- 并行计算技术的分类：
    1. Flynn分类法：依据计算机在单个时间点能够处理的指令流数量和依据计算机在单个时间点能处理的数据流数量
    2. 按存储访问结构分类：共享内存访问结构、分布式内存访问结构和分布共享式内存访问结构
    3. 按系统类型分类：多核并行计算系统、对称多处理系统（多个相同类型的处理器通过总线连接并共享存储器）、大规模并行处理系统（专用内联网连接组成）、集群（网络连接的普通商用计算机）和网格（用网络连接的一组异构计算机构成的）。
    4. 按应用的计算特征分类：数据密集型（数据量大但是计算简单）、计算密集型（数据不大但是计算复杂的并行处理）和两者混合
    5. 按并行程序设计方式：共享存储变量方式、消息传递方式、mapreduce方式和其他（mapreduce难以满足高实时性和高数据相关性的大数据处理需求）
- 并行计算面临的主要问题：
    1. 网络互联技术
    2. 存储访问体系结构
    3. 分布式数据和文件管理
    4. 并行计算的任务划分和算法设计
    5. 并行程序设计模型和语言
    6. 数据访问和通信控制
    7. 可靠性和容错性
- 大数据的5大特点：
    1. 大体量
    2. 多样性
    3. 时效性
    4. 准确性
    5. 大价值
- 大数据的分类：
    1. 结构化、半结构化和非结构化
    2. 批处理和流式计算
    3. 传统的查询分析和复杂的数据挖掘
    4. 实时和非实时
    5. 简单关系数据和复杂关系数据
- 大数据主要技术层面
    1. 基础设施层：大数据分布存储和并行计算的硬件基础和平台，比如集群等
    2. 系统软件层：分布式文件系统和数据查询管理系统，大数据并行计算模式和系统
    3. 并行化算法层和应用层
- mapreduce：基于集群的高性能并行计算平台、并行计算与运行软件框架、并行程序设计模型与方法。
- mapreduce的处理过程：
    1. 对大量顺序式数据元素或记录进行扫描；
    2. 对每个数据元素进行处理，获取中间结果信息；
    3. 排序和整理中间结果以便后续处理；
    4. 收集整理中间结果；
    5. 结果输出
    6. 1和2是map的工作，4和5是reduce的工作
- mapreduce主要功能：

1. 自动数据划分和计算任务调度
2. 数据和代码互定位（一个基本原则是本地化数据处理，即尽可能处理本地磁盘上面的数据），实现了代码向数据的迁移
3. 系统优化：减少数据通信开销，在进入reduce之前会有合并处理
4. 出错检测和恢复

- mapreduce主要技术特征：
  1. 向外扩展而不是向上扩展：选用大量便宜的而不是选一些很贵的服务器
  2. 失效被认为是常态
  3. 代码向数据迁移
  4. 顺序处理数据，避免随机访问
  5. 为开发者封装细节
  6. 平滑的可扩展性
- HDFS:hadoop给予每个从节点上的本地文件系统，构建一个逻辑上整体化的分布式文件系统，以此提供大规模可扩展的分布式数据存储功能。其中主控节点叫做NameNode，从节点叫做DataNode
- HBase：HDFS难以管理结构化/半结构化的海量数据，所以hadoop提供了一个大规模分布式数据库管理和查询系统HBase。HBase是一个建立在HDFS上的一个NoSQL，提供了对结构化/半结构化甚至非结构化的大数据的实时读写和随机访问能力。
- Hive：建立在hadoop上的数据仓库，主要用来管理存储于HDFS或HBase中的数据，可以使用类似SQL的HiveQL来进行查询，并且HiveQL能在底层实现时转换为相应的mapreduce程序
- HDFS的6种特征：
  1. 大规模数据分布存储能力
  2. 高并发访问能力
  3. 强大的容错能力
  4. 顺序式文件访问：大数据批处理主要方式是顺序处理，所以HDFS支持大数据的快速顺序读出，但是随机访问效率差
  5. 一致性模型：支持大量数据一次写入，多次读取，不支持写入数据的更新操作，但是允许在文件末尾添加
  6. 数据块存储模式：采用大粒度数据块存储文件，默认64MB
- NameNode是一个主服务器，用来管理整个文件系统的命名空间和元数据，以及处理外界的文件访问请求；NameNode主要保存了文件系统的3中元数据：
  1. 命名空间，即整个分布式文件系统的目录结构
  2. 数据块对应文件名的映射表
  3. 每个数据块副本的位置信息
- HDFS文件访问过程：
  1. 用户的应用程序通过客户端将文件名发送到NameNode
  2. NameNode接收到后在HDFS目录中检索对应的数据块，再根据数据块的信息找到保存数据块的DataNode地址，返回给客户端
  3. 客户端收到响应后与DataNode并行的进行数据传输，并将操作结果日志等提交到NameNode
- mapreduce并行编程框架基本处理过程：
  1. 各个map结点对所划分的数据进行并行处理，从不同的输入数据产生相应的中间结果输出
  2. 各个reduce结点也各自进行并行处理，各自负责处理不同的中间结果的集合
  3. 在进行reduce之前，必须等到所有的map结点处理完毕，因此在进入reduce结点前有一个同步障，这一阶段也会对map的输出做一点处理
  4. 汇集所有reduce结点的输出
- Combiner：将两个主键相同的值直接传给reducer会效率低下，所以Combiner的作用就是在输入到reducer

之前会中间结果进行优化

- Patitioner：为了保证将所有主键相同的键值对传给同一个reduce结点，要对传入reduce结点的中间结果进行处理，消除相关性。Patitioner的过程在map和reduce过程之间
- 为了实现本地化计算原则，让每个从节点同时作为DataNode和TaskTracker，NameNode和JobTracker的话可以在一起也可以不在一起
- mapreduce程序执行过程：
    1. 用户程序客户端通过作业客户端接口程序JobClient提交一个用户程序
    2. JobClient向JobTracker提交作业执行请求并获得一个JobID
    3. JobClient同时也将用户程序作业和待处理的数据文件信息准备好并存储在HDFS中
    4. JobClient正式向JobTracker提交和执行作业
    5. JobTracker接受并调度改作业，进行作业的初始化准备工作，根据待处理数据的实际分片情况，调度和分配一定的Map结点来完成作业
    6. JobTracker查询作业中的数据分片信息，构建并准备相应的任务
    7. JobTracker启动TaskTracker结点开始执行具体任务
    8. TaskTracker根据分配到的任务，获取相应的作业数据
    9. TaskTracker结点创建所需要的java虚拟机，并启动相应的map任务或reduce任务
    10. 若是map任务，则吧中间结果输出到HDFS上，若是reduce任务，则输出结果
    11. TaskTracker向JobTracker报告任务完成，若Map任务完成后后续还有Reduce任务，则分配和启动Reduce结点处理中间结果并输出最终结果
- 当一个作业被提交到hadoop系统时，这个作业的输入数据或被划分成很多等长的数据块，每个数据块都对应一个Map任务，这些map任务同时执行，并行的处理数据。Map任务的输出数据会被排序，然后被系统分发给Reduce任务作进一步处理。
- MapReduce框架下作业和任务的执行流程：
    - 作业执行流程：
        1. 作业的运行和生命周期分为：准备阶段，运行阶段和结束阶段
        2. 准备阶段：初始化，进行读入数据块描述信息，创建所有的map和reduce任务
        3. 运行阶段：等待任务被调度，当第一个任务开始执行时，开始进入真正的计算，当所有的map和reduce任务完成后，进入等待状态，此时另一个作业清理任务启动，清理作业的运行环境，进入结束阶段
        4. 结束阶段：作业清理完成后，作业最终达到成功状态，生命周期结束
    - 任务执行流程：
        1. 任务是MapReduce框架下并行化计算的基本单位
        2. 感觉不会考细节
- 数据输入格式InputFormat(抽象类)：有3个任务：
    1. 验证数据的输入形式和格式
    2. 将输入数据分割为若干逻辑意义上的InputSplit，每个InputSplit作为单独的Mapper输入
    3. 提供一个RecordReader，用于将Mapper的输入处理转化为若干输入记录
    - 使用方法：job.setInputFormatClass(KeyValueTextInputFormat.class)
    - 常用的是实现类是FileInputFormat类：用于从HDFS中读取文件并分块，这些文件可能是文本文件或者顺序文件
    - TextInputFormat：默认输入格式，读取文本文件的行：键：当前行的偏移位置；值：当前行内容；对应的是LineRecordReader，读入每一行的数据记录；使用此类时，Mapper的输入应该是<LongWritable,Text>
    - KeyValueTextInputFormat：将行解析为键值对；键：行内守鹤制表符前的内容；值：其余内容;对应的是KeyValueLineRecordReader，将第一个制表符前的作为键，后面的都是值；Mapper的输入应该

是<Text,Text>

- InputSplit：将单独的作为一个Mapper的输入，也就是说Mapper的数量==InputSplit的数量；InputSplit的类型根据选择的InputFormat来决定；常用的是FileSplit
- Mapper类：4个方法：setup(),map(key,value,context),cleanup(context),run(context)；
- Combiner类：一个Mapper结点输出的键值对首先需要进行合并处理，以便将key相同的键值对合并为一个，减少网络传输，系统默认提供了一个Combiner，也可以用户自己定制；Combiner实际上就是一个Reducer，所以也是继承于Reducer类，特别注意：Combiner不能改变原来Mapper的输出键值对的类型。
- Partitioner：为了避免在Reduce计算过程中不同Reduce节点存在数据相关性。因为Reduce处理的数据可能来自不同的Mapper，所以Mapper的中间输出结果需要进行分区处理，保证相关的数据送到一个Reduce上面。Hadoop默认自带了一个HashPartitioner,对键值对中的key取hash值并按Reducer的数目取模来分配；也可以用户自己定制。使用方式：job.setPartitionerClass(MyPartitioner)
- Sort：感觉不考
- Reducer：没啥内容
- OutputFormat：没啥内容
- HBase技术特点：列式存储；表数据是系数的多维映射表；读写的严格一致性；读写速度快；线性扩展性；海量存储能力；数据自动分片；失效恢复；javaAPI
- HBase基本数据结构：行关键字（唯一，字典排序）、列族（相当于一个容器，列族之间独立存放）和列名以及时间戳
- HBase查询模式：通过三元组[行key，列（列族名：列名），时间戳]来确定一个存储单元；HBase的查询方式：
    1. 通过单个行key访问
    2. 通过行key的范围
    3. 全表遍历
- HBase不支持事务，跨行的原子性无法实现
- HBase编程案例：P157
- K-means算法：
    1. 定义簇的数据结构

```
public class Cluster implements Writable{
    private int id;
    private long numOfPoints;
    private Instance center;
}
```

    2. 初始化K个中心点：扫描整个数据集，若中心点集未满，则添加当前点，若已满，则按1/(1+K)的概率替换其中一个点
    3. Mapper中的setup():读入初始簇重心；map()方法：为每个传入的数据点找到离奇最近的簇，并输出<簇的id，该点>

```
public void map(LongWritable key,Text value,Context context) throws Exception{
    Instance instance = new Instance(value.toString());
    int id;
    try{
        id = getNearest(instance);
        if (id == -1){
```

```
            throws Exception(-1);
        }else{
            Cluster cluster = new Cluster(id,instance);
            cluster.setNumOfPoints(1);
            context.write(new IntWritable(1),cluster);
        }catch{
            //
        }
    }
}
```

4. Combiner：