一种按需分配增量卷的自动扩容方法研究

高岩 曹瑞 甄彩军 徐广治 王刚 刘晓光
南开大学计算机系，天津, 300071
monciel@163.com, caorui12001@yahoo.com.cn, zhenqiansha@126.com, colyviva@gmail.com, wgzwp@163.com, liuxguang@gmail.com

An Automatic Extension Method for Allocating-On-Demand Incremental Volume

Gao Yan, Cao Rui, Zhen Caijun, Xu Guangzhi, Wang Gang and Liu Xiaoguang
(Department of CS, Nankai University, Tianjin, 300071)

Abstract A non-cow snapshot system is proposed, which based on Allocating-On-Demand Incremental volume, as well as multiple snapshots may exist, the physical space occupation of a volume is not equal to its logical size. To implement the separation of physical space and logical space, two-layer volume is presented. Based on that, the automatic extension mechanism for physical and logical volumes is also developed. For each volume, a daemon process monitors whether the space usage exceeds the threshold, if so, the extension process will be called. Compared with manual extension, the space overflow can be avoided effectively. Besides, the extension for the logical space is allowed on-demand. We put forward a way of index array, which can manage the dispersive metadata chunk conveniently and also organize the original volume and increased part effectively. It is easier to implement than the tradition method of LVM and has a small system overhead.

Keywords ADI；Allocating-On-Demand；Automatic Extension；Non-COW

摘要 本文介绍了一种基于按需分配增量卷的非写前拷贝快照系统，该系统中存在多个快照版本，实际占用的物理空间可以与逻辑大小不相等。为了实现按需分配增量卷的物理空间与逻辑空间的分离，我们设计了两层卷的结构。在此基础上，设计并实现了对其物理空间及逻辑空间自动扩容的解决方案。对每个卷来说，都有一个守护进程对磁盘空间使用率进行监控，当该使用率达到阈值时将自动启动扩容进程。较之人工扩容，这种自动物理空间扩容技术有效地避免了溢出。另外，用户可以按需要对逻辑空间进行扩容。我们提出了一种索引数组的方式，可以方便地管理分散的元数据区，并且将原有卷与新增部分有效地组织在一起。与传统 LVM 的实现方法相比，该方法实现简单且系统开销小。

关键词 按需分配增量卷；按需分配；自动扩容；非写前拷贝快照

随着信息技术的发展，企业或个人对信息存储容量的需求在日益增长，磁盘的容量越来越大，在满足人们对存储空间的需求的同时如何合理利用这些空间也成了一个热点问题。按需分配的方法无疑可以提高磁盘空间的利用率，这种方法允许系统根据需求动态地分配空间，闲置空间减少，磁盘空间得到了合理地利用从而避免了空间的浪费。

Parallax[1][2]是一个按需分配的分布式存储系统，为虚拟化服务提供了良好支撑，适用于处理大规模存储需求。它在底层以一个分布式存储池（block-store）的形式，为每个虚拟机提供一个或多个虚拟磁盘映像（virtual disk image）。另外，Parallax 大量借鉴了虚拟内存管理技术，采用细粒度的地址空间，将虚拟磁盘映像使用 Radix 树[3]的形式管理，每个 Radix 树的根结点对应一个虚拟磁盘映像。这样的结构使得磁盘的数据组织更为灵活，为磁盘的扩容提供了方便。该系统采用元数据写前拷贝与数据的追加写技术来实现快照功能，不仅降低了快照的系统开销，

基金项目：论文工作得到国家自然科学基金（60903028），国家“863”计划（2008AA01Z401），教育部博士点基金（20070055054）和天津市科技发展计划（08JCYBJC13000）资助。

也避免了因磁盘空间不足而导致快照失效的情况，为扩容的时间提供了较为宽松的条件。

Esnap[4]、Esnap2[5]实现了快照卷的自动扩容技术并对 Linux LVM 中传统的方法进行了改进。在 LVM 中，随着源卷更新数据的增多，快照卷可用的空间会越来越少而快照卷初始分配的空间往往小于源卷，当写前拷贝数据量超出快照卷的可用空间时，会导致数据溢出，快照将会失效。Esnap 提出的快照卷的自动扩容技术有效地避免了这个问题的出现。与传统的 LVM 逻辑卷不同的是，Esnap 没有将原来的 pv 数组销毁，然后再重建一个更大的数组，而是将新增的卷作为一个节点加到原来的卷上这样新增的卷与原有卷形成一个数组为节点的链表，而每次扩容即是在链表后面再添加一项，从而有效地节省了扩容时间并降低内存消耗。Esnap 还提出了扩容次数增加导致链表过长继而影响查询速度的解决方案。然而，Esnap、Esnap2 中算法的实现较为复杂。

本文介绍了一种自动扩容方法，它针对的是一种按需分配的非写前拷贝快照系统。这种系统中的数据卷是一种按需分配增量卷 ADI（Allocating-On-Demand Incremental Volume），数据卷实际使用空间是动态增加的。系统中存在多个快照版本，所以数据实际的物理空间与其逻辑大小多数情况是不相等的。为实现数据卷逻辑空间和物理空间的分离我们设计了两层卷的结构。在此基础上，我们设计并实现了对数据卷逻辑空间和物理空间的自动扩容解决方案。

1 非写前拷贝快照系统简介

1.1 非写前拷贝快照系统设计思想

非写前拷贝快照系统是基于 LVM2[6]，采用虚拟试图技术[7]而设计的。与传统写前拷贝系统不同的是，非写前拷贝系统没有利用 COW 技术（Copy On Write，写前拷贝）保存源数据。为实现非写前拷贝，对写请求的处理是采用了追加写的方式。当有数据更新时，为所修改的数据直接分配一块新的空间，将新数据直接写入这块空间。然后修改元数据，正确引用这块新数据。此时修改的实际是源卷的元数据，在创建快照时已将元数据复制了一份，并冻结，即为快照。显然，这种机制会带来元数据更新的代价，为此，我们采取了组合、推迟的策略来缓解额外的元数据更新、访问开销。所以，在非写前拷贝系统中，对数据块的写操作，最好情况只有一次 I/O 操作。而 LVM2 缺省的基于写前拷贝的快照系统，每次写操作会导致三次 I/O 操作。显然，该系统极大地提高了系统的性能。而且该系统中没有源卷和快照卷的区分，所有的数据卷是统一进行管理的。在数据卷中区分了元数据及数据区域，元数据区用于保存 ADI 系统中的所有卷的元数据，用户的数据保存在数据区。只需在建立快照时拷贝元数据，可以看出，这种方法实际是将传统快照的数据拷贝转化为元数据拷贝，大大减少了磁盘复制的工作量。

为避免空间的浪费，非写前拷贝快照系统采用了按需分配的机制。初始分配的实际的物理空间可以小于用户需求的空间大小，追加写方式也使这种动态分配成为可能。随着数据的增多，可用的物理空间将渐渐减少，为此，我们设计并实现了数据卷物理空间的自动扩容机制。当已用空间占全部可用空间的比例达到某个阈值时，系统将自动为其增加一定的物理空间。与之对照，人工扩容不但增加了用户的负担，而且可能会因为扩容不及时而导致空间溢出，所以我们选择将物理扩容的过程交由系统自动完成，这一过程，对用户来说将是透明的。逻辑空间的扩容则由用户任意指定。

1.2 非写前拷贝快照系统体系结构

如图 1 所示，是非写前拷贝快照系统的数据卷结构，总体可分为元数据区和数据区。

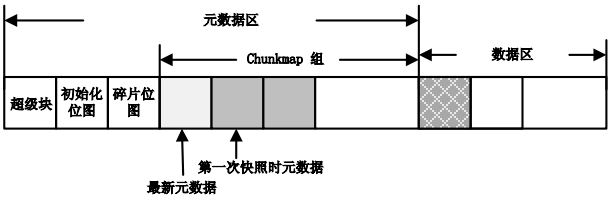

图 1 非写前拷贝快照系统的数据卷结构

元数据区包括超级块、初始化位图、碎片位图和 Chunkmap 组。超级块负责保存设备号、快照次数和每个快照的起始地址等卷的全局信息。Chunkmap 组保存用户地址到物理存储空间的映射表。初始化位图用于加速初始化进程。在创建卷的时候，我们需要将 Chunkmap 全部设置为非法物理地址，表示所有用户地址均尚未写入。如果卷逻辑大小非常庞大，则映射表的规模也会很大，那么初始化过程将十分耗时。为解决该问题，我们使用了初始化位图。位图中的每一位对应第一个 Chunkmap（最初的源卷）中的一片连续的区域。初始化时不对磁盘 Chunkmap 组进行填写，而是用位图中对应位为 0 表示 Chunkmap 相应区域尚未初始化。这样，当访问这部分元数据时，就不必读取磁盘，只需直接认为其为非法地址（对应用户地址的数据尚未写入）。当这部分元数据被更新到磁盘时，将初始化位图中对应位置为 1。显然，这种推迟初始化的方法有效地提高了初始化速度。由于我们采用追加写而非覆盖写方式，则一个用户地址数据在系统中可能存在多个版本，从而造成垃圾内存和碎片问题。

碎片位图就是用来进行碎片整理，回收垃圾碎片的。位图中的每一位对应数据区部分的一个数据块，标识该数据块是否是垃圾内存。数据区部分用于存储用户的数据。

在创建卷时，用户指定所需的逻辑大小，元数据区按此分配所需空间，进行初始化，而数据区则可分配小于逻辑大小的物理空间。

对于一个 ADI 来说，元数据区部分的大小可以根据逻辑大小计算出来。假定块大小为 4K，地址为 64 位，那么碎片位图、Chunkmap 组及数据区的比例是 1：64：32K（超级块及初始化位图可忽略，所需的空间相对其他部分很小）。元数据区与总（逻辑）空间的大小比例 P 就为

$$P = \frac{S_F + S_C}{S_F + S_C + S_L} \qquad (1)$$

式（1）中 S_F、S_C、S_L 分别表示碎片位图、Chunkmap 组及逻辑空间大小。计算得，P 的值约为 1.97‰，是一常量。这样，用户逻辑大小指定后，元数据区的大小就是固定的。在进行物理空间扩容时，也不会改变。但在逻辑空间扩容时，显然还需扩展元数据区，并应在读写逻辑中考虑逻辑空间扩容带来的元数据分散的问题。在创建 ADI 时，初始分配的数据区大小可以小于逻辑空间大小，我们将比例取值为 1/10。而物理空间的扩容比例也取为逻辑空间大小的 1/10。

2 按需分配增量卷自动扩容方法

2.1 两层卷结构

在非写前拷贝快照系统中，数据空间的按需分配对用户来说是透明的，物理空间占用与逻辑空间大小是不等的。为实现逻辑大小和物理大小相分离，我们提出了一种两层卷的结构，即逻辑卷和物理卷。对用户可见的是逻辑卷，它只是 Linux 内核中的一个数据结构，并不占用实际磁盘空间，负责将用户地址转换为物理卷地址。而物理卷则是实际保存元数据和数据的底层卷。这种结构如图 2 所示，它显然方便了逻辑空间和物理空间的分别管理。

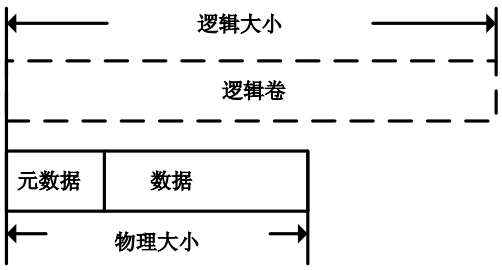

图 2 两层卷结构

逻辑卷是一个虚卷，其大小由用户创建卷时指定，称之为逻辑大小，也就是用户希望创建的卷所能使用的最大空间，该值是固定的。物理卷是底层实际卷，它的大小即为物理大小。在数据卷创建的初始化阶段，我们可以分配规模小于逻辑大小的物理空间。在用户使用的过程中，随着用户的数据量的增多，物理空间的占用率逐渐提高。当物理空间使用过多，超过一个阈值时，应继续为其分配空间，满足用户更多的存储要求。故物理大小是可变的，是元数据区与已分配的数据区的大小之和。

2.2 自动扩容

两层卷的机制使得按需分配增量卷的逻辑大小和物理大小得到了分离，可以根据需求对其中一个卷进行扩容。

（1）物理卷扩容

对于用户来说，物理卷的大小是透明的。用户完全不必关心实际有多少的物理空间，只需在逻辑的层面了解卷的大小。那么合理的物理卷的分配增量过程应该是系统在后台自动完成的，也就是实现物理卷的自动扩容。

对于每个物理卷来说，系统中会有一个守护进程负责对其空间使用情况进行监视。系统可以实时地判断当前空间使用率是否达到阈值以及是否需要启动扩容进程。

物理空间自动扩容的核心问题是用户态守护进程如何获取空间使用率。与传统块设备不同，按需分配增量卷由于采用了追加写方式，可以自然地掌握存储空间分配（使用）情况，即，它具备了文件系统的某些功能。因此，计算/获取空间使用率是十分容易的。在系统完成一次写操作后，当前数据区尾指针的位置即为空闲数据块首的位置。根据该位置与数据区空间的比值，就可以计算出空间使用率。显然，这两个值都需在内核的请求处理模块中获得。所以，如何达到用户态程序与内核态程序高效交互是一个主要的问题。

LVM 中提供了获得卷信息的用户态接口，如 lvs/lvdisplay。我们可以考虑借助于这两个函数来获得物理卷的空间使用率。首先，修改内核请求处理模块，将空间使用率作为 LVM 卷的属性信息写入磁盘元数据区域。同时修改用户态 LVM 代码中的 lvs/lvdisplay 函数，使之能读取这个属性。这样守护进程就可以通过调用 lvs/lvdisplay 来获得空间占用率了。然而，这种做法存在以下不足。首先，LVM 卷的磁盘元数据结构的修改相当复杂，在内核态实现非常困难。其次，以读写磁盘元数据方式交换数据，系统代价较高。如果交换频率过高，可能影响系统正常请求，使得处理

性能下降。如果交换频率过低，则不能及时掌握当前空间使用率，造成扩容不及时，从而导致溢出。

/proc 文件系统是一个虚拟文件系统，专门用于内核和用户态程序的信息交互。/proc 下的每个文件绑定到一个内核函数上，当文件被读时即时产生文件内容。其中文件的读写并不是真正地读写磁盘，而只是内存中的读写操作，所以几乎没有任何额外开销，可以很好地解决我们的问题。

我们的做法是在/proc 目录下建立一个 proc 接口，负责记录即时空间使用率，并且在每次写操作后更新空间使用率。为方便管理，在/proc 目录下创建一个子目录 snapproc，然后在该子目录下创建文件 full，记录空间使用率_full。

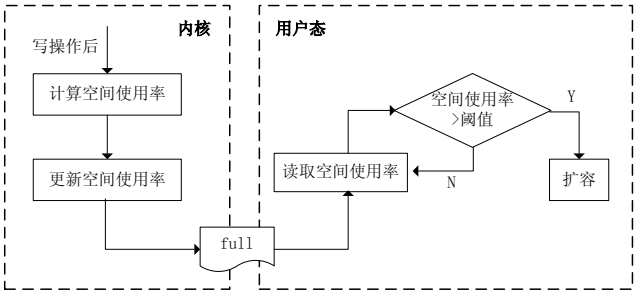

图 3 内核态程序与用户态程序信息交互示意图

如图 3，显示了内核态程序与用户态程序对空间使用率的交互过程。内核部分负责计算及更新空间使用率，用户态程序实时读取 full 文件中的数值，并以此判断是否进行扩容。

为实现 proc 接口，首先要在源码文件中包含 linux/proc_fs.h 这个头文件来定义正确的函数。Proc 接口的读写方法等都封装在 proc_dir_entry 结构体中。我们定义了两个 proc_dir_entry 结构体，分别是 proc_snap 和 proc_entry。前者用于生成 snapproc 子目录，后者用于创建实际的文件并实现具体的读写方法。

proc_snap 和 proc_entry 的初始化过程是在 proc_full_init 函数中进行的。根据需求分析，空间使用率应该是由内核态负责写，以便用户态读，所以只声明并实现 proc_entry 的读方法即可，而把写方法置为空。相关函数的伪代码如下所示。

```
static char _full[10]; /*记录空间使用率*/
static struct proc_dir_entry *proc_entry; /*用于生成
                                    snapproc 子目录*/
static struct proc_dir_entry *proc_snap;/*用于创建实
                        际的文件并实现具体的读写方法*/
Procedure：proc_full_init()
{
    proc_snap = proc_mkdir("snapproc",NULL);/* 创
                                    建 snapproc 子目录*/
    proc_entry =create_proc_entry("full",0644,
                        proc_snap);/*实现 proc 接口*/
    if(proc_entry 为空)
        提示错误信息并返回;
    else{
        proc_entry->read_proc = proc_full_read;/* 声
    明读方法*/
        proc_entry->write_proc = NULL;/* 写方法
    置为空*/
        proc_entry->owner = THIS_MODULE;
    }
}
Procedure：proc_full_read()
{
    return _full;/*返回空间使用率*/
}
```

我们在每次写操作后更新_full 的值。因此，在每次写操作后计算出空间使用率， 将其赋值给_full，进行数据更新。在读取 full 文件中的数值时，调用 proc_entry 读方法，返回_full 的值。通过该 proc 接口，守护进程就可以实时地读取/proc/snapproc/full 文件中的数值，掌握即时空间使用率。

当空间使用率达到预设的阈值（例如 80%）时，守护进程会启动扩容程序。系统扩容的过程是先暂时挂起该卷并启动自动扩容进程，在 LVM 物理空间中为扩容卷分配所需的新增空间。若 LVM 当前空闲空间可满足扩容需求，空间分配成功，则更新 LVM 卷元数据，并通知内核新的数据空间分布。自动扩容结束后，将卷恢复，继续执行操作。

（2）逻辑卷扩容

在用户使用的过程中，可能会出现原计划的逻辑空间大小不能满足实际的需求的情况，需要更改逻辑空间的大小。

如前所述，元数据区的大小是与逻辑大小呈线性关系的。因此，当逻辑卷进行扩容时，新增空间的元数据需要我们分配相应的物理空间来保存。

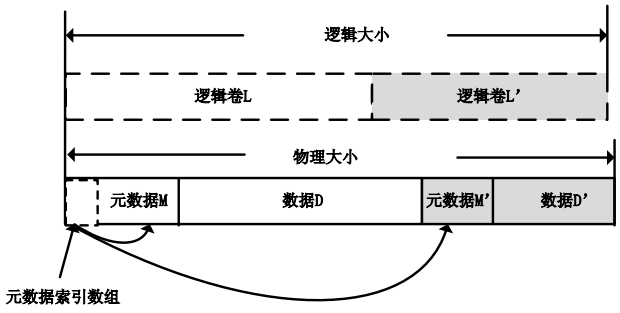

图 4 逻辑卷扩容示意图

如图 4 所示，是逻辑卷扩容方案的示意图。逻辑卷 L 是原分配的逻辑卷，元数据区 M 是 L 对应数据区的元数据区，数据区 D 是逻辑卷扩容前已分配的数据区。逻辑卷 L’是逻辑空间的新增部分，其大小由用户指定。由前面的介绍已经知道，新增的元数据区的大小应该是 L’的空间值与元数据区所占总空间的比例 P 的乘积，这样就得到元数据区 M’的大小。在逻辑卷扩容时，底层物理卷的扩容增量应该是 M’的空间值。

逻辑扩容后，物理卷的元数据区就由元数据区 M 和 M’组成。为统一管理这些分散的元数据区，所有元数据区的地址将保存在元数据索引数组中。元数据区的超级块部分大小固定为 4K，但目前只用到了极小的一部分，我们可将元数据索引数组保存在元数据区的超级块中。当有对物理卷的 I/O 操作时，通过该索引数组就可以确定物理卷的位置。

通过以上描述可以知道，逻辑卷的扩容的时间及大小都允许用户任意指定，具有很好的灵活性。传统 LVM 逻辑卷扩容的做法会导致很大的系统开销。Esnap[4][5]利用链表的方法改进了这一不足。在我们的方法中，元数据索引数组就可以起到将原有部分和新增部分组织在一起的作用。每次逻辑空间扩容，只需将新增部分对应的元数据区地址保存在该索引数组中即可。该方法实现非常简单且系统开销很小。

3 测试及结果分析

本文对不同容量的按需分配增量卷在扩容不同增量时所用的时间以及逻辑卷扩容后的读写性能进行了测试。所用操作系统为 RedHat AS server5（内核版本 2.6.8-128.el5），LVM2.2.02.39，device mapper1.02.28，所采用的工具是 IOmeter[8]。测试服务器为戴尔 Power Edge Server，3.20GHz Intel Xeon CPU，3GB 内存。

我们首先测试了物理卷自动扩容的速度。测试分三组，每组创建的卷的大小分别是 1G、10G 和 100G。在每组实验中，进行了十次扩容，测试在数据卷大小相同的情况下，扩容不同增量时所用的扩容时间，增量取值分别是 10% 、20%、 30%、 …、 100%。对于第一组来说，第一次扩容后，数据卷将增加 1G*10% 的空间，测试结束后，将卷销毁并重建一个 1G 的数据卷，然后按照 20%的增量对其扩容，以此类推。其他两组也是同样的测试方法。

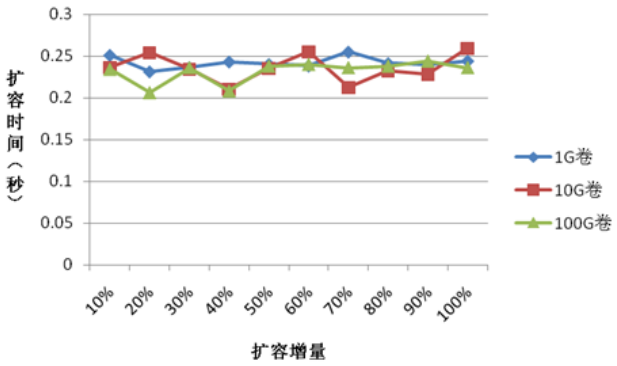

图 5 物理卷自动扩容速度测试结果

实验结果如图 5 所示，三组数据的值相差不大，不会因为卷容量或扩容增量的不同而有明显的变化，几乎是个常量，而且这个值很小。

可见，数据卷的自动扩容速度很快，用户不会因扩容而暂停正常业务，即使增量很大时也可瞬间完成，高速的扩容过程不会给用户的使用带来影响。

逻辑卷扩容测试对象是一个 200G 的卷及初始大小为 100G 经一次扩容后达到 200G 的卷。分别对两者的读写性能进行了测试。图 6、图 7 依次显示了两者的随机写、顺序写的测试对比结果。由测试结果可以看出，两个卷的写性能基本一致。同样，两个卷的读性能也相差不大，测试结果如图 8、9 所示。可见，逻辑卷的扩容不会对数据卷的读写操作产生影响。

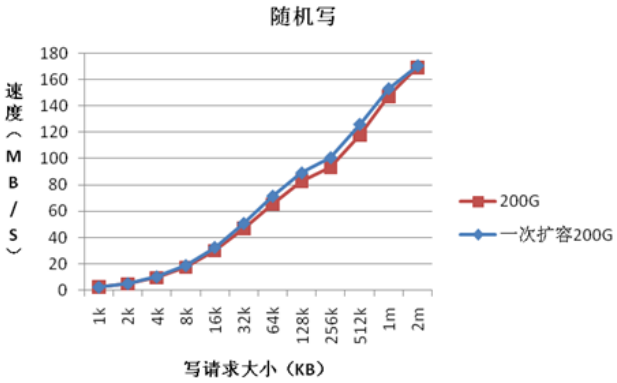

图 6 逻辑卷测试随机写性能测试结果

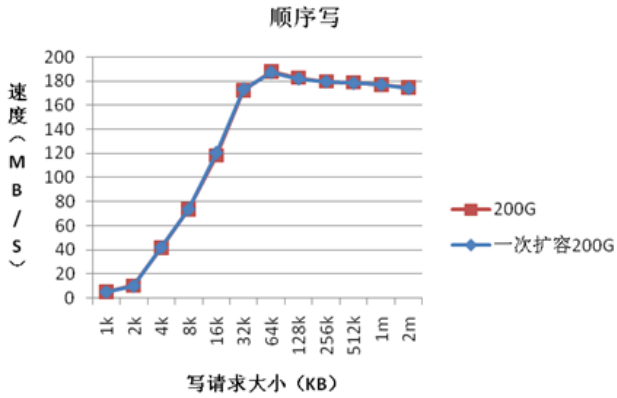

图 7 逻辑卷测试顺序写性能测试结果

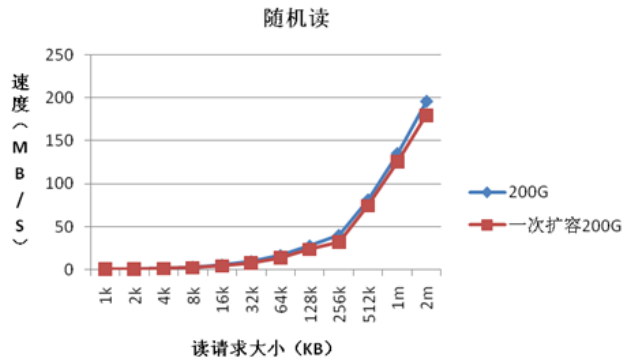

图 8 逻辑卷测试随机读性能测试结果

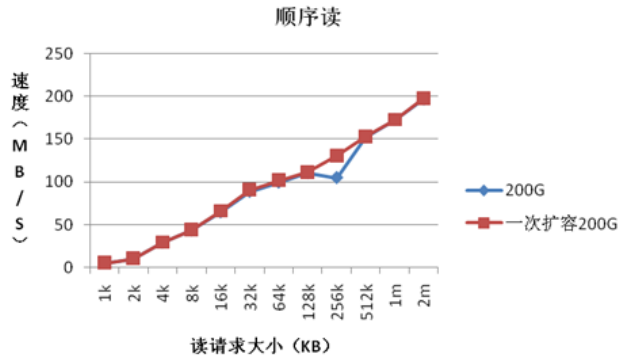

图 9 逻辑卷测试顺序读性能测试结果

4 结束语

本文介绍的非写前拷贝快照系统采用了按需分配的机制，系统中存在多个快照版本，数据卷实际的空间与逻辑值可以不相等。为实现数据卷物理空间与逻辑空间的分离，本文设计了两层卷的结构，即逻辑卷和物理卷。逻辑卷是一个对用户可见的虚卷，而物理卷是底层的实际的卷。在此基础上，我们设计并实现了对逻辑卷及物理卷进行自动扩容的解决方案。在创建数据卷时，初始分配数据区的空间可以小于用户指定的逻辑大小。当分配的空间不能满足用户的需求时，系统可以对数据卷的物理空间进行自动扩容。对每个卷来说，都有一个守护进程对磁盘空间使用率进行监控，当该使用率达到阈值时，系统会将卷挂起并自动启动扩容进程，待扩容结束后将卷恢复。与人工扩容相比，这种自动物理空间扩容技术有效地避免了溢出。逻辑空间扩容可以由用户任意指定，有很好的灵活性。为方便管理逻辑扩容后分散的元数据区，我们将元数据区的地址保存在元数据区索引数组中。与传统 LVM 的实现方法相比，该方法实现简单、系统开销小，而且可以将原有卷与新增部分有效地组织在一起。我们对物理卷自动扩容的速度及逻辑卷扩容后卷的读写性能进行了测试，实验结果表明扩容速度很快，不会影响用户的正常使用。逻辑卷经扩容后读写性能不会受到影响。

参 考 文 献

[1] D. T. Meyer, G. Aggarwal, B. Cully, G. Lefebvre, M. J. Feeley, N. C. Hutchinson, A. Warfield. Parallax: Virtual disks for virtual machines[J]. EuroSys 2008, 2008.

[2] A. Warfield, R. Ross, K. Fraser, C. Limpach, S. H. Parallax: Managing storage for a million machines[J].In Proceedings of the 10th Workshop on Hot Topics in Operating Systems, 2005.

[3] Daniel P. Bovet, Marco Cesati. Understand the linux kernel[M]. O'Reilly Media Inc.2005

[4] 付根希. 逻辑卷管理技术研究[D]. 天津:南开大学,2008.

[5] 谢广军. 虚拟化存储系统关键技术研究[D]. 天津:南开大学,2009.

[6] LVM2 user manual. http://sources.redhat.com/lvm2/.

[7] Ramac virtual array. http://www.redbooks.ibm.com/.

[8] Iometer. http://www.iometer.org/.

高岩，女，1986 年生，硕士研究生，研究方向：快照、非写前拷贝快照系统的自动扩容、网络存储系统。

曹瑞，男，1986 年生，硕士研究生，研究领域：为非写前拷贝快照系统的研究与应用。

甄彩军，女，1985 年生，硕士研究生，研究领域：为非写前拷贝快照系统的研究与应用。

徐广治，男，1987 年生，本科生，研究领域：为非写前拷贝快照系统的研究与应用。

王刚，男，1974 年生，教授，研究方向：并行计算、网络存储。

刘晓光，男，1974 年生，副教授，研究领域：并行计算、网络存储。