一种固态盘的读写性能优化调度方法

朱 玥[1] 吴 非[1,2] 熊 钦[1] 谢长生[1,2]
(武汉光电国家实验室 武汉 430074)[1] (信息存储系统教育部重点实验室 武汉 430074)[2]

摘 要 相比于传统机械硬盘,基于NAND Flash的固态盘由于具有非易失性、高性能、低功耗等优点,被广泛应用于数据中心、云计算、在线事务交易等场景。然而,由于NAND Flash中的读操作速度远远快于写操作速度,当读写请求并发执行时,读请求可能被写请求阻塞,从而表现出极大的读延时。在许多以读请求为主的场合,尤其是在线事物交易中(读请求占总请求的比例超过90%),读延时的急剧增加严重影响了系统的整体性能。提出一种读写性能优化调度的策略,通过在闪存转换层之下动态调整读写请求的优先序列,使读性能获得显著的提升。实验中,通过对固态盘仿真器的设计与实现,对读写调度策略的有效性进行了系统的评估。实验结果表明,在该调度策略下,系统中读延时的最大值和平均值均得到了显著的减少,且降幅分别达到了72%和41%。

关键词 闪存,固态盘,数据调度,队列管理
中图法分类号 TP303 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.06.008

Read-Write Performance Optimization Scheduling Scheme for SSD

ZHU Yue[1] WU Fei[1,2] XIONG Qin[1] XIE Chang-sheng[1,2]
(Wuhan National Laboratory for Optoelectronics, Wuhan 430074, China)[1]
(Key Laboratory of Data Storage System, Ministry of Education, Wuhan 430074, China)[2]

Abstract Compared with traditional hard disk drives (HDDs), NAND-Flash-based solid-state drives (SSDs) are non-volatile and can provide better performance as well as lower power consumption. Therefore, they have achieved extensive application in data centers, cloud computing and online transaction trading, etc. However, in NAND Flash memory, the speed of read operation is significantly faster than the write operation. Hence, for a concurrent workload with a mixture of read and write requests, reads may be blocked by writes, which exhibites an enormous read latency. In many read-intensive applications, especially the online transaction trading, in which the proportion of read requests is than 90%, the sharp increase of the read latency influences the overall performance of the system severely. In this paper, we proposed a read-write performance optimization scheduling scheme which achieves remarkable improvement about the read performance by dynamically adjusting the priority sequence of read and write requests beneath the flash translation layer. In the experiment, we designed and built an SSD simulator to evaluate the effectiveness of the scheduling scheme. Experimental results show that by implementing the proposed scheme, the maximum and the average read latency in the system are substantially reduced, with the reduction of 72% and 41%, respectively.

Keywords Flash memory, SSD, Data scheduling, Queue management

1 引言

基于NAND Flash的固态盘具有非易失性、能耗低、访问速度快等诸多优良的特性[1],因而被广泛应用于各个场景。与传统的机械硬盘相比,基于NAND Flash的固态盘不受限于机械部件的速度,因此可以提供更高的I/O性能,并有望缓解在许多重要的数据密集型应用中的I/O瓶颈问题。然而,由于在NAND Flash中读写操作的速度不对称,当读写并发执行时,一部分读请求可能被写请求阻塞而拥有10倍以上的读延时,从而造成读性能的剧烈波动[6]。在许多以读请求为主并且对读请求的响应时间有较高要求的场合,尤其是在线事物交易中,这种读性能的剧烈波动严重影响了系统的整体性能。

传统的I/O调度器建立在闪存转换层(Flash Translation Layer,FTL)中,通过对垃圾回收[2-3]、地址映射[4-5]以及并行算法[7-9]等方面的设计,在提高I/O性能方面取得了显著的成果。然而,在FTL中的读写调度具有一定的局限性。FTL中的调度器可以通过将可能存在读写冲突的请求尽可能分配到不同的物理地址,来减少请求间的冲突,提高系统性能。但

到稿日期:2016-11-11 返修日期:2017-01-05 本文受国家自然科学基金(61300047),中央高校基本科研业务费(HUST:2016YXMS019)资助。
朱 玥(1994-),女,硕士生,主要研究方向为固态存储技术,E-mail:yuezhu@hust.edu.cn;吴 非(1975-),女,博士,副教授,主要研究方向为计算机系统结构、新型存储器件及体系结构、大规模存储系统能耗优化和高效能策略分析等,E-mail:wufei@hust.edu.cn(通信作者);熊 钦(1990-),男,博士生,主要研究方向为固态存储、实时分析;谢长生(1957-),男,博士,教授,博士生导师,主要研究方向为计算机系统结构、新型存储技术、网络存储系统等。

由于读请求的物理地址具有不可修改性，读写请求间的冲突不能完全避免。相比之下，在 FTL 下的调度器由于能够基于物理地址对读写请求进行操作，因此更灵活。

针对上述问题，提出一种新型的读写性能优化调度方法，区别于传统的 I/O 调度器，该方法在 FTL 下实现。调度器能够根据请求的物理地址对存在读写冲突的请求进行判断，并动态地调整读写请求的执行顺序，从而在产生读写冲突时优先执行速度较快的读请求，以写性能的轻微下降为代价获取读性能的显著提高，进而得到系统整体性能的提升。在本文所提出的调度策略中，还充分考虑到写饥饿、读写相关性以及请求间公平性的问题，对调度器的功能进行了完善。

2 基于 NAND Flash 的固态盘

2.1 NAND Flash 存储原理

NAND Flash 将信息存储在一系列由浮栅晶体管组成的存储单元(memory cell)中[12]，如图 1 所示。一个存储单元存储 1bit 数据的 NAND Flash 称为 SLC，存储 2bit 数据的称为 MLC，存储 3bit 数据的称为 TLC。

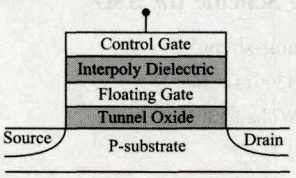


图 1 NAND Flash 存储单元

如同其他 MOS 晶体管，NAND Flash 存储单元顶部为控制栅(control gate)。但不同于其他 MOS 晶体管，在其控制栅下存在着一个被氧化层完全包围的浮栅(floating gate)。浮栅位于控制栅和 MOSFET 通道之间，由于被其周围的氧化层进行了电子隔离，存放在浮栅中的电子一直存储在其中，不会因为掉电而消失，因此 NAND Flash 具有非易失性。

当浮栅中的电子数高于阈值时，对应的是晶体管逻辑“0”的状态；当浮栅中的电子数低于阈值时，对应的是晶体管逻辑“1”的状态。因此，通过控制往浮栅中注入的电子数，可以向晶体管中写入不同数据。

在 NAND Flash 中，存在读、写和擦除 3 种基本操作。

读操作：读操作通过判断浮栅中电子数是否高于阈值来判断晶体管的状态为“0”或“1”。如图 2(a)所示，在进行读操作时，将 P 型衬底(P-substrate)接地，并在控制栅加正电压 V_{ref}(Read)。若浮栅中电子数低于阈值，则该电压能够让源极(source)和漏极(drain)导通；若浮栅中电子数高于阈值，由于浮栅中电子产生的电场(E_2)削弱了给控制栅加正电压所产生的电场(E_1)，从而源极和漏极无法导通[10]。通过检测源极和漏极是否导通，便可以判断晶体管的状态。若导通，则晶体管的状态为“1”；若不导通，则晶体管的状态为“0”。

写操作：写操作是指将晶体管的状态由“1”变为“0”的操作。如图 2(b)所示，将衬底接地，并在控制栅加正电压 V_{pp}(Program)，通过隧道效应使电子穿过氧化层注入浮栅，便可将晶体管的状态置为“0”。

擦除操作：与写操作相反，擦除操作是指将晶体管的状态由“0”变为“1”的操作。如图 2(c)所示，将控制栅接地，并在衬底上加正电压 V_{pp}(Erase)，通过隧道效应使电子离开浮栅，便可将晶体管的状态置为“1”。

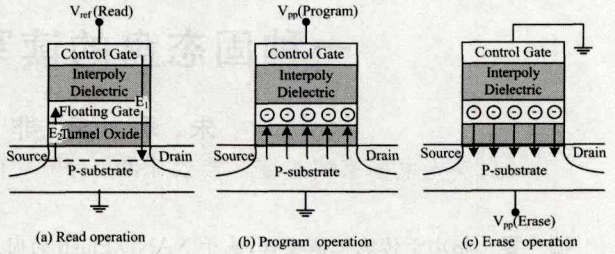


图 2 NAND Flash 读、写、擦除操作

读操作的过程只涉及到检测浮栅中电子的数目，以判断储存的是“0”还是“1”；而写操作的过程涉及到通过施加编程电压，使电子通过隧道氧化层进入浮栅中。由于原理上的不同，在 NAND Flash 中，写操作的时间远大于读操作的时间。NAND Flash 读、写和擦除操作时间的典型值[11]如表 1 所列。

表 1 NAND Flash 读、写、擦除操作典型值/μs

Attribute & cell types	SLC	MLC	TLC
Page read time	20	100	100
Page write time	200	1400	2400
Block erase time	3000	3000	4000

2.2 NAND Flash 层次结构

闪存介质具有芯片(chip)-晶圆(die)-分组(plane)-块(block)-页(page)的五级层次结构。

页：如图 3 所示，将若干个存储单元的控制栅连接在一条字线(word line)上，则这若干个存储单元共同组成一个页[12]。页是 NAND Flash 中读写操作的基本单元。

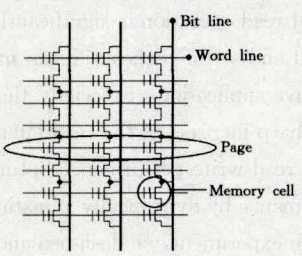


图 3 NAND Flash 中块的组织架构

块：将若干个页按照图 3 的方式组织起来，相邻存储单元的源极和漏极相连，便构成一个块。由于在一个块中，所有存储单元共用一个衬底，因此块是 NAND Flash 中擦除操作的基本单元。

分组：如图 4 所示，将若干个块按照一定的方式组织起来，便构成一个分组[14]。在每个分组中还设有一个或多个寄存器，以提高读写速度。

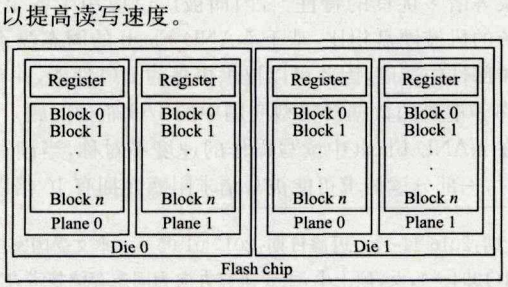


图 4 NAND Flash 中芯片的组织架构

晶圆：若干分组按照一定的方式组织起来，构成一个晶圆。在每个晶圆内部，有一条工作状态信号线，可以用来查询

各晶圆当前的工作状态。由于具有该工作状态信号线,晶圆可以独立地接收和执行 Flash 命令,而晶圆之下的层次并不具备此功能。因此,晶圆是接收和执行 Flash 命令的基本单元。除了读、写、擦除这 3 个基本命令外,闪存厂商还提供了一些高级命令,其中的一种多分组操作命令(multi-plane)能够对同一晶圆上的多个分组同时进行读或写,但在一个晶圆内不能同时执行不同类型的操作。因此,当对一个晶圆内的某个页进行写操作时,不能对其中的其他页进行读操作。

芯片:若干晶圆以一定的方式组织起来便构成一个芯片。为了节约成本和空间,一个芯片中的各晶圆共用一套外围电路和外部信号线。

2.3　闪存转换层

SSD 由接口、缓存、处理器、闪存控制器以及闪存芯片等部分组成[7],其基本框图如图 5 所示。

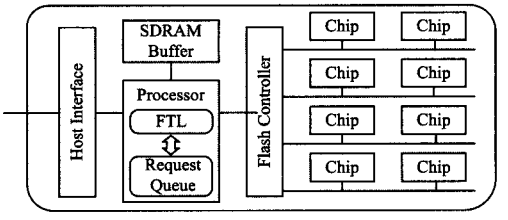


图 5　NAND Flash 中块的组织架构

位于处理器中的闪存转换层是 SSD 的重要组成部分,它主要包含 3 个功能:地址映射、垃圾回收和损耗均衡。

地址映射:Flash 的基本原理决定了其在进行写操作时只能将晶体管的状态由"1"变为"0",而将晶体管的状态由"0"变为"1"只能通过擦除操作来完成。然而,由于写操作和擦除操作的基本单位不同,并且为了保证请求的响应速度,Flash 不能在原地进行数据更新。文件系统下发读/写请求的地址为逻辑地址,闪存介质中页的地址为物理地址,在 FTL 内部维护着逻辑地址和物理地址间的映射表。当接收到文件系统的请求时,FTL 通过查询映射表将请求的逻辑地址转换为物理地址,再由闪存控制器根据物理地址执行请求。当需要对某个页中的数据进行更新时,可以把更新的数据写到一个空白页中,并修改映射表中对应的映射关系,被更新的页置为无效页。

垃圾回收:当执行了大量的更新操作后,闪存中会出现较多的无效页,由于无法对无效页进行写操作,无效页的增加会减少 SSD 中的可用空间。通常,在可用空间低于一定阈值时,会触发垃圾回收操作。垃圾回收操作是指将无效页集中起来统一进行擦除以释放 SSD 空间的操作。

损耗均衡:在闪存中,每个存储单元只有有限的写/擦除次数(P/E Cycle),当写/擦除次数达到一定数量时,存储单元就会变得不可靠,因此不能够使用[13]。为了避免某些存储单元因为被反复执行写/擦除操作而无法使用,需要尽可能地将写/擦除操作均衡地分布在各存储单元中,这便是损耗均衡。

3　读写调度器的设计与实现

3.1　基本思想

NAND Flash 的存储原理和层次结构决定其它存在两个特性。

1)读写速度不对称:在 NAND Flash 中,写操作的延时远大于读操作的延时;

2)晶圆是接收和执行 Flash 命令的基本单元:当对晶圆中的某个页进行写操作时,不能对同晶圆内的其他页进行读操作。

以上两个特性导致了一个问题:目标地址位于同一晶圆内的读写请求之间可能存在冲突,导致读请求被写请求所阻塞,从而严重影响系统的读性能。

由于读延时和写延时存在着高达一个数量级的倍数关系,因此,对于被写请求所阻塞的读请求来说,读延时可能达到不被阻塞时的 10 倍以上;相反,对于被读请求所阻塞的写请求来说,写延时相对于不被阻塞时可能只有细微的增长。

为了突出说明上述问题且不失普遍性,以图 6 为例进行详细的阐述。如图 6 所示,读请求"R"和写请求"W"在队列中依次等待执行。参考表 1,读操作的时间为 20μs,写操作的时间为 200μs。同时,为了让结果更加直观,约定:1)该队列中的读写请求间相互阻塞;2)忽略命令和数据传输时间。

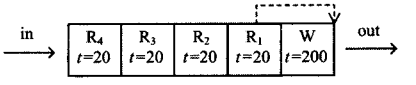


图 6　读写调度的基本思想

图 6 中,写请求"W"将首先被执行,在其执行过程中,读请求"R_1"至"R_4"由于与"W"存在冲突而被阻塞。通过计算可知,此时读请求"R_1"的延时高达 220μs,达到了不被阻塞时的 11 倍。在对读请求响应时间要求较高的系统中,如此大的读延时显然是不可接受的。

相对地,若通过调度优先执行读请求"R_1",则"R_1"的读延时由 220μs 减少至 20μs,与调度前相比,读延时的降幅高达 91%。与此同时,写请求"W"由于被推后执行了 20μs,其延时由 200μs 增加至 220μs,增加幅度仅为 10%。

基于以上分析,本文提出一种在 FTL 下的读写调度的方法。如图 6 中的虚线部分所示,通过调度,优先执行速度较快的读请求"R_1",推后执行速度较慢的写请求"W",从而在牺牲小部分写性能的前提下显著提升系统的读性能。此外,由图 6 可知,"R_1"和"W"间位置的交换并不影响位于其后的读请求"R_2","R_3"和"R_4",因此,调度范围之外的请求延时仍可保持不变。

3.1.1　写请求推后执行的可行性

在固态盘内,写请求下发后,数据不需要立即写入到介质中,而被暂时存放在缓存里,等到合适的时机再写入。因此,在对读写请求执行序列进行调整的过程中,不需要额外的空间来存放待写入的数据,这为读写调度的实现提供了条件。

3.1.2　擦除操作

与写操作相比,擦除操作具有更大的延时。然而,由于擦除操作通常由垃圾回收触发,一般情况下,在执行垃圾回收时,可用空间已经降低到一定的阈值。若此时还将擦除操作推后执行,可能会导致可用空间不足,从而严重影响固态盘的性能。因此,本文不将擦除操作列入调度范围。

3.2　总体架构

在本设计中,读写调度器需要依次查看预取队列中各请求的类型、目标地址等信息,并将发生读写冲突的请求按照策略进行调度。由于对读写冲突的判断是基于请求的物理地址而非逻辑地址,并且鉴于闪存转换层可以完成请求从逻辑地

址到物理地址的映射，本设计将读写调度器放置在闪存转换层之下的位置。调度完成后，请求再由闪存控制器依次下发至闪存介质中执行。因此，读写调度器在系统中位于闪存转换层之下、闪存控制器之上，如图 7 中虚线框部分所示。

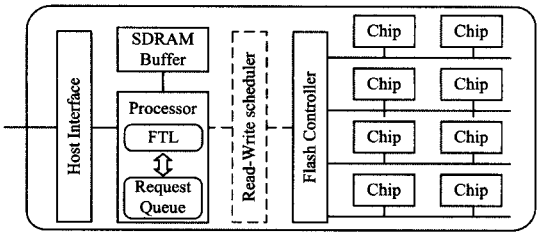


图 7 总体架构

在读写调度器中，维护着一个预取队列（称作“等待队列”）。由文件系统下发的请求通过闪存转换层完成目标地址从逻辑地址到物理地址的映射后，便进入等待队列中等候闪存控制器依次执行。此时，读写调度器需要依照调度策略来对等待队列中存在冲突的读写请求进行调度，从而改变请求的执行顺序，优先执行速度较快的读请求，以显著提高系统的读性能。

3.3 调度策略

通过对 NAND Flash 中读写操作基本特性的分析，在读写调度基本思想的基础上提出如下调度策略。

（1）读请求尽可能提前

由 3.1 节的分析可知，通过调度，适当地优先执行读请求，可以以轻微牺牲写性能为前提获得读性能的显著提升。因此，在读写调度器的设计过程中，读请求被赋予了较高的执行优先级。

（2）写延时不超过上限

虽然读请求被赋予了较高的执行优先级，但事实上读请求不能被无限制地提前，否则写请求可能会因被不断地推后而最终导致写饥饿的情况，这同样是不可接受的。因此，在本设计中，写请求的延时存在某一上限，即写延时上限（write time upper bound），一旦读写调度器计算出某一写请求的延时超过写延时上限，该写请求将获得较读请求更高的执行优先级，从而使得在预取队列中位于其后的读请求不能继续向前移动。本文将请求的延时定义为请求从产生到执行完成的时间。值得注意的是，此处所描述的写请求上限为软上限，如果某一写请求在产生时，调度器计算出其延时已超过写延时上限，则无需将该写请求在队列中向前移动，以满足写延时上限的要求。

（3）若存在读写相关性，则先写后读

在读写调度器的实施过程中，还需要考虑对读写相关性的检测。当一个读请求和一个写请求的目标地址相同（为同一个页）时，则称这两个请求具有读写相关性。在等待队列中，如果一个读请求之前存在着与它具有相关性的写请求，那么需要遵循先写后读的原则，不能够将读请求移动到该写请求之前，否则读出来的可能是该页执行擦除后没有进行写入的数据（一般为全 1），而不是执行写操作后该页中的数据，从而造成读数据错误。

（4）相同类型的请求之间位置关系不变

为保证公平性，在读写调度过程中，两个相同类型的请求（两个读请求或两个写请求）之间的位置关系不能够发生改变。在等待队列内将某一读请求向前移动的过程中，若遇到其他读请求，则停止向前移动，否则会破坏两个读请求之间的公平性。

3.4 调度方案

读写调度通过固定写请求而将读请求向前移动的方法实现。在等待队列中，调度器对到来的新请求逐个进行判断。若新请求为读请求，则将其逐步向前移动，直到满足调度策略中的各条件；若新请求为写请求，则不对其进行移动。

为了更加直观地理解读写调度的过程且不失普遍性，以图 8 为例来阐述读写调度策略的实现。

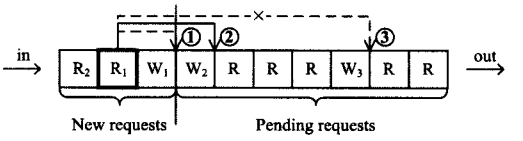


图 8 等待队列中请求的调度

图 8 展示了一个存放待执行请求的等待队列。某一时刻，等待队列中到来了 3 个新请求（new requests）：1 个写请求“W_1”以及 2 个读请求“R_1”和“R_2”。

调度器首先判断第一个新请求“W_1”的类型，由于“W_1”为写请求，因此不对其进行操作，继续判断下一个请求。

第二个新请求“R_1”为读请求，因此调度器在等待队列中将其逐步向前移动。过程①中，将“R_1”移至“W_1”之前，通过计算发现“W_1”的延时不超过写延时上限，并且“R_1”与“W_1”之间不存在读写相关性，因此记录“R_1”在此时的位置，并将其继续向前移动；过程②中，将“R_1”移至“W_2”之前，通过计算发现“W_2”的延时也不超过写延时上限，并且“R_1”与“W_2”之间也不存在读写相关性，因此记录“R_1”在此时的位置，并将其继续向前移动；过程③中，将“R_1”移至“W_3”之前，通过计算发现“W_3”的延时超过了写延时上限。因此，最终将“R_1”移至上一次记录的位置（即“W_2”之前，如图 8 中的过程②所示）。

事实上，即使在过程③中将“R_1”移至“W_3”之前，“W_3”的延时也没有超过上限并且“R_1”和“W_3”之间也不存在读写冲突。根据调度过程不改变相同类型请求之间位置关系的原则，仍不能将“R_1”在等待队列中继续向前移动，否则会破坏“R_1”和中间 3 个“R”请求之间的公平性。

完成对“R_1”的移动后，调度器继续判断下一个新请求的类型。由于第三个新请求“R_2”也为读请求，因此对“R_2”执行与“R_1”相同的操作，根据读写调度的策略将其在等待队列中逐步向前移动至合适的位置。

4 实验结果及分析

4.1 实验设置

为了验证本文所提出的读写调度方法的有效性，设计并搭建了固态盘仿真器，以实现对固态盘内读写调度及请求执行过程的模拟。

该固态盘仿真器分为请求产生、等待队列、读写调度、请求执行以及结果统计 5 个模块，如图 9 所示。其中，等待队列及读写调度模块依照调度策略实现，此处不再赘述。

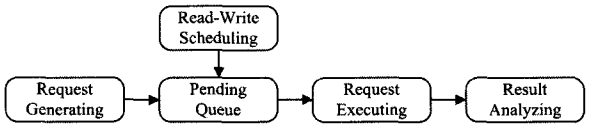


图 9　固态盘仿真器模块

请求的产生:在固态盘中,上层文件系统将请求下发,并由闪存转换层将请求的目标地址由逻辑地址映射到物理地址。由于本文提出的读写调度方法需要在闪存转换层之下实现,若采用 trace 进行测试,则需要通过 FTL 将其目标地址由逻辑地址转换为物理地址后再进行测试。由于采用不同的 FTL 算法可能导致不同的仿真效果,从而不便于进行结果的比较,因此本设计采用请求产生模块直接生成目标地址为物理地址的读、写请求。同时,为了突出读写冲突的问题,不考虑擦除操作。为了对真实负载进行模拟,请求的产生分为两个部分:一部分由均匀的、周期性产生的请求(periodical request)组成,另一部分由间隔一段时间、爆发性产生的请求(burst)组成。为保证固态盘中通道及闪存介质资源的利用率,请求产生的具体参数如表 2 所列。

表 2　请求产生相关参数设置

Parameters	Values
Periodical request period/μs	40
Number of periodical request(s) per period	1
Burst period/μs	1200
Number of request(s) per burst	10

请求的执行:请求的执行分为命令地址传输、数据传输以及读/写介质 3 个阶段。其中,由于命令地址传输阶段的时间相对于其他两个阶段十分短暂,因此将其忽略不计[14]。数据传输、介质读/写所需时间以及固态盘仿真器通道的内部参数[14]如表 3 所列。其中,页是读/写的最小单位,也是请求中目标地址所对应的单位;晶圆是接收和执行 Flash 命令的最小单位,因此,晶圆之间请求的执行相互独立,而晶圆内部的读写请求相互冲突,并且各晶圆共用通道资源。

表 3　请求执行相关参数设置

Parameters	Values
Data transmission/μs	10
Page read/μs	20
Page write/μs	200
Number of channel	1
Chip(s) per channel	4
Die(s) per chip	2
Plane(s) per die	2
Block(s) per plane	2048
Page(s) per block	64

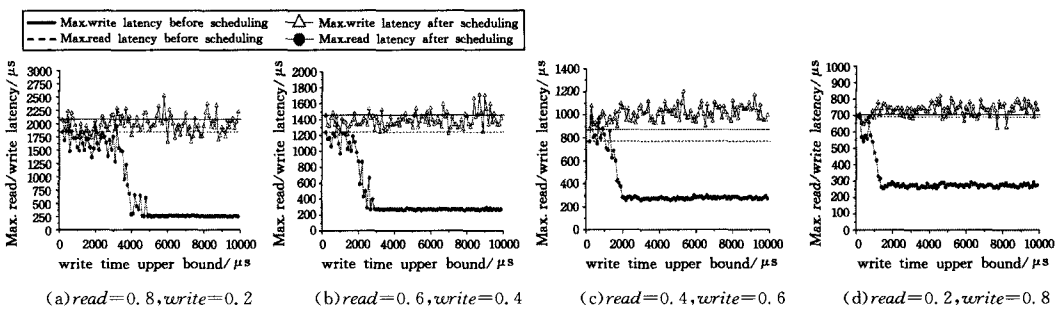
结果的统计:为便于对请求进行追踪,固态盘仿真器在产生请求的过程中为每个请求分配了一个 ID;并且,为了获取更准确的统计结果,在统计过程中,对某一段 ID 范围内的请求执行结果进行收集,而非某一时间段内执行完成的请求。通过对指定 ID 范围内请求的特征进行统计和处理,最终得到读写性能的相关参数。

在多通道系统中,由于各通道之间请求的执行相互独立,因此,单通道内读写调度的仿真结果对多通道系统同样适用。因而,本仿真器设定为对单通道内的读写调度以及请求执行过程进行模拟,并且为了使读写冲突的问题更加突出,不考虑固态盘中各级的并行性。

为了体现不同读写比情况下的读写调度结果,本文选取读请求比例为{0.8,0.6,0.4,0.2}时的情况来分析不同读请求比例时调度结果的变化趋势,并分别从最大读/写延时和平均读/写延时两个方面对读写性能进行评估。最后,通过对比调度前后的读写性能来对读写调度方法的有效性进行评估。

4.2　结果分析

图 10 示出了读请求比例分别为{0.8,0.6,0.4,0.2}时,最大读/写延时(Max. read/write latency)随写延时上限(write time upper bound)的变化而变化的情况。



(a)$read$=0.8,$write$=0.2　(b)$read$=0.6,$write$=0.4　(c)$read$=0.4,$write$=0.6　(d)$read$=0.2,$write$=0.8

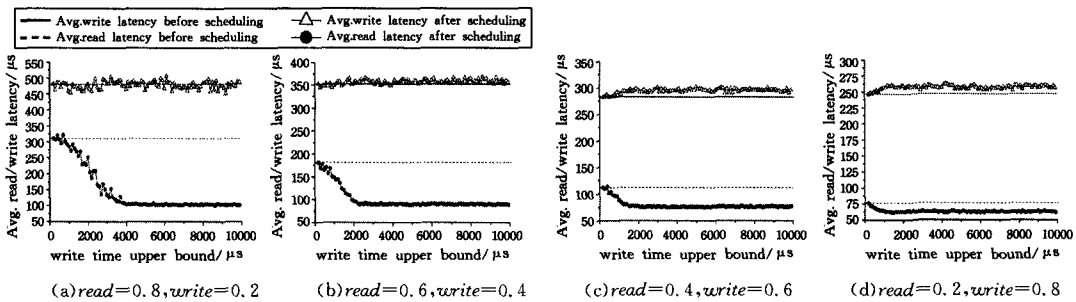
图 10　读请求比例分别为{0.8,0.6,0.4,0.2}时最大读/写延时随写延时上限的变化情况

写延时上限表示在读写调度的过程中为写延时所设置的上限。写延时上限的增加,实际上意味着调度器在等待队列中对读请求移动幅度的增大。一个极端情况是当写延时上限增加到无穷大时,只要读写请求不存在相关性,可无限制地将读请求移到写请求之前,而不需要考虑写请求的延时是否超过上限;另一个极端情况是当写延时的上限减小到 200μs 及以下时,由于写请求的执行本身需要 200μs 以上的时间,因此在该情况下等待队列中所有写请求的延时都超过了写延时上限,所以等待队列中的所有读请求均不能向前移动,即该情况下不存在读写调度。因此,本文取写延时上限为 200μs 时所对应的读/写延时为调度前的读/写延时,分别以与横坐标轴平行的虚线和实线表示,图中曲线的其余部分表示调度后的读/写延时随写延时上限的变化而变化的情况,横坐标从左至右表示调度幅度的逐渐增加。

图 10 中,在读请求比例分别为{0.8,0.6,0.4,0.2}的情况下,调度前的最大读延时(μs)为{1845,1238,770,689},最大写延时(μs)为{2075,1453,876,706}。调度后,在这 4 种情况下,随着写延时上限的增加,最大读延时先显著减小,随后逐渐趋于稳定;与此同时,最大写延时基本保持稳定,并伴有小幅的波动。在 4 种情况下,最大读延时基本稳定后,读延时

的均值(μs)分别为{257,271,276,272},写延时的均值(μs)分别为{1987,1398,1021,736}。由此可以看出,进行读写调度后,最大读延时获得了显著的减小,4 种情况下的减小幅度分别为{86%,78%,64%,61%},平均减小了 72%;与此同时,最大写延时基本保持稳定,在前两种情况下甚至存在轻微的减小,4 种情况下最大写延时的增长幅度分别为{-4%,-4%,12%,2%},平均增长了 2%。

图 11 示出了读请求比例分别为{0.8,0.6,0.4,0.2}时,平均读/写延时(Avg. read/write latency)随写延时上限(write time upper bound)的变化而变化的情况。在读请求比例分别为{0.8,0.6,0.4,0.2}的情况下,调度前的平均读延时(μs)为{296,181,112,76},平均写延时(μs)为{471,352,282,246}。调度后,在这 4 种情况下,随着写延时上限的增加,平均读延时先显著减小,随后逐渐趋于稳定;与此同时,平均写延时基本保持稳定,只是随着写延时上限的增加有轻微的增长。在 4 种情况下,平均读延时基本稳定后,读延时的均值(μs)分别为{104,90,76,63},写延时的均值(μs)分别为{478,358,295,257}。由此可以看出,进行读写调度后,平均读延时获得了显著的减小,4 种情况下的减小幅度分别为{65%,50%,32,17%},平均减小了 41%;与此同时,平均写延时只存在轻微的增长,增长幅度分别为{1%,2%,5%,4%},平均增长了 3%。



(a)$read=0.8, write=0.2$ (b)$read=0.6, write=0.4$ (c)$read=0.4, write=0.6$ (d)$read=0.2, write=0.8$

图 11 读请求比例分别为{0.8,0.6,0.4,0.2}时平均读/写延时随写延时上限的变化情况

结束语 本文针对 NAND Flash 中由于读写速度不对称而导致的读性能严重下降的问题,提出了一种读写性能优化调度的策略。该策略指出,可以通过适当地提高读请求的优先级,改变读写请求的执行顺序,优先执行读请求,从而在牺牲一定写性能的前提下获得读性能的大幅度提高。为了验证该读写调度策略的有效性,搭建了固态盘仿真器来对读写调度及请求执行的过程进行模拟。实验结果表明,该读写调度策略能够在写性能轻微下降(最大写延时和平均写延时分别增长 2%和 3%)的前提下,有效提升系统的读性能(最大读延时和平均读延时分别减少 72%和 41%)。未来的工作包括:1)进一步优化读写调度策略,为系统创建合适的排队模型,从而增强读写调度方法对系统的适应性,提高整体性能;2)在实际的 SSD 开发平台上对读写调度策略进行实现及验证。

参 考 文 献

[1] QIN Z, WANG Y, LIU D, et al. Real-time flash translation layer for NAND flash memory storage systems [C]// Proc of IEEE RTAS'12. Piscataway, NJ: IEEE, 2012: 35-44.

[2] JUNG M, CHOI W, SRIKANTAIAH S, et al. HIOS: A host interface I/O scheduler for solid state disks [J]. ACM SIGARCH Computer Architecture News, 2014, 42(3): 289-300.

[3] LEE J, KIM Y, SHIPMAN G M, et al. Preemptible I/O scheduling of garbage collection for solid state drives [J]. IEEE Transactions on CADCS, 2013, 32(2): 247-260.

[4] ABDURRAB A R, XIE T, WANG W. DLOOP: A flash translation layer exploiting plane-level parallelism [C]// Proc of IEEE IPDPS'13. Piscataway, NY: IEEE, 2013: 908-918.

[5] GAO C, SHI L, ZHAO M, et al. Exploiting parallelism in I/O scheduling for access conflict minimization in flash-based solid state drives [C]// Proc of IEEE MSST'14. Piscataway, NY: IEEE, 2014: 1-11.

[6] PARK S, SHEN K. FIOS: A fair, efficient flash I/O scheduler [C]// Proc of USENIX FAST'12. Berkeley, CA: USENIX, 2012: 1-15.

[7] PARK S, SEO E, SHIN J Y, et al. Exploiting internal parallelism of flash-based SSDs [J]. Computer Architecture Letters, 2010, 9(1): 9-12.

[8] 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]// Proc of IEEE HPCA'11. Piscataway, NY: IEEE, 2011: 266-277.

[9] PARK S, SHEN K. A performance evaluation of scientific I/O workloads on flash-based SSDs[C]// Proc of IEEE CLUSTER'09. Piscataway, NY: IEEE, 2009: 1-5.

[10] CAI Y, LUO Y, GHOSE S, et al. Read disturb errors in MLC NAND flash memory: Characterization, mitigation, and recovery [C]// Proc of IEEE DSN'15. Piscataway, NY: IEEE, 2015: 438-449.

[11] YANG M C, CHANG Y H, TSAO C W, et al. Utilization-aware self-tuning design for TLC flash storage devices [J]. IEEE Trans. on Very Large Scale Integration Systems, 2016, 24(10): 3132-3144.

[12] BEZ R, CAMERLENGHI E, MODELLI A, et al. Introduction to flash memory [J]. Proceedings of the IEEE, 2003, 91(4): 489-502.

[13] JUNG D, CHAE Y H, JO H, et al. A group-based wear-leveling algorithm for large-capacity flash memory storage systems [C]// Proc of ACM CASES'07. New York, NY: ACM, 2007: 160-164.

[14] HU Y, JIANG H, FENG D, et al. Exploring and exploiting the multilevel parallelism inside SSDs for improved performance and endurance [J]. IEEE Transactions on Computers, 2013, 62(6): 1141-1155.