Ch01:

1. 什么是云计算:

云计算是一种按使用量付费的模式，这种模式提供可用的、便捷的、按需的网络访问， 进入可配置的计算资源共享池（资源包括网络，服务器，存储，应用软件，服务），这些资源能够被快速提供，只需投入很少的管理工作，或与服务供应商进行很少的交互。

云计算是一种能够将动态伸缩的虚拟化资源通过互联网以服务的方式提供给用户的计算模式。（ppt）

云计算优点：将资本投入变成可变投入，从大型规模经济中获益，无需再猜测所需容量，增加速度和灵活性，无需再为数据中心的运行和维护投入资金，业务的快速扩展和部署。

三个主要类型：IaaS，PaaS，SaaS。（从下向上）

2. 云计算的发展历程:

IaaS 发展基于虚拟化；PaaS 发展基于分布式并行计算技术。

超级计算机（单机）→集群计算（协同计算）→分布式计算（分别计算，统一合并）→网格计算（未用资源作为分布式系统的虚拟机群）→效用计算（公共服务化）→云计算（大程序拆成小程序交给多个服务器）

3. 云计算的动因:

技术成熟（资源虚拟化技术，互联网技术带宽可靠性，Web2.0）；IT 企业的成熟和计算力过剩（摩尔定律），集中大量硬件实现规模效益；社会需求的膨胀和商业规模的扩大（面向服务架构 SOA，SaaS）。

Ch02:

1. 服务的概念:

通过一系列活动满足对方需求；以用户需求满意度为核心；活动为主，实物为辅。

特点：针对个性化需求；顾客参与度提升；更大的价值体现；以顾客满意度衡量。

2. 云的技术范畴:

资源服务化；虚拟化的计算和存储资源；运行应用的平台；种类繁多的互联网应用；服务的可伸缩性、可用性和安全性。

资源虚拟化（资源的抽象化描述）、分布式并行计算系统（海量、高并发）、资源管控（分配回收策略）

3. 云服务的基本层次:

横向：公有云（Internet）|混合云（Internet 和 Intranet）|私有云（Intranet）

纵向：IaaS（基础设施层）→PaaS（平台层）→SaaS（应用层）

4. 云的特征:

硬件和软件都是资源（分工协作）；资源动态扩展配置（按需分配）；按用计费，无需管理（租用）；物理分布式，逻辑单一整体（对用户不可见）；

5. 云的优势:

优化产业布局（规模效应）；推进专业分工（针对性强）；提升资源利用率（资源分配负载）；减少初期投资（基础设施，软件，人力）；降低管理开销（系统灵活性）

Ch03:

1．IaaS 的基本功能:

a) 资源抽象：硬件虚拟化；屏蔽硬件差异；提供统一管理接口和资源池。（粒度划分管理：虚拟机→集群→虚拟数据中心→云）。

b) 资源监控: 负载管理前提; 计算单元: 监控对象和监控层次不同, 监控方式也不同。

c) 负载管理：负载均匀（可留空结点）；避免负载过高；负载对性能资源影响。

d) 数据管理：多种数据并存（数据库）；分布式存储；完整性（log），可靠性（冗余），可管理性（粗粒度逻辑简单管理）。
e) 资源部署：IaaS 可用化，虚拟化技术简化；动态资源可伸缩性；故障恢复和硬件维护。
f) 安全管理：合法性（正确的用户、程序、分配），安全（操作审查、授权、追踪机制）。
g) 计费管理：量或时间计费（监控）。

2．PaaS 的基本功能：

a) 开发平台：应用模型（语言，元数据，打包）；API 代码库（多方 API 接口）；开发测试环境（离线开发，在线上传）。
b) 运行时环境：打包→上传→配置。隔离性（业务，数据，应用，用户），可伸缩性（动态分配资源），可复用性（资源释放回收，宏观无限，微观有限）。
c) 运营环境：更新；升级（补丁）；监控；卸载；计费。

Ch04：

实验课

Ch05:

1. 虚拟化的概念：

虚拟化是表示计算机资源的抽象方法，通过虚拟化可以用与访问抽象前资源一致的方法来访问抽象后的资源。这种资源的抽象方法并不受实现、地理位置或底层资源的物理配置限制。

在使用层和实体层之间加入了一层抽象中间层，对使用层提供统一接口（隐藏细节），对实体层提供了统一管理。

2. 服务器虚拟化的特性：

建立动态、自动化虚拟 IT 环境；高性能，可扩展性，稳定性；同一物理机运行多个 VM；快速部署；满足多种需求。

3. 服务器虚拟化的关键技术：

a) 计算虚拟化：全虚拟化（VM 在 Ring0，OS 在 Ring1 使用核心指令操作 VM）；半虚拟化（VM 在 Ring0，部分虚拟化 OS 在 Ring1 使用 Hypercall 软中断操作 VM）；硬件辅助虚拟化（VM 和硬件一层，OS 在 Ring0，两者同时核心指令操作硬件）。
b) 存储虚拟化：磁盘虚拟化（虚拟磁盘映射表）；内存虚拟化（影子页表法：表中保存虚拟机和实际内存的地址映射关系，实际地址无需变动；页表写入法：创建页表时对 VM 注册，由 VM 维护）。
c) 设备和 I/O 虚拟化：统一标准化接口。
d) 实时迁移技术：热迁移（内存页面拷贝）。

4. 其他虚拟化的相关技术：

a) 网络虚拟化：虚拟 Mac 地址；VLAN（跨网段局域网），VPN（加密技术封装数据通讯隧道）。
b) 存储虚拟化：RAID（容量，速度，安全）；NST（网络存储）。
c) 桌面虚拟化：瘦客户端。
d) 应用虚拟化：应用和底层系统硬件分离（SAE，GAE）。
e) 容器虚拟化：隔离不同容器进程和资源，共享容器和宿主的资源；不需要指令解释（同一内核，本地运行）

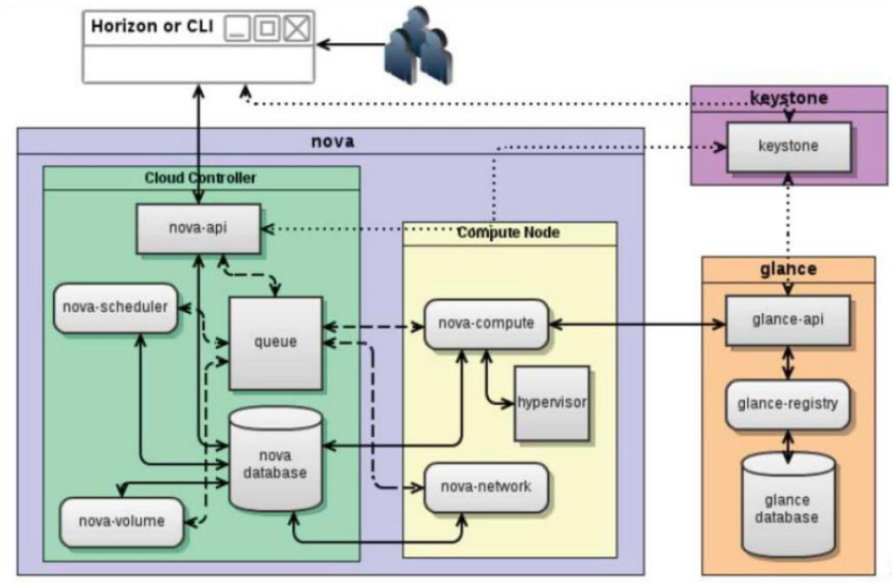
5. 典型虚拟机：

KVM、Xen。

6. 虚拟化和云计算的关系：
   资源汇聚（虚拟表示的汇聚），服务方式提供（对虚体进行检索浏览；通过虚体对实体资源分配回收；通过虚拟化接口访问实体资源）。

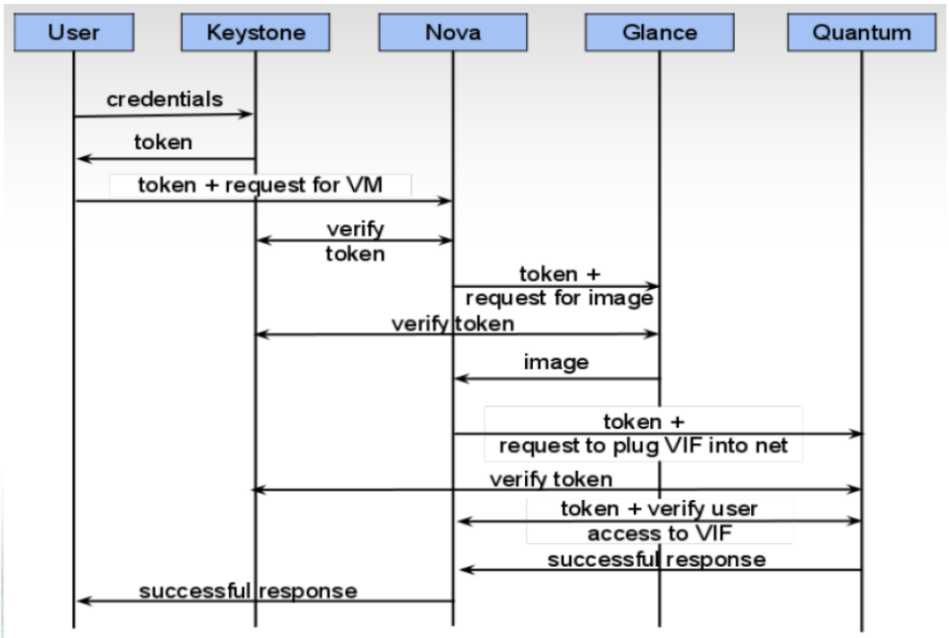
Ch06：

1. AWS 模式是什么，有什么优点：
   云计算 IaaS 和 PaaS 平台服务。AWS 面向用户提供包括弹性计算、存储、数据库、应用程序在内的一整套云计算服务。
   以 API 管理服务，通过认证和授权区分用户，完全的 SOA(面向服务的架构)，IaaS 标准，构建了完整的云计算生态系统，按需使用，按用计费。
2. IaaS 模式核心需求有哪些：
   计算虚拟技术的多样选择；存储技术/设备的多样支持；网络技术/设备的多样支持；多种 API 的支持；松耦合，通过组合组件，模块和服务来构成整个系统；组件，模块和服务功能内聚。
3. Openstack 都包含哪些核心项目，作用：
   Nova（计算），swift（对象存储），glance（镜像服务），keystone（身份服务），horizon（UI 界面），Neutron（网络和地址管理），cinder（块存储）。
4. 镜像和实例有什么区别和联系：
   镜像是一系列虚拟化硬件和对应软件的固定搭配，类似于一个类的概念。实例是对以上固定搭配的一个实例，类似于对类进行实例化。镜像相当于对实例的一个抽象化表示，便于对大量实例进行管理。
5. Nova 有哪些核心模块，工作过程是什么：
   核心模块：
   Nova-compute（虚拟机实例创建终止迁移，接受请求，执行并更新数据库状态），nova-volume（映射到实例的卷的创建附加取消），nova-network（接受网络任务，控制虚拟机网络），nova-scheduler（调度，决定哪台机器启动新的虚拟机实例），queue（守护进程传递消息），SQLdatabase（存储数据）。
   工作过程：
   用户输入命令，api 会查看这种类型的 instance 是否达到最大值，给 scheduler 发送一个消息（实际上是发送到 Queue 中）去运行这个实例。
   调度器接收到了消息队列 Queue 中 API 发来的消息，然后根据事先设定好的调度规则，选择好一个 host，之后，这个 instance 会在这个 host 上创建。
   真正创建 instance 是由 compute 完成的，通过 glance 查找镜像。
   根据找到的镜像，到 database 中查找相应的数据。
   volume 创建虚拟机实例的卷。
   network 为虚拟机分配 IP 等网络资源。

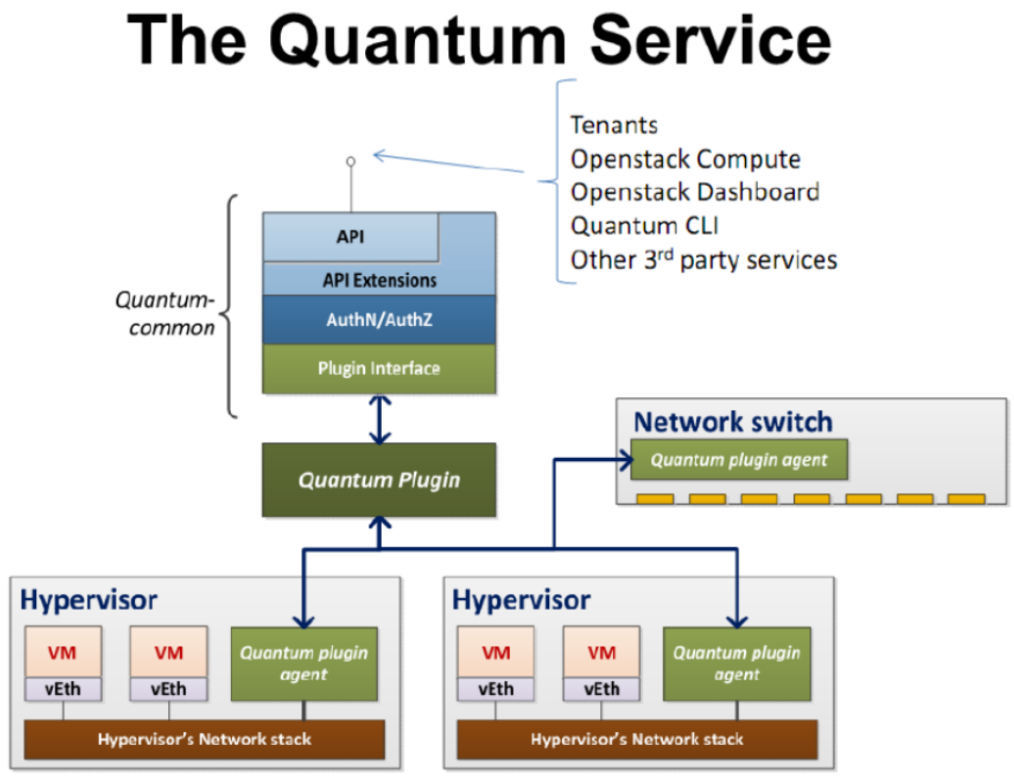


6. Keystone 权限控制过程是什么：

用户传 credential 给 keynote 进行请求，keynote 进行认证以后分配给用户一个 token（令牌），用户获得权限，将令牌和虚拟机请求传给 nova，nova 向 keynote 验证令牌，获得权限后连同对镜像的请求传给 glance，glance 向 keynote 验证令牌，把镜像传给 nova；nova 再将用户接入网络的请求传给 quantum，验证成功后，即传出成功访问的回答。



7. Quantum 原理是什么：



8. Cinder 存储的机制是什么：

   块存储单元“卷”，多个卷可以挂载到同一个虚机，卷可以在虚机间移动，同一个卷同一个时间只能被挂载的一个虚机实例，块存储管理（块设备到虚机的创建，挂载和卸载）。

9. Swift 的核心概念有哪些：

   object：对象。基本的存储实体，所有数据按照对象进行存储

   container：容器。对象的装载体，组织数据的方式，存储的隔间，类似于文件夹，但不能嵌套

   account：账户。权限单位，一个 account 拥有若干 container

10. Swift 的组件有哪些，都有什么作用：

    Proxy Server ：是提供 Swift API 的服务器进程，负责 Swift 其余组件间的相互通信。对于每个客户端的请求，它将在 Ring 中查询 Account、Container 或 Object 的位置，并且相应地转发请求

    Storage Server ：提供了磁盘设备上的存储服务

    Consistency Servers ：查找并解决由数据损坏和硬件故障引起的错误

    Ring：Swift 最重要的组件，用于记录存储对象与物理位置间的映射关系。

Ch07：

1. 大规模数据存储面临的新问题与挑战：

   传统模式的挑战——核心问题是数据量与成本的矛盾

   成本（处理能力扩充、人力等），效率（从降低的量变到不可用的质变），变更（业务逻辑），数据库的限制（表结构），其他（易用性、可靠性、安全性）。

2. 索引技术：

   索引分为 3 种类型：二叉搜索树 BST，B 树、B+树、B*树，R 树。

   BST：小数在左子节点，大数在右子节点。同一组数据可能形成不同的书。

   B 树：

根节点至少有两个子节点、每个节点有 M-1 个 key，并且以升序排列、位于 M-1 和 M key 的子节点的值位于 M-1 和 M key 对应的 Value 之间、其它节点至少有 M/2 个子节点。

B+树:

有 k 个子结点的结点必然有 k 个关键码、非叶结点仅具有索引作用，跟记录有关的信息均存放在叶结点中、树的所有叶结点构成一个有序链表，可以按照关键码排序的次序遍历全部记录。

R 树:

R 树是用来做空间数据存储的树状数据结构。例如给地理位置，矩形和多边形这类多维数据建立索引。

R 树的核心思想是聚合距离相近的节点并在树结构的上一层将其表示为这些节点的最小外接矩形，这个最小外接矩形就成为上一层的一个节点。

3. GFS 体系结构:

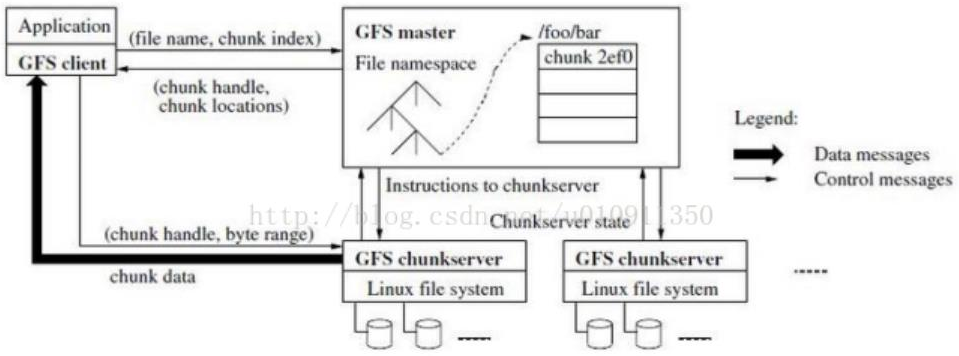


Figure 1: GFS Architecture

一个 GFS 集群包含三个角色：一个单独的 GFS Master 总控制服务器，多台 GFS Chunkserver（数据块服务器，简称 CS）和多个 GFS Client 客户端.GFS 存储的文件都被分割成固定大小的 Chunk。 Chunk 服务器把 Chunk 以 linux 文件的形式保存在本地硬盘上，并且根据指定的 Chunk 标识和字节范围来读写块数据。为了保证可靠性，Chunk 在不同的机器中复制多份，缺省情况下，使用 3 个存储复制节点，不过用户可以为不同的文件命名空间设定不同的复制级别。

Master 中维护了系统的元数据（SQL 中的说法是:数据的数据）,这些元数据包括 Chunk 名字空间、访问控制信息、文件和 Chunk 的映射信息、以及当前 Chunk 的位置信息.。Master 还管理着系统范围内的活动，比如， Chunk 租用管理无用 Chunk 的回收、以及 Chunk 在 Chunk 服务器之间的迁移。

Master 节点使用心跳信息周期地和每个 Chunk 服务器通讯, 发送指令到各个 Chunk 服务器并接收 Chunk 服务器的状态信息。

Client 代码实现了 GFS 文件系统的 API 接口函数以及应用程序的访问接口。应用程序从 Master 获取元数据，根据元数据提供的信息与 Chunk 服务器直接进行交互。从架构图可以看出，Client 和 Master 之间的交互只有控制流（指令信息），没有数据流，因此降低了 Master 的负载（因为控制流只需传送指令和状态, 数据量小）。Client 与 Chunk 之间直接传输数据流，同时由于文件被分成多个 chunk 进行分布式存储，因此 Client 可以同时并行访问多个 Chunk，从而让系统的 I/O 并行度提高。GFS 不提供 POSIX 标准的 API 的功能，因此，其 API 调用不需要深入到 Linux vnode 级别。

需要注意的是，GFS 中的客户端不缓存文件数据，只缓存 Master 中获取的元数据。

Chunk 服务器也不需要缓存数据，Chunk 以本地文件的方式保存， Linux 操作系统的文件系统缓存会把经常访问的数据缓存在内存中。

4. 云存储应用的特点：

通用的设备支持（云存储的浏览端），数据同步与共享，任意格式/大小文件，免费+付费。

Ch08：

1. 并行化思想：

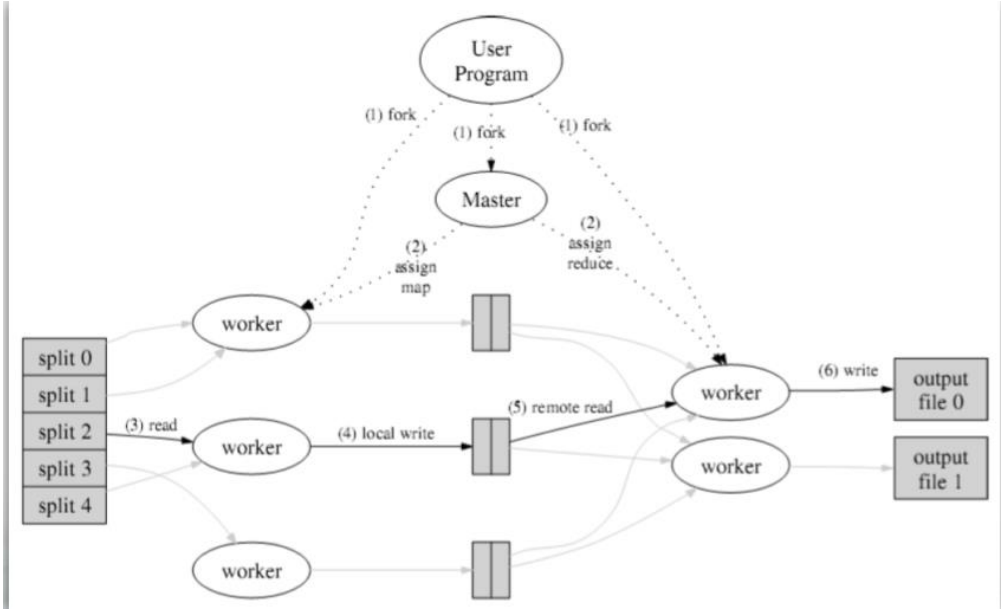
将一道指令划分为几部分，然后它们可以并发的执行。各部分的指令分别在不同的 CPU 上运行，这些 CPU 可以在单台或多台机器中，他们连接起来共同运作。

并行化计算适用问题间存在依赖关系或数据的结构一致处理逻辑相同。

2. 批量计算特点：

数据规模不会变化，由用户来驱动计算。对计算的实时性要求不高，同时存在重复性问题。计算的逻辑相对简单。是一种批量、高时延、主动发起的计算。

3. Mapreduce 算法的架构：



4. Mapreduce 算法设计思想：

对于大数据并行处理：分而治之。对于不存在依赖关系的数据，用一定的数据划分方法对数据分片，然后将每个分片交给一个节点去处理，最后汇总结果。

抽象模型 map 和 reduce：

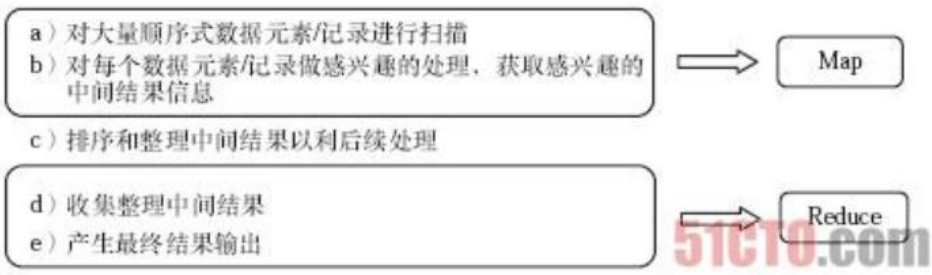


图 1-13 典型的顺序式大数据处理过程和特征

Map 操作主要负责对一组数据记录进行某种重复处理，而 Reduce 操作主要负责对 Map 的中间结果进行某种进一步的结果整理和输出。

例如：master 对文本进行分割。在分割之后的每一对<key,value>进行用户定义的 map 进行处理，生成新的<key,value>对。对输出的结果集归拢排序。通过 reduce 操作生成最后结果。

5. 算法调优：

更优化的 key-value 对设置。

Map 算法：key-value 对的生成。

Combiner 算法：对 key 的解析，对 value 的解析，对 value 的计算。为了减少中间结果量。

Partiton 算法：控制分发策略，多元 key 的解析。

Reduce 算法：对 key 的解析，对 value 的解析，对 value 的计算（类似 combiner）。

6. Mapreduce 运行过程中的各种参数及其作用：

Map:

io.sort.mb(缓冲区大小)、io.sort.spill.percent(缓冲区容量阈值)。当缓冲区被填满后，map 会阻塞直到写过程完成。

lo.sort.factor(多溢出写文件合并流数)、min.num.spill.for.combine(溢出写次数最小值)，数据在传送到 reducer 前进行 patition。

Mapred.compress.map.output 、mapred.map.output.compression.codec(压缩标志和方式)。写磁盘时压缩 map 提高输出效率。

Tracker.http.threads(tasktracker 的工作线程数)。文件的分区工作线程控制。

Reduce:

Mapred.reduce.parallel.cpoies(reduce 任务复制线程数）、mapred.reduce.copy.backoff(reduce 获取一个 map 的最大时间)。一个 map 任务完成后，reduce 任务复制输出。

Mapred.job.shuffle.input.buffer.percent(map 输出内存缓冲区占堆空间的百分比）、mapred.job.shuffle.merge.percent(缓冲区输出阈值）、mapred.inmem.merge.threshold(map 输出阈值)。Map 先写入缓冲区再合并。

lo.sort.factor(合并因子)。将副本合并成更大的排好序的文件。(40 个文件，第一次 4 个，后三次合并 10 个，最后一次 4 个文件和剩余 6 个合并)。

Mapred.job.reduce.input.buffer.percent(输入内存阈值)。内存中 map 输入大小不能超过输入内存阈值。

lo.file.buffer.size(hadoop 文件缓冲区)。Reduce 的输出结果写入 hdfs 系统。

7. 参数调优：

给 shuffle 过程尽可能多的内存空间。Map 和 reduce 函数尽量少用内存。运行 map 和 reduce 任务的 jvm 的内存尽可能大。Map 端估计 map 输出大小，减少溢出写磁盘次数。Reduce 端的中间数据尽可能多驻留内存。增加 hadoop 文件缓冲区。

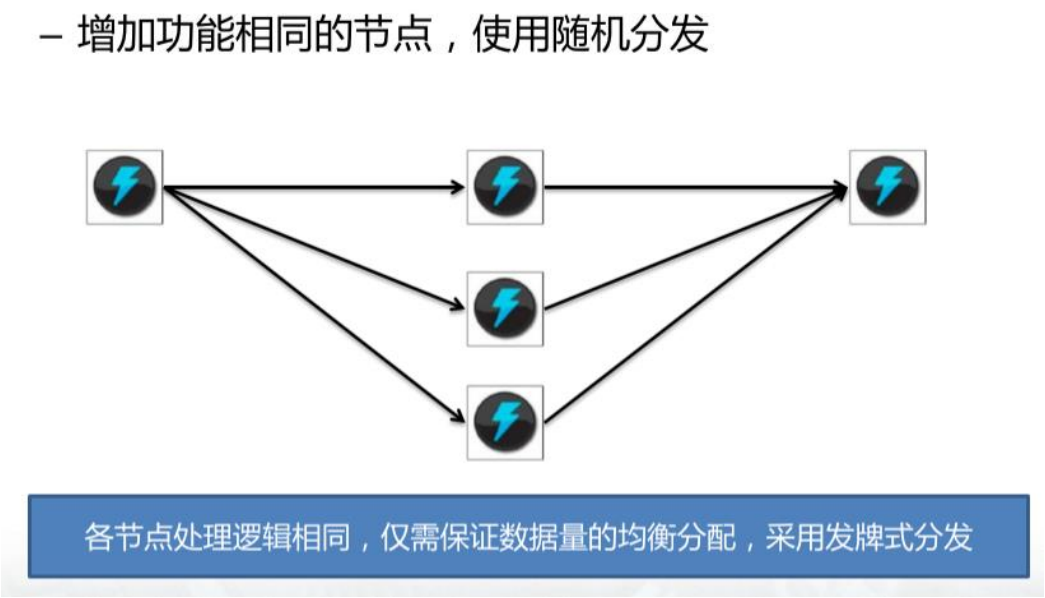
Ch09：

1. 流式计算和批量计算的区别：

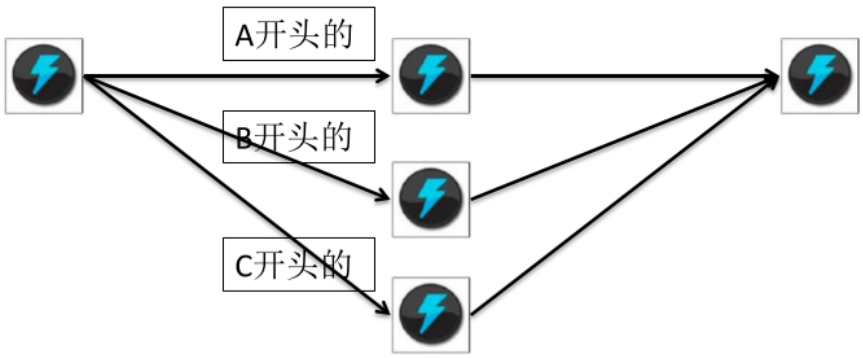
2. 流式计算的关键要素：

   对于数据来的太快的问题，需要提高分词节点的效率，所以需要增加节点的个数，核心在于制定合理的分发策略。将单个节点的处理压力分散到多个节点上。

3. 数据分发机制：

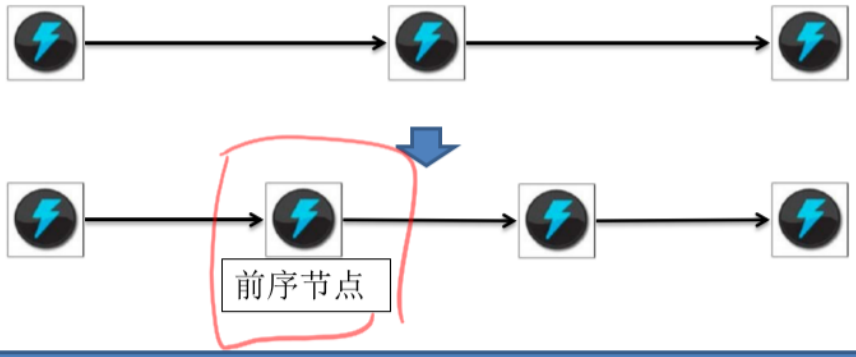


– 将处理功能分散，使用特定值分发



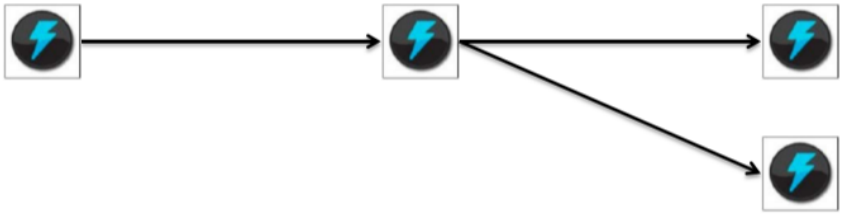
各节点负责处理的内容不同，把满足不同条件的内容分发到相应的节点上

– 增加前序节点，在处理前对数据进行某种转换



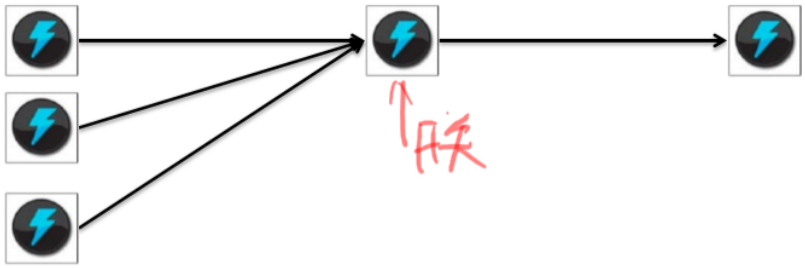
前序节点可以起到限流、过滤、变换等作用

– 如果多个后续节点都需要同一个数据，可以设置一个专门的数据转发节点，使用广播分发发给这些后续节点



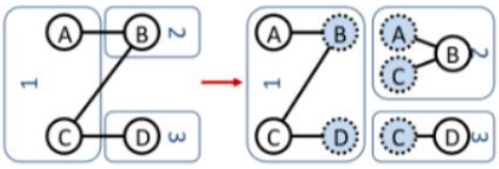
广播节点实际上起到了数据复制的作用，使得同一份数据进入不同的管道

– 某些节点可以作为同步节点，接收到来自多个上游的数据后触发下一个步骤

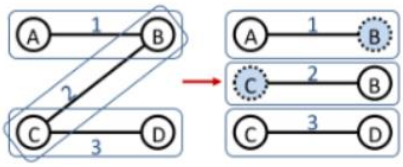


同步节点实际上起到了数据开关的作用，控制管道开关

4. 运用流式计算方法解决实际问题：
   数据采集：获取数据
   数据计算：处理数据
   数据查询：提供结果
5. 图的切分方式：
   边切分，点切分。

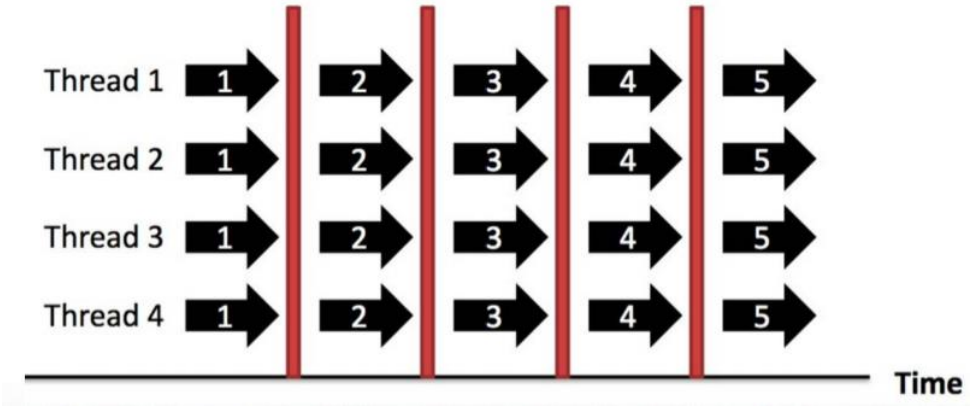


(a) Edge-Cut



(b) Vertex-Cut

6. BSP 计算模式：
   将计算分为一系列的超步的迭代。纵向上看是一个串行模式，一轮一轮顺序化串行。横向上看是一个并行的模式。每两个超步间设置一个栅栏作为整体同步点。确定所有并行的计算都完成后再启动下一轮超步。



7. 图数据计算的并行思想：

迭代各个轮次。每一轮次每个节点分别计算当前轮次的权值。每个轮次各个节点分别计算完。开始下一轮次。

8. 运用图数据计算方法解决实际问题：

并行 dijksta 算法。

Ch10：

实验课

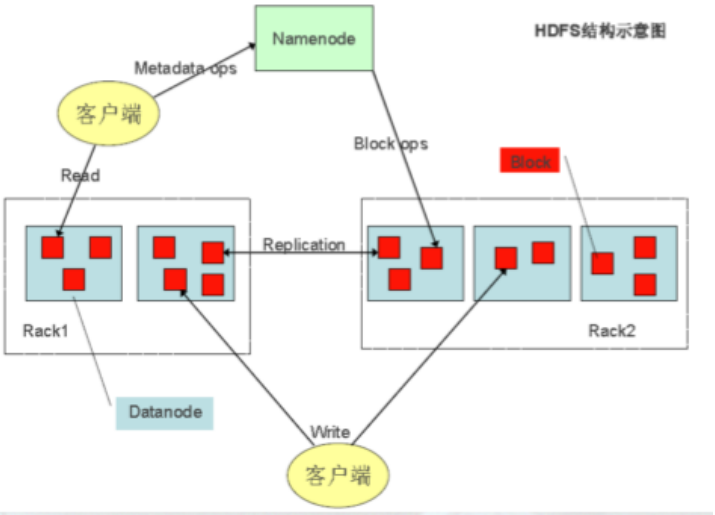
Ch11：

1. hadoop 项目的由来：

Hadoop 基于网络搜索引擎 nutch，由于缺乏可扩展的架构，借鉴 GFS 实现了 NDFS，再加入 mapreduce 系统后形成 hadoop。

其中有 mapreduce、hdfs、hbase、zookeeper、pig、hive。

2. HDFS 的体系结构：



3. HDFS 的运行机制：

读文件：

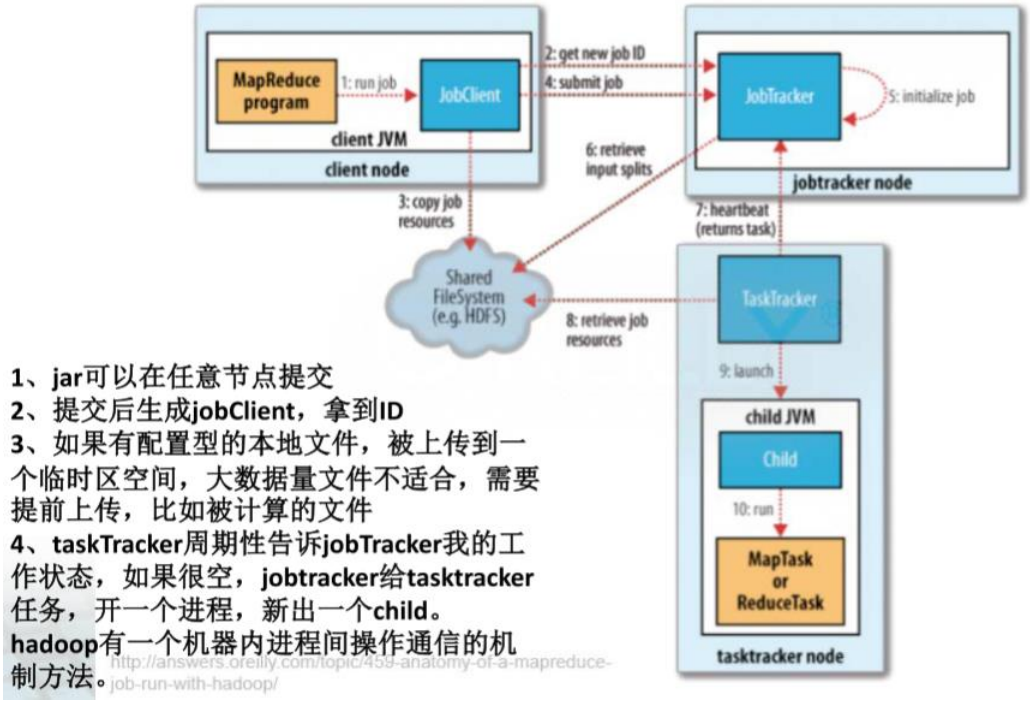
客户端调用 DistributedFileSystem 对象的 Open()方法。DistributedFileSystem 通过 RPC 联系 namenode，得到所有数据块信息，对每个数据块，namenode 返回存有该块副本的 datanode 地址，并且这些 datanode 根据他们与客户端的距离进行排序。DistributedFileSystem 类返回一个 FSDataInputStream 对象给客户端并读取数据。客户端对该对象调用 read()方法读取数据。FSDataInputStream 连接最近的 datanode 读取数据（同一节点，同一机架上不同节点，同一数据中心不同机架，不

同数据中心），数据读取完毕时 FSDataInputStream 会关闭与该 datanode 的链接，然后寻找下一块的 datanode。FSDataInputStream 可能并行读取多个 datanode，当客户端完成读取时，对 FSDataInputStream 调用 close()方法。

写文件：

客户端调用 DistributedFileSystem 对象的 create()方法创建文件。DistributedFileSystem 通过 RPC 联系 namenode，namenode 执行各种检查确保待建立的文件不存在，且客户端拥有创建该文件的权限。如果检查通过，namenode 为新文件创建一条记录，否则抛出一个 IOException 异常。DistributedFileSystem 给客户端返回一个 FSDataOutputStream 对象进行写数据。FSDataInputStream 将待写数据分成数据包并写入内部队列 dataqueue。DataStreamer 处理 dataqueue，根据 datanode 列表要求 namenode 分配适合的新块来存储数据备份。Namenode 分配的数据备份 datanode 形成一个管线（第一复本在节点本身，第二复本在随机选择的机架上，第三复本在第二复本机架随机节点），DataStreamer 将数据包传输给管线中的第一个节点，然后该节点存储完之后发送给第二个节点，以此类推。

4. Hadoop 中 mapreduce 的实现机制：



1、jar可以在任意节点提交
2、提交后生成jobClient，拿到ID
3、如果有配置型的本地文件，被上传到一个临时区空间，大数据量文件不适合，需要提前上传，比如被计算的文件
4、taskTracker周期性告诉jobTracker我的工作状态，如果很空，jobtracker给tasktracker任务，开一个进程，新出一个child。
hadoop有一个机器内进程间操作通信的机制方法。
http://answers.oreilly.com/topic/459-anatomy-of-a-mapreduce-job-run-with-hadoop/

5. Htable 的数据结构：

Row key：行主键，读取记录只能按 Row key（及其 range）或全表扫描，因此 Row key 需要根据业务来设计以利用其存储排序特性。

Column Family（列族）：在表创建时声明，每个 Column Family 为一个存储单

Column（列）：HBase 的每个列都属于一个列族，以列族名为前缀。Column 不用创建表时定义即可以动态新增，同一 Column Family 的 Columns 会群聚在一个存储单元上，并依 Column key 排序。

Timestamp：HBase 通过 row 和 column 确定一份数据，这份数据的值可能有多个版本，不同版本的值按照时间倒序排序，即最新的数据排在最前面，查询时默认返回最新版本。

Value：每个值通过 4 个键唯一索引。

6. Hbase 的运行机制：

数据存储实体为区域，表按照水平的方式划分为一个或多个区域，每个区域有一个随机 id，且区域内行为键值有序的。区域以分布式方式存储在集群内。通过区域服务器运行：

写：写数据首先写入“预写日志”；先缓存，再批量写入；完成后在日志中做标记

读：区域服务器先在缓存中查找，找到则直接服务；

合并：映射文件数量超过阈值，则区域服务器进行合并

分割：区域文件大过阈值时，按照行方式对半分割；在元信息表中生成子元信息表；主服务器在得知分割后，将子表分配给新的区域服务器服务

失效恢复：将失效服务器的区域分配给其他服务器，原“预写”日志进行分割并分配给新的区域服务器

7. Yarn 对 hadoop 的核心改进：

将原框架中的 jobtracker 和 tasktracker 拆分为 ResourceManager，ApplicationMaster 与 NodeManager 三个部分。

ResourceManager 是中心服务，监控调度启动 ApplicationMaster。负责作业和资源的调度，接受 jobsubmitter 提交的作业，分配 container 作为 appmstr。

ApplicationMaster 负责 job 生命周期中的所有工作。每个 job 都有一个 ApplicationMaster。

NodeManager 负责 container 状态维护，保持心跳。

对资源的表示用内存为单位，比之前用剩余 slot 数目更合理。

8. Spark 架构和运行机制：

Spark 架构：

Spark 架构使用了分布式计算中 master-slave 模型，master 是集群中含有 master 进程的节点，slave 是集群中含有 worker 进程的节点

master 作为整个集群的控制器，负责整个集群的正常运行

worker 相当于计算节点，接受主节点命令与状态汇报

executor 负责任务的执行

client 作为用户的客户端负责提交应用

driver 负责控制一个应用的执行

运行机制：

Spark 集群部署后，需要在主节点和从节点分别启动 master 进程和 worker 进程来控制集群。在一个应用执行中，driver 是应用逻辑执行的起点，负责作业的调度，即 Task 任务的分发，而多个 worker 用来管理计算节点和创建 executor 并行处理任务。在执行阶段，driver 会将 task 和其依赖的文件传递给 worker 机器，同时 executor 对相应数据分区的任务进行处理。

SparkContext： 整个应用的上下文，控制应用的生命周期。

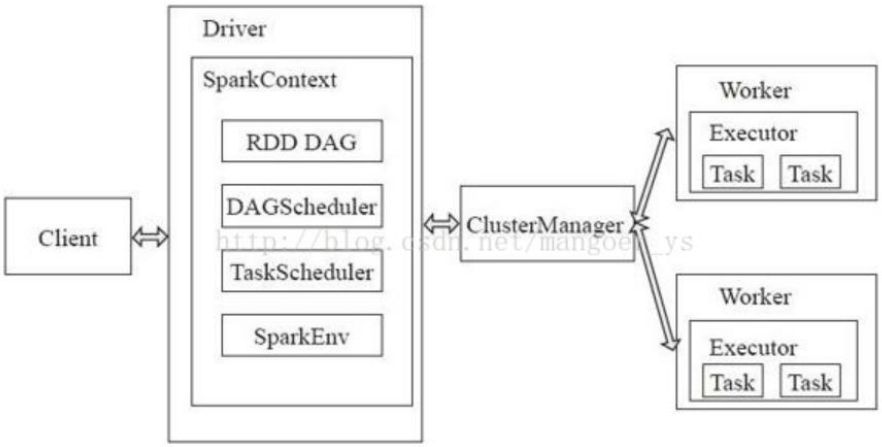
RDD： Spark 的基本计算单元，一组 RDD 可执行的有向无环图 RDD Graph。

DAGScheduler： 根据作业构建基于 Stage 的 DAG，并提交给 Stage 的 TaskScheduler。

TaskScheduler： 将任务分给 executor 执行。

SparkEnv： 线程级别的上下文，存储运行时的重要组件的引用。

运行流程：Client 提交应用，master 找到一个 worker 启动 driver，driver 向 master 请求资源，之后将应用转化为 RDD Graph，再由 DAGScheduler 将 RDD Graph 转换为 stage 的 DAG 提交给 TaskScheduler，由 TaskScheduler 提交任务给 executor。



Spark架构

9. Storm 架构和运行机制：

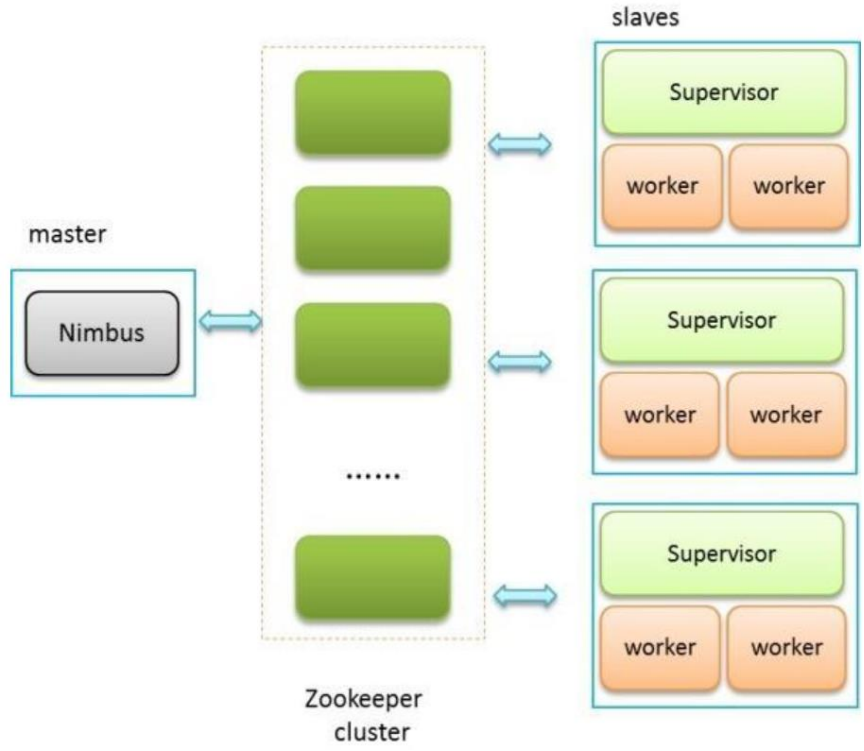
Storm 是一个分布式流处理&&实时的计算系统

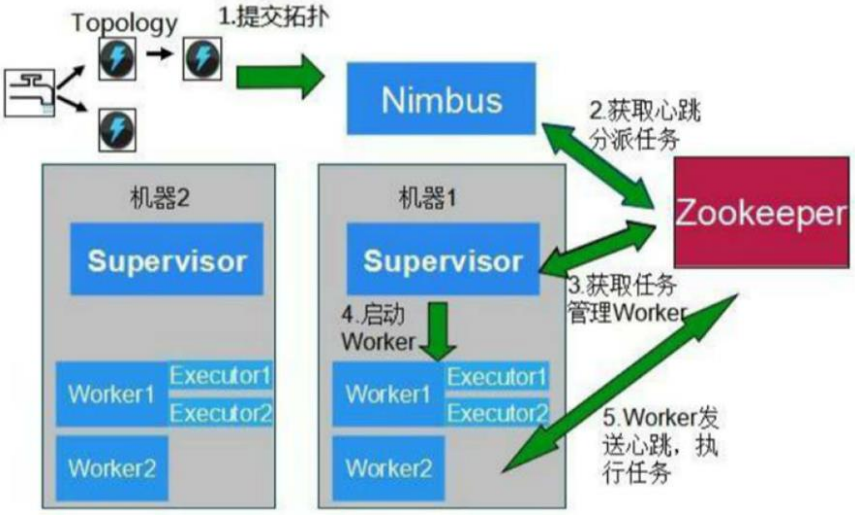
Storm 架构：

Storm 采用 Master/Slave 体系结构,分布式计算由 Nimbus（主节点）和 Supervisor（工作节点）两类服务进程实现,Nimbus 进程运行在集群的主节点,负责响应集群节点，分配任务和故障检测 ,Supervisor 运行在集群的从节点,负责收听工作指派，运行工作进程。Nimbus 和 Supervisors 之间所有的协调工作是通过一个 Zookeeper 集群实现。

Storm 实现了一种数据流模型，其中数据持续地流经一个转换实体网络。一个数据流的抽象称为一个流（stream），这是一个无限的元组序列。每个流由一个唯一 ID 定义，这个 ID 可用于构建数据源和接收器（sink）的拓扑结构。流起源于喷嘴（spout），Spout 将数据从外部来源流入 Storm 拓扑结构中。接收器（或提供转换的实体）称为螺栓（bolt）。螺栓实现了一个流上的单一转换和一个 Storm 拓扑结构中的所有处理。Bolt 既可实现 MapReduce 之类的传统功能，也可实现更复杂的操作（单步功能），比如过滤、聚合或与数据库等外部实体通信。典型的 Storm 拓扑结构会实现多个转换，因此需要多个具有独立元组流的 Bolt。Bolt 和 Spout 都实现为 Linux 系统中的一个或多个任务。

拓扑 topology：Storm 中运行的一个应用是一个拓扑（对应 Hadoop 里的 job）。





10. Kafka 架构和运行机制：

消息发布机制：

Producer 指定发布的 topic

Topic 里的信息物理上分为多个 partition 存储

Broker 存储若干 topic 的 partition，允许同 topic 的不同 partition 存储在不同的 broker 上

由 zookeeper 统一管理

Producer 可以通过指定消息的 key 控制将消息发到某个 partition

消息消费机制：

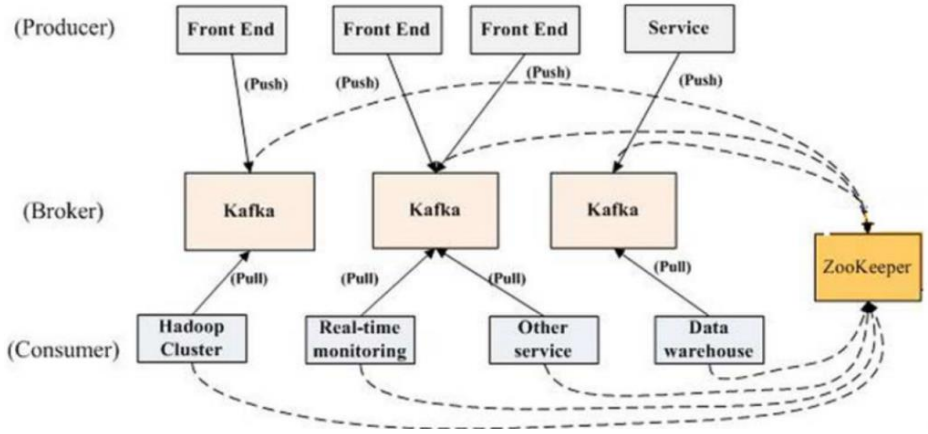
Consumer 需要指定消费的 Topic

Consumer 需要指定隶属的 consumer group

一个 Topic 里的每条信息只能被 consumer group 中的一个 consumer 消费，

属于不同 consumer group 的 consumer 可以共同消费一个 topic 的相同信息

Consumer group 里的 consumer 和 topic 的 partition 按照 id 顺序进行消费。

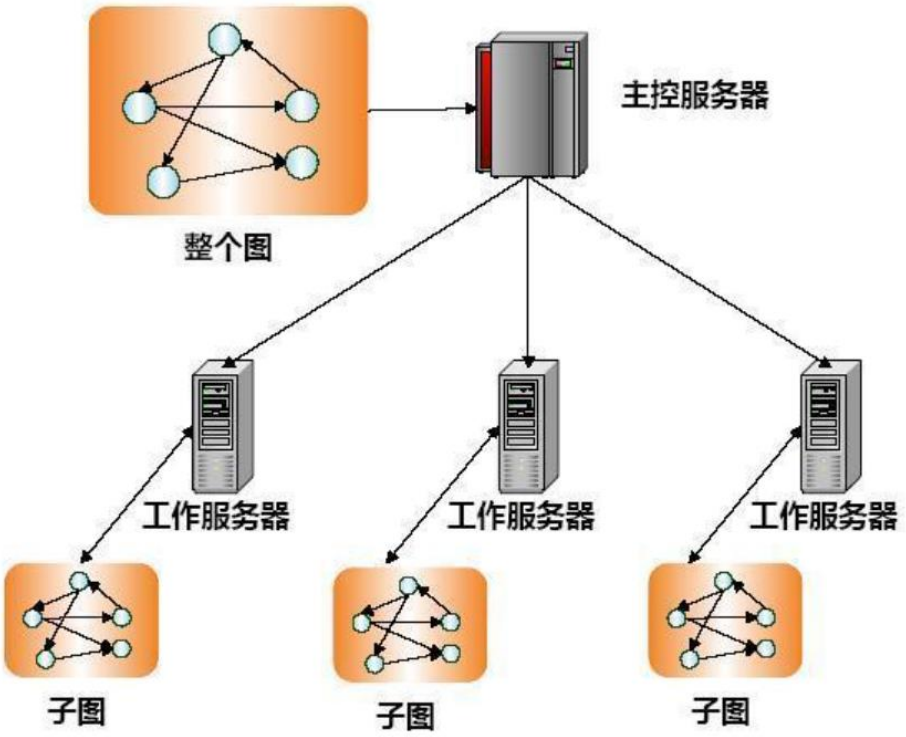


11. Pregel 架构和运行机制：

Pregel 采用了“主从结构”来实现整体功能，图 14-15 是其架构图，其中一台服务器充当“主控服务器”，负责整个图结构的任务切分，采用“切边法”将其切割成子图（Hash(ID)=ID mod n ，n 是工作服务器个数），并把任务分配给众多的“工作服务器”，“主控服务器”命令“工作服务器”进行每一个超级步的计算，并进行障碍点同步和收集计算结果。“主控服务器”只进行系统管理工作，不负责具体的图计算。

每台“工作服务器”负责维护分配给自己的子图节点和边的状态信息，在运算的最初阶段，将所有的图节点状态置为活跃状态，对于目前处于活跃状态的节点依次调用用户定义函数 F(Vertex)。需要说明的是，所有的数据都是加载到内存进行计算的。除此之外，“工作服务器”还管理本机子图和其他“工作服务器”所维护子图之间的通信工作。

在后续的计算过程中，“主控服务器”通过命令通知“工作服务器”开始一轮超级步的运算，“工作服务器”依次对活跃节点调用 F（Vertex），当所有的活跃节点运算完毕，“工作服务器”通知“主控服务器”本轮计算结束后剩余的活跃节点数，直到所有的图节点都处于非活跃状态为止，计算到此结束。



12. 各种分布式处理框架的异同点：

Ch12：

1. 分布式处理的不一致情况有哪些：
2. 有哪些原因造成了分布式处理的不一致：

   硬件损坏，节点掉线，网络延迟等。
3. Quorum protocol 是什么：

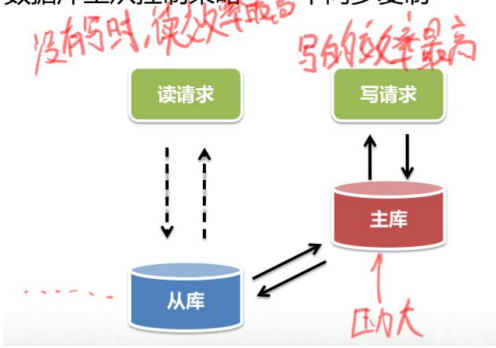
   Quorum 机制是“抽屉原理”的一个应用。定义如下：假设有 N 个副本，更新操作 wi 在 W 个副本中更新成功之后，才认为此次更新操作 wi 成功。称成功提交的更新操作对应的数据为：“成功提交的数据”。对于读操作而言，至少需要读 R 个副本才能读到此次更新的数据。其中，W+R>N ，即 W 和 R 有重叠。一般，W+R=N+1。

   假设系统中有 5 个副本，W=3，R=3。初始时数据为(V1，V1，V1，V1，V1）--成功提交的版本号为 1。

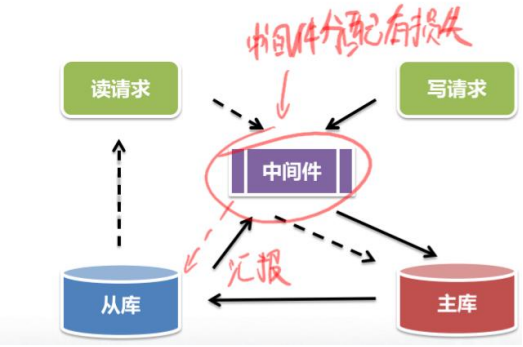
   当某次更新操作在 3 个副本上成功后，就认为此次更新操作成功。数据变成：(V2，V2，V2，V1，V1）--成功提交后，版本号变成 2

   因此，最多只需要读 3 个副本，一定能够读到 V2(此次更新成功的数据)。而在后台，可对剩余的 V1 同步到 V2，而不需要让 Client 知道。
4. 主从机制下主和从的任务特点是什么：
5. 数据库主从机制有哪些：

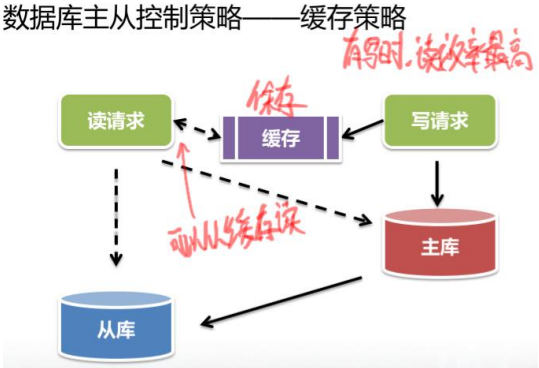
数据库主从控制策略——半同步复制



数据库主从控制策略——中间件控制



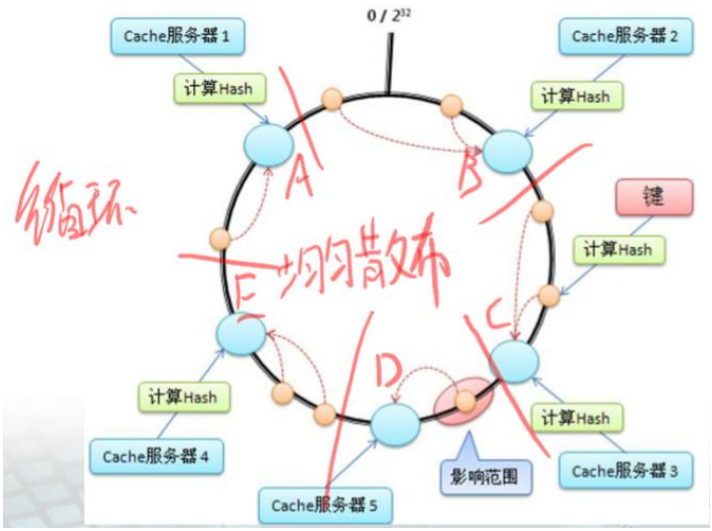
数据库主从控制策略——缓存策略



6. Ring 算法是什么，有哪些改进：

Ring 算法用在负载均衡上，防止因为节点数增加或减少时导致的哈希重新分配。

Ring算法



改进的 Ring 算法加入了虚节点，从 object→virtual node→node 的映射表。先把虚节点预设放在环上，当节点个数发生变化时调整节点和虚节点之间的映射。

7. Zookeeper 的实现机制是什么：

ZooKeeper 提供通用的分布式锁服务，用以协调分布式应用，适用于主要负载为读的应用场合

Zookeeper 架构：

ZooKeeper 是一个由多个 Server 组成的集群

一个 Leader，多个 Follower

每个 Server 都保存了一份数据副本

全局数据一致

分布式读写

Leader 负责写和同步，follower 负责读

更新请求转发，由 Leader 实施

Zookeeper 使用：

独占锁：如果分布式应用需要对某资源独占使用，可以申请独占锁，有且仅有一个 Client 可以获取到独占锁

共享锁：如果之前没有独占锁，就可以获取共享锁

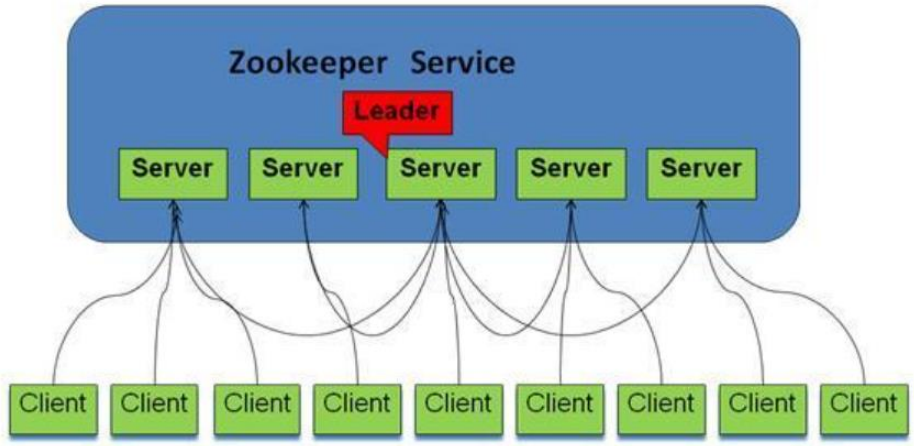
使用 ZooKeeper 的约定：

更新请求顺序执行：来自同一个 Client 的更新请求按其发送顺序依次执行

数据更新原子性：一次数据更新要么成功，要么失败。不存在部分数据写入成功或失败的情况

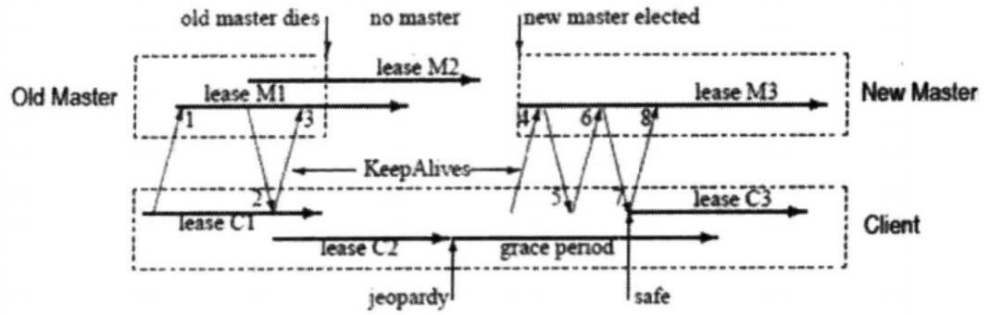
全局唯一数据视图： Client 无论连接哪个 Server，数据视图都是一致的

实时性：在一定时间范围内，Client 能读到最新数据



8. 时间片机制有什么用，chubby 释放和重连机制是什么：

时间片通信协议是一种倒计时“租期”系统，实现对操作授权的“限时”。



Ch13：

1. GFS 容错机制：

Master 容错：命名空间、Chunk 和文件名的映射靠日志容错，Chunk 副本位置靠 ChunkServer 容错。

ChunkServer 容错：多个存储副本存储在不同服务器上。每个 chunk 划分若干个 Block，每个 Block 对应校验码。

2. HDFS 容错机制：

数据复制（机架感知策略），故障检测（心跳包，块报告，数据完整性检测，日志文件，镜像文件）。

读文件容错：读数据遇到错误，尝试从最近的 datanode 读取数据，记住故障 datanode 保证以后不会继续从节点读取其他块。每个块通过校验和确认保证数据完整。发现损坏的块，在从其他 datanode 读取块之前通知 namenode。

写文件容错：维护一个确认队列，管线中所有 datanode 确认后，该数据包再被删除。Datanode 发生故障，则关闭管线将确认队列中的数据包添回数据队列最前端，将错误信息返回 namenode，从管线中删除错误节点，把剩余数据块写入 datanode。副本数量不足时，分配新的 datanode 并创建新的副本。

3. 内存数据库容错机制：

Snapshot：内存数据按照快照存储。父进程处理 client 请求，子进程讲内存内容写入临时文件。子进程将快照写入临时文件完毕后，用临时文件替换原来的快照。快照在系统重启后会丢失。

语句追加 aof：把写命令追加到文件中，系统重启时通过执行文件中保存的写命令在内存上重建数据库内容。父进程把写命令写入 aof 文件中，使用临时文件替换老的的 aof 文件并重命名。

4. 计算框架容错机制：

Worker 故障：master 周期性 ping 每个 worker，如果没收到返回信息，把这个 worker 标记成失效。重新执行该节点上已经执行或尚未执行的 map 任务。重新执行该节点上未完成的 Reduce 任务，已完成的不再执行。

Master 故障：定期写入检查点数据。从检查点恢复。

Spark 容错：Driver 中记录每个 RDD 生成的图，在 RDD 失效的时候，能够根据这个链条重新生成 RDD。RDD 在内存中备份。

5. Paxos 过程是什么：

Proposer 提出议题。Acceptor 初步接受 或者 Acceptor 初步不接受。如果上一步 Acceptor 初步接受则 Proposer 再次向 Acceptor 确认是否最终接受。Acceptor 最终接受或者 Acceptor 最终不接受。

6. 多租户的概念是什么：

探讨和实现如何于多用户的环境下共用相同的系统或程序组件，并且仍可确保各用户间数据的隔离性。希望利用多租户带来的资源高度共享模式，提高资源利用率，降低单位资源成本。

不共享，共享硬件，共享 OS，共享数据存储，共享容器，共享所有。

Ch14：

1. 什么是云安全：

通过网状的大量客户端对网络中软件行为的异常监测，获取互联网中木马、恶意程序的最新信息，传送到 Server 端进行自动分析和处理，再把病毒和木马的解决方案分发到每一个客户端。

2. 云安全的安全威胁：

共享技术漏洞，数据丢失和数据泄露，恶意内部用户，账户服务和传输劫持，不安的 APIs，服务的恶意使用，不确定风险预测。

3. 云计算的安全优势：

数据可访问性大大提高，分布式存储。

4. 物联网和云计算的关系：

云存储可以为物联网的海量数据提供足够大的存储空间，而云计算则可通过网格技

术、分布式技术等将不同类型的设备集合应用起来，协同起来对外提供数据存储以及业务分析等功能。

5. 什么是主机，什么是终端：

   云主机：主机公司将他的硬件和网络线路做成一朵云，向客户提供一些通向这朵云的网络接口 API。每个客户共享的不再是某个特定的服务器，而是云里的所有服务器。

   云终端：云的接入终端，是 UI 层，核心内容在云上。

6. 大型数据中心构建和管理主要需要做哪些事：

   构建：选址（电力，通信，安全性）、建筑要求（室内环境，布局）、电力布局、网络拓扑结构、环境控制、节能。

   上线：服务器（塔式，机架式，刀片式）、软件（操作系统，监控软件，业务软件），上线安装调优、软件部署测试。

   维护：硬件（升级，维护，更新，故障检查和处理，清洁）、软件（安装，配置，升级，监控，安全）、数据（备份，恢复，整合，存档，挖掘）、负载均衡（同一服务器不同类型资源，同一应用不同服务器，不同应用，时间不均衡）、安全（物理、数据、日志、备份恢复）。

7. 云计算中的信任机制如何构建：

   第三方机制，提高节点可信度，信任构建方式。

   区块链（分布式，去中心化，阳光化）。

8. NoSQL 相较于传统数据库的特点：

RDBMS	NoSQL
ACID: **A**tomicity, **C**onsistency, **I**solation, **D**urability	**BASE**: **B**asically **A**vailable, **S**oft State, **E**ventually consistent
强一致性(2 PC, Paxos)	最终一致性
复杂的操作(joins, queries)	简单的操作(CRUD)
Availability through scale-up	Availability through scale-out
可扩展性有限	高可扩展性
关键任务 (e.g., core banking)	Web 2.0, Internet companies

9. 常见的 NoSQL 数据类型和代表产品：

   列族数据库：Bigtable；键-值数据库：Dynamo；文档数据库：MongoDB；图数据库：Neo4j。