**《高级网络计算模式》复习重点**

**1：C/S架构、B/S架构、IT鸿沟、中间件概念**

**（IT鸿沟如何形成的、如何解决、为什么引用）**

**①二层C/S架构：**

**概念：**C/S 架构是一种典型的两层架构，其全程是Client/Server，即客户端服务器端架构，其客户端包含一个或多个在用户的电脑上运行的程序，而服务器端有两种，一种是数据库服务器端，客户端通过数据库连接访问服务器端的数据；另一种是Socket服务器端，服务器端的程序通过Socket与客户端的程序通信。

C/S 架构也可以看做是胖客户端架构。因为客户端需要实现绝大多数的业务逻辑和界面展示。这种架构中，作为客户端的部分需要承受很大的压力，因为显示逻辑和事务处理都包含在其中，通过与数据库的交互（通常是SQL或存储过程的实现）来达到持久化数据，以此满足实际项目的需要。

**优点：**1) C/S体系结构具有强大的数据操作和事务处理能力，模型思想简单，易于人们理解和接受；

2) 系统的客户应用程序和服务器构件分别运行在不同的计算机上，系统中每台服务器都可以适合各构件的要求，这对于硬件和软件的变化显示出极大的适应性和灵活性，而且易于对系统进行扩充和缩小。

3) 在C/S体系结构中，系统中的功能构件充分隔离，客户应用程序的开发集中于数据的显示和分析，而数据库服务器的开发则集中于数据的管理，不必在每一个新的应用程序中都要对一个DBMS进行编码。将大的应用处理任务分布到许多通过网络连接的低成本计算机上，以节约大量费用。

**缺点：**1) 开发成本较高；

2) 客户端程序设计复杂；

3) 信息内容和形式单一；

4) 用户界面风格不一，使用繁杂，不利于推广使用；

5) 软件移植困难；

6) 软件维护和升级困难；

7) 新技术不能轻易应用。

**②三层C/S架构：**

**优点：**1) 允许合理地划分三层结构的功能，使之在逻辑上保持相对独立性，能提高系统和软件的可维护性和可扩展性。

2) 允许更灵活有效地选用相应的平台和硬件系统，使之在处理负荷能力上与处理特性上分别适应于结构清晰的三层；并且这些平台和各个组成部分可以具有良好的可升级性和开放性。

3) 应用的各层可以并行开发，可以选择各自最适合的开发语言。

4) 利用功能层有效地隔离开表示层与数据层，未授权的用户难以绕过功能层而利用数据库工具或黑客手段去非法地访问数据层，为严格的安全管理奠定了坚实的基础

**缺点：**

1) 三层C/S结构各层间的通信效率若不高，即使分配给各层的硬件能力很强，其作为整体来说也达不到所要求的性能。

2) 设计时必须慎重考虑三层间的通信方法、通信频度及数据量。这和提高各层的独立性一样是三层C/S结构的关键问题。

**③B/S架构：**

概念：B/S架构的全称为Browser/Server，即浏览器/服务器结构。Browser指的是Web浏览器，极少数事务逻辑在前端实现，但主要事务逻辑在服务器端实现，Browser客户端，WebApp服务器端和DB端构成所谓的三层架构。B/S架构的系统无须特别安装，只有Web浏览器即可。

B/S架构中，显示逻辑交给了Web浏览器，事务处理逻辑在放在了WebApp上，这样就避免了庞大的胖客户端，减少了客户端的压力。因为客户端包含的逻辑很少，因此也被成为瘦客户端。

**优点**：1) 基于B/S体系结构的软件，系统安装、修改和维护全在服务器端解决。用户在使用系统时，仅仅需要一个浏览器就可运行全部的模块，真正达到了“零客户端”的功能，很容易在运行时自动升级。

2) B/S体系结构还提供了异种机、异种网、异种应用服务的联机、联网、统一服务的最现实的开放性基础。。

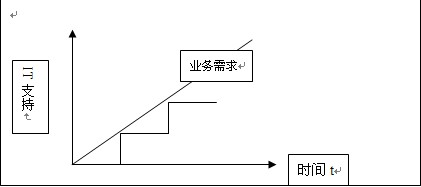
**缺点：**1) B/S体系结构缺乏对动态页面的支持能力，没有集成有效的数据库处理功能

2) B/S体系结构的系统扩展能力差，安全性难以控制

3) 采用B/S体系结构的应用系统，在数据查询等响应速度上，要远远地低于C/S体系结构

4) B/S体系结构的数据提交一般以页面为单位，数据的动态交互性不强，不利于在线事务处理应用

**④IT鸿沟：**

缩短开发周期的方法：提高重用性

**定义：**IT鸿沟指的是当业务不断发展的时候，现有的IT支持平台的发展不能满足业务的需要，而当IT支持平台发展到可以支持业务的需要的时候，业务又有新的需求和发展，这样业务与IT支持之间持续存在一个鸿沟。

**形成原因**：存在IT鸿沟的一个重要的原因就是企业中的信息孤岛问题。

**如何描述IT信息孤岛及其解决方法？**

**信息孤岛**是指相互之间在功能上不关联互助、信息不共享互换以及信息与业务流程和应用相互脱节的计算机应用系统。

**解决方法**：从用户界面集成、业务流程整合、数据层面集成、应用系统集成四个方面阐述。

**IT鸿沟解决办法：**

IT鸿沟的解决办法是EAI（企业应用集成）。一套完整的EAI层次体系包括应用接口层、应用整合层、流程整合层和用户交互层。

EAI使得企业众多信息系统都与一个由中间件组成的底层基础平台相连接，各种“应用孤岛”、“信息孤岛”通过各自的“适配器”连接到一个总线上，然后再通过一个消息队列实现各个应用之间的交流。

EAI技术层次体系最下面的一层是**应用接口层**，它要解决的是应用集成服务器与被集成系统之间的连接和数据接口的问题。

再往上去就是**应用整合层**，它要解决的是被集成系统的数据转换问题，通过建立统一的数据模型来实现不同系统间的信息转换。

应用整合层之上是**流程整合层**，它将不同的应用系统连接在一起，进行协同工作，并提供商业流程管理的相关功能，包括流程设计、监控和规划，实现业务流程的管理。

到了最上端的**用户交互层**，则是为用户在界面上提供一个统一的信息服务功能入口，通过将内部和外部各种相对分散独立的信息组成一个统一的整体，保证了用户既能够从统一的渠道访问其所需的信息，也可以依据每一个用户的要求来设置和提供个性化的服务。

**⑤中间件**

**概念：**中间件（Middleware）是一种软件，处于系统软件（操作系统和网络软件）与应用软件之间，它能使处于应用层中的各应用成分之间实现跨网络的协同工作（也就是互操作），这是运行各应用软件之下所涉及的“系统结构、操作系统、通信协议、数据库和其他应用服务”各不相同。

中间件位于C/S计算模型的操作系统、网络和数据库之上，应用软件的下层，是连接两个独立应用程序或独立系统的软件；

中间件为上层的应用软件提供运行与开发的环境，帮助用户灵活、高效地开发和集成复杂的应用软件；

分布式应用软件可借助中间件在不同的技术之间共享资源，相连接的系统即使它们具有不同的接口，但通过中间件相互之间仍能交换信息；

中间件屏蔽了底层分布式环境的复杂性和异构性，简化了分布式应用程序的开发，使分布式应用程序的健壮性、可扩展性、可重用性更好

**作用：**提供了通道功能，可保证应用系统数据的可靠传输；

起着桥梁作用，达成异构应用资源之间的互联互通；

是分布式应用集成开发框架，促成企业分布式应用的有机集成；

作为平台，可有效实现分布应用的分阶段部署；

起着过滤网作用，可有力保护关键信息的安全

**提供的最基本的系统服务**：

通信服务：用户屏蔽低层网络通信接口，为高层应用提供可靠的消息通信和远程访问；

名字服务：提供全局的名字服务空间，屏蔽低层实体分布的具体物理位置；

存储服务：提供全局的一致数据访问，集成低层分布的局部文件系统和数据库；

分布式事务服务：提供分布式事务处理，包括事务管理和调度、负载均衡和失败恢复等；

安全服务：提供存取访问控制，在中间件层次保证分布式系统的安全性。

**优越性**：

企业应用方面：缩短应用开发周期、减少项目开发风险、应用程序质量可维护、增加产品吸引力；

应用程序方面：透明地同其他应用程序交互、与运行平台提供的网络通信服务无关、具有良好的可用性和可靠性、具有良好的可扩展性。

**中间件技术的主流标准（PPT第1讲）：**DNA（Distributed Internet Applications）标准。CORBA（Common Object Request Broker Architecture）标准。J2EE（Java 2 Platform, Enterprise Edition）标准（J2EE优势及四层结构）。

**2：网格概念、五层沙漏模型、OSGA**

**（什么是网格、是什么需求引出的网格概念；五层沙漏模型怎么进化形成OSGA的，引入了什么）**

**①什么引出网格：**

由于B/S架构管理软件只安装在服务器端上，网络管理人员只需要管理服务器就行了，用户界面主要事务逻辑在服务器端完全通过WWW浏览器实现，极少部分事务逻辑在前端（Browser）实现，所有的客户端只有浏览器，网络管理人员只需要做硬件维护。

由于所有数据的运算都集中在服务器上运行，严重影响运算效率。此外应用服务器运行数据负荷较重，一旦发生服务器“崩溃”等问题，后果不堪设想。

因此，许多单位都备有数据库存储服务器，以防万一。这就需要将服务器中的数据进行分布式存储,并通过在多台服务器上进行计算来提高运算效率。网格计算在这样的需求中诞生了。

**网格思想的起源：**

网格思想是借鉴电力网的概念提出来的，其目的是希望用户像用电一样方便的使用网格的计算能力。

**网格的意义**：

网格的概念提供了与以往完全不同的计算方式，突破了以往计算方法的计算能力、地理位置、共享和协作方面的限制。

网格从各种计算资源中抽象出其中内在的“计算能力”，形成一种分布在网络上的抽象的计算能力，将原来有形的、专用的计算能力转化为一种无形的、更通用的计算能力，提供一种更强大、更方便、更高级的计算手段。

**什么是网格计算：**

网格计算在每个节点上安装中间件进行彼此间的连接，使用网格计算的用户需要参与提供计算机资源。网格计算利用互联网上计算机的CPU闲置资源处理能力来解决大型计算问题。

优点：提供高效的资源共享服务，平衡多台计算机上的负载，可把程序放到最适合的位置上。

目的：高性能计算。

网络：覆盖网络/应用层网络。

**网格计算定义**

定义：网格计算是动态多机构虚拟组织中的一个协调的共享资源和解决问题的过程（Ian Foster定义）。

网格必须满足三个条件：

在非集中控制环境下协同使用资源；

使用标准的、开放的、通用的协议（标准化）；

提供非凡的服务。

网格将网络上分布的资源聚合为一体，建立计算和数据处理的通用系统支持平台，实现资源共享和联合工作。

资源：计算、数据、存储、设备、软件和人员等；

本质：资源的共享和协同，统一和横向设计；

技术：虚拟组织之间的资源共享和协同工作技术；

方法：建立广泛的资源共享和协同的标准协议；

最终目标：数字信息基础设施。

**网格分类**

计算网格：分布式计算体系结构。

信息网格：支持用户有效共享和管理信息资源，为用户提供透明的信息服务。

知识网格（语义网格）：使用户有效的共享和管理知识资源，为用户提供知识服务。

**网格系统的特点：**异构性、可扩展性、可适应性、自治性与多重管理性

**网格的结构（PPT第2讲）：**网络基础设施。网格中间件。网格发展环境工具。网格应用和门户。

**网格体系结构：以协议为中心**的5层沙漏结构

**以服务为中心**的开发网络服务体系结构OSGA (Open Grid Services Architecture)

**②五层沙漏模型：**

构造层：提供网格服务的基础硬件，包括：计算、存储、网络等。

连接层：将硬件连接起来，并能方便地查找和使用及为保证使用安全。

资源层：将硬件转换成资源，并为资源提供管理和维护功能。有限的资源构成瓶颈。

汇集层：将所有单个的资源形成一个网格。

应用层：使用资源的用户应用程序。

**中心思想：采用以协议为中心的分层结构**

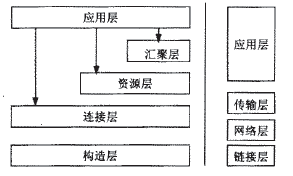
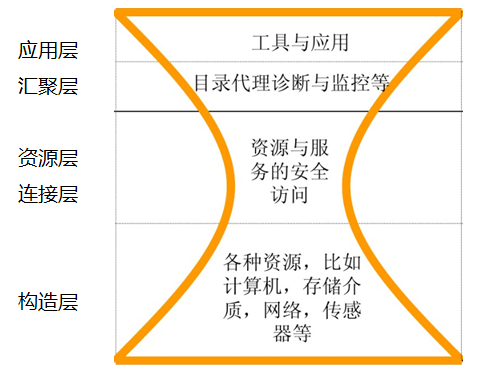
**基本思想和概念：共享、互操作性、协议、服务**

共享：深层次、广泛、动态、具有多种形式的有条件受控的共享。

互操作性：共享关系可以跨域不同的平台、语言和编程环境，跨越不同的组织边界、使用策略和资源类型。

协议：为了实现特定的操作而定义的分布式系统元素之间交互的方式以及交互过程中交换的信息结构。

服务：由所使用的协议和所实现的行为来定义。服务抽象掉了与资源相关的信息。

****

5层沙漏结构根据各组成部分离共享资源的距离，将对共享资源进行操作、管理和使用的功能分散在五个不同的层次中，越向下层就越接近共享的物理资源，与特定资源相关的成分就越多；越向上层就越感觉不到共享资源的细节特征，也就是说上层是更加抽象的共享资源的表示。

5层结构中各层协议的数量是不同的，对于最核心的协议，既要能实现上层各种协议向核心协议的映射，同时又要能实现核心协议向下层协议的映射，核心协议在所有支持网格计算的地点都应该得到支持，因此核心协议的数量不应该太多，这样核心协议就形成了协议层次结构中的瓶颈，形成沙漏形状。

**③开放网格体系服务结构OGSA**

**基本思想和概念：以服务为中心的模型。**

五层沙漏结构试图实现的是资源的共享，OGSA实现的将是对服务的共享。OGSA将一切（计算资源、存储资源、网络、程序、数据库、设备等）都看作网格服务，将网络看作可扩展的网格服务的集合。

OGSA在五层沙漏结构基础上，对Web Services进行了扩展，提出动态服务（即网格服务）的概念。网格服务是一组接口（发现、动态服务创建、生命周期管理、通知等）定义明确、遵守特定惯例的Web Services。

**OGSA的两大支撑服务（PPT第2讲）：**Globus、Web Services

两者关系：Web Services的核心是在大的异构网络上将各种应用连接起来，借助于各Web标准（UDDI、WSDL、XML等）将Internet从一个通信网络进一步发展到一个应用平台。

**3：云计算与网格技术的差别、Map-Reduce、Hadoop基本原理、数据一致性理论**

**（MapReduce的计算和处理过程；Map是怎么完成的；Reduce是怎么完成的；数据一致性中的四种同步的方式和机制）**

**①云计算和与网格技术的差别：**

相对于网格计算，在表现形式上，云计算拥有明显的特点：

第一是**低成本**，这是最突出的特点；

第二是**虚拟机的支持**，使得在网络环境下的一些原来比较难做的事情现在比较容易处理；

第三是**镜象部署**的执行，这样就能够使得过去很难处理的异构的程序的执行互操作变得比较容易处理；

第四是**强调服务化**，服务化有一些新的机制，特别是更适合商业运行的机制。

**网格计算和云计算有相似之处（特别是计算的并行与合作的特点），但他们的区别也是明显的**：

1. 网格计算的思路是**聚合分布资源**，支持虚拟组织，提供高层次的服务，例如分布协同科学研究等。而云计算的**资源相对集中**，主要以数据中心的形式提供底层资源的使用，并不强调虚拟组织的概念；

2. 网格计算用聚合资源来支持**挑战性的应用**，这是**初衷**，因为高性能计算的资源不够用，要把分散的资源聚合起来；后来逐渐强调适应普遍的**信息化应用**，特别在中国，做的网格跟国外不太一样，就是强调支持信息化的应用。但云计算从一开始就支持广泛企业计算、Web应用，**普适性更强**。

3. 在**对待异构性**方面，二者理念上有所不同。网格计算用中间件屏蔽异构系统，力图使用户面向同样的环境，把困难留在中间件，让中间件完成任务。而云计算实际上承认异构，用镜像执行，或者提供服务的机制来解决异构性的问题。

4. 网格计算用**执行作业形式**使用，在一个阶段内完成作用产生数据。而云计算支持持久服务，用户可以利用云计算作为其部分IT基础设施，实现业务的托管和外包。

5. 网格计算更多地面向科研应用，商业模型不清晰。而云计算从诞生开始就是针对企业商业应用，商业模型比较清晰。

6. 云计算是以相对集中的资源，运行分散的应用（大量分散的应用在若干大的中心执行）；而网格计算则是聚合分散的资源，支持大型集中式应用（一个大的应用分到多处执行）。

但从根本上来说，从应对Internet的应用的特征特点来说，他们是一致的，为了完成在Internet情况下支持应用，解决异构性、资源共享等等问题。

**云计算：**是并行计算、分布式计算和网格计算的发展，或者说这些计算科学的商业实现。是指基于互联网的超级计算模式——即把存储于各种异构设备上的大量信息和处理资源集中在一起，协同工作，将大规模可扩展的信息技术能力向外部客户作为服务来提供的一种计算模式。

**云计算基本思想**：数据在云端：不怕丢失,不必备份,可以任意点的恢复 ；

软件在云端：不必下载自动升级 ；

无所不在的计算：在任何时间，任意地点，任何设备登录后就可以进行计算服务；

无限强大的计算：具有无限空间的，无限速度。

**IST（美国国家标准与技术研究院）云计算定义：**

给云计算定义了五个关键特征、三个服务模型、四个部署模型

五个关键特征：按需自服务、宽带接入、虚拟化的资源“池”、快速弹性框架、可测量的服务；

三个服务模型：“SPI”模型，即软件作为服务、平台作为服务、基础设施作为服务；

四种不熟模式：公有云、私有云、社区云、混合云。

**云计算与网格计算相同之处**

**目标**：减少计算成本、增加可靠性、增加灵活性

面临很多**共同问题**：管理大型设备、定义用户发现、请求和使用资源的方法

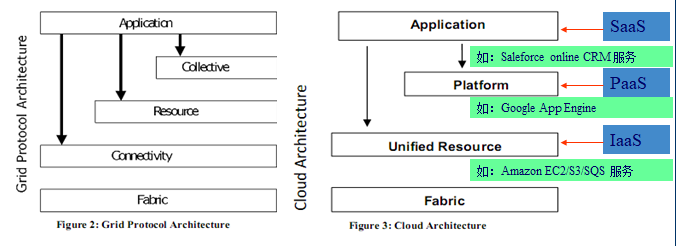
**不同之处：**业务模型、体系结构、资源管理、编程模型、应用模型、安全模型

**业务模型**

云计算：面向企业或单个用户，按需使用资源，按使用收费，如电、水、气等的使用方式一样，由单一的公司资源建立云计算环境，提供服务，资源可以使高性能设备，也可以是PC机。（**我为人人**）

网格计算：面向工程，集中于解决大型复杂的计算问题，一项任务可由来自于不同成员贡献的资源共同完成，资源一般是高性能的计算、存储设备。（**人人为我**）

**体系结构**



构造层：包括各种计算、存储、网络等硬件资源。

资源层：通过虚拟化计算，抽象为被上层或终端用户使用的资源。比如，虚拟的计算机、集群、文件系统、数据库等。

平台层：在资源层之上，整合了一系列的工具、中间件、服务等，提供应用部署平台服务。

应用层：包括云环境下运行的各种应用程序。

**资源管理——计算模型**

网格：批量计算模型，不支持交换式应用

云计算：资源同时被所有用户共享，支持延迟敏感的应用

**资源管理——数据模型**

网格：在网格环境下有专门的数据网格处理数据集中式的应用，出现了虚拟数据的概念

云计算：网络计算是数据、云计算和客户端计算的结合

**资源管理——数据位置**

云计算：能够将数据分块，存储于不同的位置，可以将任务调度到靠近数据存储位置的地方执行，提高效率

网格计算：数据存储依赖于共享的文件系统，难于利用数据位置信息，需要改进任务调度器为数据意识的。

关键需要的是**计算和数据资源管理的综合**，在访问模式中利用数据位置信息使**数据移动的数据量最小，改善终端应用的性能和可扩展性**。

**资源管理——虚拟化**

网格：每一个组织自己控制和管理自身的资源，对虚拟化的依赖程度不如云计算高

云计算：虽然不是必要的，但在实践中几乎成为不可缺少的元素

**编程模型**

网格：MPI, Globus Tookit, WSRF 等

云计算：MapReduce, Hadoop等

**应用模型**

网格：支持多种不同的应用（高性能计算、高吞吐量计算等）。

云计算：能处理和网络计算类似的应用，由于网络环境的不确定性，在高性能计算方面，云计算要稍逊一筹。支持交互式应用。

浏览器和Web 2.0技术在用户和网格与云计算交互的过程中扮演重要角色。

**安全模型**

网格：更为严格，网格安全的核心是单点登录、代理和数据隐私等。

云计算：更为简单，具有更弱的安全性。采用网络管理为用户创建账户，并允许用户通过邮件在不安全和未加密的通信中重设密码和接收新的密码。新用户通过邮件能方便容易地使用云服务。

云计算安全是云计算研究中的重要课题，普通云用户面临的安全风险主要体现在以下方面：

授权的用户接入：企业敏感数据要求只有特权用户才能访问以保证机密性。

数据隔离：保证不同用户之间的数据达到完全隔离。

数据恢复：有良好的容错机制以保证系统出错时数据不丢失。

**云计算体系结构（PPT第3讲）**

**②MapReduce分布式处理技术**、**分布式的文件系统GFS**、**结构化的BigTable存储系统**是Google的三个核心技术。

描述Map-Reduce

目标：用于大规模数据集（大于1TB）的并行计算

任务分配：Map对一组数据元素进行某种重复式的处理；Reduce对Map的中间结果进行某种进一步的整理

Master的责任

Map worker的责任

Reduce worker的责任

MapReduce不能处理不可拆分的任务，或者有依赖关系的任务

**MapReduce的计算和处理过程：**

MapReduce模式的思想：（分而治之）将要执行的问题拆解成Map（映射）和Reduce（化简）的方式，先通过Map程序将数据切割成不相关的区块，分配（调度）给大量计算机处理达到分布运算的效果，再通过Reduce程序将结果汇整，输出开发者需要的结果。

**MapReduce分布式处理技术——Map端**

map: (k1, v1) → list(k2, v2)

输入：键值对(k1, v1)表示的数据

处理：文档数据记录（如文本文件中的行，或数据表格中的行）将以“键值对”形式传入map函数；map函数将处理这些键值对，并以另一种键值对形式输出处理的一组键值对中间结果list(k2, v2)。

输出：键值对(k2, v2)表示的一组中间数据

备注: list(k2, v2) 表示有一个或多个键值对组成的列表

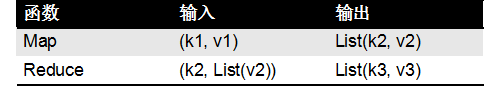
**MapReduce分布式处理技术——Reduce端**

reduce: (k2, list(v2)) → list(k3, v3)

输入：由map输出的一组键值对list(k2, v2) 将被进行合并处理，同样主键下的不同数值合并会到一个list(v2)中，故reduce的输入为(k2, list(v2))。

处理：对传入的中间结果列表数据进行某种整理或进一步的处理，并产生最终的某种形式的结果输出list(k3, v3)。

输出：最终输出结果list(k3, v3)。

****

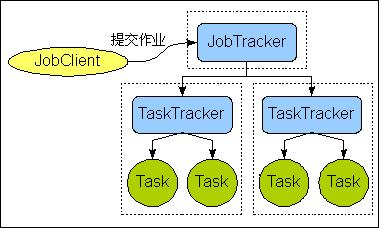
各个map函数对所划分的数据并行处理，从不同的输入数据产生不同的中间结果输出；

各个reduce各自并行计算，各自负责处理不同的中间结果数据集合；

进行reduce处理之前，须等到所有的map函数做完，并且在进入reduce前会对map的中间结果数据进行整理(Shuffle)，保证将map的结果发送给对应的reduce；

最终汇总所有reduce的输出结果即可获得最终结果。

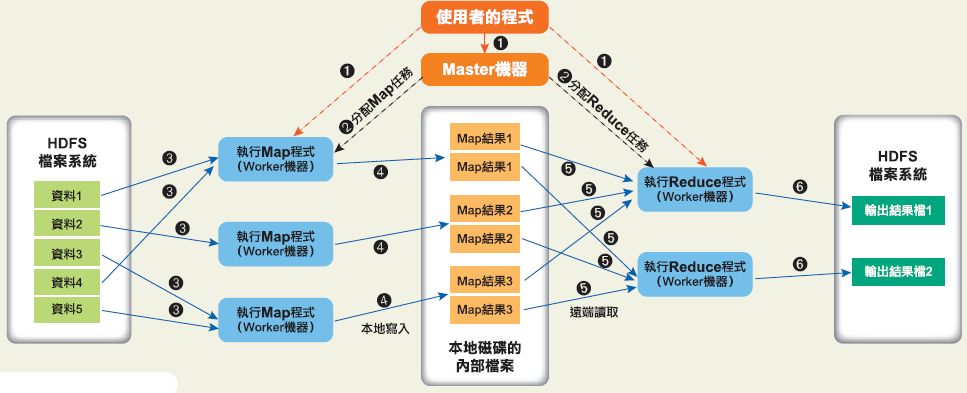
**MapReduce框架：**

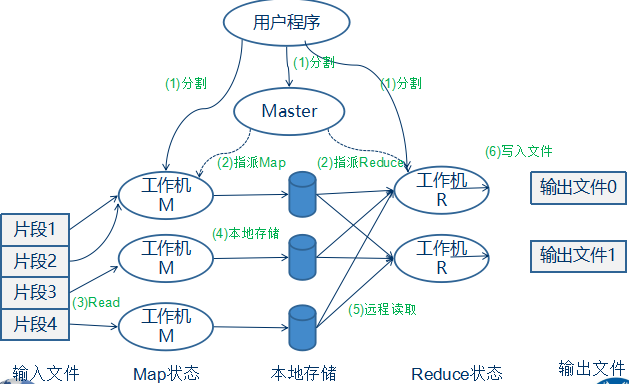


MapReduce框架由一个单独运行在主节点上的JobTracker 和运行在每个集群从节点上的TaskTracker组成。

主节点负责调度构成一个作业的所有任务，这些任务分布在不同的从节点上。主节点监控它们的执行情况，并且重新执行之前失败的任务。

**MapReduce分布式处理技术——实现机制**

****

****

(1) MapReduce函数库首先把输入文件分成M块，每块大概16MB到64MB。接着在集群的机器上执行处理程序。

(2) MapReduce算法运行过程中有一个主控程序，称为master。主控程序会产生很多作业程序，称为worker。并且把M个map任务和R个reduce任务分配给这些worker，让它们去完成。

(3) 被分配了map任务的worker读取并处理相关的输入(这里的输入是指已经被切割的输入小块splite)。它处理输入的数据，并且将分析出的键/值(key/value)对传递给用户定义的reduce()函数。map()函数产生的中间结果键/值(key/value)对暂时缓冲到内存。

(4) map()函数缓冲到内存的中间结果将被定时刷写到本地硬盘，这些数据通过分区函数分成R个区。这些中间结果在本地硬盘的位置信息将被发送回master，然后这个master负责把这些位置信息传送给reduce()函数的worker。

(5) 当master通知了reduce()函数的worker关于中间键/值(key/value)对的位置时，worker调用远程方法从map()函数的worker机器的本地硬盘上读取缓冲的中间数据。当reduce()函数的worker读取到了所有的中间数据，它就使用这些中间数据的键(key)进行排序，这样可以使得相同键(key)的值都在一起。

(6) reduce()函数的worker根据每一个中间结果的键(key)来遍历排序后的数据，并且把键(key)和相关的中间结果值(value)集合传递给reduce()函数。reduce()函数的worker最终把输出结果存放在master机器的一个输出文件中。

(7) 当所有的map任务和reduce任务都已经完成后，master激活用户程序。在这时，MapReduce返回用户程序的调用点。

(8) 当以上步骤成功结束以后，MapReduce的执行数据存放在总计R个输出文件中(每个输出文件都是由reduce任务产生的，这些文件名是用户指定的)。通常，用户不需要将这R个输出文件合并到一个文件，他们通常把这些文件作为输入传递给另一个MapReduce调用，或者用另一个分布式应用来处理这些文件，并且这些分布式应用把这些文件看成为输入文件由于分区(partition)成为的多个块文件。

③**MapReduce容错机制：**

master会定期发送命令轮询每一台worker机器。如果在一定时间内有一台worker机器一直没有响应，master就认为这个worker失效了。所有这台worker完成的map任务都被设置成为它们的初始空闲状态，并且因此可以被其他worker调度执行。类似的，所有这个机器上正在处理的map任务或者reduce任务都被设置成为空闲状态，被其他worker重新执行。

在失效机器上的已经完成的map任务还需要再次重新执行，这是因为中间结果存放在这个失效的机器上，所以导致中间结果无法访问。已经完成的reduce任务无需再次执行，因为它们的结果已经保存在全局的文件系统中了。

当map任务首先由A worker执行，随后被B worker执行的时候(因为A机器失效了)，所有执行reduce任务的worker都会被通知。所有还没有来得及从A上读取数据的worker都会从B上读取数据。

大多数MapReduce函数库都能够有效地支持很多worker失效的情况。比如，在一个网络例行维护时，可能会导致每次大约有80台机器在几分钟之内不能访问。master会简单地使这些不能访问的worker上的工作再执行一次，并且继续调度进程，直到所有任务都完成。

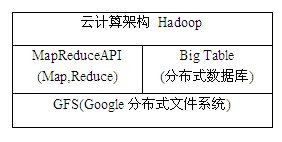
**GFS、BigTable（PPT第4讲）**

**④Hadoop基本原理：**

在Google发表MapReduce后，2004年开源社群用Java搭建出一套Hadoop框架，用于实现MapReduce 算法，能够把应用程序分割成许多很小的工作单元，每个单元可以在任何集群节点上执行或重复执行。

此外，Hadoop 还提供一个分布式文件系统GFS（Google file system），支持大型、分布式大数据量的读写操作，其容错性较强。而分布式数据库（BigTable）是一个有序、稀疏、多维度的映射表，有良好的伸缩性和高可用性，用来将数据存储或部署到各个计算节点上。

在架构中MapReduce API提供Map和Reduce处理、GFS分布式文件系统和BigTable分布式数据库提供数据存取。基于Hadoop可以非常轻松和方便完成处理海量数据的分布式并行程序，并运行于大规模集群上。

****

**Google云计算执行过程：**

（1）将要执行的MPI程序复制到Hadoop框架中的Master和每一台Worker机器中。

（2）Master选择由哪些Worker机器来执行Map程序与Reduce程序。

（3）分配所有的数据区块到执行Map程序的Worker机器中进行Map（切割成小块数据）。

（4）将Map后的结果存入Worker机器。

**⑤数据一致性理论：**

**① CAP理论：**

强一致性（Consistency）。系统在执行某项操作后仍然处于一致状态。在分布式系统中，更新操作执行成功后所有用户都应该读取到最新的值，这样的系统被认为具有强一致性。

可用性（Availability）。每一个操作总是能够在一定的时间内返回结果。

分区容错性（Partition Tolerance）。分区容错性可以理解为系统在存在网络分区的情况下仍然可以接受请求（满足一致性和可用性）。

CAP理论是在分布式环境中设计和部署系统时需要考虑的三个重要的系统需求。根据CAP理论，**数据共享系统只能满足这三个特性中的两个，而不能同时满足三个条件**。因此系统必须在这三个特性之间做出权衡。

**②最终一致性模型：**

分布式系统一般通过复制数据来提高系统的可靠性和容错性，并且将数据的不同副本存放在不同的机器上，由于维护数据副本一致性代价很高，许多系统采用弱一致性来提高性能。不同的一致性模型：强一致性、弱一致性、最终一致性。

**强一致性**：要求无论更新操作在哪个数据副本上执行，之后所有的读操作都要获得最新的数据。对于单副本数据来说，读写操作在同一数据上执行，容易保证强一致性。对于多副本数据来说，需要使用分布式事物协议（如Paxos）。

**弱一致性**：在这种一致性下，用户读到某一操作对系统特定数据的更新需要一段时间。这段时间成为“不一致性窗口”。

**最终一致性：**是弱一致性的一种特例，在这种一致性下系统保证用户最终能够读取到某操作对系统特定数据的更新。

此种情况下，“不一致性窗口”的大小依赖于交互延迟、系统负载，以及复制技术中replica的个数（可以理解为master/slave模式中，slave的个数）。

DNS系统是在最终一致性方面最出名的系统，当更新一个域名的IP以后，根据配置策略以及缓存控制策略的不同，最终所有客户都会看见最新值。

**最终一致性模型分类：**

最终一致性模型根据其提供的保证可以划分为多个模型：

**因果一致性**（Causal Consistency）：假如有相互独立的A、B、C三个进程对数据进行操作。进程A对某数据进行更新后并将该操作通知给B，那么B接下来的读操作能够读到A更新的数据值。但是由于A没有将该操作通知给C，那么系统将不保证C一定能够读取到A更新的数据。

**读自写一致性**（Read Your Own Writes Consistency）：用户更新某个数据后，读取该数据时能够获取其更新后的值，而其他用户读取该数据时则不能保证读取到最新值。

**会话一致性**（Session Consistency）：指读自写一致性被限制在一个会话范围内，也就是说提交更新操作的用户在同一个会话里读取该数据时能够保证数据是最新的。

**单调读一致性**（Monotonic Read Consistency）：指用户读取某个数据值，后续操作不会读取到该数据更早版本的值。

**时间轴一致性**（Timeline Consistency）：要求数据的所有副本以相同顺序执行所有更新操作，也称为单调写一致性（Monotonic Write Consistency）。

**③ACID与BASE：**

CAP理论指出一致性、可用性和分区容错性不能同时满足。对于数据不断增长的系统（如社会计算、网络服务的系统），他们对可用性及分区容错性的要求高于强一致性，并且很难满足事务所要求的ACID特性，因此BASE理论被提出。

**事务**是用户定义的一个数据库操作序列，这些操作要么全不做，要么全做，是一个不可分割的单位，**ACID是事务所具有的特性**：

原子性（Atomicity）：事务中的操作要么都做，要么都不做。

一致性（Consistency）：系统必须始终处于强一致状态下。

隔离性（Isolation）：一个事务的执行不能被其他事务所干扰。

持续性（Durability）：一个已提交的事务对数据库中数据的改变是永久性的。

ACID特性是传统关系型数据库中事务管理的重要任务，也是恢复和并发控制的基本单位。

**BASE方法**通过牺牲一致性和孤立性来提高可用性和系统性能，其中BASE分别代表：

基本可（Basically Availability）：系统基本能够运行、一直提供服务。

软状态（Soft-state）：系统不要求保持强一致性。

最终一致性（Eventually consistency）：系统需要在某一刻后达到一致性要求。



**④ 数据一致性实现技术：**

分布式系统在不同节点的数据采用什么技术保证一致性，取决于应用对系统一致性的需求，在关系型数据管理系统中一般会采用悲观的方法（如加锁），这些方法代价比较高、对系统性能影响较大，而在一些强调性能的系统中则会采用乐观的方法。

**Quorum系统NWR策略**

对于数据在不同副本中的一致性，此策略采用类似于Quorum系统的一致性协议实现。这个协议有三个关键值N、R、W。

* + N表示数据所有副本数。
  + R表示完成读操作所需要读取的最小副本数，即一次读操作所需参与的最小节点数目。
  + W表示完成写操作所需要写入的最小副本数，即一次写操作所需参与的最小节点数目。

该策略中，只需要保证R+W>N，就可以保证强一致性。

**两阶段提交协议**

两阶段提交协议（Two Phase Commit Protocol，2PC协议）可以保证数据的强一致性，许多分布式关系型数据库管理系统采用此协议完成分布式事务。

* + 两阶段提交协议系统一般包含两类节点：一类为协调者，通常一个系统中只有一个；另一类为事务参与者，一般包含多个，在数据存储系统中可以理解为数据副本的个数。

**时间戳策略**

* + 基于时间戳，可为每一份数据附加一个时间戳标记，在进行数据版本比较或数据同步的时候只需要比较其时间戳就可以区分它们的版本。
  + 改进时间戳，不依赖单个机器，也不依赖物理时钟同步。该时间戳为逻辑上的时钟。

逻辑时间戳用来确定分布式系统中事件的**先后关系**。假设发送或接收进程中的一个事件，分布式系统中事件的先后关系可用“->”来表示，例如：若事件a发生在事件b之前，则a->b。该关系满足：

* + 如果事件a和事件b是同一进程中的事件，并且a在b之前发生，那么a->b。
  + 如果事件a是某消息发送方进程中的事件，事件b是该消息接收方进程中接收该消息的事件，那么a->b。
  + 对于事件a、事件b和事件c，如果有a->b和b->c，那么a->c。
  + 如果两个不同的事件a和事件b，既不能得出a->b也不能得出b->a，那么逻辑上认为事件a和事件b同时发生。

**PAXOS算法**

Paxos算法常用于具有较高容错性的分布式系统，其核心就是一致性算法，该算法解决的主要问题是一个分布式系统如何就某个值（决议）达成一致。分布式系统Hadoop中的Zookeeper为Paxos算法的开源实现。

Paxos算法中有三种角色：

* + 提议者（proposer）：提出决议。
  + 批准者（acceptor）：负责批准决议。
  + 学习者（learner）：主要学习决议。

在具体实现中，一个节点可以担当多个角色。

假设节点之间通过发送信息进行通信，这里使用异步、非拜占庭模型。在该模型中：

* + 节点以任意的速度进行操作，可能因为故障而停止，也可以重新启动。并且节点所选择的决议不会因为重启等故障消失。
  + 消息可以延迟发送、多次发送或丢失，但不会被篡改（即不存在拜占庭问题）。

（四种方法的细节见PPT第5讲）

**4：P2P的概念、DHT的概念、Chord算法、Pastry算法**

**（P2P概念：第一代P2P和第二代P2P**

**三种架构：C/S 架构、纯Peer架构、离散哈希表（重点），除了离散哈希表，以前的P2P还有哪几种结构模式**

**离散哈希表中，如何将key-value对发布到分布式P2P网络中，涉及到离散哈希表的一些算法：chord 、Pastry CAN**

**要理解算法，算法有哪些特点，算法如何实现：1. 如何将key-value对发布到分布式P2P网络中2.查找时如何找到key-value对这两个过程 算法特性，拓扑适配是什么样的，查询效率）**

**①P2P概念：**

对等网络(Peer-to-Peer Networks)是分布式系统和计算机网络相结合的产物，在应用领域和学术界获得了广泛的重视和成功，被称为“改变Internet的一代网络技术”。

P2P: Peer to Peer对等网络。

peer指网络结点，在行为上是自由的——任意加入、退出，不受其它结点限制，匿名；在功能上是平等的——不管实际能力的差异；在连接上是互联的——直接/间接，任两结点可建立逻辑链接，对应物理网上的一条IP路径。

充分利用网络带宽、节点资源，提高工作效率。

**P2P体系结构（分三类）**

1、完全集中式

研究目标及重点是应用模式从C/S模式向对等模式的转变。

优点：应用模式消除了应用服务器的瓶颈问题并缓解了应用流量的不均衡性，在目录服务器获取资源索引信息之后的所有数据的交换都是在节点间完成的。简单易部署。可以模糊查询。

缺点：单点失效。尽管可以用并行服务器解决。

拓扑结构：非结构化、集中式。

典型代表：Napster

2、非集中式、非结构化

研究目标和重点是**去除体系结构上的单点失效**问题。对象查询是分布式的。查询是逐跳的，泛洪式直到成功或失败或超时。

优点：自组织的管理模式使得整个系统的鲁棒性得以大幅度增强。可以模糊查询。

缺点：**消息传递（泛洪、回溯等）的资源定位模式**制约了网络规模的可缩放性。查询效率低。

典型代表：Gnutella、Freenet、KaZaA

拓扑结构：非结构化、非集中式、无规则分布式

3、非集中式且结构化

需要解决的问题则是如何增强网络规模的可缩放特性。对象查询也是分布式的。使用**DHT技术**构造结构化拓扑。对象的查询也是逐跳的执行，经过确定的步跳可以确信是成功的。

拓扑结构：非集中式、**结构化**。

典型P2P网络如：CAN、Chord、Oceanstore等。

优点：在资源管理过程中同时拥有自组织特性、规模的强可缩放特性以及部署的廉价性等等。为规模庞大的资源整合及共享提供了可能性。

缺点：节点仅存在局部视图。缺少权威第三方的控制。不支持模糊查询。

**文档路由模型（PPT第6讲）**

实现文档路由的四大算法：Chord、CAN、Tapestry、Pastry

**第二代P2P：**

没有集中的目录服务器，但是拓扑结构有意义。这个结构意味着P2P网络拓扑被紧紧的控制。比如： Mesh, Ring , d-dimension Torus, K-ary tree。

使用DHT技术，有较好的可伸缩性和查询效率。提供负载均衡和确定性的搜索保证。但是容错性或弹性不好，尤其是在恶意攻击下。文件不是被 随机地而是以特定的位置放置，这样使得连续的查询更加容易满足。

它使用精确的定位算法和特殊的路由协议使得搜索效率提高。支持精确查询不能支持模糊查询。

**第二代P2P的代表**

Tapestry :用于覆盖网络的定位和路由机制。Tapestry具有自我管理、容错和灵活平衡负载等特点。

Pastry：是一个路由定位协议，与Tapestry有许多相似性。

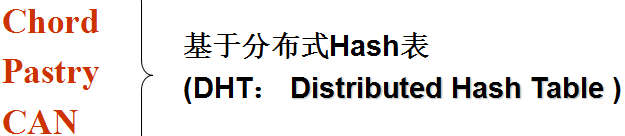
CAN：可以在Internet规模的大型对等网络上提供类似哈希表的功能，提供查询服务。CAN具有可扩展、容错和完全自组织等特点。。

CHORD：是一个分布式的非集中式的P2P查询服务，存储关键字/值对。给定一个关键字（key），将key映射到某个结点。如果给对等网络应用的每个数据都分配一个key，那么对等网络中的数据查找问题就可以用Chord很容易地解决了。

**②DHT：**

**DHT的概念、主要思想：**

什么是DHT：把网络上资源的存取像Hashtable一样，可以简单而快速地进行put、get，该思想的诞生主要是受第一代P2P（Napster）网络的影响。DHT更强调的是资源的存取，而不管资源是否是一致性的。

****

结构化P2P: 直接根据查询内容的关键字定位其索引的存放节点

DHT算法：

1.将内容索引抽象为<K, V>对

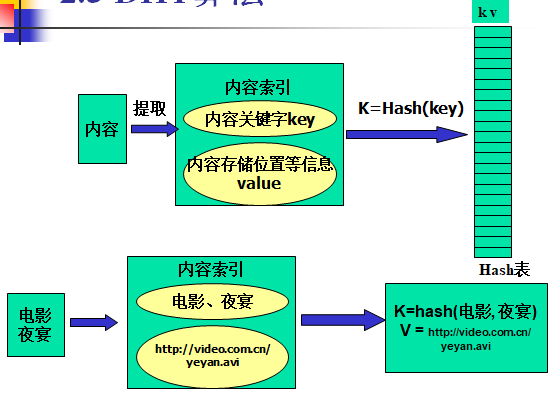
K是内容关键字的Hash摘要：K = Hash(key)

V是存放内容的实际位置，例如节点IP地址等

2.所有的<K, V>对组成一张大的Hash表，该表存储了所有内容的信息

3.每个节点都随机生成一个标识(ID)，把Hash表分割成许多小块，按特定规则（即K和节点ID之间的映射关系）分布到网络中去，节点按这个规则在应用层上形成一个结构化的重叠网络

4.给定查询内容的K值，可以根据K和节点ID之间的映射关系在重叠（Overlay）网络上找到相应的V值，从而获得存储文件的节点IP地址



**定位(Locating)**：节点ID和其存放的<K, V>对中的K存在着映射关系，因此可以由K获得存放该<K, V>对的节点ID

**路由(Routing)：**在覆盖网上根据节点ID进行路由，将查询消息最终发送到目的节点。每个节点需要到其邻近节点的路由信息，包括节点ID、IP等

**网络拓扑：**

拓扑结构由节点ID和其存放的<K, V>对中的K之间的映射关系决定

拓扑动态变化，需要处理节点加入/退出/失效的情况

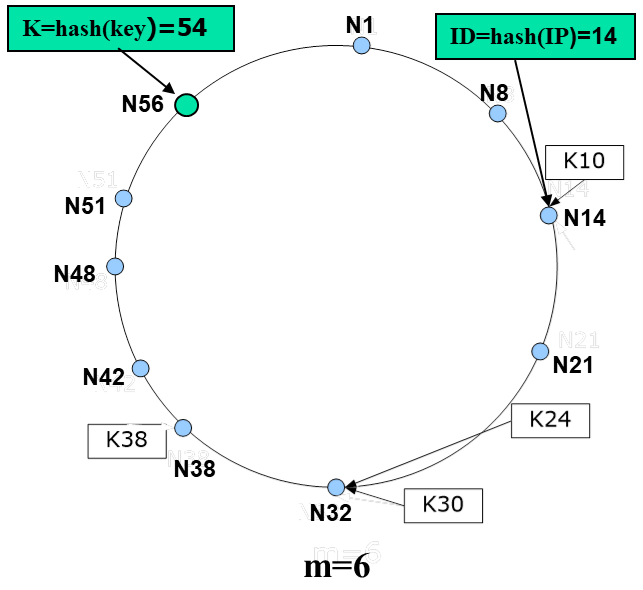
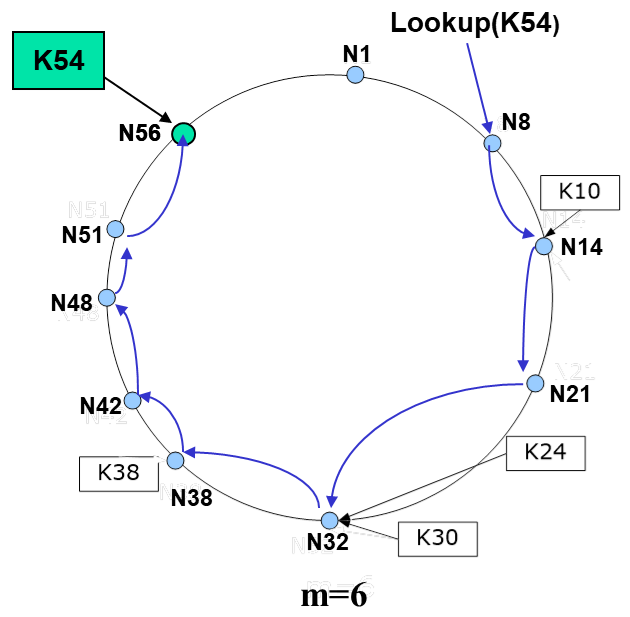
**Chord、Pastry算法细节见PPT第7讲**

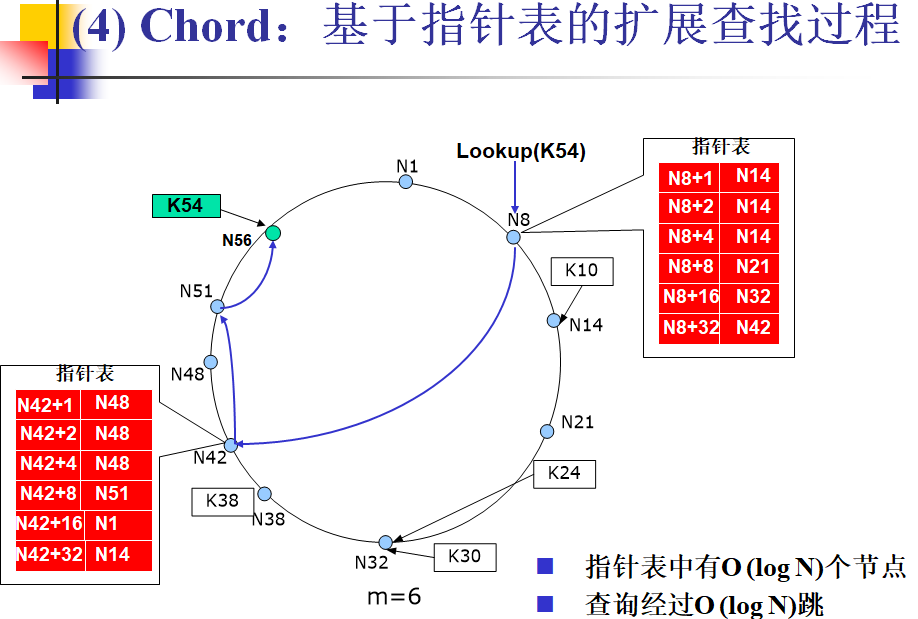
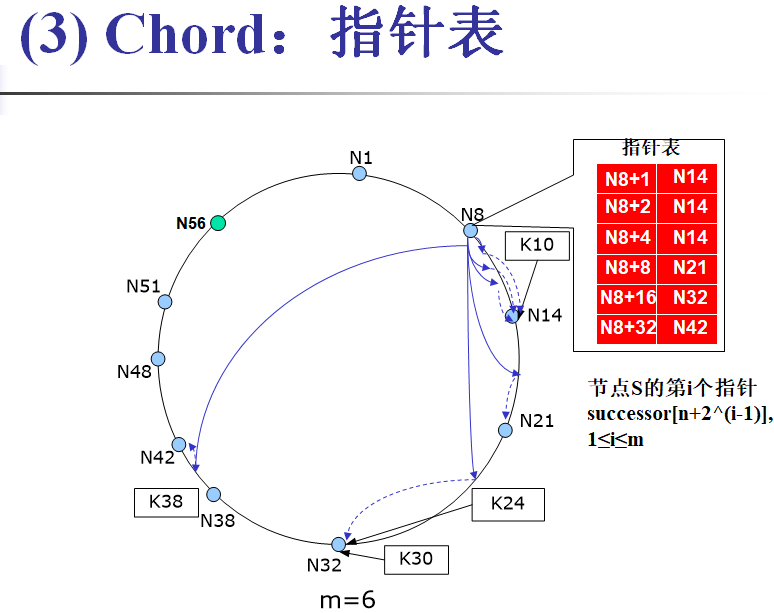
* **Chord**
  + SHA-1：目标Hash节点IP地址－>m位节点ID(表示为NID)，Hash内容关键字－>m位K(表示为KID)。<K, V>存放在顺时针方向的离该节点最近的下一个节点Successor(K)（从K开始顺时针方向距离K最近的节点 ）。Chord中每个节点上都有一张局部的指针表，表中存放2i 个节点信息（i=0，1，……，n-1）。
  + 每个节点仅维护其后继节点ID、IP地址等信息
  + 查询消息通过后继节点指针在圆环上传递
  + 直到查询消息中包含的K落在某节点ID和它的后继节点ID之间
  + 这种方法的时间效率是log（n），相当于进行折半查找。

环形拓扑（Chord环）

核心思想：解决如何在P2P网络中找到存有特定数据的节点

使用一致性哈希作为哈希算法，即SHA-1



**网络波动（Churn）：**Churn由节点的加入、退出或者失效所引起。

每个节点都周期性地运行探测协议来检测新加入节点或退出/失效节点，从而更新自己的指针表和指向后继节点的指针。

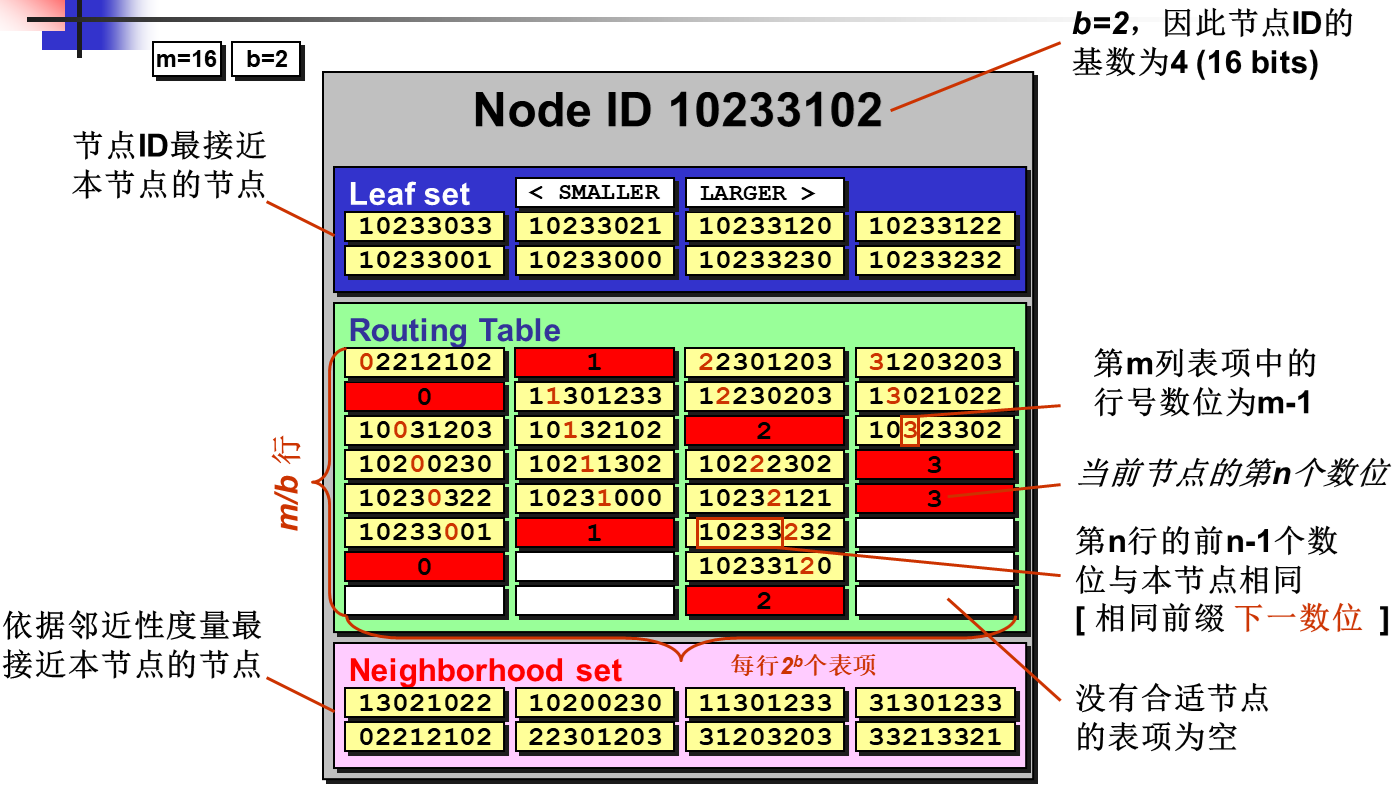
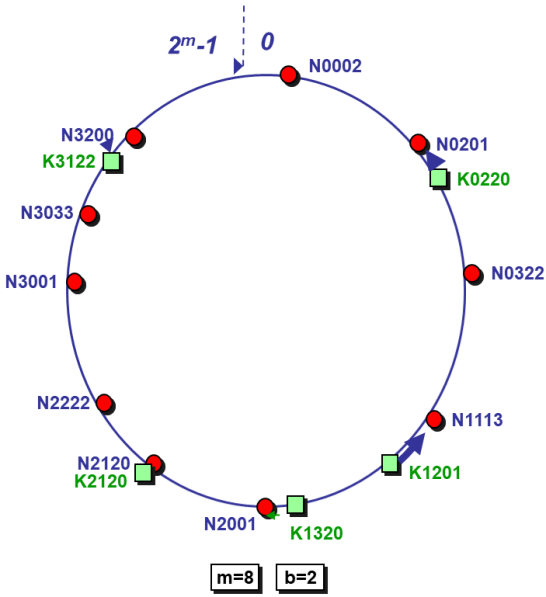
**拓扑失配问题：**

* O(LogN)逻辑跳数，但是每一逻辑跳可能跨越多个自治域，甚至是多个国家的网络
* 覆盖网络与物理网络脱节
* 实际的寻路时延较大
* 解决：
  + 提取物理网络的拓扑信息改造Chord
  + 存在w个界标站点
  + 每个节点测量它到w个界标的RTT
  + 将RTT递增排列
  + 具有相同RTT序列的节点在物理网络上临近

**Chord小结**

* 算法简单
* 负载平衡：所有的节点以同等的概率分担系统负荷，从而避免某些节点负载过大
* 可扩展：查询过程的通信开销和节点维护的状态随着系统总节点数增加成对数关系(O (log N)数量级)
* 可用性：要求节点根据网络变化动态更新查询表，能够及时恢复路由关系，使得查询可靠地进行。
* 存在拓扑失配问题
* **Pastry**
  + Hash节点IP地址－>m位节点ID(表示为NID)
  + Hash内容关键字－>m位K(表示为KID)
  + NID和KID是以为基的数，共有m/b个数位（m/b是一个配置参数，通常为4）
  + 节点按ID从小到大的顺序排列在一个逻辑环上
  + <K, V>存储在离它最接近的节点上（即存储在NID和KID数值最接近的节点上）
  + 每个节点维护一个状态表：路由表、邻居节点集、叶子节点集。

考虑网络的本地性，解决物理网路和逻辑网络的拓扑失配问题；基于应用层定义的邻近性度量，例如IP路由跳数、地理距离、往返延时等；节点ID分布采用环形结构。



**节点维护状态表**：

* **路由表R**
  + 包括 m/b行，每行包括2b 个表项
  + 第n行与节点ID的前n-1个数位相同，第n个数位不同，取值从0到2b -1，也称n-1数位前缀相同
  + 表中的每项包含节点ID，即IP地址等
  + 根据邻近性度量选择距离本节点近的节点
  + b过大，节点要维护的路由表大，但存储的邻居节点多，在转发时更为精确，b的选择反映了路由表大小和路由效率之间的折衷
* **邻居节点集M**
  + 存放在真实网络中与当前节点“距离”最近的|M|个节点的信息
  + “距离” 类似IP路由协议中的距离， 考虑转发跳数、传输路径带宽、QoS等综合因素后所得的转发开销
  + |M|的典型值为2b或者2×2b
  + 邻居节点集通常不用于路由查询消息，而是用来维护本地性
* **叶子节点集L**
  + 存放在键值空间中与当前节点距离最近的|L|个节点的信息，其中各有一半的节点标识大于或小于当前节点
  + |L|的典型值为2b或者2×2b
  + 路由时，首先检查叶子节点集

**查询过程：**

当一个K为D的查询消息到达节点A

1. 节点A首先看D是否在当前节点的叶子节点集中，如果是，则查询消息直接被转发到目的节点，也就是叶子节点集中节点ID与D数值最接近的那个节点（有可能就是当前节点），否则进行下一步。

2. 在路由表中根据最长前缀优先的原则选择一个节点作为路由目标，转发路由消息 ，如果该表项不为空，则将查询消息直接转发到该节点，否则进行下一步

3. 如果不存在这样的节点，当前节点将会从其维护的所有邻居节点集合(包括路由表叶子集合及邻居集合中的节点)中选择一个距离消息键值最近的节点作为转发目标。

路由查询消息的逻辑跳数： *O(log2b N)*

**节点状态表和查询：**

* 节点路由表R中的每项与本节点具有相同的n数位长度前缀，但是下一个数位不同
* 例如，对于节点N0201：

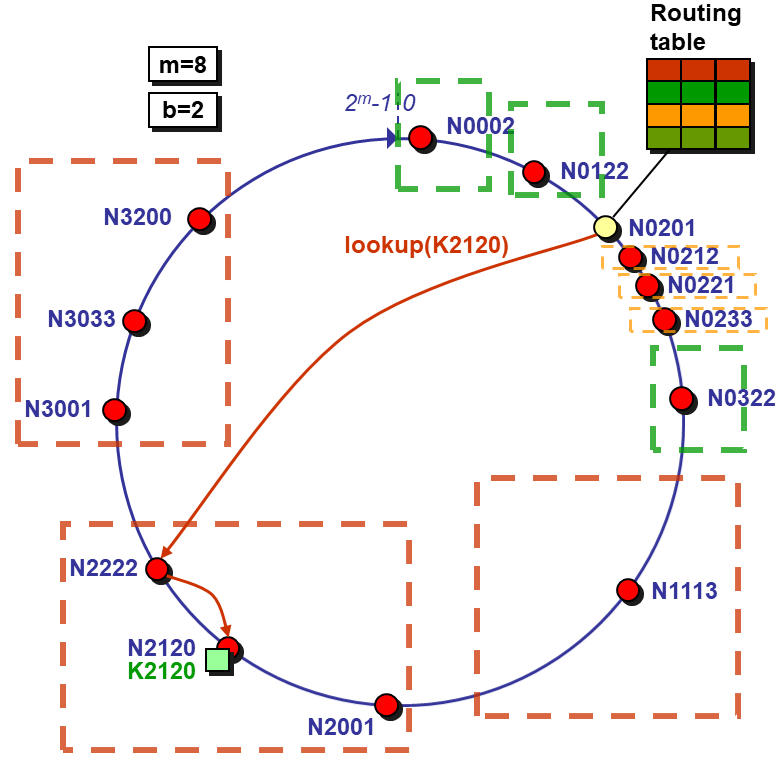
N-: N1???, N2???, N3???

N0: N00??, N01??, N03??

N02: N021?, N022?, N023?

N020: N0200, N0202, N0203

* 当有多个节点时，根据邻近性度量选择最近的节点（维持了较好的本地性）



**节点加入和失效/退出、路由本地性（PPT第7讲）**

**节点加入：**

1) 由于A与X在邻近性度量上接近，所以使用A的邻居节点集来初始化X的邻居节点集；

2) 由于Z的节点ID与X最相近，因此使用Z的叶子节点集来初始化X的叶子节点集；

3) X将Join消息经过的第i个节点的路由表的第i行作为自己路由表的第i行，因为Join消息经过的第i个节点与X的前i个数位相同；

4) 向其他相关节点通告自己的到来：新节点向邻居节点集、叶子节点集和路由表中的每个节点发送自己的状态，以更新这些节点的状态表。

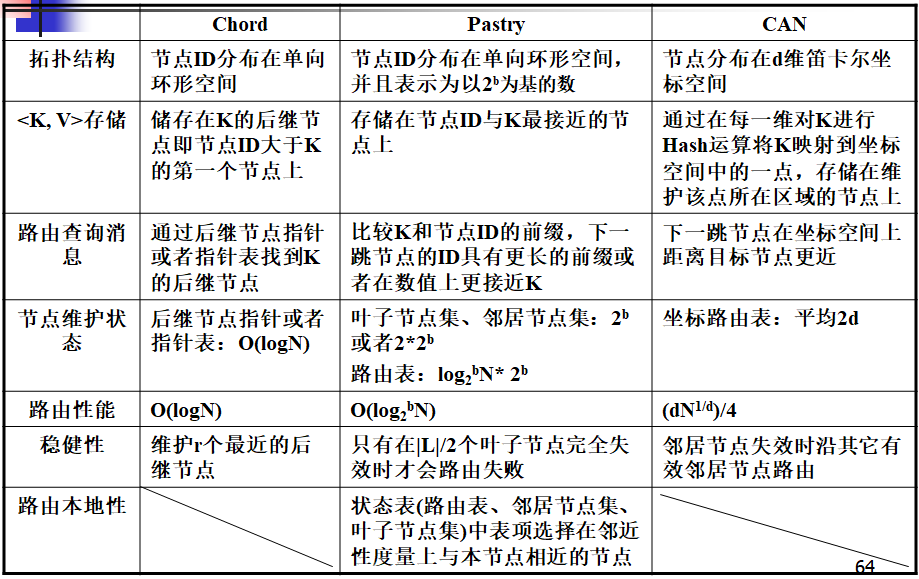
**Pastry小结：**

* 逻辑网络路由跳数*O(log2b N)*
* 路由表开销*log2b N* \* 2b
* 路由本地性：状态表（路由表、邻居节点集、叶子节点集）中的表项选择在邻近性度量上与本节点相近的节点
* 稳健性：只有在|L|/2个叶子节点完全失效时才会路由失败

**CAN（PPT第7讲）**

* **几种结构化P2P总结**
  + 完全分布式，不存在任何中心节点
  + 直接根据查询内容的关键字定位其索引的存放节点，查找具有确定性
  + 节点失效时表现出很好的健壮性
  + 可扩展性好，系统开销小
  + 自动配置，不需要手工干预就可自动把加入新节点
  + 几个需要研究的问题
    - 模糊查找问题
    - 网络波动(Churn)问题
    - 路由本地性问题
    - 负载均衡问题
    - 安全问题

**基于DHT的结构化P2P比较：**



**5：CDN网络概念、DNS重定向技术**

**（域名系统如何影响我们CDN网络分发，如何使用域名系统来进行重定向）**

**域名系统（PPT第8讲）**

* **CDN网络概念**
* **两种主流的CDN网络技术架构**
  + Akaimai 和 Limelight
  + 内容路由的实现机制
  + DNS重定向机制的阐述

**CDN（细节见第9讲）：**

全称是Content Delivery Network，即内容分发网络。

其目的通过在现有的Internet中增加一层新的网络架构，将网站的内容发布到最接近用户的网络“边缘”，使用户可以就近取得所需的内容，解决Internet网络拥挤的状况，提供用户访问网站的响应速度。

从技术上全面解决由于网络带宽小、用户访问量大、网点分布不均等原因，解决用户访问网站的响应速度慢的根本原因。

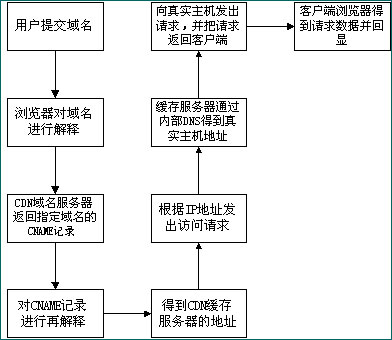
* **CDN工作原理**

当用户访问某域名上的内容信息时，首先需用通过域名解析系统将该域名转换为存储该内容信息的内容服务器的IP地址，再根据该IP地址访问内容服务器以获取内容信息。如果该域名使用了CDN技术，该域名上的内容信息将会被复制并发布到广泛分布的、位于不同地理位置的CDN网络的边缘节点服务器。用户访问使用CDN技术的域名时，其地理位置附近的CDN网络的边缘节点服务器将向其提供相关内容信息。

传统的未加缓存服务的访问过程：

http://pic001.cnblogs.com/images/2011/1/2011111514082838.jpg

使用CDN缓存后的网站访问过程：



* **Akamai CDN技术架构**

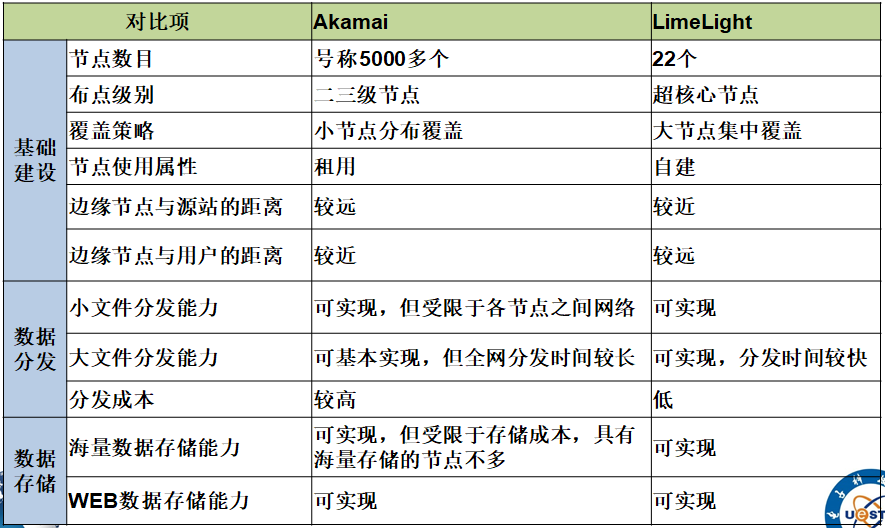
Akamai CDN在全球建立部署了几千个节点，以物理节点距离用户最近而著称；通过对各节点间的路由优化，实现信息的分发；主要提供WEB、下载、流媒体等加速服务。

* **LL CDN技术架构**

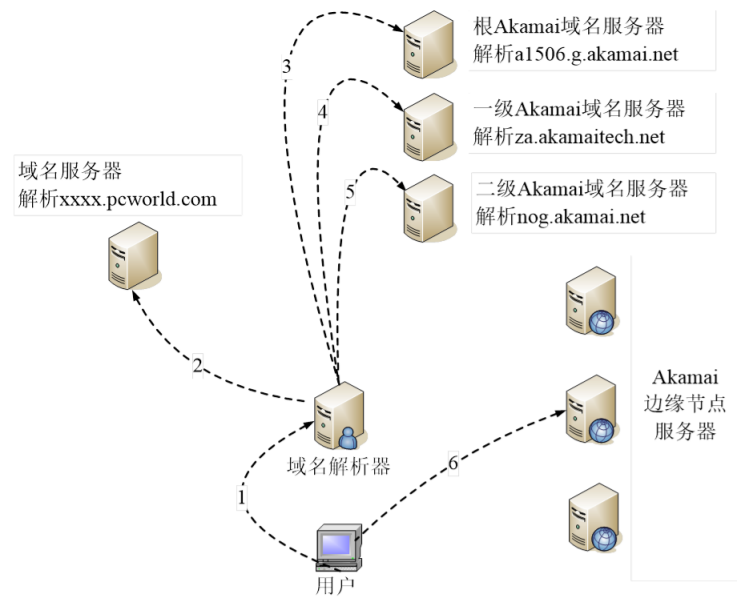
LimeLight通过在全球最重要的城市建立“超级节点”，且所有“超级节点”间采用自有光纤直连，实现全球多运营商网络的互联互通；如同在全世界建立了一个自有的、具有互联互通的“BGP”网络。

LimeLight在每个“超级节点”部署了成千上万的服务器、存储、网络设备。形成了分布式的全球“信息”存储中心。

通过“超级节点”网络结构的使用，LimeLight能够更好地满足和支持富媒体全球分布式分发，特别适合于为大资料库、大文件、海量观众提供高效率分发服务。



* **CDN关键技术**
  + 内容管理（内容、索引、访问状态信息收集）
  + 内容存储（基于缓存技术）
  + 内容分发（主动push、被动pull）
  + 内容路由（负载均衡）
* **内容路由技术**
  + 内容路由技术，即CDN实现负载均衡技术。
  + 负载均衡是整个CDN的核心，负载均衡的准确性和效率直接决定了整个CDN的效率和性能。
  + 负载均衡技术将网络的流量尽可能均匀地分配到几个能完成相同任务的服务器或网络节点上进行处理，避免部分网络节点过载而另一部分节点空闲的不利状况，既可以提高网络流量，又可以提高网络的整体性能。
  + CDN负载均衡系统实现CDN的内容路由功能，将用户导向到CDN网络中最佳的节点。
  + **LimeLight CDN的主要实现方法为IP Anycast。**
  + **Akamai CDN的主要实现方法为DNS重定向。**
* **DNS重定向**
  + 用户向域名解析器发送域名解析请求，请求解析某域名，以获取内容服务器的IP地址；域名解析请求中包含了域名解析器的IP地址，并被发送到该域名的域名服务器；
  + 该域名的域名服务器通过返回一个CNAME类的DNS记录，将此域名解析请求重定向到CDN网络的域名服务器；
  + CDN网络的域名服务器对该域名解析请求进行智能解析，根据域名解析器的IP地址、网络的拥塞状况、各CDN网络边缘节点服务器的负载情况等，将最合适的CDN网络边缘节点服务器的IP地址返回给域名解析器，域名解析器再转发给用户，用户直接访问CDN边缘节点。



* + **示例：**

首先，域名解析器访问域名xxxx.pcworld.com的域名服务器，请求解析该域名。该域名的域名服务器返回一个CNAME（Canonical NAME）类型的DNS记录。该CNAME类型的DNS记录指示了所请求解析的域名xxxx.pcworld.com的别名a1506.g.akamai.net。其作用是通过返回一个Akamai CDN网络的内部域名，将域名解析请求重定向到Akamai CDN网络的域名服务器，即根Akamai域名服务器。

然后，域名解析器访问根Akamai域名服务器，请求解析域名a1506.g.akamai.net。根Akamai域名服务器根据域名解析器的IP地址和网络状况等动态信息，返回CNAME类型的DNS记录，即域名a1506.g.akamai.net的CNAME类型的DNS记录为za.akamaitech.net，将域名解析请求重定向到一级Akamai域名服务器。

同样的方法，域名解析器访问一级Akamai域名服务器，请求解析域名za.akamaitech.net。一级Akamai域名服务器也根据域名解析器IP地址和网络状况等动态信息，返回CNAME类型的DNS记录，即域名za.akamaitech.net的CNAME类型的DNS记录为nog.akamai.net，再将域名解析请求重定向到二级Akamai域名服务器。

最后，域名解析器访问二级Akamai域名服务器，请求解析域名nog.akamai.net。二级Akamai域名服务器根据域名解析器IP地址和网络状况等动态信息，返回A类型的DNS记录，即Akamai CDN网络边缘节点服务器的IP地址。通常情况下，二级Akamai域名服务器将返回2个IP地址，用户可选择其中的任意一个边缘节点服务器发出访问请求。

**6：物联网体系结构**

**（应用案例不考，不用画图，物联网基础，几层模型，主要技术是什么）**

**按三层结构阐述：感知层、网络层、应用层**

**按五层结构阐述：感知层、接入层、网络层、中间件层、应用层**

按五层结构阐述示例：

感知层由各种传感器以及传感器网关构成，包括二氧化碳浓度传感器、温度传感器、湿度传感器、二维码标签、RFID 标签和读写器、摄像头、GPS等感知终端。感知层的作用相当于人的眼耳鼻喉和皮肤等神经末梢，它是物联网获识别物体，采集信息的来源，其主要功能是识别物体，采集信息。

接入层用来读取识别到的物体的信息并将读到的内容连到Internet。……

* **联网的概念分为广义和狭义两方面**

广义来讲，物联网是一个未来发展的愿景，等同于“未来的互联网”，或者是“泛在网络”，能够实现人在任何时间、地点，使用任何网络与任何人与物的信息交换，以及物与物之间的信息交换；

狭义来讲，物联网是物品之间通过传感器连接起来的局域网，不论接入互联网与否，都属于物联网的范畴。

物联网的一种定义是：通过射频识别（RFID）、红外感应器、全球定位系统（GPS）、激光扫描器等信息传感设备，按约定的协议，把任何物品与互联网连接起来，进行信息交换和通讯，以实现智能化识别、定位、跟踪、监控和管理的一种网络。

**物联网的内涵主要体现:**

互联网特征：对需要联网的物一定要能够实现互联互通的互联网络；

识别与通信特征：纳入物联网的“物”一定要具备自动识别与物物通信的功能；

智能化特征：网络系统应具有自动化、自我反馈与智能控制的特点。

**全面感知、可靠传送、智能处理是物联网的基本特征**。



**基本构成**：包括三部分——产品编码（EPC）、射频识别系统（FRID，包括标签和读写器）和信息网络系统（包括中间件、对象名称解析服务（ONS）、实体标记语言（PML）和信息服务（EPCIS））。

* **三层体系：感知层、网络层、应用层**

感知层：包括二维码标签和识读器、RFID标签和读写器、摄像头、CPS（Cyber Physical Systems，网络化物理系统）、各种传感器、传感器网络（指由大量各类传感器节点组成的自治网络，具有自组织、自愈合的特点）、T2T&H2T&H2H终端和传感器网关等，感知层的主要作用是感知和识别物体，采集并捕获信息。

网络层：包括各种通信网络与互联网形成的融合网络，这被普遍认为是最成熟的部分，除此之外还包括物联网管理中心、信息中心等利用网络的能力对海量信息进行智能处理的部分。也就是说网络层不但要具备网络运营的能力，还要提升信息运营的能力。网络层是物联网成为普遍服务的基础设施。

应用层：是将物联网技术与行业专业技术相结合，实现广泛智能化应用的解决方案集。物联网通过应用层最终实现信息技术与行业的深度融合，对国民经济和社会发展具有广泛影响。应用层的关键问题在于信息的社会化共享以及信息安全的保障。

* **按五层结构阐述：感知层、接入层、网络层、中间件层、应用层**

感知层：由各种传感器以及传感器网关构成，是物联网识别物体，采集信息的来源，其主要功能是识别物体，采集信息。

接入层：用来读取识别到的物体信息，并将读到的内容连到Internet。

网络层：由各种私有网络、互联网、有线和无线通信网、网络管理系统和云计算平台等组成，负责传递和处理感知层获取的信息。

中间件层：用来管理物联网上的资源和网络通讯，连接多个独立应用程序或独立系统，使它们之间能交换信息。

应用层：是物联网和用户（包括人、组织和其他信息）的接口，它与行业需求结合，实现物联网的智能应用。

* **物联网核心技术**
* 物联网编码
  + EPC编码体系（原则、结构与类型）
  + EPC标准（EPC Global、 EPC Global Gen2 ）
  + GSI全球统一识别系统
  + EPC编码策略与实现
* 识别和防碰撞问题
  + 多标签防碰撞
  + 多读写器防碰撞
* 信息采集技术
* 近程通讯技术
* 信息远程传输技术
* 海量信息智能分析与控制技术（云计算）



1. **简述Map-Reduce和Hadoop的基本原理。**

**Map-Reduce**的目标是用于大规模数据集（大于1TB）的并行运算，在面对节点失效的情况时能够保证大量的文件和数据依然可用。初始化时数据被分割成许多分块的小数据，这些数据都是以<key, value>的形式存储。用户程序会分配一个Master进程和许多Worker进程。

任务开始时，Master将用户程序的工作分成两种类型的任务（Map任务和Reduce任务），并将这些任务分配给相应的Workers。

Master的责任是给Map Workers和Reduce Workers分配相应的任务，检测是否有Workers进程死掉，将Map任务处理后的结果通知给Reduce任务。

Map Worker在得到Master的Map工作任务的指令后，Map Workers开始工作。Map Workers从input中获取分块的数据，并通过其中数据存储的方式对数据进行处理，并将处理的中间结果（<key, value>形成的链表）存放在本地磁盘中。

得到Master的有关Map处理结果的通知后，Reduce任务开始工作。Reduce工人把每个Map Workers的本地磁盘中的内容进行整合，形成有序的最终结果，并将结果写入到输出文件中。

假如一个Workers进程失败了，该Workers的任务就被分配给其他的Workers进程。如果一个Master进程失败了，那么整个Map-Reduce任务就会终止。

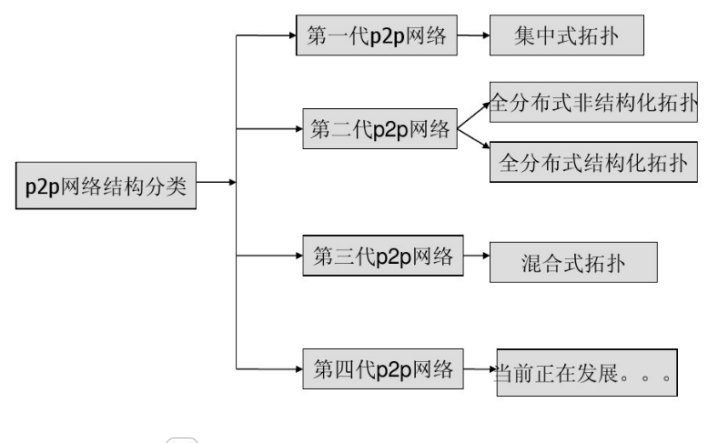
**Hadoop**是一个分布式系统基础架构。用户可以充分利用集群的威力进行高速运算和存储。Hadoop的框架核心的设计就是：HDFS和Map Reduce。HDFS为海量的数据提供了存储，则Map Reduce为海量的数据提供了计算。

Hadoop实现了一个分布式文件系统（HDFS）。HDFS有高容错性的特点，部署在低廉硬件上，提供高吞吐量来访问应用程序的数据，适合那些有着超大数据集的应用程序。

1. **简述P2P网络，阐述第一代P2P和第二代P2P的区别。**

Peer to Peer对等网络是一种分布式网络，网络的参与者共享他们所拥有的一部分硬件资源，这些共享资源需要由网络提供服务和内容，能被其它对等节点直接访问而无需经过中间实体。在此网络中的参与者既是资源提供者，又是资源获取者。第二代与第一代区别：

1. 没有集中的目录服务器，但是拓扑结构有意义。
2. 使用**DHT技术**，有较好的可伸缩性和查询效率。
3. 使用精确的定位算法和特殊的路由协议使得搜索效率提高。



1. **Chord算法原理**

**特点：**

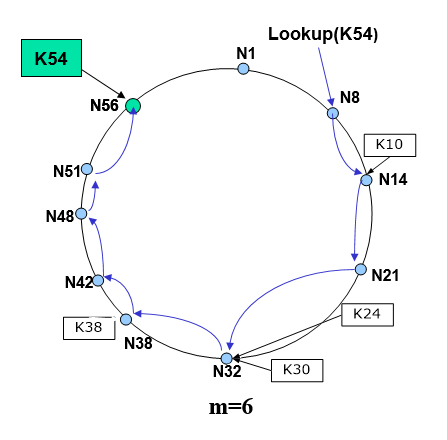
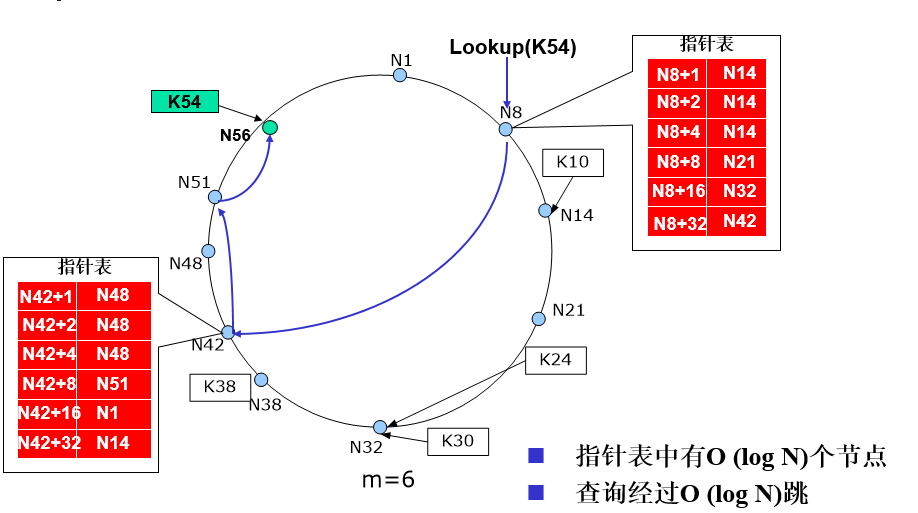
1. 采用环形拓扑(Chord环)
2. 其核心思想就是要解决在P2P应用中遇到的基本问题：如何在P2P网络中找到存有特定数据的节点
3. Chord使用一致性哈希作为哈希算法，在Chord协议中将其规定为SHA-1。

**加入过程：**

1. Hash算法：SHA-1
2. Hash节点IP地址得到m位节点ID（表示为NID）
3. Hash内容关键字－>m位K（表示为KID）
4. 节点按ID从小到大顺序排列在一个逻辑环上
5. 键值对<K, V>存储在后继节点上。即NIDi < K < NIDi+1时，键值对存在NIDi+1上。也就是从K开始顺时针方向距离K最近的节点

**改进前：**

1. 每个节点仅维护其后继节点ID、IP地址等信息
2. 查询消息通过后继节点指针在圆环上传递
3. 直到查询消息中包含的K落在某节点ID和它的后继节点ID之间
4. 速度太慢 O(N)，N为网络中节点数

改进前：改进后：

**改进后：**Chord中每个节点上都有一张局部指针表，表中存放着2m个节点信息（m是二进制位数）。

**查询过程（改进后）**

如上图所示，假设当前节点是Nc，要查找Ko 。

1. 查找当前节点是否有要查找的Ko。
2. 在指针表里面查找Ko的位置。如果找到，即Nc + 2i = Ko （i∈[0,m-1]），则直接转到对应节点Nc + 2i；若没有找到，则查找临界节点。若Nc + 2i < Ko < Nc + 2i+1 i∈[0,m-1），转到Nc + 2i 对应的节点处继续查找；若Ko> Nc + 2m-1 ，即最后一项的ID < Ko，则在Nc + 2m-1对应节点继续查找。并重复1)

**节点加入**

1. 新节点N事先知道某个或者某些节点，并且通过这些节点初始化自己的指针表，也就是说，新节点N将要求已知的系统中某节点为它查找指针表中的各个表项
2. 在其他节点运行探测协议后，新节点N将被反映到相关节点的指针表和后继节点指针中
3. 新结点N的第一个后继结点将其维护的小于N节点的ID的所有K交给该节点维护

**优点：**

1. 算法简单
2. 负载平衡：所有的节点以同等的概率分担系统负荷，从而避免某些节点负载过大
3. 可扩展：查询过程的通信开销和节点维护的状态随着系统总节点数增加成对数关系(O (log N)数量级)
4. 可用性：要求节点根据网络变化动态更新查询表，能够及时恢复路由关系，使得查询可靠地进行。

**缺点**：拓扑失配。可能本来的资源节点离自己很近，但却要通过逻辑节点绕一圈过来。

1. **简述CDN网络的概念，DNS重定向技术。**

内容分发网络的目的通过在现有的Internet中增加一层新的网络架构，将网站的内容发布到最接近用户的网络“边缘”，使用户可以就近取得所需的内容，解决Internet网络拥挤的状况，提供用户访问网站的响应速度。从技术上全面解决由于网络带宽小、用户访问量大、网点分布不均等原因，解决用户访问网站的响应速度慢的根本原因。

Akamai和Lime Light主要区别：

1. Akamai主要使用DNS重定向。Lime Light使用IP Anycast技术。
2. Akamai在全球建立部署了几千个节点，以物理节点距离用户最近而著称；通过对各节点间的路由优化，实现信息的分发；主要提供WEB、下载、流媒体等加速服务。
3. Lime Light通过在全球最重要的城市建立“超级节点”，且所有“超级节点”间采用自有光纤直连，实现全球多运营商网络的互联互通；如同在全世界建立了一个自有的、具有互联互通的“BGP”网络。Lime Light在每个“超级节点”部署了成千上万的服务器、存储、网络设备。形成了分布式的全球“信息”存储中心。通过“超级节点”网络结构的使用，Lime Light能够更好地满足和支持富媒体全球分布式分发，特别适合于为大资料库、大文件、海量观众提供高效率分发服务。

|  |  |  |  |
| --- | --- | --- | --- |
| **对比项** | | **Akamai** | **Lime Light** |
| **基础建设** | 节点数目 | 号称5000多个 | 22个 |
| 布点级别 | 二三级节点 | 超核心节点 |
| 覆盖策略 | 小节点分布覆盖 | 大节点集中覆盖 |
| 节点使用属性 | 租用 | 自建 |
| 边缘节点与源站的距离 | 较远 | 较近 |
| 边缘节点与用户的距离 | 较近 | 较远 |
| **数据分发** | 小文件分发能力 | 可实现，但受限于各节点之间网络 | 可实现 |
| 大文件分发能力 | 可基本实现，但全网分发时间较长 | 可实现，分发时间较快 |
| 分发成本 | 较高 | 低 |
| **数据存储** | 海量数据存储能力 | 可实现，但受限于存储成本，具有海量存储的节点不多 | 可实现 |
| WEB数据存储能力 | 可实现 | 可实现 |

