第 35 卷 第 1 期 计 算 机 工 程 2009 年 1 月
Vol.35 No.1 Computer Engineering January 2009
·博士论文· 文章编号：1000—3428(2009)01—0030—03 文献标识码：A 中图分类号：TP393

一种面向存储服务的缓存管理模型

孟晓烜[1,2]，李一鸣[1,2]，卜庆忠[1]，纪海涛[1]，许 鲁[1]
(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)

摘 要：给出一种适用于网络存储设备的面向存储服务的新型缓存管理模型，它支持可配置的缓存管理策略和可配置策略的缓存分配机制。与通用操作系统中的缓存管理模块相比，该模型的主要优点是使能针对具体存储应用的性能优化，使能多个存储应用之间的数据访问 QoS 控制。

关键词：网络存储设备；缓存管理；缓存分配

Storage Service-oriented Buffer Management Model

MENG Xiao-xuan[1,2], LI Yi-ming[1,2], BU Qing-zhong[1], JI Hai-tao[1], XU Lu[1]
(1. Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080;
2. Graduate University of Chinese Academy of Sciences, Beijing 100039)

【Abstract】 This paper proposes a novel buffer management model for network storage device(Service-oriented Buffer Management model, SBM). It supports not only configurable buffer management policy but also configurable policy-based buffer allocation mechanism. Compared with buffer management module in general-purpose OS, SBM enables I/O performance optimization targeted for specific storage application, and facilitates realizing storage QoS across multiple storage applications.

【Key words】 network storage device; buffer management; buffer allocation

1 概述

网络存储设备是构建存储网络的基本元素之一[1]，作为网络存储设备的一种 I/O 性能优化手段，缓存技术用于降低低速磁盘访问延迟对存储应用 I/O 性能的影响。网络存储设备中的缓存位于 I/O 路径中的第 2 级，又称为二级缓存[2]，各类存储应用原本的 I/O 模式特征在一定程度上受到存储应用服务器缓存的过滤影响，此外，网络存储设备通常需要同时支持多种存储应用，这些具有不同 I/O 性能需求、I/O 模式特征的存储应用不仅共享静态物理存储资源，而且共享动态缓存资源。

近年来，随着存储网络规模的不断扩大和通用处理器运算能力的不断增强，基于服务器架构的开放式存储设备由于其价格上的优势正在逐步取代专用存储设备。开放式网络存储设备依靠通用操作系统中的内存管理模块管理读写缓存。以 Linux 操作系统[3]为例，它采用基于类似 2Q 替换算法[2]的内存管理机制统一管理进程虚拟内存和 VFS 缓存，这种集中式管理方式显然不能很好地满足网络存储设备的缓存管理需求：(1)单一固化的缓存管理策略不能根据具体存储应用的 I/O 模式特征进行有针对性的优化[4]；(2)缓存管理策略本身是针对一级缓存特点而设计的，因此，不能高效地应用于二级缓存[2]；(3)随着网络存储服务概念的提出，提供相应的存储 QoS 保证是存储设备所必备的功能之一[5]，通用操作系统中的内存管理机制尚不能提供类似区分服务这样的存储 QoS 保证。

基于上述考虑，本文提出一种新的面向存储服务的缓存管理模型(Service-oriented Buffer Management model, SBM)以解决上述问题：(1)支持可配置的缓存管理策略，系统管理员可根据每个存储应用特有的 I/O 模式特征选择最佳缓存管理策略；(2)支持基于可配置策略的缓存分配机制，多个存储应用之间在缓存分配策略约束下，综合 I/O 性能需求、I/O 活跃度、I/O 模式特征等因素有序地竞争有限的缓存资源，从而为实现存储 QoS 保证提供有效手段。为验证 SBM 模型在实际系统中的可行性和效率，本文在 Linux-2.6.12 内核下以虚拟块设备驱动方式[2]实现了该模型，如图 1 所示，除了网络存储目标端软件[1]需要通过 I/O 接口[3]直接访问 SBM 缓存，Linux 操作系统和网络存储系统软件不需要做任何修改。实验结果表明，SBM 模型在实际系统中的处理开销可忽略不计，此外，它采用的 I/O 处理机制明显提高了磁盘 I/O 效率。

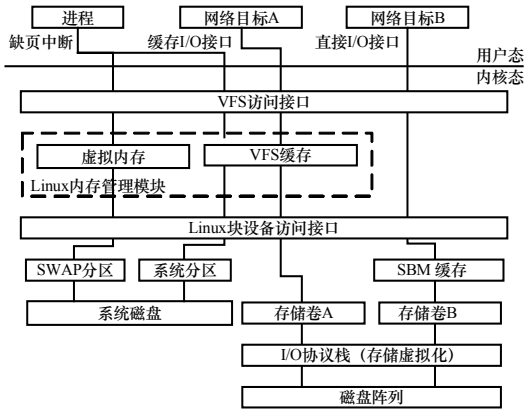

图 1 SBM 模型在 Linux 平台下的应用

基金项目：国家“973”计划基金资助项目(2004CB318205)
作者简介：孟晓烜(1980－)，男，博士研究生，主研方向：大型网络存储系统；李一鸣，硕士研究生；卜庆忠，助理研究员、博士后；纪海涛，助理研究员、硕士；许 鲁，研究员、博士生导师
收稿日期：2008-06-16 **E-mail：**mengxx@ict.ac.cn

2 SBM 缓存管理模型

SBM 缓存模型将传统单一集成的缓存管理机制抽象划分为逻辑上相互关联而实现上相互独立的 3 类功能实体：缓存对象，缓存管理对象和缓存分配对象(图 2)，其中缓存对象与存储应用相对应；根据应用的具体需求，缓存对象与特定的缓存管理对象相关联以获得最佳缓存管理功能；所有的缓存对象都通过唯一的缓存分配对象竞争缓存资源。

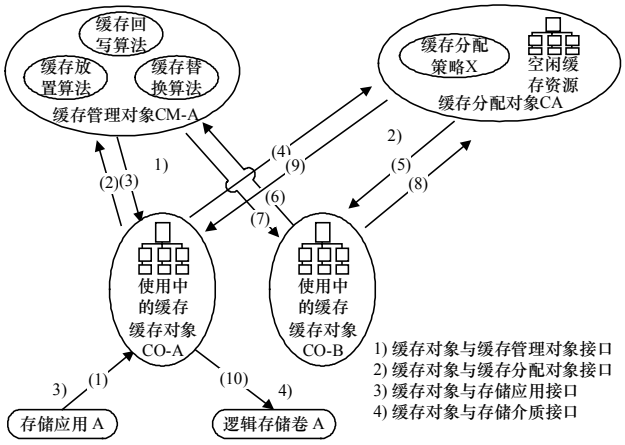

图 2 SBM 模型示意及其工作原理

2.1 缓存对象

缓存对象将存储应用和存储介质相关联，它在 I/O 路径上为存储应用提供有针对性优化的缓存管理功能。如图 2 所示，缓存对象定义了 4 类访问接口：1)与存储应用间接口，接收来自存储应用的 I/O 请求；2)与存储介质间接口，向下层物理存储介质发起 I/O 请求；3)与缓存管理对象间接口，实现缓存管理功能；4)与缓存分配对象间接口，获取缓存资源。缓存对象是整个模型的核心枢纽，不仅规范了 SBM 模型的外部 I/O 接口，同时通过内部缓存接口将模型中的各功能实体有机高效地关联在一起。

2.2 缓存管理对象

缓存管理对象为缓存对象提供基于某种缓存管理策略的缓存管理功能，利用 I/O 模式中的时间/空间局部性以提高缓存利用率[4]。在给定缓存资源的前提下，SBM 模型将缓存管理策略按功能进一步细分为以下 3 部分：(1)放置操作。决定了何时从存储介质中读取哪些数据块并置于缓存中。(2)回写操作。决定何时将缓存中哪些修改过的数据块同步至存储介质中。(3)回收操作。决定何时从缓存中替换出哪些缓存块。

SBM 模型基于的一个重要思想是：在网络存储应用模式下，存储应用应能根据 I/O 模式特征、存储介质属性[4]、网络承载能力、级间缓存容量比例和性能差异[2]等因素综合选择最佳缓存管理策略以最大化缓存使用效率。具体来说，缓存对象根据应用的具体需求配置最合适的缓存管理对象，后者为前者提供所需的缓存管理功能，且同一缓存管理对象可以同时服务于多个缓存对象。

2.3 缓存分配对象

缓存分配对象统一管理缓存资源，其基本原理为：缓存分配对象发起空闲缓存块回收操作，但它只决定何时从哪些缓存对象中回收多少缓存块；而与缓存对象关联的缓存管理对象则具体决定回收哪些缓存块。具体来说，一方面缓存对象在 I/O 处理过程中按需向缓存分配对象申请缓存资源，另一方面缓存分配对象根据系统当前 I/O 负载以静态或动态的方式[4-5]调整缓存资源的分配，从而为实现存储 QoS 控制提供必要手段。为支持多种存储 QoS 语义以适用于不同的应用模式，缓存分配对象中的分配策略是可配置的，不同的分配策略实现不同的存储 QoS 语义。以动态缓存分配方式为例，缓存对象可以通过某种期望缓存增益函数[4-5]来竞争缓存资源，原则上缓存增益越高，缓存对象获得的缓存资源也越多。

2.4 模型的工作原理

下面通过具体实例阐述 SBM 模型的工作原理：如图 2 所示，假定缓存对象 CO-A 和 CO-B 都关联于缓存管理对象 CM-A，(1)存储应用访问数据块 c；(2~3)由于 c 当前不在缓存中，CO-A 询问 CM-A 对应缓存放置方案，如是否需要预取从 c 地址开始的一组连续的数据块等；(4)CO-A 向缓存分配对象 CA 申请所需的缓存资源；(5)假设此时空闲缓存资源不足，CA 暂时挂起该申请操作并立即触发缓存回收操作：缓存分配策略 X 决定从 CA-B 中回收 *N* 块缓存；(6~8)CM-A 则相应从 CA-B 使用的缓存中选出它认为最没有价值的 *N* 块，并交还给 CA；(9)后者随即满足先前被阻塞的缓存申请；(10)CO-A 向逻辑存储卷发起磁盘 I/O。

3 系统实现

为了验证SBM模型在实际系统中的可行性，本文在Linux-2.6.12内核下以虚拟块设备驱动方式实现了该模型，系统由缓存设备驱动(Cache Block Device Driver, CBD)、系统缓存管理模块(System Cache Manager, SCM)、设备缓存管理模块(Device Cache Manager, DCM)及用户管理工具等4部分组成，其C代码实现量分别为5 510行，900行，725行和1 110行，总计为 8 250行，其中SCM模块支持可配置的缓存分配策略，它对应于SBM模型中的缓存分配对象；DCM模块则各自基于不同的缓存管理策略，它们对应于SBM模型中的缓存管理对象。

系统实现原理如下：(1)SBM 模型中的缓存对象实现为虚拟缓存块设备，如图 1 所示，它在 I/O 协议栈中位于逻辑存储卷之上，存储应用通过直接 I/O 绕过 VFS 缓存直接访问缓存设备；(2)缓存设备在创建时配置最佳缓存管理策略，基于该策略的 DCM 模块为缓存设备提供缓存管理功能；(3)缓存设备在运行中通过 SCM 模块获取缓存资源。

此外，CBD 驱动为系统提供了实际运行环境，包括与主机系统的交互、缓存设备的管理、模块间的通信机制及通用缓存处理机制等，用户管理工具则通过系统 ioctl 调用实现对 CBD 驱动的配置管理。目前已实现了缓存分配和缓存管理 2 种简单的策略：

(1)静态分区缓存分配策略。缓存设备在创建时分配一块专用缓存空间。在系统运行过程中，缓存设备间的缓存资源是非共享的，系统管理员可按需通过用户管理工具在线动态调整各设备缓存空间的大小。

(2)LRU 缓存管理策略。采用 LRU, H/L*Mark[3]和 AA[3]算法分别实现缓存回收、回写和放置功能，这 3 种算法的优点是简单、高效且易于实现。

下面通过分析缓存设备的 I/O 路径以阐述 SBM 模型在 Linux 下的 I/O 处理机制。

(1)存储应用(网络存储目标端软件)首先以设备文件方式打开缓存设备，且读写模式设为直接 I/O 方式，随后可通过 read/write 向内核发起请求。

(2)内核在接收到应用层的读/写请求后，通过 vfs_read /vfs_write 向 VFS 层转发。

(3)VFS层依次调用设备文件操作(def_blk_fops{})中的读

/写方法和设备文件映射操作(def_blk_aops{})中的直接 I/O 方法进行处理，在生成对应的 bio 结构后，通过 submit_bio 向块设备层转发。

(4)块设备层对收到的 bio 请求进行完备性校验，确认请求合法后，调用设备请求队列的 make_request_fn 方法将请求传给 CBD 驱动。

(5)make_request_fn 方法在缓存设备创建时被 CBD 驱动初始化为 cbd_make_request 函数，它是整个驱动模块的 I/O 入口，其工作流程如图 3 所示：

1)获取缓存设备的数据结构、bio 请求的地址范围和类型 1~4。

2)检查缓存设备当前是否可读写 7~9。

3)判断 bio 请求的读写类型，若为写则转入 5)执行 12。

4)调用缓存设备读操作处理 bio 请求，完成后转入 8)执行 13。

5)调用缓存设备写操作处理 bio 请求 15。

6)判断缓存设备是否需要写穿，若不需要则转入 8)执行 17。

7)调用缓存设备同步操作将 5)中修改过的数据块同步至物理存储设备中 18。

8)调用 bio 回调函数(bio->bi_end_io)通知 VFS 层读写请求已处理完毕 22。

9)退出并返回 0 以终止块设备层对 bio 请求的处理 26。

```
cbd_make_request()函数实现
* int cbd_make_request(request_queue_t *q, struct bio *bio)
0 {
1     cbd_device_t *dev = q->queuedata;
2     sector_t start = bio->bi_sector;
3     sector_t end = bio->bi_sector + bio->bi_size>>9 -1;
4     int sync = bio_sync(bio), ret = 0;
5
6     atomic_inc(&dev->dio_cnt);
7     if (unlikely(!test_bit(DEV_IO, &dev->flags))) {
8         ret = -EINVAL;
9         goto out;
10    }
11
12    if (bio_data_dir(bio) == READ)
13        ret = dev->d_ops->dev_read(dev, bio);
14    else {
15        ret = dev->d_ops->dev_write(dev, bio);
16            if (likely(!ret))
17                if (test_bit(DEV_SYNC, &dev->flags) || sync)
18                    ret = dev->d_ops->dev_sync(dev, start, end, 0);
19    }
20
21 out:
22    bio_endio(bio, bio->bi_size, ret);
23    if (unlikely(ret))
24        sbm_dev_iostat_inc(dev, bio_error);
25    atomic_dec(&dev->dio_cnt);
26    return 0;
27 }
```

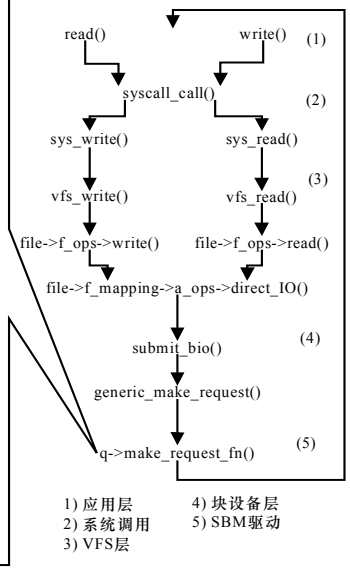

图 3　SBM 模型在 Linux 下的 I/O 处理机制示意图

4　性能评测

本节给出 SBM 和 VFS 的性能对比测试，其目的在于检验 SBM 模型在 Linux 下的系统开销和代码实现效率。为此本文对比 SBM 和 VFS 在顺序读写模式下的读写吞吐率峰值，其中，顺序读写模式避免了缓存替换算法对 I/O 性能的影响。此外，两者采用相似的预读(预读的最大粒度均设为 1 MB)和回写机制。

实验环境配置为：Gentoo-2.6.18 Kernel, Intel Xeon 3 GHz CPU, 1 GB DDR Memory, 3ware-9500RAID Card, RAID0: 7×WD2500SD(ESATA, 250 GB, 7 200 RPM), LVM2。测试中逻辑存储卷的大小均为 100 GB，测试工具为 dd，其-dio 选项支持 POSIX 标准的直接 I/O 接口。

SBM 与 VFS 性能对比如图 4 所示。

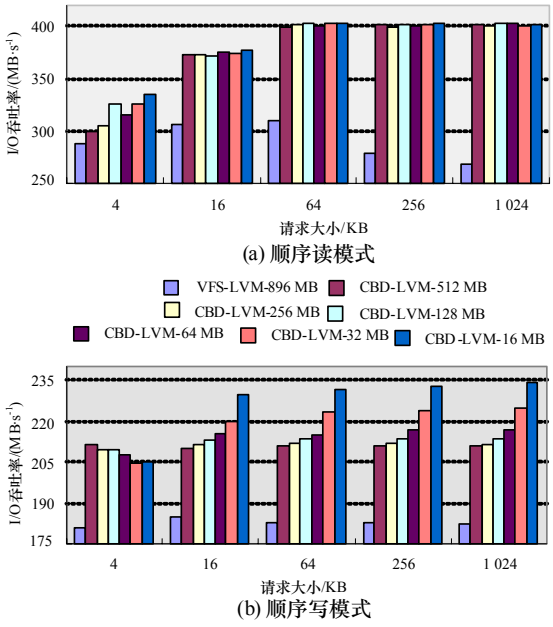

图 4　SBM 与 VFS 性能对比

测试结果显示，在各种读写粒度下，SBM 缓存 I/O 吞吐率都明显高于 VFS 缓存，其中，读最高可提升 50.05%，写提升 28.22%，这说明：(1)SBM 模型在 Linux 下实现所带来的系统开销可忽略不计；(2)SBM 模型采用的 I/O 处理机制显著提高了磁盘 I/O 效率。下一步拟通过采集磁盘阵列的读写记录进一步分析提取出 I/O 处理机制中影响磁盘 I/O 效率的关键因素，从而更好地解释了 SBM 在上述测试中表现出良好性能的原因。

5　结束语

本文描述了一种适用于网络存储设备的新型缓存管理模型，它能够支持针对单个存储应用的 I/O 性能优化，支持多个存储应用间的多种存储 QoS 语义保证。本文在 Linux-2.6.18 内核下验证了该模型在实际系统中的可行性，下一步的主要工作是：(1)研究各类典型存储应用在网络存储设备中的 I/O 模式特征，并相应设计出有针对性的高效缓存管理策略；(2)研究各种动态缓存分配策略以满足不同存储应用模式对存储 QoS 的需求。

参考文献

[1] Gibson G A, Meter R V. Attached Storage Architecture[J]. Communications of the ACM, 2000, 43(11): 37.

[2] Zhou Yuanyuan, Chen Zhifeng, Li Kai. Second-level Buffer Cache Management[J]. IEEE Transactions on Parallel and Distributed Systems, 2004, 15(6): 505.

[3] Bovet D P, Cesati M. Understanding Linux Kernel[M]. [S. l.]: O'Reilly Press, 2005.

[4] Chakraborty L, Singh A. A Utility-based Approach to Cost-aware Caching in Heterogeneous Storage Systems[C]//Proc. of the 21st International Parallel and Distributed Processing Symposium. Long Beach, CA, USA: [s. n.], 2007.

[5] Goyal P, Jadav D, Modha D, et al. CacheCOW: QoS for Storage System Caches[C]//Proc. of the 11th International Workshop on Quality of Service. Monterey, CA, USA: [s. n.], 2003.