**大数据**：指无法在可承受的时间范围内用常规软件工具进行捕捉、管理和处理的数据集合，是需要新处理模式才能具有更强的决策力、洞察发现力和流程优化能力的海量、高增长率和多样化的信息资产；在维克托·迈尔-舍恩伯格及肯尼斯·库克耶编写的《大数据时代》 中大数据指不用随机分析法（抽样调查）这样的捷径，而采用所有数据进行分析处理。大数据的 5V 特点（IBM 提出）：Volume（大量）、Velocity（高速）、Variety（多样）、Value（价值）Veracity（真实性） 。

**数据分析**：是指用适当的统计分析方法对收集来的大量数据进行分析，提取有用信息和形成结论而对数据加以详细研究和概括总结的过程。这一过程也是质量管理体系的支持过程。在实用中，数据分析可帮助人们作出判断，以便采取适当行动。数据分析的数学基础在 20 世纪早期就已确立，但直到计算机的出现才使得实际操作成为可能，并使得数据分析得以推广。数据分析是数学与计算机科学相结合的产物。

**数据挖掘**：又译为资料探勘、数据采矿。它是数据库知识发现中的一个步骤。数据挖掘一般是指从大量的数据中通过算法搜索隐藏于其中信息的过程。数据挖掘通常与计算机科学有关，并通过统计、在线分析处理、情报检索、机器学习、专家系统（依靠过去的经验法则）和模式识别等诸多方法来实现上述目标。

1、自我介绍
2、介绍简历中的项目经历。
3、常规题目：最有成就感的事，学习的动力，最喜欢的科目。
4、业务问题：有一个类似抖音的 app，请你设计推荐算法。

**常见的个性化推荐原理**

**基于用户基本信息推荐** Demographic-based Recommendation
如：领域、职位、工作年龄、性别、所在地
这个是比较基础的推荐之一，基于用户的基本信息，可以根据他的这些信息给他推荐感兴趣的或者相关的内容。

**基于物品/内容基本信息推荐** Content-based Recommendation
文章的一些显性属性如：领域、主题、类型、来源
这也是一种基础的推荐，基于被推荐物的基本信息，或者说是被推荐物的显性属性。
前面这两种都比较基础，下面这一种会复杂一些。

**协同推荐 Collaborative Filtering**
需要通过用户行为来计算出用户或者物品间的相关性。
- **基于用户的协同推荐**

以人为本，找到和你相似的人后推荐他们看了而你没有看的内容。
- **基于物品的协同推荐**

以物为本建立各商品之间的相似度关系矩阵，“用户看了 x 也会看 y”。
小张和小明都不约而同地看了产品经理和 Google，这可以说明产品经理和 Google 有相似，那么之后有看了 Google 相关内容的用户就可以给推荐产品经理的相关内容。
基于用户和被推荐物推荐不需要特别多的数据，比较适合应用在冷启动阶段。
而协同推荐是基于大数据的，所以我前面举的例子都是简化之后的，在实际的操作过程中用户的行为会比前面的例子复杂的多，但是道理都是相通的。

**如何设计一个好的推荐系统？**
UGC 、编辑、热门
- UGC: 获取更多用户行为数据，用户显性和隐形的数据
- 编辑: 处于种种目的编辑的运营

- 热门: 最火最热门的内容

**新用户的冷启动**

在没有大数据做基础的时候，可以有下面两种解决方案：

有的时候你有的总内容数量远大于用户有“打分”的内容数量，对于这种情况的解决方案：

- 用户隐形打分
- 降维。Matrix of boolean feature，投射到低维空间，再用机器学习
- 结合基于物品基本信息的推荐

**多样性**

- 看过什么，推荐相同类型的；
- 为你推荐你需要也比较适合你的；
- 基于你的性格、兴趣等，推荐甚至连自己没想到过却真正感兴趣的；

这部分推荐听起来感觉是玄学, 但是给我们推荐的一个思路就是不要仅仅局限于某一种类型的推荐上，相同类型、需求甚至挖掘兴趣，都是可以考虑的推荐内容。

**实时性**

根据用户的行为， 实时的调整。好的推荐系统是在不断更新的。

**如何判断一个推荐系统做得好不好？**

获得反馈并一直迭代

与推荐系统的交互有用吗？他们对收到的推荐结果满意吗？

设计评测标准

1.能吸引更多的用户看内容的详情页

2.促使单个用户浏览更多内容

5、一个线段上任意取两点，能组成三角形的概率？我觉得是四分之一啊，但面试官说是八分之一，不懂。

6、业务问题：有 uid，app 名称，app 类别，数据百亿级别，设计算法算出每个 app 类别只安装了一个 app 的 uid 总数。

应该用 map reduce 吧，但我不会啊。准备写个 sql，结果写了半天还是写不出。面试完走到楼下就想出来了，233。

Map: input:  key=nothing, value=all data

output: key=uid, value={appName, appCategory}  //这个 map 是为了将 user 独立处理

Reduce: input = Map’s output

中间处理过程嵌套一个 Map-Reduce/或者使用内存，我理解每个 uid 的自身数据不会太多

inner Map: input: key=nothing, value = all this uid’s data

output: key=appCategory, value = 1 //相当于 word count

inner Reduce: input = inner Map’s output

output : key= appCategory, value = count

if any count > 1, val = 0.

else val = 1

output: key = total_count_key, value = val

后面可以统计一下一个输出了多少个 1，可以在用 MapReduce。

7、业务题：有一个网页访问的数据，包含 uid，ip 地址，url，文章资料。设计算法预测用户性别。
1. 我们的输入是一个用户的 APP 列表，输出是这个用户为男性的可能性
2. 输入类似于 X = ("美图秀秀",'搜索', '腾讯新闻', '微信', '快乐炸金花', '欢乐斗地主', '欢乐斗牛', '百度', '蘑菇街', '金山电池医生')，而我们输出就是 P(male|X)这样的条件概率。
3. 完整的贝叶斯公式为 P(male|X) = P(male) * P(X|male) / P(X)，其中 P(male)是男性的概率；P(male|X)是安装了这一堆 APP 的情况下此用户是男性的概率；P(X)是着堆 APP 被安装的概率。 但是对于一组固定的 X 来说，P(X)是个常量，我们可以不计算它。
4. 由于 P(X)不用计算，我们必须分别计算出 P(male) * P(X|male)和 P(female) * P(X|female)
5. 由于训练数据集中 P(appx|male)=0 的情况，可以使用贝叶斯 m 估计的方法处理
https://blog.csdn.net/cnweike/article/details/47167411

# 1、欠拟合和过拟合的解决方法

**欠拟合**：模型拟合不够，在训练集(training set)上表现效果差，没有充分利用数据，预测的准确率比我们设计的模型远远低很多，拟合结果严重的不符合预期。

**解决办法**：增加模型的复杂度，不要用简单的线性回归，适当的采用二次回归，将训练集合扩大，采集更多的数据。

**过拟合**：模型过度拟合，在训练集(training set)上表现好，但是在测试集上效果差，也就是说在已知的数据集合中非常好，在添加一些新的数据进来效果就会差很多，造成这样的原因是考虑影响因素太多，超出自变量的维度过于多了。

**解决办法**：减少模型的复杂度，适当的将训练集合进行筛选。

# 1、 你处理过的最大的数据量?你是如何处理他们的?处理的结果。

所谓海量数据处理，其实很简单，海量，海量，何谓海量，就是数据量太大，所以导致要么是无法在较短时间内迅速解决，要么是数据太大，导致无法一次性装入内存。

那解决办法呢?针对**时间**，我们可以采用巧妙的算法搭配合适的数据结构，如 Bloom filter/Hash/bit-map/堆/数据库或倒排索引/trie/，针对**空间**，无非就一个办法：大而化小：分而治之/hash 映射，你不是说规模太大嘛，那简单啊，就把规模大化为规模小的，各个击破不就完了嘛。

至于所谓的单机及集群问题，通俗点来讲，单机就是处理装载数据的机器有限(只要考虑 cpu，内存，硬盘的数据交互)，而集群，机器有多辆，适合分布式处理，并行计算(更多考虑节点和节点间的数据交互)。

再者，通过本 blog 内的有关海量数据处理的文章，我们已经大致知道，处理海量数据问题，无非就是：

1. 分而治之/hash 映射 + hash 统计 + 堆/快速/归并排序；
2. 双层桶划分
3. Bloom filter/Bitmap；
4. Trie 树/数据库/倒排索引；
5. 外排序；
6. 分布式处理之 Hadoop/Mapreduce。

**自我介绍：**同时他也会看你的简历

**提问：**

## 2、字典是怎么实现的，key 怎么查找，key 冲突了怎么办

字典类型是 Python 中最常用的数据类型之一，它是一个键值对的集合，字典通过键来索引，关联到相对的值，理论上它的查询复杂度是 O(1) ：

```
>>> d = {'a': 1, 'b': 2}

>>> d['c'] = 3

>>> d

{'a': 1, 'b': 2, 'c': 3}
```

### 哈希表 (hash tables)

哈希表（也叫散列表），根据键值对(Key-value)而直接进行访问的数据结构。它通过把 key 和 value 映射到表中一个位置来访问记录，这种查询速度非常快，更新也快。而这个映射函数叫做哈希函数，存放值的数组叫做哈希表。 哈希函数的实现方式决定了哈希表的搜索效率。具体操作过程是：

1. 数据添加：把 key 通过哈希函数转换成一个整型数字，然后就将该数字对数组长度进行取余，取余结果就当作数组的下标，将 value 存储在以该数字为下标的数组空间里。
2. 数据查询：再次使用哈希函数将 key 转换为对应的数组下标，并定位到数组的位置获取 value。

   但是，对 key 进行 hash 的时候，不同的 key 可能 hash 出来的结果是一样的，尤其是数据量增多的时候，这个问题叫做哈希冲突。如果解决这种冲突情况呢？通常的做法有两种，一种是链接法，另一种是开放寻址法，Python 选择后者。

## 开放寻址法（open addressing）

开放寻址法中，所有的元素都存放在散列表里，当产生哈希冲突时，通过一个探测函数计算出下一个候选位置，如果下一个获选位置还是有冲突，那么不断通过探测函数往下找，直到找个一个空槽来存放待插入元素。

1. http 的状态码

   1XX：消息　continue、switching protocol processing

   2XX：成功　　OK 、created、accepted

   3XX：重定向　multiple choice、moved permanently

   4XX :请求错误 Bad request、unauthorized、payment required、forbidden、not found

   5XX、6XX：服务器错误 internal server error

2. mysql 主键插入有重复值怎么办

   1. replace into

   replace 类似于 insert，区别在于如果新插入的行的主键或唯一索引值已存在，则先删除相应的行在插入新行。

[sql] view plaincopy

```sql
create table t1 ( a int not null primary key);
replace into t1 values (1,3);
select row_count();
select * from t1;
```

2. insert...on duplicate key update

与 replace into 类似，也是在 insert 时执行主键或唯一索引判重，如果重复则执行相应的 update 操作。如果存在多个唯一索引且重复的行多于一行，则只修改一行。例子：

[sql] view plaincopy

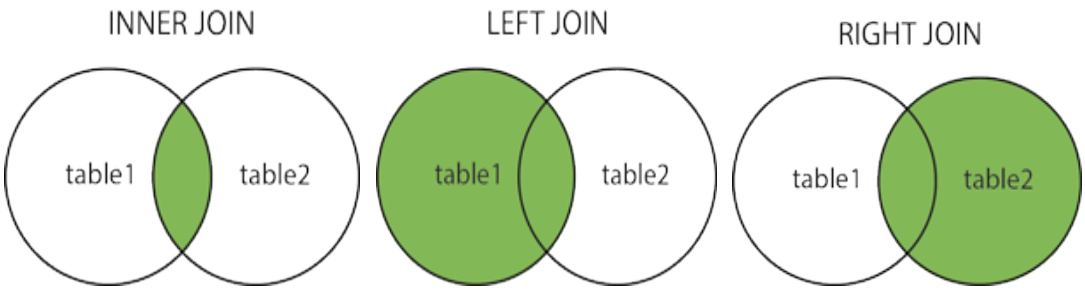
```sql
create table t1 ( a int not null unique key, b int not null unique key, c int not null);
insert into t1 values (1,1,1);
insert into t1 values (2,2,1);
select * from t1;
insert into t1 values (1,2,3) on duplicate key update c=c+1;
select row_count();
select * from t1;
insert into t1 values (1,2,3) on duplicate key update c=c+1;
insert into t1 values (1,2,3) on duplicate key update c=c+1;
select * from t1;
```

3. mysql 分组查询所有组的最大值

select uid,max(buy_time) from user_order group by uid;

4. mysql 的 join 用法

5. **INNER JOIN（内连接,或等值连接）**：获取两个表中字段匹配关系的记录。
6. **LEFT JOIN（左连接）**：获取左表所有记录，即使右表没有对应匹配的记录。
7. **RIGHT JOIN（右连接）**： 与 LEFT JOIN 相反，用于获取右表所有记录，即使左表没有对应匹配的记录。



假设 a,b 表中有相同的字段，如果从一个表中减去另一个表中的记录：

Select a.passenger_id from (select passenger_id from 表 1) a left join (select passenger_id from 表 2) b on a.passenger_id = b.passenger_id where b.passenger_id is null

## 8. 4 亿 qq 号找重复数量最多的 10 个

在 100G 文件中找出出现次数最多的 100 个 IP

搜索引擎会通过日志文件把用户每次检索使用的所有检索串都记录下来，每个查询串的长度为 1-255 字节。假设目前有一千万个记录（这些查询串的重复度比较高，虽然总数是 1 千万，但如果除去重复后，不超过 3 百万个。一个查询串的重复度越高，说明查询它的用户越多，也就是越热门），请你统计最热门的 10 个查询串，要求使用的内存不能超过 1G。

解答：我们知道，数据大则划为小的，如如一亿个 Ip 求 Top 10，可先%1000 将 ip 分到 1000 个小文件中去，并保证一种 ip 只出现在一个文件中，再对每个小文件中的 ip 进行 hashmap 计数统计并按数量排序，最后归并或者最小堆依次处理每个小文件的 top10 以得到最后的结。

但如果数据规模比较小，能一次性装入内存呢?比如这第 2 题，虽然有一千万个 Query，但是由于重复度比较高，因此事实上只有 300 万的 Query，每个 Query255Byte，因此我们可以考虑把他们都放进内存中去（300 万个字符串假设没有重复，都是最大长度，那么最多占用内存 3M*1K/4=0.75G。所以可以将所有字符串都存放在内存中进行处理），而现在只是需要一个合适的数据结构，在这里，HashTable 绝对是我们优先的选择。

所以我们放弃分而治之/hash 映射的步骤，直接上 hash 统计，然后排序。So，针对此类典型的 TOP K 问题，采取的对策往往是：hashmap + 堆。如下所示：

1. hash_map 统计：先对这批海量数据预处理。具体方法是：维护一个 Key 为 Query 字串，Value 为该 Query 出现次数的 HashTable，即 hash_map(Query，Value)，每次读取一个 Query，如果该字串不在 Table 中，那么加入该字串，并且将 Value 值设为 1；如果该字串在 Table 中，那么将该字串的计数加一即可。最终我们在 O(N)的时间复杂度内用 Hash 表完成了统计；
2. 堆排序：第二步、借助堆这个数据结构，找出 Top K，时间复杂度为 N‘logK。即借助堆结构，我们可以在 log 量级的时间内查找和调整/移动。因此，维护一个 K(该题目中是 10)大小的小根堆，然后遍历 300 万的 Query，分别和根元素进行对比。所

以，我们最终的时间复杂度是：O（N） + N' * O（logK），（N 为 1000 万，N'为 300 万）。

接下来再来说说我遇到的这个题，几乎就按这个思路来就好了：

对于 100G 的文件，先算算能有多少条 IP 呢？每条 IP 最长为 15 个字节，则 100G/15=6.7G 条，IP 一共有多少种呢，不考虑 IPv6，约有 256^4=2^32 条=4G 条，那么最极端的情况是每种 IP 都有，每个都出现那么一两条。

要解决该问题首先要找到一种分类方式，把重复出现的 IP 都放到一个文件里面，一共分成 100 份，这可以通过把 IP 对 100 取模得到，具体方法如去掉 IP 中的点转化为一个 long 型变量，这样取模为 0,1,2...99 的 IP 都分到一个文件了，那么这个分就能保证每一文件都能载入内存吗？这可不一定，万一模为 9 的 IP 特别多怎么办，可以再对这一类 IP 做一次取模，直到每个小文件足够载入内存为止。这个分类很关键，如果是随便分成 100 份，相同的 IP 被分在了不同的文件中，接下来再对每个文件统计次数并做归并，这个思路就没有意义了，起不到“大而化小，各个击破，缩小规模，逐个解决”的效果了。

好了，接下来把每个小文件载入内存，建立哈希表 unordered_map<string,int>，将每个 IP 作为关键字映射为出现次数，这个哈希表建好之后也得先写入硬盘，因为内存就那么多，一共要统计 100 个文件呢。

在统计完 100 个文件之后，我再建立一个小顶堆，大小为 100，把建立好并存在硬盘哈希表载入内存，逐个对出现次数排序，挑出出现次数最多的 100 个，由于次数直接和 IP 是对应的，找出最多的次数也就找出了相应的 IP。

这只是个大致的算法，还有些细节，比如第 90 到 110 大的元素出现次数一样呢，就随机舍弃掉 10 个吗？整个的时间复杂度分类 O(n)，建哈希表 O(n)，挑出出现最多次数的 O(nlogk)

## 9. 100 亿个数字找出最大的 10 个

1、首先一点，对于海量数据处理，思路基本上是：必须分块处理，然后再合并起来。

2、对于每一块必须找出 10 个最大的数，因为第一块中 10 个最大数中的最小的，可能比第二块中 10 最大数中的最大的还要大。

3、分块处理，再合并。也就是 Google MapReduce 的基本思想。Google 有很多的服务器，每个服务器又有很多的 CPU，因此，100 亿个数分成 100 块，每个服务器处理一块，1 亿个数分成 100 块，每个 CPU 处理一块。然后再从下往上合并。注意：分块的时候，要保证块与块之间独立，没有依赖关系，否则不能完全并行处理，线程之间要互斥。另外一点，分块处理过程中，不要有副作用，也就是不要修改原数据，否则下次计算结果就不一样了。

4、上面讲了，对于海量数据，使用多个服务器，多个 CPU 可以并行，显著提高效率。对于单个服务器，单个 CPU 有没有意义呢？

也有很大的意义。如果不分块，相当于对 100 亿个数字遍历，作比较。这中间存在大量的没有必要的比较。可以举个例子说明，全校高一有 100 个班，我想找出全校前 10 名的同

学，很傻的办法就是，把高一 100 个班的同学成绩都取出来，作比较，这个比较数据量太大了。应该很容易想到，班里的第 11 名，不可能是全校的前 10 名。也就是说，不是班里的前 10 名，就不可能是全校的前 10 名。因此，只需要把每个班里的前 10 取出来，作比较就行了，这样比较的数据量就大大地减少了。
如果需要多次 map 处理，最好用 spark 的流式处理，hadoop 比较适合一次 map-reduce 的操作。

## 10.判断别人发来的 qq 号是否在数据库中，提示我用树结构　搜索树

## 11.给你一个超级大的敏感词词典，判断用户发的语句中是否有敏感词，要考虑语义，用户体验

Hash 表

## 12.一个 7 升的桶和一个 3 升的桶，倒出 2 升的水

7L 桶倒满，用 7L 桶倒 3L 桶倒满，7L 桶剩 4L；

3L 桶倒掉，7L 桶剩下的水把 3L 桶的倒满，7L 桶剩 1L；

3L 桶倒掉，7L 桶把剩下的 1L 倒进 3L 桶；

7L 桶把水倒满，然后把只有 1L 水的 3L 桶倒满，7L 桶剩 5L 水；

3L 桶倒掉，7L 桶把 3L 桶倒满，7L 桶剩 2L 水。

## 13.Linux 下怎么查寻内存使用情况

在系统维护的过程中，随时可能有需要查看 CPU 使用率，并根据相应信息分析系统状况的需要。在 CentOS 中，可以通过 top 命令来查看 CPU 使用状况。运行 top 命令后，CPU 使用状态会以全屏的方式显示，并且会处在对话的模式 -- 用基于 top 的命令，可以控制显示方式等等。退出 top 的命令为 q （在 top 运行中敲 q 键一次）。
top 命令是 Linux 下常用的性能分析工具，能够实时显示系统中各个进程的资源占用状况，类似于 Windows 的任务管理器
　　可以选择按进程查看或者按用户查看，如想查看 oracle 用户的进程内存使用情况的话可以使用如下的命令：
　　$ top -u oracle
　　在命令行中输入　“top”
　　即可启动 top

pmap 可以根据进程查看进程相关信息占用的内存情况，(进程号可以通过 ps 查看)如下所

示：

$ pmap -d 5647

ps

如下例所示：

$ ps -e -o 'pid,comm,args,pcpu,rsz,vsz,stime,user,uid'  其中 rsz 是是实际内存

$ ps -e -o 'pid,comm,args,pcpu,rsz,vsz,stime,user,uid' | grep oracle |  sort -nrk

其中 rsz 为实际内存，上例实现按内存排序，由大到小

在 Linux 下查看内存我们一般用 free 命令：

```
[root@scs-2 tmp]# free
             total       used       free     shared    buffers     cached
Mem:       3266180    3250004      16176          0     110652    2668236
-/+ buffers/cache:     471116    2795064
Swap:      2048276      80160    1968116
```

测量一个进程占用了多少内存，linux 为我们提供了一个很方便的方法，/proc 目录为我们提供了所有的信息，实际上 top 等工具也通过这里来获取相应的信息。

/proc/meminfo 机器的内存使用信息

/proc/pid/maps pid 为进程号，显示当前进程所占用的虚拟地址。

/proc/pid/statm 进程所占用的内存

```
[root@localhost ~]# cat /proc/self/statm
654 57 44 0 0 334 0
```

查看机器可用内存

```
/proc/28248/>free
total used free shared buffers cached
Mem: 1023788 926400 97388 0 134668 503688
-/+ buffers/cache: 288044 735744
Swap: 1959920 89608 1870312
```

我们通过 free 命令查看机器空闲内存时，会发现 free 的值很小。这主要是因为，在 linux 中有这么一种思想，内存不用白不用，因此它尽可能的 cache 和 buffer 一些数据，以方便下次使用。但实际上这些内存也是可以立刻拿来使用的。

所以 空闲内存=free+buffers+cached=total-used

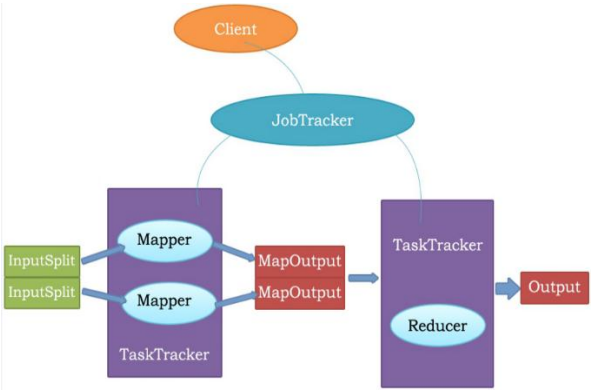
top 命令 是 Linux 下常用的性能 分析工具 ，能够实时显示系统 中各个进程的资源占用状况，类似于 Windows 的任务管理 器。下面详细介绍它的使用方法。

1、 MapReduce 原理

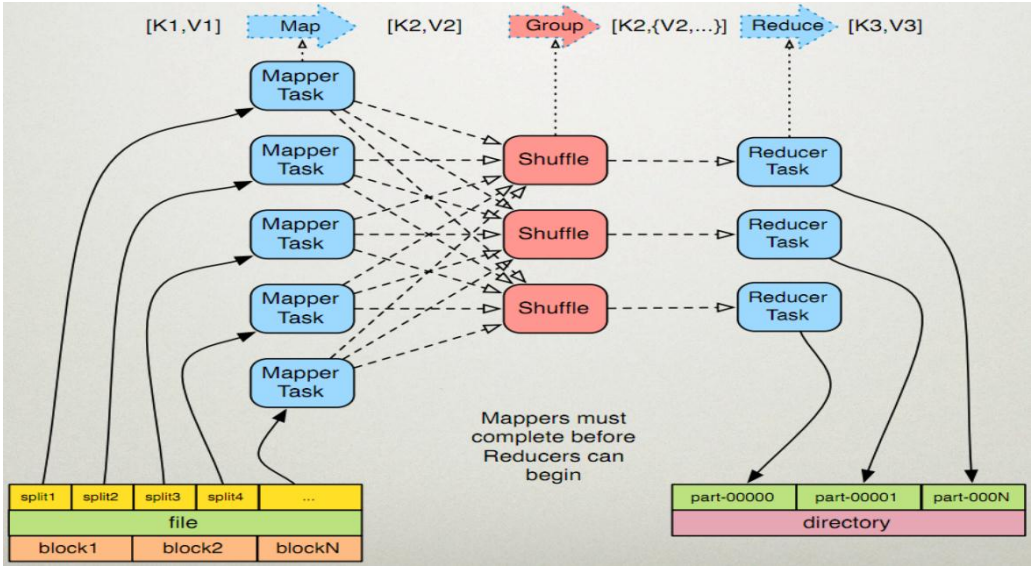
MapReduce 简介：MapReduce 是一种分布式计算模型，是 Google 提出的，主要用于搜索领域，解决海量数据的计算问题。

MR 有两个阶段组成：Map 和 Reduce，用户只需实现 map()和 reduce()两个函数，即可实现分布式计算。

MapReduce 执行流程




MapReduce 原理




MapReduce 的执行步骤： **整个 MapReduce 的过程大致分为 Map-->Shuffle（排序）-->Combine（组合）-->Reduce**

1、Map 任务处理

1.1 读取 HDFS 中的文件。每一行解析成一个<k,v>。每一个键值对调用一次 map 函数。 <0,hello you> <10,hello me>

1.2 覆盖 map()，接收 1.1 产生的<k,v>，进行处理，转换为新的<k,v>输出。<hello,1> <you,1> <hello,1> <me,1>

1.3 对 1.2 输出的<k,v>进行分区。默认分为一个区。

1.4 对不同分区中的数据进行排序（按照 k）、分组。分组指的是相同 key 的 value 放到一个集合中。 排序后：<hello,1> <hello,1> <me,1> <you,1> 分组后：<hello,{1,1}><me,{1}><you,{1}>

1.5 （可选）对分组后的数据进行归约。

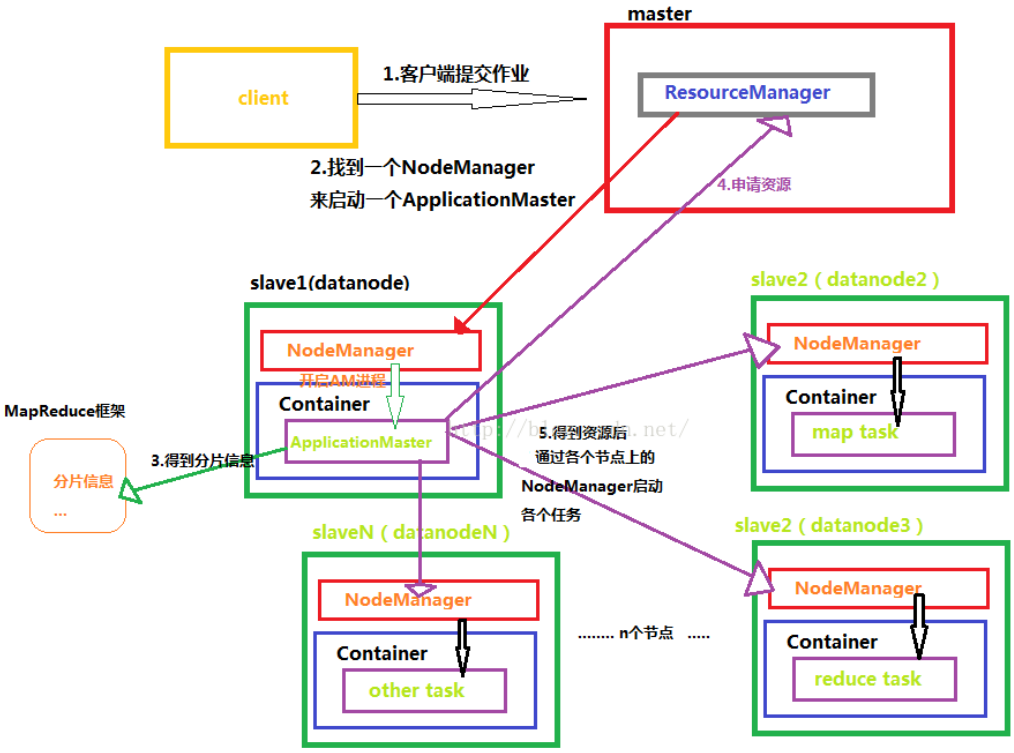
2、Reduce 任务处理

2.1 多个 map 任务的输出，按照不同的分区，通过网络 copy 到不同的 reduce 节点上。

2.2 对多个 map 的输出进行合并、排序。覆盖 reduce 函数，接收的是分组后的数据，实现自己的业务逻辑， <hello,2> <me,1> <you,1>

处理后，产生新的<k,v>输出。

2.3 对 reduce 输出的<k,v>写到 HDFS 中。




2、 大数据数据倾斜

什么是数据倾斜 :简单的讲，数据倾斜就是我们在计算数据的时候，数据的分散度不够，导致大量的数据集中到了一台或者几台机器上计算，这些数据的计算速度远远低于平均计算速度，导致整个计算过程过慢。

数据倾斜有一个关键因素是数据量大，可以达到千亿级。

数据倾斜的原理

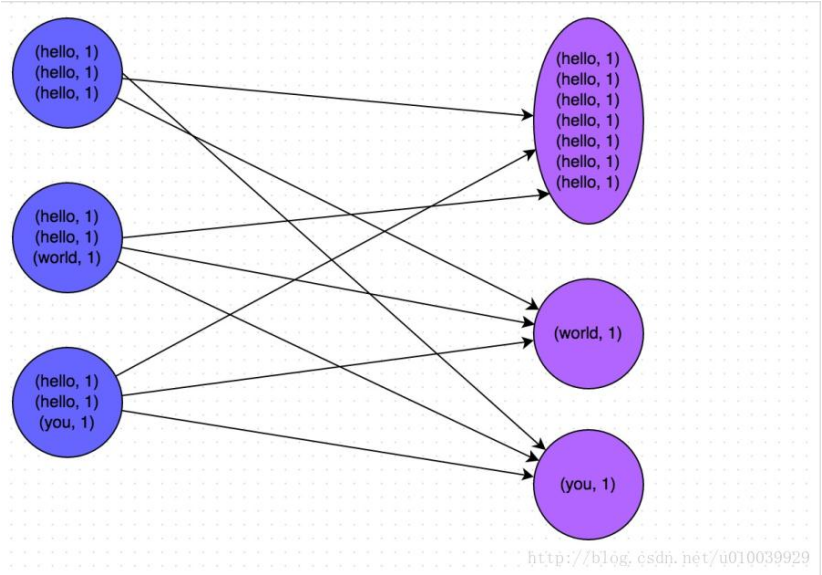
1、数据倾斜产生的原因

我们以 Spark 和 Hive 的使用场景为例。他们在做数据运算的时候会设计到，count distinct、group by、join 等操作，这些都会触发 Shuffle 动作，一旦触发，所有相同 key 的值就会拉到一个或几个节点上，就容易发生单点问题。

2、万恶的 shuffle

Shuffle 是一个能产生奇迹的地方，不管是在 Spark 还是 Hadoop 中，它们的作用都是至关重要的。那么在 Shuffle 如何产生了数据倾斜？

Hadoop 和 Spark 在 Shuffle 过程中产生数据倾斜的原理基本类似。如下图。




大部分数据倾斜的原理就类似于下图，很明了，因为数据分布不均匀，导致大量的数据分配到了一个节点。

3、从业务计角度来理解数据倾斜

数据往往和业务是强相关的，业务的场景直接影响到了数据的分布。再举一个例子，比如就说订单场景吧，我们在某一天在北京和上海两个城市多了强力的推广，结果可能是这两个城市的订单量增长了 10000%，其余城市的数据量不变。然后我们要统计不同城市的订单情况，这样，一做 group 操作，可能直接就数据倾斜了。

如何解决

一、几个思路

解决数据倾斜有这几个思路：

1.业务逻辑，我们从业务逻辑的层面上来优化数据倾斜，比如上面的例子，我们单独对这两个城市来做 count，最后和其它城市做整合。

2.程序层面，比如说在 Hive 中，经常遇到 count（distinct）操作，这样会导致最终只有一个 reduce，我们可以先 group 再在外面包一层 count，就可以了。

3.调参方面，Hadoop 和 Spark 都自带了很多的参数和机制来调节数据倾斜，合理利用它们就能解决大部分问题。

二、从业务和数据上解决数据倾斜

很多数据倾斜都是在数据的使用上造成的。我们举几个场景，并分别给出它们的解决方案。

数据分布不均匀：

前面提到的“从数据角度来理解数据倾斜”和“从业务计角度来理解数据倾斜”中的例子，其实都是数据分布不均匀的类型，这种情况和计算平台无关，我们能通过设计的角度尝试解决它。

有损的方法：

找到异常数据，比如 ip 为 0 的数据，过滤掉

无损的方法：

对分布不均匀的数据，单独计算

先对 key 做一层 hash，先将数据打散让它的并行度变大，再汇集

•数据预处理

三、Hadoop 平台的优化方法

列出来一些方法和思路，具体的参数和用法在官网看就行了。

1.mapjoin 方式

2.count distinct 的操作，先转成 group，再 count

3.hive.groupby.skewindata=true

4.left semi jioin 的使用

5.设置 map 端输出、中间结果压缩。（不完全是解决数据倾斜的问题，但是减少了 IO 读写和网络传输，能提高很多效率）

四、Spark 平台的优化方法

列出来一些方法和思路，具体的参数和用法在官网看就行了。

1.mapjoin 方式

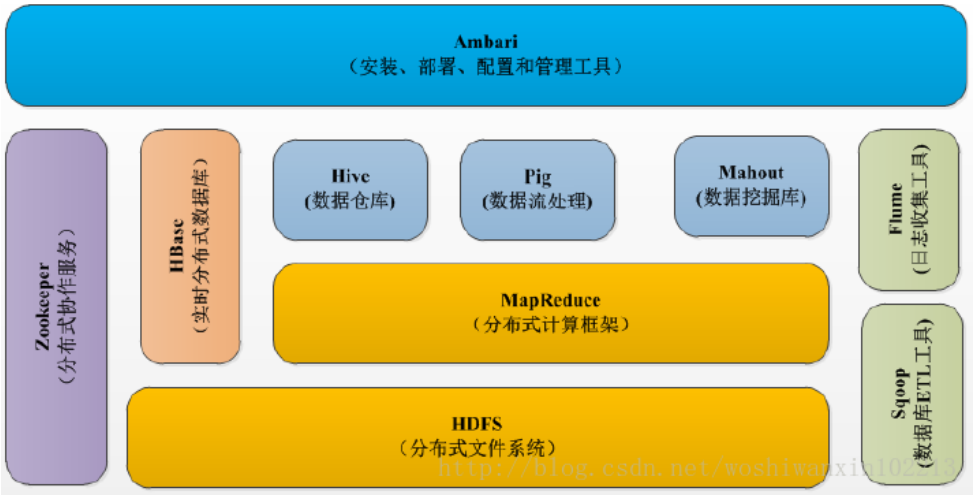
2.设置 rdd 压缩

3.合理设置 driver 的内存

4.Spark Sql 中的优化和 Hive 类似，可以参考 Hive

3、 Hadoop 生态圈、组件、工作原理、五个启动进程、namenode、datanode

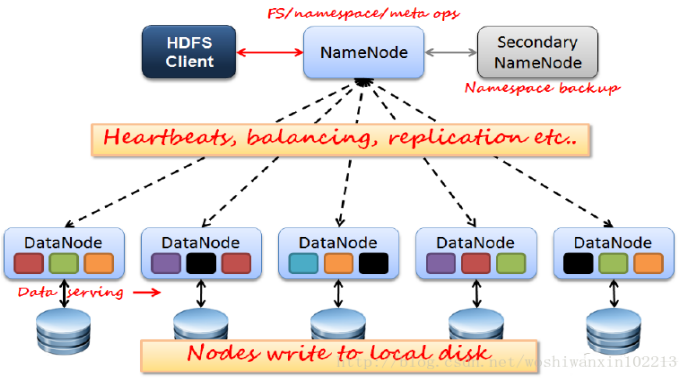
生态圈系统：https://www.cnblogs.com/dorothychai/p/4238795.html




ETL，是英文 Extract-Transform-Load 的缩写，用来描述将数据从来源端经过抽取（extract）、转换（transform）、加载（load）至目的端的过程。

2、HDFS（Hadoop 分布式文件系统）

是 Hadoop 体系中数据存储管理的基础。它是一个高度容错的系统，能检测和应对硬件故障，用于在低成本的通用硬件上运行。HDFS 简化了文件的一致性模型，通过流式数据访问，提供高吞吐量应用程序数据访问功能，适合带有大型数据集的应用程序。




**Client**：切分文件；访问 HDFS；与 NameNode 交互，获取文件位置信息；与 DataNode 交互，读取和写入数据。

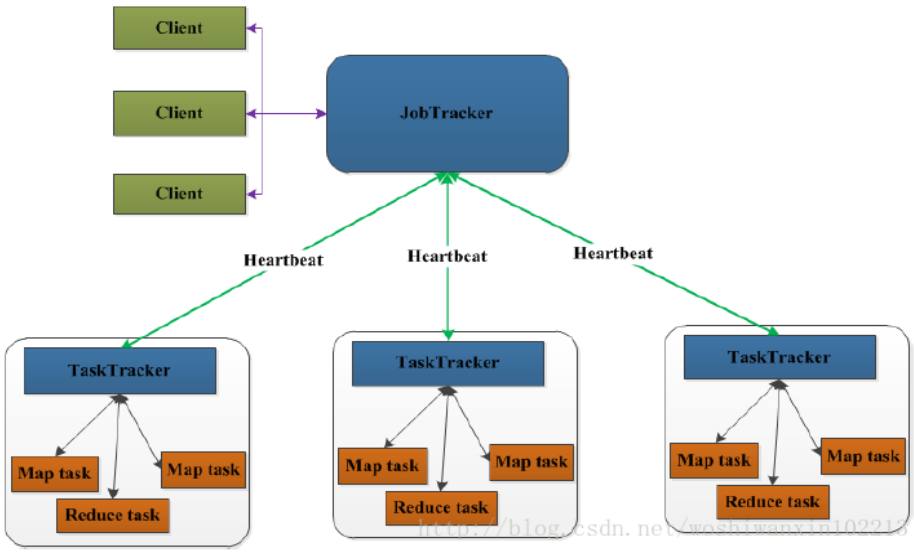
**NameNode**：Master 节点，在 hadoop1.X 中只有一个，管理 HDFS 的名称空间和数据块映射信息，配置副本策略，处理客户端请求。

**DataNode**：Slave 节点，存储实际的数据，汇报存储信息给 NameNode。

**Secondary NameNode**：辅助 NameNode，分担其工作量；定期合并 fsimage 和 fsedits，推送给 NameNode；紧急情况下，可辅助恢复 NameNode，但 Secondary NameNode 并非 NameNode 的热备。

## Mapreduce（分布式计算框架）

MapReduce 是一种计算模型，用以进行大数据量的计算。其中 Map 对数据集上的独立元素进行指定的操作，生成键-值对形式中间结果。Reduce 则对中间结果中相同“键”的所有“值”进行规约，以得到最终结果。MapReduce 这样的功能划分，非常适合在大量计算机组成的分布式并行环境里进行数据处理。




**JobTracker**：Master 节点，只有一个，管理所有作业，作业/任务的监控、错误处理等；将任务分解成一系列任务，并分派给 TaskTracker。

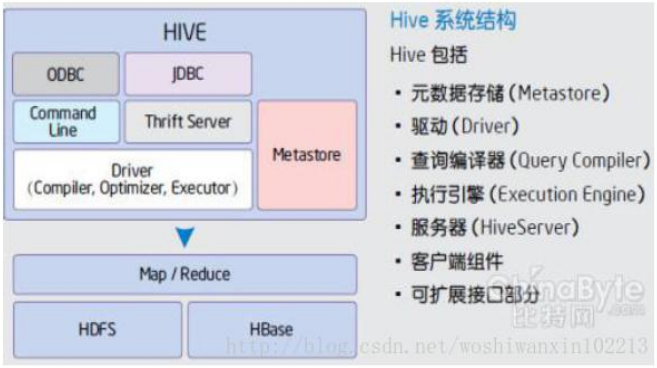
**TaskTracker**：Slave 节点，运行 Map Task 和 Reduce Task；并与 JobTracker 交互，汇报任务状态。

**Map Task**：解析每条数据记录，传递给用户编写的 map(),并执行，将输出结果写入本地磁盘(如果为 map-only 作业，直接写入 HDFS)。
**Reducer Task**：从 Map Task 的执行结果中，远程读取输入数据，对数据进行排序，将数据按照分组传递给用户编写的 reduce 函数执行。

Hive（基于 Hadoop 的数据仓库）

Hive 定义了一种类似 SQL 的查询语言(HQL),将 SQL 转化为 MapReduce 任务在 Hadoop 上执行。

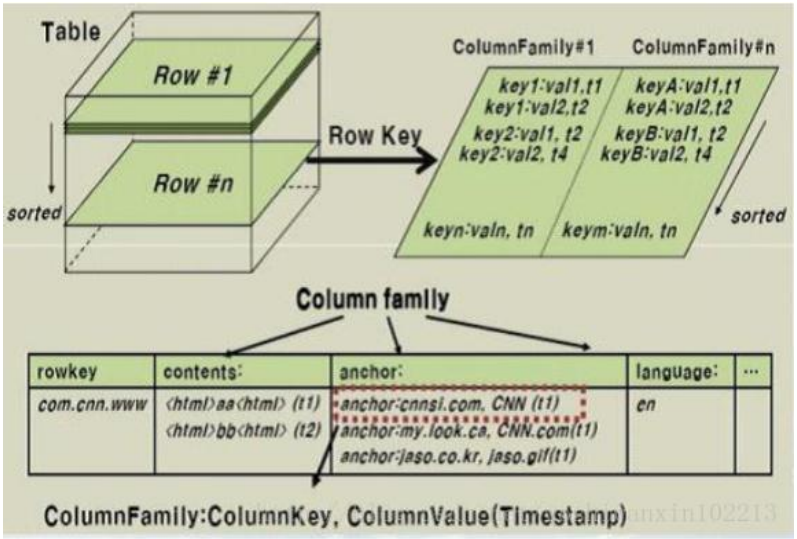
通常用于离线分析。




Hbase（分布式列存数据库）

HBase 是一个针对结构化数据的可伸缩、高可靠、高性能、分布式和面向列的动态模式数据库。和传统关系数据库不同，HBase 采用了 BigTable 的数据模型：增强的稀疏排序映射表（Key/Value），其中，键由行关键字、列关键字和时间戳构成。HBase 提供了对大规模数据的随机、实时读写访问，同时，HBase 中保存的数据可以使用 MapReduce 来处理，它将数据存储和并行计算完美地结合在一起。

数据模型：Schema-->Table-->Column Family-->Column-->RowKey-->TimeStamp-->Value



Zookeeper 解决分布式环境下的数据管理问题：统一命名，状态同步，集群管理，配置同步等。

Sqoop 是 SQL-to-Hadoop 的缩写，主要用于传统数据库和 Hadoop 之间传输数据。数据的导入和导出本质上是 Mapreduce 程序，充分利用了 MR 的并行化和容错性。

Pig（基于 Hadoop 的数据流系统）

由 yahoo!开源，设计动机是提供一种基于 MapReduce 的 ad-hoc(计算在 query 时发生)数据分析工具。定义了一种数据流语言—Pig Latin，将脚本转换为 MapReduce 任务在 Hadoop 上执行。

Mahout（数据挖掘算法库）

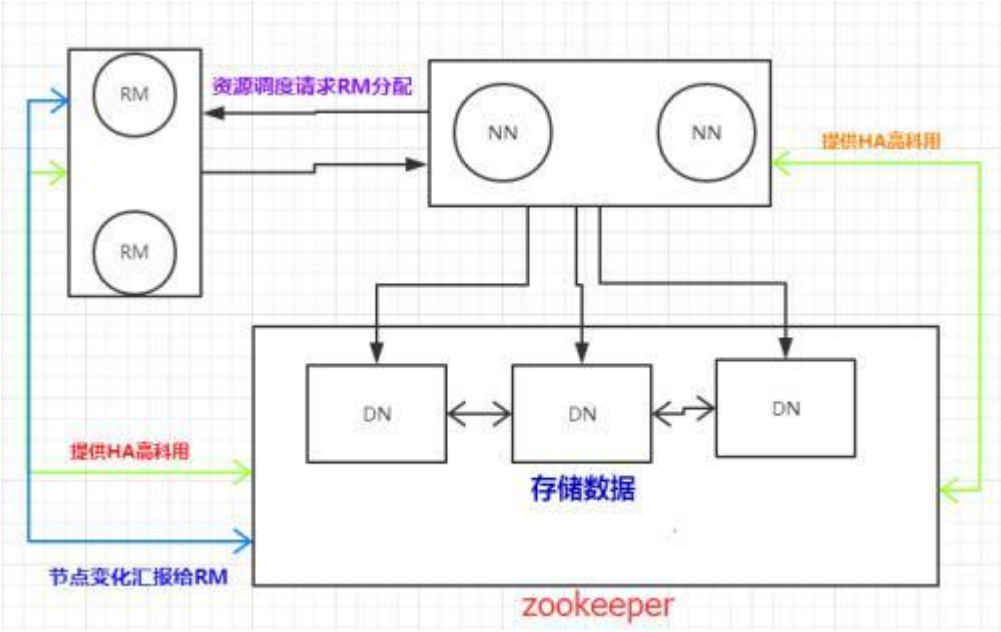
Mahout 的主要目标是创建一些可扩展的机器学习领域经典算法的实现，旨在帮助开发人员更加方便快捷地创建智能应用程序。Mahout 现在已经包含了聚类、分类、推荐引擎（协同过滤）和频繁集挖掘等广泛使用的数据挖掘方法。除了算法，Mahout 还包含数据的输入/输出工具、与其他存储系统（如数据库、MongoDB 或 Cassandra）集成等数据挖掘支持架构。

Flume（日志收集工具）

Cloudera 开源的日志收集系统，具有分布式、高可靠、高容错、易于定制和扩展的特点。它将数据从产生、传输、处理并最终写入目标的路径的过程抽象为数据流，在具体的数据流中，数据源支持在 Flume 中定制数据发送方，从而支持收集各种不同协议数据。同时，Flume 数据流提供对日志数据进行简单处理的能力，如过滤、格式转换等。此外，Flume 还具有能够将日志写往各种数据目标（可定制）的能力。总的来说，Flume 是一个可扩展、适合复杂环境的海量日志收集系统。

Hadoop 2.0 的主要改进有：
1、通过 YARN 实现资源的调度与管理，从而使 Hadoop 2.0 可以运行更多种类的计算框架，如 Spark 等。
2、实现了 NameNode 的 HA 方案，即同时有 2 个 NameNode（一个 Active 另一个 Standby），如果 ActiveNameNode 挂掉的话，另一个 NameNode 会转入 Active 状态提供服务，保证了整个集群的高可用。
3、实现了 HDFS federation，由于元数据放在 NameNode 的内存当中，内存限制了整个集群的规模，通过 HDFS federation 使多个 NameNode 组成一个联邦共同管理 DataNode，这样就可以扩大集群规模。
4、Hadoop RPC 序列化扩展性好，通过将数据类型模块从 RPC 中独立出来，成为一个独立的可插拔模块。

五个启动进程：




1,NameNode:

相当于一个领导者，负责调度 比如你需要存一个 640m 的文件

如果按照 64m 分块 那么 namenode 就会把这 10 个块（这里不考虑副本）

分配到集群中的 datanode 上 并记录对于关系 。

当你要下载这个文件的时候 namenode 就知道在那些节点上给你取这些数据了。

它主要维护两个 map 一个是文件到块的对应关系 一个是块到节点的对应关系

2，secondarynamenode：

它是 namenode 的一个快照，

会根据 configuration 中设置的值来决定多少时间周期性的去 cp 一下 namenode，记录 namenode 中的 metadata 及其它数据

3,NodeManager：

它是 YARN 中每个节点上的代理，

它管理 Hadoop 集群中单个计算节点，包括与 ResourceManger

保持通信，监督 Container 的生命周期管理，监控每个 Container 的

资源使用（内存、CPU 等）情况，追踪节点健康状况，管理日志和

不同应用程序用到的附属服务（auxiliary service）。

4,DataNode:

a,DataNode 的需要完成的首要任务是 K-V 存储

b,完成和 namenode 通信 ，这个通过 IPC 心跳连接实现。

此外还有和客户端 其它 datanode 之前的信息交换

c,完成和客户端还有其它节点的大规模通信，这个需要直接通过 socket 协议实现。

5,ResourceManager:

ResourceManager (RM) 是管理所有可用的集群资源并协助管理运行

在 YARN 上的分布式应用的主要组件。RM 与每个节点的 NodeManagers (NMs)

和每个应用的 ApplicationMasters (AMs)一起工作。

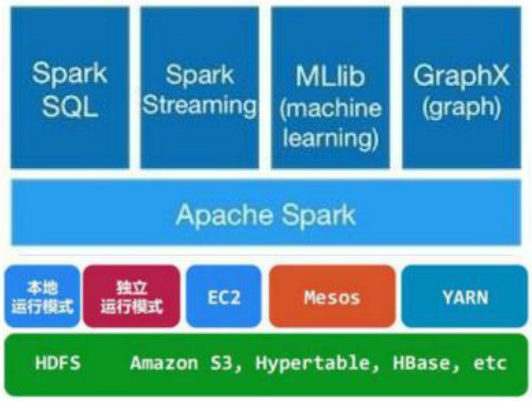
a.NodeManagers 遵循来自 ResourceManager 的指令来管理单一节点上的可用资源。

b.ApplicationMasters 负责与 ResourceManager 协商资源并
与 NodeManagers 合作启动容器

4、 Hive 工作原理、hive 和 spark 区别

Spark 基于内存迭代，快

Hive：hive 是基于 Hadoop 的一个数据仓库工具，可以将结构化的数据文件映射为一张数据库表，并提供简单的 sql 查询功能，可以将 sql 语句转换为 MapReduce 任务进行运行。可以通过类 SQL 语句快速实现简单的 MapReduce 统计。




Spark Core：包含 Spark 的基本功能；尤其是定义 RDD 的 API、操作以及这两者上的动作。其他 Spark 的库都是构建在 RDD 和 Spark Core 之上的

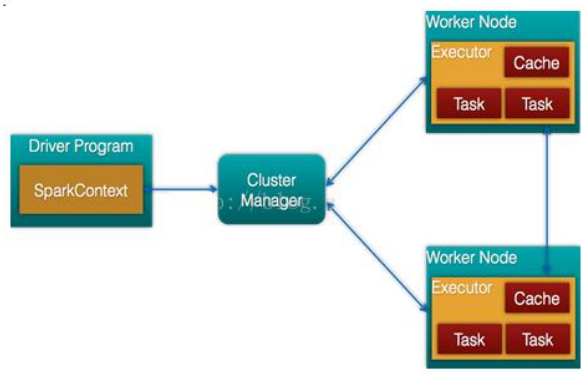
Spark SQL：提供通过 Apache Hive 的 SQL 变体 Hive 查询语言（HiveQL）与 Spark 进行交互的 API。每个数据库表被当做一个 RDD，Spark SQL 查询被转换为 Spark 操作。

Spark Streaming :对实时数据流进行处理和控制。Spark Streaming 允许程序能够像普通 RDD 一样处理实时数据

MLlib：一个常用机器学习算法库，算法被实现为对 RDD 的 Spark 操作。这个库包含可扩展的学习算法，比如分类、回归等需要对大量数据集进行迭代的操作。

GraphX：控制图、并行图操作和计算的一组算法和工具的集合。GraphX 扩展了 RDD API，包含控制图、创建子图、访问路径上所有顶点的操作

Spark 架构的组成图如下：



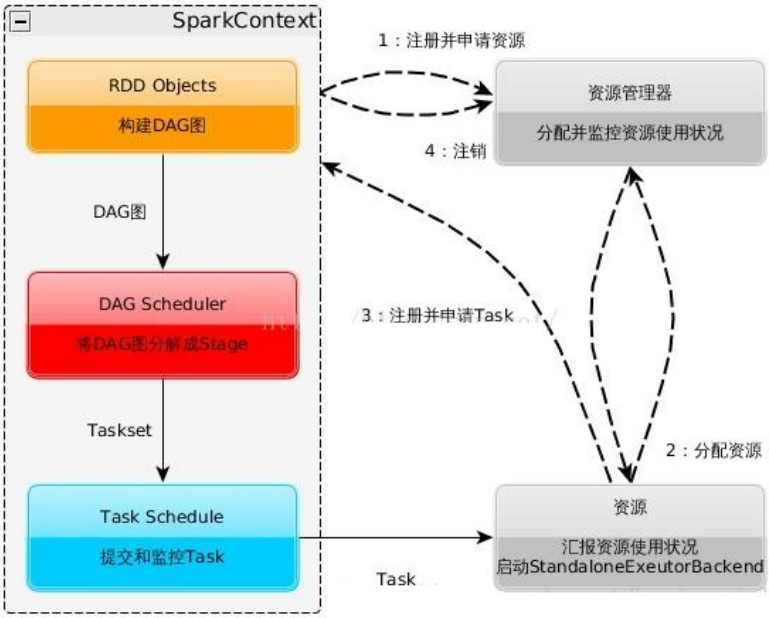
Cluster Manager：在 standalone 模式中即为 Master 主节点，控制整个集群，监控 worker。在 YARN 模式中为资源管理器
Worker 节点：从节点，负责控制计算节点，启动 Executor 或者 Driver。
Driver： 运行 Application 的 main()函数
Executor：执行器，是为某个 Application 运行在 worker node 上的一个进程

Spark 与 hadoop:

---

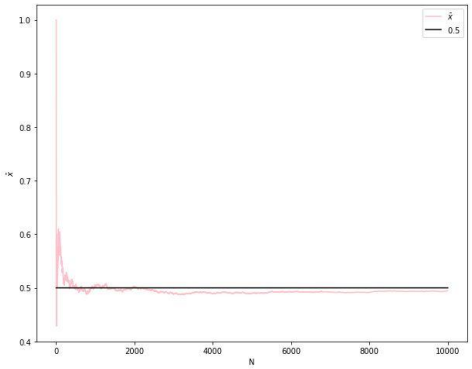
Hadoop 有两个核心模块，分布式存储模块 HDFS 和分布式计算模块 Mapreduce
spark 本身并没有提供分布式文件系统，因此 spark 的分析大多依赖于 Hadoop 的分布式文件系统 HDFS
Hadoop 的 Mapreduce 与 spark 都可以进行数据计算，而相比于 Mapreduce，spark 的速度更快并且提供的功能更加丰富
关系图如下：


spark 运行流程图如下：

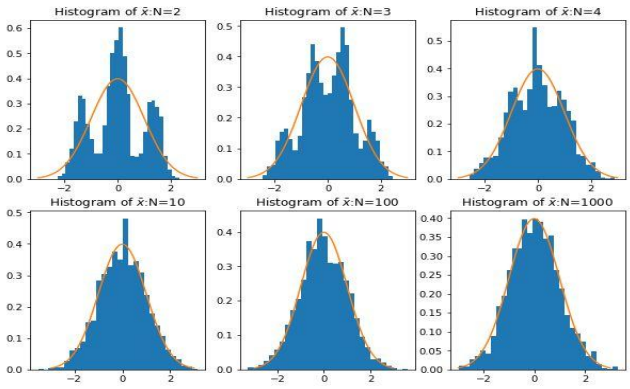



构建 Spark Application 的运行环境，启动 SparkContext
SparkContext 向资源管理器（可以是 Standalone，Mesos，Yarn）申请运行 Executor 资源，并启动 StandaloneExecutorbackend，
Executor 向 SparkContext 申请 Task
SparkContext 将应用程序分发给 Executor
SparkContext 构建成 DAG 图，将 DAG 图分解成 Stage、将 Taskset 发送给 Task Scheduler，最后由 Task Scheduler 将 Task 发送给 Executor 运行
Task 在 Executor 上运行，运行完释放所有资源

5、 Map 原理　键唯一
6、 显著性检验
显著性检验（significance test）就是事先对总体（随机变量）的参数或总体分布形式做出一个假设，然后利用样本信息来判断这个假设（备择假设）是否合理，即判断总体的真实情况与原假设是否有显著性差异。
7、 大数定律
大数定律说的是随机现象平均结果稳定性。
中心极限定理论证随机变量的和的极限分布是正态分布。
大数定律讲的是样本均值收敛到总体均值（就是期望），像这个图一样：




而中心极限定理告诉我们，当样本量足够大时，样本均值的分布慢慢变成正态分布:




8、 大数据是什么

巨量数据集合，指无法在一定时间范围内用常规软件工具进行捕捉、管理和处理的数据集合，是需要新处理模式才能具有更强的决策力、洞察发现力和流程优化能力的海量、高增长率和多样化的信息资产。

9、 UDP 传输过程

10、 TCP、UDP 区别、TCP 三次握手四次挥手

小结 TCP 与 UDP 的区别：

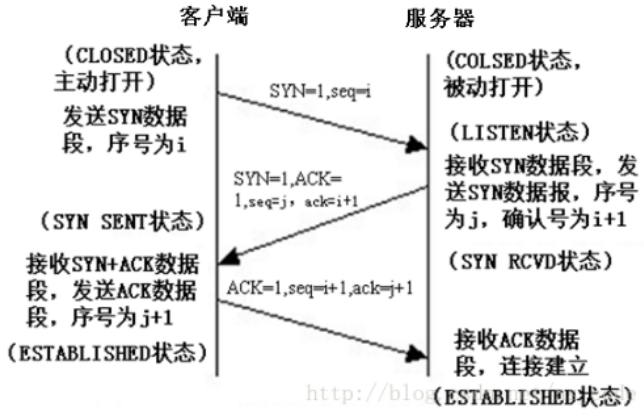
1.基于连接与无连接；

2.对系统资源的要求（TCP 较多，UDP 少）；

3.UDP 程序结构较简单；

4.流模式与数据报模式 ；

5.TCP 保证数据正确性，UDP 可能丢包，TCP 保证数据顺序，UDP 不保证。




11、 保持积极的态度

1、 多线程

2、 分布式领域 CAP 理论，

Consistency(一致性)，数据一致更新，所有数据变动都是同步的

Availability(可用性)，好的响应性能

Partition tolerance(分区容忍性) 可靠性

定理：任何分布式系统只可同时满足二点，没法三者兼顾。