Android系统优化的那10年

Danny
2022-09-21 · 苏州叶楚企业管理有限公司猎头顾问

题记

8月，看到一个新闻：
携手Unity引擎战略合作：一加Ace Pro《原神》1小时59.3帧无压力
感觉一加在上下游联合优化方面，比其它友商晚了几年。借此聊聊Android系统优化2008~2018这10年的事儿。

文章有点长，13000字，PC阅读体验要更好。感兴趣的建议点赞收藏。😊

1. 用户对手机的需求层次

- **功能齐全**: 行业起步期
- **稳定不耗电**: 行业发展期
- **性能流畅**: 行业成熟期
- **安全**：隐私问题会越来越重要

智能手机行业以苹果发布第一代Iphone开端，乘着移动互联网的东风，飞速发展。Google与此同时收购Android项目，自己主导开发。到Android 2.3时，国内雷军带领小米从做MIUI ROM转做手机硬件，发布第一款互联网营销手机小米1代，小米1借助深耕很久的MIUI，成为当时国内功能最多样的智能手机。但作为一个互联网公司转型做硬件，没有华为、oppo、vivo这类硬件公司背景的技术积累。对软件的稳定性的认识不够，小米1 各种死机重启。

司，对mac os扩展为iOS系统后的各种tuning调优和深度整合，使得Android厂商过了好几年才慢慢和iOS系统可以在同一水平。Android手机厂商对系统稳定性也足够重视。

2. 稳定性

稳定性包括，os系统自身稳定性和三方app兼容性。google 每年发布更新Android大版本，自身的系统bug缺陷也是很多的。手机厂商自身OS功能开发也会引入各类bug。同时三方应用App由于对新版本的适配不及时，导致和新系统的兼容性问题也相当严重。尤其是各种保活黑科技的引入和动态升级更新热修复。有些多进程保活方案，app自身代码缺陷，导致一直fork创建新进程，甚至出现把linux kernel能使用的最大进程数量32768都用完，导致系统不能创建进程出现死机的情况，比如猎豹浏览器。热修复方案以阿里的虚拟机dalvk和art虚拟机层hook方案和腾讯tinker方案为主流。阿里的风格比较激进，当然稳定性就差了些。腾讯的产品更看重稳定性，但少了些灵活。

手机厂商也设计出各种monkey测试、老化、压测等各种自动化，各行各类应用 top 1000、top 2000、top 5000等兼容性自动化测试。华为等公司也开发了自己的云上系统，专门针对应用在华为手机上的稳定性进行强化压力测试。
每次Android大版本升级，靠谱的手机厂商能发现各类问题缺陷上万个。甚至出现专门bug-fix的工程师。

这么多问题，怎么处理呢？靠人分析人力投入远远不够，就出现了各种监控，问题归类，自动化提单，分析等技术。以期加快问题修复速度。

发展了这么多年，手机稳定性问题，慢慢成熟，一个Android手机，好几个月也遇不到异常死机或者重启问题。

3. 续航

但Android手机的电池续航一直很差。各个手机厂商也在续航寻求突破，比如华为资助研究石墨烯新材料。但是这个革命性技术是否成功，可能需要好多年的科研。毕竟还是在实验室的玩具。一个技术性行不行，可以在实验室进行科学论证研究。但是要变成商业产品，需要工程化，规模化，量产化。柔宇的柔性屏技术就遇到过量产良品率低导致成本高的问题。

延长手机续航时间有2个方向，**让用户觉得**：更慢的耗电和更快的充电。

3.1 更快的充电

2013、14年左右，一家做充电器的小厂商实验完成了个大电流快速充电的设计，兴致勃勃的找到各大手机厂商，由于产品稳定性等安全问题。碰了一鼻子灰。oppo的一个工程师知道这个方案后，很是喜欢，自个儿私下研究，通过设计各种危险情况判断，多重模式切换阻断，防止大电流，发热，爆炸等，居然成功改进了各类问题，让它更稳定，更安全。成就了2015年的“充电五分钟，通话两小时”。这项快充技术，让oppo的产品在那年的差异化竞争尝到甜头，快充研发团队奖励了100万。继续研究，设计出了更快的闪充。中兴移动(后面改名叫努比亚)相关领导想起oppo设计出的闪充和那家找过自己的小厂商，都后悔自己没有深挖下去，错过了这个机会。

电池容量E=电压U x 电流I x 时间t。

电池容量比较固定，手机一般都3000+mAh，金立直接使用2块电池解决大容量问题，算个例外；iPhone电池出现过2000+mAh的，但是人家软硬一体化做得好，续航时长比Android系统要好，当然现在大屏，iPHone 11 好像是3500mAh左右。

oppo采用大电流方案并申请了专利，那高通、mtk、华为就走大电压方案了。整个以前是5V x 4A的20W，发展到现在都40+W了。知乎上海出现过 到底是大电流安全还是大电压安全 的问题。用了这么多年，国产手机好像没出现过一次电池爆炸事件吧。当然，三星2016年出的电池爆炸事件可能是采用了比较激进的方案，对稳定性安全性重视不够导致的手机行业大事件吧。

3.2 更慢的耗电

硬件上面的改进，不能替代软件层面的优化。金山银山，不好好持家，也会早晚败光。

手机功耗包括3方面：

- 基础功耗
- 待机功耗
- 场景功耗

3.3.1 基础功耗

手机硬件工程师会综合考虑各个器件的布板（当然，高通、mtk一般会给个公板做参考）。出了整机后，软件工程师还要对基础功耗进行摸底排查，避免出现设计缺陷，出现漏电等情况。飞行模式下、亮度最低、音量mute等各种条件下，使用power monitor看关注的sensor、器件的电流到底有多大。
比如待机电流2mA左右；
比如wifi待机功耗比不开wifi大0.25mA；wifi搜网比不开wifi大2.5~4mA；
比如音频最低音外放电流低于50mA；
...
如果发现异常，就要具体分析是什么原因。是因为漏电？还是 后台在跑xx进程？

因为电量计算其实是个模拟电路的估算值，不能完全数字化那么准确。Android上采用计算各个sensor在不同模式下的电流情况，记录到power_profiler.xml文件中，然后根据各个sensor运行的时长，继续计算求和，得到估算的整体电量消耗存放在batterystats文件里面。这个方案存在不准确性，比如频繁唤醒就可能出现实际电量消耗远远大于计算功耗。

开发电池计量器算法的工程师也会根据各个算法来优化实际电量消耗和状态栏上的电量百分比的对应关系，因为电量和电量百分比这个不是一个线性关系。有些厂商会做个取巧的方案：90%以上的时候电量百分比变化慢，但实际电量消耗已经大于10%啦，让算法在后续使用电量百分比的时候再慢慢平缓对应上。这也让用户避免出现刚充满电，还没怎么用就电量跌了这么多的心理不适。

电量的消耗是由于各个sensor硬件的运行，各个软件进程、操作系统的cpu、gpu计算导致的。

3.3 待机功耗

3.3.1 控制应用启动

由于Android的开放性和google服务在国内不能使用的原因，各个app为了保持服务器能及时发现app、通知app、控制app等，出现了各类进程保活方案。最让人讨厌的就是各类全家桶app相互唤醒，导致Android手机生态中的功耗问题相当严重。手机厂商就提出了各类自启动控制、相互启动控制等方案。

厂商直接在系统层面，对应用的广播唤醒，provider唤醒、job scheduler唤醒，父子service唤醒，多进程监控唤醒、模拟binder调用唤醒等各类黑科技进行限制。

自身对功耗控制做的很好的微信等部分国民应用，厂商也愿意给它开白名单。所以就会出现，其它app公司的研发人员被产品业务"鄙视"为什么我们做不到自启动，为啥微信可以做到?

国民级app和手机厂商的这种 相互绑架制约和相互平衡维持整个生态稳定也付出了重要贡献。部分手机厂商甚至单独出api接口给国民app，可以根据app自身运行场景进行动态调频等功能。

Android圈还出现了 绿色公约，希望各个app加入维护好整个生态。

苹果的iOS整体是封闭的，全部走苹果自身的push推送，所以app保活这块就比较风平浪静。

3.3.2 控制后台进程运行

控制功耗，自启动控制能让app不随意运行，减少系统功耗；对于运行中的app，还可以通过限制后台运行达到降低功耗的目的。

iOS系统，app退后台后可以运行3分钟左右（版本不同，策略不同），如果超时，就会被杀。如果遵守规则，就会类似signal - stop ，冻结进程。停止进程运行，达到减少系统耗电的目的。

Android上也要类似方案。2014、15年前，系统厂商主要采用定时清理进程、杀死后台运行的进程，达到控制手机功耗的目的(还有个原因是当时手机内存不大，运行的应用多可能让内存吃紧导致系统卡顿)。之后，出现了类似后台冻结进程的方案，避免了粗暴的杀进程设计(但是针对异常耗电app，还是会采用kill方式，但可能会通知栏提示用户xx-app耗电异常，可以考虑结束应用)。可以类似iOS的给进程发送stop信号，让进程挂起；也可以采用linux本身支持的cgroups方案，直接把希望冻结的应用的所有进程加入cgroups的freezer组下面管控。

手机系统耗电，大头在cpu上。对进程的调度，可以分当前应用、前台应用、后台应用等；android目前也采用分组进行控制。

cpu调度氛围 调度器、调度策略、调度优先级等；由于Arm芯片有分大小核cpu，多核cpu等，进而可以让后台进程运行更低优先级；运行在小核，降低cpu频率等降低功耗。对应当前应用，采用相反策略：
提高cpu频率、按cpuset进行分组运行在打核、taskset设置亲和性等 让 用户使用的app有尽可能多的cpu资源。

google EAS(Energy-Aware Scheduling)调度器方案，也是基于处理相同任务可以消耗更少的功耗，当然相同功耗，性能会更好。

EAS方案其实在几年前有Linaro和Arm公司合作开发。Arm公司自己不生产cpu芯片，但它设计芯片，卖协议卖授权。iPhone和Android手机都是基于ARM cpu开发。而Linaro是一个在ARM平台上连接公司企业和开源社区进行开发设计的组织。各个企业可以缴纳不同的会费成为金牌银牌赞助商。当然linaro会按赞助商的不同等级进行不同延期时间的同步内部的新技术新方案给他们。

Linux主流的CFS：'Completely Fair Scheduler' cpu公平调度调度策略是基于load负载和优先级-权重的的策略进行task任务的排序调度。优先级和权重其实就是些经验值。进程优先级有0-139，前0-99位实时进程优先级；100-139位普通进程优先级，对应nice参数为-20-19。而权重对应普通优先级的nice值，nice越小优先级越高，对应权重也就越大。实时进程对应权重为nice=-20的2倍权重。调度器先调度实时进程队列，再调度普通优先级队列。

而EAS，不仅包括了CFS，还包括cpuidle、cpufreq。把各个零散的方案整合起来，综合考虑。基于实际使用效能的调度策略。避免了原来依靠经验值各种试错tuning的方式，而是采用可测量的能效模型进行调优方法。

当然各个器件sensor也有自身的低功耗模式等，idle一小段时间可以自动suspend或者设置low power mode。

3.3.3 待机功耗优化

手机待机功耗异常，主要包括：

- 不能待机：包括应用端持锁，kernel端持锁；
- 频繁唤醒：各类网络请求，alarm等中断唤醒ap；
- 子系统不待机：modem、gps、wifi、nfc等有在工作或发生suspend异常等；

google 设计的doze mode\device idle针对系统灭屏后一段时间后，针对非白名单应用，会限制网络，取消wakelock避免系统不休眠，延迟alarm等各种定时任务同步、关闭wifi扫描等。App standby方案也是让不常用的app被限制网络等。但这类方案在中国大陆效果较差，app各类方案提高活跃度，比如某个音频app为了做日活在埋点数据上还能作假翻几倍，各个厂商还需要做中国特色的tuning增强完善。

其实除了google的这些个方案，厂商也早就研究基于不同用户的app使用习惯，针对性的优化。数据显示，一个用户，常用app 一般不超过10个。针对这个特点，进行针对性优化。比如灭屏释放waklock锁、禁止gps location调用、关闭wifi 扫描或见啥扫描频率，优化扫描算法避免无效的低信号质量wifi等。禁止app对某些频繁变化的广播响应，alarm对齐，网络心跳唤醒对齐。对kernel层ap和子系统，大多数就是bug类，需要进行修复。比如，豌豆荚app曾经做过后台静音播放一个音频文件，以期达到保活的目的，但是却让系统无法待机。还有些app出现1像素的activity保活奇葩方案。

3.4 场景功耗

所谓场景功耗优化，主要是针对用户使用频率较高的应用，比如主流游戏app、主流视频、和社交app、camera。

功耗和性能是对天平的两端，任何一端异常就让产品失败。所以各家也在找之间的平衡，有些产品主打性能，有些主打功耗。其实性能才是关键。app开发者也会面临同样的平衡。功耗也会像前面的稳定性问题一样，变成一个产品的基础体验，各家差异不会太明显。

要调优，就要找标准。可以和自己以前的产品对比，也可以和竞品对比。比如A、B两款手机，A手机玩王者荣耀电流500mA，但是有轻微卡顿；那么A款手机，电流可能有550mA，但是没有卡顿。这个根据各家产品策略进行取舍。

1. **通用优化方案：**

- GPU方案：减少draw内容，针对app动态限制应用分辨率，这个在gpu上面的优化相当明显。
- GPS方案：合并多app上报频率，动态上报频率，待机禁止乱用；
- app动态fps：除了控制cpu达到保底fps，可以通过各个应用适配性的动态fps，减少不必要的cpu、gpu消耗。卡顿主要不是由于fps低，而是由于fps不稳定，可以对比：电影24帧不卡顿，手机一般是60帧却出现卡顿；unity游戏app一般都是动态fps，开发者可以配置。
- 屏幕高Hz功耗：由于90、120Hz高帧率做出了更流畅的体验，但不是所有app都需要120Hz。针对不同app进行针对性的开启或关闭高Hz，也能一定程度降低功耗。
- 屏幕: 数据刷新有performance和on-demand模式。
- tp：固件优化采样率、点击坐标算法优化等。

2. **针对特殊app，单独优化：**
   减少layer：部分app多余layer，可以通过SurfaceFlinger进行移除。
   长连接：微信等app长连接问题可以在kernel层进行netfilter过滤动态限制，减少同步频率；

4. 性能优化-流畅不卡顿

2015年之后，由于早期的野蛮发展，各类app的各类保活方案，让国内的android生态比海外差好多。系统卡顿等问题层出不穷。各个厂商发布时候也重点突出自己的手机不卡顿，华为发布会号称18个月不卡顿，侧面反映出android手机出厂后系统调优很好，但是用久了，各类app让手机系统的原有方案不生效了，卡顿也再次出现。

产品的性能优化，除了硬件方案，还包括软件。

小米早期的宣传文案就是使用了多少频率的cpu、多少个cpu核、内存多大多大，比iPhone高多少多少。但实际情况还是没iPhone流畅。当然iPhone每代产品也是会说cpu提升xx%， gpu性能提升xx%。手机硬件的提升，app也会越做越大越复杂，以期更好的利用硬件性能，那硬件的富余也可能会出现不足。

除了硬件方面一次性交易，软件可以通过升级不对应对现实情况进行进一步调优。

2017年，苹果公司的ipad pro 120Hz屏 给人惊艳的体验。后续android也开始出现90Hz、120Hz屏。2020年，120Hz成了Android旗舰机标配。

其实性能优化涉及范围最广。android操作系统分层:
app-framework-native-|-kernel-driver-sensor/hardware

- 组件优化主要包括：CPU、gpu、flash disk、ddr memory、io、network；
- 流程优化主要包括：系统启动优化、app启动优化、app运行优化-卡顿jank优化；

当然，做app应用开发，性能稳定性等主要包括：cpu、gpu、memory、io、disk、network、卡顿、电量、稳定性、包大小，启动速度、UI；部分其实有重合。

做性能优化，主要思路包括：

- 没代码，是最牛的优化--在阶段去掉某些功能；
- 非关键逻辑，延后处理；
- 非关键逻辑，考虑提前执行；
- 关键逻辑，多线程，并发执行；
- 缓存部分结果，减少重复执行次数；
- 一次多操作，减少操作次数；

核心点：

- 别让cpu、gpu闲着，不管是block，wait、还是本身就处于idle空闲状态；关注多线程方案和锁等待问题；
- 毫秒级别的优化，小数据的串行处理，可能比多线程异步流程还快，因为没有任务调度延迟，比如tp触摸屏的input事件；
- 大数据的流程逻辑，队列、生产者消费者等异步化方案可能更稳定-削峰平谷，让整个流程稳定输出。-比如input事件在native层的处理：inputreader-inbound-inputdispatcher-outbound->transport;网卡驱动的buffer、ring长度、tcp的sync队列、socket的sb_buffer队列等；

4.1 优化方向

4.1.1 cpu优化

- 更优的调度器：CFS主要针对的是服务器设置，各个任务的公平调度；而android其实更关注的是top-app当前应用的性能，其它后台进程可以做适当的限制，Linaro的EAS更除了根据capacity容量的估算，还基于cluster按实际消耗进行调度优化。
- 开关cpu核心数量；
- 不同task任务绑定不同大小核：cpuset绑定独占cpu核心，taskset绑定任务指定cpu核心运行；
- 控制cpu核心的频率：governor 对cpu的电压频率进行控制；cpu的各种模式:on-demand、performance等
- 各个进程task根据不同的调度策略policy、优先级、进行cpu资源的动态占用，比如top-app、media进程等可能就需要设置较高的实时性策略；
- cgroups的cpu控制能限制指定task任务组cpu资源的，比如限制后台进程cpu使用率超过50%，开源节流-减少后台浪费，给前台任务更多资源。
- 开放接口给top-app，让合作的app可以根据自身的场景切换设置不同的cpu使用策略，比如cpu boost；
- 配置dtsi，cpu超频-为了性能;

以某款android手机为例：
支持的scaling_available_governors为
conservative ondemand userspace powersave performance schedutil
当前使用的governor是
schedutil

工具分析：

- 可以使用ftrace和systrace进行cpu使用情况分析，可以发现各个进程状态、cpu状态、cpu频率繁忙和空闲情况等；但是它是基于埋点的方案。
- 基于sampling采样方案；

- 转火焰图分析，快速发现系统瓶颈；

对app端来说，在cpu方面要多关注多线程问题，是否使用线程池，毕竟频繁创建线程还是有开销的，直接进行线程缓存，可以节约时间。线程数量是否太少，不能充分利用多核性能？是否太多造成调度延迟。

io密集型和cpu密集型的线程的需求其实不一样：

- cpu密集型，一般cpu-core + 1 个线程；如果系统把cpu资源都给你了，你要每个core都跑一个线程，留1个做备用应对任务调度；
- io密集型，可能就要多启动些线程，毕竟线程io block后，干不了其它事情，cpu都空闲出来了，增加线程可以增大抢占cpu的概率。

系统层面要关注cpu的饱和度，部分建议为cpu核心的0.7左右。具体还是要看实际测试情况。

android为了减少主线程任务，单独起了RenderThread线程进行draw frame。

4.1.2 gpu优化

app ui响应input操作后，通过measure、layout、draw，把具体合成功能给gpu操作。

- 优化ui布局，或交互方式，减少layer层级和对象数量；
- 部分数据可以缓存，比如字体资源等，各类资源纹理等缓存cache可以适当tuning调大；
- 裁剪分辨率，减少数据量；
- gpu的clock，频率tuning；
- gpu数据由串行改为并发执行；
- Adreno和 mali 2类gpu本身调度算法性能优化；

android 开始使用skia软件绘制，后面为了提高性能，引入gpu绘制；再后来引入vulkan。

4.1.2 memory优化

内存不足可能会导致出现各种莫名其妙的问题；内存不足出现各种页回收，碎片整理，swap交换。都可能导致卡顿。

- 维护系统可用内存最低阈值，减少卡顿

早期手机内存不大，一般采用kill杀后台进程方案；后续内存大了，就更多采用的冻结方案。
android整合利用linux的lmk：low memory killer方案，在内存不足时杀优先级低的进程，根据不同内存阈值，进行查杀，直到恢复指定内存数量。但是默认参数在国内不适用，各家也自己tuning适配的参数。
lmk在app启动阶段，不要进行kill操作，这样可以减少卡顿。

内存问题主要注意进程的内存泄漏，可以自己实现内存溢出监控-进程的绝对值和相对值/总内存的百分比阈值，也可以使用cgroups的内存控制功能实现。

“注意：不要出现swap交换。”

- 维护top app可用内存减少卡顿

app默认有最大heap使用限制，默认192MB。部分应用启动需要消耗很多内存，超过gc阈值，就会触发gc回收，造成卡顿。系统在app启动时候，可用默认调高阈值，启动完后退后台再gc调整恢复阈值。
也可以结合cgroups的 memory模块进行进程的内存限制。尤其是native的进程的限制，避免出现内存泄漏导致系统不稳定。

大型app，国民app也可以针对性的设置更大内存配额。减少gc卡顿。

- 后台、待机时内存回收和碎片整理

内存问题，大多是容量不够导致的问题。各类方案就是想方设法保证有充足的可用内存。

手机系统和传统的pc或服务器系统的使用场景不一样：

- 某时刻就那么几个可见app，和1个top app，大部分都是后台app进程；
- 手机有工作时间和待机时间--人总要睡觉。

内存优化，也可以基于性质，对后台进程进行碎片整理；长时间待机时进行更激进的整体碎片整理和内存优化。disk的trim 也是利用这个特点进行待机碎片整理，也避免开机disk check的耗时。

4.1.2.1 优化内存分配算法

- kernel：
  伙伴系统对大内存按page进行分配；
  小内存按slub/slab分配；

内存碎片问题，系统运行一段时间后，ddr上留下各种各样的空洞，导致alloc时连续空间不够。
导致page compact页面压缩。

- 用户层：
  malloc的分配方案也很多：
  高性能线程安全的:google 的tcmalloc、Facebook的jemalloc。
  android使用dlmalloc和jemalloc;

tcmalloc:

- 类似kernel分配方案；
- 各个线程存有ThreadCache局部缓存，进程使用CentralCache中心缓存；
- 小对象走局部缓存：参考slub/slab 分成100+个class等级；
- 大对象走中心缓存：从1page，2pages、3pages,..数组+链表形式。
- 同时增加了回收功能；

- 按大小分类small、large、huge；
- 采用chunk方式分割虚拟内存；
- 比tcmalloc开销大点，性能好点；

4.1.3 io/disk和filesystem

用户空间进程io系统调用到vfs虚拟文件系统；
vfs再标准化各类文件系统：ext4、f2fs等；
文件系统在对接generic block 块设备；
block dev 发起读写请求高io调度器；
io调度器类似cpu的进程调度器，看能否合并某些page以期批量处理数据，减少磁盘io操作；
时机一到，触发具体block的读写流程，交由各类块设备driver驱动进行读写磁盘。

read大致流程：

- 发起system call，系统调用 转到kernel内核态，用户线程block阻塞等待；
- vfs通过open时候的dirname找到inode对应的block上的位置返回fd，做个token标识。
- 通过读取的文件offset偏移，找到磁盘block对应的物理页映射到page cache缓存，方便下次直接读取缓存，减少磁盘io开销；
- 数据copy to user 系统调用返回进程，进程阻塞返回继续执行；

write大致流程：

- 通过fd找到对应inode的对应offset的page缓存页，没有就先预读加载进kernel；
- 更新数据到page cache，直接返回用户进程；
- kernel再选择时机一起回写到磁盘block，默认是5s；

block driver 是和disk配套的代码控制器，完成block的读写和校验，毕竟磁盘是死的。代码才是活的。

参考方案：

- 新的更适合移动设备的flash规范，新的文件系统类型；
- 根据具体具体业务类型，选择直接read/write， mmap，还是直接io等；
- 关注io操作的次数和buffer大小，按page大小倍速读写，可以减少io次数；
- io调度器电梯算法的请求队列和调度策略设置:cfs、noop、deadline；
- 顺序io可以适当增大预读size;

各类android 抓取log方案，就采用的是mmap方案，由于进程vma和kernel地址空间是共享的，可以减少1层系统调用开销。
huawei: iotrace, io monitor

对app端来说，sqlite和shared preference性能影响较大。

sqlite的优化：
page大小、buffer大小；wal模式，关闭统计功能，mmap支持。

shared preference重写优化：比如mmap方式替换；按uid通过binder进行本地server端更新。

4.1.4 网络优化

不管是系统或者是app，网络优化都相当重要。如果时不时来一下没网或者网络差，那这个产品可能没卖不出去了。

网络优化，主要关注吞吐量和延迟。重点场景为弱网和网络拥塞。

数据在 终端-传输路径-服务器 传输。网络流程从下往上包括：

- 设备外的路由间转发；
- 网卡收发数据;
- 网络层ip；
- 传输层tcp、udp;
- 应用层；

4.1.4.1 服务器端

客户端按不同ip负载均衡，多数据中心，就近的数据中心，CDN；

4.1.4.2 传输路径

有钱人用专线。终端到服务器中间可能经过各类路由器，要注意照顾最差的路由器的兼容性问题。

4.1.4.3 应用层

4.1.4.3.1 dns 问题

dns劫持比较多，app大多用httpdns；
dns预读；
大公司服务器多，还会基于ip的地理位置进行就近返回。当然也会预制几个服务器ip hash兜底，加快访问速度。
android系统层做dns本地缓存，类似dnsmasq，结合各个app进行预缓存或app下载安装时就做预解析，这个对app尤其是普通app冷启动提速帮助很大。（但别破坏了大型app的客户端做的负载均衡）。

有精力的手机厂商还可以底层监控不同gps位置，不同运营商、网络类型(固网还是基站（gps+基站id）)下的各个域名解析速度进行汇总分析，发展处第二曲线业务-dns服务商。

法等，同时配合服务器一起改造，做到0 rtt；httpdns定制等。--cronet最好有人牵头做公共网络库，毕竟有些三方sdk公司不支持这个，也会影响app整体的网络模块定制。

4.1.4.3.2 app

压缩数据、压缩算法、请求合并、长连接代替短连接、多级缓存、epoll、aio、连接池、预连接、多路复用、spdy、quic、优先ipv6等；

4.1.4.3.3 socket层

- 单个socket的optmem_max缓冲区大小；
- 整个socket的发送接收缓冲区大小；
- 传输层的tcp/udp发送接收缓冲区范围大小；
- socket option可以控制的东西很多,除了timeout、buffer，还有TCP_NODELAY、TCP_CORK等。
- 关注/proc/sys/net/core/*；

还有一个重要问题：各类超时要针对移动网络动态适配，快速超时快速重试，提升用户可感知的体验。

4.1.4.4 传输层

作为手机端的linux系统，不大可能出现服务器环境的高并发连接。它本身定位为客户端，相关问题也会简单很多，比如tcp的time_wait状态过多等，当然android上可以按uid进行端口数量控制。tcp协议是最复杂，边界问题最多的协议。

最简单的比如tcp/udp的rmem、wmem 调整、tcp_synack_retries、tcp_fin_timeout、tcp_tw_reuse等。
长连接的tcp_keepalive_time、tcp_keepalive_intvl、tcp_keepalive_probes。
根据MTU动态调整包大小，减少分片。

其它的ip_local_port_range、调大fd数量。

其实tcp最开始是基于有线网络低网速进行设计的，目前的网络拥塞冲突算法在移动网络场景不是特别优秀。google基于自身搜索引擎和youtube等实际业务的网络监控分析，提出了tcp bbr拥塞控制算法，效果喜人。华为接入的multipath tcp已经使用bbr。

4.1.4.5 链路层

- 网卡硬中断配置cpu亲和性：/proc/irq/xxx/下的smp_affinity和smp_affinity_list;
- irqbalance(默认每10s扫描interrupt情况，进行亲和性重新设置);
- 软中断：RPS（Receive Packet Steering）hash packet的四元组到对应cpu，/sys/class/net/xx/queues/rx-0/rps_cpus；
- 软中断：RFS（Receive Flow Steering)按flow绑定cpu，提高缓存命中率：/proc/sys/net/core/rps_sock_flow_entries、/sys/class/net/xxx/queues/rx-0/rps_flow_cnt；
- 网卡的多队列、队列长度、buffer等；
- cgroups net子系统按class和priority进行控制，~tc的qos；
- 协议层的部分功能下沉到网卡处理：TSO（TCP Segmentation Offload）和 UFO（UDP Fragmentation Offload）、GRO（Generic Receive Offload等。
- 参考dpdx xdp: kernel层逻辑上浮到user space和下沉到网卡;
- 由于手机系统的特殊性，定制化网卡是差异化竞争的一个亮点。

4.1.4.6 终端：移动网络和wifi

弱网：移动网络相对有线网络存在更大的不稳定性。弱网问题就相当严重。modem对小区的弱信号和小区拥塞分别进行不同的tuning，对弱信号的排除，失败快速切换其它小区，小区禁用时间的重新设置。核心点：弱信号时快速试错，拥塞时快速竞争占坑。

wifi: RF射频天线的布局，rx\tx参数调优，提高发射功率和接收增益，wifi的EDCA\QoS\竞争窗口、速率选择算法优化、wifi tcp buffer优化等。

监控接口网络数据包的收发，错误、丢包等数据，动态切换网络或增大功率等各类结构定制化优化方案。modem也可以根据gps和记录等信息优先判断适合的网络类型，加速网络建连。

4.2 场景优化

4.2.1 工具

systrace: 基于linux的ftrace进行改造的埋点跟踪技术方案。原生是配置参数，抓取日志，chrome上分析。可以改进为在线实时分析方案。pc上在chrome上实现服务端（也可以使用node.js自己实现chrome的trace分析功能）。服务端进行开关动态配置，手机端通过网络进行实时上传trace日志，pc上就可以出现流式动态跟踪数据，类似android studio的cpu profiler功能。

精简版本的卡顿监控，可以打开cpu 等基本tag，检查到卡顿自动dump trace日志上传日志服务器。
可以通过looper，choreographer、gpu，renderthread、Surfaceflinger等卡顿监控点。

gpu工具：Adreno profiler 、mali debugger等工具。

perf、systemtap、dtrace等在android上的定制化应用。

4.2.2 app启动优化

- apk redex优化；
- 尽量保证热启动；
- 启动时候禁止kill，gc等耗时任务；
- Launcher优先响应click事件，其它任务异步化或延迟执行；
- view对click事件提前串行执行；
- 启动动画单独设计和速率等调整，这个很影响速度；
- app进程预创建和缓存-可以节约30-50ms左右；

- app的theme、layout等和业务逻辑无关的资源可以预先load和缓存-可以节约上百ms；
- app的lib so使用监控，预mmap；
- 系统层so合并和缓存，提升so加载速度;
- app 减少不必要业务路径，延迟不重要逻辑；
- 系统级dns缓存-可以节约100~400ms左右；
- app启动流程task化，充分利用多核cpu，避免锁等待。
- ams启动大锁优化，有限保证top-app和system_server各个核心线程占有cpu优势资源;
- 各类boost和top-app的大核独占绑定、优先级调高，后台进程优先级调低；

4.2.3 app运行优化

- 非主activity等class，可以在startActivity后手动classloader.loadClass()；
- looper里消息分类和优先级：类型数组->优先级链表/数组->同优先级消息链表；
- tp报点优化：缩短休眠唤醒间隔、提高采样率，优化坐标计算算法；
- 不等sync信号，直接先处理input事件流程，点击提前处理，滑动阈值、摩擦力参数优化,input也可以单独线程处理；
- view对click事件考虑串行不走post，click effect异步化或优先处理onclick;
- 子线程分担ui线程input/../measure/layout任务-facebook方案可以参考；
- app动态分辨率-部分app降低分辨率不影响体验；
- app自身各类listview\recycleview的item view细化缓存；
- app自身layout优化，自定义层级、减少层级等；
- framework层measure时动态优化不必要的viewgroups，减少层级；
- view跨context缓存，切换activity时更新view的context相关内容；
- gpu数据并行化；
- SurfaceFlinger过滤多余layer，动态fps；
- 其它系统级：app线程的cpu核心绑定，cpu\gpu频率，场景化boost，预留足够内存，减少不必要的io，提高cpu、disk、net 优先级。irq balance。cgroups可以限制cpu、mem、disk 、net、io，要充分利用保证top-app优先使用。
- 注意thermal导致的限速降频等；
- 整个系统为了高扩展性移植性等架构，多采用分层结构，但客制化对应的系统可以打破原有架构设计，根据top-app等android特有性能设计更好优先级、实时性的场景化方案，比如优先响应top-app的网络请求，io请求等。
- 机器码优化运行，减少字节码解析运行--目前几乎都是arm芯片，google可以考虑放弃通用性，单独编译成机器码级别app。

其实，google现在也越来越关注开发者意见了。系统方可以开放2类api:

- app启动加速api:提供api给app 预加载缓存，提高冷启动速度；
- app运行调频api: 微信把厂商提供给它的boost api封装成Hardcoder也开源的；

4.2.4 题外话

国内android生态和国外完全两码。

后半段出现了微信等超级app，做的太好，以至于用户会觉得出问了都是手机的问题，而不是微信app的问题，间接“绑架”手机厂商，导致微信在各个厂家几乎都是各类白名单。超级app推出的小程序等，直接跳过app商店，导致分发广告等收入提成的减少，这些事情都威胁到终端公司。

国内厂商组成的android联盟，一方面要防止google的更深入的控制，一方面要避免硬件同质化和性能过剩没有卖点，担心为超级app做嫁衣，联合起来切分国民app的蛋糕，步入互联网化，慢慢达到国内android系统端的统一。

手机厂商由于有用户的所有使用习惯，结合AI、大数据，进行用户画像，以更精准的提供千人千面的手机使用体验。以各家推出的浏览器为例，虽然都是基于chromium浏览器改的，但是能更深入到系统层，提供比google的chrome更好的使用体验，毕竟是站在巨人的肩上。当然也可以基于各自的上亿用户量，推出自己用户的超级app--厂商玩这个就要差点火候了。

话题转回来：而手机设备的后续的发展方向，会慢慢的项软硬定制化深入，才能差异化。比如vivo花8000w和一加花1个亿都和三星深度定制屏幕，跑在三星自己手机的前面了。比如这次的一加和Unity合作。比如以前的华为GPU定制，和王者荣耀合作。

4.3 思路来源

思路比具体方案更重要:

- app或系统自身的各类指标监控，关键点同步到服务端--标准产品研发逻辑；
- 用户反馈问题专项攻关--能发现更广更接近用户的迫切问题；
- 各类方案借鉴：前端、后台、移动端等的业务方向，同类产品思路的相互借鉴，app、framework、native、kernel、driver、硬件等层级方向；--跨界思考
- 逆向各类rom、app；专利分析，产品宣传分析;
- 高薪招聘xx岗，面试友商相关人员，进行方案、思路获取。
- 上下游合作：华为和王者荣耀合作，一加和游戏引擎厂商unity3d合作。

思路前2点属于常规型；后3点属于扩展型。

技术方案借鉴，是“跨界”的做技术思路。目前发现的好多方案都是在不同产品或系统平台之间的相互切换移植参考。

Android 端有个好处是，都有公共的母版：google版本，然后才是其他oem rom，差异化分析相当简单。对逆向分析竞品帮助很大。是android平台比较特有的思路。

而高薪招聘幌子则是比较难以防范的歪招。这个一般是实力弱的公司团队招聘实力强的公司的人员才会用到。面试者为了证明自己能力，会对重点项目工作进行阐述，甚至细节公开。面试者可以说7分，留3分。7分里面的数据可以给大点误差，3分主要包括大坑。如果真心招聘，入职再更正。如果恶意，会能误导一部分。由于恶意招聘大多是低水平的团队，这个应对方案他们一般也发现不了。

前面方案的目标都是为了找方向。找到方向后沉下去专项攻关，就需要团队在各个方向都有专业人员储备--术业有专攻。

各类软件硬件安全问题，隐私问题会越来越重要。
出现过FPGA芯片安全漏洞，cpu指令重排漏洞，各类os系统、app等安全漏洞。
各个大公司因为隐私问题被罚款的案例也时有发生。

- android rom 反编译问题；
- prop属性按uid单独独立出来设计，加入pkgname和user group做盐；
- imei、mac等信息对app全部虚拟化，各个app获取不同的固定值，用户还可以手动不定时随机刷新。
- google版的android权限申请问题，被国内app滥用了，如果不给权限，app就退出。国内各厂商提供的passive permission control更适合隐私控制，app能正常使用，但是敏感权限控制可以拒绝某些数据读取申请。

每年google 在春节左右会举办各个上下游厂商的内部camp，国内厂商可以反馈国内android生态的典型问题，推动google优化升级。

6. 总结

所有的手机产品，都是在找超出用户预期、或者和友商差异化竞争的方案。从最开始的频硬件产品、到后面功耗、性能等。有时候，不是有好方案，就马上全上马；而是研究行业发展趋势，选取其中几个有竞争性的点，抢占先发优势，打动用户，撇开竞品。

联想当时选取了防伪基站作为卖点，只讨好了小部分的安全敏感、遇到过诈骗的用户。

华为当年的xx turbo，其实就是各种小的优化点，带来的大优化收益，不好讲，就起名个唬人的turbo，有话题，有热度，有卖点。

部分厂商，手机做得深的，会在某些方向引领产品方向，行业方向，国家标准方向。技术影响力也会辅助产品销量。

能持续的卖出更多的好产品，获取更大的利润，才是手机产品能活下去的根本。

好奇今年的卫星短信功能能有多大助攻辅助华为、苹果销售。

Android系统优化的那10年　脉脉

END

相关推荐　最新发布　大家都在看

谁在打破工业供应链的那堵墙！
养老金计算5年一档？那我交了18年的怎么算？
2023年是未来10年最好的一年，动起来就有机会！
2023年10月启动
最戳心的那10句话

评论

我来说几句...