由于回写缓冲空间的大小是有限制的,当回写缓冲区达到上限之后,会选择性地从回写缓冲空间中置换出一些数据块.此时,对于那些以部分块模式被持久化的数据块,嵌入在它们中的更新最终会以整块模式被写回到磁盘上.而这里的延迟写就是 Hitchhike 调度器能够在回写技术中实现同步写的关键.因为在整个块回写操作之前,对同一个数据块上的多次小数据写可以被积累,利用批量写达到了增加性能的目的.

2.5 快速恢复

在部分块模式持久化方式中,会将数据块中追加或更新的部分嵌入到某一个宿主数据块中,直到该数据块从回写缓冲空间中置换出来,才会以整块模式持久化到磁盘上.当系统发生意外情况出现宕机时,回写缓冲区中的数据块已经以部分块模式的模式将数据持久化到磁盘,还有一些小数据写寄存在其他宿主数据块上.因此在发生意外情况时,系统可以恢复到原本的样子.

最简单的方式是扫描整个整个磁盘,将嵌入了小数据的数据块读出来.然后取出小数据部分,并从磁盘中读取计算该小数据写的数据块,根据小数据写的时间戳,依次和该数据块进行异或运算,最后算出多次小数据同步写之后的数据.

但是全盘扫描将会很耗时间,效率比较低.由于磁盘上数据布局的优化,程序对磁盘的访问存在空间局部性,在一段时间内,程序对磁盘的访问会集中在某块区域.根据这一特性,对磁盘进行更粗粒度的逻辑划分,构建一个逻辑结构.将磁盘划分为若干存储区(zone),每个存储区(zone)的大小是固定的,并且远远大于一个块的大小.根据存储区中数据块的持久化方式,将存储区分成了 *Z*-存储区(*Z*-zone)和 *N*-存储区(*N*-zone).*Z*-存储区代表的在该区域中存在以部分块模式进行持久化的数据块,而 *N*-存储区是指该区域中所有数据块的持久化方式均以整块模式完成,具体如图 7 所示.

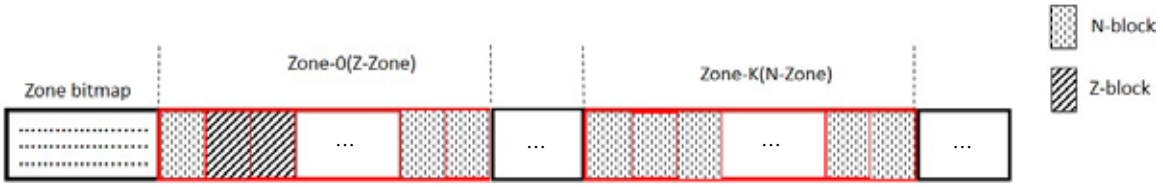

Fig.7 Disk image format

图 7 磁盘镜像格式

在图 7 中,*Z*-block 代表的是以部分块模式进行持久化的数据块,*N*-block 代表是以整块模式进行持久化的数据块.根据扇区号,磁盘被分为若干个存储区,每个存储区的类型只能是 *Z*-存储区或者 *N*-存储区,并且每种类型的存储区可能被分散到磁盘不用区域,因此,系统还需要在磁盘上维护一张存储区位图(zone bitmap),该位图将会标识每个 Zone 的类型.在最开始的时候,系统中没有以部分块模式进行持久化的数据块,此时所有存储区都为 *N*-存储区.当且仅当系统向 *N*-存储区中写入第 1 个 *Z*-block 时,需要更新存储区位图,将该存储区的类型从 *N*-存储区变为 *Z*-存储区.当 *Z*-存储区中所有的数据块都已整块模式进行了持久化之后,才会将该存储区的类型更改为 *N*-存储区.

当系统发生意外情况而进行重启时,系统首先读取存储区位图,确定磁盘上的哪些存储区是 *N* 存储区,然后再扫描 *N* 存储区中的数据块,将 *N*-block 数据块找出,然后进行数据的恢复.从而在系统恢复时减小系统扫描的磁盘区域,从而加快系统恢复.例如,当系统发生宕机时,磁盘划分如图 7 所示,当系统重启时,通过扫描存储区位图(zone bitmap),发现存储区 0 是属于 *Z*-存储区,从而系统只需扫描存储区 0 中的数据块.

3 实 验

本节对新提出的一种基于同步写的回写 IO 调度器进行验证,实验证明了该调度器的可行性.同时,把该调度器与传统调度器——Deadline 和 CFQ 进行对比,证明了该策略对小数据同步写有较好的性能提升,能够降低系统延迟,提高系统吞吐率.

3.1 实验环境

本实验中,我们是基于 linux 内核 2.6.32 版本中的 Deadline 调度器上实现了 Hitchhike 调度器原型,并且所有实现都是在一台主机上完成的,主机具体配置见表 1.

Table 1 System configuration used in the experiments

表 1 实验的系统配置

硬件	配置
RAM	2GB
CPU	Inter core i3-3240 CPU @ 3.40GH×4
Disk	500G
Operate system	Ubuntu 14.04TLS,32bit
内核	Linux 2.6.32

在实验中,我们使用的文件系统块大小为 4KB,4KB 也是文件系统块大小常用的一个值.本实验对数据块以及异或运算结果的压缩使用的是 LZ4(extremely fast compression algorithm)压缩算法,该压缩算法有较快的压缩速度和较好的压缩比.此外,本实验使用的基准测试是 Filebench.Filebench 是一款文件系统性能的自动化测试工具,它通过快速模拟真实应用服务器的负载来测试文件系统的性能,能够自动化生成负载.为了模拟小数据同步写,本实验修改 filebench 中的 filemicro_writefsync.f 文件,通过该文件,可以模拟一个小数据同步写的工作负载.该负载主要完成的内容为:循环地向一个文件中写入指定大小的小数据,每次追加后都使用 *fsync*()函数将数据同步到磁盘上,从而模拟小数据同步写环境.

为了测试 Hitchhike 调度器在不同大小的小数据同步写下的性能,本实验写入数据的大小为 64B~2 048B.每次实验运行时间为 5 分钟.为了对比 Hitchhike 调度器的小数据同步写性能,本实验还测试了 Deadline 调度器和 CFQ 调度器的小数据同步写性能.Deadline 调度器和 CFQ 调度器是使用的最为广泛的两种调度器.由于采用电梯策略,I/O 调度器会优先把靠近磁头移动方向最近的数据块写入到磁盘上,然而这会造成远离磁头的请求长时间得不到处理的现象.为了避免请求饿死,Deadline 调度器给每个请求一定的处理期限,该期限可以保证那些远离磁头移动方向的数据块可以得到及时处理.而 CFQ 调度器为了确保 I/O 带宽的公平分配,借此实现完全公平的调度算法.

3.2 实验分析

在该实验中,我们通过改变小数据同步写请求的数据大小来测试不同大小的小数据同步写在 I/O 调度器中的写,为确保每次小数据写请求能够同步到磁盘上,每次小数据写之后都执行 *fsync*(),*fsync* 函数可以确保修改过的数据块立即写入到磁盘上,并且等待磁盘写操作结束才会返回.

图 8 显示的是不同大小的小数据同步写在不同调度器中的写延迟,横坐标表示的是每次写入数据的大小,纵坐标表示的是每次同步写的写延迟.

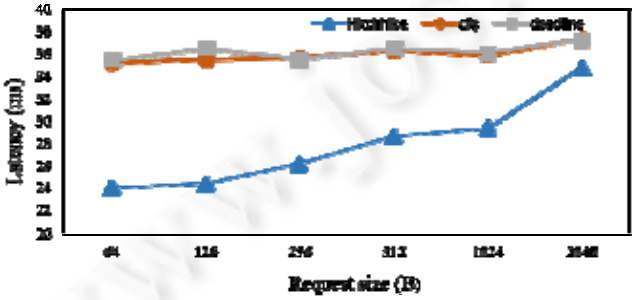

Fig.8 Latency changes of small synchronous writes under different request sizes

图 8 在不同大小的请求下,小数据同步写延迟的变化

从图 8 可知,不同大小的小数据同步写在 CFQ 和 Deadline 两种调度器上的写延迟基本相似,都固定在一个较为稳定的值.相比之下,在内核使用 Hitchhike 调度器时,不同大小的小数据同步写表现出的写延迟不相同,并且延迟均低于 CFQ 和 Deadline.从图中可以看到,随着写入数据的增大,使用 Hitchhike 调度表现出的写延迟逐渐升高,并且逐步接近于使用传统调度器的写延迟.

写延迟的降低,主要归功于 Hitchhike 调度器使用了部分块模式的持久化方式.该方式可以将小数据同步写嵌入到了其他数据块中,只需花费 1 次 I/O 操作即可实现对小数据和其他数据块的存储.从 I/O 次数方面考虑,部分块模式的持久化方式暂时节约 I/O 操作,从而使得平均写延迟降低.而且节约的 I/O 操作越多,写延迟降低得就会越多.表 2 显示了小数据写识别率.

Table 2 Recognition ratio of small writes

表 2 小数据写识别率

数据大小(B)	64	128	256	512	1 024	2 048
命中率(%)	98.5	97	93.8	87.6	75.1	50

该命中率的计算公式如下:

$$命中率=\frac{小数据写的次数}{完成的I/O操作总次数}.$$

命中率越高,表明小数据写所占比例越高,从而使用部分块模式持久化的次数也就越多,可以节约的 I/O 操作也相应地越多;相应的,写延迟也就降低了.从表 2 中可以看出:当每次写入的数据大小为 64B 时,命中率最高,达到了 98.5%,此时的写延迟也是最低的;随着数据大小的增大,命中率逐渐降低,写延迟也在逐渐增大.

图 9 显示的是不同大小的小数据同步写在不同调度器中的吞吐量.而实验中的吞吐量并不是很大,其原因主要有两个方面:(1) 本次实验模拟的是小数据写,所以基准测试生成的负载不高;(2) 每次向文件中写入小数据后,通过 *fsyn*()将小数据同步到磁盘上,而同步写的效率较低.从图 9 中可以看出:无论每次同步写入的数据多大,使用 Deadline 调度器和 CFQ 调度器时,系统的吞吐率都维持在 55ops/s.当系统使用 Hitchhike 调度器时,系统的吞吐率高于前两者.在同步写为 64B 时,使用 Hitchhike 的 I/O 调度器,系统的吞吐量比使用 CFQ 和 Deadline 的高出 48.6%;并且随着每次同步写数据量的增大,使用了 Hitchhike 的 I/O 调度器的系统的吞吐量在逐渐接近于使用了 CFQ 和 Deadline 的系统吞吐量.这个趋势也和写延迟相似.

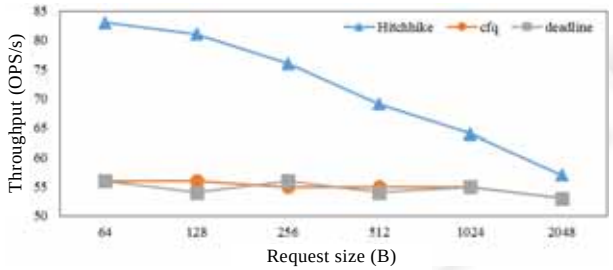

Fig.9 Throughput changes of small synchronous writes under different request sizes

图 9 在不同大小的请求下,系统吞吐量的变化

Hitchhike 调度器使用了压缩算法 LZ4 对数据块中的数据进行压缩.LZ4 是一种快速的无损压缩算法.使用压缩算法将会对系统的 CPU 造成额外开销.图 10 显示了系统处于内核状态下,CPU 使用的时间百分比.在图 10 中,横坐标表示时间,单位为 s;纵坐标表示的是 CPU 处于系统内核模式下的百分比.从图中可以看出:当系统使用 Hitchhike 调度器时,内核态下的 CPU 使用的时间百分比会稍高于使用 Deadline 调度器时的 CPU 使用时间百分比.其中,在系统使用 Deadline 调度器时,内核态下的 CPU 使用的时间百分比平均为 1.67%;在系统使用 Hitchhike 调度器时,内核态下的 CPU 使用的时间百分比平均为 2.03%,只比前者高出 0.36%.虽然 Hitchhike 调度器使用了压缩算法,这将会给内核态下的 CPU 带来一定的额外开销,但这个额外开销消耗的 CPU 资源较小.

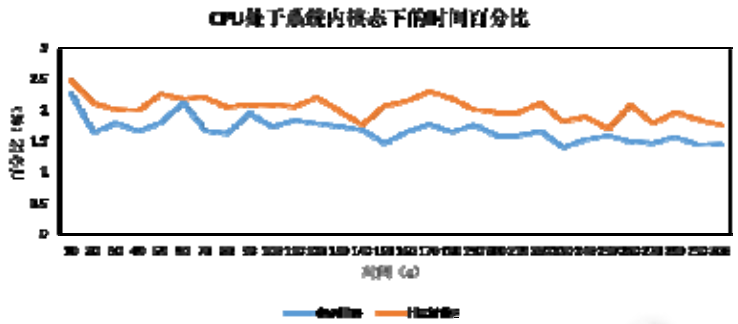

Fig.10 Percentage of time when CPU is running in system kernel space

图 10 CPU 处于系统内核态下的时间百分比

Hitchhike 调度器除了使用压缩算法 LZ4 对数据块中的数据进行压缩以外,还维护了一个回写缓存空间(write back buffer).无论是压缩算法还是回写缓存空间,都会对系统内存造成一定的额外开销.图 11 显示了在分别使用 Hitchhike 调度器和 Deadline 调度器时,系统内存的使用情况.在图 11 中,纵坐标表示已使用的内存(不包括交换分区(swap))和总内存的比值;横坐标表示时间,单位为 s.为了识别小数据,Hitchhike 调度器会在回写空间中缓存数据块.随着时间的推移,缓存的数据块越来越多,消耗的内存也就越来越多.所以随着时间的推移,系统使用的内存呈线性增长.但是回写缓存空间可存储的数据块数目是有限制的,当回写缓存空间的可用容量使用完之后,系统会使用最近最少使用的(LRU)算法置换出一些数据块.从图中可以看出,在 210s 之后,系统已使用的内存趋于稳定值,基本维持在 41%左右.使用 Deadline 调度器时,内存平均使用率为 35.98%.相比之下,使用 Hitchhike 调度器时,内存平均使用了 39.39%,稍低于前者,但是并没有给系统造成很大的系统负担.

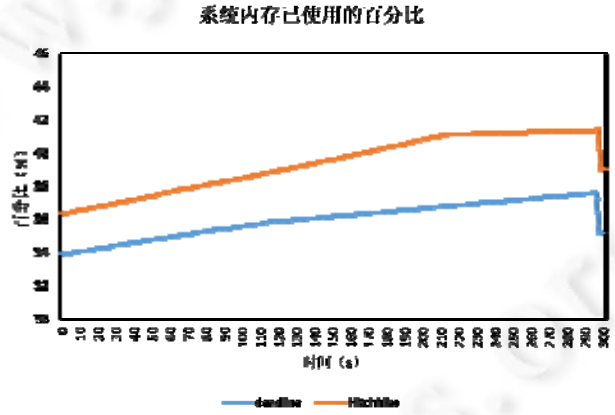

Fig.11 System memory usage

图 11 系统内存使用情况

4 小 结

本文提出了一种基于同步写的回写 I/O 调度器——Hitchhike.该调度器管理了一个回写缓存空间.该空间可以用于缓存已写入的数据块,通过对同一物理地址的数据块中的数据进行比较来识别小数据写请求.通过对其他数据块中的数据进行压缩,节省出数据所占空间,然后将小数据嵌入到节省出的空间中,从而可以在一次 I/O 操作中同时实现对其他数据块和小数据的存储.本文是基于 Linux 内核 2.6.32 版本中的 Deadline 调度器实现的,并且通过运行基准测试程序来验证 Hitchhike 调度器对小数据同步写的性能.通过和 Deadline 调度器和 CFQ 调度器比较可知,Hitchhike 调度器可以降低小数据写延迟,增大系统对小数据写的吞吐量.此外,使用

Hitchhike 调度器对系统资源的额外开销也较小.

References:

[1] Bryant RE. Data-Intensive supercomputing: The case for DISC. Pdl.cmu.edu, 2007.

[2] Hey T, Trefethen A. The data deluge: An e-science perspective. In: Proc. of the Grid Computing: Making the Global Infrastructure a Reality. 2003. 809−824. [doi: 10.1002/0470867167.ch36]

[3] Szalay AS, Kunszt PZ, Thakar A, Gray J, Slutz D, Brunner RJ. Designing and mining multi-terabyte astronomy archives: The Sloan digital sky survey. ACM SIGMOD Record, 1999,29(2):451−462. [doi: 10.1145/342009.335439]

[4] How much text verus metadata is in a tweet? 2011. http://goo.gl/EBFIFs

[5] Decandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, Sivasubramanian S, Vosshall P, Vogels W. Dynamo: Amazon's highly available key-value store. ACM SIGOPS Operating Systems Review, 2007,41(6):205−220. [doi: 10.1145/1323293.1294281]

[6] Miller EL, Greenan K. Reliable and efficient metadata storage and indexing using NVRAM. 2008.

[7] Ousterhout JK, Costa HD, Harrison D, Kunze JA, Kupfer M, Thompson JG. A trace-driven analysis of the UNIX 4.2BSD file system. ACM SIGOPS Operating Systems Review, 1985,19:15−24. [doi: 10.1145/323647.323631]

[8] Atikoglu B, Xu Y, Frachtenberg E, Jiang S, Palaczny M. Workload analysis of a large-scale key-value store. ACM SIGMETRICS Performance Evaluation Review, 2012,40(1):53−64. [doi: 10.1145/2318857.2254766]

[9] Wu X, Xu Y, Shao Z, Jiang S. LSM-Trie: An LSM-tree-based ultra-large key-value store for small data. In: Proc. of the 2015 USENIX Annual Technical Conf. USENIX Association, 2015. 71−82.

[10] Rosenblum M, Ousterhout JK. The design and implementation of a log-structured file system. In: Proc. of the ACM SIGOPS Operating Systems Review. ACM Press, 1996. 1−15. [doi: 10.1145/121132.121137]

[11] Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber RE. Bigtable: A distributed storage system for structured data. ACM Trans. on Computer Systems, 2008,26(2):205−218. [doi: 10.1145/1365815.1365816]

[12] O'Neil P, Cheng E, Gawlick D, O'Neil E. The log-structured merge-tree (LSM-tree). Acta Informatica, 1996,33(4):351−385. [doi: 10. 1007/s002360050048]

[13] Wu X, Shao Z, Jiang S. Selfie: Co-Locating metadata and data to enable fast virtual block devices. In: Proc. of the ACM Int'l Systems and Storage Conf. ACM Press, 2015. 1−11. [doi: 10.1145/2757667.2757676]

[14] Chen PM, Ng WT, Chandra S, Aycock C, Rajamani G, Lowell D. The Rio file cache: Surviving operating system crashes. ACM Sigplan Notices, 1996,31(9):74−83. [doi: 10.1145/248209.237154]

[15] Wang Y, Davis K, Xu Y, Jiang S. iHarmonizer: Improving the disk efficiency of I/O-intensive multithreaded codes. In: Proc. of the IEEE Int'l Parallel and Distributed Processing Symp. IEEE Computer Society, 2012. 921−932. [doi: 10.1109/IPDPS.2012.87]

[16] Srinivasan K, Bisson T, Goodson G, Voruganti K. iDedup: Latency-aware, inline data deduplication for primary storage. In: Proc. of the USENIX Conf. on File and Storage Technologies. USENIX Association, 2012.

[17] Chen J, Wei Q, Chen C, Wu L. FSMAC: A file system metadata accelerator with non-volatile memory. In: Proc. of 2013 IEEE the 29th Symp. on Mass Storage Systems and Technologies (MSST). IEEE, 2013. 1−11. [doi: 10.1109/MSST.2013.6558440]

[18] Condit J, Nightingale EB, Frost C, Ipek E, Lee B, Burger D, Coetzee D. Better I/O through byte-addressable, persistent memory. In: Proc. of the ACM Symp. on Operating Systems Principles (SOSP 2009). 2009. 133−146. [doi: 10.1145/1629575.1629589]

[19] Non-Volatile cache for host-based raid controls. 2011. http://www.dell.com/downloads/global/products/pvaul/en/NV-Cache-for-Host-Based-RAID-Controllers.pdf

[20] Ganger GR, Mckusick MK, Soules CAN, Patt YN. Soft updates: A solution to the metadata update problem in file systems. ACM Trans. on Computer Systems, 2000,18(2):127−153. [doi: 10.1145/350853.350863]

[21] Chen F, Koufaty DA, Zhang X. Hystor: Making the best use of solid state drives in high performance storage systems. In: Proc. of the Int'l Conf. on Supercomputing. 2011. 22−32. [doi: 10.1145/1995896.1995902]

[22] Huang H, Hung W, Shin KG. FS2: Dynamic data replication in free disk space for improving disk performance and energy consumption. ACM SIGOPS Operating Systems Review, 2005,39(5):263−276. [doi: 10.1145/1095809.1095836]

[23] Wallace G, Douglis F, Qian H, Shilane P, Smaldone S, Chamness M, Hsu W. Characteristics of backup workloads in production systems. In: Proc. of the USENIX Conf. on File and Storage Technologies. USENIX Association, 2012.

[24] Chidambaram V, Sharma T, Arpaci-Dusseau AC, Arpaci-Dusseau RH. Consistency without ordering. In: Proc. of the USENIX Conf. on File and Storage Technologies. USENIX Association, 2012.

[25] Lu Y, Shu J, Wang W. ReconFS: A reconstructable file system on flash storage. In: Proc. of the USENIX Conf. on File and Storage Technologies. USENIX Association, 2014. 75−88.

刘星(1991－),男,江西萍乡人,硕士,主要研究领域为云存储,操作系统.

江松(1969－),男,博士,研究员,博士生导师,主要研究领域为操作系统,文件和存储系统,高性能计算,互联网与云计算.

王洋(1966－),男,博士,研究员,博士生导师,CCF 专业会员,主要研究领域为并行分布计算,云计算,虚拟化技术.

范小朋(1978－),男,副研究员,CCF 专业会员,主要研究领域为移动计算,云计算,大数据分析,车联网.

须成忠(1965－),男,博士,研究员,博士生导师,主要研究领域为并行与分布式系统,互联网与云计算,高性能计算,移动嵌入式系统.