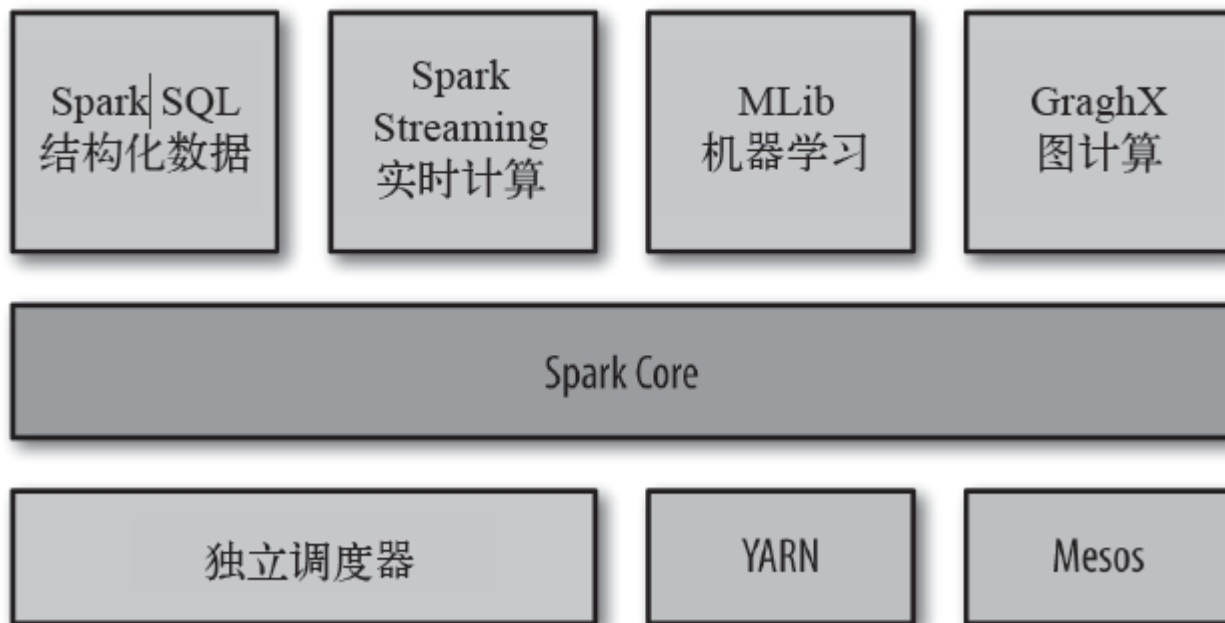
- Apache Spark 2009年在加州伯克利的AMP实验室的一个研究项目中诞生



- Spark 2010年3月开源，2013年6月交给Apache基金会，现已成为Apache顶级项目
- M. Zaharia, R. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, I. Stoica. **Apache Spark: A Unified Engine for Big Data Processing**, Communications of the ACM, 59(11):56-65, November 2016.
- M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M.J. Franklin, S. Shenker, I. Stoica. **Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing**, NSDI 2012. Best Paper Award and Honorable Mention for Community Award.

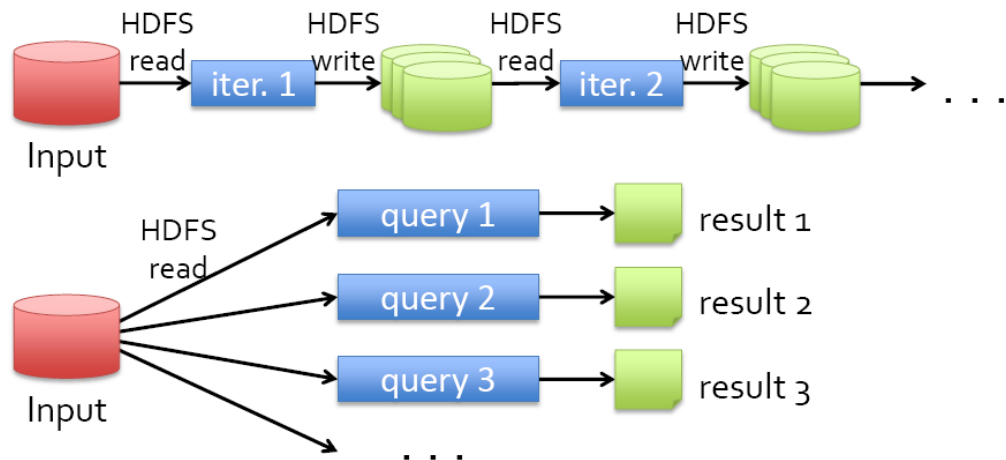


# Spark统一软件栈



- ✓ Spark Core: RDD、任务调度、内存管理、错误恢复、与存储交互
- ✓ Spark SQL: Spark用来操作结构化数据的程序包
- ✓ Spark Streaming: Spark实时数据流式处理的组件
- ✓ MLlib: Spark分布式机器学习库
- ✓ GraphX: Spark并行图计算程序库

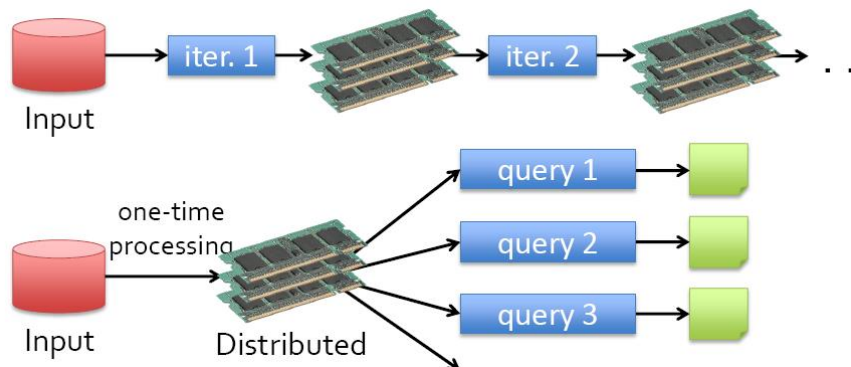
# Spark的动机



MapReduce由于数据复制、序列化和磁盘I/O导致数据处理性能较差

Spark采用内存计算，比MapReduce采用硬盘方案，数据处理性能提升10-100倍

Spark适用于需要多次操作特定数据集的应用场合，比如交互式查询、机器学习算法



# 目录

---

1

Spark简介

2

Spark RDD、编程模型及运行架构

3

Spark RDD的操作及应用实例

4

Spark SQL与DataFrame

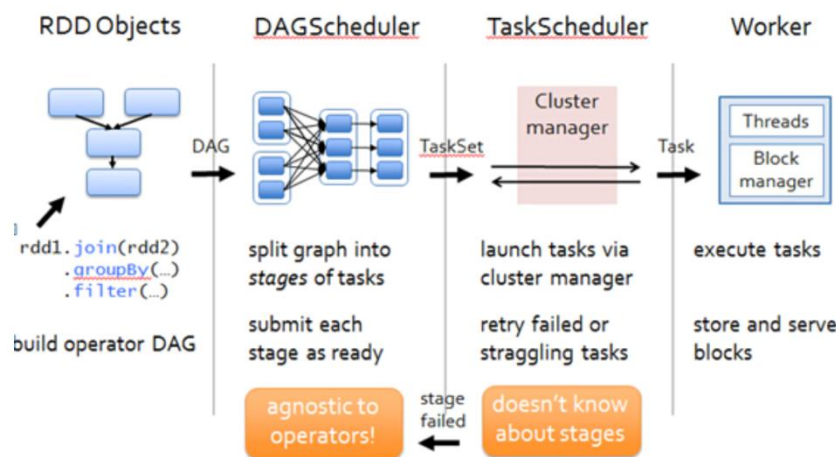
5

Spark SQL应用实例



# 什么是RDD ?

- RDD ( Resilient Distributed Datasets ) 弹性分布式数据集
  - ✓ RDD是Spark的数据抽象，其基于数据集合，而不是单个数据
  - ✓ 由确定性的粗粒度操作产生 (map, filter, join等)
  - ✓ 数据一旦产生，就不能修改 (immutable)
  - ✓ 如果要修改数据，要通过数据集的变换 (Transformations) 来产生新的数据集 (RDD)





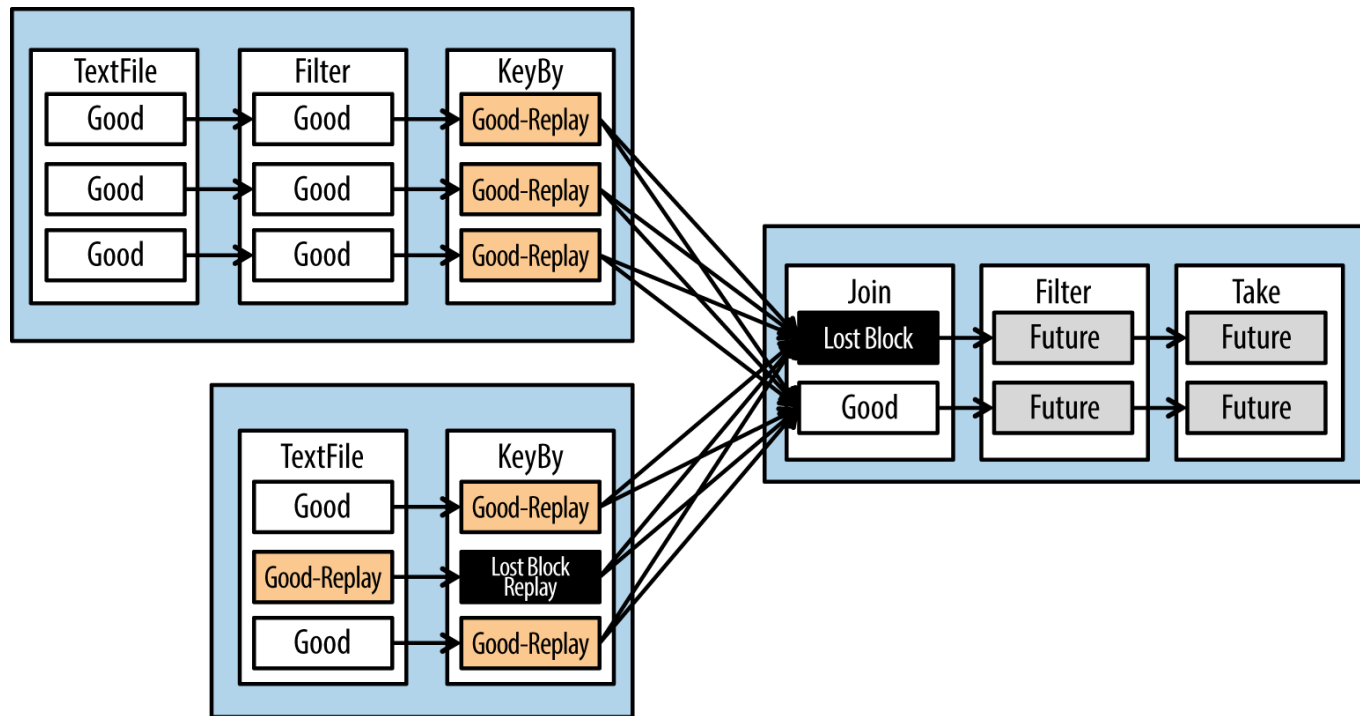
# RDD如何容错？

- 数据一旦是确定性的产生，并且产生后不会变化
  - ✓ 通过“重复计算”的方法就可以来恢复数据
  - ✓ 只要记住RDDs的生成过程就可以了，这样一次log可以用于很多数据，在不出错的情况下，几乎没有开销
  - ✓ Spark RDD的这种容错机制称为“Lineage”（血缘）

```
messages = textFile(...).filter(_.contains("error"))  
                        .map(_.split('\t')(2))
```




# RDD如何容错？

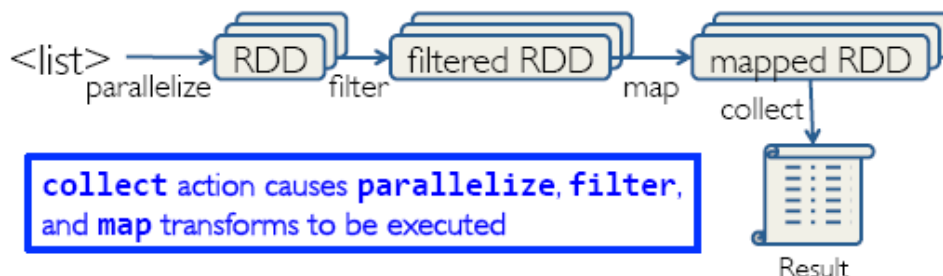


假设黑色框图表示丢失的分区，如图3-8所示。Spark 会重新执行GoodReplay 框图对应的任务，以及Lost Block 框图对应的任务，从而得到需要的数据，继续执行最终步骤。

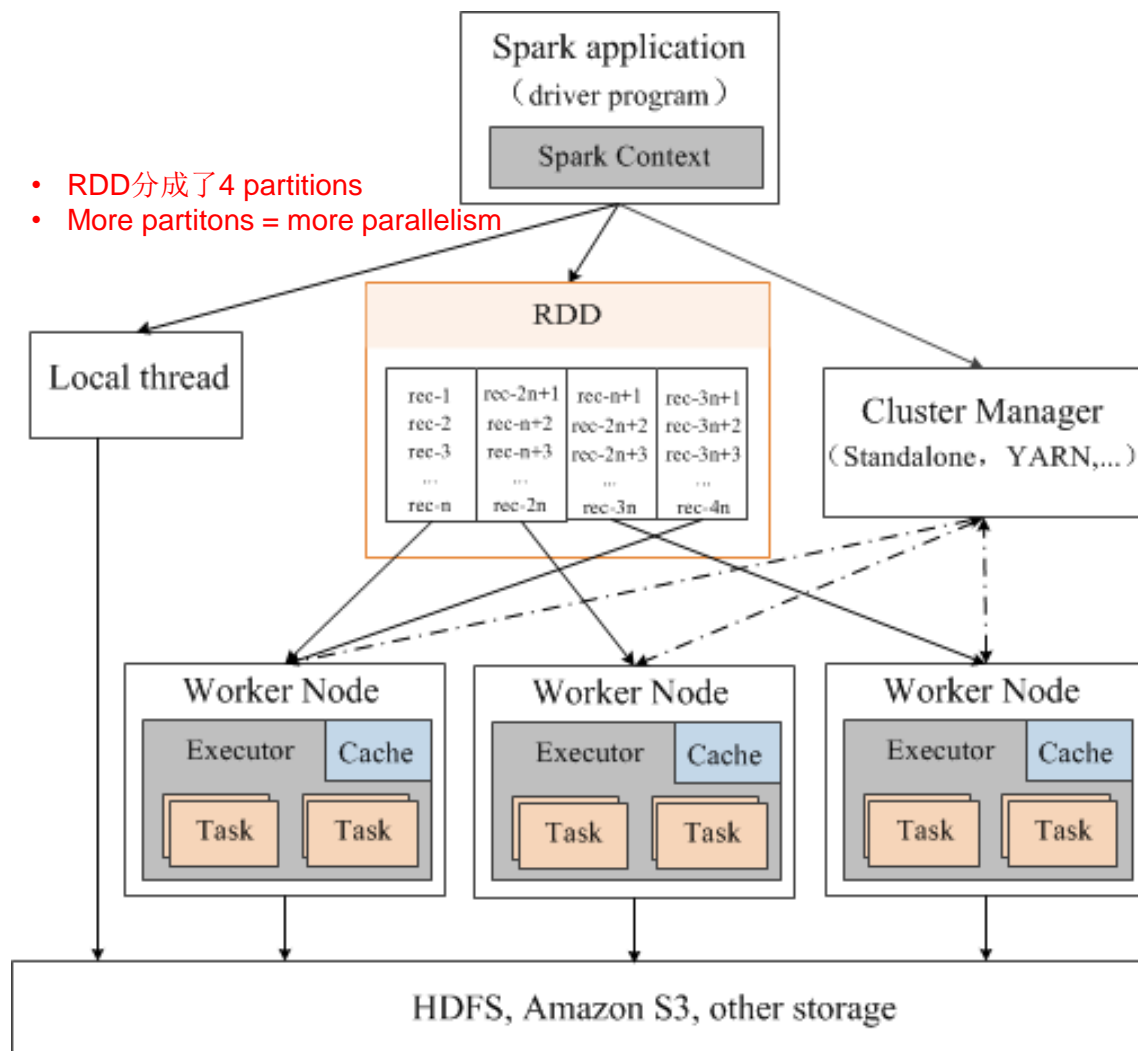
# RDDs的基本操作

- ✓ RDDs有两类操作，变换（**Transformations**）和执行（**Actions**）
- ✓ 变换（**Transformations**）操作是延迟求值的（Lazy），
- ✓ 只有当执行（**Actions**）操作触发时，RDDs的变换才真正执行
- ✓ RDDs还可以通过**Cache**操作被缓存在内存中

- Create an RDD from a data source:  <list>
- Apply transformations to an RDD: map filter
- Apply actions to an RDD: collect count

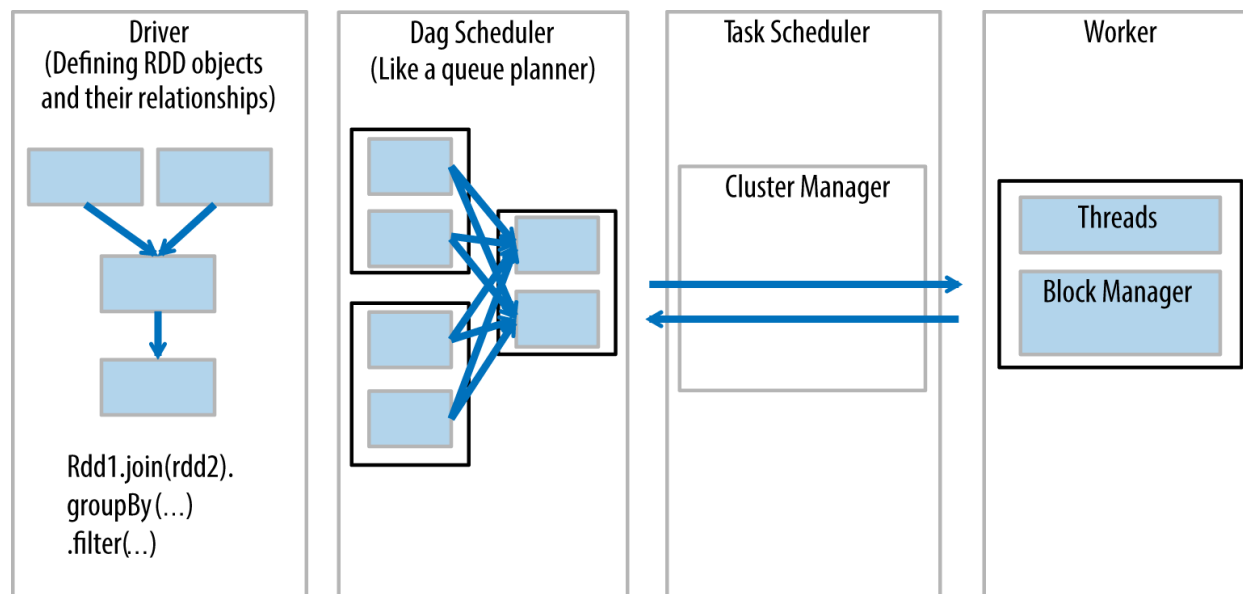


# Spark的编程模型



- ✓ 用户编写的Spark应用程序，称为**驱动程序 (driver program)**，其负责将Spark应用程序表示成高层的控制流（隐式地创建了DAG）；
- ✓ driver 程序中，用户可定义RDD的转换（Transformations）和执行（Actions）操作，这些操作在集群的worker节点上执行；
- ✓ 在driver中，一个SparkContext（SC）对象被创建，SC可连接各类集群管理器（如YARN、Mesos）；
- ✓ SC连接集群管理器，集群启动各个Worker节点中的Spark Executor，driver程序将代码和任务（Task）传给Executor，由Executor在**内存**中对RDDs执行各种操作；
- ✓ Spark作业（Job）中任务完成后，将数据写回文件系统。

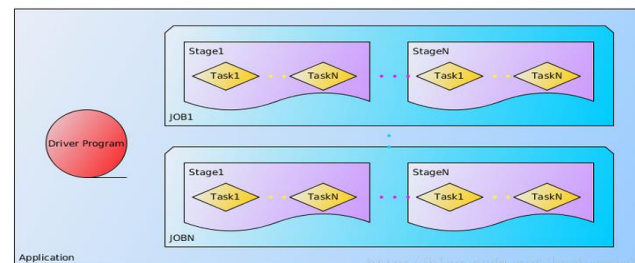
# Spark的组件



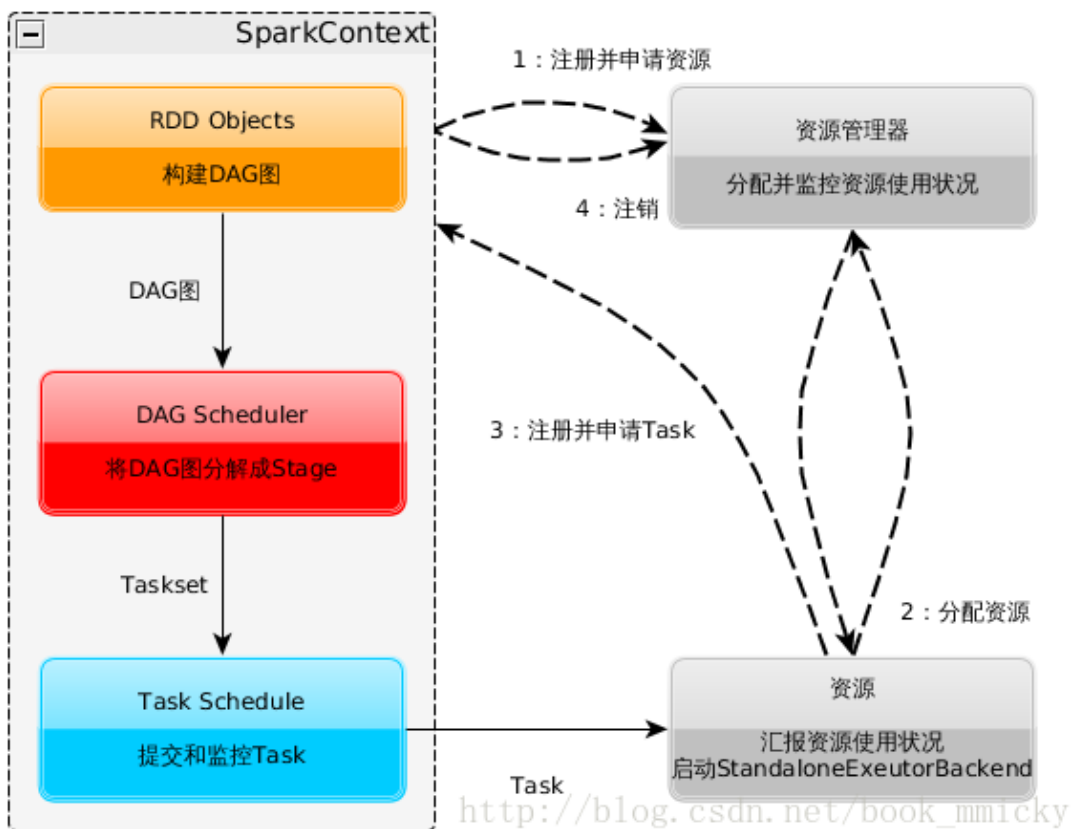
- ✓ **驱动程序**指的是包含main 函数的代码，它定义了RDD及其变换，驱动程序在Master节点中运行
- ✓ 基于RDD 的并行操作会传递给**DAG 调度器 (DAG scheduler)**，后者可以优化代码，并产生一个代表应用中数据处理步骤的高效DAG
- ✓ 最终的DAG 会发送给**集群管理器 (cluster manager)**。集群管理器知晓Worker 的信息、已分配的线程、数据块的位置等，具有分配特定处理任务到Worker 的功能
- ✓ **Worker**接受和管理分配的任务和数据。Worker 执行自身的特定任务时，并不知晓整个DAG，执行的结果会回传给驱动程序。

# Spark的运行架构——主要概念

- ✓ **Application**: 基于Spark的用户程序，包含了一个driver program 和 集群中多个的executor
- ✓ **Driver Program**: 运行Application的main()函数并且创建SparkContext
- ✓ **Executor**: 是为某Application运行在worker 上的一个进程，该进程负责运行Task，并且负责将数据存在内存或者磁盘上
- ✓ **Cluster Manager**: 在集群上获取资源的外部服务(例如: Standalone、Mesos、Yarn)
- ✓ **Worker Node**: 集群中任何可以运行Application代码的节点
- ✓ **Task**: 被送到某个executor上的工作单元
- ✓ **Job**: 包含多个Task组成的并行计算，往往由Spark Action催生，该术语可以经常在日志中看到。
- ✓ **Stage**: 每个Job会被拆分很多组task，每组任务被称为Stage，也可称TaskSet，该术语可以经常在日志中看到。
- ✓ **RDD**: Spark的基本计算单元，可以通过一系列算子进行操作（主要有Transformation和Action操作）
- ✓ **DAG Scheduler**: 根据Job构建基于Stage的DAG，并提交Stage给TaskScheduler
- ✓ **TaskScheduler**: 将Taskset提交给workers运行并回报结果

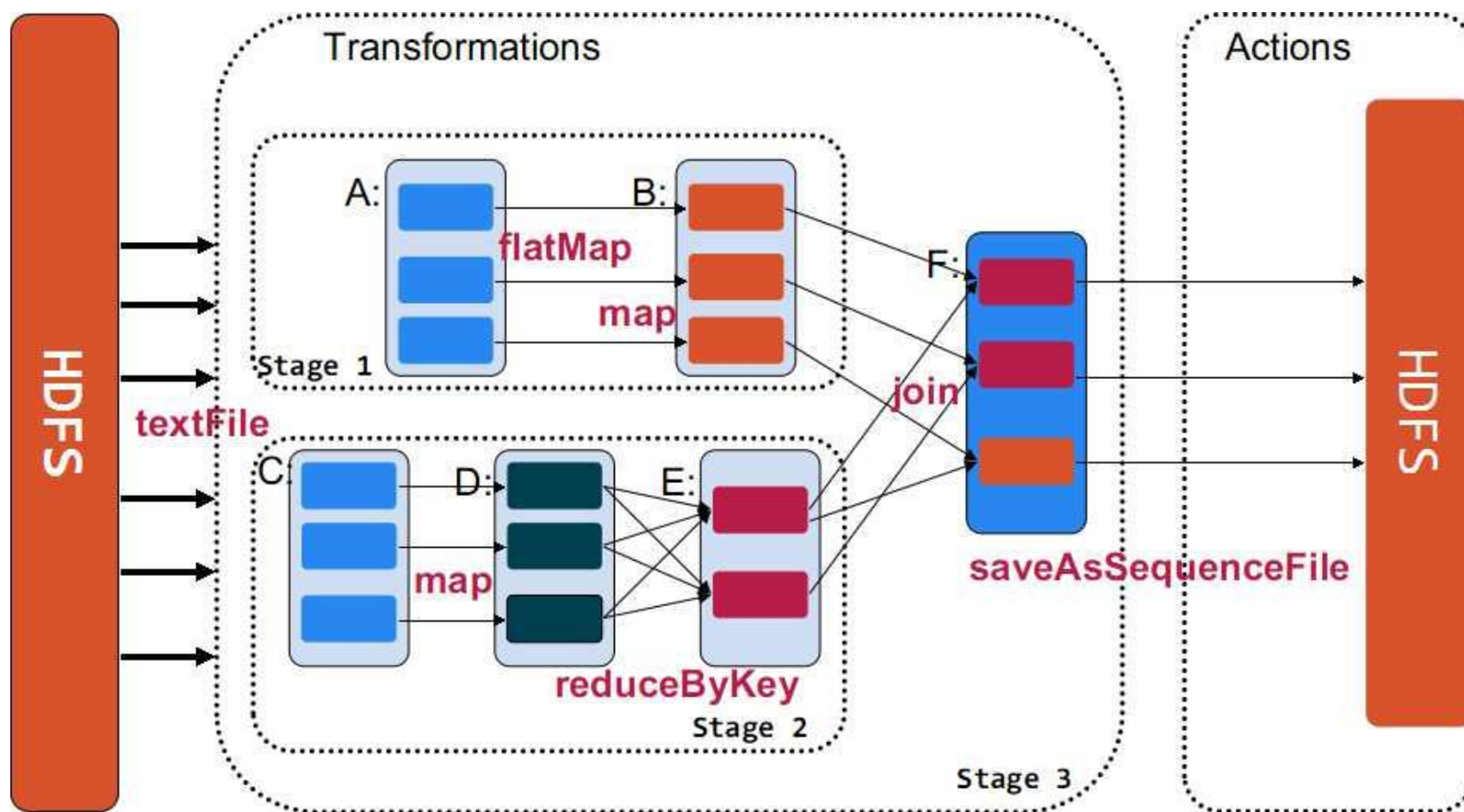


# Spark的运行架构



- ✓ 构建Spark Application的运行环境（启动SparkContext）
- ✓ SparkContext向资源管理器（可以是Standalone、Mesos、Yarn）申请运行Executor资源，并启动Standalone ExecutorBackend，executor向SparkContext申请Task。
- ✓ SparkContext将应用程序代码发放给executor
- ✓ SparkContext构建成DAG图、将DAG图分解成Stage、将Taskset发送给Task Scheduler、最后由Task Scheduler将Task发放给Executor运行。
- ✓ Task在Executor上运行，运行完毕释放所有资源。

# Spark的运行架构





# Quiz

---

■ 在Spark任务处理中，Master节点充当哪些角色：

- A.任务与资源调度
- B.节点管理
- C.执行Executor进程
- D.创建RDD Graph

答案：A、B、D

# 目录

---

1

Spark简介

2

Spark RDD、编程模型及运行架构

3

Spark RDD的操作及应用实例

4

Spark SQL与DataFrame

5

Spark SQL应用实例

# RDDs的创建

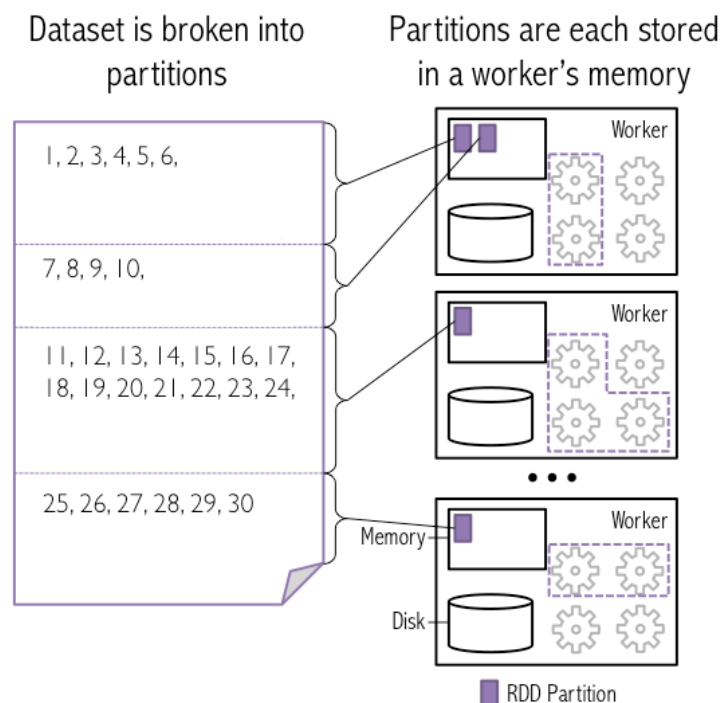
- From HDFS, text files, Hypertable, Amazon S3, Apache Hbase, SequenceFiles, any other Hadoop InputFormat

```
>> distFile = sc.textFile("README.md", 4)
```

- Create RDDs from Python collections (lists)

```
>> data = xrange(1, 31)
```

```
>> xrangeRDD = sc.parallelize(data, 4)
```



# RDDs的变换 ( Transformations )

- 输入分区与输出分区—**一对一型**

✓ `map (func)`

✓ `flatMap (func)`

```
>>> rdd = sc.parallelize([1, 2, 3, 4])
>>> rdd.map(lambda x: x * 2)
RDD: [1, 2, 3, 4] → [2, 4, 6, 8]
```

Function literals (green)  
are closures automatically  
passed to workers

```
>>> rdd = sc.parallelize([1, 2, 3])
>>> rdd.Map(lambda x: [x, x+5])
RDD: [1, 2, 3] → [[1, 6], [2, 7], [3, 8]]
```

```
>>> rdd.flatMap(lambda x: [x, x+5])
RDD: [1, 2, 3] → [1, 6, 2, 7, 3, 8]
```

✓ `mapPartitions (func)`

Review: Python `lambda` Functions

- Small anonymous functions (not bound to a name)  
`lambda a, b: a + b`  
» returns the sum of its two arguments
- Can use lambda functions wherever function objects are required
- Restricted to a single expression

## 一些Value型的Transformations

Transformation	Description
<code>map(func)</code>	return a new distributed dataset formed by passing each element of the source through a function <code>func</code>
<code>filter(func)</code>	return a new dataset formed by selecting those elements of the source on which <code>func</code> returns true
<code>distinct([numTasks])</code>	return a new dataset that contains the distinct elements of the source dataset
<code>flatMap(func)</code>	similar to <code>map</code> , but each input item can be mapped to 0 or more output items (so <code>func</code> should return a Seq rather than a single item)

# RDDs的变换 ( Transformations )

- 输入分区为输出分区子集型

✓ filter(func)

✓ distinct()

```
>>> rdd = sc.parallelize([1, 2, 3, 4])  
>>> rdd.map(lambda x: x * 2)  
RDD: [1, 2, 3, 4] → [2, 4, 6, 8]
```

Function literals (green)  
are closures automatically  
passed to workers

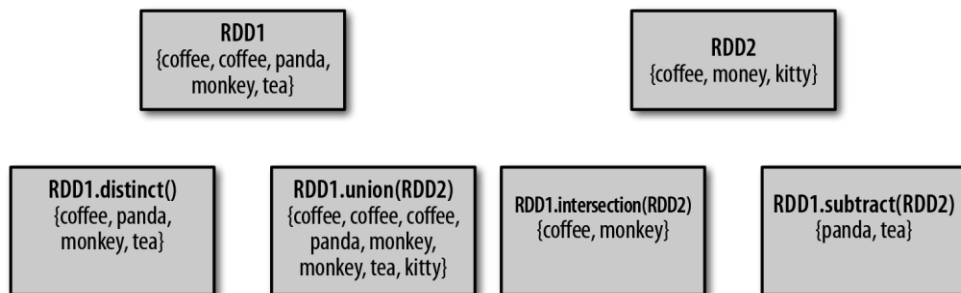
```
>>> rdd.filter(lambda x: x % 2 == 0)  
RDD: [1, 2, 3, 4] → [2, 4]
```

```
>>> rdd2 = sc.parallelize([1, 4, 2, 2, 3])  
>>> rdd2.distinct()  
RDD: [1, 4, 2, 2, 3] → [1, 4, 2, 3]
```

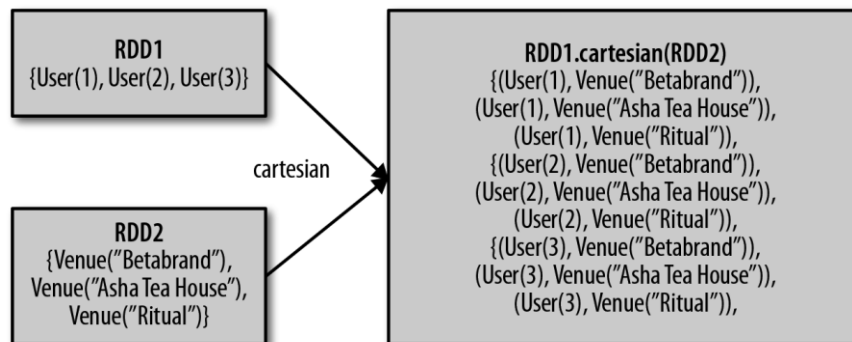
# RDDs的变换 ( Transformations )

- 输入分区和输出分区多对一型

✓ `union(otherDataset)`



✓ `cartesian(otherDataset)`



# RDDs的变换 ( Transformations )

- **Key-Value型** Transformations ( 一个数据源的**聚集**操作 )

## ✓ reduceByKey(func)

```
>>> rdd = sc.parallelize([(1,2), (3,4), (3,6)])
>>> rdd.reduceByKey(lambda a, b: a + b)
RDD: [(1,2), (3,4), (3,6)] → [(1,2), (3,10)]
```

## ✓ sortByKey([ascending], [numTasks])

```
>>> rdd2 = sc.parallelize([(1,'a'), (2,'c'), (1,'b')])
>>> rdd2.sortByKey()
RDD: [(1,'a'), (2,'c'), (1,'b')] →
      [(1,'a'), (1,'b'), (2,'c')]
```

## ✓ groupByKey ()

```
>>> rdd2 = sc.parallelize([(1,'a'), (2,'c'), (1,'b')])
>>> rdd2.groupByKey()
RDD: [(1,'a'), (1,'b'), (2,'c')] →
      [(1,['a','b']), (2,['c'])]
```

Be careful using **groupByKey()** as it can cause a lot of data movement across the network and create large Iterables at workers

Key-Value Transformation	Description
<code>reduceByKey(func)</code>	return a new distributed dataset of (K,V) pairs where the values for each key are aggregated using the given reduce function <i>func</i> , which must be of type (V,V) → V
<code>sortByKey()</code>	return a new dataset (K,V) pairs sorted by keys in ascending order
<code>groupByKey()</code>	return a new dataset of (K, Iterable<V>) pairs

# RDDs的变换 ( Transformations )

- **Key-Value型** Transformations ( 两个数据源的连接操作 )

- ✓ join(otherDataset, [numTasks])

- [X.join\(Y\)](#)

- » Return RDD of all pairs of elements with matching keys in X and Y
    - » Each pair is (k, (v1, v2)) tuple, where (k, v1) is in X and (k, v2) is in Y

```
>>> x = sc.parallelize([("a", 1), ("b", 4)])
>>> y = sc.parallelize([("a", 2), ("a", 3)])
>>> sorted(x.join(y).collect())
```

```
Value: [('a', (1, 2)), ('a', (1, 3))]
```

- ✓ leftOuterJoin(otherDataset, [numTasks])

- ✓ rightOuterJoin(otherDataset, [numTasks])

- ✓ fullOuterJoin(otherDataset, [numTasks])

- [X.fullOuterJoin\(Y\)](#)

- » For each element (k, v) in X, resulting RDD will either contain
      - All pairs (k, (v, w)) for w in Y, or (k, (v, None)) if no elements in Y have k
    - » For each element (k, v) in Y, resulting RDD will either contain
      - All pairs (k, (v, w)) for v in X, or (k, (None, w)) if no elements in X have k

```
>>> x = sc.parallelize([("a", 1), ("b", 4)])
>>> y = sc.parallelize([("a", 2), ("c", 8)])
>>> sorted(x.fullOuterJoin(y).collect())
```

```
Value: [('a', (1, 2)), ('b', (4, None)), ('c', (None, 8))]
```



# RDDs的执行 ( Actions )

- 触发Spark执行数据处理，并从Spark中获得结果

✓ `reduce(func)`

✓ `take(n)`

✓ `collect()`

## Getting Data Out of RDDs

```
>>> rdd = sc.parallelize([1, 2, 3])
>>> rdd.reduce(lambda a, b: a * b)
Value: 6
```

```
>>> rdd.take(2)
Value: [1,2] # as list
```

```
>>> rdd.collect()
Value: [1,2,3] # as list
```

✓ `takeOrdered(func)`

```
>>> rdd = sc.parallelize([5,3,1,2])
>>> rdd.takeOrdered(3, lambda s: -1 * s)
Value: [5,3,2] # as list
```

## Spark Actions

- Cause Spark to execute recipe to transform source
- Mechanism for getting results out of Spark

Action	Description
<code>reduce(func)</code>	aggregate dataset's elements using function <i>func</i> . <i>func</i> takes two arguments and returns one, and is commutative and associative so that it can be computed correctly in parallel
<code>take(n)</code>	return an array with the first <i>n</i> elements
<code>collect()</code>	return all the elements as an array <b>WARNING: make sure will fit in driver program</b>
<code>takeOrdered(n, key=func)</code>	return <i>n</i> elements ordered in ascending order or as specified by the optional key function

# Spark 支持的操作

<b>Transformations</b> (define a new RDD)	map filter sample groupByKey reduceByKey sortByKey	flatMap union join cogroup cross mapValues
<b>Actions</b> (return a result to driver program)	collect reduce count save lookupKey	

## Spark Programming Guide

<http://spark.apache.org/docs/latest/programming-guide.html#rdd-operations>

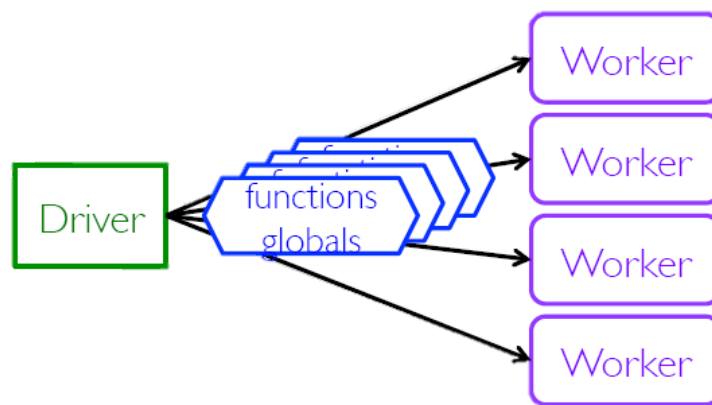
## Pyspark RDD 操作实例

<http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD>

# Demos in Zeppelin with pySpark

# Spark 中闭包 ( Clousures )

- Spark自动创建闭包：
  - ✓ Worker节点运行在RDDs上的函数
  - ✓ Worker节点使用到的任何全局变量



- Spark会自动将闭包中引用到的变量发送到工作节点上：
  - ✓ Spark会将闭包发送到每个task
  - ✓ worker之间不发生通讯
  - ✓ 在每个worker中对全局变量的修改信息并不发送给driver

**closure闭包:** 如果在一个内部函数里，对在外部作用域（但不是在全局作用域）的变量进行引用，那么内部函数就被认为是闭包。

# Spark 中闭包 ( Clousures ) ——考虑下面的情况

- Job需要非常大的全局变量(large global variables) :
  - ✓ 发送large read-only 查找表 (Lookup table) 给workers节点
  - ✓ 在机器学习算法中, 发送large 特征向量(Feature vector)给workers节点
- Job执行过程中发生的事件计数 :
  - ✓ 多少输入的行是空行
  - ✓ 多少输入记录中有脏数据 (数据不完整)

## Problems:

- Closures are (re-)sent with **every** job
- Inefficient to send large data to each worker
- Closures are one way: driver → worker

# Spark 中的共享变量



- Broadcast Variables(广播变量) :

- ✓ 让程序高效地向所有工作节点发送一个较大的只读值
- ✓ 这个广播变量可供一个或多个Spark操作使用
- ✓ 例如向所有worker节点发送一个很大的只读查询表，或机器学习算法中一个很大的特征向量



- Accumulators ( 累加器 )

- ✓ 累加器提供了将工作节点的值集合到driver程序的简单语法
- ✓ 只有driver可以访问累加器的值
- ✓ 对于tasks，累加器是write-only（只写变量）

- 累加器用来对信息进行聚集，广播变量用来高效分发较大的对象

# Spark广播变量



- Broadcast Variables(广播变量) :

- ✓ 在每一个worker中缓存一份只读的变量

- ✓ 采用高效的广播算法

- 一个简单的例子

At the driver:

```
>>> broadcastVar = sc.broadcast([1, 2, 3])
```

At a worker (in code passed via a closure)

```
>>> broadcastVar.value
```

```
[1, 2, 3]
```

- 实际应用案例 ( 处理无线电操作者的呼叫日志 )

```
def loadCallSignTable():
```

```
    f = open("./files/callsign_tbl_sorted", "r")
```

```
    return f.readlines()
```

```
signPrefixes = sc.broadcast(loadCallSignTable())
```

```
def processSignCount(sign_count, signPrefixes):
```

```
    country = lookupCountry(sign_count[0], signPrefixes.value)
```

```
    count = sign_count[1]
```

```
    return (country, count)
```

```
countryContactCounts = (contactCounts
```

```
    .map(lambda signCount: processSignCount(signCount, signPrefixes))
```

```
    .reduceByKey((lambda x, y: x + y)))
```

```
countryContactCounts.saveAsTextFile(outputDir + "/countries.txt")
```

- ✓ 从文件中载入查找表

- ✓ 该查找表可根据呼号，返回国家

- ✓ 将查找表定义为广播变量



# Spark 累加器



- Accumulators ( 累加器 ) :

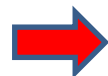
- ✓ 用来高效地实现并行计数或求和

- ✓ 只有driver可以读累加器的值

- ✓ 累加器中变量的操作需要满足交换律和结合律

- 实践应用案例 ( 对无线电操作者的呼叫日志中的空行进行计数 )

```
file = sc.textFile(inputFile)
# Create Accumulator[Int] initialized to 0
blankLines = sc.accumulator(0)
```



创建累加器，并初始化为0

```
def extractCallSigns(line):
    global blankLines # Make the global variable accessible
    if (line == ""):
        blankLines += 1
    return line.split(" ")
```

```
callSigns = file.flatMap(extractCallSigns)
print "Blank lines: %d" % blankLines.value
```

# Spark 日志分析——Apache Common Log Format

- Apache Common Log Format <http://httpd.apache.org/docs/2.4/logs.html#common>

field	实例	meaning
remotehost	127.0.0.1	Remote hostname (or IP number if DNS hostname is not available)- (远程主机名或IP)
rfc931	-	The remote logname of the user. We don't really care about this field. (远程登录的用户名)
authuser	-	The username of the remote user, as authenticated by the HTTP server. (经过HTTP服务器验证过的远程用户名)
[date]	[10/Oct/2000:13:55:36 -0700]	The date and time of the request.(请求的日期与时间)
"request"	"GET /apache_pb.gif HTTP/1.0"	The request, exactly as it came from the browser or client. (用户请求)
status	200	The HTTP status code the server sent back to the client. (返回用户的HTTP状态码)
bytes	2326	The number of bytes (Content-Length) transferred to the client. (传输给客户端的字节数)

- Apache Web Server Log

198.213.130.253 - - [03/Aug/1995:11:29:02 -0400] "GET /images/USA/logosmall.gif HTTP/ 1.0 " 304 20  
uplherc.upl.com - - [01/Aug/1995:00:00:10 -0400] "GET /images/WORLD/logosmall.gif HTTP/1.0 " 304 15

.....

(tampico.usc.edu) - - ([14/Aug/1995:22:57:13 - 0400]) "(GET) (/welcome.html) (HTTP/1.0) " (200) (790)

host date\_time method endpoint protocol response\_code content\_size



# Spark 日志分析——Spark杀手级应用（通用需求）

---

- 日志总体情况（Overall）
  - ✓ 服务器返回内容的统计信息？文件大小
  - ✓ 返回码的类型
  - ✓ 有多少404（找不到页面）错误
  - ✓ .....
- 特定时间上的状态分析（Temporal）
  - ✓ 每天有多少独立主机/IP来访问
  - ✓ 一个月内每天有多少请求
  - ✓ 每天有多少404错误
  - ✓ .....

# Spark 日志分析——Python 正则表达式

(tampico.usc.edu) ( - ) ( - ) ([14/Aug/1995:22:57:13 -0400])

host clientID userID date\_time

“(GET) (/welcome.html) (HTTP/1.0) ” (200) (790)

method endpoint protocol response\_code content\_size

```
APACHE_ACCESS_LOG_PATTERN = '^(\S+) (\S+) (\S+) \[([^\w:/]+\s[+-]\d{4})\] "(\S+) (\S+)\s*(\S*\s*)" (\d{3}) (\S+)'
```

host clientID userID

date\_time

method endpoint protocol response\_code content\_size

^ —表示从起始位置开始匹配

\S+ —表示匹配一个或多个非空字符；\s表示匹配空格

\w —表示匹配单词字符

[\w:/]+ —表示匹配单词字符或’:’或’/’

[+/-] —表示匹配+ 或 -

\d{4} —表示匹配4位数字

## Python 正则表达式

<https://docs.python.org/3.5/library/re.html>

# Spark 日志分析——日志预处理函数（清洗）

```
APACHE_ACCESS_LOG_PATTERN = '^(\S+) (\S+) (\S+) \[([w:/]+\s[+-]\d{4})\] \"(\S+) (\S+)\s*(\S*\s*)\" (\d{3}) (\S+)'
```

```
def parseApacheLogLine(logline):  
    """ Parse a line in the Apache Common Log format  
    Args:  
        logline (str): a line of text in the Apache Common Log format  
    Returns:  
        tuple: either a dictionary containing the parts of the Apache Access Log and 1,  
               or the original invalid log line and 0  
    """
```

```
    match = re.search(APACHE_ACCESS_LOG_PATTERN, logline)  
    if match is None:  
        return (logline, 0)  
    size_field = match.group(9)  
    if size_field == '-':  
        size = long(0)  
    else:  
        size = long(match.group(9))  
    return (Row(  
        host          = match.group(1),  
        client_id     = match.group(2),  
        user_id       = match.group(3),  
        date_time     = parse_apache_time(match.group(4)),  
        method        = match.group(5),  
        endpoint       = match.group(6),  
        protocol       = match.group(7),  
        response_code = int(match.group(8)),  
        content_size  = size  
    ), 1)
```

## 数据清洗

- ✓ 去除“错误”的日志
- ✓ 将内容为“-”转化为0

## 数据封装

- ✓ 将用正则表达式提取出来的字段，封装在Row对象中
- ✓ 返回元组 (row(...), 1) 或 (logline, 0)
- ✓ Row对象本质是定长的字段数组，可通过“.”获取字段的值

# Spark 日志分析——日志加载与预处理

```
logFile = "/tutorial/input/access_log_Aug95"
# The log files look like this:
# `127.0.0.1 - - [01/Aug/1995:00:00:01 -0400] "GET /images/launch-logo.gif HTTP/1.0" 200 1839`
```

```
def parseLogs():
    """ Read and parse log file """
    parsed_logs = (sc
        .textFile(logFile,3)
        .map(parseApacheLogLine)
        .cache())

    access_logs = (parsed_logs
        .filter(lambda s: s[1] == 1)
        .map(lambda s: s[0])
        .cache())

    failed_logs = (parsed_logs
        .filter(lambda s: s[1] == 0)
        .map(lambda s: s[0]))
    failed_logs_count = failed_logs.count()

    if failed_logs_count > 0:
        print 'Number of invalid logline: %d' % failed_logs.count()
        for line in failed_logs.take(10):
            print 'Invalid logline: %s' % line

    return parsed_logs, access_logs, failed_logs
```

```
parsed_logs, access_logs, failed_logs = parseLogs()
```

## 数据加载与预处理

- ✓ 从HDFS里面加载日志，并行度设为3
- ✓ 调用`parseApacheLogLine`函数，并缓存

## 正确的日志记录

## 错误的日志记录

## 返回RDD

# Spark 日志分析——日志总体信息的统计

- 服务器返回内容的统计信息

```
# Calculate statistics based on the content size.
content_sizes = access_logs.map(lambda log: log.content_size).cache()
print 'Content Size Avg: %i, Min: %i, Max: %s' % (
    content_sizes.reduce(lambda a, b : a + b) / content_sizes.count(),
    content_sizes.min(),
    content_sizes.max())
```

Content Size Avg: 17097, Min: 0, Max: 3421948

Took 1 min 25 sec. Last updated by cmg at March 08 2017, 3:57:47 PM.

- 返回码的类型

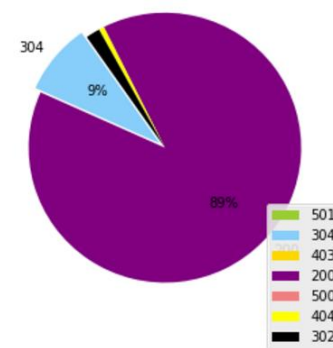
```
# Response Code to Count
responseCodeToCount = (access_logs
    .map(lambda log: (log.response_code, 1))
    .reduceByKey(lambda a, b : a + b)
    .cache())

responseCodeToCountList = responseCodeToCount.take(100)
print 'Found %d response codes' % len(responseCodeToCountList)
print 'Response Code Counts: %s' % responseCodeToCountList
```

Found 7 response codes

Response Code Counts: [(501, 27), (304, 134138), (403, 171), (200, 1398317), (500, 3), (404, 10020), (302, 26440)]

Took 45 sec. Last updated by cmg at March 08 2017, 4:00:03 PM.



Took 46 sec. Last updated by cmg at March 08 2017, 4:01:45 PM.

# Spark 日志分析——日志总体信息的统计

- 有多少404（找不到页面）错误

```
#Counting 404 Response Codes
badRecords = (access_logs
               .filter(lambda log: log.response_code==404).cache())
print 'Found %d 404 URLs' % badRecords.count()
```

- 列出404错误的被访问Endpoints

```
#Listing 404 Response Code Endpoints
badEndpoints = badRecords.map(lambda log:log.endpoint).cache()
badUniqueEndpoints = badEndpoints.distinct()
badUniqueEndpointsPick40 = badUniqueEndpoints.take(40)
print '404 URLs: %s' % badUniqueEndpointsPick40
```

- 列出404错误最多的前20个Endpoints

```
#List the Top 20 404 Response Code Endpoints
badEndpointsCountPairTuple = badRecords.map(lambda log:(log.endpoint,1))
badEndpointsSum = badEndpointsCountPairTuple.reduceByKey(lambda a,b:a+b)
badEndpointsTop20 = badEndpointsSum.takeOrdered(20,lambda s : -1*s[1])
print 'Top Twenty 404 URLs: %s' % badEndpointsTop20
```

# Spark 日志分析——特定时间上状态分析 ( Temporal )

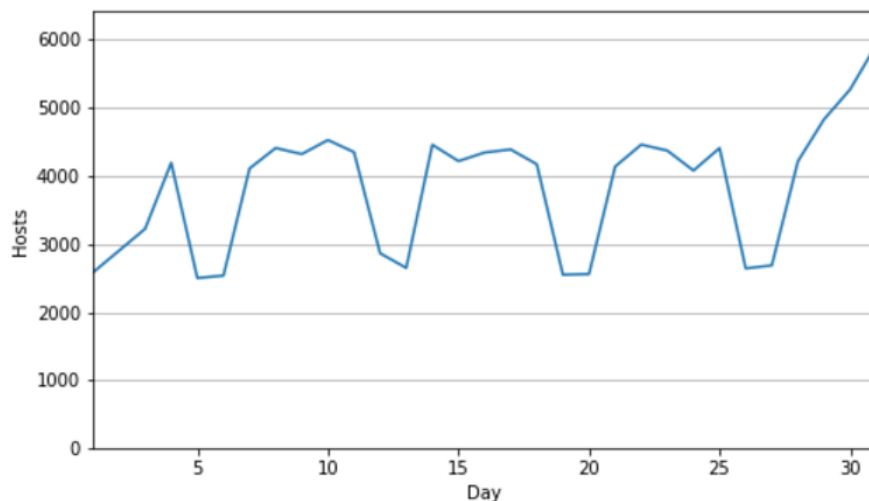
- 每天独立主机访问数

```
#Number of Unique Daily Hosts
dayToHostPairTuple = access_logs.map(lambda log: (log.date_time.day, log.host)).distinct().cache()

dayGroupedHosts = dayToHostPairTuple.groupByKey()

dayHostCount = dayGroupedHosts.map(lambda k: (k[0], len(k[1])))
dailyHosts = (dayHostCount.sortByKey()).cache()

dailyHostsList = dailyHosts.take(30)
print 'Unique hosts per day: %s' % dailyHostsList
```



# Spark VS Hadoop

---

■ Spark数据放内存，对于迭代运算效率更高

■ Spark比Hadoop更通用

- ✓ Transformations操作：map, filter, flatMap, sample, groupByKey, reduceByKey, union, join, cogroup, mapValues, sort, partitionBy
- ✓ actions操作：Count, collect, reduce, lookup, save
- ✓ 处理节点之间的通信模型不再像Hadoop那样就是唯一的Data Shuffle一种模式。
- ✓ 可以命名，物化，控制中间结果的存储、分区等

■ Spark比Hadoop可用性更高

- ✓ 支持丰富的Scala, Java, Python、R等语言及API



# Quiz

---

■ 以下针对RDD的操作中，属于Transformation的操作有哪些？

- A. map()
- B. filter()
- C. count()
- D. union()

答案：A、B、D

# Quiz

---

■ 以下针对RDD的操作中，属于Action的操作有哪些？

- A. collect()
- B. reduce()
- C. save()
- D. cogroup()

答案：A、B、C

# 目录

---

1

Spark简介

2

Spark RDD、编程模型及运行架构

3

Spark RDD的操作及应用实例

4

**Spark SQL与DataFrame**

5

Spark SQL应用实例

# 什么是Spark SQL

- Apache Spark SQL是Spark用来操作结构化与半结构化数据的接口
- 从历史上来讲，Spark SQL的前身是Shark（SQL on Spark），Shark之于Spark就相当于Hive之于Hadoop MapReduce
- Spark SQL提供了一种特殊的RDD，早些版本称为SchemaRDD，从Spark 1.3.0以后的版本，SchemaRDD已被一个称为DataFrames的编程抽象取代，DataFrames可以充当分布式SQL查询引擎
- Spark SQL：Relational Data Processing in Spark，SIGMOD 2015.



**Michael Armbrust**  
Software Engineer at Databricks  
San Francisco Bay Area | Computer Software

500+ connections

Current: Databricks  
Previous: Google, Reliable Adaptive Distributed Systems Laboratory, Standard Performance Evaluation Corporation (SPEC)  
Education: University of California, Berkeley

# Spark SQL 动机与解决方案

## 动机

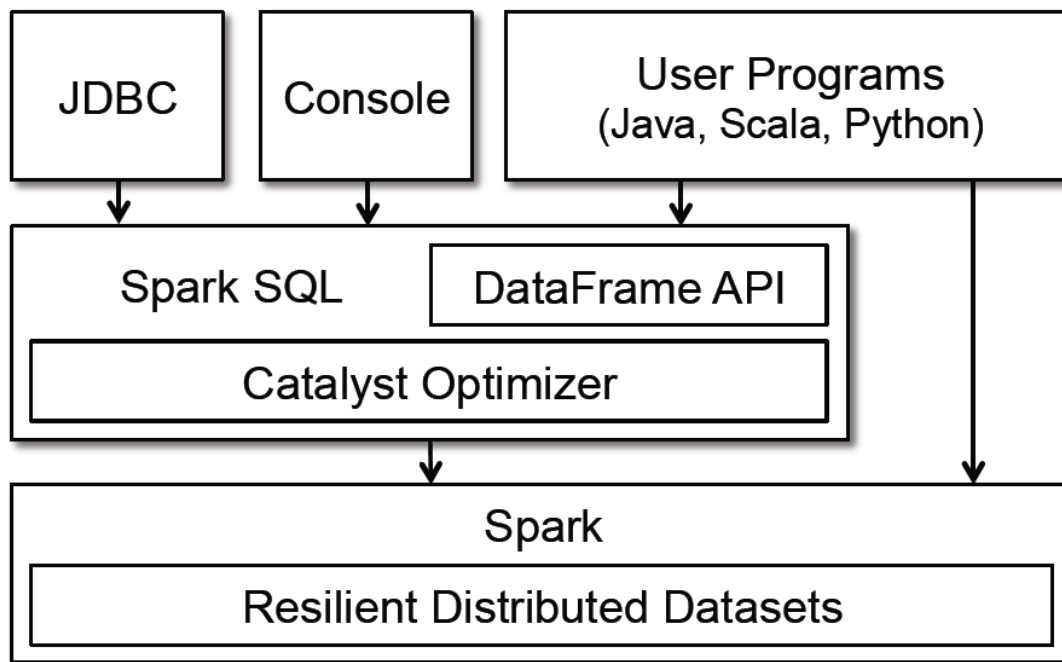
- 多数的数据处理流程既需要关系型操作，又需要过程性操作，比如：
  - ✓ 用户一方面需要关系型操作（如数据查询、统计分析）
  - ✓ 另一方面又需要过程式的操作（如ETL、机器学习与图分析等高级数据处理）
- 然而，遗憾的的是这两类操作常常是割裂的

## 解决方案（两大贡献）

- **DataFrame API** ——将关系型的处理与过程型处理结合起来，可以对外部数据源（Hive、JSON等）和Spark内建的分布式集合（RDD）进行关系型操作。
- **Catalyst** ——一个可扩展的SQL 查询优化器，采用了Scala语言的特性（pattern matching, quasiquotes）可增加组合规则、控制代码生成，定义扩展。



# Spark SQL的编程接口



**Figure 1: Interfaces to Spark SQL, and interaction with Spark.**

# Spark SQL中的DataFrame API

---

- 什么是DataFrame

- ✓ DataFrame是Spark SQL的主要数据抽象（RDD是Spark的主要数据抽象），是相同模式行的分布式集合（这些行以命名的列方式组织），等同于关系数据库的表，支持关系操作（selecting, filtering, aggregating, and plotting structured data）。
- ✓ DataFrame 支持从已存在的原生RDD或外部数据源来创建。
- ✓ DataFrame是延迟评估的，只有当用户调用输出操作，如count()，Spark SQL才构建物理计划来执行，在执行过程中采用了优化策略，比如只扫描某些列数据。

# Spark SQL中的DataFrame API

---

- DataFrame的特点

- ✓ DataFrame一旦构建就无法修改
- ✓ DataFrame跟踪血缘信息来有效重计算丢失数据
- ✓ DataFrame分布在worker中，可执行并行操作
- ✓ DataFrame的数据集都是按指定列存储
- ✓ DataFrame提供特定领域的语言（DSL）API来操作数据集



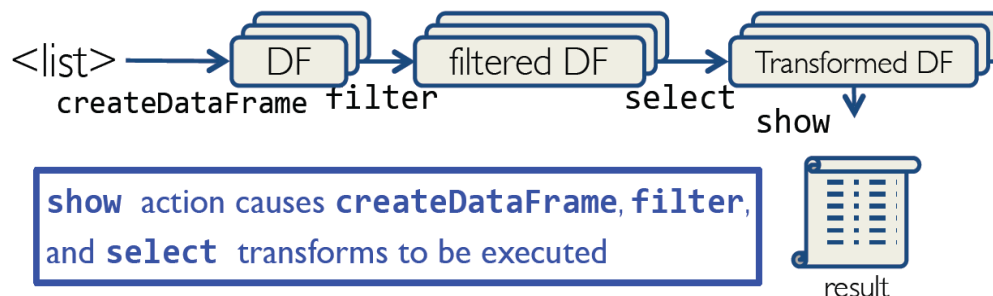
# Spark SQL中的DataFrame API

- DataFrame的操作

- ✓ DataFrame中每一行是Row对象，Row中的字段可以像属性一样被访问

```
>>> row = Row(name="Alice", age=11)
>>> row
Row(age=11, name='Alice')
>>> row['name'], row['age']
('Alice', 11)
>>> row.name, row.age
('Alice', 11)
```

- ✓ DataFrame有两类操作，分别是：transforms和actions（这点与RDD相同）



# Spark SQL中的DataFrame API

- 创建DataFrame—从Python集合（List）或Python数据分析库pandas中创建DataFrame

## ✓ 从Python集合（List）中创建DataFrame

```
>>> data = [('Alice', 1), ('Bob', 2)]

>>> df = sqlContext.createDataFrame(data)
[Row(_1=u'Alice', _2=1), Row(_1=u'Bob', _2=2)]

>>> sqlContext.createDataFrame(data, ['name', 'age'])
[Row(name=u'Alice', age=1), Row(name=u'Bob', age=2)]
```

## ✓ 从Python数据分析库pandas中创建DataFrame

```
# Create a Spark DataFrame from Pandas

>>> spark_df = sqlContext.createDataFrame(pandas_df)
```

# Spark SQL中的DataFrame API

- 创建DataFrame—从文件中创建DataFrame
  - ✓ • Spark SQL支持从HDFS、文本文件、JSON、Parquet、Hypertable、Amazon S3、Hbase、SequenceFiles、任何其它Hadoop InputFormat, 目录或通配符:/data/201404\*中读入数据, 并创建DataFrame

```
>>> df = sqlContext.read.text("README.txt")
```

```
>>> df.collect()
```

```
[Row(value=u'hello'), Row(value=u'this')]
```



Loads text file and returns a DataFrame with a single string column named "value"

Each line in text file is a row

*Lazy evaluation* means no execution happens now

# DataFrame的transformations操作

---

- 使用Select方法从DataFrame中选择一列或多列，返回一个新的DataFrame

```
>>> df.select('*')
    * selects all the columns
>>> df.select('name', 'age')
    * selects the name and age columns
>>> df.select(df.name,
              (df.age + 10).alias('age'))
    * selects the name and age columns,
      increments the values in the age column by 10,
      and renames (alias) the age + 10 column as age
```

## 使用drop方法删除指定的列，返回一个新的DataFrame

```
>>> df.drop(df.age)
[Row(name=u'Alice'), Row(name=u'Bob')]
```

# DataFrame的transformations操作

---

- 使用自定义函数，DataFrame支持自定义函数作transformations

```
>>> from pyspark.sql.types import IntegerType
>>> slen = udf(lambda s: len(s), IntegerType())
>>> df.select(slen(df.name).alias('slen'))
```

\* Creates a DataFrame of [Row(slen=5), Row(slen=3)]

# DataFrame的transformations操作

- 更多DataFrame的Transformations

Transformation	Description
<a href="#"><code>filter(func)</code></a>	returns a new <b>DataFrame</b> formed by selecting those rows of the source on which <i>func</i> returns true
<a href="#"><code>where(func)</code></a>	<b>where</b> is an alias for <b>filter</b>
<a href="#"><code>distinct()</code></a>	return a new <b>DataFrame</b> that contains the distinct rows of the source <b>DataFrame</b>
<a href="#"><code>orderBy(*cols, **kw)</code></a>	returns a new <b>DataFrame</b> sorted by the specified <i>column(s)</i> and in the sort order specified by <i>kw</i>
<a href="#"><code>sort(*cols, **kw)</code></a>	Like <b>orderBy</b> , <b>sort</b> returns a new <b>DataFrame</b> sorted by the specified <i>column(s)</i> and in the sort order specified by <i>kw</i>
<a href="#"><code>explode(col)</code></a>	returns a new row for each element in the given array or map

**func** is a Python named function or **lambda** function

# DataFrame的transformations操作

---

- UDF、select、alias、filter操作

```
>>> df = sqlContext.createDataFrame(data, ['name', 'age'])  
[Row(name=u'Alice', age=1), Row(name=u'Bob', age=2)]
```

```
>>> from pyspark.sql.types import IntegerType  
>>> doubled = udf(lambda s: s *2, IntegerType())  
>>> df2 = df.select(df.name, doubled(df.age).alias('age'))  
[Row(name=u'Alice', age=2), Row(name=u'Bob', age=4)]
```

\* selects the **name** and **age** columns, applies the UDF to **age** column and aliases resulting column to **age**

```
>>> df3 = df2.filter(df2.age > 3)  
[Row(name=u'Bob', age=4)]
```

\* only keeps rows with **age** column greater than 3

# DataFrame的transformations操作

- distinct , sort操作

```
>>> data2 = [('Alice', 1), ('Bob', 2), ('Bob', 2)]
>>> df = sqlContext.createDataFrame(data2, ['name', 'age'])
[Row(name=u'Alice', age=1), Row(name=u'Bob', age=2),
 Row(name=u'Bob', age=2)]
>>> df2 = df.distinct()
[Row(name=u'Alice', age=1), Row(name=u'Bob', age=2)]
* only keeps rows that are distinct
```

```
>>> df3 = df2.sort("age", ascending=False)
[Row(name=u'Bob', age=2),
 Row(name=u'Alice', age=1)]
* sort ascending on the age column
```

- 利用explode将list分裂为行

```
>>> data3 = [Row(a=1, intlist=[1,2,3])]
>>> df4 = sqlContext.createDataFrame(data3)
[Row(a=1, intlist=[1,2,3])]
>>> df4.select(explode(df4.intlist).alias("anInt"))
[Row(anInt=1), Row(anInt=2), Row(anInt=3)]
* turn each element of the intlist column into a Row, alias the resulting
column to anInt, and select only that column
```



# DataFrame Transformation中的聚合操作

- DataFrame通过DataFrame.groupBy()函数实现DataFrame的聚合操作

GrouppedData Function	Description
<a href="#"><code>agg(*exprs)</code></a>	Compute aggregates (avg, max, min, sum, or count) and returns the result as a <b>DataFrame</b>
<a href="#"><code>count()</code></a>	counts the number of records for each group
<a href="#"><code>avg(*args)</code></a>	computes average values for numeric columns for each group

```
>>> data = [('Alice',1,6), ('Bob',2,8), ('Alice',3,9), ('Bob',4,7)]
>>> df = sqlContext.createDataFrame(data, ['name', 'age', 'grade'])
>>> df1 = df.groupBy(df.name)
>>> df1.agg({"*": "count"}).collect()
[Row(name=u'Alice', count(1)=2), Row(name=u'Bob', count(1)=2)]
```

```
>>> df.groupBy(df.name).count()
[Row(name=u'Alice', count=2), Row(name=u'Bob', count=2)]
```

```
>>> data = [('Alice',1,6), ('Bob',2,8), ('Alice',3,9), ('Bob',4,7)]
>>> df = sqlContext.createDataFrame(data, ['name', 'age', 'grade'])
>>> df.groupBy().avg().collect()
[Row(avg(age)=2.5, avg(grade)=7.5)]
```

```
>>> df.groupBy('name').avg('age', 'grade').collect()
[Row(name=u'Alice', avg(age)=2.0, avg(grade)=7.5),
 Row(name=u'Bob', avg(age)=3.0, avg(grade)=7.5)]
```

# DataFrame Transformation中的SQL操作（以join操作为例）

---

- 2个DataFrame的join操作（内连接）

Inner Join – X.join(Y, cols)

—> Return DF of rows with matching cols in both X and Y

```
>>> df = sqlContext.createDataFrame(data, ['name', 'age'])
[Row(name=u'Alice', age=1), Row(name=u'Bob', age=2)]
>>> df2 = sqlContext.createDataFrame(data2, ['name', 'height'])
[Row(name=u'Chris', height=80), Row(name=u'Bob', height=85)]

>>> df.join(df2, 'name')
[Row(name=u'Bob', age=2, height=85)]
```

- 2个DataFrame的outer join操作（外连接）

Outer Join – X.join(Y, cols, 'outer')

—> Return DF of rows with matching cols in either X and Y

```
>>> df.join(df2, 'name', 'outer')
[Row(name=u'Chris', age=None, height=80),
 Row(name=u'Alice', age=1, height=None),
 Row(name=u'Bob', age=2, height=85)]
```

# DataFrame的Action操作

Action	Description
<a href="#"><code>show(n, truncate)</code></a>	prints the first $n$ rows of the <b>DataFrame</b>
<a href="#"><code>take(n)</code></a>	returns the first $n$ rows as a list of <b>Row</b>
<a href="#"><code>collect()</code></a>	return all the records as a list of <b>Row</b> <b>WARNING: make sure will fit in driver program</b>
<a href="#"><code>count()</code></a> <sup>+</sup>	returns the number of rows in this <b>DataFrame</b>
<a href="#"><code>describe(*cols)</code></a>	Exploratory Data Analysis function that computes statistics (count, mean, stddev, min, max) for numeric columns – if no columns are given, this function computes statistics for all numerical columns

+`count` for **DataFrames** is an action, while  
for **GroupedData** it is a transformation

# DataFrame的Action操作

---

- collect ( 返回所有行 )、show ( 打印行 )、count ( 统计行数 )

```
>>> df = sqlContext.createDataFrame(data, ['name', 'age'])
>>> df.collect()
[Row(name=u'Alice', age=1), Row(name=u'Bob', age=2)]
```

```
>>> df.show()
```

```
+-----+-----+
| name|age|
+-----+-----+
| Alice|  1|
|  Bob|  2|
+-----+-----+
```

```
>>> df.count()
2
```

# DataFrame的Action操作

---

- take ( 返回前n行 )、describe ( 计算统计数据 )

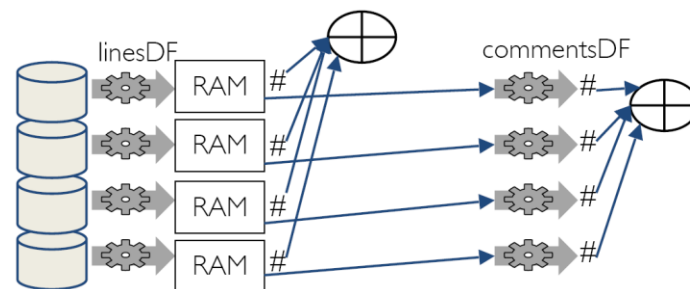
```
>>> df = sqlContext.createDataFrame(data, ['name', 'age'])
>>> df.take(1)
[Row(name=u'Alice', age=1)]
```

```
>>> df.describe()
+-----+-----+
|summary|      age|
+-----+-----+
|  count|         2|
|   mean|        1.5|
| stddev|0.7071067811865476|
|   min|         1|
|   max|         2|
+-----+-----+
```

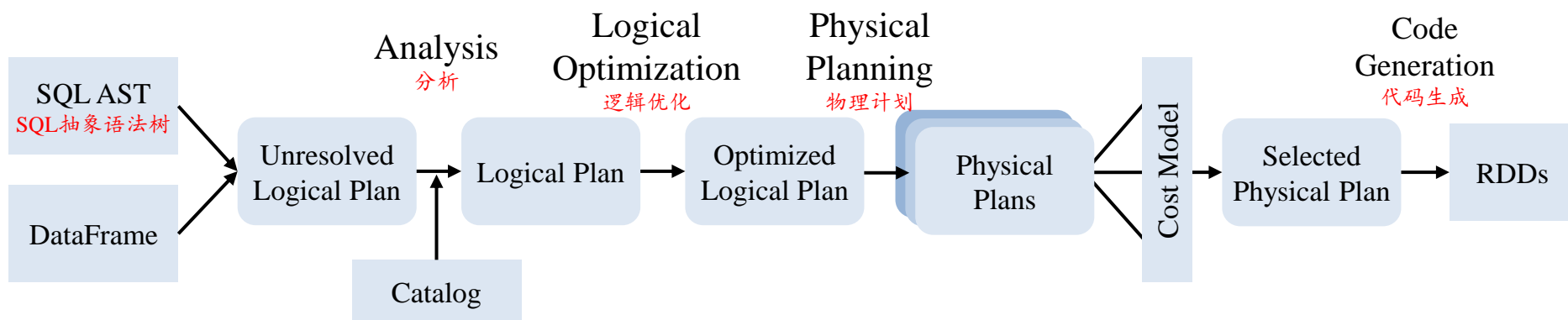
# DataFrame的Cache操作

- Spark SQL可以将热数据物化（Materialize），通常也称作缓存（Cache），以列存储的方式放到内存中，相对于JVM对象，列存储占用内存更少，因为可以用如字典编码（dictionary encoding）或者行程编码（run-length encoding）压缩方式压缩。
- 缓存对于交互式查询和机器学习中迭代算法特别有用，通过调用DataFrame的cache()方法实现。

```
linesDF = sqlContext.read.text('...')
linesDF.cache() # save, don't recompute!
commentsDF = linesDF.filter(isComment)
print linesDF.count(), commentsDF.count()
```



# Catalyst — 计划优化与执行



DataFrames 和 SQL共享相同的优化与执行流程

# 目录

---

1

Spark简介

2

Spark RDD、编程模型及运行架构

3

Spark RDD的操作及应用实例

4

Spark SQL与DataFrame

5

Spark SQL应用实例



# Spark SQL日志分析——日志加载与预处理

```
logFile = "/tutorial/input/access_log_Aug95"
```

```
# The log files look like this:
```

```
# `127.0.0.1 - - [01/Aug/1995:00:00:01 -0400] "GET /images/launch-logo.gif HTTP/1.0" 200 1839`
```

```
base_df = sqlContext.read.text(logFile)
```



数据加载

```
base_df.show(truncate=False)
```



base\_df就是DataFrame

```
+-----+
|value|
+-----+
|in24.inetnebr.com - - [01/Aug/1995:00:00:01 -0400] "GET /shuttle/missions/sts-68/news/sts-68-mcc-05.txt HTTP/1.0" 200 1839|
|uplherc.upl.com - - [01/Aug/1995:00:00:07 -0400] "GET / HTTP/1.0" 304 0|
|uplherc.upl.com - - [01/Aug/1995:00:00:08 -0400] "GET /images/ksclogo-medium.gif HTTP/1.0" 304 0|
|uplherc.upl.com - - [01/Aug/1995:00:00:08 -0400] "GET /images/MOSAIC-logosmall.gif HTTP/1.0" 304 0|
|uplherc.upl.com - - [01/Aug/1995:00:00:08 -0400] "GET /images/USA-logosmall.gif HTTP/1.0" 304 0|
|ix-esc-ca2-07.ix.netcom.com - - [01/Aug/1995:00:00:09 -0400] "GET /images/launch-logo.gif HTTP/1.0" 200 1713|
|uplherc.upl.com - - [01/Aug/1995:00:00:10 -0400] "GET /images/WORLD-logosmall.gif HTTP/1.0" 304 0|
|slppp6.intermind.net - - [01/Aug/1995:00:00:10 -0400] "GET /history/skylab/skylab.html HTTP/1.0" 200 1687|
|piweb4y.prodigy.com - - [01/Aug/1995:00:00:10 -0400] "GET /images/launchmedium.gif HTTP/1.0" 200 11853|
|slppp6.intermind.net - - [01/Aug/1995:00:00:11 -0400] "GET /history/skylab/skylab-small.gif HTTP/1.0" 200 9202|
|slppp6.intermind.net - - [01/Aug/1995:00:00:12 -0400] "GET /images/ksclogosmall.gif HTTP/1.0" 200 3635|
|ix-esc-ca2-07.ix.netcom.com - - [01/Aug/1995:00:00:12 -0400] "GET /history/apollo/images/apollo-logo1.gif HTTP/1.0" 200 1173|
|slppp6.intermind.net - - [01/Aug/1995:00:00:13 -0400] "GET /history/apollo/images/apollo-logo.gif HTTP/1.0" 200 3047|
```

# Spark SQL日志分析——日志加载与预处理

## 利用Spark SQL正则表达式解析日志

```
from pyspark.sql.functions import split, regexp_extract
split_df = base_df.select(regexp_extract('value', r'^([\s]+\s)', 1).alias('host'),
                          regexp_extract('value', r'^.*\[(\d\d\d\d/\d{3}/\d{4}:\d{2}:\d{2} -\d{4})\]', 1).alias('timestamp'),
                          regexp_extract('value', r'^.*"\w+\s+([\s]+\s)\s+HTTP.*"', 1).alias('path'),
                          regexp_extract('value', r'^.*"\s+([\s]+\s)', 1).cast('integer').alias('status'),
                          regexp_extract('value', r'^.*\s+(\d+)$', 1).cast('integer').alias('content_size'))
split_df.show(truncate=False)
```

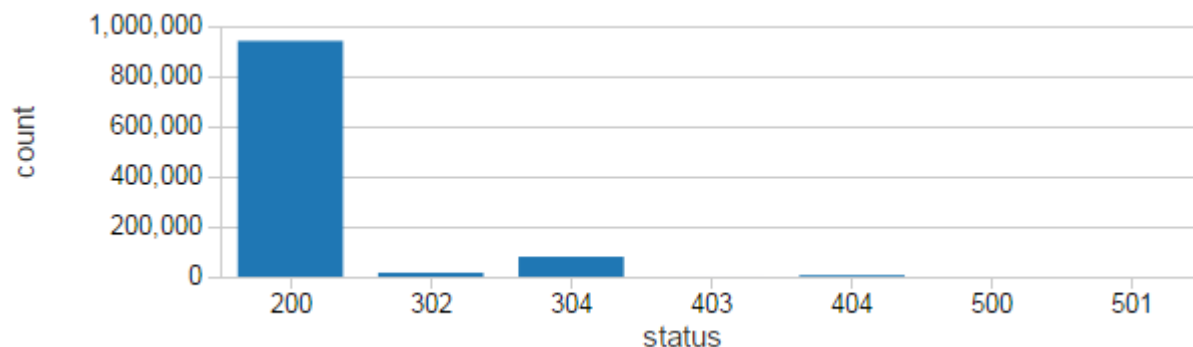
host	timestamp	path	status	content_size
in24.inetnebr.com	01/Aug/1995:00:00:01 -0400	/shuttle/missions/sts-68/news/sts-68-mcc-05.txt	200	1839
uplherc.upl.com	01/Aug/1995:00:00:07 -0400	/	304	0
uplherc.upl.com	01/Aug/1995:00:00:08 -0400	/images/ksclogo-medium.gif	304	0
uplherc.upl.com	01/Aug/1995:00:00:08 -0400	/images/MOSAIC-logosmall.gif	304	0
uplherc.upl.com	01/Aug/1995:00:00:08 -0400	/images/USA-logosmall.gif	304	0
ix-esc-ca2-07.ix.netcom.com	01/Aug/1995:00:00:09 -0400	/images/launch-logo.gif	200	1713
uplherc.upl.com	01/Aug/1995:00:00:10 -0400	/images/WORLD-logosmall.gif	304	0
slppp6.intermind.net	01/Aug/1995:00:00:10 -0400	/history/skylab/skylab.html	200	1687
piweba4y.prodigy.com	01/Aug/1995:00:00:10 -0400	/images/launchmedium.gif	200	11853
slppp6.intermind.net	01/Aug/1995:00:00:11 -0400	/history/skylab/skylab-small.gif	200	9202
slppp6.intermind.net	01/Aug/1995:00:00:12 -0400	/images/ksclogosmall.gif	200	3635
ix-esc-ca2-07.ix.netcom.com	01/Aug/1995:00:00:12 -0400	/history/apollo/images/apollo-logo1.gif	200	1173

# Spark SQL日志分析——日志总体信息的统计

- 服务器返回内容的统计信息

```
# Calculate statistics based on the content size.  
content_size_summary_df = logs_df.describe(['content_size'])  
content_size_summary_df.show()
```

summary	content_size
count	1043177
mean	17531.555702435926
stddev	68561.99906264187
min	0
max	3421948



- 返回码的类型

```
status_to_count_df = (logs_df.groupBy('status')  
                        .count()  
                        .sort('status')  
                        .cache())  
  
status_to_count_length = status_to_count_df.count()  
print 'Found %d response codes' % status_to_count_length  
status_to_count_df.show()
```

Found 7 response codes

status	count
200	940847
302	16244
304	79824
403	58
404	6185
500	2
501	17

# Spark 日志分析——日志总体信息的统计

- 有多少404（找不到页面）错误

```
not_found_df = logs_df.filter(logs_df['status']=='404').cache()
print('Found {0} 404 URLs'.format(not_found_df.count()))
```

- 列出404错误的被访问Endpoints

```
not_found_paths_df = not_found_df.select('path').cache()
unique_not_found_paths_df = not_found_paths_df.distinct()

print '404 URLs:\n'
unique_not_found_paths_df.show(n=40, truncate=False)
```

- 列出404错误最多的前20个Endpoints

```
top_20_not_found_df = (not_found_paths_df.groupBy('path')
                                .count()
                                .sort(desc('count'))
                                .cache())

print 'Top Twenty 404 URLs:\n'
top_20_not_found_df.show(n=20, truncate=False)
```

Top Twenty 404 URLs:

path	count
/pub/winwn/readme.txt	633
/pub/winwn/release.txt	494
/shuttle/missions/STS-69/mission-STS-69.html	430
/images/nasa-logo.gif	319
/elv/DELTA/uncons.htm	178
/shuttle/missions/sts-68/ksc-upclose.gif	154
/history/apollo/sa-1/sa-1-patch-small.gif	146
/images/crawlerway-logo.gif	120
/://spacelink.msfc.nasa.gov	117
/history/apollo/pad-abort-test-1/pad-abort-test-1-patch-small.gif	100
/history/apollo/a-001/a-001-patch-small.gif	97
/images/Nasa-logo.gif	85
	76
/shuttle/resources/orbiters/atlantis.gif	63
/history/apollo/images/little-joe.jpg	62
/images/lf-logo.gif	59

Command took 1.81 seconds -- by mgchen78@gmail.com at 2016/10/9 下午3:33:57 on Miles

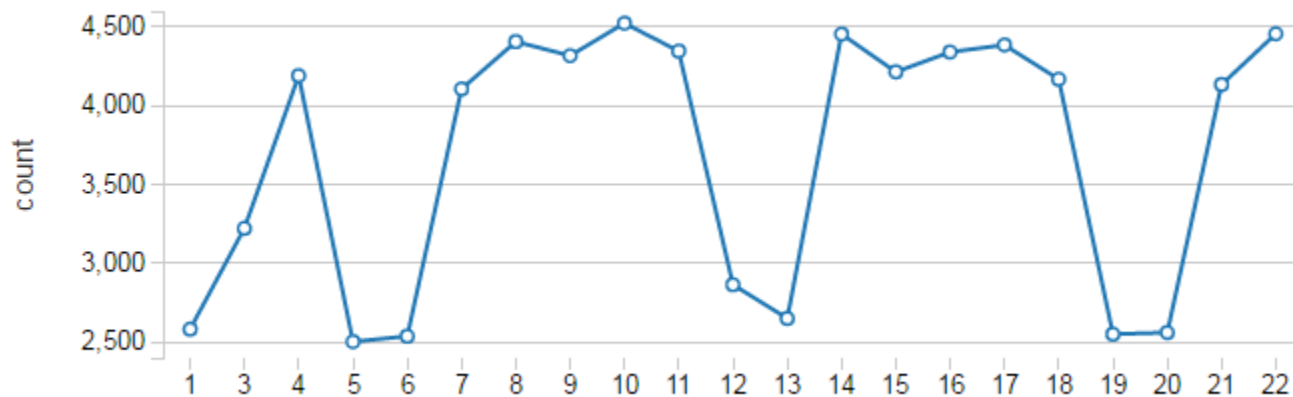
# Spark SQL日志分析——特定时间上状态分析（Temporal）

- 每天独立主机访问数

```
from pyspark.sql.functions import dayofmonth
day_to_host_pair_df = logs_df.select(logs_df['host'], dayofmonth(logs_df['time']).alias('day'))
day_group_hosts_df = day_to_host_pair_df.distinct()
daily_hosts_df = day_group_hosts_df.select(day_group_hosts_df['day']).groupBy(day_group_hosts_df['day']).count().cache()
print 'Unique hosts per day:'
daily_hosts_df.show(30, False)
```

Unique hosts per day:

day	count
1	2582
3	3222
4	4190
5	2502
6	2537
7	4106
8	4406
9	4317
10	4523
11	4346
12	2864
13	2650
14	4454
15	4214



# RDD VS DataFrame/DataSet

## ■ 使用RDD的一般场景：

- ✓ 数据集非结构化，比如，流媒体或者文本流；
- ✓ 使想用low-level的transformation和action来控制你的数据集；
- ✓ 你想使用函数式编程来操作你得数据，而不是用特定领域语言（DSL）
- ✓ 你不在乎schema，比如，当通过列处理（或访问）数据属性，不在意列式存储格式；放弃使用DataFrame/Dataset来优化结构化和半结构化数据集

## ■ 使用DataFrame/DataSet的一般场景

- ✓ 你想使用丰富的语义，high-level抽象和特定领域语言API
- ✓ 你处理的半结构化数据集需要high-level表达， filter, map, aggregation, average, sum , SQL 查询，列式访问和使用lambda函数
- ✓ 你想利用编译时高度的type-safety, Catalyst优化
- ✓ 你想统一和简化API使用跨Spark的Library

# Quiz

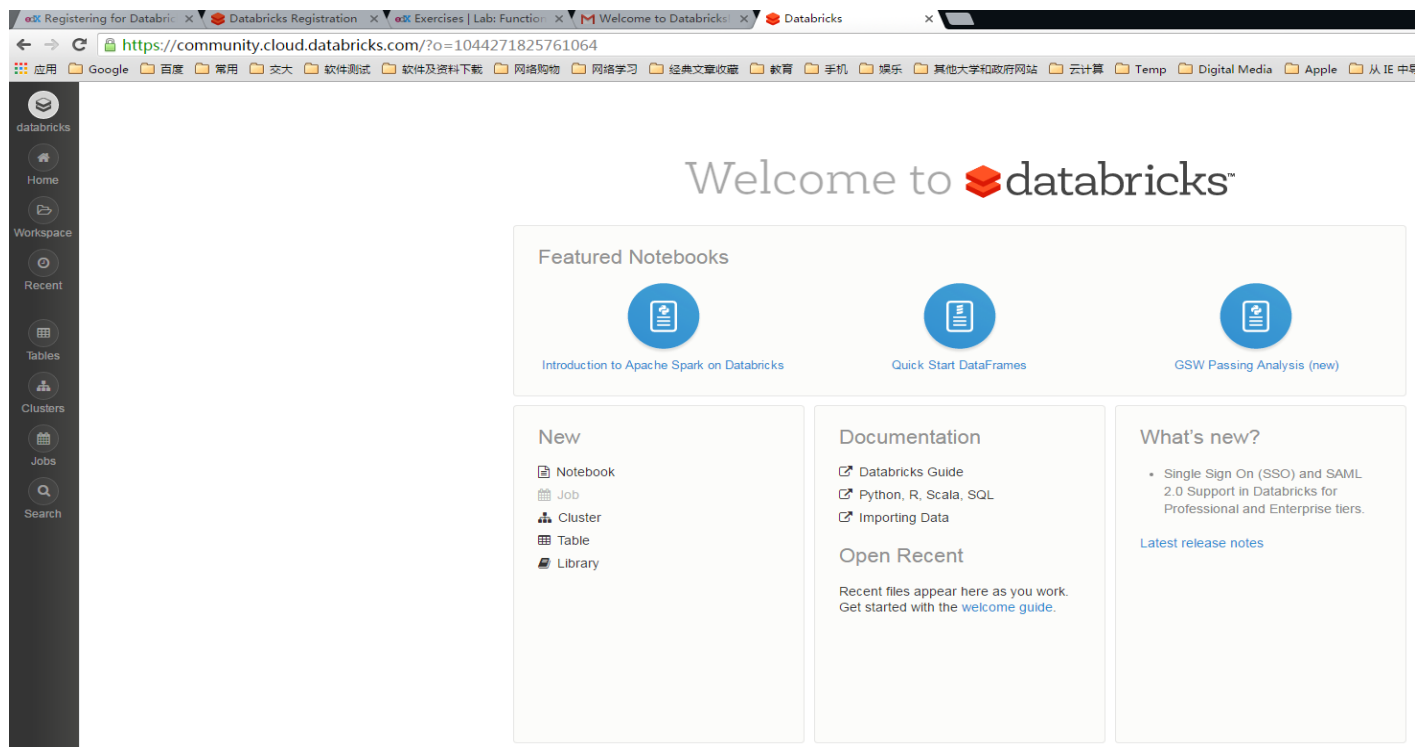
---

■ 我们可以从以下哪些数据源创建DataFrame ?

- A. Existing DataFrames
- B. Data sources, such as files
- C. pandas DataFrames
- D. R DataFrames
- E. Python List

答案：A、B、C、D、E

# Databricks基于云的Spark平台社区版



<https://databricks.com/ce>

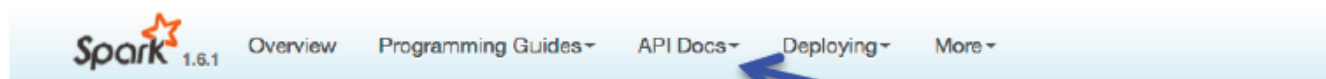
Databricks社区版是免费的在线Spark软件开发环境。用户可以免费申请一个小的Spark集群（6GB内存，Spark1.6.1 Hadoop2）



# Spark 学习资源

## Spark Online Documentation

<https://spark.apache.org/docs/latest/>

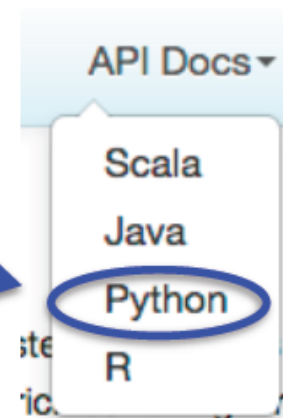


### Spark Overview

Apache Spark is a fast and general-purpose cluster computing system. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including [Spark SQL](#) for SQL and structured data processing, [MLlib](#) for machine learning, [GraphX](#) for graph processing, and [Spark Streaming](#).

### Downloading

Get Spark from the [downloads page](#) of the project website. This documentation is for Spark version 1.6.1. Spark uses Hadoop's client libraries for HDFS and YARN. Downloads are pre-packaged for a handful of popular Hadoop versions. Users can also download a "Hadoop free" binary and



# Spark 学习资源

## Databricks Guide

The screenshot shows the Databricks 'Visualizations Overview' page. On the left is a sidebar with navigation links. The main content area has a title 'Visualizations Overview' and a brief description. A right-hand menu contains links to 'Databricks Guide', 'REST API Docs', 'Community Forum', 'Feedback', and 'Shortcuts'. Two blue arrows point to the 'Databricks Guide' link and a user profile icon in the top right corner.

Visualizations / Visualizations Overview (Python)

Import Notebook  
R Dataset Util - Example

Sending Email - py  
Sending Email - scala  
Server-Side Encryption

**Visualizations**  
Visualizations Overview  
Visualizations in Python  
Visualizations in R  
Visualizations in SQL  
Visualizations in Scala  
Charts & Graphs - py  
Charts & Graphs - scala  
HTML, D3, and SVG  
Matplotlib and GGPiot

**Spark**  
Spark  
Intro Datasets  
Debugging  
Caching  
Run C++ Code - py  
Run C++ Code - scala  
Monte Carlo Simulation

### Visualizations Overview

This notebook outlines your various options for visualizations with Databricks.

### Utilize the built-in visualizations for Databricks

Spark DataFrames can be displayed visually in Databricks and can be configured with just a few clicks.

Learn about the basics of using visualizations in your preferred language:

- [Visualization Basics in Python](#)
- [Visualization Basics in Scala](#)
- [Visualization Basics in SQL](#)
- [Visualization Basics in R](#)

To get more information about the various charts and graph types:

- [Charts & Graphs in Python](#)
- [Charts & Graphs in Scala](#)
- For SQL, see either of the notebooks above. Python or Scala is just used to generate the data displayed in the graph.

Right-hand menu:  
Databricks Guide  
REST API Docs  
Community Forum  
Feedback  
Shortcuts

# Spark 学习资源

---

## Technical Blogs

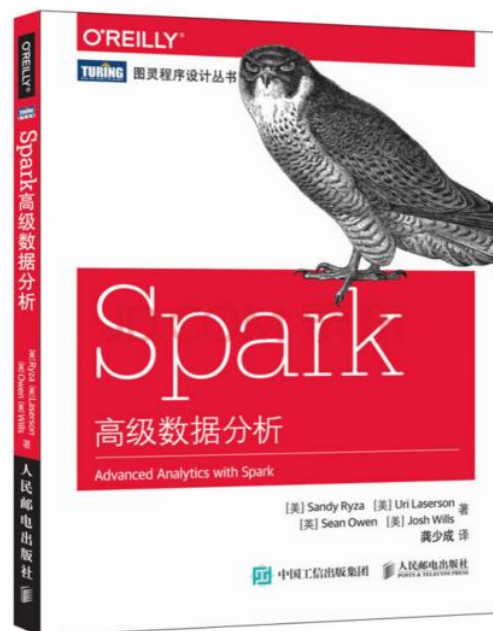
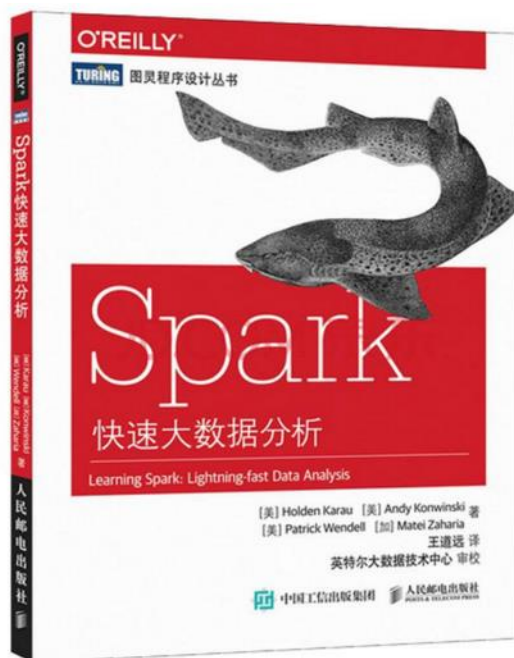
Databricks: <https://databricks.com/blog/category/engineering>

Cloudera: <http://blog.cloudera.com/blog/category/spark/>

IBM: <http://www.spark.tc/blog/>

- Hortonworks: <http://hortonworks.com/blog/category/spark/>
- Many more! (eBay, AWS, MapR, Datastax, etc)

# 推荐参考书





**Q&A!**