0254-6124/2013/33(2)-219–06 *Chin. J. Space Sci.* 空间科学学报

Guo Qingpu, Xia Junbao, Zhong Dingkun. An SSD-based disk cache architecture and its application to data system of data center of Meridian Project. *Chin. J. Space Sci.*, 2013, **33**(2): 219-224

一种基于固态盘的磁盘缓存系统架构及其在子午工程数据中心业务系统中的应用*

郭清溥[1] 夏军宝[2] 钟鼎坤[3]

1(河南财经政法大学 郑州 450002)

2(中国地质大学 北京 100083)

3(中国科学院空间科学与应用研究中心 北京 100190)

摘 要 固态盘作为内存和磁盘之间的二级磁盘缓存运用于服务器存储层次结构中, 以满足空间科学研究领域日益增长的应用级 I/O 请求. 然而, 当前主流混合存储架构未能充分发挥固态盘优势, 也没有考虑其寿命损耗. 因此, 本文提出一种联合固态盘、磁盘的 RAF (Random-Access First) 混合存储架构, 在提高系统性能的同时可延长固态盘寿命. 其主要思路是通过序列探测技术, 区分负载中的随机访问和连续访问, 并将随机和顺序访问请求重定向到固态盘和磁盘系统分别处理. RAF 的原型已在 Linux 内核 2.6.30.10 中实现. 基于子午工程真实业务数据及模拟数据的实验结果表明, RAF 与当前的主流架构 Flash Cache 相比, 在多种负载情况下可提升系统响应时间 17%, 同时减少固态盘老化率 53%, 提高了存储系统的整体性能.

关键词 固态盘, 磁盘缓存, 序列侦测

中图分类号 V 55

An SSD-based Disk Cache Architecture and its Application to Data System of Data Center of Meridian Project

GUO Qingpu[1] XIA Junbao[2] ZHONG Dingkun[3]

1(*Henan University of Economics and Law, Zhengzhou* 450002)

2(*China University of Geoscience, Beijing* 100083)

3(*Center for Space Science and Applied Research, Chinese Academy of Sciences, Beijing* 100190)

Abstract Solid-State Drivers (SSD) are integrated in server storage hierarchy as a second tier of disk cache between DRAM and disks for caching more data from disks to meet the increasingly intensive I/O demands. Unfortunately, available state-of-art hybrid storage architectures cannot fully exploit SSDs' potentials and shorten their lifetime. In this paper, RAF (Random Access First), a hybrid storage architecture that combines an SSD based disk cache and a disk drive subsystem, is

* 河南省教育厅自然科学基金项目资助 (2008B520002)

2011-11-29 收到原稿, 2012-11-08 收到修定稿

E-mail: gqp@huel.edu.cn

proposed. RAF focuses on improving the system performance while extending the lifetime of SSD through providing priority to caching random-access data. Random-access and sequential-access data are identified by sequence detection scheme, and they will be served by SSD and Hard Disk Drive (HDD) respectively. RAF is implemented in Linux kernel 2.6.30.10. The results of experiments to apply it to the simulation and real data systems of Data Center of Meridian Project show that RAF can improve the performance by an average rate of 17% and reduce the flash wear by 53% compared with the state-of-art Flash Cache architecture.

Key words SSD, Disk cache, Sequence detection

0 引言

子午工程沿北起漠河, 经北京、武汉、海南并延伸至南极中山站的东经 120° 子午线附近, 通过无线电、地磁 (电)、光学和探空火箭等多种探测手段, 监测地磁场、电场、中高层大气的风场、密度、温度和成分以及电离层、磁层和行星际空间中的相关参数. 子午工程数据中心日处理数据达 3 GB, 预计每年总数据量约为 3 TB. 这些数据大多属于小于 5 MB 的小文件, 而在数据接收、缓存、预处理、迁移、产品加工、入库、发布以及数据管理和用户共享访问等业务中, 会生成大量超过 100 MB 的处理日志, 因此, 在系统中会产生巨大的随机访问与连续访问数据和处理日志的存储需求. 由于传统的机械磁盘难以克服由其机械特性产生的性能瓶颈, 无法很好地满足这种需求. 固态盘的出现可以很好地解决传统机械磁盘所面临的问题. 相对于机械磁盘, 固态盘作为电子设备不存在磁头的寻道和旋转定位时间, 因此有更快的随机访问性能, 可以作为磁盘系统的高速缓存应用于服务器存储系统中.

在当前主流的混合存储架构中, 固态盘常被作为内存的扩展来工作. 通常, 采用哈希表 (Hash Table) 来记录固态盘数据与磁盘数据之间的映射关系. 当一个读请求到达时, 首先通过哈希表检索请求数据是否在固态盘中. 如果没有命中, 则读请求由磁盘服务. 随之, 从磁盘中读取的数据被同步复制到固态盘中, 并更新哈希表. 如果读命中, 读请求由固态盘服务, 从而节省了机械磁盘所耗费的磁头定位时间. 当一个写请求到达时, 数据被写入固态盘, 如果写命中, 则更新哈希表. 固态盘中的废数据 (dirty data) 将周期性地写回到磁盘中.

然而, 这种混合存储系统结构并没有真正发挥出固态盘的优势, 在实际应用中系统性能提升不大且是以缩短固态盘寿命为代价的. 究其原因主要有以下几点: 固态盘在读写请求混合情况下性能会急剧下降; SSD 独自承受高负载, 顺序请求占多数时会耗尽带宽, 从而增加请求的排队时间, 成为系统的瓶颈; 固态盘吸纳了整个系统的写操作, 使其老化加速; 固态盘写命中比读命中要付出更多代价, 而当前架构中没有考虑这一点.

为解决以上问题, 本文提出了 RAF 架构, 此架构主要有以下特点.

(1) 通过数据检测策略分离随机访问和连续访问. 固态盘仅处理收益高的随机存取数据而将连续存取数据交由磁盘系统直接处理.

(2) 固态盘被动态划分为读区和写区来分别服务读请求和写请求, 限制固态盘内部垃圾回收开销.

(3) 当随机访问数据被置换出内存的文件缓存时, 将其拷贝到固态盘中的读区, 这一排他性策略可以增大磁盘缓存容量并减少固态盘写命中.

(4) 写区域被作为一个循环的预写日志而不是一个废数据写回型的缓存, 从而可减少响应时间并且限制垃圾回收开销.

RAF 的原型系统在内核版本号为 2.6.30.10 的 Linux 环境中通过 1 个 Mtron 固态盘和 4 个希捷硬盘来实现. 试验结果表明, 与当前主流的 Flash-Cache 架构相比, RAF 在多种负载下可平均调高系统响应时间 17%, 同时平均减少固态盘的损耗 53%.

1 系统架构

RAF 的系统架构如图 1 所示. 内核态的 Monitor 模块被用来截获块级别的操作, 与文件系统的缓存和缓冲区进行交互, 并同步重定向这些操作请求到

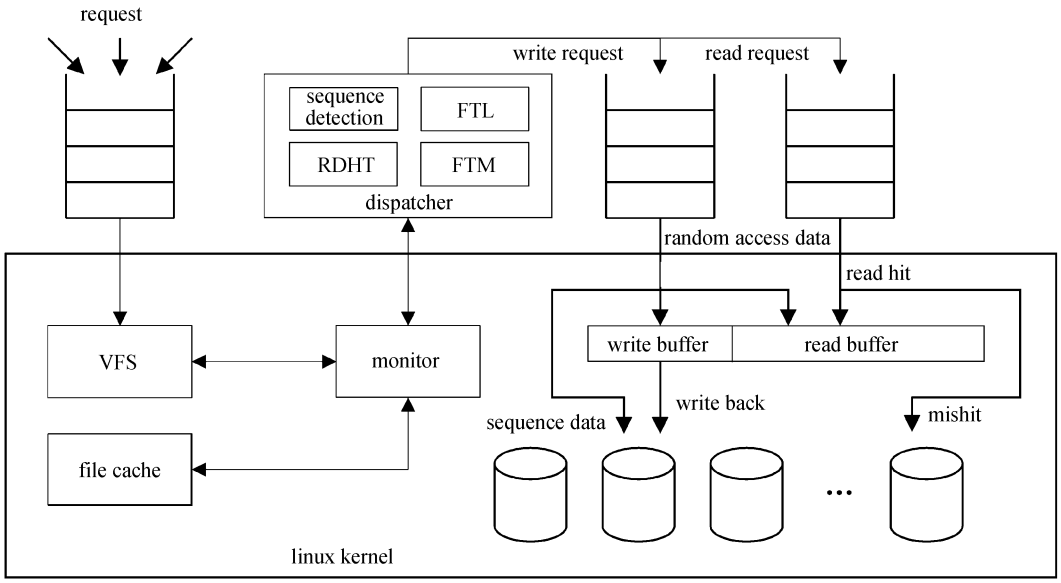


图 1 RAF 系统结构

Fig. 1 System architecture of RAF

Dispatcher. 用户态守护程序 Dispatcher 负责维护固态盘的元数据信息, 并分配读写操作到固态盘或磁盘. RAF 同样动态分配固态盘为读区和写区.

1.1 Monitor 模块

Monitor 截获块级别的请求, 并将其同步重定向到 Dispatcher. Monitor 监控 Linux 系统的文件缓存, 其目的在于把驻留在内存中即将被置换掉的随机数据拷贝到固态盘中. 传统的做法是固态盘与内存同步缓存数据, 本文策略与其相比有两种好处. 第一, 当内存中缓存数据被修改后, 固态盘不会进行无谓的同步更新; 第二, 在内存中被置换的随机数据一定是没有被修改过的, 这意味着这些数据在固态盘中被修改的可能性同样很小.

1.2 Dispatcher 模块

Dispatcher 执行了 RAF 的绝大多数功能, 例如随机数据侦测、元数据管理、固态盘读区和写区的划分和请求重定向等.

1.2.1 随机存取数据侦测

在图 1 中, Dispatcher 的序列侦测模块是识别随机访问数据及记录的核心. 通过 Dispatcher, 数据块总数小于阈值的请求序列将被归结为随机存取数据并记录下来.

序列侦测的基础是逻辑块地址 LBA (Logical Block Address). LBA 是文件系统中数据块存放地址的常见表示方法. 当逻辑连续的 LBA 地址被转换为物理地址, 其物理空间连续性被尽可能地保留. 也就是说, LBA 地址连续意味着物理空间上连续. 这样, 一个序列被定义为一定时间段内, 被访问的 LBA 地址连续的一组数据块.

一个序列并不容易被直接侦测出来, 这是因为很多请求可能相互交织在一起从而打散连续性. 例如, 一个数据块访问序列为 20 057, 57, 9557, 58, 59, 6459, 并且定义一个顺序序列必须包含 3 个数据块, 否则就是随机存取数据. 常规方法将会识别出 5 个随机访问序列, 包含 {20 057}, {57}, {9557}, {58, 59}, {6459}. 这意味着服务这些访问的寻道距离为 $|57-20\,057|+|9557-57|+|58-9557|+|6459-59|=45\,399$, 而且其寻道次数为 5 次. 在混合存储结构中, 这些随机存取数据都将被缓存在固态盘中. 然而, 通过将顺序序列中掺杂的随机序列提取出来, RAF 能够侦测出 4 个访问序列, 即 {20 057}, {9557}, {57, 58, 59}, {6459}, 这意味着平均磁头寻道距离为 $|9557-20057|+|57-9557|+|6459-59|=26400$, 固态盘只需缓存一半的随机数据块 20 057, 9557 和 6459, 而顺序访问序列 57, 58, 59 将由磁盘直接服务.

数据结构 Sequence Descriptor (SD) 用来代表与一个序列相关的内容, 包含若干描述属性, 例如最近请求的地址、修改时间戳和数据块计数器. 这里

设置了一个叫做 randomThreshold 的参数作为阈值来辨别随机序列和顺序序列. 任何数据块计数器值小于 randomThreshold 的序列将被归结为随机序列. randomThreshold 的值可被调控来支持使用积极的或保守的侦测方式. 当一个新的请求到达时, 其数据块起始地址的 LBA 被用来与所有 SD 的最近请求地址属性进行比较. 如果某一个 SD 的最近请求地址被扩展, 则这个新请求属于一个原有的序列. 这个原有序列 SD 中的数据块计数器属性增加新请求的数据块数, 时间戳属性更新为当前时间, 并更新最近请求的地址属性. 如果没有扩展原有的 SD, 意味着这是一个新队列的开始, 需要建立一个新的 SD 对象来描述该请求. 当内存中的 SD 总数达到上限时, 那些时间戳长时间没有被更新的 SD 将会被销毁. 而其中如果 SD 的数据块计数器属性值小于 randomThreshold, 其包含的所有数据块将被认定为随机数据, 其地址将被 RDHT 记录.

1.2.2 RBHT

RAF 通过哈希表 (Random-access Block Hash Table, RBHT) 来记录文件缓存中的随机访问数据地址. Dispatcher 建立序列通过请求队列. 当随机序列被 Dispatcher 侦测出来, 所有的数据块的 LBA 地址都将被记录在 RBHT 中.

RBHT 对于随机数据识别非常重要. 例如当一个内存页被回收时, 其对应的 LBA 地址在 RBHT 中被检索. 如果它被记录, 则这个页内的数据将被拷贝到固态盘的读区; 否则, RAF 认为这些数据是连续数据从而不予处理.

RBHT 带来的额外开销是可以接受的, 因为随机数据的数据量仅限制在文件缓存中, 所占存储空间很小. 另外, 相对于 I/O 操作, 对 RBHT 遍历所消耗的时间是可以忽略不计的.

1.2.3 闪存转换层

闪存转换层 (Flash Translation Layer, FTL) 维护了固态盘中逻辑地址 LBA 到固态盘物理地址的一个映射表 (Flash Mapping Table, FMP). 在 FMP 中, LBA 字段指向磁盘数据块的逻辑地址, 物理地址字段指向该数据块在固态盘中所处的物理地址. FTL 同样还要提供一些固态盘内部功能, 例如擦除操作和垃圾回收. RAF 中仅使用了简化的 FTL 通过使用一个全连接的映射. 因此, 其 FTL 提供更高的效率, 但是消耗了更多主存空间来容纳该映射表.

1.2.4 固态盘读写区划分

固态盘的读写操作速度是不平衡的. 写操作的响应时间一般为读操作响应时间的 4~5 倍. 另外, 固态盘除了读写操作之外, 还有一种耗时的擦除操作. 该操作在固态盘的块级别执行, 只有擦除过的块, 其中的页才可以进行写操作. 固态盘的写操作采用异地更新策略, 这样会产生很多无效页, 从而需要通过垃圾回收来处理散布的无效页. 总体而言, 固态盘读操作代价小于写操作, 尤其是随机的小量写操作.

与内存不同, 读写操作的命中对于固态盘意味着不同的代价, 因为写命中将会产生无效页, 从而需要垃圾回收机制来回收这些无效页. 此外, 由于 RAF 仅缓存随机数据, 即写命中的数据量相对较小, 而这是固态盘最不期望的. 因此, RAF 把固态盘划分为读区和写区来分别负责处理读写操作. 使用写区来专门负责写操作, 这样垃圾回收会限制在写区内. 即垃圾回收时检查的块数目减少, 从而可减少开销.

固态盘的读区和写区具有不同的特点. 读区只容纳随机访问数据, 这些数据是从文件系统的文件缓存替换出来的. 因为只缓存随机数据, 从而减少了固态盘的损耗, 同时由于经过在文件缓存中的过滤, 可以减少写命中. 写区以协同型的循环日志方式来处理写操作, 其目标在于使写操作速度更快, 减少系统的响应时间, 并简化垃圾回收开销. 与读区相似, 写区只缓冲随机访问数据. 写区中被写入的数据随后将被写回到磁盘.

(1) 读区域

读区域仅缓存随机访问数据. 随机访问数据在将要被 DRAM 中的文件缓存置换掉时, 被拷贝到固态盘的读区域. 这样做的目的在于减少写命中. 写命中意味着由于异地更新会在固态盘中产生无效页, 要回收这些无效页时, 就要使用垃圾回收机制来拷贝有效页和擦除块, 从而带来更大的开销.

读区域置换策略基于长期随机访问频率. 在新的随机访问数据被缓存到固态盘时, 读区在没有空余空间时要淘汰现有缓存的数据. 具体来说, 固态盘中每一个单元的随机访问命中频率将会被记录, 随机命中频率最低的数据将会被置换掉.

(2) 写区域

写区的工作方式是协同型的循环日志. 其吸收了磁盘系统的随机写小数据操作, 然后把缓冲的数据写回到磁盘系统. 之所以只在写区缓冲随机的小

数据, 是由于磁盘系统的连续写操作性能不亚于固态盘. 如果把固态盘作为写回 (write-back) 类型的缓存, 吸收磁盘写操作带来的收益并不能弥补写小数据操作给固态盘带来的性能影响.

写区的存储空间需求应与负载的需求动态匹配. 写区作为可以永久保存的短期数据缓冲区, 只需占用满足写操作峰值需求的存储容量即可. 而且, 采用循环型日志模式不需要清除零星的无效页, 从而降低了垃圾回收的额外开销.

写区的损耗必须被重视, 因为在一个相对很小的区域吸纳大量的写操作, 会造成频繁的擦除操作, 进而缩短固态盘的寿命. 由于固态盘的可靠性随着擦除次数的增加而降低, RAF 就周期性地在固态盘中变换位置, 从而均摊了整个固态盘的损耗.

2 试验

这里将讨论 RAF 的实现, 并与当前主流的固态盘磁盘缓存架构 FlashCache 进行比较. 与 RAF 不同, FlashCache 和 DRAM 中文件缓存同步缓存数据, 并按照写回 (write-back) 方式来处理写操作.

2.1 试验平台搭建

在子午工程数据中心的一个台式机上构建了 RAF 的原型系统. 该机配备有 1 个 2 核 CPU, 主频 2.2 GHz, 内存 2 GB, 1 个 Mtron 固态盘和 4 个希捷硬盘, 运行内核版本号为 2.6.30.10 的 Linux 操作系统. 仅使用了固态盘中的 1 GByte 空间. 另外, 4 个硬盘被组织为 RAID-5 模式, 其条带大小为 128 KByte.

2.2 负载

在试验中, 同时使用了人工合成和子午工程数据中心的观测数据、产品数据及安全保障系统产生的日志数据评估 RAF 性能. 这些负载的特征列于表 1. W1 中随机写操作占主导地位, W2 有大量的顺序写操作. W3 是一个综合型的负载, 其中 20% 是顺序的, 30% 是局部的, 50% 是完全随机的访问. Web 是子午工程门户网站访问的观测数据和产品数据以及安全保障系统所收集的日志数据.

3 结果分析

这里使用系统平均响应时间和平均擦除率来比较基于固态盘的磁盘缓存架构性能. 其中, 系统平均响应时间包含服务时间和排队时间, 平均擦除率表征了固态盘损耗的程度.

图 2 和图 3 给出了试验结果. 图 2(a) 考查了不同负载下两种架构的系统平均响应时间比较. 从图 2(a) 中可以看出, RAF 相对于 FlashCache 的系统平均响应时间减少了 17%. 虽然 W1 和 W2 都是以写操作为主的负载, RAF 相对于 FlashCache 的增进效果却差距很大, 推测这可能源于不同的顺序读比例. 由于 W2 的顺序读比例更高, 这样固态盘可与磁

表 1 负载的特征

Table 1 Loads feature

trace	reads /(%)	sequentiality /(%)	average transfer size /KByte	rate /ms
W1	9	10	42	322
W2	11	70	17	80
W3	67	20	8	36
Web	99	14	15	10

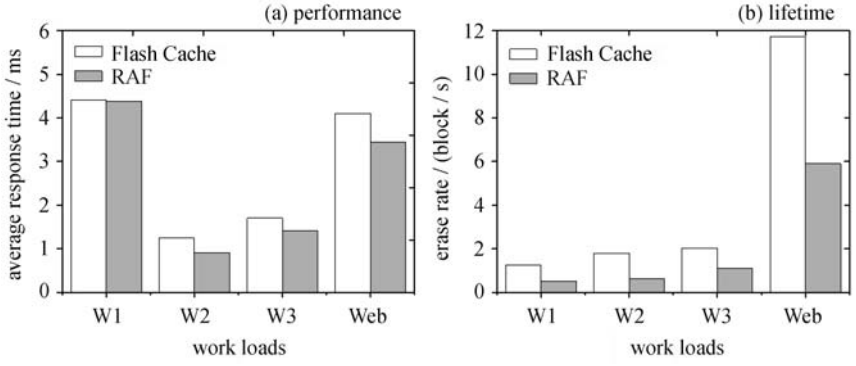


图 2 性能和寿命分析

Fig. 2 Performance and service life analysis

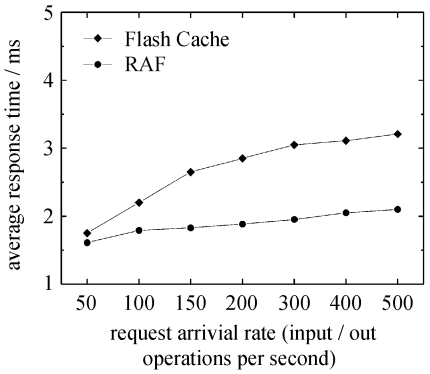


图 3 负载 W3 在不同 I/O 强度下的性能比较

Fig. 3 Performance comparison for W3 under different I/O loads

盘共同分担负载, 达到更好的并行处理效果. 图 2(b) 显示, RAF 的平均擦除率比 FlashCache 结构平均减少了 53%. 综上表明, RAF 对于固态盘的损耗更小, 可以极大地延长固态盘的寿命. 以上表明 RAF 可以在延长固态盘寿命的情况下提升系统的性能.

考虑系统负载的爆发性特征, 通过加速日志的重放速度模拟这种情况. 图 3 给出了逐步增强的 I/O 强度下系统响应时间在两种架构下的比较. 从图 3 可以看出, RAF 在剧烈的 I/O 强度下表现依然稳健, 其系统响应时间的增长远小于 FlashCache. 推测其原因, RAF 倾向于在磁盘与固态盘之间分担负载, 而 FlashCache 倾向于让固态盘尽可能地吸纳磁盘的负载, 导致固态盘由于负载过重性能急剧下降, 从而影响整个系统的性能.

4 结论

提出了一种基于固态盘的磁盘缓存架构 RAF, 在提高系统性能的同时延长固态盘的寿命. 在该架构中, 固态盘被分为读区和写区分别服务于读写请求. 读区仅缓存那些被侦测到并从文件缓存中置换出的随机数据来增加读命中并减少写命中. 写区作为一种循环写通日志来工作, 从而增加写入速度并减少垃圾回收开销.

在 Linux 环境中实现了 RAF 的原型系统, 性能和损耗速度在多种负载下进行了测试. 试验结果表明, RAF 相对于主流的 FlashCache 结构, 可以减少系统平均响应时间 17%, 平均减少损耗率 53%.

研究结果表明, 固态盘作为磁盘缓存不应过度吸收磁盘的负载. 合理地区分随机访问数据和连续访问数据并分别将其交由固态盘和磁盘处理, 有助于增进系统性能并延长固态盘的寿命. 在今后的研究中, 将进一步分析固态盘和磁盘并行工作的互补关系.

RAF 在子午工程观测数据及系统日志的访问试验结果表明, 该架构可以应用到数据中心服务器存储系统中, 提高存储系统的性能.

参考文献

[1] Kgil T, Roberts D, Mudge T. Improving NAND Flash Based Disk Caches [C]//Proceedings of the 35th International Symposium on Computer Architecture. Beijing: IEEE Computer Society Washington, 2008. 327-338

[2] Kgil T, Mudge T. FlashCache: A NAND flash memory file cache for low power web servers [C]//CASES'06: Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems. Seoul: ACM New York, 2006. 103-112

[3] Wu M, Zwaenepoel W. eNVy: A non-volatile, main memory storage system [C]//Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems. San Jose: ACM New York, 1994. 86-97

[4] Narayanan D, Thereska E, Donnelly A, *et al*. Migrating server storage to SSDs: Analysis of tradeoffs [C]//Proceedings of the 4th ACM European conference on Computer systems. Nuremberg, Germany: ACM New York, 2009. 145-158

[5] Lee S, Moon B. Design of flash-based DBMS: An in-page logging approach [C]//Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data. Beijing: ACM New York, 2007. 55-66

[6] Gupta A, Kim Y, Urgaonkar B. DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings [C]//Proceeding of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems. Washington DC: ACM New York, 2009. 229-240

[7] Uysal M, Merchant A, Alvarez G A. Using MEMS-based storage in disk arrays [C]//Proceedings of the 2nd USENIX Conference on File and Storage Technologies. San Francisco, CA: The USENIX Association, 2003. 89-101

[8] Marsh B, Douglis F, Krishnan P. Flash memory file caching for mobile computers [C]//Proceedings of the 27 Hawaii International Conference on System Sciences. Maui, HI: IEEE Computer Society, 1994. 451-460