Hadoop

解决的问题：

大数据（也就是之前无法在一台计算机上存储和处理的数据）的可靠存储和处理。并且能够实现并行计算，在不同机器上可以分别进行（分工完成），最后可以联合起来完成一个问题。

关键思想：

1.HDFS（由普通 PC 机组成的集群）

NameNode 是名空间，并且是用户层面上文件的访问、打开等，也负责文件到 DataNode 的映射，是管理者

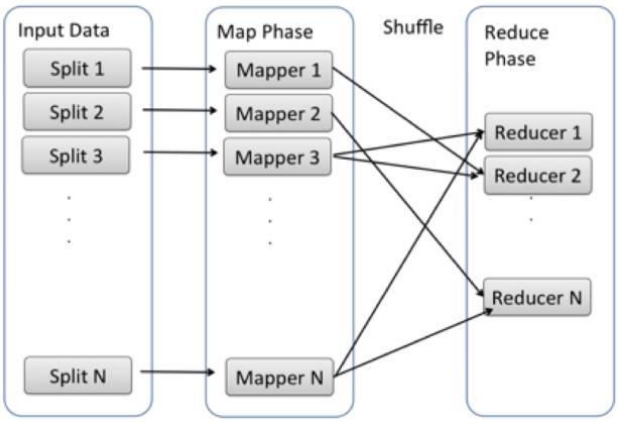
DataNode 完成具体的读写请求，文件也是被存储在一组 DataNode 上

2.Mapreduce（受 GoogleLab 的编程包影响提出的编程框架）

从 Map 到 Shuffle 到 Reduce 的过程

接收输入的数据集，切分为若干个数据块，进行任务的分配，从而可以并行的处理。每个集群进行自己的任务的计算

对于 map 任务之后的输出，框架对其进行排序（shuffle 的过程对其进行整合、排序、汇总），输入给 reduce 任务，也就是最后的任务。采用了 master/slave 的框架，一个 master 可以控制多个 slave（每个集群节点一个 slave）



Hbase

解决问题：

海量数据的实时查询、随机查询。其本身是个非关系型数据库

是与 hadoop 相比，hadoop 更适合大文件、大数据的存储，但是对于具体的、个别的数据的快速查询，hbase 更能解决

关键思想

1.Zookeeper 群、Master 群、RegionServer 群的配合

Zookeeper 负责存储、协调

Master 负责管理 Region，分配

RegionServer 存储数据，包括若干个 region，每个 region 根据 rowkey 的值进行区间维护，rowkey 可以理解为主键。每个 region 由若干个 store 构成

【store 中的 storefile 其实是存放了索引，真实的文件依然存储在 HDFS 上。同时还有 memstore 也就是写缓存，这样实现了速度的提升

2.面向列族存储，拥有良好扩张性

每一个列都由一个列族前缀和修饰符连接而成，物理上按照列族存储。
存储达到阈值时会做切分。切分的方式也有很多种。

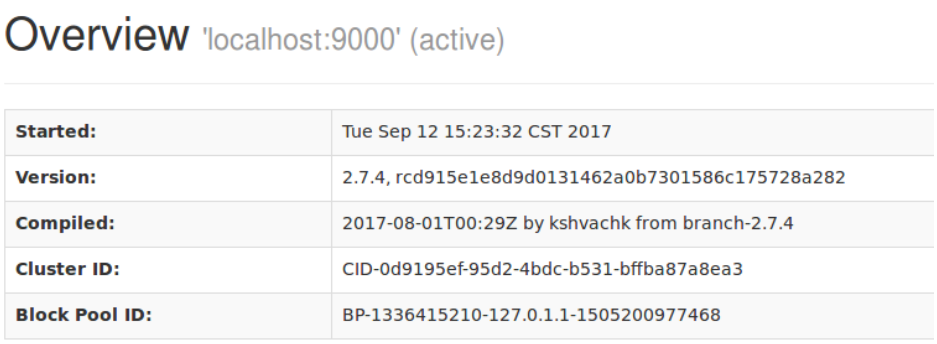
Spark
解决问题：
尽管 hadoop 已经完成了一个非常完整的分布式计算系统，但是每次读写都需要经过磁盘，速度很慢。Spark 提出的共享内存，可以将中间结果保存在内存中，提高了计算的速度，尤其是迭代计算，同时解决了编程难上手的问题

关键思想
1.需要多次反复计算的结果缓存起来，这样就不需要多次访问 hdfs（也就是磁盘）
2.RDD（弹性分布式数据集）由很多分区（partition）构成，只读分区。每个分区（也是会在不同的机器上）对应一个任务，其中还会记录依赖关系（继承、转移），最后还需要 action 进行资源的整合，同时具有容错机制（基于之前的依赖关系）

环境配置
Hadoop-2.7.4
Hbase-1.3.1
Spark-2.2.0
系统 ubantu14.04
JDK 1.8
Scala 2.12.3
Hadoop 采用单机模式进行配置（还没有进行分布式的搭建）
例子：grep 和 pi 的计算
1.使用 hadoop 自带的例子 根据范式输出结果
2.使用自带的例子进行 pi 的计算（自带的蒙特卡洛方法计算 pi 值），这个例子可以设置 map 的个数

体会：会显示 shuffle 过程中的错误信息，显示很长的步骤，对输入输出都有提示，如果是自带的例子还是比较简单，用 jar 的格式

伪分布式配置



也运行了之前单机模式下的例子，得出的结果完全不同。速度上可能是在虚拟机上运行、数据量不大还没有感受到特别大的差异

Hbase 使用

1.面向列族的存储模式，column 和后面的标识符隔开，和 SQL 中存储不一样

2.删除表格的时候需要先 disable 才能删除

3.用 get 查询的时候只能有层次的（从表到行，再到列族等等查询）

4.会給每一行新添加的内容增加时戳，timestamp，这样对之后的数据查询也很有利（数据更细直接 put 就完成）

Spark 使用

1.scala 语言在命令行里使用也十分简洁，比 hadoop 的 mapreduce 简单很多（用 shell 就可以启动 scala 的命令行的计算，简洁很多），语言里面也可以嵌入 java 的语法

2.也用 spark 自带的函数进行 pi 的计算，发现结果总是偏大（查资料发现也是蒙特卡洛的计算方法，原理也是一样但是计算结果和 hadoop 的计算结果很不一样）

3.自带也有 Python 的计算，非常简单

4.做了 RDD 的最基础的操作，可能是还没有数据集所以和一般的数值操作没有很大的比较