doi:10.3772/j.issn.1002-0470.2015.02.006

一种 Linux 环境下优化 ZFS 同步写性能的方法①

沈 岩②* 张广艳③** 杨长江***

(*首都师范大学信息工程学院 北京 100048)

(**清华大学计算机科学与技术系 北京 100084)

(***中兴通讯平台软件长沙开发部 长沙 410205)

摘 要 为了提高 Linux 环境下的 ZFS 文件系统的同步写性能,在研究分析现有 Linux 环境下实现的 ZFS 文件系统的块设备同步写性能较低,同步写负载较重时系统吞吐量明显下降的原因的基础上,提出了一种 Linux 环境下的 ZFS 同步写优化方法。该方法通过同步写合并提交策略减少日志提交所产生的 I/O 请求数;使用同步写全局负载均衡机制实现各盘之间同步写的负载均衡来提高同步写的吞吐量。实验结果表明:相比原来在 Linux 上实现的 ZFS,该优化方法提高同步写性能 23.9% ~88.14%。

关键词 块设备, 同步写, ZFS 文件系统, 负载均衡

0 引 言

ZFS(Zettabyte file system)是 Sun 公司研发的文件系统,该系统可以提供文件系统接口和块设备接口的统一存储系统。统一存储是指同时支持文件访问和块设备访问等多种访问接口的存储系统。一套系统可以支持用户变化的接口需求,比如原来通过网络文件系统(NFS)提供网络存储服务,后来通过互联网小型计算机系统接口(iSCSI)供大型数据库使用。一套系统可以同时支持不同的接口需求,并通过高级的存储功能软件实现跨应用系统的资源调配、负载均衡和数据保护。相同的硬件、相同的软件降低了系统的部署成本;相同的人员、相同的过程节省了系统的运维开销。因此,一套系统具有统一存储的特征是很重要的。

类似于 Linux 逻辑卷管理器(LVM),ZFS 提供的块设备接口可以充分利用多块磁盘的带宽资源,拥有配置灵活的特点[1]。与面向对象的存储系统设备不同,ZFS 的块设备接口可以充分利用多块盘的带宽,为通用的文件系统提供灵活的块设备接口[1,2]。ZFS 由 Solaris 移植到 Linux 环境中,称为 ZFS on Linux[3]。ZFS on Linux 的块设备同步写性能较低,原因是 ZFS 处理同步写请求时,每个写请求都单独提交日志,这种同步方式虽然保证了同步写语义,却因为牺牲了同步写的并发度而降低了同步写的吞吐量。当同步写负载较重时,性能会明显下降。为了提高同步写的吞吐量,ZFS 利用多块盘的带宽,将同步写分布到各个设备上。这种策略虽然能够保证单个逻辑块设备(ZFS 中称为 ZVOL)的同步写在其成员盘之间是负载均衡的,但在多个 ZVOL 存在同步写时不能保证底层存储设备之间的负载均衡。针对 Linux 环境下的 ZFS 的同步写性能较低的问题,本文提出了一种 Linux 环境下的 ZFS 同步写性能优化方法。该优化方法采用同步写合并提交策略,将同步写请求放入到同步写队列中并合并提交日志写,提高了同步写的吞吐量;采用同步写全局负载均衡机制确保无论单个 ZVOL 还是多个

① 国家自然科学基金(61170008, 61272055)和 863 计划(2013AA01A210)资助项目。

② 男,1990 年生,硕士生;研究方向:网络存储与分布式系统;E-mail: shenyan_001@126.com

③ 通讯作者,E-mail: gyzh@tsinghua.edu.cn

(收稿日期:2014-09-11)

ZVOL 存在同步写时在底层存储设备之间的负载均衡。本研究基于 ZFS on Linux 设计并实现了 ZFS 同步写性能优化方法,利用标准 I/O 测试工具 fio 测试了多个 ZVOL 的同步写性能。实验结果表明,相比原来在 Linux 上实现的 ZFS,该优化方法提高了同步写性能 23.9% ~88.14%。

1 Linux 环境下的 ZFS 实现

ZFS 是由 SUN 公司为 Solaris 操作系统开发的文件系统。由于 ZFS 代码具有良好的可移植性与安全性[4],目前,ZFS 已经被移植到 Linux、FreeBSD、OSX 等其它平台上。其中,ZFS on Linux 是在 Linux 内核空间实现了 ZFS 的核心功能。ZFS on Linux 应用广泛,比如目前世界排名前 10 的超级计算机 Sequoia 就采用 ZFS on Linux 作为底层存储系统,Cloudscaling 公司利用 ZFS on Linux 作为 OpenStack 开放云存储的底层存储组件。

ZFS on Linux 整体结构自上而下由接口层、数据管理层和存储池层组成,见图 1。在接口层,通过 ZPL(ZFS Posix 层)对外提供文件系统接口,通过 ZVOL(图中 ZFS 容量模拟器)提供块设备接口。数据管理层集合了事务对象处理、快照、日志等对象级实现,同时利用自适应缓存管理算法(ARC)提供高效的自适应的缓存机制。存储分配层负责维护 ZFS 的存储池,存储空间的申请与释放,以及对底层存储设备的读写控制。ZFS 不仅利用写时拷贝技术实现了存储快照的功能[6],同时通过该方法保证数据的一致性与可靠性[7-9]。使用日志写(ZFS Intent Log, ZIL)实现 ZFS 的同步写语义,保证数据可以同步到磁盘中,防止系统突然断电而导致数据丢失。

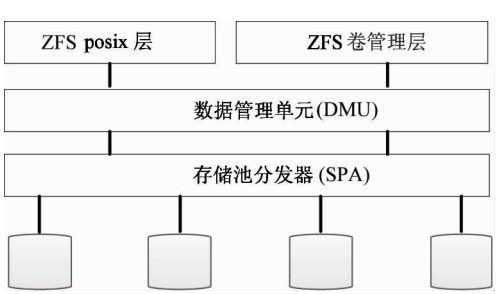

图 1 ZFS 整体架构[5]

其次,ZFS 每次执行日志写都会将整个 ZVOL 相关的内存中等待提交的日志项全部提交到磁盘上。如图 2,每个同步写请求都会提交一次日志写,并等待日志写提交结束。所以 ZFS 的同步写性能取决于 ZIL 提交到磁盘的时间。当 ZFS 的同步写负载较重时,这种同步方式会严重影响队列中其他请求的服务效率,降低同步写操作的并发度,从而明显降低系统吞吐量。

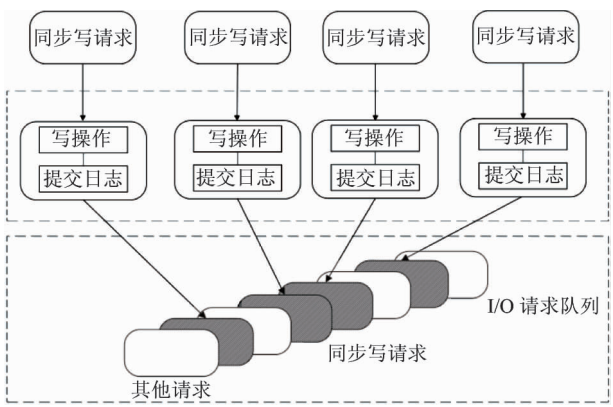

图 2 同步写请求处理过程

为了提高 ZFS 日志写的性能,ZFS 提供两种日志同步方法。第一种方法是指定快速存储设备为专门的 Log 盘,一般采用固态盘(SSD)作为 Log 盘。这种方法主要针对要求实现低延迟同步写的场景。但是当闪存芯片达到擦除重写次数后,存储块可能存在损坏,威胁数据的可靠性[10]。第二种方法是为每个 ZVOL 维护一个同步写请求分发器,将同步写请求分发到各个成员设备上,这样可以充分利用磁盘带宽资源,提高 ZIL 日志写的吞吐量。ZFS 采用类似于轮询的方法将每个 ZVOL 的请求分发到各个成员盘上。当系统中只存在一个 ZVOL 时,该同步方式可以保证底层存储设备的负载均衡。但是当系统中存在多个 ZVOL 时(见图 3),这种同步方式却难以保证底层存储设备的负载均衡。

2 Linux 环境下的 ZFS 同步写性能优化方法

同步写性能优化方法主要包括两部分:同步写合并提交策略和同步写全局负载均衡机制。

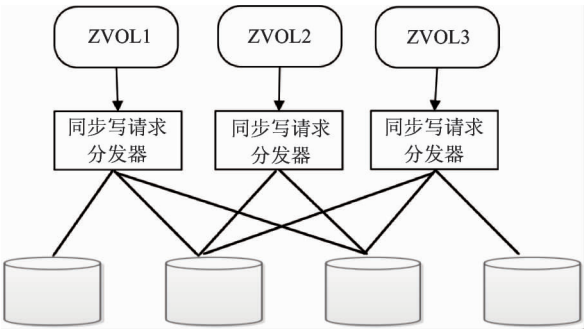

图 3　不均匀的日志写分布

2.1　同步写合并提交策略

为了优化同步写性能,本研究提出了同步写合并提交策略。为了提高同步写的效率,本文为每个 ZVOL 创建一个同步写队列,并创建了一个称作 sync _ thread 的内核线程。sync _ thread 线程既负责监听同步写队列,又负责合并提交日志。

如图 4 所示,当有同步写请求需要提交日志时,将该请求插入到同步写队列中(sync write queue)并通知 sync _ thread 线程。如果监听到其他线程向同步写队列中插入写请求,sync _ thread 会将同步写队列中待提交的日志项一次性全部提交到磁盘上。提交日志结束后,sync _ thread 删除并返回队列中已提交日志的请求以确保同步写语义。等同步写请求返回结束后,sync _ thread 检查队列状态,如果发现队列为空,sync _ thread 阻塞并条件等待同步写队列接收新的请求,否则继续处理队列中的新请求。

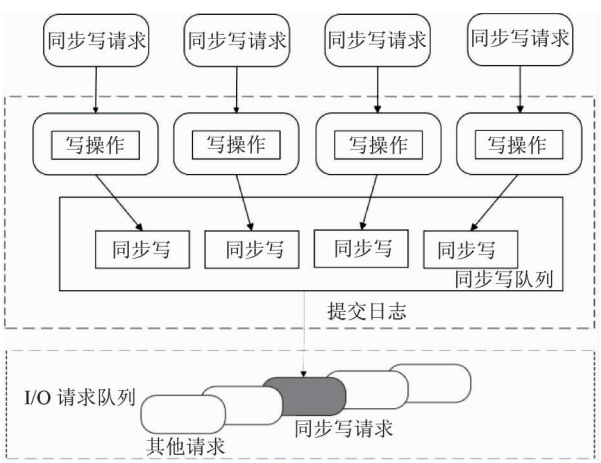

图 4　同步写合并提交策略

sync _ thread 提交日志时需要同步访问磁盘,花费时间相对较长。在此期间,同步写队列可以接收更多同步写请求。当同步写请求负载较重时,同步写队列会接收到大量的同步写请求。由于同步写合并提交策略不仅可以合并提交同步写请求,而且对同步写日志内容进行合并[11]。所以被合并的请求数越多,合并提交日志的优势越明显。同步写合并提交策略虽然延迟了部分先到队列的同步写请求的处理,却提高了同步写的并行度,从而提高了同步写的整体吞吐量。而当同步写请求负载较轻时,因为 sync _ thread 只要监听到同步写队列不为空就会提交日志,所以可以确保同步写请求得到及时响应。

2.2　同步写全局负载均衡机制

由于 ZFS 的局部同步写请求分发器难以保证底层存储设备的负载均衡,我们提出用全局同步写请求分发器来确保同步写请求可以均匀分布到 ZFS 的成员设备上。全局同步写请求分发器不区分同步写请求的来源,负责完成对所有同步写请求的合并调度和分发工作,如图 5 所示。

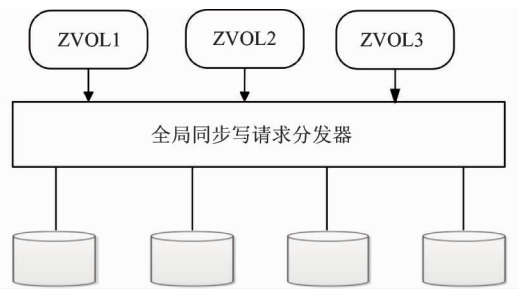

图 5　同步写全局负载均衡

同步写请求全局分发器内部维护一个轮询指针 HPTR,指向当前分配的成员设备。已知 ZFS 的成员设备组 $D_n = \{dev_i \mid i = 1,\cdots,n, n \geqslant 1\}$ 以及日志记录序列 $L_{m,q} = \{(data_i, vdev_i) \mid i = m, m+1,\cdots,q;\ m \geqslant 0;\ q > m;\}$($vdev$ 表示日志记录的成员设备编号)。对于 $L_{m,q}$ 中的日志记录 R_i 满足 $vdev_i = dev_j$,则 $HPTR = dev_j$ 且对于下一条日志记录 R_{i+1},满足 $vdev_{i+1} = dev_{(j+1) mod(n)}$。

由于同步写请求全局分发器为所有 ZVOL 共享,所以无论用户创建单个或多个 ZVOL,全局分发器都能够将同步写请求均匀分布到各个成员设备上,因而可以更好地利用成员设备的 I/O 带宽。虽然访问同步写请求全局分发器需要加锁,但是临界区内的操作只是改变轮询指针的位置,因此引入同

步写全局分发器所产生的开销很小。同步写全局分发器不会因 ZVOL 的数量的增加而导致系统性能的下降。

3 性能评价

为了验证提出的同步写性能优化方法的有效性,对经过该方法优化后的 ZFS 进行了性能测试,并与原 ZFS 进行了比较。

3.1 测试场景

测试机器的硬件配置为 Intel(R) Xeon(R) 2.13GHz四核 CPU,8MB Cache,内存大小为 8GB,磁盘均为转速为 7200r/min 、容量 500GB 的 SATA 盘。测试机器的操作系统为 Ubuntu 10.04.3(内核版本为 2.6.32)。实验选取 zfs-0.6.0-rc11(ZFS on Linux 稳定版)作为实验对比。

将三块 SATA 盘作为 ZFS 的成员设备创建了一个存储池。在存储池中创建了三个块大小为 128kB、容量均为 10GB 的虚拟块设备 ZVOL1, ZVOL2,ZVOL3。实验采用标准 I/O 测试工具 fio 测试性能。fio 是一个可以产生不同 I/O 负载的 I/O 测试工具。fio 可以定义不同的测试参数,包括执行的线程数、每个请求的 I/O 大小、I/O 访问的方式以及有关同步写的参数 fsync。其中,*fsync* = *n* 表示每执行 *n* 次 I/O,才发起一次脏数据的同步操作。fio 创建三个线程,同时写入 ZVOL1,ZVOL2,ZVOL3。为了模拟高并发的 I/O 负载,每个线程采用异步I/O 的方式访问块设备,每次提交 I/O 数设置为 100,通过调整 *fsync* 参数来保证数据被同步写入磁盘。

3.2 性能测试结果

同步写测试主要验证同步写合并提交策略与同步写全局负载均衡策略的优化效果。

图 6 为 *fsync* = 5, I/O 块大小为 4kB 时同步写带宽的变化情况。从结果来看,优化后的同步写性能平稳,且明显优于原 ZFS。然后,测试不同同步写负载下的同步写性能的每秒读写(I/O)操作的次数(input/output operations persecond, IOPS)和写操作的延迟。测试分为 4 组,fsync 分别为 1、5、10 和 20。每组测试 3 次,取 3 次 IOPS 的平均值,如图 7,图 8 所示。对于 *fsync* = 1 的情况,由于一次只同步一个 I/O,无法合并提交多个 I/O 请求,所以性能无提升。当 fsync 的值变大时,一次可同步的 I/O 数变多,同步写优化效果明显。

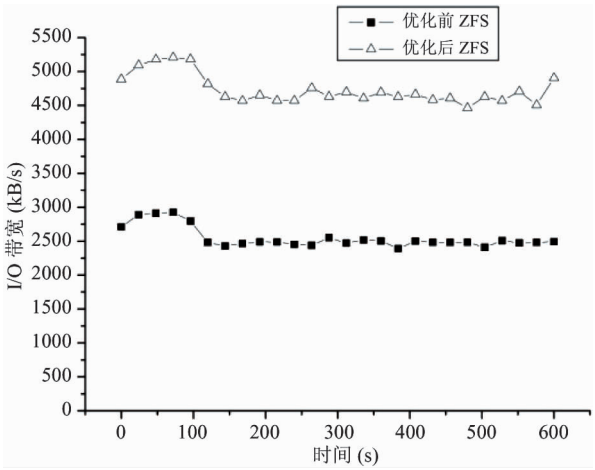

图 6 *fsync* = 5 时的同步写性能

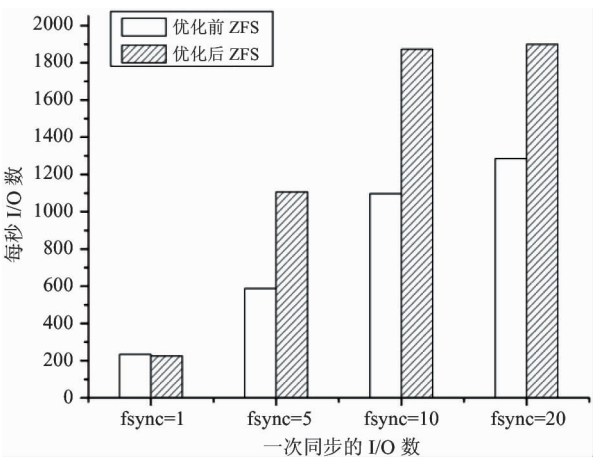

图 7 不同写负载下的同步写性能

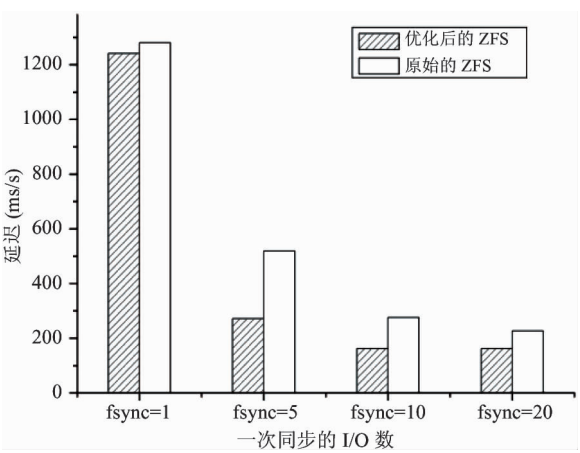

图 8 不同写负载下的同步写的延迟

从图 8 可以看出,不仅同步写的吞吐量有了很明显的提高,同步写的平均延迟也有明显的下降。本文优化方法的整体延迟降低的原因在于:虽然同步写合并提交策略延迟了部分先到队列的请求的执行,但是由于集中提交减少了 I/O 队列长度,优化后的同步写的平均延迟仍有明显下降。

图 9 为 I/O 块大小为 8kB,*fsync* = 5 对应的同步写带宽随时间变化的情况。可以看出优化后的同步写带宽变化平稳且明显优于原 ZFS。最后测试了 *fsync* = 5 时,不同的 I/O 块大小对同步写优化的影响,每组数据测试 3 次,每次 10min,取 3 次 IOPS 的平均值。图 10 给出了 I/O 块大小对同步写性能的影响,I/O 块大小为 4kB、8kB、16kB 和 32kB 时,IOPS 分别提升了 88.14%、54.9%、32.17% 和23.9%。结果表明,I/O 块越小,优化效果越明显。这是由于当 I/O 块较小时,块设备的 IOPS 较大,一次可合并的同步写较多,合并提交日志的优化效果较为明显。

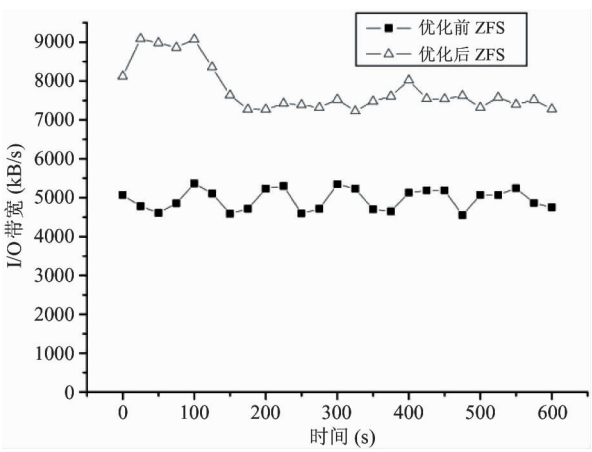

图 9　I/O 块大小为 8kB 时的同步写性能

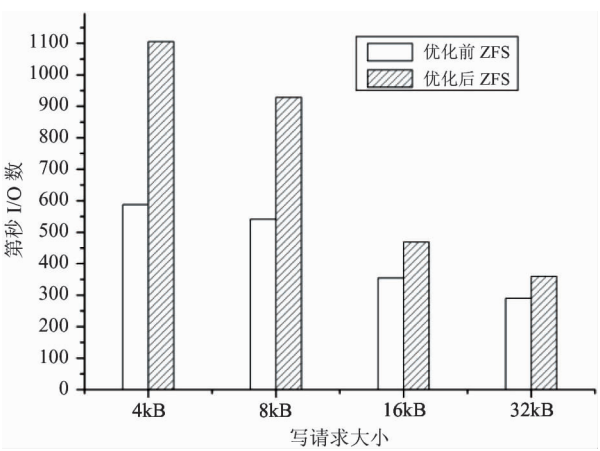

图 10　不同 I/O 块大小情况下的同步写性能

4　结　论

针对 Linux 环境下 ZFS 的块设备同步写性能较低的问题,本文提出了一种 Linux 环境下 ZFS 同步写优化方法 FS。它采用同步写合并提交策略,一次提交多个同步写请求,提高了同步写的吞吐量;采用同步写全局负载均衡策略,确保在多个 ZVOL 存在时同步写在底层存储设备之间的负载均衡。实验结果表明,本文提出的 ZFS 同步写优化方法可以很好地提高 ZFS 的同步写的吞吐量,同时缩短了同步写的延迟时间。相比原来在 Linux 上实现的 ZFS,本文方法提高了同步写性能 23.9% ~88.14%。

参考文献

[1] Phromchana V, Natawut N, Piromsopa K. Performance evaluation of ZFS and LVM (with ext4) for scalable storage system. In: Proceedings of the 8th International Joint Conference on Computer Science and Software Engineering, Nakhon Pathom, Thailand, 2011. 250-253

[2] 鞠大鹏, 刘川意, 汪东升等. 在 Linux 内实现基于内容存储的驱动器(CASD). 高技术通讯, 2009, 19(3): 224-229

[3] Brain B. ZFS on Linux. http://zfsonlinux.org/, 2014

[4] Bonwick J, Moore B. ZFS: The Last Word In File Systems . http://wiki.illumos.org/download/attachments/1146951/zfs_last.pdf, 2007

[5] Fio, flexible I/O tester . http://linux.die.net/man/1/fio, 2007

[6] 周煜, 卢正添, 易固武等. 一种基于过滤驱动的写时拷贝快照方法. 四川大学学报: 自然科学版, 2010(3): 478-482

[7] Dawidek P J. Porting the ZFS file system to the FreeBSD operating system. In: Proceedings of AsiaBSDCon, Tokyo, Japan, 2007. 97-103

[8] Prabhakaran V, Bairavasundaram L, Agrawal N, et al. IRON file systems. In: Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP 2005), Brighton, UK, 2005. 206-220

[9] Zhang Y P, Abhishek R, Andrea C, et al. End-to-end Data Integrity for File Systems: A ZFS Case Study. In: Proceedings of the 8th USENIX Conference on File and Storage Technologies, San Jose, USA, 2010. 29-42

[10] Agrawal N, Prabhakaran V, Wobber T, et al. Design Tradeoffs for SSD Performance. USENIX Annual Technical Conference. Boston Massachusetts. 2008. 57-70

[11] 张虎, 董小社, 武卫国等. 一种基于日志合并优化的数据同步机制. 小型微型计算机系统, 2006,27(12): 2183-2188

An approach to optimizing synchronous write performance of ZFS on Linux

Shen Yan[*], Zhang Guangyan[**], Yang Changjiang[***]

([*] College of Information Engineering, Capital Normal University, Beijing 100048)

([**] Department of Computer Science and Technology, Tsinghua University, Beijing 100084)

([***] Platform Software Changsha Development Department, ZTE Corporation, Changsha 410205)

Abstract

To improve the synchronous write performance of the ZFS file system working in the Linux environment, an approach to optimizing the synchronous write performance of the ZFS on Linux was proposed based on investigation of the reseans of the block device's lower synchronous write performance and the system's low throughput under the heavy load of the existing ZFS on Linun. Firstly, the approach uses the strategy of coalescing synchronous write to reduce the number of disk I/Os. Secondly, it uses global load balance of synchronous writes to achieve load balance among all disks in ZFS's storage pool to increase the throughput of synchronous writes. The experimental results showed that the approach improved the synchronous write performance of the existing ZFS on Linux by 23.9% ~88.14%.

Key words: block device, synchronous write, ZFS file system, load balance