高效的随机访问分块倒排文件自索引技术

刘小珠[1),2)] 彭智勇[3)] 陈 旭[3)]

[1)](武汉大学软件工程国家重点实验室 武汉 430072)

[2)](武汉理工大学自动化学院 武汉 430070)

[3)](武汉大学计算机学院 武汉 430072)

摘 要 针对倒排索引空间开销大、查询时间效率低以及难以同时支持连接布尔查询和排序查询的问题,提出了一种同时提高空间效率与查询时间效率的高效随机访问分块倒排文件自索引 RABIF.为了在降低空间消耗的同时支持连接布尔查询与排序查询,RABIF 将倒排列表进行合理地分块,然后对每个子块的不同部分采用相应的压缩方式,在不需要插入任何附加辅助信息的前提下实现压缩索引的快速定位与随机访问.理论分析及实验结果表明,与忽略倒排文件自索引 SIF 相比,提出的 RABIF 空间开销平均减少 5.3%,布尔查询时间平均减少 17.8%;对于 0.2%与 1%排序查询,查询时间分别平均减少 34.4%与 27.5%.

关键词 倒排文件;自索引;时间效率;空间效率;随机访问

中图法分类号 TP391 **DOI 号**: 10.3724/SP.J.1016.2010.00977

An Efficient Random Access Block Inverted File Self-Index Technology

LIU Xiao-Zhu[1),2)] PENG Zhi-Yong[3)] CHEN Xu[3)]

[1)](*State Key Laboratory of Software Engineering*, *Wuhan University*, *Wuhan* 430072)

[2)](*School of Automation*, *Wuhan University of Technology*, *Wuhan* 430070)

[3)](*School of Computer*, *Wuhan University*, *Wuhan* 430072)

Abstract In order to overcome the problems of the huge space cost, low query performance and being unable to support conjunctive Boolean query and ranking query simultaneously of inverted index, a time and space efficient random access block inverted file (RABIF) self-index is proposed. To decrease space consumption and support conjunctive Boolean query and ranking query simultaneously, the authors' RABIF appropriately divides inverted list into sub-blocks, and then it compresses different parts of each sub-block with corresponding compression method, which makes fast localization and random access of compressed index into reality without inserting any additional auxiliary information. Theoretical analysis and detailed simulation results prove that, compared with existed skipped inverted file (SIF) self-index scheme, the authors' RABIF averagely reduces space cost by 5.3%, conjunctive Boolean query time by 17.8%; for 0.2% and 1% ranking queries, RABIF decreases query time averagely by 34.4% and 27.5% respectively.

Keywords inverted file; self-index; time efficiency; space efficiency; random access

---

收稿日期:2009-02-01;最终修改稿收到日期:2010-05-10.本课题得到国家"九七三"重点基础研究发展规划项目基金(2007CB310806)、国家自然科学基金(60573095)和武汉大学 2008 年博士研究生自主科研项目(20086350101000066)资助.**刘小珠**,女,1977 年生,博士研究生,讲师,研究方向为全文检索技术.E-mail: lxz_h@163.com. **彭智勇**,男,1963 年生,博士,教授,博士生导师,研究领域为复杂数据管理、可信数据管理、Web 数据管理.E-mail: peng@whu.edu.cn. **陈 旭**,男,1984 年生,博士研究生,研究方向为专利检索与问答系统.E-mail: xu_chen@mail.whu.edu.cn.

1 引 言

随着文本信息量的迅速增长,对高性能全文检索系统的需求越来越迫切.如何快捷有效地管理和检索文本这种非结构化的数据成为一项紧迫的研究任务[1-2].由于倒排索引技术具有实现相对简单、查询速度快以及易于支持同义词查询等扩展功能的优点,被广泛应用于大规模文本集[3]与其它各种信息的检索[4-5].但面对与日俱增的海量文本信息,倒排索引空间开销大、查询效率低的弊端日益凸现[6],如何在降低倒排索引存储空间的同时提高查询效率成为全文检索的突出问题.

采用适当的编码策略对倒排索引进行压缩,不仅可以节省存储空间,而且可以减少查询时读取索引数据所需的磁盘 I/O 次数与时间,从而提高检索速度[1,7-8].因此,当前大多数研究都集中在如何对倒排索引进行有效的压缩方面[1-3].为了降低倒排索引的存储空间,Moffat 等[9]提出了一种递归的二进制内插编码(Binary Interpolative Coding,BIC)算法对倒排索引进行压缩;BIC 利用相邻两个数的信息,对单调递增的倒排列表进行紧凑的递归编码,不仅压缩率高,而且解码速度快;该算法还考虑了文档中词出现的频率分布,以聚类(clustering)的方式对倒排列表的压缩性能进行优化,有效地提高了索引的空间效率.文献[10]采用基于字行的二进制编码方式(Word-Aligned Binary Code,WABC)对索引进行压缩;该编码具有字节操作的优点,其紧凑的二进制特性不仅保证了索引的压缩性能,而且提高了查询时倒排列表的解码速度.针对 WABC 中每个字(32 比特)存在部分比特浪费的缺点,Anh 等[11]提出了一种压缩性能更高的滑动(sliding)压缩算法;其思想是充分利用每个字中的剩余比特存储索引信息,提高空间利用率;但其解压复杂性更高,在一定程度上降低了查询性能.考虑到倒排列表中数据分布的特性,文献[12]提出了一种基于聚类的混合编码压缩算法;该算法根据设定的阈值,将倒排列表中的整数按其差值与阈值之间的大小关系分成不同的簇,然后对小于阈值的簇与大于阈值的簇分别采用不同的压缩编码算法,以提高索引的空间性能;但如何确定合适的阈值以及这种混合编码方式在查询时的解码性能如何,该文并没有阐述.文献[13]中同时考虑了搜索引擎中的索引压缩与索引缓存机制,并对变比特编码等几种压缩算法的性能进行了比较分析.为了进一步提高检索性能,文献[14-17]对倒排列表文档标识(Identity,ID)整数集的升序特征进行了研究,通过更加紧凑的表示方法、快速求交集算法以及对文档 ID 顺序进行优化,提高查询效率.文献[7-8,18]通过实验研究进一步表明了对索引进行压缩不仅可以提高空间效率,而且有助于提高访问速度.

为了在降低倒排索引存储空间的同时提高查询效率,Moffat 等[3,19]提出了一种有效的忽略倒排文件(Skipped Inverted File,SIF)自索引(Self-index);SIF 将倒排列表分成多个子块,通过在相邻子块间插入少量的忽略信息(Skipping information),使得查询时只需对压缩索引进行部分解码,从而提高检索的时间效率;当选择合适的子块大小并以递归方式对倒排列表进行压缩时,可以提高存储空间与查询时间效率;但该方法存在以下问题:(1)加入的辅助信息增加了索引的空间消耗,尤其是当子块较小时,辅助信息所造成的额外空间开销将不可忽略,空间效率将大大下降;(2)无法找到合适的子块大小使得空间效率与连接布尔查询、排序查询等性能同时提高.

尽管上述研究工作对倒排索引的时间和空间性能进行了改进,但仍然存在以下问题[20]:(1)倒排文件存储空间与查询时间效率还有待进一步提高.上述方法大多只针对其中的一个性能进行研究,或提高了一个性能却导致另一个性能恶化.如何实现空间效率与时间效率的性能折衷具有相当的挑战性.(2)上述研究方法并没有充分考虑索引查询的可扩展性,无法同时适应不同查询方式快速定位与解码的要求,导致查询性能相差较大.如排序查询需要访问倒排列表中的词频信息,现有方法必须完全解码索引才能实现词频信息的访问,导致时间消耗较大.(3)当倒排列表被分成大量的小块时,现有方法不仅会导致大量的额外空间开销,而且由此产生的磁盘 I/O 时间将超过压缩所带来的性能.

本文针对倒排文件空间开销大、查询效率低以及难以同时支持连接布尔查询和排序查询的问题,提出一种同时提高存储空间与查询效率的高效随机访问分块倒排文件自索引 RABIF.RABIF 将倒排列表进行合理地分块,然后对每个子块的不同部分采用相应的压缩方式,以提高索引的压缩率;同时采用高效的二进制内插编码方式,在不需要插入任何附加辅助信息的前提下实现压缩索引查询的快速定位,使得压缩索引具有随机访问能力,在查询时只对索引中需要的部分进行解码,以提高检索的时间效率.与现有的工作相比,提出的 RABIF 自索引具有

以下特点：(1) 在不插入任何附加辅助信息的前提下，压缩的 RABIF 索引具有快速定位与随机访问能力；(2) 具有快速的连接布尔查询与排序查询性能；(3) 同时提高了存储空间效率与查询时间效率.

2　随机访问分块倒排文件自索引

为了在提高索引空间效率的同时，支持高效的连接布尔查询和排序查询，提出的倒排文件自索引 RABIF 的思想是：将倒排列表进行分块，构造一种具有随机访问功能的分块倒排列表. 每个子块由两部分组成：第一部分为定位部分. 采用 d-gap 压缩方式，实现压缩索引查询的快速定位，使得压缩索引在不解压的情况下具有子块级的随机访问能力，以提高查询时间效率；第二部分为信息部分. 以部分解码的方式支持连接布尔查询与排序查询，并以高效的二进制内插编码方式进行压缩，在不需要插入任何辅助信息的前提下，实现查询时子块内信息的随机访问与快速定位功能，在减少空间消耗的同时提高查询效率.

2.1　索引结构与创建

提出的倒排文件自索引 RABIF 的结构如图 1 所示. 索引由词典 (Dictionary) 和分块倒排列表 (Posting lists) 组成. 词典包含两部分信息：(1) 词 w_i 以及包含该词的文档数量 s_i；(2) 指向包含 w_i 具体位置信息的分块倒排列表 L_i^{block} 的指针 p_i. 分块倒排列表 L_i^{block} 由 m 个子块 SB_r ($r\in[1,m]$) 组成，每个子块 SB_r 包括定位部分 Loc_r 与信息部分 I_r. 定位部分 Loc_r 为子块 SB_r 的第一个文档标识 (Identity，ID) 与累计词频信息序列对；除了第 m 个子块 SB_m 的信息部分 I_m 保持原倒排列表 L_i 中文档 ID 与累计词频信息序列对的格式外，其它子块的信息部分 I_r 由其它 $k-1$ 个信息对分别按文档 ID 与累计词频的升序排列而成的两个列表 LD_r 与 LF_r 组成.

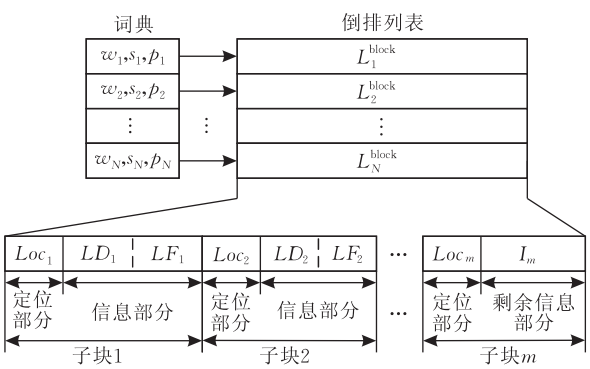


图 1　提出的倒排文件自索引结构

对于包含 n 个信息序列对 (d_j, fq_j) 的倒排列表 L_i：

$$L_i=(d_1,fq_1),(d_2,fq_2),\cdots,(d_n,fq_n) \tag{1}$$

其中 d_j 表示文档的 ID，且有 $d_j<d_{j+1}$，$j\in[1,n-1]$，fq_j 表示该词在文档 d_j 中出现的次数. 分块倒排列表 L_i^{block} 的构建算法 BIL 如算法 1 所示.

算法 1.　分块倒排列表构建算法 BIL.

输入：倒排列表 L_i，每个子块包含的信息序列对数 k

输出：分块倒排列表 L_i^{block}

1. 对 L_i 中的词频 fq_j 以累加方式得到累计词频 f_j，并用 f_j 替换 fq_j，实现 f_j 的升序排列.

2. 以 k 个信息序列对为单位将 L_i 分成 m 个子块 SB_r.

3. 对前 $m-1$ 个子块 SB_r 的信息部分 I_r，分别将文档 ID 与累计词频按照其原来的先后顺序排成文档 ID 和累计词频两个列表 LD_r 与 LF_r. 对 LD_r 与 LF_r 采用 C_2 编码方式进行压缩.

4. 对所有子块 SB_r 的定位部分 Loc_r 以及第 m 个子块 SB_m 的信息部分进行 d-gap 压缩，并采用编码方式 C_1 进行压缩.

5. 返回分块倒排列表 L_i^{block}.

根据 BIL 算法，首先用 f_j 替换词频 fq_j：

$$f_j=\sum_{r=1}^{j} fq_r,\ \ j\in[1,n] \tag{2}$$

然后将倒排列表 L_i 分成 m 个子块 SB_r ($r\in[1,m]$)，且每个子块包含定位部分 Loc_r 与信息部分 I_r. 则有

$$SB_r=(Loc_r,I_r),r\in[1,m] \tag{3}$$

$$m=\lceil n/k\rceil \tag{4}$$

其中 k 为每个子块中包含的信息序列对的数量. 对于第 m 个子块，其信息序列对的数量可能小于 k.

根据 BIL 算法的第 3 步，对于定位部分 Loc_r 有

$$Loc_r=(d^r,f^r),\ \ r\in[1,m] \tag{5}$$

其中 d^r 和 f^r 分别表示第 r 个子块的第 1 个信息序列对的文档 ID 与累计词频，且有

$$d^r=d_{k\cdot(r-1)+1},r\in[1,m] \tag{6}$$

$$f^r=f_{k\cdot(r-1)+1},r\in[1,m] \tag{7}$$

对于信息部分 I_r 有

$$I_r=\begin{cases}(LD_r,LF_r), & r\in[1,m-1]\\ (d_{k(m-1)+2},f_{k(m-1)+2}),\cdots,(d_n,f_n), & r=m\end{cases} \tag{8}$$

$$LD_r=(d_{k\cdot(r-1)+2},\cdots,d_{k\cdot r}),\ \ r\in[1,m-1] \tag{9}$$

$$LF_r=(f_{k\cdot(r-1)+2},\cdots,f_{k\cdot r}),\ \ r\in[1,m-1] \tag{10}$$

然后对 LD_r 与 LF_r 采用 C_2 编码方式进行压缩.

根据 BIL 算法的第 4 步，对所有子块的定位部

分进行 d-gap 压缩后有

$$d^{r\prime}=\begin{cases}d_1, & r=1\\ d_{k\cdot(r-1)+1}-d_{k\cdot(r-2)+1}, & r\in[2,m]\end{cases} \tag{11}$$

$$f^{r\prime}=\begin{cases}f_1, & r=1\\ f_{k\cdot(r-1)+1}-f_{k\cdot(r-2)+1}, & r\in[2,m]\end{cases} \tag{12}$$

从表达式(8)中可以看出,由于第 m 个子块 SB_m 的信息部分 I_m 维持原倒排列表 L_i 中信息序列对格式,即当 $j\in[(m-1)k+1,n]$ 时,信息部分格式为 (d_j,f_j),因此采用 d-gap 压缩后有

$$d'_j=d_j-d_{j-1},\ j\in[(m-1)k+2,n] \tag{13}$$

$$f'_j=f_j-f_{j-1},\ j\in[(m-1)k+2,n] \tag{14}$$

然后采用编码方式 C_1 进行压缩.

经过 BIL 算法后有

$$L_i^{\text{block}}=BIL(L_i)=(Loc_1,I_1),\cdots,(Loc_j,I_j),\cdots,(Loc_m,I_m),\ j\in[1,m] \tag{15}$$

2.2 分块倒排列表的编码

为了尽可能降低分块倒排列表 L_i^{block} 的空间消耗,L_i^{block} 的每个子块 SB_r 的定位部分 Loc_r、信息部分 I_r 的两个列表 LD_r 与 LF_r 分别采用不同的编码方式 C_1 和 C_2 进行压缩. 当子块 SB_r 的定位部分 Loc_r 采用编码方式 C_1 进行压缩时,所需比特数用 $B_{C_1,k}(Loc_r)$ 表示;当子块 SB_r 信息部分 I_r 的两个列表 LD_r 与 LF_r 采用编码方式 C_2 进行压缩时,存储这两种信息所需要的比特数分别用 $B_{C_2,k}(LD_r)$ 和 $B_{C_2,k}(LF_r)$ 表示.

由于每个子块 SB_r 的定位部分 Loc_r 只包含一个信息序列对,因此可以采用压缩性能好的 Golomb 编码进行压缩(当然也可以采用其它性能更好的编码算法).

为了在不加入任何辅助信息的前提下,支持倒排列表的随机访问与快速定位以提高查询的时间效率,关键的问题是寻找一种有效的编码方式,这种编码方式在对每个块中信息部分的文档 ID 和累计词频进行压缩后具有精确的寻址和随机访问能力. 这意味着子块 SB_r 的信息部分 I_r 的编码方式必须满足以下条件:(1)在不附加任何辅助信息的前提下,具有块间忽略功能,这要求在当前已知定位部分 Loc_r 与 Loc_{r+1} 的信息时,能计算出下一个子块 I_{r+1} 的地址;(2)支持块内定位功能,即在实现子块定位后,能在压缩的子块上,实现块内任意元素的随机访问、定位与解码功能.

二进制内插编码 BIC[9] 不仅可以对升序的整数集进行有效压缩,而且在已知升序的整数集的最小整数与最大整数时,能有效地计算出压缩该整数集所需的空间消耗. BIC 编码的这些特点与倒排列表中文档 ID 本身就是升序排列、所要实现的精确寻址和随机访问能力相吻合. 因此,如果通过块间忽略功能,得到当前已知定位部分 Loc_r 与 Loc_{r+1} 的信息时,就可以在满足上述条件的同时准确地计算其压缩后所需要的比特数,进而实现上述两个条件所需满足的功能. 因此本文采用高效的二进制内插编码 BIC 对信息部分的两个列表 LD_r 与 LF_r 进行压缩.

根据二进制内插编码 BIC 的原理,对于子块 SB_r 信息部分 I_r 的升序列表 LD_r,其 $k-1$ 个整数经过编码后所占用比特数取决于列表中整数的取值范围 D. 为了在无需对 LD_r 解码的前提下计算出 LD_r 所占的空间,可由已知的 SB_r 定位部分 Loc_r 的文档号 $d_{k\cdot(r-1)+1}$ 与下一个子块 SB_{r+1} 定位部分 Loc_{r+1} 的文档号 $d_{k\cdot r+1}$ 来计算 D. 因此,对于子块 SB_r 的信息部分列表 LD_r 有

$$B_{C_2,k}(LD_r)=B_{\text{BIC},k}(D)=B_{\text{BIC},k}(d_{k\cdot r+1}-d_{k\cdot(r-1)+1}-1) \tag{16}$$

同理,对于子块 SB_r 的信息部分列表 LF_r 有

$$B_{C_2,k}(LF_r)=B_{\text{BIC},k}(f_{k\cdot r+1}-f_{k(r-1)+1}-1) \tag{17}$$

对于二进制内插编码 BIC 有

$$B_{\text{BIC},k}(D)=\begin{cases}0, & D=k-1\\ (k-1)\cdot\lceil\log_2 D\rceil, & \text{其它}\end{cases} \tag{18}$$

对于子块 SB_r 的信息部分 I_r 的两个列表 LD_r 与 LF_r,其文档 $d_{k\cdot(r-1)+1+j}$ 与对应词频 $f_{k\cdot(r-1)+1+j}$ 之间的物理位置满足以下关系:

$$B_{\text{BIC},k,r}(d_{k\cdot(r-1)+1+j},f_{k\cdot(r-1)+1+j})=\frac{(k-j)}{k-1}\cdot B_{\text{BIC},k}(d_{k\cdot r+1}-d_{k\cdot(r-1)+1}-1)+\frac{(j-1)}{k-1}\cdot B_{\text{BIC},k}(f_{k\cdot r+1}-f_{k(r-1)+1}-1),j\in[1,k-1] \tag{19}$$

例 1. 某词 w_i 的倒排列表 L_i 包含 $n=10$ 个信息序列对 $L_i=(1,2),(2,3),(4,1),(5,2),(6,4),(8,2),(10,3),(12,1),(15,3),(17,2)$,当 $k=4$ 时,对 L_i 进行 BIL 编码.

根据算法 1,首先用表达式(2)求累计词频并进行替换有

$$L_i^1=(1,2),(2,5),(4,6),(5,8),(6,12),(8,14),(10,17),(12,18),(15,21),(17,23).$$

然后以 $k=4$ 个序列对为单位对 L_i 进行分块:

$$L_i^2=[(1,2),(2,5),(4,6),(5,8)],[(6,12),(8,14),(10,17),(12,18)],[(15,21),(17,23)].$$

接着对前 $m-1(\lceil 10/4\rceil-1=2)$ 个子块的信息部分分别按文档 ID 与累计词频进行升序排序有

$$L_i^3=[(1,2),(2,4,5),(5,6,8)],\ [(6,12),(8,10,12),(14,17,18)],\ [(15,21),(17,23)].$$

然后对前 $m-1$ 个子块的信息部分进行 BIC 编码，由 $Loc_1=(1,2)$ 与 $Loc_2=(6,12)$，此时 LD_1 的 3 个整数的取值范围一定在[2,5]之间，即最多有 4 个取值，只需要 2 比特对每个整数进行编码. 同理，对于 LF_1 的 3 个整数的取值范围一定在[3,11]之间，最多有 9 个取值，只需要 4 比特. 依此类推，BIC 编码后可得

$$L_i^4=[(1,2),(00,10,11),(0010,0100,0110)],\ [(6,12),(001,011,101),(001,100,101)],\ [(15,21),(17,23)].$$

接着对每个子块的定位部分及最后一个子块的信息部分进行 d-gap 压缩有

$$L_i^5=[(1,2),(00,10,11),(0010,0100,0110)],\ [(5,10),(001,011,101),(001,100,101)],\ [(9,9),(2,2)].$$

最后对该部分进行参数为 3 的 Golomb 编码有

$$L_i^6=[(00,001),(00,10,11),(0010,0100,0110)],\ [(101,11100),(001,011,101),(001,100,101)],\ [(11011,11011),(001,001)].$$

2.3　分块倒排列表的解码与随机访问

为了便于解码，需要已知当前子块 SB_r 定位部分 Loc_r 与下一个子块 SB_{r+1} 的信息来计算编码所需的比特数. 因此，其存储顺序作如下调整：

$$L_i^{\text{block}}=Loc_1,Loc_2,I_1,Loc_3,I_2,\cdots,Loc_m,I_{m-1},I_m \tag{20}$$

用 $P(Loc_1)$ 表示分块倒排列表 L_i^{block} 中第一个子块 SB_1 的定位部分 Loc_1 的物理地址，则可计算出 L_i^{block} 中任意子块 SB_r 的定位部分 Loc_r 与信息部分 I_r 的两个列表 LD_r 与 LF_r 的所有地址：

$$P(Loc_r)=\begin{cases}p_i, & r=1,\\ P(Loc_1)+B_{C_1,k}(Loc_1), & r=2,\\ P(Loc_1)+\sum_{j=1}^{r-1}B_{C_1,k}(Loc_j)+\sum_{j=1}^{r-2}[B_{C_2,k}(LD_j)+B_{C_2,k}(LF_j)], & r\in[3,m]\end{cases} \tag{21}$$

$$P(I_r)=P(LD_r)=\begin{cases}P(Loc_1)+\sum_{j=1}^{2}B_{C_1,k}(Loc_j), & r=1\\ P(Loc_1)+\sum_{j=1}^{r+1}B_{C_1,k}(Loc_j)+\sum_{j=1}^{r-1}[B_{C_2,k}(LD_j)+B_{C_2,k}(LF_j)], & r\in[2,m]\end{cases} \tag{22}$$

$$P(LF_r)=P(I_r)+B_{C_2,k}(LD_r),\ r\in[1,m] \tag{23}$$

对于子块 SB_r 的列表 LD_r 的第 j 个文档 ID 的物理地址 $P_d(LD_r,j)$ 有

$$P_d(LD_r,j)=P(LD_r)+\frac{(j-1)}{(k-1)}\cdot B_{C_2,k}(LD_r),\quad j\in[1,k-1] \tag{24}$$

其中 $P(LD_r)$ 与 $B_{C_2,k}(LD_r)$ 分别由表达式(22)与(16)计算.

同理，对于子块 SB_r 的列表 LF_r 的第 j 个文档累计词频的物理地址 $P_f(LF_r,j)$ 有

$$P_f(LF_r,j)=P(LF_r)+\frac{(j-1)}{(k-1)}\cdot B_{C_2,k}(LF_r),\quad j\in[1,k-1] \tag{25}$$

其中 $P(LF_r)$ 与 $B_{C_2,k}(LF_r)$ 分别由表达式(23)与(17)计算.

子块 SB_r 信息部分 I_r 的两个列表 LD_r 与 LF_r 中任意的文档 $d_{k\cdot(r-1)+1+j}$ 与对应词频 $f_{k\cdot(r-1)+1+j}$ 的地址，在已知 $P(I_r)$ 与 $P(LF_r)$ 的前提下，通过表达式(24)与(25)得出.

分块倒排列表的解码与随机访问算法 BILDRA 如算法 2 所示.

算法 2.　分块倒排列表解码与随机访问算法 BILDRA.

输入：$p_i,L_i^{\text{block}},k,C_1,C_2,s_i$，文档号 d

输出：d 对应的词频 fq

```
1. 根据 p_i 利用式(21)计算 P(Loc_1)，根据 C_1 解码得(d^1', f^1')，根据式(11)、(12)与式(2)计算(d_1, fq_1)
2. for(i=2; i≤⌈n/k⌉; i++){
3.    if (d==d_{k·(i-2)+1})
4.       return fq_{k·(i-2)+1}.
5.    else
6.       由式(21)计算 P(Loc_i)并解码得(d^i', f^i')，根据式(11)与式(12)与式(2)计算(d_{k·i+1}, fq_{k·i+1}).
7.       if (d∈(d_{k·(i-1)+1}, d_{k·i+1}))
8.          return BRA(P(Loc_{i-1}), I_{i-1}, Loc_{i-1}, Loc_i, d).
9.       else
```

```
10.      continue
11.  }
12. return 0.
```

BILDRA 算法首先通过对子块的定位部分解码,确定查询文档 d 所在子块.然后调用块内随机访问算法 BRA,实现子块内的快速定位与随机访问.

块内随机访问算法 BRA 如算法 3 所示.

算法 3. 块内随机访问算法 BRA.

输入:$P(Loc_r)$,I_r,Loc_r,Loc_{r+1}文档号 d

输出:d 对应的词频 fq

```
1. 根据式(22)计算 P(LD_r).
2. if (d==d_{k.r+1}+1)
3.    根据式(24)求 P_d(LD_r,1),并对 LD_r的第一个文
      档 ID 进行解码得到 d_temp.
4.    if (d==d_temp)
5.       根据式(25)和式(2)求 P_f(LD_r,1)解码计算 fq
6.       return fq.
7. else if (d==d_{k.(r+1)+1}-1)
8.    根据式(24)求 P_d(LD_r,k-1),并对 LD_r的第 k-1
      个文档 ID 进行解码得到 d_temp.
9.    if (d==d_temp)
10.      根据式(25)和(2)求 P_f(LD_r,k-1)解码计算 fq.
11.      return fq.
12. else
13.      采用折半查找算法
14.      if (∃d_j==d)
15.         根据式(25)计算 P_f(LF_{i-1},j)与 P_f(LF_{i-1},
            j-1),对 f_j 与 f_{j-1} 进行解码,计算 fq=
            f_j-f_{j-1}.
16.         return fq.
17.   else
18.         return 0.
19. return 0.
```

块内随机访问算法 BRA,首先判断查询文档 d 与当前子块定位部分文档 ID 号 $d_{k.r+1}$及下一个子块定位部分文档 ID 号 $d_{k.(r+1)+1}$之间的关系,对于 $d==d_{k.r+1}+1$ 与 $d==d_{k.(r+1)+1}-1$ 两种特殊的情况,可以直接解码得到词频,其它情况采用折半查找算法进行定位,提高解码与查询效率.

例 2. 对例 1 中的数据实现查询:文档 8 是否包含词 w_i,并返回词频.

由表达式(20)可知 L_i 经过编码后的实际存储结构 L_i^7 为

$$L_i^7=(00,001),(101,11100),[(00,10,11),(0010,0100,0110)],(11011,11011),[(001,011,101),(001,100,101)],[(001,001)].$$

根据词表指向倒排列表的指针 p_i,得 $P(Loc_1)=p_i$.根据编码方式可得 $B_{C_1,4}(Loc_1)$.由 BILDRA 算法的步 1 与步 6 可得 $(d^{1'},f^{1'})=(1,2)$ 与 $(d^{2'},f^{2'})=(5,10)$ 以及 $(d_1,f_1)=(1,2)$ 与 $(d_{4+1},f_{4+1})=(d^{1'}+d^{2'},f^{1'}+f^{2'})=(6,12)$.

由于 $8>d_{4+1}$,因此必须找到下一个子块的定位部分 Loc_3.根据 BILDRA 算法,计算出:

$$\begin{aligned}P(Loc_3)&=P(Loc_1)+B_{C_1,4}(Loc_1)+B_{C_1,4}(Loc_2)+\\&\quad B_{C_2,4}(LD_1)+B_{C_2,4}(LF_1)\\&=p_i+31.\end{aligned}$$

对 Loc_3 进行解码得 $(d^{3'},f^{3'})=(9,9)$ 以及 $(d_{8+1},f_{8+1})=(d^{3'}+d_{4+1},f^{3'}+f_{4+1})=(15,21)$.此时有 $8\in(d_{4+1},d_{8+1})$,因此对第 2 个子块的信息部分进行块内随机访问算法 BRA.

根据 BRA,采用折半查找算法,此时对 LD_2 的第 $i(i=\lfloor(k-1)/2\rfloor=1)$ 个文档 ID 进行解码.可知 LD_2 的第 1 个文档 ID 与 8 相同.因此根据式(25)计算词频的地址,并得累加词频值 14,该值减去 f_{4+1},即得频率为 2.

3 性能分析

3.1 空间性能分析

定理 1. 包含 n 个信息序列对的分块倒排列表 L_i^{block},在最坏情况下,采用 BIL 进行编码所占空间 $B_{L_i^{\text{block}}}$ 为 $2n\left(2+\log_2\left(\frac{k}{k-1}\right)+\log_2\left(\frac{N}{2n}\right)\right)$ 比特,其中 N 为 L_i^{block} 中文档 ID 与词频的最大值.

证明. 包含 n 个信息序列对的分块倒排列表 L_i^{block},有 $2n$ 个整数,其取值范围为 $[1,N]$.用 $B[N,2n]$ 表示压缩 $2n$ 个文档 ID 与词频所需要的比特数,对于 Golomb 编码有[9]

$$B[N,2n]\leq 2n\cdot(2+\log_2(N/(2n))) \quad (26)$$

当用 Golomb 编码对 L_i^{block} 的每个子块 SB_r 的定位部分 Loc_r 以及最后一个子块的信息部分进行编码时,所需要的空间为

$$B_{Loc}^{\text{Golomb}}=[2n-2(m-1)(k-1)]\cdot\left[2+\log_2\left(\frac{N}{2[n-(m-1)(k-1)]}\right)\right] \quad (27)$$

其中 m 为子块数量,由表达式(4)计算.

对于前 $m-1$ 个子块的信息部分,采用 BIC 编码,其所需空间为

$$B_I^{\mathrm{BIC}}=\sum_{i=1}^{m-1}[(k-1)\cdot\lceil\log_2(D_{D_i})\rceil]+\sum_{i=1}^{m-1}[(k-1)\cdot\lceil\log_2(F_{D_i})\rceil]\le(k-1)\sum_{i=1}^{m-1}[\log_2(D_{D_i})+1]+(k-1)\sum_{i=1}^{m-1}[\log_2(F_{D_i})+1]\le2(k-1)(m-1)\left[1+\log_2\left(\frac{N}{(k-1)(m-1)}\right)\right] \tag{28}$$

分块倒排列表 L_i^{block} 压缩后的空间大小为

$$B_{L_i^{\mathrm{block}}}=B_{Loc}^{\mathrm{Golomb}}+B_{Loc}^{\mathrm{BIC}} \tag{29}$$

考虑最坏情况下，即取 $m=\lceil n/k\rceil=\frac{n}{k}+1$ 时，对式(29)进行简化有

$$B_{L_i^{\mathrm{block}}}=2n\left[2+\log_2\left(\frac{k}{k-1}\right)+\log_2\left(\frac{N}{2n}\right)\right] \tag{30}$$

定理 1 得证.　　证毕.

从表达式(30)可以看出，$B_{L_i^{\mathrm{block}}}$ 与 k 成反比，且随着 k 的增加而减小，但 k 的取值必须满足范围约束条件 $k\in[2,n-1]$.

推论 1.　在最坏情况下，提出的 RABIF 自索引比 SIF 自索引[3,19]的空间消耗小 $B_{\mathrm{skipping\_information}}^{\mathrm{Golomb}}$ 比特. 其中 $B_{\mathrm{skipping\_information}}^{\mathrm{Golomb}}$ 表示 SIF 自索引中插入的 m 个忽略信息采用 Golomb 编码所需的空间.

证明.　对于 SIF 自索引，根据其结构所需存储空间 $B_{\mathrm{SIF}}^{\mathrm{Golomb}}$ 为

$$B_{\mathrm{SIF}}^{\mathrm{Golomb}}=B_{Loc}^{\mathrm{Golomb}}+B_{\mathrm{skipping\_information}}^{\mathrm{Golomb}}+B_I^{\mathrm{Golomb}} \tag{31}$$

其中 $B_{\mathrm{skipping\_information}}^{\mathrm{Golomb}}$ 表示插入的 m 个忽略信息 $(d^r, pointer_{d^{r+1}})$ 采用 Golomb 编码所需的空间，$pointer_{d^{r+1}}$ 为指向下一个子块忽略信息对的地址指针；B_I^{Golomb} 表示前 $m-1$ 个子块的信息部分，采用 Golomb 编码所需的空间：

$$B_{\mathrm{skipping\_information}}^{\mathrm{Golomb}}=m\left(2+\log_2\frac{N}{m}\right)+m\cdot4\cdot8 \tag{32}$$

$$B_I^{\mathrm{Golomb}}\le2(m-1)(k-1)\left[2+\log\left(\frac{N}{2(m-1)(k-1)}\right)\right]=2(m-1)(k-1)\left[1+\log\left(\frac{N}{(m-1)(k-1)}\right)\right] \tag{33}$$

由表达式(28)与式(33)有 $B_I^{\mathrm{Golomb}}=B_I^{\mathrm{BIC}}$.

因此在最坏情况下，提出的 RABIF 自索引比 SIF 自索引的空间消耗小：

$$B_{\mathrm{SIF}}^{\mathrm{Golomb}}-B_{L_i^{\mathrm{block}}}=B_{\mathrm{skipping\_information}}^{\mathrm{Golomb}} \tag{34}$$

推论 1 得证.　　证毕.

3.2　时间性能分析

根据文献[3,9]的研究表明，BIC 的解码时间性能与 Golomb 相当，而且在某些情况下略优于 Golomb. 以下分析中假设 BIC 与 Golomb 的解码时间性能相同，即两种方法解码一个 (d_j,f_j) 所需的时间均为 t_d.

定理 2.　对包含 n 个信息序列对的分块倒排列表 L_i^{block}，查找项为 q 个时，其最小查询时间 $T_{\min}$ 为 $t_d\left(1+\frac{q}{4}+\sqrt{nq}\right)+nt_r$，其中 t_d 为解码一个 (d_j,f_j) 所需的时间，t_r 为读取一个 (d_j,f_j) 所需的时间.

证明.　考虑最坏情况下，需要将整个倒排列表 L_i^{block} 读入，即需要将 n 个信息序列对都读入内存，则时间为 t_rn. 在最坏情况下，L_i^{block} 中所有 m 个子块的定位部分 $Loc_r(r\in[1,m])$ 都需要解码，则所需时间为 t_dm；每个子块的列表 LD_r 最坏情况下折半查询需要解码一半，而 LD_r 中的每一个文档 ID 解码时间为 $t_d/2$，则所需时间为 $q\cdot\frac{k-1}{2}\cdot\frac{t_d}{2}$；在已知查询项的文档 ID 地址后，$LF_r$ 部分可直接解码得到词频，则所需时间为 $q\cdot\frac{t_d}{2}$. 因此有解码时间 T_d 为

$$T_d=t_dm+q\cdot\frac{k-1}{2}\cdot\frac{t_d}{2}+q\cdot\frac{t_d}{2}=t_d\left(m+q\cdot\frac{k+1}{4}\right) \tag{35}$$

而总时间 T 是读取时间与解码时间 T_d 之和，因此有

$$T=t_d\left(m+q\cdot\frac{k+1}{4}\right)+t_rn \tag{36}$$

由于 m 满足表达式(4)，代入有

$$T=t_d\left(\lceil n/k\rceil+q\cdot\frac{k+1}{4}\right)+t_rn\le t_d\left(\frac{n}{k}+q\cdot\frac{k}{4}+\frac{q}{4}+1\right)+t_rn \tag{37}$$

对于提出的 RABIF 自索引，显然当 k 为

$$k=2\sqrt{n/q} \tag{38}$$

具有最小的解码时间 $T_{d,\min}$ 与总时间 $T_{\min}$ 值：

$$T_{d,\min}=t_d\left(1+\frac{q}{4}+\sqrt{nq}\right) \tag{39}$$

$$T_{\min}=t_d\left(1+\frac{q}{4}+\sqrt{nq}\right)+nt_r \tag{40}$$

定理 2 得证.　　证毕.

推论 2.　在最坏情况下，提出的 RABIF 自索引比 SIF 自索引的查询时间消耗小 $\frac{k-1}{4}qt_d+mt_r$.

证明.　对于 SIF 自索引，在最坏情况下，倒排列表中所有 m 个子块的辅助信息(该信息是一个信息序列对)都需要解码，则所需时间为 t_dm；另外，

SIF 中并不具备块内的随机访问能力，因此最坏情况下，SIF 需要将块内的所有 k 个信息序列对进行解码，这里我们取其平均解码时间，也就是只需要对 $k/2$ 个信息序列对进行解码所需的时间 $t_d \cdot k/2$，因此查询项为 q 个时，可得解码时间 $T_{d,\mathrm{SIF}}$ 的表达式为

$$T_{d,\mathrm{SIF}} = t_d\left(m+q\cdot\frac{k}{2}\right) \tag{41}$$

由于 SIF 自索引中插入了 m 个子块的辅助信息，因此在最坏情况下，需要将整个倒排列表读入，因此其读取时间为 $t_r(n+m)$. 则总时间 T_{SIF} 为

$$T_{\mathrm{SIF}} = t_d(m+q\cdot\frac{k}{2})+t_r(n+m) \tag{42}$$

从编码方式的角度而言，RABIF 定位部分以及最后一个子块的编码方式与 SIF 相同；而不同之处在于 RABIF 中没有插入辅助信息，RABIF 中前 $m-1$ 个子块的信息部分的编码方式采用的是 BIC 编码. 当假设 BIC 与 Golomb 的解码时间性能相同，即两种方法解码一个 (d_j, f_j) 所需的时间均为 t_d 时，可得提出的 RABIF 自索引，在最差情况下比 SIF 自索引的查询时间消耗小：

$$T_{\mathrm{SIF}} - T = \frac{k-1}{4}qt_d + mt_r \tag{43}$$

推论 2 得证. 证毕.

定理 3. 在最坏情况下，对包含 n 个信息序列对的分块倒排列表 L_i^{block}，查找项为 q 个时，同时使空间与查询时间性能最优的 k 的取值范围为 $[\min(2\sqrt{n/q}, n-1), \max(2\sqrt{n/q}, n-1)]$.

证明. 根据定理 1 可知，在最坏情况下，采用 BIL 进行编码所占空间 $B_{L_i^{\mathrm{block}}}$ 与 k 成反比，且随着 k 的增加而减小，但 k 的取值必须满足范围约束条件 $k\in[2, n-1]$. 因此 k 为 $n-1$ 时，空间性能最优.

根据定理 2，对于提出的 RABIF 自索引，显然当 k 为 $2\sqrt{n/q}$ 时具有最小的解码时间与查询时间值. 因此有同时优化空间与查询时间性能的 k 取值范围为

$$k\in[\min(2\sqrt{n/q}, n-1), \max(2\sqrt{n/q}, n-1)] \tag{44}$$

证毕.

4 实验结果

为了验证提出的 RABIF 自索引的性能，对其所占空间大小与查询时间效率进行测试，并与 SIF 自索引进行了对比. 实验数据均在以下硬件实验平台下测试获得：CPU 3.0GHz、硬盘 240GB、内存 1GB、操作系统 Windows XP. 实验数据来自武汉市专利局，数据大小为 193.2GB，该表中含有 42 个属性，其中 40 个属性是文本类型，对这 40 个文本属性均建立了全文索引. 提出的 RABIF 自索引与 SIF 自索引均在开源 Clucene① 搜索引擎上实现，系统实现中将 Clucene 搜索引擎的索引结构以及编解码方法分别替换为 RABIF 与 SIF，其它部分功能（如分词、建词表等），直接采用了 Clucene 系统中的方法. 对于 Clucene 搜索引擎，其压缩后的倒排列表文件所占空间大小为 22.5GB.

4.1 空间性能

图 2 所示为 Clucene 系统不考虑分块机制时索引大小以及经过两种自索引压缩后索引大小的实验结果.

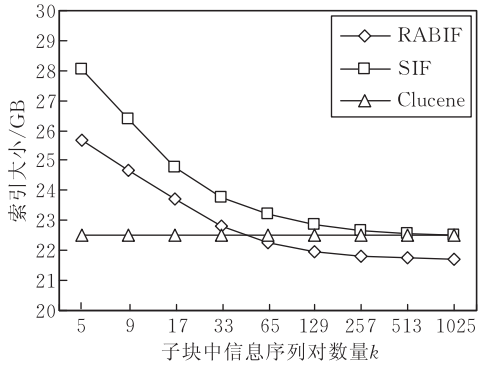


图 2 两种索引所占空间

如图 2 所示，由于没有考虑分块机制，Clucene 系统所产生的索引大小为 22.5GB，与 k 值无关. 提出的 RABIF 自索引所占的空间比例明显小于 SIF，两种自索引所占空间比例平均相差 1.16GB. 随着子块中信息序列对数量 k 的增加，两种索引所占的空间都将逐渐减小. RABIF 索引所占空间从 25.7GB 降低到 21.71GB，原因是当 k 增加时，每个子块内相邻两个数的平均间隔将越来越小，采用 BIC 编码时所需的比特数也越少，因此存储空间消耗逐渐降低. 对 SIF 其值从 28.035GB 降低为 22.52GB，原因在于当 k 增加时 SIF 索引需要插入的冗余信息比例将减少，因此所占空间比例逐渐降低. 随着 k 值的增加，两种索引的空间比例差越来越小. 上述实验结果与表达式(30)与(34)的理论分析基本一致，即 RABIF 索引所占空间与 k 成反比；随 k 值的增加 SIF 索引插入的冗余信息比例将减少，两种索引空间大小的差值减小. 另外，SIF 索引所占空间大小始终大于 Clucene 压缩后倒排列表文件大小，当 k 为 257 时，此时 SIF 的索引大小与 Clucene

① CLucene Search Engine Wiki. Retrieved October 21, 2007, from http://clucene.wiki.sourceforge.net/

系统相当. 而当 k 为 33 与 65 时，RABIF 索引所占空间分别为 22.79GB 与 22.25GB，其大小为 Clucene 系统所形成索引大小的 101.3%与 98.9%. 这表明，当 k 达到 65 时，索引所占空间将小于 Clucene 压缩后倒排列表文件大小，而且随着 k 的增加进一步降低，当 k 增加到 1025 时候，RABIF 索引所占空间为 21.71GB. 这也表明 RABIF 具有降低索引空间消耗的优势.

4.2　查询时间性能

图 3 所示为子块中信息序列对数量 k 分别取 5、129 与 1025 时，RABIF 与 SIF 两种自索引以及 Clucene 系统随连接布尔查询词增加时的性能.

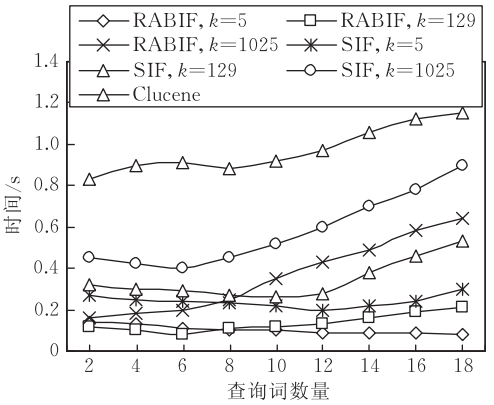


图 3　两种索引连接布尔查询时间

从图 3 可以看出，RABIF 与 SIF 两种索引的布尔查询性能明显优于 Clucene 系统的性能. 其主要原因在于 RABIF 与 SIF 两种索引采用了快速的定位功能，能明显地提高查询的性能. 同时，对于 RABIF 与 SIF，k 值越小，RABIF 与 SIF 两种索引的布尔查询性能越好. 对于相同的 k 值，RABIF 索引时间性能明显优于 SIF，布尔查询时间平均减少了 17.8%. 当 k 分别取 5、129 与 1025，与 SIF 相比，RABIF 的查询时间平均小 0.152s、0.164s 与 0.168s，时间差值基本保持在 0.16s. 这与表达式(43)的理论分析结果基本一致，即两种索引查询时间的差值与查询项 q 及子块数量 m 有关，而 t_d 的取值一般为毫秒级，因此，两种索引查询时间的差值约等于 mt_r. 另外当 k 分别取 5、129 与 1025 时，根据定理 3 的理论分析，此时由于 n 为 48671，通过分析计算此时可得 q 分别为 3893.7、5.9 与 0.1. 图 3 中所示，由于图中 q 的取值范围为[2，18]，因此 RABIF 查询性能的最小值分别出现在 q 为 18、6 与 2 时，这与理论分析一致.

当 k 为 5 时，查询词的数量对 SIF 索引性能影响并不明显. 而当 k 值为 129 与 1025 时，多词查询时间将随查询词数量的增加而呈线性增长. 因此，对于布尔查询，SIF 索引中 k 的取值应小于 129. 对于 RABIF 索引，从图 3 中可以看出，只有当 k 取 1025 时，查询时间的大小才随查询词数量的增加而增长明显. 因此 k 的取值范围更广，其值在 5～1025 之间时都具有良好的布尔查询性能.

图 4 所示为子块中信息序列对数量 k 增加时，两种自索引 0.2%排序查询的性能. 由于 Clucene 系统中并没有采用分块的方法，其 0.2%排序查询时间性能与 k 无关，时间大小为 1.73s. 如图 4 所示，RABIF 的时间性能优于 SIF，尤其是当 $k>33$ 时，性能差更明显，两者时间差从 0.28s 增加到 1.32s；与 SIF 相比，查询时间平均减少了 34.4%. 对于 SIF 索引，当 k 的取值为 5～17 时，查询时间并没有明显的变化，均小于 1.3s. 但随着 k 值的增加，查询时间从 1.5s 迅速增加到 2.8s. 因此，对于排序查询，SIF 的性能受 k 值的影响显著，且 $k<33$ 时性能较好，此时其时间性能为 1.5s，而当 k 值增加到 65 时，查询时间将增加到 1.81s，比 Clucene 系统的性能更差. 其原因在于 k 值越大，每个子块的信息对越多，在排序查询时解码所需的时间越长. 而 RABIF 的时间性能基本在 1.4s 以下，原因是 RABIF 索引中每个子块的解码与 k 的取值相关性不大，只要已知该词的分块倒排列表的指针 p_i，就可以实现随机访问与解码等功能. 这也表明 RABIF 排序查询时，k 取值范围更大.

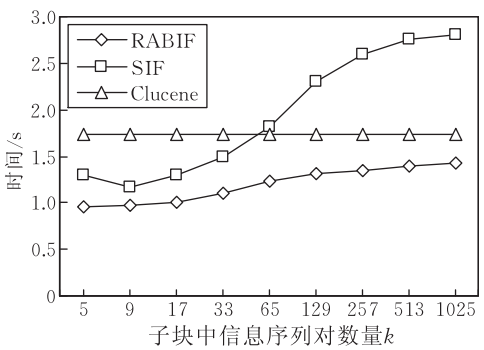


图 4　两种索引 0.2%排序查询时间性能

表 1　两种索引 1%排序查询时间性能

k	查询时间/s	
	SIF	RABIF
5	2.45	1.98
9	2.52	2.03
17	2.63	2.07
33	2.97	2.32
65	3.71	2.58
129	4.54	2.82
257	4.97	2.95
513	5.16	3.09
1025	5.68	3.43

表 1 所示为 1%排序查询时,两种索引在 k 取不同值时的查询性能.对于 1%排序查询,此时 Clucene 系统查询时间大小为 3.91s,与 k 值无关.从表 1 中的实验数据可以得出与图 4 基本一致的结论:提出的 RABIF 索引具有明显的时间性能优势,与 SIF 相比,查询时间平均减少了 27.5%,与 Clucene 系统相比总是低于其查询时间;RABIF 索引排序查询时 k 取值范围为[5,1025],而对于 SIF,当 $k>65$ 时,其性能将比 Clucene 系统差.

4.3 讨 论

通过上述实验,可以得到如表 2 所示优化性能的 k 值范围.对于空间性能,k 值越大两种索引所占的空间越小.当 $k>33$ 时,提出的 RABIF 索引所占空间小于 Clucene 压缩后倒排列表文件大小;对于 SIF,当 $k>129$ 其空间大小为 101.7%与 Clucene 压缩后倒排列表文件大小相当.对于连接布尔查询,SIF 索引中 k 的取值小于 129 时,多词连接布尔查询性能较好;而 k 值在 5～1025 之间时 RABIF 都具有良好的布尔查询性能.对于排序查询,SIF 索引 k 的取值应小于 33,而 k 值对 RABIF 的排序查询时间性能并不明显.因此,SIF 算法难以同时实现空间与时间性能的优化,而当 $k\in(33,1025)$ 时提出的 RABIF 算法性能得以优化.

表 2 优化性能的 k 值范围

	k	
	SIF	RABIF
空间性能	>129	>33
布尔查询时间性能	<129	$\in(5,1025)$
排序查询时间性能	<33	≤1025
k 取值范围	$\varnothing$	$\in(33,1025)$

5 结束语

针对倒排文件空间开销大、查询效率低以及难以同时支持连接布尔查询和排序查询的问题,提出了一种同时提高存储空间与查询时间效率的高效随机访问分块倒排文件自索引 RABIF.为了支持连接布尔查询与排序查询,RABIF 将倒排列表进行合理地分块,然后对每个子块的定位部分与信息部分分别采用 Golomb 与 BIC 编码进行压缩,降低倒排列表的空间消耗;在不需要插入任何附加辅助信息的前提下,RABIF 具有快速定位功能,使得压缩后的索引具有子块级与子块内信息的随机访问能力,显著地提高了查询时间效率.理论分析与实验结果表明,当 k 为 65 时,RABIF 索引所占空间将低于 Clucene 压缩后的倒排列表文件大小;与 SIF 相比,布尔查询时 RABIF 索引的 k 值范围更大,查询时间平均减少了 17.8%;对于 0.2%与 1%排序查询,RABIF 查询时间比 SIF 平均减少了 34.4%与 27.5%.当 $k\in(33,1025)$ 时提出的 RABIF 算法能同时提高存储空间与查询时间效率.

考虑到压缩算法性能对系统性能的影响,下一步的工作将在此基础上,考虑采用性能更优的压缩算法对倒排列表进行编码,进而提高系统的时空性能;并研究高效的索引动态更新机制.

致 谢 衷心地感谢审稿专家与同行富有建设性的意见和建议,感谢武汉大学软件工程国家重点实验室数据库新技术研究小组老师和同学们的大力支持和帮助!

参 考 文 献

[1] Navarro G, Makinen V. Compressed full-text indexes. ACM Computing Surveys, 2007, 39(1): 1-61

[2] Liu Xiao-Zhu, Peng Zhi-Yong. Time and space efficiencies analysis of full-text index techniques. Journal of Software, 2009, 20(7): 1768-1784(in Chinese)
(刘小珠, 彭智勇. 全文索引技术时空效率分析. 软件学报, 2009, 20(7): 1768-1784)

[3] Zobel J, Moffat A. Inverted files for text search engines. ACM Computing Surveys, 2006, 38(2): 1-56

[4] Seoa C, Leeb S-W, Kima H-J. An efficient inverted index technique for XML documents using RDBMS. Information and Software Technology, 2003, 45(1): 11-22

[5] Wang Chao-Kun, Li Jian-Zhong, Shi Sheng-Fei. *N*-gram inverted index structures on music data for theme mining and content-based information retrieval. Pattern Recognition Letters, 2006, 27(9): 492-503

[6] Liu Xiao-Zhu, Sun Sha, Zeng Cheng, Peng Zhi-Yong. An inverted index mechanisms based on buffers. Journal of Computer Research and Development, 2007, 44(S): 153-158(in Chinese)
(刘小珠, 孙莎, 曾承, 彭智勇. 基于缓存的倒排索引机制研究. 计算机研究与发展, 2007, 44(S): 153-158)

[7] Gupta A, Hon W-K, Shah R, Vitter J S. Compressed dictionaries: Space measures, data sets, and experiments. Lecture Notes in Computer Science, 2006, 4007: 158-169

[8] Buttcher S, Clarke C L A. Index compression is good, especially for random access//Proceedings of the 16th ACM Conference on Information and Knowledge Management. Lisboa, Portugal, 2007: 761-770

[9] Moffat A, Stuiver L. Binary interpolative coding for effective index compression. Information Retrieval, 2000, 3(1): 25-47

[10] Anh V, Moffat A. Inverted index compression using word-aligned binary codes. Information Retrieval, 2005, 8(1): 151-166

[11] Anh V, Moffat A. Improved word-aligned binary compression for text indexing. IEEE Transactions on Knowledge and Data Engineering, 2006, 18(6): 857-861

[12] Chen Jin-Lin, Zhong Ping, Cook T. Compressing inverted file index using mixed delta/flat binary code//Proceedings of the 1st International Conference on Digital Information Management. Christ College, Bangalore, India, 2006: 338-343

[13] Zhang Jiang-Gong, Long Xiao-Hui, Suel T. Performance of compressed inverted list caching in search engines//Proceedings of the 17th International Conference on World Wide Web. Beijing, China, 2008: 387-396

[14] Culpepper S J, Moffat A. Compact set representation for information retrieval. Lecture Notes in Computer Science, 2007, 4726: 137-148

[15] Sanders P, Transier F. Intersection in integer inverted indices//Proceedings of 9th Workshop on Algorithm Engineering and Experiments and 4th Workshop on Analytic Algorithms and Combinatorics. New Orleans, LA, USA, 2007: 71-83

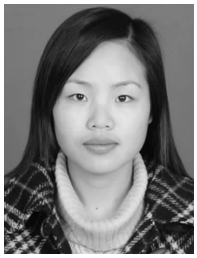
[16] Blandford D K, Blelloch G E. Compact representations of ordered sets//Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA). New Orleans, LA, USA, 2004: 11-19

[17] Yan H, Ding S, Suel T. Inverted index compression and query processing with optimized document ordering//Proceedings of the 18th International World Wide Web Conference. Madrid, Spain, 2009: 401-410

[18] Chen J, Cook T. Using d-gap patterns for index compression//Proceedings of the 16th International World Wide Web Conference. Banff, AB, Canada, 2007: 1209-1210

[19] Moffat A, Justin Z. Self-indexing inverted files for fast text retrieval. ACM Transactions on Information Systems, 1996, 14(4): 349-379

[20] Trotman A. Compressing inverted files. Information Retrieval, 2003, 6(1): 5-19



LIU Xiao-Zhu, born in 1977, Ph. D. candidate, lecturer. Her main research interests focus on full-text retrieval.

PENG Zhi-Yong, born in 1963, professor, Ph. D. supervisor. His main research interests include complex data management, trusted data management and Web data management.

CHEN Xu, born in 1984, Ph. D. candidate. His main research interests include patent retrieval and question answering system.

Background

This work is supported by the National Basic Research Program (973 Program) of China under grant No. 2007CB310806, and the National Natural Science Foundation of China under grant No. 60573095.

Efficiently management and retrieval of a huge amount of text is a key and challenge technology. As one of efficient methods for time and space efficiencies of full-text retrieval, the inverted index technique has been comprehensively studied in recent years. Most of related work concentrates on compressing the index data with appropriate coding scheme to reduce space cost and query time cost. However, such schemes have several disadvantages: (1) the existed work just improves space performance or query time performance, and can not simultaneously improve both them. How to balance the tradeoff between space and time consumption is a challenge. Especially for the huge amounts of digitally available information, it is very important to design an efficient compressed index with less or no space consumption to attain greatly time efficiency; (2) the existed schemes can not simultaneously provide extremely fast query processing of both conjunctive Boolean queries and ranking queries; (3) the existed mechanisms can incur high storage overheads if the posting lists are divided into small blocks and the increase in disk I/O time outweighs the reduction in decompression time.

In order to overcome the problems of the huge space cost, low query performance and being unable to support conjunctive Boolean query and ranking query simultaneously of inverted index, this paper proposes a time and space efficient random access block inverted file (RABIF) self-index. RABIF appropriately divides inverted list into sub-blocks, and then it compresses different parts of each sub-block with corresponding compression method, which makes fast localization and random access of compressed index into reality without inserting any additional auxiliary information. Theoretical analysis and detailed simulation results prove that, compared with the existed skipped inverted file (SIF) self-index scheme, the authors' RABIF greatly reduces space cost, conjunctive Boolean query time cost, 0.2% and 1% ranking queries time cost simultaneously.