基于 B^+ 树的多关键字密文排序检索方法

那海洋 杨 庚 束晓伟
(南京邮电大学计算机学院 南京 210003)

摘 要 针对当前社会信息量大、隐私信息需要加密存储带来的检索难度大等问题,研究分析了现有的基于密文的可搜索加密方案,提出了一种基于 B^+ 树的多关键字密文排序方法。利用向量模型构建索引和查询陷门,根据相关性分数和关键字匹配度对检索结果进行排序。在真实数据集上进行了实验,结果证明本方案具有较高的检索效率。

关键词 B^+ 树,密文检索,多关键字
中图法分类号 TP309 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.01.029

Multi-keyword Ranked Search Method Based on B^+ Tree

NA Hai-yang YANG Geng SHU Xiao-wei
(School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing 210003, China)

Abstract For the large amount of information and the storage of encrypted privacy information in society, it has become more difficult to retrieve these information for users. Based on research and analysis of the existing searchable encryption scheme, a method of multi-keyword ranked search based on B^+ tree was proposed in this paper. Specifically, the vector model is combined in the index construction and trapdoor generation, and the search results are sorted according to the relevance score and the match number of keyword. Finally, the experiments are conducted on the really dataset to demonstrate the search efficiency of the proposed scheme.

Keywords B^+ Tree, Encrypted search, Multi-keyword

1 引言

在这个信息爆炸的时代,互联网上的信息正在以几何级数的速度增长。在这种背景下,海量数据的存储成为了一个难题,如何在这些信息中快速准确地检索到需要的信息成为了一种迫切的需求。由于云服务器拥有传统计算机的存储能力和计算能力,许多用户选择将数据外包到云服务器中,用户利用云服务器充足的存储、计算和网络资源,只需较小的代价即可获得高质量的数据服务。在云环境下,当云用户发送数据到不可信的云服务器端时,他们面对的问题是如何确保其数据在云端是安全的。一个直观的想法是云用户在外包给云服务器前对数据进行加密以保护数据的隐私。

为了保护数据的隐私,云用户将数据的加密形式外包给云服务器,将加密后的数据存储在云服务器中,使得云管理者无法获取数据的明文内容,由此带来的挑战是在云中对密文数据进行检索[1]。近年来,出于功能、效率和安全性等方面的考虑,一种支持用户在密文上进行关键字查找的可搜索加密(Searchable Encryption, SE)[2]被提出。可搜索加密技术是解决这一问题的有效方法,其允许云服务器在对密文不解密的情况下进行检索,并且将密文检索结果返回给用户,由用户根据相关密钥进行解密。

针对以上问题,本文提出一种基于 B^+ 树索引的多关键字密文排序检索方案。该方案利用向量模型将索引和查询向量化,利用 TF-IDF 模型将索引和查询分别表示为 TF 和 IDF 向量,将索引和查询的向量积作为相关性分数。当检索多个关键字时,将相关性分数和关键字的匹配个数作为在 B^+ 树上的检索条件,并利用相关性分数和关键字匹配度对检索文件进行排序。最后,分析了该方案对密文查询的隐私保护,并在真实数据集上进行了实验,结果证明该方案有较高的检索时间效率。

本文第 2 节介绍密文检索方面的相关研究;第 3 节介绍基本符号及系统模型;第 4 节详细介绍基于 B^+ 树索引的检索算法,以及对 B^+ 树上向量的加密方法;第 5 节对该方案的安全性和效率进行理论分析;第 6 节通过分析实验结果,与理论分析进行比较;最后总结本文的研究内容并对未来工作进行展望。

2 相关工作

目前,国内外学者和研究人员在密文检索方面做了大量工作,实现了对密文检索功能的优化、效率的提高以及隐私的保护。2000 年,文献[3]首次提出了单关键字的对称可搜索加密方案,其检索的时间复杂度与其数据集的大小成线性关

到稿日期:2016-03-31 返修日期:2016-05-31 本文受国家自然科学基金资助项目:云计算环境下的新型访问控制理论与关键技术研究(61272084),国家自然科学基金资助项目:云计算环境中面向数据多维隐私保护的关键技术研究(61572263)资助。
那海洋(1990-),女,硕士生,主要研究方向为云环境下的密文检索,E-mail:nahaiyang1991@163.com;**杨 庚**(1961-),男,博士,教授,CCF 高级会员,主要研究方向为网络与信息安全、分布与并行计算、大数据隐私保护;**束晓伟**(1992-),男,硕士生,主要研究方向为可搜索加密、并行计算。

系。2006 年,文献[4]在文献[3]的基础上给出了更严格的安全性定义和更高效的对称密钥可检索加密方法,利用加密 Hash 表存储的关键词和密文文件标识的映射关系实现密文数据查询。为了更好地满足用户的查询需求,文献[5]于 2011 年提出了多关键字密文排名搜索问题,结合 KNN 查询技术[6],将文件和查询表示为长度为字典大小的向量,利用两个向量内积的结果进行排序。但是,此方案并未考虑不同关键字的权重,因此检索排序结果的准确性不高。2012 年,文献[7]提出了一种支持隐私保护的密文排序查询方法,基于无证书认证思想构建了 RQED 框架,设计了支持多属性数据隐私、多关键字检索、合理的密文查询排序函数。为提高密文检索的时空效率,设计了基于层次动态布隆过滤器的 RQED 索引机制。为了提高数据存储的安全性,文献[9]所提出的方案使用 2 个不同的云服务器,一个用于存储加密索引,而另一个用于存储加密的文档集合。这样的分离隐藏了加密陷门和相关文件两者之间的关联,可以防止向云服务器泄漏搜索结果。由于多核处理器的发展,文献[10]提出了次线性搜索时间的并行和动态执行的可搜索加密方案。该方案利用红黑树构建关键字索引,树中的叶子节点存储文件信息,内部节点存储两个关键字哈希表,检索从根节点开始,到达叶子节点,结果返回包含关键字的所有文件;并利用多个处理器同时对索引中的节点进行处理以实现并行。但是,该方案只能实现单关键字检索,并且对检索结果没有进行排序整理。为了方便用户查看检索结果,文献[11]提出一个安全的多关键字密文排序检索方案,该方案基于二叉平衡树构建了索引,并提出了深度优先搜索算法。将向量空间模型和广泛使用的 TF-IDF 模型相结合来生成索引向量和查询向量,并利用向量积作为其相关性得分对检索结果排序。因此,该方案使得返回给用户的文件按照与检索关键字的相关程度从高到低的顺序排序,方便了用户对文件的使用。但是,对于不同的关键词,利用向量积计算出来的相关性分数高,并不能说明该文件与关键词的相关性高。为了进一步提高检索的效率和结果的准确性,本文利用 B^+ 树构建索引,降低了索引的空间复杂度,使得检索的速度加快;同时将相关性分数和关键词匹配度结合,提高了检索的准确性,并对检索结果进行排序,进一步满足了用户的需求。

3 相关定义

3.1 符号定义

F 为文件集,包含 n 个文件,$F=\{f_1,f_2,\cdots,f_n\}$。

C 为密文文件集合,$C=\{c_1,c_2,\cdots,c_n\}$。

n 为文件集的大小。

W 为关键字集合,数量为 m_k,$W=\{w_1,w_2,\cdots,w_{m_k}\}$。

m_k 为词典的大小,即总的关键字个数。

$\boldsymbol{data}$ 为存储在索引中的词频向量,长度等于 m_k。

$\boldsymbol{I}$ 为 $\boldsymbol{data}$ 的加密形式。

$\boldsymbol{Q}$ 为待查询的关键字组成的向量,长度等于 m_k。

$\boldsymbol{T_q}$ 为 $\boldsymbol{Q}$ 的加密形式,即搜索产生的陷门。

3.2 TF-IDF 模型

TF-IDF 模型[8]是一种用于信息检索的向量模型。词频(TF)是一个关键字在一个文件中出现的频率,逆文档频率(IDF)可以用来度量一个关键字的普遍重要性,其由总的文件数目除以包含该关键字的文件的数目,再将得到的商取对数得到。在向量模型中,每个文件可以表示成一个向量,该向量由文件所包含的所有关键字的 IF 值所组成,每个查询也可以表示为由待查询的关键字的 IDF 值所组成的向量。TF 向量和 IDF 向量的长度等于总的关键字数目 m_k,而 TF 向量和 IDF 向量的向量积反应了查询与相应文件的相关度。相关性分数计算函数[9]可以由式(1)表示:

$$Score(\boldsymbol{data},\boldsymbol{Q})=\boldsymbol{data}\cdot\boldsymbol{Q}=\sum_{w_i\in w_q}TF_{w_i}\times IDF_{w_i} \tag{1}$$

在本文的方案中,当节点 u 是 B^+ 树的内部节点时,根据 u 的孩子节点的词频向量来计算 u 的词频向量,具体规则在 4.1 节介绍。当节点 u 是叶子节点时,由式(2)计算[11]:

$$TF_{w_i}=\frac{TF'_{f,w_i}}{\sqrt{\sum_{w_i\in w}(TF'_{f,w_i})^2}} \tag{2}$$

其中,TF'_{f,w_i} 是文件 f 包含的关键字 w_i 的词频值,$TF'_{f,w_i}=1+\ln N_{f,w_i}$,$N_{f,w_i}$ 是文件 f 中关键字 w_i 的数量。

查询向量由式(3)计算[10]:

$$IDF_{w_i}=\frac{IDF'_{w_i}}{\sqrt{\sum_{w_i\in w}(IDF'_{w_i})^2}} \tag{3}$$

其中,IDF'_{w_i} 是关键字 w_i 的逆文档频率,$IDF'_{w_i}=\ln(1+N/N_{w_i})$,$N_{w_i}$ 是包含关键字 w_i 的文件数量,N 是总的文件数。

3.3 系统模型

本文提出的方案由 3 个角色组成:数据拥有者、授权用户和云服务器,如图 1 所示。

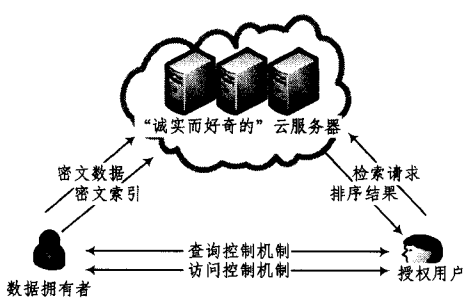


图 1 云环境下加密检索系统结构图

(1)数据拥有者

数据拥有者希望将自己的一些文件外包到云端,考虑到安全性,这些文件需要加密后再进行上传。数据拥有者首先根据文件集建立一个安全可检索的索引,然后将文件集进行加密,将加密的文件集和索引一起发送到云服务器。此外,数据拥有者还负责更新文件集,分发授权用户相应权限的密钥,更新时将更新信息发送至服务器,由服务器端完成更新。

(2)授权用户

授权用户有权限对云服务器端的加密文件进行检索。授权用户输入待检索的关键字并产生相应的陷门,服务器根据陷门进行检索,返回 k 个检索结果。用户根据相应的密钥对检索结果进行解密。

(3)云服务器

云服务器是系统模型中的存储中心,存储的数据包括数据拥有者上传的加密文件集和索引。云服务器根据授权用户发送的陷门对索引树进行检索,返回 k 个最优的加密文件。此外,云服务器根据数据拥有者发送的更新信息,对存储的索引树和加密文件集进行更新。目前,文献[5,12-14]对加密数

据检索的研究认为云服务器是诚实而好奇的，云服务器会返回给用户真实的检索结果，同时根据用户输入的检索信息去猜测更多的数据信息。本文约定云服务器是诚实而好奇的，并且不会和用户合谋攻击整个系统。

4　基于 B+ 树的多关键字检索方案

4.1　B+ 树索引

相比于红黑树、二叉树而言，B+ 树的高度要低很多。而基于树构建的索引，其检索的时间与树的高度成正比，因此本文使用 B+ 树作为索引结构。其中，B+ 树的叶子节点存储文件信息，内部节点存储其孩子节点的信息，并作为导航节点帮助完成检索过程。B+ 树是一种动态平衡的数据结构，在本方案中，树中的每个节点存储一个词频向量 ***data***，节点 u 的结构如式(4)所示：

$$u=\langle Num, fid, children[m], \boldsymbol{data}\rangle \tag{4}$$

其中，Num 是节点 u 的编号；fid 表示文件标识符，如果节点 u 是内部节点，则 fid 等于 0；$children[m]$是指向孩子节点的指针，m 是 B+ 树的阶数；***data*** 是存储 TF 值的向量，内部节点的 ***data*** 值由式(5)计算：

$$\boldsymbol{data}[i]=\max\{u.children[1]\rightarrow \boldsymbol{data}[i],\cdots,u.children[m]\rightarrow \boldsymbol{data}[i]\},i=1,\cdots,m_k \tag{5}$$

4.2　B+ 树索引的构建

在对文件进行加密之前，数据拥有者要根据文件建立索引。基于 B+ 树的索引是通过插入数据建立的，而每次操作都会将其插入到叶子节点。B+ 树中所有指向文件的指针都保存在叶子节点中，上层节点的键值是复制其孩子节点键值中最大的一个。建立索引就是 B+ 树插入数据的过程，以 B+ 树的根节点和待插入的文件作为输入，具体过程如算法 1 所示。

算法 1　Insert (BNode u, f_{id})

```
1. if (u 是叶子节点) then
2.     将 fid插入到叶子节点，更新叶子节点的 data 向量
3.     return;
4. else
5.     根据新插入节点的键值 id，在当前节点查找出向下插入的孩子节点 u'
6.     if (节点 u'的键值个数达到最大) then
7.     对节点 u'进行分裂，重新确定向下插入的孩子节点 u'
8.     endif
9.     Insert(BNode u', fid);
10. endif
```

在本方案的 B+ 树索引中，对于节点 u，$\boldsymbol{data}_u$ 向量保存的是所有关键字的 TF 值，如果 $\boldsymbol{data}_u[i]\neq 0$，表示以节点 u 为根的子树中至少存在一条路径，该路径的叶子节点指向的文件包含关键字 w_i。

4.3　基于 B+ 树索引的多关键字检索

授权用户为了检索云端的数据，发送查询信息给云服务器，希望云服务器返回合适的检索结果。为了存储检索结果，构造一个结果列表 $ResultList=\langle Score, fid, MatchNum\rangle$，列表长度为 k，是用户输入的希望返回的结果数量。$Score$ 是通过式(1)计算得到的文件 f_{fid} 的相关性分数，$MatchNum$ 是文件 f_{fid} 中的关键字与输入的关键字匹配的个数，其值小于或等于输入的关键字数。$ResultList$ 中的结果文件首先根据 $MatchNum$ 的大小进行降序排列；其次，相同的 $MatchNum$ 根据 $Score$ 进行二次排序。随着检索操作的进行，$MatchNum$ 值大的节点优先加入 $ResultList$，如果列表满了就删除相关度最低的文件。以 B+ 树的根节点作为输入，具体检索过程如算法 2 所示。

算法 2　Search(BNode u)

```
1. if(u 是叶子节点) then
2. if ((MatchNum > minMatchNum) 或 (MatchNum == minMatchNum 且 Scoreu > minScore))
3.     删除 ResultList 中 MatchNum 和 Score 最小的元素，将节点 u 加入 ResultList;
4.     return;
5. else
6.     获取节点 u 的所有孩子节点 v
7.     if (v->MatchNum > minMatchNum 或 (v->MatchNum == minMatchNu且 Scorev > minScore))
8.         Search(v);
9.     endif
10.    endif
11. endif
```

$minMatchNum$ 为当前 $ResultList$ 中匹配度的最小值，作为后面文件是否加入 $ResultList$ 的比较条件，初始值为 0。

$minScore$ 为当前 $ResultList$ 中匹配值为 $minMatchNum$ 的文件中 $Score$ 的最小值，初始值为 0。

图 2 是根据包含 n 个文件的文件集建立的索引，设置关键字的总数 $m_k=5$，查询向量 $\boldsymbol{Q}=(0.5,0,0.9,0.4,0.3)$。表 1 是根据查询向量 $\boldsymbol{Q}$ 计算出来的匹配值和相关性分数。假设用户需要返回的文件数 $k=6$，算法从根节点 N_0 开始进行搜索，经过 N_1 依次到达 N_3，N_4，此时结果列表有 f_2，f_3，f_1，f_5，f_6，f_4。然后到达 N_5，由于 f_7 和 f_8 的匹配值较大，将其加入结果列表，此时列表为 f_2，f_8，f_3，f_1，f_7，f_5。左子树搜索完成，继续搜索右子树，经 N_2 到达 N_6，此时结果列表为 f_2，f_{11}，f_8，f_3，f_1，f_7。到达 N_7 时，结果列表为 f_2，f_{11}，f_8，f_{14}，f_3，f_1。当到达 N_8 时，因为列表中当前匹配值为 3，而节点 N_8 的匹配值为 2，所以停止搜索，最终返回给用户的结果列表为 f_2，f_{11}，f_8，f_{14}，f_3，f_1。

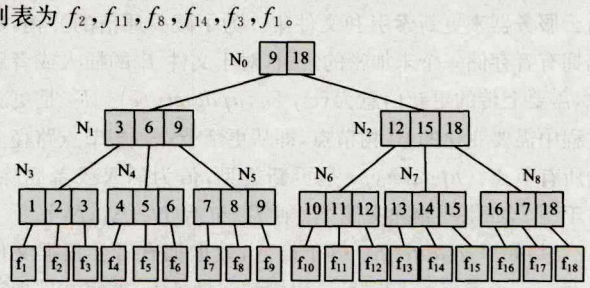


图 2　B+ 树索引

表 1　各节点匹配度和分数

f_{id}	M	Score	f_{id}	M	Score	N_{id}	M	Score
f_1	3	0.71	f_{10}	1	0.9	N_0	4	1.82
f_2	4	1.06	f_{11}	4	0.5	N_1	4	1.82
f_3	3	0.8	f_{12}	2	0.48	N_2	4	1.65
f_4	1	0.9	f_{13}	2	0.44	N_3	4	1.46
f_5	3	0.4	f_{14}	3	0.84	N_4	3	1.3
f_6	2	0.48	f_{15}	2	0.41	N_5	3	1.22
f_7	3	0.45	f_{16}	2	0.39	N_6	4	1.4
f_8	3	0.97	f_{17}	2	0.37	N_7	3	1.07
f_9	2	0.68	f_{18}	1	0.81	N_8	2	1.09

4.4　向量加密

由于可搜索加密机制的构造方法众多，因此其算法描述方法各不相同。文献[6]基于 KNN(k-nearest neighbor)查询

算法实现了对索引向量的加密。结合 KNN 查询方法，本文方案包括 5 个算法，分别为 *Setup*，*GenBTreeIndex*，*GenToken*，*calcMatch* 和 *calcScore*。

(1)*Setup*。数据拥有者随机产生一个密钥向量 $K \leftarrow \{0,1\}^k$，其长度等于字典长度 m_k。

(2)*GenBTreeIndex*。首先通过算法 1 建立未加密的索引，索引中的每个节点存放 ***data*** 向量，数据拥有者要根据密钥向量 $\boldsymbol{K}$ 将 ***data*** 向量随机拆分成两个向量 $\boldsymbol{data}_1$，$\boldsymbol{data}_2$。具体规则：如果 $\boldsymbol{K}[i]=0$，$\boldsymbol{data}_1[i]=\boldsymbol{data}_2[i]=\boldsymbol{data}[i]$；如果 $K[i]=1$，$\boldsymbol{data}_1[i]+\boldsymbol{data}_2[i]=\boldsymbol{data}[i]$，$\boldsymbol{data}_1[i]$ 和 $\boldsymbol{data}_2[i]$ 是两个和等于 $data[i]$ 的随机数。最后，索引树上节点存储的是两个加密向量 $\boldsymbol{I}=\{\boldsymbol{data}_1,\boldsymbol{data}_2\}$，索引树加密完成。

(3)*GenToken*。用户输入待检索的关键字集 w_q，生成查询向量 $\boldsymbol{Q}$。如果 $w_i \in w_q$，$\boldsymbol{Q}[i]=IDF_{wi}$，否则 $\boldsymbol{Q}[i]=0$。同样地，根据密钥向量 $\boldsymbol{K}$，将查询向量随机拆分成两个向量 $\boldsymbol{Q}_1$，$\boldsymbol{Q}_2$。具体规则：如果 $\boldsymbol{K}[i]=1$，$\boldsymbol{Q}_1[i]=\boldsymbol{Q}_2[i]=\boldsymbol{Q}[i]$；如果 $\boldsymbol{K}[i]=0$，$\boldsymbol{Q}_1[i]+\boldsymbol{Q}_2[i]=\boldsymbol{Q}[i]$，$\boldsymbol{Q}_1[i]$ 和 $\boldsymbol{Q}_2[i]$ 是两个和等于 $\boldsymbol{Q}[i]$ 的随机数。最后，该算法输出陷门 $\boldsymbol{T}_q=\{\boldsymbol{Q}_1,\boldsymbol{Q}_2\}$。

(4)*calcScore*。根据陷门和加密的索引向量，每个节点都可以计算出一个相关性分数。根据式(6)可知，由加密后的索引向量和查询向量计算出来的相关性分数与未加密向量计算出来的分数相等。

$$\boldsymbol{I}\cdot\boldsymbol{T}_q=\boldsymbol{data}_1\cdot\boldsymbol{Q}_1+\boldsymbol{data}_2\cdot\boldsymbol{Q}_2=\boldsymbol{data}\cdot\boldsymbol{Q}=Score(\boldsymbol{data},\boldsymbol{Q}) \tag{6}$$

(5)*calcMatch*。在检索过程中，当 $\boldsymbol{data}_1[i]\times\boldsymbol{Q}_1[i]+\boldsymbol{data}_2[i]\times\boldsymbol{Q}_2[i]\neq 0$ 时，$MatchNum=MatchNum+1$。该算法输出关键词匹配度 *MatchNum*。

4.5 动态更新

如果数据拥有者想要更新自己的数据，那么云服务器中的索引和文件就要进行同步的更新。由于本方案中的索引是基于 B^+ 树的，因此对文件更新就要对索引树中的节点进行更新。由数据拥有者将要更新的信息发送到云服务器端，然后由云服务器来更新索引和文件集。为了减少通信的开销，数据拥有者存储一个未加密的索引，对于文件 f_i 的插入或者删除，需要上传的更新信息为 $\{c_i, T_N, updatetype\}$。T_N 是更新过程中需要变化的 B^+ 树节点，即从更新节点到根节点路径上的所有节点。*updatetype* 是更新类型，值为插入或者删除。对于图 2，如果要删除文件 f_8，则 T_N 包括 $\{N_5, N_1, N_0\}$。

(1)如果 *updatetype* 等于插入，数据拥有者会根据文件 f_i 生成一个节点 u，同时计算出词频向量 ***data***，并将文件加密成 c_i。然后，将 u 插入到索引树的叶子节点中，并且根据式(5)更新从该节点到根节点路径上的所有节点的 ***data*** 向量。最后如 4.4 节描述的那样，利用密钥 $\boldsymbol{K}$ 对这些更新的 ***data*** 向量重新加密，得到加密的节点集 T_N。

(2)如果 *updatetype* 等于删除，数据拥有者先找到 f_i 所在的叶子节点并将其删除，如果删除节点打破了索引树的平衡，可以利用 B^+ 树本身的平衡性质保持树的平衡。然后更新根路径上的所有节点的信息，并利用密钥 $\boldsymbol{K}$ 对这些节点的向量进行加密，得到 T_N。

(3)云服务器收到数据拥有者发送过来的更新信息，用 T_N 中的节点替换原来索引中的对应节点，产生新的加密索引。如果是删除操作，就将 c_i 从加密文件集中删除；如果是插入操作，将 c_i 加入到加密文件集。

在对文件集进行更新后，文件集的总数 n 发生了变化，而关键字的 IDF 值是由文件总数计算出来的，所以 IDF 也发生了变化。但是根据文献[11]，在对文件进行删除或插入 100 到 300 个文件时，关键字的 IDF 值变化很小。因此，不需要每次更新都重新计算 IDF 值，可以在云服务器空闲或者更新的文件数达到一定数量时更新关键字的 IDF 值。

5 安全性和效率

5.1 安全性分析

(1)索引和检索机密性。索引中不存储真实的明文关键字，而是将关键字映射到索引向量的对应位置。由于 $\boldsymbol{I}$ 和 $\boldsymbol{T}_q$ 是根据密钥 K 加密的向量，云服务器在没有密钥 K 时无法得到原始的向量 ***data*** 和 $\boldsymbol{Q}$。

(2)数据隐私保护。对文件集的加密可以使用传统的对称加密，如 AES。由于文件和索引向量采用不同的方式进行加密，因此增加了攻击者的攻击难度，但这并不会影响对文件和索引的操作。对文件的检索、插入和删除操作都是基于文件标识符的，不会涉及到文件的内容，因此不会泄露文件的内容。

(3)查询的不关联性。根据数据拥有者的密钥 $\boldsymbol{K}$，将查询向量随机拆分成两个向量并生成陷门。因此对于相同的查询，每次产生的查询陷门也不相同，因此实现了查询的不关联性。

(4)关键字隐私保护。索引和检索的机密性保护了关键字的原始向量信息，云服务器能够知道的只是检索过程中计算出来的相关性分数，这没有泄露关键字的信息。因此，在云服务器只知道密文集合和索引时，关键字隐私得到了保护。但是，当云服务器知道更多的信息时，比如云服务器知道了某些关键字的词频分布情况，就可能发起词频攻击来推断这些关键字[11,13,15]。

5.2 效率分析

在 B^+ 树索引的检索过程中，对于结点 u，如果 $Score_u$ 和 $MatchNum_u$ 满足加入 *ResultList* 的条件，那么继续访问结点 u 的孩子节点，否则就返回。因此，在实际的检索中，B^+ 树索引中的许多节点都没有被访问到。由于检索是从索引树的根到叶子节点的遍历过程，因此算法的时间复杂度与树的高度相关。假设一次检索中包含待检索关键字 w_q 的子集的叶子节点的数量为 θ，用户需要返回的文件数为 k，那么 θ 一定大于 k，并且小于文件集大小 n。相关性分数计算的时间复杂度为 $O(m_k)$，当索引为二叉树时，由于二叉树的高度为 $\log n$，因此检索的时间复杂度为 $O(\theta m_k \log n)$。本方案中的 m 阶 B^+ 树索引的高度最大值为 $\log_{\lceil \frac{m}{2} \rceil}\frac{n+1}{2}+1$，因此其检索的时间复杂度为 $O(\theta m_k \log_{\lceil \frac{m}{2} \rceil}\frac{n+1}{2})$。然而，在实际的检索中，真实的检索时间一定小于 $\theta m_k \log_{\lceil \frac{m}{2} \rceil}\frac{n+1}{2}$，这是因为：1)本方案由于检索条件的限制，许多包含关键字的子树或者叶子节点都没有被访问；2)被访问到的叶子节点中，许多节点都有共同的祖先节点及相同的访问路径，不需要每得到一个结果后就从根开始重新检索。

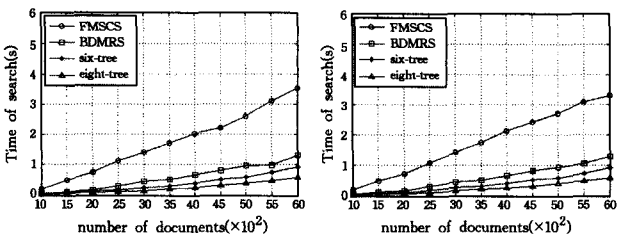
6 实验结果

本文选择英文文档作为测试数据，通过全文检索工具 Lucene[16] 对这些文档进行关键字提取，并过滤掉一些停用词

（例如 a，the 等），从而形成关键字集合，即词典。实验环境：操作系统为 Windows 7，64 位。CPU 为 Intel(R)Core(TM) i5(2.80GHz)，内存为 4GB。

检索执行效率是每一个应用系统应该首先考虑的因素，该实验分析索引的检索执行时间以及处理结果排序的时间开销。

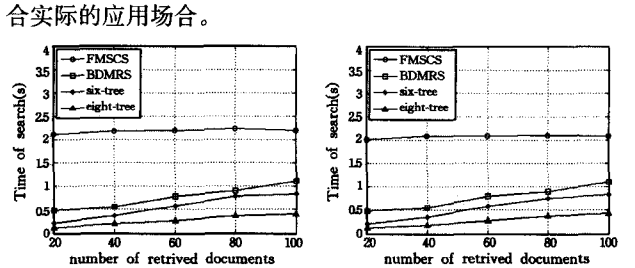
实验 1　词典大小 $m_k=4000$，设定用户返回的文件数 $k=100$。实验通过输入 10 个关键字，对 6 阶和 8 阶的 B⁺ 树的检索执行效率进行测试，并与文献[11,17]的方案进行比较。如图 3 所示，随着文件数的增加，检索时间呈递增趋势，这是因为若文件数增多，包含待检索关键字的文件就可能会增多，因而时间逐渐递增；与 FMSCS 方案[17]相比，BDMRS 和本文方案在检索速度上有显著的优势，原因是 FMSCS 方案是针对文件集中每个文件建立索引向量，却没有建立合适的索引结构，以至于在进行检索时需要遍历文件集中所有的文件，达到接近线性的搜索时间。而相比于 BDMRS 方案，本文方案的总体检索时间减少，并且阶数越大，时间越少。这是由于当阶数变大，B⁺ 树的高度会降低，检索的时间就会减少，与 5.2 节分析相符。



（a）文件集 1　　（b）文件集 2

图 3　实验 1 的结果

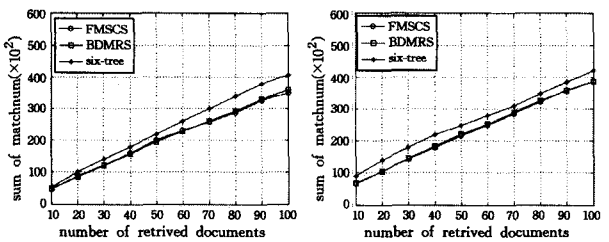
实验 2　文件数固定为 5000，词典大小 $m_k=4000$。实验输入 10 个关键字，如图 4 所示，随着返回的文件数量的增加，FMSCS 方案的检索时间基本不变，原因是无论需要返回几个文件，检索都需要遍历整个文件集，以达到准确检索的目的。而 BDMRS 和本文方案的检索时间都逐渐增加，这是因为若要返回的文件数量增多，需要访问的文件数就会增加，所以检索的执行时间是递增的；对于 B⁺ 树索引本身来说，当 B⁺ 树的叉数从 6 变为 8 时，总体的检索时间相对减少。与 BDMRS 方案相比较，本文利用 B⁺ 树降低了索引树的高度，减少了树中的节点数，从而降低了索引的空间复杂度，提高了检索的执行效率。虽然 B⁺ 树的高度越低，检索速度越快，但 m 也不是越大越好，假如 $m=n$，n 是文件集大小，那么 B⁺ 树的高度为 1，此时将退化为线性检索。按照假设，内存不可能容纳整个索引，需将索引存储在外存。在实际应用中，索引结点中存储两个 m_k 长度单位的向量，假设每次从外存中取一个节点，I/O 接口可容纳的单位数为 x，则需满足 $2m_km<x$。本文选取 6 和 8 是为了方便对比阶数不同的检索效果，m 的选取应符合实际的应用场合。



（a）文件集 1　　（b）文件集 2

图 4　实验 2 的结果

实验 3　对于用户来说，检索的目的不仅是希望快速地得到检索结果，还需要返回的检索结果能够符合自己的要求。用户需要自己判断结果是否准确与满足需求，但是根据用户输入的检索关键字，可以认为返回的结果文件中包含用户输入的关键字数越多，就越符合用户的需求。本文利用关键字匹配度 $matchnum$ 来表示每篇文章与用户输入的检索关键字的匹配个数，并将所有返回给用户的结果文件与关键字的匹配度之和 $sum=\sum_{i=1}^{k} matchnum_i$ 作为衡量检索结果的标准。由于返回结果按匹配度降序排列，因此 sum 越大，表示本次结果文件与关键字的相关度越大。该实验测试文件数为 3000，词典规模为 4000，如图 5 所示，相比于 FMSCS（设定用户对检索关键字的偏好相同）和 BDMRS 方案，本文方案的匹配度更高，原因是 FMSCS 和 BDMRS 方案以相关性分数作为检索结果的标准，不考虑实际关键词匹配的情况。本文的方案同时以匹配度和分数作为标准，在匹配度高的情况下选择分数高的文件，所以结果更贴近用户检索的需求。并且，本文方案的匹配度情况与 B⁺ 树索引的叉数无关。



（a）文件集 1　　（b）文件集 2

图 5　实验 3 的结果

实验 4　由于各方法都要对文件数据进行加密，因此对于密文数据的空间代价没有大的差别，重点体现在索引的空间代价及密钥两方面。本实验对比分析不同方法的密文索引的空间代价，设定文件数 $n=1000$，根据词典规模的变化进行实验。如表 2 所列，与 BDMRS 方案相比，本文的方案降低了索引的空间存储代价，这是因为对于相同的叶子节点数，B⁺ 树的总的节点数比二叉树少，减少了存储节点的空间代价。由于 FMSCS 方案只对文件构建索引向量，没有索引树上内部节点的向量，因此空间代价相对小一些。对于索引密钥，本文方案中用户只需要存储一个随机密钥，而 FMSCS 和 BDMRS 方案需要存储密钥矩阵，并随着词典规模变大而产生巨大的存储代价。

表 2　索引的空间代价

词典规模	FMSCS (MB)	6 叉树 (MB)	8 叉树 (MB)	BDMRS (MB)
1000	15.3	19.6	18.3	73
2000	30.5	39.2	36.6	146
3000	45.8	58.8	54.9	220
4000	61	78.4	73.2	293
5000	76.5	98	91.5	367

结束语　本文针对密文检索问题，提出了一种基于 B⁺ 树的密文排序检索方案，通过实验取得了较为理想的结果。实验结果表明，本文的方案在一定程度上能够提高多关键字密文排序检索的速度和结果的准确性。该方案利用 B⁺ 树比二叉树高度低的优势构建索引，降低了索引的空间复杂度，提高了检索的效率。对于检索的结果，该方案将相关性分数和关键词匹配度相结合并作为排序的标准，使得检索结果中不会

因为某个关键字的分数过高而影响排序结果，提高了检索的准确性。未来将着重于采用不同的加密方法对索引进行加密，使索引具有更高的安全性。

参 考 文 献

[1] FENG Chao-sheng, QIN Zhi-guang, YUAN Ding. Techniques of Secure Storage for Cloud Data[J]. Chinese Journal of Computers, 2015, 38(1): 150-163. (in Chinese)
冯朝胜，秦志光，袁丁. 数据安全存储技术[J]. 计算机学报，2015，38(1)：150-163.

[2] SHEN Zhi-rong, XUE Wei, SHU Ji-wu. Surveyon the Research and Development of Searchable Encryption Schemes[J]. Journal of Software, 2014, 25(4): 880-895. (in Chinese)
沈志荣，薛巍，舒继武. 可搜索加密机制研究与进展[J]. 软件学报，2014，25(4)：880-895.

[3] SONG D X, WAGNER D, PERRING A. Practical Techniques for Searches on Encrypted Data[C]// IEEE Symposium on Security and Privacy, 2000: 44-55.

[4] CURTMOLA R, GARAY J, KAMARA S, et al. Searchable symmetric encryption: Improved definitions and efficient constructions[C]// Proc of the 13th ACM Conference on Computer and Communications Security. New York: ACM, 2006: 79-88.

[5] CAO Ning, WANG Cong, LI Ming, et al. Privacy-Preserving Multi-Keyword Ranked Search over Encrypted Cloud Data[C]// Proc of INFOCOM. Shanghai: IEEE. 2011: 829-837.

[6] WONG W K, CHEUNG W L, KAO B, et al. Secure kNN computation on encrypted databases[C]// Proc of SIGMOD. New York: ACM, 2009: 139-152.

[7] CHENG Fang-quan, PENG Zhi-yong, SONG Wei, et al. An Efficient Privacy-Preserving Rank Query over Encrypted Data in Cloud Computing[J]. Chinese Journal of Computers, 2012, 35(11): 2215-2227. (in Chinese)
程芳权，彭智勇，宋伟，等. 云环境下一种隐私保护的高效密文排序查询方法[J]. 计算机学报，2012，35(11)：2215-2227.

[8] ZHANG Wei, XIAO Sheng, LIN Ya-ping, et al. Secure Ranked Multi-keyword Search for Multiple Data Owners in Cloud Computing[C]// Proc of Dependable Systems and Networks (DSN). Atlanta: IEEE, 2014: 276-286.

[9] IBRAHIM A, JIN H, YASSIN A, et al. Secure Rank-ordered Search of Multi-keyword Trapdoor over Encrypted Cloud Data [C]// Proc of Services Computing Conference (APSCC). Guilin: IEEE, 2012: 263-270.

[10] KAMARA S, PAPAMANTHOU C. Parallel and Dynamic Searchable Symmetric Encryption[M]// Financial Cryptography and Data Security. Berlin: Springer Berlin Heidelberg, 2013: 258-274.

[11] XIA Zhi-hua, WANG Xin-hui, SUN Xing-ming, et al. A Secure and Dynamic Multi-keyword Ranked Search Scheme over Encrypted Cloud Data[J]. IEEE Transactions on Parallel and Distributed Systems, 2016, 7(2): 340-352.

[12] WANG Cong, CAO Ning, REN Kui, et al. Enabling Secure and Efficient Ranked Keyword Search over Outsourced Cloud Data [J]. IEEE Transactions on Parallel and Distributed Systems, 2012, 23(8): 1467-1479.

[13] SUN Wen-hai, WANG Bing, CAO Ning, et al. Verifiable Privacy-Preserving Multi-Keyword Text Search in the Cloud Supporting Similarity-Based Ranking[J]. IEEE Transactions on Parallel and Distributed Systems, 2013, 25(11): 3025-3035.

[14] ZHANG Wei, LIN Ya-ping, XIAO Sheng, et al. Privacy Preserving Ranked Multi-Keyword Search for Multiple Data Owners in Cloud Computing[J]. IEEE Transactions on Computers, 2016, 65(5): 1566-1577.

[15] ZERR S, OLMEDILLA D, NEJDL W, et al. Zerber^{+R}: Top-k retrieval from a confidential index[C]// Proc of EDBT. New York: ACM, 2009: 439-449.

[16] Lucene3. 5[OL]. http://jakarta. apache. org/lucene.

[17] LI Hong-wei, YANG Yi, TOM L, et al. Enabling Fine-grained Multi-keyword Search Supporting Classified Sub-dictionaries over Encrypted Cloud Data[J]. IEEE Transactions on Dependable and Secure Computing, 2016, 13(3): 312-325.

(上接第 148 页)

[8] REZAEI F, HEMPEL M, SHRESTHA P L, et al. Achieving robustness and capacity gains in covert timing channels[C]// 2014 IEEE International Conference on Communications (ICC). IEEE, 2014: 969-974.

[9] BERK V, GIANI A, CYBENKO G, et al. Detection of covert channel encoding in network packet delays[R]. Department of Computer Science, Dartmouth College, 2005.

[10] GIANVECCHIO S, WANG H. Detecting covert timing channels: an entropy-based approach[C]// Proceedings of the 14th ACM Conference on Computer and Communications Security. ACM, 2007: 307-316.

[11] SHRESTHA P, HEMPEL M, REZAEI F, et al. A Support Vector Machine-based Framework for Detection of Covert Timing Channels[J]. IEEE Transactions on Dependable and Secure Computing, 2016, 13(2): 274-283

[12] DARWISH O, AL-FUQAHA A, ANAN M, et al. The role of hierarchical entropy analysis in the detection and time-scale determination of covert timing channels[C]// 2015 International Conference onWireless Communications and Mobile Computing (IWCMC). IEEE, 2015: 153-159.

[13] LIU Y, GHOSAL D, ARMKNECHT F, et al. Hide and seek in time—robust covert timing channels[M]// Computer Security-ESORICS 2009. Springer Berlin Heidelberg, 2009: 120-135.

[14] HOUMANSADR A, BORISOV N. CoCo: coding-based covert timing channels for network flows[M]// Information Hiding. Springer Berlin Heidelberg, 2011: 314-328.

[15] LIU Y, GHOSAL D, ARMKNECHT F, et al. Robust and undetectable steganographic timing channels for iid traffic[M]// Information Hiding. Springer Berlin Heidelberg, 2010: 193-207.

[16] GIANVECCHIO S, WANG H, WIJESEKERA D, et al. Model-based covert timing channels: Automated modeling and evasion [M]// Recent Advances in Intrusion Detection. Springer Berlin Heidelberg, 2008: 211-230.

[17] PAXSON V, FLOYD S. Wide area traffic: the failure of Poisson modeling[J]. IEEE/ACM Transaction on Networking (ToN), 1995, 3(3): 226-244.

[18] RICHARDSON A M. Nonparametric Statistics: A Step-by-Step Approach[J]. International Statistical Review, 2015, 83(1): 163-164.