一种动态更新索引结构的设计与实现

The Design and Implementation of a Updating Index

郭琦娟 （中国石油大学(华东)计算机与通信工程学院 山东东营 257061）
陈通照 （中国石油信息技术服务中心 北京 100724）

摘要:在全文检索中,要使索引具有较好的灵活性,就需有合适的索引结构及更新策略,使得既能有效地实现索引更新,又不影响查询效率。本文设计了一种基于互关联后继树模型的动态更新的索引结构,该索引结构由主索引、附加索引和删除文件列表组成,很好的解决了索引的更新问题。

关键词:全文检索 互关联后继树 实时更新 全面更新

1 引言

全文检索是一种非常有效的信息检索技术,它极大地提高了从海量数据中查找特定信息的效率。当前信息的增长速度和淘汰速度均呈现不断加快的趋势,对全文检索的动态性能要求也越来越高。由于中文信息处理的复杂性和特殊性,实现一个完善的全文检索系统还存在不少困难。因此,对全文检索技术进行更多广泛深入的研究是十分必要的。

本文描述的全文检索系统采用互关联后继树索引数据结构,能够处理中英文文档,尽管在建立时需要付出一定的代价,但是在基本不影响查询效率的前提下,大大减少了索引的更新时间。

2 互关联后继树模型简介

互关联后继树模型[1]是从中文语言特点出发提出的一种新颖的全文检索模型,它将全文看成一个字符流,利用多棵二层树组成的森林表示这个全文字符流。与其它模型相比此模型具有创建速度快,查询速度快,空间效率高,并且可以通过索引生成原文,可以直接查询任意长度的字符串等特点。在互关联后继树模型中,对任意文本 T 中的任意字符串 ab,a 称为 b 的前驱;b 称为 a 的后继。在每个文本 T 的末尾人为添加一个不在该文档中出现的字符,作为该文档的结束符,则文本最后一个字符的后继为文本结束符。

例如文档 babac#(其中#为索引库的结束符),b 有两个后继都是 a,a 有两个后继分别是 b 和 c,c 的后继为#。对于每个字符建立一棵树,树的分支为该字符的后继,如图 1 所示。

图 1 互关联后继树示例

创建该全文互关联后继树的时候,为三个字符:a,b,c 分别建立三棵树,然后按照读取的字符依次为树添加树枝。首先读入 ba,将 b 的后继 a 填入树 b 的第一个分支处,由于此时还不知道该分支对应的位置信息,留空。而后,读入下一个 b,将 b 填入以其前驱 a 为根的树的第一个分支,此分支号即为前次留空的位置信息。再读取 a,在 b 树的第二个分支填写 a,并且回填该分支号 2 到 a 树的第一个分支。……。当读入#后,在以其前驱 c 为根的后继树中增加第一个分支#,并将 1 回填。此时,将索引文件表时成树的形式,即为图 1。图中 b 树的第二个分支是(a,2),因为这个 a 在原文中的后继为 c,而 c 是 a 树的第二个分支。因此,整个互关联后继树构成的森林记载了文档中所有字符的先后位置,可以利用索引生成原文。

针对这种索引,如果用户输入查询字符串"bac",还无法直接查找,检索系统要经过一个匹配的过程。匹配是从查询串的第一个字符开始的,在"b"为根的树的分支中查找后继为"a"的分支,发现有两个分支满足条件,再根据这两个分支的序号,跳转到 a 树的 1,2 分支查看其后继是否为 c,发现第二个分支的内容为

"c",则"bac"在文档中仅有一个匹配。

作为一种优秀的全文索引模型,互关联后继树具有出色的时间效率和空间效率,创建速度也不亚于传统的索引模型。但是,索引数据的动态更新效率还不是特别理想,对于文档的修改只能够通过重新加入来实现,如何提高索引维护的整体效率是一个值得深入研究的问题。

3 索引模块的体系结构和数据结构

本系统的体系结构如图 2 所示,在该系统设计与实现的过程中,需要考虑以下关键问题:原文本库的预处理、原文本无结构化、索引的存储结构设计、索引建立、查询处理。相应地,系统提供以下几个功能模块:文档采集模块、索引模块、查询服务模块。其中,索引模块主要由三个部分组成:主索引、附加索引和删除文件列表。

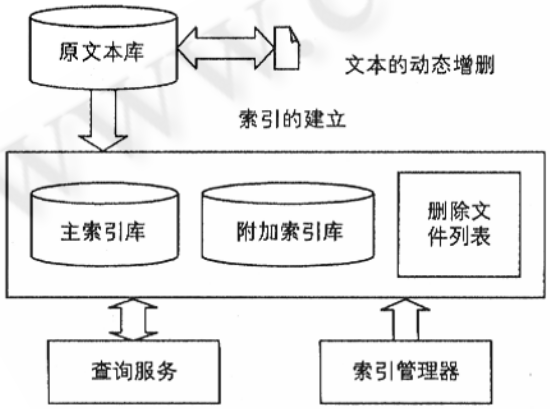

图 2　索引模块体系结构

3.1 数据准备

由于原文本库的来源非常广泛,可以包括 WWW 上的各种静动态网页,Intranet 内部的各种文档、数据资源等,在类型上有 txt 文本、html 网页等。这种来源的复杂性和类型的复杂性使得在建立全文索引以前,要对原文本库进行预处理,对加载到全文数据库中的数据进行收集、整理、归类等处理过程。

3.2 互关联后继树全文索引的表示

为使文本无结构化后可以进行统一的索引,要对待建索引的原文本进行抽取结构信息的处理,记录原文本的基本结构信息和辅助索引信息,以备查询还原时用。互关联后继树文本索引表[3]的框架结构可说明为表 1。

表 1　文本索引表结构

名称	说明
Id	对文献名,描述建索引时用
FileName	文献文件名
StartWord	文献的第一个字,帮助原文生成
FileSize	文献长度
PosNum	在数据库中占据的位子个数
StartOffSet	文献第一个字的相对位移,用来产生该文件中第一个字的后继信息
StartPosition	文献在文本数据库中的开始位置
FileDescription	文献简单描述

对于文本集 D,它的互关联后继树全文索引,可以用二元组表示为:

$FTI = <IST_D, FI>$;

这里,IST_D 和 FI 分别为文本库对应的互关联后继树和文本索引表。称 FTI 为文本库 D 的互关联后继树全文索引模型。

简化的文本索引表为一三元组表示的集合:

$FI = \{D_i, C_n, <C_n, P_{n12}>\}$;

其中,D_i 为第 i 个文献标识;C_n 为 D_i 文献第一个字符;$<C_n, P_{n12}>$ 为 C_n 字符后继树的一个叶子,标志了原文本的首。

3.3 附加索引

为了实现索引的实时更新,本系统的索引设计由三部分组成:主索引、附加索引和删除文件列表。主索引和附加索引采用相同的存储结构,都按照词汇编号排序,如上例中文档 babac#的逻辑存储结构如图 3 所示。

3.4 删除文件列表

删除文件列表中存储数据库中已被删除,但尚未从索引中删除的文档对应的编号。在部分更新时主索引内容并不发生变化,只对附加索引和删除文件列表进行更新,大大节省了更新时间,从而满足实时的要求。

3.5 索引管理器

索引管理器是模块的重要组成部分,它管理索引的并发控制,存取控制,事务处理以及异常处理等。

4 索引的建立、查询和更新

4.1 索引的初始建立

建索引时,采用两次扫描的方法:第一次扫描,扫

描的是原文本，将原文本读入内存，并统计每个字符出

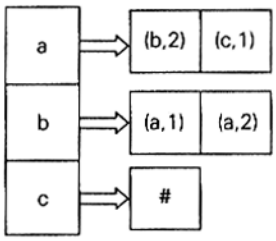

图 3 逻辑存储结构

现的字频。这样做的好处是既在下面的过程中不需要再访问外存，就可以动态的申请内存空间。第二次扫描，扫描的是已经读入内存的原文本，建立索引。

索引建立时所有的内容都存储在主索引中，不涉及附加索引和删除文件列表的操作。

4.2 索引的查询

对用户输入的查询串 s，先从主索引库中查找所有 s 的匹配，之后读取附加索引中与 s 匹配的内容，两个结果相加为最终的查询结果。然后根据删除文件列表，形成结果集。最后再根据文档信息的重要程度以及查询词的相关性对结果排序得到最终的查询结果。

4.3 索引的更新

为了使索引满足动态更新的要求，本系统设计了独特的索引结构，整个索引由三部分组成：主索引、附加索引和删除文件列表。绝大部分索引都存储在主索引中，它不支持索引的插入和删除，要进行主索引的更新需要对索引重建；附加索引存储容量小，支持文档的实时插入，具有良好的更新性能，当有文档插入时，主索引并不更新，只需要把新加入的内容存储到附加索引中即可。

互关联后继树模型更新性能差的主要原因是其互动结构其实是牵一发而动全身，一个原文索引信息的改变，可能导致所有它包含的字符的索引信息的改变。本系统在索引中采用附加索引，由于附加索引相对于主索引很小，可以把索引库的动态改变限制在一个小的范围，而不影响主索引库，从而提高索引的更新性能。

索引的更新包括文档插入、删除和修改。对文档的修改可以看成先删除再插入，因此本系统只处理插入和删除操作。本系统的索引更新方式有两种：实时更新和全面更新。当有文档插入时索引启动实时更新程序，只把文档索引插入到附加索引中，主索引并不发生变化。当附加索引的大小超过一个阈值后，系统会自动触发全面更新，对主索引和附加索引进行合并，重新构建主索引，并且把删除文件列表中的文档从索引中删除，清空附加索引和删除文件列表。

由于索引存储在主索引和附加索引两部分，这就给文档的删除带来了很大的困难，为了解决这一问题，本系统在索引结构中加入删除文件列表，当有文档需要删除时，并不需要知道该文档内容存储在索引的哪一部分，也不需要直接到索引中删除，而是将文档编号记录到删除文件列表中，这样大大节省了删除时间。检索时在返回结果前对结果集进行检查，把其中已经删除的过滤即可。

4.4 主索引和附加索引的合并算法

假设把附加索引库 B 合并到主索引库 A 中，因为主索引和附加索引采用相同的存储结构，索引库中记录的都是相对的位置信息，所以对一个索引库中所有的位置加上一个同样的增量，并同步修改文本索引表不会影响索引库的正确性。因此，索引库合并可以把附加索引库里的信息作一定修改，然后加到主索引库中。

把库 B 合并到库 A 分两个步骤：

4.4.1 合并文本索引表

(1) 库 A 的后继计数读入内存增量数组 Delta[n] 中，设库 A 的最大文件编号为 NA；

(2) 库 B 中每一个文件记录 id + = NA；

(3) 库 B 中的每一个文件记录，StartPostion + = Delta[I] (I = StartWord 的编号)；

(4) 库 B 中所有记录完成(2)(3)两步以后，该文件附加在库 A 的索引文件后。

4.4.2 合并索引文件

(1) 库 A 的后继计数读入内存增量数组 Delta[n] 中；

(2) 对库 B 中的每个索引信息单元 <C,P>，作更新 P + = Delta[C]；

(3) 对每个字符 a，将库 B 中 a 的索引信息，全部附加到库 A 中记录 a 的索引位置信息块的末尾后；

(4) 对库 B 所有字符都做过(3)中的动作以后，写入外存，合并结束。

5 性能分析

测试环境：Pentium(R)4 1.7G CPU，256M 内存，

Windows XP Professional 操作系统，jdk1.4 编译器。测试语料取自网页文件，测试系统在实现中采用了链表结构，实验结果如表 2。

表 2 索引初始建立时间

文件大小	创建时间(秒)
100M	45.895
200M	93.496

通过分析发现，程序的大部分时间消耗在 I/O 操作上。索引创建时间包括：从数据库中读取网页、解析和扫描文档、将结果写入主索引文件。索引的创建时间，是随着索引库的增大而增加的，这与互关联后继树创建算法复杂度 O(M)(M 为全文中的字符数)相符。

如果把 100M 网页文件作为需要更新的文档，更新索引所需要的时间见表 3：在更新索引为空的条件下，进行全面更新需要 46 秒，部分更新需要 45 秒。如果更新前索引中存储有 100M 网页文件的索引(与更新数据不同)，则全面更新需要 98 秒，而部分更新消耗的时间只与更新网页的大小有关，与以前索引中存储多少数据无关，仍只需要 45 秒。从表中数据不难看出对于海量数据，部分更新的速度远远大于全面更新的速度。

表 3 索引更新时间

名称	全面更新时间(秒)	部分更新时间(秒)
索引为空	46	45
索引不为空	98	45

6 结论

本文设计并实现了一种动态更新的索引结构，研究了更新策略。其索引由主索引、附加索引和删除文件列表三部分组成。采取这种索引结构具有很多独特的优点：由于附加索引较小，容易更新；采用删除文件列表避免了直接对索引进行删除操作。独特的结构很好的解决了索引的更新问题，提高了互关联后继树索引的更新效率。

参考文献

1 申展、江宝林等，全文检索模型综述[J]，计算机科学，2004，(31)5：61～64.

2 王智强、刘建毅，一种实时更新索引结构的设计与实现[J]，计算机系统应用，2005，10：79～82.

3 周益群，(中文)全文数据库索引模型研究[D]，复旦大学学位论文 2002 年：47～48.

4 申展、江宝林等，互关联后继树模型及其实现[J]，计算机应用与软件，2005，(22)3：7～9.

5 马科，面向中文的全文数据库建索引的关键技术的研究和实现[D]，复旦大学学位论文 2004 年 5 月：28～29