RM-LCDF：一种块级连续数据保护高效数据恢复方法

王 超 李战怀 刘海龙 张小芳
（西北工业大学计算机学院 西安 710072）

摘 要 块级连续数据保护技术能够提供任意时刻的数据恢复，构建可靠数据存储环境，已成为现代存储系统重要的数据保护手段。数据的高可用性对数据恢复效率提出了更高的要求，针对传统块级连续数据保护机制数据恢复效率低的问题，结合数据块级写请求的集中分布特性和连续分布特性，提出了一种块级连续数据保护数据恢复机制——RM-LCDF。RM-LCDF采用去除无效写请求、多缓冲和逻辑块地址排序3种优化策略，对数据恢复过程进行优化。形式化分析及实验结果表明，RM-LCDF能够大幅度减少恢复过程中的I/O数据量，提高I/O并发度及写I/O吞吐率，进而有效提高恢复效率。

关键词 连续数据保护，数据恢复，块数据，可用性，存储系统
中图法分类号 TP311 **文献标识码** A

RM-LCDF: A Recovery Method for Block-level Continuous Data Protection

WANG Chao LI Zhan-huai LIU Hai-long ZHANG Xiao-fang
(School of Computer Science and Technology, Northwestern Polytechnical University, Xi'an 710072, China)

Abstract Block-level continuous data protection has become an important data protection technology for modern data storage systems. It can restore data to any point in time and support reliable storage. The high availability of computerized data has a raise requirement on the data recovery efficiency, but basic data recovery method for block-level continuous data protection is low. This paper presented a recovery method on the basis of the localized and continuous distribution features (RM-LCDF) for block-level continuous data protection. RM-LCDF reforms the basic data recovery method via three aspects: (1) Invalid write requests elimination, (2) Multi-buffer, and (3) Logical block address sorting. Both mathematical analysis and experiments show preliminarily that RM-LCDF can significantly reduce recovery data, improve I/O parallelism and I/O throughput, and thus improve the recovery efficiency.

Keywords Continuous data protection (CDP), Data recovery, Block-level data, Availability, Storage system

1 引言

随着信息技术的飞速发展、数据的爆炸性增长以及数据重要性的不断提高，必须采取必要的手段来确保存储系统的可靠性、可用性以及安全性。然而数据及其存储载体很容易受到软硬件失效、自然灾害、人为破坏以及病毒入侵等威胁，从而导致数据的损坏或丢失。相关研究[1,2]表明，由于数据不可用以及数据损毁所造成的经济损失超过百万美元每小时。数据保护已经成为商业组织、政府机构以及个人需要面对的重要问题[3]，因此，数据保护技术是非常必要的[4]。

在容灾理论中，RPO(Recovery Point Objective)和RTO(Recovery Time Objective)[1,5]是衡量数据保护方法的重要指标。RPO描述了灾难后可接受的数据丢失量，RTO描述了灾难后恢复数据所需的时间。根据RPO的不同，将目前常见的数据块级数据保护技术分为定期备份(Periodical Backup)、快照(Snapshot)和连续数据保护(CDP, Continuous Data Protection)3类。定期备份[6-8]机制简单，应用广泛，但通常需要在离线状态进行，不仅耗时，而且需要占用大量存储空间。与定期备份相比，快照技术[9]支持在线备份，可以提供弹性恢复窗口，但存在不超过备份窗口的数据损失。连续数据保护(Continuous Data Protection, CDP)技术则可以提供任意历史时刻点的数据恢复。

通常，块级连续数据保护机制[5,10]按照时间顺序保存各个数据块的所有更新，恢复时只需基于初始备份依次redo所有时间戳在恢复时间点之前的历史数据即可。但由于保存数据块的历史更新需占用大量的存储空间，限制了传统块级连续数据保护技术的应用。因此，大部分的研究集中于解决传统块级连续数据保护机制的高存储开销问题。

Morrey等[10]通过对网络块存储设备Peabody的写操作进行分析，发现写操作所更新的数据块中84%的数据与更新之前相同，因此可以通过消除相同的磁盘数据来提高存储空间利用率。Zhu等[11]则通过一种能够识别之前存储过的数

到稿日期：2012-08-26 返修日期：2012-11-02 本文受国家重点基础研究发展计划(973计划)课题(2012CB316203)，国家自然科学基金(61033007/F020204)资助。

王 超(1985－)，男，博士生，主要研究方向为海量数据存储、数据保护技术等，E-mail：wch@mail.nwpu.edu.cn；李战怀(1961－)，男，博士，教授，博士生导师，主要研究方向为数据库系统、数据管理等；刘海龙(1980－)，男，博士，讲师，主要研究方向为数据库系统、数据管理等；张小芳(1971－)，女，博士，副教授，主要研究方向为分布式计算、数据容灾、软件工程理论等。

据分片的高效存储架构来节省存储空间。这些技术通常需要在进行写操作之前进行搜索操作，尽管可以使用优化算法来提高搜索效率，但大量的搜索操作仍旧对 I/O 性能造成影响。Yang 等[12]提出了一种新的块级连续数据保护机制——TRAP-4，该机制通过压缩保存写操作前后数据的异或编码的方式，极大地节省了存储空间，但压缩和解压缩操作使得恢复效率明显降低，TRAP-4 以时间为代价换取了低存储空间。虽然 Lixu 等[13]通过在 TRAP-4 的恢复链条中插入快照的方法，在一定程度上提高了 TRAP-4 的恢复效率，但其由于增加了异或编码的解码时间开销，仍旧无法达到传统块级连续数据保护机制的恢复效率。

除了存储空间占用问题之外，传统块级连续数据保护机制还存在恢复效率低的问题。通常，随着恢复时间跨度的增大，传统块级连续数据保护机制在恢复时需要进行 redo 操作的历史数据也随之增长。面对高可用环境对数据可用性的要求，提高恢复效率，降低 RTO，就变得尤为重要。但对于块级连续数据保护机制，基于这方面的研究还比较有限。

最近的研究思路是通过快速识别有效恢复时间点来提高恢复效率。SWEEPER[14]通过对多种系统事件的监控分析，能够快速地识别最佳恢复时间点，从而降低恢复时间。侯等[15]提出的“基于数据差异的 CDP 邻近时间点恢复”，对于目标时间点相邻较近的两次恢复给出了一种快速恢复算法，一定程度上加速了块级连续数据保护机制有效恢复时间点的确认过程。但是这些研究都是从提高有效恢复时间点的识别效率的角度展开，而对于所识别出的单一确定时间点并没有给出较传统连续数据保护机制更有效的恢复方法。

此外，还有些研究思路是结合数据的分布特征提高快照的检索效率，如 THVFS[16]利用目录、文件版本之间的相关性提高快照检索效率；Skippy[17]利用内存快照 Hot Data 的特性，提出了一种分层次的索引结构来提高长期快照的检索效率；结合数据分布特征和检索模式的分层二维索引结构 HC-SIM[18]能够提高高频快照索引的存储效率和检索效率。这类技术尽管可以提高快照检索效率，进而提高恢复速度，但都是基于快照技术的相关研究，并不适合在数据块级的连续数据保护系统中使用。

鉴于目前对提高块级连续数据保护数据恢复效率方面的研究还比较有限，尤其是对单一确定时间点的数据恢复，本文以块级连续数据保护的基本数据恢复机制为基础，结合数据块级写请求的集中分布特性和连续分布特性，提出了一种高效的数据恢复方法，并进行了详细的实验验证及分析。本文的主要贡献如下：

(1)本文对 SPC 发布的 trace 数据进行了具体的统计分析，统计结果表明数据块级写请求具有集中分布和连续分布的双重特性，为优化块级连续数据保护的恢复机制提供了基础。

(2)结合数据块级写请求的集中分布特性和连续分布特性，从降低数据恢复过程中 I/O 操作数据量、提高 I/O 并发度和提高写 I/O 吞吐率 3 个方面入手，以块级连续数据保护的基本恢复方法为基础，提出了采用去除无效写请求、多缓冲、逻辑块地址排序 3 种策略的恢复方法——RM-LCDF。实验结果显示，RM-LCDF 所采用的 3 种策略均可有效提高恢复效率，与基本恢复方法相比，RM-LCDF 可以降低一个数量级以上的恢复时间。

2 数据块级写请求时空特性分析

不同的块级连续数据保护机制在记录数据以及恢复数据等方面都有一定的差异，但其本质都是通过记录逻辑地址空间数据的变化，并使用时间戳以及扇区号或逻辑块地址对其进行标识，在恢复时根据特定算法将记录的数据变化进行还原，最终得到目标时刻点的数据镜像。

为了对块级连续数据保护的恢复过程进行优化，下面对数据块级写请求进行了详细分析，涉及到的数学符号及含义如表 1 所列。

表 1 符号列表

符号	定义
a	逻辑块地址
A	逻辑块地址(LBA)空间
D	时间区间[0,d]
B_D	时间区间 D 内所有被更新的逻辑块地址的集合
W_{Da}	时间区间 D 内，逻辑块地址 a 上产生的更新操作次数
S_D	B_D 中所有逻辑块，按照逻辑块地址连续的序列的集合
s_i	S_D 集合中的第 i 条序列 $0\leqslant i\leqslant\lvert S_D\rvert$，其长度 $\lvert s_i\rvert$ 为组成该条序列的逻辑块个数

2.1 数据块级写操作集中分布特性

数据块级连续数据保护的历史数据分布特性是由数据块级写操作分布特性所引起的。对于不同的应用，其写操作的分布特性也各不相同，由此导致了块级连续数据保护的历史数据分布特性的差异。

本文选用存储性能理事会(Storage Performance Council，SPC)发布的 Trace 数据，对数据块级的写请求进行了具体的统计分析。该 Trace 数据采集自某大型金融机构的 OLTP 应用，数据块大小为 512 字节。图 1 给出了连续 1、5、15、30 分钟的时间区间内写操作在逻辑地址空间上的分布曲线。

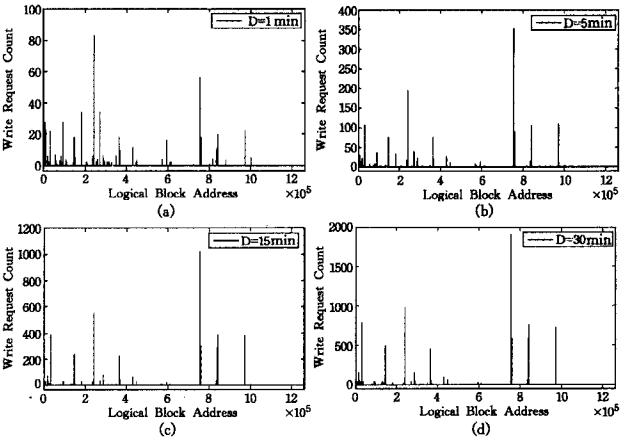

图 1 不同时间区间内，写操作在逻辑地址空间上的分布密度

统计结果表明，在整个逻辑地址空间[0,1260000]，平均每分钟产生 70000 次对逻辑块的写操作，每小时产生的写数据量高达 2GB 以上。从图 1 可以看出，写请求在逻辑地址空间上呈现出高度集中分布的特征，超过 88%的写操作集中在 3.8%的逻辑地址上，在时间区间为 30 分钟时，对单一逻辑块的写操作次数最高达到 1903 次。对同一数据块的多次写操作，尽管文件系统等应用已经在缓存中对其进行了合并处理，

但由于缓存空间有限，缓存时间较短，从长期看来写操作仍然呈现出在逻辑地址空间中高度集中的特性。该特征在 OLTP 等写操作密集的应用中尤为明显，进而导致块级连续数据保护的历史数据在逻辑地址空间内高度集中的分布特征。

为了进一步分析数据块级写请求的集中分布特性，对于不同的时间区间 D，根据式(1)计算出逻辑块在该时间区间内的写操作次数均值，结果如图 2 所示。

$$\overline{W_D}=\frac{1}{|B_D|}\cdot\sum_{a=0}^{|A|}W_{Da} \tag{1}$$

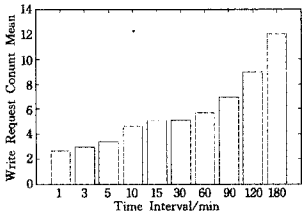

图 2　逻辑块写操作次数均值

计算结果表明，随着时间区间的不断增大，逻辑块写操作次数的均值也不断增大。仅在 1 分钟的时间区间内，写操作次数均值已达到 2 次以上；180 分钟时则高达 12 次以上。可以明显看出，数据块级写操作在逻辑地址空间中呈现出高度集中分布的特性。

通常，块级连续数据保护机制在恢复数据时，只是按照写操作提交的时间先后顺序对恢复时间区间内的所有历史数据执行 redo 操作。然而，对于任意的逻辑块，在恢复时间区间内极有可能被更新多次，但块级连续数据保护机制在数据恢复的过程中，仅有最靠近目标时刻点的那次写操作对最终的恢复结果产生影响。

因此，根据数据块级写请求集中分布的特性，将不影响最终恢复结果的无效写请求剔除，可以大幅度减少恢复过程中 I/O 操作的数据量，进而降低恢复时间，提高块级连续数据保护数据恢复的效率。

2.2　数据块级写操作连续分布特性

为了分析数据块级写操作按照逻辑块地址连续分布的特性，根据表 1 中对逻辑块序列的定义，对于不同的时间区间 D，由式(2)得到该时间区间内逻辑块序列 s 的长度均值分布曲线，如图 3 所示。与之对应，图 4 给出了该时间区间内逻辑块序列个数 $|S_D|$ 的分布曲线。

$$\overline{|s|}=\frac{1}{|S_D|}\cdot\sum_{i=1}^{|S_D|}|s_i| \tag{2}$$

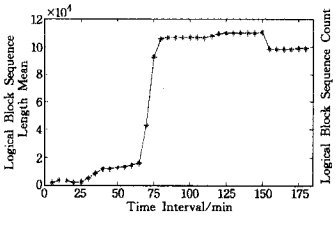

图 3　逻辑块序列长度均值随时间区间的分布曲线

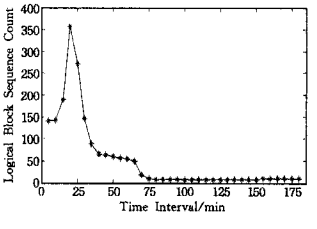

图 4　逻辑块序列个数随时间区间的分布曲线

从图 3 中可以看出，数据块级写操作呈现出按照逻辑块地址连续分布的特征。在时间区间为 5 分钟时，逻辑块序列长度均值为 1850，65 分钟后突然陡增，在时间区间达到 80 分钟后，其长度均值达到最大，约为 1.1×105，并趋于平稳。与之相对应，如图 4 所示，在时间区间达到 20 分钟、写请求达到一定数量后，系统趋于稳定，逻辑块序列的个数骤然减少，并在 75 分钟之后趋于平稳。因此，从逻辑块序列长度的增加和个数的减少中可以明显看出，对某一时间区间内所产生的写操作，按照逻辑块地址进行排序后，会在逻辑块地址空间上呈现出连续分布的特性，并且时间区间越大，连续分布特性越显著。

由于磁盘等存储介质的物理特性，顺序 I/O 的吞吐率通常远远高于随机 I/O。目前，传统的块级连续数据保护技术在恢复数据时，通常按照写操作产生的时间先后顺序对其执行 redo 操作，写操作的高随机性降低了 I/O 效率，进而影响了数据恢复的效率。虽然，在 I/O 调度层已经采用电梯算法等进行了优化，但由于其受缓存空间小、缓存时间短等限制，对于连续数据保护中通常较大的恢复时间区间(几分钟、甚至几小时)并不能起到明显作用。

因此，利用数据块级写操作在连续时间区间内呈现出的按照逻辑块地址连续的特性，在恢复数据时，将历史写请求按照逻辑块地址排序，可降低恢复过程中写操作的随机性，提高 I/O 效率，进而提高恢复效率。

3　RM-LCDF 恢复方法

3.1　基本恢复方法

通常，块级连续数据保护系统采用日志方式记录历史数据，即按照写操作产生的时间先后顺序将其保存于历史卷上；相应地，在恢复数据时，则按照历史数据在历史卷中保存的逻辑块地址顺序，即历史数据产生的时间先后顺序，进行读取并依次执行 redo 操作。

对于数据恢复的一般情况，如图 5 所示，T_0 为连续数据保护的起始时刻点，T_{target} 为恢复目标时刻点，T_{latest} 为当前时刻点。假设在 T_{latest} 时刻发生灾难造成数据损毁，需要恢复到目标时刻点 T_{target}，那么只需要对时间区间$[T_0, T_{target}]$内的所有历史写操作，依次在初始时刻 T_0 的镜像 I_0 上执行 redo 操作。

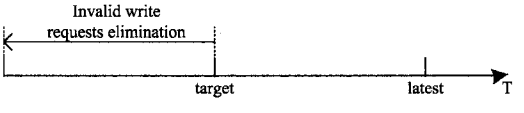

图 5　数据恢复时间点示意图

3.2　RM-LCDF 恢复方法

结合数据块级写请求的集中分布和连续分布特性，以块级连续数据保护基本恢复方法为基础，RM-LCDF(Recovery Method base on Localized and Continuous Distribution Features)方法采用以下 3 种策略进行了优化。

(1)去除无效写请求

根据 2.1 节中关于数据块级写操作集中分布特性中的讨论，剔除恢复时间区间内对同一逻辑块进行更新的无效写请求，可以大幅度减少恢复时 I/O 操作的数据量，进而降低恢复时间。

RM-LCDF 在执行目标时刻点 T_{target} 的恢复时，首先按照写操作产生时间先后顺序的逆序，即 $T_{target}\rightarrow T_0$ 的顺序，对恢复时间区间$[0, T_{target}]$内的所有历史写请求进行扫描。对于在该区间内被更新的任意逻辑块 B，仅保留最先扫描到的对

其进行更新的写请求。

(2)多缓冲

在实际应用中,块级连续数据保护机制通常将源数据卷与历史卷配置于不同的物理卷上,以提高源数据卷和历史数据卷的 I/O 并发度,降低记录历史数据时对上层应用的影响。因此,在恢复数据时,从历史数据卷读取历史数据和向源数据卷执行 redo 两个操作也可以并发进行。RM-LCDF 采用读历史数据卷和写源数据卷两个线程,读线程利用多缓冲区将待执行 redo 操作的数据异步提交给写线程,以提高恢复数据时的 I/O 并发度,从而降低恢复时间。

(3)逻辑块地址排序

根据 2.2 节中关于数据块级写操作连续分布特性中的讨论,RM-LCDF 的写线程首先将缓冲区中等待 redo 的所有写请求按目标逻辑块地址进行排序;然后对源数据卷执行 redo 操作,以降低写 I/O 的随机性,减少写 I/O 的平均寻道时间,提高恢复效率。

RM-LCDF 机制的数据恢复时序图如图 6 所示。

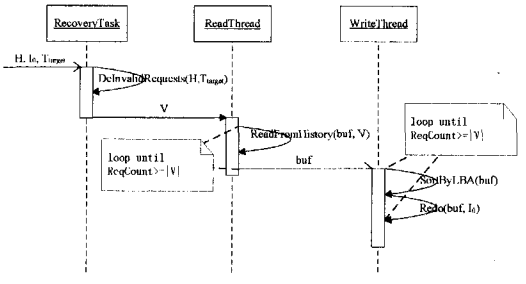

图 6 RM-LCDF 数据恢复时序图

I_0:初始数据镜像;

H:时间区间$[T_0, T_{target}]$内的历史写请求描述符序列,时间先后有序;

I_{target}:目标时刻 T_{target} 数据镜像;

V:需要执行 redo 操作的写请求描述符集合。

对于恢复目标时刻 T_{target},RM-LCDF 首先对时间区间$[T_0, T_{target}]$内的历史写请求描述符序列逆序扫描,去除无效写请求,得到需要执行 redo 操作的写请求描述符集合 V;然后读线程开始工作,每次循环都根据集合 V 从历史数据中读取出大小为 buf 的数据,然后异步提交给写线程,集合 V 中每个写请求都被读取后循环结束。此时,写线程开始同读线程并发工作,在每次循环中,写线程首先将 buf 中的历史写请求按照逻辑块地址进行排序,然后将按逻辑块地址有序的写请求写入初始镜像 I_0 中,集合 V 中每个写请求都被 redo 后循环结束,恢复完成。

4 恢复性能分析

为了分析 RM-LCDF 所采用的去除无效写请求、多缓冲和逻辑块地址排序 3 种策略对恢复效率的影响,本文采用 SPC Trace 数据进行了具体的统计分析。

4.1 去除无效写请求

对于去除无效写请求对恢复时间的影响,选取 6 组不同的时间区间:1min、3min、5min、10min、15min、30min,分别统计在去除无效写请求前后所需要进行 redo 操作的写请求个数。其中,去除无效写请求之前的写请求个数为恢复时间区间内所有写操作的个数,即$\sum_{a=0}^{|A|} W_{Da}$;去除之后的个数则为恢复时间区间内被更新的逻辑块地址个数,即$|B_D|$。统计结果如图 7 所示。

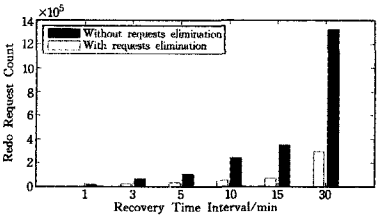

图 7 去除无效写请求前后需要进行 redo 操作的写请求个数比较

统计结果显示,去除无效写请求之后需要进行 redo 操作的写请求个数大幅度减少,平均降幅超过 78%,这也印证了数据块级写操作集中分布特性。由于读取历史写请求和 redo 写请求是影响恢复效率的两个重要因素,因此,读、写数据量 78%的锐减必然使得恢复时间锐减,进而大幅度提高恢复效率。

4.2 多缓冲和逻辑块地址排序

I/O 的随机程度对 I/O 吞吐率有着明显的影响,RM-LCDF 对缓冲中的历史写请求按照逻辑块地址排序,以降低随机 I/O 的比例。根据数据块级写操作连续分布特性,缓冲大小直接影响 I/O 吞吐率:缓冲区越大,按照逻辑块地址排序后随机 I/O 所占比例越低,写 I/O 吞吐率越高;特殊地,当缓冲足以容纳需要执行 redo 操作的所有写请求时,写 I/O 的随机性降至最低。设恢复数据时顺序写 I/O 操作次数占总 I/O 操作次数的比率为 p_D,根据式(3)分别计算在不同缓冲大小下的顺序写 I/O 比率,其中缓冲大小从 1MB 开始并按照 1MB 递增,得到顺序写 I/O 比率随缓冲大小的变化曲线,如图 8 所示。

$$p_D = 1 - \frac{|S_D|}{\sum_{i=1}^{|S_D|} \left\lceil \frac{|s_i|}{IO_{size}} \right\rceil} \tag{3}$$

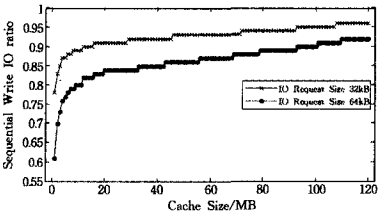

图 8 顺序写 I/O 比率随缓冲大小的变化曲线

从计算结果可以看出,随着缓冲的不断增大,顺序写 I/O 的比率也不断增长,并在 20MB 以后趋于平缓。与之相比,在不采用多缓冲及写优化策略时,顺序写 I/O 操作占总 I/O 操作的比率仅为 0.25 左右。因此,对缓冲中的历史写请求按照逻辑块地址排序可以显著提高顺序 I/O 的比例,进而有效提高恢复效率。

对于采用了去除无效写请求、多缓冲和逻辑块地址排序 3 种优化策略的 RM-LCDF 方法,其恢复时间由去除无效写请求的时间开销 $T_{eliminate}$、读取第一个缓冲区、排序并 redo 最后一个缓冲区,以及读写线程并发处理其他缓冲区的时间开销 4 部分组成。假设读取历史数据和执行 redo 操作的 I/O 吞吐率分别为 r 和 w,恢复过程需要进行 redo 操作的数据量为 R,缓冲区大小为 B,按逻辑块地址进行排序的总时间开销为 T_{sort},其恢复时间可以表示为式(4)。

$$T_{recovery}=T_{eliminate}+\frac{B}{r}+(\frac{B}{w}+T_{sort}\cdot\frac{B}{R})+\mathrm{MAX}(\frac{R-B}{r},\frac{R-B}{w}+T_{sort}\cdot(\frac{R-B}{R}))$$
$$\approx T_{eliminate}+\mathrm{MAX}(\frac{R}{r},\frac{R}{w}+T_{sort}) \quad (4)$$

恢复数据时，RM-LCDF 按顺序从历史卷上读取数据，因此可以假设读取历史数据的 I/O 吞吐率 r 是一定的，不随缓冲区大小而变化；而写 I/O 的吞吐率随缓冲大小的增加而增加；另外，在内存中排序的时间开销通常远远小于磁盘 I/O 的时间开销。所以，根据式(4)，缓冲区大小的不断增加使得写 I/O 吞吐率 w 略高于读 I/O 吞吐率 r 时，恢复时间取得最小值，缓冲大小达到最优。

5 实验分析

5.1 实验环境

为了在实际环境中对 RM-LCDF 的性能进行评估，我们在 Windows 平台下，利用磁盘过滤驱动技术，在逻辑卷层实现了块级连续数据保护原型系统。原型系统在逻辑卷层截获上层应用的写请求，并在历史数据卷中按照写请求产生的时间先后顺序进行记录。为了获得更加客观的实验数据，存储端采用独立 RAID 控制器，并且源数据卷和历史数据卷分别绑定不同的 RAID Group。实验在浪潮 AS300N 存储服务器上进行，具体的实验环境如表 2 所列。

表 2 实验环境

CPU	Intel Xeon 5504 * 2
内存	DDR III 16GB
硬盘	OS:RAID 1 (SATA 150GB * 2) Source Storage:RAID5 (SAS 300GB * 5) History Storage:RAID5 (SAS 300GB * 5)
操作系统	Windows 2003 R2 Enterprise X64 Edition

实验采用存储性能理事会发布的 Trace 数据，通过 replay 的方式在源数据卷上进行重放，重放时间为 12 小时。对于以重放开始时刻为起点、5 分钟为间隔连续递增的 120 个时间区间，分别采用基本恢复方法以及 RM-LCDF 方法中 3 种策略的不同组合进行恢复，并测量各情况下的恢复时间。

5.2 缓冲大小对 RM-LCDF 恢复效率的影响

根据 4.2 节中对缓冲大小的讨论可知，存在最优缓冲大小，使得恢复时间最短。为了寻找该最优值，在不同的缓冲区大小下对 RM-LCDF 方法的最终恢复时间进行了测量，结果如图 9 所示。

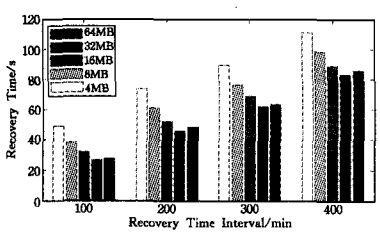

图 9 不同缓冲大小对恢复时间的影响

可以看出，在缓冲区大小为 4MB、8MB、16MB 和 32MB 时，随着缓冲区的增大，恢复时间逐渐减小。这印证了 4.2 节中的讨论，即随着缓冲区增大，按照逻辑块地址排序之后写 I/O 的随机比率降低，获得了更高的吞吐率，因此恢复时间随着缓冲区的增大而减小。但是，当缓冲区大小增加到 64MB 时，恢复时间不仅没有降低，反而略微上升，这是由于：(1)写 I/O 的吞吐率已足够高，不再是恢复过程的瓶颈，继续增大缓冲已不能降低总恢复时间；(2)由于缓冲区大小的增大，使得读取第一个缓冲区和 redo 最后一个缓冲区的时间有所增加，进而使得最终恢复时间增加。因此，在下面的实验中选择 32MB 作为 RM-LCDF 的缓冲大小。

5.3 逻辑块地址排序策略对写 I/O 的影响

取缓冲大小为 32MB，以 RM-LCDF 机制下的读 I/O 时间开销和恢复总时间作为参照，图 10 给出了采用逻辑块地址排序策略前后的写 I/O 时间开销随恢复时间区间的变化曲线。从图中可以看出，按照逻辑块地址排序的优化策略极大地提高了写 I/O 的效率，与优化前相比，写 I/O 时间开销平均降低 90%以上；甚至读 I/O 的时间开销高于写 I/O，平均高出 4 倍以上。此时，读 I/O 已成为恢复过程的瓶颈。此外，由于去除无效写请求和逻辑块地址排序也会带来一定的时间开销，因此 RM-LCDF 的总恢复时间略高于读 I/O 的时间开销。

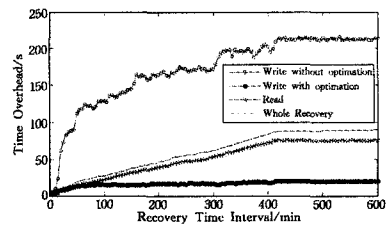

图 10 采用逻辑块地址排序策略前后写 I/O 时间开销随恢复时间区间的变化曲线

以不采用逻辑块地址排序的写 I/O 吞吐率作为参照，图 11 给出了在缓冲大小为 32MB 时，RM-LCDF 机制下的读、写 I/O 吞吐率随着恢复时间区间的变化曲线。从图中可以看出，采用逻辑块地址排序策略后，写 I/O 的吞吐率有了较大的提高，并且在恢复时间区间大于 200 分钟后略高于读 I/O 的吞吐率。同样可以看出，读 I/O 已经成为恢复过程的瓶颈。

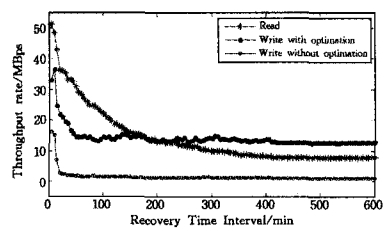

图 11 读写 I/O 吞吐率随恢复时间区间的变化曲线

5.4 去除无效写请求及逻辑块地址排序的时间开销

在实验中对除无效写请求和逻辑块地址排序所带来的时间开销进行了定量的测量。对于不同的恢复时间区间，表 3 列出了去除无效写请求的时间开销；对于不同的缓冲区大小，表 4 列出了逻辑块地址排序的时间开销。可以看出，两种优化策略所带来的时间开销仅为几十到几百个毫秒，与处理相同数据量的 I/O 相比，是可以容忍的。

表 3 去除无效写请求的时间开销

恢复时间区间 (min)	平均逻辑块个数	平均数据量 (MB)	时间开销 (ms)
5	350000	170.9	31
10	700000	341.8	46
15	2100000	512.7	109
30	4200000	1025.4	437
60	8400000	2050.8	987

表 4　逻辑块地址排序的时间开销

缓冲大小(MB)	平均逻辑块个数	平均时间开销(ms)
4	8192	0.0289
8	16384	0.0401
16	32768	0.0584
32	65536	0.0857
64	131072	0.1216

5.5　RM-LCDF 中不同优化策略组合及基本恢复方法恢复效率比较

将 SPC Trace 数据在源数据卷上进行 replay 12 小时之后，选择以 5 分钟为间隔连续递增的 120 个时间区间。对于 RM-LCDF 中的去除无效写请求、多缓冲和逻辑块地址排序 3 种优化策略的 3 种不同组合：(1)仅去除无效写请求，(2)去除无效写请求与多缓冲两种策略，(3)去除无效写请求、多缓冲和逻辑块地址排序 3 种策略，即 RM-LCDF 方法，以基本恢复方法作为参照，分别在 120 个时间区间上执行数据恢复操作，测量其恢复时间。不同恢复策略组合的恢复时间随恢复时间区间变化的曲线如图 12 所示。从图中可以看出，RM-LCDF 所采用的去除无效写请求、多缓冲以及逻辑块地址排序 3 种优化策略，每一种都可以显著提升恢复效率；与基本恢复方法相比，RM-LCDF 方法能够将恢复时间降低一个数量级以上。

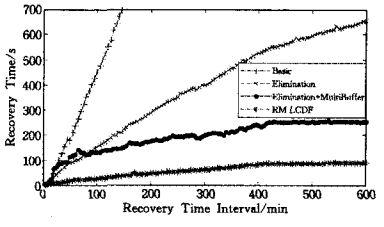

图 12　不同优化策略组合的恢复时间随恢复时间区间的变化曲线

5.6　系统资源占用比较

为了定量地分析 RM-LCDF 的 CPU 及内存资源的占用情况，以基本恢复方法作为参照，对于采用不同缓冲大小的 RM-LCDF 方法，定量测量了恢复时间区间为 60min 时的平均资源占用量，结果如表 5 所列。从结果中可以看出：(1)与基本恢复方法相比，RM-LCDF 所增加的内存占用量约为多缓冲所占用的空间大小；(2)随着缓冲的增大，读写 I/O 的并发度以及写 I/O 的吞吐率增大，使得恢复效率不断提高，因此 RM-LCDF 的 CPU 占用量有少量增加，但增幅并不明显。

表 5　系统资源占用比较

恢复方法	CPU (%)	Memory (MB)
Basic Recovery	10.548	15
RM-LCDF 4MB	10.836	25
RM-LCDF 8MB	11.267	32
RM-LCDF 16MB	11.945	49
RM-LCDF 32MB	11.996	82
RM-LCDF 64MB	11.983	145

结束语　通过去除无效写请求、多缓冲和逻辑块地址排序 3 种优化策略，RM-LCDF 机制不仅大幅度减少了恢复过程中需要执行 redo 操作的数据量，提高了读取历史数据和执行 redo 操作的 I/O 并发度，还提高了写 I/O 操作的吞吐率。实验结果表明，与基本恢复方法相比，RM-LCDF 能够降低恢复时间一个数量级以上，极大地提高了恢复效率。

参 考 文 献

[1] Keeton K, Santos C, Beyer D, et al. Designing for disasters [C]// Proc of the 3rd Usenix Conference on File and Storage Technologies. Berkeley: USENIX Association, 2004: 59-72

[2] Patterson D. Availability and Maintainability >> Performance: New Focus for a New Century [C]// Proc of the 1st USENIX Conference on File and Storage Technologies. Berkeley: USENIX Association, 2002

[3] McKnight J, Asaro T, Babineau B. Digital Archiving: End-User Survey and Market Forecast [EB/OL]. The Enterprise Strategy Group. http://www.enterprisestrategygroup.com/2006/03/digital-archiving-end-user-survey-market-forecast-2006-2010/, 2010-07-04

[4] The 451 Group. Total Recall: Challenges and Opportunities for the Data Protection Industry [EB/OL]. The 451 Group, http://www.the451group.com/reports/executive_summary.php? id=218, 2010-07-04

[5] Damoulakis J. Continuous Protection[J]. Storage, 2004, 3(4): 33-39

[6] Rock M, Poresky P. Shorten Your Backup Window [J]. Storage, special issue on managing the information that drives the enterprise, 2005(05): 28-34

[7] Chervenak A, Vellanki V, Kurmas Z. Protecting File Systems: A Survey of Backup Techniques [C]// Proc of the joint NASA and IEEE Mass Storage Conference. Piscataway, NJ: IEEE, 1998: 17-31

[8] 王迪，薛巍，舒继武，等. 海量存储网络中的虚拟盘副本容错技术[J]. 计算机研究与发展，2006，43(10)：1849-1854

[9] Cornell B, Dinda P A, Bustamante F E. Wayback: A User-level Versioning File System for Linux [C]// Proc of the 2004 USENIX Annual Technical Conference. Berkeley: USENIX Association, 2004: 19-28

[10] Morrey C B III, Grunwald D. Peabody: the time travelling disk [C]// Proc of the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies. Piscataway, NJ: IEEE, 2003: 241-253

[11] Zhu M B, Li K, Patterson R H. Efficient data storage system: US, 6928526 [P]. http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=HITOFF&p=1&u=%2Fnetahtml%2FPTO%2Fsearch-bool.html&r=1&f=G&l=50&co1=AND&d=PTXT&s1=6928526.PN.&OS=PN/6928526&RS=PN/6928526, 2002-12-20

[12] Yang Q, Xiao W J, Ren J. TRAP-Array: A disk array architecture providing timely recovery to any point-in-time [C]// Proc of the 33rd Annual International Symposium on Computer Architecture. Piscataway, NJ: IEEE, 2006: 289-301

[13] 李旭，谢长生，杨靖，等. 一种改进的块级连续数据保护机制[J]. 计算机研究与发展，2009，46(5)：762-769

[14] Verma A, Voruganti K, Routray R, et al. SWEEPER: An Efficient Disaster Recovery Point Identification Mechanism [C]// Proc of the 6th USENIX Conference on File and Storage Technologies. Berkeley: USENIX Association, 2008: 297-312

[15] 侯利曼，李战怀，胡娜. 基于数据差异的 CDP 邻近时间点恢复[J]. 计算机科学，2011，38(5)：159-163

[16] 向小佳，舒继武，郑纬民. 一种细粒度高效多版本文件系统[J]. 软件学报，2009，20(3)：754-765

[17] Shaull R, Shrira L, Xu H. Skippy: a new snapshot indexing method for time travel in the storage manager [C]// Proc of the 2008 ACM SIGMOD international conference on Management of data. New York: ACM, 2008: 637-648

[18] 吴广君，云晓春，方滨兴，等. HCSIM：一种长期高频 Block-Level 快照索引技术[J]. 计算机学报，2009，32(10)：2080-2090