软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2015,26(10):2581−2595 [doi: 10.13328/j.cnki.jos.004795]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

用户感知的重复数据删除算法*

张沪寅[1], 周景才[1,2], 陈毅波[3], 查文亮[1]

[1](武汉大学 计算机学院,湖北 武汉 430072)

[2](深圳华为技术有限公司 IT 标准与专利部,广东 深圳 518219)

[3](国网湖南省电力公司 信息通信公司,湖南 长沙 410000)

通讯作者: 周景才, E-mail: 68209669@qq.com

摘 要: 通过大量的实验分析发现:在云桌面场景下,数据拥有者之间的工作相关度越大,则该用户之间存在重复数据的概率越大.基于该实验结果,提出了用户感知的重复数据删除算法.该算法打破了数据空间局部性特征的限制,实现了以用户为单位的更粗粒度的查重计算,可以在不影响重删率的前提下,减少 5~10 倍常驻内存指纹的数量,并可将每次查重计算的指纹检索范围控制在一个常数范围内,不随数据总量的增加而线性增加,从而有效避免了因为数据总量增加而导致内存不足的问题.除此之外,该算法还能根据存储系统的负载情况自动调整重复指纹检索范围,在性能与重删率之间加以平衡,从而更好地满足主存储场景的需要.原型验证表明,该算法可以很好地解决云计算场景下海量数据的重复数据删除性能问题.与 OpenDedup 算法相比,当数据指纹总量超出内存可用空间时,该算法可以表现出巨大的优势,减少 200%以上的读磁盘操作,响应速度提升 3 倍以上.

关键词: 重复数据删除;云计算;虚拟桌面云;I/O 性能瓶颈;数据局部性

中图法分类号: TP311

中文引用格式: 张沪寅,周景才,陈毅波,查文亮.用户感知的重复数据删除算法.软件学报,2015,26(10):2581−2595. http://www.jos.org.cn/1000-9825/4795.htm

英文引用格式: Zhang HY, Zhou JC, Chen YB, Zha WL. User-Aware de-duplication algorithm. Ruan Jian Xue Bao/ Journal of Software, 2015,26(10):2581−2595 (in Chinese). http://www.jos.org.cn/1000-9825/4795.htm

User-Aware De-Duplication Algorithm

ZHANG Hu-Yin[1], ZHOU Jing-Cai[1,2], CHEN Yi-Bo[3], ZHA Wen-Liang[1]

[1](School of Computer, Wuhan University, Wuhan 430072, China)

[2](Standard & Patent Department, Huawei Technologies Co., Ltd, Shenzhen 518219, China)

[3](State Grid Information & Communication Company of Hu'nan Province, Changsha 410000, China)

Abstract: By doing a lot of experiments, if two users have more cross-project then they will own more duplication data at a virtual desktop instrument system. So, according to this finding, this paper proposes a user-aware de-duplication algorithm. This algorithm breaks the rule of data locality and can work at the new rule of user locality. According to the new rule, it just need load one user's finger print data into memory for each user group. So it can reduce 5*x*~10*x* memory requirements than other algorithm and it can control the searching scope in a limited number for each checking besides. So this algorithm can avoid a lot of read I/O operations. Meanwhile, this algorithm can adjust the searching scope dynamically according to the current workload of VDI system. Because it always tries to get the best de-duplication rate but not affect the response time of VDI system. The prototype experimental results show that it can improve the performance of de-duplication algorithm, especially when it used in a massive data storage system. Compared with OpenDedup, the algorithm can reduce more than 200% read I/O operations and can accelerate the response time more than 3*x* fast when the finger print data is bigger than available memory.

* 基金项目: 国家自然科学基金(61272454); 高等学校博士学科点专项科研基金(20130141110022)
收稿时间: 2014-07-22; 修改时间: 2014-09-29; 定稿时间: 2014-11-24

Key words: data deduplication; cloud computing; virtual desktop instrument; I/O performance bottleneck; data locality

随着云计算技术的普及,基于云计算的 VDI(virtual desktop infrastructure)系统得到快速发展.目前,无论是国内还是国外,众多大型企业和政府纷纷将自己的传统PC切换到VDI桌面云,这也将原来各个相互隔离的信息孤岛(PC 机)有机地联系了起来.有研究发现,不同用户之间的数据高达 60%是重复的[1].因此,在数据存储或传输前快速地消除用户之间的重复数据,不仅可以减少后端存储空间,而且可以减少在网络中传输的数据量,降低能量消耗和网络成本,产生巨大的社会价值.

但是,目前的重复数据删除技术应用在 VDI 桌面云中还存在一定的问题.因为当前的重复数据删除算法主要是依据数据的时间局部性和空间局部性特征设计而成,而在主存场景中,I/O 模型主要为随机的小颗粒 IO,这就会导致当前的重复数据删除算法的性能快速下降,甚至无法工作.因此,需要研究一种全新的、适合应用在云计算主存场景下的重复数据删除算法.

1 相关工作

重复数据删除算法的工作过程大致可以划分为 4 个步骤:分割数据、计算指纹、检测重复指纹和更新数据块引用关系[2−6].在这 4 个步骤中,检测重复指纹部分是最耗时的环节.如图 1 测试结果所示:OpenDedup 重复数据删除算法**的平均时间开销为 63μs,而其中的 49μs 都被检测重复指纹模块所消耗,占总开销时间的 77.7%.

```
[  701.194652] 100:2[DDP_SegExit:634][6711]SEG: DataTotalLen = 1356606722560, DdpDataLen = 1081344, CompressedDataLen = 116688.
[  701.194923] 110:4[DDP_MMClose:372][6711]Module: FunctionName: snappy_malloc, LineNum: 9, Len: 24, TotalLen: 32792, Count : 2.
[  701.194926] 110:4[DDP_MMClose:384][6711]Module: TotalMallocSize = 5088284960, TotalFreeSize = 5088252168, TotalRemainderSize = 32792.
[  701.194929] 110:4[DDP_MMClose:390][6711]Monitor: TotalMallocSize = 2319898522, TotalFreeSize = 2319898522, TotalRemainderSize = 0.
[  701.194932] 107:2[DDP_PrintPerformanceInfo:135][6711]TotalProcessedChunk = 33107110.
[  701.194934] 107:2[DDP_PrintPerformanceInfo:139][6711]TotalDedupedChunk = 264.
[  701.194936] 107:2[DDP_PrintPerformanceInfo:143][6711]TotalCompressedSize = 116688.
[  701.194938] 107:2[DDP_PrintPerformanceInfo:147][6711]TotalLoadMetaNum = 0.
[  701.194940] 107:2[DDP_PrintPerformanceInfo:151][6711]TotalWriteChunkNum = 0.
[  701.194942] 107:2[DDP_PrintPerformanceInfo:155][6711]ComputeSHA1Time = 1.3 us.
[  701.194943] 107:2[DDP_PrintPerformanceInfo:158][6711]SearchIndexTableTime = 0 us.
[  701.194945] 107:2[DDP_PrintPerformanceInfo:161][6711]SearchFPCacheTime = 0 us.
[  701.194947] 107:2[DDP_PrintPerformanceInfo:164][6711]PreocessChunksTime = 49 us.
[  701.194949] 107:2[DDP_PrintPerformanceInfo:167][6711]CompressChunksTime = 2 us.
[  701.194951] 107:2[DDP_PrintPerformanceInfo:170][6711]WriteMetaTime = 2 us.
[  701.194953] 107:2[DDP_PrintPerformanceInfo:173][6711]WriteChunksTime = 0 us.
[  701.194955] 107:2[DDP_PrintPerformanceInfo:176][6711]LoadMetaToCacheTime = 0 us.
[  701.194956] 107:2[DDP_PrintPerformanceInfo:179][6711]AvgDDTimePerIO = 63 us.
[  701.194958] 107:2[DDP_PrintPerformanceInfo:182][6711]CreateRefCntTblAvgTime = 0 us.
[  701.194960] 107:2[DDP_PrintPerformanceInfo:185][6711]DestroyRefCntTblAvgTime = 0 us.
[  701.194962] 107:2[DDP_PrintPerformanceInfo:188][6711]CreateDDPContext = 0 us.
[  701.194964] 107:2[DDP_PrintPerformanceInfo:191][6711]AvgLockTimeInSeg = 2 us.
[  701.194966] 107:2[DDP_PrintPerformanceInfo:194][6711]AvgTimeForProcChunkCB = 2 us.
```

Fig.1 Testing result: Average latency of each module of OpenDedup file system

图 1 OpenDedup 文件系统中重复数据删除模块处理时间开销测试结果

造成该问题的主要原因在于检测重复指纹时会产生大量的读磁盘操作,特别是当指纹总量超出内存的可用容量时,检索算法需要反复将磁盘中的指纹读入到内存中,从而产生大量的 I/O 操作[7−9].文献[10]中,Petros 等人曾计算过 1PB 的存储系统按 4K 规模划分数据块,大概会有 5 500GB 的指纹,这无法存储在任何一个存储阵列的内存中.所以,最早期的重复数据删除系统 SIS[2],Venti[4]以及 Store[11]的性能都受限于磁盘的性能,其吞吐率只能达到 MBps 级.

为了减少常驻内存的指纹总量,2008 年,Zhu 等人在文献[12]中介绍了基于 Bloom Filter[13]技术的重复数据删除算法.Bloom Filter 利用位数组很简洁地表示一个集合,并能判断一个元素是否属于这个集合.这样,指纹检索过程就可以分为两个步骤:首先,通过 Bloom Filter 函数判断该指纹属于哪个指纹集合?然后,再在选定的集合中检索重复指纹.这样,就可以避免读入大量无用的指纹到内存中.正如文献[12]中所描述的那样,DDFS 系统在进行重复数据删除时可以避免 99%的磁盘 I/O 操作,单节点性能可以达到 100MBps 以上.但是,采用 Bloom Filter

** http://opendedup.org 提到的 OpenDedup 是目前能够应用在主存场景下最优秀的三大开源重复数据删除技术之一(其他两个分别是 lessFS 和 ZFS).OpenDedup 能够删除 1PB 或者更多的重复数据;以 128K 块尺寸每 GB 的内存支持 3TB 以上的数据;以每秒 290MB 的速度执行内联重复数据删除.

也会带来一些新的问题.例如:如何挑选 k 个合适的 Hash 函数?每个新数据块都进行 K 次 Hash 计算的时延以及 CPU 资源占用问题等等.

2009 年,Lillibridge 等人提出的基于抽样过滤技术(sampling-based filtering)[14]的重复数据删除算法是最经典的重复数据删除算法之一.该方法主要利用了文献[12]中所描述的数据局部性特征,从 1M 大小的片段(segment)中随机挑选出一个样本指纹放在内存中.在检测重复指纹时,先比对两个 segment 的样本是否一致:如果不一致,则直接视为新数据;如果一致,再逐个比对 segment 中数据片(chunk)的指纹.这样,常驻内存的指纹数量仅为原来的 0.004,极大地减少了常驻内存指纹的总量.不过,该算法也同样存在无法忽视的缺陷:(1) segment 不能超出 1M,否则检测误报率会快速增加;(2) 主存的 I/O 模型不像备份存储的 I/O 模型是大块的、顺序的 I/O,而是随机的、小颗粒的 I/O,所以在主存系统中无法应用该算法[15].

与 Bloom Filter 和 sampling-based filtering 类似的技术还有分组索引技术(grouping index),例如,在 2013 年,Sun 等人提出的分布式相似文件元数据集合索引的构建方法[15].该方法使用位置敏感 Hash 函数[16]将具有相同数据片的相似文件元数据组成集合并建立索引,且仅将索引 lshid 保存在内存中.该方法与文献[14]相比,最大的优点是可以应用到主存场景中,且分组的粒度可以超过 1M 的限制;缺点是无法应用在块存储设备中,因为在块存储设备中无法获取文件信息.同样,Bhagwat 等人设计的 Extreme Bin 技术[17]和 Yang 等人设计的 DEBAR[18]也属于分组技术,只不过分组的粒度限制在单个文件,且同样无法应用在块存储设备中.

Li 等人提出的“有状态数据路由”和“无状态路由”重复数据删除技术[19]的基本工作原理与 Extreme Binning 类似,改进的地方主要有两点:一是放弃了基于文件的采样,采用了文献[14]中的基于 super-chunk 的采样技术(采样粒度小,该系统采用 1M 为单位),保障了重删率;二是为了避免数据存储不均衡问题的出现,他们提出了“有状态数据路由”技术.但是该算法同样也存在重删率、可扩展性和性能相互制约的问题,并最终导致集群总节点数受限.

付印金等人设计的Σ-Dedupe 系统[20]采用类似文献[19]中的超块路由粒度和容器管理策略来保持数据局部性,并提出手印(handprinting)技术来提高超块相似度的检测能力,从而能够很好地消除重复指纹检测磁盘瓶颈和重复数据删除节点信息孤岛效应.

综合对比当前经典重复数据删除算法后不难看出,目前各种重复数据删除算法都存在不同程度的限制.如表 1 所示:Sparse Index 实现简单,但只能用在备份场景中;Extreme Bin 技术适合应用到主存场景,但只能应用在文件服务器上,无法应用在块存储设备中.所以综合来看,目前还不存在一种通用的重复数据删除算法可以解决所有场景下的重复数据删除需求这一问题.

Table 1 The comparison and analysis of classic deduplication algorithm

表 1 经典重复数据删除算法对比分析

典型技术	常驻内存指纹数量与总指纹数量比	计算开销	实现复杂度	识别误报率	场景限制
Bloom Filter	1:500	大	高	高	不受限制
Sparse Index	1:250	小	低	低	仅适用于备份场景
Extreme Bin	1:86.56	中	中	中	受场景限制
HANDS[21]	1:100	大	高	低	不受限制

2 云计算场景中的重复数据分布规律分析

众多研究发现,不同应用场景下的数据重复规律是不一样的[22].例如:图片、音频以及视频文件服务器中,大多数情况下是整文件重复[23];代码库服务器则几乎不存在整文件重复;邮件服务器的正文部分一般为部分重复,而附件一般为整文件重复(in-wire deduplication of geo-replicated email database systems. http://www.pdl.cmu.edu/Retreat/retreat13.shtml).那么,云计算场景下的重复数据分布规律是什么样的呢?

如图 2 所示,在云计算场景中,云操作系统(cloud OS)首先将存储设备虚拟化,然后为每个 VM(virtual machine,虚拟机)分配一个在逻辑上完全独立的 vDisk(virtual disk,虚拟磁盘).与传统数据中心式相比,云计算场景下的数据更加集中,发现重复数据的概率也越高.除此之外,还有另外两个重要特点.

1) 每个 vDisk 都隶属于唯一的 VM,可以实现块级的隔离(根据 LUN 号区分用户),而传统文件服务器却只能实现文件级的隔离(只能根据目录区分用户);
2) 支持数据动态迁移,可以按照一定的规则将不同用户的 VM 迁移到同一台物理服务器上,更容易支持按类聚合.

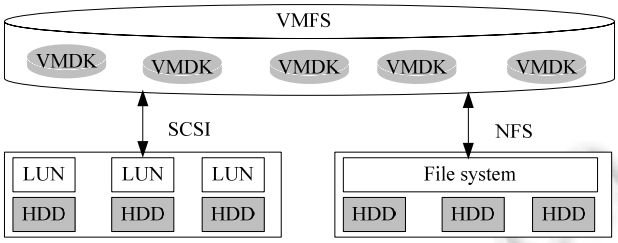


Fig.2 Storage pool in VDI system

图 2 传统数据中心的存储设备使用方式与云计算数据中心的差异

在现实中,我们经常发现相关性(包括工作相关性、业余爱好相关性、地理位置相关性等)越大的两个用户之间存在重复数据的可能性越大,而相关性越小则二者之间存在相同数据的概率越小.例如:同时使用 Windows 操作系统的两个用户之间存在重复数据的可能性比一个使用 Linux 和一个使用 Windows 的两个用户之间存在重复数据要高.为了验证该现象是否为普遍现象,我们特意设计了一个实验用于分析 vDisk 之间的重复数据量与用户的关系.实验设计如下:首先从华为的 VDI(virtual desktop instrument,虚拟桌面云)系统中(华为公司的 VDI 系统同时为该公司的 15 万员工提供云桌面服务,是目前全球最大的云计算应用系统之一),按所属组织关系随机抓取 100 个用户的 vDisk 数据(总共大约 2.3T 的数据),然后每 10 人一组进行对比分析.每组的实验对象按照工作交集度进行组合,图 3 为第 1 组实验对象的关系图.

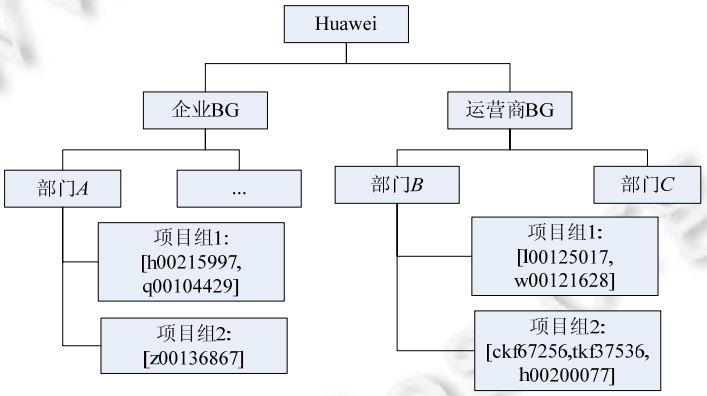


Fig.3 The user set of the first group of experiment

图 3 第 1 组实验对象所属部门关系图

实验过程如下:

步骤 1. 将上述用户 vDisk 中的文件数据按 1M 的粒度进行切割,用 SHA-1 计算每块数据的指纹,然后,每个文件 1 行记录保存到以该用户 ID 为表名的 SQLite 数据库中.考虑到隐私问题,在本次实验中,我们收集回来的数据都是指纹数据,如图 4 所示.表中的“fp”字段保存的是指纹,“fp_count”字段保存的是该文件中的总指纹个数,“file”字段保存的是对应的文件名.

步骤 2. 按字母顺序排序,挑选出一个用户的数据作为即将保存到存储介质上的新数据.

步骤 3. 按字母顺序依次选择同组中的其他用户的数据作为比对样本库,从中检索是否存在重复数据,

并记录每轮新发现的重复指纹个数.

步骤 4. 重复步骤 3,直至队列为空.

步骤 5. 重复步骤 2,直至队列为空.

步骤 6. 结束.

RecNo	id	fp	file	fp_count
		Click here to define a filter		
224068	239382	5e08c6f3cd2e21784da4d7bd38d98876e153913b	fff8e1da30fa52e4af6cc5347ce653e6f5057461	1
224069	239383	45b06845b95127ab801c1b1ec99f5165f6132b97	fff96582d3f12e1dd2ea043c6d8debf2b49bd976	1
224070	239384	df31f4bacf4f42d2dd7643c91b5bb31d1c2a79a7	fff97a8845c1142b39be07d12f580b167340102f	1
224071	239385	b8260dd72e2f4911b57871612da08c4458f4e2fc	fff9c9583a3619954ae655bc2830ea4e5f3572f9	1
224072	239386	655cf2e97385f9ebe40e82dce8e1c67f393ce27b	fff9e53073b87f5654f9a3e8a72deacadac32a74	1
224073	239387	dcd61db94c5e5873129e52c925eae05da7930e27	fffa4107305f1ba5d91e3d0c3e47845237a90f4c	1
224074	239388	e4c4d015580e5b2f7dcfa61903849ea68b8e2b38	fffab83c06dfa2f124b2ad03db69f9ddc2ef68b4	1
224075	239389	29b01b40f3762b95fc181798347754a2e8a284cf,4d805e8b2b106ab63acb4e07978fe938f9e662aa,13511f69c9754443ec0fbf410734e63d4f06e394,63747bd5a49a47bf043bcc25932ece7d29920992,0a979b64fac3b0071f4235a07d5508f492b86e34,9710e406a0683daccfe1b8b0f308900ccdd5df17	fffb32ec79874fed402691de55a2a90e912f6f5c	6

Fig.4 Fingerprint record of user

图 4 用户数据指纹库

图 5 为第 1 组的实验结果,从该图中可以清晰地看出:颜色相同的两个用户之间存在着大量的重复数据,例如 ckf67256,tkf19191,wkf37536;而颜色完全不同的用户之间,除首轮对比外,其余轮次发现的重复数据几乎为 0,例如 tkf19191 与 h00215997.不过,在数据量非常大的情况下,即使两个用户之间工作交集很弱也会存在一定数量的重复数据.例如,l00125017 与每个实验对象都存在一定数量的重复数据,因为 l00125017 的总数据量大约是其他实验对象的 10 倍左右.

	ckf67256	h00200077	h00215997	l00125017	q00104429	tkf19191	w00121628	wkf37536	y61733	z00136867
ckf67256	0	0	0	1 483	6	2 935	0	2 297	0	0
h00200077	0	0	2780	12 635	3 862	6 844	1 349	0	164	27
h00215997	0	1 502	0	20	208	0	0	9	0	50
l00125017	0	14 722	27	0	383	6 647	111	1 618	17	27
q00104429	0	13 229	220	385	0	0	34	113	89	2 202
tkf19191	0	16 556	0	6 666	0	0	750	539	0	6
w00121628	0	1 639	0	334	20	832	0	15	16	0
wkf37536	0	16 000	12	7 818	113	542	15	0	1	7
y61733	0	2 464	0	112	92	2	2	2	0	0
z00136867	0	12 786	65	2 556	2 346	80	7	0	0	0

Fig.5 Testing result of the first group of experiment (new found duplicate FPs number=C_n-C_{n-1}, C_n is the amount of duplicate FPs which be found at nth round)

图 5 每轮检索新发现的重复指纹数(新发现的指纹数=C_n-C_{n-1},C_n 为第 n 轮发现的重复指纹总数)

如果去掉样本总数的影响,进行归一化处理后,则会如图 6 所示那样:工作交集越大,则单位样本数据的重复数据率越高.

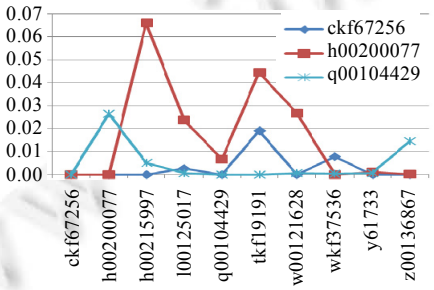


Fig.6 The duplicate data rate of unit sample data

图 6 单位样本数据的重复数据率

为了进一步分析数据的重复规律,我们统计分析了每个 vDisk 上的主要数据类型并按数据量进行排序,见表 2.从该表中可以清晰地看出:vDisk 上的数据主要由操作系统(exe,dll)、办公软件(exe,dll,xml)、开发工具(yaml,manifest)、项目代码(rb,tc,svn-base)、项目文档(PPT,txt)和用户个人数据(我们将与工作无关的数据都定义为个人数据,包括个人文档、邮件、音乐、视频、游戏等)组成,可以用公式(1)表示.

$$vDisk_i=\{操作系统,办公软件,开发工具,项目代码,项目文档,用户个人数据\} \quad (1)$$

Table 2 The amount of data (TOP 12)

表 2 各种类型文件的数据总量(仅摘录部分)

按数据量排序	Ext	q00104429 (M)	ckf67256 (M)	tkf19191 (M)	h00200077 (M)
1	svn-base	20 699	1 539	226	64 376
2	tc	15 943	5	5	59 679
3	dll	18 031	482	14 366	24 121
4	rb	18 434	3	43	30 928
5	DLL	3 281	482	14 366	24 121
6	html	30 917	77	635	8 728
7	xml	8 881	663	7 691	20 880
8	manifest	11 120	4	9 308	12 162
9	XML	356	663	7 691	20 880
10	txt	8 784	946	4 288	12 097
11	exe	4 633	1 947	9 361	9 664
12	gif	3 628	2 688	6 063	12 377
13	yaml	22 780	0	2	1 274
14	GIF	381	2 688	6 063	12 377
15	EXE	349	1 947	9 361	9 664
16	PPT	10 650	6	3 202	4 100

假设两个 vDisk 之间各类数据存在重复数据的概率分别为[$p1$,$p2$,$p3$,$p4$,$p5$,$p6$],且数据量如图 7 所示(备注:该数据来自 100 个 vDisk 的统计结果),则 $vDisk_i$ 的重复数据总量 D_{dup} 可用如下公式表示:

$$D_{dup\_i}=vDisk_i(p1_i\times 23\%+p2_i\times 8\%+p3_i\times 4\%+p4_i\times 16\%+p5_i\times 10\%+p6_i\times 39\%) \quad (2)$$

将两个 vDisk 的不同类型数据进行两两对比:user1.os_data/user2.os_data,user1.office_tools/user2.office_tools,…,则可得到如图 8 所示的统计结果.从图 8 可以清晰地看到,P_i 之间的变化趋势存在一定的关联关系.($p1$,$p2$)经常呈联动性变化,即:$p1$ 下降则 $p2$ 同步下降,$p1$ 上升则 $p2$ 上升,且幅度也经常保持一致;而($p3$,$p4$,$p5$)之间也存在类似规律;但 $p6$ 与前面所述两组之间没有明显的关联关系.

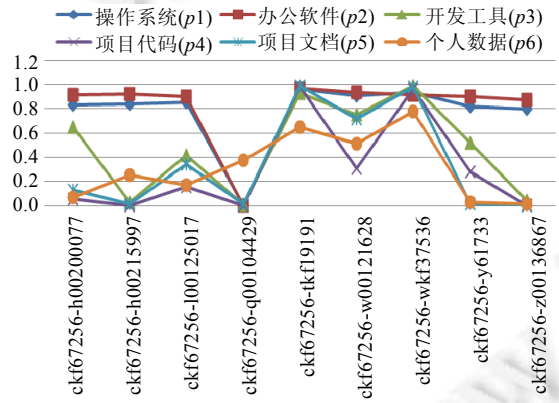


Fig.7 The value of P_i of the first group experiment

图 7 vDisk 中各类数据平均占比图,统计范围为 100 个 vDisk

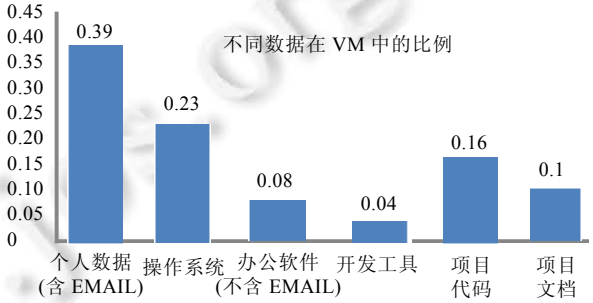


Fig.8 Average percent of data by type (counting 100 vDisks)

图 8 P_i 统计结果,统计范围:第 1 组实验

影响 P_i 分布的原因总结于表 3.

Table 3 The reasons of influence the value of P_i

表 3 影响 P_i 分布的原因总结

子项目	可选值范围	重复概率分析
$p1$: 操作系统	Windows 系列: {Windows XP,Windows 7, Windows 2000,Windows Server 2003} Linux 系列:{SuSe,Ubuntu,RedHat},...	操作系统由企业统一购买,一般只有有限的几个版本.目前, 华为 VDI 系统仅提供了 Windows 7 和 Linux SUSE 两个版本, 相同操作系统之间可能会因为补丁或版本差异而导致一定的 差异,所以 $p1$ 的取值范围一般保持在 0 或 1 两个极端
$p2$: 办公软件	{Office 系列},{Adobebat 系列}, Lotus, UltralEdit, Source insight, EA,…	办公软件由企业统一购买,可选范围大于操作系统.所以 $p2$ 的值一般 要小于 $p1$,但在总趋势上会与 $p1$ 保持一致.因为操作系统与办公软件 总体上是配套的,例如 Windows XP+Office 2003,Windows7+Office 2007
$p3$: 开发工具	{Visual Studio 系列}, {Eclipse 系列}, {华为公司自研的开发平台},...	开发工具可选择的空间较小,一般与项目强相关,要么使用 C/C++的开发工具,要么使用 Java 开发工具,所以 $p3$ 的 取值范围也非常小,要么很大,要么很小
$p4$: 项目代码	变化范围大	开发工具会决定项目代码之间的重复度,如果开发工具不一样 ($p3$ 很小),则项目代码之间的重复度一定会很小($p4$ 的值也很小); 如果开发工具一样,则 $p4$ 的值由具体项目来决定
$p5$: 项目文档	变化范围大	与具体项目相关,从事相同项目的 两个人之间的数据的重复概率高
$p6$: 个人数据	变化范围大	一般而言,同一个部门的同事之间的重复数据概率要高于非同一个 部门的同事.因为共同的部门活动,会产生大量相同的照片和视频数据, 而工作相关性又会让他们之间产生大量的相同邮件数据

综合上述实验结果和分析,我们可以得到如下结论:两个 vDisk 之间存在重复数据的总量由样本库的数据总量和重复概率 p_i 共同决定.p_i 中,[$p1$,$p2$]的取值主要由企业的软件采购策略决定,相对稳定;而[$p3$~$p6$]则相对复杂,主要由这两个 vDisk 拥有者的工作相关度、兴趣爱好、地理位置等因素综合决定.如果我们将后者统称为用户的相关度 $R_{relationship}$,并将操作系统和办公软件对应的数据统称为 *part*1,其余数据统称为 *part*2,将公式(2)简化,则可得到公式(3).

$$D_{dup}=D_{part1}\times\{0,1\}+(\alpha\times D_{part2}\times R_{relationship}) \tag{3}$$

其中,α为异常因子,取值范围为{0,1},用于排除一些异常的情况,例如工作语言(中文/英文).因为在实验中发现:即使两个人隶属于同一个部门,工作相关度非常高,但如果使用的工作语言不一样,则其重复数据的概率会大幅度降低.

有了公式(3)之后,我们再来审视实验结果(如图 6 所示),就不难理解为什么没有任何工作交集的两个用户在首轮测试时还是能找到一部分重复数据.因为 D_{part1} 部分的重复数据与用户从事什么样的工作无关,但经过首轮筛选后,剩下的主要是从 D_{part2} 中检索到的重复数据,这时会受 $R_{relationship}$ 的影响,没有任何工作交集的两个人之间的相关度有可能为 0,从而检索出来的重复数据总量也会非常少.

3 用户感知的重复数据删除算法设计

3.1 系统架构

云计算内嵌分布式重复数据删除系统部署在 Hypervisor 与 cache 之间,由一个或多个 server 组成一个集群(如图 9 所示).VM 写入磁盘的数据都先缓存到写缓存中,由写缓存将数据对齐 1M 后,再发送给重复数据删除引擎.重复数据删除引擎从任务队列中按照同一个用户数据集中处理和 FIFO(first input,first output)策略,依次对即将下盘的数据进行删冗计算,然后,将新数据写入到后端主存储 vVol(virtual volume,虚拟卷)中(虚拟卷)技术最早由 VMware 公司提出,目前已在 SNIA 标准化.该技术是架设在虚拟机和存储之间的一座桥梁,让每一个 vDisk 都可以对应到存储阵列中的一个 vVol 上,让虚拟磁盘成为存储管理和存储策略的基本单元).当 VM 需要读取数据时,如果读缓存命会失败,则该读命令会先下发给重复数据删除引擎,由重复数据删除引擎剥离引用关系后,从正确的位置读取数据并反馈给 VM.

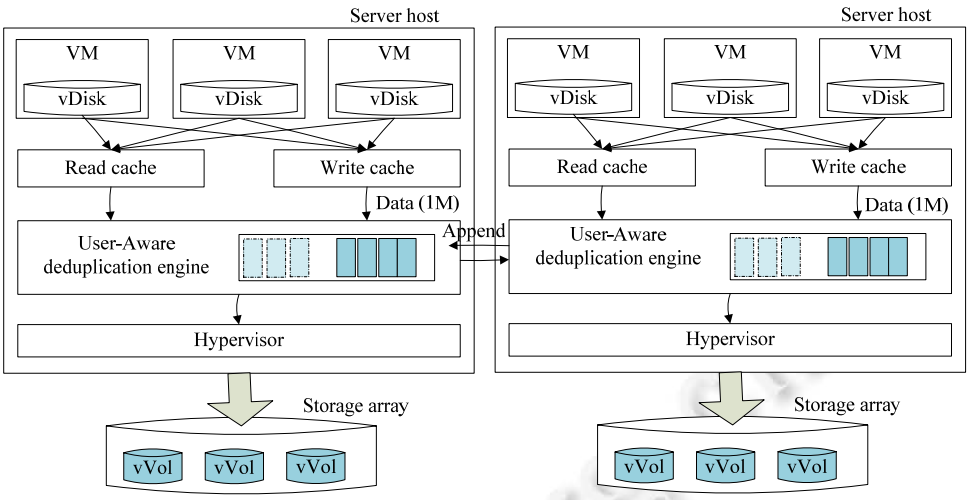


Fig.9 The architecture of vDedup system

图 9 云计算内嵌分布式重复数据删除系统架构

3.2 用户相关度计算方法

首先构建如图 10 所示的用户特征库,该特征库的数据来自两个部分:一部分直接从 VDI 系统的注册信息中获取,例如用户注册时填写的工作部门、工作时间、操作系统类型等注册信息;另一部分则需要采用用户标签自动生成算法[24]计算得到,例如用户的兴趣爱好特征.值得注意的是:特征库有一个逐步完善的过程,在数据量较少的时候,其特征信息主要来自用户的注册信息;在数据积累到一定阶段后,则可通过用户标签自动生成算法不断增加新的特征,实现更小粒度的分类聚合.$R_{relationship}$ 计算使用算法 1.

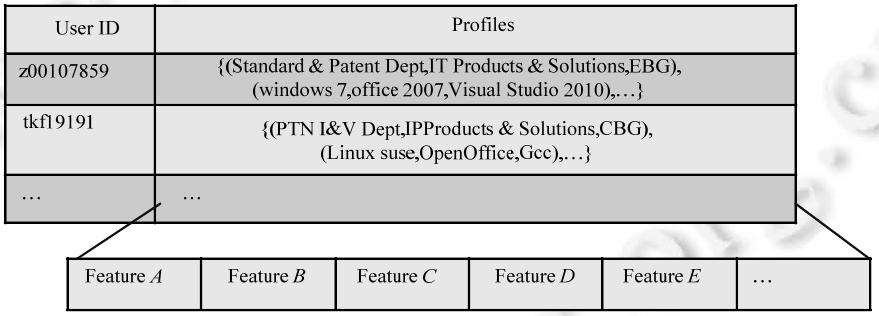


Fig.10 The user profiles library

图 10 用户特征库

算法 1. 用户相关度计算算法(user relationship evaluation algorithm,简称 UREA).

输入:*User_Profiles_DB*(用户特征库),*user_ID*1,*user_ID*2,*mask_code*(特征掩码);

输出:$R_{relationship}$ 值 //用户 1 与用户 2 之间的相关度.

1. 从 *n* 个特征项中,将影响数据量的多少作为挑选原则,选出影响数据量最大的前 4 个特征项作为比对项 ,
   放入数组变量 *profiles_list* 中;
2. i=$R_{relationship}$=0; //初始化变量
3. WHILE *profiles_list* is not *EMPTY* DO //为每一个特征项设置权重

4. $weight[i]=data_i/\sum data$; //
5. END
6. *profiles*1=*User_Profiles_DB.find*(*user_ID*1) && *mask_code*; //获取 *user*1,*user*2 的特征码,并与掩码做与运算
7. *profiles*2=*User_Profiles_DB.find*(*user_ID*2) && *mask_code*; //掩码用于动态调整需要检索的范围
8. FOR *i*=0 IN 1,…,4 //依次比对用户的特征值,如果相同,则累加上其对应的权值
9. IF *profiles*1[*i*]==*profiles*2[*i*] THEN
10. $R_{relationship}=R_{relationship}+weight[i]$;
11. END
12. return $R_{relationship}$
13. END

在算法 1 中,每个特征项对应的权重设置非常重要.例如,在企业的 VDI 系统中,个人数据占比一般不到 40%,所以与之对应的个人兴趣爱好等特征项的权重就不能超过 0.4;但在教育行业的 VDI 系统中,个人数据则有可能高达 90%,此时,个人兴趣爱好特征项的权重则需要超过工作特征的权重.

3.3 用户感知的重复数据删除算法

用户感知的重复数据删除算法由两部分组成.

- 第 1 部分负责实现指纹数据的自动聚合存储,如图 11 所示,系统首先会根据用户相关度将用户进行分组,并将同一组用户对应的指纹数据都集中存储在同一个节点上.当数据到达重复数据删除引擎时,重复数据删除引擎首先根据该数据的 *user_ID* 进行路由,从而将待查重的指纹集路由到正确的处理节点上;
- 当待查重的指纹集到达处理节点后,用户感知的重复数据删除算法第 2 部分(核心算法)将会被执行,处理过程如算法 2 所述.

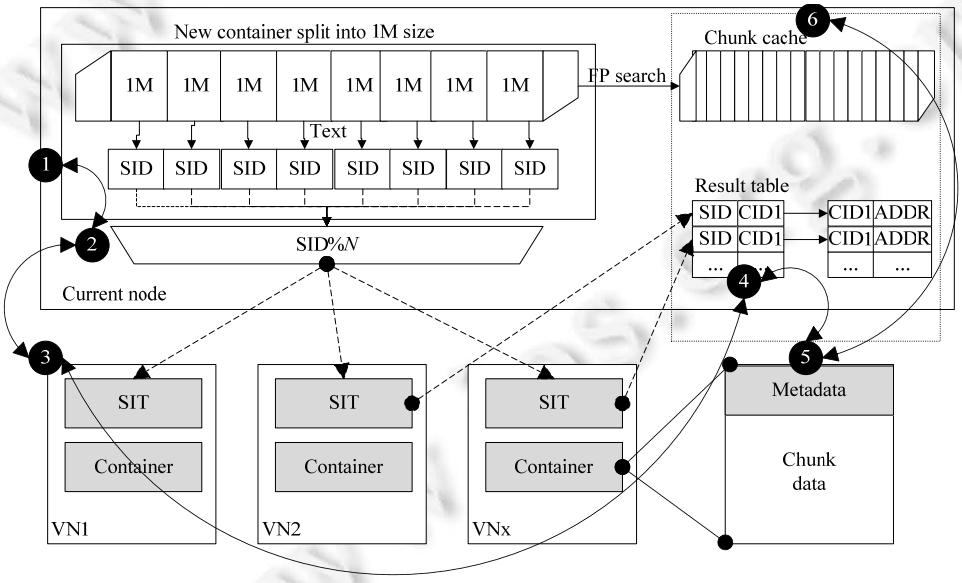


Fig.11 The proceeding of identifying duplicate data of vDedup

图 11 检查重复指纹处理流程图

算法 2. 用户感知的重复数据删除算法(user-aware deduplication algorithm,简称 UADA).

输入:*user_ID*,*mask_code*,*FP_list*(待查重指纹列表),*User_Profiles_DB*(用户特征库),*l*(最小相似度阈值);
输出:*Dup_list*(重复数据块列表),*Unique_list*(非重复数据块列表).

```
1.  BEGIN
2.  i=0, R_relationship[User_Profiles_DB.size]=0, User_list=[]   //初始化相关变量
3.  WHILE i<User_Profiles_DB.size DO   //遍历用户特征库,分别计算源用户与目标用户之间的相关度,
                                       并将大于阈值的用户添加到候选查找用户列表中
4.      R_relationship[i]=UREA(User_Profiles_DB,user_ID,User_Profiles_Database[i].User_ID,mask_code);
5.      IF R_relationship≥l THEN
6.          User_list.push(User_Profiles_DB[i].User_ID,R_relationship[i])
7.      i++;
8.  END
9.  User_list SORT BY 用户相关度 and 指纹数量;   //按用户相关度进行降序排序,相关度相同的按指纹
                                                   数量降序排序
10. //下面开始在选定的用户范围内进行查重
11. i=0, DupFP_temp_list=DupFP_list=[];
12. WHILE i<User_list.size and !FP_list.empty() DO
13.     DupFP_list=FP_list∩User_list[i].FP_list;   //将源指纹与目标指纹做交集运算,检索出重复指纹
14.     FP_list=Fp_list−DupFP_list;   //过滤掉从当前用户中检索到的重复指纹,仅将为检索到的数据
                                      进入下一轮对比
15.     DupFP_temp_list=DupFP_temp_list+DupFP_list;   //将检索到的重复指纹添加到重复数据块列
                                                        表中
16. END
17. DupFP_list=DupFP_temp_list;
18. Unique_list=FP_list−DupFP_list;
19. RETURN DupFP_list, Unique_list
20. END
```

3.4 常驻内存指纹数量与算法复杂度分析

在 UADA 算法中,仅从每个用户组中挑选出一个代表用户的指纹常驻内存,所以常驻内存的指纹数量与用户组的数量和每个 vDisk 的平均数据量成正比,如公式(4)所示.

$$Memory_{fp}=UserGroups\times vDisk/segment\_size \tag{4}$$

如果以工作相关度作为计算用户相关性的主要特征项,则一个企业的用户组个数将与该企业的最小部门数相等.根据沃顿商学院的管理学教授J.S.Mueller的研究,一个团队的最佳人数是5~11人(建设团队的最佳人数可见 http://yingyu.100xuexi.com/view/examdata/20100611/5DF7D5FF-6832-4BF5-BC32-77957C870DBD.html),则检测重复指纹时,需要从磁盘读取的数量在最好情况下为 0(在常驻内存的指纹集中全部命中),最差的情况下需要读入 n–2 个用户的指纹集.以华为公司的 VDI 系统为例,每个 vDisk 的容量为 500G,*segment_size* 为 1M,总共 15 万员工,每个团队的成员数通常在 7 人左右,采用 SHA-1 算法(计算出来的指纹长度为 20byte),则常驻内存的指纹总量仅为 200G 左右,单次检索最差情况下需要读入 5M 指纹数据.所以,UADA 算法的空间复杂度仅为 $S(n)$,n 为用户组数量.这表明:在部门数量不增加的情况下,即使 vDisk 增加,常驻内存指纹数量也不会增加.如图 12 所示,UADA 算法中的重复指纹检索部分的时间开销为 $O(\log_2 n)$,n=(最小部门人数×平均指纹数量).在用户相关度计算部分,因为需要计算两两用户的相关度,所以计算复杂度为 $O(n^2)$.但在实际应用中,用户相关度不需要实时计算,仅在新增用户或定时刷新时才需要重新计算,所以该部分的时间开销可以忽略不计.

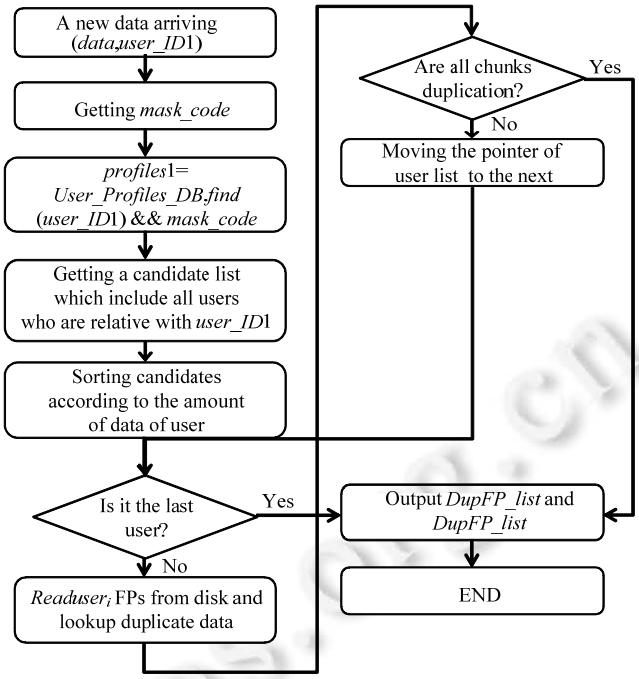


Fig.12 The flowing of user-aware deduplication algorithm

图 12 用户感知的重复数据删除算法流程图

4 实验分析

我们的实验在由 3 台服务器和 1 台 SAN 存储阵列组成的 OpenStack 云平台上进行.3 台服务器分别配备了英特尔酷睿 2 双核 E7400 2.8GHz CPU,4GB DDR-II 内存和一个 32GB 的 SSD 缓存盘.SAN 存储阵列为华为的 S5500T、50T 存储空间、RAID5.所有的机器均采用 Ubuntu 10.04.2 的 64 位服务器版作为操作系统,OpenStack 的组件部署如图 13 所示.

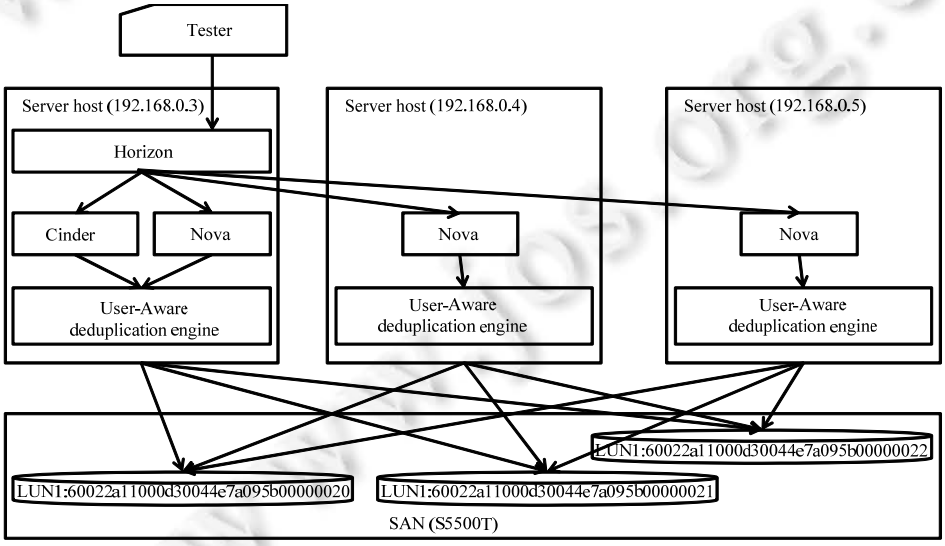


Fig.13 The deployment of test-bed

图 13 实验部署图

此外,3 台机器间均通过千兆以太网交换机连接.因此,我们假设网络传输开销对实验的性能不产生影响.

4.1 性能测试

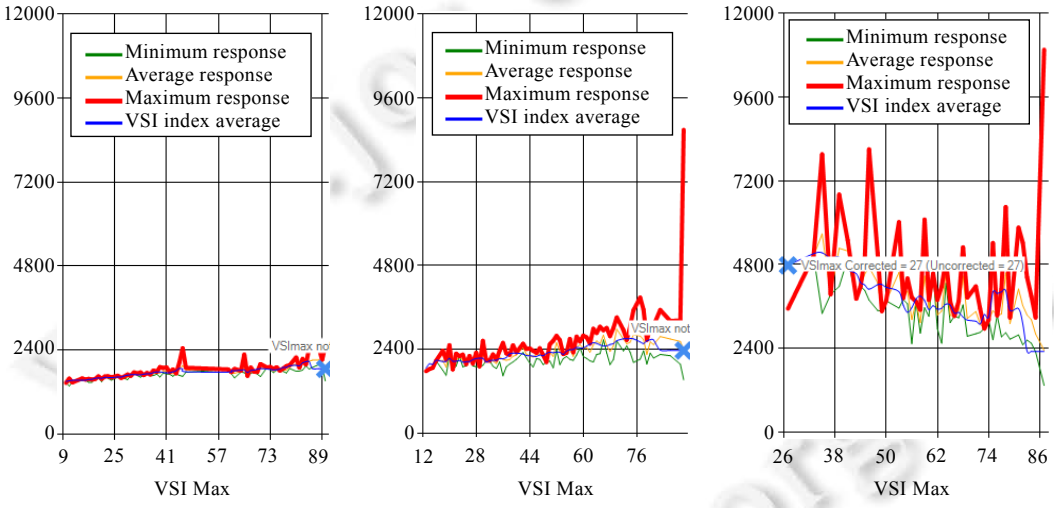
为了验证 UADA 算法在整个云计算系统实际应用中的效果,我们直接采用 LoginVSI 测试工具分别测试 VDI 系统在安装了 UADA,OpenDedup 和 lessFS 这 3 种重复数据删除算法情况下各自的性能表现.测试数据来自前面收集的 100 个 vDisk.测试过程如下:

第 1 步. 执行 Login VSI 工具,监测 VDI 系统的 I/O 响应时间;

第 2 步. 按 vDisk 编号顺序依次将 vDisk 上的数据迁移到测试环境的服务器上,并激活该 VM;

第 3 步. 统计分析测试结果.

从图 14 可以清晰地看出:在 UADA 算法测试过程中,随着 VM 数量的增加,VDI 系统的最大响应时间基本保持稳定,都控制在最佳用户体验范围内(响应时间<4800μs);但在 OpenDedup 算法中,当指纹数量达到最大可用内存的临界值时,VDI 系统的响应延迟时间突然增加 3 倍左右,达到 9 000 多μs;lessFS 在本次测试中表现最差,当 VM 数量达到 27 个时,VDI 系统的响应时延就已超出 4 800μs,不仅平均响应时延最大,而且表现极不稳定(这也说明,依赖数据的空间局部性特征设计的重复数据删除算法不适合应用在主存场景下).所以从最终应用效果来看,UADA 算法在性能上有非常大的改进,尤其是它的性能不随数据总量的增加而降低这一特点,是其他算法所不具备的.



(a) UADP 系统测试结果 (b) OpenDedup 系统测试结果 (c) lessFS 系统测试结果

Fig.14 The performance testing result

图 14 性能测试对比图

在第 1 节中我们曾分析过,影响重复数据删除算法性能的主要瓶颈之一是检索重复指纹时产生的大量磁盘读 I/O 操作.所以在上述实验中,我们特别统计了 3 种算法在检索重复指纹时累计发出的读指纹操作指令数.如图 15 所示,UADA 算法产生的读 I/O 操作数量基本上都保持在较低的水平上,几乎保持在一个常量范围,产生毛刺的地方都是在调整常驻内存的用户指纹的时候,因为在本系统中我们设计了一个自动校正模块,该模块每隔一段时间就会自动推荐每个用户组中拥有指纹数量最多的用户成为该用户组的新代表,并自动将该用户的指纹数据导入内存中,所以在每次更新时会产生大量读操作.与 UADA 算法相比,在常驻内存指纹总量未超过可用内存总量时,OpenDedup 的读操作数量是随数据量的增加而缓慢增加的,其总值大约是 UADA 的两倍;但当 vDisk 数量达到 69 个时,两者之间的差距瞬间增大到 8 倍左右.

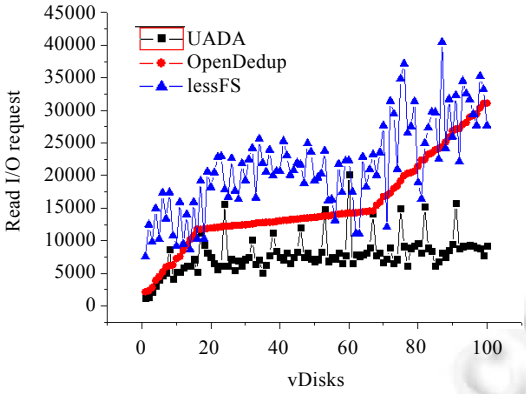


Fig.15 The migration time of each VM, under in different deduplication system

图 15 读磁盘次数对比图

除性能上的改善以外,UADA 算法在重删率上也达到了不错的效果,见表 4.所以综合来看,UADA 算法与现有的重复数据算法相比有较大的改进,特别是在存储海量数据时具有较为突出的优势.

Table 4 The total reduction @ different deduplication algorithm

表 4 重删率统计

Algorithm (chunk@64k)	Source data (GB)	Target data (GB)	Total reduction (%)
UADA	117 145.652	57 401.369	51.01
OpenDedup	117 145.652	55 175.602	53
lessFS	117 145.652	65 953.002	43.71

4.2 特征掩码与重删率测试

在主存场景中,用户体验的优先级要远大于节省存储空间的优先级.所以在前面的 UADA 算法中,我们设计了一个特征掩码来动态调整存储系统的性能与重删率,这样就可以避免与生产系统竞争计算资源,优先保障了生产系统的响应时间.在下面的实验中,我们分别测试了 l=1 和特征掩码为[255.255.255.255,255.255.255.0,255.255.0.0,255.0.0.0,0.0.0.0]这 5 种情况下的重删率和性能.0.0.0.0 是一个特殊掩码,表示仅与 UserGroup 常驻内存的指纹进行对比,一般在 CPU 负载超过 60%时被激活.测试结果见表 5:当所有检索过程全部在内存中完成时,响应时间最短,仅为 1 143μs;但与最佳重删率相比,在本轮实验中遗漏了 6.81%的重复数据.当特征掩码为 255.255.0.0 和 255.0.0.0 时,VDI 系统的平均响应时间超出了用户能够忍受的 4 800μs 时延范围.仅有当特征掩码为 255.255.255.255,255.255.255.0 和全 0 时,VDI 系统才能正常工作.从测试结果来看,特征掩码可以很好地达到主存场景下通过动态调整特征掩码来优先保障生产系统性能的目标.

Table 5 The total reduction @ different mask_code

表 5 不同特征掩码情况下的重删率和响应时间测试结果汇总表

	Total reduction (%)	Average response time (μs)
第 1 轮(255.255.255.255)	51.01	2 143
第 2 轮(255.255.255.0)	53.30	3 178
第 3 轮(255.255.0.0)	55.28	4 922
第 4 轮(255.0.0.0)	56.11	7 564
第 5 轮(0.0.0.0)	49.30	1 143

5 总 结

随着数据的快速增加和绿色 IT 进程的不断推进,重复数据删除技术将会成为网络存储领域的核心技术.从

目前的研究情况来看,分组检索是提升重复指纹检测性能的有效方法之一.而基于数据用户局部性特征的指纹分组检索具有实现简单、计算量小且误报率低的特点,并支持动态调整性能,将具有很好的工业应用价值.目前,UADA 算法已申请专利(公开号:CN103902686A),并正在应用到新产品中.在接下来的工作中,我们将重点投入到用户标记算法的设计以及不同云计算用户群体所对应的数据特征模型的分析中,从而帮助 UADA 算法实现更加精准的重复指纹检索.

致谢 感谢华为同事提供的大量实验数据和在异常数据甄别过程中所提供的帮助.

References:

[1] Fu YJ, Xiao N, Liu F, Bao XQ. Deduplication based storage optimization technique for virtual desktop. Journal of Computer Research and Development, 2012,49(S1):125−130 (in Chinese with English abstract).

[2] Bolosky WJ, Corbin S, Goebel D, Douceur JR. Single instance storage in Windows 2000. In: Proc. of the 4th Conf. on USENIX Windows Systems Symp. Berkeley: USENIX Association, 2000. 13−24. http://www.usenix.org/events/usenix-win2000/bolossky.html

[3] Quinlan S, Dorward S. Venti: A new approach to archival storage. In: Proc. of the 1st USENIX Conf. on File and Storage Technologies (FAST 2002). Berkeley: USENIX Association, 2002. https://www.usenix.org/legacy/publications/library/proceedings/fast02/quinlan.html

[4] Muthitacharoen A, Chen B, Mazieres D. A low-bandwidth network file system. In: Proc. of the ACM SOSP 2001. New York: ACM Press, 2001. 174−187. [doi: 10.1145/502034.502052]

[5] Dubnicki C, Gryz L, Held T, Kaczmarczyk M, Kilian W, Strzelczak P. Hydrastor: A scalable secondary storage. In: Proc. of the USENIX FAST 2009. Berkeley: USENIX, 2009. 197−210.

[6] Ungureanu C, Atkin B, Aranya A, Gokhale S, Rago S, Całkowski G, Dubnicki G, Bohra A. HydraFS: A high-throughput file system for the HYDRAstor content-addressable storage system. In: Proc. of the USENIX FAST 2010. Berkeley: USENIX, 2010. 165−188. https://www.usenix.org/legacy/events/fast10/tech/

[7] Ao L, Shu JW, Li MQ. Data deduplication techniques. Ruan Jian Xue Bao/Journal of Software, 2010,21(5):916−929 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3761.htm [doi: 10.3724/SP.J.1001.2010.03761]

[8] Fu YJ, Xiao N, Liu F. Research and development on key techniques of data deduplication. Journal of Computer Research and Development, 2012,49(1):12−20 (in Chinese with English abstract).

[9] Ng CH, Patrick P. RevDedup: A reverse deduplication storage system optimized for reads to latest backups. In: Proc. of the 4th ACM SIGOPS Asia-Pacific Workshop on Systems. Singapore, 2013. [doi: 10.1145/2500727.2500731]

[10] Guo F, Efstathopoulos P. Building a high-performance deduplication system. In: Proc. of the 2011 USENIX Annual Technical Conf. (USENIX 2011). Berkeley: USENIX Association, 2011. 331−345.

[11] Eshghi K, Lillibridge M, Wilcock L, Belrose G, Hawkes R. Jumbo store: Providing efficient incremental upload and versioning for a utility rendering service. In: Proc. of the USENIX FAST 2007. Berkeley: USENIX, 2007. 123−138.

[12] Zhu B, Li K, Patterson R. Avoiding the disk bottleneck in the data domain deduplication file system. In: Proc. of the USENIX FAST 2008. Berkeley: USENIX, 2008. 269−282.

[13] Bloom BH. Space/Ttime trade-offs in Hash coding with allowable errors. Communications of the ACM, 1997,13(7):422−426. [doi: 10.1145/362686.362692]

[14] Lillibridge M, Eshghi K, Bhagwat D, Deolalikar V, Trezise G, Camble P. Sparse indexing: Large scale, inline deduplication using sampling and locality. In: Proc. of the 7th Conf. on File and Storage Technologies (FAST 2009). Berkeley: USENIX Association, 2009. 111−123.

[15] Sun J, Yu HL, Zheng WM. Index of meta-data set of the similar files for inline de-duplication in distributed storage systems. Journal of Computer Research and Development, 2013,50(1):197−205 (in Chinese with English abstract).

[16] Andoni A, Datar M, Immorlica N, Indyk P, Mirrokni V. Locality-Sensitive Hashing scheme based on p-stable distributions. In: Proc. of the 20th Annual Symp. on Computational Geometry. New York, 2004. 253−262. [doi: 10.1145/997817.997857]

[17] Bhagwat D, Eshghi K, Long D, Lillibridge M. Extreme Binning: Scalable, parallel deduplication for chunk-based file backup. In: Proc. of the IEEE MASCOTS. 2009. 67−71. [doi: 10.1109/MASCOT.2009.5366623]

[18] Yang TM, Jiang H, Feng D, Niu ZY, Zhou K, Wan YP. DEBAR: A scalable high-performance de-duplication storage system for backup and archiving. In: Proc. of the Parallel & Distributed Processing (IPDPS 2010). IEEE, 2010. 1−12. [doi: 10.1109/IPDPS.2010.5470468]

[19] Dong W, Douglis F, Li K, Patterson H, Reddy S, Shilane P. Tradeoffs in scalable data routing for deduplication clusters. In: Proc. of the USENIX FAST 2011. Berkeley: USENIX, 2011. 15−30.

[20] Fu YJ, Jiang H, Xiao N. A scalable inline cluster deduplication framework for big data protection. Lecture Notes in Computer Sciences, 2012,7662:354−373.

[21] Wildani A, Miller EL, Rodeh O. HANDS: A heuristically arranged non-backup in-line deduplication system. In: Proc. of the 29th IEEE Int'l Conf. on Data Engineering. Brisbane, 2013. 254−261. [doi: 10.1109/ICDE.2013.6544846]

[22] Ma JW, Zhao B, Wang G, Liu XG. Adaptive pipeline for deduplication. In: Proc. of the 28th Symp. on Mass Storage Systems and Technologies (MSST). 2012. 117−129. [doi: 10.1109/MSST.2012.6232377]

[23] Meyer DT, Bolosky WJ. A study of practical deduplication. In: Proc. of the 9th USENIX Conf. on File and Stroage Technologies. Berkeley: USENIX, 2011. 412−425.

[24] Zhang JL, Chang YL, Shi W. Overview on label propagation algorithm and applications. Application Research of Computers, 2013,30(1):21−26 (in Chinese with English abstract). [doi: 10.3969/j.issn.1001-3695.2013.01.004]

附中文参考文献:

[1] 付印金,肖侬,刘芳,鲍先强.基于重复数据删除的虚拟桌面存储优化技术.计算机研究与发展,2012,49(S1):125−130.

[7] 敖莉,舒继武,李明强.重复数据删除技术.软件学报,2010,21(5):916−929. http://www.jos.org.cn/1000-9825/3761.htm [doi: 10.3724/SP.J.1001.2010.03761]

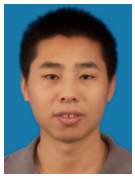
[8] 付印金,肖侬,刘芳.重复数据删除关键技术研究进展.计算机研究与发展,2012,49(1):12−20.

[15] 孙竞,余宏亮,郑纬民.支持分布式存储删冗的相似文件元数据集合索引.计算机研究与发展,2013,50(1):197−205.

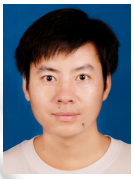
[24] 张俊丽,常艳丽,师文.标签传播算法理论及其应用研究综述.计算机应用研究,2013,30(1):21−26. [doi: 10.3969/j.issn.1001-3695.2013.01.004]

张沪寅(1962－),男,江苏苏州人,博士,教授,博士生导师,主要研究领域为高性能计算.

陈毅波(1982－),男,博士,主要研究领域为个性化推荐.



周景才(1982－),男,博士生,CCF 学生会员,主要研究领域为云计算,高性能计算.



查文亮(1988－),男,博士生,CCF 学生会员,主要研究领域为云计算,高性能计算.