智能手机应用的输入/输出特性及其对 eMMC 设计的影响

Deng Zhou
圣地亚哥州立大学 电子邮件：zhoud@rohan.sdsu.edu

Wen Pan, Wei Wang
圣地亚哥州立大学 电子邮箱 {pan, wang}@rohan.sdsu.edu

谢涛
圣地亚哥州立大学 电子邮件：txie@mail.sdsu.edu

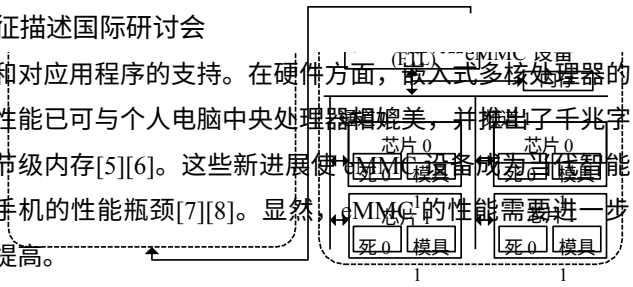
摘要--绝大多数智能手机使用 eMMC（嵌入式多媒体卡）设备作为存储子系统。最近的研究表明，存储子系统对智能手机应用程序的性能有重要影响。然而，人们对智能手机应用的块级 I/O 特性及其对 eMMC 设计的影响仍知 之 甚 少 。在这项研究中，我们收集并分析了 Nexus 5 智能手机上 18 个常见应用程序（如电子邮件和 Twitter）的块级 I/O 跟踪。我们观察到了一些 I/O 特性，并从中得出了 eMMC 设计的一些启示。例如，我们发现在 18 个跟踪记录中，有 15 个记录的大部分请求（44.9%-57.4%）是小型单页（4KB）请求。这意味着应快速处理小请求，以提高 eMMC 设备的整体性能。接下来，我们进行了一项案例研究，以演示如何应用这些含义来优化 eMMC 设计。受这两种影响的启发，我们提出了一种混合页面大小（HPS）的 eMMC。实验结果表明，HPS 方案可将平均响应时间缩短 86%，同时将空间利用率提高 24.2%。

I. 导言

智能手机的存储子系统通常通过闪存文件系统[1]或 Ext4 等块文件系统[2]使用 NAND 闪存。早期基于安卓系统的智能手机广泛使用 YAFFS2 作为默认文件系统，这是一种专用于在原始闪存上存储文件的文件系统。在 2010 年发布 Android 2.3（姜饼）平台后，当底层存储子系统从原始闪存改为 eMMC 设备时，基于 Android 的智能手机的默认文件系统被切换为 Ext4 [3]。eMMC 具有统一的协议，并为 Ext4 提供了传统的块设备接口。其控制器可在本地处理地址映射、损耗平衡和垃圾回收，这在很大程度上减轻了上述文件系统的负担。图 1 显示了 Nexus 5 等智能手机的 I/O 堆栈。智能手机应用程序的大部分文件和数据都由 SQLite 库管理，它是基于 Android 的智能手机的默认数据库管理系统（DBMS）。通常，应用程序的一次 I/O 活动会产生多个 SQLite I/O 请求，由内核层的虚拟文件系统（VFS）提供服务。每个 SQLite I/O 请求通常会调用由 Ext4 等块文件系统提供的系统调用函数，该函数会进一步生成 I/O 请求并将其发送到块层。块层负责调度这些 I/O 请求，最后将其分派给 eMMC 驱动程序。eMMC 驱动程序将每个 I/O 请求转化为 eMMC 控制器可理解的命令，并执行所有命令以满足 I/O 请求。

图 1：安卓 I/O 堆栈。

与基于闪存的固态硬盘（SSD）类似，eMMC 设备的行为类似于传统的块设备，因为它自己的闪存转换层（FTL）隐藏了所有闪存限制，如先擦除后*写入*。由于智能手机在空间、功耗和成本方面的限制，与固态硬盘相比，eMMC 设备的 FTL 和架构更简单，RAM 缓冲区也更小。因此，eMMC 设备的性能远远低于固态硬盘[4]。另一方面，智能手机的软件和硬件技术近和对应用程序的支持。在硬件方面，嵌入式多核处理器的性能已可与个人电脑中央处理器相媲美，并推出了千兆字节级内存[5][6]。这些新进展使 eMMC 设备成为当代智能手机的性能瓶颈[7][8]。显然，eMMC 的性能需要进一步提高。



要开发高性能的 eMMC 设备，就必须更好地了解智能手机应用的块级 I/O 特性。虽然最近有一些关于智能手机存储子系统的研究[9][7][4][10]在文献中有所报道，但没有一项研究关注如何根据智能手机应用程序的 I/O 特性优化存储子系统设计。据我们所知，唯一与本研究相近的作品是 [10]，它分析了 Nexus S 智能手机上 14 个安卓应用程序的 I/O 行为。其主要发现是，SQLite 和 Ext4 的组合操作会对基于 NAND 的存储产生不必要的过多写操作，这不仅会降低 I/O 性能，还会显著缩短底层 NAND 闪存的使用寿命。

978-1-5090-0088-3/15 $31.00 © 2015 IEEE
doi 10.1109/iiswc.2015.8

存储[10]。因此，这表明 SQLite 和 Ext4 需要以整合的方式进行优化，以消除多余的工作。在这项研究中，我们的重点不是了解软件层之间的交互[10]，而是利用块级 I/O 特性的影响来优化 eMMC 设计。

为了实现这一目标，我们首先实施了一个 I/O 监控软件工具，该工具使我们能够在 Nexus 5 智能手机上收集来自 18 个应用程序的 25 个块级 I/O 迹线（即 18 个单独迹线）及其组合（即 7 个组合迹线）。接下来，我们对这些痕迹进行定量分析，发现了一些有趣的智能手机应用程序的 I/O 特征。例如，我们发现在 18 个单独跟踪中的 15 个中，大多数请求（44.9%-57.4%）都是小型单页（4KB）请求。此外，在大多数跟踪中，80% 以上的请求可以在到达后立即得到服务。此外，根据我们发现的 I/O 跟踪特征，我们还得出了对 eMMC 设计的一些重要影响。例如，我们发现大多数请求都是小请求，这意味着这些小请求应迅速得到处理，从而提高 eMMC 设备的整体性能。最后，我们进行了一项案例研究，演示如何应用这些启示来优化 eMMC 设计。受这两种影响的启发，我们提出了一种混合页面大小（HPS）的 eMMC 方案。实验结果表明，与传统的纯 4KB 页面大小结构相比，HPS 方案可将平均响应时间缩短 86%。与现有的纯 8KB 页大小结构相比，HPS 可提高空间利用率达 24.2%。

本文接下来的内容安排如下。下一节，我们将介绍跟踪收集的细节。第三节分析 25 条跟踪记录在大小和时序方面的特征。第四节介绍了这些特征的影响。第五节介绍了一个案例研究，说明如何利用这些影响来优化 eMMC 设计。第六节简要总结了相关工作。最后，我们在第七部分对本研究进行总结。

II. 痕迹收集

在本节中，我们首先介绍收集跟踪信息的 环境设置。接下来，我们将介绍名为 BIOtracer（块级 I/O **跟踪器**）的 I/O 监视器的设计和实施，该监视器可收集 25 个块级 I/O 跟踪。最后，我们将解释如何以单独和组合的方式使用 18 种常见应用程序，从而记录它们的块级 I/O 活动。

A. 环境设置

我们使用 Nexus 5 作为移动平台来收集所有痕迹。它的存储子系统是 32 GB 的闪迪 INAND eMMC 4.51，没有外置 SD 卡。这款智能手机的安卓版本为 4.4（KitKat）。在 Nexus 5 上安装了我们定制的内核，并使用默认配置。我们创建了一个 I/O 记录缓冲区，用于记录每次跟踪收集过程中的 I/O 活动，其大小设置为 32 KB，可存储约 300 条请求记录。每个跟踪收集进程开始时都会生成一个日志文件，以便定期将 I/O 记录缓冲区中存储的 I/O 记录刷新到日志文件中。对于每个跟踪收集进程，系统都要重新启动

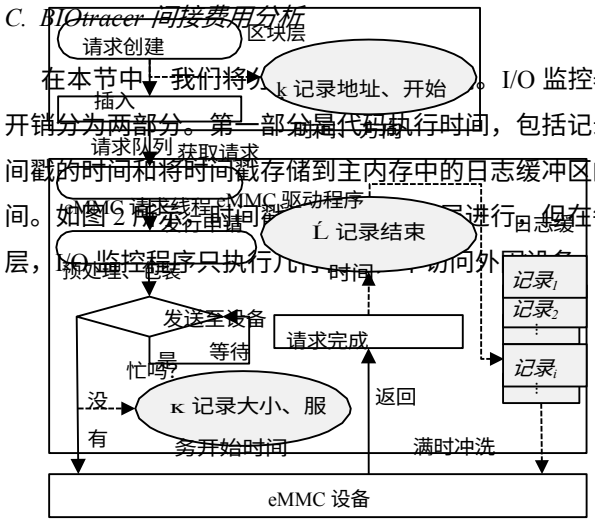


图 2：BIOtracer 的工作流程。

并清除日志文件，以尽量减少其他进程的干扰。

B. 生物追踪器

我们在 Linux 内核 3.4.0 中实施了一个名为 BIOtracer 的跟踪收集工具，以收集块层和 eMMC 驱动层的 I/O 信息（见图 1）。其架构如图 2 所示。对于每个请求，它都会在块层记录其开始时间、逻辑地址、请求大小和访问类型（读或写）（见图 2 中的步骤 1）。请求在块层创建后，会被插入块层的请求队列。然后，eMMC 请求线程获取请求，并在进行一系列状态检查后将请求发送给预处理功能。预处理功能会将块请求转化为 eMMC 请求。如果可能，打包功能会将多个写入请求合并为一个大请求。

最后，如果 eMMC 设备不忙，打包请求就会发送到该设备。否则，eMMC 请求线程必须等待设备处于空闲状态。当请求确实发送到设备时，BIOtracer 会记录请求的服务开始时间（见图 2 中的第 2 步）。当设备驱动程序完成请求时，BIOtracer 会记录其完成时间（见图 2 中的第 3 步）。请求的时间戳被合并为一条记录，存储在主机 RAM 中的 32 KB I/O 记录缓冲区中（见图 2）。缓冲区满后，所有记录都会被刷新到 eMMC 设备上的日志文件中。

C. BIOtracer 间接费用分析

在本节中，我们将分析 BIOtracer 的开销。I/O 监控器的开销分为两部分。第一部分是代码执行时间，包括记录时间戳的时间和将时间戳存储到主内存中的日志缓冲区的时间。如图 2 所示，时间戳[illegible]进行。但在每一层，I/O 监控程序只执行几行[illegible]

表 I：部分应用

应用	定义
闲置	智能手机处于闲置状态
调用	接听来电
呼出	打电话
启动	智能手机启动过程
电影	用智能手机看电影
音乐	用智能手机听歌
愤怒的小鸟	玩愤怒的小鸟游戏
摄像机视频	录制视频片段
谷歌地图	路线图和导航
信息传递	接收/发送/查看信息
推特	阅读和发布推文
电子邮件	接收/发送/查看电子邮件
在 Facebook 上	查看图片/添加评论等
亚马逊	移动在线购物
YouTube	在 YouTube 上观看视频
无线电	收听在线广播
安装	从 Google Play 安装应用程序
网络浏览	阅读《时代》周刊网站上的新闻

等待和线程切换。因此，这部分 CPU 时间可以忽略不计，因为与内核中成千上万行用于处理请求的代码相比，记录时间戳的代码量微不足道。

第二部分是日志缓冲区的刷新成本。为了获得稳定的 I/O 模式，I/O 监控程序会为每条跟踪记录收集较长时间的 I/O 数据，包括应用程序的启动、运行和关闭。因此，I/O 监控程序会定期将缓冲区刷新到 eMMC 设备上。在我们的跟踪收集实验中，缓冲区的大小设置为 32 KB，可容纳约 300 个请求的记录。每当缓冲区满时，缓冲区中的所有数据都会存储到 eMMC 设备中。我们发现，冲洗操作总是会产生 5-7 次额外的 I/O 操作（例如，同步打开、追加和关闭日志文件）。因此，I/O 监控程序造成的额外 I/O 操作数平均约为 6 次，仅为正常 I/O 请求数的 2%（即 6/300 = 2%）。

D. 应用选择

在大量智能手机应用程序中，我们选择了 14 个常用应用程序。此外，我们还收集了 4 个系统功能的 I/O 活动，包括 CallIn、CallOut、Idle 和 Booting。为简单起见，这四个系统功能也称为应用程序。表 I 总结了这 18 个应用程序，表 II 显示了如何使用这 18 个应用程序生成 25 条跟踪记录。Nexus 5 智能手机的后台服务包括电子邮件和信息接收服务以及一些基本的智能手机功能，如网络连接。除了 CallIn 和 CallOut 应用程序外，所有这些后台服务都会在程序运行时启用，以便收集跟踪信息。

III. 跟踪分析

在本节中，我们分析了 25 条跟踪记录，包括 18 条单独跟踪记录和 7 条组合跟踪记录。每个单独跟踪来自一个特定的应用程序，而组合跟踪（如 Music/WB）则由两个同时运行的应用程序生成，如音乐和 WebBrowsing。为了全面了解 I/O 模式，我们不仅研究了请求大小、请求类型和请求位置，还调查了请求响应情况。

表 II：痕迹收集详情

闲置 *（晚上 10 点至早上 6 点）*：闲置状态。
启动 *（30 秒）*：启动智能手机。
CallIn,CallOut *（1 小时）*：模拟电话采访，包括回答问题、交谈、倾听和闲逛。
CameraVideo、AngryBrid、GoogleMaps *（0.5 - 1 小时）*：录制视频、玩游戏、开车导航
Facebook、Twitter、亚马逊、电子邮件、信息 *（10 - 20 分钟）*：查看评论、搜索人或项目、查看图片和撰写回复。
网页浏览、YouTube、广播、音乐 *（1 - 1.5 小时）*：阅读新闻、观看在线视频、收听广播和听音乐。
电影，安装 *（10 分钟）*：观看本地存储的电影，安装游戏通过 WIFI 连接使用应用程序。
组合跟踪，FB/Msg 除外 *（10 分钟至半小时）*：使用 Facebook、在收听广播或音乐的同时发送信息或浏览在线新闻。
FB/Msg *（12 分钟）*：使用 Facebook，每当有新消息时就切换到阅读消息。按摩来了，回复后继续使用 Facebook。

时间、服务时间和到达间隔时间。此外，我们还分析了请求的并行程度。

A. eMMC 的吞吐量

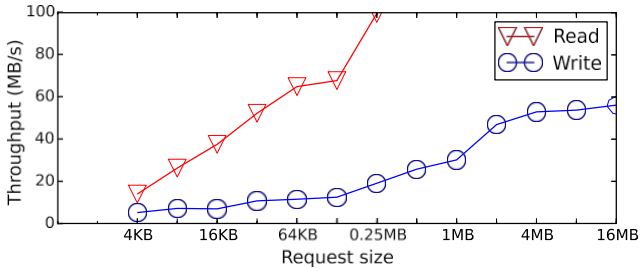


图 3：请求大小对吞吐量的影响。

图 3 显示，请求大小对 eMMC 的吞吐量性能影响很大。图中，特定请求大小的吞吐量是通过计算所有跟踪记录中该大小请求的平均访问率得出的。在收集到的 25 条跟踪记录中，最大的读取请求大小为 256 KB，而最大的写入请求大小为 16 MB。这就是读取吞吐量曲线在 100 MB/s 左右终止的原因（见图 3）。图 3 显示，当请求大小从 4 KB 增加到 16 MB 时，读吞吐量从 13.94 MB/s 变为 99.65 MB/s，而写吞吐量则从 5.18 MB/s 变为 56.15 MB/s。当请求大小为 256 KB 时，读取吞吐量达到最大值（即 99.65 MB/s），而写入吞吐量仅为 19 MB/s。这是因为在闪存中读取页面要比写入页面快得多。我们还发现，较大的请求（如 1 MB 以上）在读取和写入时都会产生较高的吞吐量，这归因于 eMMC 驱动层的打包命令。此外，图 3 还证实，eMMC 的性能明显低于固态硬盘，后者的连续读写速度分别为 550 MB/s 和 520 MB/s[11]。

B. 与尺寸有关的分析

表 III 显示了所收集的 25 条跟踪记录的请求大小相关特征。图 4 展示了 18 条跟踪记录的请求大小分布。在表 III 中，*数据大小*一栏和*请求数*一栏存储的是

表 III：25 个痕迹的大小相关统计数据

应用名称	数据尺寸 (KB)	数量的要求。	最大尺寸 (KB)	大道尺寸 (KB)	大道R尺寸 (KB)	大道W尺寸 (KB)	写要求百分比 (%)	写尺寸百分比 (%)
闲置	123,220	6,932	1,536	17.5	39.5	15.0	88.94	75.41
调用	27,300	1,491	1,536	18.0	12.0	18.0	99.93	99.96
呼出	27,364	1,569	1,536	17.0	10.0	17.5	98.92	99.37
启动	982,200	18,417	20,816	53.0	61.0	37.5	33.07	23.26
电影	130,420	4,781	512	27.0	27.5	17.0	5.40	3.37
音乐	240,060	6,913	940	34.5	62.5	9.5	52.80	14.48
AngryBrid	94,684	3,215	3,940	29.0	51.0	25.0	84.51	73.12
摄像机视频	2,283,184	9,348	10,104	244.0	38.5	736.5	29.46	88.85
谷歌地图	197,808	12,603	8,174	15.5	28.5	13.5	86.78	75.90
信息传递	63,668	5,702	128	11.0	23.0	10.5	97.30	94.38
推特	187,540	13,807	2,216	13.5	35.5	10.5	88.48	69.86
电子邮件	59,276	2,906	388	20.0	14.5	22.5	70.37	78.62
在 Facebook 上	97,436	3,897	2,680	25.0	28.5	23.5	74.42	70.70
亚马逊	67,412	3,272	1,392	20.5	24.5	18.0	63.02	55.07
YouTube	28,692	2,080	1,536	13.5	19.5	13.5	97.50	96.46
无线电	115,972	5,820	11,164	19.5	36.0	19.5	98.68	97.59
安装	1,653,900	17,952	22,144	92.0	22.0	93.0	98.26	99.58
网络浏览	95,908	4,090	1,536	23.0	21.5	23.5	80.71	81.95
音乐/WB	289,280	12,603	1,544	21.5	50.5	15.0	81.68	57.36
广播/转播	269,932	5,702	2,716	22.5	29.0	19.5	72.02	63.65
音乐/FB	442,388	13,807	2,424	12.5	38.0	8.5	87.67	62.34
广播/转播	153,776	2,906	1,368	14.5	23.0	13.5	91.68	86.92
音乐/信息	234,000	3,897	472	14.0	56.0	11.5	94.43	77.96
广播/信息	150,344	3,272	1,536	13.5	17.5	13.0	98.15	97.55
FB/Msg	182,632	2,080	732	11.5	21.5	9.5	84.72	71.72

分别是访问数据的总大小和请求总数。*最大大小*列记录了跟踪中发现的最大请求大小。*Write Reqs.*写入*大小*百分比则显示写入数据量占访问数据总大小的百分比。

由于使用了打包命令，大多数跟踪中的最大请求都大于 512 KB，这是 Linux 内核允许的最大请求大小。此外，*Ave. 大小*字段显示，数据密集型应用程序（如 CameraVideo 和 Installing）的平均请求大小远大于其他应用程序。18 个应用程序中的大多数的平均请求大小在 11 KB 到 34.5 KB 之间。

表 III 最后两列中的数值表明，除启动、电影和音乐外，大多数跟踪都是以写为主。在启动过程中，会发出许多读取请求以加载程序和配置文件，从而初始化智能手机。音乐和电影还需要从 eMMC 设备获取媒体数据，因此会产生大量读操作。CallIn 和 CallOut 只产生很少的读操作（分别不到 1%和 2%）。这是因为当用户呼叫某人时，除了一些日志文件更新外，所有其他活动都处于待处理状态。AngryBrid 会持续更新日志和播放状态，这就会产生大量的写操作。此外，CameraVideo 的数据写入率为 88.85%，而其写入请求百分比仅为 29.46%，这是因为它主要发出大型和顺序请求，这些请求可能会在驱动程序级别进一步打包。电子邮件等在线应用程序主要从互联网获取数据。为了更新日志并将数据缓存到 eMMC 设备，这些应用程序通常会发出大量的写操作。

特征 1：大多数智能手机应用程序都以写入为主。在 18 条跟踪记录中，有 15 条记录的写入请求百分比从 52.8% 到 99.9% 不等，其中 6 条记录的写入请求百分比超过 90%。

18 个应用程序的请求大小分布如图 4 所示。根据请求大小，请求被划分为不同的范围（例如，小于或等于 4 KB）。虽然范围代表一个连续的区域，但可能的大小只能是 4 的倍数，因为在文件系统级别，所有请求的大小都与闪存页面大小（即 4 KB）一致。图 4 显示，在 18 个单独跟踪中，除电影和启动外，16 个跟踪中的小请求占总请求的主要部分。不过，启动和电影等应用程序仍产生了大量的大请求。总的趋势是小请求占多数，大请求一般很少。与互联网相关的应用程序（即图 4 中的最后 10 个应用程序）显示出类似的分布，总体上也遵循相同的趋势。我们发现，数据密集型跟踪有其独特的请求大小分布，而其他跟踪则有类似的模式。例如，"电影 "有大量大小介于 16 KB 和 64 KB 之间的请求。

特征 2：在大多数应用程序中，小规模请求占请求总数的很大比例。在 18 个跟踪中的 15 个中，大多数请求（44.9%-57.4%）都是小请求（即 4 KB）。

C. 与时间有关的分析

在本节中，表 IV、图 5 和图 6 总结了 18 个应用程序与时间相关的 I/O 特性。在表 IV 中，*记录持续时间*字段中的值是从每个跟踪的开始时间到结束时间。启动的持续时间由系统性能决定，而其他应用程序的持续时间则由用户决定。虽然应用程序的 I/O 模式在很大程度上受用户习惯的影响，但我们确保每个应用程序的持续时间足够长，以便记录稳定的 I/O 模式。

到达率是指每秒到达的请求数，而数据访问率则定义为

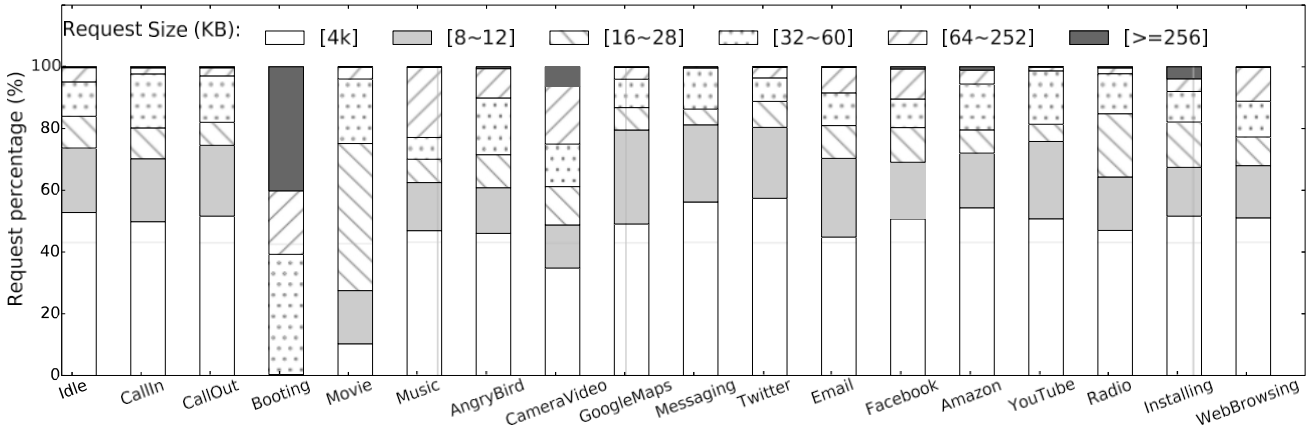


图 4：请求大小分布。

表四：25 个轨迹的时序相关统计数据

应用名称	录音持续时间（秒）	抵达速率（要求/秒）	访问速率（KB/秒）	不等待要求比率	平均值服务。（毫秒）	平均值响应（毫秒）	空间地点 (%)	时间性地点 (%)
闲置	29,363	0.24	4.20	89	7.42	9.24	25.32	34.22
调用	3,767	0.40	7.25	98	5.61	6.18	29.59	31.00
呼出	3,700	0.42	7.40	94	5.57	6.07	27.29	35.14
启动	40	460.40	24,555.00	58	1.65	4.93	28.19	19.70
电影	998	4.79	130.68	23	2.13	6.28	17.25	1.72
音乐	3,801	1.82	63.16	64	2.38	3.45	21.51	31.86
AngryBrid	2,023	1.59	46.80	84	3.44	4.06	30.08	26.07
摄像机视频	3,417	2.74	668.18	47	8.07	11.61	20.34	16.30
谷歌地图	1,720	7.33	117.76	85	1.40	2.23	21.10	42.78
信息传递	589	9.68	108.10	86	1.68	1.88	28.85	50.82
推特	856	16.13	219.09	84	1.72	2.07	26.57	52.90
电子邮件	740	3.93	80.10	63	3.01	4.09	14.49	34.87
在 Facebook 上	1,112	3.50	87.62	69	2.99	4.08	19.89	34.21
亚马逊	819	3.90	84.29	73	1.45	4.70	17.79	26.38
YouTube	4,690	0.44	6.12	96	6.90	7.19	47.61	16.35
无线电	4,454	1.31	26.04	82	3.54	6.62	23.90	29.18
安装	977	18.37	1,692.84	80	3.64	10.04	22.59	49.57
网络浏览	4,901	0.83	19.57	79	4.33	5.20	23.77	30.83
音乐/WB	2,165	6.10	133.62	65	1.70	3.61	18.40	38.40
广播/转播	1,227	9.78	219.99	69	1.86	3.30	18.66	28.48
音乐/FB	2,026	17.34	218.36	70	1.13	2.09	14.19	60.50
广播/转播	900	11.66	170.86	78	1.64	2.58	19.12	52.70
音乐/信息	926	17.82	252.70	74	1.36	2.19	20.68	53.84
广播/信息	660	16.82	227.79	89	1.63	2.04	27.25	49.48
FB/Msg	699	22.32	261.28	72	1.23	1.90	15.80	54.04

平均每秒访问的数据量（即读取和写入）。CallIn、CallOut、Idle 和 YouTube 显示的请求到达率（如每秒不到一个请求）和数据访问率（如每秒不到 10 KB）都非常低。启动仅持续 40 秒。但它仍会产生大量读取请求。它的请求访问率和数据访问率都是最高的。安装跟踪也显示了较高的请求到达率和数据访问率，因为安装过程会产生软件下载和安装。结合表 III 中的平均大小，我们可以发现数据访问率较高的应用程序（如启动、CameraVideo 和安装）的请求大小也较大。此外，我们还发现在 18 个跟踪中，有 15 个的到达率低于每秒 10 个请求。*NoWait Req.比率*表示到达时无需等待的请求的百分比。这意味着这些请求可以立即得到服务，因为没有正在服务的请求。表 IV 显示，至少有 63% 的请求

在 18 个单个跟踪中，有 15 个属于这一类，而在 18 个单个跟踪中，有 10 个有超过 80% 的请求可以立即送达。

特点 3：大多数请求在到达后都能立即得到处理。换句话说，同时到达的请求很少。

表 IV 中的*平均服务时间（Mean Serv.*(表 IV 中的平均服务时间（即平均服务时间）和*平均*响应时间（即平均响应时间）数据表明，Booting、Movie、Amazon 和 Installing 的请求队列较长，因为它们的平均响应时间与平均服务时间的比率较高。结合亚马逊不是数据密集型应用程序这一事实（见表 III），我们可以得出结论，亚马逊的 I/O 模式与其他应用程序不同。此外，我们注意到 Idle、CallIn、CallOut、YouTube 和 WebBrowsing 的请求到达率低于每秒 1 个请求。它们的平均响应时间和平均服务时间也比

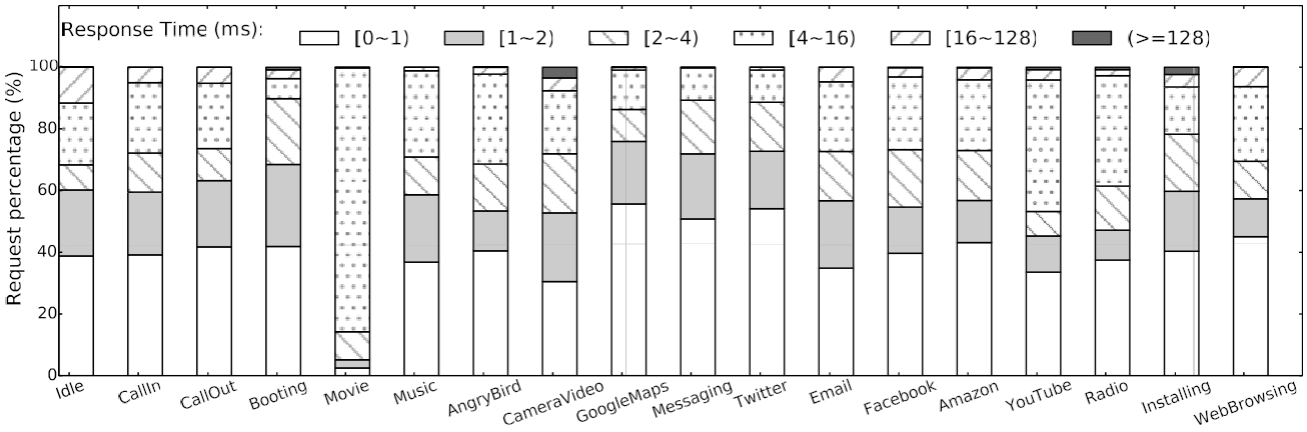


图 5：请求响应时间分布。

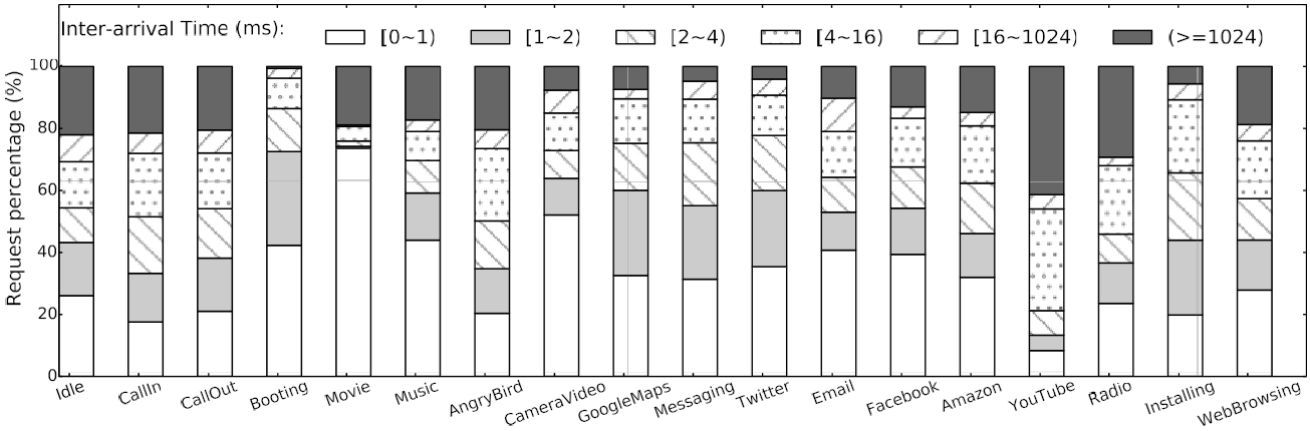


图 6：请求到达间隔时间分布。

其他（如音乐、电子邮件、Facebook）。这是因为当一段时间内没有请求到达时，eMMC 设备会进入低功耗模式。因此，eMMC 设备需要相对较长的预热时间来处理新到达的请求。在大多数其他轨迹中，平均响应时间的值约为平均服务时间的两倍。这一事实加上较高的 *NoWait Req.这一事实以及较高的NoWait Req.*

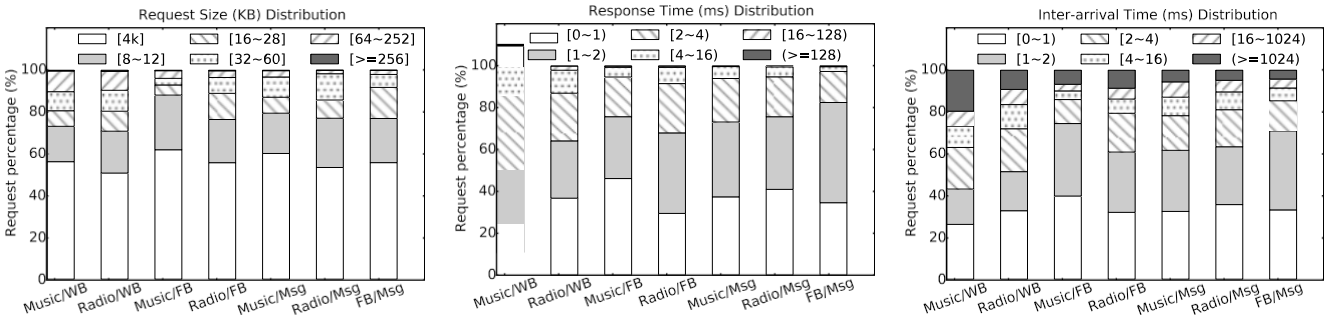
特性 4：如果请求到达间隔时间长于其省电阈值，eMMC 设备将进入低功耗模式。因此，在某些应用中可能会出现周期性模式切换。然而，频繁的模式切换会增加请求的平均响应时间。

最后两列记录了 18 个单独跟踪的位置。空间位置性的定义是连续请求访问占跟踪中请求总数的百分比。当当前请求的起始地址与前一个请求的终止地址相邻时，就会发生顺序请求访问。时间位置性是指地址命中数占总请 求 数的百分比。

当地址被重新访问时，地址命中数会增加一个。在空间局部性方面，18 条跟踪记录中有 16 条的空间局部性低于 30%，所有 18 条跟踪记录的空间局部性均低于 48%。与空间局部性相比，时间局部性总体上稍好一些（例如，18 条跟踪记录中有 11 条的时间局部性在 30.83% 到 52.9% 之间）。不过，与一些服务器级应用中观察到的 20/80 定位规则相比，所有这些定位都相对较低。

特点 5：18 个应用程序的局部性普遍较弱。此外，空间局部性低于时间局部性。

图 5 提供了一些无法从表 IV 中观察到的响应时间信息。图 5 中的总体趋势与表 IV 一致。
5 是大多数请求可在 2 *毫秒*内完成。此外，绝大多数请求可在 16 毫秒内处理完毕。在 18 个跟踪中，很少存在响应时间较长（即超过 128 *毫秒*）的请求。我们发现，响应时间分布与请求大小分布密切相关。这种高度相关性表明，请求的响应时间在很大程度上取决于请求的大小，而请求的大小又决定了响应时间的长短。



(a) (b) (c)

图 7：7 个组合轨迹的 I/O 模式。

这进一步意味着请求队列中等待的请求很少。例如，Movie 显示大小在 16 KB 到 64 KB 之间的请求比例很高（超过 65%）。它有近 70% 的响应时间在 4 *毫秒*到 8 *毫秒之间*。

图 6 提供了到达间隔时间分布的详细信息。CallIn 和 CallOut 的到达间隔时间较长，表明在通话过程中 I/O 活动较少。WebBrowsing、YouTube 和 Radio 也表明它们不会 对 eMMC 设备造成很大压力，因为它们的到达时间间隔较大。此外，在观看电影时，大部分到达时间都小于 1 毫秒（见图 6）。该图进一步证明了与互联网相关的应用程序具有相似的 I/O 模式，因为它们显示出相似的到达时间分布。与 YouTube 和广播等在线应用相比，本地应用（如启动、电影、音乐、CameraVideo）的到达时间更短。

特征 6：大多数应用程序的平均请求到达时间较长。18 个应用程序中有 13 个的平均请求到达时间至少为 200 *毫秒*。在 18 个跟踪中，有 10 个超过 20% 的请求到达时间超过 16 *毫秒*。

D. 并发应用程序的I/O 模式

在大多数情况下，智能手机用户通常在启用后台服务的情况下一次运行一个应用程序（如 YouTube）。不过，用户同时启动多个应用程序的情况并不少见。例如，用户可以一边听音乐（即音乐），一边在网站上阅读新闻（即 WebBrowsing）。另一个例子是，当用户正在使用 Facebook（即 Facebook）时，他可能会突然切换到阅读收到的信息（即信息）。回复信息后，他还可以继续使用 Facebook。因此，除了收集 18 个应用程序中的 18 个单独跟踪信息外，我们还收集了 7 个常见应用程序组合的 7 个组合跟踪信息，以了解它们的 I/O 模式。这 7 个应用程序组合包括 *{音乐、广播}* × {Facebook (FB)、Message (Msg)、WebBrowsing (WB)*}* 以及 Facebook 和 Mes- saging 之间的任务切换（即 FB/Msg）。之所以选择音乐和广播，是因为它们是其他应用（如网络浏览）的流行伴侣。图 7 显示了 7 种组合的特征

从请求大小和时间上看，包含音乐的组合跟踪与包含电台的组合跟踪相比，请求大小为 4 KB 的比例更高。图 7a 显示，与包含无线电的组合跟踪相比，包含音乐的组合跟踪中 4 KB 大小请求的比例更高。不过，7 个组合跟踪的总体请求大小分布与 15 个单个跟踪类似（见图 4）。另一方面，图 7b 显示，与单个应用情况相比，组合跟踪的响应时间并没有明显增加（见图 5）。例如，音乐/WB 的平均响应时间为 3.61 *毫秒，*而仅音乐的平均响应时间为 3.45 *毫秒*，网页浏览的平均响应时间为 5.2 *毫秒*。此外，7 个组合跟踪的到达时间间隔普遍较长，除音乐/广播外，其他所有组合跟踪的到达时间间隔超过 20%，均超 过 4 *毫秒*（见图 7c）。

7 个组合轨迹的平均到达间隔时间从 44.8 *毫秒*到 164 *毫秒不*等。在比较请求到达率和数据访问率下的数据后（见表 IV 第 3 列和第 4 列），我们可以发现，由于内存缓冲区等共享资源有限，一般情况下，组合跟踪在这两项上的数值要高于两个单独跟踪的总和。例如，Music/FB 的访问速率为 218.36 KB/s，而 Music（即 63.16 KB/s）和 Facebook（即 87.62 KB/s）的访问速率之和仅为 150.78 KB/s。不过，*NoWaitReq.Ratio* 的高值（见表四第 5 列）表明，并发应用仍不需要并行处理机制来处理同时到达的多个请求。总之，7 个应用程序组合的组合跟踪证实，即使多个应用程序同时运行，智能手机应用程序的 I/O 特性也基本稳定。

IV. 对 EMMC 设计的影响

根据这 6 个特征，我们可以为 eMMC 设计人员提供以下启示：

启示 1：在大多数跟踪中，80% 以上的请求在到达后可立即得到处理，这意味着很少有请求是同时到达的（特征 3）。这意味着，在设备层面增强并行性（如使用外部 SD 卡）或在操作系统层提供并行请求队列无助于提高性能。这是因为智能手机上的外置 SD 卡的性能明显低于

内部 eMMC 设备。例 如 ， 我们发现 Nexus 5 上 eMMC 的性能大约是 [7] 测试的主流制造商提供的 8 张 SD 卡最佳性能的三倍。对于大多数跟踪，使用外部 SD 卡可能会意外降低整体性能，因为当内部 eMMC 设备可以及时处理大多数请求时，较慢的外部 SD 卡会对整体性能产生负面影响。缩短大容量请求的处理时间可以显著提高整体性能。此外，特征 3 还表明，请求的响应时间主要由其服务时间决定。这表明大容量请求的响应时间较长，因为现有 eMMC 设备的通道数量通常非常有限。例如，闪迪的最新 eMMC 产品只提供两个通道[12]。因此，从一个大尺寸请求中分离出来的多个子请求（如超过 2 个子请求）无法以完全并行的方式进行处理。

启示 2：eMMC 中的 FTL 需要根据智能手机的 I/O 特性（特性 3 和 6）进行调整。我们发现，18 项跟踪中有 13 项的平均请求到达间隔时间超过 200 毫秒，而这一 时间 足以完成垃圾回收过程。因此，应重新设计 FTL 的垃圾回收机制，以便在执行这些非数据密集型应用时启动垃圾回收。这样，用户就不会感觉到垃圾收集导致的性能下降。在固态硬盘 FTL 中，当空闲块的数量达到预定的阈值时，通常会触发垃圾回收。eMMC 的 FTL 不应采用相同的策略，因为在空闲块数量明显变少之前，有很多机会进行垃圾回收。

启示 3：我们发现几乎所有迹线的时间局 部 性和空间局部性都很弱（来自特征 5）。例如，在大多数轨迹中，空间局部性低于 30%，时间局部性低于 40%（见表 IV）。因此，由于命中率较低，eMMC 设备内的大容量 RAM 缓冲区可能不利于性能优化。

启示 4：从特征 5 中我们还注意到，18 条跟踪记录中有 14 条的时间局部性低于 40%，16 条跟踪记录的空间局部性低于 30%（见表四）。低局部性表明，eMMC 设备级的 I/O 请求往往会访问闪存上的不同位置。因此，我们认为对于 eMMC 设备来说，简单的损耗分级策略就足够了。

启示 5：特征 2 显示，在 18 次跟踪中，有 15 次的大部分请求（44.9%-57.4%）是单页（4 KB）的小请求。这意味着，快速处理大量小型请求可提高 eMMC 设备的整体性能。更好地满足这些小请求的一个可行方法是使用 SLC（单层单元）闪存，这种闪存的读/写性能更好，但价格比 MLC 闪存高。幸运的是，MLC 闪存单元可以在 SLC 模式下有选择地使用其快速页，从而获得类似 SLC 的性能[13][14]。因此，性能的提高是以 50%的容量损失为代价的。

V. 案例研究

在本节中，我们将进行一个案例研究，演示如何利用影响提供的启示来优化 eMMC 设备的设计。启示 1 表明，加快大容量请求的服务时间可以显著提高 eMMC 设备的整体性能。显然，eMMC 设备的页面尺寸越大越好，因为它能有效处理大尺寸请求。事实上，现代固态硬盘倾向于使用更大的闪存页面大小[15]。另一方面，含义 5 表明，小尺寸请求也应得到快速处理，因为它们是大多数智能手机应用中的主要请求。然而，页面大小较大的 eMMC 设备不适合处理小尺寸请求，因为这可能会降低 eMMC 设备的性能和使用寿命。例如，当跟踪中的大多数请求都是随机写入 4 KB 数据时，大页面大小（如 8 KB）的 eMMC 性能可能会低于小页面大小（如 4 KB）的 eMMC，因为将 4 KB 数据写入 8 KB 页面比写入 4 KB 页面需要更长的时间。此外，当两个 eMMC 设备的总容量相同时（如都是 32 GB），8KB 页大小的 eMMC 的 页数比 4KB 页大小的 eMMC 少得多。因此，在有限的空闲页数被少量随机写入请求快速消耗后，eMMC 会有更多的垃圾回收（GC）操作。更多的 GC 操作会进一步降低性能，缩短设备的使用寿命。因此，需要小页数的 eMMC 设备来处理小型请求。因此，启示 1 和启示 5 促使我们提出一种混合页大小（HPS）的 eMMC，以有效地同时处理小尺寸和大尺寸请求。HPS 的基本思想是，eMMC 中的所有区块都具有相同的页面数（例如，在我们的实验中为 1,024），且区块中的所有页面都具有相同的大小（例如，4 KB）。然而，一个芯片中不同区块的页面大小可能不同（见图 1）。据我们所知，目前还不存在 HPS eMMC 设备。因此，模拟是验证其有效性的唯一方法。

表 V：三种 eMMC 设备的配置

	4PS	8PS	HPS
页面读取延迟（秒）	160	244	不适用
页面写入延迟（秒）	1,385	1,491	不适用
块擦除延迟（秒）	3,800	3,800	3,800
通道×芯片×芯片×平面	2×1×2×2	2×1×2×2	2×1×2×2
每个平面的块	1,024	512	512 个 4KB 页闪存 + 256 个 8KB 页盲区
每块页数	1,024	1,024	1024
总容量	32 GB	32 GB	32 GB

A. 模拟设置

由于成本限制，eMMC 设备的通道通常不超过两个。每个通道上连接一个或多个 MLC 闪存芯片。每个芯片由多个裸片组成，每个裸片有数千个区块。每个区块有数百个页面。由于 eMMC 设备可视为轻量级固态硬盘，我们使用名为 SSDsim [16] 的有效固态硬盘模拟器来模拟 eMMC 设备。SSDsim 是一个事件驱动的高精度模拟器。我们在 SSDsim 中实施 HPS 方案，使其能够在一个芯片中支持不同页面大小的区块。我们配置了几种混合页面大小的芯片

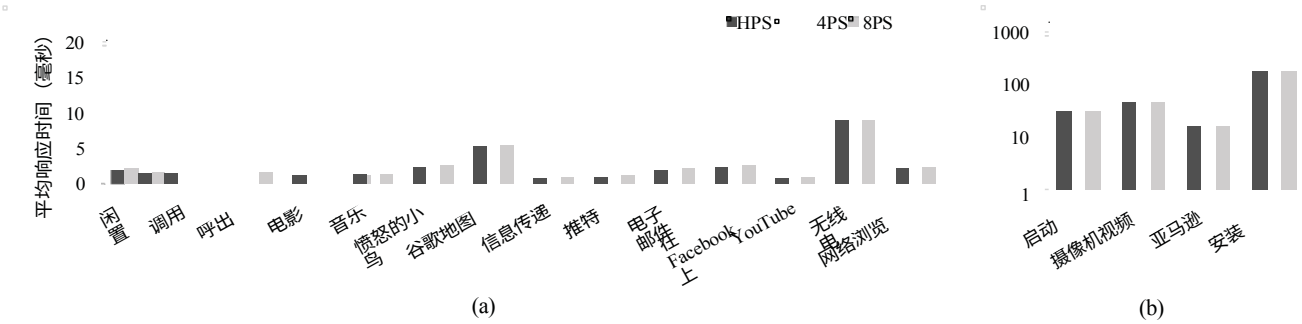


图 8：三种方案的性能比较。

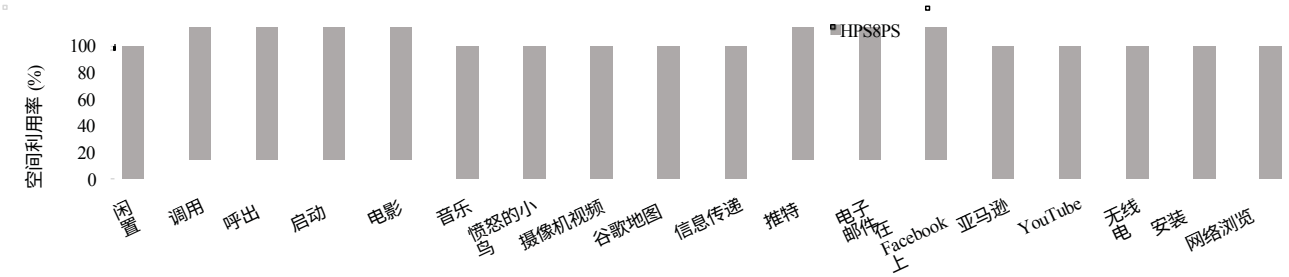


图 9：8PS 和 HPS 的空间利用率比较。

我们开发了一种新的请求分配器，用于将读/写请求分配到适当的页面。在 HPS 方案中，使用了两种大小的页面（即 4 KB 和 8 KB）。延迟相关参数来自 Micron 数据表 [17][18]。

对于 HPS 方案，我们假设平面内不同区块的页面大小各不相同。例如，在模拟器配置中，我们假设一个平面内有 512 个 4KB 块和 256 个 8KB 块（见图 10）。我们还配置了一个纯 4KB 页面方案（4PS）和一个纯 8 KB 页面方案（8PS）作为两种基准 eMMC 设备。HPS 和两种纯方案的详细配置如表 V 所示。请注意，三种方案的通道、芯片、裸片和平面数相同，因此内部并行性对三种方案的性能影响相同。根据配置，三种方案的总容量也相同（见表 V）。

请求分配器会将一个请求分割成多个页面。对于读取请求，数据会根据 eMMC 控制器维护的映射表进行检索。对于写入请求，将根据其大小以不同方式处理。例如，当写请求的大小为 20KB 时，它将被分为两个 8KB 的子请求和一个 4KB 的子请求。在单页大小的 eMMC 设备上，会浪费一些闪存空间。例如，如果使用 8KB 页大小的芯片，则需要 3 个子请求（共 24 KB）才能完成 20 KB 的写入请求。因此，会浪费 4 KB 闪存空间。写入请求的空间利用率定义为 20/24，相当于 83.3%。跟踪的空间利用率定义为写入的数据总量与消耗的闪存空间总量之比。显然，空间利用率越高，说明 eMMC 设备的使用寿命越长。

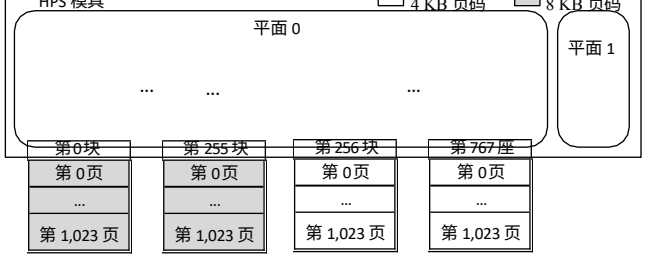


图 10：HPS 模具的结构。

B. 实验结果

我们以平均响应时间（MRT）和空间利用率来衡量每种方案的性能。所有跟踪都在模拟的全新 eMMC 设备上重放。模拟器的 RAM 缓冲层被禁用，以消除其对性能的影响。图 8 显示，在所有 18 个跟踪中，HPS 方案都优于 4PS 方案。由于 Booting、CameraVideo、Amazon 和 Installing 的 MRT 远高于其他项目，因此它们的结果以对数比例表示，并在图 8b 中分开。其余 14 项结果如 图 8a 所示。与 4PS 相比，HPS 在 "启动 "阶段的 MRT 方面取得了最显著的改进（86%）。即使在最差的情况下，即 Movie trace，HPS 仍能将 MRT 降低 24.0%。平均而言，HPS 可实现 61.9% 的性能改进。就 MRT 而言，8PS 方案的性能与 HPS 非常相似。

图 9 显示了两种方案的空间利用率。我们使用 4PS 作为基准方案。HPS 和 8PS 的结果均归一化为 4PS 的结果。由于 HPS 方案总是能达到与 4PS 相同的空间利用率，因此我们忽略了图 9 中 4PS 的结果。与 8PS 相比，HPS

在音乐轨迹中，HPS 方案的空间利用率提高了 24.2%。平均而言，与 8PS 相比，HPS 方案的空间利用率提高了 13.1%。

VI. 相关工作

随着智能手机越来越普及，了解其存储子系统与系统性能之间的关系最近开始引起业界和学术界的关注。截至 2010 年 9 月，基于 Android 的智能手机的日销售量为 20 万部，而 iPhone iOS 的日销售量为 8 万部，这是因为 Android 是一个开源操作系统[19]。因此，几乎所有关于智能手机存储子系统的现有研究都是在基于 Android 的平台上进行的。然而，据我们所知，只有少数研究调查了基于 Android 的存储子系统。Kim 等人发现，存储性能确实会影响一些常见应用的性能，如网络浏览、GoogleMaps、应用安装、电子邮件和 Facebook [7]。他们观察到，只需改变底层闪存，WiFi 性能在不同应用间的变化通常在 100% 到 300% 之间。Kim 和 Shin 在研究了 eMMC 的内部特性后，证实了 [7] 的结论。他们特别研究了 LSB 备份、打包命令和弹性组对基于 Android 的智能手机的影响。他们认为，eMMC 设备等存储子系统需要进一步优化[4]。Kim 和 Ramachandran 重新研究了智能手机的操作系统存储软件栈，以提高存储性能[8]。他们实施了一个名为 Fjord 的框架，该框架可提供细粒度控制机制，以权衡可靠性和性能。[7][8]和[4]均未重点分析 eMMC 级 I/O 跟踪及其影响。

Lee 和 Won 分析了六个不同类别共 14 个 Android 应用程序的 I/O 行为[10]。他们重点研究了软件层之间的交互：应用程序、Android 操作系统、文件系统和底层存储设备[10]。他们发现，SQLite 和 Ext4 的操作会产生不必要的过多写操作，从而大大增加了存储设备的负担[9]。虽然他们也分析了一些 I/O 特征，如 I/O 大小和随机性，以及它们对存储设备的影响，但他们主要关注的是了解应用程序和操作系统层产生的 I/O 活动，以及如何优化这些软件层。与此相反，我们的重点是如何将从 I/O 特征得出的影响应用到 eMMC 设计中。

VII. 结论

eMMC 设备的性能会明显影响智能手机应用程序的性能[7][8]。遗憾的是，在定量分析智能手机块级 I/O 特性及其对 eMMC 设计的影响方面，研究还很少。为了了解智能手机应用程序的 I/O 模式，我们在 Nexus 5 上实现了一个名为 BIOtracer 的 I/O 监控工具，并将其集成到 Android 内核 3.4 中。接下来，我们对 25 条跟踪记录进行了综合分析。我们观察到了六种 I/O 特征。下一页
根据这些特征，得出 eMMC 设计的 5 个影响。最后，我们进行了一项案例研究，演示如何应用这些影响来优化 eMMC 设计。

鸣谢

这项工作由美国国家科学基金会 CNS-1320738 号基金赞助。

参考资料

[1] W.Wang, D. Zhou, and T. Xie, "An embedded storage framework abstracting each raw flash device as an mtd," in *Proceedings of the 8th ACM Interational Systems and Storage Conference*.ACM，2015 年、
p.7.

[2] A.M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, and L. Vivier, "The new ext4 filesystem: current status and future plans," in *Proceed-ings of the Linux Symposium,* vol. 2. Citeseer, 2007, pp.

[3] T. H. open，" Android 2.3 姜饼将使用 ext4 文件系统" ，2014 年，访问时间：2015-4- 4
26. [Online].可查阅： http://www.h-online.com/open/news/item/ Android-2-3-Gingerbread-to-use-Ext4-file-system-1152775.html

[4] H.Kim and D. Shin, "Optimizing storage performance of android smartphone," in *Proceedings of the 7th Int'l Conf. on Ubiquitous Information Management and Communication*.ACM，2013 年，第 95 页。

[5] Kenlo，" 智能手机和平板电脑上的拉姆： 你需要知道的一切 " ， 2014 年，访问日期：2015-4-26。
[Online]. 见 ： http://community.giffgaff.com/t5/Blog/ RAM-On-Smartphones-amp-Tablets-Everything-You-Need-To-Know/ ba-p/7950298

[6] PHONEGG，"最快的处理器"，2014 年，访问日期：2015-4-26。[Online].
网址：http://us.phonegg.com/top/53-Fastest-Processor

[7] H.Kim, N. Agrawal, and C. Ungureanu, "Revisiting storage for smartphones," *ACM Transactions on Storage, vol.* 8, no.4, p. 14, 2012.

[8] H.Kim 和 U. Ramachandran，"Fjord：智能手机的知情存储管理" ，*《大容量存储系统和技术（MSST），2013 年电气和电子工程师学会第 29 届研讨会》，电气和电子工程师学会，2013 年，第 1-5 页*。IEEE, 2013, pp.

[9] S.Jeong, K. Lee, S. Lee, S. Son, and Y. Won, "I/o stack optimization for smartphones," in *Presented as part of the 2013 USENIX Annual Technical Conference*.USENIX, 2013, pp.

[10] K.Lee and Y. Won, "Smart layers and dumb result：*嵌入式软件第十届 ACM 国际会议论文集》*，ACM，2012 年，第 23-32 页。ACM, 2012, pp.

[11] 三星，" Samsung 850 pro ssd" ，2014 年，访问时间：2015-4-4 26. [Online].Available: http://www.samsung.com/global/business/ semiconductor/minisite/SSD/us/html/ssd850pro/overview.html?gclid= CjwKEAjwmfKpBRC8tb3Mh5rs23ASJACWy1QPGj8I7uQtfTq hMohfl7GF7ecW1b5usExP9lUZ1NtfxoCHk3w wcB

[12] SanDisk，"Sandisk inand extreme in 2014"，2014 年，访问日期：2015-4-26。[在线]。网址：http://www.anandtech.com/show/7790/ sandisk-inand-extreme-in-2014-finally-a-good-emmc-solution-for-mobile

[13] S.Im and D. Shin, "Comboftl: Improving performance and lifespan of mlc flash memory using slc flash buffer," *Journal of Systems Architecture, vol.* 56, no. 12, pp.

[14] W.Wang, T. Xie, and D. Zhou, "Understanding the impact of threshold voltage on mlc flash memory performance and reliability," in *Proceed- i n g s of the 28th ACM Int'l Conf. on Supercomputing*.ACM, 2014, pp.

[15] 美光，" Mt29f32g08cba 数据表" ，2013 年，访问日期：2015-4-26.[Online].Available: http://www.datasheetspdf.com/datasheet/ mt29f32g08cba.html

[16] Y.Hu, H. Jiang, D. Feng, L. Tian, H. Luo, and S. Zhang, "Performance impact and interplay of ssd parallelism through advanced commands, allocation strategy and data granularity," in *Proceedings of the Int'l Conf. on Supercomputing*.ACM, 2011, pp.

[17] 美光，"Mt29f128g08cbcbb 数据表"，2008 年，访问日期：2015-4-26。[在线]。 可查阅： _ _ _ _ _ _ http://www.micron.com/~/media/documents/ products/data-sheet/nand-flash/80-series/l85c plus 128gb 256gb 512gb 1tb 2tb async sync nand.pdf

[18] --，"Mt29f64g08cbaa 数据表"，2009 年，访问日期：2015-4-26。[在 线] 。 Available: http://www.datasheetarchive.com/dl/Datasheets-IS23/ DSA00448784.pdf

[19] M.Butler, "Android: changing the mobile landscape," *Pervasive Com-puting, IEEE, vol.* 10, no.