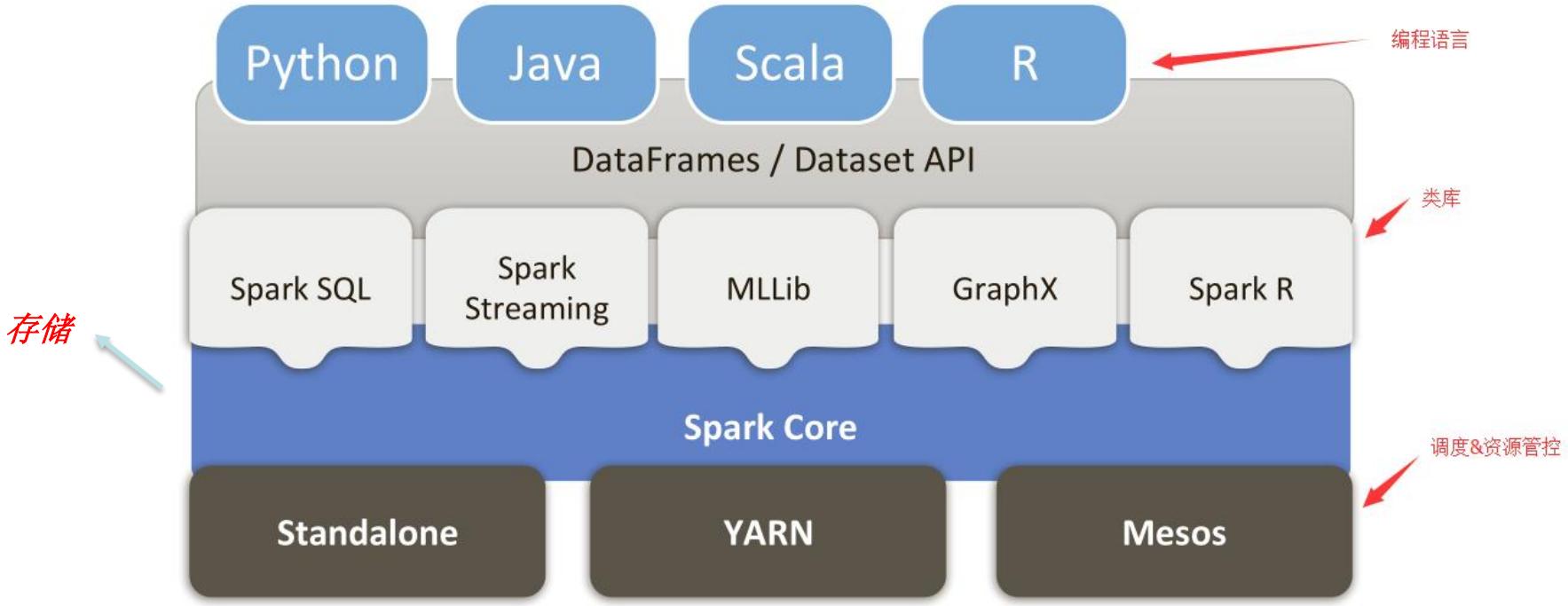
Spark研究：SparkCore/SparkSQL/SparkStreaming

易车网 宋沄剑

目录

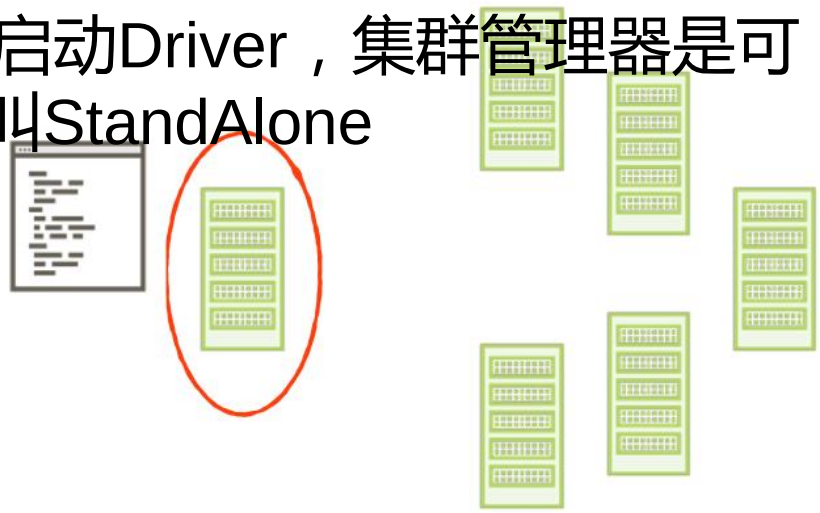
- Spark架构
- Scala语言基础
- SparkCore/RDD
- SparkSQL
- Streaming
- 打包

Spark架构



Spark架构

- Master-Slave架构
  - 一个coordinator
    - Coordinator需要ZooKeeper保证高可用
  - N个Worker
    - Worker在集群中自身就高可用
- Coordinate接收程序，并将程序分解为Task，分配给不同Worker
- 集群管理器（Cluster模式）启动Driver，集群管理器是可插拔的，内置的集群管理器叫StandAlone



Scala基础：变量

- 需要一点Scala基础，理解后面内容
- 变量声明：
  - val variablename　不可变
  - var variablename　可变
  - var variablename:String　带类型
- 变量如果不指定类型，编译器会自动推断

Scala基础：函数与方法

- 声明：

```
def Hello(name: String)(p2:String) : String ={
    s”Hello $name”
}
```

- Def定义函数,多个参数是括号分割
- 冒号后面是返回类型
- Return可以省略，Scala函数中最后一个表达式是返回值

Scala基础：接受函数作为参数

```
def Hello(name: String)(f2() = > String) : String ={
    s”Hello $name ${ f2() }”
}
```

- 第二个参数接受一个返回值为String的函数

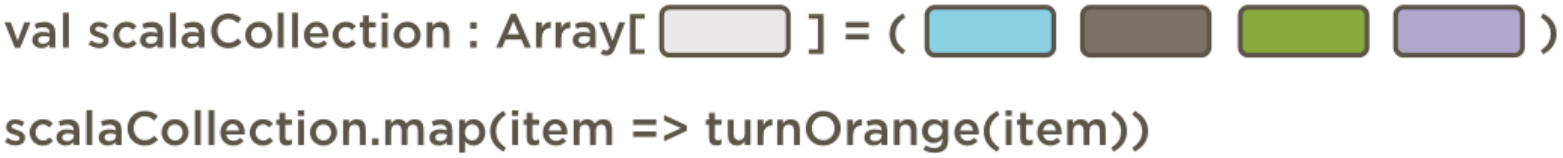
Scala基础：匿名函数&lambda表达式

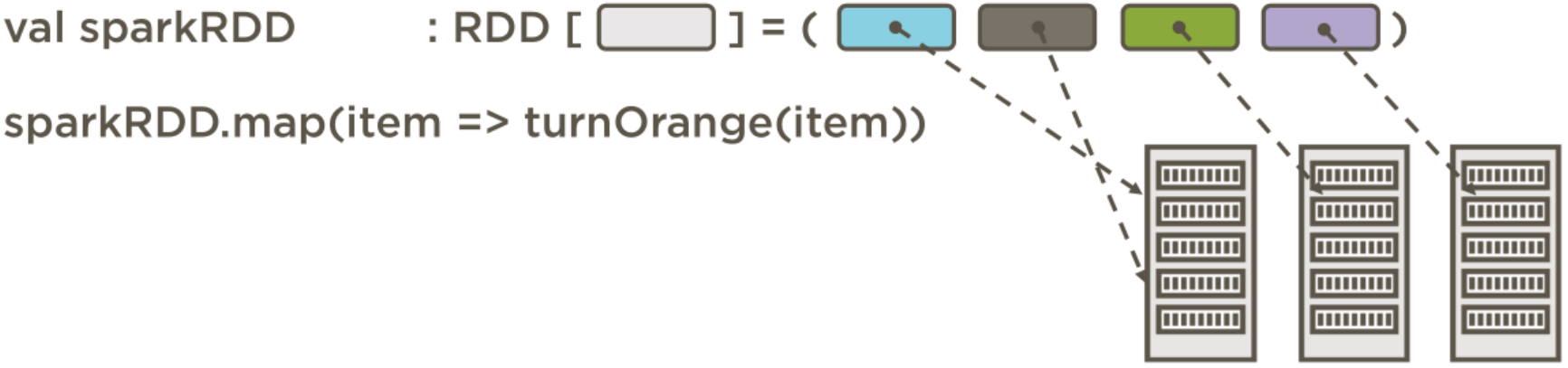
- 匿名函数只用一次，减少不必要的代码
- Lambda表达式 x = > y

Val data = Seq.range(1 ,5)
Data.map((para1:int) => {return para1*2})完全形式
Data.map((para1:int) => {para1*2}) 省略return
Data.map((para1:int) => para1*2) 省略花括号
Data.map((para1) => para1*2) 省略返回类型
Data.map(para1=> para1*2) 省略括号
Data.map(_*2) 省略输入参数与”right arrow”

RDD

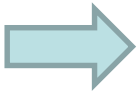
- 特殊对象集合
  - 内存中
  - 持久化





RDD Partition

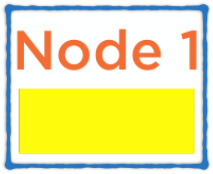
1	Swetha	30	Bangalore
2	Vitthal	35	New Delhi
3	Navdeep	25	Mumbai
4	Janani	35	New Delhi
5	Navdeep	25	Mumbai
6	Janani	35	New Delhi



1	Swetha	30	Bangalore
2	Vitthal	35	New Delhi

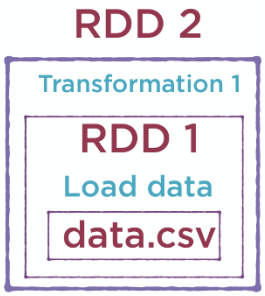
3	Navdeep	25	Mumbai
4	Janani	35	New Delhi

5	Navdeep	25	Mumbai
6	Janani	35	New Delhi

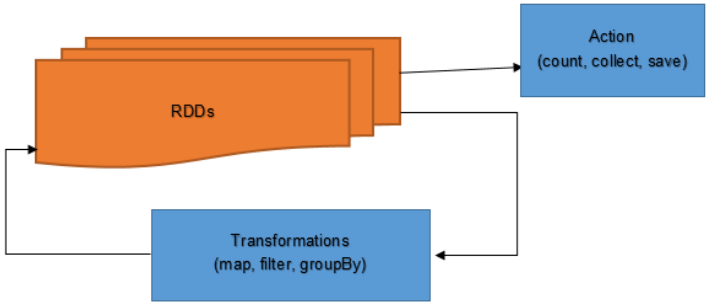


RDD操作

- Transformation（返回RDD）
  - Map（ ）
  - Groupby（ ）
- Action（汇总结果）
  - Sum（ ）
  - Reduce（ ）

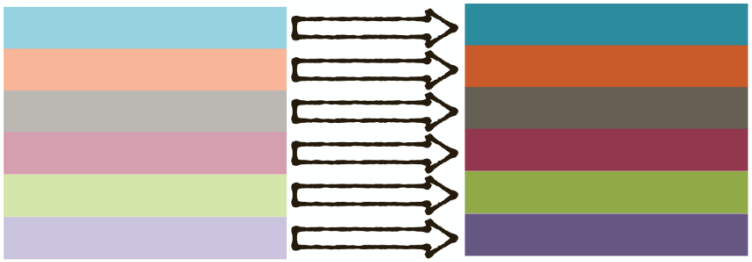


RDD总能追踪到根源，如下好处：
出现故障，从根源重新计算
Lazy Evalution，只有需要时才materialize



RDD函数式编程

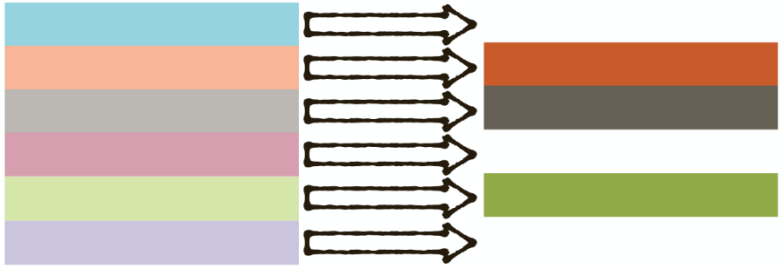
- 作用于集合(MAP操作)



```
.map(line => line.toLowerCase)
```

=

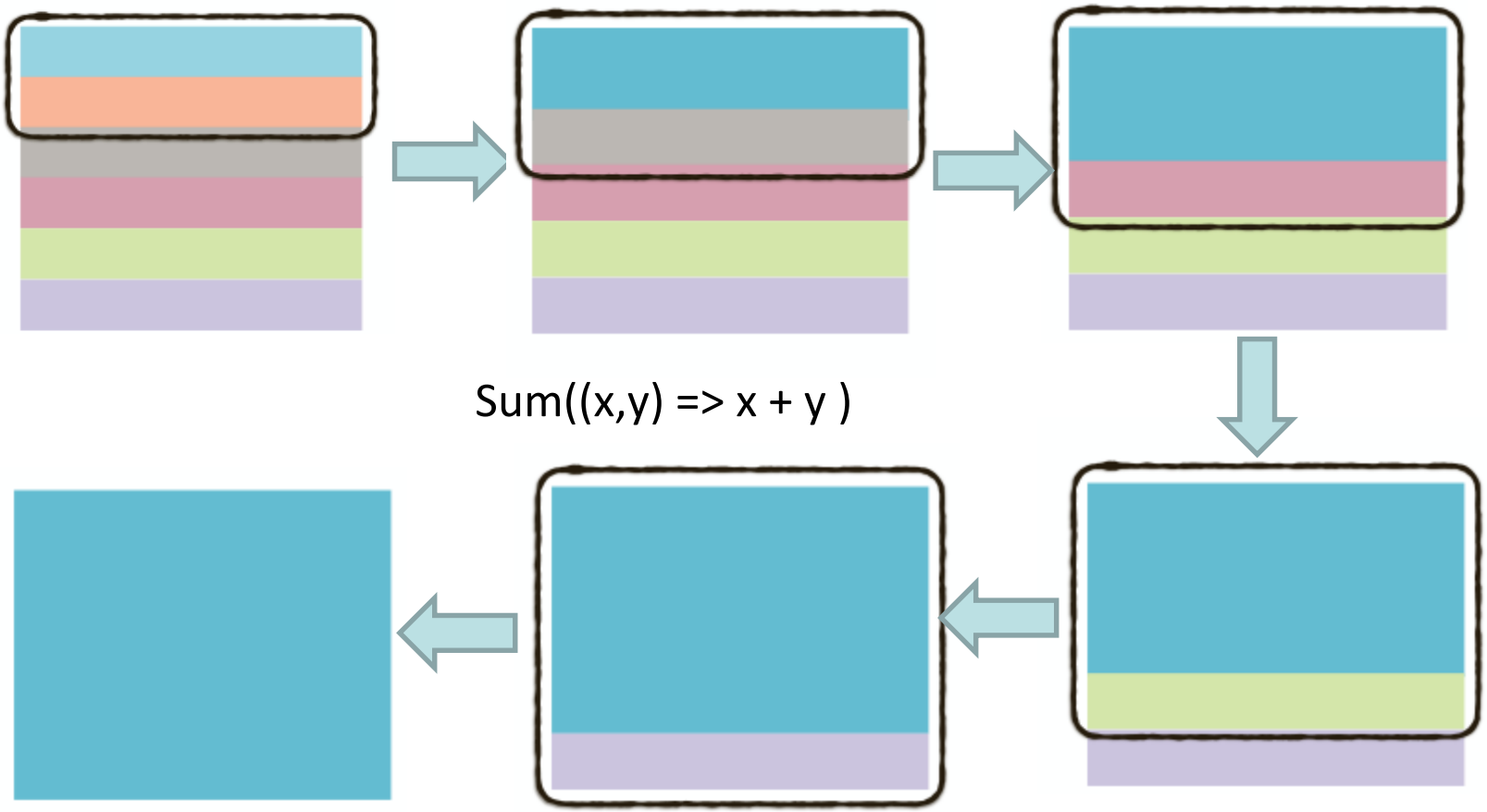
```
.map(_.toLowerCase)
```



.filter(_.startwith(“test”))

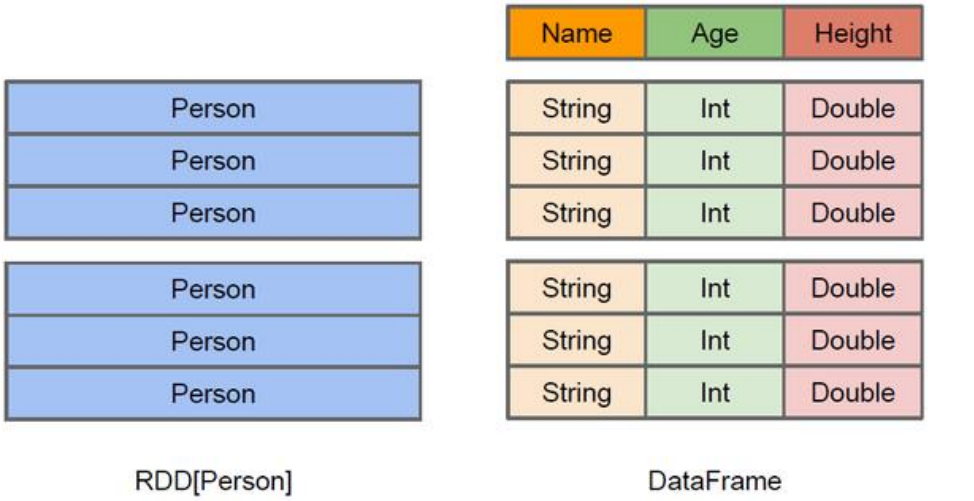
RDD函数式编程

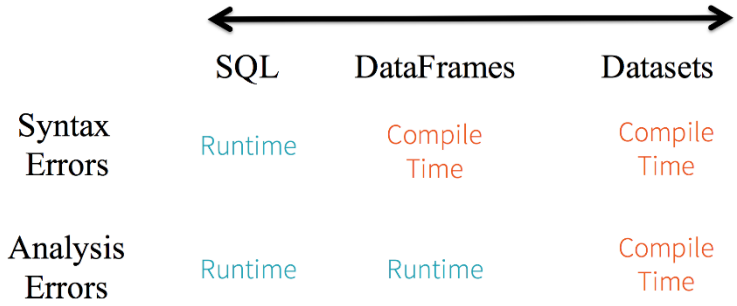
- Reduce操作（带有2个参数的函数，比如Sum）



Spark SQL

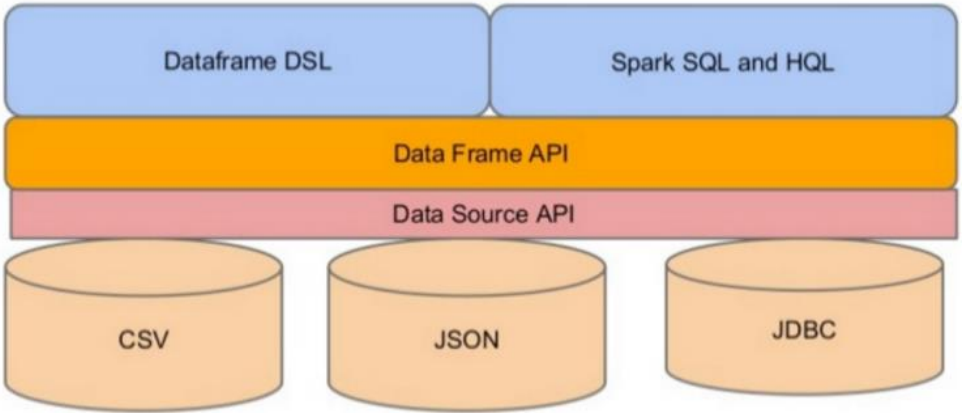
- 一个用于结构化的Spark模块
- 相比RDD，提供数据列信息
- DataFrame来源：RDD/Hive/HDFS等





Spark SQL

- 核心组件
  - 不同数据源之间的ETL：Datasource API
  - 实现SQL的两种方式：
    - Dataframe API
    - Catalyst optimizer（SQL）



Spark SQL Dataframe API AND SQL API

Using RDDs

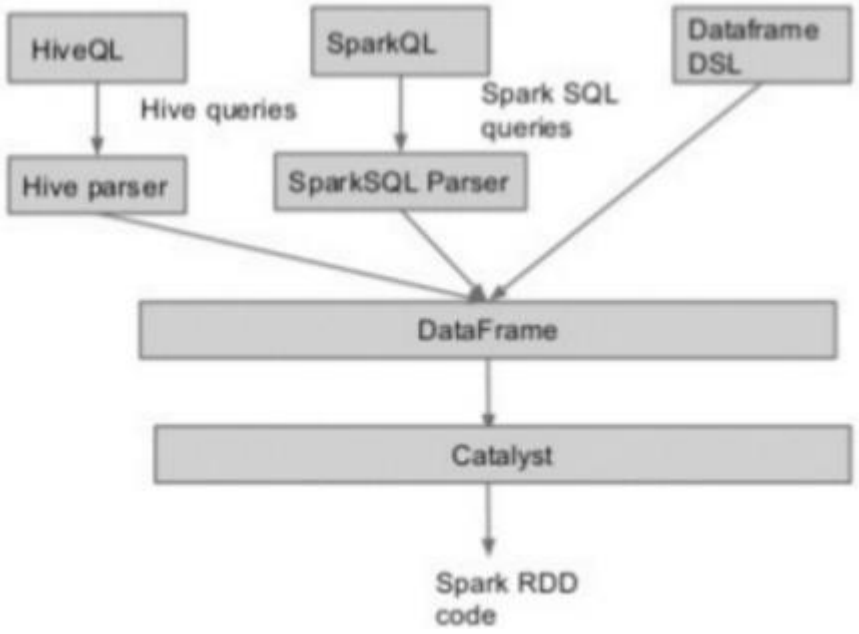
```
data = sc.textFile(...).split("\t")
data.map(lambda x: (x[0], [int(x[1]), 1])) \
    .reduceByKey(lambda x, y: [x[0] + y[0], x[1] + y[1]]) \
    .map(lambda x: [x[0], x[1][0] / x[1][1]]) \
    .collect()
```

Using SQL

```
SELECT name, avg(age)
FROM people
GROUP BY name
```

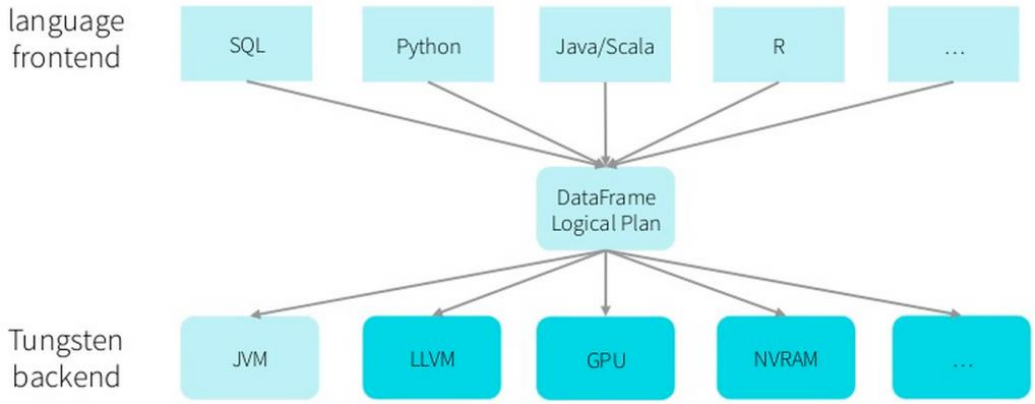
Using DataFrames

```
sqlCtx.table("people") \
    .groupBy("name") \
    .agg("name", avg("age")) \
    .collect()
```



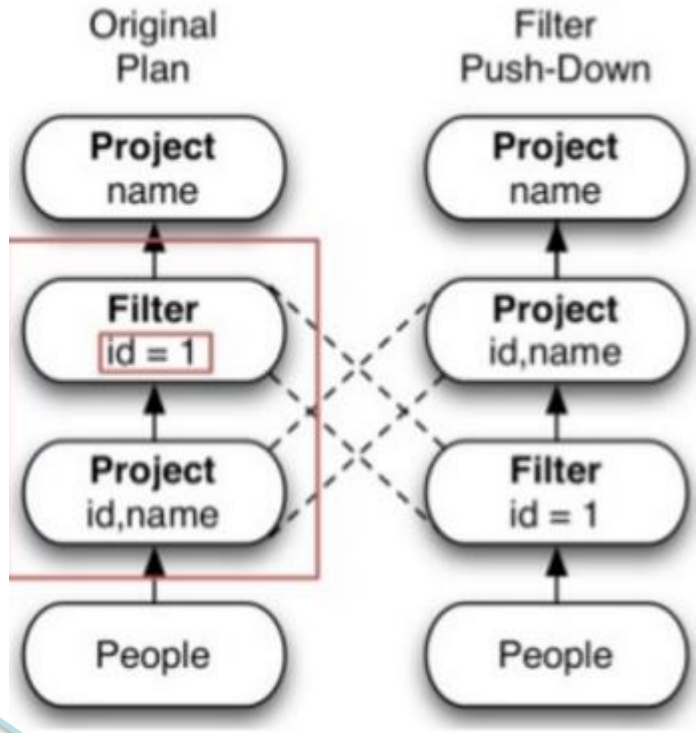
Tungsten

- 在Spark 1.4引入，1.5成为默认引擎
- 优化CPU使用，减少内存占用，优化缓存分布（CPU寄存器）
- 只能用于Dataframe，SparkSQL，Dataset

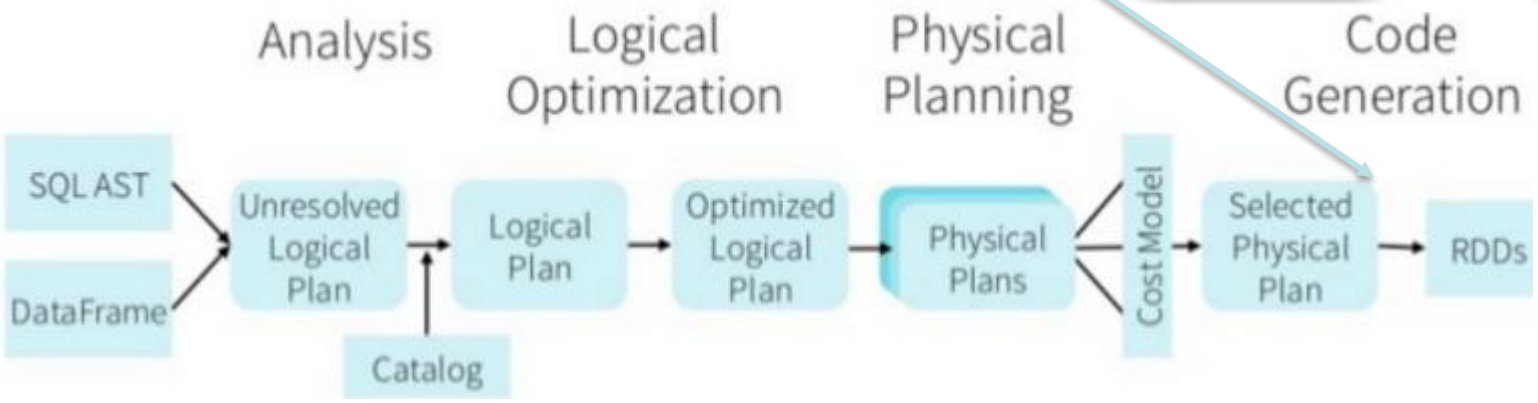


Catalyst Optimizer

- 用于将Logical plan转为physical plan



最终所有的操作还是RDD



性能比较

- 来源：https://cs.stanford.edu/~matei/papers/2015/vldb_spark.pdf

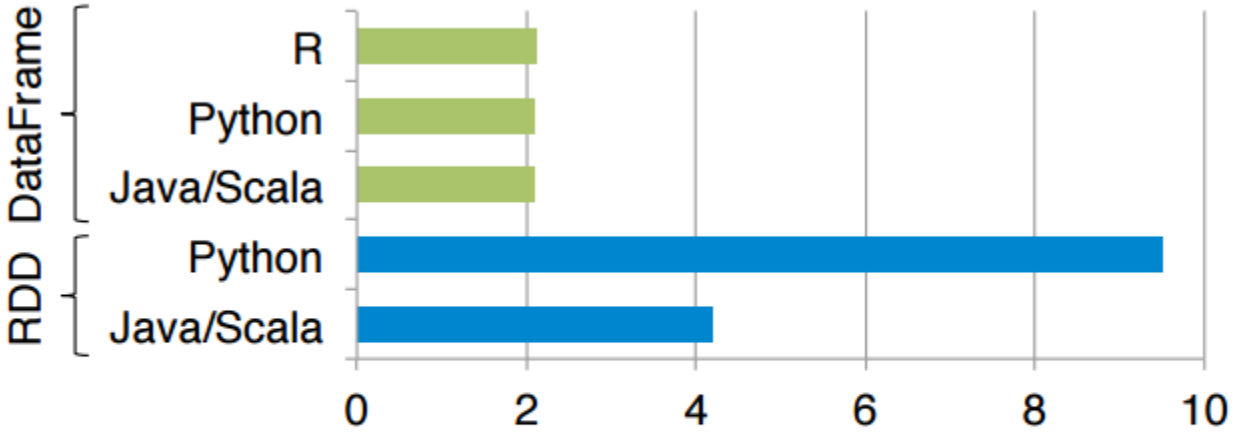


Figure 4: Running time (sec) of a simple aggregation query using DataFrames versus Spark's functional RDD API.

Spark SQL

- 接口是SQLContext,用于创建DataFrames
  - SQL Queries
  - DataFrameAPI（1.3之前叫ScheamRDD）
  - Dataset API（加入了类型安全，用于优化）
- SQLContext有两个主流实现
  - SQLContext
  - HiveContext（是SQLContext的子类）
- Spark 2.0后使用SparkSession

Demo：SQL query to multiple Source

- 表A：来自Hadoop的Json
- 表B：来自SQL Server

- 一个SQL Join这两个表

Why Dataset API

- RDD虽然是强类型，但没办法用优化器
- Dataframe虽然是支持优化器，但不是强类型
- 所以最好有强类型+优化器=>Dataset API

实例：df.select("colour") colour这列只有在运行时才知道是否存在

```
class Body(id: Int,
 width: Double,
 height: Double,
 depth: Double,
 material: String,
 color: String)
val ds = df.as[Body]
```

强类型，编码时有提示，编译时可报错

```
ds.groupByKey(body => body.color)
.agg(typedCount[Body](_.id).name("count(id)"),
typedSum[Body](_.width).name("sum(width)"),
typedSum[Body](_.height).name("sum(height)"),
typedSum[Body](_.depth).name("sum(depth)"))
.withColumnRenamed("value", "group")
.alias("Summary by color level") .show()
```

等同于:
Select count(id) as count(id),sum(width) as sum(width),sum(height) as sum(height),sum(depth) as sum(depth)
Group by color

Why Dataset API2

- Dataframe会丢失domain object信息，例如：

```
case class Person(name : String , age : Int) val personRDD =
sc.makeRDD(Seq(Person("A",10),Person("B",20)))
val personDF = sqlContect.createDataframe(personRDD)
personDF.rdd // returns RDD[Row] , does not returns RDD[Person]
```

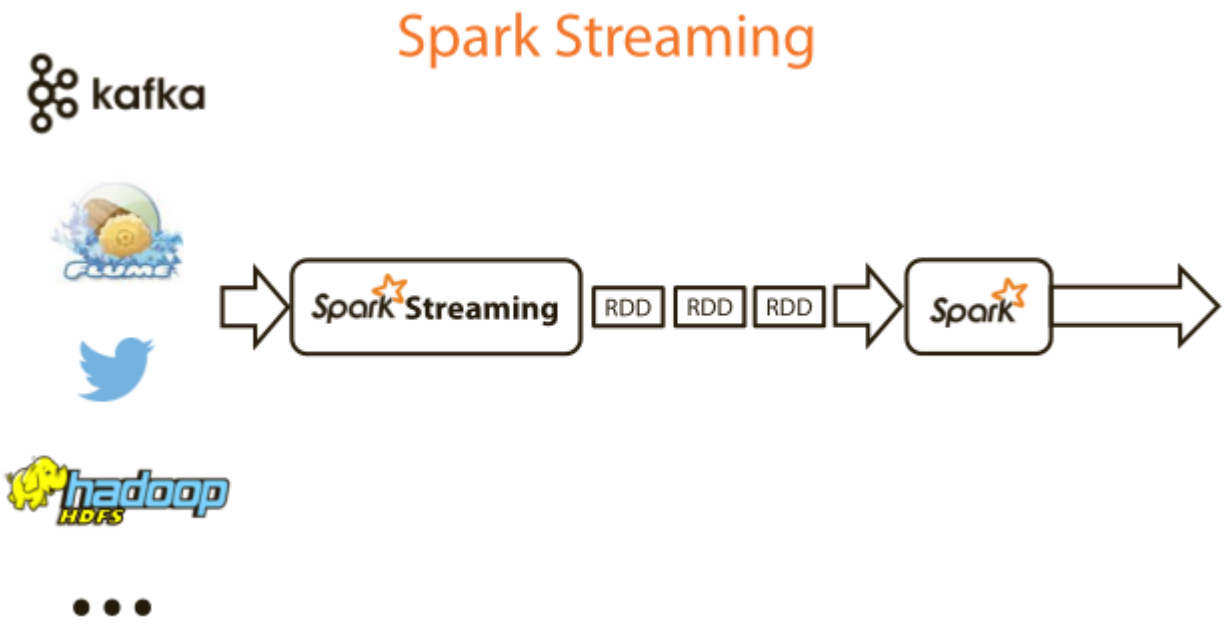
- 而使用Dataset不会

```
case class Person(name : String , age : Int)
val personRDD = sc.makeRDD(Seq(Person("A",10),Person("B",20)))
val personDF = sqlContect.createDataframe(personRDD)
val ds:Dataset[Person] = personDF.as[Person]
ds.filter(p => p.age > 25)
ds.filter(p => p.salary > 25) // 报错，salary不属于Person类
ds.rdd // 返回domain object对象 RDD[Person]
```

Dataset API

- Spark 1.6引入，但Spark 2.0才成熟
- 混合了Dataframe与RDD的优势
- 依然使用SQLContext
- DataFrame可以看作是Dataset[Row]的别名
- 该API不支持Python，只支持Scala和Java
- Dataset API虽然表面上是操作对象，但实际上会生成执行计划
- Dataset操作类似RDD，也是Lazy，transformation时不会实际调用，只有调用Action时才会计算

Streaming



流式数据

- 每一个收到的数据都可以被看作为流

```
Response: 0 1113 1490605381548,YichePage2,页面浏览,,Visitor-7,Page-4,东风御风警用系列,御风警用车
Response: 0 1114 1490605381548,YichePage3,页面浏览,,Visitor-44,Page-8,PT-HS,PTHS
Response: 0 1115 1490605381548,YichePage1,查看,,Visitor-53,Page-0,A7,海星A7
Response: 0 1116 1490605381548,YichePage4,页面浏览,,Visitor-47,Page-1,英速亚,Insignia 英速亚
Response: 0 1117 1490605381548,yicheApp,页面浏览,,Visitor-11,Page-0,LF-CC,LF-CC
Response: 0 1118 1490605381548,YichePage1,页面浏览,,Visitor-33,Page-5,ONYX,ONYX
Response: 0 1119 1490605381548,YichePage3,页面浏览,,Visitor-61,Page-2,斯宾特(进口),奔驰凌特(Sprinter) 凌特商务车 奔驰凌特Sprinter
```

Streaming

- 每个Batch的实体作为一个RDD

Discretized Stream
或Dstream

与**RDD**的关系
HashMap[time,RDD[T]]

- 根据Batch Interval形成Batch

Dstream[T]对象

- 本质上不过是RDD数组+时间
- 生成RDD：HashMap[time,RDD[T]]
- 转换
  - Map/filter/union
  - window/reduceByKeyAndWindow
  - updateStateByKey
  - 其他转换操作

- Action操作
  - Print/saveAsTextFiles
  - foreachRDD

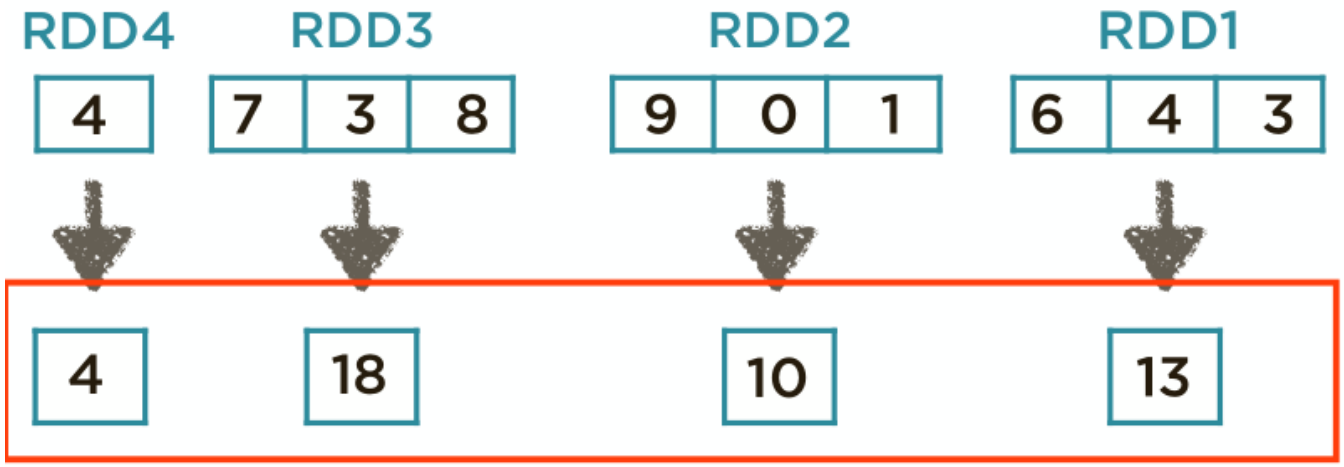
Dstream与DataFrame

- 只需要转换为RDD，再正常转换为DF即可

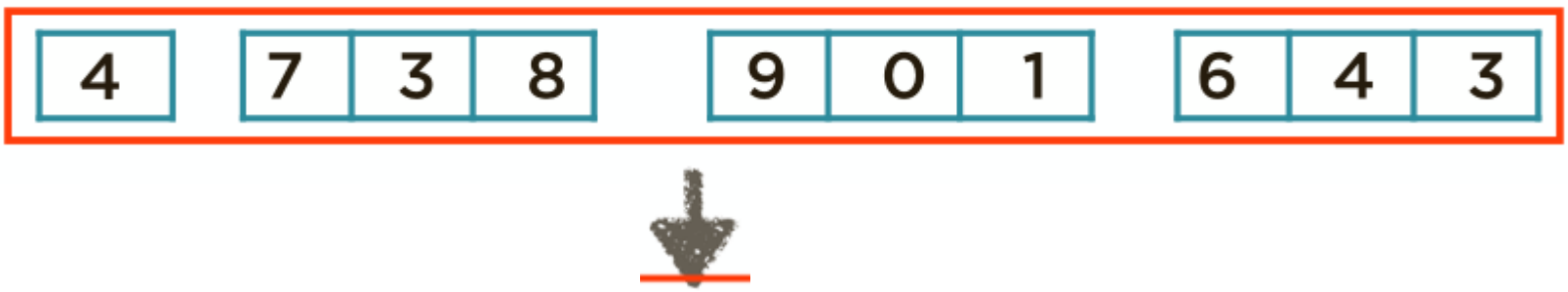
```
val words: DStream[String] = ...

words.foreachRDD { rdd =>

  // Get the singleton instance of SQLContext
  val sqlContext = SQLContext.getOrCreate(rdd.sparkContext)
  import sqlContext.implicits._

  // Convert RDD[String] to DataFrame
  val wordsDataFrame = rdd.toDF("word")

  // Register as table
  wordsDataFrame.registerTempTable("words")

  // Do word count on DataFrame using SQL and print it
  val wordCountsDataFrame =
    sqlContext.sql("select word, count(*) as total from words group by word")
  wordCountsDataFrame.show()
}
```

两种Streaming

- Stateless



- Stateful



Stateful Streaming

- 所有的Stateful Streaming都需要checkpoint
- 使用updateStateByKey更新之前的状态

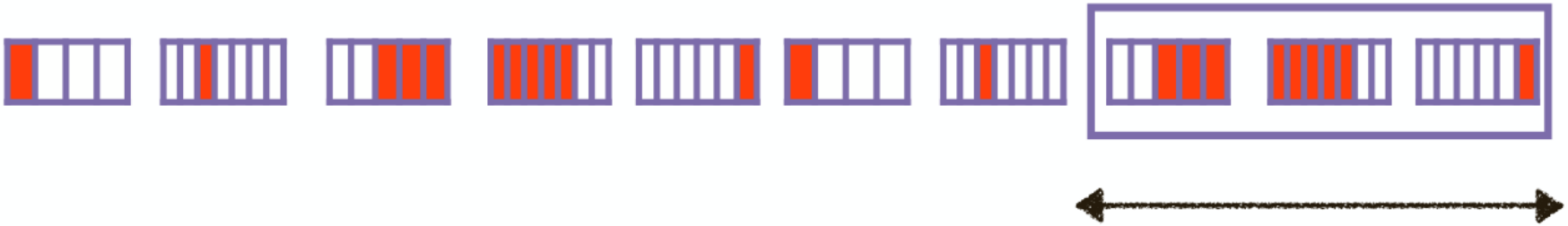
```
//stateful founction
val addFunc = (currValues: Seq[Int], prevValueState: Option[Int]) => {
  //通过Spark内部的reduceByKey按key规约，然后这里传入某key当前批次的Seq/List,再计算当前批次的总和
  val currentCount = currValues.sum
  // 已累加的值
  val previousCount = prevValueState.getOrElse(0)
  // 返回累加后的结果，是一个Option[Int]类型
  Some(currentCount + previousCount)
}

//save it to sql server
//1490605381548,YichePage1,查看,,Visitor-53,Page-0,A7,海星A7
stream.map(line => {
  val splitLines = line._2.split(",")
  (splitLines(6), 1)   //A7,1

}).updateStateByKey(addFunc).reduceByKey((x,y) => x + y )

  .foreachRDD(rdd => {

    rdd.foreachPartition(partitionOfRecords =>{
```

*必须首先**map**成**key-value**的形式*

If omit return,function will return last statement

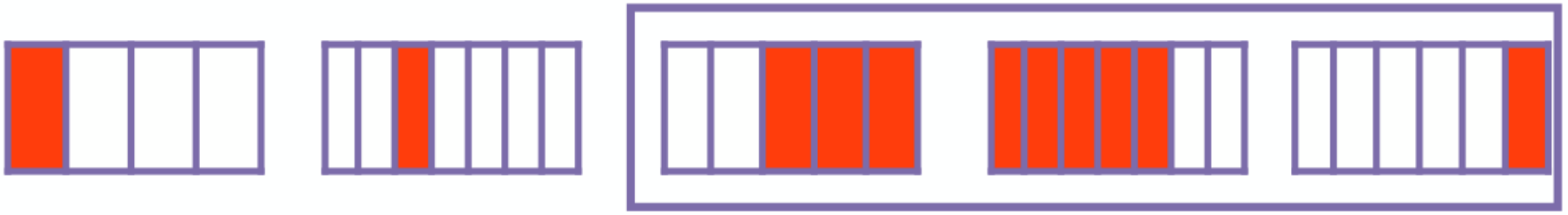
Streaming Window Operation

- 某类操作在一个更大的区间内更合适
- 比如在30秒内错误日志的数量



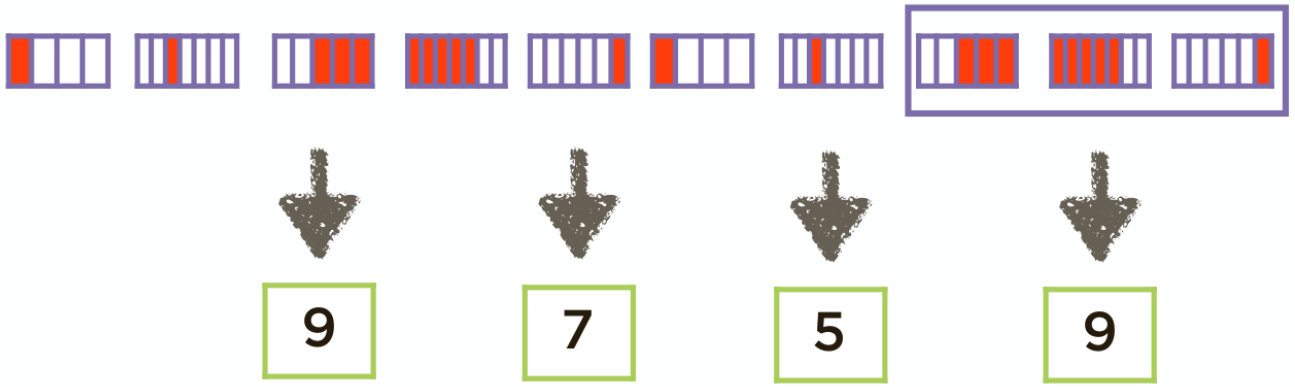
在滑动窗口内的多个RDD会合并为一个

Streaming Window



Batch interval = 10 seconds

Window size = 30 seconds

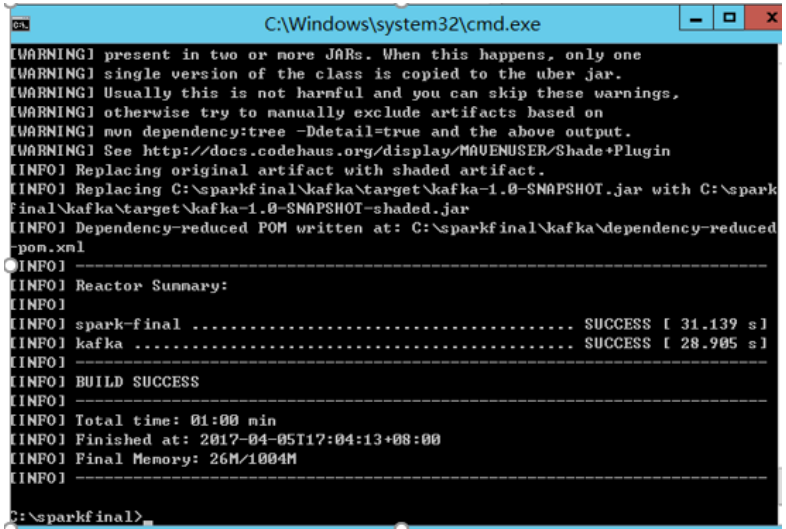


countByWindow()

Demo：模拟页面日志->kafka->streaming->SQL Server

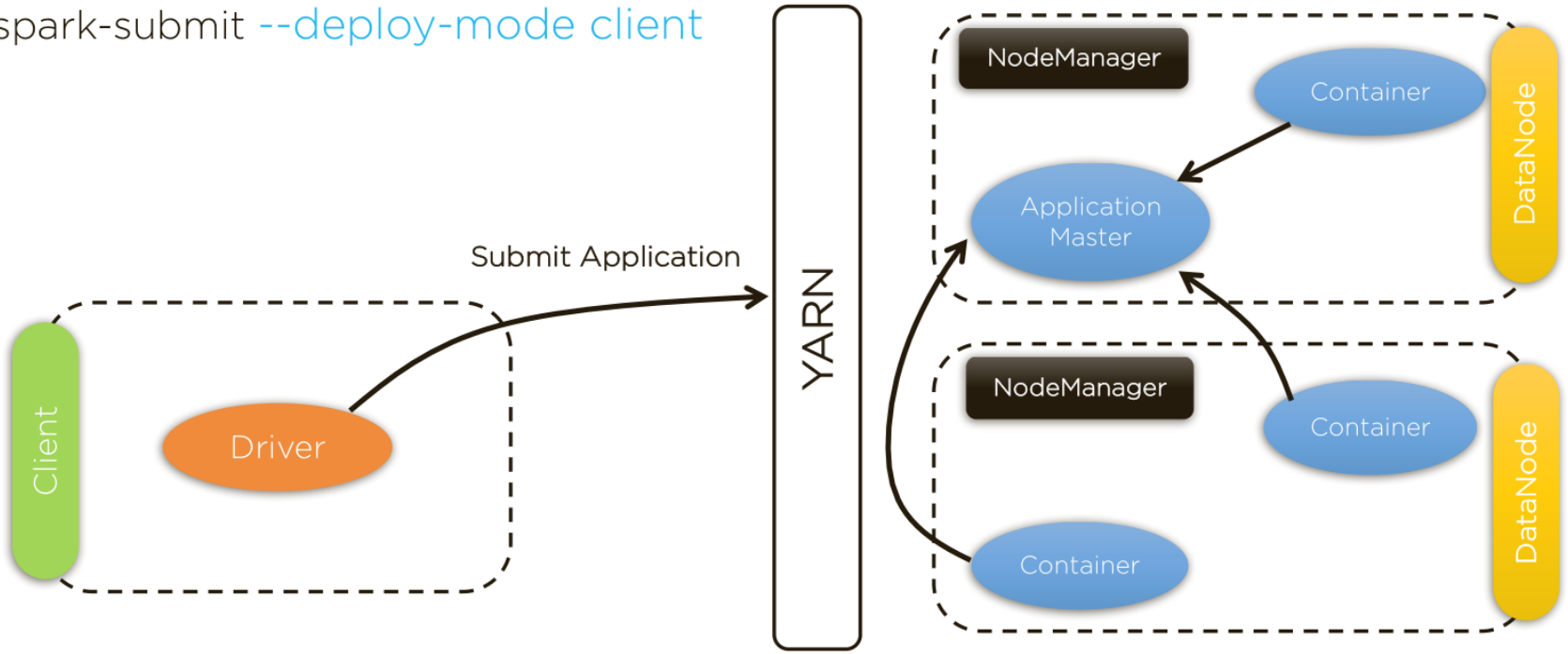
Spark打包方式

- 由于每个Spark程序都依赖独特的第三方Jar包（算法/驱动等/不同版本），因此服务器很难做到包含所有Jar包，因此客户端包含所有依赖Jar包就很方便
- 这种包含所有依赖的Jar称为：UberJar或AssemblyJar
- 可以使用Maven或SBT打包，Maven打包方式如下：
  - 配置maven-shade-plugin插件
  - 在包含POM的根目录执行mvn package



Spark提交模式-Client

./spark-submit --deploy-mode client



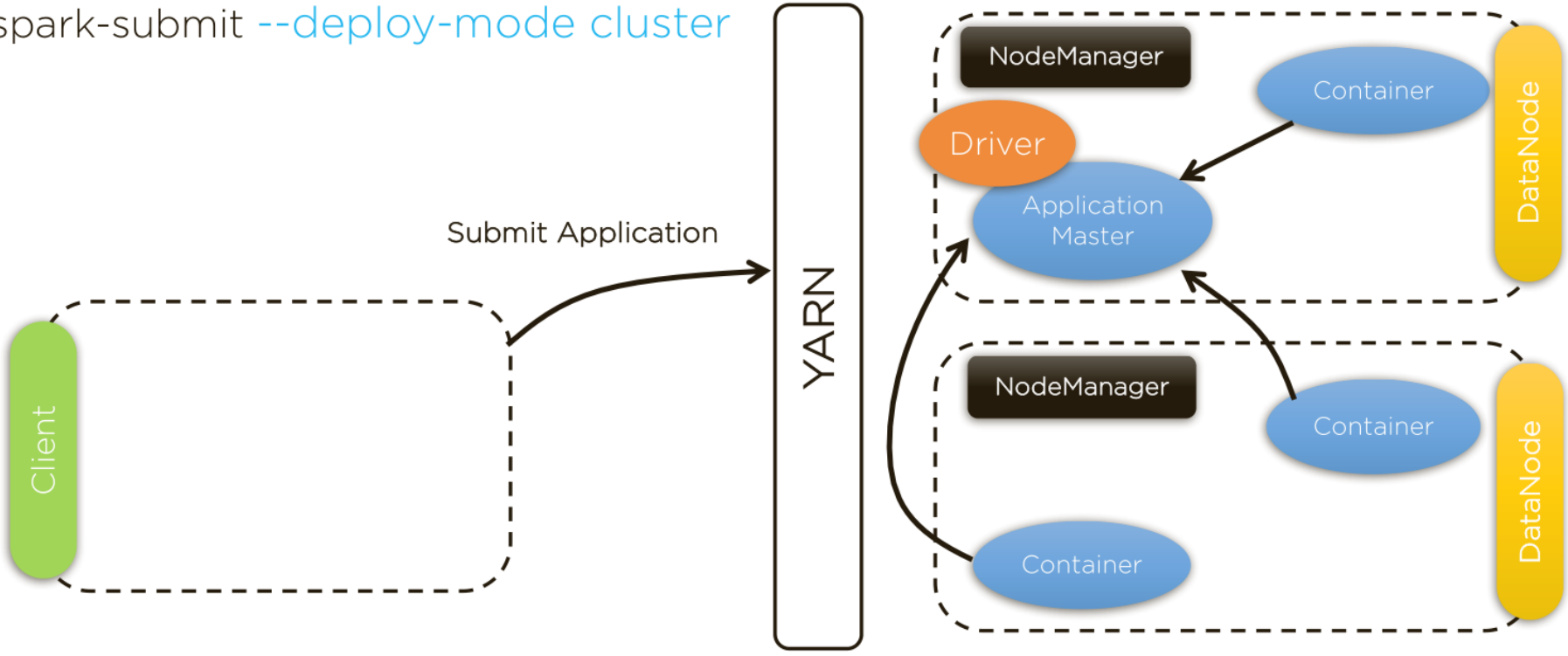
Running Applications

Application ID		Name	Cores	Memory per Node
app-20170406163632-0018	(kill)	testStreaming	7	1024.0 MB

只有worker，Driver在spark-submit的服务器

Spark提交模式-Cluster

./spark-submit --deploy-mode cluster



Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time
app-20170406163632-0018 (kill)	testStreaming	7	1024.0 MB	2017/04/06 16:36:32

Running Drivers

Submission ID	Submitted Time	Worker	State	Cores
driver-20170406163628-0009 (kill)	Thu Apr 06 16:36:28 CST 2017	worker-20170406153430-172.17.1.68-44929	RUNNING	1

Driver在集群的Master节点，客户端只收到ACK

Spark-submit两种模式客户端反馈对比

```
17/04/06 16:36:08 INFO ShutdownHookManager: Deleting directory /tmp/spark-20819360-da95-4ded-a542-d7a2c7887411
[root@dbsrvm-hadoop1 bin]# spark-submit --master spark://172.17.1.76:6066  --deploy-mode cluster --class TestStreaming.CalculateCarModel --num-executo
68:8020/SQLDATA/kafka.jar
Multiple versions of Spark are installed but SPARK_MAJOR_VERSION is not set
Spark1 will be picked by default
Running Spark using the REST application submission protocol.
17/04/06 16:36:27 INFO RestSubmissionClient: Submitting a request to launch an application in spark://172.17.1.76:6066.
17/04/06 16:36:28 INFO RestSubmissionClient: Submission successfully created as driver-20170406163628-0009. Polling submission state...
17/04/06 16:36:28 INFO RestSubmissionClient: Submitting a request for the status of submission driver-20170406163628-0009 in spark://172.17.1.76:6066.
17/04/06 16:36:28 INFO RestSubmissionClient: State of driver driver-20170406163628-0009 is now RUNNING.
17/04/06 16:36:28 INFO RestSubmissionClient: Driver is running on worker worker-20170406153430-172.17.1.68-44929 at 172.17.1.68:44929.
17/04/06 16:36:28 INFO RestSubmissionClient: Server responded with CreateSubmissionResponse:
{
  "action" : "CreateSubmissionResponse",
  "message" : "Driver successfully submitted as driver-20170406163628-0009",
  "serverSparkVersion" : "1.6.2",
  "submissionId" : "driver-20170406163628-0009",
  "success" : true
}
```

Cluster：只接受ACK返回信息

```
17/04/06 16:51:23 INFO BlockManagerInfo: Added broadcast_0_piece0 in memory on dbsrvm-hadoop2.scom.com:44909 (size: 2.1 KB, free: 511.1 MB)
17/04/06 16:51:24 INFO TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in 3307 ms on dbsrvm-hadoop2.scom.com (1/1)
17/04/06 16:51:24 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
17/04/06 16:51:24 INFO DAGScheduler: ShuffleMapStage 0 (map at package.scala:62) finished in 3.325 s
17/04/06 16:51:24 INFO DAGScheduler: looking for newly runnable stages
17/04/06 16:51:24 INFO DAGScheduler: running: Set()
17/04/06 16:51:24 INFO DAGScheduler: waiting: Set(ResultStage 1)
17/04/06 16:51:24 INFO DAGScheduler: failed: Set()
17/04/06 16:51:24 INFO DAGScheduler: Submitting ResultStage 1 (ShuffledRDD[2] at reduceByKey at package.scala:66), which has no missing parents
17/04/06 16:51:24 INFO MemoryStore: Block broadcast_1 stored as values in memory (estimated size 2.6 KB, free 8.5 KB)
17/04/06 16:51:24 INFO MemoryStore: Block broadcast_1_piece0 stored as bytes in memory (estimated size 1608.0 B, free 10.0 KB)
17/04/06 16:51:24 INFO BlockManagerInfo: Added broadcast_1_piece0 in memory on 172.17.1.76:46568 (size: 1608.0 B, free: 511.1 MB)
17/04/06 16:51:24 INFO SparkContext: Created broadcast 1 from broadcast at DAGScheduler.scala:1008
17/04/06 16:51:24 INFO DAGScheduler: Submitting 2 missing tasks from ResultStage 1 (ShuffledRDD[2] at reduceByKey at package.scala:66)
17/04/06 16:51:24 INFO TaskSchedulerImpl: Adding task set 1.0 with 2 tasks
17/04/06 16:51:24 INFO TaskSetManager: Starting task 0.0 in stage 1.0 (TID 1, dbsrvm-hadoop2.scom.com, partition 0,PROCESS_LOCAL, 1943 bytes)
17/04/06 16:51:24 INFO TaskSetManager: Starting task 1.0 in stage 1.0 (TID 2, dbsrvm-hadoop2.scom.com, partition 1,PROCESS_LOCAL, 1943 bytes)
17/04/06 16:51:24 INFO BlockManagerInfo: Added broadcast_1_piece0 in memory on dbsrvm-hadoop2.scom.com:44909 (size: 1608.0 B, free: 511.1 MB)
17/04/06 16:51:24 INFO MapOutputTrackerMasterEndpoint: Asked to send map output locations for shuffle 0 to dbsrvm-hadoop2.scom.com:37498
17/04/06 16:51:24 INFO MapOutputTrackerMaster: Size of output statuses for shuffle 0 is 149 bytes
17/04/06 16:51:24 INFO TaskSetManager: Finished task 1.0 in stage 1.0 (TID 2) in 477 ms on dbsrvm-hadoop2.scom.com (1/2)
17/04/06 16:51:24 INFO TaskSetManager: Finished task 0.0 in stage 1.0 (TID 1) in 484 ms on dbsrvm-hadoop2.scom.com (2/2)
17/04/06 16:51:24 INFO DAGScheduler: ResultStage 1 (foreachPartition at package.scala:70) finished in 0.485 s
17/04/06 16:51:24 INFO TaskSchedulerImpl: Removed TaskSet 1.0, whose tasks have all completed, from pool
17/04/06 16:51:24 INFO DAGScheduler: Job 0 finished: foreachPartition at package.scala:70, took 4.486110 s
17/04/06 16:51:24 INFO JobScheduler: Finished job streaming job 1491468680000 ms.0 from job set of time 1491468680000 ms
17/04/06 16:51:24 INFO JobScheduler: Total delay: 4.670 s for time 1491468680000 ms (execution: 4.542 s)
17/04/06 16:51:24 INFO ReceivedBlockTracker: Deleting batches ArrayBuffer()
17/04/06 16:51:24 INFO InputInfoTracker: remove old batch metadata:
```

Client：返回所有程序执行的细节，可用性取决于客户端，客户端关闭则程序停止执行

Cluster包依赖问题

- 由于Cluster模式Driver在服务端，因此需要在每一个Worker节点都有相同的Jar包，因此需要满足下述条件之一
  - 每个Worker相同路径必须有相同Jar包
  - 执行的Jar在每个节点都能访问的共享路径，比如HDFS

示例：

hadoop fs -put kafka.jar /SQLDATA

spark-submit --master spark://172.17.1.76:6066 --deploy-mode cluster --class TestStreaming.CalculateCarModel --num-executors 1 hdfs://172.17.1.68:8020/SQLDATA/kafka.jar

What About Yarn Or Mesos

- Yarn和Mesos负责资源调度

What's next

- Spark 2 new API
- Spark 2 new structure streaming