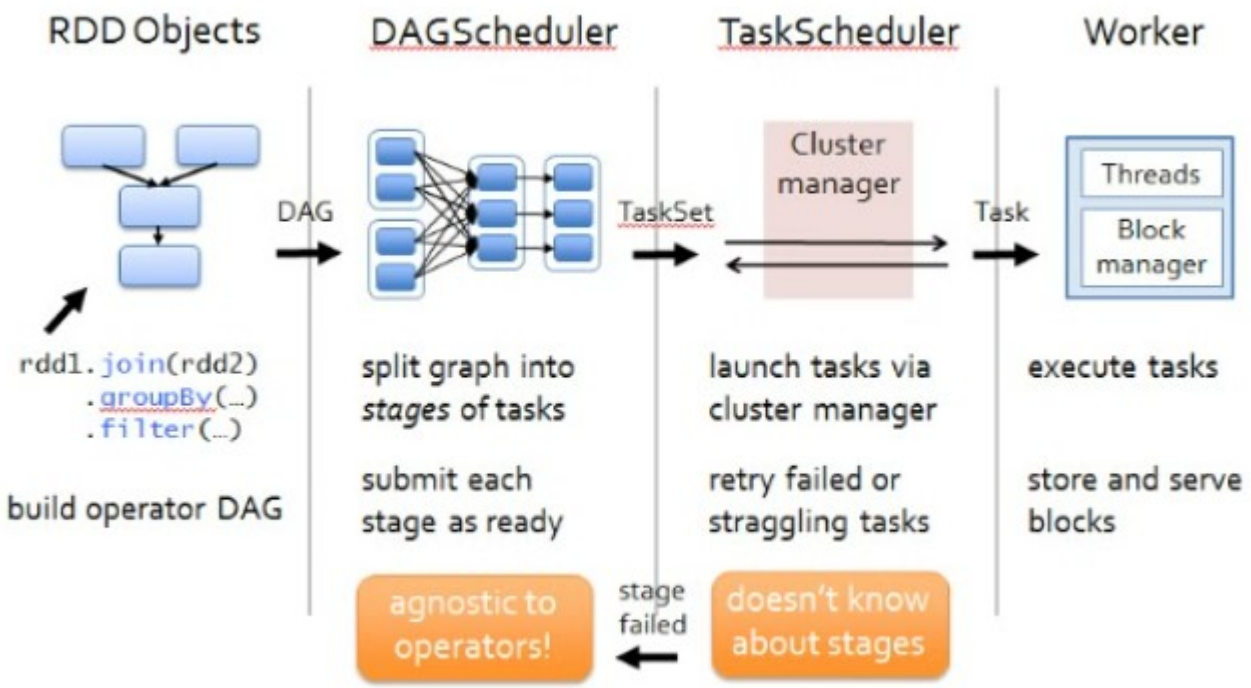


第三课： Spark 运行架构和解析

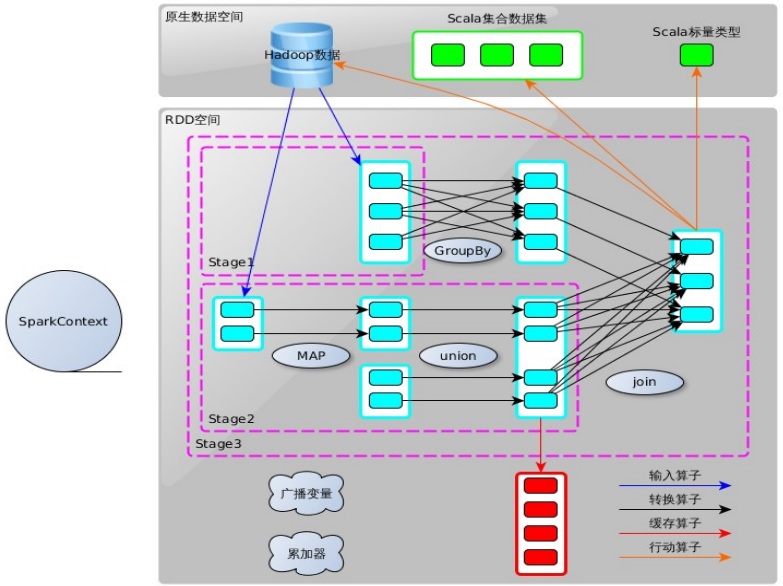
法律声明

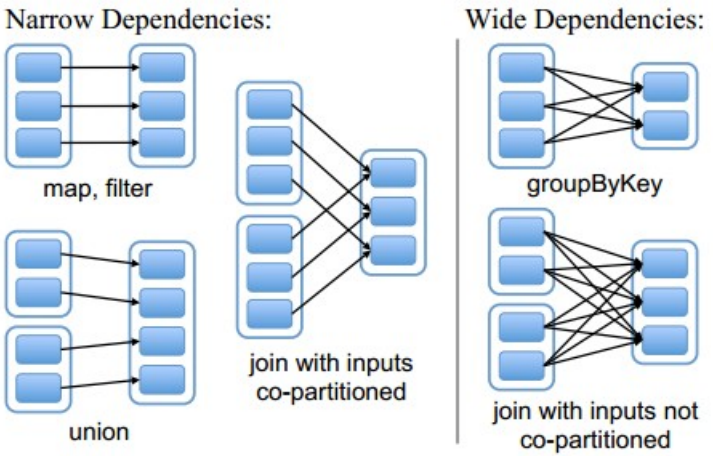
【声明】本视频和幻灯片为炼数成金网络课程的教学资料，所有资料只能在课程内使用，不得在课程以外范围散播，违者将可能被追究法律和经济责任。

课程详情访问炼数成金培训网站

http://edu.dataguru.cn

上周回顾





- Spark 编程模型几个要素
  - Driver program
  - 输入
  - Transformation
  - Action
  - 缓存
  - 共享变量
- RDD 五特性
  - 分区
  - 依赖
  - 函数
  - 分区策略（(K,V) 类型 RDD）
  - 本地性策略

本周内容

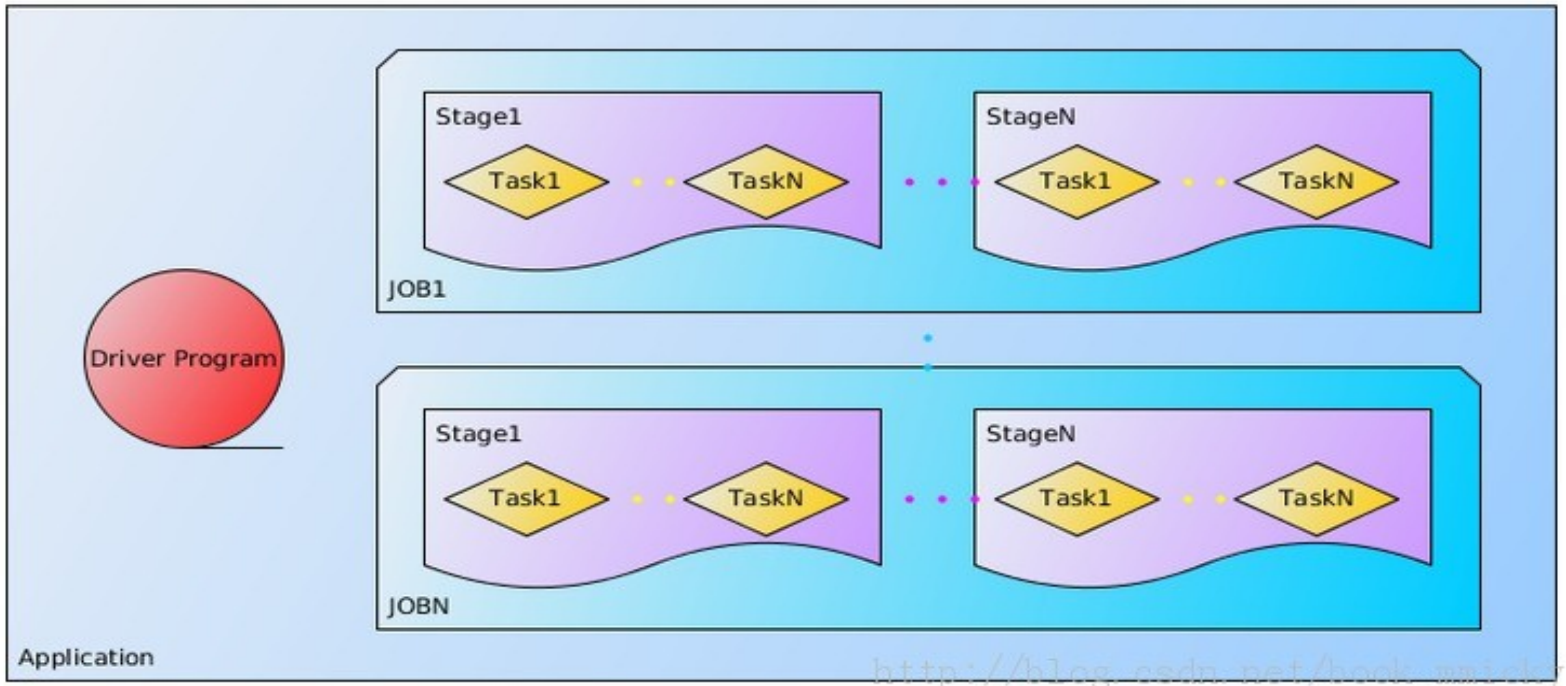
- Spark 运行架构（Spark 应用程序第二部分）
- 例子解析
- Spark 在不同集群中的运行架构

本周内容

- Spark 运行架构
- 例子解析
- Spark 在不同集群中的运行架构

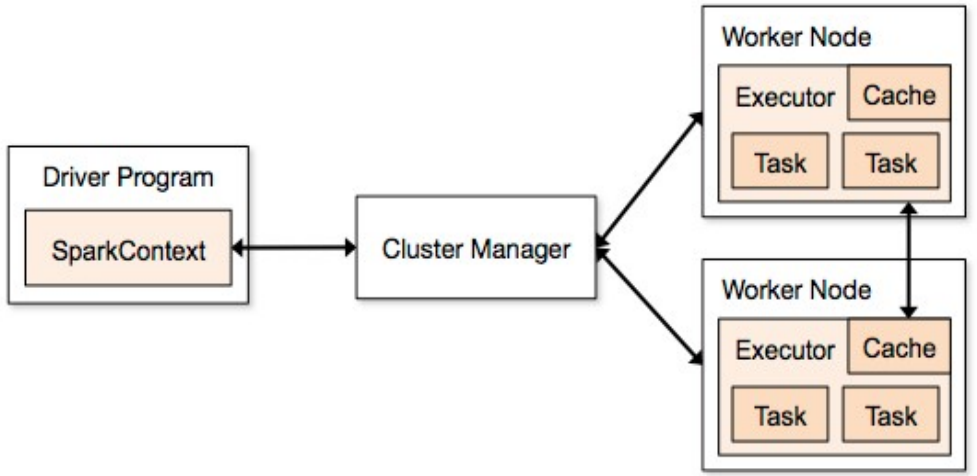
Spark 运行架构

- Job： 包含多个 Task 组成的并行计算，往往由 Spark action 催生。
- Stage： Job 的调度单位，对应于 TaskSet 。
- TaskSet： 一组关联的、相互之间没有 shuffle 依赖关系的人去组成的任务集。
- Task： 被送到某个 executor 上的工作单元

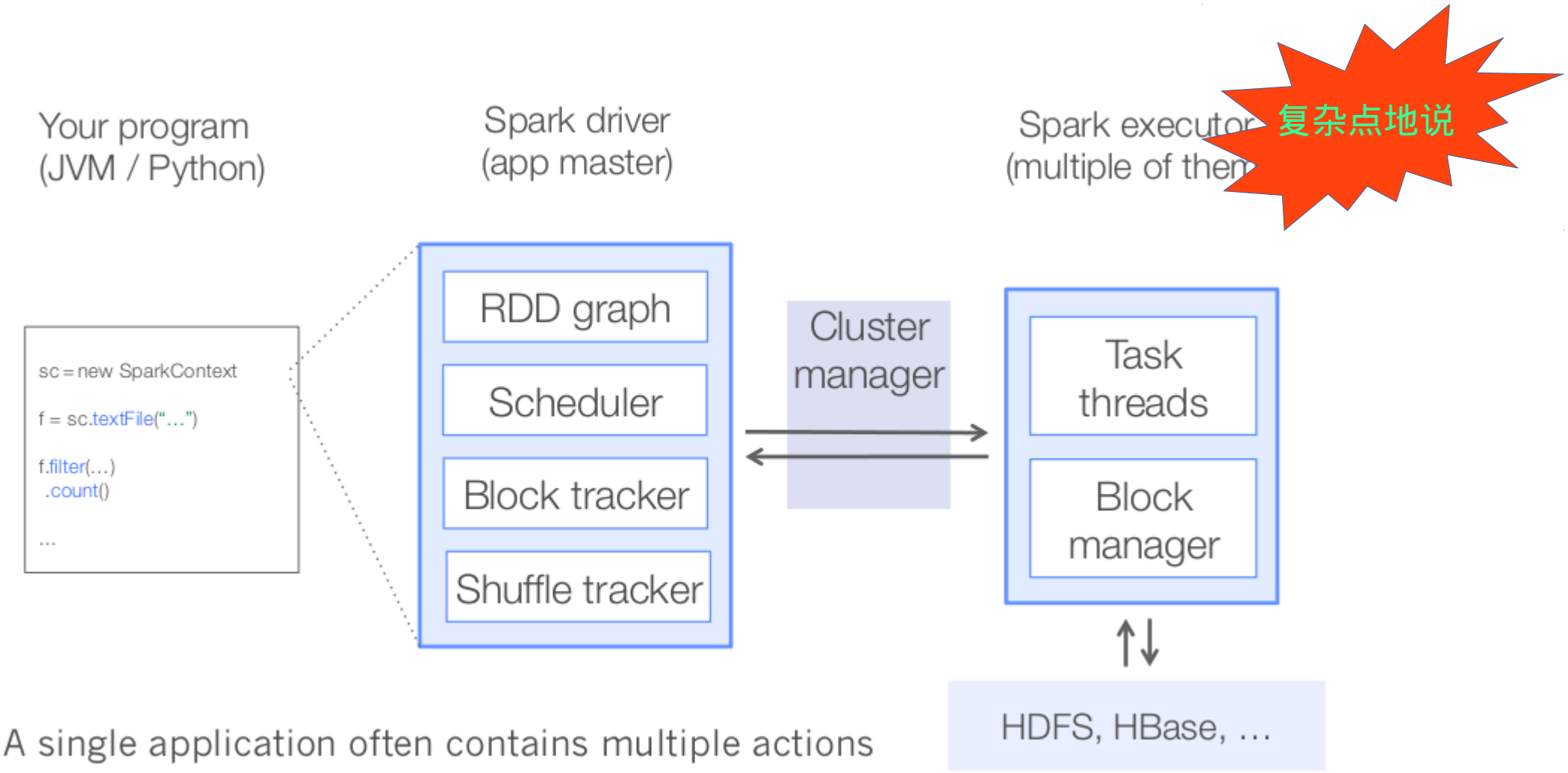


Spark 运行架构

简单点地说

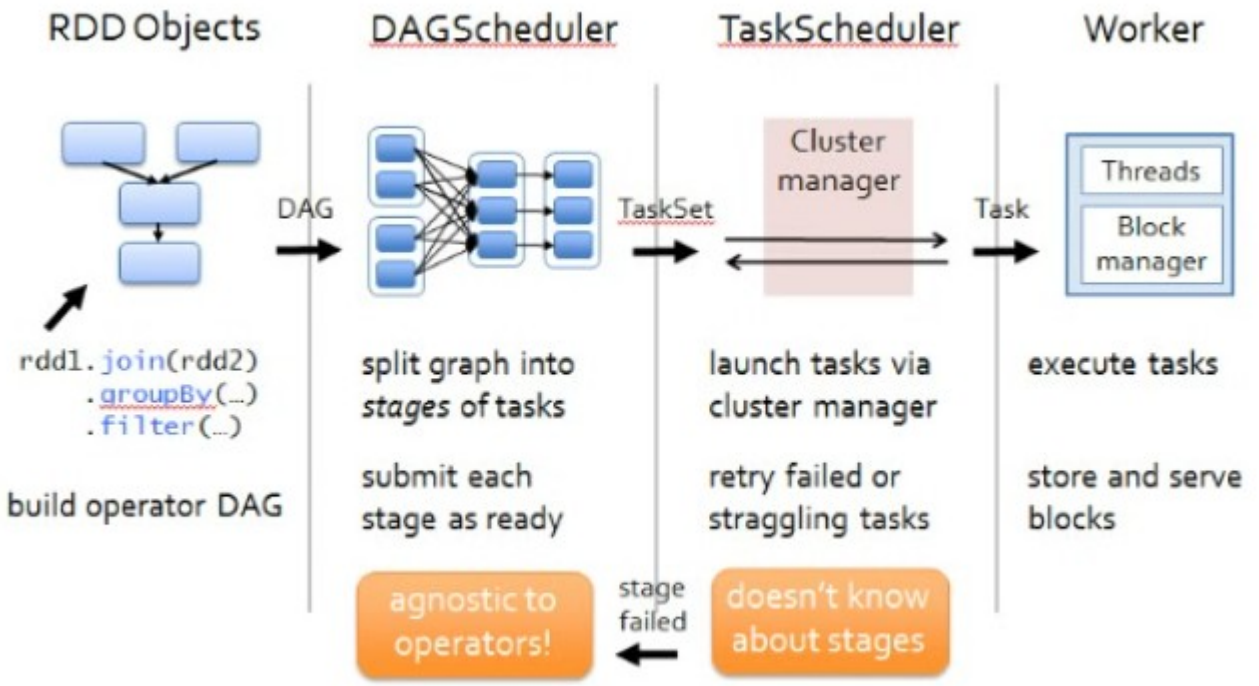


Spark 运行架构


Your program
(JVM / Python)
Spark driver
(app master)
Spark executor
(multiple of them)
复杂点地说
sc = new SparkContext
f = sc.textFile("...")
f.filter(...)
.count()
...
RDD graph
Scheduler
Block tracker
Shuffle tracker
Cluster manager
Task threads
Block manager
HDFS, HBase, ...
A single application often contains multiple actions

Spark 运行架构


全面点地说


RDD Objects
DAGScheduler
TaskScheduler
Worker
DAG
TaskSet
Task
Cluster manager
Threads
Block manager
rdd1.join(rdd2)
.groupBy(…)
.filter(…)
build operator DAG
split graph into stages of tasks
submit each stage as ready
launch tasks via cluster manager
retry failed or straggling tasks
execute tasks
store and serve blocks
agnostic to operators!
stage failed
doesn't know about stages

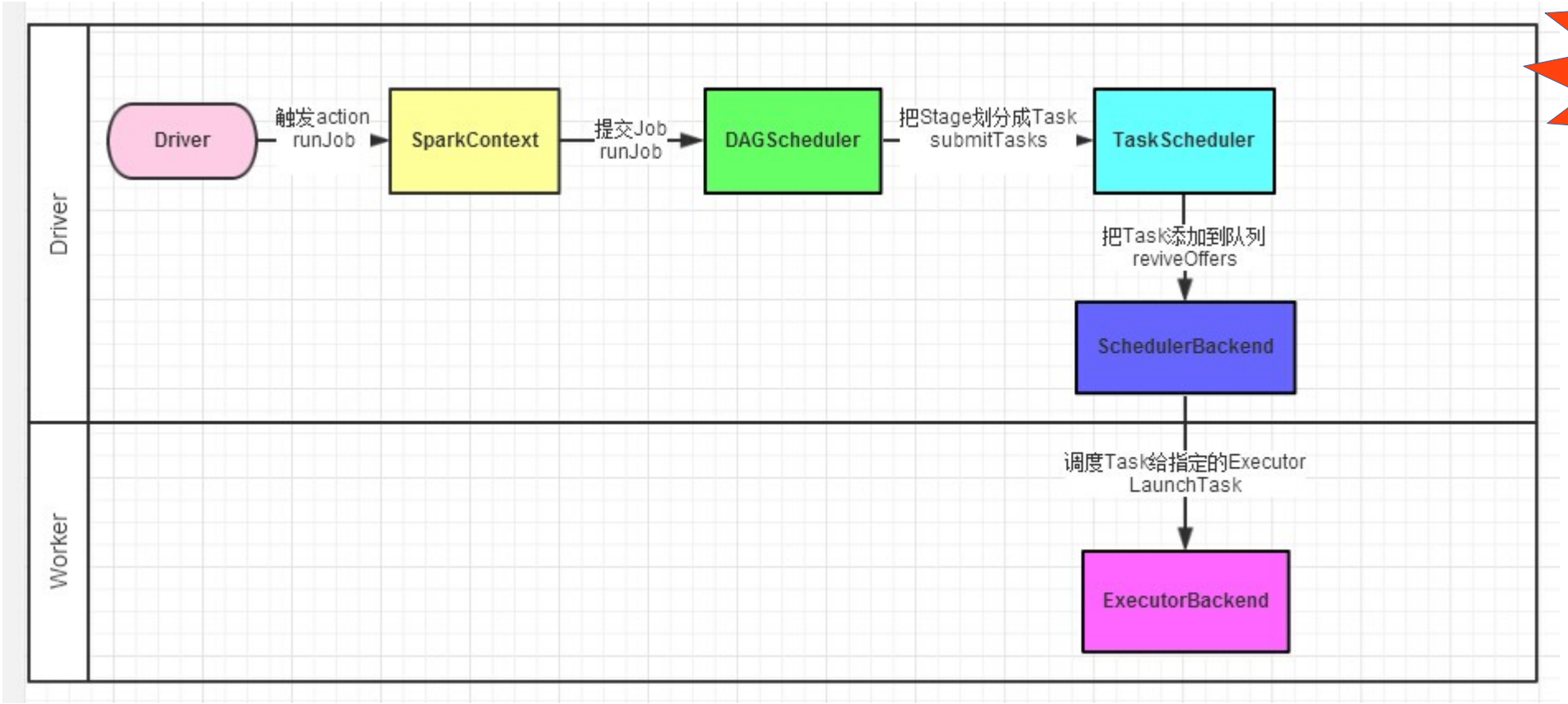
Spark 运行架构

- DAGScheduler 构建 Stage
- 记录哪个 RDD 或者 Stage 输出被物化
- 重新提交 shuffle 输出丢失的 stage
- 将 Taskset 传给底层调度器
  - spark-cluster TaskScheduler
  - yarn-cluster YarnClusterScheduler
  - yarn-client YarnClientClusterScheduler

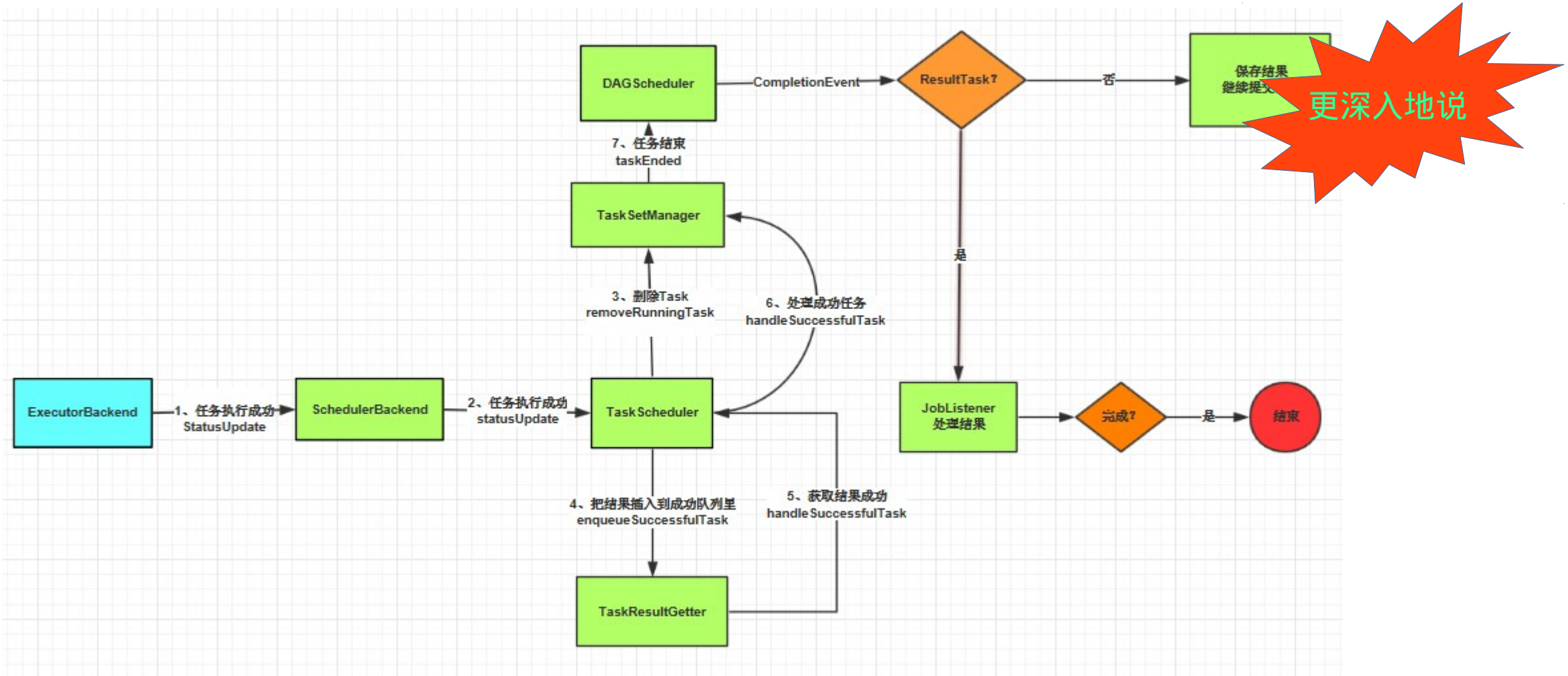
Spark 运行架构

- 为每一个 TaskSet 构建一个 TaskSetManager 实例管理这个 TaskSet 的生命周期
- 数据本地性决定每个 Task 最佳位置（process-local，node-local，rack-local and then any），
- 提交 taskset（一组 task）到集群运行并监控
- 推测执行，碰到 straggle 任务需要放到别的节点上重试
- 出现 shuffle 输出 lost 要报告 fetch failed 错误

Spark 运行架构

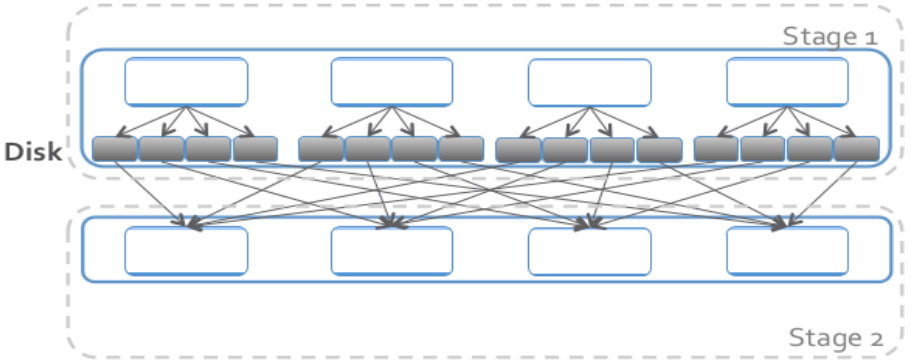

Driver
Worker
Driver
触发action
runJob
SparkContext
提交Job
runJob
DAGScheduler
把Stage划分成Task
submitTasks
TaskScheduler
把Task添加到队列
reviveOffers
SchedulerBackend
调度Task给指定的Executor
LaunchTask
ExecutorBackend
更深入地说

Spark 运行架构


DAGScheduler
CompletionEvent
ResultTask?
否
保存结果
继续提交
更深入地说
7、任务结束
taskEnded
TaskSetManager
是
3、删除Task
removeRunningTask
6、处理成功任务
handleSuccessfulTask
ExecutorBackend
1、任务执行成功
StatusUpdate
SchedulerBackend
2、任务执行成功
statusUpdate
TaskScheduler
JobListener
处理结果
完成?
是
结束
4、把结果插入到成功队列里
enqueueSuccessfulTask
5、获取结果成功
handleSuccessfulTask
TaskResultGetter

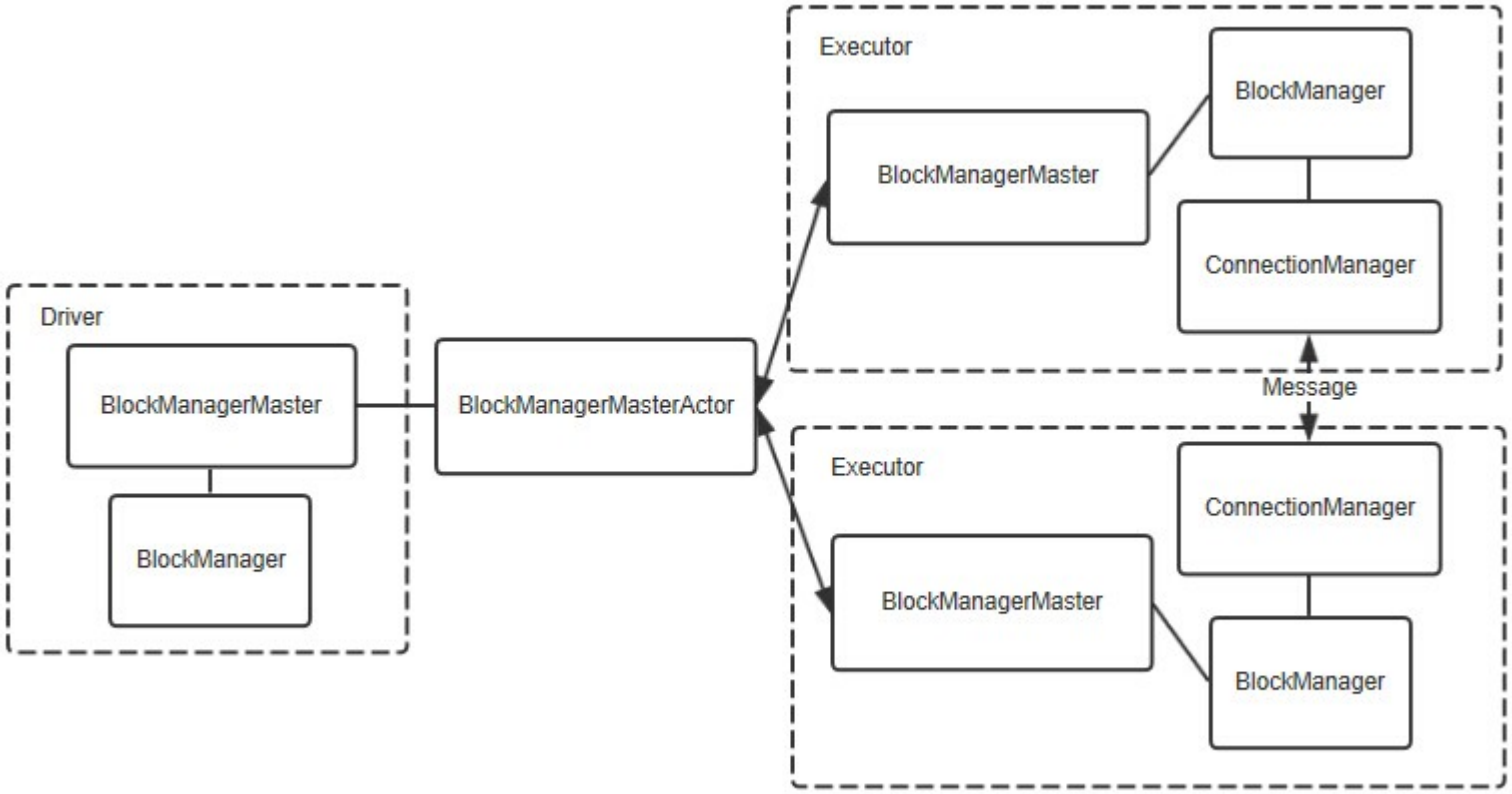
Spark 运行架构

- 二种 Task： shuffleMapTask 和 ResultTask ，被执行的 task 多数都是 shuffleMapTask
- ResultTask （FinalStage 所对应的任务），返回给 driver 的是运算结果本身
  - 结果足够小，则直接放在 DirectTaskResult 对象内
  - 超过特定尺寸（默认约 10MB ）则在 Executor 端会将 DirectTaskResult 先序列化，再把序列化的结果作为一个 Block 存放在 BlockManager 里，而后将 BlockManager 返回的 BlockID 放在 IndirectTaskResult 对象中返回给 driver
- ShuffleMapTask ，返回给 DAGScheduler 的是一个 MapStatus 对象， MapStatus 对象管理了 ShuffleMapTask 的运算输出结果在 ShuffleBlockManager 里的相关存储信息，而非结果本身，这些存储位置信息将作为下一个 Stage 的任务的获取输入数据的依据
- shuffle 的结果 patition 数目由 ShuffleDependency 中的 Partitioner 对象来决定
- Spark 内核将提供一个可拔插的 shuffle 接口。



Spark 运行架构

- 更多的细节
  - BlockManager
  - AKKA
  - NETTY
  - ...



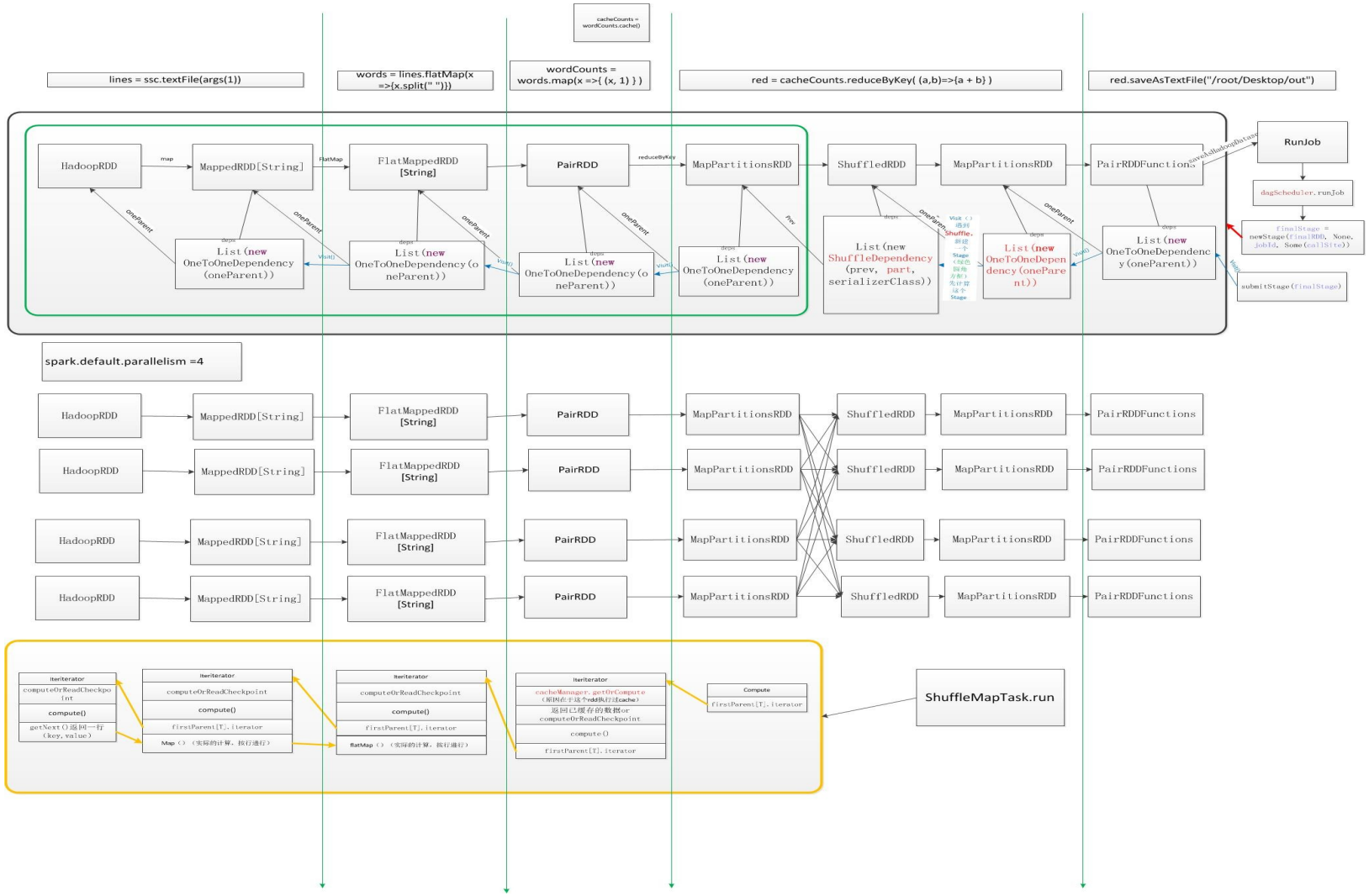
本周内容

- Spark 运行架构
- 例子解析
- Spark 在不同集群中的运行架构

例子解析

```
val lines = ssc.textFile(args(1))     //输入
val words =
lines.flatMap(x =>x.split(" "))
words.cache()          //缓存
val wordCounts =
words.map(x =>(x, 1)  )
val red = wordCounts.reduceByKey( (a,b)=>{a + b}  ,  8)
red.saveAsTextFile( “/root/Desktop/out” ,  8)        //行动
```

例子解析

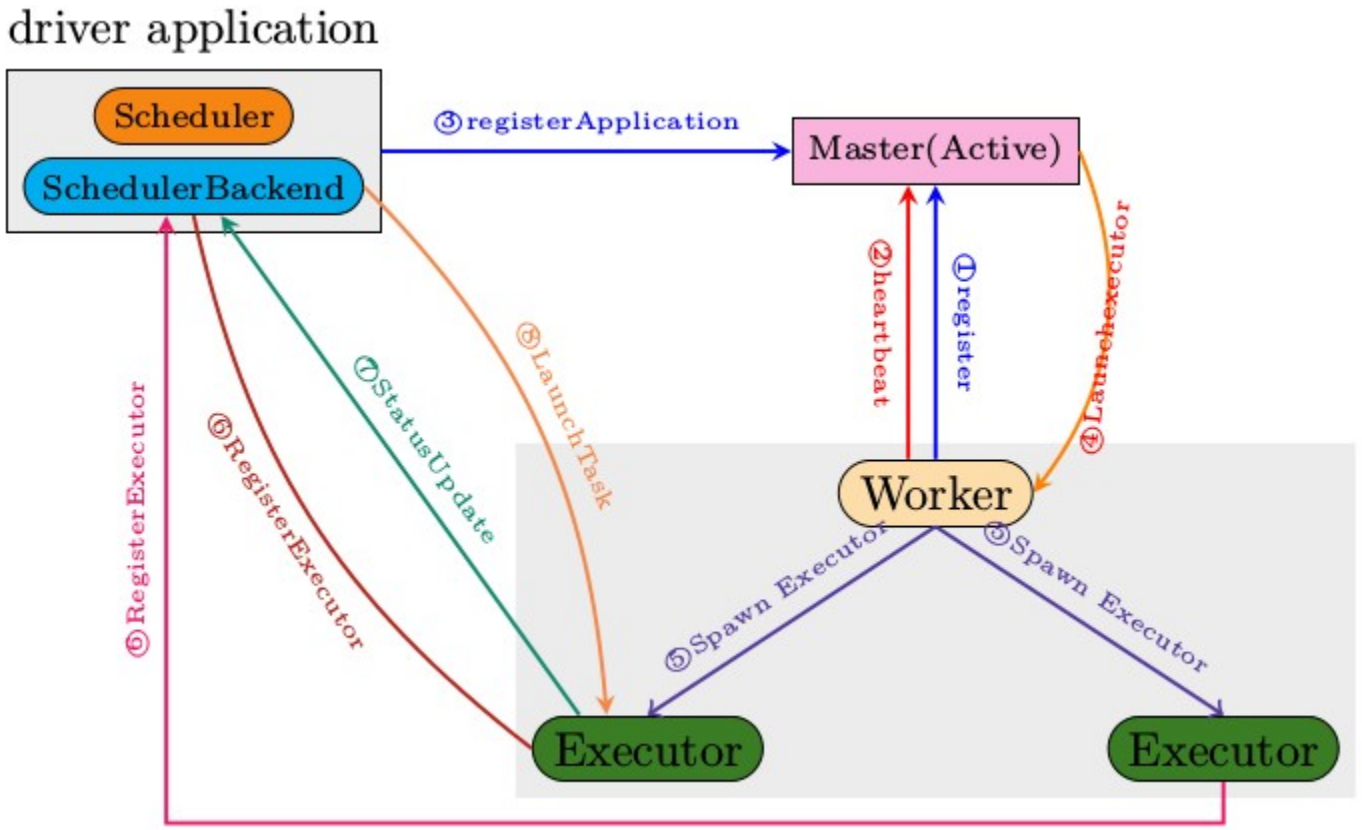

lines = ssc.textFile(args(1))
words = lines.flatMap(x =>{x.split(" ")})
wordCounts = words.map(x =>{ (x, 1) })
red = cacheCounts.reduceByKey((a,b)=>{a + b})
red.saveAsTextFile("/root/Desktop/out")
cacheCounts = wordCounts.cache()
HadoopRDD
MappedRDD[String]
FlatMappedRDD [String]
PairRDD
MapPartitionsRDD
ShuffledRDD
MapPartitionsRDD
PairRDDFunctions
RunJob
dagScheduler.runJob
submitStage(finalStage)
spark.default.parallelism =4
ShuffleMapTask.run

本周内容

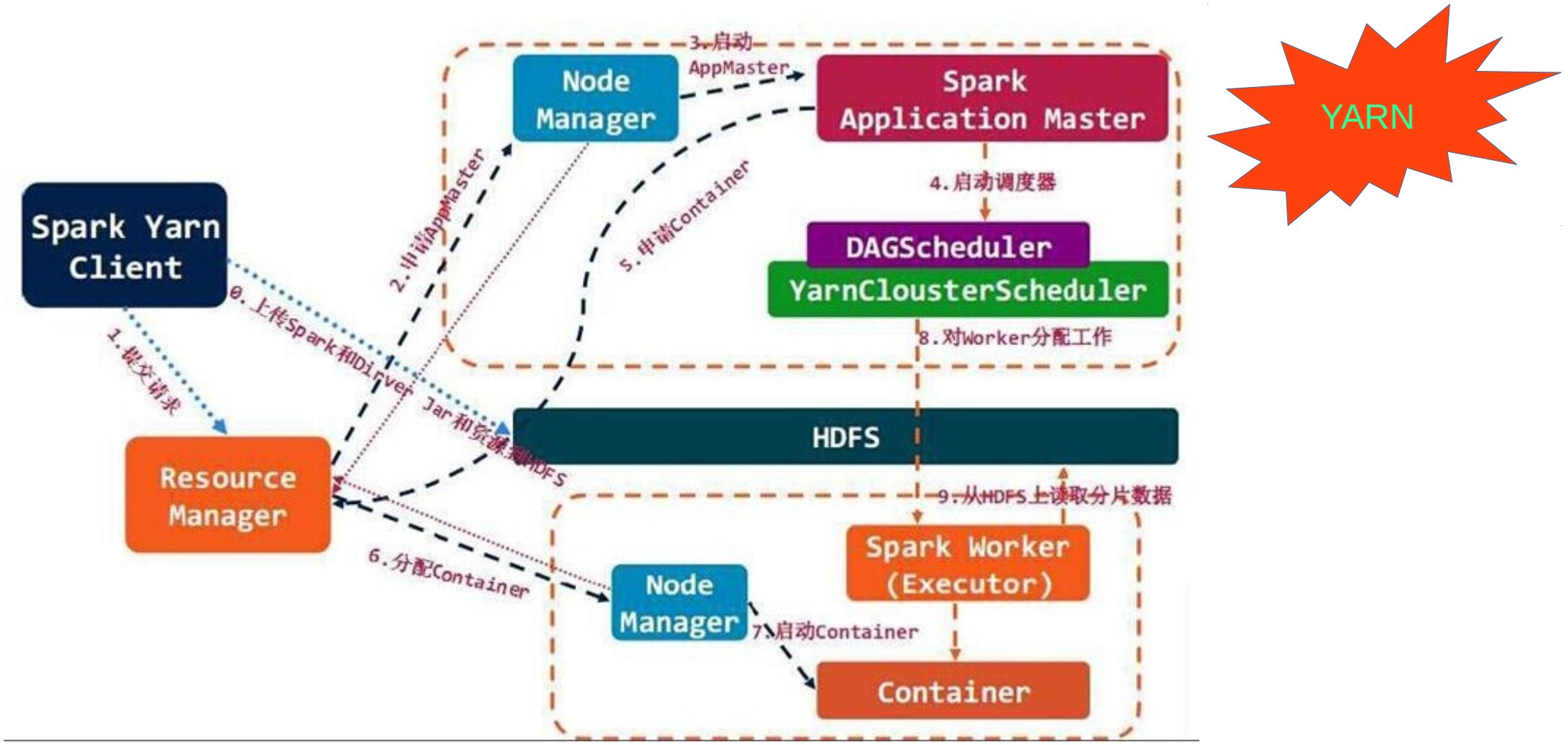
- Spark 运行架构
- 例子解析
- Spark 在不同集群中的运行架构

集群中运行架构


standalone


driver application
Scheduler
SchedulerBackend
③registerApplication
Master(Active)
②heartbeat
①register
④Launchexecutor
⑧LaunchTask
⑦StatusUpdate
⑥RegisterExecutor
⑥RegisterExecutor
Worker
⑤Spawn Executor
⑤Spawn Executor
Executor
Executor

集群中运行架构


Spark Yarn Client
Node Manager
3.启动AppMaster
Spark Application Master
YARN
2.申请AppMaster
5.申请Container
4.启动调度器
DAGScheduler
YarnClousterScheduler
8.对Worker分配工作
0.上传Spark和Dirver Jar和资源到HDFS
1.提交请求
HDFS
Resource Manager
9.从HDFS上读取分片数据
Spark Worker (Executor)
6.分配Container
Node Manager
7.启动Container
Container

集群中运行架构


```
/**
 * The entry point (starting in Client#main() and Client#run()) for launching Spark on YARN. The
 * Client submits an application to the YARN ResourceManager.
 *
 * Depending on the deployment mode this will launch one of two application master classes:
 * 1. In cluster mode, it will launch an [[org.apache.spark.deploy.yarn.ApplicationMaster]]
 *      which launches a driver program inside of the cluster.
 * 2. In client mode, it will launch an [[org.apache.spark.deploy.yarn.ExecutorLauncher]] to
 *      request executors on behalf of a driver running outside of the cluster.
 */
```

课程回顾

- Spark 应用程序的运行架构及其概念
- Spark 应用程序在不同集群中的运行架构
- 下周预告
  - Hive
  - Shark
  - SparkSQL

Thanks

FAQ 时间