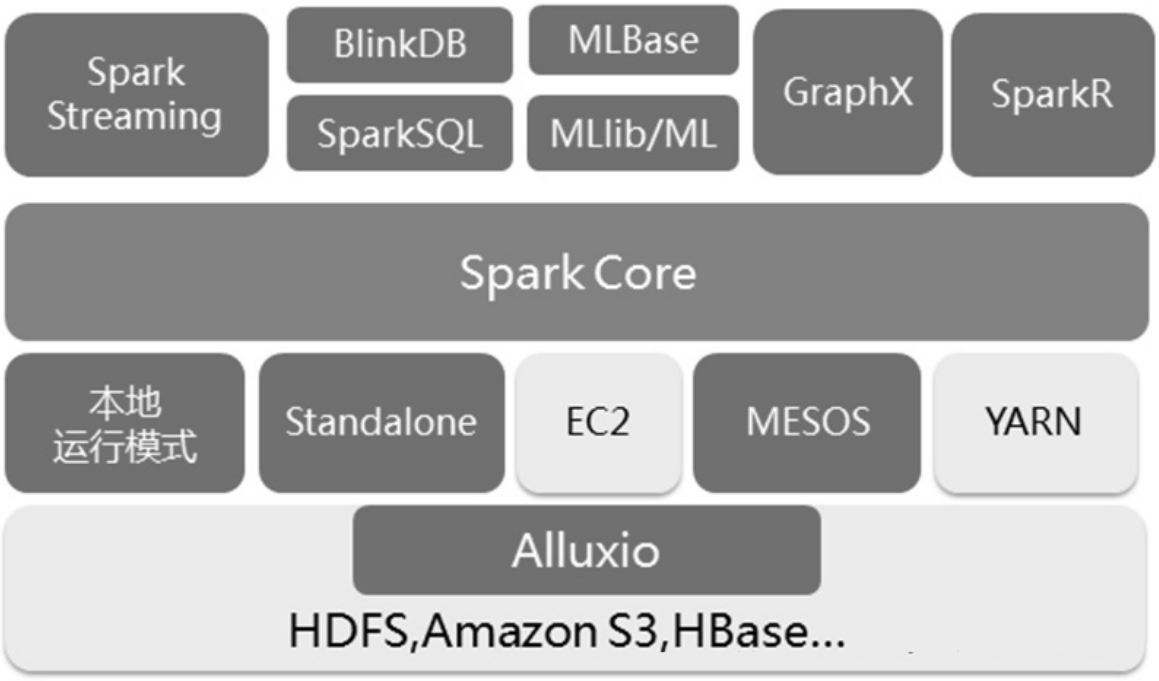
1. Spark核心功能



Spark Core提供Spark最基础的最核心的功能，主要包括：

1、SparkContext

通常而言，DriverApplication的执行与输出都是通过SparkContext来完成的，在正式提交Application之前，首先需要初始化SparkContext。

SparkContext内置的DAGScheduler负责创建Job，将DAG中的RDD划分到不同的Stage，提交Stage等功能。
MapReduce编程模型中，分成两个执行阶段：mapper阶段　reducer阶段　　阶段==stage
Spark的应用程序的编写方式：链式调用，　所以AGScheduler就是针对这个调用链，执行划分，标准就是shuffle

SparkContext内置的TaskScheduler负责资源的申请、任务的提交及请求集群对任务的调度等工作。
TaskScheduler的职责：把stage变成TaskSet

SparkContext隐藏了网络通信、分布式部署、消息通信、存储能力、计算能力、缓存、测量系统、文件服务、Web服务等内容，应用程序开发者只需要使用SparkContext提供的API完成功能开发。

2、存储体系

Spark优先考虑使用各节点的内存作为存储，当内存不足时才会考虑使用磁盘，这极大地减少了磁盘I/O，提升了任务执行的效率，使得Spark适用于实时计算、流式计算等场景。
总结：如果一个spark的应用程序中的各个stage的数据，全部都下放到磁盘。那么其实跟mapreduce区别不大了。

此外，Spark还提供了以内存为中心的高容错的分布式文件系统Tachyon（现在改名为Alluxio）供用户进行选择。Tachyon能够为Spark提供可靠的内存级的文件共享服务。

3、计算引擎

计算引擎由SparkContext中的DAGScheduler、RDD以及具体节点上的Executor负责执行的Map和Reduce任务组成。

DAGScheduler和RDD虽然位于SparkContext内部，但是在任务正式提交与执行之前将Job中的RDD组织成有向无环图（简称DAG）、并对Stage进行划分并决定各任务执行阶段任务的数量、迭代计算、shuffle等过程。

4、部署模式

由于单节点不足以提供足够的存储及计算能力，所以作为大数据处理的Spark在SparkContext的TaskScheduler组件中提供了对Standalone部署模式的实现和YARN、Mesos等分布式资源管理系统的支持。

通过使用Local、Standalone（Master，Worker）、YARN（ResourceManager，NodeManager）、Mesos、kubernetes、Cloud等部署模式为Task分配计算资源，提高任务的并发执行效率。除了可用于实际生产环境的Standalone、YARN、Mesos、kubernetes、Cloud等部署模式外，Spark还提供了Local模式和cluster模式便于开发和调试。

2. Spark应用模块

Spark的底层提供了一个 基于内存/磁盘 批处理计算引擎。spark基于这个引擎，提供了很多的高级应用模块。解决不同场景中的业务需求。

为了扩大应用范围，Spark陆续增加了一些扩展功能，主要包括：

1、Spark SQL

由于SQL具有普及率高、学习成本低等特点，为了扩大Spark的应用面，因此增加了对SQL及Hive的支持。

Spark SQL的过程可以总结为：首先使用SQL语句解析器（SqlParser）将SQL转换为语法树（Tree），并且使用规则执行器（RuleExecutor）将一系列规则（Rule）应用到语法树，最终生成物理执行计划并执行的过程。

其中，规则包括语法分析器（Analyzer）和优化器（Optimizer）。Hive的执行过程与SQL类似。

Hive相对于mapreduce就类似于 sparksql相对于 sparkcore，处理结构化数据

2、Spark Streaming

Spark Streaming与Apache Storm类似，也用于流式计算。

Spark Streaming支持Kafka、Flume、Twitter、MQTT、ZeroMQ、Kinesis和简单的TCP套接字等多种数据输入源。输入流接收器（Receiver）负责接入数据，是接入数据流的接口规范。

Dstream是Spark Streaming中所有数据流的抽象，Dstream可以被组织为DStreamGraph。Dstream本质上由一系列连续的RDD组成。

3、Spark GraphX

Spark提供的分布式图计算框架。GraphX主要遵循整体同步并行计算模式（Bulk Synchronous Parallell，简称BSP）下的Pregel模型实现。

GraphX提供了对图的抽象Graph，Graph由顶点（Vertex）、边（Edge）及继承了Edge的EdgeTriplet（添加了srcAttr和dstAttr用来保存源顶点和目的顶点的属性）三种结构组成。

GraphX目前已经封装了最短路径、网页排名、连接组件、三角关系统计等算法的实现，用户可以选择使用。

4、Spark MLlib

Spark提供的机器学习框架。机器学习是一门涉及概率论、统计学、逼近论、凸分析、算法复杂度理论等多领域的交叉学科。

MLlib目前已经提供了基础统计、分类、回归、决策树、随机森林、朴素贝叶斯、保序回归、协同过滤、聚类、维数缩减、特征提取与转型、频繁模式挖掘、预言模型标记语言、管道等多种数理统计、概率论、数据挖掘方面的数学算法。

Flink的 mllib也正在大力的完善。 Spark和Flink会趋同

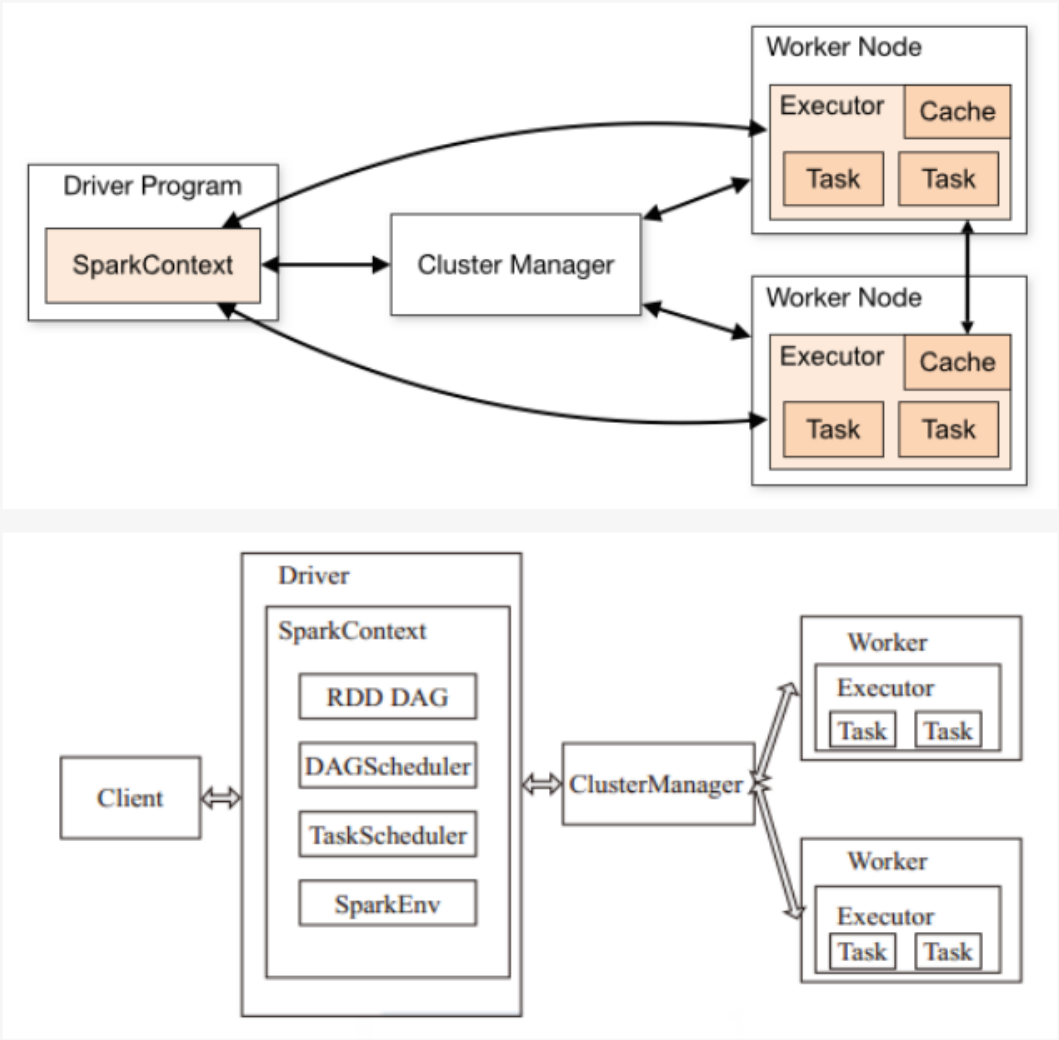
3. Spark基本架构

spark的架构分成两个方面：

```
集群的架构： master  worker  主从架构  JVM进程
应用程序架构： driver  executor 组成   JVM进程
```

从应用程序运行的角度来看，Spark集群由以下部分组成：

一个应用程序由 一个driver 和 n多个executor组成
在standalone集群中，其实master就是 cluster manager
从节点worker中会运行多个executor ， 这个任务进程中，会初始化运行一个线程池，线程池中的每个线程会运行一个Task



Cluster Manager

Spark的集群管理器，主要负责资源的分配与管理。集群管理器分配的资源属于一级分配，它将各个Worker上的内存、CPU等资源分配给应用程序，但是并不负责对Executor的资源分配。目前，Standalone、YARN、Mesos、K8S，EC2等都可以作为Spark的集群管理器。

Master

Spark集群的主节点。负责管理整个集群。

Worker:

Spark集群的工作节点。对Spark应用程序来说，由集群管理器分配得到资源的Worker节点主要负责以下工作：创建Executor，将资源和任务进一步分配给Executor，同步资源信息给Cluster Manager。

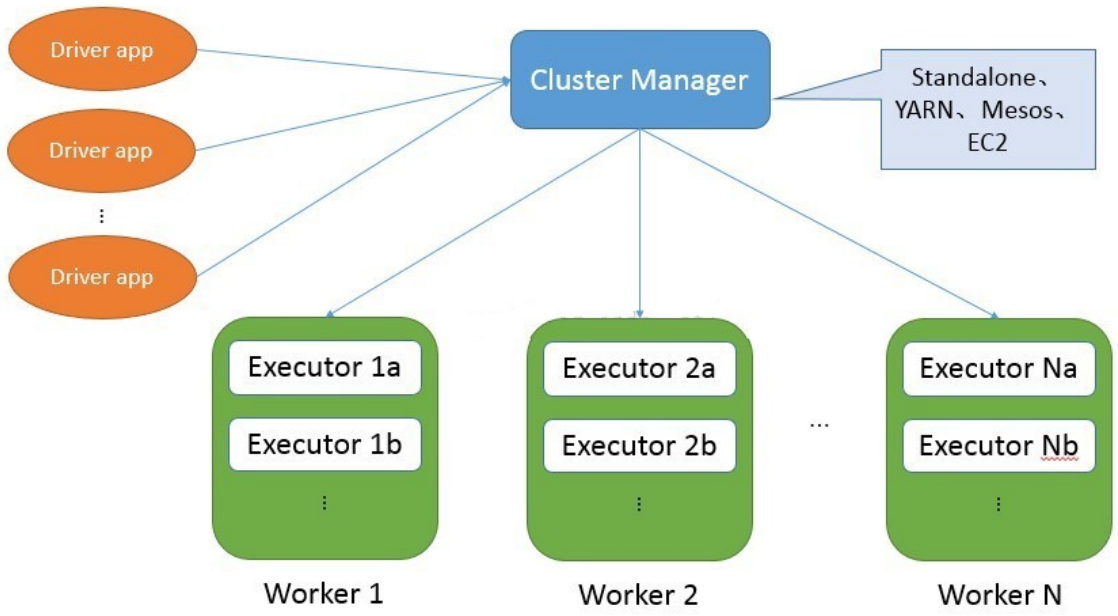
Executor:

执行计算任务的一些进程。主要负责任务的执行以及与Worker、Driver Application的信息同步。

Driver Appication

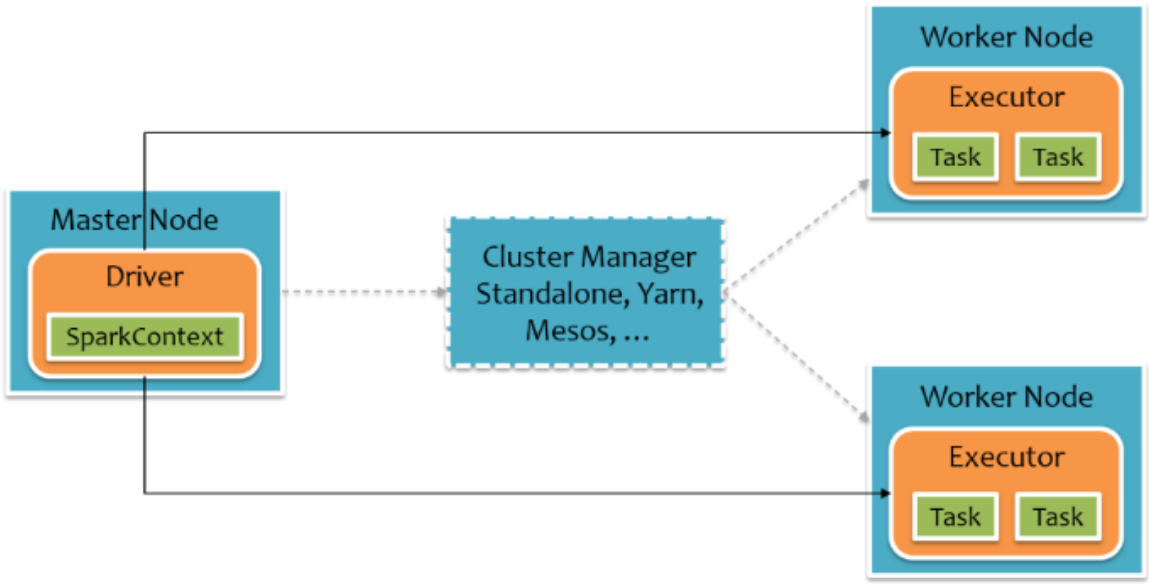
> 客户端驱动程序，也可以理解为客户端应用程序，用于将任务程序转换为RDD和DAG，并与Cluster Manager进行通信与调度。

这些组成部分之间的整体关系如下图所示：



Spark计算平台有两个重要角色，Driver和executor，不论是StandAlone模式还是YARN模式，都是：

> Driver充当Application的master角色，负责任务执行计划生成和任务分发及调度；
> executor充当Application的worker角色，负责实际执行任务的task，计算的结果返回Driver。



4. Spark核心概念

Term	Meaning
Application	User program built on Spark. Consists of a *driver program* and *executors* on the cluster.
Application jar	A jar containing the user's Spark application. In some cases users will want to create an "uber jar" containing their application along with its dependencies. The user's jar should never include Hadoop or Spark libraries, however, these will be added at runtime.
Driver program	The process running the main() function of the application and creating the SparkContext
Cluster manager	An external service for acquiring resources on the cluster (e.g. standalone manager, Mesos, YARN)
Deploy mode	Distinguishes where the driver process runs. In "cluster" mode, the framework launches the driver inside of the cluster. In "client" mode, the submitter launches the driver outside of the cluster.
Worker node	Any node that can run application code in the cluster
Executor	A process launched for an application on a worker node, that runs tasks and keeps data in memory or disk storage across them. Each application has its own executors.
Task	A unit of work that will be sent to one executor
Job	A parallel computation consisting of multiple tasks that gets spawned in response to a Spark action (e.g. `save`, `collect`); you'll see this term used in the driver's logs.
Stage	Each job gets divided into smaller sets of tasks called *stages* that depend on each other (similar to the map and reduce stages in MapReduce); you'll see this term used in the driver's logs.

大多数应该都要有实际写过Spark程序和提交任务到Spark集群后才有更好的理解

1、Application：表示你的应用程序，包含一个Driver Program和若干Executor

2、Driver Program：Spark中的Driver即运行上述Application的main()函数并且创建SparkContext，其中创建SparkContext的目的是为了准备Spark应用程序的运行环境。由SparkContext负责与ClusterManager通信，进行资源的申请，任务的分配和监控等。程序执行完毕后关闭SparkContext

3、ClusterManager：在Standalone模式中即为Master（主节点），控制整个集群，监控Worker。在YARN模式中为资源管理器。

4、SparkContext：整个应用的上下文，控制应用程序的生命周期，负责调度各个运算资源，协调各个Worker上的Executor。初始化的时候，会初始化DAGScheduler和TaskScheduler两个核心组件。

5、RDD：Spark的基本计算单元，一组RDD可形成执行的有向无环图RDD Graph。

6、DAGScheduler：根据Job构建基于Stage的DAG，并提交Stage给TaskScheduler，其划分Stage的依据是RDD之间的依赖关系：宽依赖，也叫shuffle依赖

7、TaskScheduler：将TaskSet提交给Worker（集群）运行，每个Executor运行什么Task就是在此处分配的。

8、Worker：集群中可以运行Application代码的节点。在Standalone模式中指的是通过slave文件配置的worker节点，在Spark on Yarn模式中指的就是NodeManager节点。

9、Executor：某个Application运行在Worker节点上的一个进程，该进程负责运行某些task，并且负责将数据存在内存或者磁盘上。在Spark on Yarn模式下，其进程名称为CoarseGrainedExecutorBackend，一个CoarseGrainedExecutorBackend进程有且仅有一个executor对象，它负责将Task包装成taskRunner，并从线程池中抽取出一个空闲线程运行Task，这样，每个CoarseGrainedExecutorBackend能并行运行Task的数据就取决于分配给它的CPU的个数。

10、Stage：每个Job会被拆分很多组Task，每组作为一个TaskSet，其名称为Stage

11、Job：包含多个Task组成的并行计算，是由Action行为触发的

12、Task：在Executor进程中执行任务的工作单元，多个Task组成一个Stage

13、SparkEnv：线程级别的上下文，存储运行时的重要组件的引用。SparkEnv内创建并包含如下一些重要组件的引用。

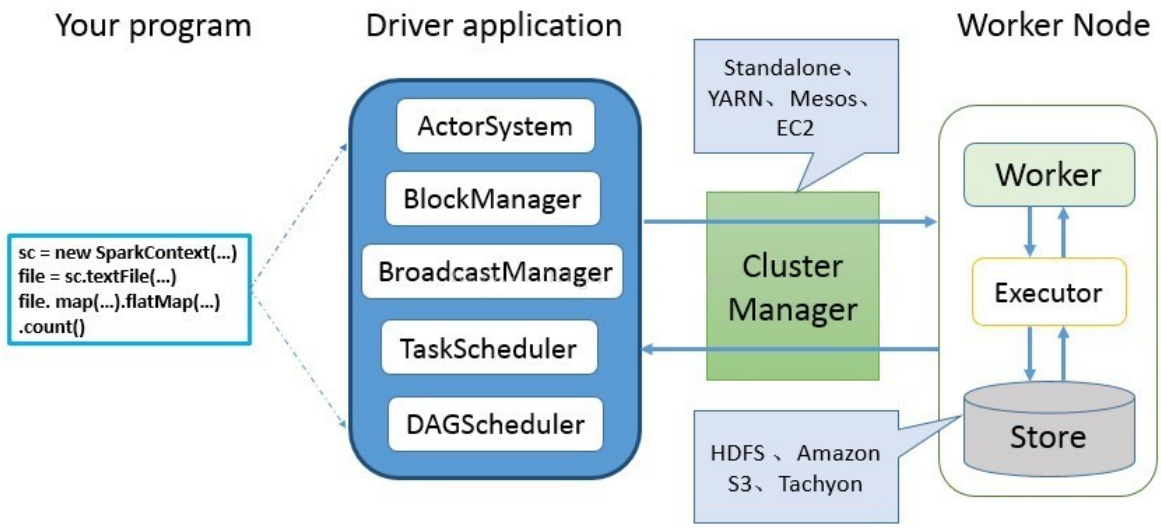
MapOutPutTracker：负责Shuffle元信息的存储。
BroadcastManager：负责广播变量的控制与元信息的存储。
BlockManager：负责存储管理、创建和查找块。
MetricsSystem：监控运行时性能指标信息。
SparkConf：负责存储配置信息。

5. Spark编程模型

整体来说，就是五个步骤：

1、初始化创建一个SparkContext编程入口
2、通过编程入口对象SparkContext加载数据源得到一个RDD
3、针对这个RDD链式调用各种算子执行各种逻辑计算得到一个结果RDD
4、针对这个结果rdd执行落盘操作
5、提交job运行

Spark应用程序从编写到提交、执行、输出的整个过程如下图所示：



图中描述的步骤如下：

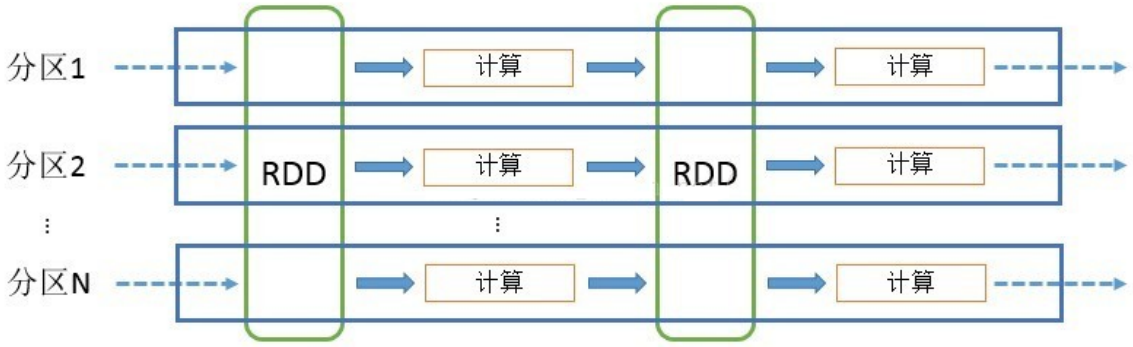
1、用户使用SparkContext提供的API（常用的有textFile、sequenceFile、runJob、stop等）编写Driver Application程序。此外SQLContext、HiveContext及StreamingContext对SparkContext进行封装，并提供了SQL、Hive及流式计算相关的API。

2、使用SparkContext提交的用户应用程序，首先会使用BlockManager和BroadcastManager将任务的Hadoop配置进行广播。然后由DAGScheduler将任务转换为RDD并组织成DAG，DAG还将被划分为不同的Stage。最后由TaskScheduler借助ActorSystem将任务提交给集群管理器（ClusterManager）。

3、集群管理器（ClusterManager）给任务分配资源，即将具体任务分配到Worker上，Worker创建Executor来处理任务的运行。Standalone、YARN、Mesos、kubernetes、EC2等都可以作为Spark的集群管理器。

计算模型：

RDD可以看做是对各种数据计算模型的统一抽象，Spark的计算过程主要是RDD的迭代计算过程，如上图。RDD的迭代计算过程非常类似于管道。分区数量取决于partition数量的设定，每个分区的数据只会在一个Task中计算。所有分区可以在多个机器节点的Executor上并行执行。

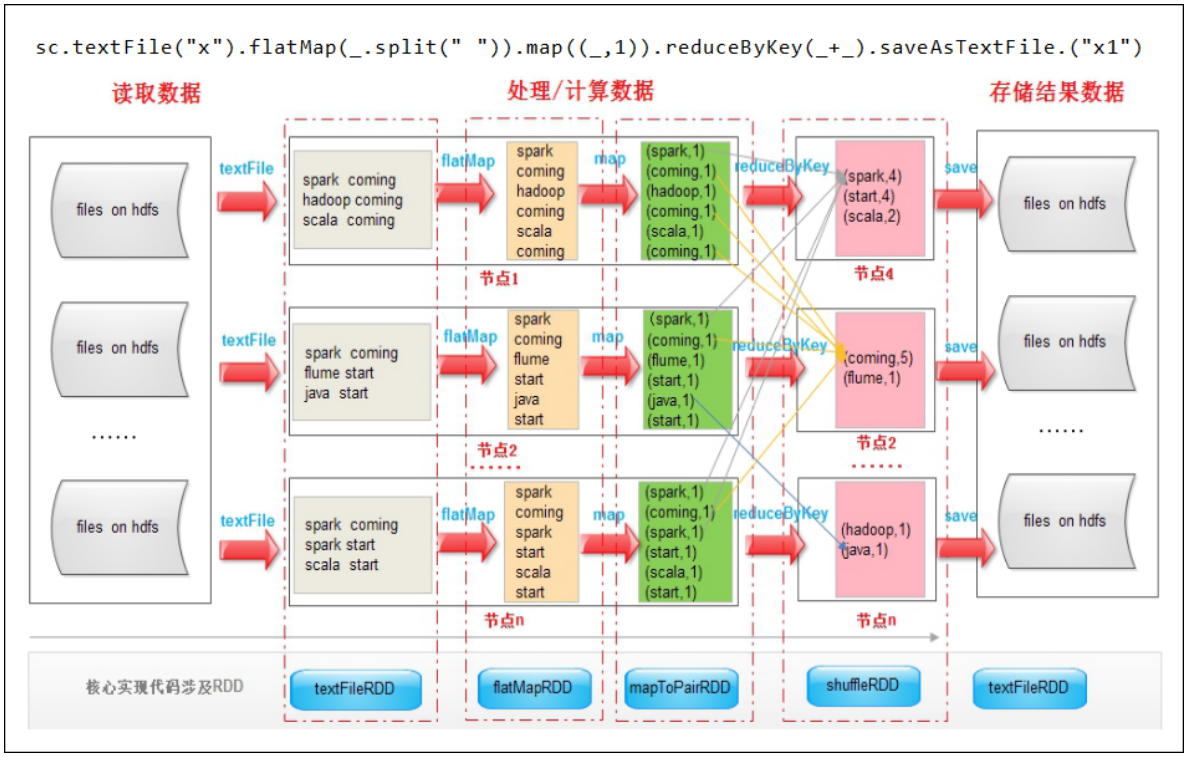


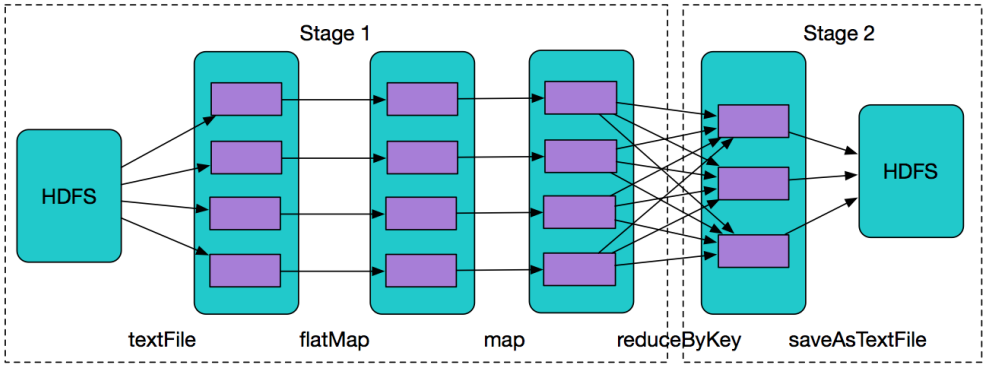
6. Spark的WordCount

代码：

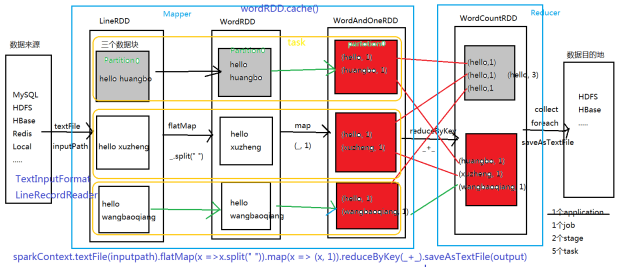
```
sc.textFile("x").flatMap(_.split(" ")).map((_,
1)).reduceByKey(_+_).saveAsTextFile("x1")
```

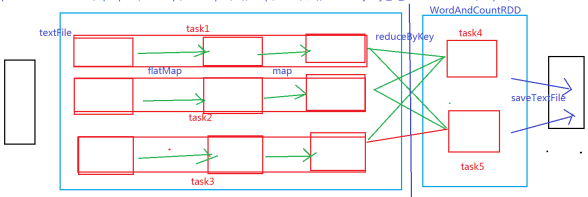
图解：





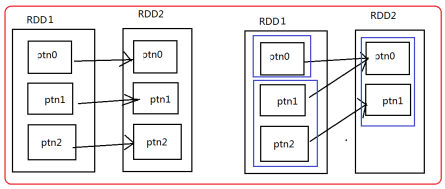

```
val textFile = sc.textFile(args(1))          // 构造RDD
val result = textFile .flatMap(line => line.split("\\s+")) .map(word => (word, 1)) .reduceByKey(_ + _) // 计算逻辑
result.saveAsTextFile(args(2))               // 数据存储
```





RDD的算子：
transformation 转换
action 动作

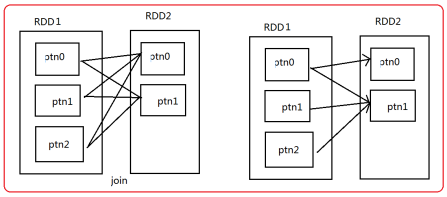
reduceByKey
切分stage的标准：从后往前寻找shuffle算子



窄依赖

RDD1：父RDD
RDD2：子RDD

独生：一个父分区的所有数据，都只会交给子RDD中的一个子分区



宽依赖

超生：
一个父分区把数据分散了之后shuffle到子RDD中的多个分区

窄依赖： transformation， 没有shuffle
宽依赖： 通俗的说：shuffle依赖， 这是影响应用程序运行效率的依赖

一个job切分成多个stage的标准：

shuffle类的算子， RDD -------> shuffle -------> RDD2

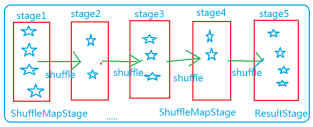
shuffle依赖：宽依赖

切分stage的标准： shuffle依赖

三个面试题中的第二个：
求交集
分别hash散列

ShuffleMapStage
ResultStage

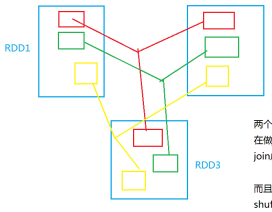
ResultStage
ShuffleStage



一般来说，一个appcation基本上，都只有一个job
这个任务，最后可能只有一个action的算子，

一个application ， 一个job

每个stage其实可以包



join:

两个父RDD的分区有key的对应关系，
在做join时候，只需要对应分区的数据，
join成一个结果分区即可

而且最后所有的数据，没有必要再进行shuffle