基于文件相似性分簇的重复数据消除模型*

王 灿[1,2], 秦志光[1,2], 王 娟[3], 蔡 博[1,2]
(1. 电子科技大学 计算机科学与工程学院, 成都 611731; 2. 网络与数据安全四川省重点实验室, 成都 611731; 3. 成都信息工程学院 网络工程学院, 成都 610225)

摘 要: 为解决现有提高重复数据消除系统吞吐量方法的局部性依赖和多节点依赖问题,提出了一种基于文件相似性分簇的重复数据消除模型。该模型将传统平面型索引结构拓展为空间结构,并依据 Broder 定理仅选择少量最具代表性的索引驻留在内存中;同时对索引进行横向分片并分布到完全自治的多个节点。实验结果表明,该方法能有效提高大规模云存储环境下重复数据消除性能和平均吞吐量,且各节点数据负载量均衡,故该模型可扩展性强。

关键词: 云存储; 重复数据消除; 吞吐量; 文件相似性分簇; 负载均衡
中图分类号: TP309.3 **文献标志码:** A **文章编号:** 1001-3695(2012)05-1684-06
doi:10.3969/j.issn.1001-3695.2012.05.022

Deduplication model based on file-similarity clustering

WANG Can[1,2], QIN Zhi-guang[1,2], WANG Juan[3], CAI Bo[1,2]
(1. *School of Computer Science & Engineering, University of Electronic Science & Technology of China, Chengdu* 611731, *China*; 2. *Network & Data Security Key Laboratory of Sichuan Province, Chengdu* 611731, *China*; 3. *School of Network Engineering, Chengdu University of Information Technology, Chengdu* 610225, *China*)

Abstract: To resolve the locality dependence and multiple-nodes dependence problems of the current throughput improving methods for deduplication system, this paper proposed a deduplication model based on file-similarity clustering. This model expanded the traditional flat index structure into spatial structure. According to the Broder's theorem, it kept only a handful of the most representative indices in RAM. It partitioned the index horizontally and distributed on several totally autonomous storage nodes. The experimental results indicate that the model can effectively improve the deduplication performance and the throughput on average in the large scale cloud-storage environment, and the data loads are balanced. Therefore, the model can be extended smoothly.

Key words: cloud-storage; deduplication; throughput; file-similarity clustering; load balancing

0 引言

近年来,随着各种新兴网络应用的飞速发展,电子数据量增长迅猛:2007 年新增的电子数据总量为 281 EB,2011 年该数字增长了十倍以上[1]。与此同时,对存储服务精细化和即时性的要求也不断提高。这使得基于云存储的备份应用成为信息存储领域的热点。

重复数据消除技术[2~4]是云存储应用中一种重要的存储优化技术,利用数据对象之间的信息冗余,可以获得远高于传统压缩方法及增量备份方法的空间利用率。该技术将数据对象划分为连续的、互不交叠的分块,计算每个分块的哈希值作为其唯一标志符,并将其存入到分块索引表中。数据对象被表示成各分块标志符按顺序连接在一起的哈希链,并以分块为单位进行存储。每当写入新数据对象时,用该数据对象的各分块标志符在索引表中进行分块存在性查询,并只存储那些新的分块及其标志符。由于重复的数据分块不再多次存储和传送,因此可以大幅度减小存储空间占用量和远程传送时的流量负载。

根据重复数据消除是在备份数据写入磁盘之前进行还是写入磁盘之后进行,可以分为在线和离线两种方式。在线方式的优点是不需要额外的磁盘空间来缓存待处理的数据,但在线重复数据消除在大规模应用环境下如果不能有效解决磁盘瓶颈问题,会严重降低重复数据消除系统的吞吐量。现有解决该问题的方法主要有 bloom filters[5]、locality preserved caching (LPC)[6]和 sparse indexing[7],其中前两种方法被综合应用在 data domain 系统[6]中,以提高系统的吞吐量。但 data domain 和 sparse indexing 都要求数据具有很强的局部性才能获得好的性能。而在云存储备份应用中,数据常常缺乏局部性,这种情况下,现有方法的吞吐量或重复数据消除性能会严重降低。

为了提高重复数据消除系统的并行性,从而获得更高的整体吞吐量,常采用 DHT 方法[8,9]对分块索引分片,并将分片分布到多个存储节点上。但该方法中,一个数据对象的重复数据消除操作通常会涉及到多个节点,各节点不能实现自治管理。

收稿日期: 2011-10-12; **修回日期:** 2011-11-28 **基金项目:** 教育部培育基金资助项目(708078);国家自然科学基金资助项目(60873075, 60973118)

作者简介: 王灿(1977-),男,四川成都人,博士研究生,主要研究方向为重复数据消除、海量数据灾备与恢复(wangcan1977@uestc.edu.cn);秦志光(1956-),男,四川隆昌人,教授,博导,博士,主要研究方向为信息安全;王娟(1981-),女,四川成都人,讲师,博士,主要研究方向为网络安全;蔡博(1987-),女,四川攀枝花人,硕士研究生,主要研究方向为存储优化技术.

为解决现有方法存在的问题,提出一种基于文件相似性分簇的重复数据消除模型(FCDM)。为后文叙述方便,提供缩略语对照表如表 1 所示。

表 1　缩略语对照表

缩略语	含义
chunk	连续、互不交叠的数据分块
chunkID	分块标志符,分块内容的 SHA-1 值
HT_{ID}	分块索引表
CEQ	分块存在性查询
fid(f_i)	文件 f_i 的标志符,文件内容的 SHA-1 值
rid(f_i)	文件 f_i 的代表 ChunkID
cluster(f_i)	rid(f_i) 对应的分簇,即 cluster(rid(f_i))
HS(f_i)	文件 f_i 的各 ChunkID 按顺序连接而成的哈希链

1　问题描述

1.1　磁盘瓶颈问题

传统的重复数据消除系统中,为了实现快速的 CEQ,所有非重复的 chunk 都对应 HT_{ID} 中一条记录,这种结构称为平面型的 HT_{ID}。随着数据规模的增大,平面型 HT_{ID} 的大小迅速增加,使得 HT_{ID} 不能全部装入内存,需要分页到磁盘上保存。由于哈希函数的特性,chunkID 可以看做是随机均匀分布的,因此传统方法无法有效缓存,这就导致几乎每次 CEQ 都会引发磁盘访问,严重降低了吞吐量。

Data domain[6] 用 LPC 技术成批地存储和预取那些可能会被一起访问概率很高的 chunkID,并用 bloom filters 技术快速判断新 chunk。Sparse indexing[7] 将 chunk 融合为更大的分段,量级为 10 MB 左右,重复 chunk 的检测在分段一级完成,检索源仅为挑选出的与待处理分段相似程度很高的少量分段。在具有显著局部性特征的传统备份负载下,这两种方法能够有效提高系统吞吐量。

1.2　数据的局部性

传统的备份负载通常是每天/每周将包含很多文件的目录树链接为较大的备份映像文件,因此传统备份负载是由数据流组成的,其特点是组成备份负载的数据对象较大,通常为 GB 级。由于同一目录下的文件在邻近的备份版本之间变动很小,并且几乎是以相同的顺序出现的,因此邻近的备份数据流之间有大段的重复数据。换而言之,如果本次备份数据流中 A、B、C 三个文件的 chunk 是按照该顺序出现的,那么在下一次备份数据流中如果 A 的 chunk 出现了,随之出现 B 和 C 的 chunk 的概率非常高。传统备份负载的这一特性称为数据的局部性。

由于对服务精细化和即时性要求的提高,在很多基于云存储的备份应用中,备份负载不再是由大的数据流组成,而是由细粒度的文件组成,这些较小的数据对象来自于分散的源,并且到达的顺序完全是随机的。例如多个用户随机提交的以文件为单位的备份和恢复请求;连续数据保护应用(CDP)中,任何文件一旦发生改变,就立即要备份。这样,在一段时间内到达的文件之间几乎是没有局部性可言的。

由于 data domain 和 sparse indexing 均依赖于数据的局部性,因此在缺乏局部性的、基于细粒度数据对象的非传统备份负载下,现有方法的吞吐量[6] 或重复数据消除性能[7] 会严重降低。

2　文件相似性快速判断算法

针对以上问题,本文提出 FCDM 模型及其算法。FCDM 是基于文件的相似性对 chunk 进行分簇和管理的,下面先对文件的相似性及其判断方法进行说明。

2.1　文件相似性定义

假设 H 为特征提取函数,$H(f_i)$ 为用 H 从文件 f_i 中提取的特征集合,根据 Jaccard 集合相似度测量指标[10],将文件的相似度规约为其特征集的相似度,定义文件 f_i 和 f_j 的相似度 sim(f_i,f_j) 如下:

定义 1　$$\mathrm{sim}(f_i,f_j)=\frac{|H(f_i)\cap H(f_j)|}{|H(f_i)\cup H(f_j)|} \tag{1}$$

有多种提取文件特征的方法[10~12]。本文提出一种在 Forman 方法[12] 基础上改进的基于内容分块的特征提取算法,其基本思想是:如果两个文件有大于平均分块长度的部分是相同的,那么这两个文件至少有一个分块相同的概率很高。该算法是基于 min-wise independence 置换族的,其定义如下[13]:

定义 2　设 S_n 是 n 元集合 $[n]$ 上所有置换组成的 n 元对称群。称置换族 $F\subseteq S_n$ 满足 min-wise independence 条件是指:对任意 $X=\{x_1,x_2,\cdots,x_m\}\subseteq[n]$ 和任意 $x_i\in X$,从集合 F 中随机、均匀地选取函数 h,计算 $H(X)=\{h(x_1),h(x_2),\cdots,h(x_m)\}$,则有下式成立:

$$Pr(\min(H(X))=h(x_i))=\frac{1}{|X|} \tag{2}$$

换而言之,X 中的所有元素在 h 的作用下都有相等的概率成为 $H(X)$ 中的最小元。在实际应用中,完全满足 min-wise independence 条件的哈希函数很难实现,通常使用近似满足 min-wise independence 条件的。

本文提出的文件特征提取算法的主要过程为:

a)用基于内容的分块算法 TTTD[14] 将 f_i 划分为连续、互不交叠的分块,记为 $f_i=\{c_1,c_2,\cdots,c_n\}$。

b)用哈希函数 h 计算各分块的 chunkID,得到特征集合 $H(f_i)=\{h(c_1),h(c_2),\cdots,h(c_n)\}$,其中 h 是近似满足 min-wise independence 条件的。

2.2　文件相似性快速判断

对于任意给定的 f_i,要从文件集合 F 中找出与其最相似的文件,最简单和直观的方法是将 f_i 与 $\forall f_j\in F$ 逐一按照定义 1 计算 sim(f_i,f_j),并从中找出最大值。但这种方法是不可扩展的,随着应用规模的扩大,其计算开销过大导致不可实施。

本文设计了如下快速判断文件相似性的方法:

a)选取 $H(f_i)$ 中最小的元素作为 f_i 的代表 chunkID,记为 $\mathrm{rid}(f_i)=\min(H(f_i))$。

b)如果 $\mathrm{rid}(f_i)=\mathrm{rid}(f_j)$,则认为文件 f_i 和 f_j 的相似度很高。

Broder 定理[13] 保证了上述相似性判断方法的正确性,该定理如下:

定理 1　两集合 S_1 和 S_2,$H(S_i)=\{h(x_k)\mid\forall x_k\in S_i\}$,$h$ 是满足 min-wise independence 条件的哈希函数,$\min(S_i)$ 代表集合 S_i 中的最小元,则

$$Pr(\min(H(S_1))=\min(H(S_2)))=\frac{|S_1\cap S_2|}{|S_1\cup S_2|} \tag{3}$$

证明 参见文献[13]。

Broder 定理说明当 h 满足 min-wise independence 条件时，集合 S_1 和 S_2 的最小哈希元相等的概率与这两个集合的 Jaccard 相似度指标相等。结合该定理和定义 1 可得到如下推论：

推论 1 $$\mathrm{sim}(f_i, f_j) = Pr(\mathrm{rid}(f_i) = \mathrm{rid}(f_j)) \quad (4)$$

因此如果两个文件 f_i 和 f_j 的相似度很高，则它们的代表 chunkID 相同的概率也非常高。

3 基于相似性分簇的重复数据消除模型

在上述文件相似性快速判断算法的基础上，设计了如下所述的分层索引结构并构建了基于文件相似性分簇的重复数据消除模型。

3.1 纵向分层 chunkID 索引结构

为了解决磁盘瓶颈问题，将传统平面型 HT_{ID} 纵向分为两层：代表 chunkID 索引（RIDI）和分簇 chunkID 索引（CIDI）。其结构如图 1 所示。

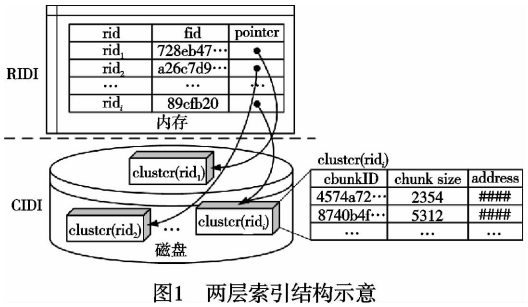

图1 两层索引结构示意

每个文件对应 RIDI 中的一条记录，该记录保存了文件的 rid、创建该记录的文件的 fid 和指向 CIDI 中对应 chunkID 簇的指针。CIDI 将 chunkID 按其所属文件的 rid 分成多个簇，每个簇中保存了具有相同 rid 的文件的全部非重复 chunkID，以及各 chunkID 对应的分块大小和存储地址等元数据信息。由于每个文件只提取一个 rid，故 RIDI 较小，驻留在内存；CIDI 中存放了所有非重复分块的元数据，故 CIDI 较大，驻留在磁盘。

3.2 基于分簇的 chunkID 检索

3.2.1 文件写入和读出

向系统写入一个文件 f_i 的主要过程如下：

a）计算 $\mathrm{rid}(f_i)$ 和 $\mathrm{fid}(f_i)$，并用 $\mathrm{rid}(f_i)$ 为条件检索 RIDI，如果返回 rid 域与 $\mathrm{rid}(f_i)$ 相同的记录，转到 b），否则转到 d）。

b）如果 $\mathrm{fid}(f_i)$ 与返回记录的 fid 域相同，说明 f_i 所有的 chunk 都是重复的，转到 e），否则转到 c）。

c）从 CIDI 中将 $\mathrm{cluster}(\mathrm{rid}(f_i))$（以下简写为 $\mathrm{cluster}(f_i)$）读入内存，然后用 f_i 的所有 chunkID 在 $\mathrm{cluster}(f_i)$ 中进行 CEQ，仅将新的 chunkID 及其元数据添加到 $\mathrm{cluster}(f_i)$ 中；当 f_i 的所有 chunkID 都处理完后，为该文件生成哈希链 $\mathrm{HS}(f_i)$，将 f_i 的新 chunk 数据、$\mathrm{HS}(f_i)$ 和更新后的 $\mathrm{cluster}(f_i)$ 写入磁盘，转到 e）。

d）为 f_i 创建一条新的 RIDI 记录和一个新的 $\mathrm{cluster}(f_i)$，并将该 RIDI 记录的指针指向该 $\mathrm{cluster}(f_i)$；然后将 f_i 的所有非重复的 chunkID 及其元数据添加到 $\mathrm{cluster}(f_i)$ 中；当 f_i 的所有 chunkID 都处理完后，为该文件生成哈希链 $\mathrm{HS}(f_i)$，将 f_i 的全部非重复的 chunk 数据、$\mathrm{HS}(f_i)$ 和新的 $\mathrm{cluster}(f_i)$ 写入磁盘，转到 e）。

e）文件 f_i 的写入过程结束。

从系统读出文件 f_i 的过程相对比较简单：

a）从磁盘上读出 $\mathrm{HS}(f_i)$，并从中提取 $\mathrm{rid}(f_i)$。

b）用 $\mathrm{rid}(f_i)$ 查询 RIDI，并根据查询返回记录的指针将对应的 $\mathrm{cluster}(f_i)$ 调入内存。

c）用组成 $\mathrm{HS}(f_i)$ 的各 chunkID 查询 $\mathrm{cluster}(f_i)$，得到组成 f_i 的各 chunk 的地址和大小等信息，然后根据这些信息将对应的 chunk 从磁盘读出、组装后得到 f_i。

从以上文件的读写流程可以看出，系统中 chunkID 的存储和检索都是基于分簇完成的。

3.2.2 ChunkID 分簇的优点

基于分簇对 chunkID 进行管理的优点主要有：

a）极大地减少了磁盘访问次数。分簇管理方法的磁盘访问次数与数据规模、数据是否具有局部性无关，每个文件读写过程中，只需要在读入对应分簇时访问磁盘一次，即可完成该文件全部 chunkID 的相关检索，因此磁盘访问的开销被组成该文件的全部 chunk 摊销了；而传统检索方法由于 chunkID 值的随机分布，无法有效缓存而导致在数据规模较大时，几乎每一次 chunkID 的查询就需要进行一次磁盘访问。

b）有效提高了对内存的利用率。由于 RIDI 只保存代表 chunkID，因此远远小于传统的平面型 HT_{ID}，可以全部装入内存，大大提高了检索和更新的速度；而传统平面型 HT_{ID} 由于非常庞大，不能全部装入内存，再加上无法有效缓存，因此即使为其分配较大的内存空间，仍需要在内存和磁盘之间频繁地进行页面的换入换出，对内存的利用率非常低。

c）保持了重复数据消除率。由于对每个文件，只选择一个 cluster 作为其重复 chunkID 的检索源，因此分簇管理方法允许系统中存在少量的重复 chunk。换而言之，对某个 chunkID，如果选中的 cluster 中没有与之相同的，但其他 cluster 中有与之相同的 chunkID，则其对应的 chunk 会被认为是新的。但由于 chunkID 是依据其所属文件之间的相似性来分簇的，因此当新来一个文件时，Broder 定理保证了选中的 cluster 是与其相似度很高的那些文件的 chunkID 的集合，故重复 chunk 的正确检出概率是非常高的。

综上所述，分簇管理方法通过引入少量的重复 chunk，可以大大提高内存利用率和减少磁盘访问，从而大幅度提高系统的吞吐量。与传统的聚类方法不同，本文提出的分簇方法是无状态的，在分簇时仅需要根据文件自身的内容，而不需要了解已有分簇的额外信息。

3.3 重复数据消除模型

3.3.1 FCDM 总体架构

云存储应用环境下的 FCDM 总体架构如图 2 所示。与传统的基于数据流的备份负载不同，FCDM 处理的是来自多个用户节点的备份文件汇集成的文件序列，其特点是文件相对较小，文件到达的顺序完全是随机的，局部性特征非常弱。

存储节点（SD）是 FCDM 中最基本的工作单元，它是具有计算和存储能力的独立工作单元，其内存和磁盘分别用于存储 RIDI 和 CIDI。单个 SD 的结构如图 2 虚线框内部分所示。

3.3.2 ChunkID 索引的横向分片及迁移策略

在大规模应用环境下，为了进一步提高系统的整体吞吐量，需要将数据分布到多个 SD 上并行处理，因此 FCDM 除了

纵向将 chunkID 索引分为两层外,还将双层的 chunkID 索引进行横向分片,并根据一定的算法将每一分片分布到某一 SD 上,该过程称为分片路由(PRT),如图 3 所示。

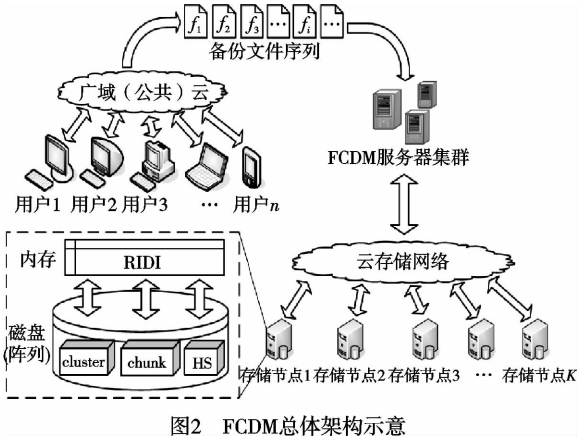

图2　FCDM总体架构示意

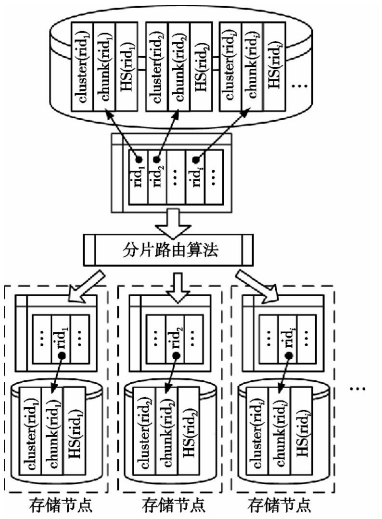

图3　索引横向分片示意

设系统中有 K 个 SD(编号为 1 ~ K),PRT 过程如下:

a)对 $\forall \mathrm{rid}_i \in \mathrm{RIDI}$,计算 $\mathrm{SD}(\mathrm{rid}_i) = \mathrm{rid}_i \bmod K$。

b)将 rid_i 对应的记录保存到编号为 $\mathrm{SD}(\mathrm{rid}_i)$ 的节点的 RIDI 中。

c)将该记录对应的分簇 $\mathrm{cluster}(\mathrm{rid}_i)$ 保存到节点 $\mathrm{SD}(\mathrm{rid}_i)$ 的磁盘 CIDI 区中,并将该分簇对应的分块数据 $\mathrm{chunk}(\mathrm{rid}_i)$ 和文件哈希链 $\mathrm{HS}(\mathrm{rid}_i)$ 也分别保存到节点 $\mathrm{SD}(\mathrm{rid}_i)$ 磁盘上的 chunk 数据区和 HS 区。

当在系统中新增或者去掉 SD,使得 SD 的数量变为了 K',此时需要重新进行 PRT。为了保证各 SD 的独立和自治,采取如下迁移策略:

a)对 $\forall \mathrm{rid}_i \in \mathrm{RIDI}$,计算 $\mathrm{SD}'(\mathrm{rid}_i) = \mathrm{rid}_i \bmod K'$。

b)将 rid_i 对应的记录从节点 $\mathrm{SD}(\mathrm{rid}_i)$ 迁移到节点 $\mathrm{SD}'(\mathrm{rid}_i)$。

c)将该记录对应的分簇 $\mathrm{cluster}(\mathrm{rid}_i)$、分簇对应的分块数据 $\mathrm{chunk}(\mathrm{rid}_i)$ 和文件哈希链 $\mathrm{HS}(\mathrm{rid}_i)$ 也一起迁移到节点 $\mathrm{SD}'(\mathrm{rid}_i)$。

3.3.3　FCDM 的主要工作流程

FCDM 处理备份负载的主要工作流程如下:

a)FCDM 服务器集群接收来自多个用户的备份文件序列 $f_1, f_2, \cdots, f_i, \cdots$。对每一个文件 f_i,根据当前各存储节点的负载情况,将其分配到某一节点 SD_c。

b)节点 SD_c 对 f_i 进行分块和特征提取,得到 $\mathrm{rid}(f_i)$,然后计算 $\mathrm{SD}_d = \mathrm{rid}(f_i) \bmod K$,并将 f_i 分配到编号为 SD_d 的节点进行重复数据消除,该过程称为文件路由(FRT)。SD_c 和 SD_d 很可能是不相同的节点,即 f_i 很有可能是在一个节点进行分块而在另一个节点进行重复数据消除。

c)由于 PRT 和 FRT 算法的一致性,保证了包含 $\mathrm{cluster}(f_i)$ 的分片也在 SD_d 上,即 SD_d 掌握了所需的全部元数据信息,可以独立完成 f_i 的重复数据消除。

d)将 f_i 的所有非重复的 chunk 数据和 $\mathrm{HS}(f_i)$ 也保存在节点 SD_d 上。

3.3.4　FCDM 的优点

从上述内容可知,FCDM 的分片方法主要具有以下特点:

a)PRT 和 FRT 算法都是无状态的,这意味着系统在决定将分片或文件路由到哪一个节点时,不需要任何关于现有节点数据的额外信息,因此算法简单,路由效率高。

b)各个节点是相互独立和自治的,任何一个文件的所有相关数据,包括 chunkID 索引、chunk 数据以及 HS 都是保存在同一个节点上的,这极大地简化了文件的管理。例如某一文件的恢复、删除、完整性校验、文件加密等不需要多个节点共同参与。

c)迁移策略保证了各节点的独立性和自治性,迁移前后不会在新旧节点之间产生任何数据的依赖关系,因此当节点环境发生了变化,重新分布数据的过程中不会造成节点之间的依赖关系。

d)一个文件的数据不被分割到多个节点保存,也提高了可靠性,因为如果一个文件的数据被分割到多个节点保存,那么其中任何一个节点的失效都会导致该文件的失效。

在分布式环境中,传统的提高系统并行性的方法是采用 DHT[8, 9] 对 $\mathrm{HT}_{\mathrm{ID}}$ 进行横向分片。例如对每一个 $\mathrm{chunkID}_i$,计算 $\mathrm{SD}_{\mathrm{DHT}}(\mathrm{ID}_i) = \mathrm{chunkID}_i \bmod K$,然后将该 chunkID 对应的索引记录保存到节点 $\mathrm{SD}_{\mathrm{DHT}}(\mathrm{ID}_i)$ 上。从上述 FCDM 模型及其算法的特点可知,FCDM 相对于 DHT 主要具有以下优点:

a)并行性更高。使用 DHT 方法时,由于 chunkID 值的随机性,在对一个文件重复数据消除的过程中,需要向多个节点发出 CEQ 请求,在最坏的情况下,甚至需要向全部的节点发出 CEQ 请求;而 FCDM 中,一个文件的所有 CEQ 请求仅需在一个节点就可完成,因此在节点数相同的情况下,FCDM 可同时处理更多的文件。

b)更加简单和便于扩展。DHT 方法中,同一个文件的 chunkID 索引和 chunk 数据是分布在多个节点上的,各个节点不是独立的,不能自治地管理数据;而 FCDM 中同一个文件的 chunkID 索引和 chunk 数据是存放在单个节点上的,各个节点自治地管理数据,文件管理更加简单,系统的可扩展性也更强。

c)资源占用更少。DHT 方法通过将 chunkID 索引分布到多个节点,使得更多的 chunkID 索引能够被装入内存,从而提高系统整体的吞吐量,但是 DHT 方法并不能减少内存占用总量;而 FCDM 由于只需要将 chunkID 的很小一部分子集装入内存,因此大大减小了内存占用总量。换而言之,在维持相同吞吐量的情况下,FCDM 所需要的节点数更少。

4 实验及分析

4.1 实验数据及实验环境

为考察 FCDM 在非传统备份负载下的性能，本文从某台文件备份服务器采集了 20 个不同用户连续两个月的备份文件作为实验数据集。采集的文件总数约为 8.79×10^6 个，总大小约为 2 TB(2 071.30 GB)，文件的平均大小约为 247 KB，这些文件在数据集中按照到达备份服务器的时间先后排序。相对于传统基于大数据流的备份负载，该数据集的特点是数据对象较小，在一段时间窗口内的数据对象之间几乎没有局部性可言。

FCDM 基于网络与数据安全四川省重点实验室的 SAN 网络搭建，用 IBM System x3850 X5 作为系统的服务器，用十台清华同方 PC 机作为存储节点。

4.2 实验结果

4.2.1 重复数据消除性能

数据压缩比(数据原始大小/消除重复数据后的大小)用来衡量重复数据消除性能。实验比较了现有主流方法和 FCDM 的压缩比，如表 2 所示。各方法的存储空间占用量随着文件总数量增加的变化趋势如图 4 所示。从表 2 和图 4 中可以看出：

a) 由于 data domain 系统采用的是平面型的 HT_{ID}，因此能够检测出所有重复的 chunk，具有最优的重复数据消除性能。

b) Sparse indexing 和 FCDM 由于使用全部索引的子集作为检索源，因此都不能检测出全部重复分块。

c) Sparse indexing 能够选出较优子集作为检索源的前提条件是备份负载是基于流的并且具有很显著的局部性，因此对基于细粒度文件的、缺乏局部性的备份负载而言，其重复数据消除性能会严重降低。

d) FCDM 由于在选择检索源时不依赖数据对象的粒度及局部性，因此能够获得接近最优的重复数据消除性能，比最优情况下仅多占用了 20.69 GB 的存储空间，仅为备份负载整体大小的 1%。

表 2 重复数据消除性能比较

方法	消冗后的总大小/GB	数据压缩比
data domain	139.70	14.83:1
sparse indexing	373.97	5.54:1
FCDM	160.39	12.91:1

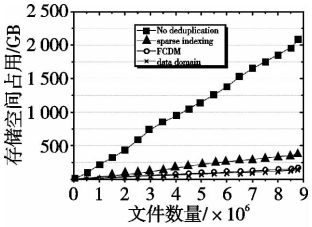

图4 存储空间占用量比较

4.2.2 内存占用

根据应用环境的复杂程度和具体实现的不同，内存中每条索引的存储开销也不同。本实验中，设内存中每条索引的存储开销为 200 Byte。实验比较了几种方法的内存总需求量，如图 5 所示。其中图 5(a)是实验所用的 TB 级数据规模得到的结果；图 5(b)是根据 TB 级实验结果，假设在文件平均大小、平均分块大小、压缩比等环境变量相同的情况下，预测的当数据量为 1 PB 时，几种方法的内存总需求量。可以看出：

a) Sparse indexing 和 FCDM 由于仅在内存中维护全部索引的一个很小的子集，因此内存总需求量远远低于 data domain 的平面型 chunkID 索引。

b) FCDM 的内存总需求量是三种方法中最低的，当数据规模达到 PB 级别时，仅为 data domain 的 1.86%，为 sparse indexing 的 44.45%。这意味着在相同条件下，如果要将索引全部装入内存，FCDM 所需要的存储节点总数远少于其他两种方法。

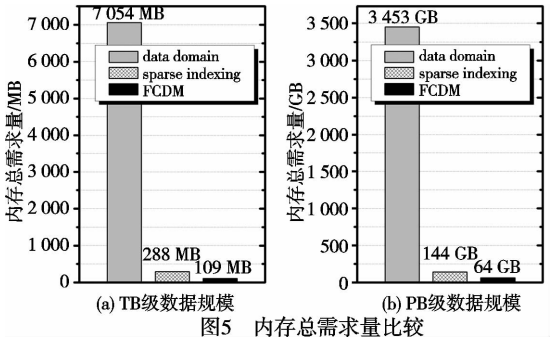

图5 内存总需求量比较

4.2.3 吞吐量

图 6 是 data domain 采用 DHT[9] 分片，文件在重复数据消除过程中所需查询节点数的情况。可以看出：

a) 当存储节点数 $K=4$ 时，有 57.46% 的文件在重复数据消除过程中需要查询 1 个以上的节点；有 27.02% 的文件需要查询全部节点。这称为 DHT 方法的多节点依赖问题。

b) 随着存储节点数的增加，多节点依赖问题更加显著。当 $K=10$ 时，有 86.75% 的文件需要查询 1 个以上的节点；有 23.15% 的文件需要查询全部节点。

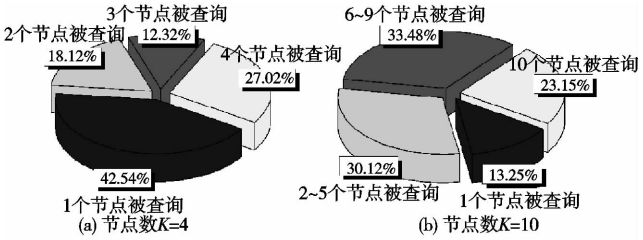

图6 DHT方法的多节点依赖问题

由于 FCDM 中，每个文件重复数据消除所需的全部信息都保存在某一个节点上，各个节点自治管理，处理每一个文件都仅需要查询 1 个节点。换而言之，在其他条件相同的情况下，FCDM 能够同时处理的文件数量比基于 DHT 分片的系统更多，这有利于提高系统整体吞吐量。

实验比较了索引不分片($K=1$)和分片($K=2 \sim 10$)的情况下，几种方法的总写入吞吐量，如图 7 所示。其中 data domain 和 sparse indexing 采用 DHT 进行分片，data domain 采用了 bloom filter 和 LPC 技术提高吞吐量。为便于比较，图中各吞吐量均以 $K=10$ 时，FCDM 的吞吐量为单位进行了归一化。可以看出：

a) 当 $K=1$ 时，data domain 的吞吐量远远低于其他两种方法，仅为 FCDM 的 13%。这是因为 bloom filter 只能快速准确断言某个 chunkID 的不存在，而对存在性的断言仍然需要查询 HT_{ID}，因此 bloom filter 对吞吐量的提高作用非常有限。而 data domain 的平面型 HT_{ID} 非常庞大，只有很小一部分能够装入内存，因此对于缺乏局部性的备份负载，LPC 不能有效缓存 chunkID，这使得 LPC 的命中率非常低，系统需要频繁地在内存

和磁盘之间交换索引页面。

b)当 $K=1$ 时,FCDM 由于大幅度减少了随机磁盘 I/O 的数量,因此相对于 data domain 有效提高了吞吐量;FCDM 每个文件仅需要提取一个 rid,并且其选取检索源的流程比 sparse indexing 更加简化,因此其吞吐量高于 sparse indexing。

c)当 $K>1$ 时,随着节点数量的增多,由于并行性的提高,各方法的吞吐量都随之提高,其中 datadomain 的增加最为显著。这是因为随着节点数的增多,总体而言装入内存的索引页面增多,从而减少了磁盘 I/O。但其吞吐量仍然远低于其他两种方法。

d)FCDM 的平均吞吐量是三种方法中最高的,并且随着节点数量的增多,与 sparse indexing 吞吐量的差异逐渐增大。如 $K=1$时,sparse indexing 的吞吐量为 FCDM 的 93.68%;$K=4$ 时,该数值为 85.05%;$K=8$ 时,该数值为 82.93%。这是因为 sparse indexing 受到 DHT 方法多节点依赖问题的影响,抵消了部分由于并行性提高而得到的吞吐量提升。换而言之,在多节点环境下,FCDM 在吞吐量方面的优势更加明显,在维持相同吞吐量的情况下,FCDM 所需的节点数少于其他两种方法。

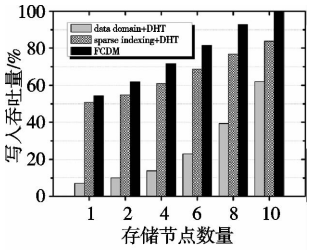

图7　写入吞吐量比较

4.2.4　负载均衡

图 8 是存储节点数 $K=10$ 时,各节点的负载数据量比较,其中无重复数据消除的系统(no deduplication)是根据 $\mathrm{fid}(f_i)$ mod K 的值将每个文件分发到对应的节点直接存储。可以看出:

a)FCDM 相对于无重复数据消除的系统大幅度减少了各节点处理的数据负载量,各节点的平均负载量仅为无重复数据消除系统的 7.74%。

b)FCDM 中各节点的负载很均衡,既没有负载很大的节点成为整个系统的瓶颈,也没有负载很小的空闲节点浪费系统资源。这一特性意味着随着数据规模的扩大,FCDM 可以平滑地扩展节点数量,稳步提升系统的总体负载能力。

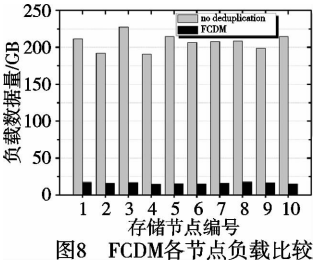

图8　FCDM各节点负载比较

5　结束语

重复数据消除是减少存储开销和网络流量负载的有效方法,如何提高重复数据消除系统的吞吐量是一个非常重要的问题。针对现有提高吞吐量方法存在的数据局部性依赖和多节点依赖问题,本文提出了一种基于文件相似性分簇的重复数据消除模型 FCDM。该模型依据 Broder 定理仅抽取少量代表性最强的索引驻留在内存中,并对索引进行横向分片和路由,显著减少了内存占用量并实现了存储节点的自治管理。真实数据集上的实验分析表明,FCDM 的重复数据消除性能接近最优值,为 sparse indexing 的 2.33 倍;平均吞吐量为 data domain 的 2.98 倍,为 sparse indexing 的 1.17 倍;内存总需求量仅为 data domain 的 1.86%,为 sparse indexing 的 44.45%。并且 FCDM 各节点的数据负载量均衡,可扩展性强。FCDM 可以有效处理云存储应用环境下基于小数据对象的、缺乏局部性特征的非传统备份负载,适用于该环境下的海量数据备份。

参考文献:

[1] GANTZ J F, CHUTE C, MANFREDIZ A, *et al*. The diverse and exploding digital universe: an updated forecast of worldwide information growth through 2011[R]. Framingham: International Data Corporation, 2008.

[2] MEYER D T, BOLOSKY W J. A study of practical deduplication [C]//Proc of the 9th USENIX Conference on File and Storage Technologies. Berkeley: USENIX Association, 2011:1-13.

[3] HARNIK D, PINKAS B, SHULMAN-PELEG A. Side channels in cloud services: deduplication in cloud storage[J]. IEEE Security & Privacy, 2010, 8(6):40-47.

[4] 王灿,秦志光,冯朝胜,等. 面向重复数据消除的备份数据加密方法[J]. 计算机应用, 2010, 30(7): 1763-1766, 1781.

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

[6] ZHU B, LI Kai, PATTERSON H. Avoiding the disk bottleneck in the data domain deduplication file system[C]//Proc of the 6th USENIX Conference on File and Storage Technologies. Berkeley: USENIX Association, 2008:269-282.

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

[8] KUBIATOWICZ J, BINDEL D, CHEN Yan, *et al*. Oceanstore: an architecture for global-scale persistent storage[C]//Proc of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2000:190-201.

[9] COX L P, MURRAY C D, NOBLE B D. Pastiche: making backup cheap and easy[C]//Proc of the 5th Symposium on Operating Systems Design and Implementation. New York: ACM, 2002:285-298.

[10] BRODER A Z. On the resemblance and containment of documents [C]//Proc of Compression and Complexity of Sequences. Washington DC: IEEE Computer Society, 1997:21-29.

[11] BRIN S, DAVIS J, GARCIA-MOLINA H. Copy detection mechanisms for digital documents[C]//Proc of ACM SIGMOD International Conference on Management of Data. New York: ACM, 1995: 398-409.

[12] FORMAN G, ESHGHI K, CHIOCCHETTI S. Finding similar files in large document repositories[C]//Proc of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining. New York: ACM, 2005:394-400.

[13] BRODER A Z, CHARIKAR M, FRIEZE A M, *et al*. Min-wise independent permutations[J]. Journal of Computer and System Sciences, 2000, 60(3): 630-659.

[14] ESHGHI K, TANG H K. A framework for analyzing and improving content-based chunking algorithms, HPL-2005-30[R]. Palo Alto: HP Labs, 2005.