基于 G-Chord的 Cache共享模型

宋志刚

(福州大学 网络与信息中心, 福建 福州 350002)

[摘要] 提出了一种基于 G-Chord算法的节点 Cache共享模型, 实现对象的搜索、存储和分发, 有效地利用了客户节点的缓存内容, 提高了客户间的合作, 减小了客户的等待时间, 降低了服务器的压力. 仿真实验证明, 采用 G-Chord算法处理节点的路由表长度有了显著的缩减, 能够保持较好的平均路径长度. 此外, 对分组数量的不同取值、节点负载的研究也为 G-Chord的分组方案提供了一定的参考依据.

[关键词] G-Chord, Cache共享, 性能
[中图分类号] TP 393 [文献标识码] A [文章编号] 1672-1292(2009)04-0077-05

A Cache-Shared Model Based on G-Chord

Song Zhigang

(Network and Information Center, Fuzhou University, Fuzhou 350002, China)

Abstract A node cache sharing model based on the G-Chord algorithm is presented, which implements the search, store and distributiton of the objects by using the content of cache of the client nodes, while raising the cooperation among the clients, decreasing the wait time of clients, and reducing the pressure on the servers. It was proved in the emulational experiment that the length of the route-table was evidently curtailed, while remaining good length of the average path. The research on the difference among the amount of groups and the value of them and the load of the nodes also provided some reference for the G-Chord grouping scheme.

Key words G-Chord, cache-shared, performance

采用 P2P技术的网络应用和服务已经成为 Internet的重要组成部分, 如分布式计算、即时通信、文件交换、协同设计等. 在这些应用中, Cache是一种常见的有效减少带宽开销、提高响应时间的技术. 目前 P2P应用中节点 Cache之间的内容共享仍需加强.

根据是否采用结构化的查询, P2P的应用及研究分为两类. 第一类采用集中式服务器或者节点完全分散的模型, 其典型实例包括 Napster(http://www.napster.com)、Gnutella(http://gnutella.wego.com)和 FreeNet[2]. 第二类则基本利用分布式哈希表(distributed hash-table, DHT)实现路由管理, 例如 Chord[3]、CAN[4]、Pastry[5]、Tapestry[6]和 Viceroy[7]等.

本文提出了一种基于 G-Chord协议的节点 Cache共享模型, 实现对象的搜索、存储和分发. 仿真实验表明, 在平均路径方面与 Chord接近情况下, 大部分节点的路由表长度得到了显著的降低. 此外, 对分组数量的不同取值、节点负载的研究也为网络的分组方案提供了参考依据.

1 G-Chord路由算法

G-Chord模型建立在 Chord基础上, 将分组的思想引入 Chord环内, 实现各分组的自治, 其逻辑拓扑结构如图 1所示. 其中, 外层的实线环表示标准 Chord网络(用 1级环表示), 假设标识符的位数为 8, 即 $N=64$. 内层的虚线环表示对 1级环进行 $n=8$等分后, 各分组代表节点形成的逻辑环(即 0级环, 位于 0级环上的节点表示各分组的代表节点). 0级环指向 1级环的虚线箭头表明该代表节点来自 1级环的某个节

收稿日期: 2008-12-18
通讯联系人: 宋志刚, 工程师, 研究方向: 网络协议与云计算. E-mail: zgsong@fzu.edu.cn

点. 图中虚线方框表示第 1组节点形成的逻辑 Chord环. 该拓扑表明组内请求和查询可以在组内完成, 组间的请求则通过各组的代表节点进行转发. 为了支持路由的实现, 非代表节点要存储同组非代表节点、本组代表节点和最邻近组代表节点的信息, 而代表节点不仅需要维护这些表项, 还需要存储其他代表节点信息.

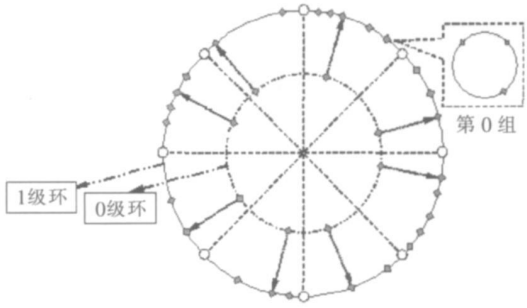


图 1　G-Chord 的逻辑拓扑

Fig.1　The logic topology of G-Chord

在 G-Chord中, 节点标识符被分为两个部分, 分别表示节点所属的组号码(GroupCode)及其在组内的 Local ID (Local ID$\in\left[0, \frac{N}{n}-1\right]$). 两部分的结合构成了标准 Chord 中的 Global ID(Global ID$\in[0, N-1]$). Gbbal ID 使用 m 位表示(即 $N=2^m$), 而 Local ID 则用 $\log_2\left(\frac{N}{n}\right)$ 位表示(n 为分组数, 且 $n=2^i$, $i\in[0, m]$).

2　节点 Cache共享模型

该模型分为两层, 底层解决数据定位, 上层则负责响应客户请求、对象的存储和分发以及 Cache对象信息的维护.

2.1　数据定位

对于组内请求, 目标与请求节点在同一组, 只需组内节点的协助转发即可. 而组间请求, 则必须代表节点、同组或其他组节点共同参与路由过程: 当节点提出组间请求后, 首先确定 key是否能够由最邻近前导或后继组处理, 如果成立, 则直接将请求发送给前导或后继组的代表节点(非代表节点从本组代表节点获得), 否则发送给本组的代表节点; 然后由代表节点根据自己维护的组间路由表信息确定下一跳的代表节点; 如果下一跳的代表节点与目标节点在同一组, 那么该代表节点将利用组内路由表继续转发, 否则将请求发送给下一个适合的代表节点.

2.2　对象的存储和分发

为了避免资源所在节点在 Flash Crowds[8]时成为失效节点, 本文提出了一种对象存储和分发方案, 如图 2所示, 其中(X. Y)表示情形 X 的第 Y步操作, 相似的操作则被省略. (1) 如果 C 的浏览器 Cache中没有对象副本, 就向 R 提交 cGET 或 GET 请求; (2) 如果 R 和 D 上存在副本, 但是 R 可以传输, 那么将返回 C 请求的对象或消息, 并将 C 同对象描述符一起添加到节点列表中去; (3) 如果存在其他 D 且 R 不打算传输副本, 那么 R 将从节点列表随机选择一个或多个 D 并返回给 C, 由 C 从 D 处得到副本, 然后 C 被添加到 R 的节点列表中去; (4) 如果 R 发现自己的对象副本过期或者出现 cache-miss, 并且没有对应的 D, 那么 R 将通知 C 从 S 直接获取对象, 并将 C 加入节点列表, 但此时设置该纪录的状态位为"不可用", 直到下载结束 C 通告 R 将状态位改为"可用", 然后由 R 根据该对象的尺寸和复制阈值(见本文 3.3节)间的比值决定是否存储该副本; (5) 如果 R 存储的副本过期并且还存在其他的 D, 那么 R 将把列表中该副本对应的所有 D 删除, 接下来的操作与(4)相类似; (6) 如果 R 上出现 cache-miss且存在 D, 那么处理过程如情形(3); (7) 如果 C 发现 D 上的副本过期, 那么将采用(4)中的操作来保证对象的更新和一致性.

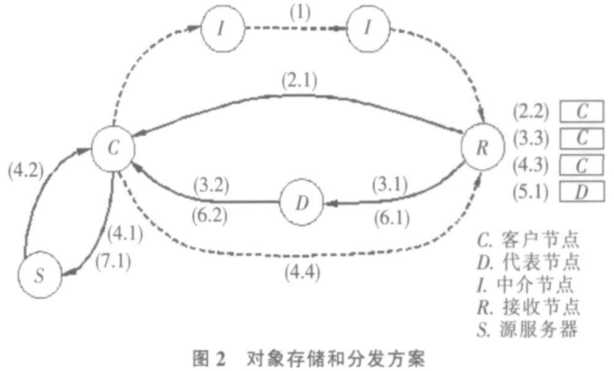


图 2　对象存储和分发方案

Fig.2　The scheme of the storing and distributing of the objects

3　实验与分析

在 Ubuntu操作系统下, 使用 C/C++语言和 Glib库开发程序. 实验采用日志驱动, 日志文件采用某大

学实验室 2005年一天的 Web浏览记录, 共计 100 801次 HTTP请求, 请求对象的尺寸总量达到 4 GB. 仿真实验分析了模型在不同节点和分组数量配置的路由性能、节点负载.

3.1 G-Chord与 Chord的路由比较

节点被分为 3组, 每组存在一个代表节点, 每个节点 Cache尺寸上限为 200 MB. 分组后非代表节点的组内路由表长度降幅约为 28%. 图 3描述了分别采用 Chord和 G-Chord算法处理请求所需非零跳数百分比的差别.

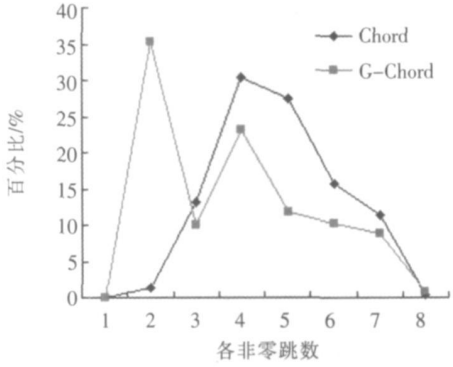


图 3 G-Chord 与 Chord 在非零跳对应请求数量方面的比较

Fig.3 The comparison of the request amount corresponding to the non-zero jump between G-Chord and Chord

由图 3可以看出, 不论是 Chord还是 G-Chord算法, 不存在仅需 1跳就能够处理的请求, 也就是说在该仿真条件下所有对象的索引与存储节点没有出现重叠, 而且二者的网络直径都是 8跳. 采用 G-Chord后, 原本占据了跳数总量大部分比例的 4、5跳请求现在只要 2跳就可以得到处理, 超过了非零跳对应请求总数的 30%. 观察节点的分组情况后, 发现造成这种结果的原因是第三组只存在代表节点自身, 到达该节点的请求一旦无法得到满足, 将立刻被转向外部服务器, 无须在该组路由请求, 因此减少了请求的转发次数. 除此之外, 4、5跳请求的百分比之和也超过了 30%, 从 3跳开始的分布曲线在外形上与 Chord类似, 说明分为 3组条件下, G-Chord并没有导致网络直径的增大, 相反许多请求的路由过程得到了改善.

3.2 不同分组取值的路由比较

除了 Cache的尺寸, 还研究了同等尺寸下分组数量对路由的影响. 在实验中, 将节点分别分为 3、5、9、16、28、36和 44组, 节点的 Cache尺寸设置为 200 MB, 仿真后的结果如图 4所示.

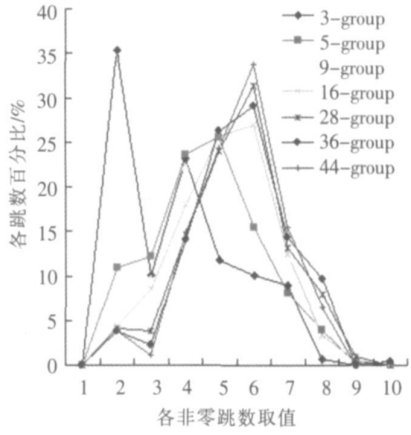


图 4 7种分组方式下非零跳对应请求的百分比分布

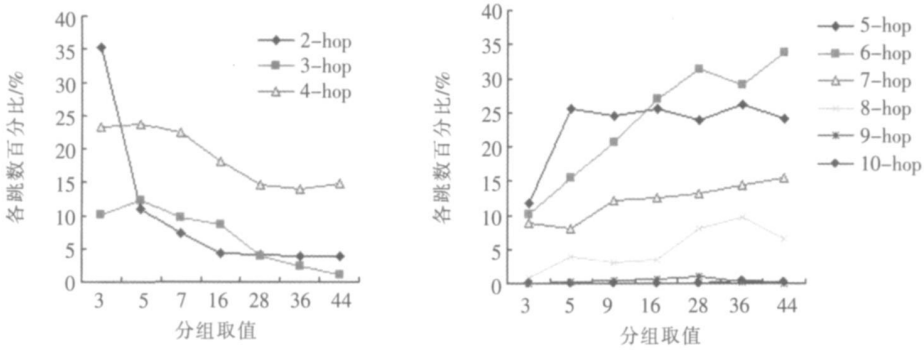
Fig.4 The distribution of the request amount corresponding to the non-zero jump under 7 kinds of grouping

与图 3相比, 图 4中 1跳对应的请求数量依然为 0, 但是出现了 9和 10跳. 网络直径增大的原因如前文所述, 分组增多后, 许多原本属于组内的请求成为组间请求, 首先需要 1跳转发到组代表节点, 然后通过组间和组内路由过程到达索引节点, 最后到达存储节点或者外部服务器, 因此最多需要 $(2+\log_2 N)$跳. 图 4中, 2、3和 4跳所占的比例随着分组数量的增大降低明显, 而 5、6、7和 8跳对应的请求数量增加较多, 相应的变化分别如图 5的(a)、(b)所示. 由于 1跳对应的请求数始终为 0, 因此不在图中绘出.

由图 4可以看出, 随着分组数量的增多, 网络直径和网络的平均路径长度都会相应增大. 分组为 3时跳数为 2的请求数量约为 35%, 但分组增多后, 5、6跳请求的数量增加, 网络直径也增加到 10跳. 这是因为, 当组数较少时, 每组的节点数量较多, 很多请求在组内就可以得到处理, 而组数增多时, 组内的节点减少, 请求可能需要经过多个代表节点才能转发到目标节点, 因而网络直径增加. 这同时也说明适当的分组是必要的, 但是一旦分组过多, 虽然非代表节点需要维护的路由表尺寸有所降低, 但是却带来更多数据定位和路由方面的开销, 此外, 由于需要更多的节点参与, 势必对节点的资源提出更多要求, 所以是不可取的.

图 5描述了分组变化对非零跳(除跳数为 1)请求数量的影响. 其中, 图 5(a)表示请求数量减少的百分比曲线, 可以看出组数从 3增加到 5时, 2跳对应请求数量的减少最为明显, 约为 25%, 原因如上所述; 而 3、4跳请求虽然在节点分为 3组、5组和 9组时基本保持不变, 但随着组数增多依然降低. 图 5(b)中, 增幅最大的跳数为 6, 约为 25%, 而 5跳对应请求数量在组数从 3到 5之后差异不大, 7跳的增加较为平稳, 而 8跳则在 16组之后明显增多. 由此, 对于实验中的节点来说, 分为 9组之内, 可以实现减少路由表长度

同时不会明显增大网络平均路径的目标.



(a) 不同分组中 2,3 和 4 跳的分布 (b) 不同分组中 5~10 跳的分布

图 5 不同分组条件下,2~10 跳对应请求的百分比分布曲线

Fig.5 The distribution curve of the request amount corresponding to 2~10 jumps under different grouping

3.3 总跳数和平均路径

在 Web Cache为 200MB,组数为 3情况下, G-Chord的非零跳数的请求总量为 44 972,虽然多于 Chord的 44 757,但是在总跳数 (即 $\sum_{i=1}^{8} \mathrm{hop}_i \times \mathrm{count}_i$) 方面, G-Chord为 171 383,而 Chord则为 214 468,减少量为 43 085,降幅约为 20%. 说明在相同条件下, G-Chord减少了平均路径长度. 同样,设置 Cache尺寸为200MB,对不同分组的非零跳数对应请求总量、总跳数和平均路径长度 3个参数同采用 Chord算法产生的结果进行比较,如图 6所示.

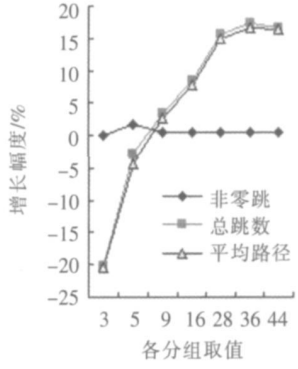


图 6 G-Chord 的 7 种不同分组方式与 Chord 算法在总跳数等参数的比较

Fig.6 The comparison of the total amount of jumps between G-Chord and Chord under 7 kinds of grouping

由图 6可以看出,非零跳对应请求的增加相对较小,最大值在分为 5组时也不超过 3%,因此平均路径的长度就由总跳数决定. 如上文所述, 在分为 3组时, 总跳数降低约 20%,随着分组的增多,总跳数也明显增大, 在 28组之后增长趋缓. 值得注意的是,平均路径在 9组时增幅出现了由负到正的改变,并且小于 5%,这也从另外一个方面为分组数量的选择提供了参考.

3.4 节点负载

引入分组后,代表节点的负载同样是本文关心的性能指标. 在实验中,记录了所有代表节点处理请求的频率及对应的数量,统计结果分为以下 3组: 每秒处理的请求数量小于 1、数量在 1到 10之间以及大于 10小于 51. 对于第一组频率对应的请求数量来说, 是请求总量的主要组成, 但是对于节点的负载影响极小,因此在此不再讨论. 第二组频率是除了第一组频率之外的主体, 其均值曲线分布如图 7所示. 其中, Chord算法的平均负载最低. 当节点分为 3组时, 代表节点负责的 ID范围较大,因而请求处理的频率较大,相应的负载也高. 随着分组的增多, 节点所代表的 ID范围较小, 负载逐渐得到均衡,这对于频率为 1、2、3时尤其明显. 值得注意的是,当分组数为 5和 9,除了频率为 1、2之外, 其他频率对应请求数量的均值与 16、28、36和 44组几乎相同.

第三组统计结果中请求数量的均值最小, 但是对节点的性能要求较高, 其分布曲线如图 9所示. 其中, Chord算法在这组统计中所有频率均值为 0. 可以看出,在这个部分的统计中, 分组为 3时的请求数量分布曲线与其他分组差别非常明显, 而且不存在频率高于 33次 /s的请求. 其他分组中虽然出现了较高的频率,但是均值始终小于 1,说明在这些分组的情形下某些节点会出现瞬时高负载,但出现的机率非常小.

结合图 7和 8,可以看出适当的分组 (如 5和 9组)既可以减少路由表长度、加快请求的转发过程,同时也不会对节点带来更多的负载.

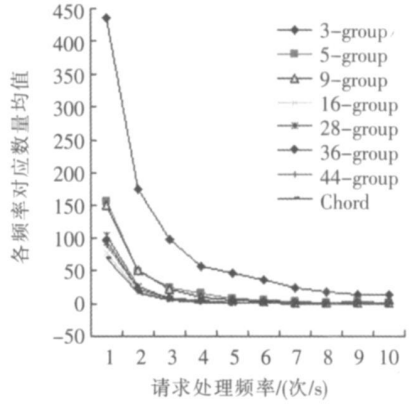


图 7 请求频率处于 1~10 的请求数量均值分布

Fig.7 The distribution of the average amount of the request corresponding to requesting frequency from 1 to 10

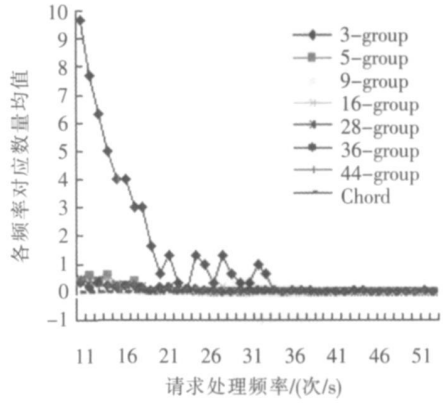


图 8 请求频率处于 11~51 的请求数量均值分布

Fig.8 The distribution of the average amount of the request corresponding to requesting frequency from 11 to 51

4 结论

G-Chord是一种基于分组的 DHT路由和数据定位算法. 本文在 G-Chord基础上提出了节点 Cache的共享模型, 有效地利用了客户节点的缓存内容, 以此提高客户间的合作, 减小客户的等待时间, 降低服务器的压力. 仿真实验证明, 该模型中大部分节点的路由表长度有了显著的缩减, 却能够保持较好的平均路径长度, 而对节点负载的研究也说明为 G-Chord的分组提供了参考依据.

[参考文献] (References)

[1] 陈刚, 吴国新, 杨望. G-Chord: 一种基于 Chord的路由改进算法[J]. 东南大学学报: 自然科学版, 2007, 37(1): 9-12
Chen Gang, Wu Guoxin, Yang Wang. G-Chord: an improved routing algrithm for Chord[J]. Journal of Southeast University: Natural Science Edition, 2007, 37(1): 9-12 (in Chinese)

[2] Clarke I, Sandberg O, Wiley B, et al. Freenet: A distributed anonymous information storage and retrieval system[C]//Proceedings of the Workshop on Design Issues in Anonymity and Unobservability. Berkeley, California, 2000: 46.

[3] Stoica I, Morris R, Karger D, et al. Chord: a scalable peer-to-peer lookup service for Internet applications[C]//Proc of ACM SIGCOMM 2001. New York, USA: ACM Press, 2001: 149-160.

[4] Ratnasamy S, Shenker S, Stoica I. Routing algorithms for DHTs: some open questions[C]//Proc of 1st International Workshop on Peer-to-Peer Systems, 2002. Berlin: Springer, 2002: 174-175.

[5] Druschel P, Rowstron A. Pastry: scalable, distributed object location and routing for large-scale peer-to-peer system[C]//Proc of the Middleware 2001. Heidelberg: Springer-Verlag, 2001: 329-350.

[6] Zhao B, Kubiatowicz J, Joseph A. Tapestry: an infrastructure for fault-tolerant wide-area location and routing[R]. Computer Science Division, University of California at Berkeley, Tech Rep: UCB/CSD-01-1141, 2001.

[7] Malkhi D, Naor M, Ratajczak D. Viceroy: a scalable and dynamic emulation of the butterfly[C]//Proc of the 21st annual ACM Symposium on Principles of Distributed Computing, 2002. New York, USA: ACM Press, 2002: 183-192.

[8] Stading T, Maniatis P, Baker M. Peer-to-peer caching schemes to address flash crowds[C]//Proc of 1st International Peer-to-Peer Systems Workshop (IPTPS 2002). Cambridge, MA, 2002: 1-21.

[责任编辑: 严海琳]