

# 大规模非结构化数据的索引技术研究

时亚南<sup>1</sup>, 张太红<sup>1,2</sup>, 陈燕红<sup>1</sup>, 郭斌<sup>1</sup>

(1. 新疆农业大学 计算机与信息工程学院, 新疆 乌鲁木齐 830052;

2. 中国农业大学 信息与电气工程学院, 北京 100083)

**摘要:**为解决搜索引擎 ASPSeek 在大规模数据下检索效率低下、占用空间大以及不利于更新等问题,提出了一种分块式存储的倒排索引组织技术,并对基于外存的 B+树索引和线性散列索引的性能进行了比较测试研究。测试结果表明,查询每万条数据耗时线性散列为 B+树索引快 57.40%,插入每万条数据耗时线性散列为 B+树索引的 2.44 倍,删除每万条数据耗时线性散列为 B+树索引的 83.52%,线性散列索引文件大小为 B+树索引文件大小的 109.56%。由测试结果可知,B+树索引具有较快的索引构建和更新速度,而线性散列索引则具有较高的磁盘空间占用率和较好的查询性能。

**关键词:**大规模数据;倒排索引;分块式存储;线性散列;B+树

**中图分类号:**TP31

**文献标识码:**A

**文章编号:**1673-629X(2014)12-0109-05

doi:10.3969/j.issn.1673-629X.2014.12.

## Study on Large-scale Unstructured Data Indexing Technology

SHI Ya-nan<sup>1</sup>, ZHANG Tai-hong<sup>1,2</sup>, CHEN Yan-hong<sup>1</sup>, GUO Bin<sup>1</sup>

(1. School of Computer and Information Engineering, Xinjiang Agricultural University,  
Urumqi 830052, China;

2. College of Information and Electrical Engineering, China Agricultural University,  
Beijing 100083, China)

**Abstract:** To solve the problem that in large-scale data condition the ASPSeek search engine retrievals inefficiently, has large disk space occupancy and can't be conducive to update, propose an inverted index-organized technique based on block storage, and make a performance comparison research test between external memory based B+ tree index and linear hash index. Test results show that, for queries per million data-consuming linear hashing to B+ tree index is 57.40%, for inserting per million data-consuming linear hash is 2.44 times to B+ tree index, for deleting every million data-consuming linear hash to B+ tree index is 83.52%, linear hash index file size is 109.56% of B+ tree index file size. According to the test results, B+ tree index has the faster index building and updating speed, while linear hash index has the higher disk space occupancy rates and better query performance.

**Key words:** large-scale data; inverted index; block storage; linear hash; B+ tree

## 0 引言

大规模数据的分析与处理技术成为当今社会人们研究和讨论的热点问题。随着互联网技术的蓬勃发展,非结构化数据的数量日趋增大,面对呈现爆炸式增长的大规模非结构化数据,如何从中快速准确地获取有价值的信息成为各行业面临的一个严峻挑战。传统的商业数据库主要用于管理结构化数据,缺乏对存储在数据库字段中的内容进行索引和分析处理的机制,

因此,它难以满足用户对大规模非结构化数据快速准确的访问需求;而搜索引擎不仅能够处理大规模数据,而且还可以为非结构化数据建立全文索引,因此可以为用户提供快速有效的查询。

搜索引擎的核心是全文索引,即它能够为文本中每个出现的词、短语、句子等建立索引,并将其保存至索引库中。常见的索引模型有署名文件、位图、PAT树、互关联后继树、倒排索引模型等,而倒排索引模型

收稿日期:2014-02-19

修回日期:2014-05-21

网络出版时间:2014-10-23

**基金项目:**新疆自治区高校科研计划项目(XJEDU2013S13);新疆维吾尔自治区科技攻关项目(200931103);新疆农业大学前期资助课题(XJAU201117)

**作者简介:**时亚南(1988-),男,硕士研究生,研究方向为农业信息检索;张太红,博士,教授,通讯作者,研究方向为农业信息化技术、数据库技术等。

**网络出版地址:**http://www.cnki.net/kcms/detail/61.1450.TP.20141023.1124.032.html

具有实现简单和检索速度快等优点,因此成为应用最广泛的索引模型。在倒排索引模型中,倒排表是目前研究的一个热点,其主要研究目标是:在不影响检索效率的前提下,尽可能减小倒排文件占用的空间体积。Moffat 等<sup>[1-2]</sup>力图通过字对齐的二进制倒排索引压缩的方式,减少访问磁盘倒排索引文件的 I/O 次数;文献[3-9]则从改进倒排索引存储结构的角度出发,研究了索引结构对索引模型整体性能的影响;杨晓波等<sup>[10-17]</sup>提出了通过设计合理的倒排索引文件缓存机制,减少访问磁盘文件开销,从而达到提升检索性能目的的方案。文中在以上研究的基础上,通过设计与研究高效的存储结构,并与合理的缓存替代算法相结合的方式,以期实现对大规模数据的快速准确检索。

## 1 索引模型构建

### 1.1 文档集合来源

文中通过农业垂直搜索引擎 ASPSeek 将互联网中的网页抓取下来,使用 zlib 压缩的方式将网页源码存储到数据库中,待抓取完成后,将其从数据库中解压出来,从数据库中读取到每个单独的 txt 文档中,并按网页编号 url\_id 命名 txt 文档,这样每个 txt 文档中其实存储的是从互联网上抓取的网页的源码。文中从互联网中抓取 200 万张网页,并随机抽取 100 万张网页作为样本集进行比较测试研究。

### 1.2 文档预处理

由于样本集中的网页带有 html 标签,所以首先需要对网页源码进行去标签化处理,文中选用常用的 html 解析库 htmlParser 对网页源码中的文本信息进行了抽取。其次,待文本信息抽取完成后,使用中科院开发的开源分词工具 iclclass 对文本信息进行分词处理,在分词过程中 iclclass 会自动加上词性标注,因此还需通过正则表达式将词性标注去除,并将分词结果写入一个临时分词文件中。最后还要将停用词过滤掉,将分出的每个非停用词加上相应的词编号 cid 及词频 count 写入到相应的词典文件中。

### 1.3 正排索引构建

从临时分词文件中读出网页编号 url\_id 和网页的内容,把 url\_id 和网页的内容对应的分词结果存入一个 HashMap 中,每当处理完一个目录下的所有 txt 文档,便将结果写入到一个指定的正排文件中。因为内存大小有限,不能将所有网页的 url\_id 和网页的内容放入其中,所以可以设置一个写入正排文件的网页阈值 FORWARDFILE\_THRESHOLD,即正排网页的个数达到此阈值便开始将其写入到正排文件中。

### 1.4 分块式倒排索引构建

倒排索引的构建过程就是将正排中的词条 term

映射成其编号 term\_id,然后将每一个正排文件解析成词条编号 term\_id 到 map<url\_id, term\_count>的映射,并写入到相应的倒排文件的过程。具体步骤如下:

(1)对于每一个正排文件目录(100 个),首先需要将正排文件中的 url\_id 到 map<term, term\_count>的集合映射解析成 term\_id 到 map<url\_id, term\_count>的映射;

(2)将 term\_id 相同的所有文档的 id 及相应的词频信息放到倒排记录表中,每个倒排记录表记录的是该词的 term\_id 对应的文档 id 及相应的词频信息;

(3)处理完一个目录后,将该内存中的倒排索引按照 url\_id%100 散列到对应的倒排文件中,散列到哪个临时倒排文件中,便写入哪个临时倒排文件中;

(4)最后,通过基于败者树的多路归并,将 100 个临时倒排文件归并成一个倒排文件。

### 1.5 索引检索

分块式倒排索引的检索过程为:将用户提交的查询条件,经过中文分词后,在词典文件中找到相应词所对应的 term\_id,首先将 term\_id 集合从 B+树索引或线性散列索引中得到 term\_id 所对应的记录标识 rid (pageNo 和 slotNo),然后再根据 rid 从倒排索引文件库中读出相应的页,在读出页中找到 url 编号 url\_id、词频 term\_count 等信息,并根据 url\_id 读出 url 及内容等信息,最后对不同的检索词得到的文档进行评分,并将结果集进行分组排序,将最终结果展示给终端用户。

索引模型构建流程图如图 1 所示。

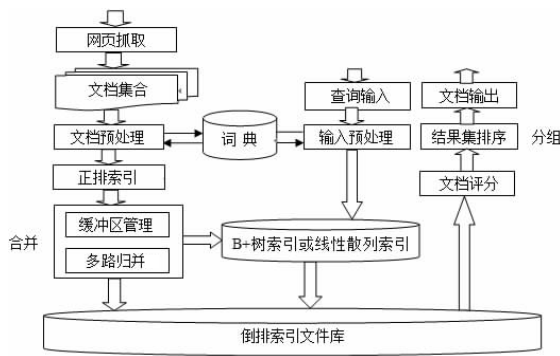


图 1 全文检索系统结构图

## 2 索引文件结构设计

### 2.1 词典文件设计

词典文件存放每个词条的编号 cid 及词条本身等信息,在磁盘上它以顺序文件的方式存储词条集合。由于词典文件本身只有几十兆,因此可以在系统加载的时候将其全部读入内存中。

### 2.2 倒排表的存储结构

倒排表实际按块存储。倒排表主要记录每个词的编号 cid 及出现的频率 count 等信息,每个倒排索引文

件包含一个头页 HeadPage,若干个目录页 CataPage 和若干个数据页 Page。头页中存储文件描述信息 (dump)、目录页个数 (cataPageCnt)、数据页个数 (pageCnt)、第一个目录页的页号 (firstCataPageNo)、第一个数据页的页号 (firstPageNo) 和当前头页的页号 (curHeadPageNo)。目录页则存储当前目录页页号 (curCataPageNo)、下一个目录页页号 (nextCataPageNo) 和数据页数组 pageEntry (包括可用空间 freeSpace 和数据页页号 dataPageNo)。而数据页则记录了数据 data、下一页页号 (nextPageNo)、当前页可用空间 freeSpace、记录标识 rid (包括当前页页号 pageNo 和记录插槽号 slotNo) 以及当前页记录数 recCnt 等信息。

### 2.3 缓存区管理机制

为提高倒排索引的构建速度和检索效率,文中设计并实现了一套专门的缓存系统,整个缓存系统包含一个用于跟踪每个缓存帧状态的缓存帧描述器 BufDesc 和一张用于将文件及页号映射到缓存池帧号的动态哈希表 BufHashTable。缓存帧描述器记录该缓存页是否被修改过、该缓存页是否可用以及该缓存页是否为有效页等信息,它通过双向链表将所有 BufDesc 类的实例链接在一起。缓存替代策略使用爱憎算法,即采用给帧加 love/hate 标记的方式选择被替代出去的页,它的效果类似于 LRU/MRU 算法。

### 2.4 基于 B+树的倒排索引

在以块为单位的存储系统中 (尤其是文件系统), B+树是一种最常用的数据结构,它可以显著提高检索的效率。主要原因是 B+tree 可以拥有大量分支 (通常大于 100,这不同于二叉树,它只有两个分支),因此可以减少在树中查找某个元素所需的 I/O 操作次数。

在基于外存的 B+树索引文件中,通常内部节点的指针为指向子树的页号 (PageNo),而叶节点中键所对应的值则为数据记录的唯一标识 (RID,由 pageNo+SlotNo 构成)。

基于 B+树的索引实现过程中,需要在原有页结构的基础上新增一些页结构,新增页结构分为叶子页 BPLeafPage 和分支页 BPBranchPage,它们均从 Page 页继承而来。

下面给出基于外存的 B+树索引构建算法,该算法的输入为当前页 currentPageNode,叶子页记录 record,当前层数 currentLayer,输出为分支页记录 newRecord,具体过程如下:

- (1) 初始化 newRecord 为空。
- (2) 如果层数 Layer 为 1,执行步骤(3)和(4)。
- (3) 如果当前页有可用空间,直接插入到当前页,返回 newRecord 值。
- (4) 如果当前页没有可用空间,分裂当前页 (分配

一个新的叶子页作为当前页的右兄弟节点,解订新页,然后将当前页中的记录和新记录平均分配到当前页和新页中,最后用新页的最小键值 Key 和页号 pageNo 设置 newRecord)。如果当前页是根节点,树长高一层,分配一个分支页作为新的根节点,解订分支页,用当前页的页号和新页的页号设置新的根节点的记录,并用新的根节点的页号设置当前页和新页的父节点的页号;如果当前页不是根节点,则直接返回 newRecord。

(5) 如果层数不为 1,则执行步骤(7)至(9)。

(6) 根据当前页和键值 Key 找到当前页节点的子树的页号,然后根据子树的页号从缓存中读取该子节点,并将该子节点页解订。

(7) 将子节点页,叶子页记录,当前层数减 1 作为新的输入值,递归调用算法自身,执行步骤(2)至(6),并将递归调用的结果赋值给 newRecord。

(8) 如果当前页未滿 (表示下层没有发生分裂),则直接返回 newRecord。

(9) 如果当前页已滿 (表示下层发生分裂),则分裂当前页 (首先分配一个新的分支页,然后解订该分支页,将该分支页的记录平均分配到当前页和新的分支页中,用当前页的最小记录设置 newRecord,用新分支页的页号设置 newRecord 的页号,最后删除新分支页的相应记录)。如果当前页是根节点,则树长高一层 (首先分配一个分支页作为新的根节点,解订该分支页,然后将新分支页的最小记录推向新的根节点中,用当前页的页号设置新根节点的最左页号,用新分支页的页号设置 newRecord 的页号,用新的根节点的页号设置当前页和新的分支页的父节点的页号,最后将层数加 1); 否则,直接返回 newRecord。

### 2.5 基于线性散列的倒排索引

线性散列是一种动态的散列技术。它不仅是一种高效的存储结构,而且也是一种常见的查找方法,它可以常数平均时间执行增删查改四种常见基本操作。与可扩展散列相比,它不需要专门用于存储数据桶指针的目录项,且能够更灵活地处理桶满情况及选择桶进行分裂的合适时机;而与线性表、二叉树、二叉排序树以及 B-树等存储结构相比,在进行查找操作时,其检索的效率不依赖于给定值与关键码的比较次数,而是将节点的关键码与节点的存储位置建立一种对应关系,这样,只通过节点的关键码便可定位该节点。因此,它是一种面向查找的存储结构,在精确匹配查询方面,它有着广泛的应用,它的缺点是不支持范围查找。

下面给出基于外存的线性散列的构建算法,该算法的输入为词条编号 term\_id 和数据记录的标识 rid,算法的输出为布尔值 (插入成功为 true,否则为 false)。具体算法描述如下:

(1)调用查找函数 search(term\_id)检查要插入的 term\_id 是否在散列表中,如果已存在的话,直接返回 false;如果不存在,执行步骤(2)至(8)。

(2)通过散列函数 hash(term\_id, level) 计算出 term\_id 散列到散列表中的地址 index,如果计算出的 index 值小于准备分裂的桶号 next(即分裂点),则 level+1 进行再散列 rehash,重新计算该词条编号在散列表中的地址。这里 level 表示当前散列函数的级别(即分裂轮数)。

(3)根据第(2)步计算出的散列表中的位置 index,得到该 term\_id 所处的页。

(4)如果该页未满,直接将记录 term\_id 和 rid 插入到该页中。

(5)如果该页已满,则需要进行分裂,执行步骤(6)至(8)。

(6)分配一个新页,并解订该新页,按 level+1 对分裂点 next 所指向页中的记录(如果有溢出页,溢出页中的记录也要处理)进行重新散列,并将所有散列出的值 newIndex 不等于 next 的记录插入到新页中,并从原有页中将这些记录删除,而散列出的值 newIndex 等于 next 的记录仍然保留在原有页中。

(7)如果新页中有记录,不为空,则将其加到散列表并移动分裂点 next,将 next 值加 1。

(8)如果分裂点 next 的值大于  $2^{level} * N - 1$  ( $N$  表示初始轮数的级别),则将分裂轮数 level 加 1,分裂点 next 值置为 0(表示分裂点回到初始位置,分裂点移动了一轮)。

3 实验设计

文中针对基于块的 B+树和基于块的线性散列两种不同存储结构的倒排索引进行比较研究,比较的主要是为了得出在相同的硬件、相同的编程环境以及相同编程语言的前提下,哪种存储结构具有较快的检索性能、较快的索引构建速度和更新速度,以及较低的磁盘占用率。测试软硬件环境见表 1。

表 1 实验环境

名称	属性
CPU	Inter(R) Core(TM)2 Duo CPU E8400 @ 3.00 GHz
内存/GB	2
OS	Windows7 旗舰版
文件系统	NTFS 页/块大小为 4 096 字节
集成开发环境	Eclipse Juno(4.2)
编程语言	Java
文档数	1 001 476
文档大小/GB	5.9(zip 压缩)
词条数	876 538
词典大小/MB	26

3.1 带缓冲的模式下 87 万词条随机检索

文中设置缓冲区大小为 400 M,块大小为 4 096 字节,比较测试了基于块的 B+树和基于块的线性散列两种存储模式下对 87 万词条的随机检索效率。测试结果如图 2 所示。

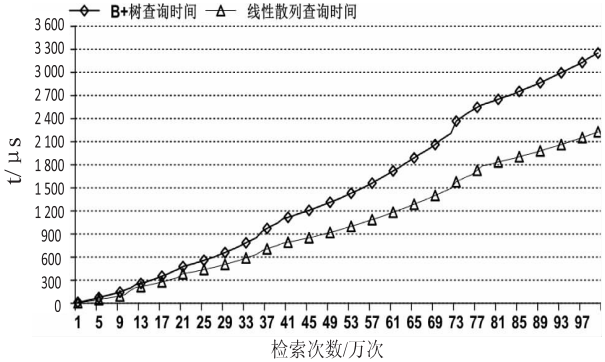


图 2 带缓冲模式下 87 万词条随机检索

通过线性拟合,可以得到 B+树索引和线性散列索引检索用时与检索次数的关系。假设检索用时用  $Y$  表示,检索次数用  $X$  表示,则 B+树索引检索用时与检索次数的关系可以表示为: $Y=0.191\ 8X+22.838$ ,线性散列索引检索用时与检索次数的关系可以表示为: $Y=0.110\ 1X+16.718$ 。从拟合结果可以得出结论,查询每万条数据耗时线性散列为 B+树索引的 57.40%。理论上,执行一次查询,线性散列索引通常需要 1-2 次 I/O,而 B+树索引则需要 4-5 次 I/O,即便将 B+树索引的前两层放入内存,仍需要 2-3 次 I/O。因此,无论是从实践的角度还是从理论的角度看,线性散列索引的查询性能均优于 B+树索引。

3.2 带缓冲模式下 87 万词条的插入删除效率

实验条件同 3.1,插入删除情况如图 3 所示。

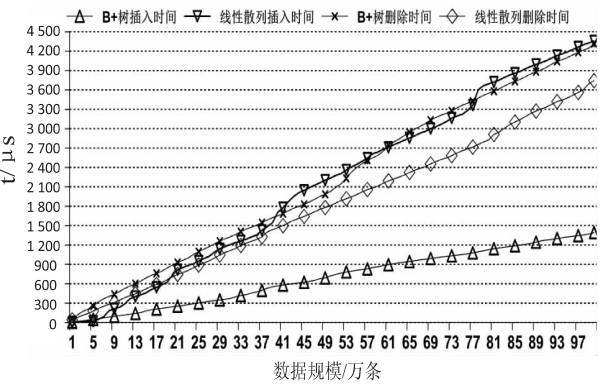


图 3 带缓冲模式下 87 万词条的插入删除效率比较

从图 3 可以看出,插入删除耗时随数据规模的增大趋于线性,通过拟合,可以得出线性散列索引插入耗时  $Y(s)$  与数据规模  $X$ (万条)之间的关系为: $Y=46.926X-193.33$ ,B+树索引插入耗时  $Y(s)$  与数据规模  $X$ (万条)之间的关系为: $Y=14.529X-32.307$ ;线性散列索引删除耗时  $Y(s)$  与数据规模  $X$ (万条)之间的

关系为:  $Y = 36.539X - 21.902$ , B+树索引删除耗时  $Y$  (s) 与数据规模  $X$  (万条) 之间的关系为:  $Y = 43.75X - 3.4389$ 。从拟合结果可以得出, 线性散列插入每万条记录平均耗时约为 B+树散列的 2.44 倍, 而删除每万条记录耗时则相当于 B+树的 83.52%。

### 3.3 两种存储模式下占用磁盘空间大小比较

如图 4 所示, 两种索引的空间复杂度趋于线性, 通过线性拟合, 得出线性散列索引文件大小  $Y$  (MB) 与插入数据规模  $X$  (万条) 之间的关系为:  $Y = 0.4708X - 2.6305$ , B+树索引文件大小  $Y$  (MB) 与插入数据规模  $X$  (万条) 之间的关系为:  $Y = 0.4258X - 0.5633$ 。从拟合结果可以得出如下结论: 线性散列索引文件大小为 B+树索引文件大小的 109.56%, 因此, B+树这种存储模式具有较低的磁盘占用率。

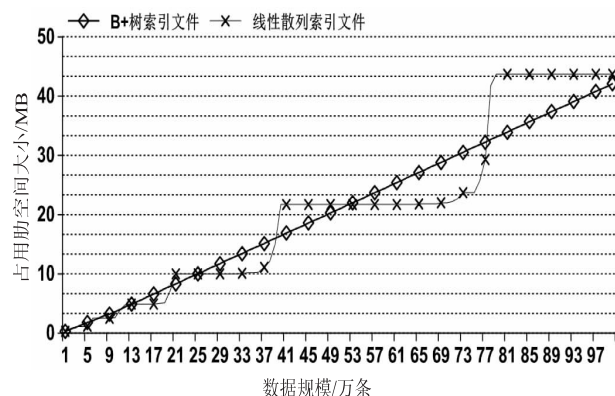


图 4 两种存储模式下占用磁盘空间大小比较

## 4 结束语

文中主要比较了基于块的 B+树和基于块的线性散列两种存储模式的倒排索引的构建用时、更新用时、检索用时及索引文件磁盘占用率。从测试结果可以看出, 基于块的线性散列索引虽然具有较高的磁盘占用率, 但却表现出较好的检索性能。由此可见, 分块式线性散列索引存储模式是以牺牲空间大小来换取检索时间降低的做法。另外, 通过测试可以发现 I/O 仍然是制约检索性能的一个重要瓶颈。因此如何进一步降低系统 I/O 成为笔者下一步需要着重解决的重点内容。

在下一步的工作中, 笔者希望增加对倒排索引块内压缩算法的研究与测试。通过索引压缩, 一方面可以降低索引的存储开销, 另一方面, 也可以增加缓存的

利用率, 加快数据从磁盘到内存的传输速度, 从而减少了 I/O 时间, 提高了倒排索引的检索性能。

### 参考文献:

- [1] Anh V N, Moffat A. Inverted index compression using word-aligned binary codes[J]. Information Retrieval, 2005, 8(1): 151-166.
- [2] Witten I H, Moffat A, Bell T C. Managing gigabytes: compressing and indexing documents and images[M]. 2nd ed. [s. l.]: Morgan Kaufmann, 2009.
- [3] Margaritis G, Anastasiadis S V. Low-cost management of inverted files for online full-text search[C]//Proc of CIKM. Hong Kong: [s. n.], 2009: 102-107.
- [4] Zobel J, Moffat A. Inverted files for text search engines[J]. ACM Computing Surveys, 2006, 38(2): 345-351.
- [5] 刘小珠, 彭智勇, 陈旭. 高效的随机访问分块倒排文件自索引技术[J]. 计算机学报, 2010, 33(6): 977-987.
- [6] 马健, 张太红, 陈燕红. 中文搜索引擎分块倒排索引存储模式[J]. 计算机应用, 2013, 33(7): 2031-2036.
- [7] 邓攀, 刘功申. 一种高效的倒排索引存储结构[J]. 计算机工程与应用, 2008, 44(31): 149-152.
- [8] 吴文娟, 车明. 搜索引擎倒排索引技术的改进[J]. 微处理机, 2006, 27(6): 83-85.
- [9] 丁华, 廖学军, 汪荣峰. 基于海量空间信息的索引技术研究[C]//中国索引会议年会暨学术讨论会论文集. 上海: 出版者不详, 2005.
- [10] 杨晓波. 倒排文件索引缓存机制的优化[J]. 计算机系统应用, 2012, 21(5): 96-99.
- [11] 杨岳. 非结构化数据统一访问平台及索引技术研究[D]. 郑州: 解放军信息工程大学, 2010.
- [12] 杨建武, 陈晓鸥. 半结构化数据相似搜索的索引技术研究[J]. 计算机学报, 2002, 25(11): 1219-1226.
- [13] 余斌. 海量非结构化数据分布式分析与检索[D]. 杭州: 浙江大学, 2012.
- [14] 熊才权, 马乐乐, 孙贤斌. 空间索引技术研究[J]. 计算机技术与发展, 2010, 20(10): 219-223.
- [15] 郑榕增, 林世平. 基于 Lucene 的中文倒排索引技术的研究[J]. 计算机技术与发展, 2010, 20(3): 80-83.
- [16] 孙春菊. 云环境下数据模型和索引技术研究[D]. 南京: 南京邮电大学, 2013.
- [17] 刘小珠, 彭智勇. 全文索引技术时空效率分析[J]. 软件学报, 2009, 20(7): 1768-1784.

# 大规模非结构化数据的索引技术研究

作者：[时亚南](#)，[张太红](#)，[陈燕红](#)，[郭斌](#)，[SHI Ya-nan](#)，[ZHANG Tai-hong](#)，[CHEN Yan-hong](#)，[GUO Bin](#)

作者单位：[时亚南, 陈燕红, 郭斌, SHI Ya-nan, CHEN Yan-hong, GUO Bin\(新疆农业大学 计算机与信息工程学院, 新疆 乌鲁木齐, 830052\)](#)，[张太红, ZHANG Tai-hong\(新疆农业大学 计算机与信息工程学院, 新疆 乌鲁木齐 830052; 中国农业大学 信息与电气工程学院, 北京 100083\)](#)

刊名：[计算机技术与发展](#)

英文刊名：[Computer Technology and Development](#)

年，卷(期)：2014(12)

引用本文格式：[时亚南](#). [张太红](#). [陈燕红](#). [郭斌](#). [SHI Ya-nan](#). [ZHANG Tai-hong](#). [CHEN Yan-hong](#). [GUO Bin](#) 大规模非结构化数据的索引技术研究[期刊论文]-[计算机技术与发展](#) 2014(12)