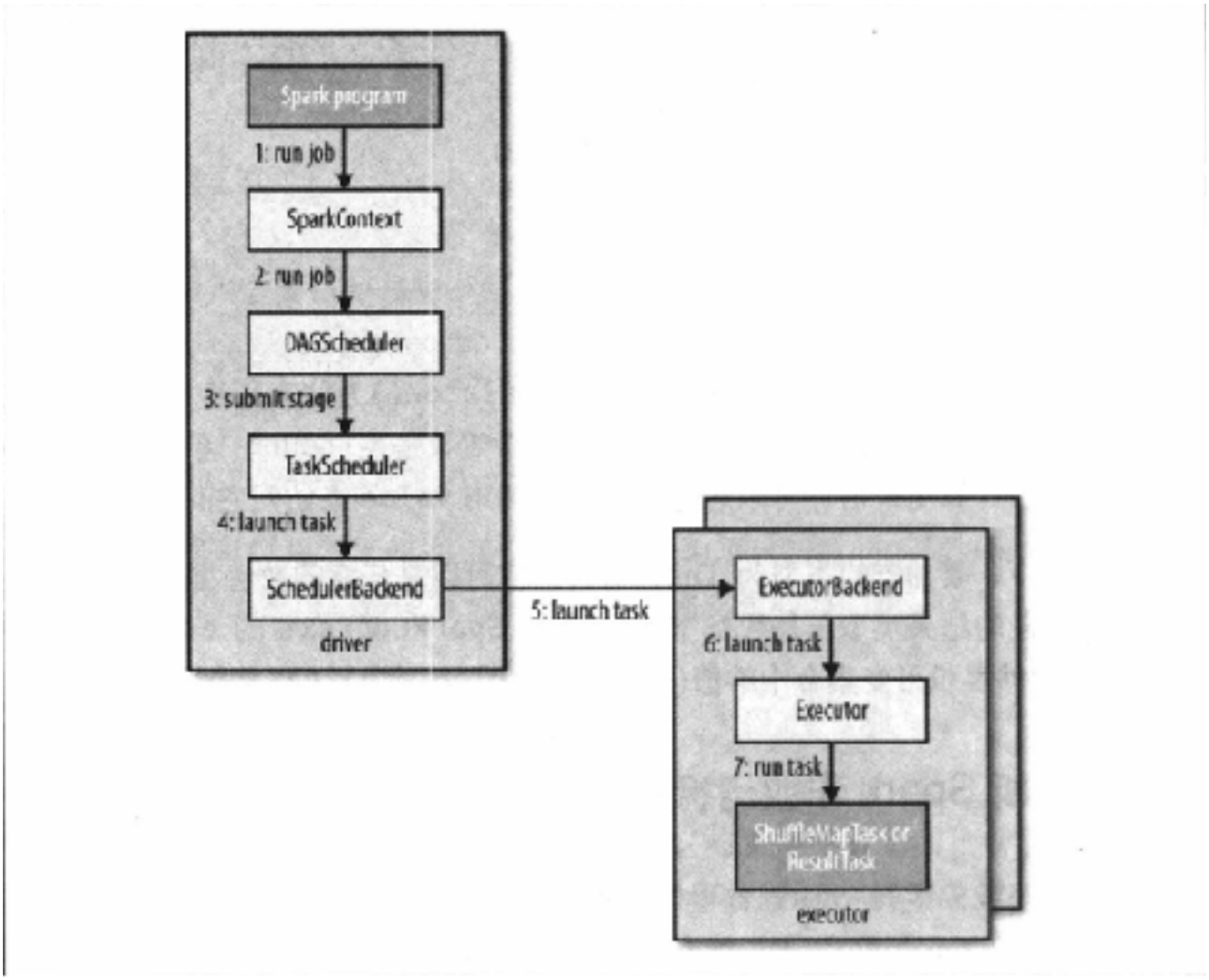
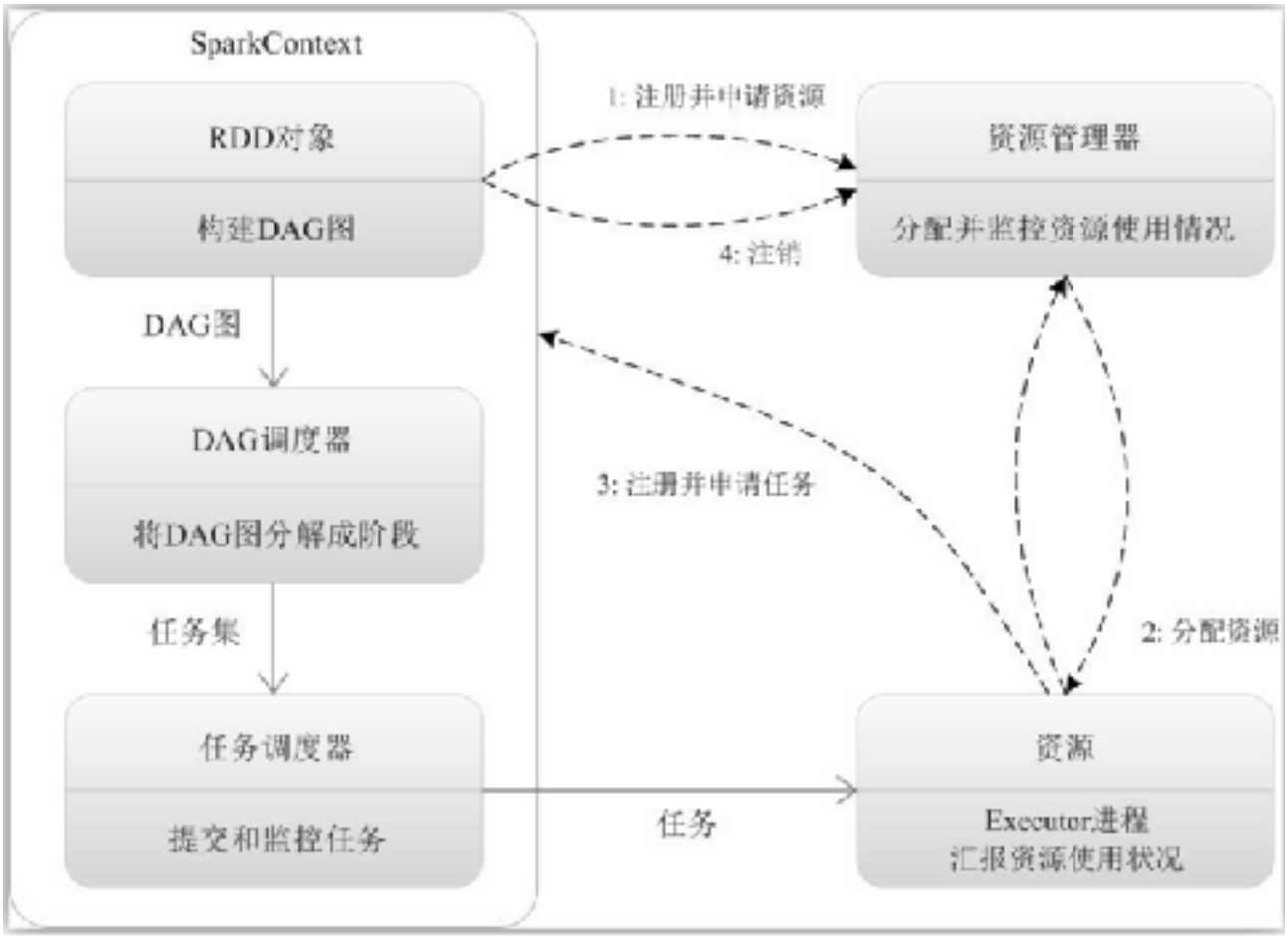
Spark应用，作业，阶段和任务：
应用>作业>有向无环图>阶段>任务->并行运行在RDD上

Spark作业是由任意的多阶段的有向无环图(DAG)构成的，其中的每个阶段大致相当于MapReduce中的map阶段或reduce阶段
这些阶段又被Spark运行环境分解为多个任务(task)，任务并行运行在分布于集群中的RDD分区上
Spark作业始终运行在应用(application)上下文中(SparkContext实例来表示)，它提供了RDD分区以及共享变量。一个应用可以串行或并行地运行多个作业，并为这些作业提供访问由同一应用的先前作业所缓存的RDD(cache)的机制

1.作业提交
当对RDD执行一个动作，会自动提交一个Spark作业。从内部看，它导致对SparkContext调用runJob()，然后将调用传递给作为driver的一部分运行的调度程序。调度程序由两部分组成：DAG调度程序和任务调度程序。
DAG调度程序把作业分解为若干阶段，并由这些阶段构成一个DAG。任务调度阶段则负责把每个阶段中的人物提交到集群。





总体而言，Spark运行架构具有以下特点：

（1）每个应用都有自己专属的Executor进程，并且该进程在应用运行期间一直驻留。Executor进程以多线程的方式运行任务，减少了多进程任务频繁的启动开销，使得任务执行变得非常高效和可靠；

（2）Spark运行过程与资源管理器无关，只要能够获取Executor进程并保持通信即可；

（3）Executor上有一个BlockManager存储模块，类似于键值存储系统（把内存和磁盘共同作为存储设备），在处理迭代计算任务时，不需要把中间结果写入到HDFS等文件系统，而是直接放在这个存储系统上，后续有需要时就可以直接读取；在交互式查询场景下，也可以把表提前缓存到这个存储系统上，提高读写IO性能；

（4）任务采用了数据本地性和推测执行等优化机制。数据本地性是尽量将计算移到数据所在的节点上进行，即"计算向数据靠拢"，因为移动计算比移动数据所占的网络资源要少得多。而且，Spark采用了延时调度机制，可以在更大的程度上实现执行过程优化。比如，拥有数据的节点当前正被其他的任务占用，那么，在这种情况下是否需要将数据移动到其他的空闲节点呢？答案是不一定。因为，如果经过预测发现当前节点结束当前任务的时间要比移动数据的时间还要少，那么，调度就会等待，直到当前节点可用。

2.DAG(有向无环图)的构建：

在阶段中运行的任务类型：shuffle map任务和result任务

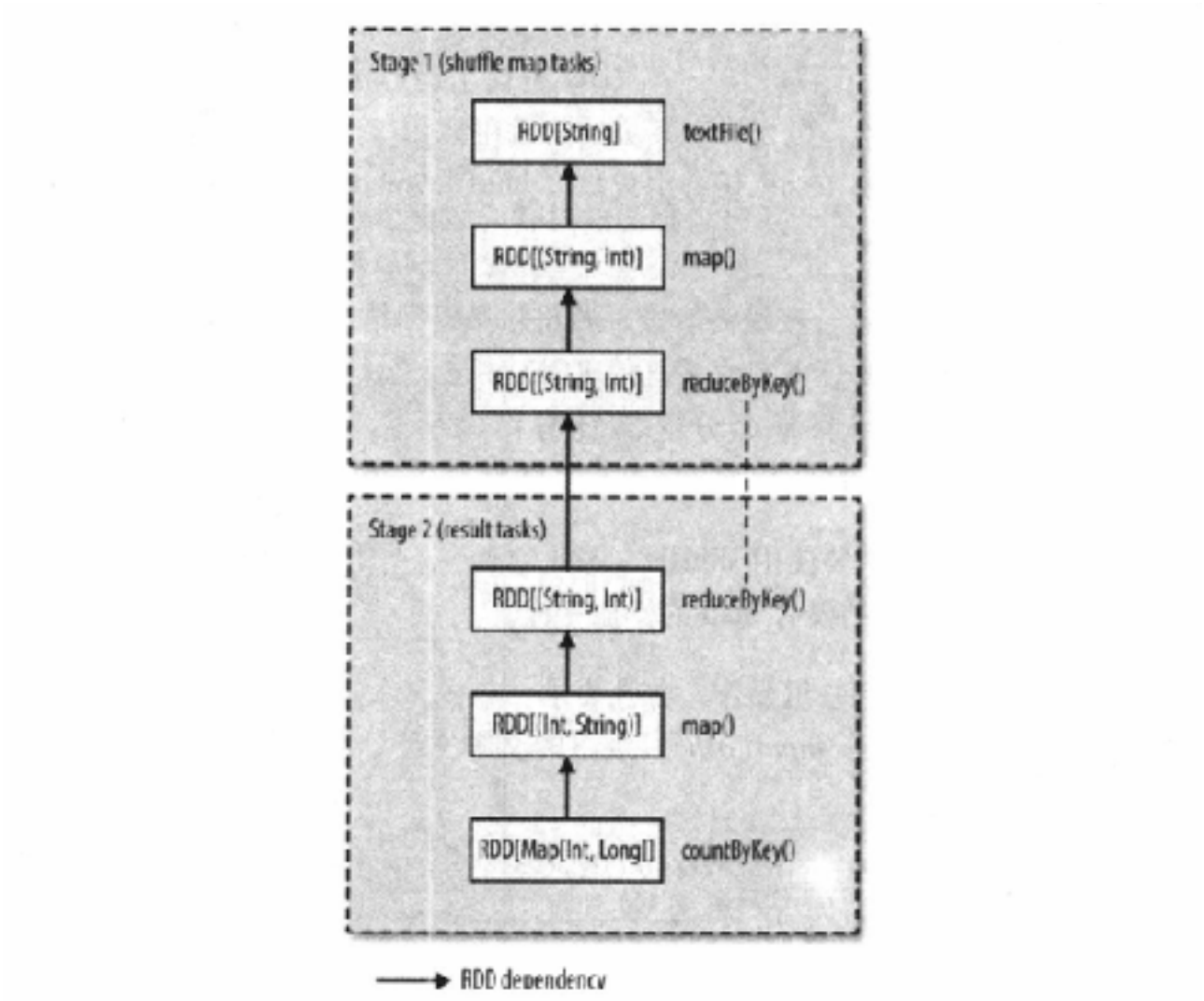
shuffle map任务就像是MapReduce中shuffle的map端部分。每个shuffle map任务在一个RDD分区上进行计算，并根据分区函数把输出写入一组新的分区中，以允许在后面的阶段中取用

result任务运行在最终阶段，并将结果返回给用户程序。每个result任务在它自己的RDD分区上运行计算，然后把结果发送回driver，再由driver将每个分区的计算结果汇集成最终结果。

3.比较复杂的作业要涉及到分组操作，并且要求一个或多个shuffle阶段。例如，该作业用于为存储在inputPath目录下的文本文件计算词频统计分布图(每行文本只有一个单词)

```
val hist: Map[Int, Long] = sc.textFile(inputPath)
.map(word => (word.toLowerCase(), 1))
.reduceByKey((a, b) => a + b)
.map(_.swap)
.countByKey()
```

前两个转换map()和recudeByKey()，用于计算每个单词出现的频率。
第三个转换是map()，它交换每个键值对中的键和值，从而得到(count, word)对。
最后是countByKey()动作，它返回的是每个计数对应的单词量(即词频分布)
通常，每个阶段的RDD都要在DAG中显示



注意，reduceByKey()跨越了两个阶段，这是因为它是使用shuffle实现的，并且就像MapReduce一样，reduce函数一边在map作为combiner运行阶段，一边又在reduce端又作为reduce运行。它与MapReduce相似的另一个地方是:Spark的shuffle实现将其输

出写入本地磁盘上的分区文件中(即使对内存中的RDD也一样)并且这些文件将由下一阶段的RDD读取