# NAS系统中卷管理器的研究与实现*)

欧阳凯 周敬利 余胜生 夏 涛
(华中科技大学计算机学院外存储国家专业实验室 武汉430074)

## The Research and Implementation of Volume Management in NAS

OU Yang-Kai ZHOU Jing-Li YU Sheng-Sheng XIA Tao
(College of Computer Science & Technology, Huazhong University of Science & Technology, Wuhan 430074)

**Abstract** The research and implementation of the volume management is the key technology in Network Attached Storage (NAS), which is also the important part in the research of virtual storage system. This paper researches the volume management technology and its error-control mechanism based on DAGs model, optimizes the online recovery mechanism, and improves the method of the hot-plug technology. Finally, we implement the optimum volume management based on FreeBSD.

**Keywords** NAS, Volume management, DAGs, Error control, FreeBSD

## 1 前言

在NAS系统中,卷管理器是实现可靠数据存储的关键技术,实现卷管理器可由专用的硬件实现,也可由系统软件实现,二者区别的关键点在于是否占用主机的机时和内存。

利用直接与主机相连接的RAID适配器,可以实现简单的卷管理器功能。这样,所有的卷操作和管理均不需要主机CPU来负担,而是由板卡上的专用处理器来执行。因此,硬件RAID卡具有高效性、易于出错控制,但是它造价昂贵、控制RAID级别不灵活。

随着CPU的迅猛发展,主机的计算能力已经不再是瓶颈,软件RAID技术越来越受到人们重视,一般是通过嵌在操作系统的文件系统与磁盘驱动之间实现卷管理器。其独立于硬件,在PC机上即可实现,而且控制灵活,可以同时实现多种RAID级别控制,满足用户不同需求。但是,PC机一般是不支持磁盘的热插拔,特别是有I/O的情况,往往会导致死机,给用户造成不可估量的损失。本文分析了软件RAID的可靠模型,在Carnegie Mellon University提出的DAGs模型[1]基础上,对其进行研究,提高了模型的强壮性,并在FreeBSD系统上实现了NAS系统中的卷管理器。

## 2 软件RAID可靠模型

磁盘和RAID系统的可靠性可用MTTF(Mean Time To Failure,平均无故障时间)值度量[2]。

RAID0由N台磁盘驱动器所构成,无校验盘,无容错功能,是典型的不可修复串联系统。设磁盘的平均无故障时间为$MTTF_{disk}$,则RAID0的MTTF为:

$$MTTF_{RAID0}=MTTF_{disk}/N$$

即RAID0的可靠性仅为单盘的1/N。

对于RAID1～5,每一级别的RAID都有D台磁盘作数据盘,而用N-D台磁盘来进行检纠错。另外,还可以加一个热备份磁盘来替换可能失效的磁盘,系统具备容错和单盘失效时系统可修复的功能,其可靠性模型属于马尔科夫串联型可修复系统。

设:(1)每个磁盘的失效率λ是一个常数,即$\lambda=1/MTTF_{disk}$;(2)每个驱动器的修复率μ也是一个参数,即$\mu=1/MTTR_{disk}$($MTTR_{disk}$为单台磁盘的平均修复时间);(3)失效事件和修复事件相互独立;(4)在RAID1～5中,磁盘总数为$N$,分成$N_g$个组,每组有$D$个数据盘,$C$个校验盘,一个热备份盘,$N=N_g(D+C+1)$。

据此,可画出如图1所示的马尔科夫可靠性模型。图中,状态"0"组内全部磁盘都工作正常,状态"1"表示有一个盘失效,系统处于降级和修复运行,仍能提供服务,状态"2"为系统的吸收状态,表示有两个盘失效,系统无法正常工作。

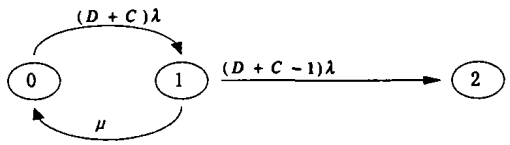


图1 单组(D+C)个盘的马尔科夫可靠性模型

依据此马尔科夫模型,可得出如下微分方程组:

$$P_0^{(1)}(t)=-(D+C)\lambda P_0(t)+\mu P_1(t)$$

$$P_1^{(1)}(t)=(D+C)\lambda P_0(t)-\mu P_1(t)-(D+C-1)\lambda P_2(t)$$

$$P_2^{(1)}(t)=(D+C-1)\lambda P_1(t)$$

对上式进行拉氏变换,并把初始条件$P_0(0)=1, P_1(0)=P_2(0)=0$代入,可解得系统可靠度为:

$$R(s)=P_0(s)+P_1(s)=\frac{(D+C)s+[2(D+C)+1]\lambda+\mu}{s^2+\{[2(D+C)+1]\lambda+\mu\}s+(D+C)(D+C-1)\lambda^2}$$

因此有:

$$MTTF=\lim_{t\to\infty}\int_0^t R(t)dt=\lim_{s\to\infty}sL\{\int_0^t R(t)dt\}=\lim_{s\to 0}sR(s)/s$$

$$=R(0)=\frac{[2(D+C)+1]\lambda+\mu}{(D+C)(D+C-1)\lambda^2}$$

当$[2(G+C)+1]\lambda<<\mu$时,有

*)课题来源:863计划,编号:2001AA111011 存储虚拟化及其文件系统研究。欧阳凯 博士研究生,研究方向为高性能网络存储技术。周敬利 教授,博士生导师,研究方向为高性能网络存储技术及多媒体计算技术。余胜生 教授,博士生导师,研究方向为数字信号处理及网络存储技术。夏 涛 讲师,研究方向为多媒体及网络存储技术。

$$MTTF=\frac{MTTF_{disk}^2}{(D+C)(D+C-1)MTTR_{disk}}$$

由于 RAID 系统共有 $N_g$ 组，最后可得 RAID1～5系统的平均无故障时间为：

$$MTTF_{RAID1\sim5}=\frac{MTTF_{disk}^2}{N_g(D+C)(D+C-1)MTTR_{disk}}$$

上式表明，RAID1～5具有相当优异的可靠性。一般地，$MTTF_{disk}$ 为数万小时，而 $MTTR_{disk}$ 仅为数小时，因而 $MTTF_{RAID1\sim5}$ 比 $MTTF_{disk}$ 提高了数千甚至上万倍，这也可以理解为在 RAID 系统中两盘同时失效的概率比单盘失效的概率要小得多。

## 3 DAGs 模型

### 3.1 DAGs 原子操作

原子操作构建了卷管理器的底层基础，实现了硬件无关性。其原子操作，如表1。内存管理用于卷管理器与共享池协商使用缓冲区；锁维护卷管理器的一系列的锁，这些锁保证了进程之间的共享、独占性；算法提供了卷管理器的数据的编码解码，以及数据的校验。Reed-Solomon 编码[3]是一种基于块设备的含有出错修复功能的编码技术。

表1 DAGs 原子操作的类型

| 类型 | 原子操作 | 描述 |
|---|---|---|
| 磁盘 | 读(Rd) | 雁磁盘向缓冲读数据 |
| 磁盘 | 写(Wr) | 从缓冲向磁盘写数据 |
| 内存管理 | MemA | 申请一个缓冲区 |
| 内存管理 | MemD | 释放一个缓冲区 |
| 锁管理 | Lock | 申请一个锁 |
| 锁管理 | Unlock | 释放一个锁 |
| 算法 | XOR | 缓冲数据校验 |
| 算法 | Q | 生成 Reed-Solomon 编码 |
| 算法 | $\overline{Q}$ | Reed-Solomon 编码解码 |

### 3.2 DAGs 基本模型[1]

我们可以通过软件实现表1定义的一系列的原子 RAID 操作，为了更好地描述各种原子操作的关系，区分 RAID 各种模型的工作原理，引入了 DAGs 模型，该模型里每一个结点表示一个原子操作。Rd-XOR-Wr 链表述读并修改数据，重新计算校验区域数据的过程；Rd-Wr 链表述读取旧数据和重新写数据；Rd-XOR 链表述从旧数据中获得校验数据；NOP 结点表示非操作结点，是个标志结点；添加 NOP 结点的意义在于简化 DAGs 的结构，使之更加简明，其具体使用在下一节结合控制机制阐述。

## 4 出错控制机制与在线恢复机制

### 4.1 出错控制机制

DAGs 出错控制理论是基于 Roll-away 恢复机制[1]。Roll-away 恢复机制是由两个部分组成，一个是向前(forward)出错恢复机制，另一个是向后(backward)恢复机制。为了更好地明白 Roll-away 恢复机制是如何工作的，我们首先必须明白冗余磁盘是怎么将数据编码存储在磁盘上的。每一个数据编码区是由两个部分组成的，一个是数据块，另一个是校验块，由两种不同类型的标志表示。为了使冗余磁盘能够容错，即在一个或多个标志丢失的情况下，用户数据信息也能保证完整性，必须设置数据编码区的有效性。譬如：当写新数据时，卷管理器则需修改相应的校验块的标志，也就是要更新校验块的数据。

恢复机制的方向性是由一个原子操作失败的时间决定的。在 DAGs 结构中，通过添加一个提交(Commit)结点来区分一次动作的两个阶段，如图2。

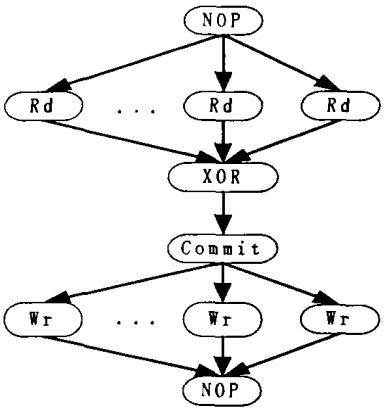


图2 一次动作的两个阶段

在第一个阶段，已经存在的数据编码块还没有被修改，在这个阶段的结点能够容易地被取消操作，譬如读磁盘数据或者奇偶校验。显然，在第二阶段，我们的原子操作涉及到修改数据编码块的标志，但是，并不是所有的原子操作都涉及到第二阶段，譬如读磁盘操作不需要修改任何数据编码区的标志，相反写磁盘操作必须修改数据编码区标志。

当一个原子操作在第一阶段失败时，DAGs 控制如图3。结点读操作发生在提交操作之前，如图3的黑体结点，这导致向前操作被终止，向后控制进程启动，它会取消刚刚完成动作的结点操作(Rd pass)，如果出错发生在提交前，则系统认为这个图从来没有被执行过。

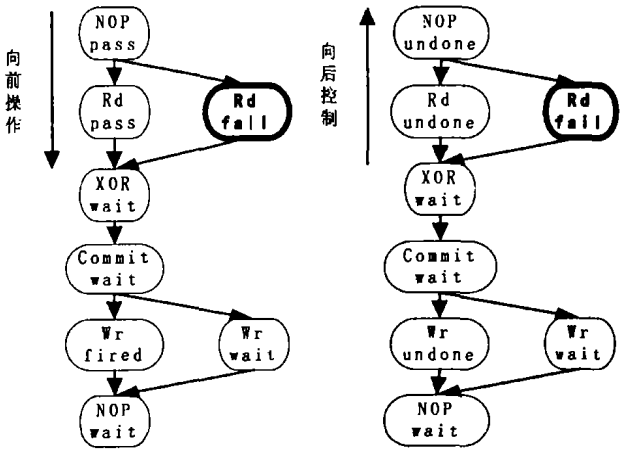


图3 提交前出错控制图

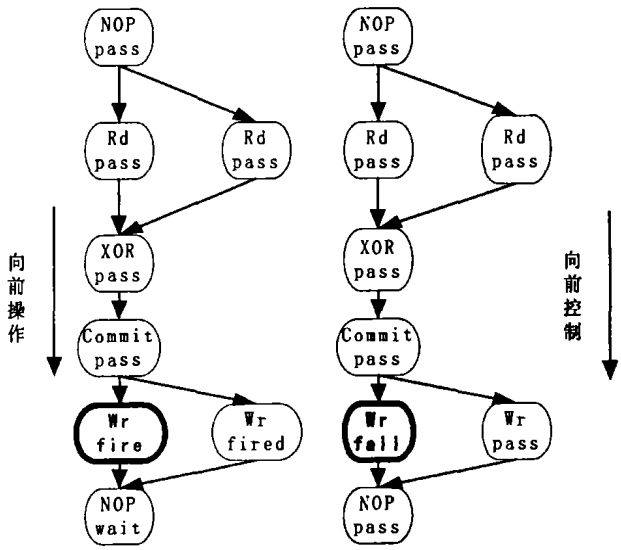


图4 提交后出错控制图

当一个原子操作在第二阶段失败时，DAGs 控制如图4。结点读操作发生在提交操作之后，如图4的黑体结点，向前操作会继续向前，Wr 结点右边的动作已经完成就会进入 pass 状态，如同没有错误发生一样，向前错误控制进程会把 fire 状态的结点设置为 fail 状态。整个执行过程会继续，系统认为该图顺利执行了。

### 4.2 在线恢复机制

作为一个完善的卷管理器的设计，在线重建数据机制是必需的，即当一个磁盘出错后，更新磁盘，使卷恢复到具备容错功能。重建数据到一个热备份盘的理论是基于磁盘调度的重构算法[4]。

通过在冗余磁盘阵列中保持一个或多个热备份盘，当具有容错机制的卷出现一个磁盘的数据丢失时，卷管理器能在不中断服务的情况下，自动启动一个热备份盘，重构那个丢失的盘的数据。这对于 NAS 系统对外提供连续服务功能来说，是至关重要的。

当一个磁盘出错时，卷自动进入降级模式(不再具有容错功能)，同时，将触发卷管理器产生一个后台重构进程使卷从降级模式恢复到正常模式，该进程会成功地重构那个出错磁盘所丢失的数据和奇偶校验单元到热备份盘上。这样的机制就叫重构机制。当重构完成后，该卷又具有容错能力。

基于磁盘调度的重构算法是，当需要重构时，会产生 C 个重构进程，C 代表了该卷的磁盘个数。其中 C-1个进程分别对应那些存在有效的磁盘，每个进程的算法是一致的，其算法如下：

do：

1. 找到该磁盘需要重构的单元的最小数目，并做标志。

2. 发出一个低优先级的请求线程：从磁盘读具有重构标志的单元到内存缓冲区。

3. 等待读操作完成。

4. 提交该单元的数据到一个缓冲区管理中心进行 XOR 操作，或者在缓冲区管理中心没有足够内存接受该单元数据时，阻塞直到有内存空间接收该单元。

While(所有的数据单元都被读入内存)

另一个单独的进程专门针对那个替换的磁盘，其算法如下：

do：

1. 缓冲区管理中心申请下一个填满数据的缓冲块。

2. 如果填满数据的缓冲块没有准备好，则阻塞等待。

3. 开一个低优先级的写线程：从缓冲块中读取数据，并处理，写向替换的磁盘。

4. 等待写完成。

While(那个出错的磁盘信息已经完全重构)

缓冲区管理提供了一个用于校验的用户数据和校验数据的仓库中心。当 C-1进程中的任何一个进程提交一个新的缓冲块，缓冲区管理器会 XORs 这些数据，并统计出奇偶校验块，直到该标志的数据都统计完成，则设置为可用(指可给替换磁盘使用的缓冲块)状态，并处于等待；当它接到一个来自那个替换的磁盘的请求信号，则将数据传给该磁盘的缓冲块，并从可用缓冲块链表中把该缓冲块释放。这种设计方法的优点总是保持一个低优先级的对磁盘的请求，这样就可以占有用户不使用的部分带宽。在基于磁盘调度的重构算法中，由于用户请求磁盘服务是随机的，某些重构进程读数据会比另一些重构进程快许多，因此，缓冲区管理器必须保存这些信息直到较慢的重构进程的相应的数据和奇偶校验数据到达。

## 5 软件 RAID 系统的实现

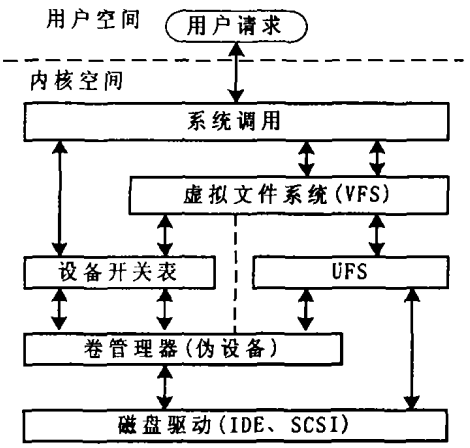


图5 卷管理器在 FreeBSD 内核的层次结构图

我们的卷管理器是基于 FreeBSD 操作系统[5]，利用了伪设备编程技术[6]实现的，其在 FreeBSD 内核中层次结构如图5所示。卷管理器是以伪设备的形式实现的，处于文件子系统和设备驱动之间。当需要使用卷管理器时，用户可以通过系统调用访问设备开关表，对伪设备层控制，即对卷管理器控制，用户对文件系统的读写操作，在具体写磁盘时，就不是直接访问物理硬盘，而是通过卷管理器控制，从而实现用户对数据的可靠、冗余需求。当用户不需要卷管理器时，可以通过系统调用直接访问 UFS 文件系统(Unix 标准文件系统)[7]。

### 5.1 内核支持

设备开关表[7]是实现伪设备技术的关键结构，也是卷管理器控制的关键结构。设备开关表一般分为字符设备开关表和块设备开关表。但是，在现在的 FreeBSD 系统中，所有的设备开关表都统一使用字符设备开关表。用户向 kernel 发出的对 raid 伪设备的各种控制信息都是通过 ioctl 系统调用实现的。用户层对 raid 伪设备的 ioctl 对应内核的 raidctlioctl( )和 raidioctl( )。

在系统启动时，内核加载伪设备(通过 raidattach()实现的)，并通过内核函数 make_dev(&raidctl_cdevsw，0，0，0，0x644，"raidctl")创建 raidctl_cdevsw 实体，把它和设备名 raidctl 联系在一起；当用户需要建一个卷时，内核会调用 disk_create (raidPtr-> raidid，&rs-> sc_dkdev，0，&raid_cdevsw，&raiddisk_cdevsw)创建 raid_cdevsw 实体，这是通过 raidinit()实现的。raidctl_cdevsw 和 raid_cdevsw 这两个数据结构是通过 raidctl_softc 和 raid_softc 控制的。其关系如图6所示。这两个 softc 的数据结构是内核控制伪设备的数据结构，不仅有指向系统调用函数指针，而且还包含了该伪设备特有的一些数据结构。

### 5.2 磁盘驱动支持

传统 FreeBSD 的 IDE 驱动层认为 IDE 磁盘是不会被拔出的；而且 IDE 磁盘协议标准[8]也不支持动态插拔磁盘，因此，对于磁盘被拔出的情况没有处理，同样地，它也认为 IDE 磁盘是不可能在线插入的。当正在写盘时，IDE 磁盘被拔出，传统的 IDE 驱动会认为写盘出错只是暂时的，会不断地试图写盘，而写盘总是失败，IDE 驱动并没有做任何处理，通常会导致系统崩溃。

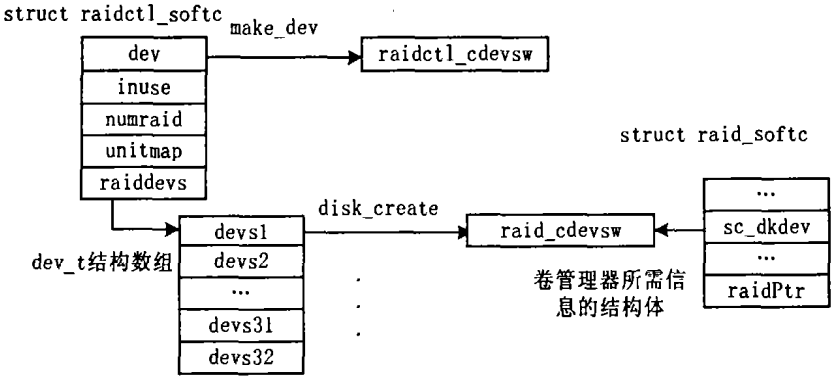


图6 卷管理器内核控制结构关系图

我们的目标是提供IDE磁盘的热插拔功能，实时支持软件RAID的出错控制和在线恢复机制。首先，采用自制转接口，使IDE在物理电气特性上支持动态插拔；在三次I/O失败后，我们认为该磁盘出错，不再可写或是被拔出；同时，给出了用户控制接口以支持磁盘在线插入功能。我们是通过两条途径操作IDE磁盘的，一是通过专门的ATA控制接口，在设备驱动一层上控制；另一个是通过FreeBSD提供的"/dev/io"设备名获得外设的I/O特权，对IDE的寄存器[8]直接访问。

添加ATA控制接口的目的是为了便于用户层和内核之间的相互通信，内核部分是利用字符设备开关表(ata_cdevsw)，在这里，我们仅需要实现ioctl的功能。通过ioctl，可以检测IDE磁盘的拔出、插入、系统ATA通道的状况，以及访问ATA状态寄存器和交换状态寄存器的基址等功能，在用户层，提供一个命令(atacontrol)，便于用户控制。

当插入一个新盘时，内核利用ata_probe()和ata_attach()可以让系统认知该盘；当拔出磁盘时，内核利用ata_reinit()重新初始化该ATA通道，则系统得知该盘已被拔出。提供ATA通道的寄存器的控制是为了方便热插拔的守护进程的轮讯，也是为了实现硬件无关性。因为不同的硬件设备提供的ATA通道的基址不是完全一致的，实时从内核中获得是最为可靠的，也是无需考虑硬件的。

**结束语** 本文对DAGs模型的出错控制机制和在线恢复机制提出改进，并在FreeBSD操作系统上实现了NAS系统的卷管理器。

NAS系统中的软件RAID技术还有许多有待完善或值得继续探讨的理论和应用问题。譬如：作为高端NAS服务器，应该在RAID的基础上提供文件系统快照功能以及双机热备份功能，以实现其高可靠性。这也是我们下一步的研究方向。

## 参考文献


1 Courtright W V II, Gibson G, Holland M, Zelenka J. A Structured Approach to Redundant Disk Array Implementation. CMU-CS-96-137 10 June 1996

2 Chen P M, Lee E K, Gibson G A, et al. RAID: High-Performance, Reliable Secondary Storage. ACM Computing Surveys, 1994, 26(2): 145～185

3 Guruswami V, Sudan M. Improved Decoding of Reed-Solomon and Algebraic-Geometric Codes. IEEE Symposium on Foundations of Computer Science, 1998

4 Holland M. On-Line Data Reconstruction In Redundant Disk Arrays. 1994

5 The FreeBSD 4.5 release source code. ftp://ftp.freebsd.org 12 July 2002

6 Writing Device Drivers. Sun Microsystems, Inc., 2000

7 Tanenbaum A S, Woodhull A S. Operating System: Design and Implementation Second Edition(影印版). 清华大学出版社, 1997. 401～503

8 Schmidt F 著, 精英科技 译. SCSI总线和IDE接口：协议、应用和编程(第二版). 中国电力出版社, 2001. 45～54


(上接第151页)

智能化程度更高。它不仅可以报告编译时的语法错误，还可以发出一些警告信息，以报告可能的用户疏漏，如变量未初始化。

规则检查工具可以以编译器插件的形式存在。它可以从编译器那里得到扫描前的源程序、词法分析结果和语法分析后的中间表示，然后进行相应的分析和规则检查，将结果以文件或者显示的形式输出。采用插件形式的好处是方便、灵活，充分利用编译器的中间运行结果。但是这需要得到编译器的接口，因此需要编译器厂商的支持。

规则检查插件最好能有供用户选择的选项。用户是否希望进行规则检查、希望进行哪些规则检查、是部分检查还是全部检查，这些都可以通过设置偏好来完成。

**结束语** 本文对程序设计规则检查做了介绍，列出了各种程序设计标准。并在规则检查如何保障软件质量的问题上，从正确性、可理解性、可移植性、安全性等几个角度作了剖析。然后介绍了几种著名的规则检查工具。最后对规则检查的应用领域推广、标准化、实用化等方面做了一些思考。

本文的目的主要在于引起相关人员对程序设计规则检查的重视：使程序员养成良好的编程习惯；使软件需求分析与设计人员可以以更严格、精确的方式描述、刻画需求和设计；并且倡导程序设计规则检查工具的使用。

## 参考文献


1 LDRA Ltd. LDRA Testbed MISRA C Checking, version 2.0. product documentation, 2000

2 Scott Meyers. Effective C ++: 50 Specific Ways to Improve Your Programs and Design. Addison-Wesley, 1992

3 Scott Meyers. More Effective C ++: 35 New Ways to Improve Your Programs and Designs. Addison-Wesley, 1995

4 MISRA. Development Guidelines for Vehicle Based Software, MIRA, CV10 0TU, 1994. http://www.misra.org.uk/

5 陈世忠. C++编码规范. 北京：人民邮电出版社, 2002