ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.20, No.3, March 2009, pp.754−765
doi: 10.3724/SP.J.1001.2009.03182
© by *Institute of Software*, *the Chinese Academy of Sciences*. All rights reserved.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种细粒度高效多版本文件系统*

向小佳[1,2+], 舒继武[1,2], 郑纬民[1,2]

[1](清华大学 计算机科学与技术系,北京 100084)
[2](清华信息科学与技术国家实验室(筹),北京 100084)

An Efficient Fine Granularity Multi-Version File System

XIANG Xiao-Jia[1,2+], SHU Ji-Wu[1,2], ZHENG Wei-Min[1,2]

[1](Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)
[2](Tsinghua National Laboratory for Information Science and Technology (TNList), Beijing 100084, China)
+ Corresponding author: E-mail: xiangxj05@mails.tsinghua.edu.cn

Xiang XJ, Shu JW, Zheng WM. An efficient fine granularity multi-version file system. *Journal of Software*, 2009,20(3):754−765. http://www.jos.org.cn/1000-9825/3182.htm

Abstract: A snapshot-based fine granularity versioning technique is presented to retain history data only for a single directory or a single file, and bring flexibility to multi-version file systems. Adopting the strategy to search in name space and version space separately, this paper also presents backward inheriting path-finding mechanism in version space. This mechanism is beneficial to the performance and management, because it can utilize the coupling relationship between versions to optimize the data layout of versions and build hierarchy in version space to accelerate the path-finding procedure. In addition, fast index structures for directory versions and file versions are designed. This prototype file system——THVFS can achieve both good performance and high availability with these technologies mentioned above. The experimental results show that the average time of searching old versions in THVFS was reduced by 34.4% than in ext3cow, the famous multi-version file system. In the trace experiment, the average read response time in THVFS was 12% less than in ext3, and only 80% extra space was needed to retain all history data when snapshots are taken every 72 minutes in THVFS.
Key words: version; backward inheriting path-finding; red black tree embedded in inode; red black tree with weight and link; red black tree lock

摘 要: 提出了基于快照的细粒度版本技术,能够克服已有多版本文件系统无法仅对系统局部目录或文件保留版本的缺点,增加了系统的灵活性;提出了版本空间的反向继承寻径,使用名字与版本独立的检索方案,可以充分利用版本间的相关性,优化数据物理布局,建立版本间的层级结构,既便于管理,又提高系统性能;设计了分别针对目录版本和文件版本的快速索引结构.评测结果表明,THVFS 的历史数据访问性能较著名的多版本文件系统 ext3cow 提高了 34.4%;Trace 实验中,相对于 ext3,THVFS 的读性能提高了 12%,同时,在每 72 分钟生成一次快照的高频率下,维护

---

* Supported by the National Natural Science Foundation of China under Grant No.60473101 (国家自然科学基金); the National Basic Research Program of China under Grant No.2004CB318205 (国家重点基础研究发展计划(973)); the Program for New Century Excellent Talents in University of China under Grant No.NCET-05-0067 (新世纪优秀人才计划)

Received 2007-06-29; Accepted 2007-09-30

© 中国科学院软件研究所 http://www.jos.org.cn

所有历史版本仅需要 80%的额外空间.

关键词: 版本;反向继承寻径;Inode 内嵌式红黑树;带权重线索红黑树;红黑树锁

中图法分类号: TP301 **文献标识码:** A

数据存储需求增长的同时,保证数据可靠的需求也越发紧迫,推动了多版本文件系统的出现与发展.多版本文件系统中,重要的目录或文件保留多个版本可以避免用户误操作和系统故障所导致的数据丢失,从而提供了可靠的数据存储服务,付出的代价是更多的存储空间.磁盘等存储介质容量的增加以及价格的下降,使多版本文件系统的流行成为可能.

多版本文件系统中,版本是目录或文件不同时期的数据映像,是系统的最小存储单元.目录或文件是由处于不同时期的目录版本或文件版本组成的集合.

多版本文件系统的核心是版本的组织管理技术,主要包括版本日志技术和基于 COW(copy on write)和 ROW(redirect on write)的快照技术.版本日志技术通过将数据记录在 Undo log 或 Redo log 中来保留历史数据,通过日志滚动操作来获取指定版本的数据.采用版本日志的多版本文件系统的典型代表有 Wayback[1],Elephant[2].Wayback 中,对每个目录和文件的历史修改操作都被记录在日志中.由于其日志实际上是拥有特殊文件名的日志文件,因此破坏了系统的命名空间.Elephant[2]使用 inode log 结构将代表文件所有历史时期版本的 inode 以版本日志的形式组织起来,便于管理和查找.快照技术实质上是轻量级的数据快速备份技术,通过复制文件系统元数据来保留系统中目录和文件的历史映像.采用快照技术的典型代表有 AFS[3],Plan-9[4],WAFL[5],Petal[6],Venti[7]和 Ext3cow[8]等.不论采用哪种技术的多版本文件系统,在可用性和性能上都存在缺陷,具体表现在如下几个方面:

首先,由于对细粒度版本机制的支持需要更复杂的元数据结构并带来额外的操作与管理开销,已有的基于快照的多版本文件系统都采用了粗粒度的版本生成与管理策略.例如,AFS,Plan-9,WAFL 和 Ext3cow,在执行快照操作时会生成整个文件系统的映像,所有目录和文件的版本数目、生成时间完全一致.粗粒度的缺点是:耗费存储资源,并且不能满足用户仅对部分目录或文件保留版本的需求,不利于管理.

其次,不能充分利用版本间的相关性.在多版本文件系统中,同一目录或文件的不同版本间有紧耦合性,相近时期的父目录版本与子目录版本、文件版本间也有相关性.利用这些相关性能够提升性能,反之则会带来不利影响.以 Elephant 和 CVFS[9]为代表的文件系统采用单一目录版本策略,不同时期的文件版本对应唯一的父目录版本,丢失了目录版本时间信息,加重了目录内检索的负担.CVFS 为提高单个版本的检索性能,打破了同一文件不同版本间的紧耦合性,以文件名和版本生成时间作输入参数,使用类哈希函数为版本建立索引.这不利于版本的管理,例如,删除同一文件的所有版本时系统不得不一一查找分散于不同位置的版本,既耗时也容易与其他并发检索操作发生冲突,甚至引起死锁.Ext3cow 通过在寻径过程中保留父目录的版本生成时间并据此指导子目录版本的搜索,能够部分利用版本间的相关性;但父目录版本的时间跨度大,粒度粗,使用其版本生成时间指导搜索不够精确,速度慢.

最后,检索版本的性能不佳.在传统文件系统中,访问一个目录或文件需要检索目录名或文件名;在多版本文件系统中,访问目录版本和文件版本既需要检索目录名或文件名,还需要检索版本号.由于版本数量随着时间的增长而线性增加,多版本文件系统的检索负担较传统文件系统有翻倍的增加.在访问较古老的历史版本时,系统往往需要遍历之前的所有版本,形成性能瓶颈.基于版本日志技术的多版本文件系统,如 Wayback,访问版本需要单向翻滚日志,Redo 或 Undo 日志记录的操作,降低了性能.Ext3cow 为改进性能,采用限长分段链表的形式来组织版本,但线性搜索的瓶颈仍旧存在,性能提高得有限.Versionfs[10]采用在普通文件系统之上增加中间层以支持版本操作,虽然增加了移植的灵活性,但中间层的增加也带来了额外消耗.

综上所述,目前没有一种多版本文件系统,能够同时具备好的可用性和优良的性能.本文提出了一种基于快照的细粒度版本技术,既能够克服已有多版本文件系统不能对局部目录和文件保留版本的缺点,使用户能够灵活配置版本生成策略,又能够利用快照技术,将耗时的元数据和数据复制工作推迟到真正需要修改时再进行,降

© 中国科学院软件研究所 http://www.jos.org.cn

低性能上的额外开销.提出了版本空间的反向继承寻径策略,在名字空间与版本空间中独立建立索引,不但可以充分利用同名版本间的紧耦合性,将逻辑上相关的各同名版本通过索引结构组织在一起,并尽量存放在相近的物理位置,而且便于在版本空间中建立层级结构,利用相近时期所创建的父目录版本与子目录版本、文件版本之间的相关性,提高了系统的整体性能以及版本的易管理性.设计了基于红黑树的目录版本和文件版本的快速索引结构,能够消除已有多版本文件系统检索版本的性能瓶颈.

在上述关键技术基础上实现的原型系统 THVFS(Tsinghua versioning file system),好的性能与可用性兼备.实验结果表明,其历史版本访问性能较著名多版本文件系统 ext3cow 提高了 34.4%.Trace 实验中,相对于 ext3,THVFS 的读性能提高 12%;与多版本文件系统 wayback、ext3cow 相比,THVFS 在写性能上的额外消耗最少;同时在每 72 分钟生成一次快照的高频率下,维护所有历史版本仅需 80%的额外空间.

本文第 1 节介绍 THVFS 的体系结构.第 2 节详细论述系统中采用的关键技术.第 3 节阐述实验方法以及具体结果.最后给出结论.

1 软件体系结构

THVFS 在 Linux 平台上实现,按功能划分,其软件模块(如图 1 所示)包括位于内核空间的系统调用(Sys call)、虚拟文件系统层(VFS layer)、文件系统功能子层(function layer)、策略管理器(policy manager)、快照驱动(snapshot driver)、日志设备(JBD module)、设备驱动(block device driver)以及位于用户空间的应用工具(utility).图中非阴影部分使用了 Linux 操作系统中的现有模块;我们自己的设计与实现集中在阴影部分.

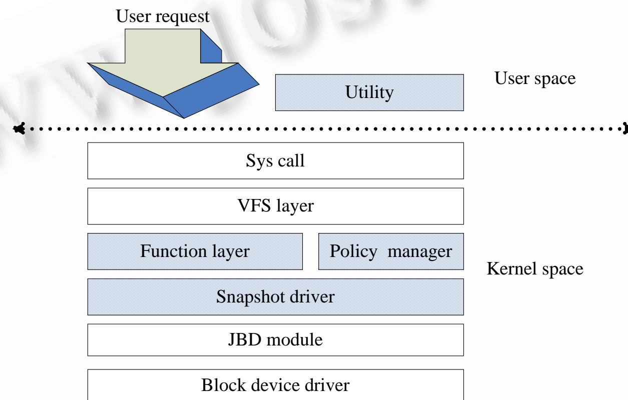


Fig.1 Modules of THVFS

图 1 THVFS 软件模块图

文件系统功能子层实现了所有关于文件和目录的操作,例如创建、删除、读写、获取属性以及访问权限控制等等,是 THVFS 的核心部分.该层沿用了传统文件系统的关键数据结构 inode 和 dentry,并针对版本作了相应的扩充.

Inode 的扩充:Inode 是文件系统中表示文件或目录的数据结构.在传统文件系统中,一个目录或一个文件仅对应一个 inode;在多版本文件系统中,一个目录或一个文件对应多个 inode,原因是:多版本文件系统中的目录和文件有多个版本,每个版本都拥有独立的 inode.为支持多版本,文件系统 inode 数据结构中至少还需要包含版本生成时间信息,后文称为 epoch[8].epoch 用来存放版本生成时的操作系统时间,一个文件或目录的不同版本对应不同的 epoch,越早生成的版本其对应的 epoch 值越小,反之亦然.THVFS 中的 inode 数据结构还增加了:(1) snapepoch 域,用来存放最近一次对本目录或文件执行快照操作的时间,用来支持细粒度快照;(2) 索引结构(index structure)域,存放用来索引其他版本的指针以及相关状态,以支持版本快速索引.

Dentry 的扩充:dentry 代表文件系统中的目录项.为了支持在版本空间建立层级结构,THVFS 借鉴了 ext3cow 的设计,在 dentry 数据结构中增加了生命周期二元组〈death_epoch,birth_epoch〉[8];birth_epoch 代表

© 中国科学院软件研究所 http://www.jos.org.cn

dentry 被创建时的系统时间,即出生时间;death_epoch 代表 dentry 被删除时的系统时间,即死亡时间.

策略管理器主要用来管理控制 THVFS 中版本生成的模式、粒度.版本生成的模式包括手动和自动两种.手动模式根据用户自己的需求来触发快照操作,生成版本.自动模式会按照事先的配置定期生成相应的版本.通过对版本生成粒度的控制,THVFS 既能对整个文件系统中的所有目录和文件同时生成新的版本,也能仅对指定子目录甚至单个文件生成新的版本.

快照驱动是 THVFS 版本生成的底层机制.THVFS 中执行快照时仅保留必要的时间信息,而将数据甚至元数据的复制工作推迟到真正需要修改时再进行,具有速度快、效率高的特点.

应用工具的主要作用包括:(1) 辅助用户进行 THVFS 的相关配置;(2) 提供手动生成指定范围目录和文件新版本的界面;(3) 对系统进行监测、调试.

2 关键技术

2.1 基于快照的细粒度版本生成技术

细粒度版本生成技术的基本思想包含如下两点:

(1) 快照生成时在系统元数据相应结构中记录快照的位置和时间.THVFS 中,快照的时间记录在执行快照操作的目录或文件当前版本的元数据中,即每个目录或文件当前版本 inode 的 snapepoch 域中.例如:对整个文件系统所作全局快照的快照时间被记录在系统根目录当前版本 inode 的 snapepoch 中.

(2) 对目录或文件的当前版本进行修改时,根据该目录或文件所处位置以及当前版本的时间信息判断是否应该生成新版本,如果需要,则复制相应的元数据和数据.这是细粒度版本生成技术的难点,原因是:一个目录或文件同时是其父目录、祖父目录以及根目录等祖先的子孙,在任一祖先上所作的快照都会对该目录或文件的版本产生影响.解决的方法是:判断一个目录或文件是否应该生成新版本需要依序遍历其所有祖先的当前版本,对各自的快照时间进行调整和比较.基本原则是:子目录或文件当前版本的快照时间应该晚于或等于父目录当前版本的快照时间;如果被修改目录或文件当前版本的生成时间(记录在 inode 的 epoch 域中)早于该版本的快照时间,说明最近一次快照操作后该版本还未被修改,则需要生成新版本,反之则不生成.

细粒度版本生成的具体流程是:细粒度快照在某个目录或文件上执行时,首先,将当前时间记录在全局变量 superepoch 和该目录或文件当前版本 inode 的 snapepoch 中.然后,当需要修改该目录或文件时,在文件系统层级结构中,自顶向下执行由根目录当前版本到该目录或文件当前版本的寻径,并作快照时间的调整(根目录除外).以目录 B 的当前版本 B*为例,设 B*的快照时间为 snapepochB,其父目录的当前版本为 A*且其快照时间为 snapepochA,则调整 snapepochB 的方法用公式表示为:snapepochB=MAX(snapepochA,snapepochB).最后,当该目录或文件的当前版本快照时间结束调整操作后,比较其 inode 中的 epoch 与 snapepoch 值,如果 snapepoch>epoch,说明最近一次快照操作后该版本还未被修改,旧数据应该得到保留,通过复制该目录或文件当前版本的元数据以及被修改的数据,生成新的版本.

细粒度版本生成技术的主要时间消耗在遍历过程中,如果每次对目录和文件的修改都需要遍历调整快照时间,将会给系统性能带来不利影响,优化方法是:在遍历过程中,除了调整快照时间外,还需要在各个目录和文件当前版本的 inode 中缓存系统当前的 superepoch 值;同时,再次执行遍历过程前,首先判断目标目录或者目标文件当前版本 inode 中缓存的 superepoch 是否过期,如果不过期,说明自上次遍历后系统中还没有触发新的快照操作,可以跳过遍历,直接进行后续比较与修改.

2.2 二维检索中的反向继承寻径策略

THVFS 采用了二维检索,其基本思想是:名字空间与版本空间的检索操作相互独立,名字空间的索引结构与版本空间的索引结构相互独立.

二维检索中版本空间索引结构的特征是:同一目录或文件的不同版本组织在一起,并尽量存放在相近的物理位置;同时,相近时期所创建的子目录版本或文件版本与父目录版本之间有从属关系,形成了版本空间的层级

© 中国科学院软件研究所 http://www.jos.org.cn

结构.图 2 示例了版本空间的层级结构.图中,横坐标代表时间;纵坐标代表目录层级;方框代表一个目录版本,并且同一层的所有目录版本按时间顺序串联成一条版本链表;t^*代表当前时间;⟨t_{death},t_{birth}⟩为目录版本的生命周期二元组,t_{death} 代表该目录版本的被删除时间,t_{birth} 代表该目录版本的被创建时间.父子目录版本间的相关性体现在:父目录一个版本的生命周期包含其子目录若干个连续版本的生命周期,即父目录一个版本对应其子目录版本链表中的一个子链表.例如,图 2 中,父目录 A 的一个版本的生命周期为⟨t^*,$t0$⟩,包含其子目录 B 的 3 个连续版本的生命周期⟨t^*,$t18$⟩,⟨$t18$,$t9$⟩和⟨$t9$,$t0$⟩.

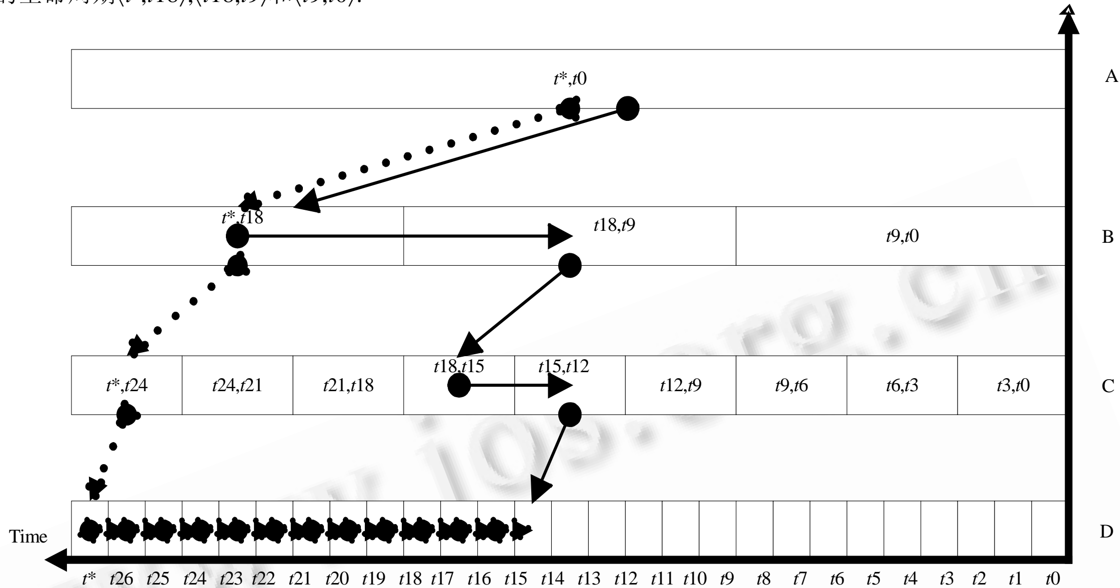


Fig.2 Comparison between two pathfinding method (t^* denotes current time)

图 2 寻径方法的对比(t^*代表系统当前时间)

反向继承寻径策略能够有效利用版本空间的层级结构,加速文件系统中定位指定目录/文件版本的过程.以图 2 为例,设系统需要以顶层目录 A 的唯一版本为起点,搜索并访问底层目录 D 的生命周期为⟨$t15$,$t14$⟩的版本(后文中简记为 v).传统多版本文件系统在版本空间中的定位过程是:以顶层目录版本为起点,定位到下一层目录版本链表的表头所代表的版本,再以该版本为起点继续向下层定位.逐层递进,一直到达 v 所在版本链表的表头.最后在该版本链表中检索 v.示例路径如图中虚线所示,一共访问了从 A~D16 个版本.反向继承寻径机制的思想是:以 v 的生命周期为依据,先在当前层版本链表中检索,寻找生命周期包含 v 的那个版本,再以该版本为起点按照其生命周期的范围定位到下一层版本链表中的相应子链表.然后在该子链表中重复上述定位过程,逐层递进,一直检索到 v 为止.示例路径如图中实线所示,一共访问了从 A~D 共 6 个版本.

通过对比可知,反向继承寻径机制能够直接利用最底层目标目录或目标文件版本的生成时间减少定位过程中需要访问的版本数,进而缩短了在版本空间层级结构中的定位时间.

2.3 版本快速索引

THVFS 中使用基于红黑树的数据结构来建立版本快速索引,提出两种方案:inode 内嵌式红黑树和带权重线索红黑树.前者的优点是数据都存储于外存,不占用内存空间,但检索中需要执行较多访问外存的操作,速度慢;后者的优点是减少了访问外存的操作,速度快,但需要占用额外的内存空间.多版本文件系统中,目录与文件的访问特点是:对目录的访问多是读取,修改少;而文件修改频繁,版本更新快,对检索性能要求高.综合考虑时间和空间的要求,我们使用 inode 内嵌式红黑树来建立目录版本索引,而使用带权重线索红黑树来建立文件版本索引.后文中称树的叶结点为外部结点;非叶结点为内部结点.

© 中国科学院软件研究所 http://www.jos.org.cn

2.3.1 索引结构

inode 内嵌式红黑树是用来检索目录版本的元数据索引结构,是普通红黑树结构在多版本文件系统中的典型应用.其上的每一个内部结点与目录的某个具体版本一一对应,其数据结构被包含在代表该目录版本的 inode 中,存放于外存,主要用来存储以下信息:结点关键值、指向父子结点的指针以及本结点的状态(颜色属性等).结点关键值是检索的依据,实现中设定为结点所对应目录版本的生成时间.外部结点是傀儡(dummy)结点,只为维持红黑树的性质而存在,没有对应的实体.

带权重线索红黑树是用来检索文件版本的元数据索引结构.其上的每一个内部结点都是建立在内存中的索引结点,数据结构中包含线索、指向父子结点的指针以及本结点的状态(颜色属性等),但不包含关键值;其上的每一个外部结点与文件的某个具体版本一一对应,数据结构被包含在代表该文件版本的 inode 中,存放于外存,主要用来存储以下信息:结点关键值、指向父结点的指针和权重,结点关键值设定为结点所对应文件版本的生成.

权重代表外部结点的重要程度,用来构造红黑树外部结点权重链表,权重最大的结点位于表头,最小的结点位于表尾.为增加系统的可靠性和灵活性,我们在生成和删除文件版本时,除了将该版本从红黑树索引中插入或删除以外,还会在红黑树外部结点权重链表中执行相应操作.外部结点权重链表可以作为红黑树索引的补充:当红黑树索引意外失效时,用户仍然可以通过权重链表访问相应的文件版本.权重的具体含义可由用户自己设定,例如:可以取外部结点被访问的次数作为权重.

线索是红黑树内部结点中的特殊指针.一个内部结点的线索指向其在红黑树中序遍历过程中的前驱和后继,并且其前驱和后继都是外部结点.增加线索的作用是赋予内部结点以关键值,原因是:内部结点与文件版本间无对应关系,而外部结点有,同时,系统中的关键值设定为文件版本的生成时间,所以只有外部结点才包含关键值,内部结点无关键值.但红黑树检索操作需要读取内部结点的关键值,作为与待检索关键值比较的对象,通过线索访问内部结点前驱或后继的关键值可以解决此问题.

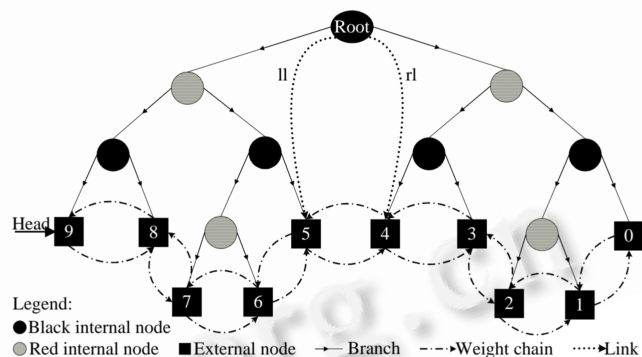


Fig.3 Red black tree with weight and link

图 3 带权重线索红黑树

图 3 为带权重线索红黑树的示例.图中以 head 为头的双向链表为按权重排序的外部结点权重链表,表头是最大权重外部结点.外部结点中的数字代表关键值,即外部结点所代表文件版本的生成时间.本例中权重与关键值相等.图中由 root 内部结点引出的带箭头的虚线代表 root 的线索:标注为“ll”的虚线代表root的左线索,指向root在中序遍历过程中的前驱;标注为“rl”的虚线代表右线索,指向 root 在中序遍历过程中的后继.以检索关键值为3的结点为例,由于root没有关键值与之比较,所以需要将其左(右)线索指向的外部结点中的关键值 5(4)读入,然后再比较.如果被访问结点关键值恰与读入关键值相等,则可以直接读入外部结点.如果不等,则转向相应子树继续检索.

2.3.2 并发访问和一致性问题

多用户同时使用元数据索引结构访问目录或文件时,不加控制会使读写出错甚至破坏索引结构,这称为并发访问问题.

我们设计了 RBTL 锁(红黑树锁),既可以保证在红黑树中执行检索、插入和删除操作的顺序性,又能够保证各操作的充分并行,进而解决了并发访问问题.按照权限高低排序,RBTL 锁共包含:写锁,记为 WriteLock;颜色锁,记为 ColorLock;读锁,记为 ReadLock.3 种锁之间有着不同的兼容关系,可以互相转换.用户在对红黑树中某个结点执行操作前必须先申请对应权限的锁,得到锁后才能执行后续操作,否则,必须等待其他用户释放锁.

© 中国科学院软件研究所 http://www.jos.org.cn

RBTL3 种锁分别对应不同的读写权限,原因是:红黑树的不同操作及操作的不同阶段需要的读写权限不同.我们将读写权限分为 6 种,其集合为{R_{key},R_{color},R_{link},W_{key},W_{color},W_{link}},其中元素分别代表读取结点关键值的权限以及读取结点颜色属性的权限、读取结点中指针的权限、修改结点关键值的权限、修改结点颜色属性的权限、修改结点中指针的权限.检索操作只需要读取相关结点,其所需权限集合为{R_{key},R_{link}},对应于 ReadLock.插入与删除操作既需要读权限,也需要修改权限.按照它们对红黑树结点的修改方式可分为两类:(1) 修改结点颜色属性的操作,其所需权限集合为{R_{key},R_{color},R_{link},W_{color}},对应于 ColorLock,该操作不会改变树体结构,且出现的频率较高;(2) 修改树体结构的操作,其所需权限集合为{R_{color},R_{link},W_{key},W_{color},W_{link}},对应于 WriteLock,该操作的出现频率较低.

Table 1 Compatibility of the locks

表 1 锁的兼容性

	ReadLock	ColorLock	WriteLock
ReadLock	1	1	0
ColorLock	1	0	0
WriteLock	0	0	0

RBTL 的 3 种锁的兼容矩阵见表 1,表中 1 代表兼容,0 代表不兼容.对红黑树的结点而言,兼容的锁可以同时持有,不兼容的锁则是互斥的.不同读写权限间存在读-写冲突和写-写冲突是造成不兼容的原因.以 ReadLock 与 ColorLock 间的兼容关系为例,因为两者对应的权限集合分别为{R_{key},R_{link}}和{R_{key},R_{color},R_{link},W_{color}},两集合间不存在任何冲突,所以 ReadLock 与 ColorLock 是兼容的;而 ReadLock 与 WriteLock 所对应的权限集合之间存在 R_{key}-W_{key} 和 R_{link}-W_{link} 冲突,所以 ReadLock 与 WriteLock 是不兼容的.

红黑树的各种操作对锁的处理如下:

检索操作按照从根至叶的顺序申请各结点的 ReadLock,且遵循典型的树锁(TL)申请原则:在申请某非根结点的 ReadLock 之前,必须已经拥有其父亲结点的 ReadLock;当获取结点的 ReadLock 之后,立即释放其父亲结点的 ReadLock.插入和删除操作在起始检索阶段按照从根至叶的顺序申请各结点的 ColorLock,申请成功后保持锁的持有状态一直到确定不再访问相应结点.持有 ColorLock 既可以避免与其他插入或删除操作因同时修改结点而发生冲突,又能够保证与检索操作并发执行,这是由锁的兼容性决定的.当插入和删除操作中需要修改树体结构时,将相应结点上已持有的 ColorLock 转换为 WriteLock.

由于操作系统故障等原因,对元数据索引结构的操作可能会异常终结,残留的部分操作结果会破坏文件系统的一致状态,这称为一致性问题.我们使用 Linux 内核中的 jbd 模块来实现红黑树插入、删除和检索操作的事务化.当操作异常终结时,启动 restart 流程,根据 jbd 事务日志记录的内容恢复文件系统的一致状态.

3 性能评测

3.1 测试方法与实验平台

首先,为测试并比较不同多版本文件系统中的版本机制对文件系统总体性能的影响,我们采用宏测试基准程序 postmark,在测评不同文件数和事务数时,传统文件系统 ext3、多版本文件系统 THVFS、ext3cow 和 wayback 完成整个测试流程的总时间.其次,为测量版本快速索引对多版本文件系统中特有的历史版本访问性能的影响,我们修改了微测试基准程序 lmbench[11]以增加对版本访问操作的测试.测试比较了在多版本文件系统 wayback,THVFS 和 ext3cow 中对文件的某一指定版本执行属性获取操作(stat)的响应时间.再次,为评价反向继承寻径策略,我们进一步扩充了 lmbench,使其能够创建不同规模的版本空间层级结构,记录、计算并比较采用不同寻径策略访问层级结构中最底层目录或文件版本的平均时间.该测试分别在都采用了二维检索并构建了版本空间层级结构的 THVFS 和 ext3cow 中进行.最后,我们使用 Berkeley trace RES[12]来评测多版本文件系统在真实使用环境中的性能,并比较不同版本生成技术的空间消耗.该测试分别在传统文件系统 ext3、多版本文件系统 THVFS、ext3cow 以及 wayback 中进行播放.

实验用服务器的配置如下:Intel Xeon 2GHz 处理器;512MB 内存;Adaptec aic7902 Ultra320 SCSI 适配卡;Linux 操作系统,内核版本 2.4.22;SEAGATE ST336607LW 硬盘,容量为 34GB,划分为一个分区,依序安装各个文件系统并测试.实验使用的 ext3cow 版本为 0.1.4,wayback 版本为 1.0.1.

© 中国科学院软件研究所 http://www.jos.org.cn

3.2 系统总体性能测试

实验中,设定测试文件集中的起始文件数为 10 000,分别针对不同的事务数运行 postmark.Postmark 中的事务是具有文件系统 cache 屏蔽作用的读(read)、追加(append)、创建(create)、删除(delete)操作的集合.

图 4 为实验结果,由图可见,版本机制的引入降低了文件系统的性能.随着事务数量从 5 000 增加到 50 000,各文件系统的测试完成时间也呈线性增长趋势.wayback 增长得最快,完成时间也最长;THVFS 和 ext3cow 次之,且性能相近;ext3 的完成时间最短;当事务数为 50 000 时,它们的完成时间分别为 409s,129s,121s 和 79s.我们分析性能变化的原因是:wayback 的用户态写日志机制导致了低性能;ext3cow 和 THVFS 中在修改文件时需要检查文件版本号,即使只修改文件的当前版本,也需要读取、比较并修改版本间的共享位图;THVFS 由于采用了细粒度版本生成策略,在最坏的情况下,修改版本时需要遍历其所有祖先,这也降低了性能.

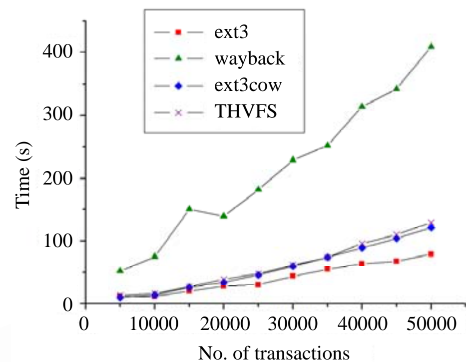


Fig.4 Postmark results in ext3, wayback, ext3cow, and THVFS

图 4 Postmark 在 ext3,wayback,ext3cow 和 THVFS 中的运行时间比较

通过如下方式得到该测试中 wayback,THVFS 和 ext3cow 相对于 ext3 的性能变化百分比:首先计算 3 种多版本文件系统在各事务点的测试完成时间变化百分比,然后再对 10 个点处的百分比求平均数.结果表明,对传统文件系统操作,Wayback,THVFS 和 ext3cow 的版本机制平均引入性能开销分别为 444%,44%和 33%.

3.3 历史版本检索性能测试

我们用修改过的 lmbench[11]测试多版本文件系统检索历史文件版本的性能.测试的流程是:第 1 步,创建测试文件集;第 2 步,以测试文件集中指定文件的指定版本为参数执行 stat()函数,完成对文件指定版本的检索并读取属性的操作,我们以该测试的执行时间作为依据,评价系统对文件历史版本的检索性能;第 3 步,删除测试文件集.测试文件集包括 100 个文件,每个文件各拥有当前版本和 100 个历史版本,后文中称为第 0 号版本、第 1 号历史版本、第 2 号历史版本,一直到第 100 号历史版本.

在 3 个多版本文件系统 wayback,THVFS 和 ext3cow 中进行 lmbench 测试的结果表明:wayback 的测试执行时间较 THVFS 和 ext3cow 高出近 4 个数量级,这是由于在 wayback 中作历史版本访问前必须先将版本从日志中恢复,而且日志回滚操作在用户空间执行,性能较低.

图 5 是 THVFS 与 ext3cow 的测试结果比较图,横坐标为文件版本的版本号,纵坐标为时间,单位为μs.由图可见,对第 1 号历史版本执行测试的时间,THVFS 为 ext3cow 的 1.5 倍;对第 25 号历史版本执行测试的时间,THVFS 为 ext3cow 的 75%;对第 50 号历史版本执行测试的时间,THVFS 为 ext3cow 的 55.5%;对第 100 号历史版本执行测试的时间,THVFS 仅为 ext3cow 的 34.8%.可见,随着版本号的增大,THVFS 的测试执行时间几乎不变,而 ext3cow 却表现出了线性增长的趋势.图中,无论是 THVFS 还是 ext3cow,曲线在第 0 号版本与第 1 号历史版本之间都出现了一个阶跃,其原因是:第 0 号版本为当前版本,可以被直接访问,不需要执行版本检索操作.由实验数据可知,当版本号低于 13 时,THVFS 执行测试的时间略高于 ext3cow,其原因是:ext3cow 使用链表结构组织版本元数据,越接近表头的版本,检索时间越短,反之亦然.

© 中国科学院软件研究所 http://www.jos.org.cn

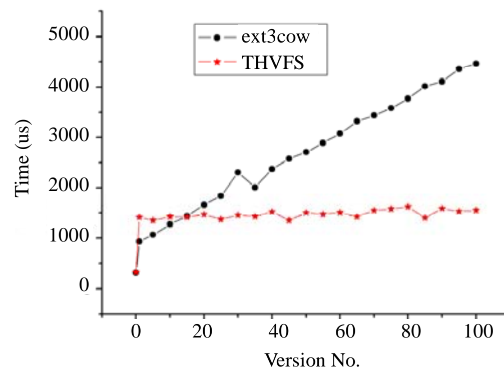


Fig.5 Comparison of version access time between ext3cow and THVFS

图 5 ext3cow 和 THVFS 版本访问时间的比较

通过如下方式得到 THVFS 相对于 ext3cow 检索历史数据的平均时间变化百分比:首先计算出 THVFS 相对于 ext3cow 对各历史版本(1~100)执行测试的时间变化百分比,然后再对这些百分比求平均数.结果表明,相对于 ext3cow,THVFS 检索历史数据的平均时间缩短了 34.4%.

3.4 寻径策略性能比较

本测试使用 lmbench,扩展其功能使其能够创建指定规模的版本空间层级结构.层级结构的规模取决于两个参数:(1) 层级结构深度 d:即层级结构的总层数,如图 2 中版本空间层级结构的总层数为 4,即 d=4;(2) 层级结构发散度 n:即层级结构中,父目录一个版本的生命周期包含其子目录或文件版本生命周期的个数,如图 2 所示,A 的一个版本的生命周期包含其子目录 B 的 3 个连续版本的生命周期,所以 n=3.

测试流程是:第 1 步,创建测试环境:首先创建(d–1)层目录,在最底层目录中创建 10 个测试文件,然后根据指定的参数 n 和 d,通过执行写入和快照操作,创建版本空间层级结构;第 2 步,以顶层目录为工作目录,指定一个版本号,对最底层的 10 个测试文件的该版本依序执行 open,stat 和 close 操作,记录测试的完成时间,最后再对不同版本号所对应的测试时间求平均,我们以该平均时间作为依据,评价系统寻径策略的性能;第 3 步,删除测试目录与文件.

测试分别在 THVFS 和 ext3cow 中进行,取值范围是 $d\in[2,9,]$,$n\in[1,10]$.为了仅对比寻径策略的性能,实验中关闭了 THVFS 中的版本快速索引.

图 6(a)为当 n=3 且固定不变时,逐级增加层级结构深度 d 的实验结果.由图可见,在 THVFS 的层级结构中执行寻径测试的平均时间比 ext3cow 中的要短.当层级结构只有 2 层时,THVFS 中的寻径测试平均时间比 ext3cow 中的短 4μs,前者为后者的 94.8%;当达到 6 层时,THVFS 中的寻径测试平均时间比 ext3cow 中的短 44μs,前者为后者的 96.9%;当达到 9 层时,THVFS 中的寻径测试平均时间比 ext3cow 中的短 424μs,前者为后者的 97.0%.

图 6(b)为当 d=4 且固定不变时,逐级增加层级结构发散度 n 的实验结果.由图可见,当 n=4 时,在 THVFS 中执行寻径测试的平均时间比 ext3cow 中的短 8μs,前者为后者的 93.8%;当 n=10 时,THVFS 中的寻径测试平均时间比 ext3cow 中的短 92μs,前者为后者的 95.0%.

通过如下方式计算 THVFS 的寻径策略相对于 ext3cow 的性能百分比:首先计算出各次实验($d\in[2,9,]$,$n\in[1,10]$)中 THVFS 的寻径测试平均时间相对于 ext3cow 的寻径测试平均时间的百分比,然后再对这些百分比求平均数.结果表明,通过采用反向继承寻径,THVFS 的平均寻径时间为 ext3cow 的 96.3%,缩短了 3.7%.性能提升有限的原因在于:文件系统中的 inode 与 dentry cache 起到了缓冲作用,缩短了两者的差距;同时,随着层级结构规模的扩大,inode 数的剧增,测试中 cache 不命中带来了较大的额外开销,也减弱了 THVFS 的优势.

© 中国科学院软件研究所 http://www.jos.org.cn

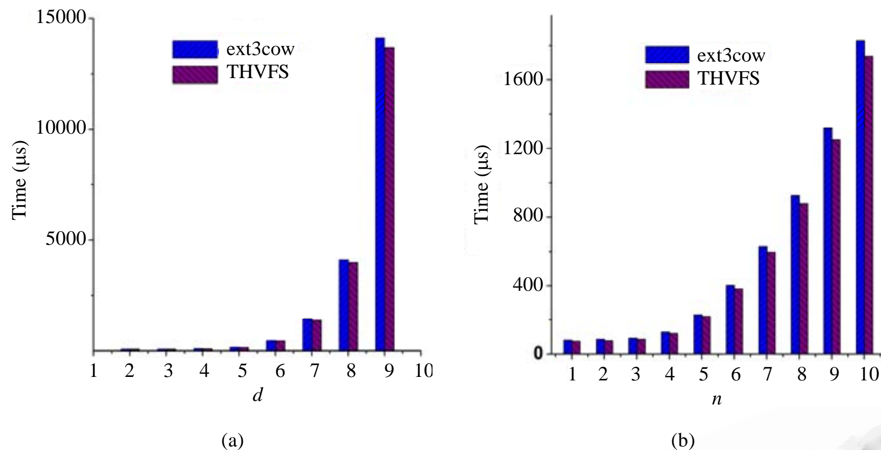


Fig.6 Comparison of pathfinding strategy between ext3cow and THVFS

图 6 ext3cow 和 THVFS 寻径策略性能比较

3.5 基于Berkeley trace的测试

该 trace 反映了科学计算用服务器的真实负载,播放时间为 24 小时.为了测试文件系统的性能,我们将 trace 播放时间分割为长为 24 分钟的 60 个等长小段(segment),分别记录各时间段内的读写 ART(平均响应时间).为了测试系统的空间消耗,我们比较了 trace 播放完成后各文件系统中已使用的 inode 数以及硬盘空间占用量.

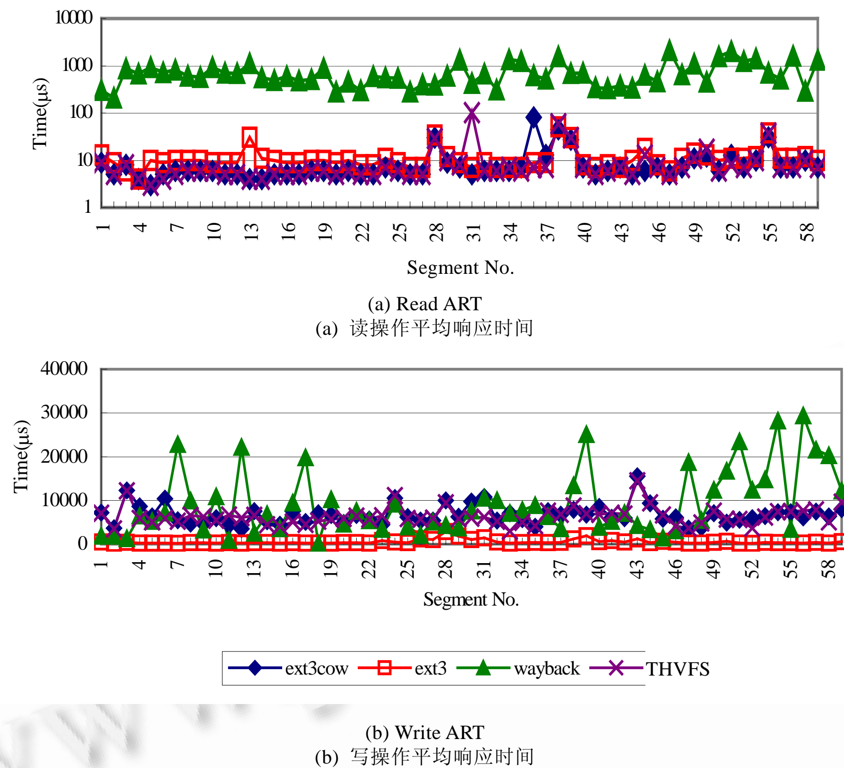
由于 THVFS 和 ext3cow 是基于快照的多版本文件系统,为了模拟真实使用环境中版本机制对性能的影响,在 THVFS 中播放 trace 时,每隔 72 分钟需要对 trace 播放器的当前工作目录作一次细粒度快照;而在 ext3cow 中播放时,则每隔 72 分钟作一次文件系统的全局快照.

图 7(a)为读操作平均响应时间,横坐标为时间段段号,纵轴为对数时间轴,单位为μs.由图可知,wayback 的读平均响应时间较其他 3 种文件系统高出近两个数量级;而 THVFS 和 ext3cow 中版本机制的引入对传统文件系统读操作性能影响较小.将图中各时间段的平均响应时间再取平均作为系统读性能的度量,可知在 trace 播放过程中,传统文件系统 ext3 的读 ART 为 11.76μs,wayback 的为 747.78μs,ext3cow 的为 9.58μs,THVFS 的为 10.37μs.wayback 的读性能较 ext3 慢了近 64 倍,而 ext3cow 提高了 18%,THVFS 提高了 12%.

图 7(b)为写操作平均响应时间,横坐标为时间段段号,纵坐标为时间,单位为μs.由图可知,wayback 的写平均响应时间起伏较大,这是与其日志提交策略相关的;THVFS 和 ext3cow 中版本机制的引入对传统文件系统写操作的性能也带来了一定的影响,这是由于在基于快照的多版本文件系统中,当新快照产生后,写入新数据时还必须执行必要的操作保留旧数据.将图中各时间段的平均响应时间再取平均作为系统写性能的度量,可知在整个 trace 播放过程中,传统文件系统 ext3 的写 ART 为 529.95μs,wayback 的为 9191.58μs,ext3cow 的为 6680.53μs,THVFS 的为 6392.25μs.wayback 的写性能较 ext3 慢了近 17 倍,而 ext3cow 慢了近 13 倍,THVFS 慢了近 12 倍.

Trace 播放完成后的空间消耗见表 2.为了比较不同版本生成策略对空间消耗量的影响,对 THVFS 和 ext3cow,实验统计了当快照间隔分别为 72 分钟、144 分钟和 288 分钟时,文件系统在 trace 播放完成后的空间占用量.由表可见,wayback 需要在日志中记录每次修改的数据及部分元数据,空间占用量比传统文件系统 ext3 高出两个数量级;ext3cow 和 THVFS 的空间占用量都随着快照频率的增加而增加,但 THVFS 中的细粒度快照技术能够仅保留指定重要目录的历史数据,因此其空间占用量小.由表中数据可知,播放完 trace 后,相对于 ext3,wayback 需要消耗额外 48 倍的存储空间,增加 93%的 inode 占用量;ext3cow 在 72 分钟作一次快照的频度下,需要消耗额外 146%的存储空间,增加 67%的 inode 占用量;THVFS 在 72 分钟作一次快照的频度下,仅需要消耗额外 80%的存储空间,增加 22%的 inode 占用量.

© 中国科学院软件研究所 http://www.jos.org.cn



(a) Read ART
(a) 读操作平均响应时间

(b) Write ART
(b) 写操作平均响应时间

Fig.7 Average response time (ART) of read and write
图 7 读写平均响应时间(ART)

Table 2 Comparison of space exhausted
表 2 空间消耗比较

File system	Space exhausted (Bytes)	Inodes exhausted
Ext3	432 750 592	8 827
Wayback	21 257 042 273 (+20 824 291 681)	16 995 (+8 168)
Ext3cow/288min	646 426 624 (+213 676 032)	10 877 (+2 050)
Ext3cow/144min	818 884 608 (+386 134 016)	12 475 (+3 648)
Ext3cow/72min	1 063 399 424 (+630 648 832)	14 771 (+5 944)
THVFS/288min	565 179 300 (+132 428 708)	9 510 (+683)
THVFS/144min	658 586 186 (+225 835 594)	10 033 (+1 206)
THVFS/72min	778 089 536 (+345 338 944)	10 808 (+1 981)

4 结 论

通过基于快照的细粒度版本技术,能够克服已有多版本文件系统不能对局部目录和文件保留版本的缺点,为系统中的不同目录和文件定制不同的版本生成策略,增加系统的灵活性;通过在名字空间与版本空间中独立建立索引并使用版本空间的反向继承寻径策略,可以充分利用版本间的相关性,既便于管理,也提高了系统性能;使用基于红黑树的快速索引结构,能够消除已有多版本文件系统检索版本的性能瓶颈.

结合以上关键技术实现的多版本文件系统 THVFS 既具有好的可用性,又具有较高的性能.评测结果表明:THVFS 的历史数据访问性能较著名多版本文件系统 ext3cow 提高了 34.4%,同时,维护版本所带来的时间空间开销都控制在可接受的范围内.

© 中国科学院软件研究所 http://www.jos.org.cn

References:

[1] Cornell B, Dinda PA, Bustamante FE. Wayback: A user-level versioning file system for linux. In: Proc. of the 2004 USENIX Annual Technical Conf. Boston: USENIX Association, 2004. 19−28.

[2] Santry DJ, Feeley MJ, Hutchinson NC, Veitch AC. Elephant: The file system that never forgets. In: Proc. of the Workshop on Hot Topics in Operating Systems. Arizona: IEEE TCOS, 1999. 2−7.

[3] Howard JH, Kazar ML, Menees SG, Nichols DA, Satyanarayanan M, Sidebotham RN, West MJ. Scale and performance in a distributed file system. ACM Trans. on Computer Systems, 1988,6(1):51−81.

[4] Pike R, Presotto D, Doward S, Flandrena B, Thompson K, Trickey H, Winterbottom P. Plan 9 from Bell Labs. Computing Systems, 1995,8(3):221−254.

[5] Hitz D, Lau D, Malcolm M. File system design for an NFS file server appliance. In: Proc. of the 1994 Winter USENIX Technical Conf. San Francisco: USENIX Association, 1994. 235−245.

[6] Lee EK, Thekkath CA. Petal: Distributed virtual disks. In: Proc. of the 7th Int’l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-7). Cambridge: ACM Association, 1996. 84−92.

[7] Quinlan S, Dorward S. Venti: A new approach to archival storage. In: Proc. of the 1st USENIX Conf. on File and Storage Technologies. Monterey: USENIX Association, 2002. 89−101.

[8] Zachary P, Randal B. Ext3cow: A time-shifting file system for regulatory compliance. ACM Trans. on Storage, 2005,1(2):190−212.

[9] Soules C, Goodson G, Strunk J, Ganger G. Metadata efficiency in versioning file systems. In: Proc. of the 2nd USENIX Conf. on File and Storage Technologies. San Francisco: USENIX Association, 2003. 43−58.

[10] Muniswamy-Reddy K, Wright C, Himmer A, Zadok E. A versatile and user-oriented versioning file system. In: Proc. of the 3rd USENIX Conf. on File Storage and Technologies. San Francisco: USENIX Association, 2004. 115−128.

[11] McVoy L, Staelin C. lmbench: Portable tools for performance analysis. In: Proc. of the USENIX 1996 Technical Conf. San Diego: USENIX Association, 1996. 279−294.

[12] Roselli D, Anderson ET. Characteristics of file system workloads. Technical Report, UCB/CSD-98-1029, Berkeley: University of California Berkeley, 1998.

向小佳(1977－),男,四川宜宾人,博士生,主要研究领域为海量存储,文件系统,计算机网络,分布式系统.

郑纬民(1946－),男,教授,博士生导师,CCF高级会员,主要研究领域为计算机体系结构,并行计算,编译器技术,网络计算,网络存储.

舒继武(1968－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为网络存储,并行计算,并行处理技术.

© 中国科学院软件研究所 http://www.jos.org.cn