分布式存储系统中的数据高效缓存方法

杨青霖，吴桂勇，张广艳
清华大学计算机科学与技术系，北京 100084

摘要
针对典型分布式存储系统存在的写放大、I/O路径过长、响应时延较高等问题，提出了一种基于SSD的分布式存储系统中数据高效缓存方法，采用读写旁路和懒惰缓存的缓存管理策略，以及兼顾最近访问时间和历史访问频率的缓存替换策略，并根据前台工作负载的变化情况，自适应地调整主动回刷脏数据的速率，显著提升了存储系统的读写性能。

关键词
分布式存储；分布式数据缓存；读写旁路；懒惰缓存；替换策略

中图分类号：TP333 文献标识码：A doi: 10.11959/j.issn.2096-0271.2021018

An approach to buffering data efficiently in distributed storage systems

YANG Qinglin, WU Guiyong, ZHANG Guangyan
Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China

Abstract
To address the problems of write amplification, long I/O path, and high access latency in distributed storage systems, an efficient SSD-based caching approach for distributed storage systems was proposed. This approach adopts read/write bypassing and lazy caching methods to manage the cache system, considers last access time and historical access frequency when performing cache replacement, and adjusts the flushing speed according to the foreground workload. It improves significantly the reading and writing performance of storage systems.

Key words
distributed storage, distributed data caching, read/write bypassing, lazy caching, replacement strategy

1 引言

分布式存储系统是由多个存储设备或者服务器（统称为节点）通过I/O总线或者互联网络连接而成的，并通过节点间的数据分散布局实现高效、低廉的数据存储。分布式存储系统因其强大的横向扩展能力在大数据场景中得到广泛应用[1]。然而，以Ceph为代表的分布式存储系统存在以下几个问题。

- 写放大。为了支持写事务，需要将数据先写入日志，然后再应用到本地文件系统中，这导致了双倍的数据写入量，降低了写入性能，且后端文件系统的写入容易成为性能瓶颈。
- I/O路径过长，响应时延较高。一个完整的I/O过程需要经过多种相关线程和模块协同，部分模块之间还需要进行网络传输或内存复制，且过程中存在多处排队等待，这些排队等待有可能阻塞读写请求，过长的I/O路径增大了读写操作的响应时延，降低了存储系统的读写性能。

现有技术中普遍将内存或固态硬盘（solid state disk，SSD）等快速设备作为底层存储系统的缓存，利用数据分层机制提高存储系统的读写性能，如Memcache分布式缓存系统、Flashcache混合磁盘技术，或者在磁盘阵列上层支持SSD客户端缓存等。因此，为了解决分布式存储系统中的性能问题，在以磁盘为介质的存储系统中引入以固态硬盘为介质的缓存是一种可行的途径。

对于单机系统中的SSD缓存技术，现有的研究和应用已经比较成熟。但是，运行在分布式存储系统上的大规模任务通常会被分割为许多小任务分散执行，这就导致数据访问模式呈现弱局部性，若采用传统的缓存机制，容易造成缓存污染的问题。另外，由于分布式场景下缓存管理的粒度通常较大，对数据块的缓存操作需要大量的网络带宽和磁盘读写开销，在传统的缓存机制中，缓存操作会落在I/O的关键路径上，增大I/O请求的响应时延。由于容错需要，缓存层和存储层还需要做数据冗余，导致数据缓存和脏数据回刷的开销更大。因此，在分布式场景下，传统的缓存机制并不完全适用，不能发挥SSD等快速设备的优秀特性，不能充分提升分布式存储系统的读写性能。

本文提出一种分布式存储系统中数据高效缓存方法——基于热点检测（hot spot detection，HOSD）的缓存方法，该方法通过充分利用分布式系统访问模式信息来提高缓存池中数据的命中率，最终提高缓存系统的读写性能。首先，HOSD采用读写旁路和懒惰缓存的方法降低了缓存操作对I/O时延的影响，且避免了缓存污染；其次，HOSD采用高效的缓存管理机制及缓存替换策略降低了缓存管理开销，提高了对缓存中冷数据的识别精度，使得更多的热数据得以保留在缓存中，提高了缓存效率和分布式场景下弱局部性访问的I/O性能；最后，HOSD通过跟踪前台工作负载的变化情况自适应地调整主动回刷脏数据的速率，以避免在发生缓存替换时密集回刷数据对系统I/O性能造成很大影响。

2 系统架构设计

HOSD的基本思想是冷热数据分离，用相对快速的固态存储设备组成一个热数据缓存池，后端用相对慢速的磁盘设备组建冷数据存储池（如**图1**所示）。HOSD在分布式存储系统内部支持数据缓存，数据在缓存池与存储池之间的迁移由HOSD软

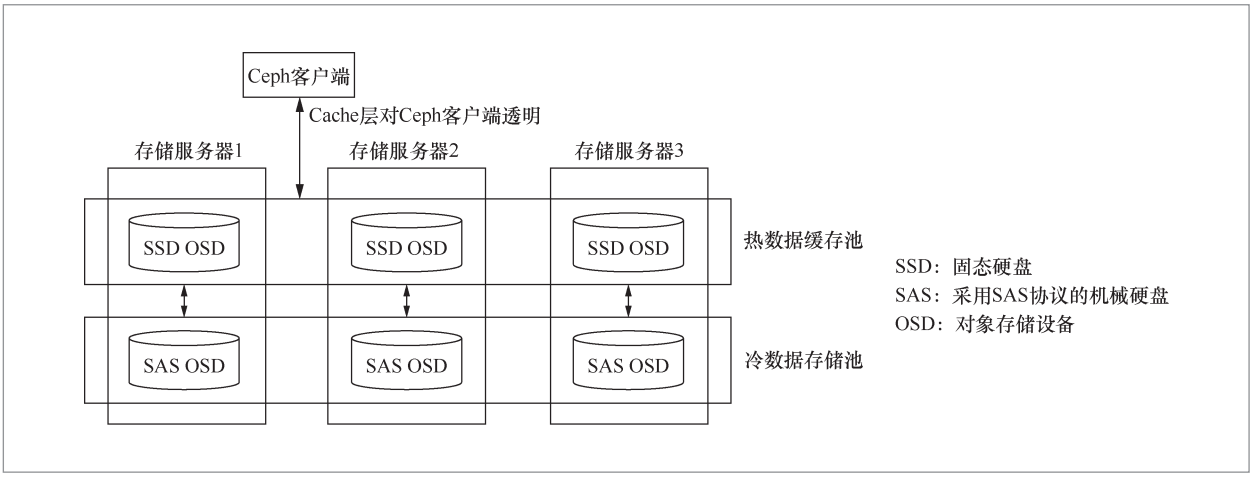


图1 缓存方法的软件逻辑架构

件模块控制，这个过程对客户端透明，因此无须在提供块接口、文件接口或对象接口的客户端上进行改动。

HOSD主要包括以下软件模块（如图2所示）。

- Objecter模块：属于客户端，负责将读写请求路由到缓存池或存储池。
- Filter模块：根据数据的访问热度，判断是否将数据调入缓存池。
- Promotion模块：负责将存储池中的数据调入缓存池。
- Agent模块：负责将脏数据适时回刷到存储池，或者将冷数据从缓存池中剔除出去。

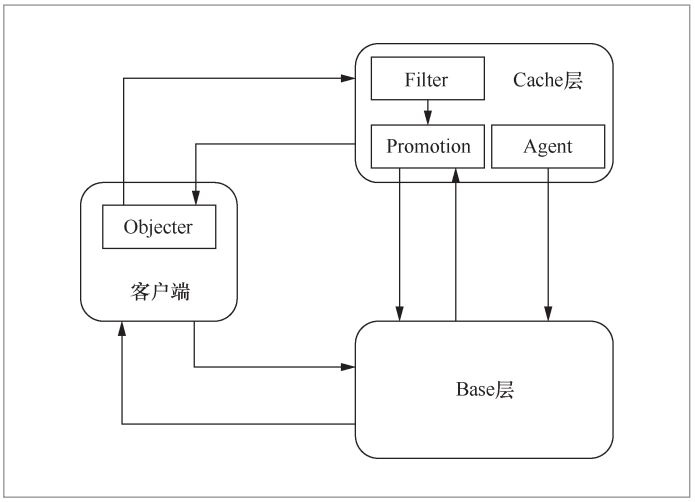


图2 缓存方法的软件模块设计

分布式存储系统中数据缓存方法HOSD的工作流程如下。当客户端要读写的数据在缓存池命中时，则直接在缓存池进行读写，不会访问相对慢速的存储池。若未命中，则旁路读写请求，将请求转发给存储池，让客户端在存储池中进行读写。另外，由Filter模块负责热数据发现，判断该数据是否被频繁访问，若是，则由Promotion模块将数据迁移到缓存池；否则，该数据进入缓存候选状态，暂不调入缓存池。当缓存池的占用率达到一定阈值时，Agent模块负责将脏数据回刷到存储池，或者将冷数据剔除出缓存池。

由上述HOSD缓存方法的工作流程可以看出，其缓存管理主要涉及3种数据交换操作，分别如下。

- 调入缓存池：客户端读写时，依据HOSD调入算法将数据从存储池迁移到缓存池中。
- 脏数据回刷：对于写入缓存池中的数据，经过一定的时间，HOSD Agent模块会将这部分数据回刷到存储池中，以免缓

存池中脏数据过多。

● 冷数据替换：在缓存池装满的情况下，HOSD Agent模块会将缓存池中的一些数据块剔除出去，让新的数据可以调入进来。

上述数据交换操作会带来大量存储和网络开销，因此希望通过设计高效的缓存机制来提高缓存效率，并减少这种数据流动开销。

3 分布式存储系统缓存方法

本文提出的分布式存储系统中数据高效缓存方法HOSD主要包括如下3个技术点：

● 通过读写旁路和懒惰缓存相结合的方法，避免缓存操作落在I/O路径上，增大I/O请求的响应时延，且通过热点数据发现的方法来选择性地将数据调入缓存，提高缓存中的数据质量，避免缓存污染；

● 通过高效的缓存管理机制，减少缓存管理操作带来的额外磁盘访问和计算开销，并采用兼顾最近访问时间和历史访问频率的缓存替换策略来提高替换操作的准确性，使得更多的热数据得以保留在缓存中，提高数据访问的缓存命中率；

● 通过跟踪前台工作负载的轻重程度，自适应地调整主动回刷脏数据的速率，减少缓存替换时的脏数据回刷，并减轻数据回刷操作对系统读写性能的影响。

前两个技术点保证了缓冲池中的数据质量，提高了缓存命中率和系统读写性能；第三个技术点保证了缓存方法能够尽可能地使用富余的网络和磁盘带宽来管理缓存，降低缓存管理操作对系统性能的影响。

3.1 读写旁路与懒惰缓存

传统的缓存机制采用按需调入策略，即一旦发现缓存缺失，就将对应的数据调入缓存，再响应客户端的读写请求。这样做，一方面会使缓存操作落到I/O路径上，增加I/O时延；另一方面，该数据块可能只是被偶发性地访问，后续不会被频繁地访问，那么，将该数据块调入缓存就会造成缓存污染，且带来大量开销，还会导致一些原本在缓存中较热的数据块被替换出去。因此，HOSD缓存方法采用读写旁路和懒惰缓存相结合的方法来解决上述问题，从而保证缓存池中的数据质量。

HOSD处理读写请求的主要流程如**图3**所示，当一个I/O请求达到时，如果对应数据块在缓存池中，则由缓存池直接响应用户请求；如果对应数据块不在缓冲池中，则HOSD会将该请求转发给存储池，笔者把这种处理请求的方式叫作读写旁路。对于缓存缺失的数据块，HOSD采用懒惰缓存策略。具体来讲，就是预先定义一个数据重用距离和缓存阈值，当该数据块在重用距离内被访问的次数超过了设定的缓存阈值时，将其调入缓存。另外，数据重用距离也可以根据缓存的命中情况来动态调整。

通过读写旁路和懒惰缓存的方法，HOSD让缓存操作异步进行，避免了缓存操作对I/O时延的影响，同时减少了缓存池和存储池之间的数据流动开销。事实上，按需的缓存调入策略相当于缓存阈值为1的特例，HOSD通过增大迁移阈值来延缓对数据块的缓存，最终提高缓存池中数据块的命中潜力。

3.2 高效的缓存管理机制和替换策略

HOSD缓存管理机制的工作流程如**图4**所示，通过在内存中维护一个将访问热度作为优先级的序列来对缓存池中的数据进行排列，当缓存的占用率达到一定的阈值

时，通过缓存替换，将访问热度较低的数据块从缓存池中剔除出去。同时，当发生读写请求时，需要对数据块序列进行更新，为了避免这部分开销对I/O时延造成影响，需要对缓存中的数据做逻辑划分，也就是在内存中维护多个缓存序列，以此来减小锁粒度，降低缓存管理带来的额外开销。

缓存信息维护在内存中，如果发生节点故障，会造成缓存信息的丢失。因此，HOSD周期性地将缓存信息打包成一个对象，通过存储系统现成的写入逻辑，将缓存信息以检查点的形式持久化到存储系统中。对于检查点之间丢失的缓存信息，HOSD也可以通过存储系统的变更操作的日志信息将其重建出来。通过上述方法，保证HOSD缓存方法在节点故障的情况下仍能正常工作，保证了存储系统的容错能力。

当进行缓存替换时，需要选择访问热度最低的数据块，将其从缓存中剔除，以便为即将调入缓存中的数据块腾出空间。HOSD缓存方法采用基于访问历史的替换策略，综合考虑数据块的历史访问频率和最近访问时间，并构造了两级缓存，一级缓存完整的数据，另一级仅缓存元数据，避免了由缓存量不足导致的缓存预测不准确的问题，保证了从缓冲池中替换出去的数据块都具有较低的命中潜力。

HOSD缓存替换策略的主要数据结构由4个链表构成（如图5所示）：MRU（most recently used）链表、MFU（most frequently used）链表、MRUG（MRU ghost）链表、MFUG（MFU ghost）链表。

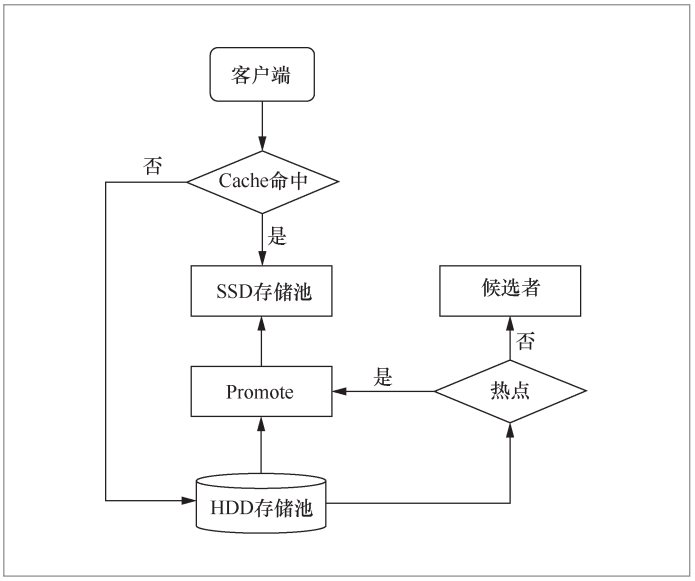


图 3 HOSD 缓存方法读写流程

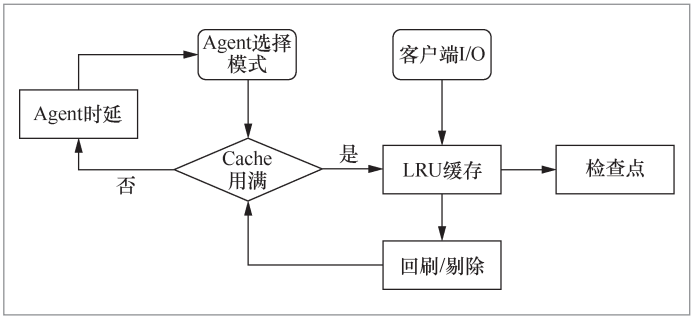


图 4 HOSD 缓存管理机制工作流程

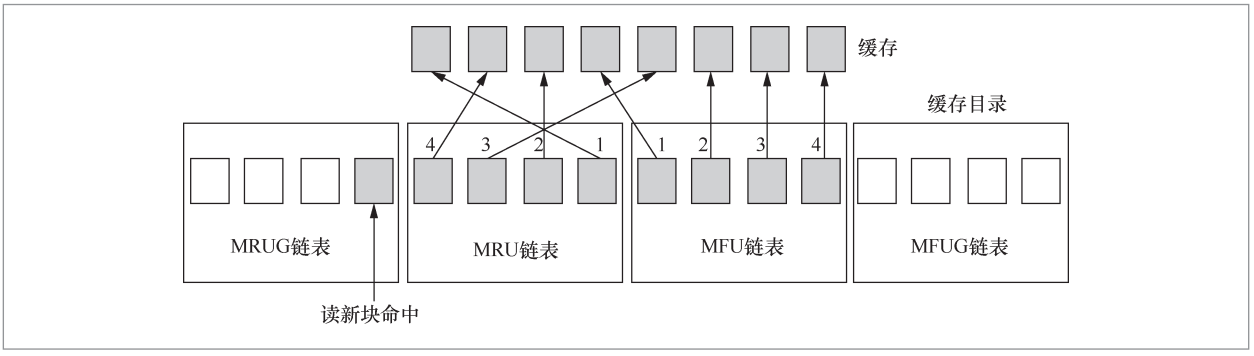


图 5 HOSD 缓存方法替换策略

其核心思想如下。

- 当数据块进入缓存时，HOSD先将数据放入MRU链表中。MRU链表是依据数据块的访问时间进行排序的一个有限序列。当一个新的数据块进入MRU链表时，队列LRU（least recently used）端的数据块将被替换出去。如果MRU链表中的某个数据块在被替换之前被二次访问，那么将该数据块放入MFU链表中的MFU端。MFU链表也是依据数据块的访问时间进行排序的一个有限序列。不同的是，每发生一次二次命中，都是把MFU链表中的对应数据块放到MFU链表头部。
- 如果有数据需要进入缓存中，而此时缓存中的数据块数目已经到了设定的阈值，HOSD会从LRU和LFU（least frequently used）端删除元素，并将对应的元数据信息送入ghost链表中，即MFUG链表和MRUG链表。MFUG链表和MRUG链表不存储数据，只存储数据块的访问记录。将MFU链表的数据送入MFUG链表中，同时释放该数据块在MFU链表中占用的存储空间；如果释放的数据块在MRU链表中，那么将该元素从中删除，并送入MRUG链表中。
- 当再次访问该数据块时，数据如果在MRUG链表或者MFUG链表中，那么从存储池读取该元素，并重新将其插入MRU链表或MFU链表中。HOSD会根据MRUG链表或者MFUG链表中发生伪命中次数的多少，动态地调整MRU或MFU两个链表应包含的元素的个数。

HOSD缓存替换策略在传统算法上构造了一层元数据缓存，能够较快地读取刚被替换的缓存，并兼顾数据块的最近访问时间和历史访问频率信息，可以准确识别出缓存中访问热度较低的数据块，将其替换出缓存池，从而保证缓存中的数据块都具有较高的命中潜力。此外，在真实系统中实现的HOSD缓存策略支持多线程访问。

3.3 自适应的主动回刷机制

分布式存储系统中的数据回刷操作和前台应用会竞争系统中的I/O、网络和CPU等资源。为了减小对前台性能的影响，HOSD采用主动回刷的策略，并根据应用负载自适应地调整回刷速率。目标包括两个方面：一是让系统将缓存池中的脏数据尽快回刷到存储池，二是减少回刷操作过程对前台应用的影响。

当缓存中的数据量达到一定的阈值时，HOSD周期性地主动将缓存中较冷的脏数据回刷到存储池，而不是等到替换发生时再回刷。自适应的主动回刷机制需要跟踪系统中的应用负载，进而动态调整数据回刷的速率，实现回刷操作快速完成，并对前台应用性能影响较小。

在回刷速率控制方面，HOSD采用一个开/关逻辑阀来进行调控（如**图6**所示）。是否在下一时间周期内执行数据回刷操作取决于当前时间周期内负载的轻重。如果当前时间周期内负载较轻，则下一时间周期执行数据回刷操作；反之，下一周期不执行数据回刷操作。也就是说，利用当前周期内的负载来近似估算下一周期的负载，因此时间周期不能设置得太长。

4 性能测试

4.1 测试环境及方法

HOSD是基于Ceph 0.94.2开发的缓存管理系统，为了测试HOSD原型系统的性能，本文构建了一个分布式存储系统。其中，存储服务器通过专用的高速存储网络连接起来。每台机器都有一定数量的硬盘

驱动器(hard disk drive, HDD)用于构建存储池，以及一定数量的固态硬盘用于构建缓存池。

实验所用的测试环境描述如下：实验过程中共使用了3台机器，每台机器配有2个E5-2609V2 CPU、4个 DDR3 RECC 8 GB内存、6块ST2000NM0033 2T SASA磁盘、2块Intel S3500 480 GB固态硬盘、一块Intel S3500 120 GB固态硬盘、一块LSI 9260-8IRAID卡以及一块Intel X520-SR2万兆网卡，每台机器均安装了Redhat 7.0操作系统。这些机器通过万兆光纤交换机连接起来，形成一个高速集群存储系统。

通过对比HOSD和Ceph自带的分布式缓存系统Cache Tiering的性能，评价HOSD的性能。通过播放有代表性的读写请求trace，收集系统的每秒输入输出量(input/output per second, IOPS)和读写带宽等数据。实验采用典型的I/O测试工具FIO(flexible I/O tester)生成有代表性的工作负载。FIO能生成具有不同特征的工作负载，特征参数包括读写比例、传输请求大小和同时发出请求的最大数目等。

4.2 测试结果

4.2.1 人工负载下的性能

第一组实验使用FIO默认的均衡随机负载进行测试。实验中磁盘存储池大小为200 GB，SSD缓存池大小为40 GB，Ceph对象大小设置为1 MB，FIO播放负载所用的iodepth参数值设为128。分别将请求大小参数设置为4 KB、1 MB和4 MB，分别测试Cache Tiering和HOSD的IOPS及读写带宽。

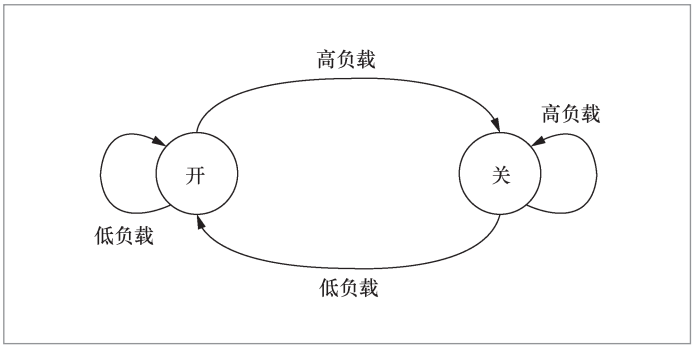


图 6 HOSD 缓存脏数据回刷控制机制

图7统计了每种情况下HOSD相对于Cache Tiering的加速比，加速比在2.52和5.96之间波动。为了统计HOSD的平均加速效果，笔者计算了各个加速比的几何平均值，以减小极端数据对平均值的影响。可以得到，HOSD带来高达3.48的平均加速比。因此，相对于Cache Tiering，HOSD带来了明显的性能提高。这种性能提升的原因是：Cache Tiering将大量冷数据调入缓存，既占用了缓存空间，又增加了系统的I/O开销；HOSD则通过懒惰缓存的方法，保证了缓冲池中的数据具有较高的命中率，并通过自适应的主动回刷机制，保证了分布

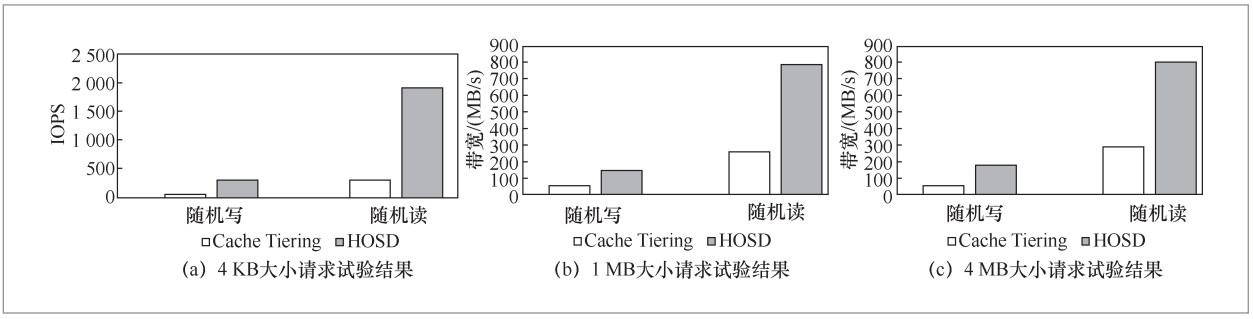


图 7 FIO 随机负载下的 I/O 性能

式缓存系统具有较低的维护开销。

4.2.2 真实负载下的性能

为了在真实的磁盘I/O负载下比较Cache Tiering和HOSD的性能，第二组实验使用FIO同时播放多个trace进行测试。实验中磁盘存储池大小为100 GB，SSD缓存池大小为8 GB，Ceph对象大小设置为1 MB，FIO播放负载所用的iodepth参数值设为64。同时播放的trace包括prxy_1、wdev_0、prxy_0、webmail和hm_0，分别测试Cache Tiering和HOSD的IOPS。

表1统计了第二组实验中每种情况下HOSD相对于Cache Tiering的加速比，加速比在1.01和1.13之间波动。为了统计HOSD的平均加速效果，笔者计算了各个加速比的几何平均值。可以看出，HOSD带来高达1.08的平均加速比。因此，相对于Cache Tiering，HOSD带来了一定的性能提高。

第二组实验的平均加速比相对第一组实验明显下降，原因分析如下：笔者统计了实验所用的5个I/O trace，发现每个trace的访问都局限在一个较小的地址空间。这样带来的结果是无论使用Cache Tiering还是HOSD，绝大多数I/O请求会在SSD缓存区命中。在命中的情况下，二者性能几乎没有差距。不同的是，HOSD维护缓存区中数据的数据交换操作成本要比Cache Tiering低一些，且对访问时间序列的维护更加精确，因此带来了8%的性能提高。

表1 播放真实trace负载下的I/O性能

trace名称	测量指标	Cache Tiering	HOSD	加速比
prxy_1	读IOPS	2 248	2 542	1.13
	写IOPS	1 185	1 341	1.13
wdev_0	读IOPS	343	373	1.09
	写IOPS	1 326	1 469	1.11
prxy_0	写IOPS	1 568	1 668	1.06
webmail	读IOPS	525	570	1.09
	写IOPS	1 143	1 159	1.01
hm_0	读IOPS	425	438	1.03
	写IOPS	1 079	1 122	1.04
几何平均值				**1.08**

5 相关工作

随着闪存的日益普及，无论是工程师还是研究人员，都对将其集成到存储栈中表现出较大兴趣。硬盘制造商支持混合磁盘已有多年。主流操作系统上也已经有如ReadyBoost和bcache的软件实现。Facebook将固态盘作为数据中心磁盘进行高速缓存。Solaris ZFS支持使用固态盘作为二级缓存[2]。NetApp公司也使用固态硬盘来加速存储系统[3]。

早期的一些研究主要关注如何改进基于闪存的高速缓存。Kgil T等人[4]提出将闪存作为内存扩展或磁盘缓存来减少功耗。他们采用能够改变纠错码（error correction code，ECC）强度和闪存芯片单元存储密度的可编程闪存控制器，能够在保证性能牺牲不到5%的情况下延长闪存使用寿命。Mesnier M P等人[5]利用从文件系统和数据库管理系统获得的语义信息改进基于SSD的磁盘缓存，通过对元数据块和小文件赋予更高的优先级，能够获得比不考虑语义的缓存算法更好的性能。Pritchett T等人[6]将SSD作为多个服务器的全局高速缓存，以便最大化它的利用率。他们用更少的SSD设备获得更高的命中率，比为每个服务器单独配置缓存更加高效。同时他们还通过懒惰缓存机制来延长SSD设备的寿命。

近期的研究工作[7]利用内部垃圾回收行为提高了固态硬盘的缓存性能。垃圾回收操作对SSD的性能和寿命有较大的影

响。研究人员将SSD分为3个区域（未使用区域、读区域和写区域），动态调整这些区域的大小，在提高命中率的同时，减少了擦除操作次数。同时，SSD也可以作为数据库管理系统的扩展缓冲池。参考文献[8]提出了基于访问频率的替换策略——温度感知缓存（temperature-aware caching，TAC）。参考文献[9]在Microsoft SQL Server上实现了一个专用模型，作者还评估了3个不同的将脏页替换出内存缓存池的策略。

除了缓存，存储分层[10-12]是另一个热门的混合存储技术。一个分层存储系统会动态选择热门的数据子集，并将它们移动到速度更快但容量较小的设备上。这项技术被广泛应用于企业级存储阵列，以满足大容量、高性能与低成本的需求。一个典型例子是，苹果公司通过Fusion Drive推出了消费级分层存储产品。

缓存和分层存储都属于分级存储。作为替代，横向混合存储系统使用SSD存储数据的特定部分，以更好地利用SSD和HDD。这种体系结构背后的想法是一个块的访问模式取决于它存储了什么样的数据。例如，元数据占用的存储空间极小，但是其访问频率较高[13]。因此，将它们固定存储在SSD中可以提高总体性能，同时避免了来回移动数据块的开销。这种架构的例子包括ChunkStash[14]和I-CASH[15]。Wang S C等人[16]发现了HDD读写性能的波动性，从而设计了调度器，在HDD和SSD的混合存储中根据HDD的波动情况将I/O调度至HDD或者SSD，在节省SSD存储空间的同时，降低了尾部时延。

Liu Z X等人[17]指出，在大规模分布式存储系统中，负载均衡是影响服务等级目标（service-level objective，SLO）的关键因素，而一个快速的缓存可以保证负载均衡。由此设计了一个两层的缓存系统DistCache，通过两个不同的哈希函数将热数据分散在不同的缓存层，从而提升缓存性能。Zhang Y等人[18]通过记录存储集群中各节点的缓存命中情况来动态调整集群中的缓存分配情况，从而提升总体缓存命中率，减少与底层存储系统的数据交换总量。Berger D S等人[19]针对用户的网络服务存在的长尾时延问题，提出了RobinHood方法，该方法可以动态地将缓存从富余的地方分配至紧张的地方，有效地降低了尾部时延。在纠删码被越来越多地应用于现代存储集群中的情况下，Luo T Q等人[20]提出了使用编码缓存来减少峰值数据传输量的方法。

6 结束语

本文提出了一种分布式存储系统中数据高效缓存方法HOSD，采用读写旁路和懒惰缓存的办法，降低了缓存操作对I/O时延的影响，并减少了对冷数据的缓存，避免了缓存污染情况的发生。同时，采用高效的缓存管理机制降低了缓存管理对存储系统读写性能的影响，并通过兼顾最近访问时间和历史访问频率的缓存替换策略，提高了对缓存中冷数据的识别精度，使得更多的热数据得以保留在缓存中，提高了缓存的效率和分布式场景下弱局部性访问的I/O性能。另外，通过跟踪应用负载轻重而自适应地主动回刷脏数据，减轻了数据回刷操作对系统性能的影响。最后，通过定期对内存中维护的缓存信息进行持久化来保证存储系统和缓存方法的容错能力。实验表明，相对于Ceph自带的Cache Tiering缓存系统，本文所提分布式缓存方法HOSD实现了3.48的平均加速比，带来了明显的性能提升。未来，将进一步探索本文提出的缓

存策略在大规模存储集群及纠删码存储系统中的应用。

参考文献:

[1] 陈游旻, 李飞, 舒继武. 大数据环境下的存储系统构建：挑战、方法和趋势[J]. 大数据, 2019, 5(4): 27–40.
CHEN Y M, LI F, SHU J W. Building storage systems in big data era: challenges, methods and trends[J]. Big Data Research, 2019, 5(4): 27–40.

[2] LEVENTHAL A. Flash storage memory[J]. Communications of the ACM, 2008, 51(7): 47–51.

[3] PETERS M. NetApps solid state hierarchy[Z]. ESG White Paper, 2009.

[4] KGIL T, ROBERTS D, MUDGE T. Improving NAND flash based disk caches[C]//The 35th Annual International Symposium on Computer Architecture. Piscataway: IEEE Press, 2008: 327–338.

[5] MESNIER M P, AKERS J B. Differentiated storage services[C]//The 23rd ACM Symposium on Operating Systems Principles. New York: ACM Press, 2011: 57–70.

[6] PRITCHETT T, THOTTETHODI M. SieveStore: a highly–selective, ensemble–level disk cache for cost–performance[J]. ACM SIGARCH Computer Architecture News, 2010, 38(3): 163–174.

[7] OH Y, CHOI J, LEE D, et al. Caching less for better performance: balancing cache size and update cost of flash memory cache in hybrid storage systems[C]//The 10th USENIX Conference on File and Storage Technologies. New York: ACM Press, 2012: 25.

[8] CANIM M, MIHAILA G A, BHATTACHARJEE B, et al. SSD bufferpool extensions for database systems[J]. Proceedings of the VLDB Endowment, 2010, 3(1–2): 1435–1446.

[9] DO J, ZHANG D H, PATEL J M, et al. Turbocharging DBMS buffer pool using SSDs[C]//The 2011 International Conference on Management of Data. New York: ACM Press, 2011: 1113–1124.

[10] GUERRA J, PUCHA H, GLIDER J, et al. Cost effective storage using extent based dynamic tiering[C]//The 9th USENIX Conference on File and Storage Technologies. Berkeley: USENIX Association, 2011: 20.

[11] CHEN F, KOUFATY D A, ZHANG X D. Hystor: making the best use of solid state drives in high performance storage systems[C]//The International Conference on Supercomputing. New York: ACM Press, 2011: 22–32.

[12] APPUSWAMY R, VAN MOOLENBROEK D C, TANENBAUM A S. Integrating flash–based SSDs into the storage stack[C]//2012 IEEE 28th Symposium on Mass Storage Systems and Technologies. Piscataway: IEEE Press, 2012: 1–12.

[13] WANG A I A, REIHER P L, POPEK G J, et al. Con–quest: better performance through a disk/persistent–RAM hybrid file system[C]//The 2002 USENIX Annual Technical Conference. New York: ACM Press, 2002.

[14] DEBNATH B, SENGUPTA S, LI J. ChunkStash: speeding up inline storage deduplication using flash memory[C]//The 2010 USENIX Conference on USENIX Annual Technical Conference. Berkeley: USENIX Association, 2010: 16.

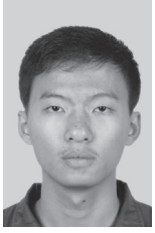
[15] YANG Q, REN J. I–CASH: intelligently coupled array of SSD and HDD[C]//2011 IEEE 17th International Symposium on High Performance Computer Architecture. Piscataway: IEEE Press, 2011: 278–289.

[16] WANG S C, LU Z Y, CAO Q, et al. BCW: buffer–controlled writes to HDDs for SSD–HDD hybrid storage server[C]//The 2020 USENIX Conference on File and Storage Technologies. Berkeley: USENIX

Association, 2020: 253–266.
[17] LIU Z X, BAI Z H, LIU Z M, et al. DistCache: provable load balancing for large-scale storage systems with distributed caching[C]//The 17th USENIX Conference on File and Storage Technologies. Berkeley: USENIX Association, 2019: 143–157.
[18] ZHANG Y, HUANG P, ZHOU K, et al. OSCA: an online-model based cache allocation scheme in cloud block storage systems[C]//The 2020 USENIX Conference on USENIX Annual Technical Conference. Berkeley: USENIX Association, 2020: 785–798.
[19] BERGER D S, BERG B, ZHU T, et al. RobinHood: tail latency aware caching – dynamic reallocation from cache-rich to cache-poor[C]//The 13th USENIX Symposium on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2018: 195–212.
[20] LUO T Q, AGGARWAL V, PELEATO B. Coded caching with distributed storage[J]. IEEE Transactions on Information Theory, 2016: 99.

作者简介

杨青霖（1995– ），男，清华大学计算机科学与技术系博士生，主要研究方向为分布式系统、纠删码存储。



吴桂勇（1992– ），男，清华大学计算机科学与技术系硕士生，主要研究方向为分布式系统、存储系统。

张广艳（1976– ），男，博士，清华大学计算机系长聘副教授、博士生导师，主要从事大数据存储与分析的理论和方法研究，包括大数据计算、存储系统与分布式处理等方面。研究得到了国家杰出青年科学基金项目、国家重点研发计划项目、国家973项目和国家863项目等的支持。近年来提出了大规模存储系统构建及访问的方法与关键技术，有效提高了存储系统的性能、扩展性和可用性。发表学术论文40余篇，其中在FAST、USENIX ATC、*ACM TOS*、*IEEE TC*、*IEEE TPDS*等计算机系统领域高水平国际会议和期刊发表论文20余篇。近五年以第一发明人获得美国发明专利授权1项、中国发明专利授权7项。

收稿日期： 2021-01-21
通信作者： 张广艳，gyzh@tsinghua.edu.cn
基金项目： 国家重点研发计划资助项目（No.2018YFB0203902）；国家自然科学基金资助项目（No.62025203）
Foundation Items: The National Key Research and Development Program of China (No.2018YFB0203902), The National Natural Science Foundation of China (No.62025203)