一种针对存储系统设计的应用级缓存回收策略

孟晓烜[1,2],司成祥[1,2],那文武[1,2],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)

E-mail: mengxx@ict.ac.cn

摘 要:针对存储系统中的缓存管理单元设计一种区分应用优先级的缓存回收策略,简称 PARP.该策略基于分区缓存管理模型,它能够根据应用优先级区分回收缓存资源以实现对各应用缓存分区容量的在线动态调节进而达成应用级缓存管理语义.实验数据表明 PARP策略能够在实际系统中有效的支持区分应用优先级,这不仅可以用于实现存储系统的服务质量保证同时也能够改善存储系统的整体性能.

关 键 词:应用优先级;缓存回收策略;分区缓存管理模型;存储系统

中图分类号: TP333 **文献标识码**: A **文 章 编 号**: 1000-1220(2010)03-0456-04

Application-level Buffer Reclaim Policy Designed for Storage System

MENG Xiao-xuan[1,2], SI Cheng-xiang[2], NA Wen-wu[1,2], XU Lu[2]

[1](Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080, China)

[2](Graduate School of the Chinese Academy of Sciences, Beijing 100039, China)

Abstract: Presented a priority-based application-level buffer reclaim policy-PARP designed for the buffer management unit in storage system. Based on partitioned buffer management model, it can differentially reclaim buffer resource from multiple applications according to their priorities. This realizes the dynamically adjustment of the application's buffer capacity to provide application-level buffer management semantics. Experiment results show that PARP effectively supports application priority in real system, which can not only be utilized to achieve storage QoS but can also be used to optimize the overall storage performance.

Key words: application priority; buffer reclaim policy; partitioned buffer management model; storage system

1 引 言

在网络存储[1]应用模式下,多种不同类型的应用共享存储资源,这些应用通常具有不同的优先级和负载特征,而这里的共享资源[2]主要包括处理器、网卡带宽、磁盘机械臂和缓存空间等四种,其中缓存是决定存储系统性能的重要因素,这就要求存储系统中的缓存管理单元应能够提供应用级的缓存区分服务[2]以适应网络存储系统中多应用共享存储的应用模式特点.目前主流操作系统中的缓存管理单元不能支持应用级的缓存区分服务,以 Linux操作系统中的缓存管理单元(Linux-MM)[3]为例,它采用全局缓存替换算法[4]来集中统一管理系统中的物理内存,其主要包括虚拟内存和 I/O缓存两类,两者所拥有缓存资源比例是根据当前系统负载变化而动态调节的.尽管 Linux-MM 在一定程度上可以做到区分虚拟内存和 I/O缓存,但它显然无法实现更细粒度的应用级缓存管理语义.

基于上述考虑,本文在分区缓存管理模型[5]基础上提出一种区分应用优先级的缓存回收策略-PARP(priority-based application-level buffer reclaim policy)以满足网络存储系统中多应用共享模式对缓存管理的需求.这里分区缓存管理模型与传统基于全局缓存替换算法的缓存管理模型所不同之处在于:它对系统缓存资源进行分区管理,每个应用具有自己相互独立的缓存分区.在系统在运行过程中,缓存回收策略定时从各应用分区中回收一定数目的缓存资源以保证系统在任意时刻始终保持一定数量的空闲缓存资源以满足应用的同步 I/O需求.本文提出的 PARP策略的基本思想如下:根据应用的重要性、I/O性能需求或负载特征的不同可为应用配置不同的优先级,策略在系统运行过程综合应用优先级、当前系统负载和缓存资源分布等三种因素,通过对缓存资源的区分回收以实现各缓存分区容量的在线动态调节.在给定应用负载特征和缓存替换策略的前提下,缓存容量决定了应用的 I/O性能.PARP策略的基本原则是优先级高的应用所获得缓存资源就越多反之亦然,其优点在于:一方面,支持应用优先级就能够在多应用共享存储系统中保证高优先级应用的 I/O性能从而实现存储服务语义;而另一方面,由于负载特征的差异而导致不同应用具有不同的缓存效用[5],为缓存效用高的应用赋予高优先级可以提高存储系统的整体性能.

2 背景介绍

在介绍 PARP策略之前,本节首先介绍该策略所基于的动态分区缓存管理模型[4],该模型由缓存分区、缓存管理模

收稿日期:2008-11-07 收修改稿日期:2009-02-05 基金项目:国家"九七三"重点基础研究发展规划基金项目(2004CB318205)资助. 作者简介:孟晓烜,男,1980年生,博士研究生,研究方向为网络存储,缓存管理;司成祥,男,1982年生,博士研究生,研究方向为分布式文件系统;那文武,男,1977年生,博士研究生,研究方向为网络存储,RAID技术.

块和缓存分配模块等三者构成,其中缓存分区承载应用当前的活跃访存数据;缓存管理模块为缓存分区提供基于特定缓存替换策略的缓存空间管理服务;而缓存分配模块则负责管理系统中的空闲缓存资源,一方面它为各应用的缓存分区提供按需缓存资源分配;另一方面它定期从各缓存分区中回收空闲缓存资源以保证任意时刻系统始终具有少量可用缓存资源以避免等待缓存资源回收而带来的同步读写阻塞[3],这其中如何进行资源回收则是由缓存回收策略所决定. 模型的基本工作原理如图 1所示:缓存分配模块以共享缓存池的方式统一管理系统中的空闲缓存资源. 当系统处于初始状态时,共享缓存池中存放了系统中所有的缓存资源;随着系统的运行,各应用按需从共享缓存池中分配缓存资源;当共享缓存池中空闲缓存块数第一次不足低临界阈值后,缓存分配模块相应

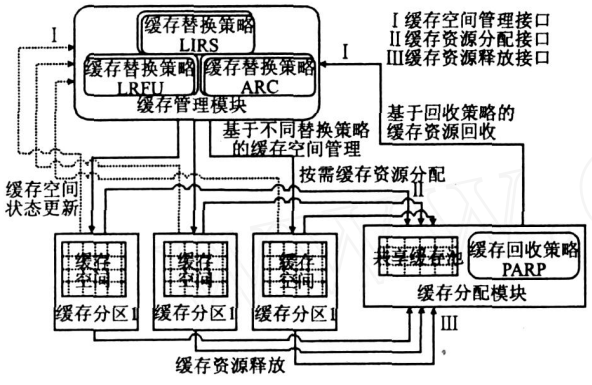

图 1　分区缓存管理模型的原理示意图

Fig. 1　Principles of partitioned buffer management model

发起了第一轮的缓存回收操作,至此系统进入了饱和状态. 在系统进入饱和状态后,周期性的缓存资源回收使得缓存池中的空闲缓存块数始终保持在高/低临界值之间. 如前所述,缓存回收策略决定了缓存回收方案,它在按需分配的基础上通过基于某种缓存区分服务语义的区分缓存资源回收来控制各分区缓存分区容量的在线动态调节,只有那些分配的缓存块数超过其被回收数的缓存分区才能得以扩张,反之亦然. 显然在给定缓存替换策略的条件下,缓存分区的容量决定了应用的 I/O性能,因此该模型所采用的这种按需分配和区分回收相结合的缓存分配机制可以有效的实现应用级的缓存管理语义,且后者取决于所配置的缓存回收策略. 本文所提出的 PARP策略实现了一种区分应用优先级的缓存管理语义,该策略的优点在于它不仅可以有助于实现存储系统的服务质量保证;同时又可以用于提高存储系统的整体性能. 下文将详细阐述 PARP策略的工作原理和算法流程.

3　PARP策略

PARP策略的基本原理如下:该策略为每个缓存分区定义一个静态优先级(prio),其取值范围为[1, 8],系统管理员可以根据应用的重要性或缓存效用等因素相应设置应用优先级. PARP策略在系统资源回收时,根据每个缓存分区的优先级、当前缓存资源占有率以及最近负载强度等三种因素相应在线计算该缓存分区的资源回收友好值(nice),并根据上述动态计算得到的 nice值按比例从各缓存分区中回收一定数量的空闲缓存资源,其中当前 nice值越高的缓存分区所回收的缓存资源就越多. 原则上,优先级越高的应用在系统运行过中所拥有的缓存资源就越多,反之亦然.

PARP策略算法

```
1. parp_rebalance_spaces()
2. {
3.    tart_blks = BLK_HIGH - curblks;
4.
5.    for each space_i in parp do
6.       set_parp_nice(space_i);
7.       if(space_i. nice)
8.          tot_nice + = space_i. nice;
9.          add space_i to rcl_list
10.      fi
11. end
12.
13.    do
14.       _blks = tgrt_blks, _nice = tot_nice;
15.       all_is_ok = 1;
16.       for each space_i in rcl_list do
17.          if_blks * space_i. nice > spae_i. curblks * _nice)
18.             tot_nice - = space_i. nice;
19.             tgrt_blks - = space_i. curblks;
20.             set_space_to_recalim (space_i, space_i. curblks);
21.             remove space_i from rcl_list
22.             all_is_ok = 0;
23.          fi
24.       end
25.    while(all_is_ok)
26.
27.    for each space_i in rcl_list do
28.       rcl_blks = space_i. nice * tgrt_blks/tot_nice;
29.       set_space_to_recalim (space_i, rcl_blks);
30. end
31. }
32.
33. parp_set_nice()
34. input: space
35. {
36.    alloc_rto = space. curblks - space. preblks/BLK_HIGH;
37.    space. alloc_rto = space. alloc_rto * age + * 1 - age) * allocrto;
38.    use_rto = space. curblks/BLK_TOT;
39.
40.    space. nice = (p * space. alloc_rto + (1 - p) * use_ratio)/space.
prio;
41. }
```

图 2　PARP缓存回收策略的算法流程图

Fig. 2　Diagram of PARP buffer reclaim policy

图 2给出了 PARP策略的算法流程图,它依次分为以下 4个步骤:

步骤 1　(3)计算当前所需回收的空闲缓存资源总量(tgrt_blks),其中 BLK_HIGH为系统空闲缓存资源的高临界值(见 2节),而 curblks为当前系统剩余空闲缓存资源,二者差值即为本次回收操作所需回收缓存资源数;

步骤 2　(5-11)更新当前 PARP策略管理域内每个缓存分区的回收 nice值,将 nice值非 0的缓存分区加入资源回收链表(rlc_list)并统计回收链表中缓存分区的 nice值得累加和

(tot_nice),其中 nice值的更新是由 parp_set_nice()函数完成(见下文);

步骤 3 (13-25)从回收链表中找出那些可回收资源不足的缓存分区,其判断依据为分区中的缓存容量是否小于其按 nice值比例回收数 $-\ blks * \frac{space_i.\ nice}{\_nice}$ (17),对于那些资源不足的缓存分区,标记回收其当前所有的缓存资源(20).并相应更新 tgrt_blks和 tot_nice值,最后将该缓存分区从回收链表中删除,这其中 set_space_to_reclaim()函数标记该缓存分区应回收多少缓存资源;

步骤 4 (27-30)对于每一个在回收链表中的缓存分区,按比例标记所需回收的缓存资源数.

parp_set_nice()函数完成对单个缓存分区的状态更新,它首先计算缓存分区的当前负载强度 alloc_rto(36),即当前缓存资源分配率;接着根据缓存分区的历史负载强度(space. alloc_rto)利用自回归滑动平均法得出缓存分区最近负载强度(37),其中 age[0, 1]为历史老化因子;再者计算缓存分区当前资源占有率(use_rto),其中 BLK_TOT为系统缓存资源总量;最后求出当前资源占有率和最近负载强度的加权平均值并将该值除以缓存分区的静态优先级得到当前缓存分区的 nice值,其中 p[0, 1]为负载强度的权值,这里需要指出的是 age和 p两参数主要用于控制 PARP策略算法敏感度,优化的参数配置可以使得 PARP策略即能够快速的响应应用负载的相对变化同时又能够避免负载短时变化噪音[6]所带来分区缓存容量抖动.

4 策略实现

我们在 DPCache缓存管理系统[4]平台下实现了 PARP策略,DPCache系统提供了动态分区缓存管理机制,它采用虚拟块设备驱动技术实现于 Linux内核中.在该系统中,每个应用对于一个缓存虚拟块设备,后者在应用与物理存储介质之间的 I/O路径上提供缓存管理服务.DPCache系统采用模块化的设计方法,系统由缓存驱动管理框架、缓存管理模块和缓存分配模块等三者构成,其中缓存分配模块需配置支持特定缓存管理语义的缓存回收策略并通过下述三种标准接口方法与所配置的缓存回收策略进行交互:

attach_space(): 将应用缓存分区纳入策略管理域内;

detach_space(): 将应用缓存分区从策略管理域中移出;

rebalance_spaces(): 计算系统缓存资源回收方案,即对策略管理域内各应用缓存分区容量进行调整;

set_space(): 配置缓存分区的控制参数;

PARP策略相应实现了上述四种方法,其中 rebalance_spaces()方法实现了 PARP策略的核心处理逻辑,set_space()方法用于配置/修改缓存分区的优先级.经过大量实验我们发现,当参数 age和 p(见第 3节)分别设为 0.8和 0.7时系统行为表现最为平稳,因此在下一节的各组性能测试案例中,PARP策略均采用上述参数配置组合.

5 性能评测

本节我们在 DPCache系统平台下对 PARP策略进行性能评测.测试中使用的存储服务器配置参见表 1,使用的测试工具为 xdd和 kernio,其中后者是我们自主开发的一种运行于内核空间的块级 I/O负载回放工具.此外测试使用 oltp和 websearch两种符合 SPC-1标准[7]的 I/O负载,两者的测试时间均为 1小时左右,其中 oltp负载的时间局部性强、数据访问量小;而 websearch负载正好与之相反.此外测试中应用缓存分区所使用的替换算法均为 lru.

表 1 实验环境配置

Table1 Experiment settings

网络存储设备	Gentoo-2.6.18 Kernel Intel(R) Xeon(TM) 1.60GHz CPU 2GB DDR Memory 3ware-9500RAID Card 2 * RAID0: 2 ×WD2500SD (ESATA, 250GB, 7200RPM)

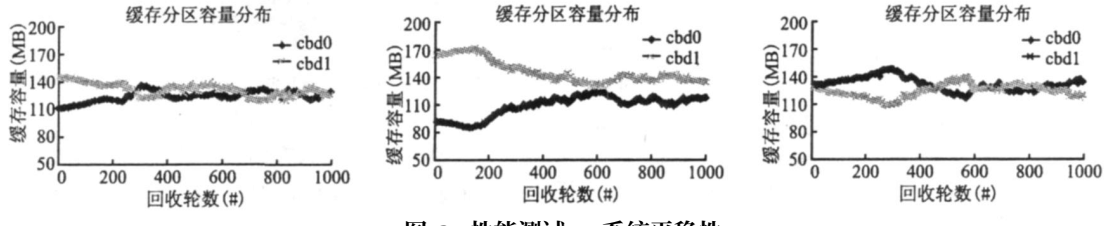

图 3 性能测试一:系统平稳性

Fig. 3 Performance test case1: system stability

第一组测试用以检验 PARP策略的算法稳定性(见第 3节),为此我们利用 xdd工具分别同时顺序读/顺序写 cbd0/cbd1两块虚拟缓存磁盘以观测二者在极端突发性(bursty) I/O负载模式下的缓存分区容量变化分布.测试中的读写粒读均为 1MB,且系统缓存总容量设为 256MB,实验结果如图 3(a-b)所示,显然系统在上述两种极端突发性 I/O负载下表现非常平稳,各缓存分区容量始终相对稳定没有因为负载强度的交替变化而产生抖动.

在第二组测试中我们同时向 cbd0/cbd1回放 oltp负载,通过对比 cbd0/cbd1在不同优先级组合下的性能差异以检验 PARP策略能否实现基于优先级的缓存区分服务语义.考虑到 oltp负载的数据访问集仅为 275MB,因此测试中系统缓存总容量设为 256MB.实验结果如图 4(a)所示,由于 cbd1的优先级始终设为 1,因此随着 cbd0优先级的增大,cbd0的性能

相应提高而 cbd1则正好相反. 图 4(b)给出了测试中 cbd0/cbd1缓存分区大小的动态分布,从图中可以看出随着 cbd0相对优先级的增大,其缓存分区容量在系统达到稳态后也越多. 正是由于缓存容量上的差异导致了 cbd0/cbd1在性能上的差异. 本测试说明了 PARP策略能够在多应用共享存储系统中有效保证高优先级应用的性能.

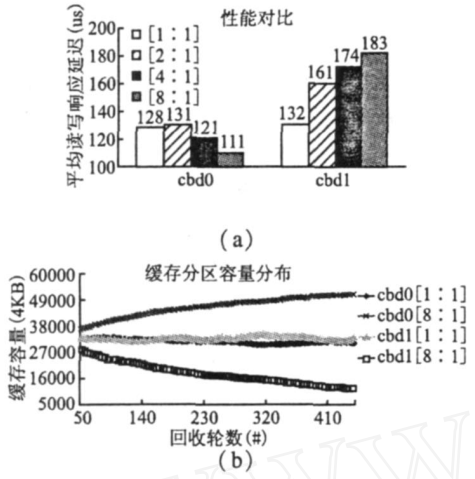

图 4　性能测试二:区分服务

Fig. 4　Performance test case2: differential service

在第三组测试中我们同时向 cbd0/cbd1分别回放 oltp和 websearch两种不同类型的负载,通过配置不同的优先级组合以检验 PARP策略能否实现基于优先级的缓存效用优先语义. 相比于 oltp, websearch负载的数据访问局部性弱,且其数据访问量和数据访问集均更高,因此后者的缓存效用明显低于前者. 基于此我们尝试为 oltp负载配置更高的优先级以观察随之带来的性能差异. 该组测试以 Linux-MM为性能评测基准,系统缓存总容量设为 1GB. 图 5(a)给出了在两种优先级组合下 DPCache相比于 Linux-MM对 oltp和 websearch两种单负载以及系统整体性能的提升,其中系统整体性能提升是依据 I/O访问频度对两种单负载性能提升的加权平均. 从图中可以看出,当优先级组合为 8/1时 DPCache系统整体性能比 Linux-MM提升了近 30%. 图 5(b)给出了 oltp负载在两种优先级组合下的缓存容量分布,当优先级设为 8时, oltp负载的容量显著高于其优先级为 1时. 一方面,由于 oltp负载的缓存效用高,因此容量上的增长直接转换为性能上的提升, oltp单负载性能提升了约 70%;而另一方面,尽管 oltp负载容量上的增长必然随之带来 websearch负载容量的相应下降,但由于后者的缓存效用低,因此其性能几乎没有受到影响. 由此可知,如果以缓存效用为原则合理的配置应用优先级, PARP策略能够有效改善存储系统的整体性能.

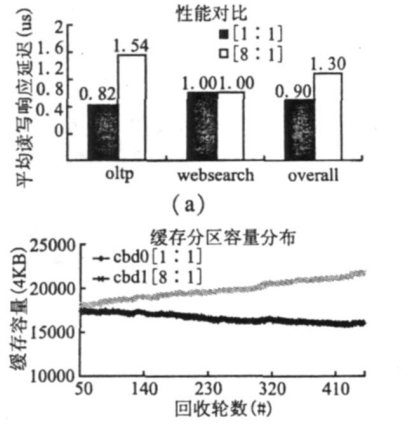

图 5　性能测试三:缓存效用

Fig. 5　Performance test case3: buffer utility

6　总　结

本文介绍了一种针对存储系统设计的区分应用优先级的缓存回收策略 -PARP,它在动态分区缓存管理机制的基础上能够根据应用的优先级区分回收应用的缓存资源,从而实现对应用缓存分区容量的在线动态调节进而达成应用级的缓存管理语义. 实验数据表明 PARP策略能够在实际系统中有效的支持区分应用优先级,这不仅可以满足网络存储系统在多应用共享存储模式下对存储服务质量保证的需求同时可以与缓存效用原则相结合用以提高存储系统的整体性能.

References:

[1] Lofgren A. Decisions about storage area networks and network attached storage should not be based purely on capacity[EB/OL]. http://www.forrester.com/research, 2002.

[2] Goyal P, Jadav D, Modha D, et al. CacheCOW: QoS for storage system caches[C]. Heidelberg, German: Proc of the 11th International Workshop on Quality of Service, 2003, 498-515.

[3] Bovet D P, Cesati M. Understanding Linux kernel 3rd[M]. Cambridge: O'Reilly Press, 2005.

[4] Meng X X, Na W W, Xu W, et al. A dynamic partitioned buffer cache system designed for storage server[C]. Xian China: China National Computer Conference, 2008.

[5] Chakraborty L, Singh A. A utility-based approach to cost-aware caching in heterogeneous storage systems[C]. Los Alamitos, CA, USA: Proc of the 21th International Parallel and Distributed Processing Symposium, 2007, 1-10.

[6] Ko B J, Lee K W, Amiri K, et al. Scalable service differentiation in a shared storage cache[C]. Washington, DC, USA: Proc of the 23rd International Conference on Distributed Computing Systems, 2003, 184-194.

[7] UMass Trace Repository. OLTP application I/O and search engine I/O[EB/OL]. http://traces.cs.umass.edu/index.php/Storage/Storage, 2007.

附中文参考文献:

[4] 孟晓烜, 那文武, 徐　伟,等. 一种针对存储服务器设计的动态分区缓存管理系统[C]. 西安:中国计算机大会, 2008.