**FAST I/O: QoS Supports for Urgent I/Os in NVMe SSDs**

作者：  
***Kyusik Kim，Seongmin Kim，Taeseok Kim***

*Dept. of Computer Engineering, Kwangwoon University, Seoul, Republic of Korea*

ICCIT: **International Conference on Intelligent Information Technology**

DOI：https://doi.org/10.1145/3385209.3385221

Publisher：Association for Computing Machinery New York, NY, United States

1. 摘要

NVMe SSDs通过大量的I/O提交队列同时调度I/O请求，积极发挥了ssd的并行性。虽然该架构可以提高NVMe ssd的I/O带宽和IOPS（每秒读写操作数），但无法保证每个I/O请求的QoS（服务质量）。特别是当提交队列中有很多I/O请求处于等待状态时，需要紧急处理的I/O请求可能会被延迟。在本文中，我们提出了一种几乎无延迟处理紧急I/O请求的方案。由于无法在主机中控制提交队列中的挂起I/O请求，我们使用主机内存缓冲区(HMB)作为处理紧急I/O请求的快速通道。该方案没有通过传统的I/O路径将紧急I/O请求发送到ssd，而是直接将它们插入到可以从主机和ssd访问的HMB中，从而消除了延迟。通过在仿真器上的实验，我们证明了所提出的方案能够显著降低平均延迟和尾延迟（如果在系统中引入实时监控，总会有少量响应的延迟高于均值，我们把这些响应称为尾延迟（Tail Latency））。

1. 背景
2. NVMe (Non-volatile memory express)是为固态硬盘(ssd)设计的最新逻辑接口之一，克服了传统接口(如高级主机控制器接口(AHCI))的局限性。与只有一个队列的传统接口不同，NVMe提供了许多可扩展的提交队列（SQ），用于ssd内的并行I/O处理。
3. 主机操作系统也进行了更改，以充分利用NVMe SSD架构。例如，Linux用多队列块层取代了现有的单队列块层，以响应NVMe的多个提交队列。
4. NVMe的特性--the host memory buffer (HMB), 它允许ssd在需要更多的DRAM时使用主机的一部分DRAM，它被视为主机和SSD控制器的共享区域，通过HMB通信可以立即处理紧急I/O请求。在支持HMB的情况下，SSD控制器可以通过NVMe协议支持的高速PCIe接口访问主机内存，将主机内存的一部分用作地址映射表或常规数据的存储缓存。
5. HMB可以用作映射表缓存或数据缓冲区，改善了无DRAM的ssd由于控制器中没有DRAM而导致的性能下降。

HBM: 它是只分配给SSD的主机DRAM的一部分，SSD的制造商决定如何使用它。大多数情况下，HMB由多个物理上连续的内存空间组成。

激活HMB需要两个步骤;

1. 通过主机内存缓冲首选大小(HMPRE)和主机内存缓冲最小大小(HMMIN)来确定HBM的大小。如果HMPRE不为零，则表示主机支持HMB。SSD所需的HMB大小不小于HMMIN，也不大于HMPRE。
2. 主机发出NVMe Set features命令用于激活HMB并发送一些与分配空间相关的信息，并且SSD成功响应请求
3. 动机和挑战

这种可扩展的架构提供了较高的I/O带宽和IOPS，但（1）不能为每个I/O请求提供QoS支持。如果提交队列中有大量挂起的I/O请求，则应该紧急处理的I/O请求不能立即得到服务。例如，当ssd在后台处理突发I/O工作负载时，需要低延迟的应用程序启动或前台进程执行可能会延迟。

（2）由于主机操作系统无法控制NVMe提交队列中的挂起的I/O请求，在这种情况下，块层上现有的I/O调度算法没有多大帮助。

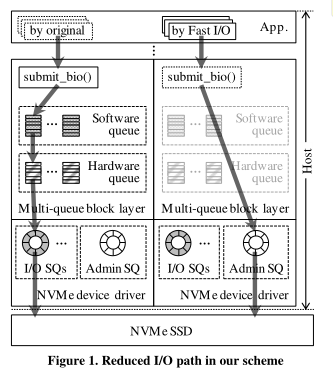
（3）在为紧急I/O请求提供QoS时，不能违反NVMe接口的协议。

解决方法：通过使用管理队列来解决这个问题，它是一个用于处理NVMe管理命令的队列。

1. 设计

通过使用HMB作为I/O快速通道处理紧急I/O。

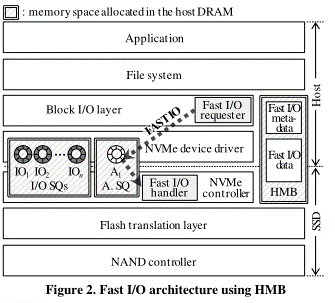
处理I/O和快速I/O的原始过程之间的区别如图1所示：



原始的I/O处理过程，I/O通过在多队列块层中的软件队列（按每个处理器（多核）分配的）和硬件队列（按每个I/O提交队列（I/O请求不同）分配的），然后通过NVM e device driver到达SSDs进行处理。

Fast I/O处理通过绕过软件队列和硬件队列，它们可以快速地发送到SSD。

使用HMB的Fast I/O架构如图2



两个关键模块--Fast I/O requester，Fast I/O handler

**Fast I/O requester**：位于主机侧，（1）判断请求是否是Fast I/O request，是的话还要进行相关信息的转化（将请求信息、bio结构中描述的扇区号和要读取的扇区数量转换为SSD的逻辑页单元）。

（2）发送FAST IO命令到Fast I/O handler 进化处理。

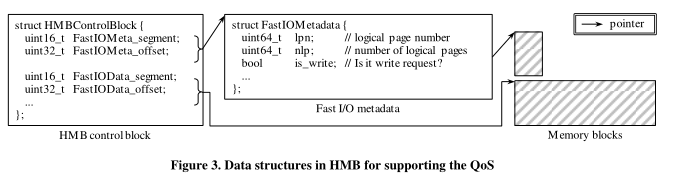
**Fast I/O handler**：位于NVMe SSD侧，基于Fast I/O 中的metadata，进行数据的复制（在主机pages和Fast I/O data间或NAND flash memory和Fast I/O data间）。

**HMB:**

**Fast I/O metadata，**接收Fast I/O requester的转化信息，具有起始逻辑页码(lpn)、逻辑页码的数量(nlp)以及请求类型是读还是写(is\_write)。

**Fast I/O data，**保存主机和SSD交换的数据。

数据结构如下图3



**整个读请求过程（通过Fast I/O）的步骤：**

**（1）在块层的入口处会通过submit\_bio()调用Fast I/O请求器，同时检查这些请求是否经过Fast I/O处理。检查的方法这里选择判断I/O进程的优先级。**

**（2）是的话，Fast I/O请求器会将请求信息、bio（Block I/O layer）结构中描述的要读取的扇区编号和扇区数量转换为固态硬盘的逻辑页单元。**

**（3）转化完成后，Fast I/O requester将转化信息写到HMB的Fast I/O metadata中。**

**（4）然后Fast I/O requester发送FAST IO命令到Fast I/O handler 进化处理。**

**（5）基于Fast I/O 中的metadata，handler 会从NAND flash memory 中复制数据到**

**Fast I/O data中，，并将完成信息入队到admin 完成队列中。**

**（6）主机收到完成信息后，主机就从 Fast I/O data中复制数据到主机的pages中，这些pages的位置信息由bio结构描述，然后调用 bio\_endio()完成这个请求。**

**对于写的话：**

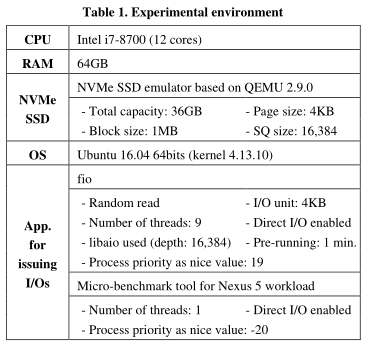
**第（3）和第（5）步不同：**

**在第（3）步中根据Fast I/O metadata中的is\_write为真，Fast I/O data会将主机中要写入的数据复制到这来，然后在执行（4）**

**（5）Fast I/O handler 将从Fast I/O data中复制数据到NAND flash memory中，其他相同**

1. 实验效果

实验环境



为了验证当SSD已经有I/O负载时，使用Fast I/O是否是为I/O提供快速通道的有效方法，我们进行了各种实验。

通过使用在Nexus 5 (Android智能手机之一)上收集的I/O负载来验证快速I/O的有效性。

在执行微基准测试工具之前，我们首先执行fio 1分钟，以在每个提交队列按预期将I/O请求放入队列后启动实验（为后续优先级调度做准备）。

微基准测试工具在三种情况下的实验：

（1） (Idle) the micro-benchmark tool is only executed and processed by the original procedure

（2）(Busy (original)) the micro-benchmark too executed with the fio and processed by the original procedure

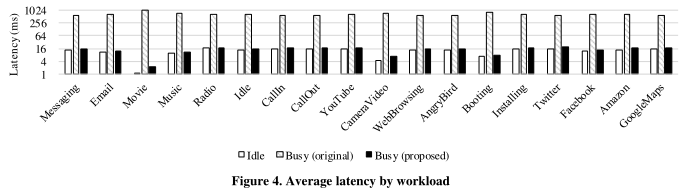
（3）(Busy (proposed)) the micro-benchmark too executed with the fio and fully processed by Fast I/O

第一个是SSD中没有I/O负载，直接用Fast I/O运行

第二个是SSD中有I/O负载并且由原始程序运行

第三个是SSD中有I/O负载但由Fast I/O运行

实验一：平均延迟

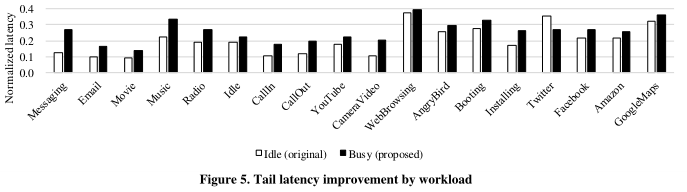


当SSD硬盘有I/O负载，且未使用Fast I/O时，所有工作负载的平均时延增加37.6~885.3倍，这意味着当前考虑进程优先级的CPU调度并不能充分缓解I/O处理延迟。

通过将每个工作负载发出的所有I/O处理为Fast I/O，时延提高了31.6~459.7倍。

**结论**：**就平均延迟而言，当SSD及其提交队列都有I/O负载时，Fast I/O充分发挥了作为I/O快速通道的作用。**

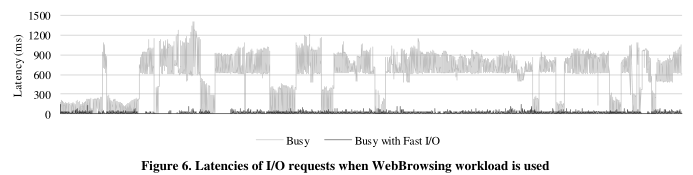
实验二：尾延迟，图5显示了将它们标准化为Busy(原始)的结果



通过应用Fast I/O，在所有工作负载下，尾延迟提高了2.5~7.2倍。

Busy(proposed)的尾部延迟(它执行fio和微基准测试工具)平均是Idle(只执行微基准测试工具)的1.4倍。

实验三：使用WebBrowsing工作负载时I/O请求的延迟（就尾延迟而言，这个工作负载是最糟糕的情况）



大多数I/O都被Fast I/O显著改善。

结论：Fast I/O不仅是一种提供比原始过程更快的I/O处理的有效方法，而且是一种保证最差结果的好方法。

1. 总结

为了让用户满意，在应用程序启动或前台进程执行时，减少I/O延迟非常重要。在本文中，我们提出了一种保证在NVMe ssd中紧急I/O请求QoS的方案。为此，我们通过NVMe的特性HMB在主机和控制器之间进行通信，减少了处理紧急I/O请求的I/O路径。

通过大量的实验，我们表明，当使用不同的智能手机工作负载时，提出的方案可以显著提高平均和尾部延迟。