

Usage and Comparisons of Control Group in Android AOSP: Marshmallow and Before

2016/1/11

Yoshi

State-of-the-Art

- In Seattle on August 20th, 2015
 - Topic: big.LITTLE Support in Android
 - Todd Kjos & Tim Murray from Google

What We've Seen

- Most vendors use HMP patchset + modifications
- Tunings generally minimize use of A57s except for very obviously heavy tasks by default (large window, large upmigrate)
- Lots of special touchboost-like events to tune for specific actions (scroll, zoom, launch new app, ...)

Problems

- Basically impossible to tune
- Unimportant work can end up on the big cores
- No guarantee that work you really care about will end up on big cores when necessary

Improvements in Marshmallow

- Android Marshmallow brings better support for big.LITTLE using cpusets and new hooks
- Basic idea: ActivityManager knows a lot about relative importance of apps, so use that to better inform the scheduler

cpusets in Android

- Two cpusets set up at init: foreground and background
- Foreground should contain all cores, background contains a small subset (generally one little core)
- cpuset configuration is in device-specific init.rc files

Question 1: Differences between
/dev/cpuctl/ & /dev/cpuset/

init.rc @ /alps/system/core/rootdir/

```
# Create cgroup mount points for process groups
mkdir /dev/cpuctl
mount cgroup none /dev/cpuctl cpu
chown system system /dev/cpuctl
chown system system /dev/cpuctl/tasks
chmod 0666 /dev/cpuctl/tasks
write /dev/cpuctl/cpu.shares 1024
write /dev/cpuctl/cpu.rt_runtime_us 800000
write /dev/cpuctl/cpu.rt_period_us 1000000

mkdir /dev/cpuctl/bg_non_interactive
chown system system /dev/cpuctl/bg_non_interactive/tasks
chmod 0666 /dev/cpuctl/bg_non_interactive/tasks
# 5.0 %
write /dev/cpuctl/bg_non_interactive/cpu.shares 52
write /dev/cpuctl/bg_non_interactive/cpu.rt_runtime_us 700000
write /dev/cpuctl/bg_non_interactive/cpu.rt_period_us 1000000
```

Only
bg_non_interactive
group is made

$52/(1024+52) \approx 5\%$

New cgroup in init.rc

```
# sets up initial cgroups for ActivityManager
mkdir /dev/cpuset
mount cpuset none /dev/cpuset
mkdir /dev/cpuset/foreground
mkdir /dev/cpuset/background
# this ensures that the cgroups are present and usable, but the device's
# init.rc must actually set the correct cpus
write /dev/cpuset/foreground/cpus 0
write /dev/cpuset/background/cpus 0
write /dev/cpuset/foreground/mems 0
write /dev/cpuset/background/mems 0
chown system system /dev/cpuset
chown system system /dev/cpuset/foreground
chown system system /dev/cpuset/background
chown system system /dev/cpuset/tasks
chown system system /dev/cpuset/foreground/tasks
chown system system /dev/cpuset/background/tasks
chmod 0664 /dev/cpuset/foreground/tasks
chmod 0664 /dev/cpuset/background/tasks
chmod 0664 /dev/cpuset/tasks
```

AMS!!

Reply to Question 1

- Before Android M, we've already see **bg_non_interactive** as a control group
 - cpuctl is for “% of CPU Usage”
 - Background = 5%
 - Foreground = 95%
- In Android M, cpuset is *another control group* for setting big.LITTLE

Question 2: Does Google Have Similar Concepts of APO?

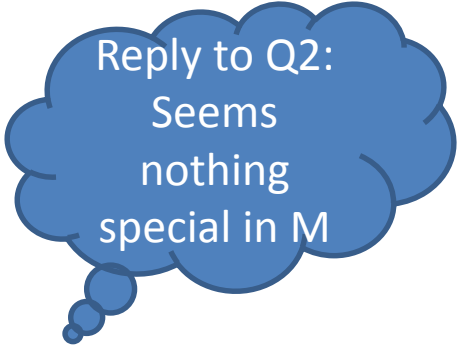
Similar to APO =

Interceptions in **activityResumed** and
activityPaused in AMS

Check Pause and Resume Methods

```
@Override
public final void activityResumed(IBinder token) {
    final long origId = Binder.clearCallingIdentity();
    synchronized(this) {
        ActivityStack stack = ActivityRecord.getStackLocked(token);
        if (stack != null) {
            ActivityRecord.activityResumedLocked(token);

            /// M: App-based AAL @[
            updateSmartBacklightLocked(token);
            /// M: App-based AAL @}
        }
    }
    Binder.restoreCallingIdentity(origId);
}
```



Reply to Q2:
Seems
nothing
special in M

```
@Override
public final void activityPaused(IBinder token) {
    final long origId = Binder.clearCallingIdentity();
    synchronized(this) {
        ActivityStack stack = ActivityRecord.getStackLocked(token);
        if (stack != null) {
            stack.activityPausedLocked(token, false);
        }
    }
    Binder.restoreCallingIdentity(origId);
}
```

Question 3: How CPUSET cgroup is used?

Find CPUSETS

```
[mtk05583@mtks1t203 system]$grep USE_CPUSETS * -R
core/libcutils/Android.mk:LOCAL_CFLAGS += -DUSE_CPUSETS
core/libcutils/Android.mk:LOCAL_CFLAGS += -DUSE_CPUSETS
core/libcutils/sched_policy.c:#ifdef USE_CPUSETS
core/libcutils/sched_policy.c:#ifndef USE_CPUSETS
```

Find CPUSETS

android_os_Process_setProcessGroup
@base/core/jni/android_util_Process.cpp

```
#ifdef ENABLE_CPUSETS
    int err = set_cpuset_policy(t_pid, sp);
    if (err != NO_ERROR) {
        ALOGE("Error set_cpuset_policy pid:%d errno:%d \n", pid, errno);
        signalExceptionForGroupError(env, -err);
        break;
    }
#endif
    continue;
}
int err;
#ifdef ENABLE_CPUSETS
    // set both cpuset and cgroup for general threads
    err = set_cpuset_policy(t_pid, sp);
    if (err != NO_ERROR) {
        ALOGE("Error set_cpuset_policy pid:%d errno:%d \n", pid, errno);
        signalExceptionForGroupError(env, -err);
        break;
    }
#endif
#endif
```

attachApplicationLocked
@
base/services/core/java/com/android/server/am/ActivityManagerService.java

... Several calls

applyOomAdjLocked
@
base/services/core/java/com/android/server/am/ActivityManagerService.java

setProcessGroup
@
base/core/java/android/os/Process.java

android_os_Process_setProcessGroup
@
base/core/jni/android_util_Process.cpp

set_cpuset_policy
@
system/core/libcutils/sched_policy.c

set_sched_policy

Reply to Question 3

- Seems to be used when adjusting Out-of-Memory adjustment
 - One-time call, not resume/pause every time

Summary

- We went through how new CPuset is used in AOSP 6.0
- We compared the differences between Google's idea and APO
- Cgroups interface and functionalities are changed by Google (Not presented in the slides)
- Partial conventional cgroups in Vanilla's kernel works
 - Freezer & *vanilla's* cpuset have been tested
 - Why??

Ongoing

- TOSHIBA's study on improving real-time tasks' performance
 - http://elinux.org/images/8/84/Real-Time_Tasks_Partitioning_using_Cgroups.pdf
- Should we consider APO, EAS based on cgroup?
- AAF had a meeting for discussing cgroups
- SCHED_DEADLINE introduces *constant bandwidth server* scheduling policy in Linux Kernel 3.14
 - Android L uses 3.10
 - Android M uses 3.18

Active Discussion in LPC Android u-Conference (2015/8/20)

- Most of the EAS discussion focused on energy
 - Interactive cpufreq governor in Android is mostly focused on latency
 - how does latency come into the picture and how can it be controlled?
 - The **schedtune sysfs** knob allows for similar boosting
 - » When an event occurs, **Android user space** could provide some short-term boosting to improve latency response
 - Riley Andrews from Google noted that **the interactive cpufreq governor gives this benefit without user space having to do anything**
 - Lelli pointed out that the governor does it a bit blindly, so this allows a more informed and focused response that could save power
 - » I think the basic concept is the same as APO

Xiaomi's Smart CGroup

- The new Smart CGroups will distribute the CPU usage of process groups better between cores: so instead of making one core do all the work, the work is divided as balanced as possible to make the processor run as efficient as possible.
 - <http://en.miui.com/thread-35193-1-1.html>

Reference

- Binder's priority propagation
 - <https://github.com/keesj/gomo/wiki/AndroidScheduling>
- Summary of the LPC Android microconference
 - <http://lwn.net/Articles/657139/>
- Google I/O 2015 on cgroup and big.LITTLE
 - <http://www.linuxplumbersconf.org/2015/ocw//system/presentations/3399/original/LPC-tkjios.pdf>

BACKUP

Sched-TUNE

Userspace power/performance tunable knob*

- Stacked on top of Sched-DVFS
- Interacts with CFS scheduler (for the time being)
- Global and CGroup based (per-task) interface
 - global: `/proc/sys/kernel/sched_cfs_boost`
 - cgroup: `/sys/fs/cgroup/stune/performance/schedtune.boost`

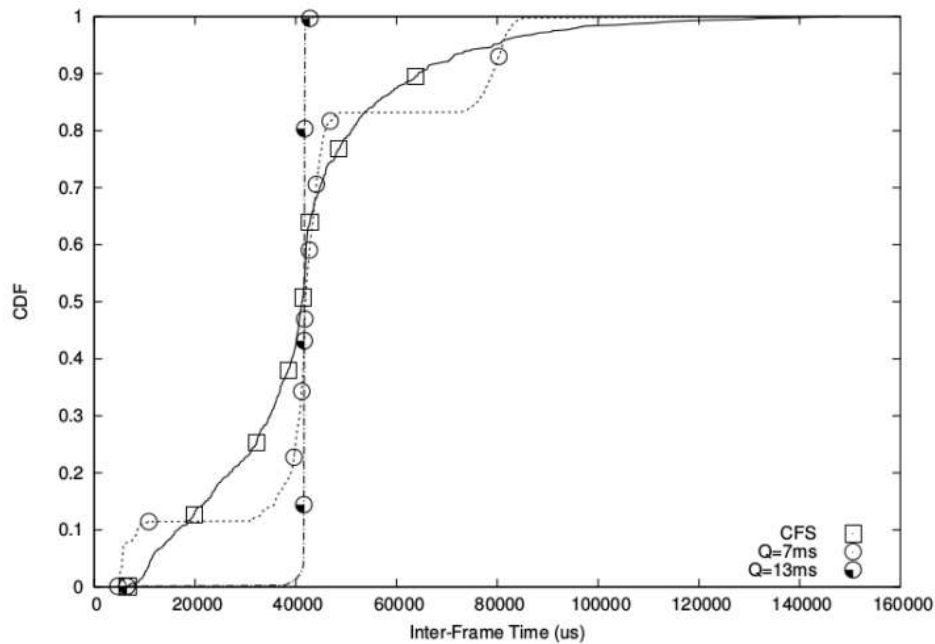
Deadline scheduling (a.k.a. SCHED_DEADLINE)

it's not only about deadlines

- it's a relatively new addition to the Linux scheduler
- real-time scheduling policy
- higher priority than SCHED_NORMAL and SCHED_FIFO
- allows explicit per-task latency constraints
- enables predictable task scheduling, avoids starvation and
- enriches scheduler's knowledge about tasks' QoS requirements
- it implements
 - Earliest Deadline First (EDF): tasks with earlier deadlines have higher priorities
 - Constant Bandwidth Server (CBS): reservation based scheduling
- CBS it's the cool thing here

SCHED_DEADLINE: some numbers

- MPlayer with HD movie
- QoS metric: IFT, difference between DT of current and previous one
- Variation in IFT \rightarrow video doesn't play smoothly
- frame rate = 23.9fps, IFT = $41708\mu s$
- $P = \text{IFT}$ (for SCHED_DEADLINE)
- 6 other instances of ffmpeg in background
- CFS QoS highly dependent on system load
- With SCHED_DEADLINE player not affected (with reasonable CPU usage)



Discussion

- Better quality of service provisioning
- Additional information for the scheduler
- How do you like the interface?
- Does it look usable?
- How about using it for audio pipeline instead of SCHED_FIFO (cit. Google I/O 2013 - High Performance Audio) ?
- SurfaceFlinger maybe?

-Combined dalvik/vm/Thread.c and -frameworks/base/include/utils/threads.h

Thread.priority	Java name	, Android property name	, Unix priority
1	MIN_PRIORITY	ANDROID_PRIORITY_LOWEST,	19
2		ANDROID_PRIORITY_BACKGROUND + 6	16
3		ANDROID_PRIORITY_BACKGROUND + 3	13
4		ANDROID_PRIORITY_BACKGROUND	10
5	NORM_PRIORITY	ANDROID_PRIORITY_NORMAL	0
6		ANDROID_PRIORITY_NORMAL - 2	-2
7		ANDROID_PRIORITY_NORMAL - 4	-4
8		ANDROID_PRIORITY_URGENT_DISPLAY + 3	-5
9		ANDROID_PRIORITY_URGENT_DISPLAY + 2	-6
10	MAX_PRIORITY	ANDROID_PRIORITY_URGENT_DISPLAY	-8

Binder and priorities

The binder mechanism also propagates priorities. That is the binder process called will run with the same priority as the caller.