存储设备 IO 优化向导

文件标识：RK-KF-YF-146

发布版本：V1.1.1

日期：2021-05-12

文件密级：□绝密 □秘密 □内部资料 ■公开

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

前言

概述

产品版本

芯片名称	内核版本
全系列	通用

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师

软件开发工程师

修订记录

日期	版本	作者	修改说明
V1.0.0	陈谋春	2018-03-01	初始版本
V1.1.0	林涛	2021-02-04	格式升级以及添加io工具分析
V1.1.1	黄莹	2021-05-12	修改格式

目录

概述

虽然存储设备的性能在近年来也一直在稳步提升，从 RAW NAND 到 EMMC，再到现在 UFS 都在进步，但是和 CPU 相比依然是一个低俗外设，并且 IO 是不可抢占的，所以应用程序在交互中去等待 IO 会带来非常糟糕的用户体验，大部分程序员都意识到了这一点，所以绝大多数的程序尽量都会想办法改善这一点，例如预读和异步 IO 等。但是有一些场景是很难规避 IO 的影响，比如启动应用和本地音视频播放。如果在这两个场景中，还有其他程序来竞争 IO，那对用户体验来说可能是无法接受的。

现有的优化

我们已前面说的两个场景中的本地音视频播放[1]细化，设想有这样一个场景：某一天用户心血来潮要整理设备上的照片或其他数据，通过文件浏览器把大量的文件拷贝到一张新的 TF 卡上，这个过程是漫长的，这时候他打开一个本地视频观看，但是不幸的事情来了，视频卡顿非常严重。

碰到这种情况，工程师们肯定能理解，文件拷贝和视频播放在竞争 IO，导致解码器没有及时得到新数据，进而导致丢帧或音视频不同步。但是用户不一定能接受这个解释，他觉得为什么不能让文件拷贝的速度放慢一些，优先保证视频播放的用户体验。

这时候又轮到工程师来想办法了，在这个场景中，IO 实际上可以分两类，视频播放和文件拷贝，把其中视频播放的 IO 认为是关键 IO，问题就变成了如何避免关键 IO 拥塞。

让我们先看看万能的谷歌是怎么处理这个问题的，最新的 Android 会设置所有视频相关进程的 IO 优先级到 RT 级，这种优化的效果肯定是有的，但是还有几个问题：

- **效果有限**：RT 只能改善，而不能避免关键 IO 的拥塞，因为文件拷贝的进程会更频繁的发起 IO 请求，所以经常会碰到这种情况，视频播放的 IO 请求到来的时候，设备驱动正在执行文件拷贝的

IO，由于 IO 不可抢占，此时只能等这个请求完成，视频播放的请求才能得到服务

- **对第三方应用支持有限**：如果第三方应用是通过 MediaCodec 或 OMX 来实现音视频播放的，此时流媒体数据的请求是应用自己发起的，并不会被自动配置成 RT 级
- **没有考虑启动应用的场景**: 实测在文件拷贝的同时去启动应用，速度慢了近两倍

如何改进

结合 Cgroup 控制

Kernel 除了 IO 优先级以外，还有一个 BLK Cgroup 控制，目前支持两种控制策略，权重和节流。二者都是通过标准的 Cgroup 方式提供配置接口的，这里就不详细描述了。把进程分组以后，可以给不同的分组配置不同的权重，例如给视频播放进程配置 1000 的权重，给文件拷贝分配 10 的权重，则内核会尽量按 99：1 的比例来分配存储设备的带宽，当然实际上的带宽比例不会是这样的，因为两个进程发起 IO 的频度是不一样，内核只能保证短时间内的带宽按配置的权重来分配。而节流的话，则是可以给每个分组每个设备设置一个性能阈值，内核来保证性能不超过这个阈值，例如给文件拷贝分配 2MB/s 的读带宽，则内核会严格按这个指标分配带宽，即使此时设备带宽还没有用完。

提升第三方应用兼容性

大部分媒体应用都是直接调用 MediaPlayer 来实现播放，这样会通过 MediaServer 来读取流媒体数据，而这个进程 Android 已经配置了 RT 优先级。但是还有一部分应用是直接调用 MediaCodec 来实现播放的，比如特殊的加密格式视频，又或则为了实现帧合成等特殊处理，在这种情况下是由视频应用本身来读取流媒体数据，然后再送给 MediaCodec 的。为了改善这个问题，可以在 dequeueInputBuffer 函数中把当前线程的 IO 优先级和权重都配置到最高。

识别启动应用的场景

所有的应用启动都要经过 Activity Manager Service(后面简称 AMS)，所以在 AMS 的 startActivity 中识别启动应用开始，在 Activity.onStart 中认为启动应用结束。

限制文件拷贝性能

为了进一步提升前台应用和媒体应用的用户体验，还需要降低文件拷贝的 IO 性能，即在识别到启动应用和音视频播放的场景后，会同时根据一个白名单主要是各种文件浏览器）降低特定活动进程的 IO 优先级和权重，同时发出一个 BROADCAST_IO_JANK 消息，各种系统服务如 mtp 服务、下载器等可以在收到这个消息的时候降低 IO 优先级和权重

提供框架层接口

框架层提供 IO 优先级和权重的配置接口，但是普通应用只支持降低优先级和权重，只有 system 权限的应用可以提高优先级和权重。

整体流程

- Step1: 开机启动过程中，init 会为每个系统服务设定 IO 优先级和权重，其中 Media 相关的进程全部都是 RT 和最高权重，debuggerd、logcatd 等都是 IDLE 和最低权重
- Step2: AMS 会在 Activity 的不同生命周期，设置不同的 IO 优先级和权重，其中 TOP_APP > FORGROUND > BACKGROUD
- Step3: 媒体播放过程中会设置一个属性 media.codec.running=true，来表示当前正在进行音视频编解码，并且会发出 BROADCAST_IO_JANK 消息，通知系统服务调低 mtp、下载器和其他白名单中应用程序的 IO 优先级和权重到最低，并在播放结束后延迟发出 BROADCAST_IO_RESUME（如果在延迟时间内触发新的 BROADCAST_IO_JANK，会先移除这个延迟消息），通知系统服务恢复

到正常优先级和权重

- Step4: 在应用启动过程中，如果 media.codec.running==false，则会发出 BROADCAST_IO_JANK 消息，通知系统服务调低 mtp、下载器和其他白名单中应用程序的 IO 优先级和权重到最低，并在启动结束后延迟发出 BROADCAST_IO_RESUME（如果在延迟时间内触发新的 BROADCAST_IO_JANK，会先移除这个延迟消息），通知系统服务恢复到正常优先级和权重

  根据本文介绍的这些方法，整理了一个参考补丁:*blkio_patch.tar.bz2*，工程师可以根据需求定制和修改。

系统IO情况分析

1. Linux内核中通过/proc/diskstats来统计系统各磁盘的IO情况，通过连续cat此节点，可以看到每个磁盘在间隔时间内的数据访问量情况，可以用于粗略估计IO负载。

```
254       0 zram0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
179       0 mmcblk2 6804 2136 814432 7756 272 16 5072 239 0 3884 5534 0 0 0 0
179       1 mmcblk2p1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
179       2 mmcblk2p2 0 0 0 0 0 0 0 0 0 0 0 0 0 0
179       3 mmcblk2p3 0 0 0 0 0 0 0 0 0 0 0 0 0 0
179       4 mmcblk2p4 0 0 0 0 0 0 0 0 0 0 0 0 0 0
179       5 mmcblk2p5 0 0 0 0 0 0 0 0 0 0 0 0 0 0
179       6 mmcblk2p6 0 0 0 0 0 0 0 0 0 0 0 0 0 0
179       7 mmcblk2p7 75 0 4096 162 3 0 24 1 0 164 164 0 0 0 0
179       8 mmcblk2p8 0 0 0 0 0 0 0 0 0 0 0 0 0 0
179       9 mmcblk2p9 0 0 0 0 0 0 0 0 0 0 0 0 0 0
179      10 mmcblk2p10 45 27 1228 16 14 1 120 14 0 27 27 0 0 0 0
179      11 mmcblk2p11 30 25 458 9 8 1 72 12 0 17 20 0 0 0 0
179      12 mmcblk2p12 6223 308 776082 7184 0 0 0 0 0 3717 5064 0 0 0 0
179      13 mmcblk2p13 421 1776 32488 379 197 14 4856 181 0 427 480 0 0 0 0
179      64 mmcblk2boot1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
179      32 mmcblk2boot0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
253       0 dm-0 5743 0 695714 6509 0 0 0 0 0 3484 6530 0 0 0 0
253       1 dm-1 30 0 274 6 0 0 0 0 0 7 7 0 0 0 0
253       2 dm-2 721 0 75794 993 0 0 0 0 0 670 994 0 0 0 0
253       3 dm-3 129 0 3114 39 0 0 0 0 0 40 40 0 0 0 0
253       4 dm-4 18 0 1170 6 0 0 0 0 0 4 7 0 0 0 0
```

从左至右分别对应主设备号，次设备号和设备名称。后续的11个域的函义将在下面解释。除了第9个域，所有的域都是从启动时的累积值。用户主要关心的是第6个域和第10个域的情况，可以将用户分区的两次cat数值第6和第10个域相减，获取到单位时间内的读和写的sector数。

第4个域：读完成次数 ----- 读磁盘的次数，成功完成读的总次数。

（number of issued reads. This is the total number of reads completed successfully.）

第5个域：合并读完成次数， 第6个域：合并写完成次数。为了效率可能会合并相邻的读和写。从而两次4K的读在它最终被处理到磁盘上之前可能会变成一次8K的读，才被计数（和排队），因此只有一次I/O操作。这个域使你知道这样的操作有多频繁。

（number of reads merged）

第6个域：读扇区的次数，成功读过的扇区总次数。

（number of sectors read. This is the total number of sectors read successfully.）

第7个域：读花费的毫秒数，这是所有读操作所花费的毫秒数（用**make_request()到end_that_request_last()测量）。**
（number of milliseconds spent reading. This is the total number of milliseconds spent by all reads (as measured from make_request() to end_that_request_last()).）

第8个域：写完成次数 ----写完成的次数，成功写完成的总次数。

（number of writes completed. This is the total number of writes completed successfully.）

第9个域：合并写完成次数 -----合并写次数。

（number of writes merged Reads and writes which are adjacent to each other may be merged for efficiency. Thus two 4K reads may become one 8K read before it is ultimately handed to the disk, and so it will be counted (and queued) as only one I/O. This field lets you know how often this was done.）

第10个域：写扇区次数 ---- 写扇区的次数，成功写扇区总次数。

（number of sectors written. This is the total number of sectors written successfully.）

第11个域：写操作花费的毫秒数 — 写花费的毫秒数，这是所有写操作所花费的毫秒数（用__make_request()到end_that_request_last()测量）。

（number of milliseconds spent writing This is the total number of milliseconds spent by all writes (as measured from __make_request() to end_that_request_last()).）

第12个域：正在处理的输入/输出请求数 – -I/O的当前进度，只有这个域应该是0。当请求被交给适当的request_queue_t时增加和请求完成时减小。

（number of I/Os currently in progress. The only field that should go to zero. Incremented as requests are given to appropriate request_queue_t and decremented as they finish.）

第13个域：输入/输出操作花费的毫秒数 ----花在I/O操作上的毫秒数，这个域会增长只要field 9不为0。

（number of milliseconds spent doing I/Os. This field is increased so long as field 9 is nonzero.）

第14个域：输入/输出操作花费的加权毫秒数 ----- 加权， 花在I/O操作上的毫秒数，在每次I/O开始，I/O结束，I/O合并时这个域都会增加。这可以给I/O完成时间和存储那些可以累积的提供一个便利的测量标准。

2. 由于diskstats的粒度是针对分区的，且显示效果不够清晰。如果需要针对各个应用/进程进行单独统计，则可以利用ioblame脚本搭配ftrace功能进行IO信息采集。

- 首先需要将内核的CONFIG_BLK_DEV_IO_TRACE, CONFIG_FUNCTION_TRACER开启
- 将linux设备终端连接到PC机，确保ADB能够连接成功。
- 在PC上执行ioblame脚本：./ioblame.sh -r -w -v 其中参数r,w,v分别表示统计读、写以及是否显示进程PID
- 执行脚本后的任意时间内，使用ctrl+c进行中断脚本的执行
- 脚本收到中断执行信号后，将采集到的信息输出到PC的控制台

```
restarting adbd as root
Found rk3566_r Device
Unknown Device rk3566_r
OK to kill sleep when test is done
signal INT received, killing streaming trace capture
success test is done
Killing adb shell cat /sys/kernel/debug/tracing/trace_pipe

READS :
_______
```

```
FILE VIEW:
File: /
            app_process64 Reads: 4 KB
            netd Reads: 4 KB
            Total Reads: 8 KB i_size: 11 KB
File: /bin/hw/android.hardware.audio.service
            init Reads: 12 KB
            Total Reads: 12 KB i_size: 9 KB

//例如上述log显示了在ioblame脚本执行的时间内，android.hardware.audio.service文件一共向磁盘发起了12kb的数据读请求

PID: netd
            /system/bin/netd Reads: 804 KB i_size: 808 KB
            / Reads: 4 KB i_size: 11 KB
            /system/apex/com.android.resolv/lib64/libnetd_resolv.so Reads: 852 KB i_size: 862 KB
            /system/apex/com.android.tzdata/etc/tz/tzdata Reads: 116 KB i_size: 497 KB
            /system/bin/iptables Reads: 16 KB i_size: 475 KB
            /system/bin/netd Reads: 804 KB i_size: 808 KB
            /system/lib64/android.system.net.netd@1.0.so Reads: 84 KB i_size: 83 KB
            /system/lib64/android.system.net.netd@1.1.so Reads: 108 KB i_size: 104 KB
            /system/lib64/libcrypto.so Reads: 16 KB i_size: 1109 KB
            /system/lib64/libjsoncpp.so Reads: 68 KB i_size: 126 KB
            /system/lib64/libmdnssd.so Reads: 36 KB i_size: 35 KB
            /system/lib64/libnetutils.so Reads: 40 KB i_size: 36 KB
            /system/lib64/libpcap.so Reads: 328 KB i_size: 325 KB
            /system/lib64/libqtaguid.so Reads: 12 KB i_size: 11 KB
            /system/lib64/libssl.so Reads: 68 KB i_size: 339 KB
            /system/lib64/libsysutils.so Reads: 8 KB i_size: 48 KB
            /system/lib64/netd_aidl_interface-V4-cpp.so Reads: 324 KB i_size: 471 KB
            /system/lib64/netd_event_listener_interface-V1-cpp.so Reads: 40 KB i_size: 39 KB
            /system/lib64/oemnetd_aidl_interface-V1-cpp.so Reads: 40 KB i_size: 38 KB
            /system/usr/share/zoneinfo/tzdata Reads: 112 KB i_size: 497 KB
            Total Reads: 3880 KB
//log显示在ioblame脚本执行的时间内，netd这个进程一共向磁盘发起了3880KB的读请求。

Grand Total File DATA KB Reads 175200
Debug Grand Total KB READ 173980
//log显示了开机以来，系统总共向磁盘发起了175MB的读请求

Writes :
_______

FILE VIEW:
File: /tombstones/#479
            crash_dump64 Writes: 103 KB
            Total Writes: 103 KB i_size: 97 KB
//log显示在ioblame脚本执行的时间内，/tombstones/#479这个文件一共向磁盘发起了103KB的写请求。
```

```
PID VIEW:
PID: crash_dump64
            /tombstones/#472 Writes: 106 KB i_size: 97 KB
            /tombstones/#474 Writes: 106 KB i_size: 97 KB
            /tombstones/#477 Writes: 103 KB i_size: 97 KB
            /tombstones/#479 Writes: 103 KB i_size: 97 KB
            Total Writes: 418 KB
//例如上述log显示了ioblame脚本执行时间内，crash_dump64进程一共向磁盘写入了418kb的数据。

Grand Total File DATA KB Writes 1992
//显示了系统开机以来，系统总共向磁盘发起了1.9MB的写请求。
```

---

1. 在线音视频缓存流媒体数据一般都是直接存在内存中的，一是可以保证性能，二是Flash设备的寿命有限，所以不需要考虑IO优化↩