一种迭代式快照系统的设计与实现

刘振军[1,2] 许 鲁[1] 冯 硕[1] 尹 洋[1,2]
[1](中国科学院计算技术研究所,北京 100080)
[2](中国科学院研究生院,北京 100039)
E-mail:TULiuzj@ict.ac.cnUT

摘 要 VSVM是一个提供了迭代式快照机制的逻辑卷管理系统。该快照机制针对于普通应用,允许快照逻辑卷正常读写,支持对任意快照逻辑卷(可能是经过多次快照迭代操作产生的逻辑卷)继续创建快照。快照的可迭代性确保了对快照数据的高效灵活的管理。VSVM在快照逻辑卷的数据映射上采用了直接映射的方式,使得快照迭代操作次数的增加并不影响其处理复杂度,并在设计方面理论上消除了对快照操作迭代次数的限制。论文实现了VSVM的迭代快照原型并给出了性能分析。试验结果表明系统具有较强的快照性能,单个系统可以支持上百个迭代式快照逻辑卷的并发密集读写,迭代快照适于普通应用使用。

关键词 迭代快照 卷管理 存储虚拟化

文章编号 1002-8331-(2006)14-0011-05 **文献标识码** A **中图分类号** TP302

The Design and Implementation of an Iterative Snapshot System

Liu Zhenjun[1,2] Xu Lu[1] Feng Shuo[1] Yin Yang[1,2]
[1](Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080)
[2](Graduate School of the Chinese Academy of Sciences, Beijing 100039)

Abstract: VSVM is a logical volume manager providing an ordinary-application-oriented iterative snapshot mechanism. In VSVM, any snapshot volume can be writable. No matter how many iterative operations have been produced in generating a snapshot volume, another snapshot based on it can still be created. This ensures that snapshot data can be managed efficiently and flexibly. We introduce a direct mapping method to snapshot volume data mapping, with which the complexity of snapshot logic won't be affected by the increase of iterative operations. This method also theoretically removes the limit of maximum number of snapshot iterative operations in design. We have implemented a VSVM prototype and analyzed its performance. Experimental results show that VSVM has nice snapshot performance; one single system can support concurrent intensive I/O on over one hundred iterative snapshot volumes. Therefore, the iterative snapshot volume is suitable for ordinary usage.

Keywords: iterative snapshot, volume management, storage virtualization

1 引言

随着SAN和NAS技术的出现,许多在应用上分离的数据开始在存储上趋于集中。这一变化使得一些数据类同的应用利用快照技术进行优化成为可能。在企业办公环境里,计算机使用相同操作系统、相同应用软件的情形非常普遍;同构集群中多个节点的数据环境也无显著区别。如果我们将这些集中存储后的相似数据等价看待为同一数据的不同快照副本,那么运用快照技术将能够优化和拓展对这类数据的管理,并有一些在传统存储模式下应用所不具备的有趣特性。例如,快照技术支持快速生成数据的即时副本,利用这一点可使同构集群系统具备快速提供节点数据环境的能力。

新应用将带来新问题。要用快照数据代替普通数据以获取更好特性,首先需要屏蔽掉快照数据与普通数据在应用属性与可管理性上的差异。这就意味着不但快照副本要能够被正常读写,更重要的是快照可以按需创建,并对其能完成备份、历史记录追踪等全面的数据管理。一个自然的需求是能对这些快照副本继续创建快照,即支持迭代式的快照逻辑。

然而,目前的快照研究和相关产品要么针对备份和历史存档,它们无法对快照再次创建快照;要么只有非常简单的迭代逻辑,无法支持大规模频繁迭代式应用。它们都不能很好地满足上述集中存储模式下快照数据应用与管理的需求。

针对上述情况,本研究提出了一种支持迭代快照能力的逻辑卷管理系统VSVM。该系统存在如下特点:

(1)迭代快照的能力。即可以对快照逻辑卷再次创建快照,并且理论上迭代次数没有限制。这是VSVM逻辑卷管理的关键特点。

(2)可正常读写的快照逻辑卷。满足实际应用使用。

(3)写拷贝(copy-on-write)型快照机制。该机制使得数据

基金项目:国家973基础研究发展规划资助项目(编号:2004CB318205);中国科学院百人计划基金资助项目

作者简介:刘振军(1976-),男,博士研究生,研究方向:网络存储,备份容灾。许鲁(1962-),男,博士,研究员,博士生导师,研究方向:操作系统、体系结构、网络存储。冯硕(1979-),男,研究实习员,研究方向:网络存储、集群管理。尹洋(1980-),男,博士研究生,研究方向:网络存储、集群管理。

副本可以快速创建,同时也节省了存储资源。

本文描述了 VSVM 的迭代式快照核心的设计、实现以及性能测试。本文其余部分的结构如下:第 2 部分介绍相关研究。第 3 部分对整个 VSVM 系统的体系结构和迭代式快照逻辑进行描述。第 4 部分详细描述 VSVM 迭代式快照的设计与实现。第 5 部分为 VSVM 原型的快照性能测试结果。第 6 部分给出了本文总结和未来研究方向。

2 相关研究

目前已有存储系统的快照大都针对备份等只读应用,很少用于读写目的,支持迭代式快照逻辑的系统则更少。

Linux 中的 LVM[1]与 IBM 的 EVMS[2]具有典型的以备份为主要目的的快照机制。系统可对其中的普通逻辑卷创建快照。旧版 LVM 中只有只读型快照,EVMS 以及现在的 LVM2 都能够创建可写快照。系统中快照以逻辑卷的形式存在。但是,这两个系统都不能对快照的逻辑卷创建快照。因此对使用中可写快照的备份等工作并不能很方便地进行。

SNAP[3]是对象存储系统 Thor 的高性能快照系统。相对于针对备份恢复的快照系统,其更关注于快照服务的性能,目的是确保应用该快照的 back-in-time 事务的效率。其快照机制针对应用进行设计,可以很频繁地创建快照而不影响其应用性能。但是,SNAP 仅支持只读类型的快照,无法提供给普通读写应用使用。SNAP 不能也不必支持迭代式的快照逻辑,因为只读快照上的数据状态不会发生改变。

EMC 公司的 SnapView[4]有两种快照类型:分离镜像类型的 Clone 和写拷贝类型的 Snapshot。SnapView 对每一个卷最多支持 8 个 Clone 和 8 个 Snapshot,对每一个 Clone 又可以创建 8 个 Snapshot。从这个意义上,SnapView 可以支持快照的简单"一次迭代"。Clone 允许进行读写,但创建前必须与逻辑卷进行镜像同步,因而无法按需随时创建。Snapshot 虽可立即获得,但却是只读类型的快照。SnapView 不能进行快照的多次迭代,迭代产生的快照也不能支持普通的读写。

造成迭代式快照"稀缺"的原因主要是存储系统通常不将快照作为承载生产数据的主体,因而对其性能、数据可恢复性等需求相对较低,而迭代快照又在逻辑维护上较普通快照更为复杂,因此支持这一功能得不偿失。但是,在快照作为实际应用存储的情况下,迭代式快照的需求与必要性就凸现出来。

3 VSVM 系统概述

VSVM 关注于迭代式快照功能的支持。VSVM 是基于 Linux 平台开发的逻辑卷管理系统。它将多个存储设备虚拟化为一个大的存储池,并利用其中资源构建逻辑卷供应用程序使用。与 LVM 一样,VSVM 的快照也以逻辑卷形式存在;所不同的是,系统可以对任何一个逻辑卷创建快照,无论该逻辑卷是普通逻辑卷,还是任意次快照迭代产生的逻辑卷。这就意味着在理论上 VSVM 快照的迭代次数没有限制。

在 VSVM 中快照都是写拷贝类型,快照创建的时间通常在 1s~2s 之内。换言之,快照可以在任何需要的时候即时获得。由于写拷贝快照与其源逻辑卷共享没有改变的数据,在逻辑卷彼此数据大都相同的情况下可以显著地节省存储空间。

尽管可以创建只读类型的快照逻辑卷,VSVM 能够将任一个快照创建为可写的类型。因此,快照逻辑卷的可应用性不受其快照迭代次数的影响。

上述这些特点为快照提供正常应用带来了极大便利。任何时候只要有应用需要某个(快照)逻辑卷的数据,都可以立即对这个逻辑卷创建可写快照并交付使用。

3.1 VSVM 体系架构

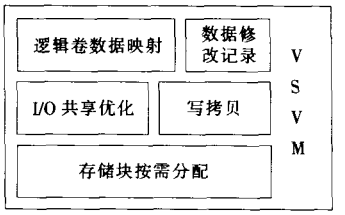

图 1 VSVM 的体系架构

图 1 描述了 VSVM 的体系架构。该架构由逻辑卷数据映射、数据修改记录、写拷贝机制、I/O 共享优化以及存储块按需分配等 5 个功能模块组成。其中:

(1)逻辑卷数据映射模块是 VSVM 的核心部分,其负责完成逻辑卷上数据到存储的映射。写拷贝类型的快照都是通过映射来维护快照关系以及判断是否进行写拷贝。VSVM 的迭代快照逻辑就在这个部分体现。

(2)数据修改记录模块负责记录逻辑卷数据的修改情况,主要用于支持逻辑卷的增量备份以及快照回滚。此模块还对迭代式快照的备份和恢复提供了特有的支持,保证迭代快照逻辑得以恢复到原状。

(3)当逻辑卷间需要进行写拷贝时,写拷贝模块负责写拷贝任务的执行。其不但是一个执行者,更是一个决策者。当系统支持多个写拷贝策略时,此模块根据快照状况选择最适合的策略来执行。

(4)I/O 共享优化模块也是 VSVM 的关键部分。由于 VSVM 大量应用快照逻辑卷,数据的存储共享十分普遍,因此针对这种共享的 I/O 路径的共享优化可以显著提高系统的聚集 I/O 性能。

(5)存储块按需分配模块处于 VSVM 的最底层,负责逻辑卷与存储资源的按需绑定。它一方面使存储资源得到了充分利用,另一方面也为多种写拷贝策略提供了基础支持。

本文着重描述 VSVM 的核心迭代式快照机制的设计与实现。不对架构中数据修改记录、I/O 共享优化、存储块按需分配这三部分作进一步的详细介绍。

3.2 VSVM 快照逻辑结构

VSVM 的迭代式的快照逻辑与写拷贝机制决定了逻辑卷的逻辑组织形态。按照逻辑卷间的快照关系,一个逻辑卷以及基于它的一系列迭代式快照构成了一种树状逻辑。树的根节点是初始创建的逻辑卷,其孩子节点是这个逻辑卷的快照。同样,树中任意节点都是其父亲节点逻辑卷的快照。由于每个快照都继承了它的源逻辑卷在快照创建时刻的数据以及存储共享状态,因此任一节点逻辑卷都可能共享其任何祖先节点上的数据。

图 2 是 VSVM 中逻辑卷组织形态的一个示例。如图,有迭代快照关系的逻辑卷 1~5 组成了一颗树,其中逻辑卷 2 是逻辑卷 1 的快照,逻辑卷 3、4 是逻辑卷 2 的快照,逻辑卷 5 又是逻辑卷 3 的快照。根据快照关系以及各个逻辑卷数据(块)修改情况决定了数据的存储共享状态。例如,逻辑卷 1 的数据块 1 被逻辑卷 2、3、5 共享,逻辑卷 2 的数据块 3 被逻辑卷 3、4、5 共享等等。

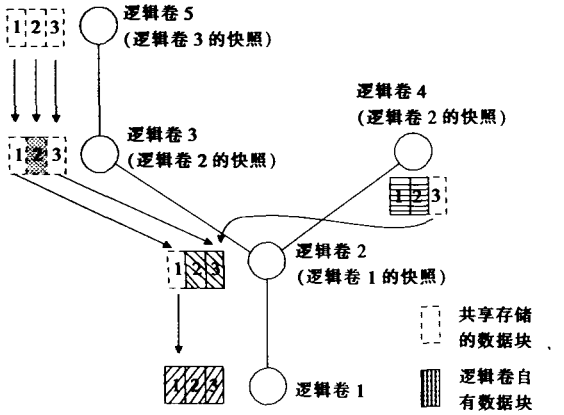

图2 VSVM的逻辑卷组织形态

从上面的简单示例就可以看出,VSVM迭代快照的复杂度要高于普通的快照系统。数据的共享和写拷贝由原来的逻辑卷一对一关联变为了一对多关联,由此引发数据快照正确性维护、多逻辑卷数据读写同步、以及并发读写时I/O性能保证等问题的复杂度剧增,其实现难度较之于普通快照系统也大了许多。接下来的部分就详细描述本迭代快照系统的设计考虑与具体实现。

4 迭代式快照的设计与实现

4.1 直接映射式实现架构

许多之前系统的快照映射采用了间接映射的实现方式:快照逻辑卷的读写首先在其上查找待访问数据所映射的存储,假如映射不存在,则表明写拷贝没有发生,那么再到其源逻辑卷上查找对应的映射并完成相应读写操作。这种方式的优点是节省了一些映射所需的内存开销,但它并不十分适用于迭代式快照的情况。

如前所述,一个逻辑卷及其迭代快照组成了树状的逻辑结构,逻辑卷可能共享其祖先逻辑卷的数据。如果按照这种间接映射方式来处理某个快照逻辑卷的数据请求,那么映射的查找过程就可能涉及到它的多个祖先逻辑卷。这一方面会带来查找上的延迟,另一方面在逻辑卷并发访问时的锁机制也会涉及到所有这些逻辑卷,因而导致并发IO效率下降。快照的迭代次数越多,间接映射带来的这些不利情况就越严重,这对强调快照迭代和应用性能的VSVM来说是不可接受的。因此,我们在访问逻辑上将迭代的树状快照映射方式设计为平面式的直接映射。即让每个逻辑卷都有其全部数据的映射,可直接对应到数据的存储地址。如图3所示:逻辑卷2是逻辑卷1的快照,逻辑卷3是逻辑卷2的快照。访问逻辑卷3的数据块1时,如使用间接映射,则需要依次查找逻辑卷3、2、1才能最终获得数据的实际映射;改用直接映射后,逻辑卷3虽然是逻辑卷2的快照,但当其映射的数据位于逻辑卷1时,将不经过逻辑卷2而直接映射至逻辑卷1所对应的数据。

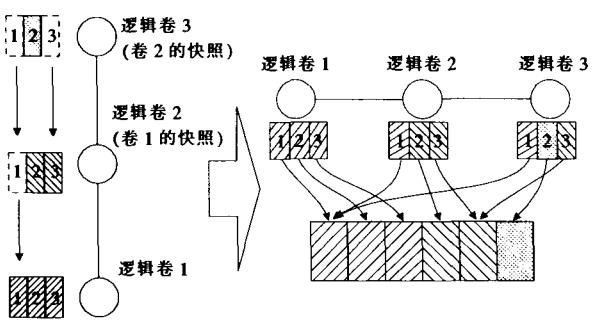

图3 间接映射与直接映射

通过这样的设计,一个数据请求以及相应写拷贝操作直接关联到对应的逻辑卷,能够有效缓解上述因间接映射造成的并发效率低下的问题。并且如图3(b)所示,各个逻辑卷在数据映射上是平等的,映射处理过程不会因为迭代快照的增加而变得更加复杂,因而本系统的快照迭代次数在理论上可以没有限制。

4.2 写拷贝策略

写拷贝操作执行时,有两种情况:(1)逻辑卷需要对属于自己的数据块进行写入,但此数据块正被其它逻辑卷共享。此时的操作是要消除其它逻辑卷的共享,然后完成此数据块的写入。(2)逻辑卷需要对共享另一逻辑卷的数据块进行写入。此时的操作是获得属于自己的数据块副本,同时消除对另一逻辑卷的共享,然后对自己的数据块进行写入。在情况(1)中,首先需要知道此数据块被逻辑卷共享的情况,以便确定写拷贝的目的和拷贝策略;而情况(2)则不必知道此数据块的共享状况。

直接映射下逻辑卷数据块的共享情况十分明确,但可采用的写拷贝策略却有许多种。例如在情况(1)中,逻辑卷可对所有共享此数据块的逻辑卷进行写拷贝,以此消除它们的共享;也可以仅对其中一个逻辑卷进行写拷贝,并使其余逻辑卷转而共享此新拷贝的数据块;在逻辑卷的存储构成足够灵活的情况下(例如存储块可按需分配和映射),逻辑卷甚至可将自己数据块的所有权移交给其它逻辑卷,然后转换为情况(2)即共享其它逻辑卷数据的情况进行处理。

我们的原型系统采取了数据块固定属主,且当逻辑卷对自有数据块写入时,对所有共享此数据块的逻辑卷进行写拷贝的策略。本原型目前仅支持了这一种策略。此策略简化了原型系统的实现复杂度,但它并不是对性能最有利的类型。例如一些系统如VxFS[5]所支持的单次写拷贝操作类型的策略,可使得逻辑卷I/O性能独立于共享该数据块的逻辑卷的个数。类似的策略可在今后的写拷贝模块中进行扩充。

4.3 快照算法

一些概念:

物理存储单元Chunk:组成逻辑卷的基本单位,由一组连续的物理设备块(Sector)构成。其也是一次写拷贝的单元。

逻辑卷:是由多个大小相同的Chunk组成的有序集合$\{C_1, C_2 \cdots C_n\}$,其中C表示Chunk,n为逻辑卷中Chunk的数目。

属于关系:对于一个具体的Chunk,有且仅有一个逻辑卷使得此Chunk属于它。Chunk属于一个逻辑卷的含义是指,该Chunk永远在这个逻辑卷的有序集合中。当这个Chunk发生写拷贝操作时,它总是被保留在其属于的逻辑卷中。而另一逻辑卷则总是将其替换为拷贝出来的Chunk副本。

在本系统中,写拷贝总是发生在一个Chunk属于的逻辑卷和共享此Chunk的逻辑卷之间。

系统中有两种类型的逻辑卷:

源逻辑卷:创建时给定Chunk集合的逻辑卷,其集合中的任意元素C都有C属于逻辑卷。

快照逻辑卷:创建时Chunk序列是另一逻辑卷的Chunk序列的拷贝。其集合初始状态时任意元素C都有C不属于逻辑卷,即此逻辑卷的所有Chunk都是共享的。此时称此快照逻

辑卷是其拷贝源的快照。

算法流程：

(1)快照逻辑卷创建：

快照创建的过程很简单：主要的工作是复制源逻辑卷的Chunk集合序列，并在系统中标示此快照逻辑卷已共享这些Chunk。

(2)逻辑卷读写：

逻辑卷读写请求到来时，分为两种情况：

①若请求对应的Chunk属于逻辑卷，如果是读操作，则直接完成读请求。否则是写操作，若Chunk不被共享，直接完成写请求；否则发起一个COW_TASK任务，查询每一个共享此Chunk的逻辑卷是否已经发起了此Chunk的写拷贝，若没有则发起之，假如写拷贝已经发起，则也将此写拷贝纳入本COW_TASK中，同时将此写请求挂入等待队列中等待此次COW_TASK任务完成。

②若请求对应的Chunk不属于逻辑卷，查询拥有这个Chunk的逻辑卷是否以及对本逻辑卷执行了该Chunk的写拷贝操作，若是，则读写请求都将挂入等待队列等待这个写拷贝完毕；否则没有开始写拷贝操作，若是读请求则直接完成；若是写请求，逻辑卷发起此Chunk的写拷贝，并将写请求挂入等待队列等待此写拷贝完毕。

写拷贝完毕后，首先将写拷贝新产生的Chunk副本替换到共享此Chunk的逻辑卷中，并记此副本Chunk属于这个逻辑卷。然后将等待此写拷贝完毕的读写请求定位到这个副本Chunk，并完成读写请求。如果这个写拷贝还被纳入到某个COW_TASK中，通知COW_TASK这个写拷贝已经完成。

当COW_TASK中最后一个写拷贝已经完毕，则将等待此COW_TASK完毕的写请求完成。

5 系统原型测试

下面是我们对VSVM迭代式快照的I/O性能相关的测试结果。试验硬件环境采用一个服务器节点，其配置为Intel Xeon™ 2.40GHz处理器、1GB内存、IDE7200转120G硬盘。软件环境是Linux Redhat 8.0，内核版本为2.4.20。

5.1 写拷贝单元大小

由于写拷贝操作对性能的影响是基于此类技术的快照性能的关键因素，我们对VSVM试验的第一步是确定写拷贝单元的大小。在我们的试验中发现写拷贝单元太小或太大对VSVM的快照I/O效率都有影响，太小的写拷贝单元由于发起的写拷贝次数较多，会造成I/O性能下降，并且由于映射表项数目较大，相应内存开销也会增大。但写拷贝单元太大，又会存在小数据量写入(通常一次写是一个4KB的文件块)导致大量数据拷贝的问题，造成性能下降以及空间浪费。为此，我们通过对不同的写拷贝单元大小进行测试，确定了对快照I/O性能最有利的单元大小。

在VSVM中写拷贝单元是Chunk。表1为我们对逻辑卷刚创建单个快照的初始情况下分别对不同Chunk大小的快照源和快照进行Bonnie++块读写测试的情况。Chunk大小的变化从16KB至256KB，由于测试结果中Chunk大小的变化对块读和Seek的性能几乎没有影响，因此表中不再给出相应的具体数据。从表中我们可以看到Chunk大小在32KB和64KB情况下块顺序写性能达到最好。更大或更小的Chunk性能都有所下降。由于在几十个至上百个层叠快照同时读写的规模测试中64KB的Chunk在整体性能上较32KB好，因此我们最终选择了64KB作为我们性能测试的写拷贝单元大小。

表1 单快照逻辑卷Bonnie++测试(2 016M文件尺寸)

Chunk Size测试		快照源写		快照写	
Bonnie++块测试		Sequential Write		Sequential Write	
		per block	rewrite*	per block	rewrite
		KB/s	KB/s	KB/s	KB/s
Chunk/KB	16	6 758	17 775	6 449	18 089
	32	7 287	20 012	9 489	19 070
	64	13 311	10 354	8 355	19 114
	128	8 083	9 489	7 047	19 225
	256	5 452	16 350	6 937	12 143

5.2 快照性能

本部分对VSVM和LVM进行快照性能的对比测试。测试使用了文件系统的读写性能的测试软件Bonnie。读写测试文件大小为2 047M(Bonnie支持的最大值)，是内存大小的两倍，可以避免系统缓存对测试的影响。测试时首先创建一个6GB大小的逻辑卷，在其上创建一个ext3文件系统。然后对逻辑卷创建快照后对此逻辑卷进行Bonnie测试。

图4给出了VSVM和LVM的逻辑卷在不同快照数目下块写入性能(Write)和读写性能(ReWrite，先读出，再写入)的比较。上述性能结果两者都是在源逻辑卷对所有共享该数据的快照逻辑卷执行写拷贝的情况下获得的。可以看出，VSVM的迭代式快照性能相对于LVM要好。由于VSVM的快照机制中快照与快照源处于平等的位置，因此本测试结果也可间接反映出快照逻辑卷的写性能指标。

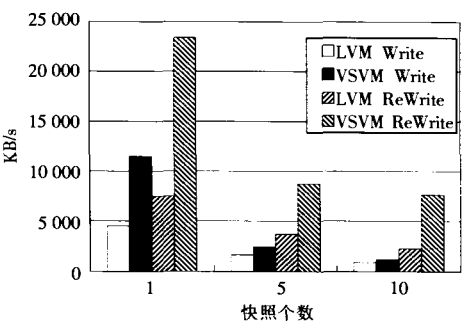

图4 带快照的逻辑卷数据块写性能

5.3 迭代快照并发I/O性能

Bonnie++和Bonnie性能测试基于单个文件读写并在测试过程中取系统时间来计算性能，在系统负载很大的情况下，测试结果不太稳定。为了测试迭代快照并发应用的性能，我们采用了大量快照同时读写并计时(使用time命令)的方式进行规模测试，更符合系统的实际应用情况。

测试的主体步骤是：

(1)建立一个新普通逻辑卷并写入一个Linux内核源代码目录(190M，17 390个文件)；

(2)以此逻辑卷为根创建一颗N个节点的迭代快照树(快照树由程序随机生成)；

(3)同时对所有的快照及源逻辑卷拷贝入两个大文件(共210M)，并在拷贝完成后与拷贝源做diff操作(并发I/O过程)。

图5记录了在N分别等于5、10、20、40、60、80、100的情况下系统运行测试的总耗时。耗时情况可以反向反映出快照逻

辑卷的应用性能。

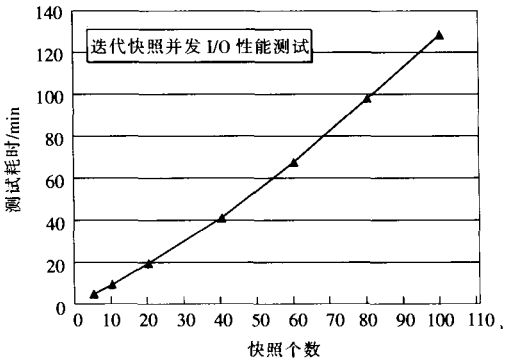

图 5 迭代快照并发 I/O 测试

可以看到,VSVM 的快照逻辑卷并发数与其应用性能呈较为合理的线性反比关系,系统可以同时支持 100 个以上迭代快照的并发 I/O。

6 结论

本文描述了虚拟共享卷管理系统 VSVM 的迭代式快照核心。设计这一新型快照机制的目的是提供满足正常应用的快照存储,这是许多快照系统所不擅长的领域。VSVM 中任何快照逻辑卷都可以是可写的类型,系统可对快照逻辑卷继续创建快照,这就对快照数据的备份等管理提供了必要的支持。我们采用了一种逻辑卷数据直接映射的设计,将树状的迭代快照逻辑转换为平面化的实现模式。这不仅在理论上突破了快照的迭代次数的限制,并且简化了迭代快照的实现复杂度,利于快照应用性能的改善。试验表明 VSVM 的快照性能较好,系统可支持上百个迭代快照的并发密集读写,快照逻辑卷适合正常应用使用。目前 VSVM 已作为蓝鲸 SonD 动态服务部署系统[6]的存储子系统,对 SonD 灵活部署应用数据环境起到了关键作用,并取得了较好的应用效果。

信息时代中数据快照的应用范围日趋广泛,不同类型的数据应用对快照的获取、功能与性能上的要求各异。例如,备份需要快照提供数据的只读一致映像;容灾则在此基础上强调快照的实时性,即利于远程复制的多版本、频繁、对生产系统影响较小的快照;而使用快照的正常应用则更关注于快照本身的 I/O 性能和可管理性。即使应用模式相同,不同具体应用乃至某一应用的不同时刻,读写特性也不会完全类似,甚至可能存在较大的差异。因此,使用固定的快照机制(如快照组织,写拷贝模式等)极难完全满足应用的实际需求。发展根据具体应用情况动态调整写拷贝策略、甚至快照类型的快照自主管理技术是我们未来的研究方向。(收稿日期:2006 年 2 月)

参考文献

1.D Teigland,H Mauelshagen.Volume Managers in Linux[C].In:the USENIX Annual Technical Conference,Boston,MA,2001:185~198
2.Steven L Pratt.EVMS:A Common Framework for Volume Management[C].In:Ottawa Linux Symposium,Ottawa,Ontario Canada,2002:451~458
3.Shrira L,Xu H.Data Engineering,SNAP:Efficient Snapshots for Back-in-Time Execution[C].In:ICDE 2005,Proceedings,21st International Conference,2005-04:434~445
4.Richard Hou,Steve Feibus,Patty Young.Data Replication and Recovery with Dell/EMC SnapView 2.0 and MirrorView.Dell,2003-02
5.Steve D Pate.UNIX Filesystems Evolution,Design,and Implementation (VERITAS Series)[M].Wiley Publishing,2003
6.刘振军,许鲁,尹洋.蓝鲸 SonD 服务动态部署系统[J].计算机学报,2005;7:1110~1117

(上接 6 页)

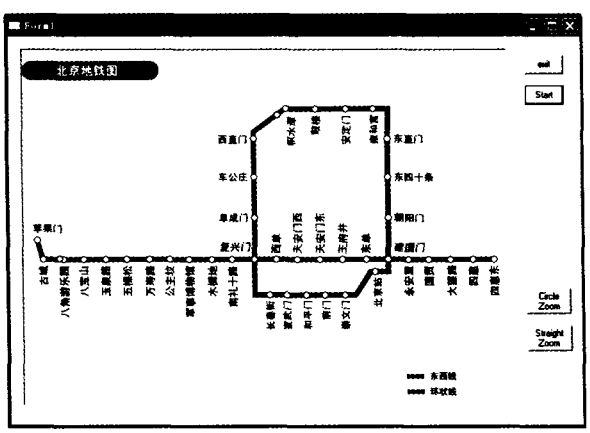

图 3 地铁运行进行动态模拟

4 结束语

(1)分析了城市轨道交通中的两种信号系统:固定闭塞系统和移动闭塞系统的特点,在研究移动闭塞条件下地铁列车的运行规律的基础上,建立了地铁列车的动力学模型;

(2)采用基于事件的控制理论和编队技术,通过引入运动参考变量,可以根据列车的走行距离对多列列车的运行进行实时调整,从而优化列车的运行、增加发车密度;

(3)地铁列车运行系统是一个很复杂的控制系统,经常会遇到一些不确定因素的干扰,如何在运行受到干扰情况下,提高列车的准点运行还要进行进一步研究。

(收稿日期:2006 年 2 月)

参考文献

1.Burrage K W et al.Railway Control Systems[M].London:A&C Black Publishers Limited,IRSE,1991
2.Nock O S et al.Railway Signalling[M].London:A&C Black Publishers Limited,IRSE,1980
3.于建国,苗彦英.地铁电动车组运行模型的研究[J].大连铁道学院学报,2001;22(2):43~45
4.毛明平,陶生桂,王曰凡.上海地铁 2 号线牵引仿真计算研究[J].城市轨道交通研究,2001;14(2):22~27
5.钮泽全.牵引计算学[M].北京:中国铁道出版社,1984:30~38,52~55
6.Ning Xi.Event-based motion planning and control for robotic system[D].Doctor of Science Dissertation.Washington University,St Louis,MO,1993
7.路飞,宋沐民,李晓磊.基于移动闭塞原理的地铁列车追踪运行控制研究[J].系统仿真学报,2005;17(8):1944~1948
8.W Kang,N Xi.Formation Control of Multiple Autonomous Vehicles[C].In:Proc IEEE International Conference on Control Applications,1999:1027~1032
9.W Kang,N Xi,Andy Sparks.Theory and Application of Formation Control in a Perceptive Referenced Frame[C].In:Proc of 39th IEEE Conference on Decision and Control,2000:352~357