基于DHT的分布式索引技术研究与实现

吴 炜 苏永红 李瑞轩 卢正鼎
(华中科技大学计算机科学与技术学院 武汉 430074)

摘 要 针对索引创建和维护效率不高的问题,设计了一种基于DHT(Distributed Hash Table)的分布式倒排索引构建算法。该算法利用基于改进的Chord网络的分布式哈希表技术,将分词后的结果分散到多个索引服务器上并行构建索引,同时采用前驱列表定位和减少服务器定位延迟的技术,大大缩短了索引构建时间。通过采用统一调度的基于分块的增量式倒排索引更新策略,索引更新时不再需要移动已有的索引文件,提高了索引更新效率。利用周期性稳定算法和前驱列表定位提高了系统的稳定性、容错性和索引的一致性。

关键词 分布式索引,分布式哈希表,Chord网络

Research and Implementation of Distributed Index Based on DHT

WU Wei SU Yong-hong LI Rui-xuan LU Zheng-ding
(School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)

Abstract A distributed inverted index's building method based on DHT (Distributed Hash Table) was adopted to improve the index's creating and updating efficiency. The arithmetic, using the DHT technology based on improved Chord network, hashes the terms and their relational information to the distributed index servers and builds the index parallely. This method reduces the index's building time through distributing a task to many nodes. The strategies of scheduling the index building task through chained index management servers and the incremental distributed inverted index updating method were used, which could assure index's consistency and updating efficiency.

Keywords Distributed index, Distributed hash table, Chord network

随着计算机和网络技术的不断发展,计算机的应用领域不断扩大,数据的规模越来越大,查询也越来越复杂。对于具有海量特征的数据,集中式处理技术具有检索速度慢、可扩展性不强、容易出现访问瓶颈等缺点,分布式技术以其高性能的特点而逐渐成为解决这类复杂问题的有效手段,关于这方面的研究也成为了研究热点。WonGoo[1]是一种在CAN(Content Addressable Network)内容可寻址网络环境下,基于DHT[2](Distributed Hash Table)的质心法的文本内容索引方法,其索引能够较好地体现文本内容语义,同时具有短小、存储消耗小、均衡性好的特点。但其DHT算法随着结点不断加入,会将CAN网络区域划分得支离破碎,而且由一个结点负责多个结点的情况将越来越多,直到负载超过结点能力上限,其可扩展性不太好,需要采用更加丰富的、经验证有效的传统信息索引技术来进一步提高系统负载能力和扩展性。ALVIS[3]是Toan. Luu等提出的利用DHT,HDKS(Highly Discriminative Keys)[4]和关键词语义相关性来建立结构化P2P网络中基于关键字的分布式索引,但其DHT算法中的语义相关机制导致了索引尺寸的加大,需要大量空间来存放索引信息,影响索引的构建速率,并且索引结点加入时会发送大量消息到网络上的其他结点,这会占用大量带宽,带宽的不足和索引结点占用大量空间都会影响索引的查询速率。著名的Web搜索引擎Excite利用网页索引构建一个全文分布式数据库[5],基于动态空间槽的分布式并行空间索引树DPSIR+[6],但也存在索引构建复杂、负载不均衡的缺点。综上所述,目前国内外在基于DHT的分布式索引方面的研究都不太成熟,主要是因为在设计模型时对索引问题不够重视或考虑不全面。因此,需要设计一个检索效率高、负载平衡性好、可扩展性好和适用性广的基于DHT的分布式索引。本文提出的这种新型构建分布式索引技术,可以弥补其它分布式索引技术在这方面的缺点。

1 基于DHT的分布式索引构建

1.1 分布式索引系统模型

本系统是在基于改进的Chord协议的基础上建立包含超结点的两层多环形分布式索引结构。在Chord协议中,所有的结点被组织在一个单环中,而在本系统中有多个环,每个普通环负责单独的分布式索引内容,环中的每个结点有一个与关键词相关的拥有它源的主机IP地址列表和与结点相关的

到稿日期:2009-03-06 返修日期:2009-05-25 本文受国家自然科学基金项目(60873225,60773191,70771043),国家高技术研究发展计划(863计划)项目(2007AA01Z403)资助。

吴 炜 博士生,研究方向为信息检索、分布式计算,E-mail:eggwalk@hotmail.com;苏永红 硕士生,研究方向为对等网络、分布式异构系统的安全;李瑞轩 博士,副教授,研究方向为分布式异构系统、分布式系统安全;卢正鼎 教授,博士生导师,研究方向为分布式系统、智能信息系统、信息安全。

前后关系列表，环中的每个结点不一定拥有它们负责的实际资源，但有与匹配的资源相关的 IP 地址列表。为了使普通环被发现，超环是包含指向其它普通环的指针的超结点组成的环，即索引管理服务器结点组成的环。当某个索引管理服务器离开或失效时，将它所负责的资源传给相邻的超结点。图 1 展示了多环结构的两层索引系统模型，有些处理方式如路由表处理和系统稳定性处理继承于标准的 Chord 协议，每个结点的 Chord 路由表中有指向结点的前驱和后继结点表的指针，当发起一次新的查询时，先返回相同关键词以前的查询结果，如果继续查询，则通过分布式哈希表发布查询并与以前的查询结果一起结合来返回新的查询结果。环中结点的加入采用懒惰环方式处理，即当有结点提出加入申请时并不立即被加入环，而是当申请的结点达到一定数量时再一起加入，并修改相关前驱和后继结点的哈希表和路由表。

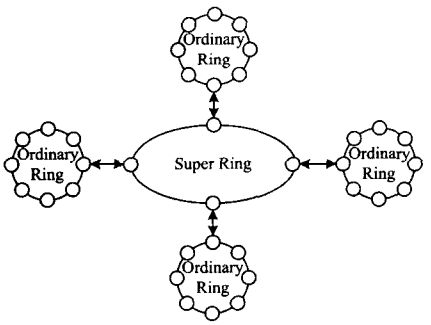

图 1　索引系统模型

1.2　分布式哈希表算法

分布式哈希表算法用于将索引文件分布式存储和管理。分布式哈希表是一个广域范围内大量结点共同维护的巨大散列表，它被分割成不连续的块，每个结点被分配给一个属于自己的散列块，并成为这个散列块的管理者。通过加密散列函数，一个对象的名字或关键词被映射为 160 位的散列值，DHT 结点被映射到一个空间。DHT 算法能够自适应结点的动态加入或退出，结点分配具有均衡性，结点查找具有准确性。

我们采用基于改进的 Chord 网络的分布式哈希表算法，通过 SHA-1(Secure Hash Algorithm)散列算法，将对象映射为 160 位散列值。对 Chord 结构的改进在于采用前驱列表定位来提高定位容错性，通过选择服务器来减少定位延迟，对结点 ID 进行认证以防止 ID 伪造和 ID 虚报。原来的 Chord 定位中，最重要的是通过调用 *find_predecessor*(*id*)函数返回对象 *id* 的直接前驱，这里对此做了改进(如算法 1 所示)，该函数每次通过调用 *preceding_node_list*(*id*)子函数获得一个前驱列表，然后从这些前驱列表中选出最大的前驱 n'，再通过 n'来获得 *id* 前驱列表，这个过程循环执行下去，直到最终找到 *id* 的直接前驱。虽然改进后的方法每次返回的信息比过去多，但定位速度、容错性比以前要好得多。

算法 1　改进后的 Chord 前驱寻找函数、获得前驱列表的子函数

```
//请求结点 n 寻找 id 的前驱
n. find_predecessor(id)
  n' = n;
  while (id ∉ (n', n'. successor])
      l = n'. preceding_node_list(id);
      n' = max n'', n''∈l;
  return n';
//请求结点 n 返回一组结点，这组结点在 n 的路由表中，
//或者是 n 的后继列表中位于 id 之前的结点
n. preceding_node_list(id)
  return {n', n'∈{fingers∪successors}&(n, id]};
```

为了减少定位时延，系统在 Chord 层加入了服务器选择，即定位过程中选择下一跳时，尽量选择那些时延小、在底层物理网中靠近的结点。以上面的前驱寻找函数为例，假设某一步结点 m 返回给结点 n 的 *id* 前驱列表，列表中除了结点位置信息，还有每个结点到 m 的时延(这些时延在 m 初始化、更新自己的路由表时获得)。基于这些时延，n 对列表中的每个结点进行评价，选取它认为最合理的作为下一跳，这就是“服务器选择”。设 $C(n_i)$表示 n 取 n_i 作为下一跳的预计开销，网络中采用如下公式评价 $C(n_i)$，从而选择最小的 $C(n_i)$作为下一跳：

$$C(n_i)=d_i+d'\times H(n_i) \tag{1}$$

$$H(n_i)=ones((n_i-id)\gg(160-\log N)) \tag{2}$$

其中，d_i 表示结点 m 在前驱列表中告诉 n 它到 n_i 的时延，d'表示 n 所取的每跳平均时延(基于 n 的过去的经验值计算)。$H(n_i)$表示 n 取 n_i 作为下一跳后估计剩余的跳数，N 表示 n 对网络总结点数的估计(基于 n 的邻近结点密度)，因此 log(*N*)是当前网络中真正决定跳数的高比特位数，≫为位运算右移符号。结点 ID 使用 160 位散列值，*ones*(*x*)函数计算 x 中有多少位为 1，因此(n_i-id)被右移$(160-\log N)$位后，*ones*()函数实际上计算了 n_i 到 *id* 间近似的跳数 $H(n_i)$。

在计算出 $H(n_i)$后，将 $H(n_i)$乘以每跳平均时延 d'，即得到取 n_i 作为下一跳后的估计剩余时延，加上到 n_i 本身的时延，就是取 n_i 所要的总时延的预计值 $C(n_i)$。这里假设了一个前提：n 到 *id* 的定位时延大致等于 n 到 n_i 的时延加上 n_i 到 *id* 的定位时延。通常情况下，因特网是近似符合这个特点的，因此采用服务器选择的 Chord 定位一般工作得较好，从而能有效地降低时延。

结点 ID 认证的目的是防止恶意结点伪造 ID 或者虚报 IP 地址等信息。当一个新的结点 n 加入网络时，一些现存的结点会考虑是否将它加入到自己的路由表中。每个这样的现存结点首先发送一条消息到 n 所宣称的 IP 地址，并附加一个“现时”(一个很难预测、伪造的随机数，仅用来标识本次事务)。如果该 IP 地址的结点承认它拥有所宣称的 ID，并且将此 ID 收到的“现时”以及其他信息(如虚拟结点号)回发给现存结点，那么现存结点首先对比自己发出的“现时”与收到的“现时”，结果相同证明没有虚报 IP 地址(否则 n 不可能收到该“现时”值)，然后计算 n 的 IP 地址和其他信息的哈希值，如果该哈希值与结点 n 宣称的 ID 相同，证明 n 没有伪造 ID，此时现存结点才将 n 的 ID，IP 地址等信息加入自己的路由表。

在分布式哈希表网络中，网络结点根据 IP 地址和端口号等信息通过散列运算分配一个唯一的结点标识符(Node ID)，资源对象通过散列运算产生一个唯一的资源标识符(Object ID)，每个关键字和结点都分别拥有一个 m 比特的标识符，超结点与普通结点网络分别形成一维的环形空间，结点和关键字的 ID 从$(0\cdots 2m-1)$中选取，是文件和 IP 地址的 Hash 值。关键字通过 SHA-1 散列运算转化为 160 位二进制，然后转化为 40 位十进制，取前中后 3 个十位相加，取模，模空间为结点

空间。关键字标识为 K 的(K,V)对存储在这样的结点上，该结点的结点标识等于 K 或者在环上紧跟在 K 之后，这个结点被称为 K 的后继结点，表示为 $Successor(K)$，$Successor(K)$就是从 K 开始顺时针方向距离 K 最近的结点。

图 2 给出了网络中 DHT 算法的结点资源分配过程。图中给出了一个 $m=3$ 的 Chord 环，环中分配了 4 个结点，每个结点中列出了其后继结点表的示意图。$Successor(id,i)=(id+2^i) \bmod m, 0\leqslant i\leqslant m-1$，表中 id 表示结点编号，i 表示编号为 id 的结点的后继指针列表的指针编号，$Succ$ 表示编号为 id 的结点的指针表的第 i 个指针的后继指针编号。如对于结点 0，它的第一个指针的后继指针编号为 $Succ(0,0)=(0+20) \bmod 8=1$，结点 id 的后继指针表中第 i 项是圆环上标识大于等于$(id+2i) \bmod m$ 的第一个结点。当结点 id 加入网络时，某些原来分配给它的后继结点的关键字将分配给结点 id；当结点 id 退出网络时，所有分配给它的关键字将重新分配给它的后继结点，除此之外网络中不会发生其它变化。

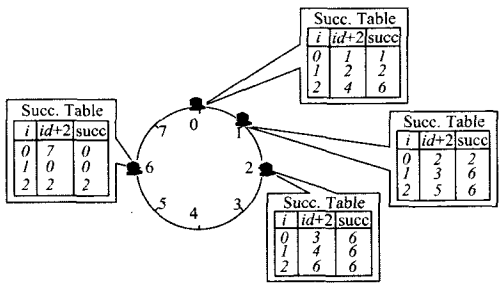

图 2　DHT 算法的结点资源分配图

网络的查询过程是：任何一个结点收到查询关键字 K 的请求时，首先检查 K 是否落在该结点标识和它的后继结点标识之间，如果是的话，这个后继结点就是存储目标(K,V)对的结点。否则，结点将查找它的指针表，找到表中结点标识符最大但不超过 K 的第一个结点，并将这个查询请求转发给该结点。通过重复这个过程，最终可以定位到 K 的后继结点，即存储有目标(K,V)对的结点。通过这种散列算法，新找到的结点对象 id 的距离通常比原来少一半，因此可以保证在 $O(\log 2N)$跳内找到结点(N 是总结点数)。

1.3　索引结点的实现

1.3.1　索引管理服务器结点

索引管理服务器主要负责索引文件的读取、索引的分析、索引的映射和对索引结点的管理。为了对文档数据建立索引，第一步就是要把这些需要建立索引的文档数据转换成可以处理的类型，如对 PDF 文档、Word 文档或者 HTML 文档建立索引，这可以通过文档抽取工具来实现，对于 PDF 格式的文档，可以使用 XPDF 工具来抽取，对于微软公司的 Word 和 Excel 程序，可以使用 Jacob 工具来抽取。

在建索引和检索过程中，分析是很重要的一个环节，本系统使用分析器来对各种各样的输入进行分析，分析器的运行性能和分析能力直接影响到索引的构建和搜索效果。所谓“分析”，用更具体的话说就是“分词”和“过滤”，它位于索引和文本资源之间，所有进入索引库的文本资源都应当经过分析器的分析，以此来控制索引中的内容。分词器就是用于对文本资源进行切分，将文本按规则切分为一个个可以进入索引的最小单位。过滤器的功能是对这种最小单位进行预处理，比如大写转小写、复数转单数、“停止词”分析等。对于英文分词，使用 Lucene 搜索引擎自带的分析器。对于中文分词，由于 Lucene 自带的中文分词器处理效果不是很好，在此使用中国科学院计算技术研究所研制的中文分词工具，该工具基于多层隐马模型的汉语词法分析系统 ICTCLAS(Institute of Computing Technology, Chinese Lexical Analysis System)，具有97.58%的分词正确率(最近的 973 专家组评测结果)。

索引的映射采用远程方法调用(RMI, Remote Method Invocation)和分布式哈希表(DHT)方式实现。在分布式索引管理服务器的实现过程中，首先将上传文件转化为系统可以识别的 Document 文档，并以 XML 文件形式存取，通过引用库文件包调用上传文件，然后通过调用分词、DHT 函数和读取索引服务器台数，将上传文件分词，并将分词后的文件通过分布式哈希表方式映射到不同服务器，其中服务器台数、服务器 IP 号和端口号通过 XML 文件读取，也方便以后修改，再通过 RMI 方式远程调用服务器端桩程序，在服务器端构建索引。图 3 是索引管理服务器上的索引分布过程。

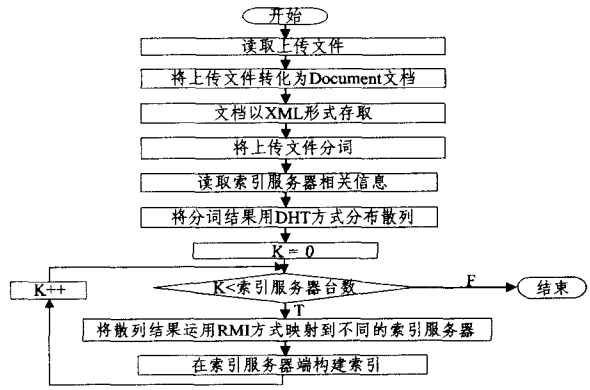

图 3　索引分布流程图

索引管理服务器结点具有较大的容量和较好的处理能力，每个结点除了具有普通结点的功能外，还与相关的普通结点具有路由表和分布式哈希表联系并对普通环中的索引进行管理，包括索引结点的路由管理、索引结点的备份管理、超结点对普通结点检索的支撑管理等。索引结点的路由管理用于管理结点的路由、路由表的更新，通过路由时间的比较确定检索路线。索引结点的备份管理用于对突然断电、服务器出现故障、文件损坏或丢失等特殊情况的管理，可以提高系统的可靠性。超结点对普通结点检索的支撑管理就是把一定的管理信息存储起来，其他结点只有索引等基本的数据，包括索引、路由、版本、数据恢复等管理。这些管理功能通过在管理结点的索引上设置相应的索引项，通过相关项比较、周期性扫描等方式实现。

1.3.2　索引服务器结点

索引服务器结点负责具体的构建索引过程。首先分析索引数据结构，然后分析索引分布式存储的具体过程。下面以某个索引服务器为例加以介绍，假设有下面 2 个文本。

文本 1：华科大硕士研究生热爱祖国。

文本 2：硕士毕业论文体现学生学术水平。

假设索引服务器收到的分布式哈希结果为下面 5 个分词。

[毕业][华科大][论文][硕士][研究生]

表 1 为字符次数表。

表 1　字符次数表

1,1	…	1,1	…	1,1	…	2,2	…	1,1	…
毕业		华科大		论文		硕士		研究生	

相应的字符次数列表用(C_i, W_i)表示，其中 C_i 表示字符在所有文档中出现的次数，W_i 表示包含字符串的文档个数。

其中，每个字符在索引中所需的字节数为：

$$(C_i + W_i \times 2) \times 2 \tag{3}$$

根据式(1)－式(3)可以计算出每个字符在最终的索引中所需的字节数。索引数据结构如下：

$$\{D_i, T_i, [S_{i1}, S_{i2}, \cdots]\} \tag{4}$$

其中，D_i 表示字符串所在文档号，T_i 表示字符串在文档中出现的次数，S_{ij} 表示字符串在文档中的位置。根据$(C_i + W_i \times 2) + 2$，式(3)可以确定索引头和索引体的保留字节数，根据$\{D_i, T_i, [S_{i1}, S_{i2}, \cdots]\}$，式(4)可以计算出每个字符在最终的索引中所需的字节数(为简单起见，设所有数据均占用两个字节)。

在索引服务器端，还需要实现索引分布式存储的具体过程。由于采用 RMI 方式，首先需要声明创建索引的公共接口 CreateIndexable，它继承了 Remote 类，方便索引服务器和索引管理服务器的访问，然后要创建生成索引的桩程序，以此作为本地对象索引管理服务器和运行于索引服务器的远程对象之间的中间人，各个远程对象都以客户端上的桩类表示，最后要创建索引服务器程序，实现索引服务器与注册表的绑定，当索引管理服务器映射部分索引到远程的索引服务器上时，通过访问注册表实现远程引用，完成索引的分布式存储。

在上述程序中，首先获取项目根路径和索引存放路径，分解客户端传入的字符串，然后对每一个字符串，构建一个索引域，并在此基础上生成一个 Document，然后通过将索引写入 Document，完成索引的构建过程。在索引管理服务器对索引服务器创建索引程序的引用中，需要通过索引服务器端的注册绑定程序来调用。

2　分布式索引的更新

2.1　倒排索引更新的几种方法

对倒排索引动态性的研究与其说是算法的研究，不如说是数据结构的研究，经典的动态数据结构大多是基于树实现的[7]，一般是 3 种：平衡二叉树、B 树、键树。以这 3 种结构为基础，可以得到多种变种结构。文献[8]中提出在倒排索引的字典表上建一个 B 树索引，以支持字典表的扩充。B 树的索引项就是单词，一个单词对应的倒排列表放在一个 B 树块中，如果一个块装不下，则在堆文件(heap file)中分配一片连续的空间，再从 B 树叶子结点链一条指针到堆文件中。此后，若倒排表还要膨胀，超出了已有空间，则重新在堆文件中分配一片空间，把原倒排列表迁移到新空间中，再将新增倒排列表合并到后面。然后将旧空间标记为空闲。对这个堆文件的管理可以采用动态存储空间分配的算法。文献[9]提出了一种 Dual-Structure 索引结构，它动态地区分长、短倒排列表，并对长、短列表采用两种不同的存储结构及配套的检索、更新策略。一种是针对倒排列表的。某词通过静态散列函数映射到桶里，一个桶可以装多个倒排表，桶数是固定的，且每个桶只有一个存储块。每个词的倒排列表初始都是短倒排列表，当某个桶溢出时，将桶中最长的倒排列表升为长倒排列表，在字典表中做上标记后，将它迁移到长倒排列表使用存储结构中。这样就动态地实现了长、短倒排列表的分区。另一种是将长倒排列表放在变长的连续块中，且末尾预留一部分空闲空间，这些块不让其他词的倒排列表共享。对于给定的词，可以通过查词表来判定它是否对应一长倒排列表。

2.2　分布式索引的更新

分布式索引的更新包括文档的更新和结点的更新。对于文档的更新，首先对上传文档进行分析，将分词后的结果用分布式哈希表策略计算并分发到不同的索引结点上，如果索引结点上有该索引词，采用追加方式将相关信息链接到倒排链表末尾，否则创建该索引词的头结点，采用倒排索引链表方式链接相关信息。索引结点采用增量式分块的倒排索引更新策略，即把一个庞大的倒排文件分割成字节数相等的若干个小部分。这样在对索引进行更新时不是把后添加的索引内容存入到原索引文件中，而是利用指针链接，把新创建的索引块与原索引文件链接起来。这样，在索引更新时，不需要移动操作，而是直接追加，从而极大地提高了系统索引更新速度，使其具有较高的动态性与灵活性。

对于索引结点 n 的插入，采用自适应路由表策略，告诉其后继将应该由 n 负责的数据对象索引传递给 n。对于索引结点的退出，采用后继列表定位的方式，将要退出的索引结点的信息加入到后继列表中最近的容量和性能综合最好的索引结点中。下面根据上述思想对索引更新中文档的更新和结点的更新分别进行讨论。

2.2.1　文档的更新

根据上一节的索引创建算法，首先在索引管理服务器上对新添加进来的文档进行分析，将分词后的结果用分布式哈希表策略计算并分配到不同的索引结点，然后对各个索引结点计算出本次索引更新中每个字符项所需要的索引空间，并对每个字符项判断上次索引更新所分配的空间中剩余多少空间，是否够本次更新所用，如果小于本次所需的空间，则根据需要进行存储块的分配，并在上次索引块中设一指向新分块的指针；如果是一个新字符串，则在索引头中增加一指针项，分配一个所需要的空间，并且指针指向所分配的空间。

分布式索引更新的过程如下：

(1)将需要更新的文档输入索引管理服务器并进行分析，获得索引词和索引词相关信息，如出现次数 C_i、包含字符串的文档个数 W_i 等；

(2)采用分布式哈希表策略将索引词和索引词信息散列到不同的索引结点；

(3)根据相关规定设置数据块的大小 W，并根据分布式散列过来的分析结果，利用式(3) 计算每一个字符所需要的字节数 m；

(4)如果是第一次索引更新，则直接分配所需要的块数为：

$$\mathrm{Int}(m/W) + 1 \tag{5}$$

并在原索引文件中设置一个指向本块的指针；如果不是首次更新，则设上次所剩空间为 H，则本次所需要的块数为：

$$\mathrm{Int}((m - H)/W) + 1 \tag{6}$$

分配存储块后，在上一个索引块中设置一个指向本块的指针；

(5)在内存中进行索引创建;

(6)当一个索引块写满时,把相应的索引块写出。

当某一文本需要删除时,对索引做标记,使该文本成为禁检文本。若删除的文本较少,采用这一方法应不成问题,关键是若有大量文本已被删除,但索引中还保留有这些文本的索引,这必然造成极大的浪费,也降低了检索效率。对于这种情况,必须从索引文件中删除已删除文本的索引记录。为了支持文本删除,启用一个 Deletable 文件,它记录了一组文本的 docID,这些文本在原文件中已经删除,但还没有从倒排索引中清除,只有当删除文本数积累到一定数量时,再集中整理索引。根据分块索引的策略,采用移位式删除的方法,即对每个字符的索引块从头往后依次检查,若已有文本被删除,则把该文本对应的这个字符的索引记录全部删除,后续的索引向前移即可,如果最后存储的索引块的索引内容全部移到前面存储块中,则把该空存储块添加到未使用的存储块链表中。

2.2.2 结点的更新

在动态网络环境中,结点可以在任何时刻加入或者离开,对于分布式网络来说要求在动态环境下仍然能够正确地定位每个数据对象。网络采用"稳定化"(stabilization)的自适应算法来保证结点后继的正确性,避免了并发操作或不正常操作如不做通知的结点退出导致的错误。在后继正确的前提下,进一步使用后继信息来验证和更新路由表项,这样定位可以恢复到和正确时一样。算法 2 是网络自适应算法的代码,它包括 3 个部分:①并发的结点加入函数 $join(n')$;②稳定函数 $stabilize()$;③路由表更新函数 $fix\_fingers()$。

算法 2 网络自适应算法

```
n. join(n')
    predecessor = nil ;
    successor = n'. finde_successor(n) ;
//周期性地修正 n 的后继,并通知其后继修正前驱
n. stabilize()
  x = successor. predeccessor;
  if (x∈(n,successor))
      successor = x ;
  successor. notify(n);
//n' 认为 n 可能是自己的前驱
n. notify(n')
  if (predecessor is nil or n'∈(predecessor,n))
      predecessor = n' ;
//周期性地修正路由表项
  n. fix_finger()
    i= random index > 1 into finger[] ;
    finger[i]. node = find_successor(finger[i]. start) ;
```

这样通过自适应函数,网络中每个结点周期性地调用稳定函数 $stabilize()$ 和路由表更新函数 $fix\_finger()$,前者首先修正自己的后继,然后通过调用子函数 $notify(n')$ 来通知其后继修正前驱;后者在前者的基础上随机修正自己的路由表项,这对提高定位效率很有意义,路由表的正确度、定位效率能提高多少,取决于它被执行的周期的长短。

网络中没有明显的结点退出操作,而是采用周期性探测的方法。对任何结点 n,在间隔时间内,只要索引管理服务器结点在规定间隔时间内没有收到索引结点的新的保持请求,就可以认为该结点已退出,并根据副本策略和后继列表修改后继关系。环上每个结点维护一个后继列表,其中保存了该结点在环上的 r 个后继,除非 r 个后继都失效,否则结点的后继关系就可以修复。本分布式哈希表算法将每个数据块复制 k 份放到 k 个索引服务器中以提高数据可用性,并在索引服务器不断加入、退出的情况下自动维护 k 个副本。将数据块的 k 个副本放在环上数据后继结点的 k 个后继上,这样就可以很容易地找到它们,因为每个索引结点都维护一个后继列表,并且后继列表项 r 大于 k。这样通过后继列表和复制策略,对于结点的退出,可以快速地修正结点的后继关系,然后根据后继关系修正路由表,这种方法具有较好的容错性。

3 性能分析

3.1 索引构建性能

下面对采用改进的 Chord 结构的 DHT 方法分布式构建索引的效率与集中式构建索引的效率和前面提到的其它基于 DHT 的分布式检索系统 WonGoo 和 ALVIS 的分布式索引构建效率进行比较,来测试与分析分布式索引的性能。用于测试的计算机配置为 WindowsXP 操作系统,Pentium 4 3.0GHz CPU、120G 硬盘、1GB 内存。网络环境为 100Mbps 的局域网,分布式系统在 5 台机器上进行测试,运行环境为 Tomcat6.0 + MySQL 5.1。分 10 次在文档数量递增的条件下分别在集中式构建索引和分布式构建索引两个系统中测试,用于测试的单个文档大小在 20kB 左右,文档数量从 1000 篇增加到 27000 篇。

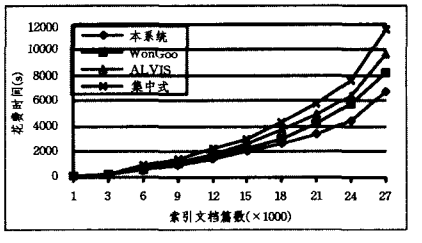

图 4 索引构建时间比较图

从图 4 可以看出,当索引文档数量少于 6000 篇时,分布式索引的构建效率不如集中式构建索引的效率,随着文件数量和文档总大小的增加,分布式构建索引的优势逐渐体现出来,在基于 DHT 的分布式索引构建方法中,本系统的构建方法与 WonGoo 和 ALVIS 比较,具有较高的性能,因为 ALVIS 的语义索引信息构建占用较多空间,影响索引构建效率,所以 WonGoo 比 ALVIS 的索引构建具有较高的效率,又由于 WonGoo 是基于 CAN 网络的,其网络区域的划分方式不如改进的 Chord 网络,因为本系统是基于改进的 Chord 网络的,所以本系统比 WonGoo 具有较好的性能,通过本系统与其他基于 DHT 的分布式索引构建方法比较,可以发现基于改进的 Chord 网络的 DHT 算法的索引构建方法比其他分布式索引构建方法有较高的效率,因此证明了本策略的有效性。

3.2 索引更新性能

为了降低更新的时间开销,采用批量更新的方法,每次文本文件发生更新时并不立即更新索引,而将索引的更新积压起来,当达到一定数量时再写出到外存,这样能有效地节省索引更新总的 I/O 次数。

相比之下,基于 B 树的倒排索引采用伙伴系统管理倒排列表,可能导致频繁地申请和释放空间以及频繁地迁移索引

数据，而且对于较长的倒排列表，迁移可能涉及多个块。因此，更新的代价很高。

Dual-Structure 索引结构为每个倒排列表都在末尾预留了一块空间，虽然能缓解倒排列表迁移的次数，但是一旦发生迁移，就需要为整个倒排列表重新分配空间，所以更新开销依然较高[10]。

下面对设计的索引更新时间与基于 B 树的倒排索引和 Dual-Structure 索引更新时间进行测试比较。测试过程为：文本文件开始大小为 40MB，不断向其中添加 25MB 大小的纯文本文件，分别记录每次索引更新所需要的平均时间。机器性能与第 1 节实验所用机器配置相同，实验数据是纯文本文件，如图 5 所示。

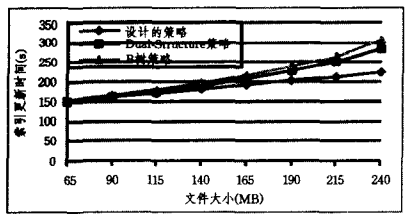

图 5　更新时间分析

由图 5 可以看出，设计的索引更新时间随着原索引文件的增加变化并不是很明显，而相对来说 Dual-Structure 策略和 B 树策略的索引更新时间随着索引文件的增加变化较大。当索引更新时，只对原索引文件进行追加，然后从索引头中找到该词项的最后一个存储块，进行指针的链接，原来的索引文件并没有移动，这样索引更新的时间其实与索引创建时间基本相同，并不会随着索引文件的增加而使索引时间有较明显的增加。

3.3　稳定性分析

系统不稳定的主要原因是结点的加入和退出，由于解决系统不稳定的主要方法是“周期性”稳定协议。改进的 Chord 网络对于结点的大量加入和退出是敏感的，而通过调整稳定性协议在固定时间内的执行次数可以保持系统稳定性。在这个实验中，在一台机器上模拟 500 个索引结点，分 5 次在 2 小时内执行稳定算法，执行次数逐次加 1，每次有 1～10 个结点加入和退出，取每次结点加入和退出失败率的平均值，通过周期性稳定算法执行次数与结点加入和退出的关系来测试系统的稳定性，如图 6 所示。周期性稳定算法通过调整结点前驱和后继列表指针列表修改路由表来调整结点加入或退出导致的索引信息的变化，结点加入时通过调整前驱列表指针来反映结点的加入，结点退出时通过调整后驱列表指针来反映结点的退出。从图中可以看出，在固定时间内，稳定算法执行次数越多，结点加入和退出的失败率越低。并且对于系统来看，这个稳定算法可以使系统达到较好的稳定性。

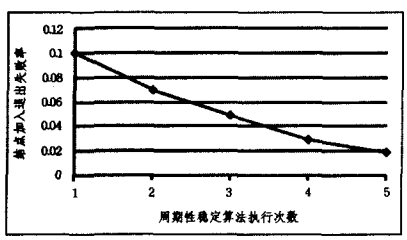

图 6　结点加入、退出失败率

结束语　本文对基于 DHT 的分布式索引创建和分布式索引更新进行了深入的研究，主要完成了以下两个方面的工作。

(1)分析了现有分布式索引构建算法的不足，设计了一种基于 DHT 的分布式索引构建方法。利用改进的对等网络 Chord 中的分布式哈希表算法，将分词后的索引词散列到不同的索引结点上，然后分布式构建索引。本算法将集中式的索引构建分布到不同索引结点上并行构建，提高了索引构建效率，本系统的分布式哈希表策略可以确保系统数据分配的均衡性、可扩展性和检索的准确性，相对于其他分布式策略具有较好的性能。

(2)采用了一种新的基于分块的分布式索引更新策略和自适应的索引结点更新策略。索引的更新通过索引管理服务器统一管理调度，得到更新任务的索引结点在不需要更新原有索引的基础上采用指针链接追加到索引块末尾的方式，减少了索引的移动，增加了索引的更新效率。对于索引结点的加入退出，采用路由监测自适应和前驱后继列表的方式。索引结点定期地向周围结点和索引管理服务器发送存在信息，若规定的一段时间一直没收到某索引结点的信息，则认为此结点已退出网络，并根据结点前驱和后继结点信息修正网络，将退出结点的信息存储到其后继列表的结点中。

参 考 文 献

[1] 程学旗，吕建明，周昭涛. 基于对等网络的全文信息检索[J]. 计算机研究与发展，2004，41(12)：2148-2156

[2] Ratnasamy S, Shenker S. Routing algorithms for DHT : Some open questions[C]// Proceedings of IPTPS. Boston, MA, 2002: 278-293

[3] Luu T, Kelemm F. ALVIS Peers: A Scalable Full-text Peer-to-Peer retrieval Engine[M]. Arlington, Virginia, USA, 2006: 383-395

[4] Podnar I, Rajman M, Luu T, et al. Scalable peer-to-peer web retrieval with highly discriminative keys[R]. September 2006

[5] Zakis J D, Pudlowski Z J. The World Wide Web as Universal Medium for Scholarly Publication, Information Retrieval and Interchange[J]. Global Journal of Engineering Education, 1997, 1(3)

[6] 左朝树，刘心松，陈小辉，等. DPSIR+：一种基于动态空间槽的分布式并行空间索引树[J]. 计算机科学，2006，33(2)：121-126

[7] 刘兴宇. 基于倒排索引的全文检索技术研究[D]. 华中科技大学，2004，05：24-34

[8] Cutting D, Pedersen J. Optimization for Dynamic Inverted Index Maintenance[C]// Proceedings of the 13th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. New York: ACM Press, 1990: 405-411

[9] Tomasic A, Garcia-Molina H. Incremental Updates of Inverted Lists for Text Document Retrieval[C]// Proceedings of 1994 ACM SIGMOD International Conference on Management of Data. Minneapolis. New York: ACM Press, 1994: 289-300

[10] 吴恒山，刘兴宇，左琼. 一种基于可扩展散列表的倒排索引更新策略[J]. 计算机工程，2005，8(30)：83-84