

对等系统的数据定位

唐志福, 邹恒明

(上海交通大学计算机系, 上海 200240)

摘 要: 数据定位是对等系统的核心问题, 在诸多对等系统数据定位方法中, 应用分布式哈希表接口进行数据定位的方法优点突出。论文讨论了对等数据定位的常用方法以及用分布式哈希表接口进行对等数据定位的优点, 对 4 种实现了分布式哈希表的接口算法, 即 CAN、Chord、Pastry、Tapestry 算法作了分析, 并对这 4 种算法的性能进行了比较。

关键词: 对等; 分布式哈希表; 内容可定位网络; Chord; Pastry; Tapestry

Location of Data in Peer-to-Peer System

TANG Zhifu, ZOU Hengming

(Dept. of Computer, Shanghai Jiaotong University, Shanghai 200240)

【Abstract】 Location of data is a heart problem in P2P field. Among many techniques of data location, distributed hash table (DHT) is most advantageous. The paper discusses most of popular techniques of data location and the benefits of DHT. It analyses the four algorithms of implementing DHT, and compares their preferences.

【Key words】 Peer-to-peer; Distributed hash table (DHT); Content-addressable network (CAN); Chord; Pastry; Tapestry

对等 (Peer-to-Peer) 文件共享应用的成功激起了对等领域的其他应用。对等系统的优点如没有集中的控制或者阶层式组织和每个结点身份同等, 使大家对对等技术产生了浓厚的兴趣。在未来完全分布式的系统完全有可能成为大规模系统组建方式。

数据定位是对等系统的核心问题。它研究如何在一个对等系统以可扩展性方式定位数据, 并且没有任何中心服务器或层次组织作为辅助。它是任何对等系统的核心问题。目前这个问题没有得到很好的解决。这个问题的解决思路是解决其他对等系统难题的模式。

1 数据定位

某一个数据提供者提供了一个数据 X 到这个系统中, 譬如 X 是个文件。在这之后某个消费者想获得 X, 一般是提供者可能已经没有连接系统网络了。消费者如何找到 X 的副本的所在的主机位置呢?

第 1 种技术是应用中心数据库建立文件名和存储该文件的主机之间映射关系 (Mapping)。Napster^[1]是采用了该技术, 低可靠性 (Reliability) 和低扩展性 (Scalability) 是该技术的缺点。这些缺点可能导致数据容易遭到攻击, 系统脆弱。

第 2 种技术是为消费者广播文件 X 请求消息给所有邻居结点。当结点收到这则请求消息, 首先检查一下本地文件系统。如果这个结点有 X 文件, 它就来响应这个消息。否则它会继续广播这则消息给它的邻居。同样它的邻居也完成这样协议。Gnutella^[1,2]是采用该种技术。而该技术不具有强扩展性。因为系统带宽会被广播消息占据, 而且计算资源也会因处理请求消息而耗尽。

第 3 种技术是第 2 种技术的改进。为了减少广播请求消息的消耗, 研究者们把这些网络结点组织成一个层次状结构, 就像互连网的域名服务器 (DNS) 一样。查找从最顶层开始, 沿从一个结点到下一个结点的前进指引, 遍历一条向下通向

存储目标数据结点的路径。引导式的一条路径遍历的资源消耗比广播式的资源消耗少得多。Kazaa、Grokster 和 MusicCity Morpheus 采用该技术。层次状结构方法的缺点是位于层状顶层结点的负荷大, 因此顶层结点使系统变得脆弱。

第 4 种技术为对称分布式数据定位。对称分布式数据定位技术避免了以上技术的缺点。查询沿从一个结点到下一个结点的指引直到找到存储该数据结点而展开。查询从任何一个结点开始, 每个结点只涉及系统中一小部分的查找路径。因此在查询中没有一个结点会消耗过多的资源。该技术具有强扩展性、结点维护信息少, 并且容易高效率覆盖网络。

2 分布式哈希表

哈希表 (DHT) 接口是分布式数据定位算法的重要基础, 一个分布式哈希表只要实现一个操作: lookup(key), 该操作返回一个结点的标识 (如 IP 地址), 返回的标识结点存储键值对应的数据。下面是一个应用 DHT 接口的简单例子: 提供者结点需要把文件的全局唯一名字通过哈希函数如 SHA-1 转换成一个数字键值, 然后调用函数 lookup(key), 提供者结点发送这个文件, 将文件存储到返回结点中。读该文件的结点需要获得文件名, 转换成它的键值, 调用 lookup(key), 然后向返回结点请求该文件副本。

为了实现分布式哈希表, 数据定位算法必须解决如下问题: 建立负载均衡的键值和主机的映射关系, 推进键值查询到适当主机, 建立一个路由表。

3 分布式哈希表算法

3.1 CAN

CAN^[2,3]使用了 d 维笛卡儿积坐标空间来实现 DHT 接口。

作者简介: 唐志福 (1976 -), 男, 硕士生, 主研方向: 高可靠软件研究, 分布式系统研究; 邹恒明, 教授

收稿日期: 2005-11-03 **E-mail:** zftang@sjtu.edu.cn

这个坐标空间被划分为数个超长方形，这些超长方形称为区域。每个结点都对应一个区域而且用它所在区域的边缘作为ID。一个键值映射到一个坐标空间点，它存储在该点所在区域的主机上。图 1(a)表示了一个 2 维 $[0,1] \times [0,1]$ CAN 有 6 个结点。每个结点负责维护存储有邻居结点信息的路由表。邻居结点具有d-1 个坐标值。

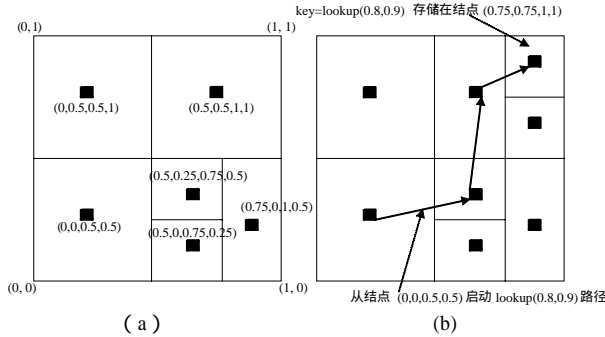


图 1 DHT 接口

查询操作沿几乎是直线的路径从消息查询发起结点到键值存储结点来推进一个数据定位查询消息。接收到数据定位消息，结点把消息推进给它的最近邻居结点，直至找到存储该键值的结点。图 1(b)显示了查询键值(0.8,0.9)的路径。每个结点维护 $O(d)$ 个状态，并且这个查询成本为 $O(d \cdot N^{1/d})$ 。如果 $d = O(\log N)$ ，CAN 的查询次数及存储空间需要是最优的。

当新结点加入到系统时，新的结点首先要在这个坐标空间里选择一个随机点 P，然后要求一个已经在网络中的结点找到区域包含点 P 的结点 n，结点 n 把这个区域分开两半，把一半分配给新结点。新结点很容易初始化它的路由表，因为除结点 n 本身外，结点 n 的邻居也就是新结点的邻居。一旦加入这个网络，新结点应该向它的邻居结点报告它的存在，以便邻居结点更新它们的路由表。

当一个结点要离开系统时，这个结点把它的区域给其中一个邻居。如果两个区域合并成了有效区域，那么这两个区域就变成一个更大区域。如果不能，邻居结点就会临时处理这两个区域，如果该邻居结点失败，CAN 实现一个协议，这个协议让最小区域的邻居来处理这个事情。一个潜在的问题是，在结点处理大量区域合并操作过程中，如果多次失败可能导致这个坐标空间有很多碎片。为了处理这个问题，CAN 潜在地运行一个特别算法实现结点重排。这个算法尽量把能够合并成一个有效区域的区域分配给同一个结点，然后把这些区域合并。

CAN 采用了两种技术来减少数据查询定位的延迟：(1) 结点接收到一个查询，它会以前进幅度与网络延迟时间比值最大的方式来推进这个消息给邻居结点。(2) 多实现。使用多坐标空间比较数据查询延迟。在每个坐标空间里每个结点分配给不同的区域。为了推进一个查询消息，选择最小数据定位延迟空间实现来推进消息。

为了达到负载的平衡，CAN 采用统一分配空间。在结点加入系统操作时，结点在分裂它的区域前要检查一下它的邻居结点。如果邻居结点有更大区域，则由这个更大区域的邻居结点来分裂它的区域。

3.2 Chord

Chord^[3,4]在一维空间分配ID给键值和结点。存储键值K的结点称为K的后续结点，后续结点的ID是最靠近K的。

Chord 查找时间复杂度为 $O(\log N)$ ，N 为结点数。Chord

中每个结点拥有保存 $\log N$ 条记录的指针表。结点的指针表包含从本身开始第半圆数个结点 IP，第 1/4 圆数个结点 IP 和第 2 的 n 次方分之一圆数个结点 IP。如图 2 所示，一个 Chord 查找键值 54 的路径，从结点 8 开始。每个箭头代表查询的推进，通过结点指针表项，结点 8 推进查询到结点 42；因为结点 42 是结点 8 的第半圆数个结点，下一个是第 1/4 圆数个结点 51。这个例子中结点 8 的指针表储存着结点 42、51、56 信息。

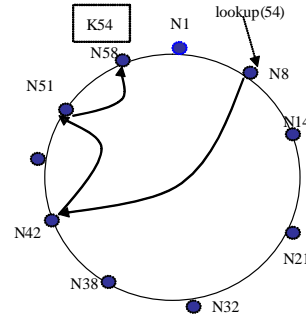


图 2 Chord 的查找路径

一个结点把键值 K 查找消息推进到它的指针表中 ID 最大且比键值 K 小结点。指针表的 2 的次方结构确保了结点总是能把查询消息最少推进剩余 ID 空间一半的距离。因此 Chord 查询使用了 $O(\log N)$ 条消息。

新结点 n 如何确定自己在 Chord 系统中位置，Chord 通过如下方法实现：为了这个新结点能正确地加入这个系统，系统需要做的是更新它的和它前键结点的后续列表。新结点通过请求其他已存在的结点来查找将分配结点 n 的 ID。即使在有相似 ID 的结点同时加入系统中，Chord 也能确保其正确性。更新的结点和已经存在的结点指针表只要蕴式地进行，因为它只是在性能上要求的，不是正确性上要求的。新的结点同时必须获得它存储键值的所有相关的数据。后继关系确保所有键值都可以从新结点的后续结点获得。

随机性保证了 Chord 键值和结点 ID 初步统一地分布在 ID 空间里，确保结点间负载均衡。Chord 对存储在结点的键值空间片通过虚拟结点来控制。在 Chord 系统中的每个物理结点有多份虚拟结点 ID，增加虚拟 ID 会增加结点存储键值的数量。Chord 能够路由一个消息可以在低查询延迟时间的路径上进行；它是通过充分利用存在许多指针表记录更容易把查询推进到目的结点。每个这样的记录都有显著的代价（查询延迟时间）和益处（在 ID 空间的进度）。

3.3 Pastry

Pastry^[2-4]每个结点都有一个随机选定的ID，概念上它表明了结点在标识圈内的位置。Pastry路由键值消息到一个活跃结点，这个结点的ID数字和这个键值很相似。ID空间的位数是 2^b 的倍数。其中b是一个算法的参数，一般b为 4，ID空间的位数就 128。

Pastry 使用一个基于前缀推进机制。每个结点 n 维护一个叶子集合 L，该集合中有 $|L|/2$ 个最靠近结点 n 的且比 n 小的结点，有 $|L|/2$ 个最靠近结点 n 的且比 n 大的结点。叶子集合的正确性是消息正确推进的唯一要求，除非 $|L|/2$ 个 ID 邻近的结点同时失败，否则数据定位消息就能正确推进。叶子结合在概念上和 Chord 的后继列表相似。

为了优化消息的推进性能，Pastry 维护一个指向分布在 ID 空间其他结点的路由表。这个路由表是一个 $\lceil \log_b N \rceil$ 行，每

行有 2^{b-1} 个项。结点 n 路由表的第 i 行的每个项指向 ID 的前 i 位和结点 n 相同的结点, 并且它的第 $i+1$ 位是不同的(最多有 2^{b-1} 种可能)。如图 3 所示, 假设 $b=2$, 结点 ID 空间 8 位, 结点 10233102 的叶子集合; 路由表和邻居结点的结构图。所有数据都是 4 的倍数, 路由表顶行是第 0 行, 每行的阴影单元格表示当前结点 ID 对应的位, 在路由表中每一项结点 ID 都是用特定形式 (10233102 共享前缀 - 共享前缀的下一位 - 剩下位) 来描述的。

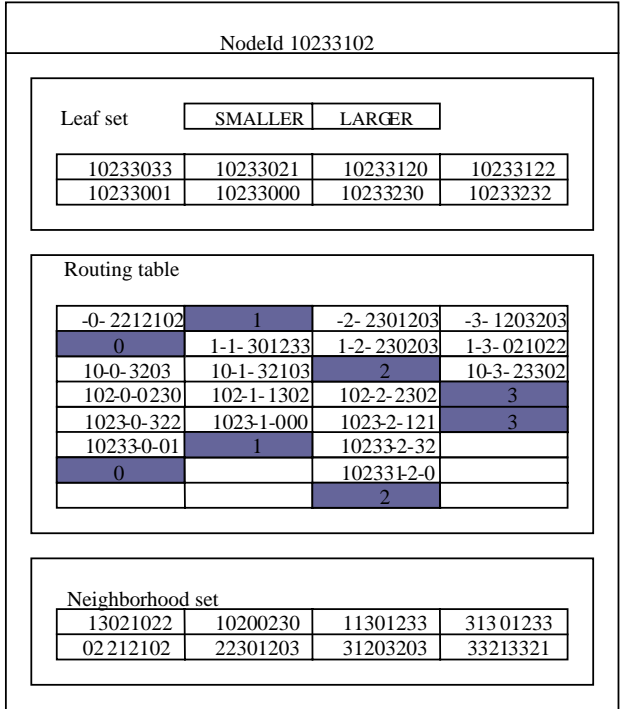


图 3 结点 10233102 的叶子集合, 路由表和邻居结合结构

给定叶子集合和路由表, 结点 n 推进查询消息: 如果要找的键值存储在叶子集合中的结点, 查询消息就推进到给这个结点。一般来说, 直到查询消息推进到 ID 较靠近键值的结点时才会发生这样情况。在这种情况下, 查询消息会被推进到路由表中 ID 和结点 n 具有较长共同前缀的结点。

结点 n 无法到达的结点项可能会从路由表中丢失, 在该种情况下结点 n 推进一个查询消息给 ID 的共同前缀和结点 n 一样长的结点, 并且这个结点的 ID 数字上更靠近键值。这样的结点一定在 n 的叶子集合中, 除非这个查询已经到达数字上和这个键值更相近 ID 的结点或者到达了它的直接邻居。

如果路由表和叶子集合都正确, 预期的 Pastry 路由一个键值到正确的结点的路由数至多为 $\lceil \log_b N \rceil$ 。因为每步路由结点和键值间的共同位数就增加一位。最后查询消息接触到叶子集合只需要一步。

Pastry 有个加入操作协议, 它沿着出发结点或者 ID 空间的最靠近结点到新结点路径获得信息来建立路由表和叶子集合。这个方法简化了新结点的叶子集合正确性的维护, 然后蕴式地建立路由表, 就像在 Chord 一样。结点离开时也使用这个方法。当一个结点企图到达一个不存在的结点和无法到达的结点, 那么就要马上更新叶子集合结点, 和更新路由表信息。

3.4 Tapestry

Tapestry^[5]是Plaxtonet等人开发的基于数据定位机制^[4]。像所有其他的数据定位机制一样, Tapestry映射结点与键值标识成数字串, 并且通过一次增加一位共同位来推进数据定位到正确的结点。详见文献[5]。

4 性能比较

表 1 是 4 种算法的性能对照表。Node state 行标识每个结点被多少个其他结点所知。Lookup 和 Join 行表明为每个操作需要多少条消息。 N 是系统的总结点数, d 是 CAN 的维数。

表 1 各个算法代价比较

	CAN	Chord	Pastry	Tapestry
Node state	d	$\log N$	$\log N$	$\log N$
Lookup	$d \cdot N^{1/d}$	$\log N$	$\log N$	$\log N$
Join	$d \cdot N^{1/d} + d \cdot \log(N)$	$\log^2 N$	$\log^2 N$	$\log^2 N$

操作代价: 表 1 总结了主要操作的代价。Chord, Pastry 和 Tapestry 几乎一致; 不管系统大小 CAN 的路由表大小为常数, 但是查询代价 CAN 比其他系统增加得更快。

查询延迟时间: Pastry, CAN, 和 Tapestry 都有一个启发式选择路由表中的指向系统中附近结点路径, 减少了查询的延迟时间。

恶意结点: Pastry 使用安全证书来证明结点的身份, 这将抵制恶意结点。但代价就是信任一个认证机构。本文描述的算法都有潜能执行交叉检测来监测因为恶意或错误导致的不正确路由, 因此在推进路由消息时, 核实 ID 空间是否有进度是可以做到的。数据的权限是要通过加密保证的。

总之, 4 种对等数据定位算法优点和缺点反映了设计者对不同问题考虑的优先性。这些算法扩展性强, 查找延迟时间短, 扩展地处理结点的接入和分离, 结点路由表维护信息少以及负载平衡处理得好。在适当的场合采用适合的算法, DHT 将会在国际互联网应用上很有价值。

参考文献

- 1 Balazinskam. A Scalable Peer-to-Peer Architecture for Intentional Resource Discovery[C]. Proc. of Int. Conf. on Pervasive Computing, Zurich, Switzerland, 2002-08.
- 2 Dabek F. Wide-area Cooperative Storage with CFS[C]. Proc. of the 18th ACM Symposium on Operating Systems Principles, 2001-10.
- 3 Hildrum. Distributed Object Location in a Dynamic Network[C]. Proc. of the 14th ACM Symp. on Parallel Algorithms and Architectures, 2002-08.
- 4 Malkhi D. A Scalable and Dynamic Emulation of the Butterfly[C]. Proceedings of the 21th Annual ACM Symposium on Principles of Distributed Computing, 2002-07.
- 5 Maymounkov. A Peer-to-Peer Information System Based on the XOR Metric[C]. Proc. of the 1st International Workshop on Peer-to-Peer Systems, 2002-05.