分　数：	
评卷人：	

華中科技大學

研究生（数据中心技术）课程论文（报告）

题 目：关于数据中心上数据去重技术的优化

学　　号 M202073400

姓　　名 王　丰

专　　业 电子信息

课程指导教师 施展 童薇

院（系、所） 计算机科学与技术学院

2020 年 12 月 20 日

关于数据中心上数据去重技术的优化

王丰[1)]

[1)](华中科技大学计算机科学与技术学院 湖北 武汉 430074)

摘 要 随着现有电子信息和数据的爆炸式增长，数据占用的空间越来越大。全球年新增数据 2015 年为 8ZB，2018 年为 33ZB，预计到了 2025 年，全球年新增数据会猛增到 180ZB，也就是 1800 亿 TB，全球数据领域的扩张是无止境的。同时，微软和 EMC 的研究表明他们的存储系统中有 50%到 80%的数据是冗余的；而美因资大学和普渡大学分别研究了欧洲四个高性能数据中心和数据中心的 525 个虚拟机镜像，结果显示欧洲四个高性能数据中心至少有 20%的数据冗余，数据中心的数据冗余率约为 60%。这些数据表明数据存储系统有很大的冗余数据，数据去重能够为数据存储系统节省更多的存储空间。

数据去重是一项删除数据系统中的冗余数据从而节省存储空间的技术，目前已经广泛运用于很多大型存储数据系统。数据去重过程可以分为 4 个阶段：（1）、数据分块，（2）、计算数据块指纹，（3）、数据块指纹索引，（4）、存储数据块和元数据。

在本文中，我们对数据去重技术在数据中心中分布式存储系统上的应用、数据分块的加速、数据还原的加速三个方面的相关研究进行了概述。对数据去重技术的发展情况进行了简单的叙述。

关键词 数据去重；数据分块；分布式存储系统；数据还原；数据中心

About the optimization of data deduplication technology on the data center

Wang Feng[1)]

[1)](School of Computer Science and Technology, University, Huazhong University of Science and Technology, Wuhan, Hubei 430074)

Abstract With the explosive growth of existing electronic information and data, data takes up more and more space. The global annual new data was 8ZB in 2015 and 33ZB in 2018. It is estimated that by 2025, the global annual new data will soar to 180ZB, which is 180 billion terabytes. The expansion of the global data field is endless. At the same time, studies by Microsoft and EMC show that 50% to 80% of the data in their storage systems are redundant; while Mainz University and Purdue University have studied four high-performance data centers and 525 data centers in Europe. The results show that the four high-performance data centers in Europe have at least 20% data redundancy, and the data redundancy rate of the data centers is about 60%. These data indicate that the data storage system has a lot of redundant data, and data deduplication can save more storage space for the data storage system.

Data deduplication is a technology that deletes redundant data in a data system to save storage space. It has been widely used in many large storage data systems. The data deduplication process can be divided into 4 stages: (1), data block, (2), calculate data block fingerprint, (3), data block fingerprint index, (4), store data block and metadata.

In this article, we summarize the relevant research in three aspects: the application of data deduplication technology in distributed storage systems in data centers, the acceleration of data block, and the acceleration of data restoration. The development of data deduplication technology is briefly described

Key words Data Deduplication; Data Chunk; Distributed Storage System; Data Restore; Data Center

1 绪论

1.1 概述

近年来随着物联网的发展与普及，人们身边可用的智能设备越来越多，实现万物互联的目标将产生海量的数据。据预测，2025 年全球个人智能终端数将达 400 亿，其中智能手机数将达 80 亿，平板电脑和 PC 数将达 30 亿，各类可穿戴设备数将达到 80 亿；平均每人拥有 5 个智能终端，20%的人将拥有 10 个以上的智能终端；近 200 亿实时在线的智能家居设备，将成为个人和家庭感知的自然延伸[1]。每天人们与智能设备的互动都将产生大量数据，根据相关预测，全球每年的新增数据从 2015 年的 8ZB，猛增到 2025 年的 180ZB，也就是 1800 亿 TB[1]，将近九成的数据都在近两三年产生并且积累。显然，怎么样高效地管理这些海量的数据是现在数据存储系统所面临的挑战。

表 1.1 各研究机构关于存储系统中冗余数据的调查结果

研究机构	数据来源	数据规模	冗余数据/原始数据量
微软	857 个桌面文件系统	162TB	约 42%
	15 个微软服务器文件系统	6.8TB	15%-90%
EMC	约 1 万个商用备份存储系统	700TB	69%-93%
美因茨大学	欧洲四个高性能计算数据中心	1212TB	20%-30%
普渡大学	数据中心的 525 个虚拟机镜像	-	约 60%

表 1.1 显示了工业界和学术界对于不同存储系统的冗余数据负载的调查结果。2011 年微软研究院[2,3]对 857 个桌面文件系统进行了冗余数据负载的调查，发现用户的文件系统中有将近 40%的冗余数据；对于用户之间的共享数据，冗余数据占比更是高达 68%。微软研究院[2]又在 2012 年公布了微软桌面软件服务器文件系统的冗余数据情况，发现其的冗余数据负载要比微软研究的桌面文件系统更高，约为 15%-90%。因此，2012 年，微软公司[4]为提高存储效率在新推出的 Windows Server 8 操作系统中添加了数据去重功能。2012 年 EMC[5,6]也针对约 1 万个商用备份存储系统进行了冗余数据的研究，结果表明他们的存储系统中有 69%到 93%的数据是冗余数据。

在学术界，德国美因资大学[7]调查了欧洲四个高性能数据中心，结果显示在科学计算这种应用场景，存储系统也有 20%-30%的数据冗余；同时，普渡大学[8]展示了对于虚拟机这一应用场景的数据冗余调查，发现 525 个虚拟机文件系统的数据冗余率约为 60%。

上述研究结果表明，现在的大规模存储系统中存在着大量冗余数据，消除这些冗余数据，不仅能帮助提高存储效率，节约有限的存储空间，降低存储设备的支出成本；还能减少在网络中传输数据的大小，避免占用宝贵的网络带宽；同时，对于用户而言，自然就减少了数据的传输时间，降低了购买更多存储硬件的成本。对于企业和个人用户而言，减少这些冗余数据存储，就能减少了其存储成本；而减少冗余数据的传输，就相当于无形中提高了带宽。因此，众多 IT 公司都在探索冗余数据技术来提高其企业信息化工作效率，这一情况也在 2011 年 IDC[9]的研究报告中得到了证明，其中 80%被调查的公司都在使用数据压缩技术来消除企业 IT 系统中的冗余数据。

1.2 数据去重

数据去重作为数据压缩的一种，可以理解为是一种无损压缩。对于重复的数据，它只会存储一份，其余的副本用一个指针指向真实存储数据的地址。在数据去重技术出现之前，人们使用传统的数据压缩来进行冗余数据消除。比如在网页上进行图片上传时，网站一般都会对其进行一次压缩，这样既加快了上传的速度，也节约了服务器的存储空间。IBM 研究院[10]2013 年也进行了一项类似研究，发现传统的经典压缩技术 DEALATE[11]可以节省约 18%-53%的存储空间。但是传统的压缩方法（比如 lz4、lz5）都是字节粒度级别的压缩，数据去重作为一种处理粒度更大的冗余数据消除方法，更适合用于现代海量数据存储中，并且大部分存储系统也会将数据去重后的数据在采用传统压缩进行进一步处理，争取节省出更多的存储空间。

数据去重可以分为文件级去重和块级去重两种。文件级去重是以文件为基本去重单位的方法。文件级去重首先对文件进行一次哈希计算（比如SHA1、SHA256）得到指纹，然后检查备份文件索引来比较文件的属性，如果存在相同的文件，则会添加一个指向现有文件的指针，否则的话，它会更新并且存储索引值。因此该文件只有一个实例被存储，也被称为单实例存储。对整个文件的哈希计算方法很容易使用。由于文件哈希值很容易生成，文件级去重对运算能力的要求比较低。但是文件级去重方法有一定的局限性，任何较少的字节变动甚至是单个字节变动都会导致哈希值的变化，从而被认为是与之前不同的文件并要被重新存储。

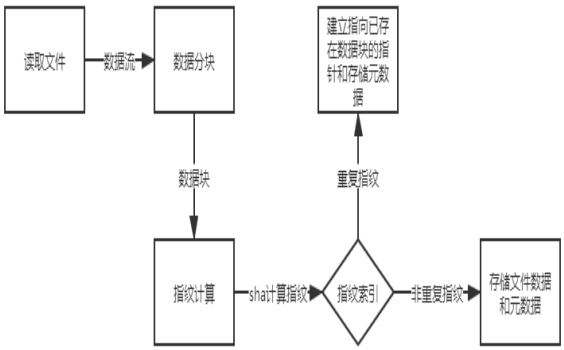

图 1.1 数据分块流程

同文件级去重不同，块级去重以数据块为基本单位来进行重复数据的检测。对于大文件，块级去重系统直接对其进行分块，而对于大量小文件，块级去重系统一般会先将小文件进行打包，组装成一个大文件，然后再进行重复数据的检测。如图 1.1 所示，一般块级去重过程可以分为 4 个阶段：（1）、数据分块，（2）、数据块指纹计算，（3）、指纹索引及去重，（4）、数据与元数据存储。如图 1.1，数据流在数据分块阶段被分割成较小的数据块，再在计算指纹阶段对各个数据块取指纹，然后在索引查询阶段通过匹配数据块的指纹辨识出重复的数据块，最后将不重复的数据块进行写入，而重复的数据块无需写入。显然块级去重因为更小的处理粒度，要比文件级去重检测到更多重复数据，因此被广泛用于现有的数据去重系统中。如果没有特别说明，本文后面提到的数据去重都是基于块级的数据去重。

2 分布式存储系统上的全局重删设计

2.1 研究背景

对于现有的数据中心来说，其主存储系统一般都是分布式存储系统，并且为了避免单点故障，常见的分布式存储系统比如 Ceph 是没有一个中心化的元数据服务器（Metadata Server）的，然而在之前提到的数据去重流程是单机上的数据去重，它的指纹索引部分的实现需要一个集中式的元数据存储管理服务器，这样的话之前的数据去重流程特别是指纹索引的部分很难在分布式系统上实现。

Myoungwon 等人在 2018 年提出了一个在分布式存储系统上的全局重删设计[12]，这个设计是基于目前使用率较高的 Ceph 分布式存储系统实现的，其在保持 Ceph 原有架构的情况下实现了数据内容和存储位置的关联（Content-Addressable Storage），将全局重删技术在分布式存储系统上进行了重新设计与适应，提出了双重哈希和去重速率控制两个方面。

2.2 双重哈希

分布式存储系统 Ceph 与传统数据去重存储系统不同的地方是，传统数据去重系统将分块（Chunk）的哈希指纹与分块的位置信息一同保存在指纹的索引结构里，当查询一个分块的哈希指纹时就能直接得到这个分块的位置信息；然而在分布式存储系统中对象（Object）的存储位置由一个分布式哈希算法（如 CRUSH）来决定，这个算法的输入是对象的 id。因此，传统数据去重存储系统根据分块的哈希指纹来得到数据分块的存储位置，而分布式存储系统的根据对象的 id 来得到对象的存储位置。

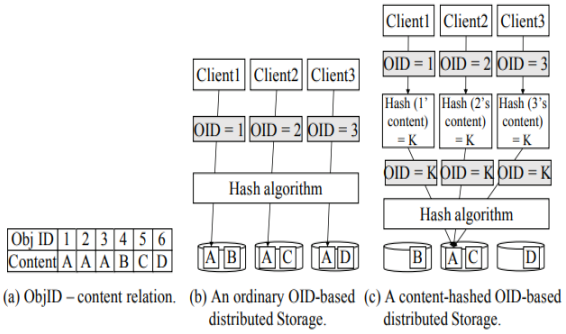

图 2.1 双重哈希设计

双重哈希（Double hashing）通过将上述所说的分块哈希指纹与对象 id 进行组合来实现分布式存

储系统上的全局重删设计。图 2.1-(a)显示了对象 id 与数据内容的关联，可以看到对象 id 为 1、2、3 的对象拥有相同的内容。图 2.1-(b)显示了一个普通的分布式存储系统的对象存储方式，每一个对象都可以根据它的对象 id 通过预先定义的哈希算法（比如 CRUSH）找到其数据内容存储的位置。但是，其对象 id 和对象的数据内容是没有任何关联的，在这个情况下，相同的数据内容会在多个存储节点同时存在；这个时候，如果要使用传统的数据去重方法在所有存储节点上找到相同的数据内容会导致极大的指纹索引开销。图 2.1-(c)使用了一次额外的哈希计算来将原本的对象 id（object ID 1，2，3）转化为对象内容（object K）作为哈希算法的输入值，因此分布式存储系统在对象数据内容和存储位置之间建立了关联。

这个算法的优势在于其直接去除了数据去重中指纹索引部分在分布式存储系统上带来的额外开销，而且保留了分布式存储系统原有的可扩展性，并且不需要客户端进行额外的适配操作。

2.3 去重速率控制

为了最小化后台的数据去重过程对于分布式存储系统带来的性能影响，其设计采用了去重速率控制并且实现了选择性去重。在分布式存储系统上的全局重删设计中，后台数据去重线程定期地执行一次数据去重任务，并且这个线程的后台 I/O 是受到速率控制的；同时，数据去重线程会根据数据的状态来区分热数据与冷数据，其只对冷数据进行数据去重来避免对热数据去重导致的读取与写入的延迟增加。

内联（inline）重复数据删除具有节省空间的优势。然而，由于内联处理，它不可避免地需要额外的等待时间。例如，在分布式存储系统中，重复数据删除中的元数据查找和数据处理在网络中进行。因此，延迟可能比本地节点上的元数据和数据处理的延迟更长。在内联处理中，此开销可能很大，因为系统需要立即进行重复数据删除。因此，内联处理很难保证分布式存储系统的性能。另一方面，通过使用后台（offline）处理，我们可以获得两个主要好处。第一，通过 I/O 速率控制，可以保证分布式存储系统恒定的吞吐量。第二，后台（offline）处理可以避免读取和写入的延迟问题，因为前台 I/O 不需要进行额外的操作，而且后台重复数据删除线程是在后续过程中才执行重复数据删除作业。但是，前台 I / O 作业还是可能会受到后台重复数据删除任务的干扰。因此，我们采用了速率控制技术，其可以将前台 I / O 干扰降到最低。并且，它可以避免经常修改的对象（热数据）被执行重复数据删除操作。在后台处理中，后台重复数据删除线程读取数据，然后执行重复数据删除过程。因此，我们可以控制是否对热数据进行重复数据删除。

2.4 实现效果

该论文在 Ceph 12.0.2 上实现了目标方法。在实验中，一个 Ceph 集群由四个服务器节点组成，每个服务器具有 Intel Xeon E5-2690 2.6Ghz（12 个内核），128GB 的 RAM 和四个 SSD（SK Hynix 480GB）。

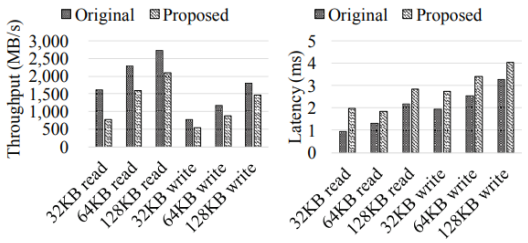

图 2.2 双重哈希设计

如图 2.2 所示，在读取的情况下，块大小较小时，与原始文件相比，性能会降低一半。这是因为在较小的块大小（所需的等待时间较短）的情况下，重定向（从元数据池到块池）的开销会增加。但是，当块大小较大时，开销相对减少。在 128KB 的情况下，并行向块池请求 32KB 的块，从而提高了吞吐量和延迟。理想情况下，读取性能应与原始性能相似。但是，由于重复数据删除计数会导致碎片化（顺序读取变为随机读取），因此性能降低。我们的设计无法完全消除这种开销。但是，使用高速缓存管理器，可以防止严重的性能下降，因为在元数据池中处理了热对象。测量写入性能。由于重复数据删除在元数据池中以恒定的速率执行，因此与原始性能相比，只有有限的性能下降，而与客户端请求的块大小无关。

表 2.1 ：分布式存储系统去重率

	16KB	32KB	64KB
理想去重率 (%)	46.4	44.8	43.7
存储数据 (TB)	1.82	1.88	1.89
存储元数据 (GB)	163	82	41
真实去重率 (%)	41.7	42.4	43.3

表 2.1 显示了基于块大小的重复数据删除率的结果。 理想的重复数据删除率意味着仅有数据的

重复数据删除率。 可以看出，重复数据删除率随着块大小的增加而降低。通过包括附加到数据的元数据的大小来计算实际重复数据删除率。 对象的存储除了数据本身还需要元数据对象和块对象的元数据。此附加元数据的大小会随着块大小的变小而按比例增长。 因此，尽管最小的块大小显示了最高的重复数据删除率，但其实际的重复数据删除率却最低。 另一方面，如果块大小较大，则重复数据删除率将降低，这将再次降低总体重复数据删除率。

3 数据去重的算法加速

3.1 研究背景

之前提到的分布式存储系统上的全局重删设计为了尽可能避免引入数据去重技术带来的额外开销，所使用的数据分块算法是计算简单的定长分块算法（Fixed-size Chunk），其根据预先设定的块长大小对数据流进行直接分块，计算开销小，但是存在文件偏移问题。

与定长分块不同，基于内容的分块（Content-Defined Chunk）算法解决了文件偏移问题。最经典的 CDC 算法是华盛顿大学于 2000 年提出的 Rabin 分块算法[13]，Rabin 分块算法相对于之前的定长分块，比较好地解决了文件边界偏移的问题。 Rabin 分块算法使用了一个滑动窗口来处理输入数据流的内容，这个滑动窗口会负责 Rabin 指纹的计算。

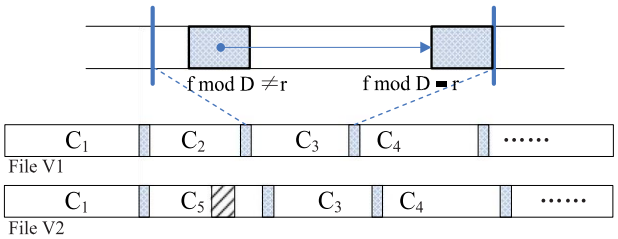

图 3.1 边界偏移问题

在分块时，Rabin 使用一个窗口 w 去计算该窗口内容的 Rabin 指纹，如果 Rabin 的指纹 f 与预先设定好的值 D 的模的值为另一个预先设定的值 r，那么这个窗口的位置就会被看作一个合适的切点，或者说数据分块的边界，否则的话继续按字节为单位往下移动。可以看图 3.1，文件 V1 可以被分为数据块 C1、C2、C3、C4，在 C2 分块插入一小段新数据后，得到了文件 V2，这时候虽然文件有变化，但是并没有出现文件的边界偏移问题，由于 Rabin 分块算法根据符合条件的切点作为分块条件，所以虽然 C2 分块出现了变化，但是并没有影响到插入数据之后的 C3、C4 分块。Rabin 分块算法解决了数据边界偏移的问题，去重率要比定长分块提高很多。

虽然基于内容的分块算法比定长分块的去重率要更高，但是其存在的一个问题是计算开销大，这一点是之前提到的数据中心上的分布式存储系统所不能接受的。显然，分布式存储系统需要一个计算开销更小的基于内容分块算法。

Wen xia 等人在 2020 年提出了 FastCDC[14]这一基于内容的分块算法，其大大减少了数据去重中数据分块过程的计算开销，同时保证了数据去重率。

3.2 设计与实现

FastCDC 的设计目的是实现更高去重性能的基于内容的分块（Content-Defined Chunk）算法。衡量 CDC 算法性能的指标有三个：

（1） 去重率：去重后数据大小与原数据大小的比值

（2） 分块速度：分块算法的吞吐量

（3） 平均去重块长：平均去重块长反映了去重阶段的元数据开销，越大的平均块长意味着越少的分块，也就是说去重需要处理的元数据也越少。

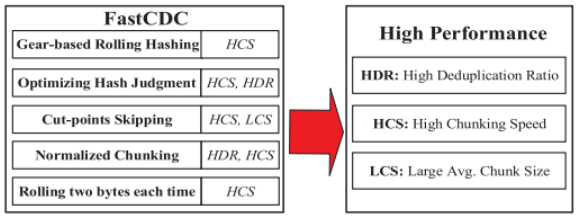

图 3.2 FastCDC 使用的五个关键技术

显然，这三个指标是冲突的，比如说更小的平均块长意味着更高的去重率，但是同时也会因为元数据处理的增多导致分块速度的下降。FastCDC 使用图 3.2 提到的五种技术来在这三个性能指标之间取得比较好的平衡。

3.2.1 Gear-Based 滚动哈希

表 3.1 各研究机构关于存储系统中冗余数据的调查结果

名字	伪代码	速度
Rabin	$fp = \left(\left(fp^{U(a)}\right) \ll 8\right) \lvert b^{T[fp \gg N]}]$	慢
Gear	$fp = (fp \ll 1) + G(b)$	块

Gear-Based 滚动哈希是 Ddelta[15]第一次提出用于数据差量压缩的哈希算法，同时其也被建议用于基于内容分块的候选滚动哈希算法。由于其哈希的

简单性和滚动有效性，Gear 哈希被证明是 CDC 最快的滚动哈希算法之一。可以看到表 3.1，Gear 哈希相对于传统的 Rabin 分块算法需要的计算次数更少，因此其的分块速度相对于 Rabin 算法有着很大的提升。

3.2.2 优化哈希判断

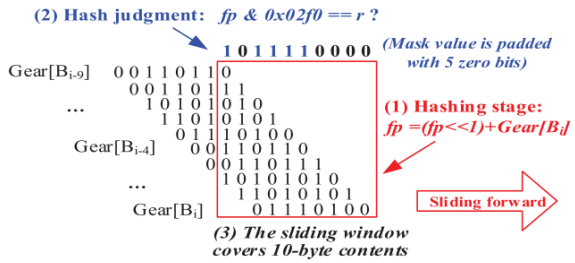

图 3.3 零填充和优化哈希判断

在本小节中，我们在基于 Gear 的 CDC 上优化哈希判断阶段，这有助于进一步加速分块过程并提高重复数据删除率，从而达到基于 Rabin 的 CDC 的重复数据删除率。

首先，FastCDC 通过零填充技术来增大滑动窗口的大小。基于 Gear 的 CDC 采用与基于 Rabin 的 CDC 相同的常规哈希判断，其中使用指纹的一定数量的最低位来判断块切割点。FastCDC 通过将多个零位填充到掩码值中来扩大滑动窗口的大小。如图 3.3 的示例所示，FastCDC 将五个零位填充到掩码值中，并将哈希判断语句更改为“ fp&mask==r”。如果 fp 的掩码位与阈值 r 匹配，则将当前位置声明为块切割点。由于 Gear 哈希使用一个左移和一个加法运算来计算滚动哈希，因此这种零填充方案使 10 个字节（即 B_i; ...;B_{i-9}）而不是原始的五个字节参与到最后的哈希判断，因此使分块位置冲突的可能性最小。结果表示 FastCDC 能够实现与基于 Rabin 的 CDC 分块算法一样高的重复数据删除率。

其次，在基于 Rabin 的 CDC 中使用的常规哈希判断为“$fp \bmod D == r$”。在 FastCDC 中，当与上面介绍的零填充方案结合使用时，如图 xx 所示，哈希判断语句可以优化为“$fp \,\&\, mask == 0$”，等效于“$!fp \,\&\, mask$”。因此，FastCDC 的哈希判断语句减少了用于存储阈值 r 的寄存器空间，并避免了将“$fp \,\&\, mask$”和 r 之间的不必要的比较操作，从而进一步加快了 CDC 的过程。

3.2.3 切点跳过

大多数基于内容分块（CDC）的重复数据删除系统都对数据块的最大最小块长进行了限制来避免产生很多很大或者很小的分块的情况。根据公式 $P(X \le x) = F(x) = \left(1 - e^{-\frac{x}{8192}}\right), x \ge 0$可以知道小于 2KB 和大于 64KB 的块分别占总块数的 22.22%和 0.03%。这说明加入最大块长限制对重复数据删除率的影响非常小，但是加入最小块长限制，或者说次最小块切点跳过，将会显著得影响重复数据删除率。鉴于 FastCDC 的目标是最大程度地提高分块速度，扩大最小块大小和跳过次最小分块切入点，将避免在跳过区域进行哈希计算和判断操作，从而帮助 FastCDC 达到更高的分块速度。但是，这种速度的提高是以降低重复数据删除率为代价的。为了解决这个问题，我们将开发一种标准化的分块方法，将在下一节中介绍。

3.2.4 标准化分块

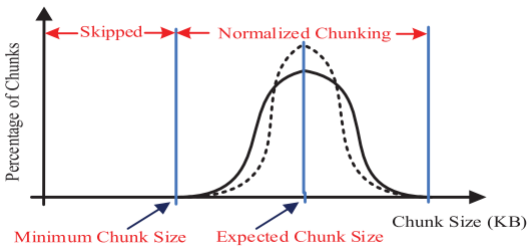

图 3.4 归一化块长分布情况

本小节提出了一种新颖的分块方法，称为归一化分块，以解决面向切点跳过方法的重复数据删除率降低的问题。如图 3.4 所示，归一化分块生成的块都是大小被归一化到期望块长的指定区域的分块块。归一化分块后，几乎没有大小小于最小块大小的块，这意味着归一化分块可以跳过次要分块的切点，以减少不必要的分块计算，从而加快分块速度。

在归一化分块的实现中，FastCDC 使用了两个阈值来进行哈希判断，在分块长度小于期望块长的时候，选择一个更难获得切点的阈值 Mask1 来进行哈希判断，在当前分块长度大于期望块长时，使用一个更容易获得切点的阈值 Mask2 来进行哈希判断，这样使得生成的块尽可能接近我们预先设定的期望块长大小。

归一化分块有三个好处：

a) 如图 3.4 所示，归一化分块减少了小块的数量，这使其可以与切点跳过方法结合使用，从而在不牺牲重复数据删除率的情况下实现了更高的分块速度。

b) 归一化分块通过减少大块的数量进一步

提高了重复数据删除率，从而弥补了FastCDC中由于减少小块的数量而导致的重复数据删除率的降低。

c) FastCDC的实现不会添加其他计算和比较操作。 它只是将哈希判断分为预期的块大小之前和之后的两部分。

3.2.5 每次滚动两个字节

通过在哈希阶段减少每次哈希运算的运算量，可以进一步加快分块速度，方法是每次滚动两个字节以计算哈希阶段的分块指纹，然后分别判断偶数和奇数字节。

3.3 实验结果

（1）实验环境。 为了评估FastCDC，我们在Ubuntu 18.04.1操作系统上实现了重复数据删除系统，该操作系统在2.1 GHz的IntelXeon®Gold 6130处理器上运行，具有128 GB RAM。 为了更好地评估分块速度，还使用了另一个3.2 GHz的Intel i7-8700处理器进行比较。

表 3.2 重复数据删除率评估

Dataset	FIXC	RC-v1	RC-v2	RC-v3	AE-v1	AE-v2	FC-v1	FC-v2	FC-v3
TAR	15.77%	46.66%	47.42%	45.37%	43.62%	46.41%	46.65%	47.39%	45.40%
LNX	95.68%	96.30%	96.28%	96.19%	96.25%	96.13%	96.31%	96.28%	96.19%
WEB	59.96%	75.98%	83.16%	80.39%	83.08%	83.18%	83.20%	83.29%	80.92%
VMA	17.63%	36.70%	37.79%	36.52%	38.10%	38.17%	36.40%	37.66%	36.39%
VMB	95.68%	96.12%	96.17%	96.11%	95.82%	96.15%	96.08%	96.17%	96.11%
RDB	16.39%	92.57%	92.96%	92.24%	88.82%	92.83%	92.58%	92.97%	92.23%
SYN	79.46%	97.36%	97.91%	97.67%	97.54%	97.86%	97.37%	97.90%	97.67%

表 3.3 生成平均块长大小评估

Dataset	FIXC	RC-v1	RC-v2	RC-v3	AE-v1	AE-v2	FC-v1	FC-v2	FC-v3
TAR	10239	12449	12664	14772	12187	12200	12334	12801	14918
LNX	6508	6021	7041	7636	6274	6162	6012	7042	7636
WEB	10240	11301	12174	14148	11977	11439	11552	11880	13951
VMA	10239	13071	13505	15628	13098	13559	13150	13595	15746
VMB	10239	11937	12970	15094	12303	12254	12138	13034	15166
RDB	10239	10964	12587	14728	11943	12102	10970	12583	14725
SYN	10240	11663	12221	14271	11956	11997	11598	12239	14289

本小节中结合了基于齿轮的滚动哈希，优化哈希判断，切点跳过，每次滚动两个字节以及归一化分块五项关键技术来全面评估FastCDC的性能。测试了十二种CDC方法：

1) RC-v1（或RC-MIN-2 KB）是LBFS中使用的基于Rabin的CDC [6]； RC-v2和RC-v3指的是使用最小化大小分别为4 KB和6 KB的归一化分块的基于Rabin的CDC。
2) FC-v1是FastCDC，它使用优化哈希判断和切点跳过的技术，最小块大小为2 KB；FC-v2和FC-v3指的是使用全部四种技术的FastCDC，它们的最小块大小分别为6 KB和8 KB。
3) FC′-v1，FC′-v2和FC′-v3是FastCDC，使用分别在FC-v1，FC-v2和FC-v3顶部每次滚动两个字节的技术。
4) AE-v1和AE-v2指的是基于AE的CDC [16]及其优化版本[17]。
5) FIXC指定长分块算法，并使用10 KB的平均块大小进行分区。

表3.2中的评估结果表明，在大多数情况下，FC-v1，FC-v2，AE-v2和RC-v2的重复数据删除率几乎与RC-v1相同，这表明归一化分块方案在Rabin和FastCDC都有较好的效果。另外，FIXC在数据集LNX和VMB上去重率较高，因为LNX的许多文件都小于固定大小的10 KB块（因此，平均生成的块大小也小于10 KB），并且VMB具有许多结构

化备份数据（因此 VMB 适用 FIXC 算法）。

表 3.3 显示 RC-v1，RC-v2，AE-v1，AE-v2，FC-v1 和 FC-v2 产生相似的平均块大小。但是 RC-v3 和 FC-v3 的方法具有更大的平均块大小，这意味着它生成的块更少，因此用于重复数据删除处理的元数据也更少。同时，RC-v3 和 FC-v3 仍可达到相当的重复数据删除率，如表 4 所示，略低于 RC-v1，但同时提供了更高的分块速度。

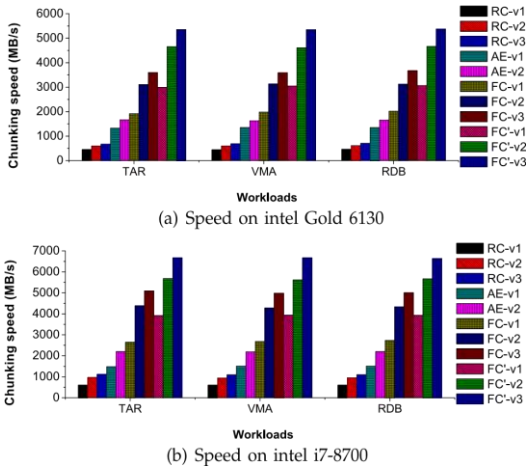

图 3.5 11 个 CDC 算法的分块速度

图 3.5 表明 FC'-v3 具有最高的分块速度，比基于 Rabin 的方法快约 12 倍，比 FC-v1 快约 2.5 倍。这是因为 FC’-v3 是使用所有五种技术来加快分块阶段的最终版本 FastCDC。 此外，FC’-v2 也是分块算法的不错选择，因为它具有比较好的重复数据删除率同时，也在分块速度和平均生成块大小的其他两个指标上表现得比较好。 另外，归一化分块还有助于加速基于 Rabin 的 CDC（即 RC-v2 和 RC-v3），同时实现比较好的重复数据删除率和平均块大小。 但是这种加速是有限的，因为基于 Rabin 的 CDC 的主要瓶颈仍然是滚动哈希计算。

4 数据还原的加速研究

4.1 研究背景

由于重复数据删除之后，数据块分散在许多不同的容器中，数据恢复时会频繁读取容器，因此数据块碎片会严重阻碍数据恢复性能，造成 I/O 性能的下降，这导致了读放大和数据碎片问题。对于数据中心的分布式存储系统，每秒请求的数据访问量是非常大的，同时，如果要对去重后的数据进行请求，那么首先重复数据删除系统要对其进行还原，这会大大提高访问延迟和用户等待时间，为了解决读放大和数据碎片问题，使用一个能够快速还原的数据还原算法是非常有必要的。

Cao 等人在 2019 年提出了一个使用可变计数引用重写方案和滑动回溯窗口的数据还原方案[18]，其在数据还原速度和数据去重率上取得了较好的平衡，在保证数据去重率的同时大大提高了数据去重系统的还原吞吐量。

4.2 设计思路

4.2.1 可变容器计数引用重写方案（Flexible Container Referenced Count）

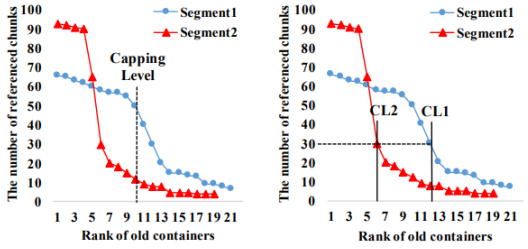

图 4.1 可变容器计数引用与 capping level

之前的 capping 重写方案[19]将原始数据流被划分为固定大小的段，并且每个段具有 N 个容器的大小（例如 N •4MB）。该方案的目标是限制将数据段恢复到 T +C 所需的容器读取次数。C 是对数据段进行重复数据删除时生成的新容器的数量，T 是 capping 方案的设置的 capping level。因此，capping level 限制了将读取以加载重复块的容器的数量。在 capping 中，第一步是计算属于旧容器（称为容器引用计数（CNRC））的段中的数据块（包括重复项）的数量。然后，将这些至少包含该段中重复数据块的旧容器及其 CNRC 降序排序。如果容器的排名低于 T，则该容器中的重复数据块将与唯一数据块一起写入当前活动的容器中。这样，capping 可确保每个段中容器读取次数的上限。

Capping 使用固定的 capping level 来控制所有段的重写策略，以便可以获得更高的容器读取次数。但是，在该方案中，重复数据删除率不被视为优化目标。我们发现，通过对 Capping 方案进行一些更改，可以在容器读取数和重复数据删除率之间实现更好的取舍。如果我们考虑某个段中涉及的旧容器的数量以及这些旧容器的 CNRC 分布，我们发现不同段的旧容器的数量和 CNRC 的分布可能会

非常不同。如果我们希望限制重复数据删除率的降低，则某些分段可以保留的容器读取次数少于目标上限，而其他区段可能必须超出目标上限。例如，在一个段中，按照旧容器的降序对旧容器进行排序后，我们可以绘制出这些容器的 CNRC 分布，如图 4.1-(a)所示。 X 轴是容器的等级，Y 轴是所涉及容器的引用数据块的数量，不同的细分具有不同的分布。考虑图 4.1-(a)中的分段 1 和分段 2 的 capping level 为 10，则 capping level 右侧的容器中的所有重复数据块都必须重写。

在示例中，这两个段的容器读取数以 20 加新生成的容器数为上限。对于分段 1，从 10 到 12 排名的容器具有相对大量的引用数据块。在这些容器中重写数据块的副本将导致重复数据删除率的进一步降低。根据图中分布，分段 1 的理想上限水平应为 12 或 13。对于分段 2，由于数据块更多地集中在较高级别的容器中，因此，在排名超过容器 6 或 7 而不是容器 10 的容器中重写重复的数据块将减少更多的容器读取次数，而重写的重复数据块的数量将比较少地增加。因此，如图 4.1-(b)所示，如果我们使用 12 作为分段 1 的新 capping level（图中的 CL1），使用 6 作为分段 2 的新 capping level（图中的 CL2），则容器的数量读取次数将减少，而被重写的重复数据块的数量也将更少。因此，对不同的分段应用不同的 capping level 可以进一步减少容器读取并实现更少的数据块重写。但是，如何确定不同分段的“capping level”是一个有挑战性的问题。

为了解决使用固定上限级别的上述限制，我们提出了 capping 的改进：基于灵活容器引用计数的方案（FCRC）。FCRC 不会使用固定的上限级别作为选择阈值，而是使用 CNRC 的值作为新阈值。它从 CNRC 低于阈值的旧容器中重写重复的数据块。这样，不同的细分将具有不同的实际上限水平。实际的封顶级别由这些段的阈值和 CNRC 的分布决定。可以通过目标上限水平来估计 CNRC 阈值 T_{cnrc}。即，一个段中重复数据块（包括重复项的所有副本）的总数除以目标 capping level。从统计上讲，如果一个段中的每个读入容器可以贡献超过 T_{cnrc} 数量的重复数据块，则恢复该段的旧容器读取总数将受到目标 capping level 的限制。因此，在具有小于 T_{cnrc} 的 CNRC 的容器中重写重复数据块可以实现与在上限方案中使用相同的目标 capping level 相似的容器读取次数。

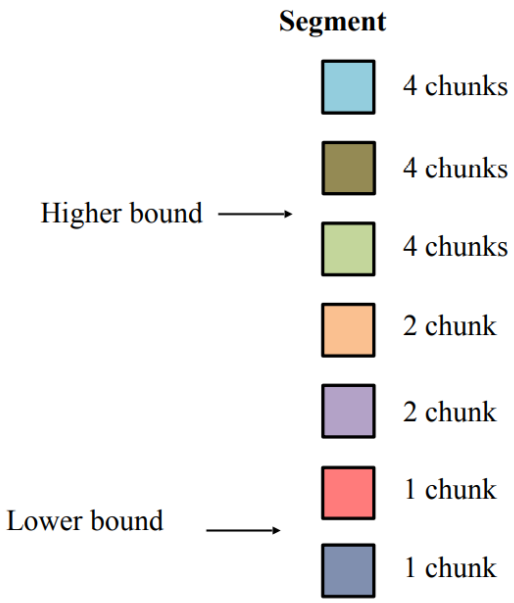

图 4.2 两个边界限制

如图 4.2 所示，灵活容器引用计数的方案（FCRC）提出了两个边界限制：去重率边界和容器读取次数边界，容器重写的阈值将会在这两个边界之间浮动，以此在数据还原速度和数据去重率之间取得更好的平衡。

4.2.2 滑动回溯窗口（Sliding Look-Back Window）

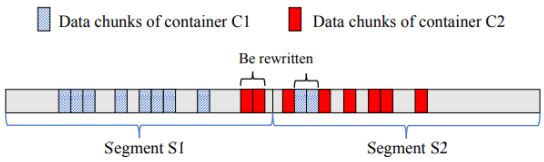

图 4.3 固定分段在 capping 中的问题

在 capping 方案和 FCRC 方案中，分段的边界附近的重复数据块的重写决策可能会有问题。如图 4.3 所示的示例。有两个连续的分段 S1 和 S2。容器 C1 的大部分数据块出现在 S1 的前面，而 C2 的大部分数据块出现在 S2 的前面。由于分段的分割，容器 C1 的一些重复块也位于 S2 的前面，容器 C2 的一些重复块也位于 S1 的末尾。根据 capping 方案，S1 中的 C1 的数据块的数量和 S2 中的 C2 的数据块的数量排列显然高于 capping level。这些块将不会被重写。但是，容器 C1 在 S2 中的排名和容器 C2 在 S1 中的排名超出 capping level。由于我们不知道在对 S2 进行重复数据删除时有关 S1 中 C1 的过去信息以及对 S2 在对 S1 进行重复数据删除时

S2 中有关 C2 的未来信息，因此这些块（即，S1 中来自 C2 的一些数据块和 S2 中来自 C1 的一些数据块）将被重写。当我们还原 S1 时，将在缓存中读取 C1 和 C2。恢复 S2 时，C2 已在缓存中，并且不会触发其他容器读取。因此，重写了出现在 S1 中的 C2 的数据块，并且浪费了 S2 中的一些 C1 的数据块。如上述示例所示，在 capping 和 FCRC 方案中，基于固定大小段的容器和数据块信息均不包含有关过去和将来段的信息。一方面，分段前面附近的数据块将和同一段中的后续块一起进行重写决策评估，但它们不会受到数据块引用信息和在前一个分段中进行的重写决策的影响。另一方面，可以使用分段中较早的块来评估接近段末尾的数据块，但是没有有关下一个可用分段的信息。如果其容器的大部分数据块出现在随后的几个段中，则接近该段末尾的数据块被重写的可能性更高。结果，仅根据当前分段的统计信息做出的重写决策不太准确。

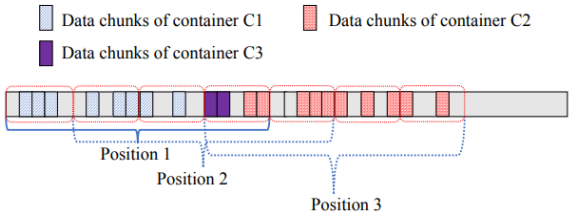

图 4.4 双重哈希设计

因此我们提出了滑动回溯窗口（Sliding Look-Back Window）来对分块的前后信息进行综合考虑，从而更好的做出重写决策。

如图 4.4 所示的简单示例。假设我们使用固定的参考计数值 4 作为阈值。也就是说，一旦一个旧容器在窗口中具有四个以上的重复数据块，则该容器中的重复数据块将不会被重写（非重写条件）。当 LBW 位于位置 1 时，来自容器 1、2 和 3 的数据块的数量分别为 8、2 和 2。基于非重写条件，来自容器 1 的块将不会被重写。来自容器 2 和 3 的新标识的数据块不能满足不可重写条件。我们将这些数据块添加到重写候选缓存中。当 LBW 移到位置 2 时，来自容器 2 的数据块的数量已经是 5，满足不重写条件。来自容器 2 的数据块将不会被重写。因此，丢弃了重写候选缓存中容器 2 的数据块。对于容器 3 中的数据块，我们仍然需要延迟才能做出重写决定。当 LBW 移到位置 3 时，来自容器 3 的数据块已经在 LBW 的末尾。此时，来自容器 3 的数据块的数量仍低于阈值。因此，来自容器 3 的两个数据块将被重写到活动容器中。同时，配方缓存中的相应元数据条目将使用活动容器信息进行更新。随着最旧的容器移出 LBW，相应的元数据条目将被写入存储。

滑动回溯窗口考虑了数据去重系统的还原时的缓存效应，并且利用了数据的局部性来最大程度上避免不需要的数据块的重写，大大提高了数据去重系统的去重率，减少了数据碎片的问题。

4.3 实验结果

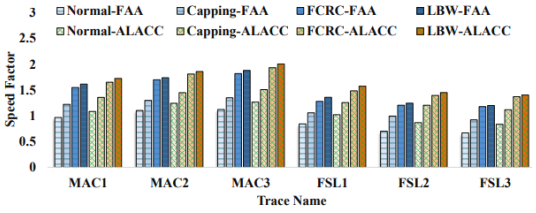

图 4.5 双重哈希设计

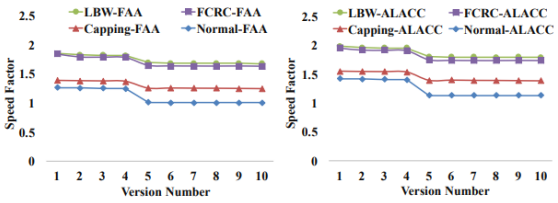

图 4.6 双重哈希设计

图 4.5 所示。如果在重复数据删除过程中未应用数据块重写（正常设计），即使使用 FAA[19]或 ALACC[20]作为还原缓存方案，速度因数始终低于 1.3。在三个 FSL 数据集中，速度因子甚至低于 1，这意味着读取 4MB 大小的容器甚至无法还原 1MB 的数据。这表明数据块是碎片化的，并且这两种缓存方案无法有效地减少大量的容器读取。通过重写重复的数据块，提高了速度因子。对于所有 6 个数据集，我们发现在相同的重复数据删除率下，当使用相同的还原引擎时，LBW 始终实现最高速度因子（最佳还原性能）。 FCRC 的速度因子低于 LBW，但高于 Capping。由于表 1 中所示的平均自引用计数比（ASR）较低，FSL 数据集的重复数据块比 MAC 数据集的数据块更加分散。因此，FSL 数据集的总体速度因子低于 MAC 数据集的速度因子。通常，当使用相同的还原缓存方案时，LBW 的速度因子比普通的速度因子高 97%，比 capping 的速度因子高 41%，比 FCRC 的速度因子高 7%。

我们还比较了数据集 MAC2 中不同备份版本的速度因素。 使用 FAA 和 ALACC 作为恢复缓存

方案的评估结果分别如图 4.6-(a)和图 4.6-(b)所示。在这 10 个版本中，重复的数据块在版本 4 之后更加分散。因此，版本 5 的速度因子明显下降。如两个图中所示，通过使用 FCRC，与常规模式相比，速度因子有了很大改善。 和 capping 方案与 FCRC 相比，LBW 通过解决由固定分段引起的重写精度问题，并考虑了缓存有效范围以及容器读取效率，进一步提高了速度因子。 因此，LBW 的速度因数在所有 10 个版本中始终是最高的。 通常，在重复数据删除率相同的情况下，LBW 可以有效减少容器读取的次数，从而使其还原性能始终是最佳的。

5 论文总结

近年来随着物联网的发展与普及，数据中心将存在越来越多冗余数据，重复数据删除技术作为消除冗余技术，节省存储空间和网络传输带宽的关键技术，将成为数据中心不可缺少的一部分。重复数据删除技术在分布式存储系统上的应用和性能提高是重复数据删除技术应用于数据中心技术的关键。本综述从重复数据删除技术在分布式存储系统上的设计、数据分块、数据还原三个方面进行了优化设计，提高了重复数据删除系统的吞吐量和去重率。

分布式存储系统上的全局重删设计提出了双重哈希（Double hashing）技术，实现了数据内容和存储位置的关联（Content-Addressable Storage）；同时提出了去重速率控制方案，实现了选择性去重的去重策略，最大程度上避免了去重任务对于分布式存储系统的延迟和性能影响。实现了分布式存储系统上的全局重删，其基于广泛应用的分布式存储系统 Ceph 的设计思路和实现方案具有很强的现实意义。

FastCDC 结合了（1）、Gear 哈希算法（2）、优化哈希判断计算（3）、最小块长切点跳过（4）、标准化分块（5）、窗口滑动两个字节等五种方案来提高基于内容分块算法的去重率和吞吐量，极大提高了重复数据删除系统的吞吐量并且保持了去重率。

Cao 提出了可变容器计数引用和滑动回溯窗口两个方案来解决数据还原过程中的数据碎片和读放大问题。并且对分块的前后信息进行综合考虑，从而更好的做出重写决策。与此同时还利用了数据还原过程中的缓存效应，极大地减少了数据恢复的读取次数，提高了数据恢复的 I/O 吞吐量。

参考文献

[1] 华为技术有限公司. GIV 2025: 打开智能世界产业版图[R/OL]. (2018-04-17) [2020-04-14]. http://www.huawei.com/minisite/giv.

[2] Meyer D T, Bolosky W J. A study of practical deduplication[J]. ACM Transactions on Storage, 2012, 7(4).

[3] Elshimi A, Kalach R, Kumar A, et al. Primary data deduplication-large scale study and system design[C]. usenix annual technical conference, 2012: 26-26.

[4] Microsoft corporation: Windows Server 8 data deduplication[R/OL]. http://research.microsoft.com/enus/news/features/deduplication-101311.aspx.

[5] Wallace G, Douglis F, Qian H, et al. Characteristics of backup workloads in production systems[C]. file and storage technologies, 2012: 4-4.

[6] Shilane P, Huang M, Wallace G, et al. WAN-optimized replication of backup datasets using stream-informed delta compression[J]. ACM Transactions on Storage, 2012, 8(4).

[7] Meister D, Kaiser J, Brinkmann A, et al. A study on data deduplication in HPC storage systems[C]. ieee international conference on high performance computing data and analytics, 2012: 1-11.

[8] Jayaram K R , Peng C , Zhang Z , et al. An empirical analysis of similarity in virtual machine images[C]. Middleware Industry Track Workshop. ACM, 2011.

[9] Dubois L, Amaldas M, Sheppard E. Key considerations as deduplication evolves into primary storage[J]. white paper, 2011.

[10] Harnik D, Kat R I, Margalit O, et al. To Zip or not to Zip: effective resource usage for real-time compression[C]. file and storage technologies, 2013: 229-242.

[11] Deutsch L P. DEFLATE compressed data format specification version 1.3[R/OL]. RFC Editor, http://tools.ietf.org/html/rfc1951, 1996.

[12] Oh M, Park S, Yoon J, et al. Design of global data deduplication for a scale-out distributed storage system[C]//2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS). IEEE, 2018: 1063-1073.

[13] Spring N, Wetherall D. A protocol-independent technique for eliminating redundant network traffic[C]. acm special interest group on data communication, 2000, 30(4): 87-95.

[14] Xia W, Zou X, Jiang H, et al. The Design of Fast Content-Defined Chunking for Data Deduplication Based Storage Systems[J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 31(9): 2017-2031.

[15] Xia W, Jiang H, Feng D, et al. Ddelta: A deduplication-inspired fast delta

compression approach[J]. Performance Evaluation, 2014, 79: 258-272.

[16] Zhang Y, Jiang H, Feng D, et al. AE: An asymmetric extremum content defined chunking algorithm for fast and bandwidth-efficient data deduplication[C]//2015 IEEE Conference on Computer Communications (INFOCOM). IEEE, 2015: 1337-1345.

[17] Zhang Y, Feng D, Jiang H, et al. A fast asymmetric extremum content defined chunking algorithm for data deduplication in backup storage systems[J]. IEEE Transactions on Computers, 2016, 66(2): 199-211.

[18] Cao Z, Liu S, Wu F, et al. Sliding look-back window assisted data chunk rewriting for improving deduplication restore performance[C]//17th {USENIX} Conference on File and Storage Technologies ({FAST} 19). 2019: 129-142.

[19] Lillibridge M, Eshghi K, Bhagwat D. Improving restore speed for backup systems that use inline chunk-based deduplication[C]//Presented as part of the 11th {USENIX} Conference on File and Storage Technologies ({FAST} 13). 2013: 183-197.

[20] Cao Z, Wen H, Wu F, et al. {ALACC}: Accelerating restore performance of data deduplication systems using adaptive look-ahead window assisted chunk caching[C]//16th {USENIX} Conference on File and Storage Technologies ({FAST} 18). 2018: 309-324.