视频压缩第一次实验报告

提示：可以直接跳到最后看结论：

# 一、前置说明

## **1.1命名规则：**

iphone拍的视频由ip打头，小米6x的以m6x大头，荣耀8的以hor8\_打头,所有视频都按小视频标准拍摄，统一为15s。

## 1.2测试环境：

Linux机器：单核单线程 Intel(R) Xeon(R) CPU E5-26xx v4

Windows机器：2核4线程 core i5-4300u 2.0G hz

## 1.3有关视频的几个概念：

1. 视频文件格式：由于项目最终在手机网页端演示，而现在手机摄像产生的视频只有两种统一的格式——MOV (ios系统)和MP4（Android系统），所以测试只涉及这两种格式。
2. 视频封装格式：仅介绍两种相关格式。

**Mpeg**格式：有 .mpg、.mpeg、.mpe、.dat、.vob、.asf、.3gp、.mp4 等等

**QuickTime File Format** 格式，对应的文件格式是 .mov，是 Apple 公司开发的一种视频格式，默认的播放器是苹果的 QuickTime。

1. 视频编解码方式：视频编解码的过程是指对数字视频进行压缩或解压缩的一个过程。常见的视频编码方式有：H.261、H.262、H.263、H.264、H.265。其中，H.264是现在的主流视频编解码方式，H.265正逐步成为下一代的标准。可以把「视频封装格式」看做是一个装着视频、音频、「视频编解码方式」等信息的容器。一种「视频封装格式」可以支持多种「视频编解码方式」。一个封装方式是 QuickTime File Format，编码方式是 H.264的视频记为H.264/MOV。
2. 帧速率：帧速率，我们看到的视频是由一帧帧的画面，也就是一张张的图片构成；如果一秒钟跑过去的图片有24张，帧速率就是24帧/秒（fps）（低于这个帧率，人眼会感觉卡顿）；
3. 分辨率：我们所看到的图像都是由一个个的像素点构成的；分辨率就是所显示图像的精密程度，也可以理解为像素点的多少。在屏幕大小一定的情况下，分辨率越高，图像就会越清晰；在分辨率一定的情况下，屏幕越小，图像越清晰；如1080x720，可以理解为每行1080个像素点，一共720行。
4. 码率（在视频传输时，和比特率等同）是指每秒钟视频或音频的数据量，一般用兆／秒或K／秒表示。其他条件相同的情况下，码率越大，视频越清晰，但文件也越大；

# 二、测试

单个ffmpeg实例：每个原始视频都为1920\*1080的分辨率

## 2.1 ip1.MOV

### 2.1.1 linux环境

命令：ffmpeg -i /data/minitrill/video/temp/ip1.MOV -b:v 2000k -bufsize 2000k -vf scale=960:540 -strict -2 /data/minitrill/video/hasDealVideo/ip1.MOV

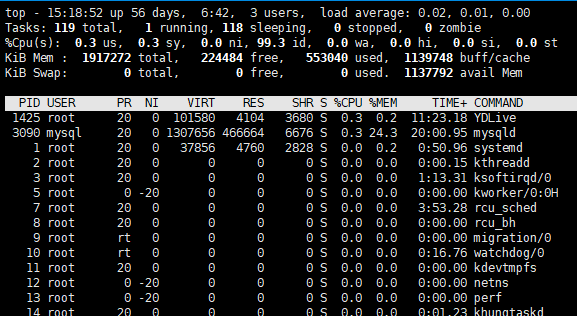
解释命令：-i 设定输入流 -b:v控制平均码率

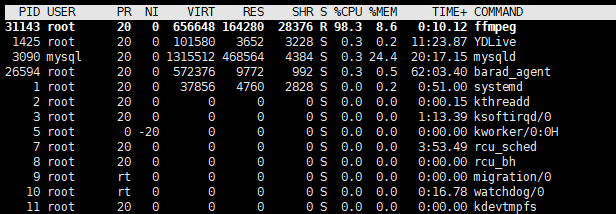
-bufsize 用于设置码率控制缓冲器的大小，设置的好处是，让整体的码率更趋近于希望的值，减少波动。

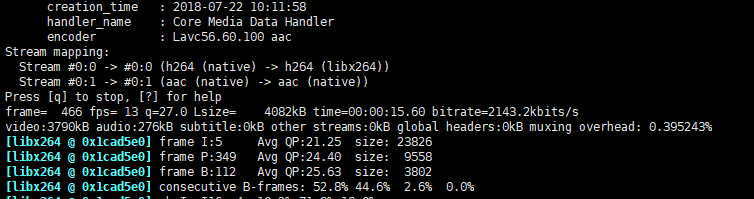
-vf 控制分别率

-strict -2：禁止某个库的错误提示

最后一个参数表示输出结果







用时：35s

处理前：28.8M

处理后：4.0M

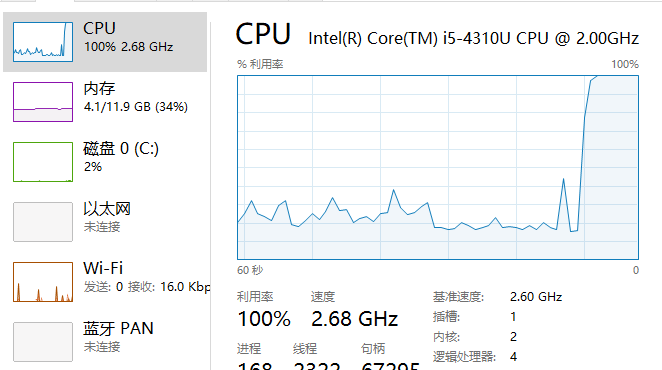
### 2.1.2 Windows：

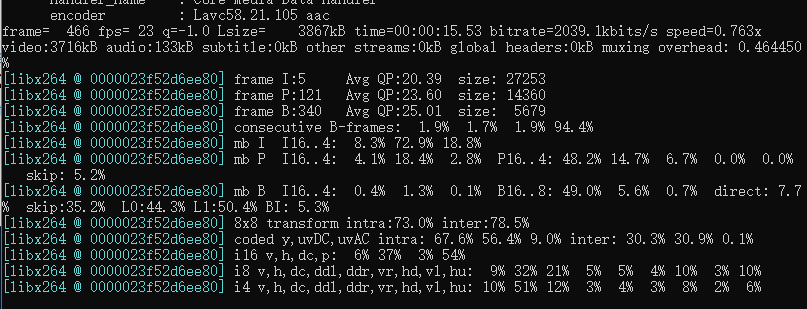
ffmpeg -i C:\Users\calebzhang\Desktop\ip1.MOV -b:v 2000k -bufsize 2000k -vf scale=960:540 -strict -2 C:\yasuoshipin\dealip1.MOV

用时：23s

处理前28.8M

处理后：3.77M

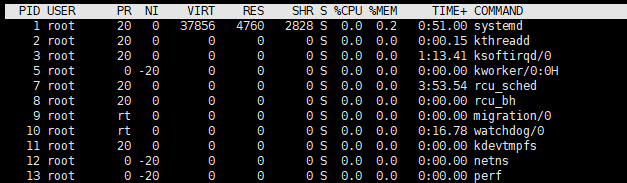


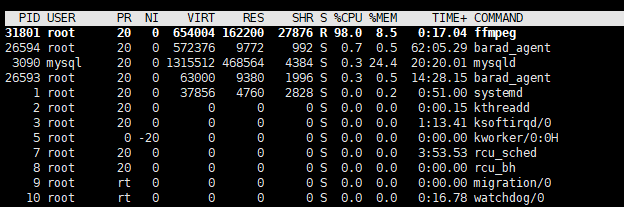


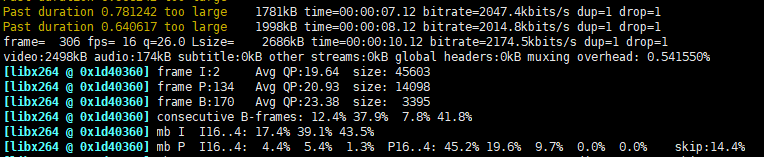
## 2.2 m6x1.mp4

### 2.2.1 Linux

sudo ffmpeg -i m6x1.mp4 -b:v 2000k -bufsize 2000k -vf scale=960:540 -strict -2 ../after/dealm6x1.mp4







提示：压缩之后1M大小左右视频在网页端可以满足一般用户体验。

处理前：15.8M 1920\*1080

处理后：

2000K 37.5s 2.7M 很清晰（数据含义：码率(bps)，处理时间(s)，处理后大小）

900k 29.6s 1.3M 较清晰

400K 23.5s 682K 很模糊

### 2.2.2 Windows：

2000K: 22.5s 2.46M

900K: 17s 1.15M

400K：14s 568K

ffmpeg -i C:\Users\calebzhang\Desktop\m6x1.mp4 -b:v 400k -s 960x720 C:\yasuoshipin\dealm6x1\_6.mp4 9s 658k 清晰度比单纯400k好点(默认比例为1920x1080 )

ffmpeg -i C:\Users\calebzhang\Desktop\m6x1.mp4 -b:v 400k C:\yasuoshipin\dealm6x1\_5.mp4 15s 687k

ffmpeg -i C:\Users\calebzhang\Desktop\m6x1.mp4 -b:v 400k -strict -2 C:\yasuoshipin\dealm6x1\_5.mp4 15.2s 687k

hor8\_1.mp4

ffmpeg -i C:\Users\calebzhang\Desktop\hor8\_1.mp4 -b:v 400k C:\yasuoshipin\dealhor\_8\_1\_1.mp4 29s 31.8M 1019k 很模糊

ffmpeg -i C:\Users\calebzhang\Desktop\hor8\_1.mp4 -b:v 400k -s 960x720 C:\yasuoshipin\dealhor\_8\_1\_1.mp4 15s, 31.8M，1003k，比模糊好一些

ffmpeg -i C:\Users\calebzhang\Desktop\hor8\_1.mp4 -b:v 900k C:\yasuoshipin\dealhor\_8\_1\_1.mp4 34s 31.8M 1944k

# 三、多进程测试

由于linux环境为单核单线程，测试多进程和多线程没有意义，所以多进程多线程的测试都在2核4线程的windows环境下测试：

多进程测试：

压缩命令：ffmpeg.exe -i C:\Users\calebzhang\Desktop\\* -b:v 900k C:\yasuoshipin\\*

1. 分别按顺序串行压缩m6x1.mp4(15.8M)，hor8\_1.mp4(31.8M), ip1.MOV(28.8M)，压缩效果分别为（1382K，19s）（1944K，34s）（1943K, 35.6s）

总用时为 19+34+35.6 = 88.6 s

1. 同时开三个进程，每个进程压缩一个视频，压缩总用时为：90.2 s

# 四、多线程测试

由于ffmpeg本身支持多线程，所以只需设置压缩命令即可：

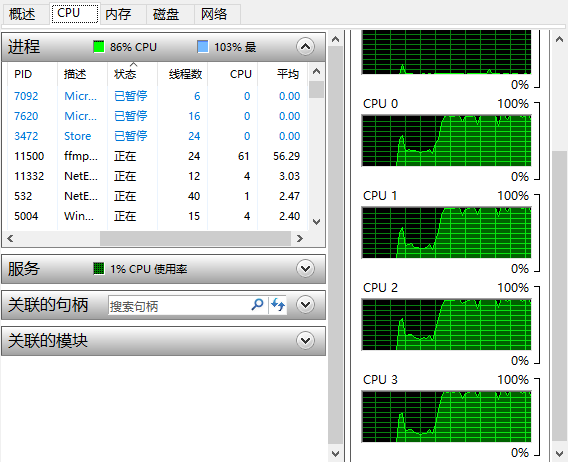
ffmpeg.exe -threads 2 -i C:\Users\calebzhang\Desktop\\* -b:v 900k C:\yasuoshipin\\* (表示开启两个线程)

ffmpeg.exe -i C:\Users\calebzhang\Desktop\\* -b:v 900k C:\yasuoshipin\\*（默认状态）

ffmpeg.exe -threads 0 -i C:\Users\calebzhang\Desktop\\* -b:v 900k C:\yasuoshipin\\* (和默认状态一致)

默认状态：在没有其他计算任务的情况下，cpu有几个逻辑处理器，打开几个线程

分别测试了在0、1、2、3、4个线程下的压缩效率以及cpu的使用率，和默认状态下几乎都是一样的，打开任务管理器，发现无论开多少个线程，4个逻辑处理器都会启动，cpu利用率突破95%，当我不打开其他应用程序时，单单ffmpeg进程平均cpu占有率超过75%，当打开chrome，word，音乐播放器时，ffmpeg进程平均cpu占有率超过60%。



# 五、结论

视频压缩涉及大量数学运算，是一个非常消耗cpu资源的任务，鉴于不同手机摄像头像素不同，导致一个15s的短视频占用磁盘空间不同（大概在20M到30M之间）。根据实验，压缩至1M大小，原视频大小平均每增加1M，压缩时间就增加1.13s，30M的短视频压缩时间超过30s，所以未改进的算法影响用户体验。

由于我们前端开发不像大不数短视频应用采用Android开发，难以利用手机CPU和GPU进行视频压缩处理，集中交给后端非常容易造成性能瓶颈。

# 六、后续

算法改进以及具体方案需进一步研究