初识spark

一个人的心算能力是有限的，所以在很多年前，人们就开始了在各种东西上进行演算。最开始，在沙摊上，后来开始在布上，再后来开始在纸上，再后来有了计算器，再往后有了计算机。

在计算机上进行演算的确是很快，而且随着摩尔定律的一步步被验证（以后还这样下去可能有点够呛了），计算机处理效率越来越高。

虽然计算机是进步了，可是我们人类产生数据的速度简直是超乎想象的快，现在很多企业数据在单台计算机上已经难以处理（当然并不是说单台计算机处理数据的价值就比较低，需要分场景）。

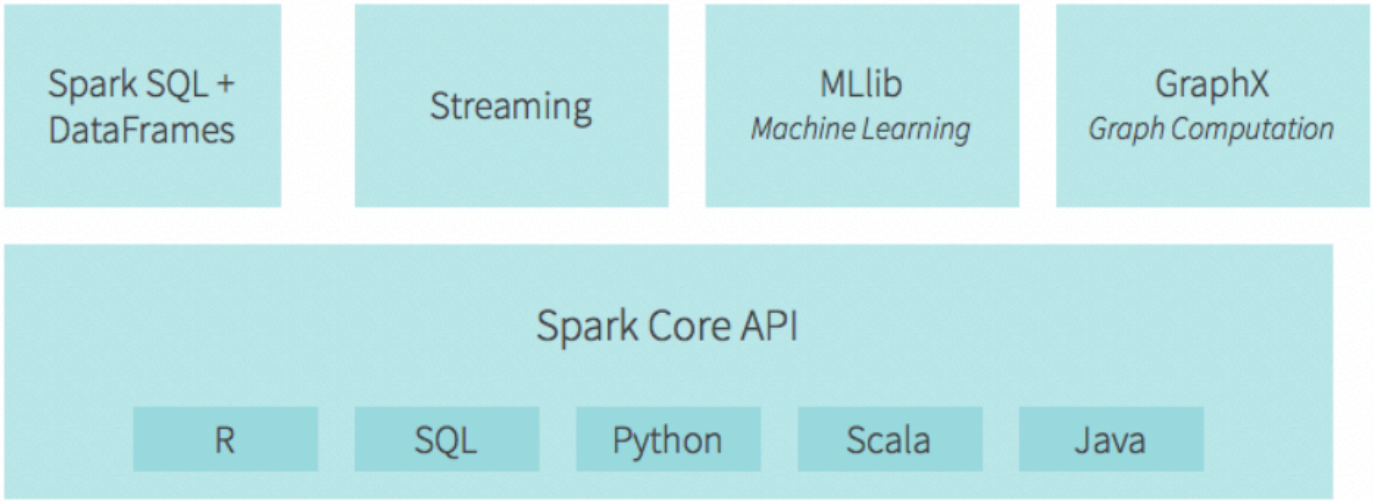
为了可以处理更多的数据，人们把很多机器链接起来，形成了分布式集群，在这些集群上存储数据是可以，那么如何高效的处理这些数据，于是Apache Apark出现了，它是一个开源、强大的分布式查询和处理引擎，再此之前，有MapReduce也可以做分布式数据处理，但是spark更强。

spark酷毙的地方是除了它的快（内存中比Hadoop快100倍，读磁盘也快10倍），还有它惊人的灵活性。

在数据处理的时候，它支持被每个数据分析师烂熟于心的SQL，支持看惯关系型数据库的DataFrame，支持关联挖掘的GraphFrames，支持机器学习的ML，甚至在深度学习上都有TensorFrames。

在使用语言上，它基于Java，原生代码是Scala，但是也支持Python，SQL，R，可以说对程序员的照顾是发自内心的。除了这些，spark还支持流式计算Streaming，也就是说不用等到数据全部完成落到分布式集群上了再开始计算，在线实时采集一会儿就可以。

简单看一下它的结构



多说一句，在分布式集群上执行数据处理，那必然会有主节点和多个工作节点，而spark通过有向无环（DAG）来组织这些依赖关系。

既然是处理数据，那么spark的数据是什么样的呢？答案是RDD。RDD中文叫弹性分布式数据集，是不可变Java虚拟机（JVM）对象的分布式集合，数据就是存储在这些JVM中的，这里的弹性，也说明了spark依赖数据的灵活性。

要做到快，就尽可能的把串行搞成并行，而RDD也正是这么做的。RDD包括两两组并行操作：

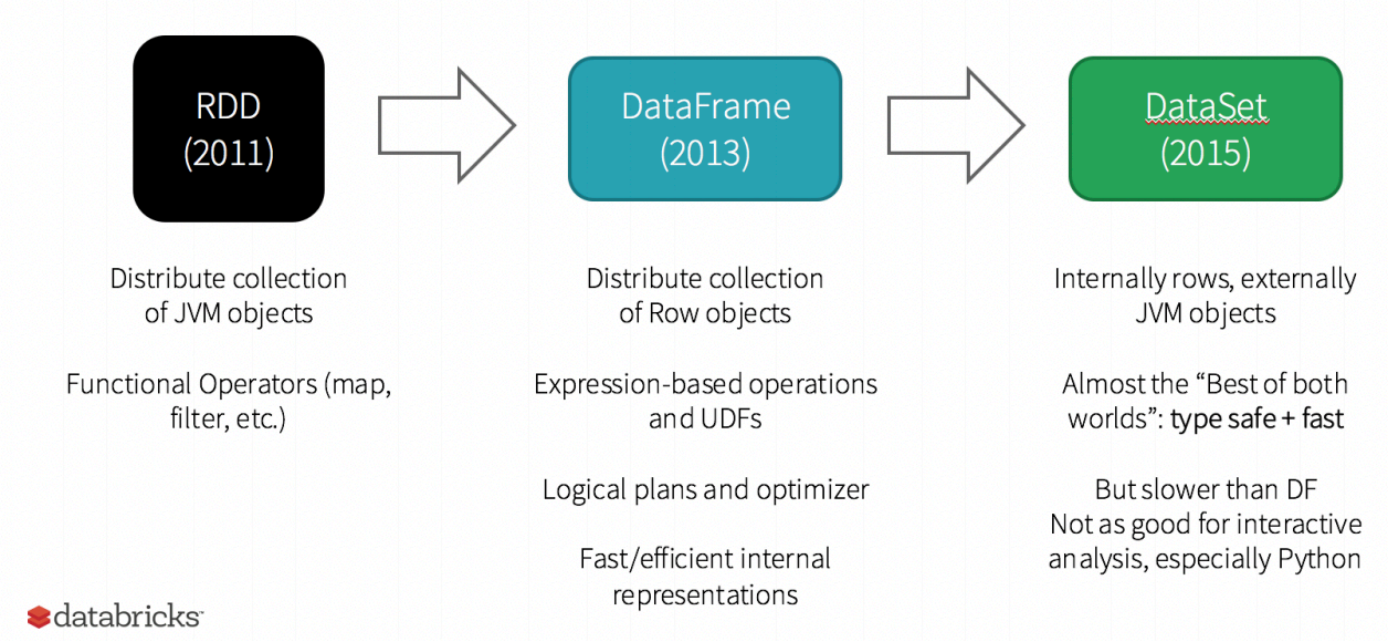
- 转换，反馈指向新的RDD指针
- 动作，运行计算

而且，spark还很“懒”，在没有收到具体的动作指令前，它不会做任何的转换，只有动作执行了而且需要有结果反馈时，才会计算，这种延迟计算可以让它有针对性的查询优化。也算“懒”得很有理由（对于熟悉深度学习的朋友，这个tensorflow的计算图差不多）。

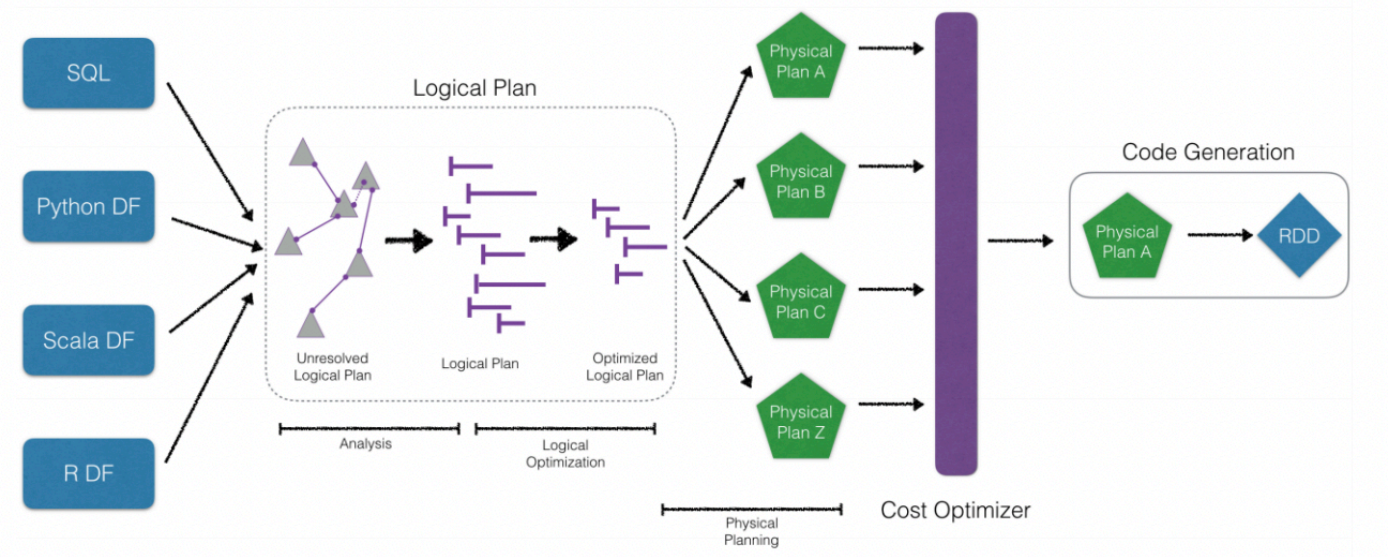
为了对更多的程序员友好，光有RDD当然不够，毕竟RDD也不能支持那么多语言，所以RDD基础上衍生出了对Python、Pandas、R更友好的DataFrame，这就和关系型数据库非常类似了，这也是spark被很多数据工程师广泛应用的重要原因。

除了提高支持更多语言灵活性，在速度上spark也追求更快，所有有了Dataset，不过有点遗憾，Dataset由于设计的缘故，特别不适合做交互式分析，特别是Python，所以现在也不支持Python。

他们之间在spark 2.0上关系为



RDD是spark原生的数据结构，快是应该的，可是既然说了灵活，也不能因为转换到SQL，Python上就慢了，不妨看看慢的原因，主要就是python这类新的API与RDD之间的通信拖慢了计算速度，所以spark有了Catalyst优化器，他将计算过程进行逻辑优化，极大的提高了计算速度，优化过程大概长这样。



spark的核心——RDD

前面说了spark的快主要是因为RDD的并行操作，那这种并行操作是个什么概念了，我们来举个例子。

1. 统计出某一列中不同值出现的次数
2. 选出以字母A开开头的
3. 输出结果

spark在计算的时候会先算2，再算1，最后算3，而不是1，2，3依次算，如果一个数据有1000万行，而A的只有10万行，这种计算速度的提升不言而喻了。

知道了并行的好处，现在开始创建RDD。创建方式有两种，一种的自己写，一种是读外部的。

- 自己写通过.parallelize()，这里面的数据结构非常丰富，像元组、字典、列表啥的都可以
- 读外部的通过.textFile()，指定读取的文件地址就可以，还可以顺便定个分区

前面我们说过，spark并行的两个东西分别是转换和动作，建好数据，现在就可以转换了。

RDD本身是一行一行的数据，所以自然首当其冲会有转换行的需求，可以使用.map()。

这种转换以后还是一行一行的，可是如果想转换到同一行呢，那就弄成水平的，用.flatMap()。

行转换完了，有时候我们也不需要全部的元素，选择条件就出现了，用.filter()，这就和SQL里面的where非常像了。

同样，去重也和SQL中的一样，用.distinct()，关联的时候用.join()，.leftOuterJoin()。当比join更严格，只要完全相同的时候，那就用.intersection()。

对于有明确条件的可以用filter，只想随机选择的时候就用.sample()。

spark是基于分布式集群的，那必定会涉及到分区，前面创建RDD的时候也会指定分区，那如果要改变分区了，可以用.repartition()。

说完转换，下面说动作。

对大量数据而言，我们最常用的就是看下数据大概长什么样。

比如反馈前面几行啥的，可以用.take()。

如果实在要看全部数据，那就用.collect()，不过这最好慎用。

当然，查看数据一般不能看完，如果想数一数有多少行，可以用.count()。

在转换的时候对行转换有map，那使用指定的方法呢，用.reduce()，这个是不是感觉用hadoop的mapreduce有点说不出的感觉了。

reduce是用一个函数，如果用相同的方法迭代的处理每个元素呢，那就用.foreach()。

最后，我们读了数据，最后也要记得把处理过后的数据保存下来，可以用.saveAsTextFile()。

pyspark的核心——DataFrame

如前面所说，DataFrame和Catalyst优化器的意义在于非优化的RDD查询时提升PySpark查询的性能，这里提升的性能主要是Python与JVM之间的通信开销。

要分析DataFrame数据，首先创建DataFrame数据，创建方法和RDD相同，不过需要经过一次到DataFrame的转化，使用spark.raed.json()。前面有说到spark是惰性的，所以只read了还不行，实际使用的时候可以转换为临时表，使用.createOrReplaceTempView()。

从RDD转换到DataFrame，共计有两种方式。

- 第一种是直接把json格式的数据给Dataframe，让spark自动推断是什么类型，这也叫反射推断模式。
- 另一种是定义StructTtpe定义schema，在CreateDataFrame的时候指定schema，这种叫编程指定模式。

DataFrame创建完成后，可以使用python进行高效的查询，查询方法主要有两大类。

- 使用DataFrame API指定，比如.show()和.head()都可以查看前几行，.count()可以统计行数，而且直接支持select操作。
- 另一种方式是用过SQL查询，使用spark.sql()。这里正常的SQL语句都可以，非常方便。

数据建模准备

理解完基础数据RDD和DataFrame后，现在开始准备数据建模。

知道数据建模或机器学习的朋友应该都知道，在数据建模时，基本上80%的工作都是整理清洗处理数据。

准备数据无外乎就是让实际数据变得更可用，比如去重，缺失值处理，异常数据等，为了做到这些，PySpark提供了比较丰富的方法。

首先看重复数据，为了检测到重复数据，可以可用常用的.distinct()，检测到了，使用.dropDuplicates()可以删除重复项，这里可以增加subset参数指定特定的列。

对于缺失数据，处理缺失值最简单的方法就是益处，这和去除数据的方法一样，但是直接移除可能会对数据集的可用性带来比较大的影响。所以通常情况下，我们会采用稍微折中一些的方式处理缺失值。

- 比如，当数据是离散布尔型时或已经是分类的，我们可以添加新列——Missing，并将其转化为一个分类变量。
- 如果数据是顺序类或数值类的，则可以使用描述性统计指标进行填充，如均值、分位数、众数等。

如果是缺失值比较少，可以使用.dropna()删除，如果缺失值比较多就需要使用上面的方法填充，填充的方式是.fillna()，该方法会将所有缺失值都是用该值填充，以平均值填充为例，需要先计算平均值，然后将平均值传递给.fillna()方法。

除了缺失值，还有一个对原数据会产生较大影响的还有离群值，离群值可以看作是一种特殊的离群值，填充方法都是相同的，不过这种离群值怎么检测出来呢?

常用于检测离群值的方法为四分位法（Q1 - 1.5IQR, Q3 + 1.5IQR），这个方法对于经常做数据分析的朋友来说应该是非常熟悉。在PySpark中使用.approxQuantile()方法可以或得分位数，获取后就可以计算IQR。

为了快速了解数据，我们可以打印前面几行查看数据内容，比如前面所说的take，head和show，但是有时候除了数据本身，我们还需要看数据每个列的类型，就可以使用.printSchema()，进一步为了描述数据的描述性统计指标，可以使用.describe()。如果想使用更多的统计指标，如峰度、偏度等，可以使用agg函数指定。

除了单列特征的统计性指标，我们还会经常描述两个特征之间相关性，PySpark同样提供了相应的方法，为.corr()，这里是Pearson相关系数。

除了直接的指标外，要了解数据，通过图像也是一个非常好的方式，特别是直方图和散点图，但是对于大数据来说，绘图是一件不太理智的行为，这里就不说了。

机器学习模型的福音——ML

ML是支持DataFrame的一个机器学习库，对于RDD，有MLlib支持（现在并未被积极维护）。对于一个机器学习模型而言，无外乎就是处理数据、建立模型，不过在spark惰性计算的条件下，ML的使用和python的一般使用方法略有不同。

PySpark将ML建模分为了三个部分，分别是转换器、评估器和管道，直观来说，转换器就是处理数据的过程，评估器就是建模的过程，管道则是整个建模的过程。

从高层次看，当用转换器的抽象类派生时，每个新的转换器需要实现.transfrom()方法，该方法需要传递一个待被转换的DataFrame。下面简单列一些转换器及介绍。

- Binarizer: 根据指定的阈值将连续变量转换为对应的二进制值
- Bucketizer: 根据阈值列别将连续变量转换为多项式
- ChiSqSelector: 只用卡方（Chi-Square）进行参数特征选择
- CountVectorizer: 用于文本标记
- DCT: 离散余弦变换取实数值向量，并返回不同长度的向量，但余弦函数只和在不同频率下震荡
- ElementwiseProduct: 返回传入该方法方向量和另一个传入参数scaling Vec向量的乘积
- HashintTF: 返回一个带有计数的有预定长度的hash转换向量
- IDF: 逆向逆向文件频率
- IndexToString: 使用StringIndexerModel对象中的编码将字符串索引反转到原始值
- MaxAbsScaler: 数据调整到[-1.0, 1.0]范围内
- MinMaxScaler: 数据调整到[0.0, 1.0]范围内
- NGram: 返回结果包含一些列n-gram
- Normalizer: 是用p范数将数据缩放为单位范数（默认为L2范数）
- OneHotEncoder: 分类列编码成二进制向量
- PCA: 主成分分析数据降维
- PolynomialExpansion: 向量的多项式展开
- QuantileDiscretizer: 与Bucketizer类似，不过传递的是一个numBuckets，计算数据的近似分位数进行分割
- RegexTokenizer: 使用正则表达式对字符串进行分词
- RFormula: 使用公式生成新列
- SQLTransformer: 使用SQL生成新列
- StandardScaler: 标准化（均值0，标准查1）
- StopWordsRemover: 删除文本中的停用词
- StringIndexer: 生成一个索引向量
- Tokenizer: 以空格为分割词进行分词
- VectorAssembler: 多个数字列合并为一列向量
- VectorIndexer: 为类别列生成索引向量
- VectorSlicer: 给定一个索引列表，其从特征向量只能够提取值
- Word2Vec: 将字符串转换为{string, vector}格式

转换器是数据的加工过程，评估器则是建模的过程，可以被视为需要评估的统计模型，对观测的对象做预测和分类。

常用的评估器有分类、回归和聚类。ML中一共提供了7中分类模型，7中回归模型和4中聚类模型。

分类模型有：

- LogisticRegression: 逻辑回归
- DecisionTreeClassifier: 决策树
- GBTClassifier: 梯度提升树
- RandomForestClassifier: 随机森林
- NaiveBayes: 朴素贝叶斯
- MultilayerPerceptronClassifier: 多层感知器
- OneVsRest: 将多分类问题简化为二分类问题

回归模型有：

- AFTSurvivalRegression: 生存回归，适用于明确的阶段性过程建模
- DecisionTreeRegression: 决策树回归
- GBTRegression: 梯度提升树回归
- GeneralizedLinearRegression: 广义线性回归
- IsotonicRegression: 保序回归
- LinearRegression: 线性回归
- RandomForestRegression: 随机森林回归

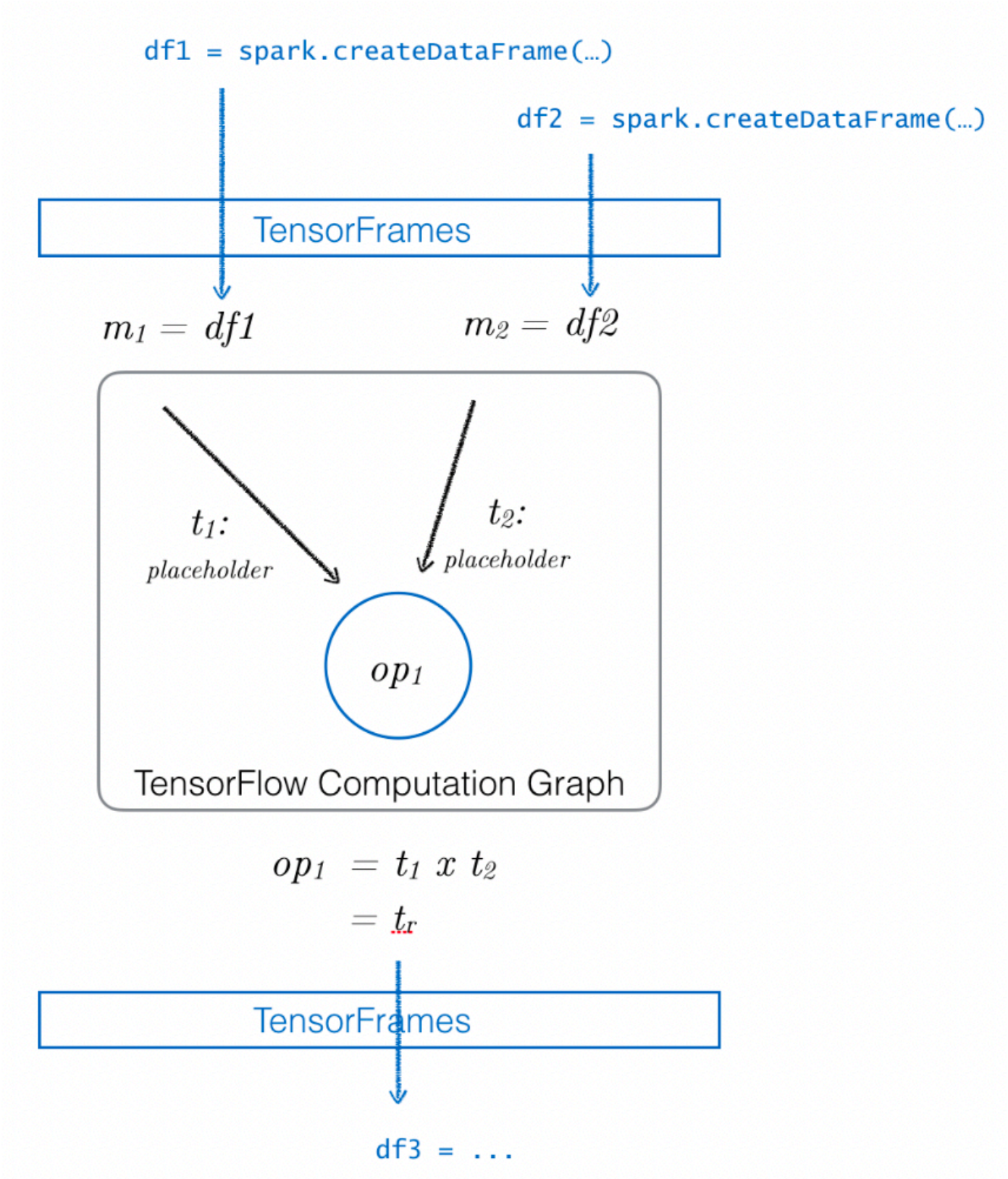
聚类模型有：

- BisectingKMeans: 二分k均值算法
- KMeans: k均值算法
- GaussianMixture: 高斯混合模型
- LDA: 隐含狄利克雷分布模型

PySpark ML 中的管道用来表示从转换到评估的端到端的过程。一个管道可以被认为是一系列不同阶段组成，通常情况下，前一阶段的输出会成为下一阶段的输入。

大数据上的深度学习——TensorFrames

TensorFrames是利用TensorFlow来操作Spark DataFrame，其大概结构为：



TensorFrames可以将DataFrame作为输入应用到TensorFlow计算图中，同时还允许将TensorFlow计算图输出返回到DataFrame中进行下一步到Spark处理。

TensorFrames当前在工业界生产环境中未有比较典型的应用，毕竟tensorflow自身提供了更友好的分布式学习架构。这里简单说两句深度学习。

深度学习是机器学习的一部分，其主要包含三个层次结构，分别是输入层、隐藏层和输出层。每个层由一个或多个具有连接的节点组成，输入层被动接受信息，隐藏层和输出层主动修改数据。深度学习相比传统机器学习，可以做非常复杂的特征工程，让神经网络中的不同神经元自动去学习输入数据的特征结构。

结构化图数据——GraphFrames

除了前面我们熟知的机器学习与深度学习，spark还有一个亮点是支持图结构数据的相关运算，图是我们生活中非常普遍的数据结构，比如人与人之间的社交关系，人与商品之间的消费关系，商品与地址的物流关系等等都是图结构。

而GraphFrames就是利用DataFrame来进行图计算的利器，图中的点和边由DataFrame表示，允许存储每个节点和边的任意数据，这里有必要说一下GraphFrames与GraphX的关键性区别。

- GraphFrames利用了DataFrame API的性能优化和简单性
- GraphFrames可以使用Python、Java和Scala访问，但是GraphX只能使用Scala访问

GraphFrame构建图形时需要对点和边对命名做一些特殊处理。

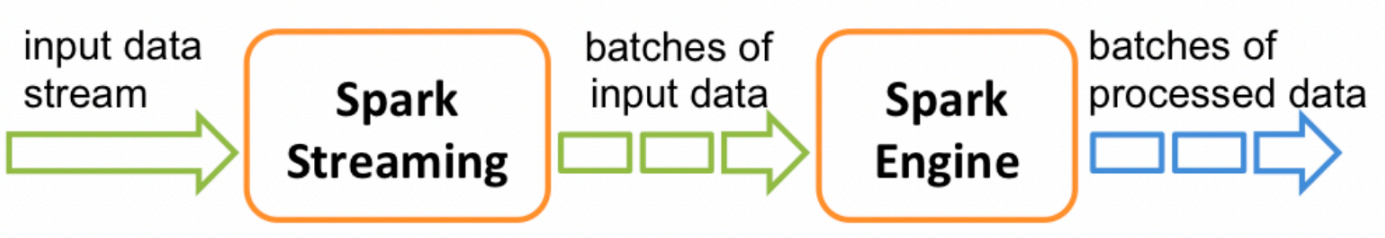
- 表示节点的列需要id的名称
- 表示边的列需要一个起点（src）和一个终点（dst）

在查询过程中，可以使用.edges.filter()选择边，通过degree可以查询图中节点的度，并且还有inDegree和outDegree。

说到图，不得不说一下Google Search Engine中的PageRank。在越是重要的网站接收到的其他网站的链接就越多的假设下，其工作原理是对连接页面的数量和质量进行计数，从而估计该页面的重要性。GraphFrames中已经包含了PageRank的API，可以使用.PageRank(resetProbability, maxIter)，其中resetProbability表示复位到随机节点的概率，maxIter表示最大迭代次数。

结构化流——Spark Streaming

Spark Streaming是一种可扩展、容错的数据流系统，它采用RDD的批量处理模式并加快处理速度，它的工作流大致如下：



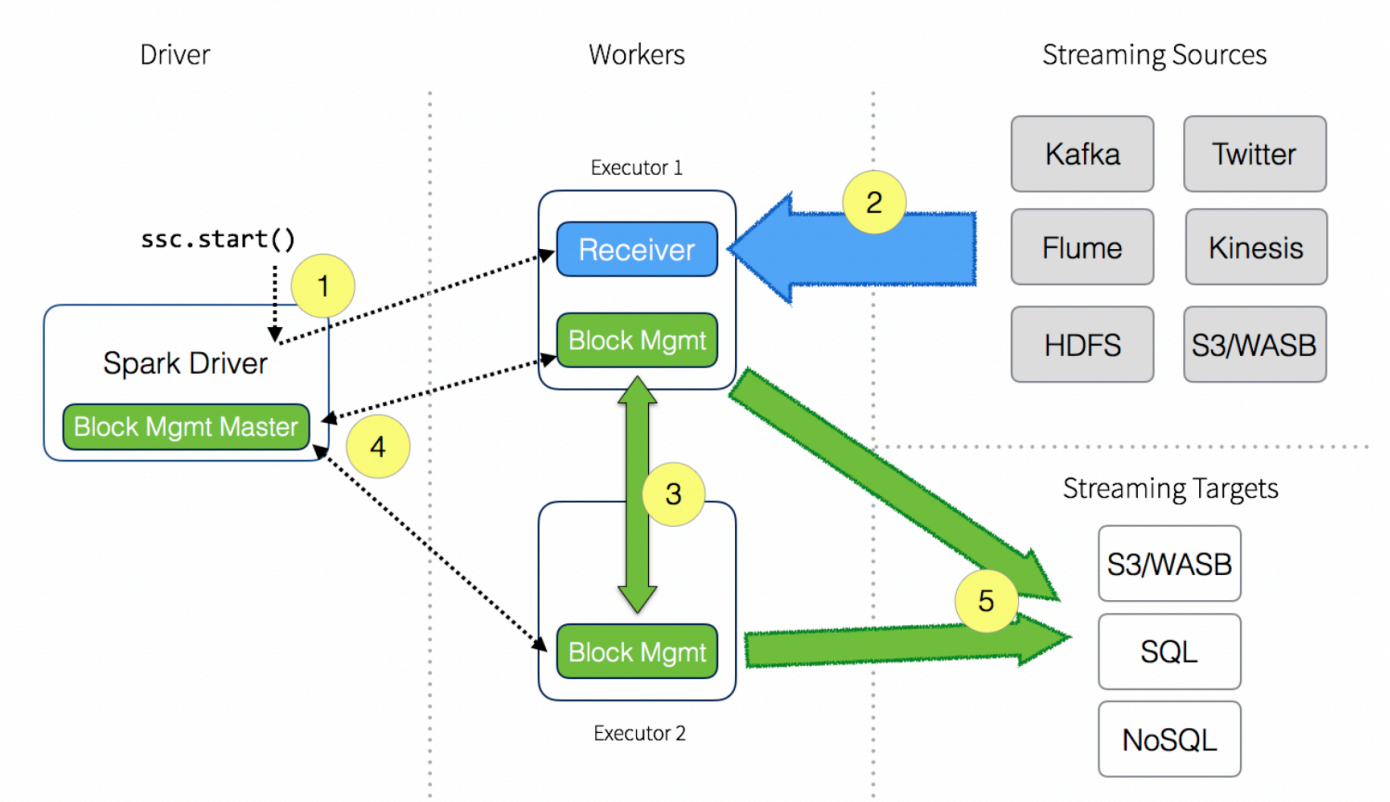
Spark Streaming接收输入数据流，并在内部将数据流分为多个较小的batch，Spark引擎将这些输入数据的batch处理后，输出被处理过的batch结果集。

当前Spark Streaming已经支持流非常多的组建，比较常用的有：Kafka，Flume，HDFS/S3，Kinesis和Twitter等。

结成了这么多组建，适应的应用场景也很刚需，Spark Streaming常用场景有：

- 流ETL：将数据推送到下游系统之前进行持续的清洗和聚合
- 粗发器：实时检测行为和异常事件，及时触发下游动作
- 数据浓缩：将实时数据与其他数据集连接，进行更丰富的分析
- 复杂会话和持续学习：持续分析关联数据，以更新机器学习模型

在了解完具体的应用场景后，最后来了解一下数据流的工作流程，大致工作流程如下图所示：



1. 当Spark Streaming上下文启动时，驱动进程将对executor执行长时间运行的任务
2. executor中的Receiver从Streaming源接收数据流，Receiver将输入的数据流分为多个数据块并将这些块保留在内存中
3. 为流避免丢失，这些块会被复制到另一个executor中
4. 块ID信息被传送到driver上的Block Management Master。
5. 对于在Spark Streaming Context内配置的批次间隔，驱动程序将启动Spark任务来处理这些数据，然后被持久化到任意数据的目标数据存储中，比如云存储，关系数据库和NoSQL。

打包spark——spark-submit

描述性分析数据的时候可以在jupyter等交互式分析界面完成，但是当需要同时运行一堆程序时，就需要对应用程序打包，spark-submit就提供这样一个API，通过配置一些参数就可以将一堆程序跑起来，这些参数有：

- --master: 设置主节点URL的参数，支持的语法有
  - local: 执行本地机器的代码
  - spark://host:port : Spark单机集群
  - mesos://host:port : 部署在Mesos上的Spark集群
  - yarn: 从运行Yarn的头节点提交作业
- --deploy-model: 是否在本地启动Spark驱动程序，或在集群内的其中一台机器上启动
- --name: 应用程序名称
- --py-files: .py、.egg或.zip的Python应用程序，这些文件会被交付给每一个执行器
- --files: 以逗号分割的文件列表
- --conf: 应用程序的配置（spar应用程序配置优先级SparkContext > spark-submit > spark-defaults.conf）
- --properties-file: 配置文件，与spark-defaults.conf相似
- --driver-memory: 在驱动程序上分配多少的内存

- --executor-memory: 每个执行器上分配的内存
- --help: 帮助信息
- --verbose: 打印附加调试信息
- --version: 版本信息
- --driver-cores: 驱动程序内核数量（仅在单机或Yarn上可用）
- --queue: 在Yarn上运行的队列（仅在Yarn上可用）
- --num-executors: 指定执行器数量（仅在Yarn上可用）
- --supervise: 当驱动程序丢失或失败的时候，重启该驱动（仅在单机或Mesos上可用）
- --kill: 用于杀死任务，赋予submission_id
- --status: 请求指定的应用程序状态

小结

整体而言，PySpark作为一个大数据工具，性能是杠杠的，里面的方法也比较简单，可常用的SQL、Python相关方法联系都比较紧密，使用成本并不高。

spark上的MLlib（对scala）模块和ML（对python）模块对大数据机器学习而言是不过的选择，只是在模型选择上相对稀缺，而且不易自定义一些模型。同样，对于Graphx也是如此。至于深度学习，虽然有tensorDataFrame，但是咋生产环境上不建议使用，如果真要用深度学习，还是用深度学习原生框架会比较好。

总之，spark处理大数据是非常有优势的，但是过多、过细、过于复杂的数据还是尽量不要在上面进行，毕竟在非常大的数据上使用非常复杂的模型也得不偿失。

reference：learningPySpark

完～