第 36 卷 第 8 期 计 算 机 工 程 2010 年 4 月
Vol.36 No.8 Computer Engineering April 2010
·网络与通信· 文章编号：1000—3428(2010)08—0120—03 文献标识码：A 中图分类号：TP393

一种懒惰的 Chord 指向表更新算法

汪 昱，陈荣华，叶德建
(复旦大学软件学院，上海 201203)

摘 要：在节点加入退出或者失效情况下，结构化 P2P 算法 Chord 搜索指向表(FT)出现大量指向错误，降低搜索效率。针对该问题，讨论和比较几种搜索 FT 的更新策略，分析各算法维护指向正确的开销，提出一种懒惰算法解决搜索 FT 更新效率低下的问题。该算法最小化搜索 FT 更新的消耗，可作为一种有效的错误恢复机制。通过实验对比证明了该算法的有效性。

关键词：对等网络；分布式散列表；指向表

Lazy Chord Finger Table Update Algorithm

WANG Yu, CHEN Rong-hua, YE De-jian
(Software School, Fudan University, Shanghai 201203)

【Abstract】 As lots of errors in Chond search Finger Table(FT) appears, it depresses the searching efficiency when nodes go into or drop out. By discussing and comparing several algorithms of search FT updating, this paper proposes a lazy one to refresh the search FT recursively. The algorithm minimizes the consumption of FT refvesh, it can be an effective fault tolerance mechanism as well. Experimental result shows the correctness and effectiveness of the algorithm.

【Key words】 Peer-to-Peer(P2P) network; Distributed Hash Table(DHT); Finger Table(FT)

1 概述

结构化 P2P 克服了非结构化 P2P 在寻找资源上的不确定性，被广泛地应用在分布式文件共享(BT)和分布式多媒体(skype)等领域。Chord[1]作为一种结构化 P2P 环境的分布式资源发现服务，以其结构简单、可塑性强的特点一直都是这方面研究的热点。本文主要研究如何维护和更新 Chord 用于加速搜索的搜索指向表(Finger Table, FT)。

Chord 依赖 FT 加速搜索过程，相应地，如果指向表项发生错误，则会大大降低搜索的效率。为了解决以前算法 FT 正确性维护开销过大的问题，本文提出一种把指向表的修正延迟到搜索时触发的懒惰更新算法，最小化了 FT 更新的消耗，虽然一定程度上增加单次搜索的跳数，但是在整体效率上有很大的提高。本文还对该算法通过实验模拟进行验证和横向比较，获得了搜索平均花费的跳数等性能指标。

2 Chord 搜索指向表更新

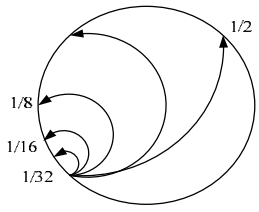
2.1 Chord 和分布式散列表

Chord[1]由美国麻省理工学院提出，是一种比较常见的环状结构化 P2P 拓扑，用分布式散列表[2](Distributed Hash Table, DHT)的方式，由广域范围大量节点共同维护的巨大 Hash 表。对 Chord 算法的改进已经有很多研究，改进其覆盖网或者物理拓扑[3]是比较常见的方式，本文主要从 Chord 搜索的核心 FT 进行研究。

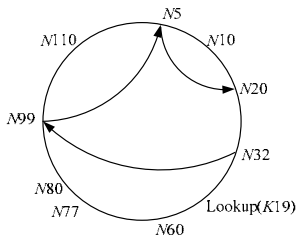
2.2 搜索指向表及其更新问题

在没有引入 FT 之前，节点只维护后继节点信息，为了定位到资源索引，Chord 上的搜索必须从发起搜索的位置向后面节点逐一进行，搜索的复杂度为 $O(N)$。

在引入 FT 后，节点维护了一个到后继指数级递增位置的表，分别存放了与当前节点距离为 $1,2,4,8,\cdots,2^n$ 的位置的信息。这样，当资源索引存储的位置大于表中某一项时，可以直接跳到对应节点，搜索效率提高到 $O(\log N)$。FT 加速下的搜索过程如图 1 所示。



(a)指向表



(b)搜索过程

图 1 FT 加速下的搜索过程

构建正确的 FT 后，搜索伪码过程如下：

```
Lookup(my-id, key-id)
  Look in local finger table for
      Highest node n my-id<n<key-id
```

作者简介：汪 昱(1984－)，男，硕士研究生，主研方向：宽带网络，互动多媒体；陈荣华，讲师；叶德建，副教授、博士

收稿日期：2009-12-20 **E-mail：**ywang.fdu@gmail.com

```
If n exists
    Call Lookup(key-id) on node n
Else
    return my successor
```

但 FT 同样出现了新的问题：在 FT 指向错误时，搜索便不能正确完成。很多问题可以导致 FT 的不正确指向，最常见的是节点的加入和退出，一次变动可能影响上万个节点上的 FT。另外，新节点也需要足够的信息来初始化 FT。在分布式系统中，确定哪些节点上的 FT 需要改变以及通知其如何改变等的代价都是相当昂贵的，在搜索/更新比比较低的系统中更是如此。

3 FT 更新算法

FT 更新评价的关键在于，当一个或者多个节点发生变化时，以尽量小的代价来确定具体哪几个节点需要被更新以及如何更新。要随时保证所有节点上的每一个 FT 项是正确的，这显然是不现实的，因为其代价昂贵，且很难在并发时保证一致。在改进 Chord 路由和搜索方式[4]，也就是更好地利用 FT 方面，已经有不少研究，但是对 FT 本身维护的探讨还比较少。

3.1 已有的周期更新算法分析

Chord 最初的更新算法主要是基于后台周期运行的 stabilization 和 fix_fingers 这 2 个协议，前者保证节点加入退出时前后继节点链的正确，后者通过周期搜索 FT 中的各项的正确位置进行更新，新节点的初始化同样通过 fix_fingers 搜索相应的 key 来进行。在一次搜索中，如果遇到 FT 未及时更新导致失败，则退化(如果 Finger[*n*]失败，使用 Finger[*n*-1])到 FT 中的上一项，最差的情况退化到 Finger[0]，即把搜索请求传递给直接后继节点处理。

该算法的问题是 fix_fingers 的运行周期不易确定，如太短，则频繁搜索修正开销过大，如太长，则 FT 中错误项过多，退化影响整个搜索算法效率，而且搜索退化的方式过于消极，在错误项累计到一定程度的时候，搜索效率可以退化到 $O(N)$。

3.2 懒惰更新

针对上述周期更新周期难以确定以及消极退化规避错误引起搜索效率下降的问题，本文设计了懒惰算法。其主要理念是除非必要，决不提前做任何 FT 的修正行为，把某些 FT 路由表项的修正推迟到访问时，从而有效减少总体更新开销。

考虑到对从未使用的 FT 项的更新是一种浪费(FT 被更新后从未被访问，或者节点加入后迅速离开)，甚至在节点加入时对 32 项 FT 逐一初始化都是多余的，在懒惰算法里，只有在搜索过程中发现 FT 错误时，才主动要求并强制更正 FT 后再重新开始搜索过程。

具体做法是在搜索过程中包含上一跳节点的 FT 指向信息，比如还是以在图 1 中插入 *N*50 并且从 *N*32 发起搜索 *K*49 为例。由于 FT 错误，搜索下一跳被传递给 *N*60，只要把搜索请求传递给 *N*60 的同时告知 *N*60，*N*32 是以 32+16=48 为搜索依据，*N*60 根据自己的维护区间[51, 61]可以判断出 *N*32 的 FT 有误，从而驳回搜索请求，通知 *N*32 更新第 5 项 FT 后，再重新进行搜索过程。

*N*32 通过搜索的形式，利用出错 FT 表项的上一项，以搜索资源同样的形式，搜索 *K*(32+16)的位置，从而更新对应的表项。如果在这个过程中同样出现 FT 错误，则递归进行 FT 的更新请求过程。

更新递归过程一定会结束，因为每个节点维护直接前继和后继节点的信息，FT 表中的第 1 项，即距离为 1 的后继节点位置总是保持正确的。递归过程如果到达这一项时必定会结束。

下面用一个实际的过程来解释懒惰算法的工作。假设在图 2 中，*N*5 是新加入的节点，在加入时不做任何更新 FT 的操作，那么此时很多节点上的 FT 指向错误。在图 2 中，L 指搜索；E 表示 FT 错误；F 表示找到资源；FL 表示 FT 的修正搜索。

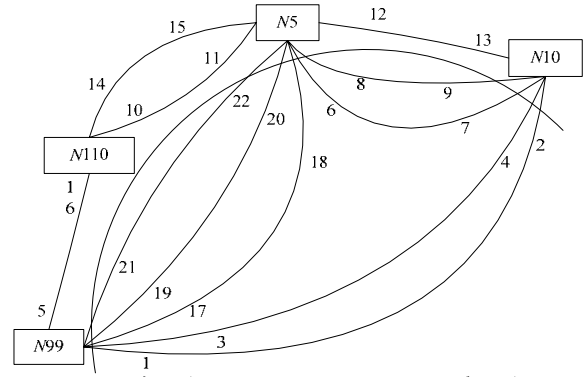


图 2　搜索及更新过程

如果此时节点 *N*99 执行如下操作：lookup(*K*4)。*N*99 不知道 *N*5 的存在，它会根据 FT *K*(99+32) *N*10 (错误！)直接指向 *N*10。节点 *N*10 发现自己的责任区间[6, 10]，并不持有 *K*4 的资源索引，通知 *N*99 其路由项可能有误。*N*99 试图更正其路由表，做 lookup(99+32)，通过 FT 表项上一项跳转 *K*(99+16)=*K*(115)　*N*10(仍然错误)；同样，*N*10 再次返回路由错误信息。*N*99 试图更正路由表，做 lookup(99+16)…直到在修正 lookup(99+16)时，*K*(99+8), *N*110。*N*110 做 lookup(107)，根据 *N*110 的路由表 *K*(110+4), *N*10(还是错误)，根据 *N*10 返回的路由错误信息，*N*110 试图修正 *K*(110+4)，做 lookup(114)根据路由表 *K*(110+1)来跳转，由于 *K*(110+1)总是保持为 *N*110 的后继节点，为 *K*(110+1)= *N*5(正确)，因此 *K*(110+4)成功得到修正，依次返回修正结果并且重新发起搜索，最终 *N*99 找到 *K*4。

此次查询过程中一共修正错误路由 4 项，找到资源 1 个，耗费通信过程 22 个。

节点退出或者失败的情况下，情况类似，只是不需要对方反馈，本节点在发现 FT 指向的位置已经失败后，就可以主动发起更新 FT 的请求。

该算法不但避免了无意义的操作浪费，而且因为容许几乎任何程度的错误(但是 FT 错误率高时平均搜索的消耗会变大)，本身可以作为一种非常有效的错误修正机制。

4 实验及其效果评估

4.1 实验设置

为了论证懒惰算法的实际效果，用程序模拟的方式对以上的更新策略做了实验。

以循环链表模拟 chord 的环状拓扑，随机生成 32 位无符号整数作为加入节点的 key。搜索时随机挑选环上任意一个

节点，随机生成 32 位搜索 key 交给 chord 进行搜索，记录所用的跳数。

节点变动和搜索行为随机发生，发生的比率遵照实验预先设置的搜索/变动比触发。节点加入的几率固定，离开(或失败)的几率与环上的节点数目呈正比，在 chord 规模到达一定程度时，节点数目趋于稳定。为了对比的一致性，在不同的搜索/变动比设置下，节点稳定的目标规模都定在 10 000。

作为对照，在每个节点上部署 2 套不同的 FT，其中，第 1 套采用了时刻保证 FT 全部正确的更新方式，任何节点的加入离开失败都重新更新所有相关节点的 FT 表；第 2 套运行懒惰算法。由此，可以得到在严格相同的操作顺序下，不同策略的数据表现，其中，对照组的更新采用变动时计算可能涉及节点所处的区域范围，优化消除重叠区域的冗余后，搜索区域起始位置按后继节点链依次修正。

4.2 实验结果

下面是在不同的搜索更新比之下，不同策略平均每个操作耗费的跳数与平均每次搜索耗费的对比，见表 1、表 2 和图 3、图 4。

表 1 搜索/变动比与操作数比重

编号	搜索/变动	节点数	加入节点数	离开节点数	搜索节点数
1	10:1	8 706	14 112	5 587	200 301
2	20:1	8 577	14 248	5 671	400 081
3	30:1	8 704	14 351	5 647	600 002
4	40:1	8 635	14 218	5 583	800 199
5	50:1	8 634	14 283	5 649	1 000 068
6	60:1	8 686	14 270	5 584	1 200 146
7	70:1	8 706	14 363	5 657	1 399 980
8	80:1	8 641	14 375	5 734	1 599 891
9	90:1	8 727	14 498	5 771	1 799 731
10	100:1	8 631	14 433	5 802	1 999 765

表 2 极值对比和最终 FT 正确率(即时/懒惰)

编号	最大搜索跳数	最大变动跳数	FT 正确率/(%)
1	312/14	0/716	100/92.40
2	243/14	0/704	100/93.99
3	218/14	0/724	100/94.86
4	252/14	0/713	100/95.34
5	199/14	0/714	100/96.03
6	186/14	0/708	100/96.27
7	195/14	0/720	100/96.36
8	259/15	0/726	100/96.78
9	178/14	0/716	100/97.02
10	165/15	0/714	100/97.03

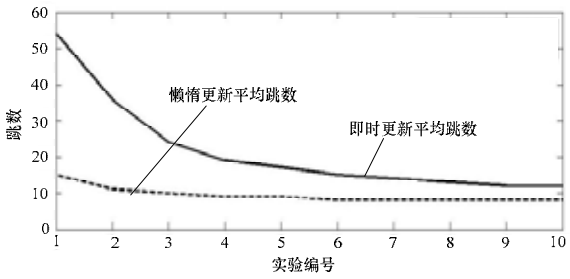


图 3 平均跳数对比

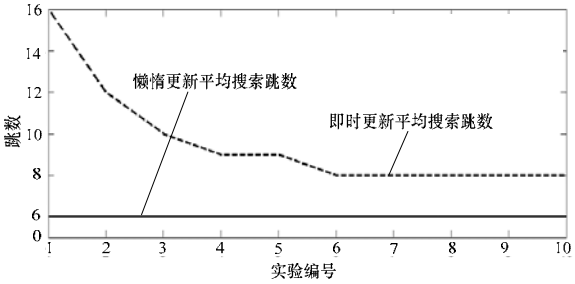


图 4 平均搜索跳数对比

以上的实验共进行 50 次，平均跳数和搜索平均跳数基本保持稳定。

从实验结果可以看出，懒惰算法的平均跳数明显少于即时更新的算法，特别是在搜索/更新比比较小的时候。由于需要额外的修正，因此在搜索的平均消耗上懒惰算法略逊于即时更新。懒惰算法在最大搜索跳数方面更加依赖于 FT 的正确率，在搜索路径错误比较多的情况下，最差情况的消耗比较大，但是相对于节点变动时可能最差 700 多跳的消耗还是有了很大的提高，并且这一点会随着搜索/变动比的增大而有所改观。即使是在搜索/变动比比较小的环境中，92%左右的 FT 正确率也是可以接受的。懒惰算法牺牲一部分搜索的性能大大缩减了节点变动的开销，适合用在对节点加入退出速度要求比较高、搜索/变动比相对比较大的应用环境中。

5 结束语

本文提出一种懒惰算法解决搜索指向表更新效率低下的问题。实验结果证明，虽然在平均搜索跳数上做了一定牺牲，但是在平均的操作跳数上却有很大优势。修正搜索递归包含触发修正时完整的搜索信息，无须在单个节点上保存未完成搜索的状态，工程角度也易于实现，在笔者参加的一个 P2P 语音项目中，使用该算法达到了较好的效果。

分布式资源搜索服务系统具有广阔的应用前景，但是由于结构化的限制和利用 Hash 搜索的特性，在模糊查询和错误恢复方面有不可避免的缺陷。下一步将在应用模糊查询、提高系统错误抵抗能力以及改进拓扑和搜索方式等方面进行深入的研究。

参考文献

[1] Stoica I, Morris R, Karger D. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications[C]//Proceedings of ACM SIGCOMM'01. New York, USA: ACM Press, 2001.

[2] Zhu Yingwu, Hu Yiming. Efficient, Roximity-aware Load Balancing for DHT-based P2P Systems[J]. IEEE Transactions on Parallel and Distributed Systems, 2005, 16(4): 349-361.

[3] 邹东尧, 宋美娜, 宋俊德. 一种基于物理网络拓扑的高效 Chord 模型[J]. 计算机工程, 2008, 34(6): 127-129.

[4] 孙道平, 王于同. Chord 路由算法的改进[J]. 机电工程, 2007, 24(12): 92-94.

编辑 索书志