Linsched

Making it useful!

Dhaval Giani

Slides are available

http://goo.gl/GFqxV

What if we could run the kernel scheduler in userspace?

Linsched

- Developed at UNC
- Picked up by Google
- Can be run in a variety of ways
  - busy tasks
  - Sleep/Run following some distribution
    - Sleep times follow a normal distribution
    - Run times follow a lognormal distribution

Current Status

- Updated to v3.5
  - ~20 lines diff from mainline
  - Somewhere on github
- Has another branch based on tip
  - Still sucks a bit!
- Linsched in its own architecture
  - Based on x86
  - Lots of ugliness hidden in the architecture
- Still fragile
  - A lot of the kernel is stubbed inside the architecture
  - Changes in kernel API cause linsched to break
    - Recent example, cgroup API changing from 3.3 to 3.4

Short Demo

Linsched Results

```
 level:   SYSTEM
cpu  0:  0.0000%
average imbalance: 0.000000
dhaval@cluster020:~/kernel/github/linux/tools/linsched/tests/results/uniprocessor-results$ cat sim-1
TOPO = uniprocessor, tg_file = ../mcarlo-sims/sim-1, duration = 60000
Task id = 3 (1), exec_time = 464472352, run_delay = 10223332775, pcount = 823
Task id = 4 (2), exec_time = 517638853, run_delay = 10892902679, pcount = 935
Task id = 5 (3), exec_time = 575519264, run_delay = 14318013251, pcount = 1076
Task id = 6 (4), exec_time = 553062005, run_delay = 13682721126, pcount = 1044
Task id = 7 (5), exec_time = 609886402, run_delay = 15414402898, pcount = 1147
Task id = 8 (6), exec_time = 615271351, run_delay = 14948568297, pcount = 1125
Task id = 9 (7), exec_time = 599811941, run_delay = 15502850994, pcount = 1090
Task id = 10 (8), exec_time = 565220830, run_delay = 14082440937, pcount = 1082
Task id = 11 (9), exec_time = 597146808, run_delay = 15043017675, pcount = 1112
Task id = 12 (10), exec_time = 762585899, run_delay = 19046836849, pcount = 1456
Task id = 13 (11), exec_time = 608258321, run_delay = 15072663796, pcount = 1136
Task id = 14 (12), exec_time = 579548408, run_delay = 13702970649, pcount = 1065
Task id = 15 (13), exec_time = 932878554, run_delay = 27694854046, pcount = 1791
Task id = 16 (14), exec_time = 1053604367, run_delay = 30170557698, pcount = 2075
Task id = 17 (15), exec_time = 1142163841, run_delay = 33905673797, pcount = 2246
Task id = 18 (16), exec_time = 1201417072, run_delay = 37515087371, pcount = 2344
Task id = 19 (17), exec_time = 1121573955, run_delay = 34779290183, pcount = 2198
Task id = 20 (18), exec_time = 897421817, run_delay = 24490012824, pcount = 1790
Task id = 21 (19), exec_time = 1084524499, run_delay = 30281071850, pcount = 2089
Task id = 22 (20), exec_time = 1268353064, run_delay = 37615146003, pcount = 2469
Task id = 23 (21), exec_time = 1179441931, run_delay = 34588775024, pcount = 2237
Task id = 24 (22), exec_time = 1055963428, run_delay = 32367640222, pcount = 2066
Task id = 25 (23), exec_time = 1378192030, run_delay = 56764903460, pcount = 1410
Task id = 26 (24), exec_time = 1380357634, run_delay = 57175439111, pcount = 1348
Task id = 27 (25), exec_time = 1403381413, run_delay = 58140121515, pcount = 1441
Task id = 28 (26), exec_time = 1405119018, run_delay = 58527597130, pcount = 1308
Task id = 29 (27), exec_time = 1388704837, run_delay = 57282205741, pcount = 1435
Task id = 30 (28), exec_time = 1384946477, run_delay = 57265012629, pcount = 1373
Task id = 31 (29), exec_time = 1404665371, run_delay = 58342743144, pcount = 1448
Task id = 32 (30), exec_time = 1380971076, run_delay = 57129617159, pcount = 1402
Task id = 33 (31), exec_time = 1404950714, run_delay = 58358379762, pcount = 1393
Task id = 34 (32), exec_time = 1386691809, run_delay = 57166033090, pcount = 1463
Task id = 35 (33), exec_time = 1404586404, run_delay = 57606601207, pcount = 5221
Task id = 36 (34), exec_time = 1404912590, run_delay = 57583850027, pcount = 5067
Task id = 37 (35), exec_time = 1404450630, run_delay = 57605731495, pcount = 5176
Task id = 38 (36), exec_time = 1405067341, run_delay = 57546566567, pcount = 5425
Task id = 39 (37), exec_time = 1404595647, run_delay = 57536202849, pcount = 5194
Task id = 40 (38), exec_time = 1404695366, run_delay = 57588302342, pcount = 5241
Task id = 41 (39), exec_time = 1404757836, run_delay = 57511148305, pcount = 5431
Task id = 42 (40), exec_time = 1404508973, run_delay = 57547496060, pcount = 5423
Task id = 43 (41), exec_time = 1404455859, run_delay = 57726787747, pcount = 4578
Task id = 44 (42), exec_time = 1404743638, run_delay = 57597975280, pcount = 5138
Task id = 45 (43), exec_time = 1404513072, run_delay = 57526400368, pcount = 2788
Task id = 46 (44), exec_time = 1404813689, run_delay = 57345555076, pcount = 2792
Task id = 47 (45), exec_time = 1404814039, run_delay = 57486466555, pcount = 2762
Task id = 48 (46), exec_time = 1404925313, run_delay = 57581256149, pcount = 2800
Task id = 49 (47), exec_time = 1405291126, run_delay = 57457237709, pcount = 2802
Task id = 50 (48), exec_time = 1405082280, run_delay = 57565658899, pcount = 2742
Task id = 51 (49), exec_time = 1405328734, run_delay = 57568832844, pcount = 2803
Task id = 52 (50), exec_time = 1404961040, run_delay = 57539631655, pcount = 2793
Task id = 53 (51), exec_time = 1404531500, run_delay = 57549097933, pcount = 2728
Task id = 54 (52), exec_time = 1405159583, run_delay = 57460244010, pcount = 2788
Total exec_time = 59999940001
CGroup = / (0), exec_time = 59999940001

------ group runtime
CGroup = / (0), exec_time = 59999940001
------ sched stats
version 15
timestamp 4294727296
cpu0 0 0 128752 0 96281 96281 59999853894 2222421926752 128109
------ nohz residency
Time spent with tick disabled over 59999 ms:
 level:   SYSTEM
cpu  0:  0.0000%
average imbalance: 0.000000
dhaval@cluster020:~/kernel/github/linux/tools/linsched/tests/results/uniprocessor-results$
```

Issues

- All this information is useful
  - We can already process a lot of it
  - diff-mcarlo-sim tool
  - Gives good view of how the load balancer is performing

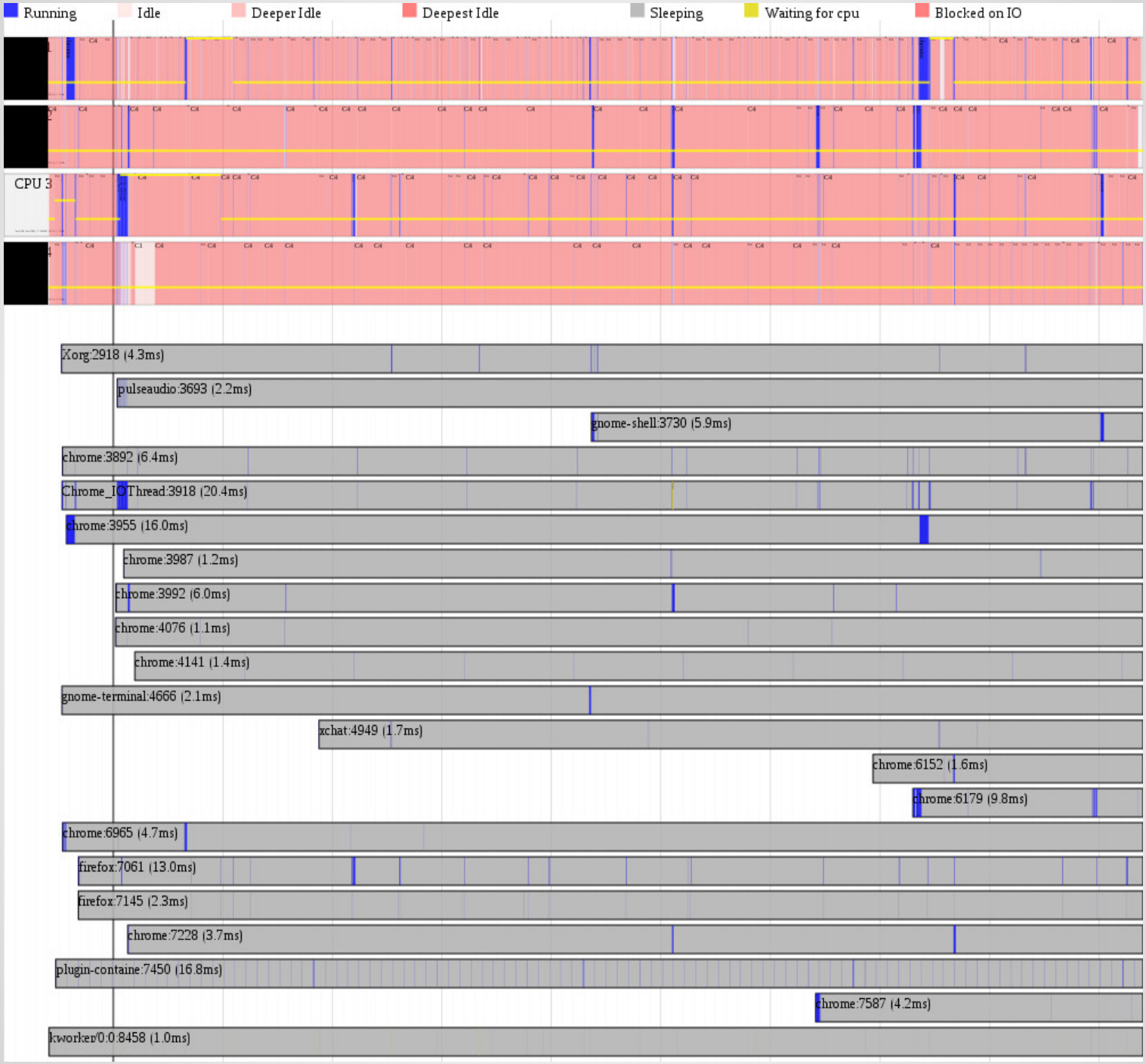
diff-mcarlo tool

```
dhaval@cluster020:~/linux/tools/linsched/tests$ ./diff-mcarlo-500
results/ ~/kernel/github/linux/tools/linsched/tests/results/
avg_results
dual_cpu-results -77.1444480822
dual_cpu_mc-results -71.6951469319
hex_cpu_dual_socket_smt-results 200.542241158
quad_cpu-results -94.1485217335
quad_cpu_dual_socket-results -109.407197707
quad_cpu_mc-results -79.4337981142
quad_cpu_quad_socket-results 358.981127431
uniprocessor-results 0.0
dhaval@cluster020:~/linux/tools/linsched/tests$
```

April 2012

PJT: What I would really like is a view similar to what we get in time chart

- But, timechart code is C, processes a perf. data file.
- Hmm, well, we are using kernel infrastructure, linsched is its own architecture
- We could have linsched perf events


Running
Idle
Deeper Idle
Deepest Idle
Sleeping
Waiting for cpu
Blocked on IO
CPU 3
Xorg:2918 (4.3ms)
pulseaudio:3693 (2.2ms)
gnome-shell:3730 (5.9ms)
chrome:3892 (6.4ms)
Chrome_IOThread:3918 (20.4ms)
chrome:3955 (16.0ms)
chrome:3987 (1.2ms)
chrome:3992 (6.0ms)
chrome:4076 (1.1ms)
chrome:4141 (1.4ms)
gnome-terminal:4666 (2.1ms)
xchat:4949 (1.7ms)
chrome:6152 (1.6ms)
chrome:6179 (9.8ms)
chrome:6965 (4.7ms)
firefox:7061 (13.0ms)
firefox:7145 (2.3ms)
chrome:7228 (3.7ms)
plugin-containe:7450 (16.8ms)
chrome:7587 (4.2ms)
kworker/0:0:8458 (1.0ms)

Perf support in linsched

- Need to stub out functions we don't need
- Provide support for events
- Provide support for software events (maybe not needed ?)
- HAVE_PERF_EVENTS

perf.data format ?!

Tuesday evening

acme: Why don't you just reuse the perf-infrastructure as it is. Modify perf to call into linsched as opposed to a syscall, and load linsched from perf itself, reuse the ringbuffer from perf.

- Hindsight is 20/20!

Discussions!

linsched

- https://github.com/linsched/linux/tree/linsched-tip
- https://github.com/linsched/linux/tree/linsched-35

Acknowledgements

- Paul Turner
- Stephane Eranian
- Arnaldo Carvalho de Melo
- Steven Rostedt