基于SSD-SMR混合存储的LSM树键值存储系统的性能优化

王洋洋 韦皓诚 柴云鹏
(中国人民大学信息学院 北京 100872)

摘 要 大数据对存储系统的可扩展性、性能和成本等方面提出了更高的要求。瓦记录(Shingled Magnetic Recording,SMR)硬盘由于存储密度高、价格便宜,正逐步被广泛应用于大数据存储系统。但是,SMR硬盘的随机写性能较差,与快速的基于闪存的固态硬盘(Solid State Drive,SSD)一起构成混合存储时可以显著提升性能。同时,基于写优化的日志结构合并(Log-Structured Merge,LSM)树的键值存储已被广泛应用于许多NoSQL系统,如BigTable,Cassandra和HBase等。因此,如何基于新型的SSD-SMR混合存储构建出高性能的LSM树键值存储系统是一个具有很大研究价值的问题。首先建立基于SSD-SMR混合存储的LSM树键值系统的性能模型,然后针对SSD和SMR的硬件特征以及LSM树键值存储的软件特点,设计了一套面向SSD-SMR混合存储进行性能优化的LSM树键值存储系统,并基于LevelDB实现了该系统。在仅仅使用0.4%~2%空间的SSD的情况下,所提方法可以使SSD-SMR混合存储方案比普通磁盘方案的随机写性能提高20%,随机读性能提高5倍。

关键词 大数据,日志合并树,瓦记录磁盘,闪存,混合存储
中图法分类号 TP333 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2018.07.009

Performance Optimization of LSM Tree Key-value Storage System Based on SSD-SMR Hybrid Storage

WANG Yang-yang WEI Hao-cheng CHAI Yun-peng
(School of Information,Renmin University of China,Beijing 100872,China)

Abstract Because of the higher requirements on the scalability,performance,and cost for storage systems proposed by big data,Shingled Magnetic Recording (SMR) disks are widely used in big data storage systems due to the high storage density and low cost. However,since the random write performance of SMR disks are usually weak,the hybrid storage consisted of both SMR disks and the fast Flash-based Solid State Drives (SSDs) can promote the performance significantly. Meanwhile,the write-optimized Log-Structured Merge (LSM) Tree-based key-value storage system have been widely used in many NoSQL systems,such as BigTable,Cassandra,HBase,etc. Therefore,how to construct a fast LSM tree key-value storage system based on SSD-SMR hybrid storage is a research problem with great practical significance. This paper first modeled the performance model of LSM tree key-value storage system based on SSD-SMR hybrid storage,and then designed a performance-optimized LSM tree key-value storage system and implemented it based on LevelDB. The evaluation results indicate that the system based on SSD-SMR hybrid storage improves the random-write performance by 20% and improves random-read performance by 6 times coupled with only a very small SSD (i.e.,0.4%~2% of disk capacity) compared with the HDD-based solution.

Keywords Big data,LSM tree,SMR HDD,Flash,Hybrid storage

根据IDC统计,全球数据量将在2020年达到44 ZB(即4400万PB)[1]。例如,欧洲中程天气预报中心(ECMWF)的存储规模在2015年已经达到了100 PB,而且每年以45%左右的速度增长[2]。而在大数据中,随着Web 2.0应用的快速增长和云计算平台的大规模部署[3-5],相当一部分大数据存储于键值(Key-Value,KV)存储系统中。与传统关系数据库相比,KV存储具有更高的可扩展性和效率,更适应大数据的要求。相对于B+树[6-7]或哈希表[8],基于日志结构合并树(LSM树)的KV存储由于具有很好的写性能和范围查询能力而被广泛使用。很多主流的KV存储系统都基于LSM树

到稿日期:2017-07-16 返修日期:2017-11-25 本文受国家重点研发计划"云计算和大数据"重点专项项目(2018YFB1004400),国家自然科学基金重点项目(61732014),北京市自然科学基金面上项目(4172031),中国人民大学预研委托项目(团队基金)(16XNLQ02),计算机体系结构国家重点实验室开放课题(CARCH201702)资助。

王洋洋(1995-),男,硕士生,主要研究方向为分布式系统、大规模存储系统;**韦皓诚**(1993-),男,硕士生,主要研究方向为分布式系统、大规模存储系统;**柴云鹏**(1983-),男,博士,副教授,CCF会员,主要研究方向为分布式系统、大规模存储系统、云存储、Flash存储、节能存储系统,E-mail:ypchai@ruc.edu.cn(通信作者)。

来进行数据管理，如 BigTable[9]，HBase[10]，Cassandra[11]，SSDB[12]等。

由于大数据的数据规模巨大，数据增速快，因此大数据存储系统对存储设备的存储密度和成本都非常敏感。磁盘由于存储密度高且价格便宜，成为大数据存储的一种非常重要的存储介质。但是，目前磁盘广泛采用的垂直磁记录即将达到密度极限，存储密度很难进一步增长，难以与数据规模增长的速度匹配。因此，最新的瓦记录(SMR)[13-14]技术正在逐步取代传统磁盘技术。SMR 技术通过磁道之间彼此部分重叠来提高存储密度，将是磁盘在未来很长一段时间内的主要形式。

随着数据的增长，大规模数据的产生对大数据存储系统的可扩展性、性能和成本等方面提出了更高的要求。由于 SMR 硬盘具有存储密度高、价格便宜等优点，大数据存储系统应该基于 SMR 磁盘来构建，包括基于 LSM 树的 KV 存储系统。但是，SMR 磁盘存在比较严重的写放大现象，随机写操作性能比较差[13-14]。对此，本文提出用随机访问性能很好的闪存固态盘(SSD)和 SMR 磁盘构成混合存储，并基于这种混合存储进行基于 LSM 树的键值存储的性能优化。

具体来说，一方面针对 SMR 磁盘适合大粒度 I/O 的特点，通过增加写文件的大小来降低 SMR 写放大，提高 SMR 的写性能；另一方面通过结合 LSM 树 KV 存储中不同存储对象的访问特征来提高 SSD 承担的 I/O 量的比例，从而提高 KV 存储的整体性能。基于 LevelDB 的实验结果表明，本文方法使得 SSD-SMR 混合存储方案比传统磁盘方案的随机写性能提高 20%，随机读性能提高 5 倍。

本文第 1 节将给出基于 SSD-SMR 混合存储的 LSM 树键值存储系统的性能模型；第 2 节设计 LSM 树键值系统下的 SMR 磁盘访问的性能优化方案；第 3 节设计面向 SSD-SMR 混合存储特征优化性能的 LSM 树键值系统；第 4 节给出实验结果与分析；第 5 节介绍相关研究工作；最后总结全文。

1 基于 SSD-SMR 混合存储的 LSM 树键值存储系统的性能模型

本节首先介绍了 SMR 硬盘和 LSM 树键值存储系统，然后给出基于 SSD-SMR 混合存储的 LSM 树键值存储系统的性能模型，并分析如何提高系统性能。

1.1 SMR 硬盘

SMR 磁盘一般可以分为驱动器管理式(Drive-Managed)和主机管理式(Host-Managed)两大类。目前，主流的 SMR 磁盘产品多数是驱动器管理式，例如希捷发布的 5 TB 和 8 TB 的驱动器管理式 SMR 磁盘产品。

SMR 的结构如图 1 所示。SMR 通过磁道的部分重叠来提高存储密度，读磁头较小，即使很窄的磁道也能读出数据。而写磁头较大，如果进行随机写入，会覆盖临近磁道的内容，因此必须重写整个磁盘或很多磁道的内容，这就是 SMR 的写放大现象。

为了减少写放大，SMR 内分成多个 Band，Band 内包含重叠轨道，而 Band 之间有间隔，从而使得每次写放大不需要跨越多个 Band。因此，SMR 磁盘进行写操作时，需要先把整个 Band 的内容读到内存，在更改内容后再把整个 Band 的内容写回磁盘。例如希捷的 5 TB SMR 产品中，Band 的大小在 17～36 MB 之间，假如写 1 MB 的数据，则需写一个 20 MB 的 Band，即写放大 20 倍。

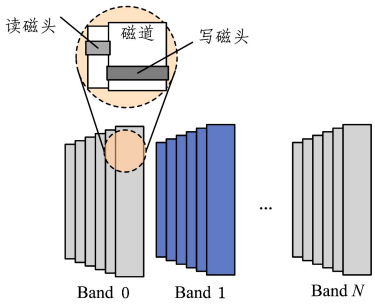

图 1 SMR 磁盘的内部结构图

Fig. 1 Internal structure of SMR disk

由于 SMR 本身存在写放大的特性，随机写性能很差，特别是小粒度的随机写，因此 SMR 硬盘作为存储层时适合存放较大的文件，并且适合追加写的方式，而不适合小粒度随机更新的方式。

1.2 LSM 树键值存储

基于 LSM 树(Log-Structured Merge Tree)的键值存储系统在读、写操作性能方面能够达到比较好的平衡，近年来在新型的 NoSQL 系统中逐渐得到广泛应用。其中，比较典型的开源实现是 LevelDB[15]。LevelDB 是 Google 开源的持久化键值存储引擎，使用 LSM 树组织数据，能够把随机写变成批量的顺序写，通过牺牲部分读性能使得写性能得到大幅度提高。

LevelDB 的结构如图 2 所示。在内存中，数据在 Memtable 中有序存储；在外存中，数据在 SSTable 中有序存储。LevelDB 有很多层，一般下一层大小是上一层的 10 倍，内存 Memtable 被放满时，数据会放到第 0 层成为 SSTable。当某层数据量超过预设的阈值时，会选择一个 SSTable 与下层有重叠 *key* 范围的 SSTable 做合并操作，将生成的新 SSTable 放到下一层，这个操作被称为压缩(Compaction)。第 0 层的 SSTable 因为是由内存的 Memtable 直接放下来的，所以它们之间无序，除此之外，每层的 SSTable 之间有序。当读数据时，先读内存的 Memtable，在无法读取的情况下才从外存的第 0 层开始依次往下读。

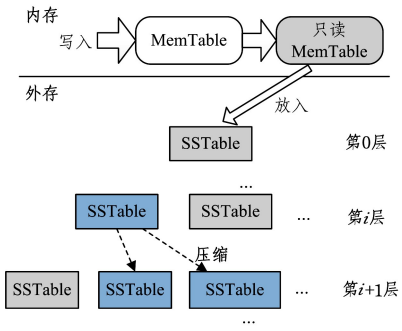

图 2 LevelDB 结构示意图

Fig. 2 Schematic diagram of LevelDB structure

LSM 树在内存中是把随机写转化为批处理写；在外存中是将各个 SSTable 进行压缩合并来生成新的 SSTable，也是批

处理的写。SMR 硬盘的存储密度高、价格便宜，但随机写性能较差，大粒度的批处理写性能较好，因此 LSM 树有可能与 SMR 硬盘特性匹配。

1.3　性能模型

SMR 硬盘存储密度高、价格便宜，但是 I/O 访问速度，尤其是小粒度的随机写性能较差；SSD 的 I/O 性能比磁盘高很多，但是价格昂贵。将 SSD 和 SMR 组成混合存储系统能够充分发挥二者的优势，弥补各自的缺点。

对于基于 SSD-SMR 混合存储的 LSM 树键值存储系统，一部分数据存储在 SSD 上，另一部分数据存储在 SMR 磁盘上，其性能主要取决于 I/O 操作的数量和速度。系统的整体性能 P 可以由下式表示：

$$P=K_{SSD}*P_{SSD}+(1-K_{SSD})*P_{SMR}$$

其中，K_{SSD} 表示 KV 存储系统访问 SSD 的 I/O 量与总 I/O 量的比例，P_{SSD} 表示 SSD 的访问性能，P_{SMR} 表示 SMR 磁盘的访问性能。一般来说，P_{SSD} 远远高于 P_{SMR}，而 SMR 磁盘存在写放大现象，不同的访问模式带来的写放大会有很大的差异，P_{SMR} 的值也会有很大的波动。因此，要想提高 LSM 树键值存储系统的整体性能，可从以下两个方面入手：

1）降低 SMR 磁盘的写放大率，从而提高 P_{SMR}，具体的优化方法将在第 2 节详细阐述。

2）提高 SSD 的 I/O 量占比 K_{SSD}，利用 SSD 访问速度快的优势，在读、写两个方面都能减少对慢速的 SMR 磁盘的访问，从而提高系统的整体性能，具体策略将在第 3 节详细阐述。

2　LSM 树键值存储系统下的 SMR 磁盘性能优化

本节首先基于一系列实际系统测试来分析如何优化 LSM 树键值存储系统下的 SMR 磁盘访问性能，之后总结测试得出的结论。

2.1　优化 LSM 写操作的粒度

LSM 树的典型实现系统为 LevelDB，本文即在 LevelDB 上进行实验。LevelDB 存储数据单元（SSTable）的大小默认为 2 MB，如果存储在 SMR 硬盘上，则存在写放大。为观察写放大的影响，在 LevelDB 中测试了在 HDD（普通硬盘）和 SMR 下，不同大小的存储数据单元对写性能的影响。本文使用 YCSB[16]（Our Yahoo! Cloud Serving Benchmark）进行测试，其中，key 的大小为 31 B，value 的大小为 1000 B，本文测试随机写 1 千万条 key-value 对，大小相当于 10 GB 数据。

图 3 给出了 SSTable 为 2 MB，20 MB，100 MB 和 200 MB 时，在 HDD 和 SMR 硬盘上随机写 10 GB 数据的吞吐量。从图中看出，随着 SSTable 的增大，吞吐量增大，当 SSTable 为 100 MB 时，吞吐最大，之后吞吐量开始下降，即这是一条先增后减的曲线。同时，当 SSTable 为 2 MB 和 20 MB 时，由于 SMR 存在写放大，其性能低于 HDD；当 SSTable 为 100 MB 和 200 MB 时，SMR 写放大的影响不大，其性能优于 HDD。因为 SMR 硬盘存储密度高，同时基于 fio 的测试发现在大粒度随机写下 SMR 硬盘的性能优于 HDD，虽然实验测试的 HDD 转数为 7200，而 SMR 硬盘转数只有 5900。因此，在设置存储数据单元大小时应大于 Band。

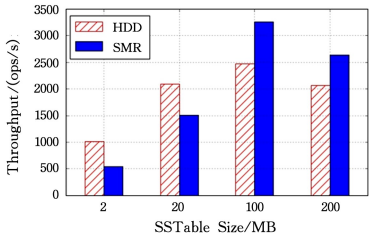

图 3　随机写 10 GB 数据的吞吐量

Fig. 3　Throughput of randomly writing 10 GB data

为分析系统性能先上升后下降的原因，本文统计了 LevelDB 产生的 I/O 量，实验结果如图 4 所示。从图中可知，随着 SSTable 变大，I/O 量增加，写文件大小的增大会导致磁盘性能上升，刚开始时磁盘性能上升带来的优势能弥补增加的 I/O 量，因此性能上升，当优势不能弥补增加的 I/O 量时，性能便下降。

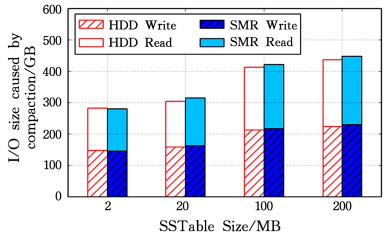

图 4　LevelDB 产生的合并压缩操作的 I/O 量

Fig. 4　I/O amounts caused by compactions of LevelDB

2.2　SSTable 大小对读性能的影响

本文基于 YCSB 测试了不同大小的存储数据单元对读性能的影响，在 20 GB 数据中随机读取了 1 GB 数据，实验结果如图 5 所示。从图中可知，SSTable 大小对随机读性能无影响，SMR 随机读性能优于 HDD，因为 SMR 的存储密度更高。

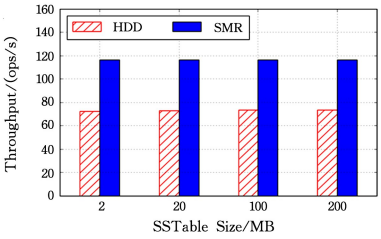

图 5　随机读 1 GB 数据的吞吐量

Fig. 5　Throughput of randomly reading 1 GB data

2.3　总结

总的来说，LSM 树键值存储系统的 I/O 特征比较适合 SMR 磁盘，因为 LSM 树是通过牺牲部分读性能来优化写性能，把随机写转化为批处理写，并且大粒度情况下 SMR 硬盘的性能优于 HDD。对于 SMR 磁盘这种优点和缺点都非常明显的硬件来说，软件的 I/O 访问特征如果非常匹配，那么可以充分发挥其优势，同时弥补其劣势。

不论是来自 Memtable，还是不同层之间的合并压缩操作，LSM 树键值存储系统的写入操作粒度都是 SSTable 大小。那么，这个写入粒度应该设置为多大呢？通过前文的测试和分析发现，粒度应该超过或接近 SMR Band 的大小，且为 100 MB 时性能最好。粒度过大时不能得到较好的性能，因为

SSTable 越大，压缩过程中产生的 I/O 量也越大。另外，SSTable 大小对读性能几乎没有影响。

3 SSD-SMR 混合存储的 LSM 树键值系统

在写操作方面，LSM 树结构是自上往下进行的，因此前几层会频繁写入数据，I/O 量比较大。在读操作方面，LSM 树结构中每一层内会按照键(key)的大小排列，但是层之间不能保证是有序的。读操作也需要从 LSM 树的顶端开始，逐层往下，直到读到目标数据为止，因此前几层的读访问量也相对较大。利用快速的 SSD 来存储 LSM 树中前几层的数据，能提高 SSD 的 I/O 量占比和平均 I/O 访问速度。

由前面的分析可知，要想提高 LSM 树键值系统的 SMR 访问性能，须设置存储数据的单位大于 SMR 的 Band。最简单直接的方案是 Rekha 等[18]为减少 SMR 写放大，设置 Memtable 和 SSTable 与 Band 一样大。但是这样也会带来一些问题：1)Band 大小是不固定的，在 17～36 MB 之间取值，很难准确按照 Band 的大小来设置 SSTable 的大小；2)很多用户选择 LevelDB 的一个重要理由是它使用的内存少，增大 Memtable 即增大了使用内存；3)由于前几层 I/O 比较频繁，同时 SSTable 很大，合并操作的 I/O 量很多，会造成性能下降；4)如果想把 I/O 访问密集的前几层放入 SSD 来加速，且前几层的 SSTable 很大，那么空间有限的 SSD 中无法装入很多 SSTable。

因此，本文提出一种逐层增加文件大小的分布方案，如图 6 所示。以 LevelDB 为例，设置 Memtable 仍为 4 MB 不变，以控制对内存的消耗；而将 SSTable 的上限设置为 100 MB。这样，上面的 SSTable 比较小，但通过逐渐合并，在下面的某个层次开始会达到上限 100 MB。另外，设置一个“小文件阈值”，假设为 17 MB，小于阈值的文件都被放入 SSD 中存储，大于阈值的文件被放入 SMR 磁盘。在这种情况下，前几层的小文件 I/O 访问密集，可以充分利用 SSD 的高性能；后面几层的大文件接近或超过 SMR Band 的大小，写入 SMR 磁盘时会控制写放大率的大小，提高 SMR 的访问速度。

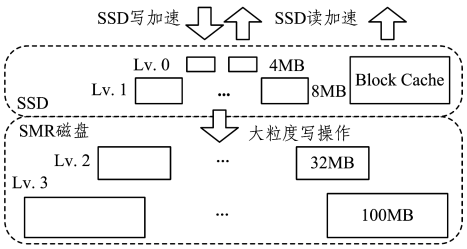

图 6 LevelDB 中 SSTable 大小的分布示例

Fig. 6 Example of SSTable size distribution in LevelDB

对于写操作来说，写入粒度是 SSTable 的大小，上面几层被密集访问。按照以上设计，可以充分利用 SSD 的性能优势。但是，键值存储系统的读操作粒度非常灵活，可以是一个很小的键值对，也可以是一个范围内的很多键值对，因此存于 SMR 磁盘下层的数据中也会有一些热度较高的数据。对于这部分热门数据，由于内存缓存(Block Cache)的空间非常有限，很难带来足够高的命中率，因此也可以利用 SSD 的一部分空间作为扩展的 Block Cache 来缓存热门数据，以进一步提高键值存储系统的读操作性能。

综上所述，本文提出了 LSM 树键值系统下的 SSD-SMR 混合存储，即将 SSD 分为两个空间，一个空间用于存放小文件，大文件存储在 SMR 磁盘上，当该空间满时小文件也存储在 SMR 磁盘上；另一部分空间作为辅助缓存，用于提高随机读性能，缓存读单位数据块，使用先进先出策略管理缓存队列。

SSD-SMR 混合存储需要的 SSD 空间并不多，因为 SSD 小文件空间只是存放了前几层的小文件，并且这些小文件会被频繁地生成和删除，同时在 SSD 小文件空间的文件并不多。在小文件的阈值为 17M 的情况下，设置 SSD 小文件空间为 100 MB 即可。因此，SSD 的引入并不会给系统带来明显的成本提升，但是对性能的提升却比较明显。

4 实验结果与分析

本文在 LevelDB 中实现了 LSM 树键值系统下的 SSD-SMR 混合存储，主要包括两个部分：1)控制数据按照定义的规则在 SSD 和 SMR 磁盘上合理分布；2)用 SSD 扩展 Block Cache。

测试中 LevelDB 的具体设置如下：SSTable 上限为 100 MB，Memtable 大小为 4 MB，SSD 小文件空间默认为 100 MB，小文件阈值为 17 MB，SSD 辅助缓存空间为 100 MB。

实验环境是 Linux 3.19.0，8 GB 内存，7200 转 500 GB 希捷普通硬盘，5900 转 8 TB 希捷 SMR 硬盘，750 GB PCIe SSD。4.1 节给出写性能测试的结果与分析，4.2 节给出读性能测试的结果与分析。

4.1 写性能测试的结果与分析

本文用 LevelDB 自带的 db_bench 测试了数据规模分别为 10 GB，20 GB 和 50 GB 3 种情况下随机写的性能。在这项测试中，分别将 LevelDB 架构在传统磁盘(HDD)、SMR 磁盘、SSD 以及 SSD 和 SMR 磁盘的混合存储之上。实验结果如图 7 所示，其中纵坐标表示随机写的吞吐量。由图 7 可知，基于纯 SSD 存储的系统写性能最好；SMR 性能优于 HDD，这是因为 SMR 磁盘的存储密度大，在系统中以大粒度写入为主；而基于 SSD-SMR 混合存储的性能优于 SMR 磁盘，大约提高了 20%，考虑到总共使用的 SSD 空间为 200 MB，占数据规模的比例为 0.4%～2%，因此 20%的性能提升还是非常明显的。

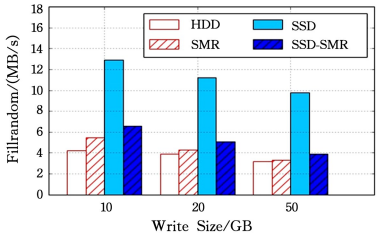

图 7 各存储方式的随机写性能对比

Fig. 7 Performance comparison of random write of storage methods

4.2 读性能测试的结果与分析

本文用 LevelDB 自带的 db_bench 测试了数据规模为 10 GB 情况下随机读 1 GB 数据的性能。实验结果如图 8 所示，其中纵坐标表示随机读操作的平均吞吐量。除了写性能测试中参与对比的 4 种存储方案，新增了一组 No Cache 方案，其表示只利用 SSD 来存储小文件，而不用 SSD 作为辅助缓存。

由图 8 可知，基于纯 SSD 存储的随机读性能依然最好，基于 SMR 磁盘的性能仍然优于 HDD，而两种 SSD-SMR 混合存储方案(SSD-SMR 和 NO Cache)的性能都明显较好，其中 SSD-SMR 方案优于 No Cache。由于 SSD 随机读性能明显好于 SMR 磁盘，SSD 设备的读性能也明显优于自身的写性能，因此 SSD-SMR 混合存储方案的随机读性能提升比写操作的提升更为明显，提升到 6 倍。

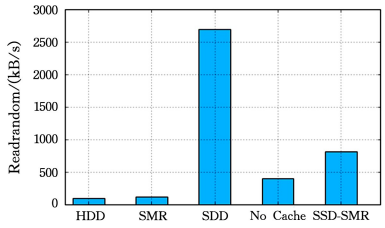

图 8　随机读 1 GB 数据的性能对比

Fig. 8　Performance comparison of randomly reading 1 GB data

此外，本文还统计了 SSD-SMR 混合存储实验中 LevelDB 读数据块的各位置的比例，实验结果如图 9 所示。图中的 Block Cache 表示内存中的 block cache，SSD 表示存储在 SSD 上的 SSTable 文件，SSD Cache 表示作为辅助缓存的 SSD 空间，SMR 表示存储在 SMR 上的 SSTable。由图 9 可知，有 36.28%的读在 100MB 的 SSD 辅助缓存空间上，相比内存 8 MB 的 block cache，提高了 11 倍左右；添加了 SSD 辅助缓存后，数据从 SSD 上读数据块占比 44.76%，因此 SSD－SMR 的读性能比 HDD 提升了 5 倍。

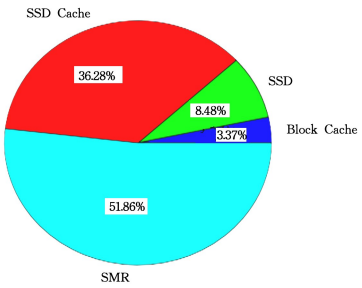

图 9　SSD-SMR 读数据块位置的占比统计

Fig. 9　Statistics of SSD-SMR read block location

5　相关工作

1)SMR 磁盘与 LSM 树键值存储结合。Rekha 等[17]提出 SMRDB，使用 SMR 存储数据，在 LevelDB 上实现，为降低 SMR 写放大的影响，提出把 Memtable 和 SSTable 的大小设置得与 Band 一样；但是他们并没有采用真实的 SMR 设备，而是采用普通的磁盘来模拟。基于 YCSB 的测试表明，SMRDB 的性能超过 LevelDB 的 8.8%～123.6%。

华中科技大学的 Yao 等[18]提出一种名为 LWC-tree 的方法来降低 LSM 树在 Compaction 过程中的写放大，在 LevelDB 上减少了 80%的写放大。LWC-tree 在 SMR 上比 LevelDB 在 HDD 上的写性能提高了 467%。

2)混合存储。基于闪存的 SSD 由于其高性能和高存储密度等优点，在企业存储系统中得到了广泛应用。混合存储系统由 SSD 和磁盘组成[19-21]，也是现代数据中心的重要存储形式，因为它们具有成本低、性能高和数据可靠的优势。此外，还有许多工业产品或开源系统来优化混合存储中的 SSD 缓存，例如 Facebook Flashcache[22]，Linux dm-cache[23]，EMC FAST 缓存[24]和 Exadata 中的 Oracle Smart Flash Cache 数据库机[25]。除了一些传统的通用缓存算法，如 LRU，MQ[26]，LIRS[27]，ARC[28]等，还有一些特别优化的基于 SSD 的缓存算法，如 SieveStore[29]，LARC[30]以及 Solaris ZFS 文件系统中的二级 ARC(L2ARC)技术[31]，其添加了不同类型的新数据过滤器，以提高缓存数据的质量，同时降低缓存更新的频率。

3)优化 LSM 树键值存储系统。压缩合并过程中的 I/O 开销较高，是 LSM 树面临的严重挑战。有些应用通过使用硬件并行性(如 CPU 和 I/O 设备)来加速压缩。RocksDB[32]支持配置任意数量的 Memtables，并且可以多线程压缩。PCP[33]也将压缩分解成几个子任务，并利用 I/O 和 CPU 资源的并行性来以流水线的方式处理这些子任务。LOCS[34]扩展了 LevelDB，以利用定制的开放式 SSD 的并行性，并优化调度和调度策略以提高效率。

SSD-SMR 混合存储与 LSM 树的结合暂时没有相关研究成果。本文只使用了少量的 SSD 空间，相对于传统 HDD 存储可以降低存储成本并提升性能，相对于 SMR 存储可以提升性能。

结束语　本文提出的基于 SSD-SMR 混合存储的 LSM 树键值存储系统通过增大写操作的粒度来减少 SMR 磁盘的写放大，从而提升 SMR 磁盘的性能，以适应 SMR 磁盘硬件的特征；同时，通过把小文件放到 SSD 上来提高 I/O 量的占比，并把 SSD 一部分空间作为辅助缓存来提高随机读性能。最后，将所提出的方案在开源系统 LevelDB 中实现，实验测试表明，当实验数据分别为 10 GB，20 GB 和 50 GB 时，使用 200MB 的 SSD 空间可以使随机写性能提高 20%，使随机读性能提升到 6 倍。SSD-SMR 混合存储只需要引入少量的 SSD 空间，便可降低存储成本，提升性能。

参 考 文 献

[1] TURNER V，GANTZ J F，REINSEL D，et al. The digital universe of opportunities：Rich data and the increasing value of the internet of things[Z]. IDC Analyze the Future，2014.

[2] GRAWINKEL M，NAGEL L，PADUA F，et al. Analysis of the ECMWF storage landscape[C]// Usenix Conference on File and Storage Technologies. USENIX Association，2015：15-27.

[3] Amazon Web service[EB/OL]. https://aws. amazon. com.

[4] Microsoft azure[EB/OL]. https://azure. microsoft. com.

[5] Aliyun. com：An integrated suite of cloud products，services and solutions[EB/OL]. http://www. alicloud. com.

[6] Apache couchdb[EB/OL]. http://couchdb. apache. org.

[7] Tokyo cabinet：A modern implementation of dbm[EB/OL]. http://fallabs. com/tokyocabinet.

[8] Redis[EB/OL]. https://redis. io.

[9] CHANG F，DEAN J，GHEMAWAT S，et al. Bigtable：a distributed storage system for structured data[J]. Acm Transactions on Computer Systems，2008，26(2)：1-26.

(下转第 89 页)

[7] ZHAO G S，CHEN M. Congestion control mechanism based on accepting threshold in delay tolerant networks[J]. Journal of Software，2013，24(1)：153-163. (in Chinese)
赵广松，陈鸣. 基于接收阈值的容延网络拥塞控制机制[J]. 软件学报，2013，24(1)：153-163.

[8] YAN H C，GUO J，ZHANG H J. Performance evaluation of routing algorithms on space delay/disruption tolerant networks[J]. Institute of Spacecraft System Engineering，2016，36(4)：38-46. (in Chinese)
燕宏成，郭坚，张红军. 空间延迟/中断容忍网络路由算法性能评估[J]. 空间科学与技术，2016，36(4)：38-46.

[9] ZHOU X B，ZHOU J，LU H C. Analysis of Delay Model in DTN[J]. Journal of Computer Research and Development，2008，45(6)：960-966. (in Chinese)
周晓波，周健，卢汉成. DTN 网络的延时模型分析[J]. 计算机研究与发展，2008，45(6)：960-966.

[10] WANG E，YANG Y J，LI L. Game of Life Based Congestion Control Strategy in Delay Tolerant Networks[J]. Journal of Computer Research and Development，2014，51(11)：2393-2407. (in Chinese)
王恩，杨永健，李莅. DTN 中基于生命游戏的拥塞控制策略[J]. 计算机研究与发展，2014，51(11)：2393-2407.

[11] QIU K. Performance of DTN Bundle Routing Protocol in Deep Space[D]. Harbin：Harbin Institute of Technology，2012. (in Chinese)
邱坤. 深空 DTN 集束层路由协议研究[D]. 哈尔滨：哈尔滨工业大学，2012.

[12] 陈敏. OPNET 网络仿真[M]. 北京：清华大学出版社，2004.

(上接第 65 页)

[10] Apache hbase[EB/OL]. http://hbase.apache.org.

[11] LAKSHMAN A，MALIK P. Cassandra：a decentralized structured storage system[J]. Acm Sigops Operating Systems Review，2010，44(2)：35-40.

[12] Ssdb-a fast nosql database for storing big list of data[EB/OL]. https://github.com/ideawu/ssdb.

[13] AMER A，LONG D D E，MILLER E L，et al. Design issues for a shingled write disk system[C]// IEEE，Symposium on MASS Storage Systems and Technologies. IEEE Computer Society. 2010：1-12.

[14] AMER A，HOLLIDAY J A，DE LONG D，et al. Data management and layout for shingled magnetic recording[J]. IEEE Transactions on Magnetics，2011，47(10)：3691-3697.

[15] Leveldb：a fast and lightweight key/value database library by google[EB/OL]. http://code.google.com/p/leveldb.

[16] FCOOPER B，SILBERSTEIN A，TAM E，et al. Benchmarking cloud serving systems with ycsb[C]// ACM Symposium on Cloud Computing. 2010：143-154.

[17] PITCHUMANI R，HUGHES J，MILLER E L. SMRDB：Key-Value Data Store for Shingled Magnetic Recording Disks[C]// 8th ACM International Systems and Storage Conference. 2015.

[18] YAO T，WAN J G，HUANGY P，et al. A Light-weight Compaction Tree to Reduce I/O Amplification toward Efficient Key-Value Stores[C]// 33rd International Conference on Massive Storage Systems and Technology. 2017.

[19] SAXENA M，SWIFT M M，ZHANG Y Y. Flashtier：a lightweight，consistent and durable storage cache[C]// Proceedings of the 7th ACM european conference on Computer Systems. 2012：267-280.

[20] KGIL T，ROBERTS D，MUDGE T. Improving NAND Flash Based Disk Caches[C]// International Symposium on Computer Architecture. IEEE，2008：327-338.

[21] YANG Q，REN J. I-CASH：Intelligently Coupled Array of SSD and HDD[C]// IEEE，International Symposium on High PERFORMANCE Computer Architecture. IEEE，2011：278-289.

[22] SRINIVASAN M，SAAB P. A general purpose，write-back block cache for linux[EB/OL]. https://github.com/facebookarchive/flashcache.

[23] THORNBER M S J，MAUELSHAGEN H. dm-cache[EB/OL]. https://en.wikipedia.org/wiki/Dm-cache.

[24] Emc fast cache：A detailed review[EB/OL]. http://www.emc.com/collateral/software/white-papers/h8046-clariioncelerra-unified-fast-cache-wp.pdf.

[25] Exadata smart flash cache features and the oracle exadata databas machine[EB/OL]. http://www.oracle.com/technetwork/serverstorage/engineer/d-systems/exadata/exadata-smart-flash-cache-366203.pdf.

[26] ZHOU Y Y，CHEN Z F，LI K. Second-level buffer cache management[J]. IEEE Transactions on parallel and distributed systems，2004，15(6)：505-519.

[27] JIANG S，ZHANG X. Lirs：An efficient low inter-reference recency set replacement policy to improve buffer cache performance[C]// Proceeding of 2002 ACM SIGMETRICS. 2002.

[28] NMEGIDDO，MODHA D. Arc：a self-tuning，low over-head replacement cach[C]// Proceedings of the 2nd USENIX Symposium on File and Storage Technologies. 2003.

[29] PRITCHETT T，THOTTETHODI M. SieveStore：a highly-selective，ensemble-level disk cache for cost-performance[C]// International Symposium on Computer Architecture. ACM，2010：163-174.

[30] HUANG S，WEI Q，CHEN J，et al. Improving flash-based disk cache with lazy adaptive replacement[C]// Proceedings of the 29th International Conference on Massive Storage Systems and Technology. 2013.

[31] GREGGB. L2arc[EB/OL]. https://blogs.oracle.com/brendan/entry/test.

[32] Under the hood：Building and open-sourcing rocksdb[EB/OL]. http://goo.gl/9xulVB.

[33] ZHANG Z，YUE Y，HE B，et al. Pipelined Compaction for the LSM-Tree[C]// IEEE，International Parallel and Distributed Processing Symposium. IEEE Computer Society，2014：777-786.

[34] WANG P，SUN G Y，JIANG S，et al. An efficient design and implementation of lsm-tree based key-value store on open-channel ssd[C]// Proceedings of the Ninth European Conference on Computer Systems. 2014.