一种基于内容分块的层次化去冗优化策略

李建江,马占宁,张 凯
(北京科技大学计算机科学与技术系,北京 100083)

摘 要: 在过去的数十年中,信息数据量呈现指数级增长,如何存储和保护这些大量信息数据成为一个难题. 云存储和冗余去重技术成为解决上述难题的主要技术. 去冗技术在云存储系统中得到广泛应用,但主流的云存储系统存在索引信息的膨胀以及数据分块的不确定性等不足,而这些弊端会导致内存空间的浪费和数据分块的不可预知性. 针对这些问题,提出了一种基于内容分块的层次化去冗优化策略,并构建了对应的算法,解决了云存储系统中索引信息表过大和数据分块过大或过小的问题. 并且选取 CNN 新闻的页面内容作为测试集进行实际测试,通过比较去冗比和去冗时间可以看出,相比于目前主流的去冗策略,本文提出的基于内容分块的层次化去冗优化策略能够提升 3% 左右的去冗比,同时降低 2% 左右的去冗时间.

关键词: 云存储; 冗余去重技术; 数据分块; 层次化; 去冗比
中图分类号: TP312 **文献标识码:** A **文章编号:** 0372-2112 (2019) 05-1094-07
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2019.05.017

An Optimal Hierarchical Deduplication Strategy Based on Content Defined Chunking

LI Jian-jiang, MA Zhan-ning, ZHANG Kai
(*Department of Computer Science and Technology, University of Science and Technology Beijing, Beijing* 100083, *China*)

Abstract: In the past decades, the amount of information data is growing in an unexpected speed and how to store and protect these large amounts of data becomes a problem. Cloud storage and data deduplication technology are the principal technology to solve the above problem. Deduplication technology is widely used in cloud storage systems. However, there are some shortcomings such as the expansion of the index information and the uncertainty of the data block in the current mainstream cloud storage technology, which lead to the waste of space and a big difference in the length of the data block. To overcome these shortcomings, through the study of cloud storage and data deduplication, this paper presents an optimal hierarchical deduplication strategy based on content defined chunking and proposes the corresponding algorithm, and achieves the purpose of saving memory space and obtaining better compression performance. Finally, this paper selects the content of the CNN news as a test set. By comparing the compression ratio and the compression time, the optimal hierarchical deduplication strategy has increased compression ratio by 3% and reduced compression time by 2% compared with the current mainstream deduplication strategy.

Key words: cloud storage; data deduplication; data block partition; hierarchical; compression ratio

1 引言

人类已经进入信息技术高速发展的时代,数据将有可能成为企业的重要财富. 但是每年数据增长的速度和规模远超人们的预期,大数据所带来的首要挑战是如何存储和保护数据. IDC 预计从 2009 年到 2020 年单位容量数据的管理投资将降低近 30 倍[1]. 因此如何高效管理数据成为当前研究的重点,尤其对于一些大型企业来说. 数据的快速增长往往会消耗企业数据中心的大量存储空间,现有的存储设备很难满足大量数据存储的需要. 并且在存储设备高度集中的情况下,大量数据的传输会占据大部分网络带宽,造成网络拥堵,给使用者造成较差的用户体验.

在数据存储系统中,存在大量的重复数据和类似数

收稿日期:2017-12-26;修回日期:2018-07-13;责任编辑:蓝红杰
基金项目:国家重点研发计划课题(No. 2017YFB0202104, No. 2017YFB0202003)

据[2],其中有些重复数据可能是为了确保数据的安全性和可靠性,比如对重要数据进行备份;有些可能是由于错误操作或者其他无意的原因,对同一份数据进行多次存储. 对于这类数据,可以通过有效的算法进行去冗,对相同数据只存储 1 份,有效降低存储空间、设备购置成本和能耗[3]. 数据量的爆炸式增长对现有存储系统的各个方面都带来新的挑战,消除冗余信息进而优化存储效率成为缓解存储容量瓶颈问题的重要手段. 冗余去重技术是通过算法消除不同文件之间的重复数据,实现一定的缩减比例,减少数据存储空间. 目前该技术已被广泛应用到云存储系统,使用冗余去重技术的两个最主要的好处就是节省存储空间和数据传输的网络带宽.

本文的主要贡献如下.

(1)在分析基于内容的分块算法存在的弊端的前提下,提出了一种基于内容分块的层次化去冗优化策略,对索引表和冗余去重采用了分层次的方法,减少了索引表在内存的占有率,节省了大量内存.

(2)在已有的内容分块算法的基础上,优化策略解决了云存储系统中数据分块过大或过小的问题.

2　相关工作

冗余去重技术最早由 Data Domain 和 Avamar 公司提出[4]. 从 2006 年开始,冗余去重技术就已经成为数据存储领域的重要技术,在此之后得到了进一步的发展. 如今随着大量数据的存储需求,对于冗余去重技术的研究显得更为重要.

一般采用 SHA-1 和 MD5 等高可靠性 Hash 算法进行文件级重复数据消除. SHA-1 和 MD5 算法与其他的 Hash 算法相比,具有较低的冲突率,所以成为目前主流的 Hash 算法.

文件级重复数据消除算法在早期就被提出[5],但是由于块级重复数据消除算法具有更好的压缩性能,因此成为目前应用最广泛的重复数据消除算法. Kumar[6] 提出了基于 bucket 的重复数据消除技术,大量数据流根据固定长度的分块算法创建固定长度的块,然后将这些块提供给 MD5 算法模块以生成块的 Hash 值,并应用 MapReduce 模型来确定 Hash 值是否重复. FingerDiff 算法[7] 使用较小的预期块来检测更多的重复项,然后将连续重复或唯一的块合并,并且基于其与先前存储的对象的相似性动态地选择数据对象的分区策略,以此改善存储和带宽利用率. 文献[5~7]等没有考虑到文件、数据块数量大造成的索引表在内存中占比过高的情况.

在 Low-Bandwidth File System(LBFS)[8] 中使用 Rabin 指纹[9] 把文件划分成了变长的数据块,使得存储系统对于修改位置不再敏感. 针对基于 Hash 的 CDC(Content-defined Chunking)算法执行时间较长的问题,Widodo[10] 等人提出了一种名为快速非对称最大值的高吞吐量非 Hash 分块算法;算法使用字节值来声明切点,并利用固定大小的窗口和可变大小的窗口来查找作为切点的最大值字节. HP 实验室提出 Two Thresholds Two Divisors(TTTD)算法[11],其具有更高的概率找到切点并且可以减小分块大小的方差. 自适应数据分块算法[12] 通过对分类文件的样本进行预处理去重,然后选择适当的算法参数来提高重复数据删除率;同时,采用基于内容的快速数据分块算法(FastCDC),解决了 CDC 重复数据删除速度低的问题. 在物联网的无线射频识别(RFID)领域,吕石磊等人[13] 基于 EPC 网络架构提出一种称为 MRRE 的阅读器去冗余算法,其使用中间件的标签信息进行冗余阅读器判别,但不对标签写入信息,减少了标签信息的冗余. 文献[7,9,10]等提出的算法等均无法保证分块大小的上下限,所以可能会有极大、极小块的情况出现.

本文提出了基于内容分块的层次化去冗优化策略,并构建了对应的算法,解决了云存储系统中索引信息表过大和数据分块过大或过小的问题. 结合层次化去冗和内容分块优化策略达到了节约内存空间的目的,同时也取得了更好的去冗性能.

3　基于内容分块的层次化冗余去重优化策略

3.1　层次化冗余去重

基于内容分块的层次化冗余去重优化策略分为层次化去冗和内容分块优化算法两个部分,如图 1 所示. 层次化去冗部分对索引信息优化进行了分层,并把第 2 级索引信息存放在硬盘上;基于内容的分块优化算法部分对传统的基于内容分块算法进行了改进,防止极大极小块的产生.

在层次化冗余去重策略中,有两方面的分层.

第 1 个分层结构:元数据索引表的分层. 对于海量数据而言,索引表的内存占有率一直居高不下,影响了系统性能. 本文设计了一种分层索引结构,只将文件级索引表放置于内存,将数据块索引表放置于可快速访问的设备上,在文件级索引表上只存储指向数据块索引表的指针,从而起到释放内存压力,提升系统性能的作用. 基于内容的层次化冗余去重策略中索引组织的结构,如图 2 所示.

由图 2 可知,文件索引表(主索引表)的每条记录都以文件 ID 作为主键,同时表中还保存着文件 Hash 值(文件指纹)、文件出现次数以及 1 个数据块指针. 在文件索引表区域中,根据文件 ID 有序地存放记录,以此来保证查找的效率. 数据块索引表(从索引表)中保存着所属文件的所有非冗余数据块的详细信息、每条记录包括文件块 ID、数据块的 Hash 值、数据块引用次数、偏

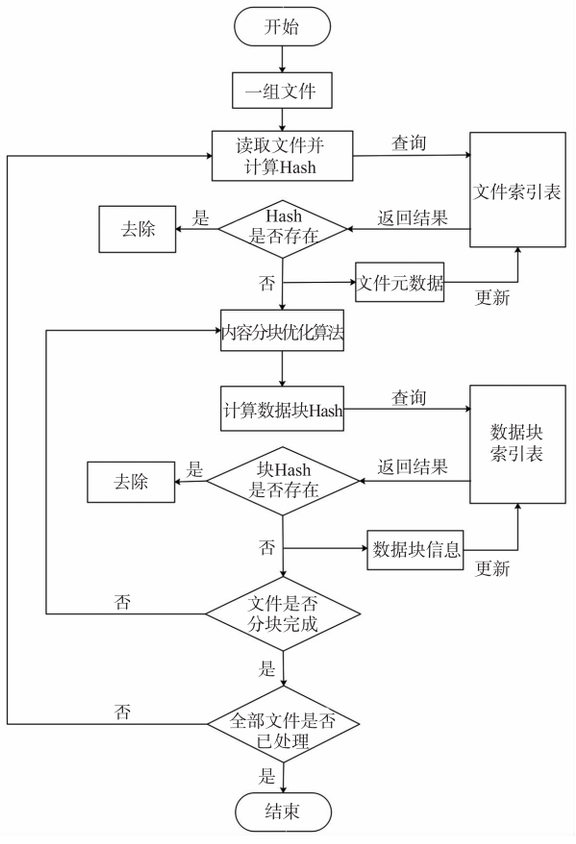

图1 基于内容分块的层次化冗余去重优化策略流程图

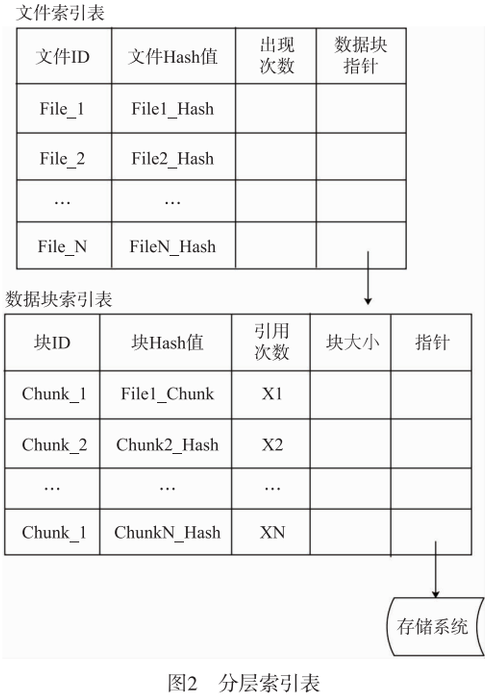

图2 分层索引表

移量大小(数据块大小)和 1 个指针,该指针指向数据实际的物理位置. 数据块索引表保存了所有非冗余数据块信息,所占的存储空间较大,因此为了节省内存,不应将其一直存储在内存,将其放置在可快速访问的设备上便是其中一种解决算法. 这样不仅可以提高访问效率,而且可以减少内存占有率,实现内存的高效使用.

第 2 个分层结构:对待存储文件进行多层过滤. 在第 1 个分层结构中,将索引表分为文件级索引表和数据块级索引表,相应的对文件冗余的检测,也要分层进行. 首先是文件级的冗余检测,若两个文件的 Hash 值相同,则将该文件丢弃,不用重复存储,只需将所包含的数据块引用次数增加 1 次即可;若不重复,在下层进行更细粒度的文件数据块的冗余检测,后续的冗余检测模块处理的都是不重复的文件,减少了工作量.

层次化冗余去重策略如算法 1 所示. 算法过程的描述如下.

算法 1 层次化冗余去重算法

```
输入:文件集合(S);
输出:文件元数据和数据块信息.
1    接收文件集 S;
2    for each file_i in S
3        打开 file_i;
4        计算 file_i_Hash;
5        if ( file_i_Hash is exist)  then
6            更新 file_i 引用次数;
7            删除 redundancy;
8            continue;
9        else
10           收集 file_i_info;
11           更新 file_i_info_table;
12       end if
13       while file_i is not finish
14           优化的基于内容分块得到 block_k;
15           计算 block_k_Hash;
16           if (block_k_Hash is exist)  then
17               更新 block_k 出现次数;
18               删除 redundancy;
19           else
20               收集 block_k_info;
21               更新 block_k_info_table;
22           end if
23       end while
24   end for
```

第 1 ~4 行:表示层次化冗余去重策略的主循环的开始,每次从文件集 S 中取一个文件,记为 file_i. 对 file_i 使用 SHA-1 Hash 函数,计算出该文件的 Hash 值,记为 file_i_Hash;

第 5 ~12 行:若存在相同 Hash 值,说明已存储完全相同的文件,只需把该文件的引用次数加 1 即可,继续处理下一个文件;若不存在相匹配的 Hash 值,表示该文

件不存在存储系统中,收集该文件的元数据信息,将该信息添加到文件索引表中,后续进行文件的分块操作.

第 13 ~ 24 行:采用基于内容的分块优化算法将文件分割成数据块,并计算每个数据块的 Hash 值,在数据块索引表中进行查询. 若存在相同 Hash 值,表示该数据块已出现,无须存储,将引用次数加 1 并令该文件的文件索引表指向描述该数据块的记录即可;若不存在相匹配的 Hash 值,表示该数据块是全新的数据块,收集其块信息,将块信息记录添加到数据块索引表中,同时保证文件级索引表指向该记录.

3.2　基于内容分块的优化算法

传统的基于内容分块的算法需要预先定义数据块的期望长度 D 和常量 r,其中 $r < D$,本文所使用的数据块的期望值 D 为 0.5KB,r 为 0 ~ D 中任意值(可任意指定),基于内容分块的优化算法首先以一个长度为 L 的滑动窗口从文件头部向文件尾部滑动. 本文规定窗口长度 L 为 32 字节,采用 SHA-1 算法计算每个位置上的指纹值. 假设对于位置 i,滑动窗口分割的数据块指纹值为 H_i,如果 H_i 模 D 得到余数为 r,则将位置 i 的右边界设为文件的 1 个分割点. 该算法中可以取得较好分块效果的前提是选择的 Hash 函数足够离散,这样数据块的 Hash 值 H 模 D 余 r 的概率才能无限接近于 $1/D$,理论情况下,按照此概率进行运算,滑动窗口每移动 D 就会出现一个符合设定参数的 Hash 值,这样就能成功得到一个文件分割点. 当然,D 毕竟是一个期望值,在实际应用中,数据块的长度会长短不一,甚至差别很大,因此可能会划分出多个极大或极小的数据块.

本文提出了一种基于内容分块的优化算法,即设定期望值的上下限范围(即最小长度和最大长度)的优化算法. 优化算法的基本思想是,首先让滑动窗口向前滑动,忽略窗口内指纹的计算和取值,直到该数据块的长度等于设定的最小长度,之后窗口的滑动按照上述计算方式依次进行,同时要保证该数据块的最大长度小于设定的最大长度. 若等于最大长度时,该数据块仍然不能满足分块条件,在此位置进行强制分块. 基于内容分块的优化算法的算法描述如算法 2 所示,具体执行过程如图 3 所示.

算法 2　基于内容分块的优化算法

```
输入:一个文件(f);
输出:多个数据块.
定义:  数据块最小长度 Min,数据块最大长度 Max,预先定义的变量
       r,滑动窗口的大小 size.
1      while f is not finish
2      L1:
3          if ( f is not feof( ) )  then
4              读取 size 窗口大小的文件内容;
5          else
6              补位到 size 大小;
7          end if
8          while block_size < Min
9              goto L1;
10         end while
11         计算指纹 F;
12         if (F mod D ! =r)  then
13             窗口前移;
14             计算指纹 F;
15             if (block_size < Max)  then
16                 goto L1;
17             else
18                 goto L2;
19             end if
20         else
21     L2:
22             得到一个数据块;
23             计算数据块 Hash 值 H;
24             if (H is exist)  then
25                 continue;
26             else
27                 storage the block;
28             end if
29         end if
30     end while
```

第 1 ~ 10 行:标签 L1 开始的位置,若数据块的长度小于 *Min* 时,须跳转到此处. 每次读取 *size* 大小的文件内容,若在文件尾时,剩余内容不足 *size*,需要补位达到 *size* 长度. 若数据块长度小于预先定义的最小数据块长度,须跳转到标签 L1 处继续滑动窗口,直至划分出的数据块长度大于 Min.

第 11 ~ 19 行:计算滑动窗口的指纹值 F. 当指纹值 F 模 D 不等于 r 时,滑动窗口需要继续前移,并计算相应的指纹值,当数据块的长度小于事先定义的数据块最大长度 Max,此时需要跳转到 L1 处继续滑动窗口;当数据块长度等于 Max,此时需要强制划分 1 个数据块,跳转到 L2 处进行相应处理.

第 20 ~ 30 行:当指纹值 F 模 D 等于 r 时,此时成功划分出 1 个数据块,计算该数据块的 Hash 值,与索引表中的数据块信息进行匹配. 若存在相同 Hash 值,说明该数据块为冗余数据,不必进行存储,只需将引用数增加 1;若不存在,说明该数据块为新数据,需要更新索引表并存放到存储系统中.

4　实验与结果

4.1　冗余去重的主要性能指标

(1)去冗比

去冗比(Radio)是去冗系统效果的一个重要指标,定义为:R = 1-输出长度/输入长度,即被成功检测出来

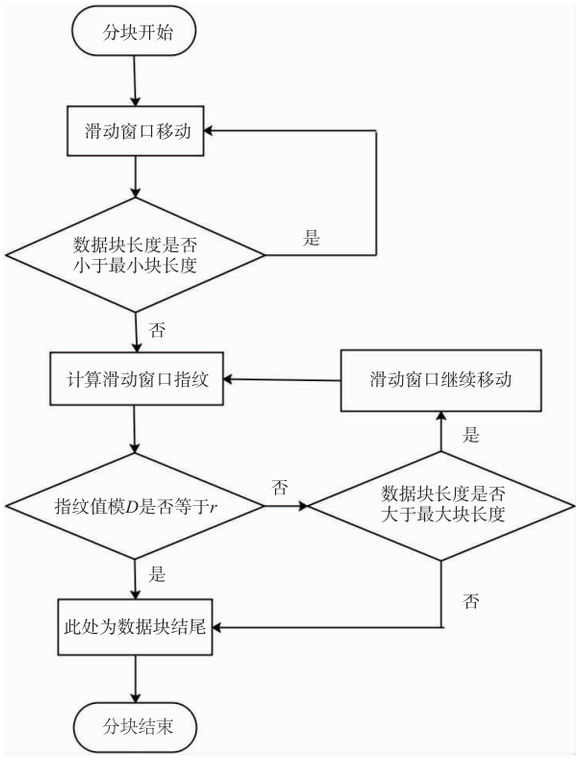

图3 基于内容分块的优化算法流程

并去除掉的冗余数据长度占输入数据长度的比例. 其中,$0 < R < 1$.

(2)去冗时间

去冗时间,也可定义为去冗时延,它也是衡量冗余去重性能的一个很重要的指标. 若去冗时延会比较小,对性能的提升就比较显著;反之,可能会造成整个数据存储系统的性能低下.

4.2 测试环境及算法

表1给出了测试所用运行环境的相关配置情况.

表1 测试环境

硬件配置	节点1	系统	Redhat 6.4
		内存	DDR3 4096M
		处理器	Intel i3-4160
		硬盘	320G/7200 转固态硬盘
	节点2	系统	Redhat 5.6
		内存	DDR3 2048M
		处理器	Intel T6640
		硬盘	240G/7200 转固态硬盘
软件配置	使用语言		C 语言
	编译器		GCC4.9

4.3 测试集及测试流程

该部分的测试数据来自 CNN 官方网站下载的新闻页面(仅文字部分),分为大中小三种规模(三种规模可以得出类似的结论,但每种规模测试结果均较多,所以以小型规模进行说明). 总共使用了100个文件,每个文件中包含多个新闻内容,文件的大小在100KB ~ 200KB,然后分别在20,40,60,80和100个文件情况下进行测试,测试服务器节点个数为文件数量除以20所得整数(最少2个),测试其在不同分块大小情况下,比较传统的固定长度分块算法、基于内容分块算法以及其优化算法的去冗比和去冗时间.

具体步骤如下.

(1)选择20个文件进行读取,分别用基于固定长度分块、基于内容分块和本文提出的基于内容分块的优化算法进行去冗测试. 一般来说,使用基于固定长度分块时,固定分块长度越大,去冗比越小,去冗时间越短;反之,固定分块长度越小,去冗比越大,去冗时间越长. 经过实验与分析,并且综合考虑去冗比和去冗时间,选取0.5KB作为本实验环境下固定长度分块算法最优的长度. 使用基于内容的分块算法时,期望数据块长度为0.5KB,滑动窗口长度为16字节,不设上下界限. 内容分块优化算法的基本设定同内容分块算法,但设定上下界限分别为0.25KB和0.75KB. 分别测试三种算法的去冗比和去冗时间.

(2)分别令文件数量为40,60,80和100,重复步骤1进行测试.

(3)将基于内容分块和及其优化算法的滑动窗口长度分别调整为32字节,64字节,重复步骤1、2进行测试,得到所有测试结果.

4.4 测试结果

(1)去冗比的比较

首先进行定长分块算法、内容分块算法和优化的内容分块算法对测试集文件的去冗比的比较. 其中的定长分块的数据块大小均选取0.5KB作为本实验环境下固定长度分块算法的长度,基于内容分块和本文提出的基于内容分块的优化算法的滑动窗口长度分别为16byte,32byte和64byte,分别测试三种算法对同一文件集的去冗比. 定长分块算法、基于内容分块算法及其优化算法的去冗比的测试结果如图4,5和6所示. 从图4,5和6可以直观的看出,在同一滑动窗口条件下,基于内容分块的优化算法的去冗比比基于内容分块算法稍微高一些. 相比较而言,内容分块算法和内容分块优化算法的去冗比 R 比定长分块算法的去冗比明显更高,更能够起到节约磁盘空间的目的. 单就节约磁盘空间而言,内容分块算法和内容分块优化算法效果更好,能够节约更多的空间. 测试结果说明,在去除冗余数据方面,本文提出的内容分块优化算法在规避了基于内容分块算法的弊端的同时,去冗比略微提升.

(2)去冗时间比较

对同一组文件分别进行定长分块算法、传统的基于内容分块算法以及优化的内容分块算法去冗时间的

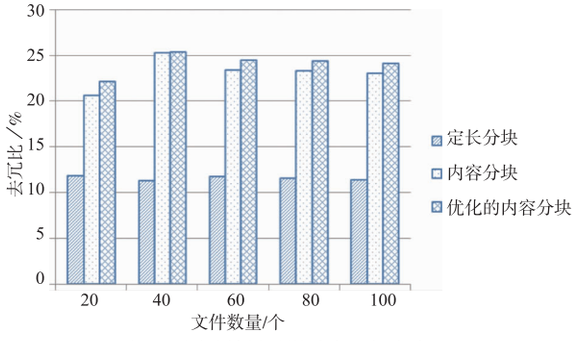

图4　去冗比测试结果（滑动窗口长度为16byte）

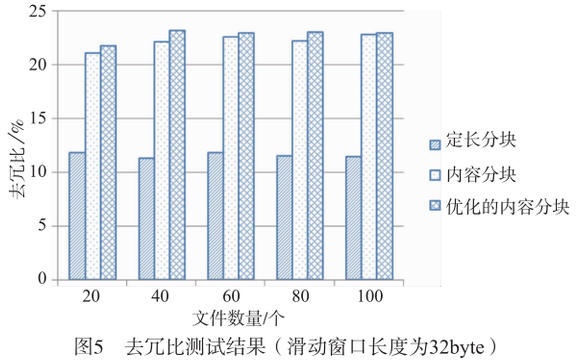

图5　去冗比测试结果（滑动窗口长度为32byte）

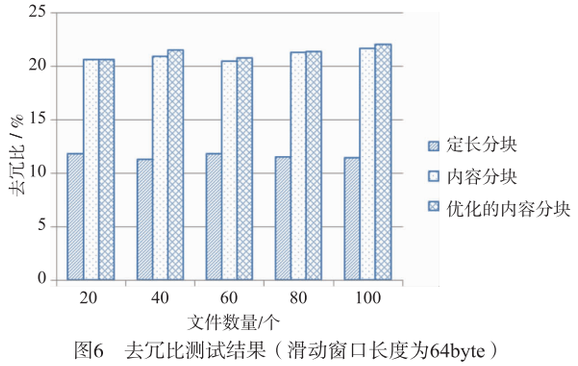

图6　去冗比测试结果（滑动窗口长度为64byte）

比较,情况如图 7,8 和 9 所示.

从图 7,8 和 9 可以直观地看出,定长分块算法的去冗时间远远小于基于内容分块和基于内容分块优化算

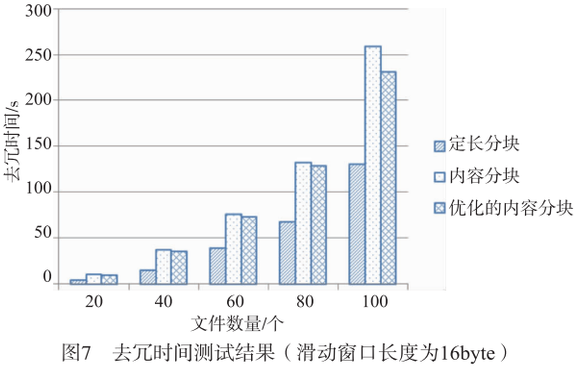

图7　去冗时间测试结果（滑动窗口长度为16byte）

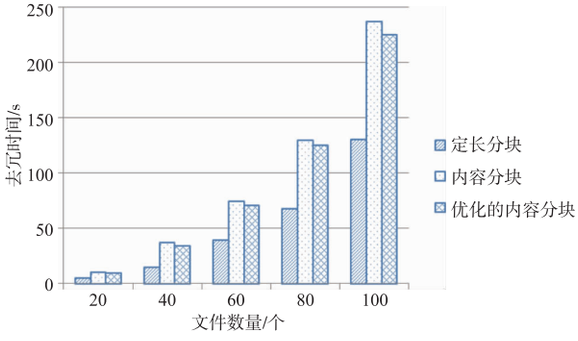

图8　去冗时间测试结果（滑动窗口长度为32byte）

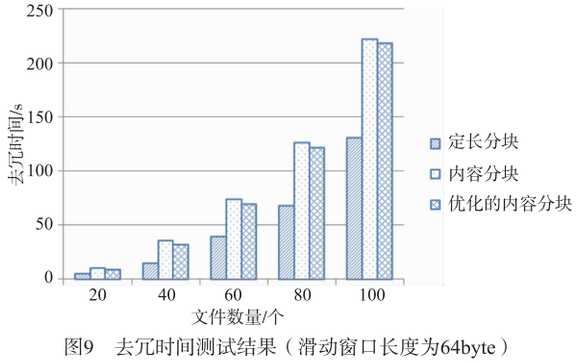

图9　去冗时间测试结果（滑动窗口长度为64byte）

法两种算法,符合定长分块去冗速度快但去冗比低下的特点. 比较内容分块算法和本文的内容分块优化算法的去冗时间,可以看出内容分块的优化算法的去冗时间略低于内容分块,这表明优化算法在规避了内容分块算法的弊端的同时,节约了一部分时间.

5　总结

本文提出了一种基于内容分块的层次化冗余去重优化策略. 首先对元数据索引表进行了分层,然后对文件级和数据块级的冗余去重进行了分层,同时在数据块级别采用了更加智能且综合性能更好的内容分块优化算法. 本文在深入研究基于内容分块算法的原理和实现步骤的基础上提出了基于内容分块的优化算法,在详细介绍基于内容分块的层次化去冗优化策略的核心技术和操作步骤之后,对该策略进行了系统测试. 结合实验数据,对三种算法的整体性能进行了综合分析. 结果显示,本文提出的策略更加智能,性能更好,有更广的应用前景.

本文提出的基于内容分块的层次化去冗优化策略仍然存在一些不足和需要继续完善的地方. 现有的冗余去重技术往往需要较高的计算能力和额外的存储资源开销,这会带来一定的负载压力. 下一步研究的一个方向是在保证去冗比例较好的前提下,使冗余去重算法减少资源占用.

参考文献

[1] Reinsel G D, Gantz J. The digital universe decade-are you ready? [R]. IDC White Paper, 2010. 1 – 16.

[2] Reinsel D. Our expanding digital world: Can we contain it? Can we manage it? [A]. Intelligent Storage Workshop (ISW) [C]. USA: University of Minnesota, 2008. 13 – 14.

[3] Nath P, Urgaonkar B, Sivasubramaniam A. Evaluating the usefulness of content addressable storage for high-performance data intensive applications [A]. Proc of the 17th Int Symp on High Performance Distributed Computing [C]. New York: ACM, 2008. 35 – 44.

[4] Gantz J, Reinsel D. As the economy contracts, the digital universe expands [R]. IDC Multimedia White Paper, 2009. 1 – 10.

[5] Sapuntzakis C P, Chandra R, Pfaff B, et al. Optimizing the migration of virtual computers [J]. ACM SIGOPS Operating Systems Review, 2002, 36 (SI): 377 – 390.

[6] Kumar N, Rawat R, Jain S C. Bucket based data deduplication technique for big data storage system [A]. Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), 2016 5th International Conference on. IEEE [C]. USA: IEEE Press, 2016. 267 – 271.

[7] Bobbarjung D R, Jagannathan S, Dubnicki C. Improving duplicate elimination in storage systems [J]. ACM Transactions on Storage (TOS), 2006, 2 (4): 424 – 448.

[8] Muthitacharoen A, Chen B, Mazieres D. A low-bandwidth network file system [A]. ACM SIGOPS Operating Systems Review [C]. New York: ACM, 2001, 35 (5): 174 – 187.

[9] Broder A Z. Some Applications of Rabin's Fingerprinting method [M]. Berlin: Springer, 1993. 143 – 152.

[10] Widodo R N S, Lim H, Atiquzzaman M. A new content-defined chunking algorithm for data deduplication in cloud storage [J]. Future Generation Computer Systems, 2017, 71: 145 – 156.

[11] Eshghi K, Lillibridge M, Wilcock L, et al. Jumbo store: providing efficient incremental upload and versioning for a utility rendering service [A]. USENIX Conference on File and Storage Technologies [C]. California: USENI, 2007. 123 – 138.

[12] Zhang X, Zhu G, Wang E, et al. Data de-duplication with adaptive chunking and accelerated modification identifying [J]. Computing and Informatics, 2016, 35 (3): 586 – 614.

[13] 吕石磊, 余顺争. 一种基于中间件的 RFID 系统阅读器去冗余算法 [J]. 电子学报, 2012, 40 (5): 965 – 970.
LÜ S L, YU S Z. A middleware-based algorithm for redundant reader elimination in RFID systems [J]. Acta Electronica Sinica, 2012, 40 (5): 965 – 970. (in Chinese)

作者简介

李建江 男, 1971 年出生, 四川人, 2005 年毕业于清华大学计算机科学与技术系, 获博士学位. 于 2014 年 1 月至 2015 年 1 月在天普大学以访问学者的身份进行学术交流. 现为北京科技大学计算机与通信工程学院副教授. 主要研究方向为并行计算、云计算、并行编译和大数据.
E-mail: jianjiangli@163.com

马占宁 男, 1988 年出生, 山东人, 2016 年毕业于北京科技大学计算机科学与技术系, 获硕士学位. 主要研究方向为分布式存储和数据冗余去除.

张 凯 (通信作者) 男, 1993 年出生, 浙江人, 2016 年毕业于北京科技大学自动化系, 获学士学位. 现为北京科技大学计算机科学与技术系研究生. 主要研究方向为云计算和并行计算.
E-mail: zhchhz@163.com