南京大学
NANJING UNIVERSITY

Spark简介

南京大学软件学院
任桐炜 李传艺
{rentw, lcy}@nju.edu.cn

目录

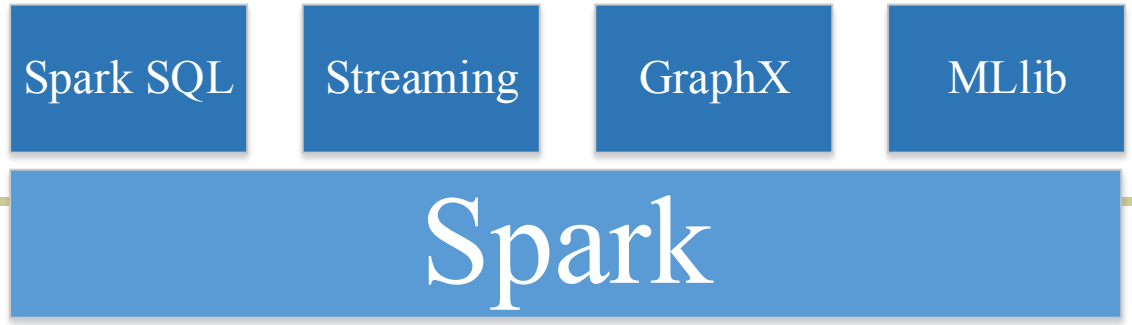
- 简介
  - 概览、功能、集群管理、存储、生态系统
- 本课程实践目标
  - 系统搭建
  - 原理理解
    - 运行模式
    - 运行过程
    - RDD
    - 资源调度
  - 项目实践
    - Spark Streaming
    - Spark GraphX
    - Spark MLlib
- 课程作业

简介(1)——概览

- 官网：**http://spark.apache.org/**
  - **Apache Spark™** is a fast and general engine for large-scale data processing.
  - 快
    - 基于内存的MapReduce计算比Hadoop快100x倍，基于硬盘的则快10x倍
  - 易用
    - 支持Scala、Java、Python和R语言开发
  - 功能强
    - Spark SQL、Spark Streaming、Spark GraphX、Spark MLlib
  - 更加通用
    - 适用于多种不同的集群管理框架
      - Standalone cluster mode、Apache Mesos、Hadoop YARN、in the Cloud（EC2）
    - 适用于多种不同的数据存储方式：数据读取接口
      - HDFS、HBase、MongoDB、Cassendra

简介(2)——功能

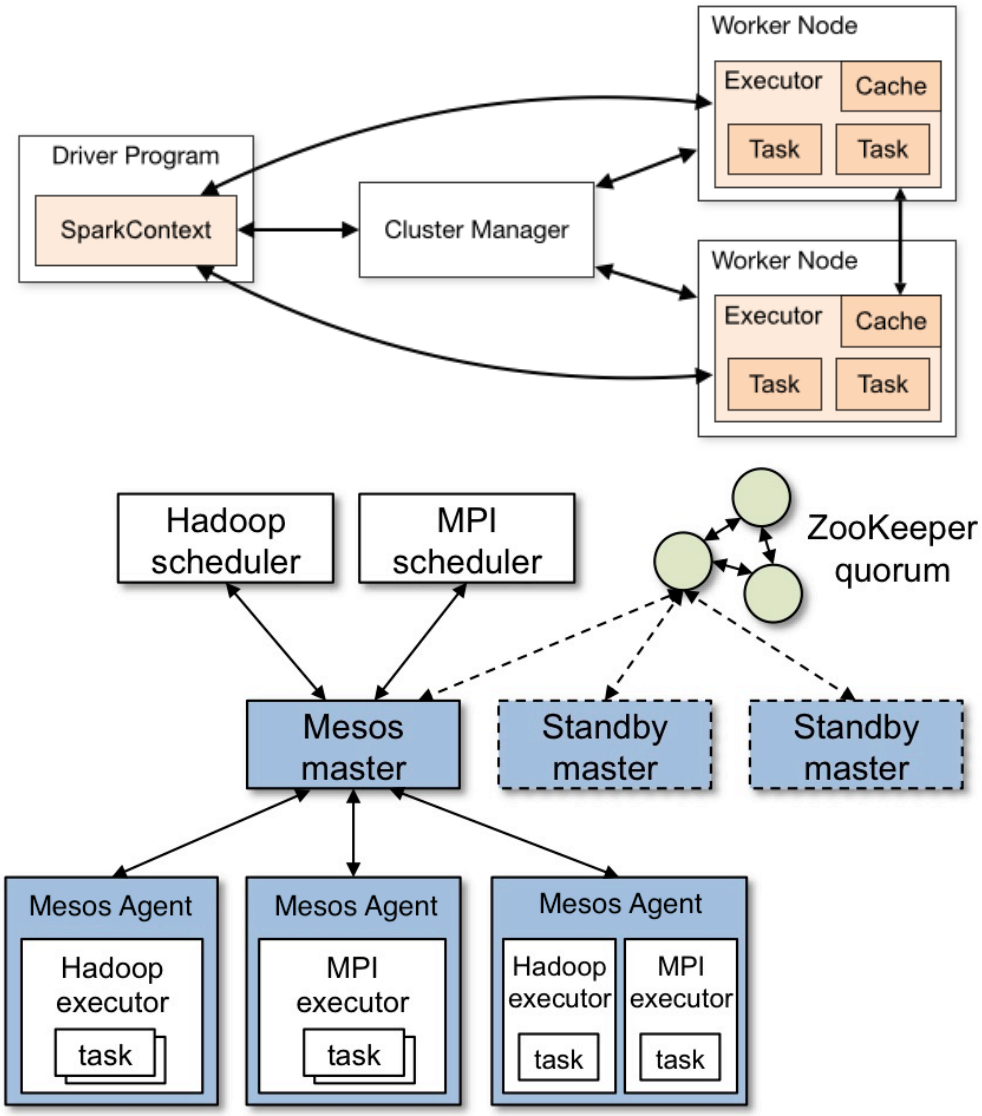
- Spark SQL
  - 前身是Shark，基于Hive的Spark SQL，代码量大、复杂，难优化和维护
  - 交互式查询、标准访问接口、兼容Hive
  - 专门用于处理结构化数据：分布式SQL引擎；在Spark程序中调用API
- Spark Streaming
  - 实时对大量数据进行快速处理，处理周期短
- Spark GraphX
  - 以图为基础数据结构的算法实现和相关应用
- Spark MLlib
  - 为解决机器学习开发的库，包括分类、回归、聚类和协同过滤等

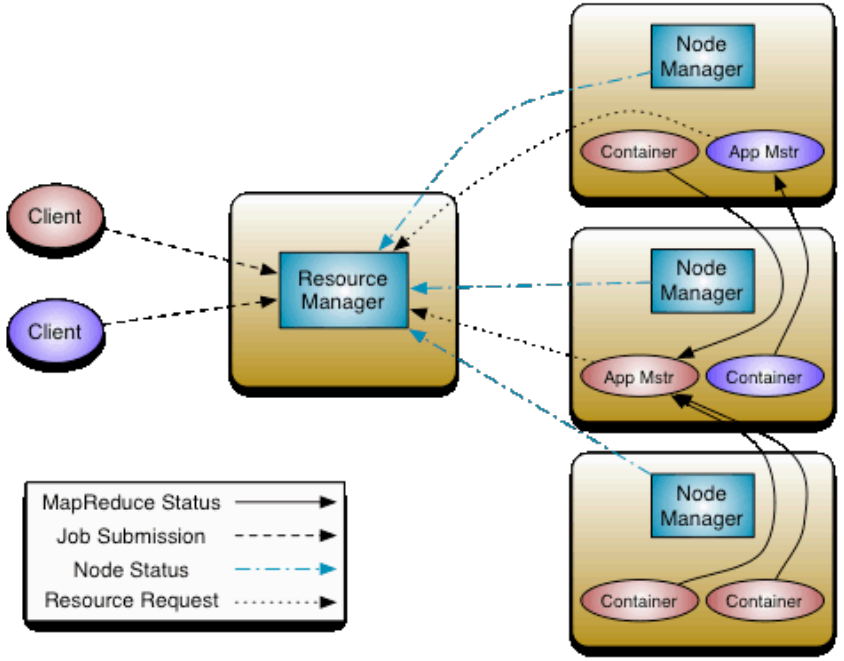


简介(3)——集群管理框架

- Standalone mode
  - 原生集群管理功能：任务调度、资源分配等
- Apache Mesos
  - 从分布式计算节点上抽象CPU, memory, storage, and other compute resources给其他框架使用，实现了静态资源分配功能
- Hadoop Yarn
  - Hadoop MapReduce的第二个版本架构，把资源管理和任务管理剥离开；实现了静态资源分配和动态资源分配功能；
- EC2
  - Amazon EC2云平台，提供一个安装了Spark、Shark和HDFS的集群，可直接登录到集群，把它当作你实验室的集群使用

简介(4)——集群管理框架图


Worker Node
Executor
Cache
Task
Task
Driver Program
SparkContext
Cluster Manager
Worker Node
Executor
Cache
Task
Task
Hadoop scheduler
MPI scheduler
ZooKeeper quorum
Mesos master
Standby master
Standby master
Mesos Agent
Hadoop executor
task
Mesos Agent
MPI executor
task
Mesos Agent
Hadoop executor
task
MPI executor
task

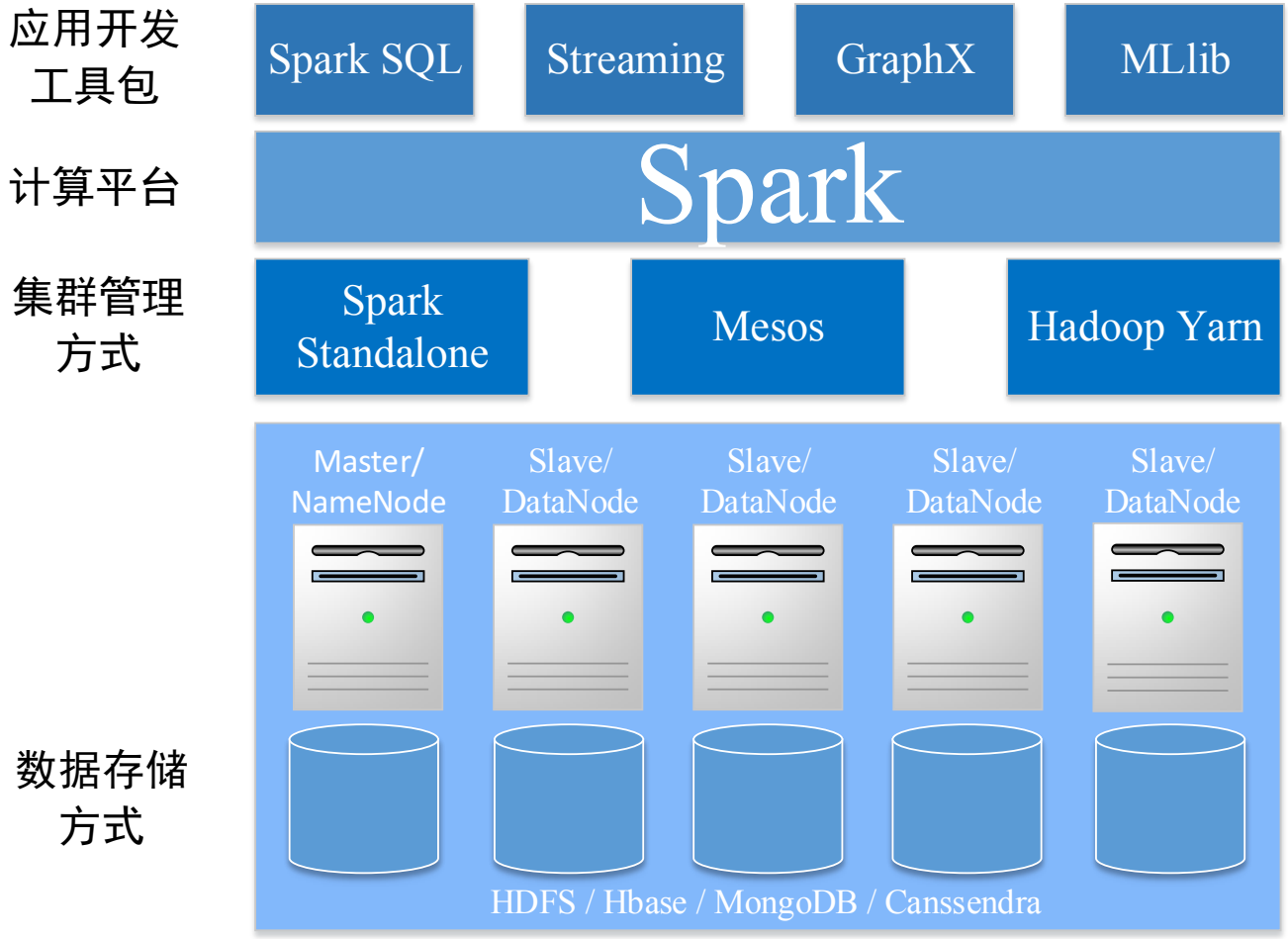

Client
Client
Resource Manager
Node Manager
Container
App Mstr
Node Manager
App Mstr
Container
Node Manager
Container
Container
MapReduce Status
Job Submission
Node Status
Resource Request

简介(5)——存储方式

- HDFS
  - Hadoop分布式文件系统
- Hbase
  - 基于HDFS的非关系型数据库(NoSQL数据库)
- MongoDB
  - 基于分布式存储的数据库，介于关系型和非关系型数据库之间
  - 是NoSQL数据库中最像关系型数据库的一个
  - 功能非常强大：支持多种开发语言、支持完全索引、支持查询等
- Cassendra
  - 基于列的分布式数据库，易扩展、模式灵活、按照范围查询等

- 它们和Spark的关系：Spark是一个计算程序，需要读取数据和存储数据，它们就是数据存储的地方

简介(6)——Spark生态系统



目录

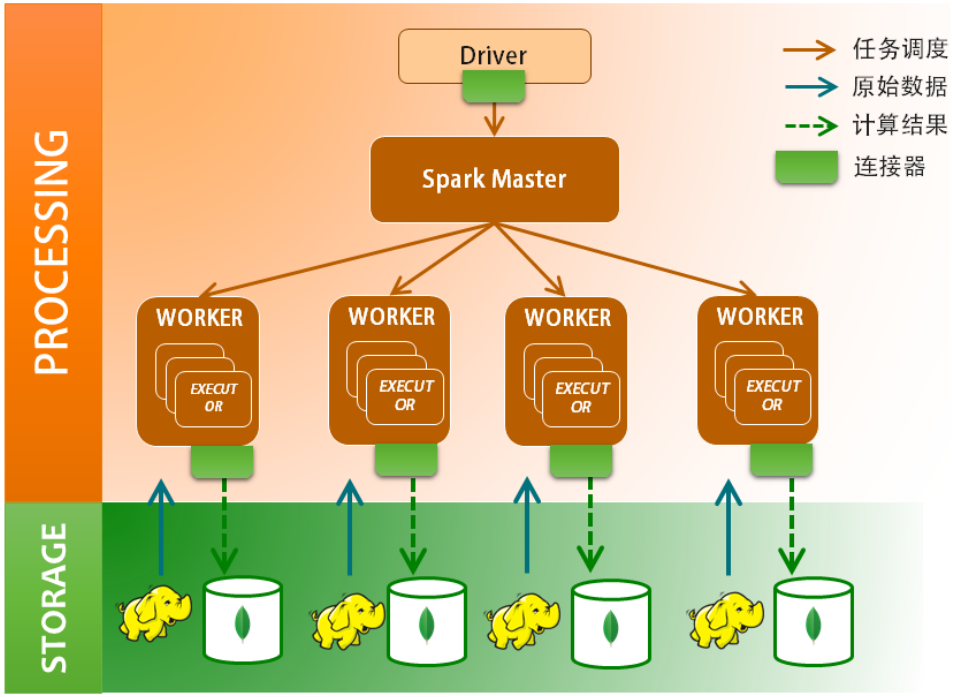
- 简介
  - 概览、功能、集群管理、存储、生态系统
- 本课程实践目标
  - 系统搭建
  - 原理理解
    - 运行模式
    - 运行过程
    - RDD
    - 资源调度
  - 项目实践
    - Spark Streaming
    - Spark GraphX
    - Spark MLlib
- 课程作业

本课程的实践目标

- Spark Standalone + HDFS + MongoDB

1. 能够搭建一个左图混合系统；
2. 利用Spark进行数据处理：
2.1 理解Spark原理
2.2 使用Spark
2.2.1 结合HDFS和MongoDB读取和存储数据
2.2.2 使用Spark Streaming
2.2.3 使用Spark GraphX
2.2.4 使用Spark MLlib

Spark MongoDB HDFS混合架构



Mongo Spark Connector 连接器

双向支持：读出与写入

条件下推

Co-Lo 部署，本地数据访问

https://github.com/mongodb/mongo-spark

目标1：系统搭建

- 安装HDFS
  - 下载Hadoop：http://hadoop.apache.org/releases.html
  - 配置Hadoop集群：http://hadoop.apache.org/docs/r3.0.0-alpha4/hadoop-project-dist/hadoop-common/ClusterSetup.html
  - HDFS自然就可以使用了
- 安装Spark
  - 下载Spark：http://spark.apache.org/downloads.html
  - 配置Spark Standalone集群：http://spark.apache.org/docs/latest/spark-standalone.html
- 安装MongoDB
  - 下载MongoDB：https://www.mongodb.com/download-center?jmp=nav#community (页面有安装指南)
- 下载Mongo Spark Connector
  - https://www.mongodb.com/products/spark-connector

目标1：补充步骤

- 确定物理机并安装操作系统（推荐Redhat）
- 如果没有网络要事先下载所有安装包
- 设置SSH无密码登陆
- 安装JDK、Scala

- 助教

目标2：利用Spark处理数据

- 理解Spark原理
  - 开发Spark程序：开发环境、程序提交、运行模式
  - 内核讲解：RDD
  - 工作机制：任务调度、资源分配
- 使用Spark
  - Spark读取、存储HDFS、MongoDB
  - Spark Streaming
  - Spark GraphX
  - Spark MLlib

开发环境搭建——IDEA

- Scala语言
  - http://blog.csdn.net/yirenboy/article/details/47440371
  - http://blog.csdn.net/oh_mourinho/article/details/52701696
  - 下载安装IDEA
  - 安装JDK
  - 安装Scala插件
  - 实战例子
- Java语言
  - http://blog.csdn.net/dream_an/article/details/54915894
- Python
  - https://www.2cto.com/net/201704/626215.html
  - （不全面，需要自己找一下）

- 到Spark、IDEA官方网站找资料

Spark原理(1)——Spark提交应用程序

```
./bin/spark-submit \
  --class <main-class> \
  --master <master-url> \
  --deploy-mode <deploy-mode> \
  --conf <key>=<value> \
  ... # other options
  <application-jar> \
  [application-arguments]
```

main方法所在的类的路径

集群管理器的地址

应用程序部署的模式，是交互式还是集群式

Spark的配置参数及值

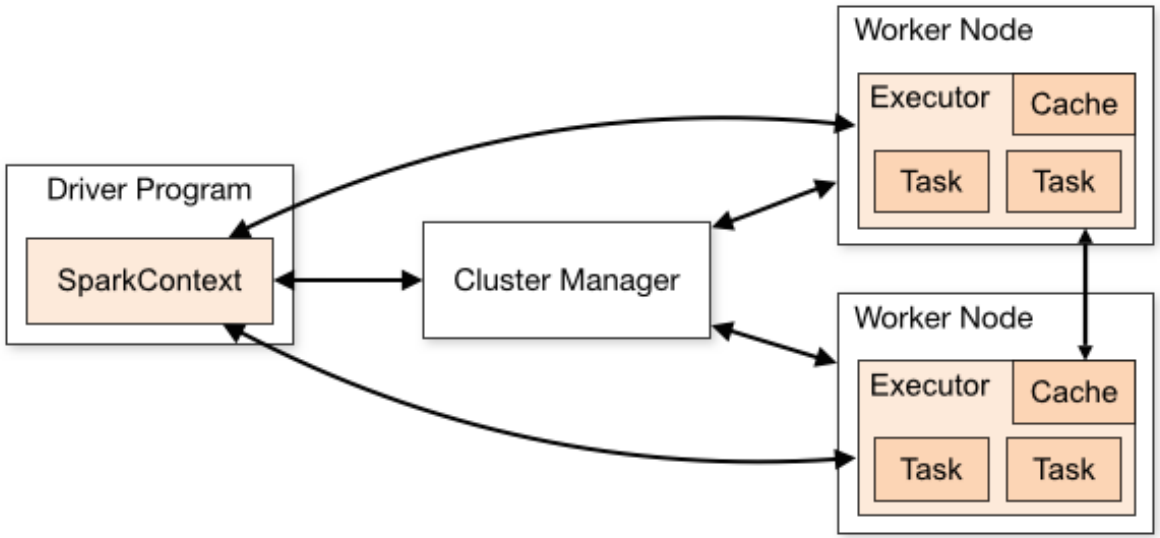
应用程序及所有依赖的包所在位置

Spark原理(2)——Driver程序

- 应用执行过程
  - Spark应用的入口程序：Driver
    - 在集群模式下用户开发的Spark程序称为Driver
- 应用的执行位置
  - Local
  - 集群
- 应用的部署模式
  - 集群模式：如果在远离（网络远离）计算节点的本地机器上提交Spark程序到集群内部执行，称为集群模式，当程序提交后，本地机器就结束了和集群的交互，程序的运行结果也不会返回到本地，只保存在集群内部。
  - 客户端模式：如果在靠近计算节点的机器上执行（Master或者就是Worker）程序，可以选择客户端模式，就是通过Spark提供的Shell执行，程序会和集群产生持续的通信，集群的任务执行完成后，会把执行结果返回到客户端。

Spark原理(3)——SparkContext

- SparkContext对象
  - 使得应用程序能够和集群进行沟通，实现CPU、内存等资源的分配
  - 每个Driver程序都有一个SC对象
    - 如果是交互式编程，则Spark Shell会自动创建一个SC对象
  - 程序启动后，SC会告诉集群管理器在Worker Node上创建执行器
    - 执行器是每一个Spark程序在计算节点上专属的进程
  - 程序代码会发送到对应的Worker Node上
  - SC分发任务(Task)到各个执行器

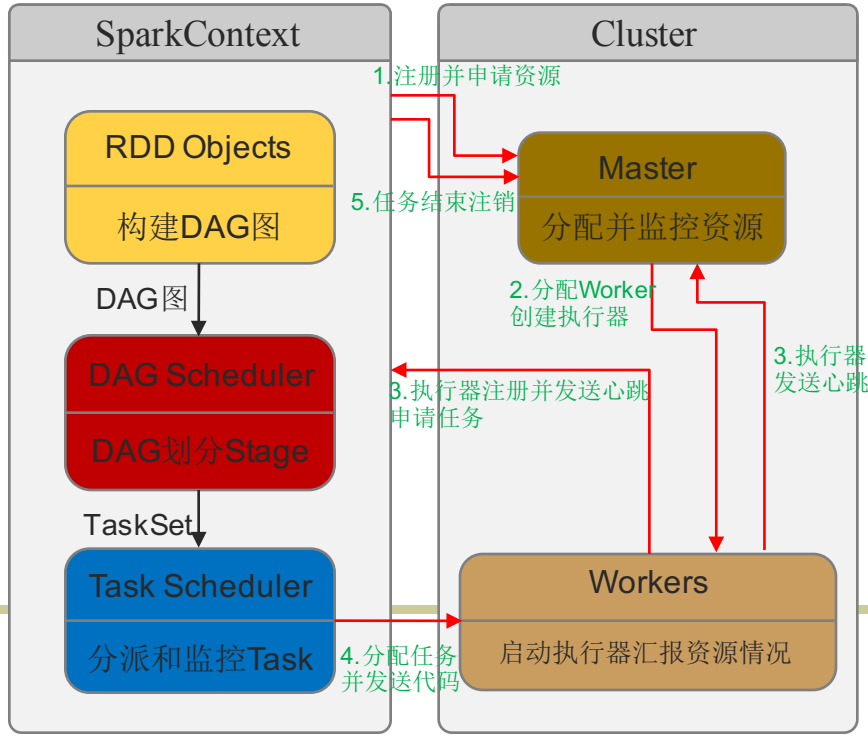


Spark原理(4)——其它相关名词

- Application
  - 独立的Spark程序
- Master（Cluster Manager），Worker
- 执行器（Executor）
  - Spark程序在对应计算节点上启动的专属线程，负责执行Task
- Job
  - 一次RDD Action称为一个Job，是一个概念
- Stage
  - 介于Job和Task之间，是一个Task集合
- Task
  - 在执行器上执行的最小单元，例如在某个计算节点对一个RDD的分区进行的Transformation操作

Spark原理(5)——具体执行过程

- 分为两部分
  - Driver程序
    - 客户端模式下，Driver在本地，不会把Driver提交给Master，而直接提交任务
    - 集群模式下，首先Driver被提交给Master，Master首先找一个Worker运行Driver
  - 集群节点



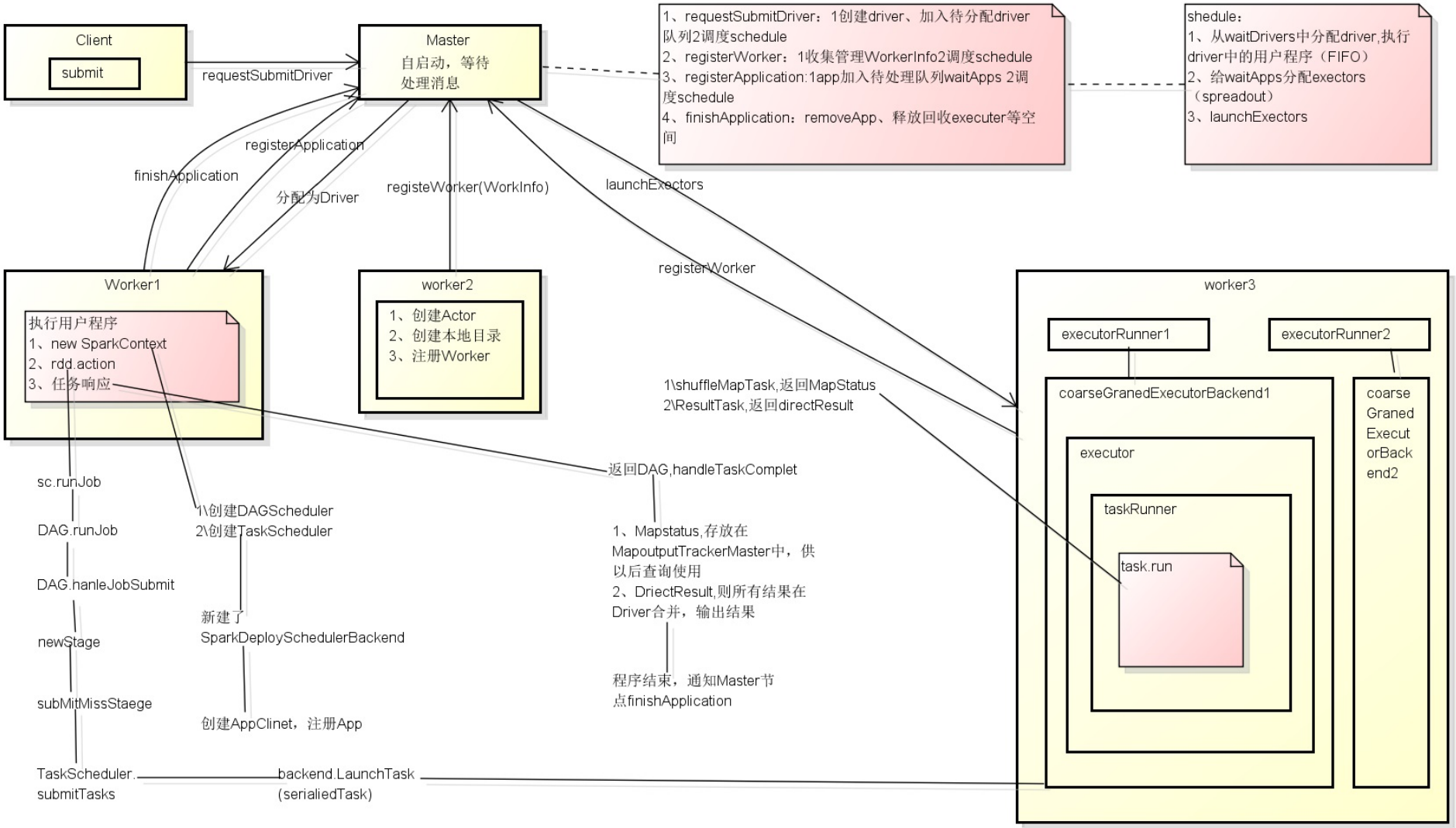
执行器启动后运行Driver程序的节点不需要和管理器有太多交互，而与worker间交互密切，因此Driver程序应该放在距离Worker节点较近的地方。

1. RDD
2. DAG Scheduler
3. Task Scheduler

4. Master分配资源

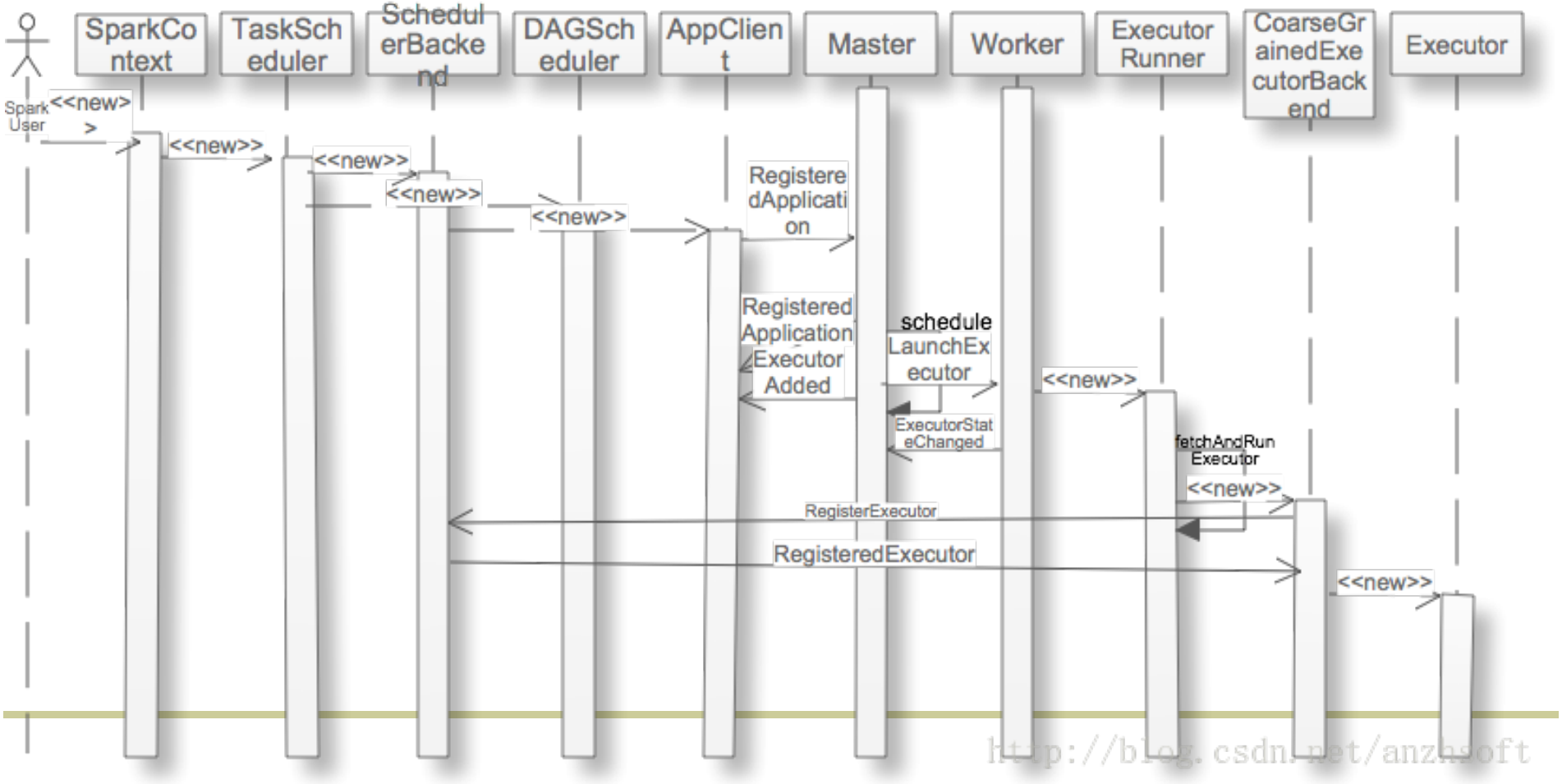
详细的应用执行过程展示

http://blog.csdn.net/thomas0yang/article/details/50352261



结合源代码分析的Spark执行过程解读

http://blog.csdn.net/u011007180/article/details/52388783
http://blog.csdn.net/zero__007/article/details/53121003



Spark原理(6)——RDD(1)

- Job、Task、DAGScheduler的定义均依赖RDD
- Resilient Distributed Dataset 弹性分布式数据集
- 是Spark的核心数据结构
  - 是一个数据集，就像Array、List和Set等一样的数据结构
  - 内容是平铺的，可以顺序遍历，像Array和List一样
  - RDD是分布式存储的，支持并行计算
  - RDD的分布是弹性的，某些操作使不同的数据块重新汇聚和分布
  - RDD是只读的
  - RDD可以缓存在内存中
  - RDD可以通过重复计算获得（实现高可靠性）

Spark原理(6)——RDD(2)

- RDD定义：包含5个属性
  - 分区列表：记录了数据块所在的分区位置；一个RDD对应的数据是切分为数据块存储在集群的不同节点上的
    - @transient def getPartitions_ : Array[Partition] = null
  - 依赖列表：记录了当前这个RDD依赖于哪些其它的RDD
    - private var dependencies_ : Seq[Dependency[_]] = null
  - 计算函数compute，用于计算RDD各个分区的值
    - def compute(split: Partition, context: TaskContext): Iterator[T]
  - 可选：分区器，子类可以重新指定新的分区方式：Hash和Range
    - @transient val partitioner: Option[Partitioner] = None
  - 可选：计算各分区时优先的位置列表
    - 例如从HDFS文件生成RDD时，HDFS文件所在位置就是对应生成的RDD分区所在位置的优先选择
    - protected def getPreferredLocations(split: Partitiion): Seq[String] =Nil

Spark原理(6)——RDD(3)

- RDD的种类：继承自基础的RDD类
  - HadoopRDD，newAPIHadoopRDD
  - MapperedRDD
  - FlatMappedRDD
  - FilteredRDD
  - PairedRDD
- 从外部数据读入并初始化为RDD
  - 文件系统：单个文件（textFile）、文件目录（wholeTextFiles）
  - Hadoop Inputformat：不同的读取函数返回HadoopRDD
- 从Driver数据集生成RDD
  - SparkContext的parallelized函数
- 对RDD的一些操作后产生新的RDD

Spark原理(6)——RDD(4)

- 对RDD的操作
  - Transformation
    - 接收一个RDD作为输入，返回一个新的RDD
    - 但是并不会真实执行计算，只是定义生成的新RDD并且设置其与前一个RDD的依赖关系
    - 操作对RDD依赖关系的区分
      - 窄依赖：只依赖前一个RDD的确定的几个分区
      - 宽依赖：依赖前一个RDD的所有分区
  - Action
    - 输入还是RDD，但是输出就不是RDD了
    - 调用了Action之后，整个从输入RDD到输出值的Transformation链条会被执行

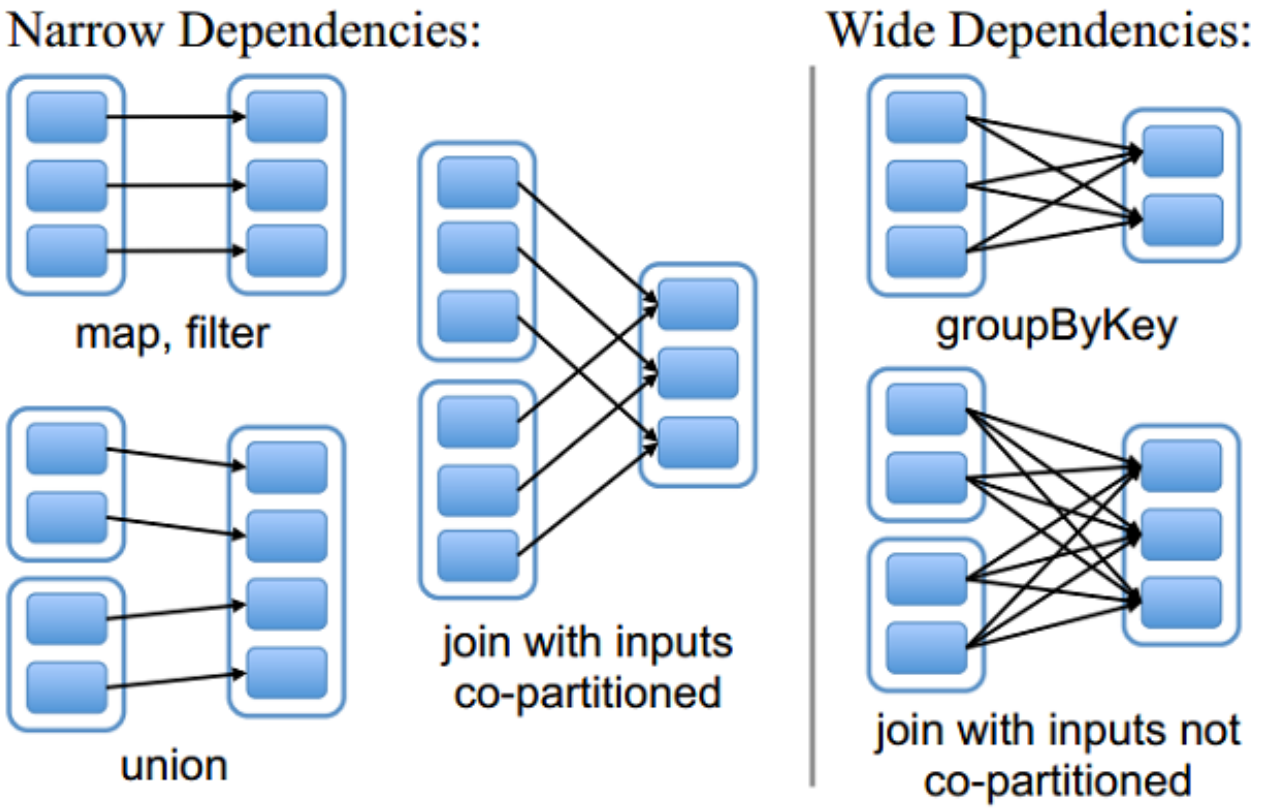
Spark原理(6)——RDD(5)

- Transformation

Transformation操作	说明
map(func)	对源RDD中的每个元素调用func，生产新的元素，这些元素构成新的RDD并返回
flatMap(func)	与map类似，但是func的返回是多个成员
filter(func)	对RDD成员进行过滤，成员调用func返回True的才保留，保留的构成新RDD返回
mapPartitions(func)	和map类似，但是func作用于整个分区，类型是Iterator<T> => Iterator<T>
mapPartitionsWithIndex(func)	…
union(otherDataset)	…
reduceByKey(func, [numTasks])	对<key, value>结构的RDD聚合，相同key的value调用reduce，func是(v,v)=>v
join(otherDataset, [numTasks])	…

Spark原理(6)——RDD(6)

- Transformation操作对应的依赖关系（例举）



Spark原理(6)——RDD(5)

- Action

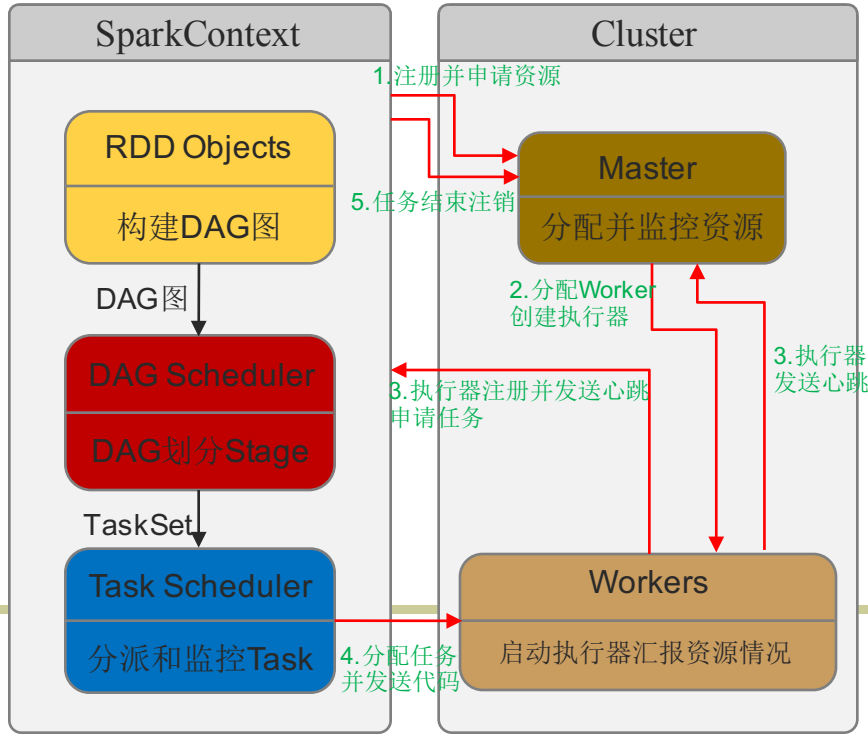
Action操作	说明
reduce(func)	对RDD成员使用func进行reduce操作，func接收两个值只返回一个值；reduce只有一个返回值。func会并发执行
collect()	将RDD读取到Driver程序里面，类型是Array，要求RDD不能太大
count()	返回RDD成员数量
first()	返回RDD第一个成员，等价于take(1)
take(n)	…
saveAsTextFile(path)	把RDD转换成文本内容并保存到指定的path路径下，可能有多个文件；path可以是本地文件系统路径，也可以是HDFS路径，转换方法是RDD成员的toString方法
saveAsSequenceFile(path)	…
foreach(func)	对RDD的每个成员执行func方法，没有返回值

Spark Map-Reduce 实例

```
object MyFunctions{
        def lineSplit(line: String) : Array[String] = {
                line.split(" ")
        }
}
val textFile = sc.textFile("README.md")
val words = textFile.flatMap(MyFunctions.lineSplit)    #先func再map
val wordPairs = words.map(word => (word, 1))           #先func再map
val wordCounts = wordPairs.reduceByKey((a, b) => a + b)
wordCounts.collect()
```

Spark原理(7)——DAG Scheduler(1)

- 把一个Spark Job转换成Stage的DAG（Directed Acyclic Graph有向无环图）
- 根据RDD和Stage之间的关系找出开销最小的调度方法，然后把Stage以TaskSet的形式提交给TaskScheduler



Job：一次Action的所有工作就是一个Job

如何划分Stage？

1. 以宽依赖为分界
2. 宽依赖之前的所有Transformation为一个Stage

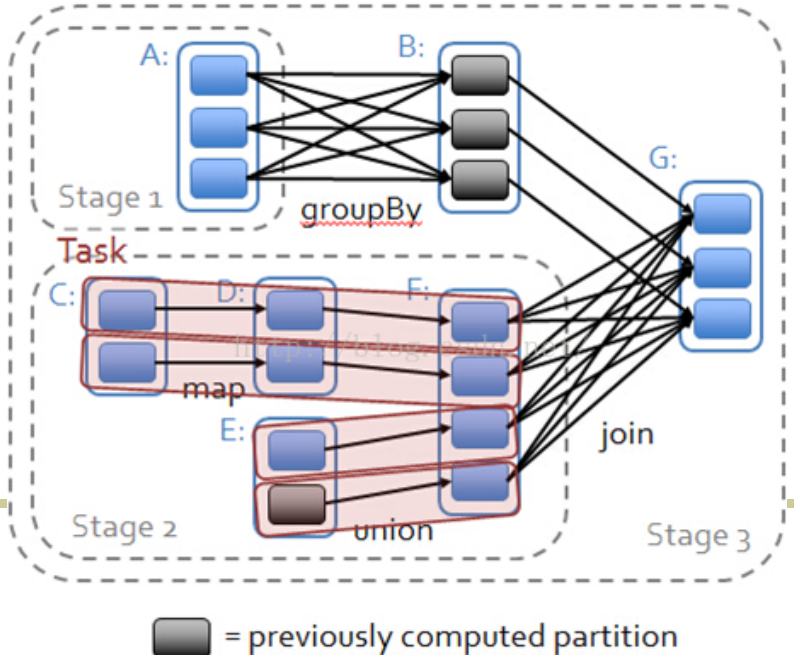
数据是否需要重组

一个Stage包括一个或多个Transformation

一个Transformation在一个RDD分区上执行是一个Task

Spark原理(7)——DAG Scheduler(2)

- 从Action开始构造Stage DAG
- 最后的Action构造一个ResultStage
- 填补Stage需要的输入RDD，并根据对RDD的依赖关系划分Stage
- 直到所有的RDD依赖关系都补充完整
- 开始按照DAG计算需要的RDD
- 将每一个Stage封装程TaskSet交给TaskScheduler调度

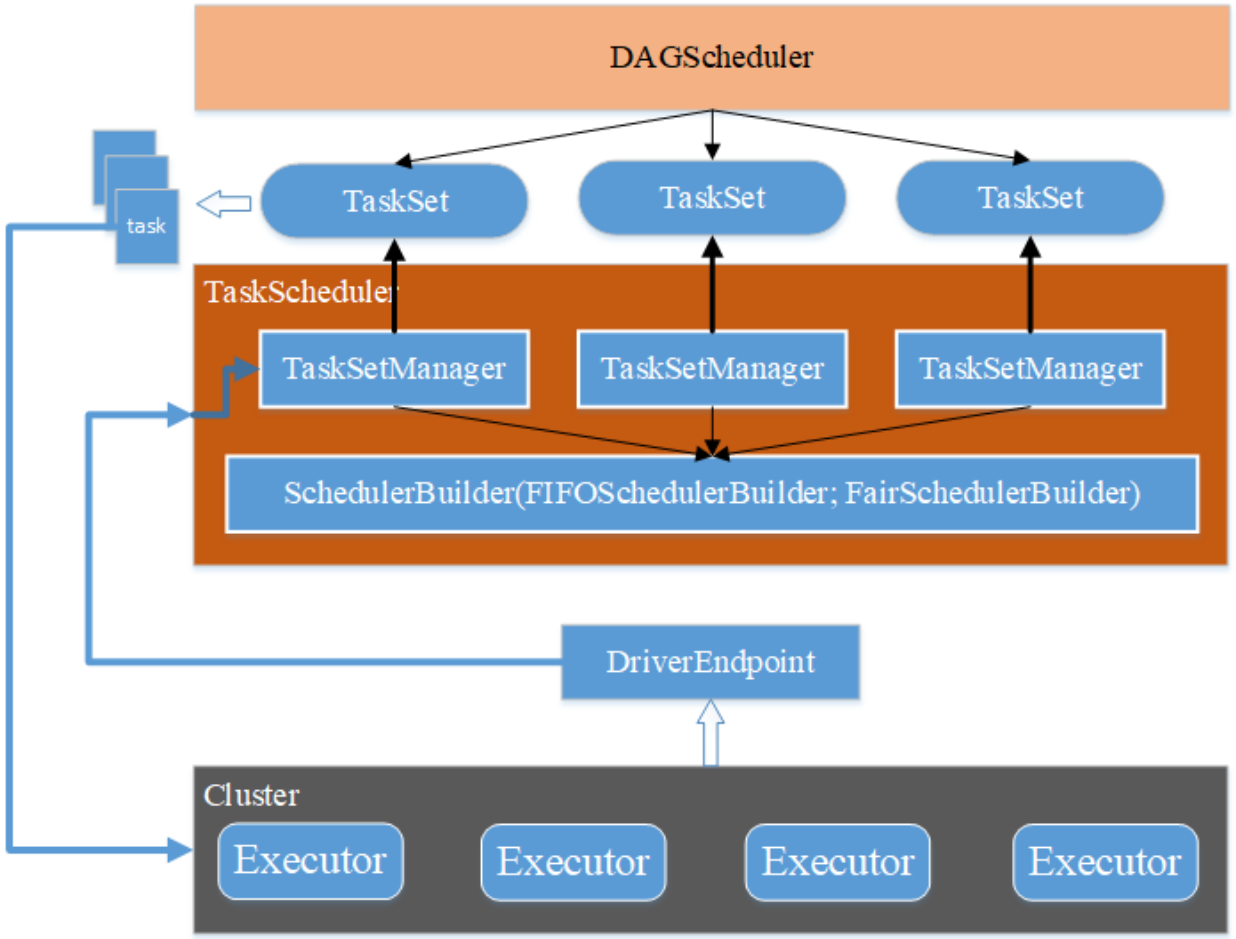


Spark原理(8)——TaskScheduler(1)

- DAGScheduler最后一步：计算Task最佳执行位置；为每一个RDD分区创建Task；将一个Stage的Task封装成TaskSet交给TaskScheduler
- TaskScheduler为每个Stage的TaskSet创建一个TaskSetManager，负责跟踪task set中所有task，包括失败重启等
- 使用SchedulerBuilder管理TaskSetManager，决定TaskSet的调度顺序
  - FIFOSchedulerBuilder：先来先调度
  - FairSchedulerBuilder：公平调度，按照资源情况调度
- driverEndpoint通过makeOffers找出计算资源
- TaskScheduler根据计算资源为TaskSet中每一个Task分配Executor
  - 遍历TaskSet
  - 使用TaskSetManager遍历TaskSet里面的Task
  - 按照“本地性”分配Executor

Spark原理(8)——TaskScheduler(2)

- DAGScheduler
- TaskSet
- TaskSetManager
- SchedulerBuilder
  - FIFOSchedulerBuilder
  - FairSchedulerBuilder
- DriverEndpoint
  - makeOffers
- TaskScheduler
  - resourceOffers
- TaskSetManager
  - resourceOffer



Spark原理(9)——资源分配

- 只考虑Standalone集群
  - 最简单的集群模式
- Application调度
  - 只支持先进先出(FIFO)的应用调度方式
  - 如何给应用分配资源？
    - 静态分配：简单
      - 一次性分配所有资源，直到应用退出才回收
    - 不支持：动态分配：复杂
      - 在运行过程中按照需要进行分配，有请求、移除策略
- Job调度
  - FIFO
  - Fair：调度池共享资源，Job加入一个池默认行为（池内Job按照FIFO调度）
- TaskSet调度
  - FIFO
  - Fair

目标2：利用Spark处理数据

- 理解Spark原理
  - 开发Spark程序：开发环境、程序提交、运行模式
  - 内核讲解：RDD
  - 工作机制：任务调度、资源分配

- 使用Spark
  - Spark读取、存储HDFS、MongoDB
  - Spark Streaming
  - Spark GraphX
  - Spark MLlib

Spark实践(1)——数据读取、存储

- 从数据源到RDD
  - parallelize()
  - textFile(path)
  - hadoopFile(path)
  - sequenceFile(path)
  - objectFile(path)
  - binaryFiles(path)
  - …
- 从RDD目标数据
  - saveAsTextFile(path)
  - saveAsSequenceFile(path)
  - saveAsObjectFile(path)
  - saveAsHadoopFile(path)
  - …

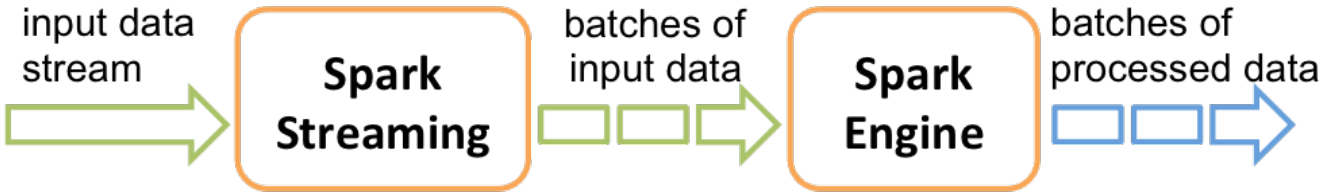
读取、存储MongoDB例子核心部分：

```
import com.mongodb.hadoop.MongoOutputFormat;

Configuration config = new Configuration();
config.set("mongo.input.uri",
          "mongodb://10.1.50.124:27017/ligf.student");
config.set("mongo.output.uri",
          "mongodb://10.1.50.124:27017/ligf.test");

JavaPairRDD<Object, BSONObject> mongoRDD =
    sc.newAPIHadoopRDD(config,
        com.mongodb.hadoop.MongoInputFormat.class,
        Object.class, BSONObject.class);

rdd.saveAsNewAPIHadoopFile("file:///bogus",
          classOf[Any], classOf[Any],
          classOf[com.mongodb.hadoop
          .MongoOutputFormat[Any,Any]],
          config)
```

Spark实践(2)——Spark Streaming(1)

- 实时对大量数据进行快速处理：处理周期短；连续不断地计算
- 计算基本过程

- 数据分批



Spark实践(2)——Spark Streaming(2)

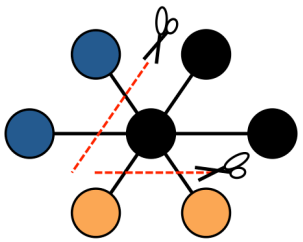
- StreamingContext (类似SparkContext)
- Dstream (类似RDD)
  - 内部是通过RDD实现的
  - 每个时间点对应一个RDD
- 程序执行过程
  - 初始化StreamingContext
  - 创建Dstream
  - 对DStream的操作
    - Transformation操作
      - 参考RDD
      - transform操作：直接操作内部RDD
      - 窗口(Window)操作：合并几个时间点的RDD
    - Output操作
      - print()
      - saveAsTextFiles/saveAsObjectFiles()
      - foreachRDD(func): func一般是对RDD的Action操作

Spark实践(3)——Spark GraphX(1)

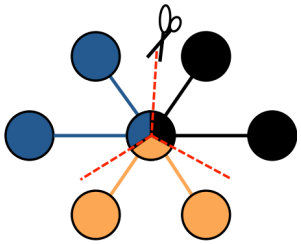
- 图计算
  - 以图为数据结构基础的相关算法及应用
- 数据结构——图
  - G=<V, E>
    - V表示顶点集合；E表示边集合
  - 两个顶点u,v相连，表示为边(u,v)
    - 无向边；任意边为无向边则称为无向图
    - 有向边；每一条都为有向边则称为有向图
- GraphX提供的API
  - 图生成
  - 图数据访问
    - 查询顶点数、边数；计算某个点的入度、出度等。
  - 图算法
    - 遍历顶点、边；计算连通性；计算最大子图；计算最短路径；图合并等

Spark实践(3)——Spark GraphX(2)

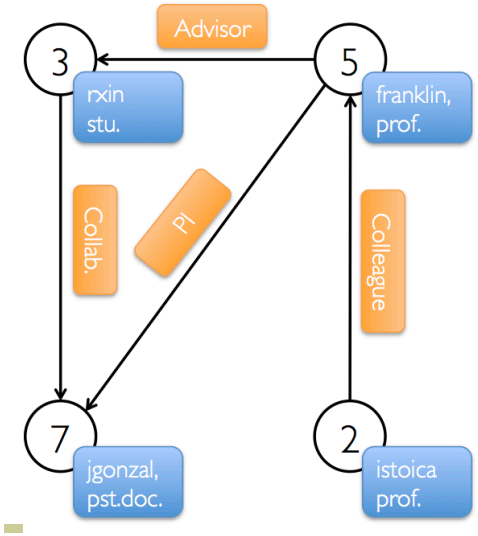
- GraphX的实现
  - 核心是Graph数据结构，表示有向多重图
    - 两个顶点间允许存在多条边，表示不同含义
  - Graph由顶点RDD和边RDD组成
  - Graph的分布式存储方式



Edge Cut



Vertex Cut

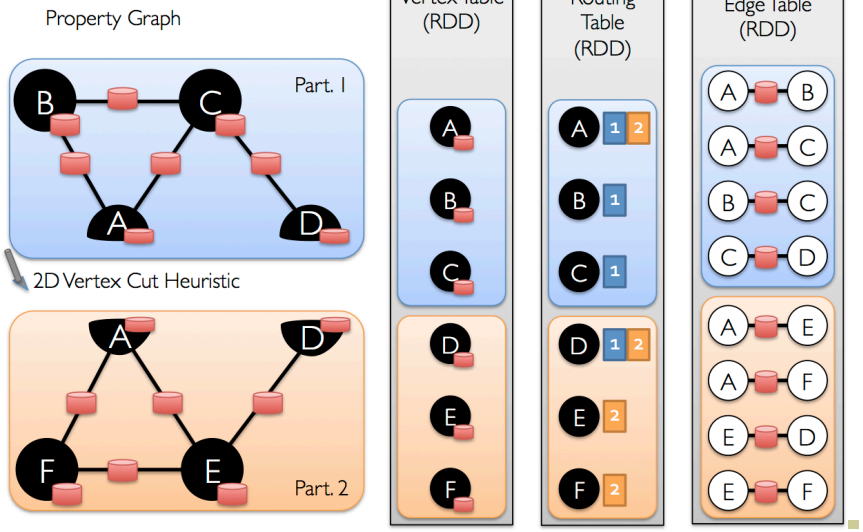


Vertex Table

Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

Edge Table

SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI



Spark实践(3)——Spark GraphX(3)

- 图生成
  - 读入存储关系信息的文件，构造EdgeRDD：eRDD = sc.textFile()
  - 从Edge RDD构造Graph：graph = Grapth.fromEdges(eRDD)
- 基本接口
  - 获取边数：numEdges；获取节点数：numVertices
  - 获取入度、出度：inDegrees, outDegrees
  - 结构操作：reverse，subgraph, mask
- 关联类操作
  - joinVertices:
  - outerJoinVertices:
- 聚合类操作
  - 分布式遍历所有的边，执行自定义的sendMsg函数；
  - 在节点上执行mergeMsg函数
- http://spark.apache.org/docs/latest/graphx-programming-guide.html#summary-list-of-operators 官网API说明

Spark实践(4)——Spark MLlib(1)

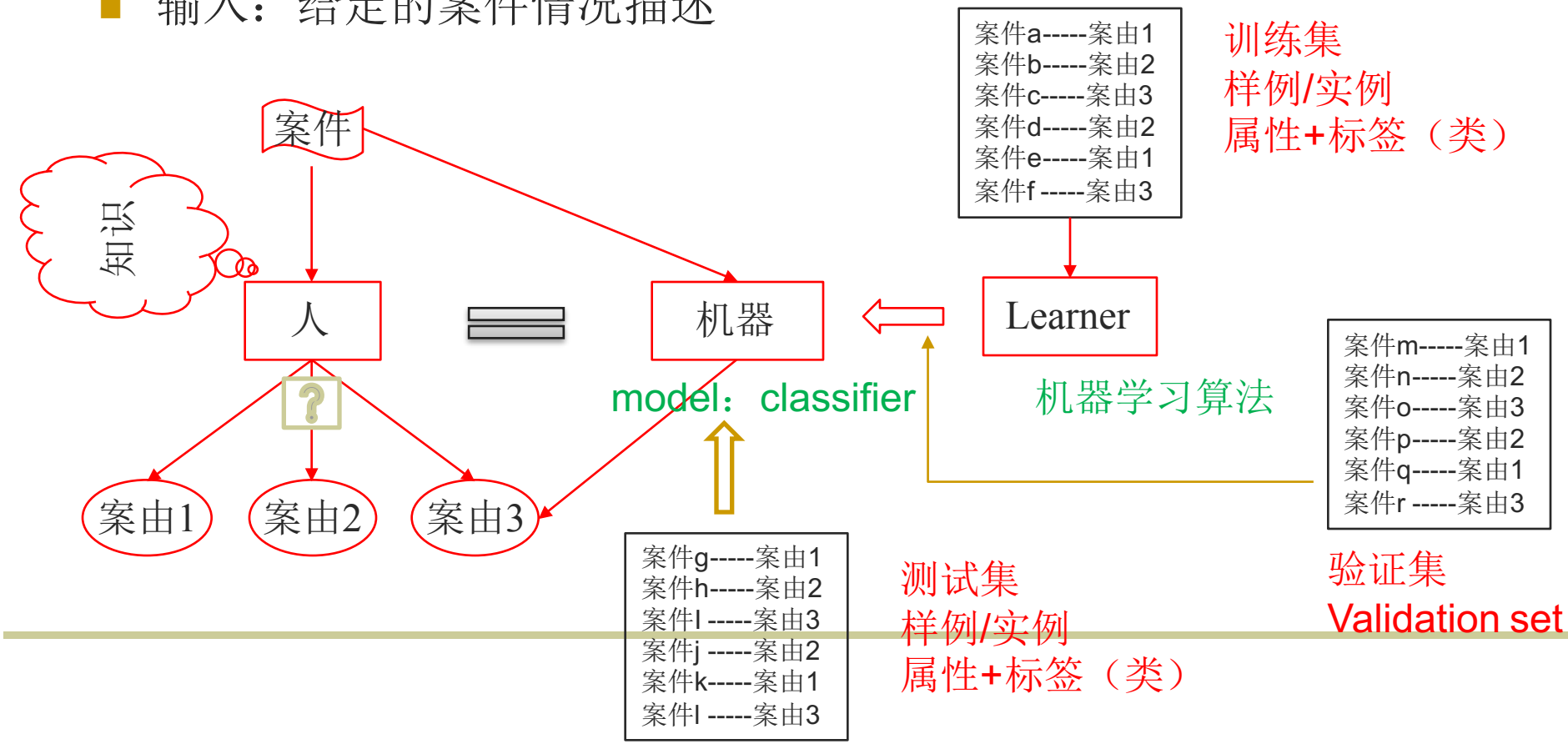
- Spark为机器学习问题开发的库
  - 分类、回归、聚类和协同过滤等
- 机器学习简介

学习方法的类别——按输入数据分

- 监督学习
  - Decision trees, neural networks, nearest-neighbor algorithms, Bayesian learning, hidden Markov models, SVM
  - 分类问题
    - 输出是非连续的有限集合
  - 回归问题
    - 输出是连续的实数集合
- 无监督学习
  - 聚类问题
    - 基于元素的相似度计算规则
- 半监督学习
- 强化学习
  - Markov Decision Processes, temporal difference learning, Q-learning

分类问题的例子

- 判定一个给定案件的案由
- 目标：离婚纠纷、产品质量纠纷、债务纠纷、其它
- 输入：给定的案件情况描述



Spark实践(4)——Spark Mllib(2)

- Spark为机器学习问题开发的库
  - 分类、回归、聚类和协同过滤等
- 机器学习简介
- MLlib简介
  - 基础数据类型
    - 向量
    - 带标注的向量：用于监督学习
    - 模型：训练算法的输出
  - 主要的库
    - mllib.classification:分类算法，二分类、多分类、逻辑回归、朴素贝叶斯、SVM等
    - mllib.cluster：聚类，K-Means、LDA等
    - mllib.recommendation：使用协同过滤的方法做推荐
    - mllib.tree：决策树、随机森林等算法
  - 其它常用库
    - mllib.evaluation：算法效果衡量方法

实践作业

- 重点来了！
- 困了、已经睡着的同学都清醒一下，占用你们几分钟时间！

- 感谢助教黄圣达同学一起设计了实践作业！

Spark实践(5)——作业1

- 流数据处理
- 第一阶段
  - 数据爬取和存储到MongoDB（作为以后实验的数据源）
  - 流数据模拟
- 第二阶段
  - 流数据处理
  - 结果展示
- 选题范围
  - 京东不同种类商品的评论——统计关注点变化曲线
    - 笔记本电脑、手机、微单
  - “雪球网”每支股票的评论和用户拥有的股票信息——情绪曲线
  - 阿里商品销售数据集（商店和用户）——按时间排序后统计店铺销售单数曲线
  - 豆瓣用户小组信息——更新用户画像（假设爬取顺序就是加入顺序）

Spark实践(5)——作业2

- GraphX
- 第一阶段
  - 构造图
  - 展示图
- 第二阶段
  - 基于图的操作方法作计算
    - 例如：好友推荐、股票推荐、对给定的句子分词
    - 发挥想象力，寻找有趣的、有意义的、有价值的图计算应用
- 选题范围
  - 如作业1

Spark实践(5)——作业3

- MLlib
- 分类
  - 利用抓取的数据构建一个可以使用分类方法解决的需求
  - 整理训练集、测试集
  - 使用MLlib工具训练分类器
  - 展示效果
- 聚类
  - 同样，提出一个聚类能够解决的需求
  - 整理数据
  - 执行分类算法
  - 展示结果

谢 谢！