# Load balancing via scalable task stealing

# LPC 2018

Steve Sistare
Senior Software Architect

Oracle Corporation

# Current CFS load balancing

- Task wakes:
  - Push task to an idle core or CPU
  - Search in LLC, limited by avg idle and cost
  - See wake_up_process, select_task_rq_fair, select_idle_cpu

- CPU goes idle:
  - Pull a task from other CPU
  - Search all domains, limited by avg idle and cost
  - Costs 10's - 100's usec.
  - See pick_next_task_fair, idle_balance
- Periodically:
  - Rebalance load across all CPUs
  - See rebalance_domains

# CFS Stealing

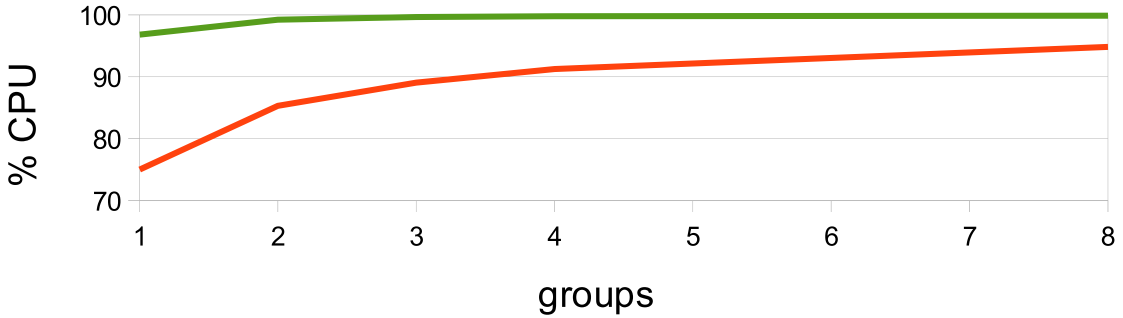
- CPU goes idle:
  - Search CPUs in same LLC. Not limited.
  - Find first with nr_running > 1 (aka overloaded)
  - Steal 1 task
  - Costs 1 - 2 usec

- Maintain bitmap of overloaded CPUs to speed search.
  - Set bit when nr_running exceeds 1
  - Clr bit when nr_running shrinks to 1
  - Bitmap is sparse, struct sparsemask
    - 8 significant bits per 64 bytes; others ignored.
    - API similar to cpumask
    - Reduces cache contention when threads concurrently set, clear, visit elements.

patch v3: https://lkml.org/lkml/2018/11/9/1173

# Performance Results

X6-2: 1 socket * 10 cores * 2 hyperthreads = 20 CPUs
Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz
hackbench <grps> process 100000
sched_wakeup_granularity_ns=15000000



```
baseline
grps   time   %cpu    slice  sched   idle      wake %find  steal
1      8.084  75.02    0.10  105476  46291     59183  0.31      0
2     13.892  85.33    0.10  190225  70958    119264  0.45      0
3     19.668  89.04    0.10  263896  87047    176850  0.49      0
4     25.279  91.28    0.10  322171  94691    227474  0.51      0
8     47.832  94.86    0.09  630636 144141    486322  0.56      0
```

```
new
grps   time   %cpu    slice    sched    idle      wake %find   steal  %speedup
1      5.938  96.80    0.24    31255    7190     24061  0.63    7433  36.1
2     11.491  99.23    0.16    74097    4578     69512  0.84   19463  20.9
3     16.987  99.66    0.15   115824    1985    113826  0.77   24707  15.8
4     22.504  99.80    0.14   167188    2385    164786  0.75   29353  12.3
8     44.441  99.86    0.11   389153    1616    387401  0.67   38190   7.6
```

# NUMA issue and limitation

- Stealing causes regression for hackbench on larger NUMA systems.
  - More cross-node migrations → higher CPU time.
- Root cause:

```
wake_affine_idle()
   if (sync && cpu_rq(this_cpu)->nr_running == 1)
     return this_cpu;       // move the task
```

- Stealing smooths the load. nr_running is 1 more often.

- Stealing is disabled for nodes > 2.
- Specific to hackbench?
  - Sender/receiver co-location trumps all.
  - No affinity to data (other than the socket)
- To override:

```
vmlinuz ... sched_steal_node_limit=<n>
```

# Details of hackbench NUMA performance

X5-8: 8 sockets * 18 cores * 2 hyperthreads = 288 CPUs
Intel(R) Xeon(R) CPU E7-8895 v3 @ 2.60GHz
Average of 10 runs of: hackbench <groups> processes 50000

```
          --- base --     --- new ---
groups     time %stdev     time %stdev  %speedup
     1    3.627   15.8    3.876    7.3      -6.5
     2    4.545   24.7    5.583   16.7     -18.6
     3    5.716   25.0    7.367   14.2     -22.5
     4    6.901   32.9    7.718   14.5     -10.6
     8    8.604   38.5    9.111   16.0      -5.6
    16    7.734    6.8   11.007    8.2     -29.8
```

Total CPU time increases (data not shown).

CPU time increases uniformly across all functions.

Due to NUMA migrations?  Let's look.

# Details of hackbench NUMA performance

```
base                                              --- domain2 ---    --- domain3 ---
grp time %cpu  sched   idle   wake steal remote  move pull remote  move pull
  1 20.3 10.3  28710  14346  14366     0    490  3378    0   4039     0    0
  2 26.4 18.8  56721  28258  28469     0    792  7026   12   9229     0    7
  3 29.9 28.3  90191  44933  45272     0   5380  7204   19  16481     0    3
  4 30.2 35.8 121324  60409  60933     0   7012  9372   27  21438     0    5
  8 27.7 64.2 229174 111917 117272     0  11991  1837  168  44006     0   32
 16 32.6 74.0 334615 146784 188043     0   3404  1468   49  61405     0    8
```

```
new                                             --- domain2 ---    --- domain3 ---
grp time %cpu  sched   idle   wake steal remote  move pull remote  move pull
  1 20.6 10.2  28490  14232  14261    18      3  3525    0   4254     0    0
  2 27.9 18.8  56757  28203  28562   303   1675  7839    5   9690     0    2
  3 35.3 27.7  87337  43274  44085   698    741 12785   14  15689     0    3
  4 36.8 36.0 118630  58437  60216  1579   2973 14101   28  18732     0    7
  8 48.1 73.8 289374 133681 155600 18646  35340 10179  171  65889     0   34
 16 41.4 82.5 268925  91908 177172 47498  17206  6940  176  71776     0   20
```

Moves are significantly higher for steal.

Correlates with longer run times.

# Future Work

- RT task stealing
- Remove the core and socket levels from idle_balance()
- Cross-node stealing.  Replace idle_balance().
- Stealing misfit tasks (see discussion with Valentin Schneider)
- Consider NUMA node load in wake_affine().
  - Eg, weight(cfs_overload_cpus)
- Sparsemask for idle cores, idle cpus