doi:10.11959/j.issn.1000-436x.2015226

基于压缩近邻的查重元数据去冗算法设计

姚文斌[1,2]，叶鹏迪[3]，李小勇[4]，常静坤[1,2]

(1. 北京邮电大学 智能通信软件与多媒体北京市重点实验室，北京 100876；
2. 北京邮电大学 计算机学院，北京 100876；
3. 中国铁道科学研究院 机车车辆研究所，北京 100081；
4. 北京邮电大学 可信分布式计算与服务教育部重点实验室，北京 100876)

摘 要：随着重复数据删除次数的增加，系统中用于存储指纹索引的清单文件等元数据信息会不断累积，导致不可忽视的存储资源开销。因此，如何在不影响重复数据删除率的基础上，对重复数据删除过程中产生的元数据信息进行压缩，从而减小查重索引，是进一步提高重复数据删除效率和存储资源利用率的重要因素。针对查重元数据中存在大量冗余数据，提出了一种基于压缩近邻的查重元数据去冗算法 Dedup^2。该算法先利用聚类算法将查重元数据分为若干类，然后利用压缩近邻算法消除查重元数据中相似度较高的数据以获得查重子集，并在该查重子集上利用文件相似性对数据对象进行重复数据删除操作。实验结果表明，Dedup^2 可以在保持近似的重复数据删除比的基础上，将查重索引大小压缩 50%以上。

关键词：重复数据删除；查重元数据；近邻压缩规则

中图分类号：TP391 **文献标识码**：A

Deduplication algorithm based on condensed nearest neighbor rule for deduplication metadata

YAO Wen-bin[1,2], YE Peng-di[3], LI Xiao-yong[4], CHANG Jing-kun[1,2]

(1. Beijing Key Laboratory of Intelligent Telecommunications Software and Multimedia, Beijing University of Posts and Telecommunications, Beijing 100876,China;
2. School of Computer Science, Beijing University of Posts and Telecommunications, Beijing 100876,China;
3. The Locomotive and Car Research Institute, China Academy of Railway Sciences, Beijing 100081,China;
4. Key Laboratory of Trustworthy Distributed Computing and Service of Ministry of Education, Beijing University of Posts and Telecommunications, Beijing 100876, China)

Abstract: Building effective deduplication index in the memory could reduce disk access times and enhance chunk fingerprint lookup speed, which was a big challenge for deduplication algorithms in massive data environments. As deduplication data set had many samples with high similarity, a deduplication algorithm based on condensed nearest neighbor rule, which was called Dedup^2, was proposed. Dedup^2 uses clustering algorithm to divide the original deduplication metadata into several categories. According to these categories, it employs condensed nearest neighbor rule to remove the highest similar data in the deduplication metadata. After that it can get the subset of deduplication metadata. Based on this subset, new data objects will be deduplicated based on the principle of data similarity. The results of experiments show that Dedup^2 can reduce the size of deduplication data set more than 50% effectively while maintain similar deduplication ratio.

Key words: deduplication; deduplication metadata; condensed nearest neighbor rule

收稿日期：2014-07-07；**修回日期**：2015-06-05

基金项目：国家自然科学基金资助项目（61370069）；国家高技术研究发展计划（“863”计划）基金资助项目（2012AA012600）；中央高校基本科研业务费专项基金资助项目（BUPT2011RCZJ16）

Foundation Items: The National Natural Science Foundation of China (61370069); The National High Technology Research and Development Program of China (863 Program) (2012AA012600); Fundamental Research Funds for the Central Universities (BUPT2011RCZJ16)

1 引言

随着信息量的爆炸式增长，数据占用空间及带宽越来越大，企业面临的快速备份和恢复的时间点越来越多，管理、保存、传输数据的成本及数据中心空间和电源的耗费也变得越来越昂贵。研究发现，应用系统所保存的数据，高达60%是冗余的，而且随着时间的推移会变得越来越严重，重复数据删除技术受到越来越多的关注。

常用的基于块的重复数据删除算法将数据切分成定长或可变长的数据块，并计算每个数据块的散列值作为数据块指纹，拥有相同指纹的数据块即被认为是重复的。这种基于块的重复数据删除算法对那些变化缓慢、尤其是修改较少的备份数据具有较好的效果。然而随着数据集的增大，数据块指纹等元数据信息会迅速超过内存容量，并且由于散列算法的天然随机性，很难对这些指纹实现有效的缓存，容易造成频繁访问磁盘、降低重复数据删除的性能。

为了解决数据块指纹检索过程中面临磁盘瓶颈问题，DDFS[1]、Sparse Indexing[2]提出通过利用备份数据流中的数据块局部性特征来构建内存中的查重索引，借此提高块指纹检索的命中率，减少磁盘操作。Extreme Binning[3]、Silo[4]和重复数据删除系统[5~7]通过比较数据对象之间的相似性，将与待去重数据对象较为相似的数据对象的块指纹数据读入内存来构建查重索引，在数据局部性特征较少的情况下，也能解决指纹检索面临磁盘瓶颈问题。然而，常用的相似数据检测算法如 shingle detection[8]、Bloom filter[9]都是利用较小的数据片段来代表原始数据对象以实现文件间的相似性检测，这些片段的长度与数据对象大小相关，在大文件较多的环境下，所产生的较长的数据片段会加重存储资源开销。

CDFS[10]基于定长的 traits[11]指纹来计算数据对象之间的相似性。Simdedup[12]基于 simhash[13]算法，用 simhash 指纹值来代表原始数据对象，通过比较 simhash 指纹值来计算数据对象之间的相似性，并基于相似数据对象的块指纹信息来构建查重缓存，减少磁盘读写次数，由于 simhash 指纹值的长度固定且极小，因此可以将大量数据对象的指纹索引保存在内存中作为相似索引，实现在保持较少的额外系统资源开销的基础上，提高重复数据删除效率。

然而，每个重复数据删除系统都需要额外的空间来存储删重过程中产生元数据信息。例如清单文件，其中保存着包含数据块指纹值及指向该数据块所在的磁盘存储位置的指针的数据块描述符。通过这些清单文件，顺序读取数据块描述符，同时加载并串联描述符所指向的数据块，便可以重构数据内容。这些数据块描述符和标准文件系统中的数据块指针的不同之处就是大小的不同，一般文件系统中的数据块指针都是 8 byte，而重复数据删除系统中的文件描述符一般至少是 20 byte。

现有的对重复数据删除技术的研究往往会忽略这些元数据信息，认为这些文件一般不会成为系统吞吐量的性能瓶颈。然而，随着数据集的增加，元数据也会随之增长，并会占用大量的存储空间。在重复数据删除技术的帮助下，存储数据所需的磁盘空间随时间缓慢增长，如 Zhu 等[1]通过应用重复数据删除技术，在每天备份的环境下，对数据达到了近 40:1 的压缩比，数据对存储空间的需求逐步趋缓。而清单数据等元数据信息却会随着去重次数的增加而稳步增长。假设一个数据块大小为 4 KB，那么 1 TB 的数据就至少需要 5 GB 的空间来存储这些清单文件。在备份系统环境下，假如每周执行一次全备份，且保留期为 52 周，那么对于 20 TB 的备份数据来说，就会产生至少 5 TB 的清单数据。因此，在海量数据环境下，保存这些元数据信息将会产生巨大的额外存储资源开销。

Simdedup 这类基于相似性的重复数据删除算法对每个已去重数据对象都会产生查重元数据，供以后的重复数据删除操作使用，由于数据对象之间存在较高的相似性，那么，所产生的清单文件等元数据也会包含大量的相似数据，增加了存储资源开销。此外，随着数据集持续增大，用于检测数据相似性的相似索引的数据量也会不断增加并最终导致索引过大以致于难以存放在内存中，影响相似数据检索的效率，进而影响重复数据删除效率。在重复数据删除过程中，使用更大的平均数据块固然能够减少数据块数量，降低清单文件的大小，然而，这种方法会导致重复数据删除率的下降[14~16]。随着人们对数据资源的愈发重视，全备份的时间间隔越来越短，且数据保存周期越来越长，清单文件等元数据信息的压缩已经成为重复数据删除系统中不容忽视的环节。

常用的元数据压缩技术中，Zero-Chunk 压缩技术[17,18]只能通过对内容全是 0 的数据块给予较短的编码，来间接对元数据进行压缩，无法处理其他类型数据块。基于重复序列的压缩技术[19~21]只能通过

合并重复序列来间接降低元数据大小，无法应对实际环境中，数据块本身存在冗余，但序列相对较乱的情况。基于统计的压缩技术[14,22]除了维护数据块使用率索引外，还需要维护额外的编码索引，该索引的存放位置是一个需要权衡的问题，如果存储在内存中，固然会提高访问速度，但却会限制编码的数量，如果存储在磁盘上，编码的数量不会有太多限制，但却会需要额外的 I/O 操作。因此，在不对重复数据删除比造成太大影响的前提下，消除查重元数据集中重复度较高的样本，降低元数据集的数量，使在海量数据环境下，依然维持较低的系统资源开销，是亟待解决的问题。

本文在 Simdedup 的基础上，基于压缩近邻[23]的思想，提出了查重元数据的去冗算法 Dedup^2，在维持重复数据删除比的同时，可以有效降低用于查重的数据量，使资源开销保持在一个较低的位置。

2　Dedup^2 设计思路

Simdedup 进行重复数据删除操作时，由于数据段之间存在较高的相似性，那么，所产生的元数据信息中也必然包含大量的相似数据。

因此，Simdedup 中元数据压缩面临的问题的描述如图 1 所示。图中的每个点都表示一个对应的数据段，2 个数据段距离的远近表示两者之间相似度的高低，那么按照两两之间的相似性，可以对数据段进行聚类，最后获得 k 个类，图 1 所示为 k=3 的情况。由于类中的数据段拥有较高的相似性，那么可以合并相似度高的数据段的清单文件，消除其中的重复数据块指纹描述符，以减少清单文件大小。

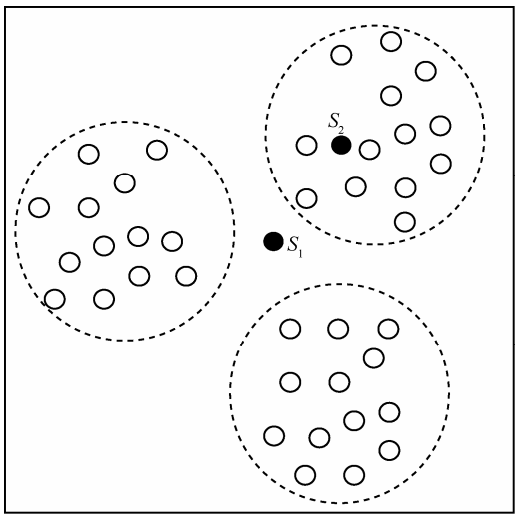


图 1　问题描述

然而，单纯通过聚类的方法无法处理相似索引 SFIndex，该索引数据量的增加依然会给系统带来较大的负担，因此需要针对各个类中数据对象的重要性，消除 SFIndex 中对重复数据删除率贡献相对较小的数据段的指纹。

对于一个新的数据段来说，基于 Simdedup 算法，其相似特征会出现 2 种情况，分别如图 1 中的黑点 S_1 和 S_2 所示。其中，黑点 S_1 表示该数据段在平面中的位置处于 3 个类边界的外部，其最相似数据段就是与其距离最近的分属于不同类别几个白点，这些白点都在类的边界部位；黑点 S_2 表示该数据段在平面中的位置处于类的内部，其最相似数据段就是与其距离最近的同一类中的几个白点，由于同类中数据对象相似度较高，所以选择类边界的数据段来构建查重缓存，也能提供较好的重复数据删除率。

因此，消除 Simdedup 元数据中处于类中心位置的数据段信息，尽可能保留类边界位置的数据段信息，便能够实现元数据的压缩，并缩小相似索引 SFIndex，使之能够完整存放于内存中，提高重复数据删除效率，同时维持较低的系统资源开销。

压缩近邻（CNN, condensed nearest neighbor rule）算法主要用于寻找样本的一致子集。对于一个集合 E 的一个子集 E'，如果利用最近邻算法（1-NN），E'中样本可完全正确地分类 E 中的样本，那么 E'就是集合 E 的一致子集。从集合 E 中创建一致子集 E'时，首先将所有少数类中的样本以及随机选取的一个多数类中的样本加入 E'中进行初始化。然后用 E'中的样本以最近邻算法（1-NN）对 E 中样本分类，将所有错分的样本加入到 E'中。

压缩近邻算法保留了多数类中边界附近的样本，同时去掉了多数类中远离边界的样本，使一致子集在保留最少量样本的条件下，仍能对原有全部样本用最近邻法正确分类，那么也就能够对待识别样本进行分类，并保持正常识别率。因此，其设计思想和 Simdedup 消除相似度较高的元数据，降低查重元数据大小后，仍需保持相近重复数据删除率的需求是等同的。

基于压缩近邻的思想，从相似度过高的元数据对重复数据删除率影响有限的角度出发，提出了查重元数据去冗算法 Dedup^2，首先对查重数据集进行聚类，然后利用压缩近邻算法获得查重子集，并基于该查重子集消除相似度较高的元数据，进而降低查重索引大小。消除相似度较高的元数据，在维持

重复数据删除率的同时，可以有效降低元数据的数量，进一步降低系统资源开销。

3 Dedup^2 系统架构

重复数据删除算法的系统整体结构如图2所示。Simdedup 中查重所需要的元数据信息数据被分成 2 个索引分别保存，分别是 simhash 指纹索引（SFIndex）和数据块指纹索引（CFIndex）。其中，SFIndex 保存所有的数据段的 simhash 指纹，而每个数据段所包含的所有数据块的指纹值将存储在磁盘的一个清单文件中，这些清单文件信息则保存在 CFIndex 里面。由于 Simdedup 对每个已去重的数据对象都会产生这 2 种元数据信息，并保存在查重索引中，因此，随着数据集持续增大，查重索引的数据量也会不断增加，最终必然导致查重索引无法有效存放在内存中。

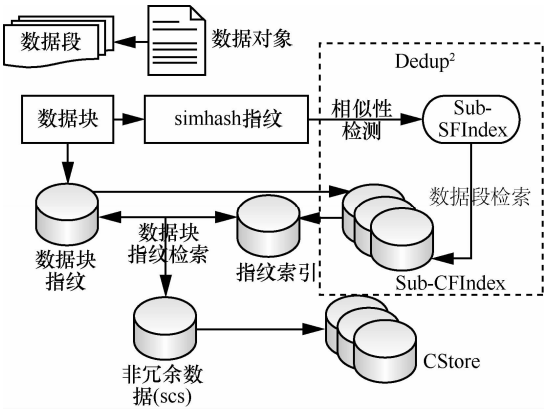


图2 系统结构

Dedup^2 通过离线处理的方式对查重的元数据信息进行周期性去冗，获得较小的查重子集。通过消除 SFIndex 中冗余度较高的元素，可以获得较小的 simhash 指纹子索引 Sub-SFIndex。同时，合并 CFIndex 中冗余度较高的清单文件，可以获得较小的清单文件子索引 Sub-CFIndex。

对一个数据对象作重复数据删除操作时，每个数据段都利用可变长分块算法进行切块，并利用 SHA-1 算法计算该数据段所包含数据块的指纹值。基于所得到的数据块，可以计算得到每个数据段的 simhash 指纹。若用于构建内存中数据块指纹索引的数据段个数设为 w，那么通过相似性检索，比较该指纹和 Sub-SFIndex 中的其他指纹，可以按照相似度的高低，获得与待去重数据段最相似的 w 个数据段。基于这些数据段，便可以通过 Sub-CFIndex 检索清单文件，获得各数据段对应的块指纹信息，进而将这些指纹值读到内存中，构建查重缓存。最后，比对新数据段的指纹值和查重缓存中的指纹值，就可以消除重复的数据块。

Dedup^2 能够消除查重元数据中的重复数据，大大降低查重索引的大小，从而使查重索引能更好地存放在内存中，减少磁盘访问，保证数据块指纹检索速度和较低的系统资源开销，同时，保持相似的重复数据删除比。

4 Dedup^2 算法描述

Dedup^2 基于压缩近邻的思想，以消除元数据中的冗余信息，获取精简的查重子集为目的，整个算法流程如图3所示，可以分为聚类阶段和去冗阶段2个阶段。该方法首先将查重数据集聚成 k 类$\{C_1, C_2, \cdots, C_k\}$，从每一类别样本中各随机选择一个样本以构成初始子集，然后按最近邻原则用该子集对剩余训练集分类，并将误分样本加入查重子集，遍历整个查重数据集后，便可获得所需查重子集。

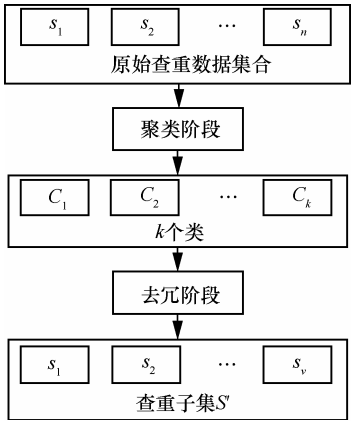


图3 Dedup^2 流程

基于该查重子集，消除元数据中对重复数据删除率影响有限的样本，进而降低元数据的大小，从而使 simhash 指纹索引能更好地存放在内存中，减少磁盘访问，保证数据块指纹检索速度和较低的系统资源开销，同时，保持相似的重复数据删除比。下面对 Dedup^2 算法的聚类阶段和去冗阶段分别作详细描述。

4.1 聚类阶段

由于在使用压缩近邻算法前需要对各个元数据进行归类，同时因为元数据的主要指标是 simhash 值，且无法通过 k-means 聚类算法对 simhash 值计算欧氏距离，所以需通过 k 中心点聚类算法进行归类。partitioning around medoids (PAM)算法是较常用的 k 中心点聚类算法，具有数据顽健性强、聚类结果与输

入顺序无关以及对小的数据集聚类效果明显等特点。

为了后续描述清晰，对后续所用符号进行统一定义：定义待查重数据的 simhash 指纹集合为 $S=\{s_1, s_2, \cdots, s_n\}$；定义聚类前的簇为 $C=\{C_1, C_2, \cdots, C_k\}$，聚类过程完成后的簇为 $C'=\{C_1', C_2', \cdots, C_k'\}$；定义 2 个相似数据段之间的距离度量为 $dist(s_i, s_j)$，2 个 simhash 指纹值的海明距离为 $Hamming(s_i, s_j)$。

本文选用 PAM 算法对元数据进行聚类，以 SFIndex 中所保存的 simhash 指纹值信息来表示一个数据对象，获得查重数据的指纹集合 $S=\{s_1, s_2, \cdots, s_n\}$。然后，对集合 S 中的数据对象进行聚类，将数据段分为 k 类 $C'=\{C_1', C_2', \cdots, C_k'\}$。

2 个相似数据段之间的距离度量表示为两者 simhash 指纹值的海明距离，即

$$dist(s_i, s_j)=Hamming(s_i, s_j) \quad (1)$$

所以，整个聚类过程可以描述如下。

1) 从 S 中任意选择 k 个代表对象 $\{m_1, m_2, \cdots, m_k\}$ 作为初始的中心点。

2) 指派每个剩余对象给离它最近的中心点所代表的簇 $C=\{C_1, C_2, \cdots, C_k\}$。

3) 对于每一个簇 C_i，$i\in\{1, 2, \cdots, k\}$，遍历簇中的 l_i 个非中心点对象 s_j。

a) 计算用 s_j 代替中心点 m_i 的总代价

$$cost=\sum_{j=1}^{l_i} dist(m_i, s_j)=\sum_{j=1}^{l_i} Hamming(m_i, s_j) \quad (2)$$

b) 选择总代价最小的那个对象作为新的中心点；

4) 重复 2)，3)，直到 k 个中心点不再发生变化。

最终所获得的 k 个簇 $C'=\{C_1', C_2', \cdots, C_k'\}$，就是所需要的 k 类相似数据段。

4.2　去冗阶段

去冗阶段利用压缩近邻算法，消除相似度过高的元数据。这里定义压缩后的查重子集 $S'=\{s_1', s_2', \cdots, s_v'\}$。去冗余阶段的过程可以描述如下。

1) 同样以数据段的 simhash 指纹值指代各个数据段，获取 SFIndex 中的所有数据段 simhash 指纹集合 $S=\{s_1, s_2, \cdots, s_n\}$。

2) 对集合 S 设置 2 个存储器 store 和 grabbag。

3) 将 S 的所有样本放入 grabbag 中。

4) 从 grabbag 中随机取一个数据段 simhash 指纹 s_1，放入 store。

5) 从 grabbag 中随机取出一个数据段 simhash 指纹 s_k，用 store 中的 simhash 指纹做参考集。

a) 采用近邻法对 s_k 进行分类，从 store 中找到一个与 s_k 最近的 s_{nb}，若 $s_{nb}\in C_i$，且 $s_k\in C_i$，$i\in\{1, 2, \cdots, k\}$，则认为分类正确，删除 s_k；

b) 否则，s_k 作为新的一个类别放入 store 中。

6) 对 grabbag 中所有样本进行步骤 5)的操作，直到 grabbag 为空。

7) 此时，store 中存放的就是压缩后的查重子集 $S'=\{s_1', s_2', \cdots, s_v'\}$。根据查重子集 S'中的元素，通过消除 SFIndex 中冗余度较高的元素，同时，合并 CFIndex 中冗余度较高的 cfs 清单文件，以实现对元数据进行压缩，进而获得新的 Sub-SFIndex 和 Sub-CFIndex。

5　实验分析

为了验证 Dedup^2 算法的有效性，利用一台计算机作为平台进行了一系列的实验，其配置如下：CPU 为 Intel Core Duo 2.93 GHz，内存为 2 GB，磁盘为 SATA 5 600 转。在此实验平台上利用 Java 对算法进行了编码实现，利用 Linux 源码包作为待去重数据，对查重数据的去冗效果，让 Simdedup 重复数据删除算法分别运行在原查重数据集和通过 Dedup^2 产生的查重子集上，从元数据去重比和重复数据删除比这 2 个方面进行了对比实验。

5.1　元数据去冗分析

实验通过查重索引去重比，即已消除的冗余数据占原查重数据集的比重，来验证 Dedup^2 对查重数据集的去冗效果。在不同聚类个数 k 的条件下，在 2 个数据集中进行实验，获得的元数据去重比如图 4 所示。其中数据集 1 包含 40 GB 的 Linux 源码包，数据集 2 中包含 80 GB 的 Linux 源码包。

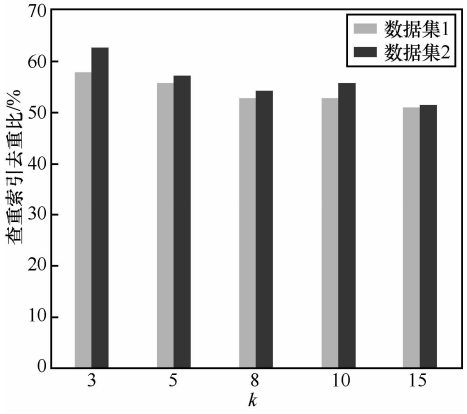


图 4　查重子集占原查重数据的比重

由实验结果可见，总体来看，Dedup^{2}对原查重数据集的去冗效果可以达到50%以上。聚类个数k的越小，最后产生查重子集的也越小，但彼此间的差距并不大。这是由于聚类个数k越大，那么聚类得到的类别也越多，那么在去冗阶段采用近邻法对某个样本进行分类时，尤其是那些位于2个类边界上的样本，发生错分的概率也会相应增加，因此，该样本就会被保留下来，进而影响了去冗效果。

此外k=8去重比相对k=10的时候要高一点。这是由于PAM算法对初始中心是随机选择的，聚类结果会随着初始点选择的变化而改变，容易陷入局部最优的情况，进而会影响最后的聚类效果。而压缩近邻算法基于聚类的结果运行，那么去冗效果也相应的会受到影响。k=8与k=10的类别个数相差不大，就比较容易发生这种因为聚类效果的差异，而导致聚类个数k的较小，但最后冗余数据量也较少的情况。但从大趋势来看，例如比较k=5、k=10以及k=15这三者，还是能清晰地得出聚类个数k越小，消除的冗余数据量越高的结论。

此外，还可以发现当数据集增大时，最后产生查重子集的也越小。这说明数据集越大，数据中的冗余数据越多，便越能产生更好的去冗效果。

5.2 重复数据删除比分析

重复数据删除比的实验在数据集1上进行。在对新数据段执行重复数据删除操作时，用于构建内存中数据块指纹索引的数据段个数w越多，那么指纹索引中所包含的已存储数据块指纹信息也越多，便越能找出重复数据。

在用于构建查重缓存的最相似数据段数量w=3，而聚类个数k不同的条件下产生多个查重子集，基于这些查重子集和原始查重数据集，分别对数据对象完成去重操作的重复数据删除率对比结果如图5所示。从实验结果可以看出聚类个数k不同对重复数据删除比没有特定的影响。此外，图中重复数据删除比最低的是基于k=15时获取的查重子集所获得，为60.3%；而重复数据删除比最高的则是基于原始查重数据集所获得，为61.4%，该结果说明虽然查重子集的大小仅为原始查重数据集的一半不到，但依然能够获得近似的重复数据删除比，进而证明了Dedup^{2}的有效性。

在所用的查重子集为聚类个数k=5时生成，而用于构建查重缓存的最相似数据段数量w不同的条件下，基于该查重子集和原始查重索引，分别对数据对象完成去重操作后的重复数据删除比对比结果如图6所示。对比结果说明随着用于构建查重缓存的最相似数据段数量w增加，利用查重子集进行重复数据删除操作时，重复数据删除比也会逐步升高，w=3之前的上升幅度较高，之后仍然会缓慢上升。而当利用原查重索引进行重复数据删除操作时，同样w=3之前的上升幅度较高，然而在w=5之后，重复数据删除比则基本停止上升。这便说明了原查重索引中，w=5之后获得的查重数据与之前的数据重复度过高，所以虽然构建查重缓存的最相似数据段数量w增加了，但对重复数据删除比却没有帮助。这也证明了Dedup^{2}元数据去冗的有效性。

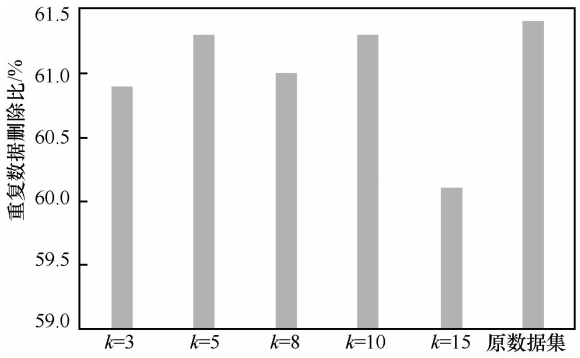


图5 k不同条件下的重复数据删除比（w=3）

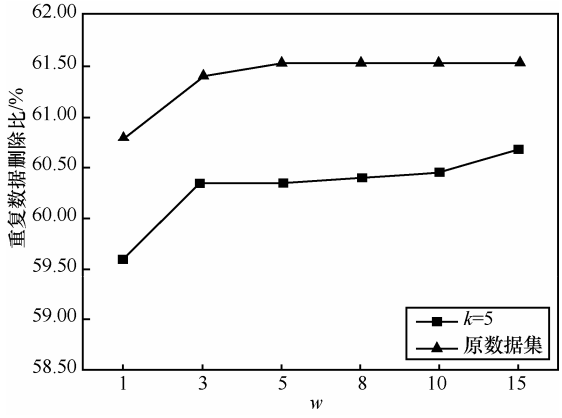


图6 w不同条件下的重复数据删除比

6 结束语

本文提出一种基于压缩近邻的查重元数据去冗算法Dedup^2，该算法定期对用于查重的元数据集进行聚类，并在此基础上利用压缩近邻算法去除查重元数据中的冗余数据，以获得精简的查重子集，基于该子集消除相似度较高的元数据，进而降低查重索引大小。结合Simdedup重复数据删除系统，可以基于文件相似性获取子集中与待去重数据对象相似度高的索引数据构成查重缓存，以完成重复

数据删除操作。实验结果证明，Dedup^2 可以在保持近似的重复数据删除比的前提下，有效消除查重元数据中的冗余信息，降低查重索引的大小。

参考文献：

[1] ZHU B, LI K, PATTERSON H. Avoiding the disk bottleneck in the data domain deduplication file system[A]. Proceedings of the 6th USENIX Conference on File and Storage Technologies, USENIX Association[C]. 2008. 1-14.

[2] LILLIBRIDGE M, ESHGHI K, BHAGWAT D, *et al*. Sparse indexing: large scale, inline deduplication using sampling and locality[A]. Proccedings of the 7th Conference on File and Storage Technologies, USENIX Association[C]. 2009. 111-123.

[3] BHAGWAT D, ESHGHI K, LONG D, *et al*. Extreme binning: scalable, parallel deduplication for chunk-based file backup[A]. In Modeling, Analysis & Simulation of Computer and Telecommunication Systems, IEEE International Symposium[C]. IEEE, 2009. 1-9.

[4] XIA W, JIANG H, FENG D, *et al*. SiLo: a similarity-locality based near-exact deduplication scheme with low RAM overhead and high throughput[A]. Proceedings of the 2011 USENIX Annual Technical Conference (ATC), USENIX Association[C].2011. 26-28.

[5] ARONOVICH L, ASHER R, BACHMAT E, *et al*. The design of a similarity based deduplication system[A]. Proceedings of SYSTOR 2009, The Israeli Experimental Systems Conference[C]. ACM, 2009. 1-14.

[6] ROMAŃSKI B, HELDT Ł, KILIAN W, *et al*. Anchor-driven subchunk deduplication[A]. Proceedings of the 4th Annual International Conference on Systems and Storage[C]. 2011.16-28.

[7] ZHANG Z, BHAGWAT D, LITWIN W, *et al.* Improved deduplication through parallel binning[A]. Performance Computing and Communications Conference (IPCCC), 2012 IEEE 31st International[C]. 2012. 130-141.

[8] DOUGLIS F, IYENGAR A. Application-specific deltaencoding via resemblance detection[A]. Proceedings of the 2003 USENIX Annual Technical Conference[C]. San Antonio, Texas, 2003. 113-126.

[9] BRODER A Z, MITZENMACHER M. Network applications of Bloom filters: a survey[J]. Internet Mathematics, 2004, 1(4): 485-509.

[10] TAN L J, YAO W B, LIU Z Y. *et al*. CDFS: a cloud-based deduplication filesystem[J]. Advanced Science Letters, American Scientific Publishers, 2012, 9(1): 855-860.

[11] TEODOSIU D, BJORNER N, GUREVICH Y, *et al*. Optimizing file replication over limited-bandwidth networks using remote differential compression[R]. Technical Report MSR-TR-2006-157, Microsoft Research, 2006.

[12] YAO W B, YE P D. Simdedup: a new deduplication scheme based on simhash[A]. In Web-Age Information Management[C]. Springer Berlin Heidelberg, 2013. 79-88.

[13] CHARIKAR M. Similarity estimation techniques from rounding algorithms[A]. Proc 34th Annual Symposium on Theory of Computing (STOC2002)[C]. 2002. 380-388.

[14] MEISTER D, BRINKMANN A. Multi-level comparison of data deduplication in a backup scenario[A]. Proceedings of SYSTOR 2009, The Israeli Experimental Systems Conference[C]. ACM, 2009.

[15] MEYER D T, BOLOSKY W J. A study of practical deduplication[J]. ACM Transactions on Storage (TOS), 2012, 7(4): 14.

[16] WALLACE G, DOUGLIS F, QIAN H, *et al.* Characteristics of backup workloads in production systems[A]. Proceedings of the Tenth USENIX Conference on File and Storage Technologies (FAST'12)[C]. 2012.

[17] WEI J, JIANG H, ZHOU K, *et al*. MAD2: a scalable high-throughput exact deduplication approach for network backup services[A]. Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium[C]. IEEE, 2010. 1-14.

[18] KAISER J, MEISTER D, BRINKMANN A, *et al*. Design of an exact data deduplication cluster[A]. Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium[C]. IEEE, 2012. 1-12.

[19] BALACHANDRAN S, CONSTANTINESCU C. Sequence of hashes compression in data de-duplication[A]. Data Compression Conference, DCC 2008[C]. IEEE, 2008. 505.

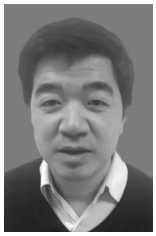
[20] CONSTANTINESCU C, PIEPER J, LI T. Block size optimization in deduplication systems[A]. Data Compression Conference, DCC'09[C]. IEEE, 2009. 442-442.

[21] ESHGHI K, LILLIBRIDGE M, WILCOCK L, *et al*. Jumbo store: providing efficient incremental upload and versioning for a utility rendering service[A]. FAST[C]. 2007. 123-138.

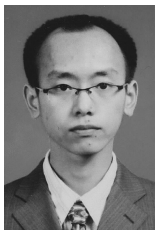
[22] MEISTER D, BRINKMANN A, SÜß T. File recipe compression in data deduplication systems[A]. Proceedings of 11th USENIX Conference on File and Storage Technologies (FAST)[C].2013. 175-182.

[23] HART P E. The condensed nearest neighbor rule[J]. IEEE Transactions on Information Theory IT-14, 1968: 515-516.

作者简介：



姚文斌（1972-），男，黑龙江哈尔滨人，北京邮电大学教授、博士生导师，主要研究方向为灾备技术、信息安全、可信计算等。



叶鹏迪（1986-），男，浙江台州人，中国铁道科学研究院助理研究员，主要研究方向为列车网络控制。

李小勇（1975-），男，甘肃天水人，北京邮电大学副教授，主要研究方向为分布式计算、网络数据分析预处理、可信计算、网络安全等。

常静坤（1988-），男，河南焦作人，北京邮电大学博士生，主要研究方向为服务计算、云灾备。