异构多固态盘的吞吐量优化

杨良怀 万凯明 范玉雷
(浙江工业大学计算机科学与技术学院 杭州 310023)

摘 要 固态盘具有低访问延迟、抗震性、内部并行性等诸多优良特性,已被广泛使用。如何利用固态盘提高系统性能是当前研究议题之一。首先通过一系列不同读写比例的负载实验来探索固态盘的特性,发现在较大I/O请求粒度场景下,较高的读请求比例有利于提升各类固态盘的吞吐量。基于实验结论,提出了一种I/O只读负载分离方法RODI,通过合理放置只读数据来分离只读负载到合适的固态盘上,以提升异构多盘阵列整体的吞吐量。大量实验表明,在较大I/O粒度的异构多盘环境中,相比传统的RAID技术,RODI方法对于改善多盘总体吞吐量更具优势。

关键词 异构多固态盘,吞吐量优化,固态盘特性
中图法分类号 TP315 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.02.037

Throughput Enhancement for Heterogeneous Solid-state Drives

YANG Liang-huai WAN Kai-ming FAN Yu-lei
(School of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310023, China)

Abstract Solid-state drives are widely used nowadays for its superiority of low latency, shock resistance and internal parallelism. How to enhance the performance of solid-state drives (SSDs) is one of the hot research topics. We explored SSDs' external behavior through experiments and found that high ratio of read requests is favorable for better SSDs' throughput. Based on these observations, a method called RODI (Read-only Data Isolation) was put forward, which isolates read-only load to certain SSDs from the rest to increase heterogeneous SSDs' total throughput. Experiment results show that RODI method can effectively improve heterogeneous solid-state drives' throughput.

Keywords Heterogeneous solid-state drives, Throughput enhancement, Solid-state drives characteristics

1 引言

随着存储技术的飞速发展,固态盘(SSD)、相变内存(PCM)和忆阻器(Memristo)[1]相继诞生。作为其中最成熟的解决方案,固态盘已经凭借其高性能、低功耗、抗震等优点成为硬盘(HDD)的有力替代者。不同于硬盘,固态盘没有机械运动部件,其特性与硬盘有较大差异,如具有极高的顺序访问吞吐量和极低的访问延迟、早期固态盘随机读/写性能的差异等,这激发了学术界和工业界对固态盘的研究兴趣[2-5]。过去的一些工作[3,6-8]设法利用固态盘的快速随机读来避免随机写。随着固态盘技术的发展,固态盘引入了一些新特性,如内部并行性以及NCQ(Native Command Queuing)技术[9-10]。早期的一些假设,如"随机写开销大应加以避免,读开销相对小",在新一代的固态盘中不再成立[11],颠覆了学术界早期在固态盘研究中的部分认知,给固态盘的算法实现和优化提出了新问题与新挑战。

自Berkeley大学的Patterson Gibson和Katz于1987年提出磁盘阵列(RAID)技术以来,RAID已经在生产环境中得到了大量使用。随着固态盘的普及,基于固态盘RAID的读写性能优化成为研究热点[21-23],尤其是在使用固态盘的情形下RAID吞吐量的优化需要得到更深入的研究。

随着固态盘技术的不断更迭,不同厂商、不同型号的固态盘不断涌现。随着业务规模的扩大以及时间的推移,企业信息系统在设备更新时往往引进了不同品牌或者型号的固态盘,由于经济上的原因,一般仍然继续使用原有的旧型号固态盘,从而带来了RAID中固态盘异构性问题。大数据时代的来临,使得海量数据分析、处理成为常态,较大的I/O请求粒度常被用于提高吞吐量。例如,Hadoop的分布式文件系统HDFS默认采用64MB大小的块,联机分析处理也可采用较大的数据块来增加吞吐量。如何在大粒度存取情形下优化异构固态盘阵列是本文的研究议题。

本文的主要贡献如下:

(1)考察了不同粒度下读/写混合对OCZ、三星、Intel系列的固态盘吞吐量的影响,分析了优化固态盘吞吐量的方法;

(2)提出了只读数据隔离(RODI)方法,在理论上分析了RODI方法相比于传统RAID 0技术在提升异构固态盘吞吐量方面的优势。实验表明,与传统RAID 0方法相比,该方法在异构多盘SSD环境下的吞吐量方面有明显的优势。

到稿日期:2016-01-21 返修日期:2016-06-01 本文受浙江省基金项目(LY14F020017,LQ15F020007),国家基金项目(61070042)资助。
杨良怀(1967—),男,博士,教授,主要研究方向为数据库系统,E-mail:yanglh@zjut.edu.cn;万凯明 硕士生,主要研究方向为数据库系统;范玉雷(1984—),男,博士,讲师,主要研究方向为数据库系统,E-mail:fyl815@wjut.edu.cn(通信作者)。

本文第 2 节阐述了相关工作;第 3 节考察混合读/写对各品牌固态盘吞吐量的影响,分析并得出优化吞吐量的方法;第 4 节描述 RODI 方法,并对其相比于 RAID 0 具有优势需具备的条件进行了系统的分析;第 5 节通过实验验证 RODI 方法在优化吞吐量方面的有效性;最后总结全文。

2　相关工作

围绕新一代存储技术开展的相关研究已有不少。针对固态盘,早期研究集中在改善写操作性能相比于读性能较差所带来的一系列问题[2-8]。连接是直接影响数据库系统整体性能的一类核心操作,针对固态盘条件下如何优化连接操作,学术界已经提出了多种优化方法。Tsirogiannis 等[12]提出了通过利用连接索引以及延迟物化来减少内存使用以及数据溢出的连接算法 FlashJoin。Shah 等[13]提出了 RARE-join 连接算法,该算法通过存取两个关系的相应连接属性列进而计算连接索引来计算连接结果,通过只存取连接列属性以及 PAX 小页面来减少 I/O 次数,同时利用固态盘的小随机读延迟。与 RARE-join 相似,Li 等[14]提出了使用传统行式页面布局的 DigestJoin 连接算法。文献[15-16]对固态盘中各种连接算法的性能进行了评价。

最近,针对固态盘的研究主要围绕固态盘内部并行性展开。新一代固态盘开始支持内部并行性以及 NCQ 技术[9-10],闪存控制器并行访问多通道和多路架构是固态盘获得高性能的一种有效设计方法[2]。Park 等[19]通过优化 SSD 内部读写请求队列调度算法来提升闪存片级和颗粒级的并行性;Kim[20]与 Chen 等[9]通过探索固态盘内部构造来发挥固态盘的并行处理能力,固态盘内部并行性在数据吞吐量方面起着重要作用;Roh 等[18]提出了在单个进程中利用固态盘内部并行性的一种方法,对 B+树索引的存取进行了优化;Lai 等[17]通过探测固态盘内部结构的方法确定固态盘内部物理架构以及数据放置规律,据此进行初始数据放置,在后续扫描和连接中直接根据数据的物理地址进行访问以发挥固态盘内部的并行性。该方法由于要利用数据物理放置的底层信息,与所使用的固态盘直接相关,在可推广性方面具有一定局限性。

另一方面,读写混合对固态盘的性能影响也引起了关注[9,18]。Roh 等[18]也考察了固态盘的两种随机访问模式,一种是读/写比例是 1∶1 的高度混合 I/O 负载,另一种是非高度混合负载,I/O 请求粒度固定为 4kB,最终发现读写混合对性能具有重要影响;Chen 等[9]比较了两种负载的性能,一种是顺序读写,另外一种是随机读写,实验中队列深度设为 2,请求块大小设为 4kB。他们发现将顺序读和任意写模式混合不会造成负面影响,而将随机读和任意模式的写混合则会降低固态盘的性能。

前述诸多方法从固态盘本身的特性出发,据此来优化算法与组织数据等,过分依赖于内部结构的使用。本文认为应当采用"黑盒"方法来研究固态盘,因为上层应用并不知悉固态盘中数据的物理分布,在实际系统中无法随时获知这类底层细节,且由于固态盘内部磨损均衡的需要,即使是静态数据,其在固态盘的物理分布也在不停地变更。因此,根据数据的物理分布这种固态盘内部细节来进行优化在实际应用中具有很大的局限性。本文的"黑盒"思路是通过一系列负载测试,了解固态盘在不同读/写混合比例下其吞吐量的变化规律,以此来优化固态盘吞吐量。

3　读写混合对固态盘吞吐量的影响

读写混合是 DBMS 查询处理中典型的 I/O 存取模式。研究固态盘在读写混合时的性能表现,对算法的设计具有重要作用;海量数据分析中,批处理、离线分析架构都需要采用较大的 I/O 请求粒度来获得大吞吐量。以往读/写混合对吞吐量影响的实验所采用的 I/O 粒度都较小。本节实验在不同读/写混合比例下,分别以 2MB,8MB,32MB 大 I/O 请求粒度对不同品牌的固态盘进行测试,考察对其吞吐量的影响。

3.1　读写混合与固态盘吞吐量变化的实验

实验平台和测试环境配置参见第 5 节。本节使用 IOmeter 测试工具(www. iometer. org)生成各类不同配置的 I/O 负载,负载调控因素包括读写比率、读写模式(顺序 I/O、随机 I/O)、请求粒度以及队列深度(并发的 I/O 任务数),负载数据的逻辑寻址范围是 20GB。实验测试相应负载下固态盘的吞吐量,了解固态盘的外部特性。实验中读负载占总负载量的比例从 5%递增至 95%,模拟不同混合程度的读写负载。实验所用固态盘信息如表 1 所列。

表 1　实验用固态盘信息

SSD 名称	容量/GB	简称
Samsung SSD 830 series	128	SSD-830
Samsung SSD 850 series	128	SSD-850
OCZ vector 150	120	SSD-OCZ
Intel SSD 530 series	240	SSD-530

先考察在负载为顺序、混合读写、队列深度为 2 的情况下 4 块固态盘的性能,实验结果如图 1 所示。由图 1(a)、图 1(e)和图 1(i)可知,在读请求比例相同的 I/O 负载中,读写请求粒度大小为 2MB,8MB 和 32MB 的 3 种情况下固态盘 SSD-830 的吞吐性能表现相近,并且无论读写请求粒度大小为 2MB 还是 8MB 或是 32MB,固态盘 SSD-830 的总吞吐量都随着读请求比例的增加而逐渐升高,趋势相近,最终达到 SATA3 限制的吞吐量峰值。由此可见,当读写请求粒度大于 2MB 时,固态盘 SSD-830 的吞吐量只与读写比例相关。图 1(b)、图 1(f)和图 1(j)示出了固态盘 SSD-850 在读写请求粒度为 2MB,8MB 和 32MB 的情况下,随着读请求比例的增加,总吞吐量有少量的增加,总吞吐量一直保持在较高的水平。固态盘 SSD-OCZ 在 2MB,8MB,32MB 情况下,随着读比例的增加,总吞吐量快速增加。当写比例较高时,SSD-OCZ 具有较低的吞吐量,如图 1(c)、图 1(g)和图 1(k)所示。但是图 1(d)、图 1(h)和图 1(l)显示固态盘 SSD-530 在读写请求粒度为 2MB,8MB 和 32MB 的情况下,随着读请求比例的增加,总吞吐量变化微弱,这说明 SSD-530 在任何比例的读写混合负载下都会表现出较优的吞吐性能。该组实验表明,在较大的读写请求粒度(2MB,8MB 和 32MB)和顺序读写混合 I/O 负载情况下,除了固态盘 SSD-830 和 SSD-OCZ 的吞吐性能对读请求比例敏感之外,其余固态盘对读请求比例并不敏感。这也在某种程度上验证了不同固态盘制造厂商对固态盘的顺序读写混合 I/O 负载的吞吐性能做了不同程度的优化。

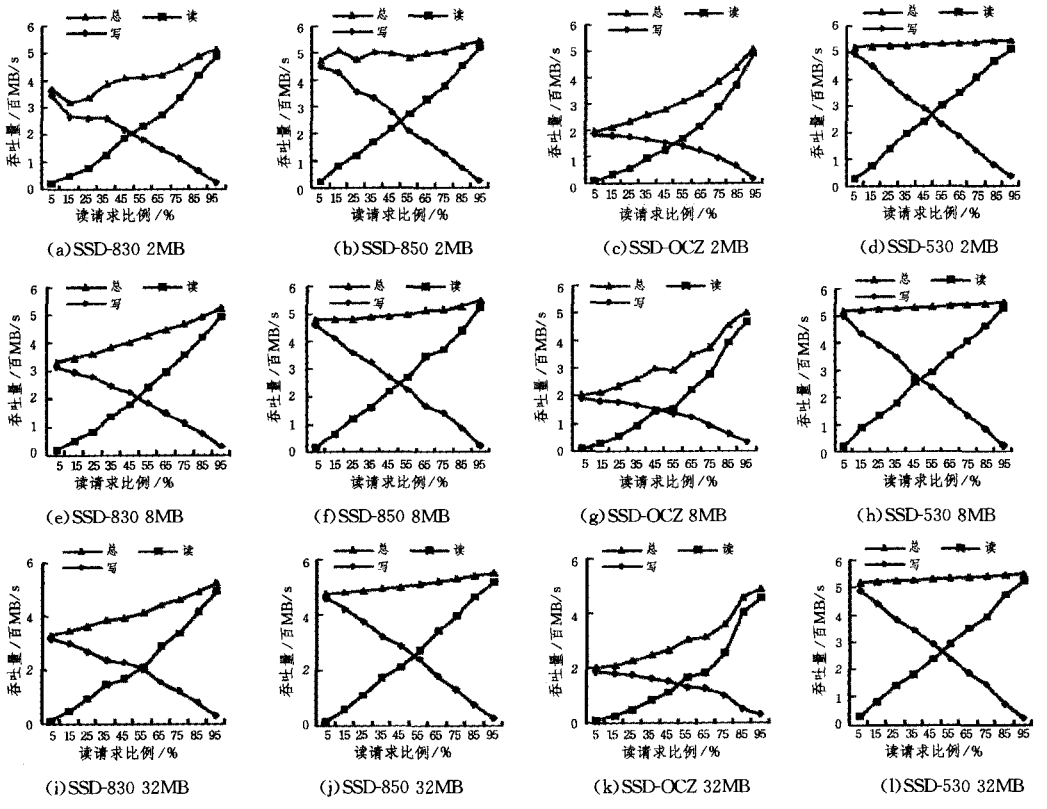


图 1 不同读写混合比例和粒度下吞吐量的变化(顺序 I/O)

现实应用中存在大量随机读写混合 I/O 负载,图 2 示出了 4 块固态盘在此负载情况下队列深度为 2 时的吞吐性能表现。

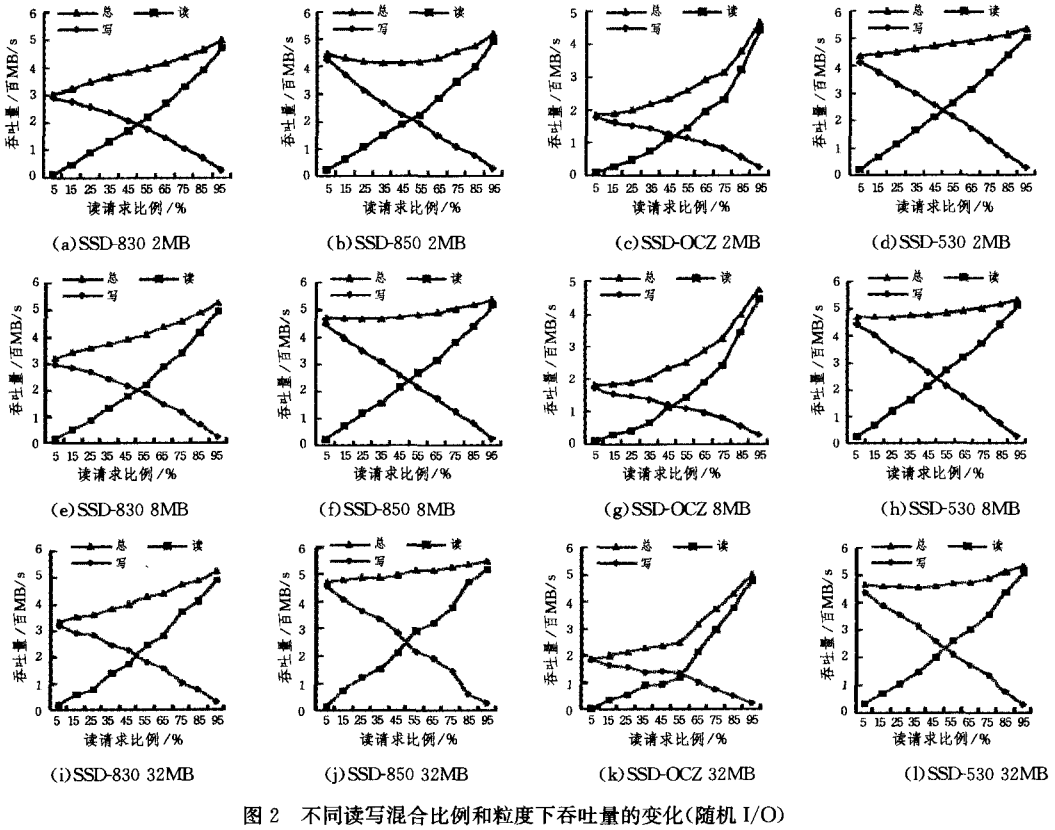


图 2 不同读写混合比例和粒度下吞吐量的变化(随机 I/O)

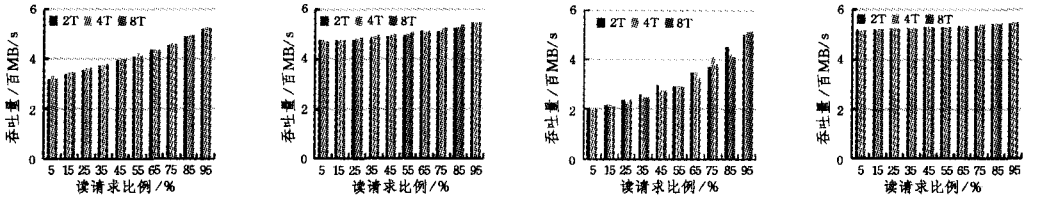
如图 2(a)、图 2(e)和图 2(i)所示,固态盘 SSD-830 在随机读写混合 I/O 负载情况下表现的吞吐性能与顺序读写混合 I/O 负载情况下表现的吞吐性能极其相似,这说明 SSD-830 的吞吐性能不受顺序和随机两种模式的影响,只与读写比例有关。对于固态盘 SSD-850,图 2(b)说明当读写请求粒度为 2MB 时,总吞吐量先下降后升高,表示其在此种情况下对读写混合比例很敏感,读写比例越接近 1 或者 0,吞吐性能越好,这与顺序 I/O 模式下的同增趋势完全不同;当读写请求粒

度为 8MB 和 32MB 时，总吞吐量与顺序读写混合 I/O 负载情况下表现的吞吐性能极其相似，如图 2(f)和图 2(j)所示。因此在读写请求粒度达到 8MB 时，固态盘 SSD-850 的吞吐性能只与读写比例有关，而在读写请求粒度小于 2M 的情况下，其吞吐性能还与顺序和随机模式有关。对比图 1(c)、图 1(g)和图 1(k)和图 2(c)、图 2(g)和图 2(k)可知，固态盘 SSD-OCZ 在随机模式下的吞吐性能与其在顺序 I/O 模式下的吞吐性能基本相似，即随着读比例的增加，总吞吐量快速增加，且当写比例较高时，SSD-OCZ 具有较低的吞吐量。固态盘 SSD-530 的吞吐性能在读写请求粒度为 2MB 的情况下随着读写比例增加而增加的趋势相较顺序 I/O 负载下吞吐性能的变化更加明显，如图 2(d)所示；在读写请求粒度为 8MB 和 32MB 的情况下，固态盘 SSD-530 的吞吐性能随着读写比例增加而有缓慢、少量增长，如图 2(h)和图 2(l)所示。

综上所述：1)固态盘 SSD-830 的吞吐量在读写请求粒度大于 2MB 的情况下随读请求比例的增大而增大，且与随机(顺序)I/O 模式无关，适合承担读多负载；2)固态盘 SSD-850 的吞吐量在读写请求粒度小于 2MB 的情况下不但与顺序和随机模式有关，还与读请求比例有关，顺序 I/O 负载下适合承担读多负载，而在随机 I/O 负载下不但可以承担读多负载，也可以用于承担写多负载；在读写请求粒度大于 8MB 的情况下与顺序和随机模式无关，并且对读请求比例不敏感，即此时适合任何类型负载；3)固态盘 SSD-OCZ 的吞吐性能在顺序模式负载和随机模式负载下差别不大：无论是顺序读写还是随机读写的吞吐性能对读请求比例都非常敏感，即 SSD-OCZ 适合承担读多负载；4)固态盘 SSD-530 的吞吐性能在读写请求粒度小于 2MB 的情况下不但与顺序和随机存取模式有关，还与读请求比例有关，顺序 I/O 负载下适合承担任何读写比例的负载，而在随机 I/O 负载下适合承担读多负载；在读写请求粒度大于 8MB 的情况下与顺序和随机读写模式无关，并且对读请求比例不敏感，即此时适合任何类型负载。

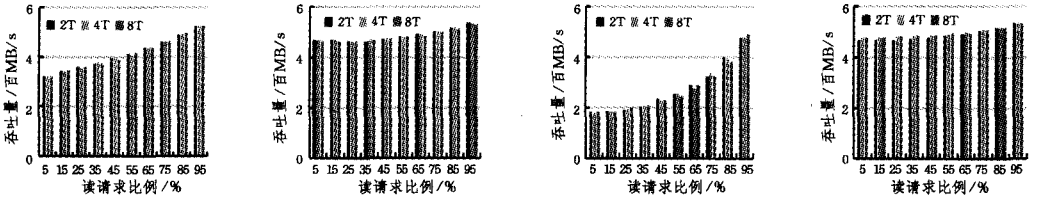
3.2　队列深度与固态盘吞吐量

固态盘存在内部并行性，本节考察队列深度的变化对固态盘吞吐性能的影响，结果如图 3 和图 4 所示。在顺序读写混合 I/O 负载且请求粒度为 8MB 的情况下，4 块固态盘的吞吐性能在队列深度为 2，4 和 8 下测得的实验结果如图 3 所示。实验结果表明 4 块固态盘吞吐性能变化趋势不受队列深度的影响，即队列深度为 2，4，8 时均表现相近的吞吐性能。而图 4 则说明在随机读写混合 I/O 负载且请求粒度为 8MB 的情况下，队列深度的增大也不会影响固态盘的吞吐性能。因此在大粒度场景下，队列深度变化对固态盘的吞吐量影响不大，即不同队列深度下，同一种固态盘在各个请求粒度下的变化趋势与队列深度为 2 时的吞吐性能变化趋势基本一致。因此，实验中所用队列深度为 2。



(a)SSD-830 8MB　(b)SSD-850 8MB　(c)SSD-OCZ 8MB　(d)SSD-530 8MB

图 3　不同队列深度下吞吐量的变化(顺序 I/O)



(a)SSD-830 8MB　(b)SSD-850 8MB　(c)SSD-OCZ 8MB　(d)SSD-530 8MB

图 4　不同队列深度下吞吐量的变化(随机 I/O)

3.3　只读收益

综上所述，在粒度较大的场景下，无论是顺序读写混合 I/O 负载还是随机读写混合 I/O 负载，读操作比例的提高均有利于提高固态盘的吞吐量，只是对不同的固态盘提升的程度不同。例如在随机读写混合 I/O 负载的场景下，通过增加读 I/O 任务的比例，对改进固态盘 SSD-830 和 SSD-OCZ 的吞吐量的效果均比较显著，而固态盘 SSD-850 和 SSD-530 的吞吐性能无论是在随机读写混合 I/O 负载还是在顺序混合读写 I/O 负载情况下均无显著变化，尤其是读写请求粒度达到 8MB 以上时。为了更好地衡量吞吐性能变化与读请求比例变化的关系，引入了“只读收益”，即纯读负载时的固态盘吞吐量相比整体不同混合比例负载的平均吞吐量所带来的吞吐性能提升量，如式(1)所示。

$$P=RT-AT \tag{1}$$

其中，P 表示只读收益，RT 表示纯读负载下的吞吐量，AT 表示不同读请求比例下吞吐量的总体平均值。

表 2 和表 3 总结了不同队列深度(用 d 表示队列深度)和访问模式下的只读收益。

通过表 2 和表 3 的数据可以看到，无论是顺序 I/O 还是随机 I/O，在固态盘上只运行只读任务将更加有利于发挥其吞吐性能。在大粒度的场景下，所有实验的固态盘都呈现出一种较为一致的规律，即只读收益均为正数。从表中数据可以看到，SSD-OCZ 和 SSD-830 无论在顺序 I/O 还是在随机 I/O 下均具有较好的只读收益。不同的读写粒度对 SSD 只读

收益的影响不大。除了 SSD-850 在随机 I/O 下的只读收益有所增长外，其他 SSD 在顺序 I/O 和随机 I/O 下都呈现出较为接近的只读收益值。

表 2 纯读负载时带来的只读收益(顺序 I/O)

固态盘名称(粒度)	2MB	8MB	32MB
SSD-830(d=2)	103.86	105.28	108.24
SSD-830(d 个=4)	115.62	111.20	109.37
SSD-830(d=8)	126.74	110.15	111.45
SSD-850(d=2)	42.93	47.37	39.80
SSD-850(d=4)	53.01	45.29	40.29
SSD-850(d=8)	55.22	42.95	39.78
SSD-530(d=2)	13.86	15.26	15.13
SSD-530(d=4)	16.33	15.33	16.46
SSD-530(d=8)	15.80	16.50	12.02
SSD-OCZ(d=2)	192.38	182.79	181.06
SSD-OCZ(d=4)	191.82	197.57	174.02
SSD-OCZ(d=8)	198.21	202.85	201.37

表 3 纯读负载时时带来的只读收益(随机 I/O)

固态盘名称(粒度)	2MB	8MB	32MB
SSD-830(d=2)	105.46	111.73	103.61
SSD-830(d=4)	111.73	112.28	108.51
SSD-830(d=8)	111.43	111.72	111.90
SSD-850(d=2)	79.15	50.34	39.51
SSD-850(d=4)	72.59	49.25	39.40
SSD-850(d=8)	72.12	46.78	45.15
SSD-530(d=2)	54.01	44.93	55.15
SSD-530(d=4)	27.95	38.41	48.64
SSD-530(d=8)	26.19	36.45	45.70
SSD-OCZ(d=2)	192.64	201.09	204.81
SSD-OCZ(d=4)	191.60	203.95	203.89
SSD-OCZ(d=8)	181.97	215.10	210.74

4 使用 RODI 改善吞吐量

面对大数据，高通量的数据分析与处理是所期望的。在这种应用场景下，数据存取的吞吐量是关键。本节探索该场景下异构固态盘的吞吐量优化。

由第 3 节的实验结果可知，多款固态盘只读请求的带宽明显大于混合读写请求负载的带宽，这种特性可以用于异构固态盘的吞吐量优化。对于面向数据分析服务的场景，大量数据是只读的，基于这些考虑，本文的方法是筛选一些只读收益较高的固态盘用于只读盘，分离出具有一定只读负荷的只读数据，将其单独存放到只读固态盘阵列，由此提升异构固态盘阵列的整体吞吐量。该方法被称为只读数据分离(Readonly Data Isolation，RODI)。使用 RODI 方法时，基于只读吞吐量和各异构固态盘的特性，根据各只读收益值从大到小选择合适数目的固态盘作为只读盘的候选盘，分离出来的只读数据放到候选盘上。上文实验中 SSD-OCZ 和 SSD-830 可作为只读盘的候选盘。

下面比较 RAID 0 的情形下采用 RODI 方法前后的固态盘阵列性能差异，从理论上分析采用 RODI 方法在改善异构多盘吞吐量上所起的作用。假定 RODI 方法将固态盘划分成两类。第一类是 m 块面向混合读写负载的固态盘，第二类是 n 块面向只读负载的固态盘。假设分离出来的只读负荷需 k 个第二类的固态盘承载，设 k 个固态盘的平均吞吐量为 TP_1，其余 $m+n-k$ 块固态盘的平均吞吐量记为 TP_2，全体 $m+n$ 块固态盘阵列的整体总吞吐量为 TP_{RODI}；使用 RAID 0 时，阵列采用条带化存储方式，各个固态盘上的平均吞吐量设为 TP_0，全体固态盘整体总吞吐量记为 TP_{RAID}。有以下计算公式：

$$TP_{RODI}=k\times TP_1+(m+n-k)\times TP_2 \tag{2}$$

$$TP_{RAID}=(m+n)\times TP_0 \tag{3}$$

对于 RAID 0 阵列，不妨假设阵列由 $DISK_1$ 和 $DISK_2$ 两块异构固态盘构成，假设某种负载下 $DISK_1$ 的吞吐量比 $DISK_2$ 的吞吐量更高。当要读取跨越两个盘的数据块时，比如读取分别存储在 $DISK_1$ 和 $DISK_2$ 上的块 B_1 和 B_2，此时由于不同的固态盘的吞吐量性能不同，B_1 先完成，B_2 后完成。若应用层需要两个数据块同时完成读取时才能工作，则此时磁盘阵列的吞吐量取决于最低吞吐量的那块盘，异构盘构建 RAID 0 引起了“木桶效应”。RAID 0 固态盘阵列中吞吐性能最差的那块固态盘的吞吐量即为各盘的平均吞吐量 TP_0。可知，$TP_0<TP_1$，$TP_0\leqslant TP_2$。

使用 RODI 方法获得的吞吐量提升为：

$$\begin{aligned}\Delta TP&=(TP_{RODI}-TP_{RAID})\\&=[k\times TP_1+(m+n-k)\times TP_2]-(m+n)\times TP_0\\&=(m+n)\times(TP_2-TP_0)+k\times(TP_1-TP_2)\\&=(m+n-k)\times(TP_2-TP_0)+k\times(TP_1-TP_0)>0\end{aligned} \tag{4}$$

由此可见，采用 RODI 方法可以改善总体吞吐量。

5 实验

本节对在 RAID0 的情形下采用 RODI 方法前后的固态盘阵列的性能差异进行比较。实验所采用的硬件设备及其相应参数如表 4 所列。

表 4 实验硬件配置

硬件部件	参数
CPU	Intel i5 3.3GHz
Main Memory	Kingston DDR2 1066MHz 4GB * 2
SSD	SSD-830 * 1，SSD-OCZ * 1，SSD-530 * 1，SSD-850 * 1
存储接口	SATA 3
OS	Windows 7 Ultimate

5.1 实验方案

实验设备具备 3 个 SATA 3 接口，因此本文采用 SSD-850，SSD-530 和 SSD-OCZ 3 块不同品牌的固态盘来进行实验。实验通过 IOmeter 调整请求粒度与读比例来生成不同负载。IOmeter 生成的负载数据为 20GB。对于 RAID 0 的构建，采用 Windows 操作系统自带的磁盘工具将 3 块固态盘生成软 RAID 0，按条带化存放数据。

实验分别在顺序 I/O 负载和随机 I/O 负载下比较采用 RODI 方法前后的吞吐性能。由上一节实验结果可知，在大粒度场景下队列深度对吞吐量的影响可以忽略不计，因此实验中每个固态盘的队列深度为 2，3 块固态盘总计有 6 个 I/O 任务；读写粒度取值为 2MB，8MB 与 32MB。当采用 RODI 方法时，3 块固态盘按其上不同的负载 I/O 分为只读盘和混合读写盘。

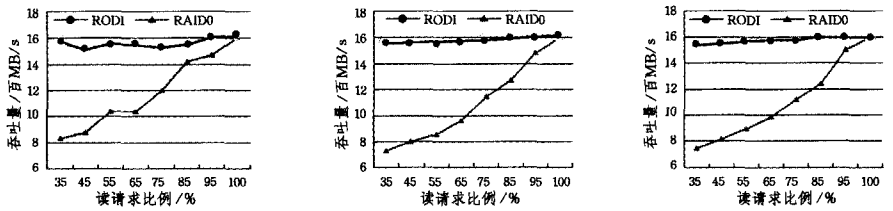
当使用一块盘作为只读盘(在该盘上读请求比例为

100%)，另外两块盘作为混合读写盘(读请求比例最小为 0，即为纯写盘)时，3 块盘构成的整体的读请求比例最小为$\frac{1}{3}$。随着两块混合读写盘的读请求比例的逐渐升高，整体的读请求比例逐渐升高。因此，实验测试 RODI 模式下的整体吞吐量时读请求比例由 35%变化到 100%。当使用两块盘作为只读盘(在此两块盘上读请求比例为 100%)，另外一块盘作为混合读写盘(读请求比例最小为 0)时，3 块盘构成的整体读请求比例最小为$\frac{2}{3}$。实验测试具有两块只读盘 RODI 模式下整体吞吐量时读请求比例变化由 70%变化到 100%。通过 IOmeter 可以控制 I/O 负载按照不同的读写比例混合，从而调节总负载的读负载比例。

在本节实验中，若采用 1 个只读盘，选取具有最高只读收益的 SSD-OCZ；若采用 2 个只读盘，则选取只读收益较高的 SSD-OCZ 和 SSD-530。

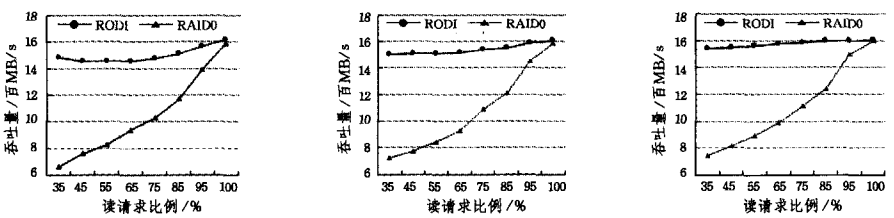
5.2 实验结果

在不同请求粒度与访问模式下，采用 RODI 方法前后在不同读负载比例下的吞吐量如图 5—图 8 所示。



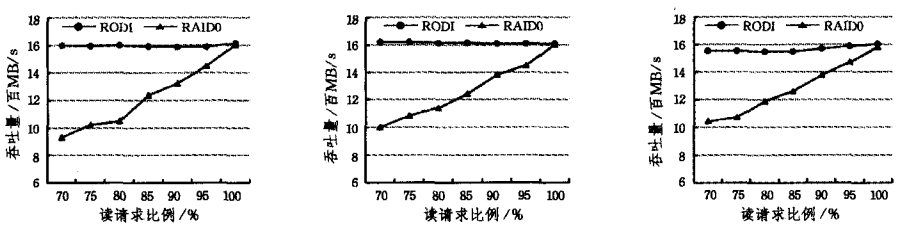
(a)I/O 粒度 2MB　(b)I/O 粒度 8MB　(c)I/O 粒度 32MB

图 5　采用 RODI 方法前后在不同 I/O 粒度下的吞吐量对比实验(1 个只读盘，顺序 I/O)



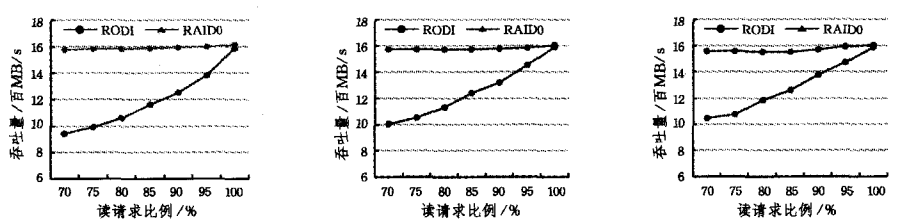
(a)I/O 粒度 2MB　(b)I/O 粒度 8MB　(c)I/O 粒度 32MB

图 6　采用 RODI 方法前后在不同 I/O 粒度下的吞吐量对比实验(1 个只读盘，随机 I/O)



(a)I/O 粒度 2MB　(b)I/O 粒度 8MB　(c)I/O 粒度 32MB

图 7　采用 RODI 方法前后在不同 I/O 粒度下的吞吐量对比实验(2 个只读盘，顺序 I/O)



(a)I/O 粒度 2MB　(b)I/O 粒度 8MB　(c)I/O 粒度 32MB

图 8　采用 RODI 方法前后在不同 I/O 粒度下的吞吐量对比实验(2 个只读盘，随机 I/O)

实验表明：采用 RODI 方法后在异构多盘环境下具有明显的吞吐量优势。当负载的读比例自 35%增加至 100%时，在各种粒度和访问模式下采用 RODI 方法均取得了较好效果。图 5 示出了采用 1 块只读盘 SSD-OCZ 和 2 块读写混合盘 SSD-530 和 SSD-850，在顺序 I/O 负载下开展的实验结果。图 5(a)示出了在 2MB 粒度时，在读请求比例是 35%的负载下，采用 RODI 方法前阵列整体吞吐量是 836MB/s，而采用 RODI 后，整体性能提升到 1567MB/s，随着读请求比例的提升，未采用 RODI 方法的系统其性能才逐渐上升到最佳状态，其余粒度类似。

随机 I/O 负载情形如图 6 所示，配置与图 5 对应的实验相同。在图 6(a)中，I/O 粒度为 2MB。当使用 RODI 时，作为只读盘的 SSD-OCZ 能按照其最大吞吐能力提供服务。SSD-530 和 SSD-850 作为读写混合盘，由第 3 节图 2(b)和图 2(d)，可知 SSD-530 的吞吐量随着读比例的增加而增加，而 SSD-850 的吞吐量与读比例的关系是在读占比在 0%～45%

时吞吐量会随着读比例增加而降低；当读占比大于 45%时又随着读比例的增加而增加。当使用 RODI 方法时，结合前文单盘特性，总吞吐量先有细微的降低，然后再随着读比例的增加而增长。当 I/O 粒度为 8MB 和 32MB 时，由图 2(b)可知，SSD-850 不再存在这种吞吐量比例随读占比先降低再上升的情况。因此，图 6(b)和图 6(c)RODI 的吞吐量曲线也变成随着读比例的增加而缓缓增加。不采用 RODI 且读负载在总请求中占比为 35%时，可认为每个盘负载分布相等，即所用盘负载读比例为 35%。由第 3 节可知，SSD-OCZ 在写比例较高的负载下吞吐量较低，由异构阵列“木桶效应”可知，不采用 RODI 方法的 RAID 0 的整体吞吐量较低。随着负载的读比例的提升，异构阵列“木桶效应”带来的影响降低，吞吐量也迅速开始提升。

图 7 与图 8 中使用了 2 块只读盘 SSD-850 和 SSD-OCZ，SSD-530 用于混合读写。注意，该场景的负载不同于前述实验，分离出来的只读数据负载大于前述实验相应的只读负载。由实验结果可知，系统总体吞吐量获得了较大提升，吞吐量最大提升 670MB/s。

综上所述，采用 RODI 方法对改善系统读写混合负载的吞吐量具有明显作用。在 RODI 方法适用范围内，当负载的写比例较高时，使用 RODI 能够较大程度地解决 RAID 0 由于异构阵列“木桶效应”所带来的问题；随着负载中读比例的增加，RODI 方法的效果减弱；当负载是纯读时，RODI 方法相比 RAID 0 不再起到改善吞吐量的作用。

结束语 本文针对异构固态盘阵列的背景，在请求粒度场景下提出了一种 I/O 只读负载分离方法 RODI，该方法通过合理放置只读数据来分离只读负载到合适的固态盘。与传统 RAID 0 相比，新方法在改善异构多盘阵列整体吞吐量上具有明显优势。

参 考 文 献

[1] FREITAS R F, WILCKE W W. Storage-class memory: the next storage system technology[J]. IBM Journal of Research and Development, 2008, 52(4/5): 439-448.

[2] AGRAWAL N, PRABHAKARAN V, WOBBER T, et al. Design tradeoffs for SSD performance[J]. Security & Privacy, IEEE, 2008, 7(2): 57-70

[3] LEE S W, MOON B. Design of flash-based DBMS: an in-page logging approach[C]// SIGMOD Conf.. 2007: 55-66.

[4] LEE S, MOON B, PARK C, et al. A case for flash memory SSD in enterprise database applications[C]// SIGMOD Conf.. 2008: 1075-1086.

[5] KIM H, AHN S. BPLRU: A buffer management scheme for improving random writes in flash storage[C]// USENIX Conf. on File and Storage Technologies. 2008: 1-14.

[6] KIM G J, BAEK S C, LEE H S, et al. LGeDBMS: a small DBMS for embedded system with flash memory[C]// VLDB Conf.. 2006: 1255-1258.

[7] ATHANASSOULIS M, AILAMAKI A, CHEN S, et al. Flash in a DBMS: where and how? [J]. IEEE Data Eng. Bull., 2011, 33(4): 28-34.

[8] KOLTSIDAS I, VIGLAS S. Data management over flash memory[C]// SIGMOD Conf.. 2011: 1209-1212.

[9] CHEN F, LEE R, ZHANG X. Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing[C]// High Performance Computing Architecture. 2011: 266-277.

[10] CHEN F, KOUFATY D A, ZHANG X. Understanding intrinsic characteristics and system implications of flash memory based solid state drives[C]// SIGMETRICS/Performance. 2009: 181-192.

[11] BJØRLING M, BONNET P, BOUGANIM L, et al. The necessary death of the block device interface[C]// Conf. on Innovative Data Systems Research. 2013.

[12] TSIROGIANNIS D, HARIZOPOULOS S, SHAH M A, et al. Query processing techniques for solid state drives[C]// SIGMOD Conf.. 2009: 59-72.

[13] SHAH M, HARIZOPOULOS S, WIENER J, et al. Fast scans and joins using flash drives[C]// DaMoN Conf.. 2008: 17-24.

[14] LI Y, XU J L, CHOI B. DigestJoin: exploiting fast random reads for flash-based joins[C]// Int'l Conf. on Mobile Data Management. 2009: 152-161.

[15] DO J Y, PATEL J M. Join processing for flash SSDs: remembering past lessons[C]// DaMoN Conf.. 2009: 1-8.

[16] PARK S S, LEE S W. Hash join in commercial database with flash memory SSD[C]// Proc. of Int'l Conf. on Computer Science and Info. Tech.. 2010: 265-268.

[17] LAI W, FAN Y, MENG X. Scan and join optimization by exploiting internal parallelism of flash-based solid state drives[C]// WAIM Conf.. 2013: 381-392.

[18] ROH H, PARK S, KIM S, et al. B^+-tree index optimization by exploiting internal parallelism of flash-based solid state drives [J]. VLDB Conf., 2012, 5(4): 286-297.

[19] PARK S Y, SEO E, SHIN J Y, et al. Exploiting Internal Parallelism of Flash-based SSDs[J]. Computer Architecture Letters, 2010, 9(1): 9-12.

[20] KIM J, SEO S, JUNG D, et al. Parameter-Aware I/O Management for Solid State Disks [J]. IEEE Trans. on Computers, 2012, 61(5): 636-649.

[21] PARK K, LEE D H, WOO Y, et al. Reliability and performance enhancement technique for SSD array storage system using RAID mechanism[C]// The 9th Int'l Symposium on Communications and Information Technology. 2009: 140-145.

[22] OH Y, CHOI J, LEE D, et al. Improving performance and lifetime of the SSD RAID-based host cache through a log-structured approach[C] // Workshop on Interactions of NVM/FLASH with Operating Systems and Workloads. 2013.

[23] BALAKRISHNAN M, KADAV A, PRABHAKARAN V, et al. Differential RAID: rethinking RAID for SSD reliability[J]. ACM Transactions on Storage (TOS), 2010, 6(2): 1-22.