1. 安装spark，下载：<http://spark.apache.org/downloads.html>
   1. 加环境变量：SPARK\_HOME，并把SPARK\_HOME\bin加入到PATH环境变量
   2. 下载：<https://github.com/SweetInk/hadoop-common-bin.git>，加环境变量：HADOOP\_HOME（如：D:\big\_data\hadoop-common-bin\2.7.3），并把HADOOP \_HOME\bin加入到PATH环境变量
   3. 现在就可以运行pyspark了，本目录下有一个用python写spark应用的代码模板：spark\_tpl.py
2. Spark中的parquet文件，为列式 存储格式，他包含表的schema信息，DataFrame可以直接保存为parquet文件，parquet文件也可以直接读取为DataFrame：
   1. peopleDF = spark.read.json("examples/src/main/resources/people.json")

peopleDF.write.parquet("people.parquet")

* 1. parquetFile = spark.read.parquet("people.parquet")

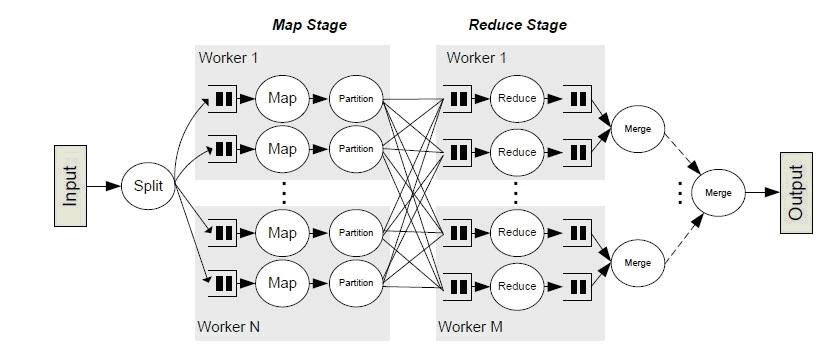
parquetFile.createOrReplaceTempView("parquetFile")

teenagers = spark.sql("SELECT name FROM parquetFile WHERE age >= 13 AND age <= 19")

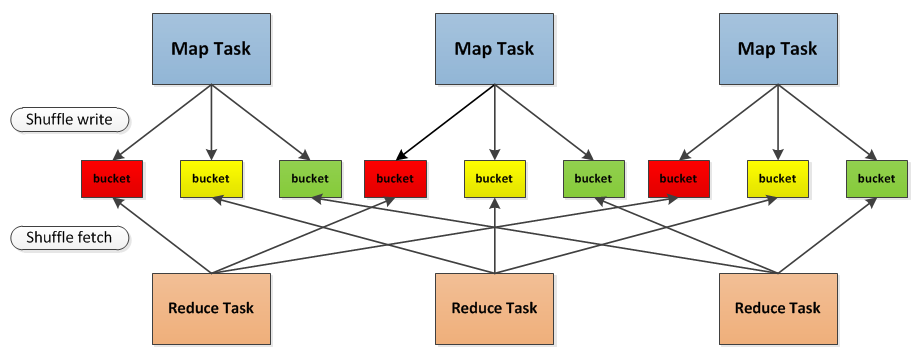
列式存储原理比较复杂，格式也比较复杂，目的是为了更快的定位并读取到所需要的数据，参考：<https://blog.csdn.net/yu616568/article/details/50993491>，理解它有助于写出更高性能的代码，如在使用Parquet的时候可以通过如下两种策略提升查询性能：1、类似于关系数据库的主键，对需要频繁过滤的列设置为有序的，这样在导入数据的时候会根据该列的顺序存储数据，这样可以最大化的利用最大值、最小值实现谓词下推。2、减小行组大小和页大小，这样增加跳过整个行组的可能性，但是此时需要权衡由于压缩和编码效率下降带来的I/O负载。

1. spark-shuffle（洗牌）解释：

下面这幅图清晰地描述了MapReduce算法的整个流程，其中shuffle phase是介于Map phase和Reduce phase之间：



Spark中shuffle的整一个流程：



首先每一个Mapper会根据Reducer的数量创建出相应的bucket，bucket的数量是M×R，其中MM是Map的个数，RR是Reduce的个数。

其次Mapper产生的结果会根据设置的partition算法填充到每个bucket中去。这里的partition算法是可以自定义的，当然默认的算法是根据key哈希到不同的bucket中去。

当Reducer启动时，它会根据自己task的id和所依赖的Mapper的id从远端或是本地的block manager中取得相应的bucket作为Reducer的输入进行处理。

这里的bucket是一个抽象概念，在实现中每个bucket可以对应一个文件，可以对应文件的一部分或是其他等。Apache Spark 的 Shuffle 过程与 Apache Hadoop 的 Shuffle 过程有着诸多类似，一些概念可直接套用，例如，Shuffle 过程中，提供数据的一端，被称作 Map 端，Map 端每个生成数据的任务称为 Mapper，对应的，接收数据的一端，被称作 Reduce 端，Reduce 端每个拉取数据的任务称为 Reducer，Shuffle 过程本质上都是将 Map 端获得的数据使用分区器进行划分，并将数据发送给对应的 Reducer 的过程。

1. parquet文件支持逐渐增加列，这样会得到多个parquet文件，这些文件的schema不相同但是是兼容的，使用代码可以控制进行合并：

sc = spark.sparkContext

squaresDF = spark.createDataFrame(sc.parallelize(range(1, 6))

.map(lambda i: Row(single=i, double=i \*\* 2)))

squaresDF.write.parquet("data/test\_table/key=1")

cubesDF = spark.createDataFrame(sc.parallelize(range(6, 11))

.map(lambda i: Row(single=i, triple=i \*\* 3)))

cubesDF.write.parquet("data/test\_table/key=2")

mergedDF=spark.read.option("mergeSchema","true").parquet("data/test\_table")

mergedDF.printSchema()