北京大学
PEKING UNIVERSITY

云计算思考题串讲

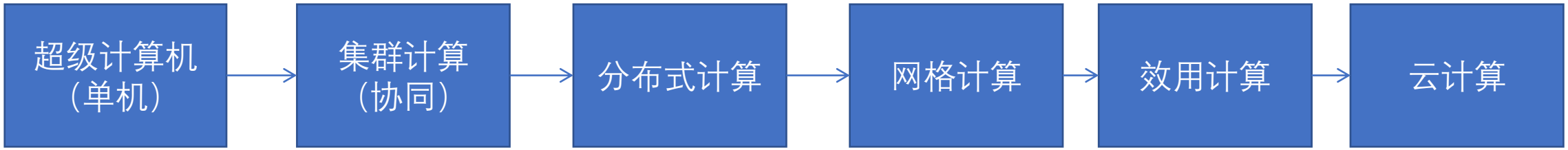
第一讲 云计算概述

1. 什么是云计算?
2. 云计算的发展历程
3. 云计算的动因

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。
- 三个主要类型：IaaS，PaaS，SaaS。

云计算的发展历程？



将一个巨大的问题分拆成许多小问题

服务提供商提供客户需要的计算资源和基础设施管理，并根据应用所占用的资源情况进行计费

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
- 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
- 商业规模的扩大（面向服务架构 SOA，SaaS）

第二讲 云计算与服务

1. 服务的概念
2. 云的技术范畴
3. 云服务的基本层次
4. 云的特征
5. 云的优势

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。

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）

云服务的基本层次

- 横向：公有云（Internet）|混合云（Internet 和 Intranet）|私有云（Intranet）【访问权限】
- 纵向：IaaS（基础设施层）→PaaS（平台层）→SaaS（应用层）【提供服务】

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）；

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）

第四讲 典型云服务

1. IaaS的基本功能
2. PaaS的基本功能
3. SaaS的典型应用

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。

IaaS的基本功能

- 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）。

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。

SaaS的典型应用

- 面向广大受众的标准化应用（a）满足用户日常生活办公需求(b)登陆认证(c)文档编辑(d)日程管理
- 定制的个性化服务应用(a)面向企业和机构用户的解决方案(b)财务管理(c)供应链管理(d)客户关系管理(e)物流管理
- 用户开发的多元化应用(a)独立软件开发商或第三方团队开发(b)为满足极为特定需求的创新应用(c)一般在公有云上创建

第五讲 典型云服务

1. 虚拟化的概念
2. 服务器虚拟化的特性
3. 服务器虚拟化的关键技术
4. 其他虚拟化的相关技术
5. 典型虚拟机
6. 虚拟化与云计算的关系

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。

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。

服务器虚拟化的关键技术

- 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）。

服务器虚拟化的关键技术

- 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）。
- 设备和 I/O 虚拟化：统一标准化接口。
- 实时迁移技术：热迁移（内存页面拷贝）。

其他虚拟化的相关技术

- 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）。
- 存储虚拟化：RAID（容量，速度，安全）；NST（网络存储）。
- 桌面虚拟化：瘦客户端。
- 应用虚拟化：应用和底层系统硬件分离（SAE，GAE）。
- 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）

典型虚拟机

- KVM:Keyboard Video Mouse.利用一组键盘、显示器和鼠标实现对多台设备的控制，在远程调度监控方面发挥着重要作用.
- Xen:是一个开放源代码虚拟机监视器

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

第六讲 虚拟化资源管理

1. AWS 模式是什么，有什么优点
2. IaaS模式核心需求有哪些?
3. Openstack都包含哪些核心项目，作用是什么?
4. 镜像和实例有什么区别和联系
5. Nova有哪些核心模块，工作过程是什么?
6. Keystone权限控制过程是什么?
7. Quantum原理是什么?
8. Cinder存储的机制是什么?
9. Swift的核心概念有哪些?
9. Swift的组件有哪些，都有什么作用?

AWS 模式是什么，有什么优点

- Amazon Web Services
- 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。

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

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）

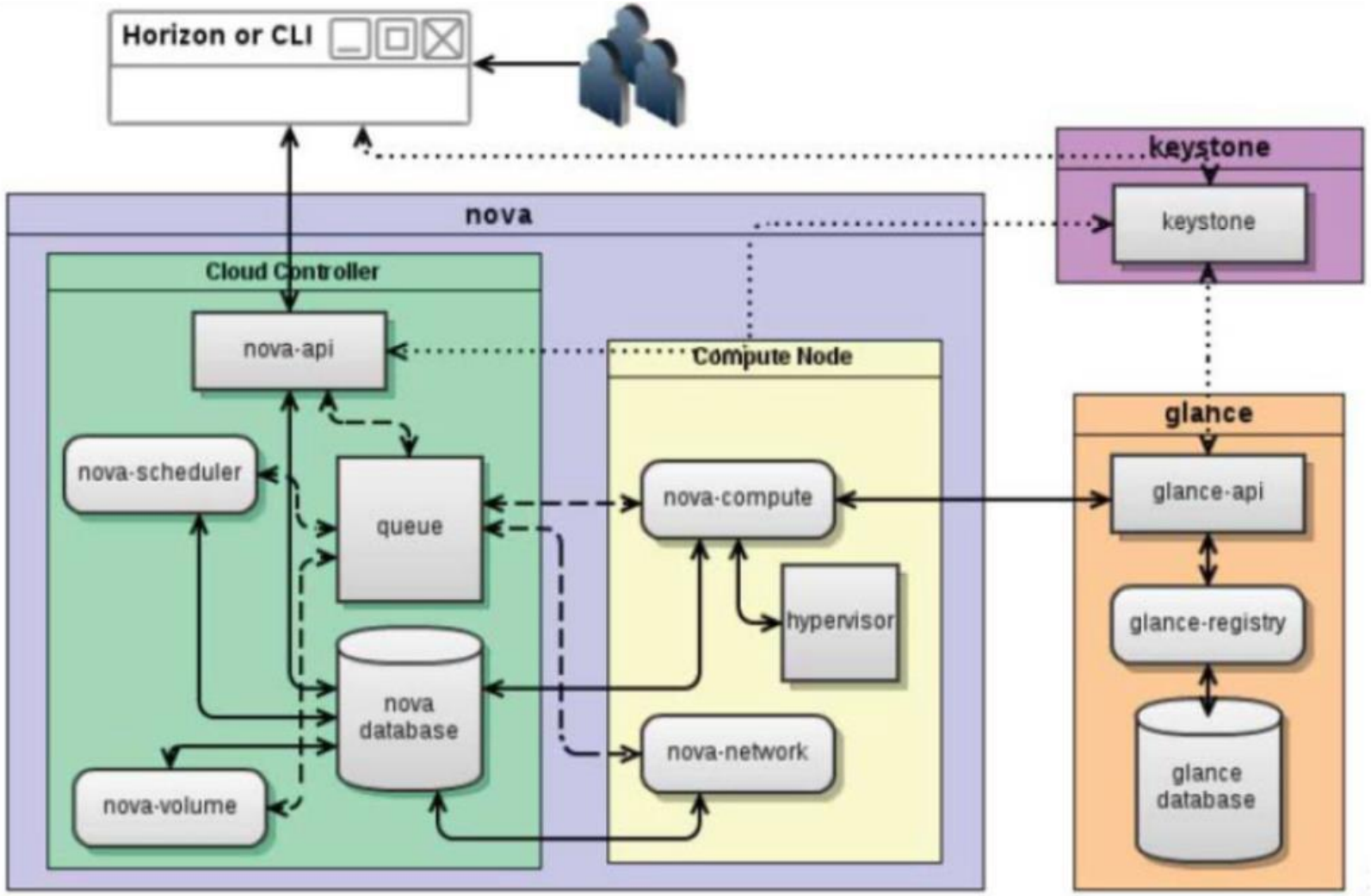
镜像和实例有什么区别和联系

- 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。

Nova 核心模块，工作过程是什么

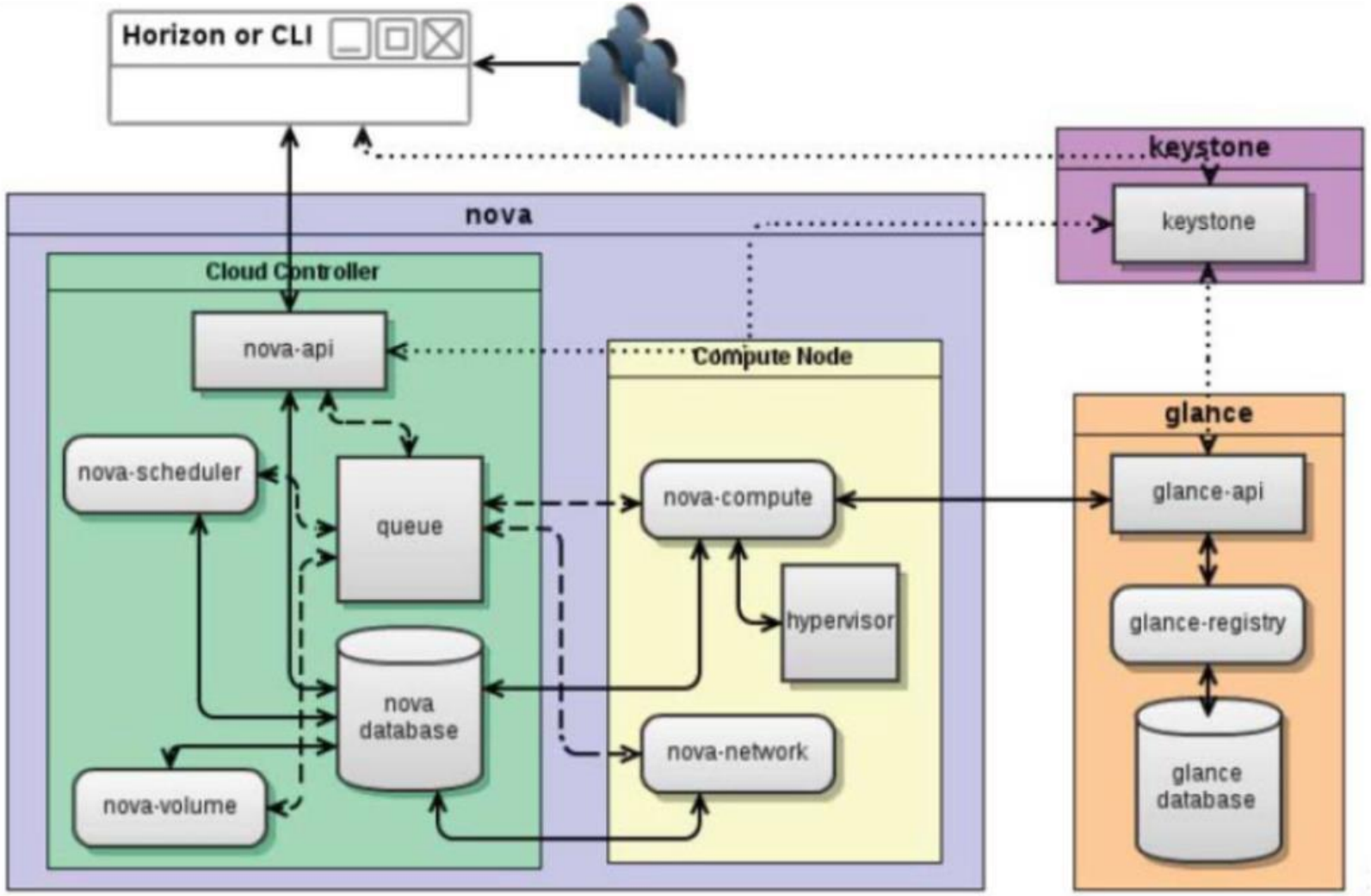
- Nova-compute（虚拟机实例创建终止迁移，接受请求，执行并更新数据库状态）
- nova-volume（映射到实例的卷的创建附加取消）
- nova-network（接受网络任务，控制虚拟机网络）
- nova-scheduler（调度，决定哪台机器启动新的虚拟机实
例）
- queue（守护进程传递消息）
- SQLdatabase（存储数据）。

Nova 核心模块，工作过程是什么



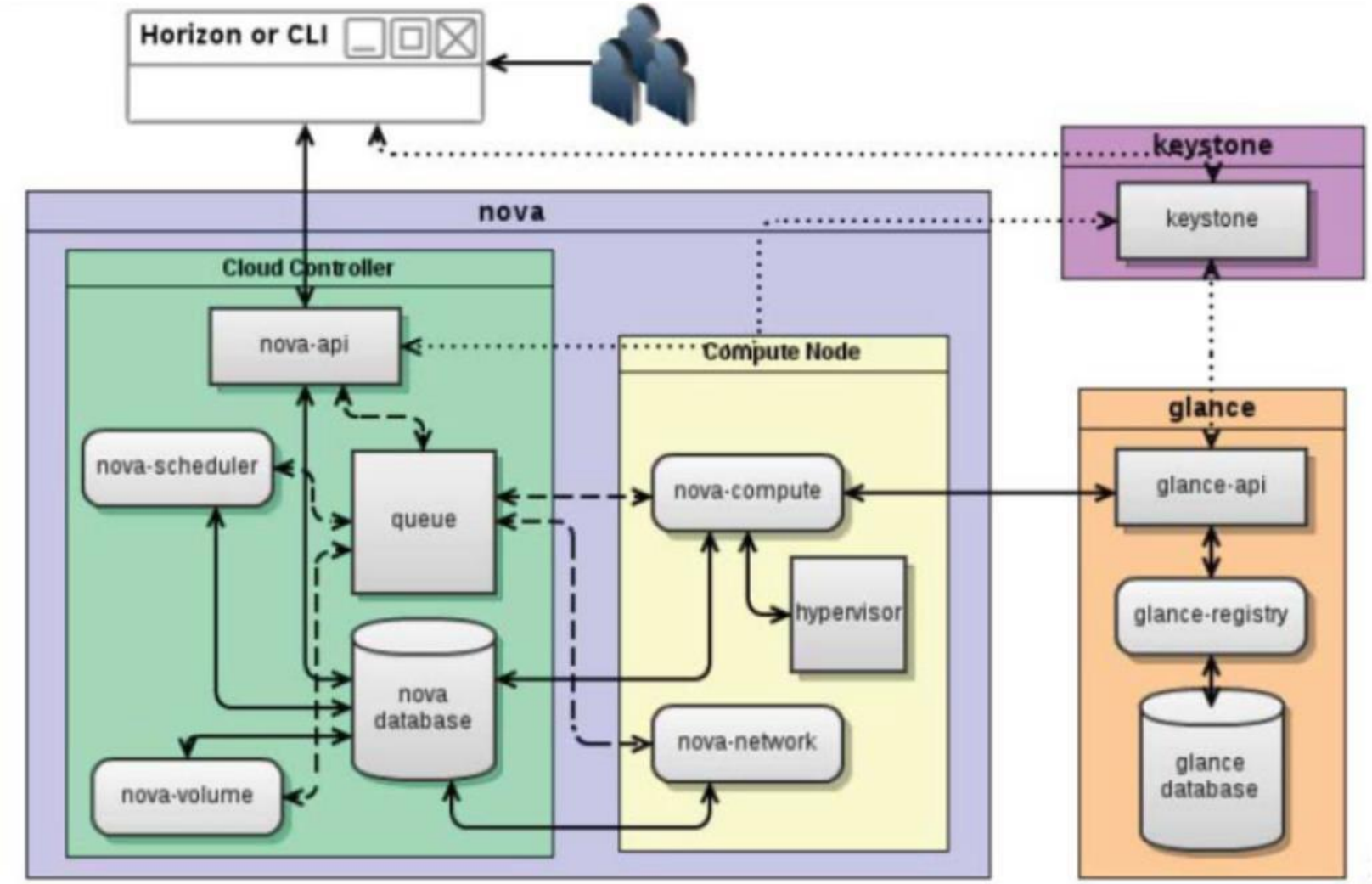
- 用户输入命令，api会查看这种类型的instance 是否达到最大值，给scheduler 发送一个消息（实际上是发送到 Queue 中）去运行这个实例。

Nova 核心模块，工作过程是什么



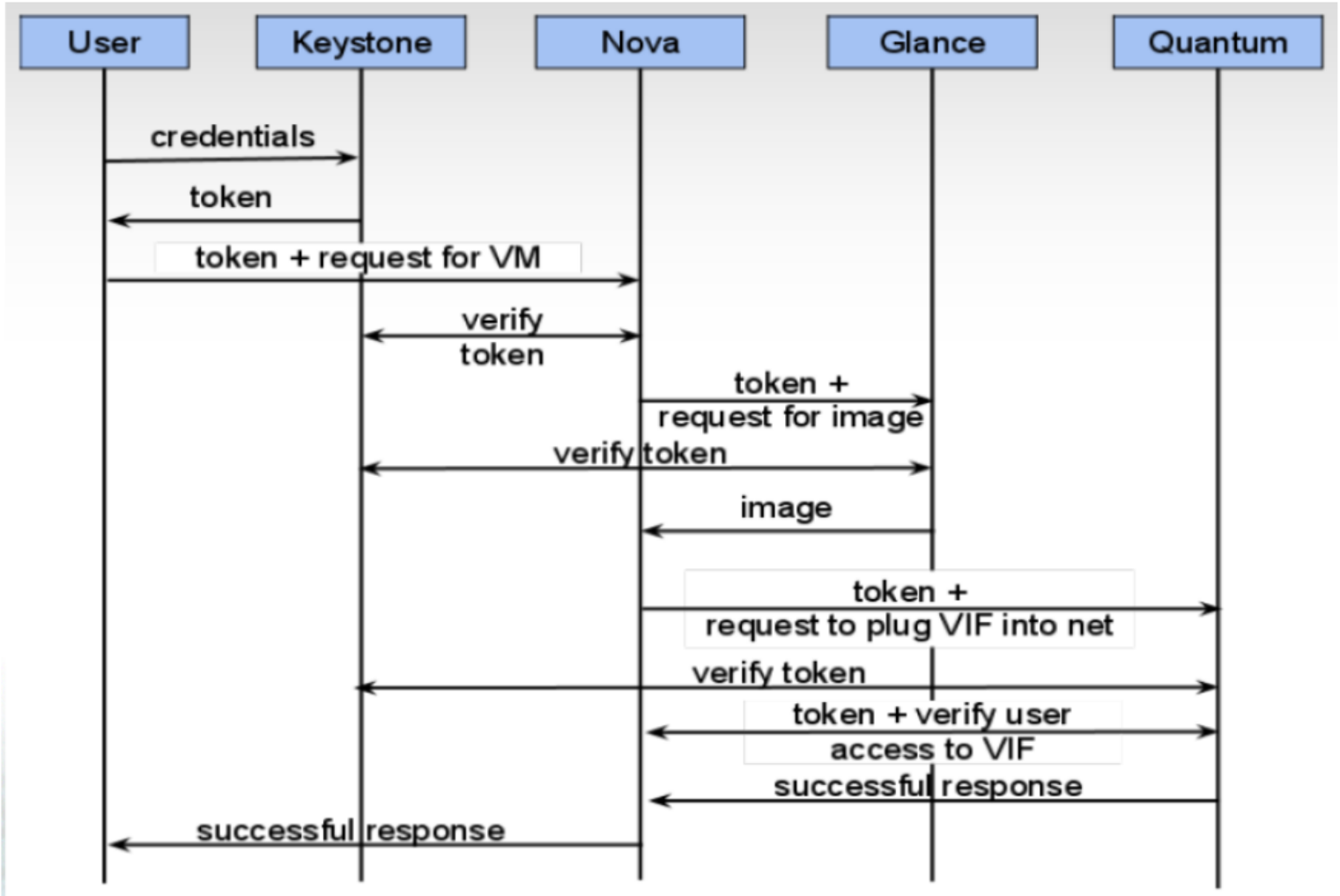
- 调度器接收到了消息队列 Queue 中 API 发来的消息，然后根据事先设定好的调度规则，选择好一个 host，之后，这个 instance 会在这个 host 上创建。

Nova 核心模块，工作过程是什么



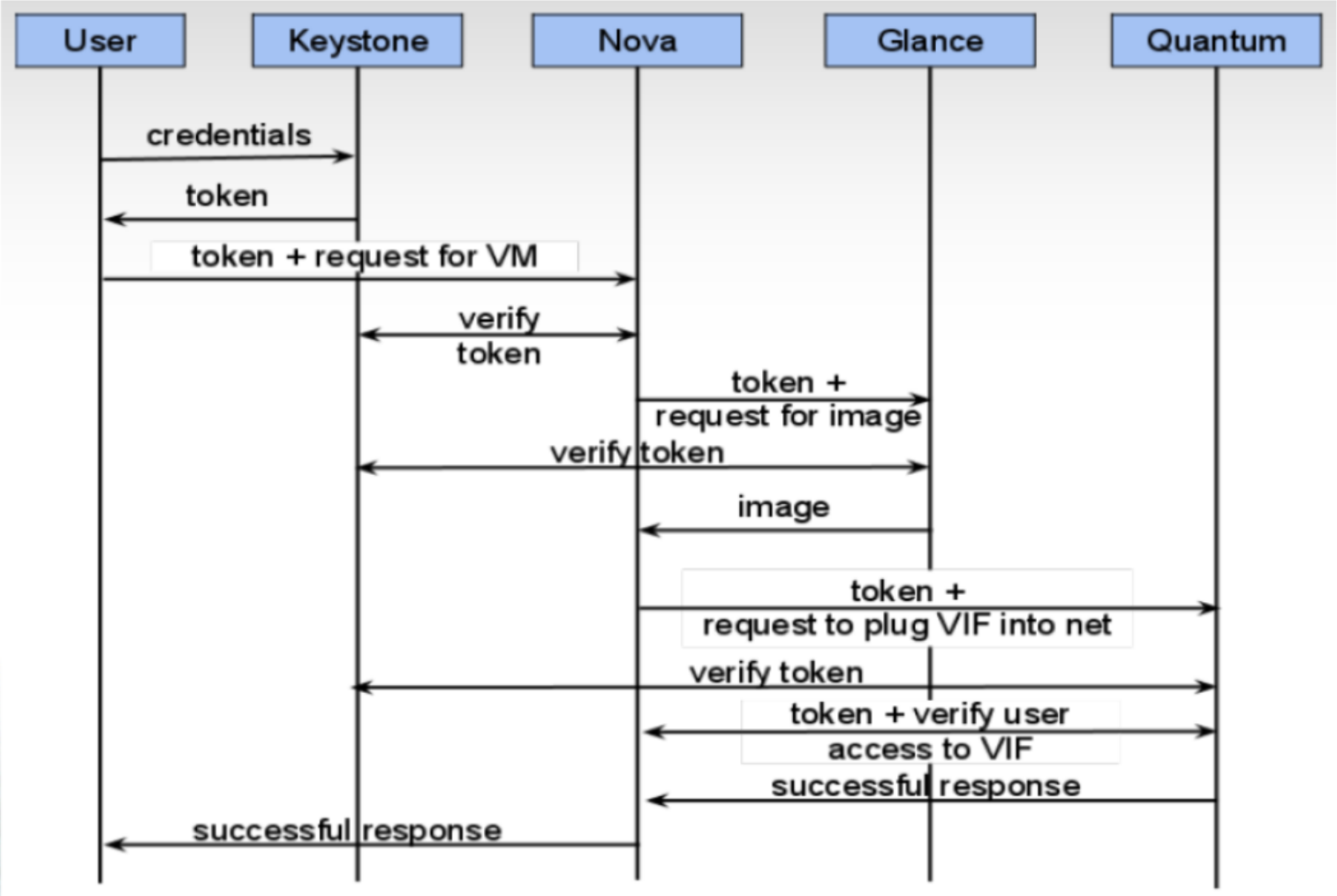
- 真正创建 instance 是由 compute 完成的，通过 glance 查找镜像。
- 根据找到的镜像，到 database 中查找相应的数据。
- volume 创建虚拟机实例的卷。
- network 为虚拟机分配 IP 等网络资源。

Keystone 权限控制过程



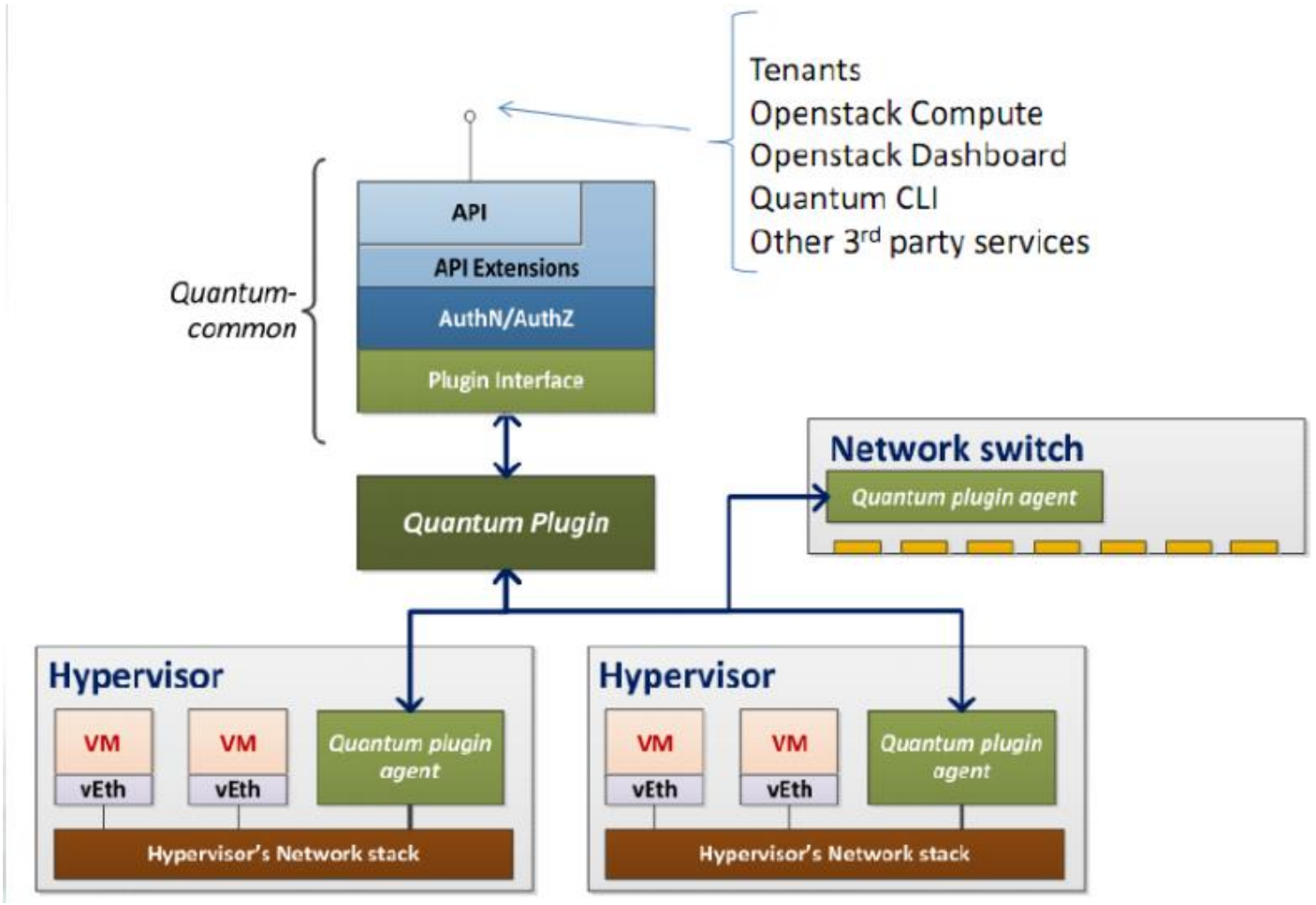
- 用户传 credential 给 keynote 进行请求，keynote 进行认证以后分配给用户一个 token（令牌），用户获得权限，将令牌和虚拟机请求传给 nova，nova 向 keystone验证令牌，获得权限后连同对镜像的请求传给 glance

Keystone 权限控制过程



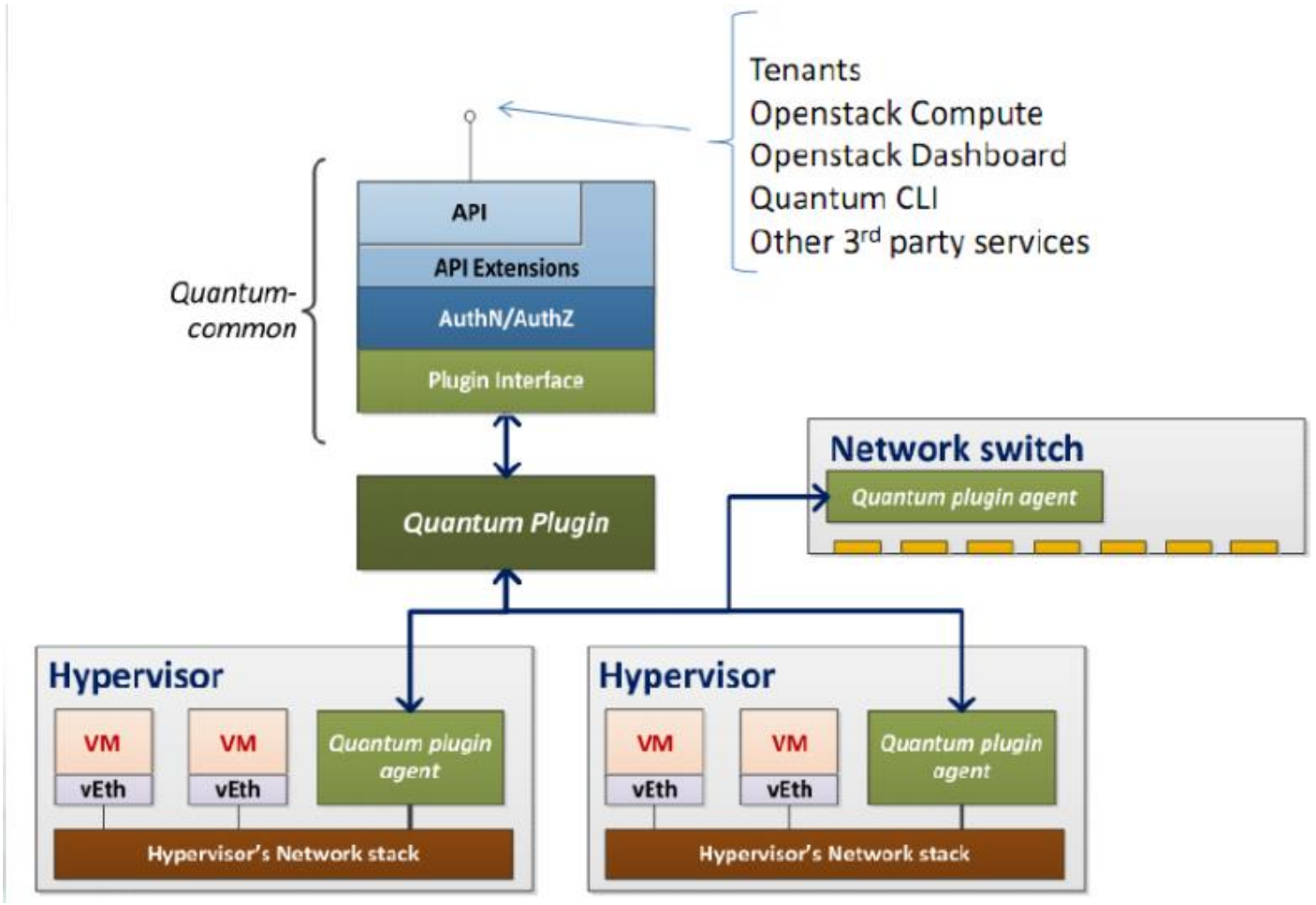
- glance 向 keynote 验证令牌，把镜像传给 nova；
- nova 再将用户接入网络的请求传给 quantum
- 验证成功后，即传出成功访问的回答

Quantum原理是什么？



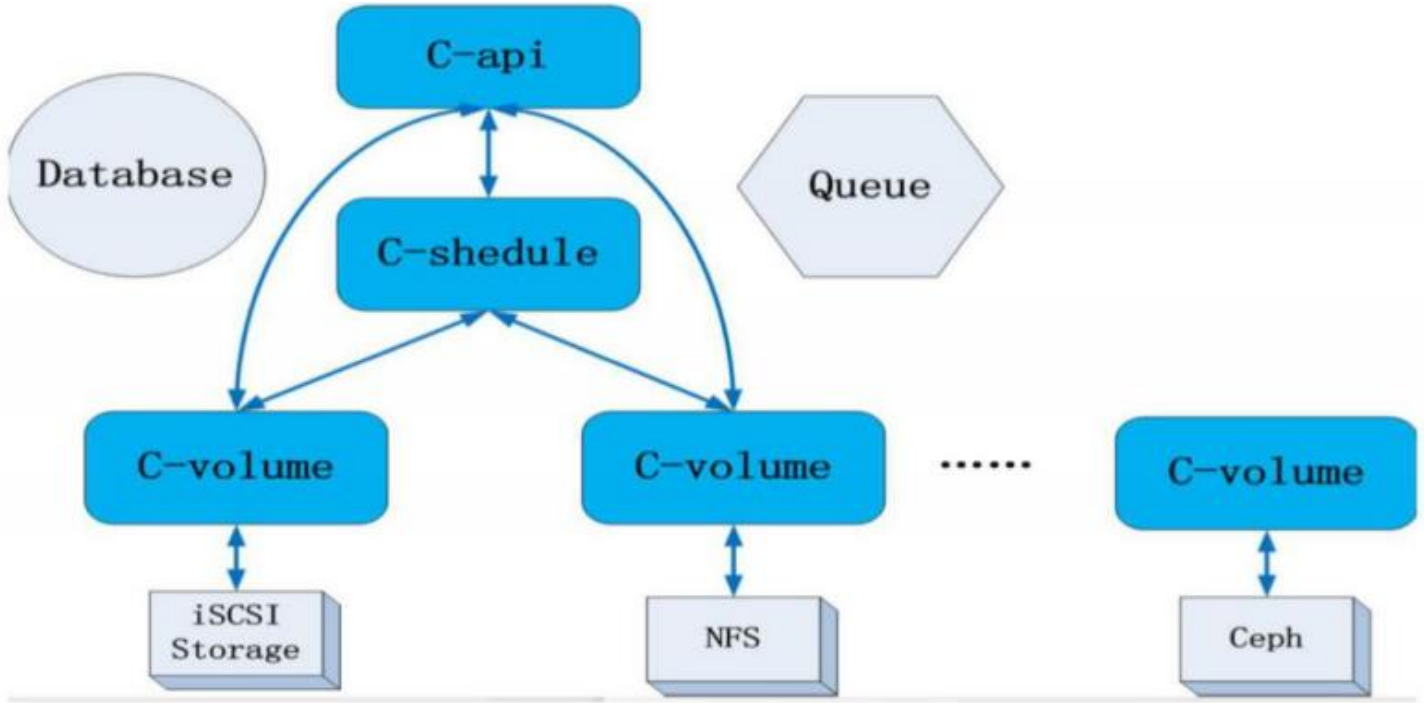
- neutron-server对外提供OpenStack网络API，接收请求，并调用plugin处理请求
- 插件(plug-ins)处理neutron-server发来的请求，维护OpenStack逻辑网络的状态，并调用agent处理请求
- 代理(agents)处理plugin的请求，负责在network provider上真正实现各种网络功能

Quantum原理是什么？



- 代理(agents)处理plugin的请求，负责在network provider上真正实现各种网络功能，比如端口插拔、创建网络或者子网、以及提供IP地址等

Cinder的存储机制是什么



- 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）

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

Swift 的组件有哪些及作用

- 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

第七讲 云存储

1. 大规模数据存储面临的新问题与挑战
2. 索引技术
3. GFS体系结构
4. 云存储应用的特点

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）

大规模数据存储面临的新问题与挑战

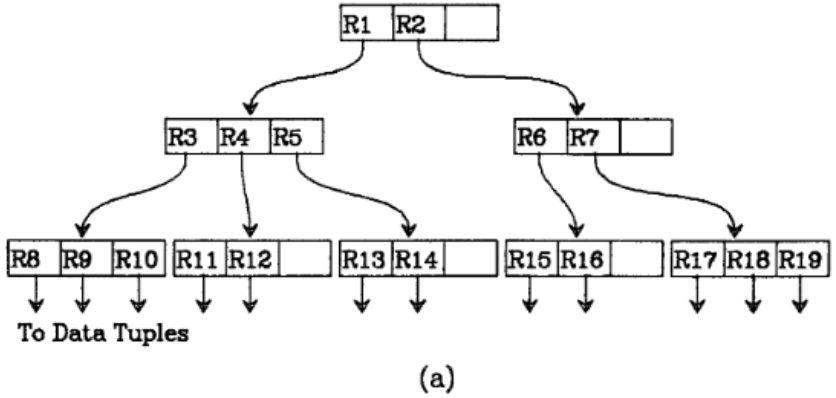
- 传统模式的挑战——核心问题是数据量与成本的矛盾
- 成本（处理能力扩充、人力等），效率（从降低的量变到不可用的质变），变更（业务逻辑），数据库的限制（表结构），其他（易用性、可靠性、安全性）

索引技术

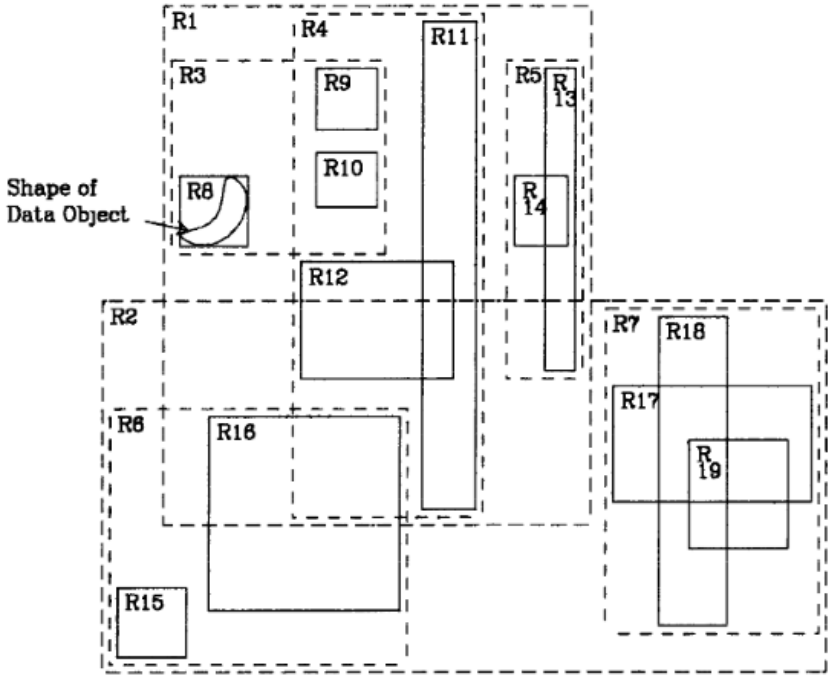
- BST：小数在左子节点，大数在右子节点。同一组数据可能形成不同的树。
- B 树：根节点至少有两个子节点、每个节点有 M-1 个 key，并且以升序排列、位于 M-1 和 M key 的子节点的值位于 M-1 和 M key 对应的 Value 之间、其它节点至少有 M/2 个子节点。
- B+树：有 k 个子结点的结点必然有 k 个关键码、非叶结点仅具有索引作用，跟记录有关的信息均存放在叶结点中、树的所有叶结点构成一个有序链表，可以按照关键码排序的次序遍历全部记录。

索引技术

- 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。



(a)



(b)

GFS体系结构

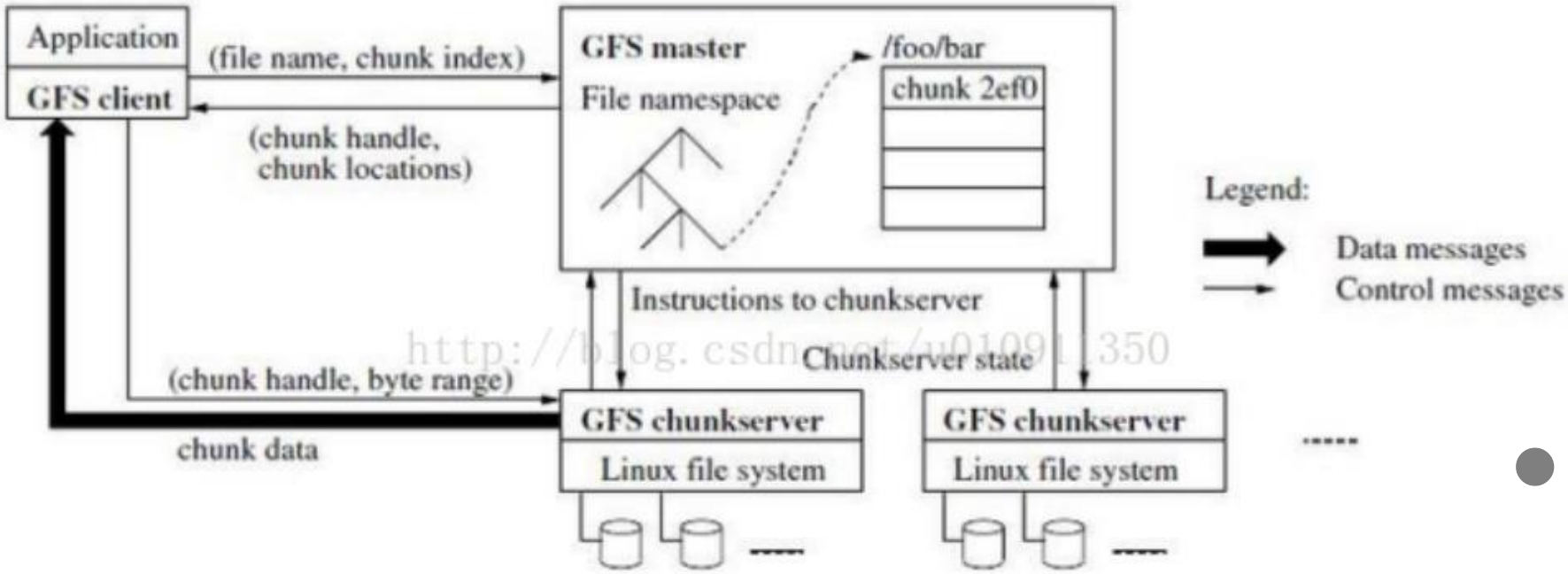


Figure 1: GFS Architecture

- 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。 以 linux 文件的形式保存在本地硬盘上

GFS体系结构

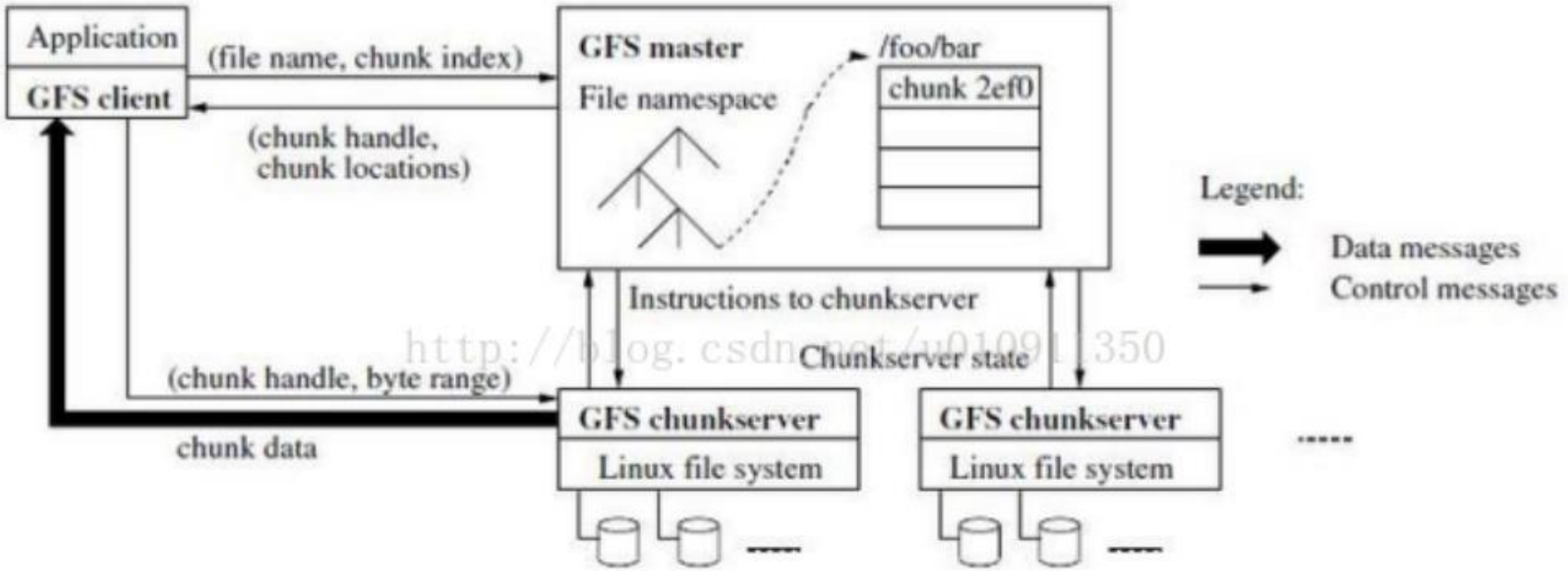


Figure 1: GFS Architecture

- 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。

GFS体系结构

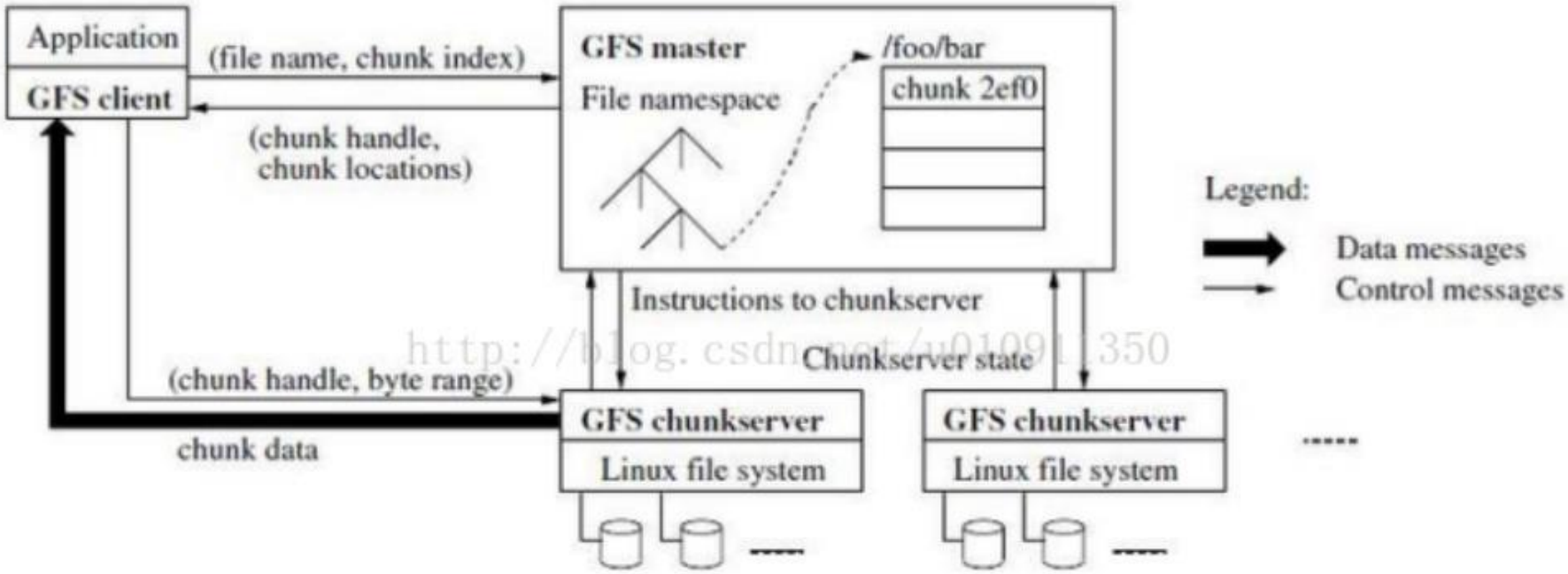


Figure 1: GFS Architecture

- Master 中维护了系统的元数据,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 的迁移。

GFS体系结构

- 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。。

GFS体系结构

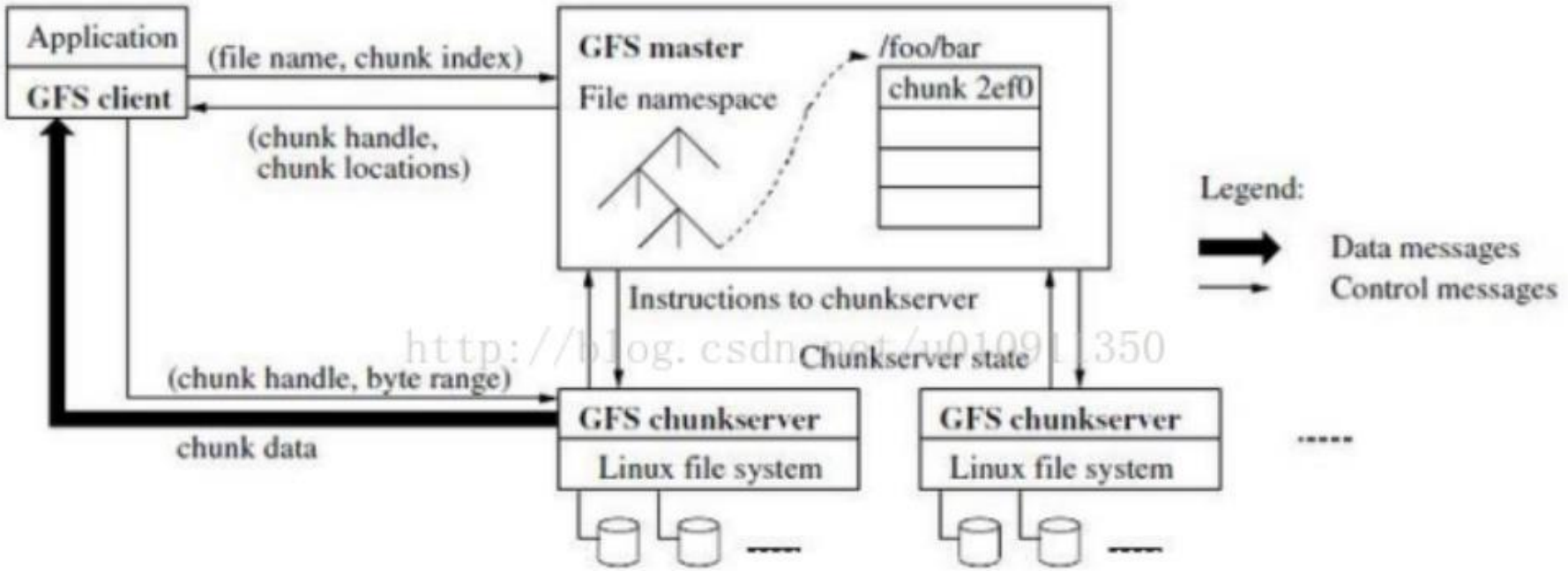


Figure 1: GFS Architecture

- 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。

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

第八讲 虚拟化资源管理

1. 并行化思想
2. 批量计算特点
3. Mapreduce算法的架构
4. Mapreduce算法设计思想
5. 算法调优
6. Mapreduce运行过程中的各种参数及其作用
7. 参数调优

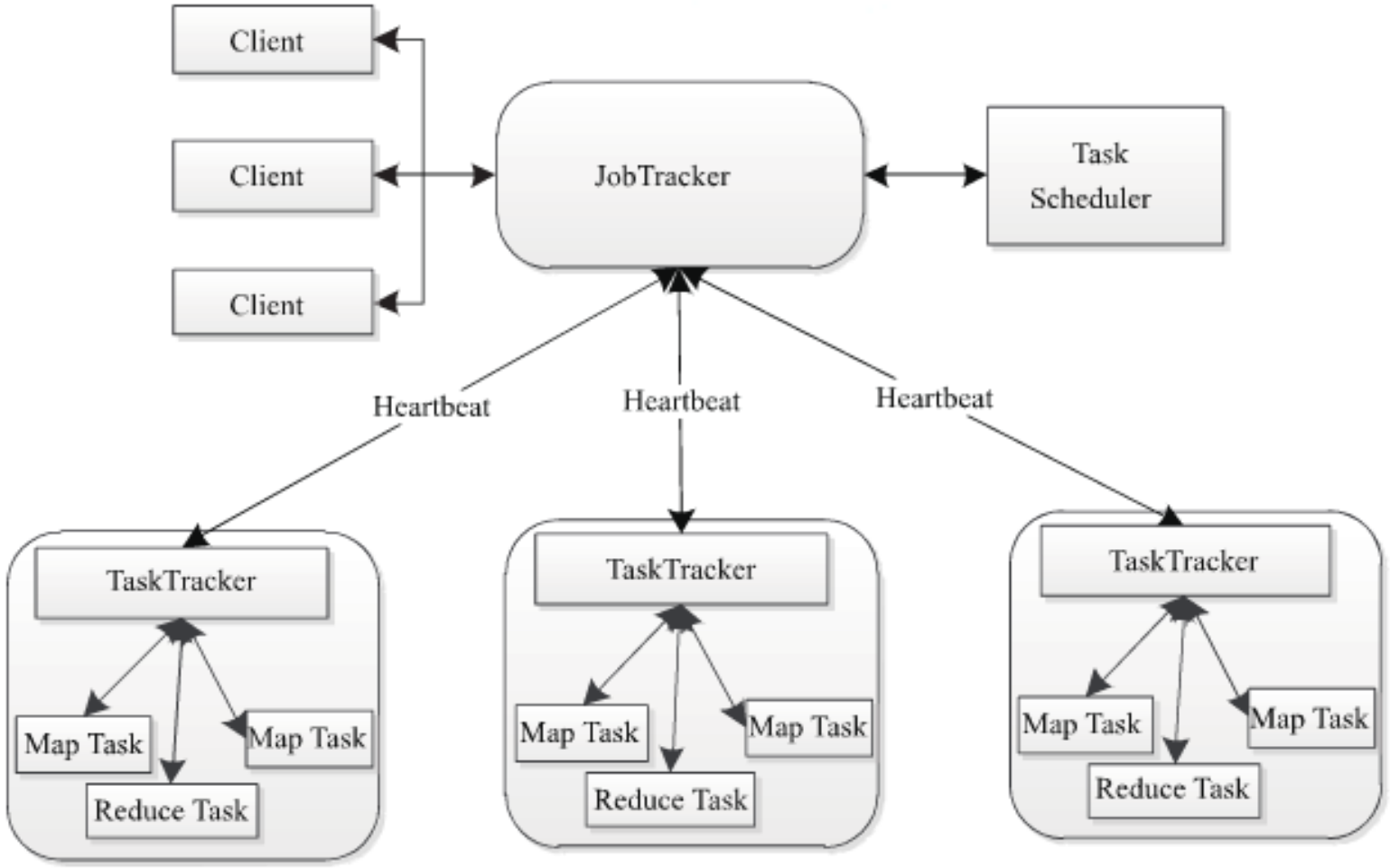
并行化思想

- 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。

批量计算特点

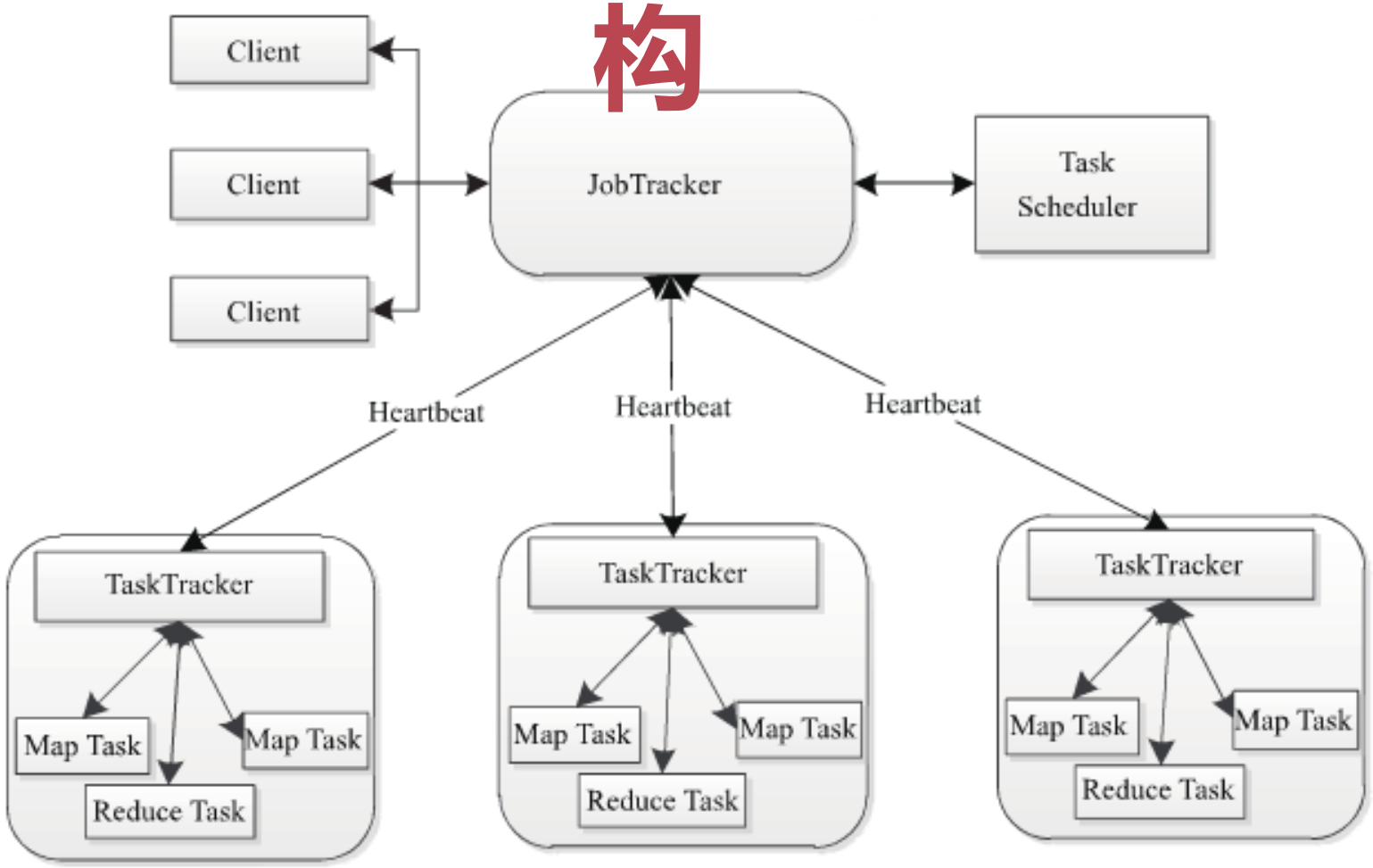
- 数据规模不会变化，由用户来驱动计算。
- 对计算的实时性要求不高，同时存在重复性问题，计算的逻辑相对简单
- 是一种批量、高时延、主动发起的计算

Mapreduce算法的架



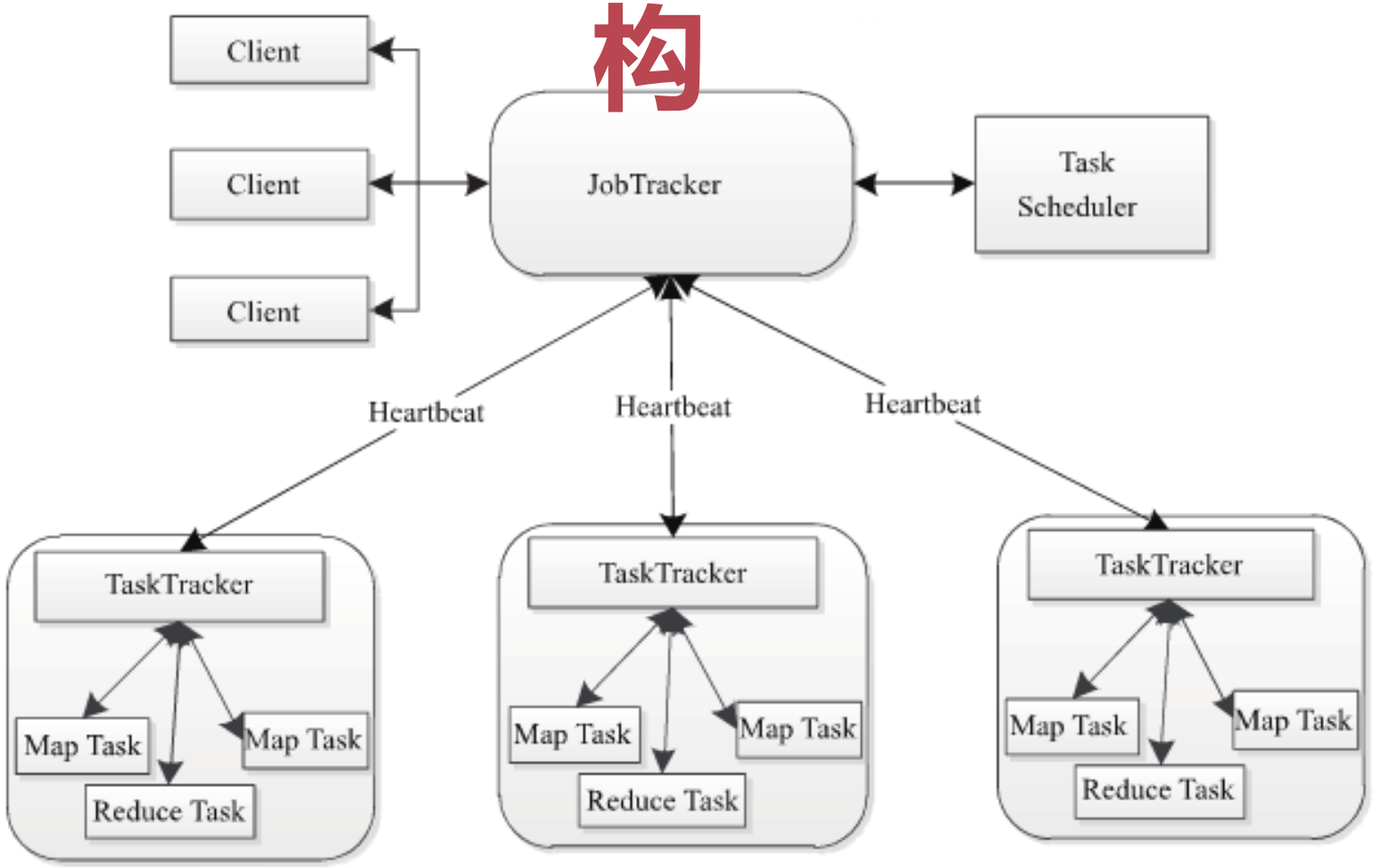
- client客户端
- 每一个Job都会在用户端通过Client类将应用程序以及参数配置Configuration打包成Jar文件存储在HDFS，并把路径提交到JobTracker的master服务，然后由master创建每一个Task（即MapTask和ReduceTask），将它们分发到各个TaskTracker服务中去执行

Mapreduce算法的架构



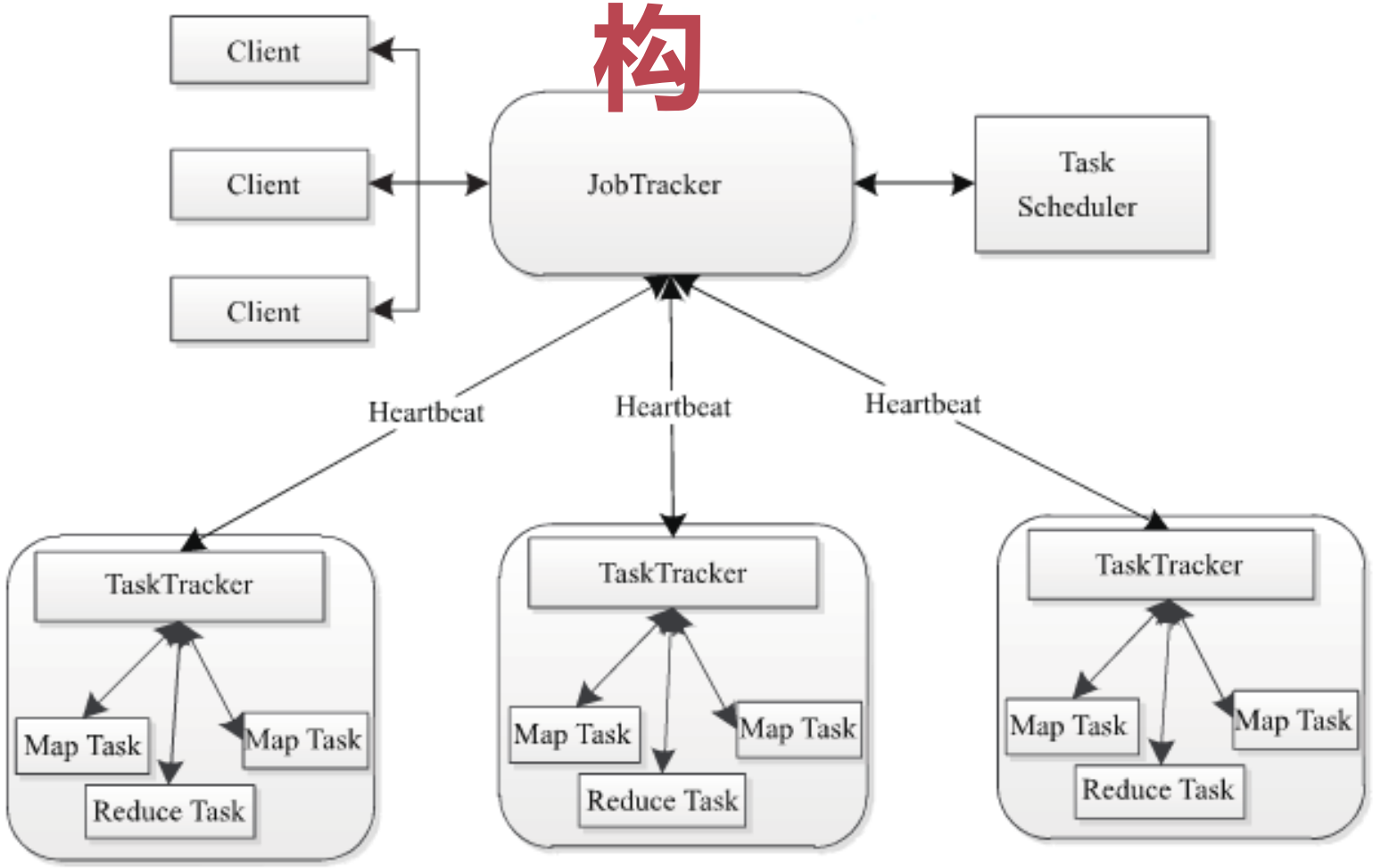
- JobTracker负责资源监控和作业调度。JobTracker监控所有的TaskTracker与job的状况，一旦发现失败，就将相应的任务转移到其它节点；同时JobTracker会跟踪任务的执行进度，资源使用量等信息，并将这些信息告诉任务调度器，调度器会在资源出现空闲时，选择合适的任务使用这些资源。任务调度器是一个可插拔的模块，可以根据自己的需要设计相应的调度器

Mapreduce算法的架构



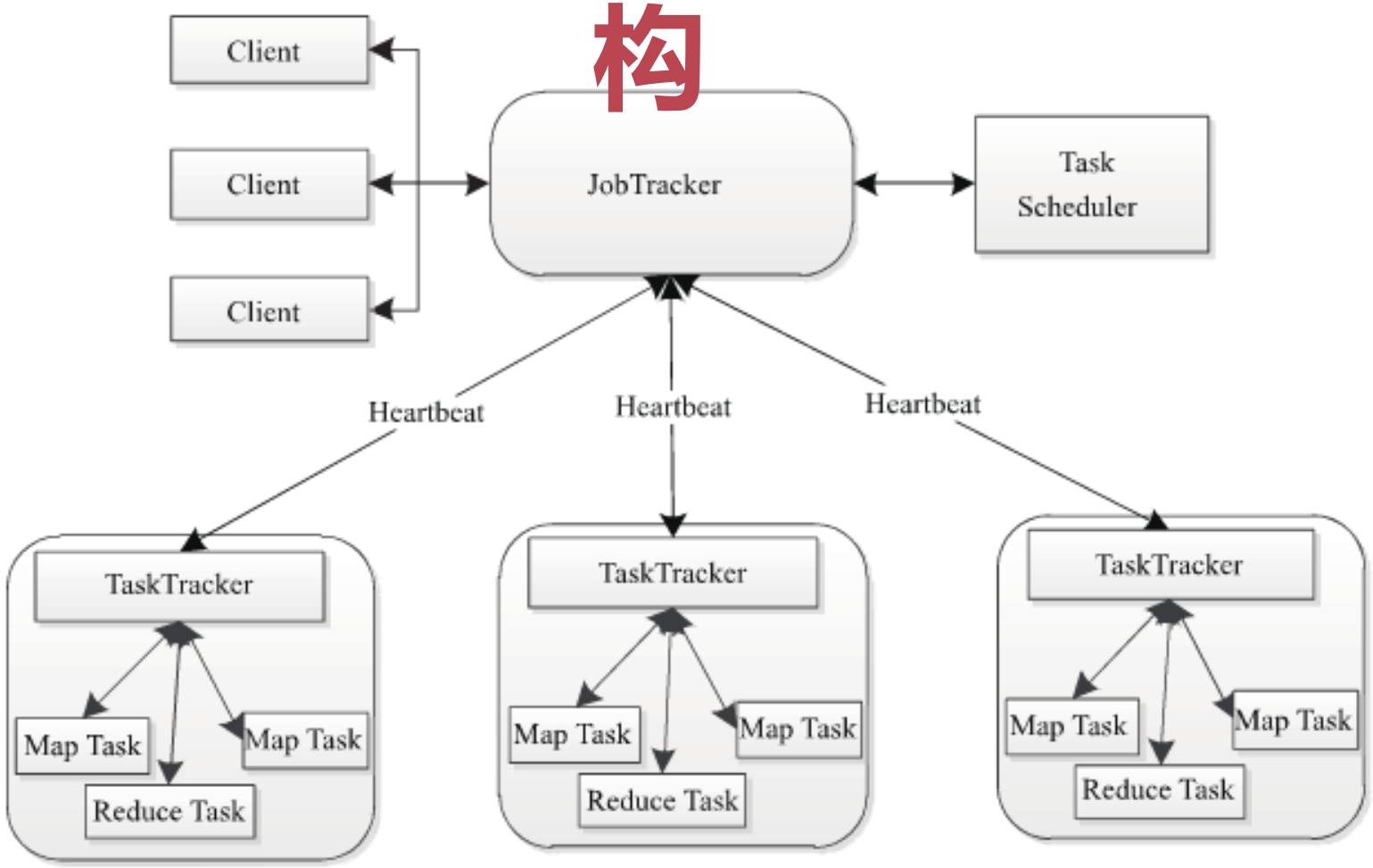
- TaskTracker会周期性地通过HeartBeat将本节点上资源的使用情况和任务的运行进度汇报给JobTracker，同时执行JobTracker发送过来的命令 并执行相应的操作（如启动新任务，杀死任务等）。TaskTracker使用“slot”等量划分本节点上的资源量。“slot”代表计算资源（cpu，内存等）

Mapreduce算法的架构



- 一个Task获取到一个slot之后才有机会运行，而Hadoop调度器的作用就是将各个TaskTracker上的空闲slot分配给Task使用。slot分为MapSlot和ReduceSlot两种，分别提供MapTask和ReduceTask使用。TaskTracker通过slot数目（可配置参数）限定Task的并发度。

Mapreduce算法的架构



- Task分为MapTask和Reduce Task两种，均由TaskTracker启动。HDFS以固定大小的block为基本单位存储数据，而对于MapReduce而言，其处理单位是split。split是一个逻辑概念，它只包含一些元数据信息，比如数据起始位置、数据长度等。它的划分方法完全由用户自己决定。

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

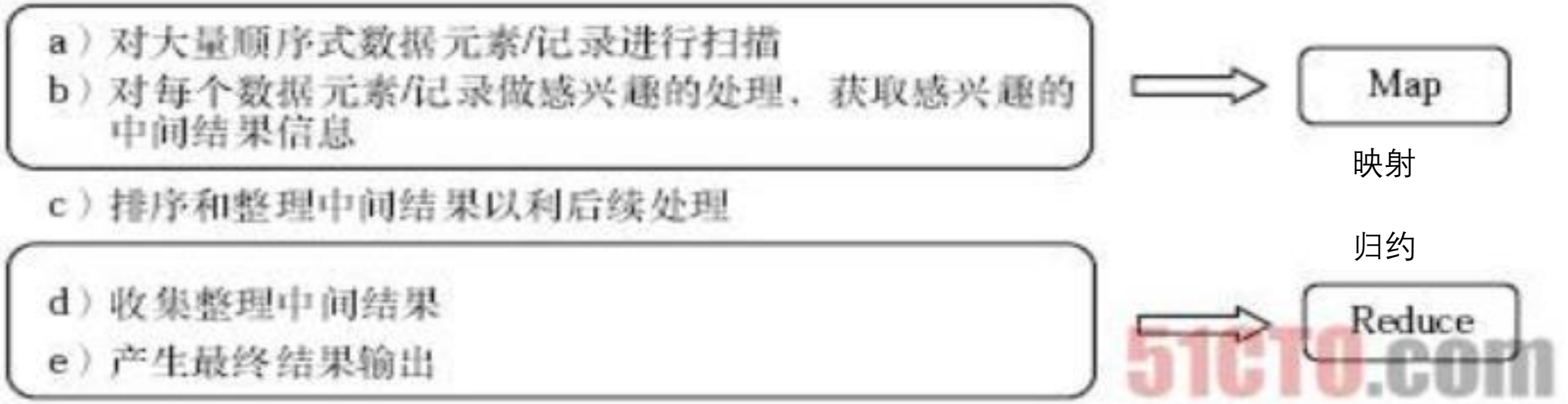


图 1-13　典型的顺序式大数据处理过程和特征

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）。

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。
- Io.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。

Mapreduce 运行过程中的各种参数及其作用

- 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。
- Io.sort.factor(合并因子)。将副本合并成更大的排好序的文件。
- 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。
- Io.file.buffer.size(hadoop 文件缓冲区)。Reduce 的输出结果写入 hdfs 系统

参数调优的原则

- 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

溢写（Spill）：每个map task都有一个内存缓冲区，存储着map的输出结果，这个内存缓冲区是有大小限制的，默认是100MB。当map task的输出结果很多时，就可能会撑爆内存，所以需要在一定条件下将缓冲区中的数据临时写入磁盘，然后重新利用这块缓冲区

第九讲 虚拟化资源管理

1. 流式计算与批量计算的区别
2. 流式计算的关键要素
3. 数据分发机制
4. 运用流式计算方法解决实际问题
5. 图的切分方式
6. BSP计算模式
7. 图数据计算的并行思想
8. 运用图数据计算方法解决实际问题

流式计算与批量计算的区别

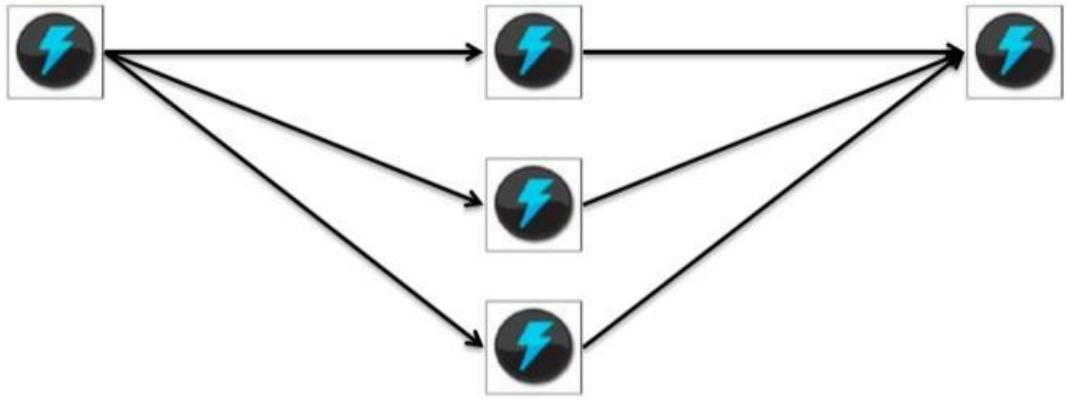
批量计算	流式计算
离线计算、存在延时	实时计算、延时较少
先将数据保存起来，然后处理	来一个处理一个
用户驱动计算请求	数据驱动计算请求
拉式获取计算结果	推式获取计算结果

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

数据分发机制

– 增加功能相同的节点，使用随机分发

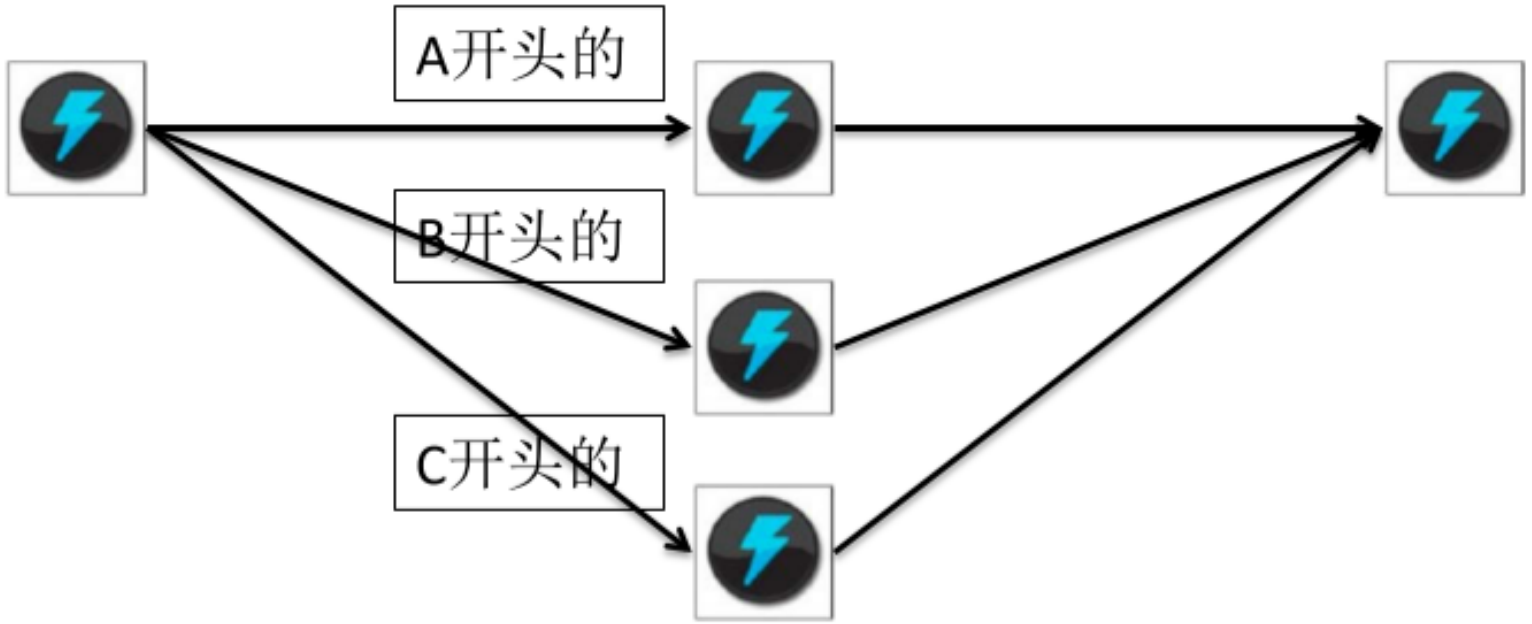


各节点处理逻辑相同，仅需保证数据量的均衡分配，采用发牌式分发

- 核心目的是为了解决节点处理压力问题
- 将单一节点的处理压力分散到多个节点上
- 关键是如何保证流经多个节点与流经一个节点的处理逻辑不变

数据分发机制

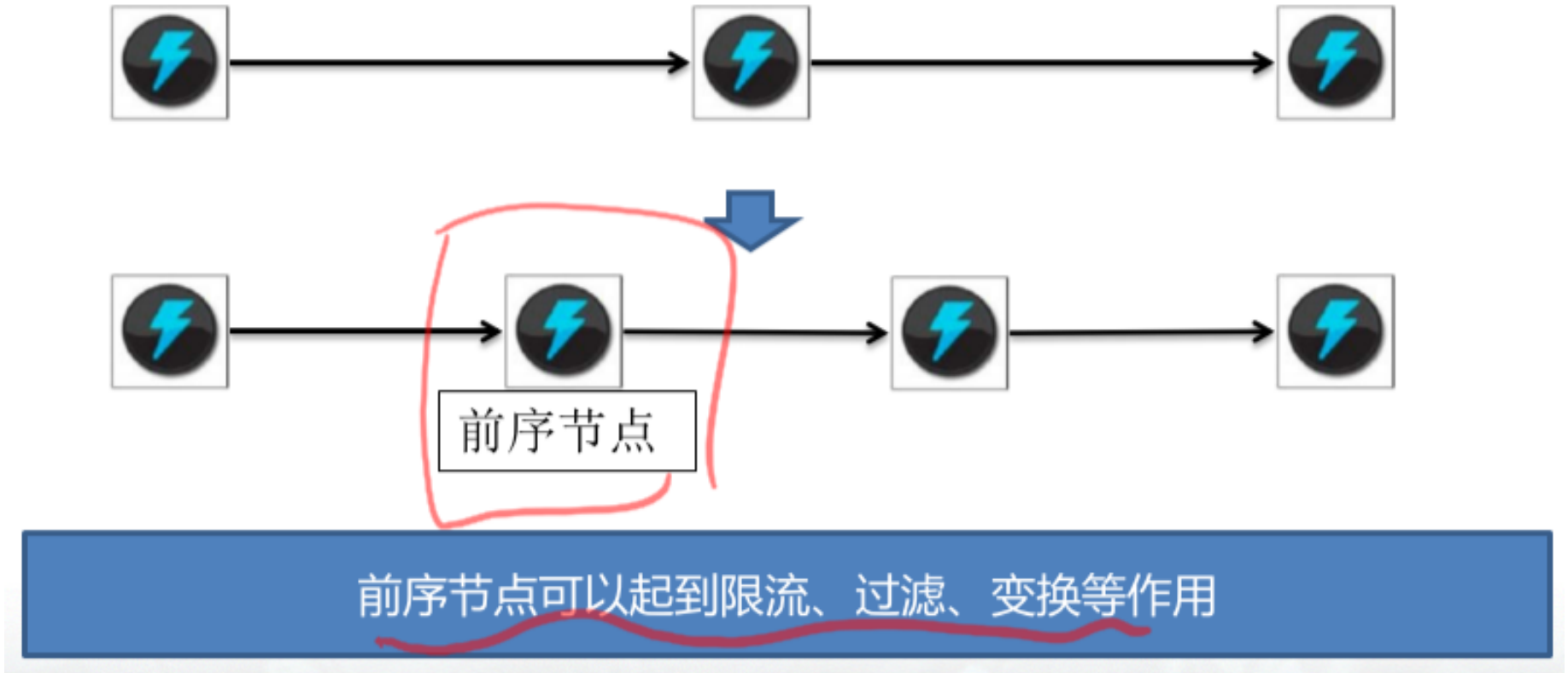
– 将处理功能分散，使用特定值分发



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

数据分发机制

– 增加前序节点，在处理前对数据进行某种转换



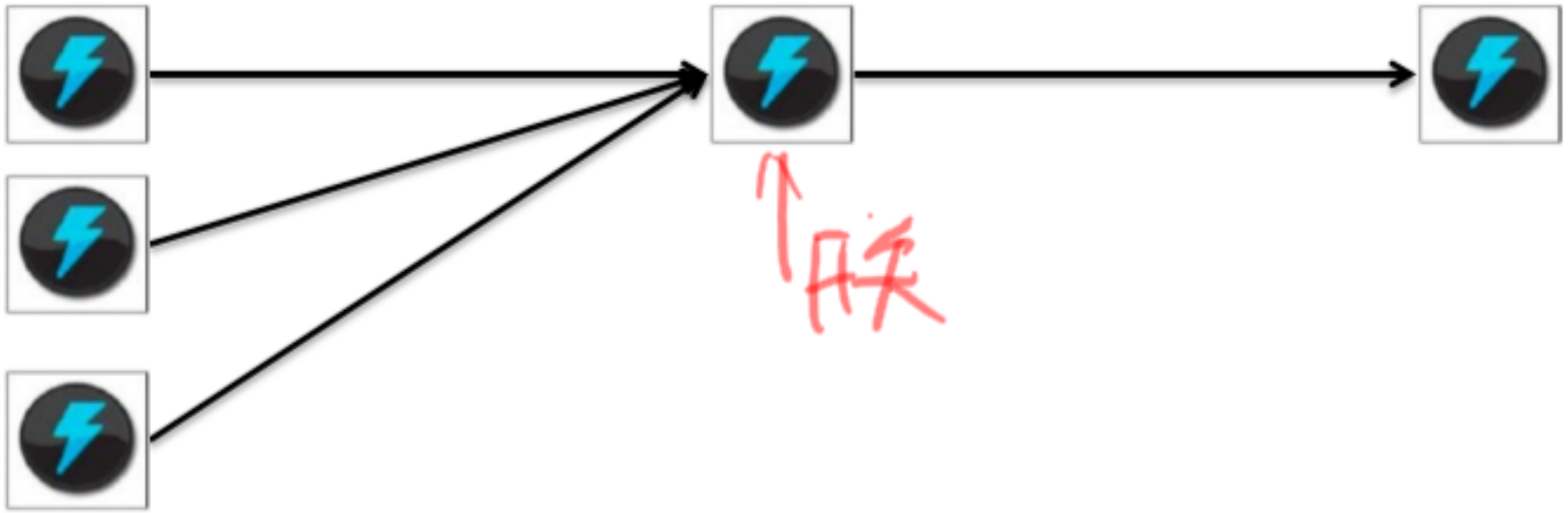
数据分发机制

- 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

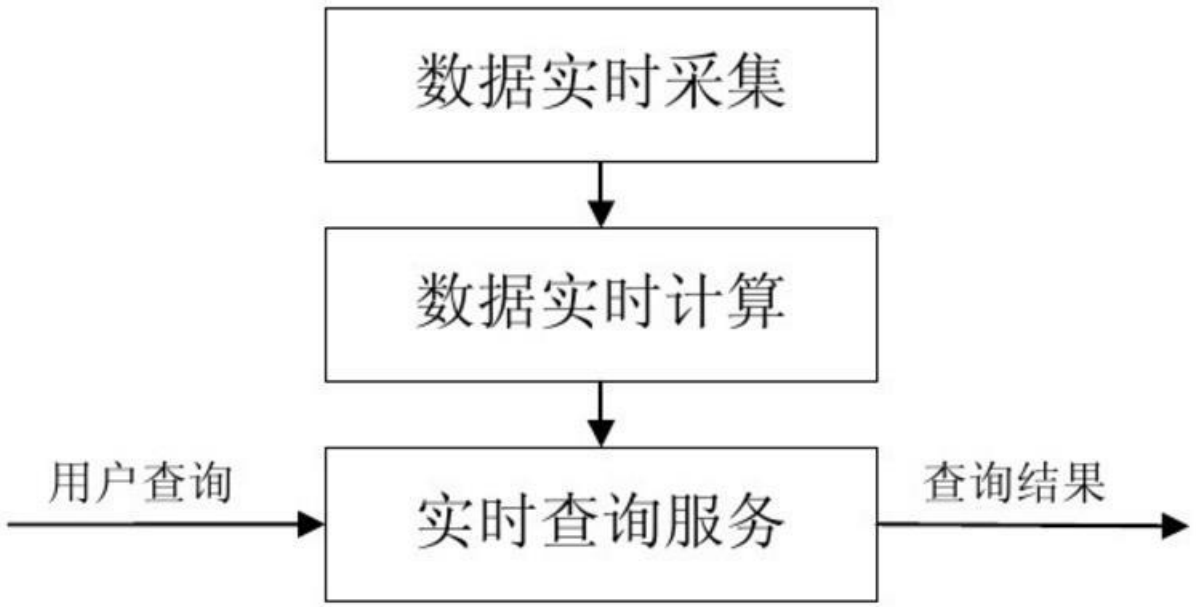
数据分发机制

– 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



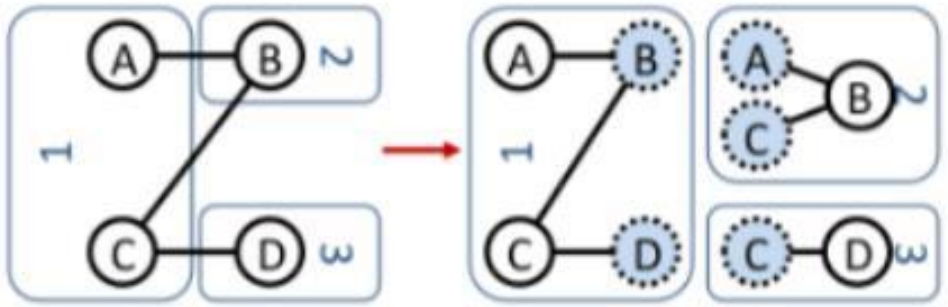
同步节点实际上起到了数据开关的作用，控制管道开关

运用流式计算方法解决实际问题



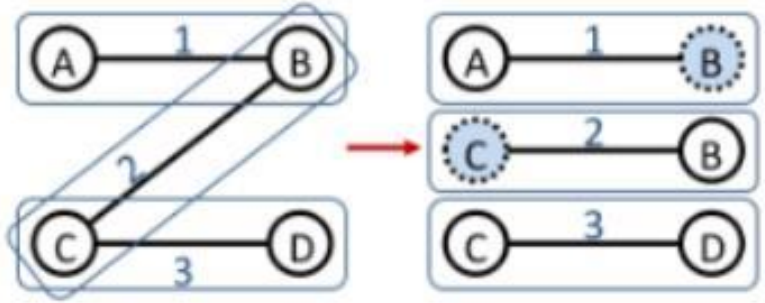
- 数据采集：获取数据
- 数据计算：处理数据
- 数据查询：提供结果

图的切分方式



(a) Edge-Cut

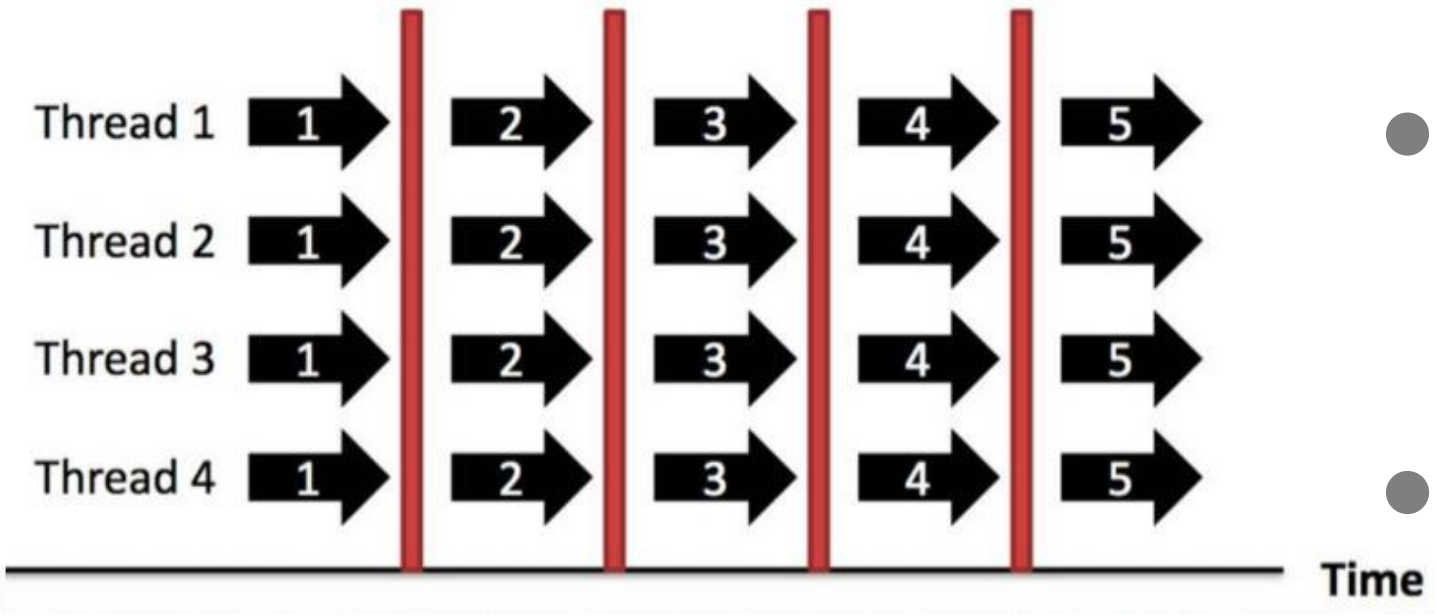
上下两个虚线圈代表了不同的涵义



(b) Vertex-Cut

- 边切分：每个点vertex的数据都只存储在一台机器上，如例子中的点A，点A的完整数据只存放在机器1上，B的完整数据只存放在机器2上这样存储的好处是对于读取某个点的数据时，只要到一台机器上就可以了。
- 点切分：这种存储方式特点是任何一条边只会出现在一台机器上，每个点有可能分布到不同的机器上，例如上图的点B就被分配到了2，3两条机器上。当点被分割到不同机器上时，是相同的镜像，但是有一个点作为主点(master),其他的点作为虚点(ghost)，

BSP 计算模式



- Bulk Synchronous Parallell 整体同步并行
- 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

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

运用图数据计算方法解决实际问题

- 并行 dijkstra 算法
- 图的拆分：键值对
- – ID；distance，color，weight
    - ID：顶点
    - Distance：距离，MAX表示无穷大
    - Color：着色

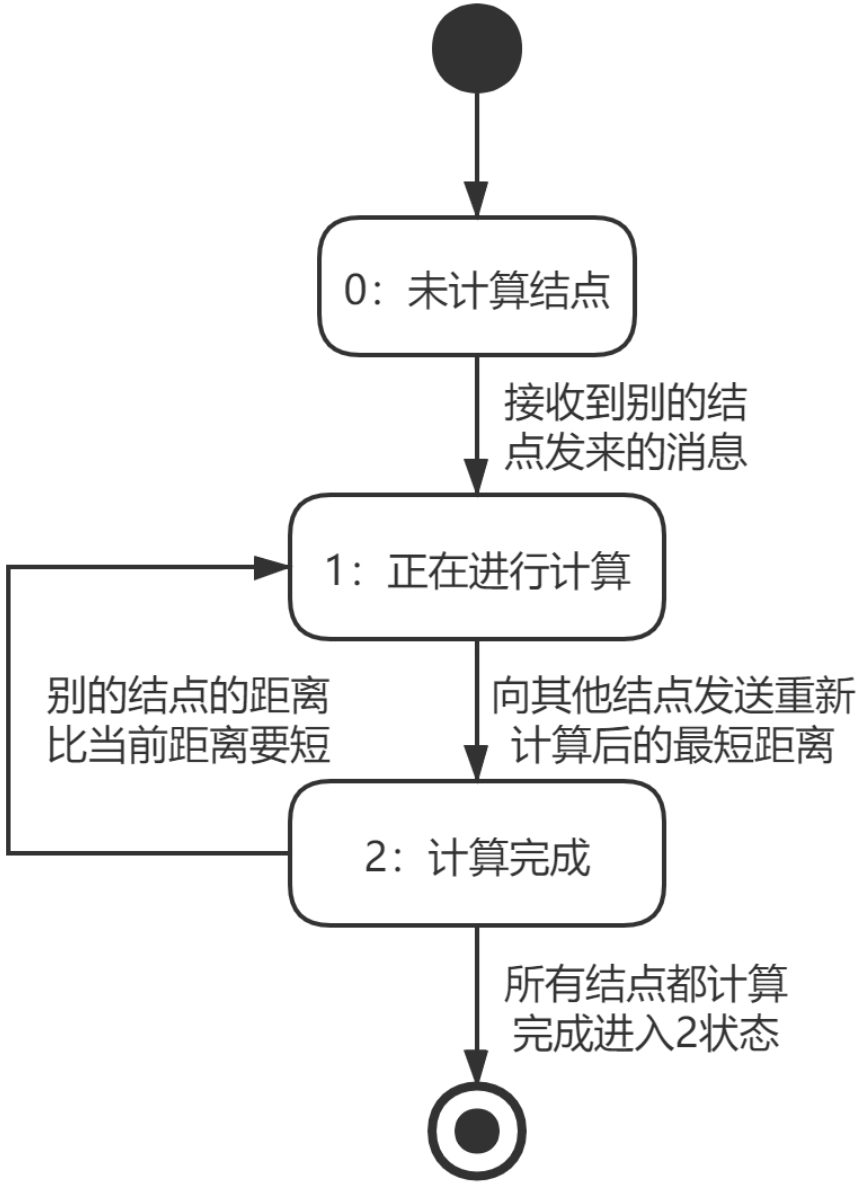
0：白色，未被计算的节点，即未连通的节点，距离为MAX

1：灰色，计算过程中节点

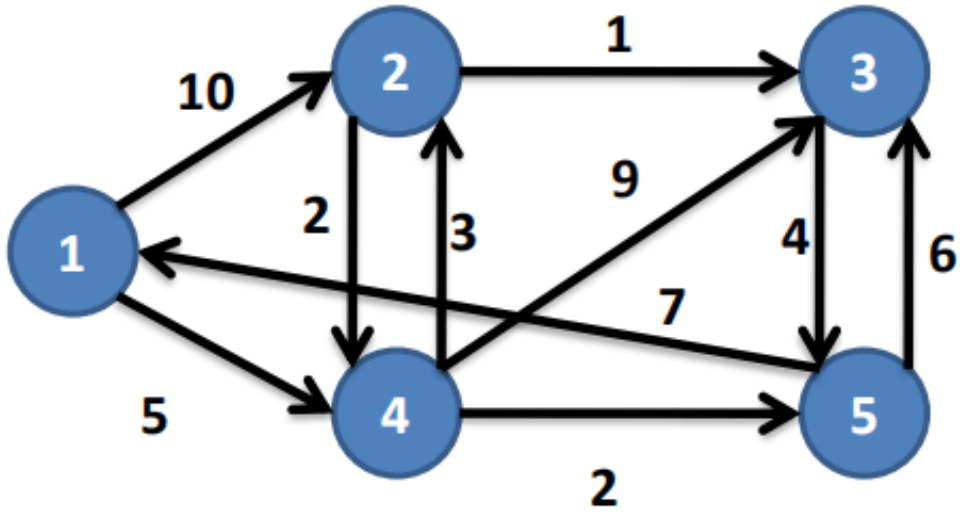
2：黑色，已经完成最短路径计算的节点

并行dijkstra算法

1	0；1；2、10，4、5
2	MAX；0；3、1，4、2
3	MAX；0；5、4
4	MAX；0；2、3，3、9，5、2
5	MAX；0；3、6，1、7



并行 dijksta 算法



1	0；1；2、10，4、5
2	MAX；0；3、1，4、2
3	MAX；0；5、4
4	MAX；0；2、3，3、9，5、2
5	MAX；0；3、6，1、7

并行dijkstra算法

- 迭代一次

1	0；2；2、10，4、5
2	10；1
4	5；1
2	MAX；0；3、1，4、2
3	MAX；0；5、4
4	MAX；0；2、3，3、9，5、2
5	MAX；0；3、6，1、7

1	0；1；2、10，4、5
2	MAX；0；3、1，4、2
3	MAX；0；5、4
4	MAX；0；2、3，3、9，5、2
5	MAX；0；3、6，1、7

1	0；2；2、10，4、5
2	10；1；3、1，4、2
3	MAX；0；5、4
4	5；1；2、3，3、9，5、2
5	MAX；0；3、6，1、7

并行dijkstra算法

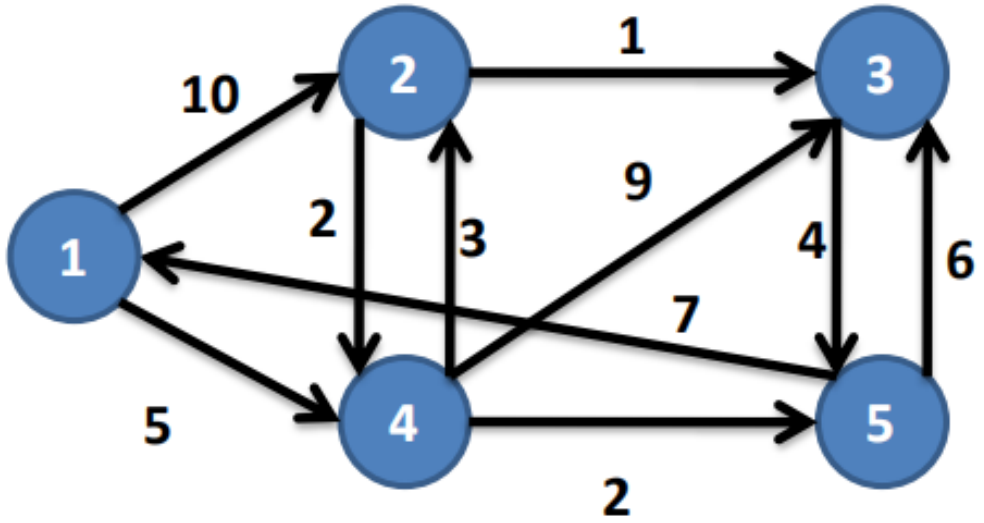
- 迭代两次

1	0；2；2、10，4、5
2	10；2；3、1，4、2
3	11；1
4	12；1
2	8；1
3	14；1
5	7；1
3	MAX；0；5、4
4	5；2；2、3，3、9，5、2
5	MAX；0；3、6，1、7

1	0；2；2、10，4、5
2	10；1；3、1，4、2
3	MAX；0；5、4
4	5；1；2、3，3、9，5、2
5	MAX；0；3、6，1、7

1	0；2；2、10，4、5
2	8；1；3、1，4、2
3	11；1；5、4
4	5；2；2、3，3、9，5、2
5	7；1；3、6，1、7

并行dijkstra算法



1	0；2；2、10，4、5
2	8；2；3、1，4、2
3	9；2；5、4
4	5；2；2、3，3、9，5、2
5	7；2；3、6，1、7

第十一讲 分布式计算框架

1. Hadoop项目的由来
2. HDFS的体系结构
3. HDFS的运行机制
4. Hadoop中MapReduce的实现机制
5. Htable的数据结构
6. Hbase的运行机制
7. Yarn对Hadoop的核心改进
8. Spark架构及运行机制

第十一讲 分布式计算框架

9 Storm架构及运行机制

10 Kafka架构及运行机制

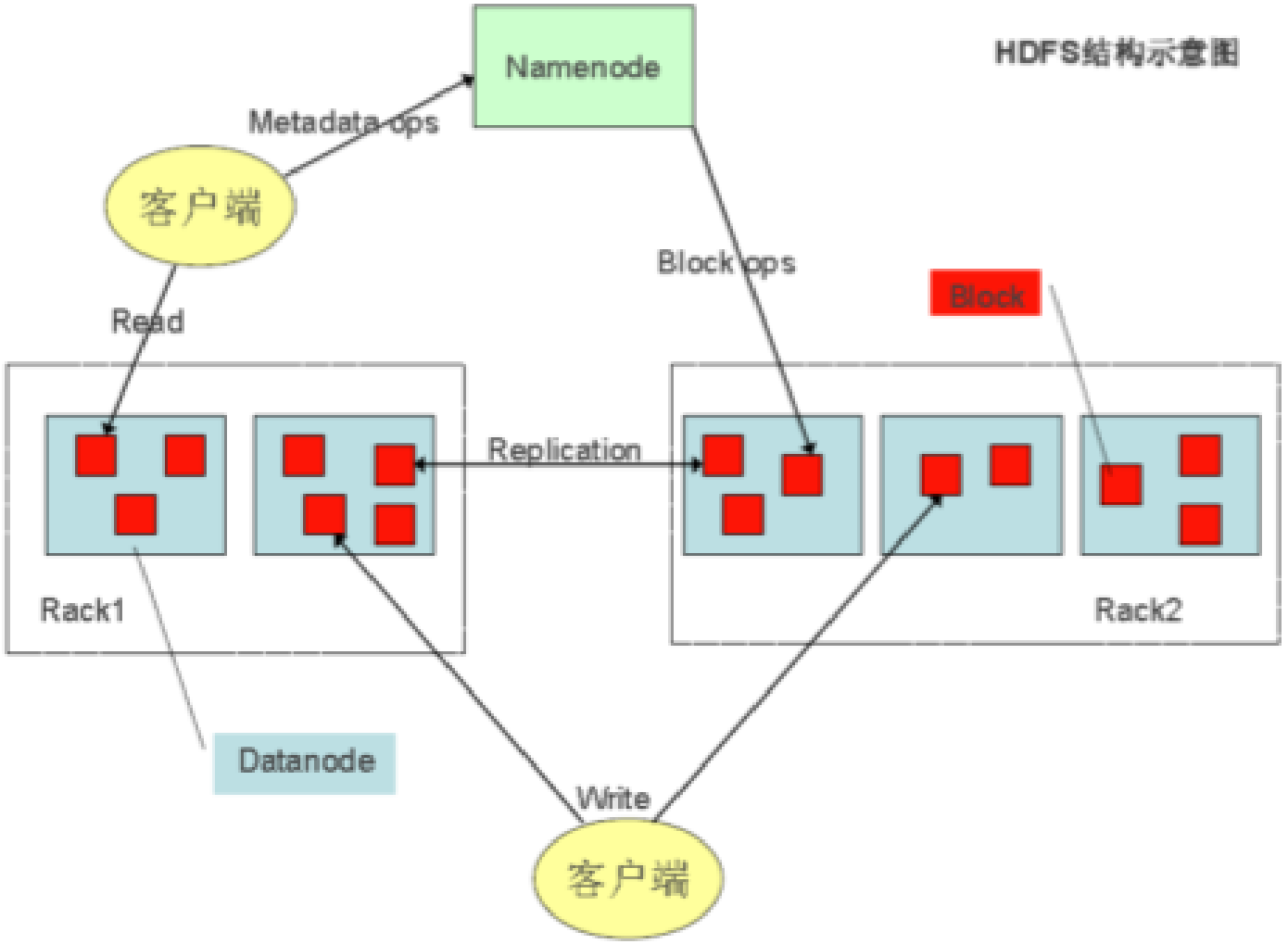
11 Pregel架构及运行机制

12 各种分布式处理框架的异同点

Hadoop项目的由来

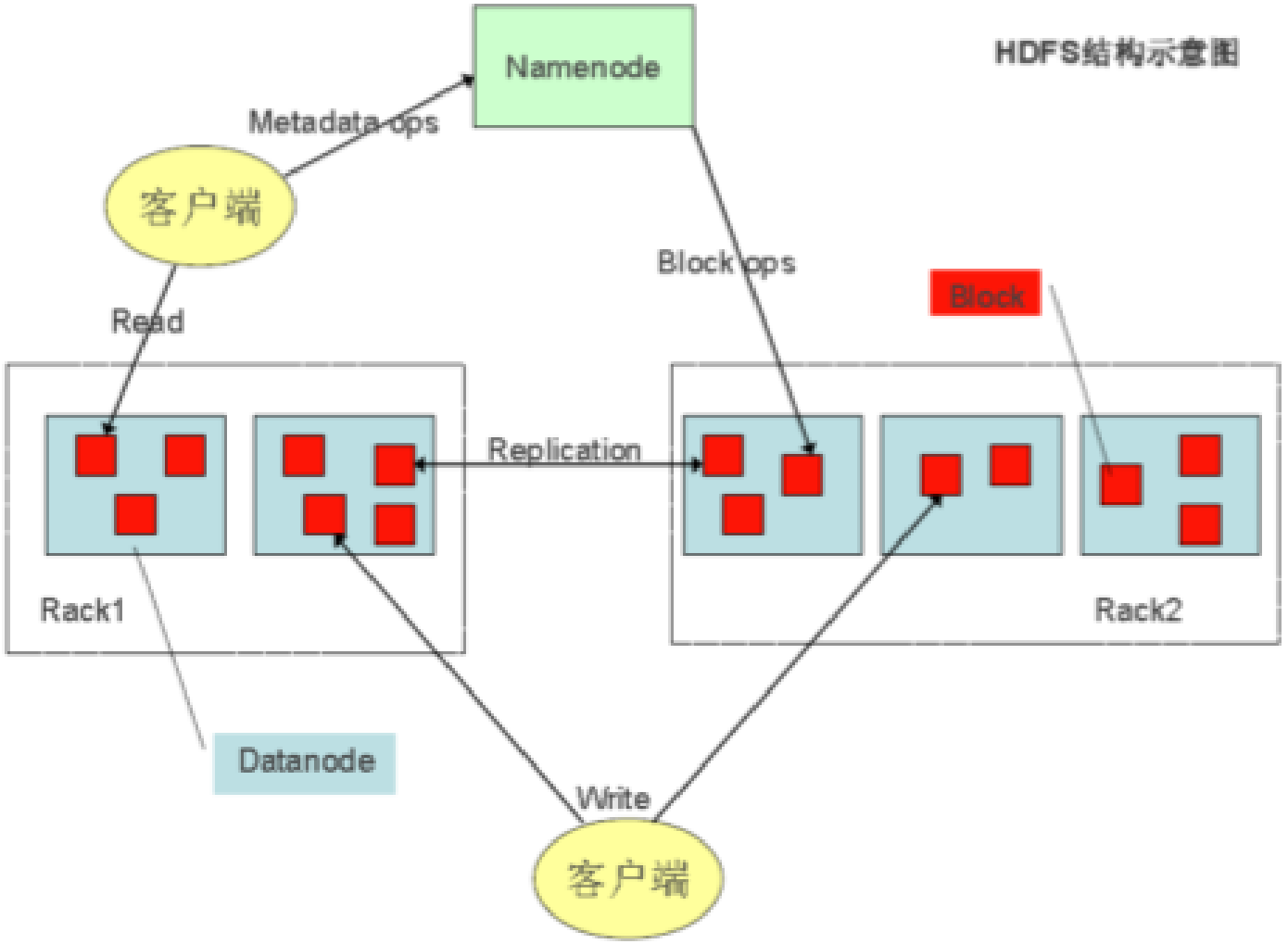
- Hadoop 基于网络搜索引擎 nutch，由于缺乏可扩展的架构，借鉴 GFS 实现了 NDFS，再加入 mapreduce 系统后形成 hadoop。

HDFS的体系结构



- NameNode，用来管理文件系统的命名空间，维护整个系统的目录树和文件索引目录；并以2种形式存储在NameNode上，一个是Namespace image(命名空间镜像)，一个是Edit log（编辑日志）

HDFS的体系结构



- DataNode，是数据/文件块存储的节点，这些节点可以有多个，并且通过心跳机制向NameNode传送所存储的块的信息。
- Secondary NameNode，这类节点是对NameNode的一种补充，对Namespace Image和Edit Log

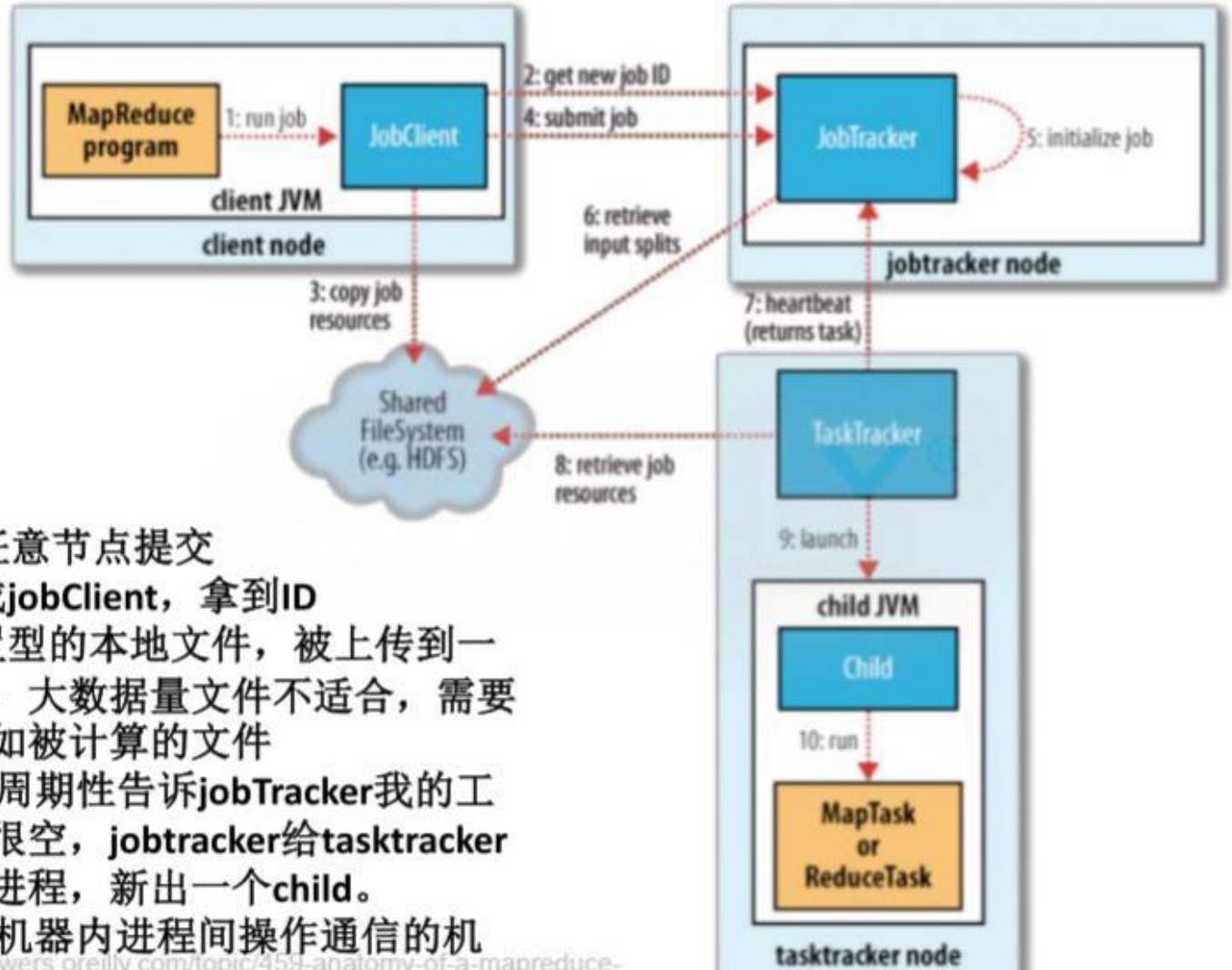
HDFS的运行机制

- 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

HDFS的运行机制

- 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

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

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

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

Hbase的运行机制

- 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

北京大学
PEKING UNIVERSITY

感谢大家的包容！

2001210374-孟庆博