第一个课件

1. 了解大数据的4个V各是什么含义？

Volume：数据体量巨大（从TB级别，跃升到PB级别）；

Variety：数据类型繁多（网络日志、视频、图片、地理位置信息等等）；

Velocity：处理速度快（1秒定律，有别于传统的数据挖掘技术）；

Value：价值密度低。

1. 超大规模计算适合什么平台？

第二个课件

1. Memcached内存替换策略，删除回收为什么是惰性回收？

3.1 内存替换策略

1）优先使用已超时的记录的空间。

2）如果还存在追加新记录时空间不足的情况， 使用最近最少使用（LRU）机制来替换已有缓存内容。 （引用计数非零则不替换）

3.2 惰性回收

记录超时不会释放已分配的内存，只是客户端无法再看见该记录，其存储空间此时可重复使用。内部不实时监视记录是否过期，而是在get时查看记录的时间戳是否过期。

原因：这样可以不在过期监视上耗费CPU时间。

1. DynamoDB比memcached多了多副本功能，怎么解决多副本临时失效的问题？

DynamoDB节点临时失效处理：hinted handoff（暗示接力）

Hinted Handoff技术：为了保证每次都能写到W个副本，读到R个副本，我们每次读和写都是发送给N个节点。如果这N个节点有节点失效，那么往后继续找一个不同的节点，暂时的代替失效的节点，当该后续节点定期监测到故障节点恢复，则将暂时代为保管的数据写回复活节点。

例：N=3，某数据的preference list是节点A、B、C。若A节点失效，则对该数据的写请求将发送到节点B、C、D上。D暂时取代A的角色，那些原本应该写到A上的数据存放在D中的一个特定的文件夹中（意味着这些数据不是D本该拥有的，而是别的节点的）。D上会启动一个线程定期检查A的状态，当发现A恢复后，就将D上存放的这些A的数据写回到A。

该策略保证了节点失效时系统的高可用和数据持久性。

1. DynamoDB一致性hash模型：R+W>N. vector-clock更新算法是什么？带来的问题和解决方案（怎么解决无限增长）

5.1 一致性hash模型：改进

原一致性hash模型采用设置虚拟节点，主要缺点：虚拟节点位置随机大小随机，如果有新节点加入，需要扫描所有节点上的所有数据对象，判断全部数据对象是否需要迁移，这种全局的扫描造成很大的开销。

改进：固定所有虚拟节点的大小和位置，只改变虚拟节点和节点的对应关系。

将整个地址空间平均分成Q个虚拟节点，每个物理节点（假设有S个）分配Q/S个虚拟节点。当有节点加入时，从现有节点每个拿出等量的虚拟节点分给新节点；当有节点离开时，将此节点的所有虚拟节点平均分配给余下的节点，保证系统中每个节点始终都有Q/S个虚拟节点。

固定所有虚拟节点的大小和位置的策略下，为实现结点加入和离开系统时影响较小，要求Q>>S。

如果业务快速发展，使得主机不断增加，从而导致Q不再满足Q>>S，则该策略将不断的退化。(实际应用中还有存储备份的问题，假设每个数据存储N个备份 则要满足Q>>S\*N)。

5.2 NWR理论（读写协议）（WernerVogels在讲“EventuallyConsistent”时提出）。设一个存储系统有如下属性：

N=每个数据的副本数

W=每次写操作时，必须同步确认写成功的副本数

R=每次读操作时，需要读取的副本数

则当W+R>N时，该存储系统可以提供强一致性。

强一致性等价于R中至少包含一个最新的副本，即(R-(N-W))>0，即W+R>N。

5.3 Vector Clock：每个机器结点维护一个逻辑时钟的向量VC。向量VC有如下属性：

（1）VCi[i] 是到目前为止机器i上发生的事件的个数；

（2）VCi[k] 是机器i知道的机器k发生的事件的个数(即机器i对机器j的知识)。

Vector Clock的更新算法：

（1）机器i本地发生一个事件时将VCi[i]加1；

（2）机器i给机器j发送消息m时，将整个VCi存在消息内；

（3）机器j收到消息m时，VCj[k]=max(VCj[k],VCi[k])，同时VCj[j]+1。

Vector Clock是一种维护因果关系(causality)的手段，Vector Clock在机器之间传递，从而给对方传递自己知道的其他机器的知识。

* 1. Vector Clock的无限增长问题

现实生活中，如果有很多的决策者，相当于有很多的客户端，整个向量时钟的长度就无限制增长了，这对于存储系统来说，需要想办法解决。

* 采用服务器向量：不要用client来标识向量空间，用server来标识向量空间。因为server的数量是可控的，向量标签不再是客户端，而是用server标识。

问题：会因为网络传输延时而丢失数据。

如：Dave收到两个消息

Tue X:2,Y:1  with Ben

Thu X:1,Y:1   from Cathy （该消息比较新）

发现后者是前者的祖先，所以抛弃祖先，选择Ben的消息，这样就把Cathy给他的消息给丢了。

* 向量时钟的剪枝：

Riak系统用四个参数来避免向量时钟空间的无限增长：

small\_vclock

big\_vclock

young\_vclock

old\_vclock

small\_vclock和big\_vclock参数标识向量时钟的长度，如果长度小于small\_vclock就不会被剪枝掉，如果长度大于big\_vclock就会被剪枝掉。

young\_vclock和old\_vclock参数标识存储这个向量时钟时的时间戳，剪枝策略同理，大于old\_vclock的才会被剪枝掉。

向量剪枝尽量只丢掉一些向量时钟的信息，而不是丢掉实实在在的数据。

但是有一种情况会有问题：一个客户端保持了一个很久之前的向量时钟，然后继承这个向量时钟并提交了一个数据，此时会产生冲突（因为服务器已经没有这个很久之前的向量时钟信息了，可能已经被剪枝掉了），所以客户端提交的此次数据，在服务端无法找到一个祖先。此时Riak会创建一个sibling。

所以，剪枝策略是一个tradeoff权衡，一方面是无限增长的向量时钟的空间，另一方面是偶尔的会有“false merge”，产生兄弟数据，但不会丢失数据。

从这个意义上看，防止向量时钟空间的无限增长，剪枝策略优于用server标识向量时钟的策略

1. DynamoDB的merkleTree怎么实现层次化？工作原理？典型应用？哪种性能保证了可以被广泛应用？

6.1 Merkle tree工作原理？实现层次化方法？

每个叶子节点对应一个数据项，并记录其hash值 ；

每个非叶子节点记录其所有子节点的hash值。

Dynamo为每一个分片维护一个Merkle Tree，需要比较分片是否相同时，自根向下的比较两个Merkle Tree的对应节点，可以快速发现并定位差异所在。

* 1. 典型应用

文件校验（BitCommit， BitTorrent种子）

副本同步（DynamoDB）

可信计算

区块链交易认证

6.4 优点，被广泛利用的保证

Merkle Tree的主要优点是树的每个分支可以独立地检查，而不需要下载整个树或整个数据集。

此外，MerkleTree有助于减少为检查副本间不一致而传输的数据的大小。

Merkle tree可以通过部分hash就能校验整个文件的完整性

例：如果两树的根哈希值相等，且树的叶节点值也相等，那么节点不需要同步。如果不一致,通过这种二叉树的结构可以在log(N)的复杂度快速定位到出错的数据块。

第三个课件（一般性了解）

1. 淘宝升级前有哪些存储方面的进展？如存储区域网的特点，通讯模式

采用NetApp 的 NAS(NetworkAttached Storage：网络附属存储) 存储作为数据库的存储设备，加上 Oracle实时应用集群 (RAC，Real Application Clusters，)来实现负载均衡。

隐患：NAS的NFS(Network File System)协议传输的延迟很严重；

SAN （Storage Area Network，存储区域网）（存储扩容）（光纤通讯）

SAN是一个高速的子网，子网中的设备可以从主网卸载流量。通常SAN由RAID阵列连接光纤通道组成，SAN和服务器和客户机的数据通信通过SCSI命令而非TCP/IP，数据处理是“块级”（block level）。

SAN是一种高速的专用网络，它建立起服务器、磁盘阵列、磁带库之间的一种直接连接。它如同扩展的存储器总线，将专用的集线器、交换器以及网关或桥路互相连接在一起。

高性能的光纤通道交换机和光纤通道网络协议是SAN的关键。

SAN利用光纤通道协议（FCP，Fibre Channel Protocol ）上加载SCSI协议来达到可靠的块级数据传输。

在一些关键应用中，传输块级数据要求必须使用SAN，尤其是多个服务器共同向大型存储设备进行读取。

SAN的优点和问题

1）由于在数据传输时被分成小段，使SAN对服务器处理的依赖较少，可以有效地传送爆发性的块数据，性能及可靠性就得到了充分的发挥。

2）通过城域网（MAN， Metropolitan Area Network ），SAN可以实现远程灾难恢复。使用E3信道SAN可以在不降低性能的同时将部件间的距离增加至150km。

3）SAN的管理集中而且高效，可以在线添加/删除设备、动态调整存储网络以及将异构设备统一成存储池等。

问题：光纤通道设备的互操作性差，采用光纤通道技术的系统造价非常昂贵。

NAS vs. SAN

1）NAS是一台特殊的含有大量硬盘空间的计算机，连接在以太网上，其它计算机通过网络映射硬盘使用空间。

SAN是一种容易扩容的光纤的磁盘阵列机，是多台服务器共享使用多台阵列机，可以安装各种软件，可跨平台。

2）SAN是光纤协议，NAS是TCP/IP协议。NAS是利用现有网络，SAN是在sever端再架设一个网络。

3）NAS以文件方式访问数据，而SAN以sectors方式访问数据。

SAN对于高容量块级数据传输具有明显的优势，易扩展且管理设备高效，可运行关键应用（如数据库、备份等）。

NAS更加适合文件级别的数据处理。可作为日常办公中需要经常交换小文件的存储配置（如存储网页）。

4）SAN更多的是强调范围，高效。  
 NAS则主要是强调共享。

NAS使用的文件传输协议意味着当把数据库建立在NAS上时，取得一条记录需要对整个数据文件进行传输（如果数据库不更改数据访问方式）。

第四个课件

1. GFS写数据时数据传播模式，副本间数据推送方式？

8.1 写数据的流程

1)客户机向Master节点询问哪个Chunk服务器持有当前租约，以及其它副本的位置。

如果没有Chunk持有租约，Master就选择其中一个副本建立一个租约。

2)Master将主Chunk的标识符以及其它副本（secondary副本/二级副本）的位置返回给客户机。客户机缓存这些数据以便后续操作。

此后只有在主Chunk不可用或主Chunk回复信息表明它已不再持有租约时，客户机才重新联系Master节点。

3）客户机把数据推送到所有的副本上（可按任意的顺序推送）。

Chunk服务器接收到数据并保存在它的内部LRU缓存中，直到数据被使用或过期交换出去。

4）当所有的副本都确认收到了数据后，客户机发送写请求到主Chunk服务器。

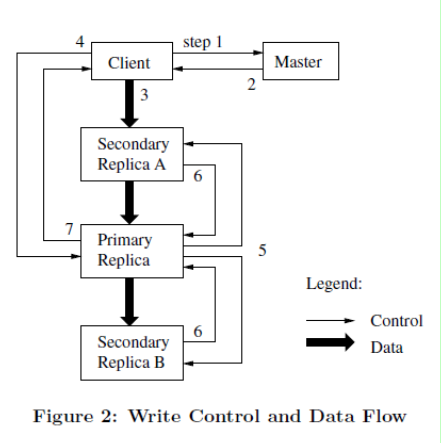
主Chunk为接收到的所有操作分配连续的序列号（这些操作可能来自不同的客户机，序列号保证了操作顺序执行），按序列号的顺序把操作应用到primary自己的本地状态中。

5）主Chunk把写请求传递到所有的二级副本。每个二级副本依照主Chunk分配的序列号以相同的顺序执行这些操作。

6）所有的二级副本回复主Chunk“它们已经完成了操作”。

7）主Chunk服务器回复客户机（任何副本产生的任何错误都会返回给客户机）。

出现错误时，写操作可能在主Chunk和一些二级副本执行成功。（如果主Chunk上操作失败，操作不会被分配序列号，也不会传递）客户端请求被确认为失败，被修改的region处于不一致状态。客户机代码通过重复执行失败的操作来处理这样的错误。在从头开始重复执行之前，客户机会先尝试几次步骤（3）~（7）。



1. GFS采用单一master形式，面向不稳定集群，怎么选择哪些数据不持久化，哪些数据持久化？
2. GFS有哪些特殊的操作（原子性操作）

原子性的记录追加操作，保证多个客户端能同时进行追加操作，不需要额外的同步操作来保证数据的一致性。

记录追加：GFS提供的特殊的修改操作，一种原子性的数据追加操作，除了在主Chunk有些额外的控制逻辑，遵循一般的写操作控制流程：

1）客户机把数据推送给文件最后一个Chunk的所有副本。

2）发送请求给主Chunk。

3）主Chunk检查这次记录追加操作是否会使Chunk超过最大尺寸（64MB）。如果超过则主Chunk先将当前Chunk填充到最大尺寸，之后通知所有二级副本做同样的操作，然后回复客户机要求其对下一个Chunk重新进行记录追加操作。

记录追加：客户机只需指定要写入的数据，GFS来实现至少有一次原子的写入操作成功执行（写入一个顺序的byte流），写入的数据追加到GFS指定的偏移位置上，之后GFS返回这个偏移量给客户机。

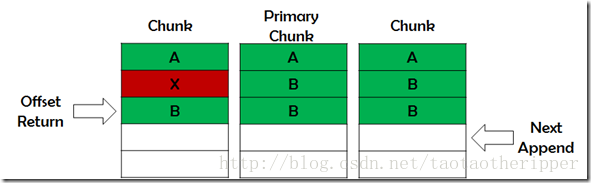
如果记录追加操作在任何一个副本上失败了，客户端会重试追加操作。

↓重新记录追加

一个Chunk的不同副本可能包含不同的数据（重复包含一个记录全部或者部分的数据）。

GFS并不保证Chunk的所有副本在字节级别是完全一致的，它只保证数据作为一个整体原子的被至少写入一次。

失败的记录追加操作可能导致Chunk间字节级别不一致，但当最终追加成功后，所有副本在返回的偏移位置一致已定义，之后的追加操作不受影响。



原子性？

当用O\_APPEND模式打开文件后，若用 lseek移动文件指针到其他的位置，然后再用write写，此时数据写到末端还是lseek移动到的位置？

↓

答案：在末端。

因为 O\_APPEND打开后，是一个原子操作：移动到末端，写数据。

记录追加O\_APPEND打开文件的作用：提供了无锁的文件追加方式，中间的插入是无效的。

第五个课件

1. Bigtable基本设计思想。Map关键字是哪三个？map怎么用三层寻址的方式实现的？

11.1 基本设计思想（数据模型）

数据访问依据下标进行：

数据的下标 = 行和列的名字（名字可是任意的字符串）。

存储的数据都视为字符串，Bigtable本身不去解析这些字符串。

↓

客户程序可以把各种结构化或者半结构化的数据串行化（序列化）到这些字符串里。

**设计思想**

* 简单数据模型
* 稀疏、分布式、多维度排序Map（行关键字+列关键字+时间戳，不解析）
* 列族
* Tablet、SStable、数据块
* 行关键字下的原子性操作

**体系架构**

* 分布式锁服务组件Chubby（存储BigTable的模式信息、自引导指令）
* Master（负载均衡、修改模式）

11.2 Map 关键字：Map的索引=行关键字+列关键字+时间戳。

11.3 三层寻址（Root Tablet/MetaData Tablet/User Tablet）；（运行管理下的）

Tablet层次结构

一个存储在Chubby中的文件，包含Root Tablet的位置信息。

第一层：Root Tablet，包含了一个特殊的METADATA表，记录METADATA里所有的Tablet的位置信息。

第二层：METADATA表，其中的每个Tablet包含了一个用户Tablet的集合。

第三层：用户表tablet。

Root Tablet：

1）实际上是METADATA表的第一个Tablet；

2）永远不会被分割 ，从而保证Tablet的位置信息存储结构不会超过三层。

METADATA表中，每个Tablet的位置信息都存放在一个行关键字下

行关键字=Tablet所在的表的标识符

⊕Tablet的最后一行编码

METADATA的每一行存储约1KB的内存数据。

METADATA表还存储次级信息(secondary information)，主要包括每个Tablet的事件日志（例如一个服务器何时开始为该Tablet提供服务）。

↓

有助于排查错误和性能分析。

客户端可能未缓存某个Tablet的地址信息（例如缓存为空），或发现缓存的地址信息不正确（过期）。

↓

客户程序在树状的存储结构中递归的查询Tablet位置信息。

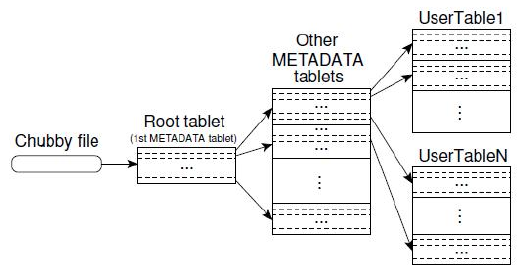
情形1：寻址算法通过三次网络来回通信寻址，其中包括一次Chubby读操作。

情形2：寻址算法可能需要最多６次网络来回通信更新数据（只有在缓存中没有查到数据的时候才能发现数据过期，三次通信发现缓存过期，另外三次更新缓存数据）。

1）发现region缓存失效；

2）本地取meta表的位置缓存，再次发现失效；

3）取ROOT的本地缓存，结论是缓存过期。



1. Paxos基本思想，CAP定理，解读CAP的两种典型应用场景（金融，互联网）怎么应对CAP定理。

12.1 paxos基本思想

Paxos算法目的：解决分布式环境下一致性的问题。

多个节点并发操纵数据，如何保证在读写过程中数据的一致性，要求解决方案能适应分布式环境下的不可靠性 。

Paxos算法通过多个监督者来增强可靠性，基本思想：

1）通过监督者们的投票来表决数据的状态变化，

2）保证所有对数据的访问都遵从这种表决。

算法分两个阶段（发起者和监督者之间的两轮通讯）

第一阶段准备（prepare）

监督者们接收请求、

产生判断、

返回承诺；

发起者依据返回的承诺决定是否继续更新数据而进入下一阶段的请求；

第二阶段接受（accept）

监督者们接受更新请求并完成多副本（多数派）更新。

法定集合性质：Paxos有效的基本保障。（paxos基于的原理）

法定集合性质：任意两个法定集合，必定存在一个公共的成员。

法定集合：将一个超过半数的集合称之为法定集合，比如数字1、2、3、4、5，共5个元素，{1,2,3}有三个元素就是法定集合。

12.2 CAP定理

CAP定理（1998年首次被提出，1999年被发表为文章，2000年Brewer在PODC大会演说时将CAP作为假设，2002年被证明为定理。CAP定理属于理论计算机科学的内容。）

分布式系统中有三种衡量特性：

一致性（Consistency）：每一个更新成功后，分布式系统中的所有节点都能读到最新的信息，即所有节点相当于访问同一份内容。这样的系统被认为是强一致性的。

可用性（Availability）：每一个请求都能得到响应。请求只需要在一定时间内返回即可，结果可以是成功或者失败，也不需要确保返回的是最新版本的信息。

分区容错性（Partition tolerance）：在网络中断、消息丢失的情况下，系统照样能够工作。网络分区是指由于某种原因网络被分成若干个孤立的区域，区域之间互不相通。

CAP定理：

一个分布式系统最多只能同时满足一致性（Consistency）、可用性（Availability）和分区容错性（Partition tolerance）这三项中的两项。

12.3 （金融，互联网）怎么应对CAP定理。

对于互联网应用，主机多，数据大，部署分散。节点故障、网络故障是常态。此时舍弃一致性C虽然会影响客户体验，但换来了对AP的保证，才能不影响用户使用流程。

对于金融领域，必须要保证C和A，因此会舍弃分区容错性P。所以金融领域的网络设备故障，可能会造成用户无法使用。



1. \*Raft算法 – 日志包含哪些东西？（经日志复制同步做一致性） -复杂流程（ 集群转换过程中如何利用过度配置）？

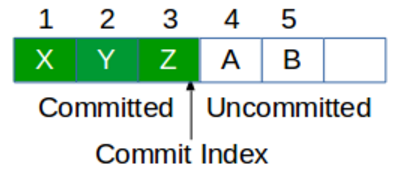
Raft将用户数据记为日志（Log），存储在日志队列中，队列里的每个日志都一个序号，该序号连续递增且不能有缺失。

每个节点都有自己的日志队列。

日志队列中有一个重要的位置叫做提交日志（Commit Index），将日志队列里的日志分为了两个部分：

已提交日志：对应已经复制到超过半数节点的数据，这些日志是可以发送给应用程序去执行的日志。

未提交日志：对应还未复制到超过半数节点的数据。



每个日志项主要记录三样内容：

（1）一条应用于状态机的指令；

（2）从leader收到本日志项时的term号；（日志项中的term号可用于判断一些日志间的不一致状态。）

（3）指明自己在日志中位置的index值。

13.3 过度配置 机群成员变更：过渡配置

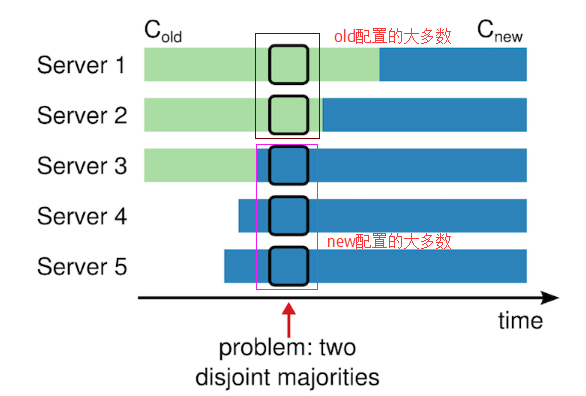
过渡配置是指集群具有 old + new 中所有机器的配置。

处于过度配置的成员，在Leader选举与提交日志时规则改变，改为分别要得到Old与New两个配置下的多数成员同意才行。

集群转换时怎么利用过度配置？

直接转换配置的不安全因素如下：需要过度配置保证其安全性

集群配置从3台机器变成5台，直接扩容转换时存在这样的一个时间点：两个不同的Leader在同一个任期里都可以被选举成功，一个是通过旧的配置，一个通过新的配置。



过度配置通过两阶段方法保证安全性：

记旧配置为C(old)，新配置为C(new)，集群先切换到一个过渡的配置，称之为共同一致配置C(old-new)。共同一致配置期间需要同时在old和new中达成多数派才行。

一旦C(new)被提交，系统就切换到新的配置上。

共同一致配置策略的好处：

（1）可以让集群在配置转换的过程中依然能响应服务器请求。

（2）允许各个服务器在不影响安全性的前提下，各自在不同的时间进行配置转换过程。

机群成员变更流程

原则：配置变更过程中，不能出现两个合法的leader。  
约定：任何一个follower在收到新的配置后，就采用新的配置确定多数派。

变更流程：  
（1）leader收到从C(old)切换到C(new)配置的请求；  
（2）创建配置日志C(old-new)，该日志需要在C(old)和C(new)中同时达成多数派；  
（3）任何一个follower收到C(old-new)配置后，采用C(old-new)策略来确定日志是否达成多数派(即使C(old-new)这条日志还没达成多数派)；  
备注：在C(old-new)没有可能成为多数派之前，如上述（1~3）步，只有可能C(old)节点成为leader。

（4）C(old-new)日志达成多数派 (commit)；  
（5）创建新配置日志C(new)，广播到所有节点；  
（6）任何一个follower收到C(new)配置后，采用C(new)策略来确定日志是否达成多数派；  
（7）C(new)配置日志commit，此时C(old-new)无法再达成多数派；  
（8）对于不在C(new)配置的节点，可以自动关闭，变更完成。  
 备注：

在（4~6）步，只有可能含有C(old-new)配置的节点成为leader，因为此时C(new) 仍未提交。  
在（7~8）步，只有可能含有C(new)配置成为leader。  
所以整个过程中永远只会有一个leader。

1. 迭代中有一些东西可以优化，如重复的？？，除此之外还有网络传播开销，计算开销，怎样针对这些优化mapreduce的计算代码，框架原理？如：如何把这个优化思想用到大规模图中？怎么设计这个算法？map函数怎么写？reduce函数怎么写？能不能用数学语言把找多效邻居的过程描述出来？
2. Storm集群有特殊节点zookeeper，用拓扑逻辑把任务分解了（分布式），如何理解这个分布式系统？1，task之间2.先后task之间流水线 这两种的运行方式。Storm通过拓扑把任务分解了，task集合要放到gpu上跑，有两个概念：1，task（逻辑层面，拓扑不变）

2.物理层面，多任务共享，会变。

16，storm高效保障机制（了解过程）

第九个课件

1. Spark的基本运行原理（部署，job执行的过程中，资源怎么分配，释放），图是怎么画的？

第十个课件（了解一下）