HCSIM：一种长期高频 Block-Level 快照索引技术

吴广君[1)] 云晓春[2)] 方滨兴[1),2)] 王树鹏[2)] 余翔湛[1)]

[1)](哈尔滨工业大学计算机网络与信息安全技术研究中心 哈尔滨 150001)

[2)](中国科学院计算技术研究所 北京 100190)

摘 要 高频快照技术应用于备份时，能够为物理错误和人为错误提供数据保护，构建可靠数据存储环境. 针对长期、高频 block-level 快照检索效率低下问题，在对目前常见的 block-level 快照技术建模、分析基础上，提出结合数据分布特征和检索模式的分层次二维索引结构——HCSIM. 实验、分析结果表明：HCSIM 索引技术应用于长期、高频快照管理时，比时空索引结构 Overlapping B+ Tree 索引技术显著提高索引的存储效率和检索效率；通过定性的分析，HCSIM 是目前存在的 block-level 快照索引模式中存储效率和检索效率相对平衡的索引技术.

关键词 快照；安全存储；索引；灾难恢复；块数据

中图法分类号 TP393 **DOI 号**：10.3724/SP.J.1016.2009.02080

HCSIM：An Indexing Method for Long-Lived Frequent Block-Level Snapshot

WU Guang-Jun[1)] YUN Xiao-Chun[2)] FANG Bin-Xing[1),2)] WANG Shu-Peng[2)] YU Xiang-Zhan[1)]

[1)](*Research Center of Computer Network and Information Security Technology*, *Harbin Institute of Technology*, *Harbin* 150001)

[2)](*Institute of Computing Technology*, *Chinese Academy of Sciences*, *Beijing* 100190)

Abstract Snapshot-based backup techniques can protect data from accidental and physical errors and support reliable storage. The long-lived frequent snapshot is low in search efficiency, because of the shared data between versions. Up to now there is no efficient indexing method for the long-lived frequent block-level snapshot. Adapting the block-level snapshot distributing features, this paper proposes a general-purpose two-dimensional indexing structure: Hierarchical Clustering Snapshot Indexing Method (HCSIM). The experimental results show that HCSIM can dramatically increase indexing storage efficiency than traditional temporal and spatial indexing structure Overlapping B+ Tree. At the meantime, HCSIM can achieve better query efficiency in the context of long-lived snapshot management. The theoretical analysis exposes that HCSIM structure can achieve better balance between storage and query efficiency than nowadays block-level snapshot indexing method.

Keywords snapshot; reliable storage; indexing; disaster recovery; block-level

---

收稿日期：2009-07-15；最终修改稿收到日期：2009-08-21. 本课题得到国家“八六三”高技术研究发展计划项目基金（2007AA01Z406，2009AA01A403，2009AA01Z437）和国家“九七三”重点基础研究发展规划项目基金（2007CB3111003）资助.**吴广君**，男，1981 年生，博士研究生，主要研究方向为数据容灾、海量存储、网络安全. E-mail：wuguangjun@{software.ict.ac.cn，gmail.com}. **云晓春**，男，1971 年生，博士，教授，博士生导师，主要研究领域为网络与信息安全. **方滨兴**，男，1960 年生，教授，博士生导师，中国工程院院士，主要研究领域为信息安全、计算机网络、并行计算. **王树鹏**，男，1980 年生，博士，主要研究方向为数据容灾、网络安全. **余翔湛**，男，1973 年生，博士，副教授，主要研究方向为计算机网络生存性技术、容灾.

1 引　言

随着网络技术的发展和磁盘技术的成熟，数据的存储量急剧地增长，到 2010 年将超过 27E(27×10^{18}Bytes)①. 数据的安全存储已成为信息安全研究领域的重要课题. 8%的企业称仅宕机 1h，会造成 1 百万美元的损失②，数据丢失直接导致企业的破产. 安全存储首先要保证数据的可靠性. 引起数据丢失原因可以分为两类：物理错误和业务逻辑错误. 物理错误通常是由介质的损坏、丢失以及雪灾、地震，恐怖袭击等灾难性事件引起的数据损毁，这类错误引起的数据丢失占总比例的 75%左右[1]. 逻辑错误是由于业务操作导致的数据丢失，如删除、修改操作引起的数据污染. 目前通常采用快照备份技术同时解决上述两类错误.

快照技术是通过数据的拷贝或复制，获取在某个时间点完整、可用的数据镜像③. 快照备份技术通过保存历史版本快照数据，提供数据恢复. 高频快照技术应用于备份时，具有支持在线备份、提供弹性恢复窗口等优点[2](以下简称为高频快照). 但是由于快照数据共享产生了复杂的依赖关系，随着版本数目、版本频率的增加导致高频快照检索效率低下，直接影响了高频快照的可用性. 高频快照产生于现代存储技术，已逐渐引起国内外研究机构的重视. 可以把当前快照数据管理技术的研究概括为如下几个方面.

一类快照管理技术是采用日志方式记录历史数据，这类技术广泛存在于文件系统、逻辑卷管理器和设备驱动器等层次中，如 WayBack[3] 使用数据日志记录所有的变化数据，Elephant[4] 通过链接变化数据的 inode 记录多版本文件；Clotho[5] 在逻辑卷层次通过 Device Version List 依次保存早期快照数据；Peabody[6] 面向 iSCSI 设备驱动器把变化的 block 依次存储. 这类技术在检索早期快照数据时通过 Redo 或者 Undo 方式，按序遍历版本日志，在面向长期、高频快照管理时检索效率低下. 尽管 TRAP[7] 利用了相邻版本的 block 具有相似性，通过编码提高存储效率，但是在数据检索时增加了解码过程，进一步降低了数据的检索效率.

一类快照技术通过 Copy-On-Write(COW)或 Redirect-On-Write(ROW)方式，结合文件系统的元数据进行快照数据管理，如 WAFL[8] 通过专有文件系统实现快照管理；Ext3cow[9] 通过修改 Ext3 文件系统实现快照多版本管理，这类技术一方面依赖于文件系统实现快照管理，与文件系统的语义相关；另一方面为了不影响正常业务的操作效率，往往限长版本个数，如 WAFL 最多管理 255 个版本快照数据，Ext3cow 采用限长分段方式提高版本组织效率.

最近的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[10] 在建立局部快照时，利用目录、文件版本之间的相关性提高快照检索效率；Skippy[11] 利用内存快照 Hot Data 的特性，提出分层次的索引结构，提高长期快照的检索效率. 尽管这类技术在应用中可以提高快照的检索效率，但是具有应用场合的限制，缺乏通用性.

因此，目前高频快照管理中存在着检索效率的瓶颈问题，尚无一种相对通用的面向长期、高频的快照索引技术，为此本文提出一种面向长期、高频的 block-level 快照索引技术. Block-level 快照技术可以在系统软件、逻辑卷管理器以及设备驱动程序等不同层次上实现，支持应用无关的数据复制. 现代存储技术基于 block-level 高频快照，可以实现验证数据的正确性以及业务迁移等操作. 本文在对目前常见的 block-level 快照技术建模分析基础上，结合以前的研究工作，提出了一种结合快照数据分布特征和访问模式的通用 block-level 高频快照索引技术，并给出了详细的比较、分析，本文的主要贡献如下：

(1) 本文给出了目前常见的 block-level 快照技术的建模、分析，并在 SPC-1 trace 下进行具体的运算. 通过计算得出 block-level 快照数据具有“集中”、“连续”的双重时空分布特征，为 block-level 高频快照索引技术提供分析基础；

(2) 结合 block-level 快照数据的分布特征和检索模式，提出以区间索引技术、快照融合为基础的分层次索引结构 Hierarchical Clustering Snapshot Indexing Method(HCSIM). HCSIM 不仅具有结构简单、操作方便的特点，而且通过与多版本数据库中通常采用的 Overlapping B+ Tree[12] 综合比较，在高频快照管理中，可有效减少索引数据的存储量，提高长期快照的检索效率. 同时本文给了 HCSIM 索引技术与当前常见的 block-level 快照索引技术的综合分析，HCSIM 在索引存储效率和检索效率方面具

① McKnight J, Asaro T, Babineau B. Digital archiving: End-user survey and market forecast 2006～2010. The Enterprise Strategy Group, January, 2006

② ONLINE SURVEY RESULTS 2001 COST OF DOWNTIME. http://contingencyplanningresearch.com/2001%20Survey.pdf

③ SNIA 快照定义：http://www.snia.org/education/dictionary/s/

有更好的平衡关系.

2 Block-level 快照时空特性分析

快照备份数据主要用来支持数据恢复,在容灾理论中针对各类数据错误和恢复有详细的讨论,使用 RPO 与 RTO[13] 作为子计划衡量指标,容灾理论已超出本文范围,不作详细讨论.为了使本文提出的索引技术、分析方法具有实用价值,以下的建模分析中结合了 RPO 与 RTO 的物理意义进行讨论.

2.1 基本模型

根据数据保护时间粒度的不同,可以把目前常见的 block-level 层次数据保护技术进行分类.如图 1 所示,传统周期备份技术以天为单位产生增量备份数据,以周为单位产生全量备份数据,以磁带为存储介质.这类技术的备份窗口大,恢复效率低.快照技术应用于备份时,可以提高备份效率,降低备份窗口.当备份窗口无限缩小,记录每次写操作产生的数据信息时,可以实现持续数据保护服务(CDP).

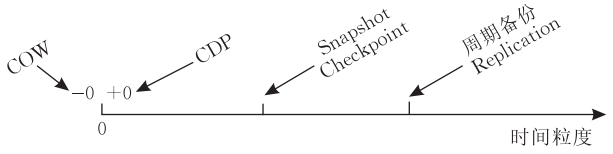

图 1 不同时间粒度的数据保护技术

尽管 block-level 数据保护技术的应用背景和实现技术的差异很大,但是具有相似的特征:本质是记录逻辑地址空间内的更新数据,利用时间点区分不同版本的数据,使用扇区号或逻辑块号(Logical Block Number)标识逻辑地址.为了建立一种相对通用的 block-level 快照索引技术,下面对常见的 block-level 数据保护技术进行统一分析,涉及到的数学符号及物理意义如表 1 所示.

表 1 数学符号列表

数学符号	物理含义
x	逻辑地址
T	数据保护粒度
$w_T(x)$	在数据保护粒度为 T,逻辑地址 x 上的写密度函数
S_D	由 $w_T(x)$ 在时间段 $[0,D]$ 内产生的数据量
l	在逻辑地址空间内连续分布的数据块长度
$E_w(l)$	在写操作 $w_T(x)$ 下,数据连续分布的期望值
$\delta_w(l)$	在写操作 $w_T(x)$ 下,数据连续分布的标准差

首先引入写密度函数 $w_T(x)$,其物理意义是在时间间隔 T、逻辑地址 x 内最后一次写操作产生的更新数据,其量值可以用式(1)表达,$w_T(x)$ 称为在数据保护粒度 T 下的写密度分布函数,简称写密度函数.使用 $w_T(x)$ 可对多种 block-level 数据保护技术进行抽象.COW 快照技术记录写操作更新之前的数据,可以设置时间保护粒度趋近于 -0,使用式(2)表示.CDP 技术记录每次写操作产生的更新数据,可以设置时间保护粒度趋近于 $+0$ 使用式(3)表示.根据极限的定义可知 COW 和 CDP 两种技术产生的数据量相同,以下主要讨论 $T>0$ 时的快照技术.对于周期快照技术、传统数据备份技术,设复制周期为 T_S,可以利用式(4)表达.

$$\|w_T(x)\| = \begin{cases} 0, \text{在时间粒度 } T \text{ 内,} x \text{ 没有发生写操作} \\ 1, \text{在时间粒度 } T \text{ 内,} x \text{ 发生写操作} \end{cases} \tag{1}$$

$$\lim_{T\to -0} w_T(x) = w_{-0}(x) \tag{2}$$

$$\lim_{T\to +0} w_T(x) = w_{+0}(x) \tag{3}$$

$$w_T(x)\big|_{T=T_s} = w_{T_s}(x) \tag{4}$$

Block-level 数据保护技术应用于容灾计划时,数据保护粒度 T 表示最多丢失时间间隔 T 内的所有操作数据,体现 RPO 的物理含义,因此 $w_T(x)$ 不仅可以抽象目前常见的 block-level 数据保护技术,在容灾背景下,还可以量化 RPO 的值.根据写密度函数 $w_T(x)$ 可以计算某些快照数据的分布特征,如由写密度函数 $w_T(x)$ 在一次快照中产生的数据量为 S_T,可以表示为

$$S_T = \int_0^{+\infty} \|w_T(x)\| \,\mathrm{d}x \tag{5}$$

2.2 快照集中分布特性分析

高频快照的分布特性是由于写操作分布特性引起的.对于一个确定的逻辑卷,设其地址空间为 $[0,N]$,由写操作在时间段 $[0,D]$ 内产生的数据量,结合式(5),可以表达为式(6):

$$S_D = \int_0^N \int_0^D \|w_T(x)\| \,\mathrm{d}T\mathrm{d}x = \int_0^N \sum_{T_i\in D} \|w_{T_i}(x)\| \,\mathrm{d}x \tag{6}$$

本文选用美国存储理事会发布的 SPC-1 Trace 进行具体计算.图 2 给出在前 6 个逻辑卷、连续 12 个小时的写密度分布曲线.

根据式(6)分别计算整个逻辑卷的地址空间和前 $[0,819200]$ 地址空间内产生的更新数据量,即在 $N_1=+\infty$, $N_2=819200$ 情况下分别计算 S_D,如式(7)所示.

$$S_D = \int_0^N \sum_{T_i\in D} \|w_{T_i}(x)\| \,\mathrm{d}x\big|_{N_1=+\infty;N_2=819200;D=12\mathrm{h}} \tag{7}$$

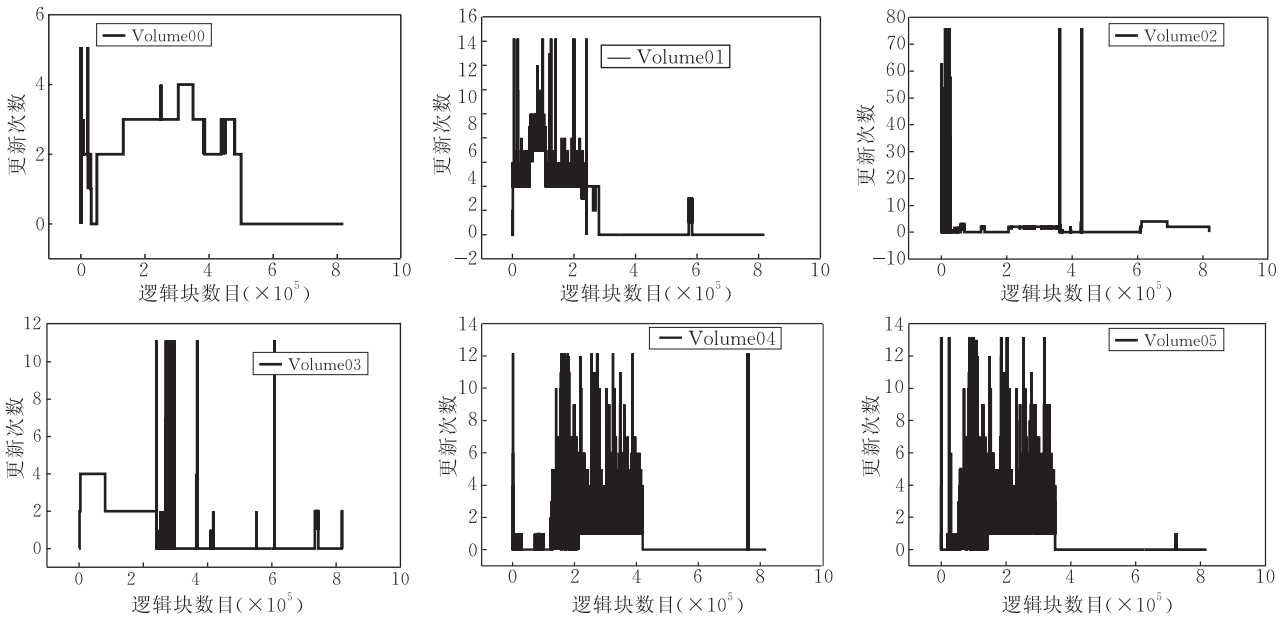

图 2　$w_{+0}(x)$在前 6 个逻辑卷的累积分布曲线(D=12h)

计算结果表明，前 20 个逻辑卷产生的写数据总量高达 12.7GB 以上，其中有 16 个逻辑卷在地址[0,819200]空间内包含了 90%以上的数据. 尽管文件系统已经对重复写的 block 在 Cache 中进行过融合处理，但是由于 Cache 的空间有限，数据缓存时间一般为 30s 左右. 从长期来看，在密集的写操作业务中(如 OLTP 业务)，仍呈现出写操作在逻辑地址空间内高度集中的特性，进而导致长期 block-level 快照数据在逻辑地址空间内高度集中的分布特征.

2.3　快照连续分布特性分析

为了分析快照数据连续分布的特性. 设快照数据连续分布的长度 l 为随机变量，分别计算在不同快照周期下的期望值 $E_w(l)$，与标准差 $\delta_w(l)$. 其中 T 分别取 1min，10min，15min，30min 以及 1h，根据式(8)、(9)得到 $E_w(l)$与 $\delta_w(l)$的分布曲线，如图 3、图 4 所示.

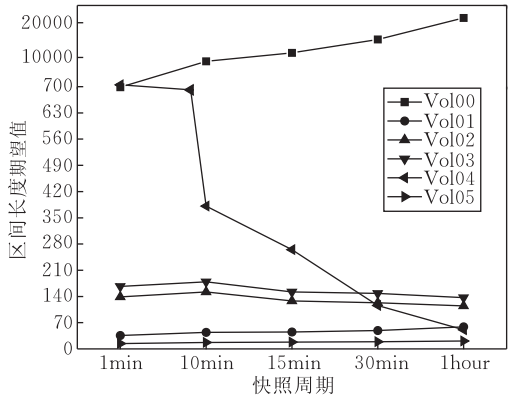

图 3　期望值分布曲线

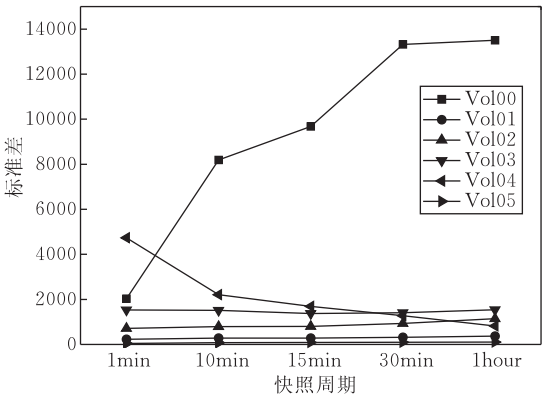

图 4　标准差分布曲线

$$E_w(l)=\sum_{i=1}^{n} l_i/n \tag{8}$$

$$\delta_w(l)=\sqrt{\sum_{i=1}^{n}(l_i-\bar{l}\ )^2/n} \tag{9}$$

计算结果显示，连续分布的快照数据长度与卷号、快照周期有关，即使相同的逻辑卷在同一个快照周期内其连续分布的长度也是动态变化的. 快照数据在逻辑地址空间内连续分布的状态，影响了数据管理中基本单位长度的划分. 目前通常采用位图技术表示快照数据在逻辑地址空间的分布，位图中的每一位表示固定长度的地址空间，文献[8]中每个 bit 表示 4K 的数据空间，文献[5]中每个 bit 表示 32K 的长度. 位图空间增大，可以减少元数据量，提高检索效率，但是降低了数据的备份和恢复效率；反之，缩小位图空间，会提高备份、恢复效率，但是增加了元数据量，降低了检索效率. 文献[14]中引入区间索引技术，以最大可连续区间作为块数据管理的基本单位，可以利用块数

据分布连续的特征，提高元数据的存储效率，在快照数据检索时对逻辑地址重叠的区间进行动态的分割.

综合以上分析，block-level 快照数据在同一版本内分布连续且连续的长度动态变化；从长期来看，不同版本的快照数据分布高度集中. Block-level 快照数据的分布特性为快照管理提供了分析基础.

3 基本索引结构

快照是一种 Point-In-Time(PIT)技术，采用基于 Time 的检索模式. 为了能够充分利用快照数据分布“连续”的特性，可以提取快照索引的时间属性信息，形成二维属性的索引结构. 空间属性由逻辑地址(LBN)表示，时间属性由快照时间戳(Time Point)表示. 不同时间点的快照索引按照时间的顺序依次链接. 如图 5 所示. 每个时间点的快照使用独立的 B+ Tree 索引，基本的索引项表达为〈*LBN Interval*，*Pointer*〉，其中 *LBN Interval* 表达一个连续分布的快照逻辑地址区间；*Pointer* 表示快照数据的存储位置.

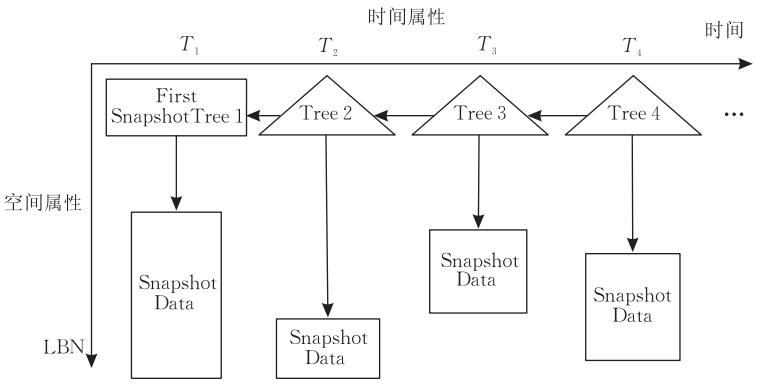

图 5 Block-level 快照二维索引结构

快照的基本检索过程包括：版本内的快照检索和版本之间的快照检索两部分. 版本内检索与 B+ Tree 基本操作相同. 版本间检索主要是解决快照数据共享问题，基本过程是按照时间顺序依次检索每个版本的快照，直到满足条件，或者到达最早快照索引 Tree 为止. 版本间检索涉及到快照融合操作. 快照融合是指相邻版本的快照，逻辑地址相同的快照数据(或快照索引)使用新版本代替老版本的过程. 融合后的结果快照等于在整个融合周期内最新版本的快照. 引入区间索引技术后具体快照合并算法见文献[14]，融合过程分为数据融合和索引融合两部分，统一记为 *Merge*().

由于时间属性和空间属性进行分离，二维索引结构在空间属性上结合了数据分布“连续”的特征，利用区间索引技术，减少元数据量，但是在时间属性上，快照融合操作是检索过程的主要时间开销，在长期多版本快照检索中仍然效率低下.

4 HCSIM 索引技术

4.1 HCSIM 基本原理

长期快照数据检索效率低下问题在文献[15-16]中有部分讨论，本文以二维索引结构为基础，进一步讨论提高长期 block-level 快照检索效率的方法. 首先给出具体的实例分析，设初始时刻为 T_1，源端的逻辑地址分别为 I，II，III，IV；数据均为 1，表达为 T_1｛I(1)，II(1)，III(1)，IV(1)｝. 在 T_2 时刻逻辑地址 I 内的数据由 1 变为 2，表示为 T_2｛I(1)—>I(2)｝，同理 T_3，T_4 时刻现场数据的变化轨迹分别为 T_3｛II(1)—>II(3)｝，T_4｛I(2)—>I(4)｝. 假设两个时间点之间正好是一个快照周期，产生的历史快照数据如图 6 所示.

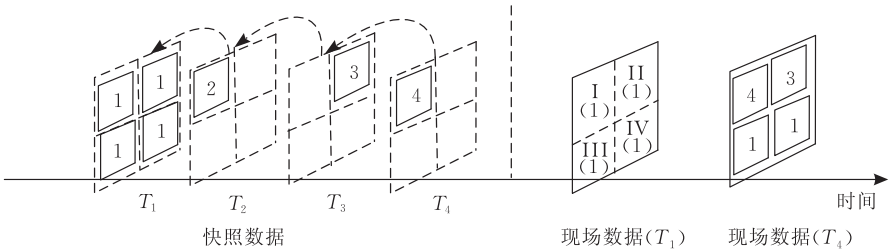

图 6 快照检索路径

如果要检索 T_4 时刻,逻辑地址 I,II,III,IV 的快照数据,需要遍历所有时间点的快照,检索的结果快照为 $T_4\{I(4)\}$,$T_3\{II(3)\}$,$T_1\{III(1),IV(1)\}$. 因此,长期多版本快照的检索顺序是从目标时间点开始,沿着时间轴向后检索. 这种检索顺序本文称为后向路经,提高长期快照数据检索效率的主要途径是缩短后向路经的长度.

可以进一步把后向路经的检索过程抽象为无环有向图的遍历过程. 其中图的源点表示第一个版本的快照索引文件,其它的结点表示增量快照产生的索引文件,有向边表示结点之间的依赖关系. 长期快照检索过程概括为从图中任何一个结点到源点的遍历过程. 图 7 中给出了简单的遍历示意图.

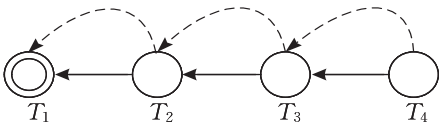

图 7　快照检索的无环有向图表示

根据 Block-level 快照数据分布"集中"的特点,不同版本的快照数据在逻辑地址空间内相对集中,可以把早期相邻版本的索引 Tree 进行快照融合,融合后的索引 Tree 仍然保持着索引间的依赖关系,形成分段、分层的索引结构 HCSIM(Hierarchical Clustering Snapshot Indexing Method). 上层的结点反映下层快照融合后的结果,融合过程仅在索引层面上进行,快照数据不发生变化.

图 8 给出了分段、分层索引结构示意图. 图中每个结点使用 $T_{[i,j]}$ 表示,i 表示层号;j 表示版本号. 上层结点的版本号等于合并周期内最大的版本号. 如 $T_{[2,2m+1]}$ 表达下层快照 $T_{[1,m+1]}$,…,$T_{[1,2m+1]}$ 的索引结点融合结果,同时有 $T_{[2,m]}$,$T_{[2,2m+1]}$,…,$T_{[2,m\times m]}$ 仍然保持索引之间的依赖关系.

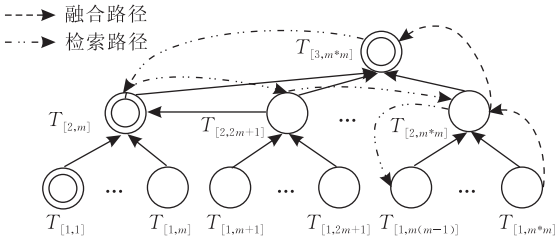

图 8　HCSIM 索引结构

HCSIM 构成了以最左边为源点的层次结构. 一次完整的遍历的过程是从当前结点开始,到达源点的寻径过程. 在 HCSIM 的实现过程中,根据层间融合发生的时间,可以进一步把 HCSIM 分为 Eager-Merge HCSIM(EHCSIM)和 Lazy-Merge HCSIM (LHCSIM)两种. EHCSIM 在每次快照数据备份过程中,把本次产生的索引 Tree 融合到父结点,融合过程直到根结点为止. EHCSIM 增加了索引的维护开销,但是可以有效缩短最坏检索路径长度,EHCSIM 检索路径长度可以表达为 $Length_{\text{EHCSIM}}$. LHCSIM 只有在本分段长度为阈值时才进行层间融合. Lazy-Merge 减少了索引生成时的融合操作,但是增加了最坏情况下的检索路径长度(m 表示分段的长度,即分段内结点的个数).

$$Length_{\text{EHCSIM}}=O(\log_m n+m) \tag{10}$$

$$Length_{\text{LHCSIM}}=O((m-1)\log_m n) \tag{11}$$

4.2　HCSIM 算法实现

在引入分层次的索引结构后,二维索引的基本操作原理不发生变化,但是在索引检索过程中需要结合 HCSIM 的结构特征,把原来版本间的顺序检索过程扩展为层内和层间两种检索顺序. 在 HCSIM 中查询版本为 i 的快照检索过程可以描述为:在版本号小于等于 i 的结点构成的子图中,从结点 i 达到根结点的遍历过程,最后把遍历路径上版本号不同的结点进行快照融合. 一种有效的实现方式是按照从顶至下的递归检索顺序. 具体算法如下所示.

算法 1.　HCSIM 实现.

i:目标快照版本号;H:HCSIM 索引结构;

V:结点集合;$Tree_{\text{TargetSnapshot}}$:目标快照索引 $Tree$;

$Version(node)$:结点 $node$ 的版本号;

$Root(H)$:HCSIM 层次结构 H 的根结点.

输入:i,H

输出:$Tree_{\text{TargetSnapshot}}$

Search(i,H){

1. if (H==NULL) return;
2. *AddNodeSet* (i,*Root*(H),V);

//从根结点 *Root*(H)开始遍历,生成结点集合 V;

3. Merge distinct version number nodes in V and set $Tree_{\text{TargetSnapshot}}$;
4. return $Tree_{\text{TargetSnapshot}}$;}

AddNodeSet 是递归遍历的过程,具体算法如下:

AddNodeSet(i,*Node*,V){

5. if ($node$==NULL) return;
6. if ($version(node)$==i){
7. add $node$ to V; return;}
8. if ($version(node)$>i)
9. *AddNodeSet*(i,*left-most child node*,V);
10. if($version(node)$<i){
11. add $node$ to V;
12. *AddNodeSet*(i,*right-cousin node*,V);}}

图 8 中给出检索版本号为 $m(m-1)$快照时的检索路径示意图，从图示中可以看出，HCSIM 算法的检索路径长度与版本长度由原来的线性关系变为对数关系，因此 HCSIM 可以有效缩短检索路径的长度. 在实际使用中，为了平衡索引的存储效率和检索效率，可以把下层大量的索引数据保存在外存中，根据需要调入内存；而上层索引数量相对较少，可以直接保留在内存中，提高快照检索、融合的效率.

5 性能分析

5.1 HCSIM 与灾难恢复

在灾难恢复中，通常使用 RTO 表示灾难恢复效率. RTO 表达从发生错误一直到恢复正常业务所需的时间间隔. 基本的灾难恢复策略包括数据回迁和业务迁移两种，现代存储技术中，基于 block-level 快照数据可以实现业务迁移操作，进行快速的灾难恢复，此时 RTO 可以表达为

$$RTO = T_{检测} + T_{检索} + T_{重启} \tag{12}$$

(1) $T_{检测}$表示从灾难发生到检测到数据丢失的时间；

(2) $T_{检索}$表示在存储端检索目标时间点的快照数据时间；

(3) $T_{重启}$表示源端业务状态迁移以及业务重启的时间；

$T_{检测}$与 $T_{重启}$往往与具体的业务和部署环境相关，因此在业务迁移恢复策略中决定恢复效率的主要因素是快照检索所需的时间.

5.2 实验环境

为了在实际应用环境中比较算法的性能，本文在 Windows 平台下，利用磁盘过滤驱动技术，实现逻辑卷层次的数据备份/恢复系统. 在源端通过监控源逻辑卷(逻辑分区)位图的变化(以 sector 为单位)，记录更新的数据块. 基本的复制策略是初始时刻产生源逻辑卷的全量快照，以后根据位图监控结果，复制变化的数据块. 存储端是多台存储服务器搭建的存储集群，根据不同的索引技术对多版本快照备份数据进行管理. 具体的实验环境为：源端是 XP 操作系统的 PC 机，配置为 Intel(R) Core(TM)2 CPU4300@1.8GHz/1.79GHz，1GB 内存，NTFS 文件系统，源逻辑卷为 20GB. 存储端 4 台存储服务器的具体配置为：两台 Inter(R)Core(TM) Duo CPU E4500 2.2GHz/2.19GHz，1GB 内存，160GB 硬盘，Realtek 100M(2)；两台 Inter(R) Core(TM) Duo CPU E4500 2.2GHz/2.19GHz，2GB 内存，250GB 硬盘，Realtek 100M(2). 实验数据集分别采用 NTFS 文件系统以及 Oracle 数据库应用背景下产生的多版本 block-level 快照备份数据.

5.3 HCSIM 与 Overlapping B+ Tree 比较

快照数据多版本检索是数据时空属性的检索过程，选择多版本数据库中通常采用的 Overlapping B+ Tree 进行比较，主要分析索引的维护效率、存储效率以及检索效率. 在实验中 Overlapping B+ Tree 以逻辑块号(实验中选取逻辑块长度为 4KB)和时间戳作为索引关键字. 在快照备份时为新的快照数据产生索引项，源卷中没有发生变化的数据直接引用以前版本的子树或分支. 在检索过程中 Overlapping B+ Tree 可以从每个版本的根结点直接读取该版本对应的全量快照.

利用 5.2 节介绍的备份/恢复系统，在 NTFS 文件系统正常工作的模式下，包括文档的读/写、视频文件的下载等操作，产生 11 个版本 block-level 备份数据. 图 9 中给出索引数据总量与版本数目的变化关系的比较. Overlapping B+ Tree 的索引数据量与备份数据总量相关. 实验中 t_1 产生全量快照，其它时间点产生增量快照，由于增量快照备份数据量相对较少，Overlapping B+ Tree 的索引数据总量变化不明显. HCSIM 索引技术产生的索引数据量包括版本内的索引数据量和由于分层引入的层间索引数据量两部分. 版本内利用了快照分布“连续”的特征减少索引数据量. 例如一次快照产生 100MB 的连续快照数据，在 HCSIM 索引结构中只需要一个索引项表示，而传统的 Overlapping B+ tree 则需要产生 25K 的索引项. 层间索引是记录融合后的结果快照数据，根据快照分布“集中”的特性，在快照融合处理中不同版本间逻辑地址相同的快照数据被覆盖掉，层间索引记录的快照数据量要远小于融合周期内各个版本快照数据量的总和. 因此，层间索引数据

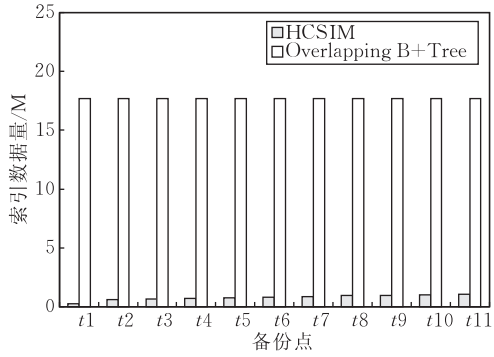

图 9 索引数据量比较

量小于底层索引数据量. HCSIM 索引技术综合利用了快照数据时空分布特性减少索引数据量.

图 10 给出两种索引的维护时间开销比较. 由于 Overlapping B+ Tree 在每次数据项插入时,都需要与前面所有的版本 Tree 进行分支共享、子树复制等操作,随着快照版本的增加,索引维护的时间逐渐增大. HCSIM 索引技术通过减少索引数据量降低索引的维护开销.

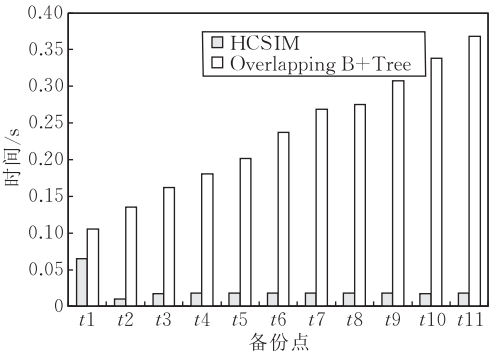

图 10　索引生成的时间开销比较

尽管 Overlapping B+ tree 构造时间长,但是 Overlapping B+ tree 通过静态维护整个时间点的全量快照索引,支持快速的查询. HCSIM 检索过程中伴随着快照融合过程,在检索少量版本快照时,效率低于 Overlapping B+ Tree,主要时间消耗发生在快照融合处理上. 本文提出 HCSIM 索引结构的基本目标就是降低索引之间的依赖关系,减少多版本快照间融合次数,提高长期快照的检索效率. 为了进一步描述 HCSIM 中检索路径的长度,引入平均路径长度 R. R 表示 HCSIM 中所有结点到达根结点的平均路径长度. 表示为式(13).

$$R=\sum_{i=1}^{n} r_i/n \tag{13}$$

式(13)中 r_i 表示 HCSIM 中结点 i 到达根结点的路径长度. 在 HCSIM 检索过程中,R 的物理意义表示在等概率索条件下,检索任意版本快照所需的平均快照融合次数.

表 2 中给出在分段长度 $m=5$ 时,R 与结点个数 n 的关系. 根据表 2,HCSIM($R=2, m=5$)最多可以管理 13 个版本的快照数据.

图 11 中给出 HCSIM 中在 $R=2, m=5$ 时,两种索引技术的检索效率比较情况. 在图 11 中可以看出 HCSIM 在版本数大于 5 时仍然具有较好的检索效率,因此在长期快照检索中,HCSIM 算法的检索效率仍然会高于 Overlapping B+ Tree 索引结构.

表 2　R 与结点个数的关系列表

平均路径长度 R	结点个数 $n(m=5)$
0	1
0.5	2
1	3
1.5	4
2	13
2.5	18
3	25

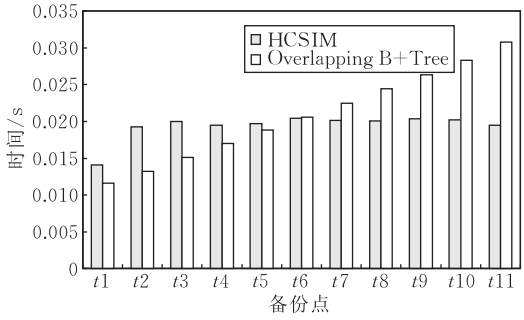

图 11　不同索引技术的检索效率

HCSIM 索引的层次结构随着版本数目的增长会发生变化. HCSIM 采用 Lazy-Merge 和 Eager-Merge 实现层次间的调整,根据式(10)、(11)、(13)可知 Eager-Merge 通过缩短 R 的长度,提高数据的检索效率,但是在每次生成快照索引时,都需要伴随着索引的融合过程. 图 12 给出了 Eager-Merge 策略下索引的维护时间开销与 Overlapping B+ Tree 的比较. 由图示可以看出,HCSIM 在引入 Eager-Merge 调整层次间索引结构时其索引维护开销仍然好于 Overlapping B+ Tree 结构.

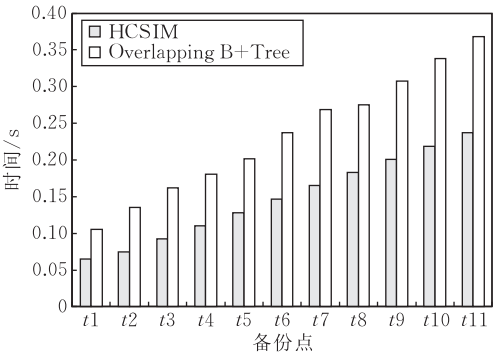

图 12　Eager-Merge 模式下索引维护效率比较

HCSIM 索引技术通过减少快照的融合次数,有效提高了长期快照的检索效率. 但是 HCSIM 是一种面向快照数据的检索模式,即查询某个时间点的所有的快照数据,而对于单一索引项的检索(基于 Time 和 Key 的检索模式),HCSIM 效率会低于传统的算法.

5.4 HCSIM 与其它快照索引技术比较

5.4.1 理论分析

目前 block-level 快照管理技术实现差异很大，本文不再一一进行比较，为了不失一般性，本文把目前常见的 block-level 快照管理技术抽象成 3 种基本的索引管理模式，与 HCSIM 索引模式进行定性比较、分析. 由于在实际应用中，快照索引数据量直接影响着检索效率、索引数据的存储效率以及系统的整体性能，下面把不同模式下的索引数据量作为比较参数，设时间点 i 生成的快照对应的索引数据量为 l_i，分别给出不同索引模式的索引数据量理论值.

间接依赖关系快照索引模式 ISIM（Indirect Snapshot Indexing Method）：这种索引模式根据快照生成时相对顺序，依次链接每个版本的快照索引，在快照检索时需要遍历所有时间点的快照索引. 这类快照管理技术广泛存在于快照文件系统和存储设备中[7,9]，此时索引存储效率最高，但是检索效率最低，仅支持有限长的版本的快照检索，索引的存储效率可以表示为 S_{ISIM}：

$$S_{\mathrm{ISIM}}=O\left(\sum_{i=1}^{n} l_i\right) \tag{14}$$

直接依赖关系快照索引模式 DSIM（Direct Snapshot Indexing Method）. 部分存储子系统在快照管理中，从创建快照开始，一直保留所有历史快照数据及对应的索引，直到该版本的快照删除为止，如 LVM 等. 这种模式下，快照检索时可以直接遍历该时间点的索引数据，具有最高的检索效率，但是索引的存储效率最低，可以表示为 S_{DSIM}：

$$S_{\mathrm{DSIM}}=O\left(\sum_{i=1}^{n}\sum_{j=1}^{i} l_j\right) \tag{15}$$

分段依赖关系快照索引模式 CSIM（Clustering Snapshot Indexing Method）. 一种折中的考虑是为固定分段的增量快照保存所有的历史数据[17]，这种技术在存储效率和检索效率取一次折中. CSIM 模式下索引数据量表示为 S_{CSIM}，其中 m 为分段的长度.

$$S_{\mathrm{CSIM}}=O\left(\frac{n}{m}\sum_{i=1}^{n} l_i\right) \tag{16}$$

本文提出的索引技术 HCSIM 引入了分层结构，索引数据量可以表示为 S_{HCSIM}，为了与 CSIM 进行统一比较，HCSIM 中取相同的分段长度 m，但实际的使用中可以灵活的配置，不会影响算法的可用性.

$$S_{\mathrm{HCSIM}}=O\left(\log_m n\sum_{i=1}^{n} l_i\right) \tag{17}$$

5.4.2 实验比较

在 5.2 节介绍的备份/恢复系统中，源逻辑卷运行于 Oracle 数据库工作背景下，产生 50 个版本 block-level 快照备份数据. 本文根据前 50 个版本快照索引信息，通过建立拟合函数，分析在长期、高频快照应用背景下不同索引模式的效率比较.

图 13、图 14 中给出平均每个版本的索引数量以及对应的全量快照索引数量，同时图中给出了函数拟合的结果. 通过拟合函数，本文给出在版本个数为 1000 时不同索引模式的效率比较. 图 15 给出 4 种索引模式的比较结果，图 16 进一步给出 HCSIM 在不同分段长度下的性能比较（图 15、图 16 中 y 轴取以 2 为底的对数函数）.

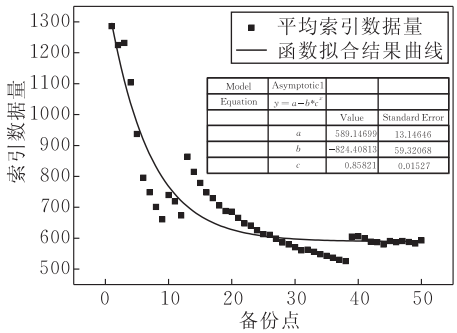

图 13 每个版本平均索引数据量

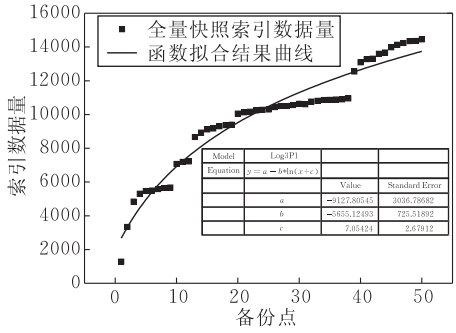

图 14 每个版本全量快照索引数据量

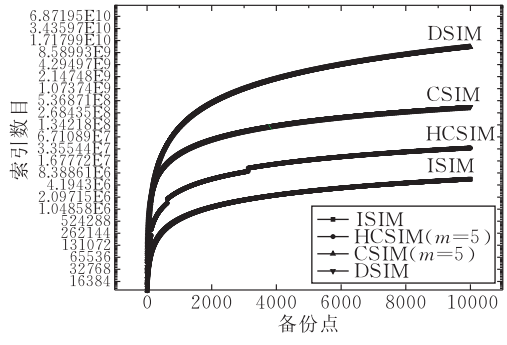

图 15 4 种快照索引模式的比较

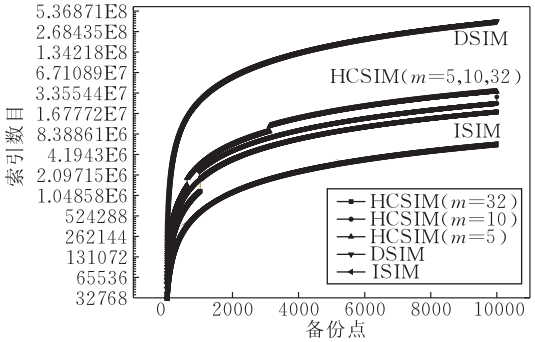

图 16　m取不同值时 HCSIM 的存储效率比较

通过上述比较、分析可以得出 HCSIM 索引模式比目前常用的 DSIM、CSIM 索引模式进一步减少索引数据量，而检索路径长度仅呈对数关系增长. 从折中角度分析 HCSIM 在 block-level 快照数据管理中可以取得索引存储效率和检索效率更好的平衡关系.

6　总　结

高频快照技术应用于备份时可以为物理错误和人为错误提供细粒度的数据恢复机制，构建可靠数据存储环境. 长期、高频快照检索效率低下是由于快照数据共享时产生复杂的依赖关系形成的，成为高频快照检索的瓶颈问题，目前逐渐引起国内外研究机构的重视. 本文提出的 HCSIM 索引技术及分析方法，可以为长期、高频 block-level 快照管理提供一种解决问题的思路.

参　考　文　献

[1] Patterson D, Brown A. Broadwell P et al. Recovery oriented computing (ROC): Motivation, definition, techniques, and case studies. University of California at Berkeley, Computer Science Technical Report: UCB/CSD-0201175, 2002

[2] Patterson H, SnapMirror R. File system based asynchronous mirroring for disaster recovery//Proceedings of the Conference on File and Storage Technologies. Monterey, CA, 2002: 117-129

[3] Cornell B, Dinda P A, Bustamante F E. Wayback: A user-level versioning file system for Linux//Proceedings of the 2004 USENIX Annual Technical Conference. Boston, 2004: 19-28

[4] Santry D J, Feeley M J, Hutchinson N C et al. Deciding when to forget in the elephant file system. Operating Systems Review, 1999, 34(5): 110-123

[5] Michail D F, Angelos B. Clotho: Transparent data versioning at the block I/O level//Proceedings of the 21st IEEE Conference on Mass Storage Systems and Technologies/12th NASA Goddard Conference on Mass Storage Systems and Technologies. Greenbelt, Maryland, USA, 2004: 315-328

[6] Morrey III C B, Grunwald D. Peabody: The time traveling disk//Proceedings of the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies. San Diego, California, USA, 2003: 241-253

[7] Yang Q, Xiao W J, Ren J. TRAP-array: A disk array architecture providing timely recovery to any point-in-time//Proceedings of the 33rd Annual International Symposium on Computer Architecture (ISCA'06). Boston, USA, 2006: 289-300

[8] Hitz D, Lau J, Malcolm M. File system design for an NFS file server appliance//Proceedings of the USENIX Winter Technical Conference. San Francisco, CA, 1994: 235-245

[9] Peterson Z, Burns R. Ext3cow: A time-shifting file system for regulatory compliance. ACM Transactions on Storage, 2005, 1(2): 190-212

[10] Xiang Xiao-Jia, Shu Ji-Wu, Zheng Wei-Min. An efficient fine granularity multi-version file system. Journal of Software, 2009, 20(3): 754-765(in Chinese)
(向小佳，舒继武，郑纬民. 一种细粒度高效多版本文件系统. 软件学报，2009，20(3)：754-765)

[11] Shaull R, Shrira L, Xu H. Skippy: A new snapshot indexing method for time travel in the storage manager//Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data. Vancouver, Canada, 2008: 637-648

[12] Tzouramanis T, Manolopoulos Y, Lorentzos N. Overlapping B+-trees: An implementation of a transaction time access method. Data and Knowledge Engineering, 1999, 29(3): 381-404

[13] Keeton K, Santos C, Beyer D et al. Designing for disasters//Proceedings of the 3rd Conference on File and Storage Technologies. San Francisco, CA, 2004: 59-62

[14] Wu G J, Yun X C, Wang S P. Design and implementation of multi-version disk backup data merging algorithm//Proceedings of the 9th International Conference on Web-Age Information Management. Zhangjiajie, China, 2008: 526-531

[15] Shrira L, Xu H. Snap: Efficient snapshots for back-in-time execution//Proceedings of the 21st International Conference on Data Engineering. Washington, DC, USA, 2005: 434-445

[16] Shrira L, Xu H. Thresher: An efficient storage manager for copy-on-write snapshots//Proceedings of the 2006 USENIX Annual Technical Conference. Boston, MA, USA, 2006: 57-70

[17] Xiang Xiao-Jia, Shu Ji-Wu, Yu Hong-Liang. Design and implementation of a cluster virtualization based fine snapshot system. Journal of China Institute of Communications, 2009, 30(2): 28-33(in Chinese)
(向小佳，舒继武，余宏亮. 基于集群虚拟化的高精度快照的设计与实现. 通信学报，2009，30(2)：28-33)

WU Guang-Jun, born in 1981, Ph. D. candidate. His research interests include disaster tolerant, massive storage and network security.

YUN Xiao-Chun, born in 1971, professor, Ph. D. supervisor. His research interests include network and information security.

FANG Bin-Xing, born in 1960, professor, Ph. D. supervisor, member of Chinese Academy of Engineering. His current research interests include computer architecture, information security and computer network.

WANG Shu-Peng, born in 1980, Ph. D.. His research interests include disaster tolerant and network security.

YU Xiang-Zhan, born in 1973, Ph. D., associate professor. His research interests include network survivability and disaster tolerant.

Background

This paper is supported by the National High Technology Research and Development Program (863 Program) of China under grant Nos. 2007AA01Z406, 2009AA01A403, 2009AA01Z437 and the National Basic Research Program (973 Program) of China under grant No. 2007CB3111003.

With the explosive growth of information service and maturity of disk techniques, the data protections have became top priority for business organizations and individuals. The block-level snapshot-based incremental backup techniques support online replication and improve backup frequency and recovery efficiency than traditional daily incremental backup technique. Despite of these importances, the challenge is that there is no efficient access method for the long-lived online multi-version snapshots. The problem is that the frequent incremental snapshot not only produces large volume size but also generates complex dependency between versions. Currently, the multi-version snapshots are maintained in recovery log or in dedicated snapshot system. These strategies are not fit for long-lived block-level multi-version snapshot management.

This paper proposes a general purpose Hierarchal Clustering Snapshot Indexing Method (HCSIM). The authors first analyze block-level protection techniques by mathematical model and get the "continuity" and "localization" distribution features. Adapting the these features, HCSIM uses an interval indexing entry to index a continuous chunk of data and hierarchical snapshot consolidating method to shorten traveling path in the long-live snapshot retrieval process. HCSIM can dramatically increase indexing storage and retrieval efficiency than traditional multi-version DB indexing structure Overlapping B+ Tree. At the meantime, HCSIM can achieve better balance between storage and query efficiency than nowadays block-level snapshot indexing method.

The work of this paper belongs to the project "The Key Techniques of High-end Massive Storage System and Application Demonstration for Next Generation". The authors design and implement the long-lived block-level snapshot indexing method to directly support key techniques for data backup and data protections by software. They also present a basic idea for application independent data protection technique in third-party disaster tolerant application.