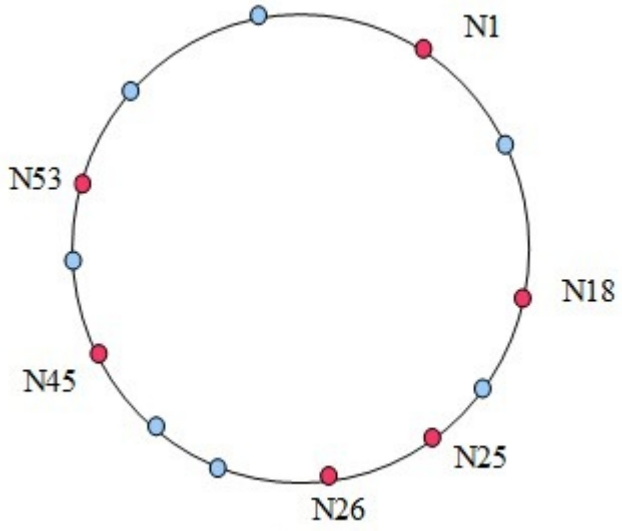
可行性分析

1.理论依据

1.1Chord 算法（结构化网络）

Chord 通过把 Node 和 Key 映射到相同的空间而保证一致性哈希，为了保证哈希的非重复性，Chord 选择 SHA-1 作为哈希函数，SHA-1 会产生一个 2160 的空间，每项为一个 16 字节（160bit）的大整数。我们可以认为这些整数首尾相连形成一个环，称之为 Chord 环。整数在 Chord 环上按大小顺时针排列，Node（机器的 IP 地址和 Port）与 Key（资源标识）都被哈希到 Chord 环上，这样我们就假定了整个 P2P 网络的状态为一个虚拟的环，因此我们说 Chord 是结构化的 P2P 网络。

称 Chord 环上的每个节点为标志符
如果某个 Node 映射到了某个标志符，则继续称该标准符为 Node
按顺时针，节点前面的成为前继(predecessor),节点后面的成为后继（successor)；
同理，第一个 predecessor 称之为直接前继，第一个 successor 称之为直接后继
如图：



Chord 环(红色点为 Node，蓝色为标志符)

很显然，分布在 Chord 环上的 Node 数远远小于标志符数（2160 是一个无法衡量的天文数字），这样 Chord 环上的 Node 就会很稀疏地分布在 Chord 环上，理论上应该是随机分布，但如前面一致性哈希的讨论，如果节点数量不多，分布肯定是不均匀的，可以考虑增加虚拟节点来增加其平衡性，如果在节点较多（比如大型的 P2P 网络有上百万的机器）就不必引入虚拟节点。

任何查找只要沿 Chord 环一圈结果肯定可以找到，这样的时间复杂度是 O(N)，N 为网络节点数，但对一个上百万节点，且节点经常加入、退出的 P2P 网络来说，O(N)是不可忍受的，因此 Chord 提出了下面非线性查找的算法：

1. 每个节点都维护一个 Finger 表，该表长度为 m（m 就是位数，在 Chord 中为 160），该表的第 i 项存放节点 n 的第(n+2i-1) mod 2m 个 successor(1<=i<=m)
2. 每个节点都维护一个 predecessor 和 successor 列表，该列表的作用是能快速定位前继和后继，并能周期性检测前继和后继的健康状态
3. 就是说存放的 successor 是按 2 的倍数等比递增，自所以取模是因为最后的节点的 successor 是开始的几个节点，比如最大的一个节点的下一个节点定义为第一个节点
4. 资源 Key 存储在下面的 Node 上：沿 Chord 环，hash(Node)>=hash(key)的第一个 Node，我们称这个 Node 为这个 Key 的 successor
5. 给定一个 Key，按下面的步骤查找其对应的资源位于哪个节点，也就是查找该 Key 的 successor：（假如查找是在节点 n 上进行）

- 查看 Key 的哈希是否落在节点 n 和其直接 successor 之间，若是结束查找，n 的 successor 即为所找
- 在 n 的 Finger 表中，找出与 hash(Key)距离最近且<hash(Key)的 n 的 successor，该节点也是 Finger 表中最接近 Key 的 predecessor，把查找请求转发到该节点
- 继续上述过程，直至找到 Key 对应的节点

从直觉上来说，上次查找过程应该是指数收敛的，类似二分法的查找，收敛速度应该是很快的；反过来，查找时间或路由复杂度应该是对数即的，证明略。

1.2Gnutella 协议（洪泛机制）

该机制应用于分布式非结构化拓扑中。查询节点在覆盖网络上发送一个资源发现请求，该请求被洪泛到直接相连的所有节点，这些节点再向其直接相连的邻居洪泛该请求，直到该请求被响应或达到最大洪泛次数。

最早的 Gnutella 使用洪泛机制发现网络中的文件。Gnutella 协议使用以下 4 种消息：

1） Ping：节点使用该消息宣告自己。

2） Pong：对 Ping 消息的响应，包含响应主机的 IP 地址、能接受响应的端口、本机共享文件的数量和本机所有共享文件的空间大小。

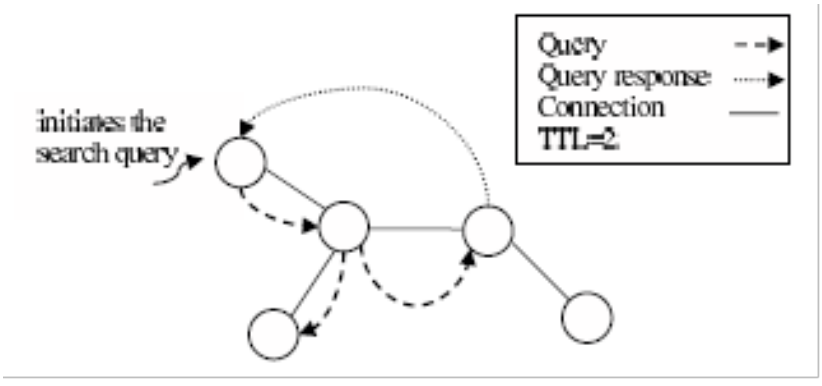
3） Query：搜索请求，包含一个搜索字符串和对响应主机的最小速度要求。

4） QueryHit：对 Query 消息的响应，包含响应主机的 IP 地址、能接受连接的端口、连线速度和响应查询的结果集，查询结果集包含匹配的文件数量以及每个匹配文件的索引、文件大小及文件名（可以根据文件名的部分或者完全匹配进行查找）。

一个 Gnutella 客户机通过与另一个当前在网络中的客户机建立连接来使自己与网络相连。获取另一个客户机的地址不在 Gnutella 协议的定义中，目前客户机地址缓存是自动获取 Gnutella 客户机地址的方式。一旦网络上另一个客户机的地址被获取，一个与该客户机的 TCP/IP 连接将被创建。加入 Gnutella 网络后，节点向每一个所连接的节点发送一个 Ping 消息，收到 Ping 消息的每个节点发送一个 Pong 消息，同时也会将 Ping 消息继续向其邻居传播。Gnutella 最初使用 TTL 来限制 Ping 和 Query 的传播范围，每一次消息转发时 TTL 减 1，减为 0 时消息被丢弃。为避免循环每个消息都带有全局唯一的标识符，用于检测和丢弃重复的消息。当收到一个 QueryHit 消息时，表明在响应主机上找到文件。下图是查询与响应过程的示意图。

尽管洪泛协议在小规模网络中可以获得最好的查询结果，但扩放性不好。在较大的网络中，TTL 限制了每个节点的查询消息可以到达的范围，从而不能保证一定能找到最佳匹配；但如果取消 TTL，则网络中将充斥大量的查询消息。

1.3Gossip 协议（数据一致性）



Gossip Protocol 利用一种随机的方式将信息散播到整个网络中。正如 Gossip 本身的含义一样，Gossip 协议的工作流程即类似于绯闻的传播，或

者流行病的传播。具体而言 Gossip Protocol 可以分为 Push-based 和 Pull-based 两种。Push-based Gossip Protocol 的具体工作流程如下：

1. 网络中的某个节点随机的选择其他 b 个节点作为传输对象。
2. 该节点向其选中的 b 个节点传输相应的信息
3. 接收到信息的节点重复完成相同的工作

Pull-based Gossip Protol 的协议过程刚好相反：

1. 某个节点 v 随机的选择 b 个节点询问有没有最新的信息
2. 收到请求的节点回复节点 v 其最近未收到的信息

当然，为了提高 Gossip 协议的性能，还有基于 Push-Pull 的混合协议。同时需要注意的是 Gossip 协议并不对网络的质量做出任何要求，也不需要 loss-free 的网络协议。Gossip 协议一般基于 UDP 实现，其自身即在应用层保证了协议的 robustness。

在 Gossip 性能中，我们可以认为: β=b/n（因为对每个节点而言，被其他节点选中的概率就是 b/n)。我们令 t=clog(n)，可以得到:
y≈(n+1)−1/n^(cb−2)
这表明，仅需要 O(log(n))个回合，gossip 协议即可将信息传递到所有的节点。 根据分析可得，Gossip 协议具有以下的特点:

1. 低延迟。仅仅需要 O(log(n))个回合的传递时间。
2. 非常可靠。仅有 1/n^(cb−2)个节点不会收到信息。
3. 轻量级。每个节点传送了 cb*log(n)次信息。

于此同时，Gossip 协议的容错性比较高，例如，50 的丢包率等价于使用 b/2 带代替 b 进行分析；50 的节点错误等价于使用 n/2 来代替 n，同时使用 b/2 来代替 b 进行分析，其分析结果不用带来数量级上的变化。

1.4NAT 概念

NAT（Network Address Translation，网络地址转换）是 1994 年提出的。当在专用网内部的一些主机本来已经分配到了本地 IP 地址（即仅在本专用网内使用的专用地址），但现在又想和因特网上的主机通信（并不需要加密）时，可使用 NAT 方法。即让地址重用。

最先提出的是基本的 NAT，它的产生基于如下事实：一个私有网络（域）中的节点中只有很少的节点需要与外网连接（这是在上世纪 90 年代中期提出的）。那么这个子网中其实只有少数的节点需要全球唯一的 IP 地址，其他的节点的 IP 地址应该是可以重用的。

因此，基本的 NAT 实现的功能很简单，在子网内使用一个保留的 IP 子网段，这些 IP 对外是不可见的。子网内只有少数一些 IP 地址可以对应到真正全球唯一的 IP 地

址。如果这些节点需要访问外部网络，那么基本 NAT 就负责将这个节点的子网内 IP 转化为一个全球唯一的 IP 然后发送出去。(基本的 NAT 会改变 IP 包中的原 IP 地址，但是不会改变 IP 包中的端口)

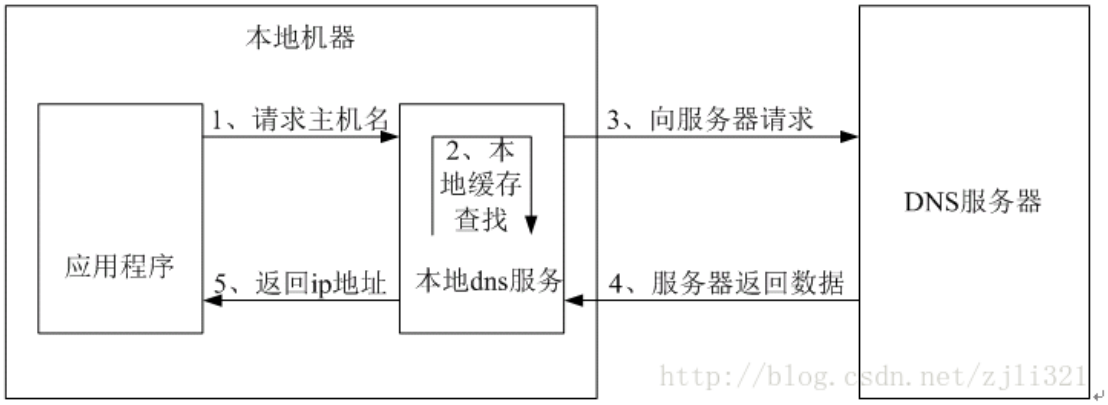
1.5 路由收敛原理

路由收敛的含义是指互联网中所有路由器运行着相同的、精确的、足以反映当前互联网拓扑结构的路由信息。
网络服务提供商用收敛时间来度量路由器设计和网络体系结构的性能。
路由器的收敛性与许多因素有关，例如，下层事件检测时间（路由失败，协议失败），SPF 处理时间，协议公告时间，以及转发信息库（FIB）的更新时间等。
通过路由收敛可以使路由域中所有路由器对当前的网络结构和路由转发达成一致的状态。
收敛时间是指从网络的拓扑结构发生变化到网络中所有路由设备中路由表重新保持一致的状态转换过程。
触发条件：
路由器失效、连接失效、管理度量调整等

1.6DNS 相关原理

DNS（Domain Name System，域名系统），万维网上作为域名和 IP 地址相互映射的一个分布式数据库，能够使用户更方便的访问互联网，而不用去记住能够被机器直接读取的 IP 数串。通过域名，最终得到该域名对应的 IP 地址的过程叫做域名解析（或主机名解析）。
dns 服务器主要做三个工作：
1、记录 dns 服务器的 ip 地址（请求进行域名解析的地方）；
2、向 dns 服务器请求进行域名解析（dns 实质工作）
3、缓存已经解析过的主机名与 ip 地址对应关系（这样已缓存的就不需要再向 dns 服务器请求解析了）。

一个 dns 的解析过程如下：
1、 应用程序请求一个主机名解析，如：应用程序跟 dns 说，我需要知道 www.baidu.com 的 ip 地址，你给我查一下；
2、 Dns 先在本地缓存中查询，若查到，返回 ip 地址给应用，流程结束。若没查到， 进行
下一步；
3、 Dns 把需要查询的主机名打包进 dns 数据包，然后把 dns 数据包发送到 dns 服务；
4、 DNS 服务器返回查询的主机名的 dns 数据包；
5、 Dns 解析数据包，把主机名对应的 ip 地址返回给应用程序；



流程图

1.7CAP 原理和 BASE 理论

CAP 由 Eric Brewer 提出，是 Eric Brewer 在 Inktomi 期间研发搜索引擎、分布式 web 缓存时得出的关于数据一致性(consistency)、服务可用性(availability)、分区容错性(partition-tolerance)的猜想，并在提出两年后被证明成立，成为我们熟知的 CAP 定理：

1. 数据一致性(consistency)：如果系统对一个写操作返回成功，那么之后的读请求都必须读到这个新数据；如果返回失败，那么所有读操作都不能读到这个数据，对调用者而言数据具有强一致性(strong consistency) (又叫原子性 atomic、线性一致性 linearizable consistency)
2. 服务可用性(availability)：所有读写请求在一定时间内得到响应，可终止、不会一直等待
3. 分区容错性(partition-tolerance)：在网络分区的情况下，被分隔的节点仍能正常对外服务

在某时刻如果满足 AP，分隔的节点同时对外服务但不能相互通信，将导致状态不一致，即不能满足 C；如果满足 CP，网络分区的情况下为达成 C，请求只能一直等待，即不满足 A；如果要满足 CA，在一定时间内要达到节点状态一致，要求不能出现网络分区，则不能满足 P。

C、A、P 三者最多只能满足其中两个，和 FLP 定理一样，CAP 定理也指示了一个不可达的结果(impossibility result)。

BASE 是 Basically Available（基本可用）、Soft state（软状态）和 Eventually consistent（最终一致性）三个短语的缩写。BASE 理论是对 CAP 中一致性和可用性权衡的结果，其来源于对大规模互联网系统分布式实践的总结， 是基于 CAP 定理逐步演化而来的。BASE 理论的核心思想是：即使无法做到强一致性，但每个应用都可以根据自身业务特点，采用适当的方式来使系统达到最终一致性。接下来看一下 BASE 中的三要素：

1、基本可用

基本可用是指分布式系统在出现不可预知故障的时候，允许损失部分可用性----注意，这绝不等价于系统不可用。比如：

（1） 响应时间上的损失。正常情况下，一个在线搜索引擎需要在 0.5 秒之内返回给用户相应的查询结果，但由于出现故障，查询结果的响应时间增加了 1~2 秒

（2） 系统功能上的损失：正常情况下，在一个电子商务网站上进行购物的时候，消费者几乎能够顺利完成每一笔订单，但是在一些节日大促购物高峰的时候，由于消费者的购物行为激增，为了保护购物系统的稳定性，部分消费者可能会被引导到一个降级页面

2、软状态
软状态指允许系统中的数据存在中间状态，并认为该中间状态的存在不会影响系统的整体可用性，即允许系统在不同节点的数据副本之间进行数据同步的过程存在延时

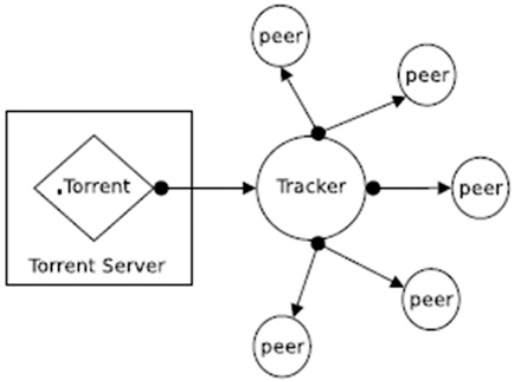
3、最终一致性
最终一致性强调的是所有的数据副本，在经过一段时间的同步之后，最终都能够达到一个一致的状态。因此，最终一致性的本质是需要系统保证最终数据能够达到一致，而不需要 实时保证系统数据的强一致性。
总的来说，BASE 理论面向的是大型高可用可扩展的分布式系统，和传统的事物 ACID 特性是相反的，它完全不同于 ACID 的强一致性模型，而是通过牺牲强一致性来获得可用性，并允许数据在一段时间内是不一致的，但最终达到一致状态。但同时，在实际的分布式场景中，不同业务单元和组件对数据一致性的要求是不同的，因此在具体的分布式系统架构设计过程中，ACID 特性和 BASE 理论往往又会结合在一起。

2.技术依据

2.1 P2P 技术

2.1.1 P2P 技术概论

对等（peer-to-peer，P2P）网络的出现打破了这种传统的客户-服务器模式。在对等网络中，每个节点的地位都是相同的，每个节点都有一些资源（如处理能力、存储空间、网络带宽、内容等）可以提供给其它节点，每个节点也都可以使用其它节点的资源，节点之间直接共享资源，不需要服务器的参与。也就是说，对等网络中的节点具备客户和服务器双重特性，可以同时作为服务的使用者和提供者。
当前的 P2P 主要有以下几种应用
（1）提供文件或其他内容共享的 P2P 网络
典型例子有 Nepster、Gnutella、Bit Torrent 等。



BT 网络示意图

BT 网络主要由 Torrent 服务器、Tracker 服务器、种子节点和下载节点组成（如上图所示）。Torrent 服务器负责 Torrent 文件的存储和检索，Tracker 服务器负责维护网络中的节点，种子节点负责提供一份文件的完整内容。BT 的基本原理是首先种子节点把一个文件分成了 N 个分片（默认是 256KB），节点 X 从种子节点随机下载第 L 个分片，Y 随机下载第 M 个分片，然后节点 X 与 Y 可以交换彼此拥有的分片，这样既减轻了种子节点的负担，又提高了节点下载的速度与效率。而且，节点在下载的同时也在上传（其他节点下载该节点拥有的文件分片），即在享受别人提供的服务的同时也在贡献。

（2）P2P 媒体网

P2P 的传输形式非常适合于流媒体直播与点播。P2P 在当前一个非常广泛的应用就是网上实时电视。由于采用 P2P 方式，提供节目的成本很低，用户却可以得到较好的收视质量，因此这些软件迅速流行，如 PPLive 和 PPStream 等。

（3）即时通信交流和安全的 P2P 通讯与信息共享

VoIP 是一种全新的网络电话通信业务，目前最典型的就是 Skype 这款 VoIP 软件。Skype 的出现给传统电信业带来强烈的冲击，它从 2003 年下半年出现以来便广为流传。截至目前，Skype 全球注册用户数已达 2.5 亿，每天增加的会员有 15.5 万人。到 2005 年 3 月 14 日为止，Skype 在全球的通话量累计已经达到 60 亿分钟。Skype 仍在迅速向各个国家渗透，最新的统计表明：使用 Skype 技术呼叫的分钟数已经占到美国 VoIP 分钟数的 46.2%，这部分用户基本是“免费”享用电话业务的。

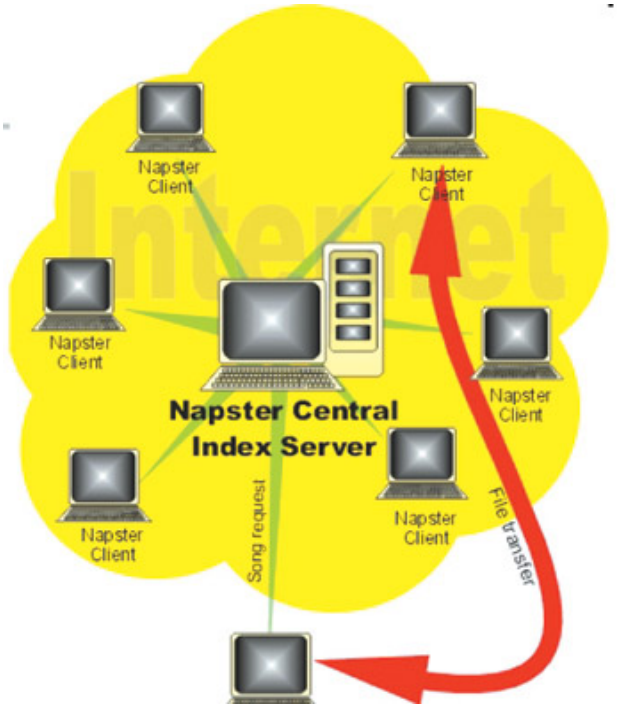
2.1.2 P2P 网络的拓扑结构及举例

P2P 目前在技术上比较权威的定义为：P2P 系统是一个由直接相连的节点所构成的分布式系统，这些节点能够为了共享内容、CPU 时间、存储或者带宽等资源而自组织形成一定的网络拓扑结构，能够在适应节点数目的变化和失效的同时维持可以接受的连接能力和性能，并且不需要一个全局服务器或者权威的中介支持。

根据 P2P 系统的分散程度，可以将 P2P 架构分成纯分散式和混合式两种。根据结构关系可以将 P2P 系统细分为四种拓扑形式：中心化拓扑，全分布式非结构化拓扑，全分布式结构化拓扑和半分布式拓扑。

（1）中心化拓扑

中心化拓扑也称集中目录式结构，因为仍然具有中心化的特点也称为非纯粹的 P2P 结构。最经典的微 Napster 结构（如下图所示）。Napster 使用一个中央索引服务器保存所有 Napster 用户上传的音乐文件索引和存放位置，当用户需要某个音乐文件时，先查询 Napster 中央索引服务器，服务器返回存有该文件的节点信息，用户再根据网络流量和延迟等信息选择合适的节点建立直接连接，而不必再经过中央索引服务器。

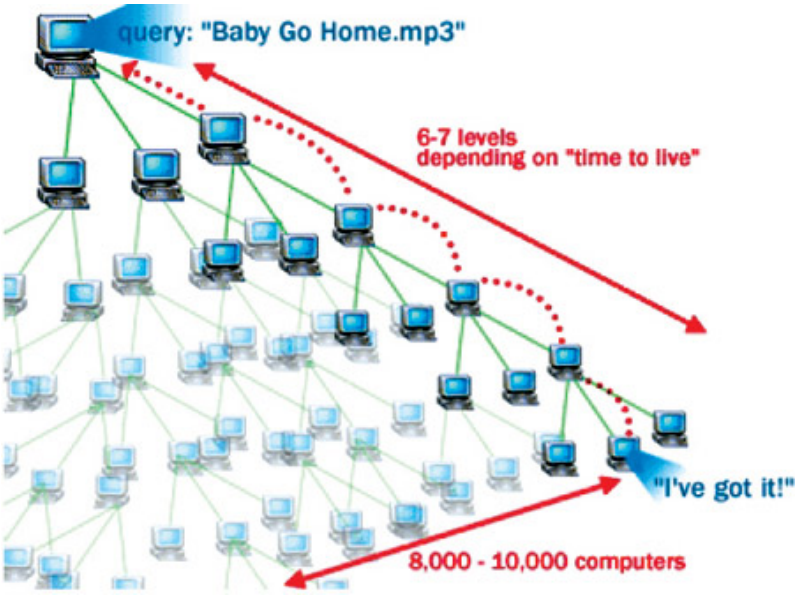


Napster 的拓扑结构

（2）全分布式非结构化拓扑

全分布式非结构化拓扑也称为纯 P2P 结构，它取消了中央服务器。每台机器是真正的对等关系，既是客户机又是服务器。Gnutella 是应用最广泛的全分布式非结构化拓扑。下图显示了洪泛的工作过程：当一台机器要下载一个文件时，它首先以文件名或关键字生成一个查询，把这个查询发送给与它相连的所有计算机；这些计算机如果存在这个文件，则与查询的机器建立连接，如果不存在则继续向自己的邻居节点转发这个查询，直到找到文件为止。为了控制搜索消息的传播范围，一般通过 TTL 值来控制查询的深度。

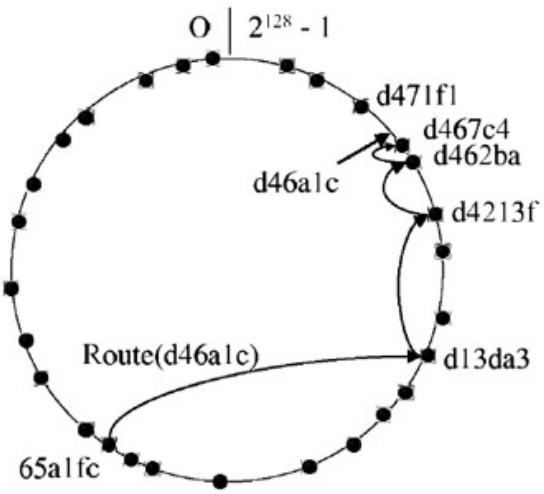
全分布式非结构化拓扑的 P2P 网络在覆盖网络中采用了随机图的组织方式，节点度数服从幂律分布，从而能够较快发现目的结点，面对网络的动态变化体现出较好的容错能力，因此具有较好的可用性。



Gnutella 的拓扑结构和文件检索方法

（3）全分布式结构化拓扑

全分布式结构化拓扑采用分布式散列表技术来组织网络中的节点。DHT 是一个由广域范围内大量节点共同维护的巨大散列表，散列表被分割成不连续的块，每个节点被分配一个散列块，并成为这个散列块的管理者。每个节点按照一定的方式被赋予一个惟一的节点标识符，资源对象的名字或关键词通过一个散列函数被映射为 128 位或 160 位的散列值，资源对象存储在 Node ID 与其散列值相等或相近的节点上。需要查找资源时,采用同样的方法可定位到存储该资源的节点。下图展示了基于 DHT 的节点组织，每个节点通过散列其 IP 地址，得到一个 128 位的节点标识符，所有节点标识符形成一个环形的 node ID 空间，范围从 0 到 2^{128}-1。

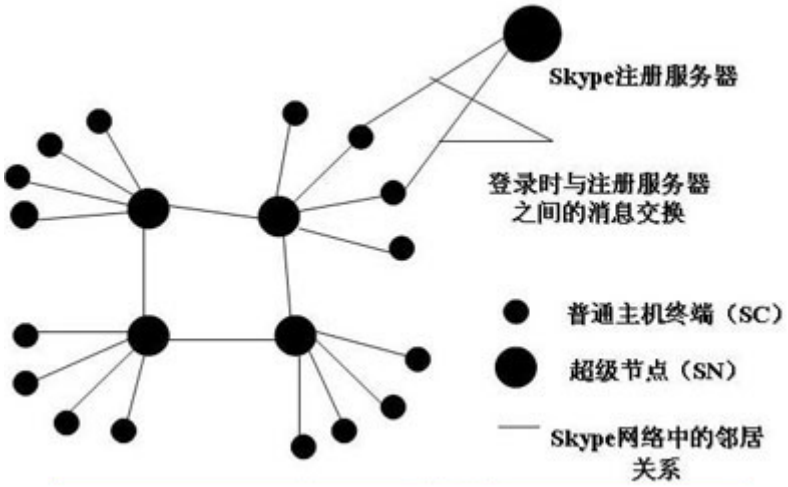


当前 DHT 结构的最大问题是维护机制较为复杂,尤其是节点频繁加入/退出造成的网络波动(Churn)会极大地增加 DHT 的维护代价。DHT 面临的另一个问题是 DHT 仅支持精确关键词匹配查询,无法支持内容/语义等复杂查询。

（4）半分布式结构

该机构亦称混合结构，其吸取了中心化结构和全分布式非结构化的优点，选择性能（处理、存储、带宽) 较高的节点作为超级节点，在各个超级节点上存储系统中其他部分节点的信息。发现算法仅在超级节点之间转发，超级节点再将查询请求转发给适当的叶子节点。

该结构较成熟的应用例子为 Skype。Skype 的网络中除了注册服务器，没有其他集中的服务器，只是将用户节点分为普通节点和超级节点。Skype 的系统连接结构如下图所示。



注册服务器是 Skype 惟一需要维护的设备，负责完成客户端的注册，存储并管理用户名和密码信息；当用户登录系统时，对用户进行身份认证。注册服务器还需要检验并保证用户名的全球惟一性。普通节点即普通主机终端，只需要下载了 Skype 的应用，就具有提供语音呼叫和文本消息传送的能力。超级节点实际上是满足某些要求的普通节点，这些要求包

括：具有公网地址、具有足够的 CPU、存储空间足够大、具有足够的网络带宽。换言之任何符合条件的主机终端都可以成为超级节点。

2.2 编码技术-Erasure Code

2.2.1 概述

Erasure Code 是一种编码技术，它可以将 n 份原始数据，增加 m 份数据，并能通过 n+m 份中的任意 n 份数据，还原为原始数据。即如果有任意小于等于 m 份的数据失效，仍然能通过剩下的数据还原出来。

2.2.2 分类

纠删码技术在分布式存储系统中的应用主要有三类，阵列纠删码（Array Code: RAID5、RAID6 等）、RS(Reed-Solomon)里德-所罗门类纠删码和 LDPC(LowDensity Parity Check Code)低密度奇偶校验纠删码。

其应用场景分别为：

a) RAID--磁盘阵列存储
b) RS--云存储
c) LDPC--通信、视频和音频编码等领域。

2.2.3Reed-Solomon Code

RS code 是基于有限域的一种编码算法，有限域又称为 Galois Field，是以法国著名数学家伽罗华（Galois）命名的，在 RS code 中使用 GF(2^w)，其中$2^w \geq n + m$.

RS code 的编解码定义：

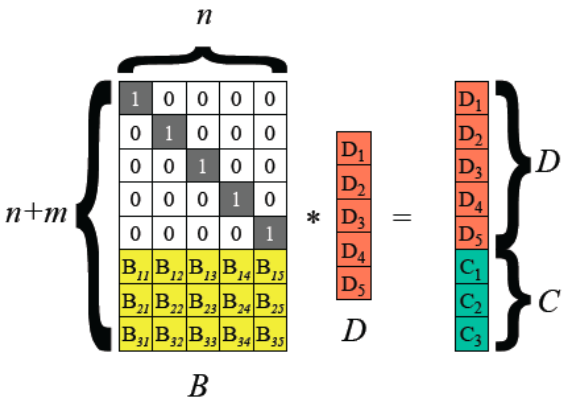
编码：给定 n 个数据块（Data block）D1、D2……Dn，和一个正整数 m，RS 根据 n 个数据块生成 m 个编码块（Code block），C1、C2……Cm。

解码：对于任意的 n 和 m，从 n 个原始数据块和 m 个编码块中任取 n 块就能解码出原始数据，即 RS 最多容忍 m 个数据块或者编码块同时丢失。

2.2.4RS code 编码原理

RS 编码以 word 为编码和解码单位，大的数据块拆分到字长为 w（取值一般为 8 或者 16 位）的 word，然后对 word 进行编解码。

把输入数据视为向量$D = (D_1, D_2, D_3, \dots, D_n)$, 编码后数据视为向量
$(D_1, D_2, D_3, \dots, D_n, C_1, C_2, \dots, C_m)$，运算过程如图：



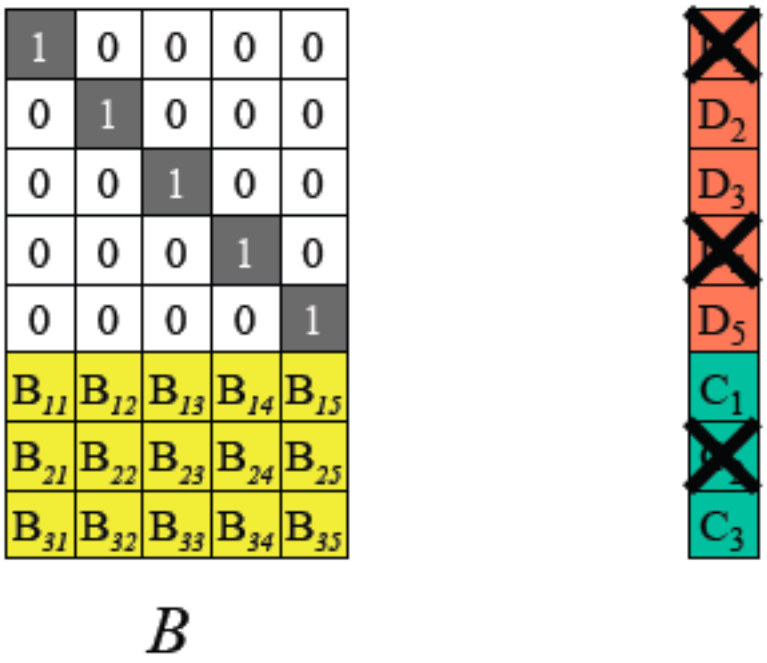
上图最左边是编码矩阵（或称为生成矩阵、分布矩阵，Distribution Matrix），编码矩阵需要满足任意 n*n 子矩阵可逆。

为方便数据存储，编码矩阵上部是单位阵（n 行 n 列），下部是 m 行 n 列矩阵。下部矩阵可以选择范德蒙德矩阵或柯西矩阵。

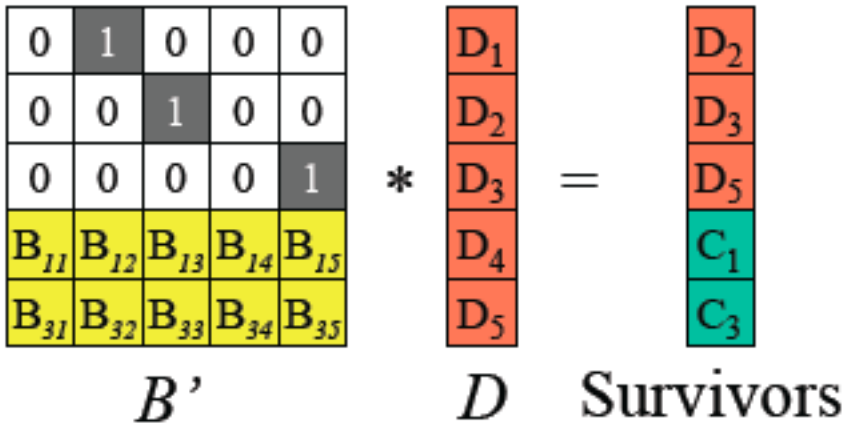
2.2.5RS code 编码数据恢复原理

RS 最多能容忍 m 个数据块被删除。 数据恢复的过程如下：

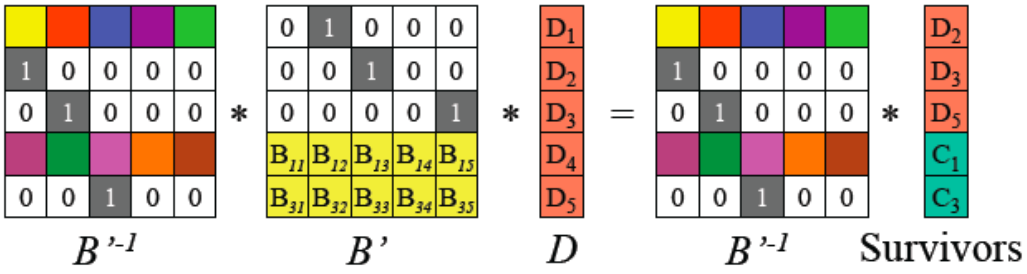
（1）假设D_1, D_4, C_2丢失，从编码矩阵中删掉丢失的数据块/编码块对应的行：



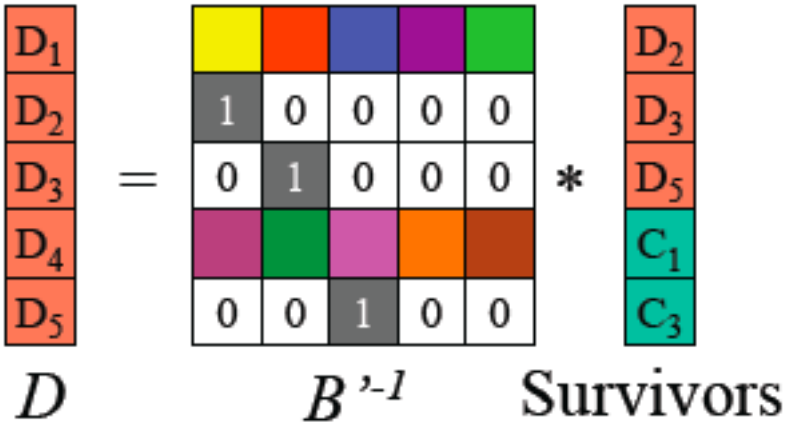
根据图 1 所示 RS 编码运算等式，可以得到如下 B' 以及等式：



(2) 由于B′是可逆的，记B′的逆矩阵为B'^{-1}，则$B'B'^{-1}$= I单位矩阵。两边左乘B′逆矩阵：



(3)得到如下原始数据 D 的计算公式:



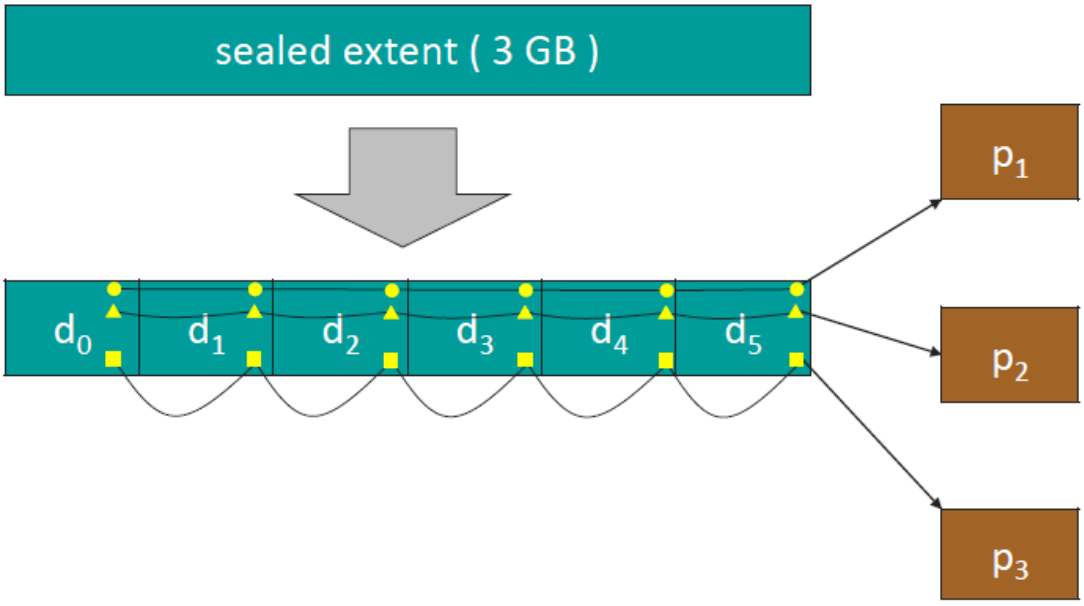
对 D 重新编码，可得到丢失的编码码.

2.2.6 各个公司的使用案例:

Google: RS(6,3)

Google GFS II 中采取了最基本的 RS(6，3)编码，将一个待编码的数据单元(Data Unit)分为 6 个 data block，再添加 3 个 parity block，最多可容纳三个数据块的丢失。

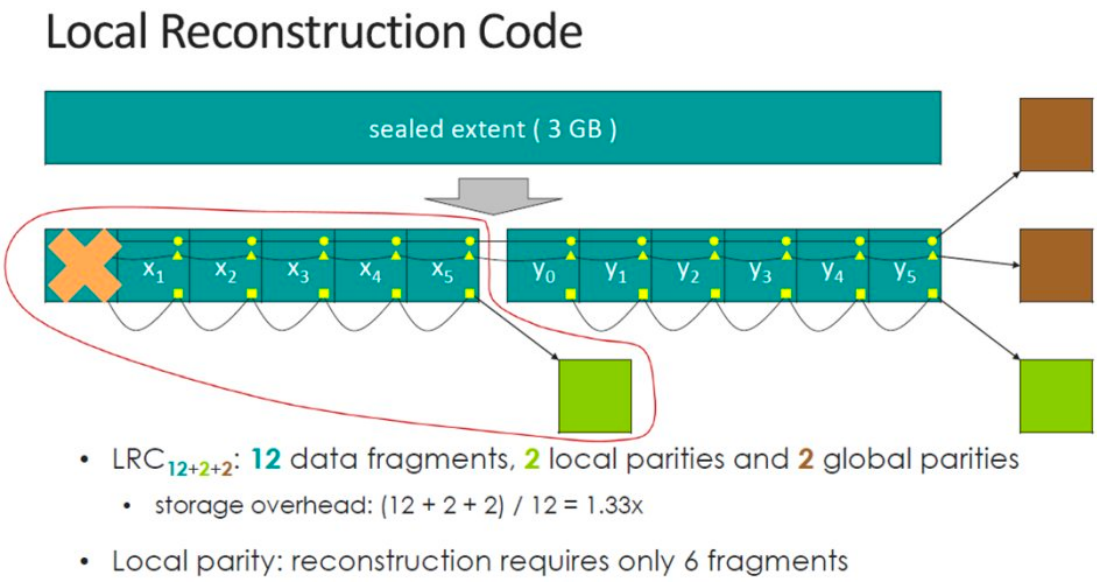
其存储的 space overhead 为 1.5，数据恢复的网络 I/O 开销为：恢复任意一个数据亏啊需要 6 此 I/O，通过网络传输 6 个数据块



Microsoft: Erasure code in WAS (Windows Azure Stroage)

为减少数据恢复时的网络 I/O，微软采用了 LRC 编码策略。其核心思想为：将校验块分为全局校验块和局部校验块。

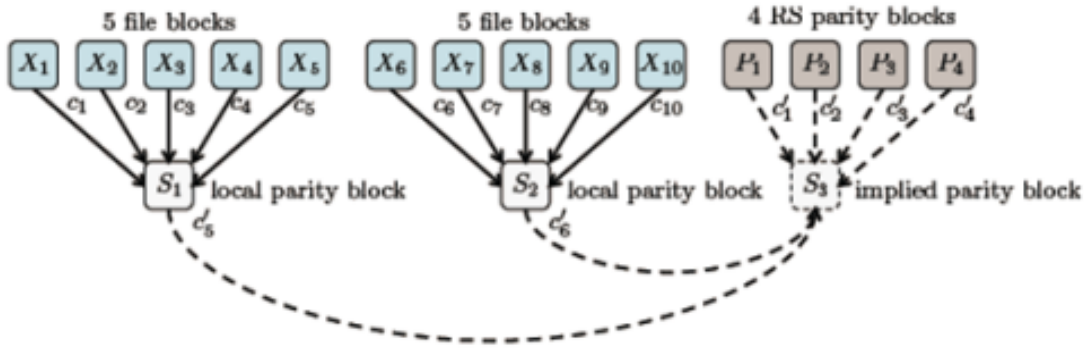
微软 LRC(12,2,2)编码将一个待编码数据块分为 12 个 data blocks，并进一步这 12 个数据块均分为两组，每组包含 6 个数据块，为每个组计算出一个局部校验块。并为所有的 12 个数据块计算出 2 个全局校验块。当发生任何数据块错误时，恢复代价由传统 RS(12,4)的 12 变为 6，恢复过程的网络开销减少一半。其 space overhead 为 1.33。



Facebook: 从 RS(10,4)到 LRC(10,6,5)

将数据分成 10 个数据块，之后添加 4 个校验块，并将 10 个原始数据块分为两组，并为每组生成一个局部校验块。

恢复数据块的错误时的网络 I/O 为 5 次，space overhead 为 1.6



2.2.7 总结：

Erasure Code 对存储空间使用率带来很大提升，但由于引入额外的编码解码运算，对分布式计算本省造成造成一定程度的性能损失。目前 EC 仅仅适用于对冷数据的离线处理阶段。LRC 减少了网络 I/O 传输的数据量，但是却牺牲了可靠性和空间利用率。

2.3 主流 DHT 协议

2.3.1 缓冲阵列路由协议(CARP，Cache Array Routing Protocol)

协议简介

CARP 是由微软公司的 Vinod Valloppillil 和宾西法尼亚大学的 Keith W. Ross 在 1997 年提出的。该协议可以将 URL 空间映射到一个仅有松散关联关系的 Web cache 服务器(在协议中称为“代理”，Proxy)阵列中。支持该协议的 HTTP 客户端可以根据要访问的 URL 智能选择目标代理。该协议解决了在代理阵列内分布存储内容的问题，避免了内容的重复存储，提高了客户端访问时 Web Cache 命中的概率。

哈希算法

哈希使用的关键字有 2 个，一个是代理的标识符(每个代理均有唯一的标识)，另一个是 URL 本身。存储内容时，每个代理负责缓冲哈希键值最大的 URL。这样，当缓冲代理阵

列发生少量变化时(新的代理加入或旧的代理退出)，原有的 URL 还有可能仍然被映射到原来的代理上，仍可以按照原有的方式访问。

路由算法

客户端(HTTP 浏览器)首先加载一个代理配置文件，该文件中存储了代理的标识符和 IP 地址等用于哈希的关键参数。浏览器在访问网页时，可以根据 URL 和代理标识获得代理的位置信息(IP 地址)，从而可以直接访问缓冲代理中的页面。

讨论

CARP 的哈希过程比较简单，路由查找更是简单到至多只有一跳(O(1))。但是 CARP 在 P2P 的应用环境中有一些致命的缺陷：

每个节点必须知道其它所有节点的信息。在大规模的重叠网环境中，由于可能存在大量的(数百万)节点，加之节点都是动态加入和退出网络，因此这一条件几乎不可能满足。

在缓冲阵列发生较大变化时(这在 P2P 网络中非常常见)，原有的 URL 和代理之间的对应关系可能发生改变，从而使得原有的配置文件失效。

2.3.2 一致性哈希(Consistent Hash)

协议简介

一致性哈希算法在 1997 年由麻省理工学院提出(参见 0)，设计目标是为了解决因特网中的热点(Hot pot)问题，初衷和 CARP 十分类似。一致性哈希修正了 CARP 使用的简单哈希算法带来的问题，使得 DHT 可以在 P2P 环境中真正得到应用。

哈希算法

一致性哈希提出了在动态变化的 Cache 环境中，哈希算法应该满足的 4 个适应条件：

平衡性(Balance)

平衡性是指哈希的结果能够尽可能分布到所有的缓冲中去，这样可以使得所有的缓冲空间都得到利用。很多哈希算法都能够满足这一条件。

单调性(Monotonicity)

单调性是指如果已经有一些内容通过哈希分派到了相应的缓冲中，又有新的缓冲加入到系统中。哈希的结果应能够保证原有已分配的内容可以被映射到新的缓冲中去，而不会被映射到旧的缓冲集合中的其他缓冲区。

简单的哈希算法往往不能满足单调性的要求，如最简单的线性哈希：

$x \rightarrow ax + b \bmod (P)$

在上式中，P 表示全部缓冲的大小。不难看出，当缓冲大小发生变化时(从 P1 到 P2)，原来所有的哈希结果均会发生变化，从而不满足单调性的要求。

哈希结果的变化意味着当缓冲空间发生变化时，所有的映射关系需要在系统内全部更新。而在 P2P 系统内，缓冲的变化等价于 Peer 加入或退出系统，这一情况在 P2P 系统

中会频繁发生，因此会带来极大计算和传输负荷。单调性就是要求哈希算法能够避免这一情况的发生。

分散性(Spread)

在分布式环境中，终端有可能看不到所有的缓冲，而是只能看到其中的一部分。当终端希望通过哈希过程将内容映射到缓冲上时，由于不同终端所见的缓冲范围有可能不同，从而导致哈希的结果不一致，最终的结果是相同的内容被不同的终端映射到不同的缓冲区中。这种情况显然是应该避免的，因为它导致相同内容被存储到不同缓冲中去，降低了系统存储的效率。分散性的定义就是上述情况发生的严重程度。好的哈希算法应能够尽量避免不一致的情况发生，也就是尽量降低分散性。

负载(Load)

负载问题实际上是从另一个角度看待分散性问题。既然不同的终端可能将相同的内容映射到不同的缓冲区中，那么对于一个特定的缓冲区而言，也可能被不同的用户映射为不同的内容。与分散性一样，这种情况也是应当避免的，因此好的哈希算法应能够尽量降低缓冲的负荷。

从表面上看，一致性哈希针对的是分布式缓冲的问题，但是如果将缓冲看作 P2P 系统中的Peer，将映射的内容看作各种共享的资源(数据，文件，媒体流等)，就会发现两者实际上是在描述同一问题。

路由算法

在一致性哈希算法中，每个节点(对应 P2P 系统中的 Peer)都有随机分配的 ID。在将内容映射到节点时，使用内容的关键字和节点的 ID 进行一致性哈希运算并获得键值。一致性哈希要求键值和节点 ID 处于同一值域。最简单的键值和 ID 可以是一维的，比如从 0000 到 9999 的整数集合。

根据键值存储内容时，内容将被存储到具有与其键值最接近的 ID 的节点上。例如键值为 1001 的内容，系统中有 ID 为 1000，1010，1100 的节点，该内容将被映射到 1000 节点。

为了构建查询所需的路由，一致性哈希要求每个节点存储其上行节点(ID 值大于自身的节点中最小的)和下行节点(ID 值小于自身的节点中最大的)的位置信息(IP 地址)。当节点需要查找内容时，就可以根据内容的键值决定向上行或下行节点发起查询请求。收到查询请求的节点如果发现自己拥有被请求的目标，可以直接向发起查询请求的节点返回确认；如果发现不属于自身的范围，可以转发请求到自己的上行/下行节点。

为了维护上述路由信息，在节点加入/退出系统时，相邻的节点必须及时更新路由信息。这就要求节点不仅存储直接相连的下行节点位置信息，还要知道一定深度(n 跳)的间接下行节点信息，并且动态地维护节点列表。当节点退出系统时，它的上行节点将尝试直接连接到最近的下行节点，连接成功后，从新的下行节点获得下行节点列表并更新自身的节点列表。同样的，当新的节点加入到系统中时，首先根据自身的 ID 找到下行

节点并获得下行节点列表，然后要求上行节点修改其下行节点列表，这样就恢复了路由关系。

讨论

一致性哈希基本解决了在 P2P 环境中最为关键的问题——如何在动态的网络拓扑中分布存储和路由。每个节点仅需维护少量相邻节点的信息，并且在节点加入/退出系统时，仅有相关的少量节点参与到拓扑的维护中。所有这一切使得一致性哈希成为第一个实用的 DHT 算法。

但是一致性哈希的路由算法尚有不足之处。在查询过程中，查询消息要经过 O(N)步(O(N)表示与 N 成正比关系，N 代表系统内的节点总数)才能到达被查询的节点。不难想象，当系统规模非常大时，节点数量可能超过百万，这样的查询效率显然难以满足使用的需要。换个角度来看，即使用户能够忍受漫长的时延，查询过程中产生的大量消息也会给网络带来不必要的负荷。

下文中讨论的几种 DHT 协议都对路由做出了优化，提出了各自的算法

2.3.3Chord 协议

Chord 在 2001 年由麻省理工学院提出(参见 0)，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。与前两种协议不同，Chord 专门为 P2P 应用设计，因此考虑了在 P2P 应用中可能遇到的特殊问题，这些内容将在路由的部分进行讨论。

哈希算法

Chord 使用一致性哈希作为哈希算法。在一致性哈希协议中并没有定义具体的算法，在 Chord 协议中将其规定为 SHA-1。

路由算法

Chord 在一致性哈希的基础上提供了优化的路由算法：

在 Chord 中，每个节点同样需要存储 m 个其他节点的信息，这些信息的集合被称为查询表(Finger Table)。一致性哈希中的节点同样具有这样的表格，但在 Chord 中，表格中的节点不再是直接相邻的节点，它们的间距(ID 间隔)将成 2i 的关系排列(i 表示表中的数组下标)。这样形成的节点之间路由关系实际上就是折半查找算法需要的排列关系。

在查询的过程中，查询节点将请求发送到与键值最接近的节点上。收到查询请求的节点如果发现自身存储了被查询的信息，可以直接回应查询节点(这与一致性哈希完全相同)；如果被查询的信息不在本地，就根据查询表将请求转发到与键值最接近的节点上。这样的过程一直持续到找到相应的节点为止。不难看出，查询过程实际上就是折半查找的过程。

经过 Chord 的优化后，查询需要的跳数由 O (N)减少到 O(log(N))。这样即使在大规模的 P2P 网络中(例如 N = 100,000,000)，查询的跳数也仅为 O(8)，每个节点仅需存储 27 个(log2100000000)其他节点的信息。

Chord 还考虑到多个节点同时加入系统的情况并对节点加入/退出算法作了优化。

讨论

Chord 算法本身具有如下优点：

负载平衡

这一优点来自于一致性哈希，也就是一致性哈希中提到的平衡性。所有的节点以同等的概率分担系统负荷，从而可以避免某些节点负载过大的情况。

分布性

Chord 是纯分布式系统，节点之间完全平等并完成同样的工作。这使得 Chord 具有很高的鲁棒性，可以抵御 DoS 攻击。

可扩展性

Chord 协议的开销随着系统规模(结点总数 N)的增加而按照 O(logN)的比例增加。因此 Chord 可以用于大规模的系统。

可用性

Chord 协议要求节点根据网络的变化动态的更新查询表，因此能够及时恢复路由关系，使得查询可以可靠地进行。

命名的灵活性

Chord 并未限制查询内容的结构，因此应用层可以灵活的将内容映射到键值空间而不受协议的限制。

2.3.4 内容寻址网络(Content-Addressable Network，CAN)

CAN 在 2001 年由加州大学伯克利分校提出(参见[3])。与 Chord 一样，CAN 也是 DHT 的一个变种。

哈希算法

CAN 的哈希算法与一致性哈希有所不同。Chord 中，哈希得到的键值总是一维的，而在 CAN 中，哈希的结果由 d 维的笛卡尔空间来表示。d 是一个由系统规模决定的常量。

路由算法

CAN 的路由查询将在 d 维笛卡尔空间中进行。

在 CAN 中，每个节点自身的 ID 经由哈希后得到的 d 维向量。经过这样的映射后，整个 P2P 系统将被映射到一个 d 维笛卡尔空间中，每个节点的位置由其自身 ID 决定。CAN 对邻居节点的定义并不要求成 2i 的关系排列，而是改为用在笛卡尔空间上相邻来表示：在 d 维笛卡尔空间中，2 个节点的 d 维坐标中有 d-1 维是相等的，剩余的一维是相邻的节点称之为相邻节点。

CAN 中的节点仅存储相邻节点表。由于在 d 维的空间中最多有 2d 个相邻的节点，因此节点的相邻节点表最多有 2d 个表项。

在查询的过程中，查询节点首先计算被查询内容的键值(d 维向量)，然后在节点列表中查找在笛卡尔空间中与该键值最为接近的相邻节点，找到后向该节点发送查询请求(这一策略被称为贪婪策略)。查询请求中将携带被查询内容的键值。收到查询请求的节点如果发现自身存储了被查询的信息，可以直接回应查询节点(这与一致性哈希完全相同)；如果被查询的信息不在本地，就根据相邻节点表将请求转发到与键值最接近的节点上。这样的过程一直持续到找到相应的节点为止。在查询过程中，被查询节点到目标节点的笛卡尔空间距离单调地减少。

如果查询节点或转发节点发现邻居节点表中无法找到可用的下一跳节点，则采用非结构化 P2P 常用的扩展环搜索(Expanding Ring Search，使用无状态，受控的泛洪算法在重叠网中搜索)以找到合适的(符合贪婪策略)下一节点。

经过 CAN 的优化后，查询需要的跳数由 O（N)减少到均值为(d/4)(n1/d)的随机制，考虑到 d 为常数，这一值可以表示为 O(n1/d)或 O(dn1/d)。

讨论

CAN 和 Chord 的主要区别在于路由算法不同。相比之下，在节点数量非常大时，CAN 的平均查询跳数要比 Chord 增加得更快。而且 CAN 查询过程中需要的运算量也要高于 Chord。但 CAN 使用的 d 为预先设置的常量，因此并不假设系统节点数量。在节点总数动态变化范围很大的系统中，CAN 的相邻节点表结构保持稳定，这在 P2P 的应用中也是很重要的优点。

2.3.5Pastry

Pastry 在 2001 年由位于英国剑桥的微软研究院和莱斯(Rice)大学提出(参见[4])。Pastry 也是 DHT 的一个变种。

哈希算法

Pastry 使用一致性哈希作为哈希算法。哈希所得的键值为一维(实际上使用的是 128bit 的整数空间)。Pastry 也没有规定具体应该采用何种哈希算法。

路由算法

在 Pastry 协议中，每个节点都拥有一个 128bit 的标识(Node Id)。为了保证 Node ID 的唯一性，一般由节点的网络标识(如 IP 地址)经过哈希得到。

Pastry 中的每个节点拥有一个路由表 R(Routing table)，一个邻居节点集 M(Neighborhood Set)和一个叶子节点集合 L(Leaf set)，它们一起构成了节点的状态表。

路由表 R 共有 logBN(B = 2b 为系统参数，典型值为 16，N 表示系统的节点总数)行，每行包括 B-1 个表项，每个表项记录了一个邻居节点的信息(节点标识、IP 地址、 当前状态等)。这样就形成了拥有(B-1)logBN 个条目的二维表格。路由表第 n 行的表项所记录的邻居节点的 Node ID 前 n 个数位和当前节点的前 n 个数位相同，而第 n + 1 个数位则

分别取从 0 到 B-1 的值(除了与当前节点第 n + 1 数位的值)。这样形成的路由表很类似 IP 路由中最长掩码匹配的算法。参数 b(或 B)大小非常关键：B 过大则节点需要维护很大的路由表，可能超出节点的负载能力，但路由表大些可以存储更多的邻居节点，在转发时更为精确。平均每次路由查找需要的跳数在 Pastry 中计算的结果是 logBN，因此 B 的选择反映了路由表大小和路由效率之间的折衷。

叶子节点集合 L 中存放的是在键值空间中与当前节点距离最近的|L|个节点的信息，其中一半节点标识大于当前节点，另一半节点标识小于当前节点。|L|的典型值为 2b 或者 2*2b。

邻居节点集合 M 中存放的是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”的定义在 Pastry 中非常类似 IP 路由协议中对距离的定义，也就是考虑到转发跳数、传输路径带宽、QoS 等综合因素后所得的转发开销(可以参见 OSPF 等路由协议)。Pastry 并未提供距离信息的获取方法，而是假设应用层可以通过某种手段(人工配制或自动协商)得到信息并配置邻居节点集合。|M|的典型值为 2b 或者 2*2b。

图 3 给出了一个 Pastry 节点状态表的例子，该图来源于[4]。

在节点状态表中，节点本身的 ID 为 10233102。叶子集合中有 8 项，每一项都代表一个当前节点已知的其他节点的信息。路由表共有 4*8 项，可以看出由上至下节点 ID 重合的位数(前缀)不断增加。邻居集合中的节点 ID 由于来源于应用层，一般没有规律性可循。

Pastry 的路由过程如下：

首先，路由查询消息中将携带被查询对象 ID(Object Id)，又称消息键值。当收到路由消息时，节点首先检查消息键值是否落在叶子节点集合的范围内。如果是，则直接把消息转发给叶子节点集合中节点标识和消息键值最接近的节点；否则就从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。

从上述过程中可以看出，每一步路由和上一步相比都更靠近目标节点，因此这个过程是收敛的。如果路由表不为空，每步路由至少能够增加一个前缀匹配数位，因此在路由表始终有效时，路由的步数至多为 logBN。

讨论

Pastry 的路由利用了成熟的最大掩码匹配算法，因此实现时可以利用很多现成的软件算法和硬件框架，可以获得很好的效率。

与 Chord 和 CAN 相比，Pastry 引入了叶子节点和邻居节点集合的概念。在应用层能够及时准确地获得这两个集合的节点信息时，可以大大加快路由查找的速度，同时降低因路由引起的网络传输开销；不过在动态变化的 P2P 网络中如何理想地做到这一点的确有很大的难度。

趋势分析

目前 DHT 算法的发展方向非常多，不断有新的改进算法被提出来。就笔者目前了解到的信息而言，至少有以下一些方向：

接近性(Proximity)

文中提到的 DHT 算法中，除了 Pasrty 以外，均未考虑重叠网络拓扑结构与真实的 IP 网络之间的重合关系。节点之间进行对等通信时，不会考虑优先选取距离自己最近的节点。这样就使得最终形成的重叠网结构混乱，效率低下。因此如何让节点获得并利用接近性信息就非常重要。

结构化

目前基于 DHT 的应用尚未大规模展开，很多工程上的细节问题尚待解决。例如：目前有很多种类的 P2P 应用，如文件存储和共享、电子邮件、流媒体等。这些应用在处理 P2P 路由算法、拓扑维护和信息检索上使用的方法均有很大差别，导致即使是同类的应用也无法实现互通。如何为各种 P2P 的应用抽象出一个通用的层次，也是目前研究的热点问题之一。

信息查询

基于分布式哈希表的查询是一种单关键字的精确匹配，尽管相对于非结构化系统它使得系统资源可被确定性地查询到，但它也极大地限制了查询的应用范围。目前有许多改进的结构化查询算法已经被提出来。

2.3.6 内容寻址网络(Content-Addressable Network，CAN)

CAN 在 2001 年由加州大学伯克利分校提出(参见[3])。与 Chord 一样，CAN 也是 DHT 的一个变种。

哈希算法

CAN 的哈希算法与一致性哈希有所不同。Chord 中，哈希得到的键值总是一维的，而在 CAN 中，哈希的结果由 d 维的笛卡尔空间来表示。d 是一个由系统规模决定的常量。

路由算法

CAN 的路由查询将在 d 维笛卡尔空间中进行。

在 CAN 中，每个节点自身的 ID 经由哈希后得到的 d 维向量。经过这样的映射后，整个 P2P 系统将被映射到一个 d 维笛卡尔空间中，每个节点的位置由其自身 ID 决定。CAN 对邻居节点的定义并不要求成 2i 的关系排列，而是改为用在笛卡尔空间上相邻来表示：在 d 维笛卡尔空间中，2 个节点的 d 维坐标中有 d-1 维是相等的，剩余的一维是相邻的节点称之为相邻节点。

CAN 中的节点仅存储相邻节点表。由于在 d 维的空间中最多有 2d 个相邻的节点，因此节点的相邻节点表最多有 2d 个表项。

在查询的过程中，查询节点首先计算被查询内容的键值(d 维向量)，然后在节点列表中查找在笛卡尔空间中与该键值最为接近的相邻节点，找到后向该节点发送查询请求(这一策略被称为贪婪策略)。查询请求中将携带被查询内容的键值。收到查询请求的节点如果发现自身存储了被查询的信息，可以直接回应查询节点(这与一致性哈希完全相同)；如果被查询的信息不在本地，就根据相邻节点表将请求转发到与键值最接近的节点上。这样的过程一直持续到找到相应的节点为止。在查询过程中，被查询节点到目标节点的笛卡尔空间距离单调地减少。

如果查询节点或转发节点发现邻居节点表中无法找到可用的下一跳节点，则采用非结构化 P2P 常用的扩展环搜索(Expanding Ring Search，使用无状态，受控的泛洪算法在重叠网中搜索)以找到合适的(符合贪婪策略)下一节点。

经过 CAN 的优化后，查询需要的跳数由 O（N)减少到均值为(d/4)(n1/d)的随机制，考虑到 d 为常数，这一值可以表示为 O(n1/d)或 O(dn1/d)。

讨论

CAN 和 Chord 的主要区别在于路由算法不同。相比之下，在节点数量非常大时，CAN 的平均查询跳数要比 Chord 增加得更快。而且 CAN 查询过程中需要的运算量也要高于 Chord。但 CAN 使用的 d 为预先设置的常量，因此并不假设系统节点数量。在节点总数动态变化范围很大的系统中，CAN 的相邻节点表结构保持稳定，这在 P2P 的应用中也是很重要的优点。

2.4Trie 树

2.4.1 简介

Trie 与二叉查找树不同，键不是直接保存在节点中，而是由节点在树中的位置决定。一个节点的所有子孙都有相同的前缀，也就是这个节点对应的字符串，而根节点对应空字符串。一般情况下，不是所有的节点都有对应的值，只有叶子节点和部分内部节点所对应的键才有相关的值。键不需要被显式地保存在节点中，而是由根节点到当前节点的完整路径上的所有字符依次连接而成。Trie 树有一些特性：

1）根节点不包含字符，除根节点外每一个节点都只包含一个字符。

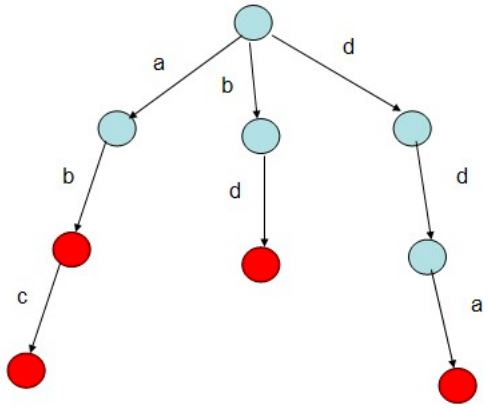
2）从根节点到某一节点，路径上经过的字符连接起来，为该节点对应的字符串。

3）每个节点的所有子节点包含的字符都不相同。

4）如果字符的种数为 n，则每个结点的出度为 n，这也是空间换时间的体现，浪费了很多的空间。

5）插入查找的复杂度为 O(n)，n 为字符串长度。

2.4.2 数据结构



Trie 树的根结点不包含任何信息，第一个字符串为"abc"，第一个字母为'a'，因此根结点中数组 next 下标为'a'-97 的值不为 NULL，其他同理，构建的 Trie 树如图所示，红色结点表示在该处可以构成一个单词。很显然，如果要查找单词"abc"是否存在，查找长度则为 O(len)，len 为要查找的字符串的长度。而若采用一般的逐个匹配查找，则查找长度为 O(len*n)，n 为字符串的个数。显然基于 Trie 树的查找效率要高很多。

如上图中：Trie 树中存在的就是 abc、ab、bd、dda 四个单词。在实际的问题中可以将标记颜色的标志位改为数量 count 等其他符合题目要求的变量。

已知 n 个由小写字母构成的平均长度为 10 的单词，判断其中是否存在某个串为另一个串的前缀子串。下面对比 3 种方法：

1、 最容易想到的：即从字符串集中从头往后搜，看每个字符串是否为字符串集中某个字符串的前缀，复杂度为 O(n^2)。

2、 使用 hash：我们用 hash 存下所有字符串的所有的前缀子串。建立存有子串 hash 的复杂度为 O(n*len)。查询的复杂度为 O(n)* O(1)= O(n)。

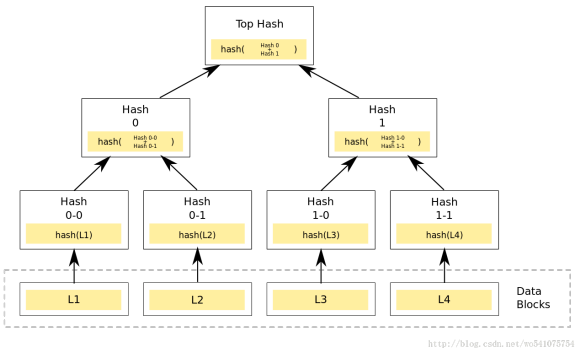
3、 使用 Trie：因为当查询如字符串 abc 是否为某个字符串的前缀时，显然以 b、c、d....等不是以 a 开头的字符串就不用查找了，这样迅速缩小查找的范围和提高查找的针对性。所以建立 Trie 的复杂度为 O(n*len)，而建立+查询在 trie 中是可以同时执行的，建立的过程也就可以成为查询的过程，hash 就不能实现这个功能。所以总的复杂度为 O(n*len)，实际查询的复杂度只是 O(len)。

2.6Merkle Hash Tree

2.6.1 简介

Merkle Tree，通常也被称作 Hash Tree，顾名思义，就是存储 hash 值的一棵树。Merkle 树的叶子是数据块(例如，文件或者文件的集合)的 hash 值。非叶节点是其对应子节点串联字符串的 hash。

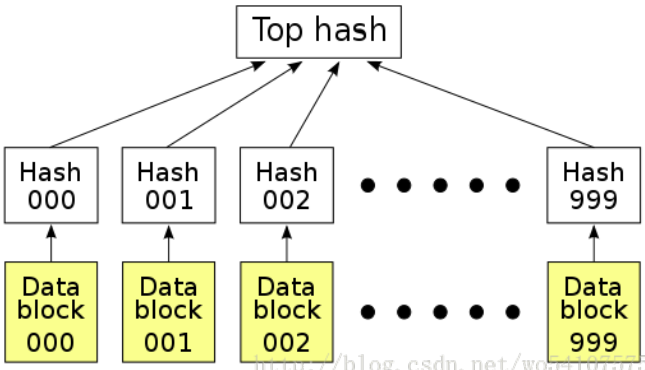
2.6.2Hash List



在点对点网络中作数据传输的时候，会同时从多个机器上下载数据，而且很多机器可以认为是不稳定或者不可信的。为了校验数据的完整性，更好的办法是把大的文件分割成小的数据块（例如，把分割成 2K 为单位的数据块）。这样的好处是，如果小块数据在传输过程中损坏了，那么只要重新下载这一快数据就行了，不用重新下载整个文件。

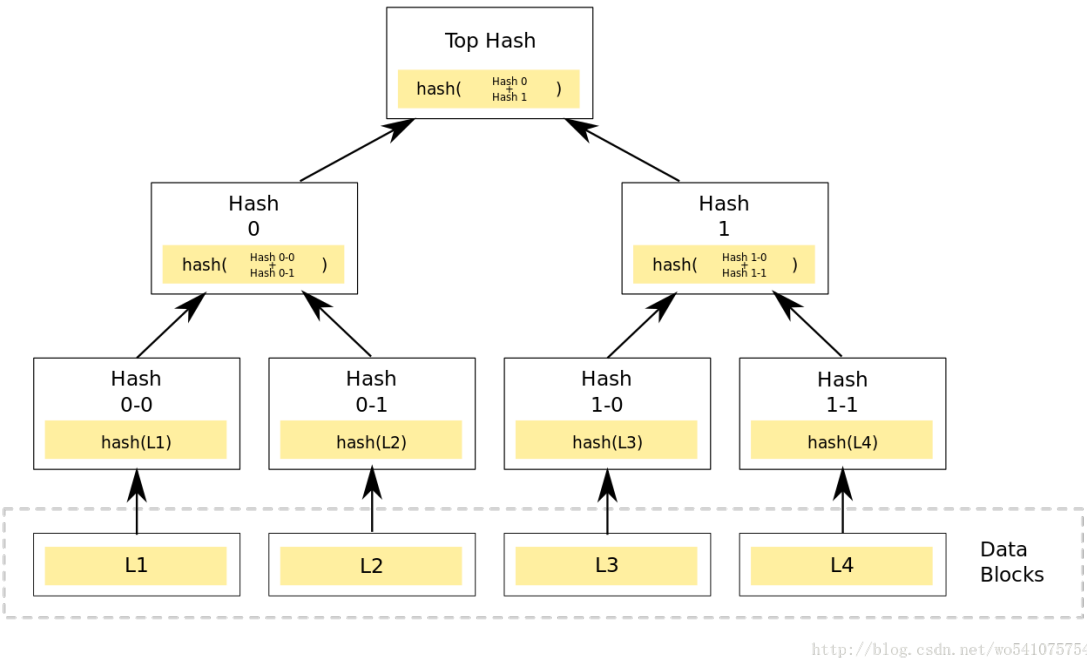
如何确定小的数据块没有损坏哪？只需要为每个数据块做 Hash。BT 下载的时候，在下载到真正数据之前，我们会先下载一个 Hash 列表。那么如何确定这个 Hash 列表本身是正确的？解决方法是把每个小块数据的 Hash 值拼到一起，然后对这个长字符串在作一次 Hash 运算，这样就得到

Hash 列表的根 Hash(Top Hash or Root Hash)。下载数据的时候，首先从可信的数据源得到正确的根 Hash，就可以用它来校验 Hash 列表了，
然后通过校验后的 Hash 列表校验数据块。



2.6.3Merkle Tree

在最底层，和哈希列表一样，我们把数据分成小的数据块，有相应地哈希和它对应。但是往上走，并不是直接去运算根哈希，而是把相邻的两个哈希合并成一个字符串，然后运算这个字符串的哈希，这样每两个哈希就得到了一个” 子哈希 “。如果最底层的哈希总数是单数，那到最后必然出现一个单身哈希，这种情况就直接对它进行哈希运算，所以也能得到它的子哈希。于是往上推，依然是一样的方式，可以得到数目更少的新一级哈希，最终必然形成一棵倒挂的树，到了树根的这个位置，这一代就剩下一个根哈希了，我们把它叫做 Merkle Root。
在 P2P 网络下载网络之前，先从可信的源获得文件的 Merkle Tree 树根。一旦获得了树根，就可以从其他从不可信的源获取 Merkle tree。通过可信的树根来检查接受到的 Merkle Tree。如果 Merkle Tree 是损坏的或者虚假的，就从其他源获得另一个 Merkle Tree，直到获得一个与可信树根匹配的 Merkle Tree。
Merkle Tree 和 Hash List 的主要区别是，可以直接下载并立即验证 Merkle Tree 的一个分支。因为可以将文件切分成小的数据块，这样如果有一块数据损坏，仅仅重新下载这个数据块就行了。如果文件非常大，那么 Merkle tree 和 Hash list 都很大，但是 Merkle tree 可以一次下载一个分支，然后立即验证这个分支，如果分支验证通过，就可以下载数据了。而 Hash list 只有下载整个 Hash list 才能验证。



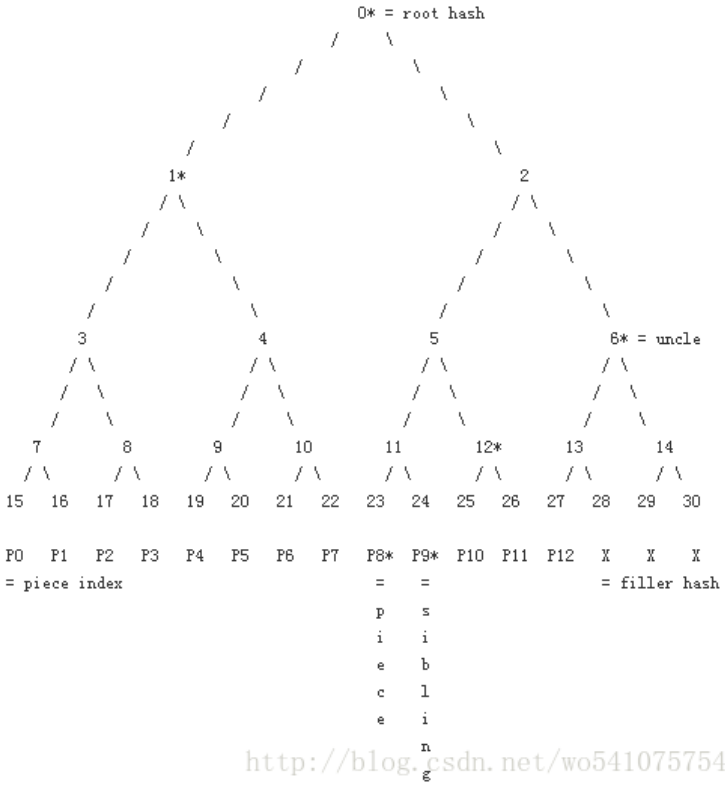
2.6.4Merkle Tree 的特点

1.MT 是一种树，大多数是二叉树，也可以多叉树，无论是几叉树，它都具有树结构的所有特点。
2.Merkle Tree 的叶子节点的 value 是数据集合的单元数据或者单元数据 HASH。

3.非叶子节点的 value 是根据它下面所有的叶子节点值，然后按照 Hash 算法计算而得出的。
通常，加密的 hash 方法像 SHA-2 和 MD5 用来做 hash。但如果仅仅防止数据不是蓄意的损坏或篡改，可以改用一些安全性低但效率高的校验和算法，如 CRC。

2.6.5Merkle Tree 的应用

1.数字签名
最初 Merkle Tree 目的是高效的处理 Lamport one-time signatures。 每一个 Lamport key 只能被用来签名一个消息，但是与 Merkle tree 结合可以来签名多条 Merkle。这种方法成为了一种高效的数字签名框架，即 Merkle Signature Scheme。
2.P2P 网络
在 P2P 网络中，Merkle Tree 用来确保从其他节点接受的数据块没有损坏且没有被替换，甚至检查其他节点不会欺骗或者发布虚假的块。BT 下载就是采用了 P2P 技术来让客户端之间进行数据传输，一来可以加快数据下载速度，二来减轻下载服务器的负担。
要进行下载必须从中心索引服务器获取一个扩展名为 torrent 的索引文件，torrent 文件包含了要共享文件的信息，包括文件名，大小，文件的 Hash 信息和一个指向 Tracker 的 URL。Torrent 文件中的 Hash 信息是每一块要下载的文件内容的加密摘要，这些摘要也可运行在下载的时候进行验证。大的 torrent 文件是 Web 服务器的瓶颈，而且也不能直接被包含在 RSS 或 gossiped around(用流言传播协议进行传播)。一个相关的问题便是大数据块的使用，因为为了保持 torrent 文件的非常小，那么数据块 Hash 的数量也得很小，这就意味着每个数据块相对较大。大数据块影响节点之间进行交易的效率，因为只有当大数据块全部下载下来并校验通过后，才能与其他节点进行交易。
解决上述两个问题的方法是用一个简单的 Merkle Tree 代替 Hash List。设计一个层数足够多的满二叉树，叶节点是数据块的 Hash，不足的叶节点用 0 来代替。上层的节点是其对应孩子节点串联的 hash。Hash 算法和普通 torrent 一样采用 SHA1。



2.7 最长掩码匹配

路由掩码最长匹配原则是指 IP 网络中当路由表中有多条条目可以匹配目的 IP 时，一般就采用掩码最长的一条作为匹配项并确定下一跳。

例如，考虑下面这个 IPV4 的路由表：

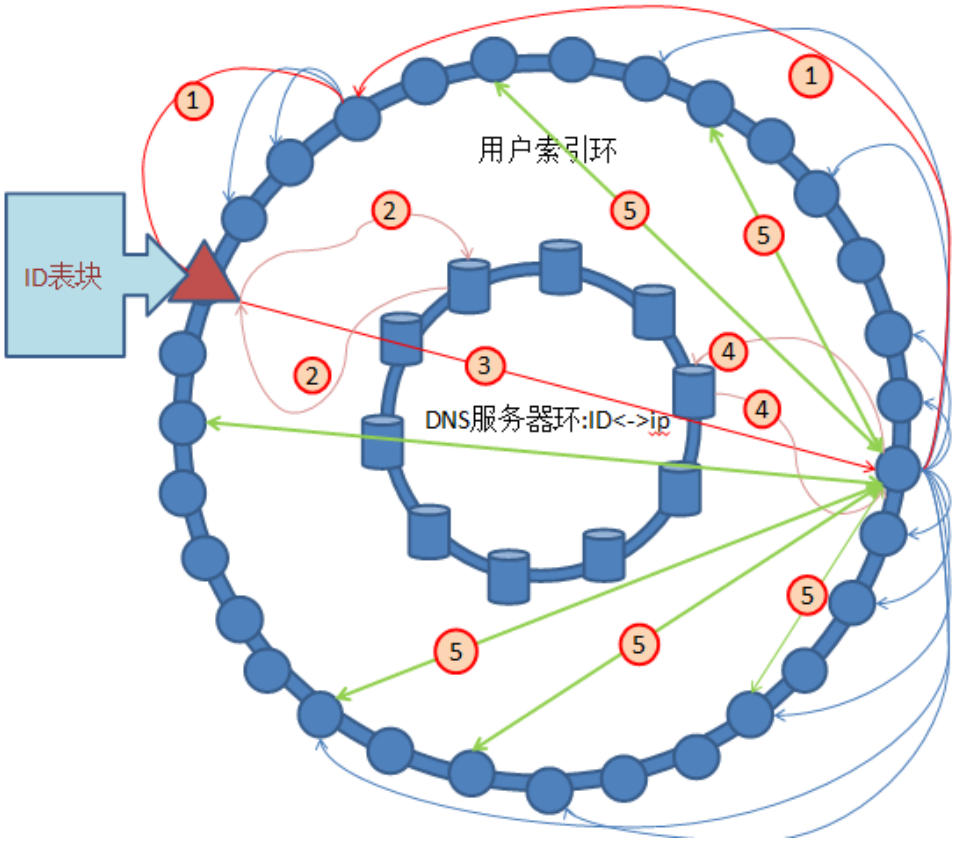
192.168.20.16/28 e0

192.168.0.0/16 s0

在要查找地址 192.168.20.19 时，不难发现上述两条都“匹配”，即这两条都包含要查找的目的地址。此时就应该根据最长掩码匹配原则，选择第一条进行匹配（更明确），所以数据包将通过 e0 发送出去。

3.设计方案

3.1 总体架构图



3.2 基本结构

1. DHT 环

- 标识符 ID：通过一致性 hash 映射得到 ID 标识符，用于唯一地标识一个逻辑节点
- 用户节点：采用 DHT 结构，随机分配每个节点的唯一标识符 ID
- 文件 ID 表块节点：根据文件名与 ID 表块块号拼接，进行一致性 hash 后得到存储这个 ID 表块的节点。

1. DNS 服务器环：提供 ID 与 ip 之间的映射关系，真/假 DNS？取决于 DNS 提供的服务是否足够，若服务功能不足够，使用较强节点作为 DNS 服务器环节点（逻辑结构）。

3.3 详细结构

用户节点：

在架构中，用户节点存储 1～2 部分内容：

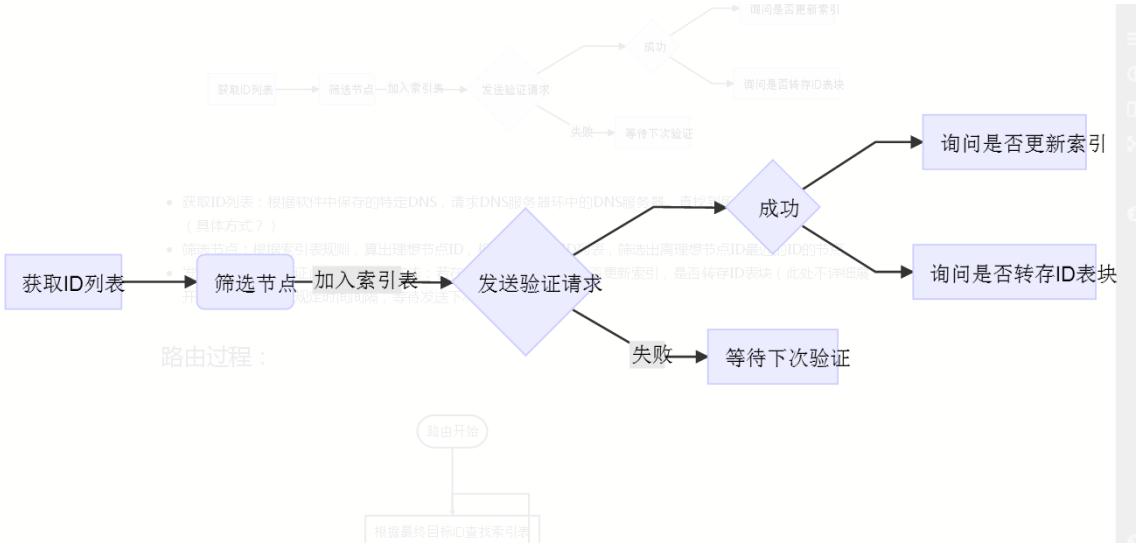
- 索引：1.采取 chord 的多分查找表 2.采取 CAN 的 d 维笛卡尔空间查找表

- ID 表块：若此用户节点被选为文件 ID 表块节点，则负责存此 ID 表块

DNS 节点：

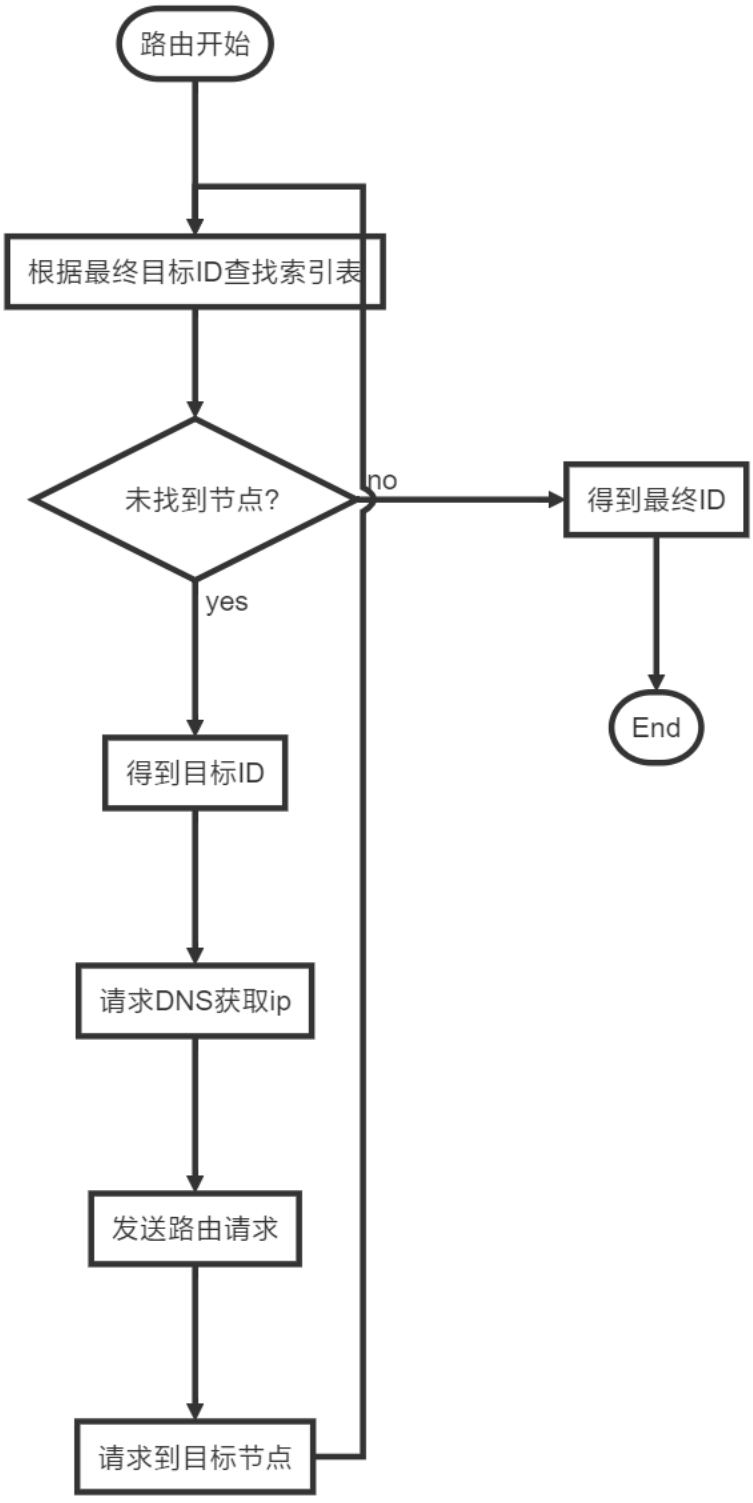
在小型网络中可以自己搭建服务器，存储数据库：包含所有 ID 与 ip 的映射关系（每一个都包含）。 若是搭建广义的 DNS 服务器，可以额外增加一些功能，简化网络。

3.4 入网过程



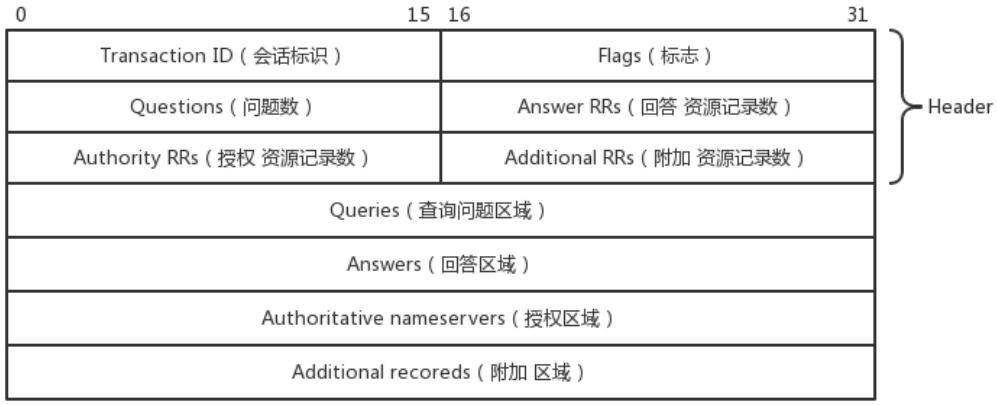
- 获取 ID 列表：根据软件中保存的特定 DNS，请求 DNS 服务器环中的 DNS 服务器，查找到所有的 ID 列表。
- 筛选节点：根据索引表规则，算出理想节点 ID，根据请求到的 ID 列表，筛选出离理想节点 ID 最近的 ID 的节点。
- 发送验证请求：验证此 ID 此时是否在线：若在线，对方节点查询是否更新索引，是否转存 ID 表块（此处不详细展开），若不在线，规定时间间隔，等待发送下次验证请求。

3.5 路由过程


路由开始
根据最终目标ID查找索引表
未找到节点?
no
得到最终ID
End
yes
得到目标ID
请求DNS获取ip
发送路由请求
请求到目标节点

3.6 请求细节

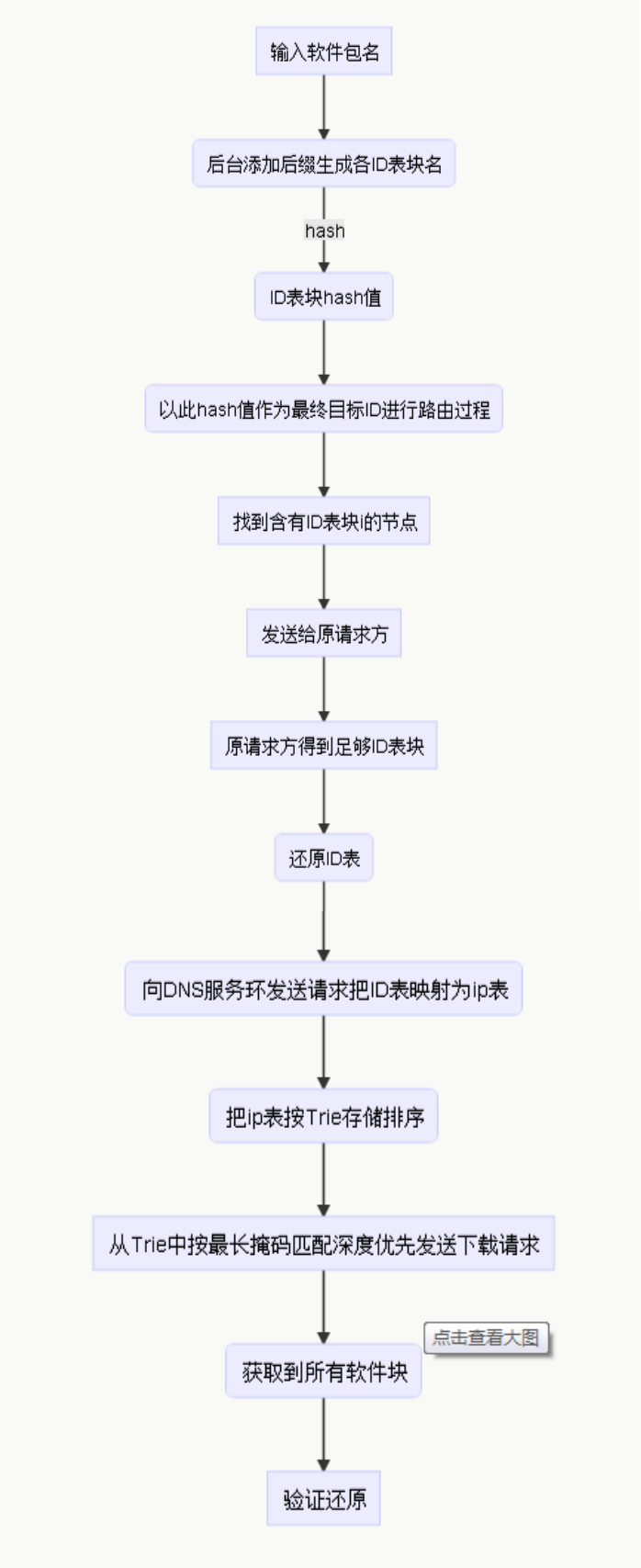
- 节点之间的通信使用 RPC 框架，包括三个远程调用函数： (1)PING 测试是否节点存在 (2)STORE 通知存储的资料 (3)FIND 通知其他节点帮助寻找
- 向 DNS 服务器请求 IP 地址使用基本的 DNS 协议



DNS协议报文格式

- 新节点加入时向 DNS 服务器请求网络中所有 IP 地址，使用远程调用函数 QUERY_ALL，服务器响应返回一个包括所有 IP 与 ID 对应的表
- 由于 hash 后的节点并不一定真实存在于网络中，协议会选取离 hash 后的 id 最接近的节点存储协议表

3.7 下载过程


输入软件包名
后台添加后缀生成各ID表块名
hash
ID表块hash值
以此hash值作为最终目标ID进行路由过程
找到含有ID表块i的节点
发送给原请求方
原请求方得到足够ID表块
还原ID表
向DNS服务环发送请求把ID表映射为ip表
把ip表按Trie存储排序
从Trie中按最长掩码匹配深度优先发送下载请求
点击查看大图
获取到所有软件块
验证还原

3.8 更新过程

新的软件包版本当作新的软件包对待，默认为查找最新版本。

- 当一个用户提供一个软件包时，这个 Donut 会根据软件内协议，算出软件包 ID 表块应该存储在的理论 ID 节点，根据索引，得到实际 ID 节点。
- 然后向这几个节点发送请求，要求更新 ID 表 **拥有同一软件包的 ID 表块的几个节点（以下节点均表示这些节点）：**
- 分配此 ID 应该存在哪一块，需要修改的那一块请求还原回原 ID 表，然后进行修改，编码，由它重新分发。
- 若分发的节点在线，则分发成功，但分发成功的节点依然保留旧版本 ID 表块。若不在线，则分发失败，设定时间间隔，一段时间后重新进行请求分发。
- 等到全部分发成功，此节点设定标志位（表示可以删除旧版本 ID 表），并发送信号给另外几个在线节点删除旧版本 ID 表，并也都设定标志位。对于那些未在线的节点，等到它们上线时，由于存在新旧两个版本 ID 表，它们会向其它在线节点查询标志位，若标志位已设上，则删除旧版本 ID 表。