RAID 阵列中高速 Cache 管理的优化*

钟永锋　张江陵
（华中理工大学计算机系　武汉　430074）

摘　要

文中分析了独立冗余磁盘阵列的性能瓶颈，论述了在磁盘阵列中引入 Cache 的必要性，在分析了一种典型的 Cache－buffer 管理模块后，综合评述了优化 Cache 的几种途径，文章最后提出一种新的 Cache 管理策略。

关键词：磁盘阵列，高速 Cache
中图法分类号：TP333.22

Optimizing Strategy of Cache Management in RAID

Zhong Yongfeng　Zhang Jiangling
（Department of Computer Science
Huazhong University of Science and Technology　Wuhan　430074）

Abstract: For eliminating the bottleneck of performance, it's necessary to introduce cache in Redundant Arrays of Independent Disks. After analyzing a typical module of cache－buffer management, this paper appraises several ways of optimizing cache management and presents a new strategy of cache management.

Keyword: RAID, cache, buffer
Class number: TP333.22

一、前言

近年来，随着芯片设计水平和制造工艺的不断提高，计算机主机性能有了大幅度的提高，这就要求外存储器容量和存取速度与处理器的性能相匹配。虽然现在由于磁盘制造工艺和磁头技术的改进，磁盘性能已经有了很大的提高。但硬盘作为计算机系统中的机电部件，本身受到机械执行惯性的制约，其 I/O 性能始终不能令人满意，成为计算机系统性能提高的瓶颈。为了解决这一问题，出现了采用冗余技术构造的 RAID 阵列。RAID 阵列由于采用数据分块技术，即在多个磁盘上交叉存放数据使得多盘可并行操作，I/O 响应时间得到改善，同时利用冗余容错技术，极大地提高了磁盘阵列的可靠性和可用性。

然而人们对速度的追求是无止境的，当初 RAID 的提出是为了解决 I/O 性能上的瓶颈，经过十几年的发展，现在对 RAID 的性能的提高要求更为迫切。反观 RAID 阵列，一方面单个磁盘的存取速度与普通磁盘一样，仍停留在毫秒级。另一方面 RAID 阵列的校验结构决定了它在下列情况时会出现性能下降：①随机的小块写。②磁盘失效，系统降级运行。这些都使 RAID 阵列比同样磁盘组成的非校验阵列性能差。如果不进一步研究，寻找到提高 I/O 的途径，RAID 将成为新的

* 文稿收到时间：1998 年 4 月 3 日

瓶颈。围绕着如何提高现有 RAID 的传输率，减少响应时间，人们采用高速 Cache 优化阵列性能。利用 Cache 存取速度快的优点，配以一定的 Cache 数据管理策略，从一定程度上提高了数据响应速度和数据传输率。可以说，Cache 已成为目前提高 RAID 产品性能的关键技术之一。

二、磁盘阵列控制软件中的 Cache 管理

1. Cache 管理的功能

典型的磁盘阵列控制软件主要由以下几大模块组成：初始化参数设置，命令分解，数据分块/重组，Cache－Buffer 管理，I/O 调度，校验信息计算，数据重构，数据修复。如下图所示：

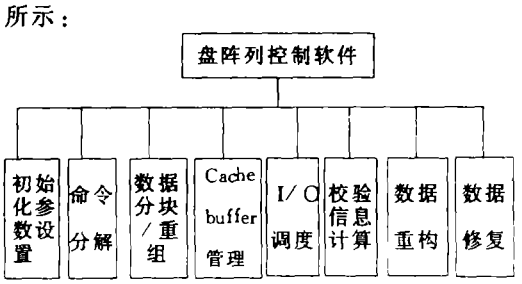


图 1

其中的 Cache－Buffer 管理子模块就是在上下位机之间，调度数据的移动，决定流动的方式，确定是否进行磁盘操作。对于读数据命令，先检查 Cache 中是否包含该数据，如果读命中，就可以立即响应上位机的请求。不命中，就生成相应的读磁盘操作；写命令比读命令的处理复杂，即使命中，也必须修改该数据所在校验串的校验块，为此必须先进行读盘操作，读出其它数据块，计算出校验后还必须进行磁盘操作更新校验块。为了优化，一般为每一个磁盘分配包括数据块 Cache 在内的多块 Cache。非数据块 Cache 的目的主要是对 Cache 内容加以索引，以加快对 Cache 中数据的搜寻速度。对读写命令，为减少数据在 Cache 中的拷贝，将数据分块/重组与命令分解过程结合进行。对于写命令，命令分解时，每个写子命令经分块得到的数据通过指针指向其头地址，同时给出数据块长度；对于命令，分解后的每一个读子命令带有数据重组时的目的地址和长度，以便读操作完成时直接进行数据重组。通过这些手段，加快了 Cache 处理的速度，相应地提高了 I/O 响应速度和数据传输率。

2. 存在的不足

我们在研究中发现，尽管采用了这些优化方法，但整个系统性能提高的程度并没有预期的大。究其原因，主要有以下几个，一是当 Cache 满时，就必须停下所有操作，将 Cache 中需要回写的数据写入磁盘，我们称之为 Cache 腾空。腾空 Cache 时上位机不能下传数据，这就意味着腾空时，上位机的请求得不到响应，急剧降低了 I/O 响应速度。二是小块数据的写操作涉及太多的 I/O 操作，降低了数据传输率。因为小块数据写时，必须先读出它所在的校验组中的其他旧数据和旧校验，再计算新校验，才能写入。如果同时有对同一个校验组的多个小块数据的写操作，只能一个一个块的进行读－更新－写，所费系统时间反而比大块操作来的多。

3. 解决的方法

对第一个问题，考虑通过改进调度策略或 Cache 管理策略，以减少 Cache 饱和的机会，尽可能降低 Cache 腾空的次数。对第二个问题，可采用小块数据不立即对盘操作，而合并成大块数据再进行磁盘操作，以减少 I/O 次数和系统等待盘操作延迟的时间。总之，就是利用 Cache 来提高现有 RAID 阵列的性能。具体分以下几个方面。

①增大 Cache 容量：

由于内存芯片的价格不断降低，采用更大容量的 Cache 已不必象过去那样付出昂贵的代价。而增大 Cache 容量带来的益处是显而易见的。首先，增大了 Cache 容量，Cache 中的数据块更多，可以预取更多的数据，

Cache的命中率可以提高，加快了响应速度。Cache腾空的次数也相应减少，当然就降低了磁盘操作次数。二是小块数据可利用较大的Cache合并成大块，并行读写磁盘，提高了数传率。实验结果表明，通过增大Cache容量甚至可获得大约300%的性能增长。当然，Cache容量的增大也并不是无限制的，一方面受板卡插槽的限制，另一方面，研究表明，当Cache容量增大到一定程度，会出现性能饱和，那时性价比会急剧下降。

②选择合适的Cache预取和替换策略：

由局部性原理可知，存储访问有集成一组的倾向，以至一旦某个位置被访问，很有可能它附近的位置也要被访问。所以每当Cache不命中，需要从磁盘读取数据时，除了读取不命中数据外，还需附带读取一定的预取数据，这样可以提高相临操作的Cache命中率。研究表明，普通磁盘大约55%－77%的数据请求是顺序的，所以预取可能是读出整个磁道的数据，而不仅仅是所需的块。由于预取涉及更多的数据，当然存在一个覆盖原有Cache数据的问题，究竟覆盖那些数据即采用何种替换策略，往往又与具体应用环境有关。可以保持对Cache中每一数据块的统计记录，诸如某一数据自它被从磁盘读入Cache后被应用了多少次。借助存取记录文件，采用LRU策略，来决定替换和覆盖哪些数据。

③Cache命令合并：

我们知道，随上位机到达的命令中的数据大小不同，有的数据刚好是一满校验组，这种情况下可并发操作所有盘的读写，仅需一次I/O操作，是最理想情况。如果是不满校验组的读，可直接对涉及的盘进行读操作。如果是不满校验组的写，就必须从同一校验组的其它盘先读出旧的数据和旧的校验，计算出新的校验，再启动磁盘操作，所以总共需至少两次磁盘I/O操作。可以将Cache中的小块数据的I/O请求按其是否相邻进行合并，而暂不对盘操作，一旦形成大块命令，再按大块命令对盘进行I/O操作，这样可以降低由于小块I/O带来的对盘的频繁操作，减少磁盘操作等待延时，提高数传率。

④减少上位机等待Cache腾空的几率和时间：

Cache满了以后就无法再接受数据，必须先将数据写回到盘上。此时上位机不得不处于等待状态。为了减少出现等待的几率，就必须保证Cache中总有空间可用，这就意味着必须不断地将Cache中的修改过的内容写回到磁盘，这样在上位机需要空间存放数据时，这部分空间就是可用的。实际操作中，Cache满的情况是不可避免的，这时必须尽量减少Cache腾空的时间，腾空时间主要由写盘的命令个数决定，可以将同一盘的相邻数据合并为同一个写命令，显然这比直接执行命令队列的命令效率更高，I/O操作要少。

三、一种新的Cache数据管理方法

基于上述分析，我们提出了新的磁盘阵列的Cache管理策略。这种策略与以前常用的Cache管理策略相比，有以下几点不同。

1.等容量双Cache策略

从我们已经完成的研究和其它国内文献来看，阵列Cache可以将实际的读盘和写盘操作转变为对Cache数据的读写，大大缩短了响应时间，提高了阵列性能。但有一点也必须注意到，一旦出现Cache满的情况，Cache管理模块不得不先将Cache中的数据写到盘上，以腾出Cache接受新的数据。Cache越大，腾空Cache所需的时间越长，从上位机来的请求得不到响应的时间也就越长。这往往是造成阵列I/O响应性能恶化的重要原因。采用双Cache，将为每一个硬盘分配的Cache分为容量相同的两部分，使用时如果一个满了或是容量不足，就启动使用另一个，同时利用并行处理机制，将已满的

Cache的内容派生一个任务处理，写到盘上，保证写命令到达时，系统总有一个可用的Cache。只有在写命令执行过程中出现两个Cache都满时，才会出现让上位机等待的最不利情况。很显然，两个Cache同时满的概率比只用一个Cache时低。双Cache都写满时，才会需要停止接受上位机的命令来进行腾空。腾空时还是基于减少I/O操作的思想，对需要写盘的命令进行扫描，找出同一盘数据相邻的命令，合并成一个命令，再将多个盘的命令并行执行，降低腾空等待时间。

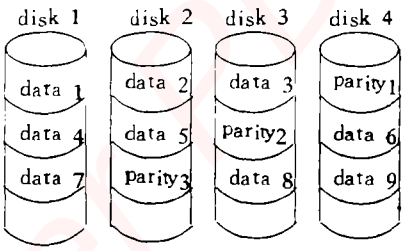
2.并行预取策略

读数据时，如果所需的数据不在Cache中，即读不命中，按以往的读盘操作，往往根据数据局部性原理，采用基于磁道的预读方法，将同磁盘的其余数据也读入Cache。这对普通单盘是有一定效果的，但对磁盘阵列而言，由于阵列是将数据分块后散布在相邻磁盘上，所以磁盘阵列与普通单盘的数据分布有一些不同，普通单盘的数据分布是逻辑上连续的物理上往往也连续，而磁盘阵列恰恰相反，逻辑上连续的物理上一定分布在另一个磁盘上。所以对缺失块所在磁道的数据预取可能效果不佳，真正需要预取的可能是同一校验组的其它数据。因此，为了提高读操作的效率，我们提出，在读未命中数据块时，不但将各块所在磁道的若干数据块一并读入，而且将该数据块所在校验组的其余数据一起读出。具体读多少块可由Cache大小和数据分块大小决定。

③小块写与预读结合进行。

磁盘阵列的小块写问题一直是制约阵列性能提高的一个难题。主要是小块写必须经过读——更新——写的过程。这中间有两次I/O操作。既然无法减少这个时间，能不能提高每次I/O操作的效益呢？还是出于数据局部性的考虑，我们提出了将小块写与预读结合进行的策略。如图2所示。

假设数据分布如下：数据块①②③加校验块①组成一个校验串，数据块④⑤⑥+校验块②组成一个校验串。假设现在需要写数据块③④⑤，考虑到数据块①④、②⑤、数据块③+校验块②、校验块①+数据块⑥分别处于同一磁盘的相邻位置，所以可以充分利用四个盘的一次并行读将这些数据同时读入Cache，在Cache中计算和更新校验。这种策略在同样的操作内，既提供了所需数据，也预读了部分相邻数据，这些数据根据局部性原理，极有可能在紧接着的数据请求中命中。



四、结语

通过以上的分析可以看出，高速Cache作为提高磁盘阵列性能的一项技术，还存在着种种不完善之处，甚至还可能造成性能瓶颈，所以有必要对其进行更深一步的研究，鉴于国内外对此研究也刚起步不久，至今还没有提出一个完善的磁盘阵列Cache策略，我们将在今后的研究中，对上述Cache管理策略，进一步加以完善。

参考文献

1 "RAID: Theory and Practice", comp, arch. storage FAQ, 1997

2 The RAID Book, Published by The RAID Advisory Board, 1994

3 "RAID: High - Performance, Reliable Secondary Storage", Peter M. Chen, Edward K. Lee

4 "It's All in the SmartCache", Patrick Norton, 1997

5 "廉价冗余磁盘阵列(RAID)Cache浅析", 刘桂兰，祝天龙，电子计算机外部设备，1995

嵌入式资源免费下载

总线协议：

1. 基于 PCIe 驱动程序的数据传输卡 DMA 传输
2. 基于 PCIe 总线协议的设备驱动开发
3. CANopen 协议介绍
4. 基于 PXI 总线 RS422 数据通信卡 WDM 驱动程序设计
5. FPGA 实现 PCIe 总线 DMA 设计
6. PCI Express 协议实现与验证
7. VPX 总线技术及其实现
8. 基于 Xilinx FPGA 的 PCIE 接口实现
9. 基于 PCI 总线的 GPS 授时卡设计
10. 基于 CPCI 标准的 6U 信号处理平台的设计
11. USB30 电路保护
12. USB30 协议分析与框架设计
13. USB 30 中的 CRC 校验原理及实现
14. 基于 CPLD 的 UART 设计
15. IPMI 在 VPX 系统中的应用与设计
16. 基于 CPCI 总线的 PMC 载板设计
17. 基于 VPX 总线的工件台运动控制系统研究与开发
18. PCI Express 流控机制的研究与实现
19. UART16C554 的设计
20. 基于 VPX 的高性能计算机设计
21. 基于 CAN 总线技术的嵌入式网关设计
22. Visual C 串行通讯控件使用方法与技巧的研究
23. IEEE1588 精密时钟同步关键技术研究
24. GPS 信号发生器射频模块的一种实现方案
25. 基于 CPCI 接口的视频采集卡的设计
26. 基于 VPX 的 3U 信号处理平台的设计
27. 基于 PCI Express 总线 1394b 网络传输系统 WDM 驱动设计
28. AT89C52 单片机与 ARINC429 航空总线接口设计
29. 基于 CPCI 总线多 DSP 系统的高速主机接口设计
30. 总线协议中的 CRC 及其在 SATA 通信技术中的应用
31. 基于 FPGA 的 SATA 硬盘加解密控制器设计
32. Modbus 协议在串口通讯中的研究及应用

VxWorks：

1. 基于 VxWorks 的多任务程序设计
2. 基于 VxWorks 的数据采集存储装置设计
3. Flash 文件系统分析及其在 VxWorks 中的实现
4. VxWorks 多任务编程中的异常研究
5. VxWorks 应用技巧两例
6. 一种基于 VxWorks 的飞行仿真实时管理系统
7. 在 VxWorks 系统中使用 TrueType 字库
8. 基于 FreeType 的 VxWorks 中文显示方案
9. 基于 Tilcon 的 VxWorks 简单动画开发
10. 基于 Tilcon 的某武器显控系统界面设计
11. 基于 Tilcon 的综合导航信息处理装置界面设计
12. VxWorks 的内存配置和管理
13. 基于 VxWorks 系统的 PCI 配置与应用
14. 基于 MPC8270 的 VxWorks BSP 的移植
15. Bootrom 功能改进经验谈
16. 基于 VxWorks 嵌入式系统的中文平台研究与实现
17. VxBus 的 A429 接口驱动
18. 基于 VxBus 和 MPC8569E 千兆网驱动开发和实现
19. 一种基于 vxBus 的 PPC 与 FPGA 高速互联的驱动设计方法
20. 基于 VxBus 的设备驱动开发
21. 基于 VxBus 的驱动程序架构分析
22. 基于 VxBus 的高速数据采集卡驱动程序开发

Linux：

1. Linux 程序设计第三版及源代码
2. NAND FLASH 文件系统的设计与实现
3. 多通道串行通信设备的 Linux 驱动程序实现
4. Zsh 开发指南-数组
5. 常用 GDB 命令中文速览
6. 嵌入式 C 进阶之道
7. Linux 串口编程实例
8. 基于 Yocto Project 的嵌入式应用设计
9. Android 应用的反编译
10. 基于 Android 行为的加密应用系统研究
11. 嵌入式 Linux 系统移植步步通

12. 嵌入式 CC++语言精华文章集锦
13. 基于 Linux 的高性能服务器端的设计与研究
14. S3C6410 移植 Android 内核
15. Android 开发指南中文版
16. 图解 Linux 操作系统架构设计与实现原理（第二版）
17. 如何在 Ubuntu 和 Linux Mint 下轻松升级 Linux 内核
18. Android 简单 mp3 播放器源码
19. 嵌入式 Linux 系统实时性的研究
20. Android 嵌入式系统架构及内核浅析
21. 基于嵌入式 Linux 操作系统内核实时性的改进方法研究
22. Linux TCP IP 协议详解
23. Linux 桌面环境下内存去重技术的研究与实现
24. 掌握 Android 7.0 新增特性 Quick Settings

Windows CE：

1. Windows CE.NET 下 YAFFS 文件系统 NAND Flash 驱动程序设计
2. Windows CE 的 CAN 总线驱动程序设计
3. 基于 Windows CE.NET 的 ADC 驱动程序实现与应用的研究
4. 基于 Windows CE.NET 平台的串行通信实现
5. 基于 Windows CE.NET 下的 GPRS 模块的研究与开发
6. win2k 下 NTFS 分区用 ntldr 加载进 dos 源代码
7. Windows 下的 USB 设备驱动程序开发
8. WinCE 的大容量程控数据传输解决方案设计
9. WinCE6.0 安装开发详解
10. DOS 下仿 Windows 的自带计算器程序 C 源码
11. G726 局域网语音通话程序和源代码
12. WinCE 主板加载第三方驱动程序的方法
13. WinCE 下的注册表编辑程序和源代码
14. WinCE 串口通信源代码
15. WINCE 的 SD 卡程序[可实现读写的源码]
16. 基于 WinCE 的 BootLoader 研究

PowerPC：

1. Freescale MPC8536 开发板原理图

2. 基于 MPC8548E 的固件设计
3. 基于 MPC8548E 的嵌入式数据处理系统设计
4. 基于 PowerPC 嵌入式网络通信平台的实现
5. PowerPC 在车辆显控系统中的应用
6. 基于 PowerPC 的单板计算机的设计
7. 用 PowerPC860 实现 FPGA 配置
8. 基于 MPC8247 嵌入式电力交换系统的设计与实现
9. 基于设备树的 MPC8247 嵌入式 Linux 系统开发
10. 基于 MPC8313E 嵌入式系统 UBoot 的移植
11. 基于 PowerPC 处理器 SMP 系统的 UBoot 移植

ARM：

1. 基于 DiskOnChip 2000 的驱动程序设计及应用
2. 基于 ARM 体系的 PC-104 总线设计
3. 基于 ARM 的嵌入式系统中断处理机制研究
4. 设计 ARM 的中断处理
5. 基于 ARM 的数据采集系统并行总线的驱动设计
6. S3C2410 下的 TFT LCD 驱动源码
7. STM32 SD 卡移植 FATFS 文件系统源码
8. STM32 ADC 多通道源码
9. ARM Linux 在 EP7312 上的移植
10. ARM 经典 300 问
11. 基于 S5PV210 的频谱监测设备嵌入式系统设计与实现
12. Uboot 中 start.S 源码的指令级的详尽解析
13. 基于 ARM9 的嵌入式 Zigbee 网关设计与实现
14. 基于 S3C6410 处理器的嵌入式 Linux 系统移植
15. CortexA8 平台的μC-OSⅡ及 LwIP 协议栈的移植与实现

Hardware：

1. DSP 电源的典型设计
2. 高频脉冲电源设计
3. 电源的综合保护设计
4. 任意波形电源的设计
5. 高速 PCB 信号完整性分析及应用

6. DM642 高速图像采集系统的电磁干扰设计
7. 使用 COMExpress Nano 工控板实现 IP 调度设备
8. 基于 COM Express 架构的数据记录仪的设计与实现
9. 基于 COM Express 的信号系统逻辑运算单元设计
10. 基于 COM Express 的回波预处理模块设计
11. 基于 X86 平台的简单多任务内核的分析与实现
12. 基于 UEFI Shell 的 PreOS Application 的开发与研究
13. 基于 UEFI 固件的恶意代码防范技术研究