Making it useful!
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: 0.000k
average imbalance: 0.00000
chord[cluster02]:/kernel/gsthub/linux/tools/linsched/tests/results/unprocessor_results: cat sim-1
1000: unprocessor: cg file: /var/log/900/900: duration: 0.0000
Task id: 4 [1]. exec_time: 46442752, run_delay: 102356275, pcoumt: 825
Task id: 2. exec_time: 51785838, run_delay: 108850538, pcoumt: 998
Task id: 5 [3]. exec_time: 57935756, run_delay: 143813522, pcoumt: 1306
Task id: 4 [4]. exec_time: 55365805, run_delay: 169327121, pcoumt: 1304
Task id: 7. exec_time: 62094392, run_delay: 154145028, pcoumt: 240
Task id: 6 [5]. exec_time: 65327153, run_delay: 140495602, pcoumt: 1125
Task id: 9 [7]. exec_time: 59331594, run_delay: 150235964, pcoumt: 1050
Task id: 10 [8]. exec_time: 56522081, run_delay: 143844087, pcoumt: 1082
Task id: 21 [9]. exec_time: 59974508, run_delay: 154457078, pcoumt: 1112
Task id: 12 [10]. exec_time: 76256589, run_delay: 190488849, pcoumt: 1456
Task id: 14 [12]. exec_time: 59948100, run_delay: 137629756, pcoumt: 1005
Task id: 15 [13]. exec_time: 95256554, run_delay: 276468464, pcoumt: 1793
Task id: 16 [14]. exec_time: 108940847, run_delay: 381706768, pcoumt: 2573
Task id: 17 [15]. exec_time: 114393894, run_delay: 395057797, pcoumt: 2346
Task id: 18 [16]. exec_time: 121017072, run_delay: 275139879, pcoumt: 2344
Task id: 19 [17]. exec_time: 112157995, run_delay: 367792691, pcoumt: 2198
Task id: 21 [19]. exec_time: 108452246, run_delay: 383281790, pcoumt: 3389
Task id: 22 [20]. exec_time: 136686284, run_delay: 796141459, pcoumt: 3439
Task id: 23 [21]. exec_time: 117944183, run_delay: 345867754, pcoumt: 2237
Task id: 24 [22]. exec_time: 105566342, run_delay: 328764022, pcoumt: 2696
Task id: 25 [23]. exec_time: 137949009, run_delay: 267696540, pcoumt: 1010
Task id: 26 [24]. exec_time: 138057624, run_delay: 571574901, pcoumt: 1348
Task id: 28 [26]. exec_time: 140315018, run_delay: 585279673, pcoumt: 1308
Task id: 29 [27]. exec_time: 129874939, run_delay: 572622279, pcoumt: 1635
Task id: 30 [28]. exec_time: 1394944077, run_delay: 5725041629, pcoumt: 1379
Task id: 31 [29]. exec_time: 140450731, run_delay: 584274344, pcoumt: 1499
Task id: 32 [30]. exec_time: 138097076, run_delay: 571096175, pcoumt: 1402
Task id: 33 [31]. exec_time: 140450714, run_delay: 586857767, pcoumt: 1636
Task id: 34 [32]. exec_time: 138069489, run_delay: 571663880, pcoumt: 1483
Task id: 35 [33]. exec_time: 140498404, run_delay: 5756091207, pcoumt: 2121
Task id: 36 [34]. exec_time: 140391259, run_delay: 572504107, pcoumt: 2097
Task id: 37 [35]. exec_time: 140450630, run_delay: 576057342, pcoumt: 2716
Task id: 38 [36]. exec_time: 140507391, run_delay: 575646052, pcoumt: 2405
Task id: 39 [37]. exec_time: 140456647, run_delay: 576824589, pcoumt: 4164
Task id: 40 [38]. exec_time: 140459566, run_delay: 575585234, pcoumt: 544
Task id: 41 [39]. exec_time: 1404572836, run_delay: 575114596, pcoumt: 4931
Task id: 42 [40]. exec_time: 140406979, run_delay: 570673068, pcoumt: 4528
Task id: 43 [41]. exec_time: 140449589, run_delay: 572769774, pcoumt: 4753
Task id: 44 [42]. exec_time: 140474638, run_delay: 573697528, pcoumt: 5188
Task id: 45 [43]. exec_time: 140451372, run_delay: 575565238, pcoumt: 2788
Task id: 46 [44]. exec_time: 140481389, run_delay: 575405597, pcoumt: 2792
Task id: 47 [45]. exec_time: 140494528, run_delay: 574965522, pcoumt: 2762
Task id: 48 [46]. exec_time: 140492313, run_delay: 575811251, pcoumt: 2800
Task id: 49 [47]. exec_time: 140425112, run_delay: 574823975, pcoumt: 2922
Task id: 50 [48]. exec_time: 140462828, run_delay: 575888869, pcoumt: 2742
Task id: 51 [49]. exec_time: 1405589774, run_delay: 575583304, pcoumt: 2593
Task id: 52 [50]. exec_time: 140492149, run_delay: 575396314, pcoumt: 2729
Task id: 53 [51]. exec_time: 140451350, run_delay: 575490793, pcoumt: 2728
Task id: 54 [52]. exec_time: 140510538, run_delay: 574500060, pcoumt: 2788
Total exec_time: 59999944001
Cgroup = /10. exec_time: 5999994001
------- group runtime
Cgroup = /10. exec_time: 5999994001
------- sched stats
version 1
maxtasks 409673706
cpu 0: 0.00000
average imbalance: 0.00000
chord[cluster02]:/kernel/gsthub/linux/tools/linsched/tests/results/unprocessor_results: cat sim-1
```

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
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 ?!
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

Acknowledgements

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