基于 Simhash 的压缩文档相似性检索研究

张斌[1,2,3]，邹学强[3,4]，刘庆云[1,2]，张鹏[1,2]，杨嵘[1,2]*，陈少鹏[5]

(1. 中国科学院信息工程研究所，北京 100093； 2. 信息内容安全技术国家工程实验室，北京 100093；

3. 中国科学院大学， 北京 100049;4. 国家计算机网络应急技术处理协调中心,，北京 100029

5. 北京英孚泰克信息技术股份有限公司)

摘　要：随着云文档存储服务平台的快速发展和普及，大量文档存储在云平台上，为用户提供便利的服务。为了节约存储空间，降低传输开销，这些文档在云平台上以压缩形式存储。然而，这样的存储方式为文件的检索服务带来了困难，为了检索这些文档，需要对文档进行解压缩后才能根据其关键字构建索引，耗费了大量的时间并恶化服务体验。因此，本文提出了一种基于压缩文档 SIMHASH 的快速相似性文档检索框架，通过特征向量降维降低了相似度的计算复杂度，基于文档的检索减少了对于关键字选取的依赖，以达到无需对文档进行全部解压缩即可对其进行快速索引的目的。经过试验验证，本框架相对于直接解压缩然后构建索引的方法，时间开销减少了将近 44.26%，检索效率大大提升。

关键词：simhash，压缩算法，相似性检索，云平台

中图分类号：TP302　　**文献标识码**：A　　**文章编号**：

Smart Similarity Search Based on Simhash over Compressed Data in Cloud Computing

Bin Zhang[1,2,3], Xueqiang Zou[3,4] , Qingyun Liu[1,2], Peng Zhang[1,2], Rong Yang[1,2], Shaopeng Chen[5]

(1. Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China;

2. National Engineering Laboratory for Information Security Technologies, Beijing 100093, China;

3. University of Chinese Academy of Sciences, Beijing 100049, China;

4. National Computer Network Emergency Response Technical Team/Coordination of China, Beijing 100029

5. Information Technologies Co., (Beijing) Ltd)

Abstract: With the rapid development and popularization of cloud platform document storage service, a large number of documents stored in the cloud platform to provide users with convenient services. In order to save storage space and reduce transmission spending, these documents on the cloud platform stored in compressed form. However, the form that files are stored as compression archives makes it difficult, to retrieve these documents, we need to build an index of the documents based on their keywords after decompression, which spends a lot of time and deterioration of the service experience. Therefore, this paper presents a smart similarity search framework based on simhash over compressed data, the dimensionality reduction reduces the computational complexity of similarity by feature vectors, and also retrieval based on document reduces the dependence on selected keywords. You can retrieve the documents you need as fast as you can without fully decompressing these archives. After the test verification, compared with constructing the index after decompressing the archives, the time this framework costs have reduced by nearly 44.26 percent, so retrieval efficiency is greatly improved.

Key words: simhash, compression algorithm, similarity search, cloud platform

*通讯作者 杨嵘(yangrong@iie.ac.cn)

1 引言

最近几年，云平台已经开始广泛商用，而且由于它减轻了数据存储和数据管理的极大的负担，云平台也将极大的发展壮大。所以个人的用户数据，从邮件到个人健康记录正在越来越多的放到公开的云服务上面，比如亚马逊的云服务[1]，微软的Azure[2]，苹果的 iCloud[3]，谷歌的 APPEngine[4]，阿里的云平台[5]、百度的云存储[6]等等。与此同时，个人对于云平台的需求也越来越强烈，为了传输的便利性，有些上传到云服务器上的命令不支持对于目录的传输，用户因此需要进行打包后上传，而打包就需要进行压缩处理。同样的，用户为了传输的快捷，会去压缩传输内容的空间，而且平台提供的存储容量限制了用户传输内容的大小，压缩传输文件大小势在必行。平台提供商为了数据的存储空间利用的最大化，同样都推荐用户对数据压缩。用户在看到网上的压缩文档之后，或者找到自己的压缩文档之后，想要求推荐跟该文档相似的文档，就需要基于压缩文档进行相似性匹配。然而数据压缩对于现有的相似性文档搜索提出了严峻的考验，因为现有的相似性文档搜索大都基于非压缩的基础上进行的算法研究。如果本文直接在服务器上面，解压缩再计算相似度进行匹配，显然是不现实的，因为大量的计算资源的浪费，在服务器上可行性太低。

实际上，已经有很多论文提出了可用搜索的框架[7]，但大都是加密搜索相关的，针对压缩文档进行的相似性搜索很少。压缩文档常见的是 gzip、tar.gz、rar、7z 格式等等，大部分都是用了 deflate 压缩算法[8]、LZ77[9]变种 LZW[10], LZ78[11]等算法，本文选取一种 gzip 格式作为压缩格式说明，并进行优化，其它几种压缩格式都可用相同的方法进行改进。而文档相似性搜索有的基于关键字和基于文档的。通过关键字去进行匹配搜索需要先进行解压缩，然后从文档中提取关键字，然后再基于关键字去进行匹配搜索已经构建好的索引。但是这样的效率往往比较低下，而且效果往往依赖于提取的关键字。关键字提取容易不准确，不同文化背景的人选取的关键字很大可能不同，从而导致不同的搜索结果，有时候结果往往不会令人满意。第二种方法是，本文基于整个文档作为请求，然后去通过文档来检索相似性的文档，这样子就不用依赖于关键字的好坏来决定搜索相似需求的文档的结果。但是该方法也需要解压缩文档，传统的相似性搜索算法通常都比较复杂而且极其消耗计算资源。

基于文档搜索的相似性搜索问题由一系列文档特性、特征串提取、请求查询、相似性衡量等问题所构成。同图像提取特征类似[12-13]，文本中，在信息检索领域，大多数有代表性的搜索框架中普遍应用的是向量空间模型(VSM, vector space model)[14]，通常空间向量模型用"TF × IDF"来作为特征权重用来描绘文本的特征。词频 (term frequency, TF) 指的是某一个给定的词语在该文件中出现的次数。这个数字通常会被归一化（分子一般小于分母，区别于 IDF），以防止它偏向长的文件。（同一个词语在长文件里可能会比短文件有更高的词频，而不管该词语重要与否。）逆向文件频率 (inverse document frequency, IDF) 是一个词语普遍重要性的度量。某一特定词语的 IDF，可以由总文件数目除以包含该词语之文件的数目，再将得到的商取对数得到。本文对每个文档做 TF-IDF 之后形成每个文档特征串，然后就可以用余弦定理在两两之间比较文档计算它们的相似度。但是该方法因为向量多且维度高需要消耗大量的计算资源，很耗时间，以及需要大量的存储空间。所以该方法并不高效，就需要用 simhash[15]来降维。

然而 TF-IDF 不能直接应用在压缩文件上，都是必须对其解压缩，所以本文针对压缩算法进行了改进，加快了提取特征串的提取速度。Gzip 压缩算法分为两步，第一步是 LZ77，第二步是霍夫曼编码。本文首先进行霍夫曼解码，然后针对 LZ77 提取特征串，对于这种半解压缩的文档，本文跳过 LZ77 的指针，对正常文本进行 TF-IDF 提取特征。然后对提取的特征串进行 Simhash 降维。Simhash 是局部敏感哈希，可以用来判断文件的相似性，计算复杂度比余弦相似度计算方法代价小很多，只需要计算两两之间的海明距离即可，接下来，本文在服务器上根据压缩文档提取的一维向量特征进行索引的构建。当用户需要根据压缩文档进行搜索的时候，按照同样的方法进行提取特征串，simhash 降维，然后在服务器上进行检索，本文选取 top k 个相似的文档作为返回结果。这样本文就实现了针对压缩文档的相似性文档搜索。

本篇论文剩余的构成如下：第二节介绍相关工作，主要是针对压缩搜索领域相关的论文以及搜索框架的相关论文包括 gzip 压缩算法、局部敏感哈希

*通讯作者 杨嵘(yangrong@iie.ac.cn)

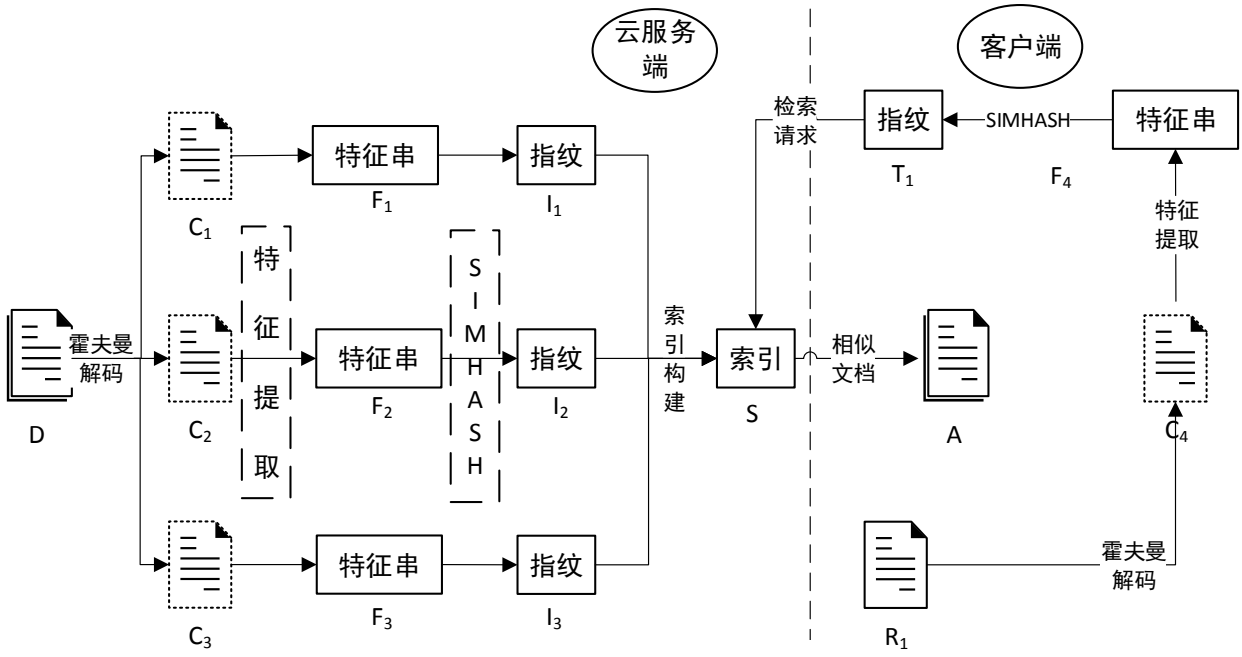

图 1 相似性搜索框架

以及搜索框架概要做了简介。第三节介绍针对压缩文档本文做出的改进以及跟现有的搜索框架结合起来的试验方法，详细阐述实验框架的整个过程原理。第四节介绍根据数据集所作出的实验结果以及效果验证。第五节总结结论。

2 相关工作

2.1 GZIP 压缩算法

Gzip[16]广泛应用于文档压缩，最常见的结尾是 gz，或者是 tar.gz。gzip 基于 DEFLATE 算法，而 DEFLATE 算法应用了两种压缩算法结合在一起：LZ77 压缩算法，然后下一步将输出做 huffman 压缩，分为自适应性 huffman 压缩和静态 huffman 压缩。本文稍微详细阐述如下两个压缩步骤：

LZ77 压缩算法：LZ77 压缩算法的最基本思想是如果前面已经出现过的字符在后面同样的出现过，本文就用指向之前重复内容(内容将会出现在一个 32k 的滑动窗口中)的指针来代替重复出现的内容，这样重复出现的大块内容就变为简短的指针，减小了文档的大小。本文用(距离，长度)对来标志一个指针，距离取值范围是 1-32768(32k)，表示之前重复的字符串出现距离现在重复内容的距离，长度取值范围是(3-258)，表示重复的长度。举个例子，”abcdefabcd”压缩之后变为”abcdef(6,4)”，其中(6,4)就是指针，表示在指针之前 6 个字节处，重复了长度为 4 的内容。

霍夫曼编码(huffman)[17]：霍夫曼编码完全依据字符出现概率来构造异字头的平均长度最短的码字。本文会提前进行预处理将霍夫曼进行提前解压缩处理。

2.2 局部敏感哈希(LSH)

本文中所用到的一种哈希算法是局部敏感哈希[18]，它的一个特点是将高维的数据降维到低维度，这样就会减少大量的数据计算复杂度。局部敏感哈希广泛应用在信息检索领域中的快速相似性搜索方面。将高维的特征串信息降维到一维二进制码，形成特征指纹，然后比较指纹之间的相似性，这样本文就能检测到文档之间的相似性。本文将应用局部敏感哈希中的 simhash，用 simhash 做出的指纹，本文可以通过衡量其指纹的海明距来衡量其文档的相似性。对于文档 A，B 来说：

$$\Pr[h(A) = h(B)] = \mathrm{sim}(A, B)$$

其中 $\mathrm{sim}(A,B) \in [0, 1]$是一种特定的相似矩阵。Simhash 有两个特点：(1)文档的指纹是文档特征的 simhash 值。(2)相似的文档具有相似的指纹。

最后本文将指纹构建成字典树，进行剪枝，提高大量文件计算海明距的效率。

相似性检索框架

基于压缩的检索框架大部分都是基于关键字的高效多模式匹配算法[19]，文中应用的是 AC 算法[20]的改进算法，但是对于基于文档的搜索却差别甚大，而且没有相似性的衡量。而大量的基于对称加密的搜索算法，同样存在基于文档进行相似性搜索的文章，例如 Zhangjie Fu[20]等人在对称搜索上做的基于 simhash 的相似性文档搜索，但实质上是基于明文进行的索引构建，基于压缩文档并不适用，需要对其解压缩处理，这样会导致效率低下。

3 搜索框架

3.1 系统框架总览

首先看图一，本文的搜索框架由两部分构成：左边一部分是服务器端服，对于服务器拥有大量的用户文档集 D，首先经过霍夫曼解码后得到 C，再次经过特征串提取生成指纹 F，最后根据指纹构建成索引。如果用户发来检索请求，则根据用户的指纹信息在索引上进行检索，最终结果将返回 top k 个相似文档，并最终发送给用户，完成检索请求；另一部分是用户部分，负责发起检索请求，起始是 gzip 压缩文档 R，同样的进行霍夫曼解码，形成文档 C，然后做提取特征串，对其进行 simhash 生成指纹。然后用户把指纹提交到服务器，服务器根据索引进行检索，如果需要用户可以向服务器提交参数 k，用来返回最相似的 k 个文档。用户将服务器返回的结果取回，然后解压缩即可看到压缩内容。

3.2 概念解释

为了便于算法框架的理解，本文解释文中以及图一中用到的变量概念：

- D – 压缩文档集，在服务器端，本文定义 $D = \{D_1, \dots, D_m\}$。
- C – 对文档进行霍夫曼解码之后的文档，此时文档是 LZ77 编码之后的形式。$\mathrm{C} = \{C_1, \dots, C_m\}$。
- F – 从 LZ77 编码文件中提取的特征串。$\mathrm{F} = \{f_1, \dots, f_m\}$。
- I – 对特征进行 simhash 降维处理处理后得到的一维特征。同理，$\mathrm{I} = \{I_1, \dots, I_m\}$。
- T – 请求要检索的特征串，由请求检索的文档经过霍夫曼解码、提取特征串、simhash 降维处理后得到。
- R – 要进行相似性文档检索的压缩文档。
- π – 哈希函数，$\{0,1\}^k \times \{0,1\}^* \to \{0,1\}^h$
- A – 发出请求过后，经过服务器检索得到的最相似的文档集合。

3.3 基本搜索框架

本基于压缩文档的相似性搜索框架的主要算法如下：

1) 霍夫曼解码：去除 gzip 头部，然后根据文档动态的构建霍夫曼树，并且根据树的信息还原文档，进行解码。
2) 索引构建：在文档集 D 的基础上，本文霍夫曼解码，提取特征串，然后进行 simhash 降维，然后将二进制形式的特征串构建成字典树的形式，并与文件关联起来。这样就形成了索引。
3) 请求文档指纹构建：跟索引构建类似，同样的霍夫曼解码，提取特征串，形成一维特征指纹。最终将指纹跟可选参数 k 发送到服务器端。
4) 搜索：通过请求的指纹信息，本文在已经构建好的索引上面进行匹配，两两之间检测相似度，并得到一个分值，最后选取分值最高的 k 个文档返回给用户。搜索可以利用字典树的特点进行剪枝操作。

3.4 主要步骤的详细描述

3.4.1 霍夫曼解码

霍夫曼分为自适应性霍夫曼解码跟静态解码，静态解码只需要根据 gzip 默认的字典树进行还原即可，将每个对应的二进制还原成树中的字符。而自适应性霍夫曼需要逐步构造霍夫曼树，并在构造的过程中还原二进制串的字符。最终本文得到 LZ77 编码的文档。

3.4.2 特征提取

因为关键字是文本中代表文档特征的有利工具，所以本文将会提取文档关键字来作为其主要特征。首先利用特征提取中最经典的”TF-IDF”方式来作为特征提取算法。”TF”代表单词在文本中的重要程度。而”IDF”代表单词在不同文本中的区分度。LZ77 编码的文档中会将 1-32768(32k)范围内的重复的字符串用指针代替。代表文档的特征本文主要关心 IDF，对于因为指针而减少统计的 TF 值并不影响对于整个文档的特征的表示，而且本文可以通

过增加特征词汇数量来减少词频的影响。所以本文可以直接统计非指针部分，这样本文就减少了大量重复的工作。本文用如下公式计算每个 LZ77 文档 C_i 中的每个单词的 TF×IDF，然后对其排序，选取前 z 个单词作为特征。

$$S_{wD_i} = \ln(1 + f_{d,w}) \times \ln(1 + \frac{m}{f_w})$$

其中$f_{d,w}$表示文中单词的词频，f_w表示包含单词w的文档的数量，而 m 代表整个文档的总数量。所以最终本文提取的特征为$F_i = \{f_{i_1}, \dots, f_{i_z}\}$，其中$f_{i_j}$表示提取的单词。

3.4.3 simhash

通过上一步本文提取了文档 D_i 的单词特征 F_i。服务器会用 simhash 算法来进一步取得 F_i 的指纹。徒儿表示了 simhash 的过程，其步骤如下：

(1) 初始化 h 维的向量 V，每一个位置为空，即 0；

(2) 对于 Fi 中的单词特征f_{i_j}，本文用 π 哈希函数将它哈希成一个 h 位的签名。如果这 h 位的签名中是一，那么向量 V 对应位置加一，否则建议。

(3) 最终，本文将产生一个 h 位的指纹 I_i，如果指纹位大于 0，那么设为一，否则设为 0。

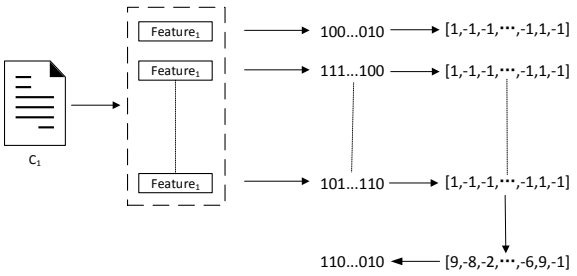

图 2 simhash 处理过程

3.4.4 索引构建

本文把 3.4.3 小节中的指纹信息做完之后就可以开始构建本文的索引树了。如图三所示，索引树是以字典树[21-22]为基础的，相同的前缀的指纹会有一部分相同的路径。图中 R 代表根节点，是搜索发起的开始位置，其中"0","1"代表指纹信息。最终的 D_i 表示字典树到达了终点之后所关联的文档，因为长度一样，所以是叶结点。叶结点关联了文档，当本文搜索到需要的叶结点，可以直接通过关联找到压缩的文档。同样的，因为字典树的优势，本文在两两之间搜索的时候就可以通过剪枝来减少不必要的搜索比较，节省了大量的服务器计算资源。

3.4.5 搜索

在指纹生成之后，客户发来通过文档 D_i 提取的请求 T_i，然后本文就可以通过比较索引与指纹的信息得出相似性数值。文章之间的相似度可以通过海明距来衡量：

$$\mathrm{sim}(D_i, D_q) = H(I_i, T)$$

其中 $H(S_1, S_2)$是两个相同长度的二进制串 S_1 和 S_2 之间的海明距。海明距的计算是通过计算两个串之间的不同的位数来决定的。比如，H(1011101, 1001001) = 2。海明距越小表示两篇文档的相似度越高。

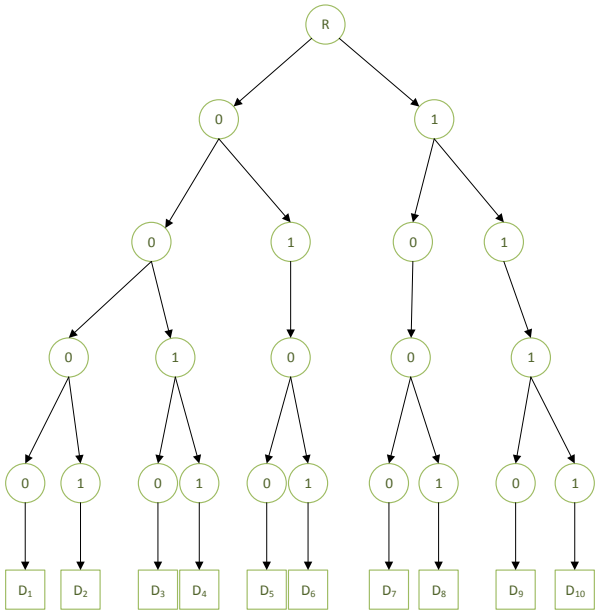

图 3 基于字典树的索引树

根据索引，本文采用深度优先搜索的思路。从树的根节点开始自上往下搜索。每次选择海明距离小的子节点。本文选取现有的已经选取的第 k 个节点的海明距作为本文的海明距上限 M_k，并且在访问节点的时候保持更新，M_k 可以用一个优先队列或者最小堆来维护。如果到达的节点超过限制海明距离 M_k，就返回到父节点中，并继续深度优先搜索，否则无需返回父节点，继续深度优先搜索。如果一个前缀大于了 M_k，本文就不需要继续访问该节点的子节点了，这样本文就减少了好多不必要的访问，节省了大量的时间。如图三所示，本文搜索的指纹是"1111"，其中 k 为 2，那么根据本文算法剪枝之后，本文只需要访问 D_9 和 D_{10} 即可。而其它节点本文就可以成功的避免掉。

4 实验与分析

为了衡量本文搜索框架的性能，本文测试机器的物理环境为 Intel(R) Xeon(R) CPU E5-4620 + DDR3 132GB Memory，测试机的软件环境为 RED HAT 6.4。数据集为在线采集的 10,000 个 gzip 压缩文件(因为网页大部分都是 gzip 压缩的形式，本文利用网页的 content-encoding 为 gzip 的网页进行的在线采集)。

4.1 霍夫曼解压缩

本文在 gzip 官网[23]上下载了 gzip 源码，并对 gzip 源代码进行了更改，去除了 LZ77 解码部分，这样本文只进行霍夫曼解码。后面本文会对比直接对解压缩后的文档提取指纹与对 LZ77 编码文档的提取指纹时间。

4.2 特征提取

特征单词提取需要经过如下步骤：(1)去除 LZ77 中的指针；(2)过滤停用词；(3)转换单词，将大写变为小写；(4)词根化，这里用的是 Porter Stemming 算法[24]；(5)最后本文计算每个单词的 TF-IDF 值，并选取 top10 的单词作为特征。

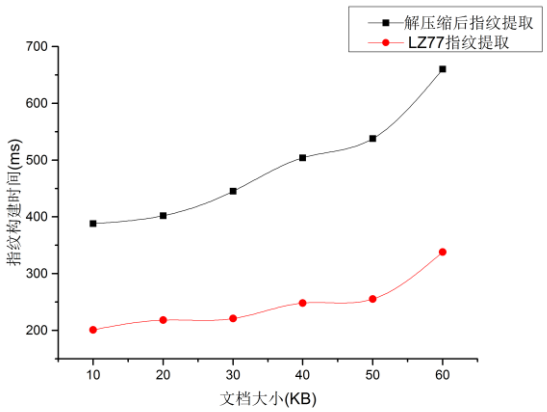

图 4 指纹提取时间对比

4.3 指纹提取

为了降维，本文需要进行 simhash 算法，首先用 π 函数，这里本文用 SHA-1 将单词哈希成一个 128 位的向量，最后根据算法形成一维的特征值指纹。如图四所示，图中横坐标是本文选取的不同文档，单位为文档的大小 KB，纵坐标是指纹提取的时间，以毫秒为单位。本文对比直接在 LZ77 编码上的文件跟解压缩之后的文档构建索引的时间比较发现，借助 LZ77 编码的优势，本文节省了将近一半的时间来生成指纹。

4.4 索引构建

本文将 4.3 小节中的一维特征向量构建成字典树的形式。在数据集为 10,000 的情况下，本文构建索引的时间为 34 分钟。相比较如果本文在解压缩的情况下，整个索引构建的过程需要的时间却长达 61 分钟。可见，整个索引的构建跟特征提取的时间息息相关。在整个索引构建的过程中，单词特征的提取最费时间，而 simhash 以及索引构建所占的时间相比却占了很少的一部分，几乎可以忽略不计。也就是说本文基于压缩文档的半解压缩，直接在 LZ77 上面进行指纹提取让本文索引构建的效率提高了 44.26%，几乎跟指纹提取时间成正比。

4.5 检索

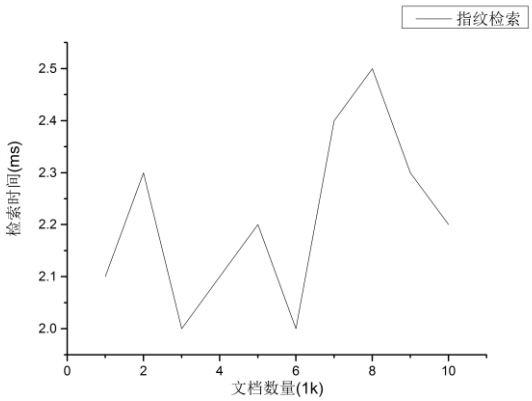

图 5 指纹检索时间

客户端需要检索一个压缩文档，将指纹提前到服务器，服务器对其进行海明距离的计算。图中显示了检索的时间，横坐标为文档的数量，以 1k 个文档为单位。纵坐标为检索的时间，单位为 ms，可以看到检索时间基本在 2ms 多一点，基本呈常数时间复杂度的搜索时间。所以短至 2ms 检索并不会影响用户的体验。

5 结论

在本论文中，本文提出了一种基于压缩文档的相似性检索框架。传统的快速检索框架并不适合本

文的压缩场景，所以本文针对压缩算法提出了基于压缩文档的改进。进行霍夫曼解压缩之后本文直接在 LZ77 上面进行了特征的提取。而特征提取是最耗时间的过程，因此整个索引构建的时间也因为算法的改进而提升了 44.26%的时间。所以本文的算法极大的提升了服务器处理的效果。同样实验也有效的证明了本文算法的有效性。

参考文献

[1] Amazon Web Services, 2016, http://aws.amazon.com

[2] Microsoft Azure, 2016, http://www.windowsazure.com.

[3] Apple iCloud, 2016, https://www.icloud.com/

[4] Google AppEngine, 2016, https://appengine.google. com/

[5]Aliyun,2016, https://www.aliyun.com/

[6]Yunbaidu,2016, http://yun.baidu.com/

[7] Fu, Zhangjie, et al. "Privacy-Preserving Smart Similarity Search Based on Simhash over Encrypted Data in Cloud Computing." Journal of Internet Technology 16.3 (2015): 453-460.

[8] L. Peter Deutsch (May 1996). DEFLATE Compressed Data Format Specification version 1.3. IETF. p. 1. sec. Abstract. RFC 1951. Retrieved 2014-04-23.

[9] Ziv, Jacob; Lempel, Abraham (May 1977). "A Universal Algorithm for Sequential Data Compression". IEEE Transactions on Information Theory 23 (3): 337–343. doi:10.1109/TIT.1977.1055714. CiteSeerX: 10.1.1.118.8921.

[10] Welch, Terry (1984). "A Technique for High-Performance Data Compression" (PDF). Computer 17 (6): 8–19. doi:10.1109/MC.1984.1659158

[11] Ziv, Jacob; Lempel, Abraham (September 1978). "Compression of Individual Sequences via Variable-Rate Coding". IEEE Transactions on Information Theory 24 (5): 530–536. doi:10.1109/TIT.1978.1055934. CiteSeerX: 10.1.1.14.2892.

[12] Jian Li, Xiaolong Li, Bin Yang and Xingming Sun, Segmentation-Based Image Copy-Move Forgery Detection Scheme, IEEE Transactions on Information Forensics and Security, Vol.10, No.3, 2015, pp.507518.

[13] Hui Zhang, Q. M. Jonathan Wu, Thanh Minh Nguyen and Xingmin Sun, Synthetic Aperture Radar Image Segmentation by Modified Student's t-Mixture Model, IEEE Transaction on Geoscience and Remote Sensing, Vol.52, No.7, 2014, pp.4391-4403.

[14] Bin Gu and Victor S. Sheng, Feasibility and Finite Convergence Analysis for Accurate On-Line ν-Support Vector Learning, IEEE Transactions on Neural Networks and Learning Systems, Vol.24, No.8, 2013, pp.1304-1315.

[15] Moses S. Charikar, Similarity Estimation Techniques from Rounding Algorithms, Proc. of the Thiry-Fourth Annual ACM Symposium on Theory of Computing, Montreal, Canada, May, 2002, pp.380-388.

[16] RFC 1952 – GZIP file format specification version 4.3

[17] Huffman, D. (1952). "A Method for the Construction of Minimum-Redundancy Codes" (PDF). Proceedings of the IRE 40 (9): 1098–1101. doi:10.1109/JRPROC.1952.273898.

[18] Piotr Indyk and Rajeev Motwani, Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality, Proc. of the Thirtieth Annual ACM Symposium On Theory of Computing, Dallas, TX, May, 1998, pp.604-613.

[19] Bremler-Barr, Anat, and Yaron Koral. "Accelerating multipattern matching on compressed HTTP traffic." IEEE/ACM Transactions on Networking (TON) 20.3 (2012): 970-983.

[20] Piotr Indyk and Rajeev Motwani, Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality, Proc. of the Thirtieth Annual ACM Symposium On Theory of Computing, Dallas, TX, May, 1998, pp.604-613.

[21] Black, Paul E. (2009-11-16). "trie". Dictionary of Algorithms and Data Structures. National Institute of Standards and Technology. Archived from the original on 2010-05-19.

[22] Franklin Mark Liang (1983). Word Hy-phen-a-tion By Com-put-er (Doctor of Philosophy thesis). Stanford University. Archived from the original (PDF) on 2010-05-19. Retrieved 2010-03-28.

[23]gzip org, 2016, http://www.gzip.org/

[24] Martin F. Porter, An Algorithm for Suffix Stripping, Program, Vol.14, No.3, 1980, pp.130-137.