Idle Scheduling in Linux

Nikhil Rao
Google, Inc.

Outline

- Workloads and CPU QoS goals
- Lessons learnt
- Limitations in balancing low weight task groups
- Solutions
- Future work

Types of Workloads

Latency sensitive applications

- High priority workload
- Strict latency, throughput guarantees
- Typical request-response tasks
- User-facing services like search, maps, etc.

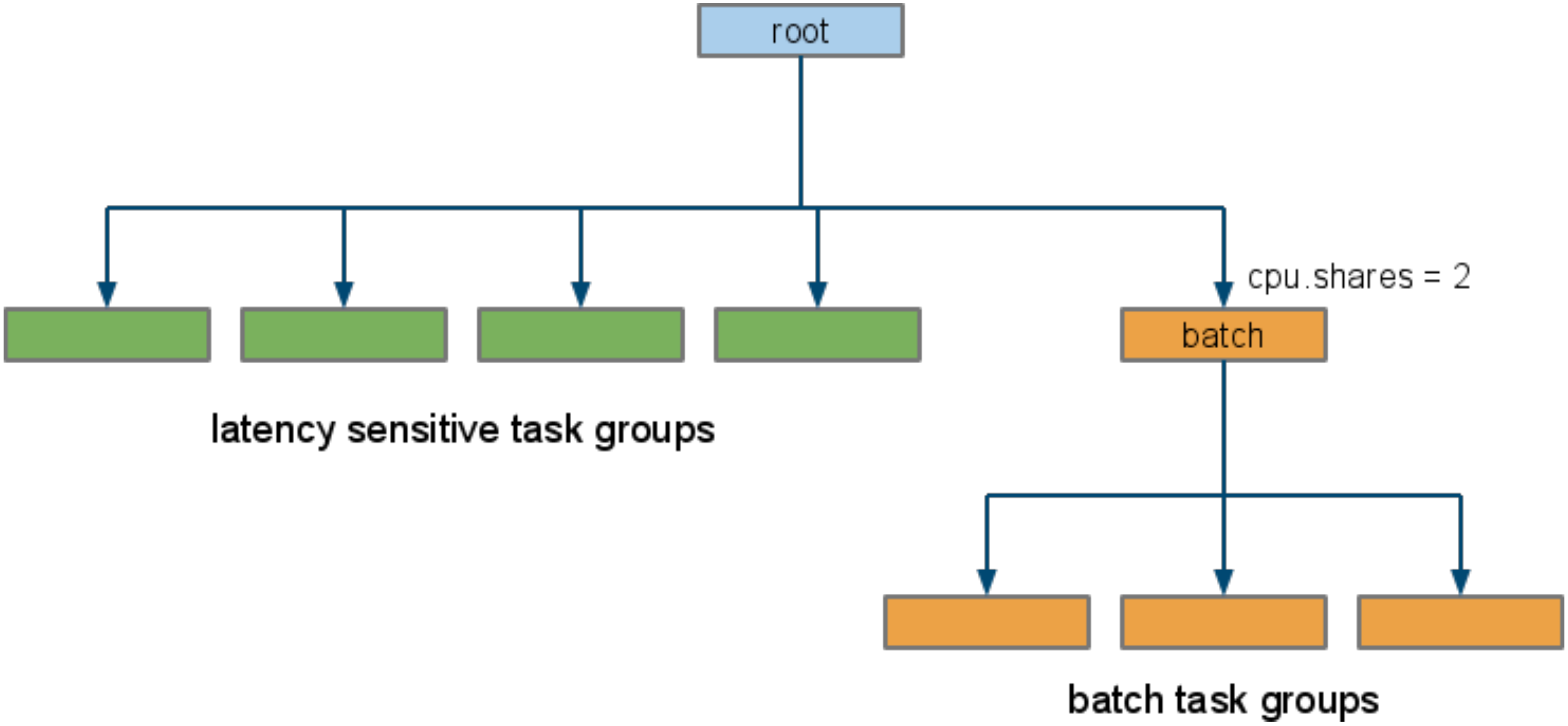
Batch applications

- Low priority workload
- Soak up idle resources on the machine
- Usually cpu soakers with little or no I/O
- Long running batch jobs like video transcoding, etc.

QoS Requirements

- Isolation between latency sensitive applications
- Guaranteed latency response, fairness for latency sensitive applications
- Strict priority for latency sensitive over batch
- Maximize utilization if there is demand for cpu

Task group structure



Lessons Learnt

Wins!

- Group scheduling works well!
  - Good isolation, fairness between latency sensitive applications
- Shares is simple abstraction for application developers
- Good isolation between latency sensitive and batch tasks
  - Could be improved, future work

Pain Point:

- Starvation and degraded end-to-end latency reported for batch tasks!
  - Large weight differentials lead to sub-optimal utilization
  - Insufficient resolution for balancing low weight groups

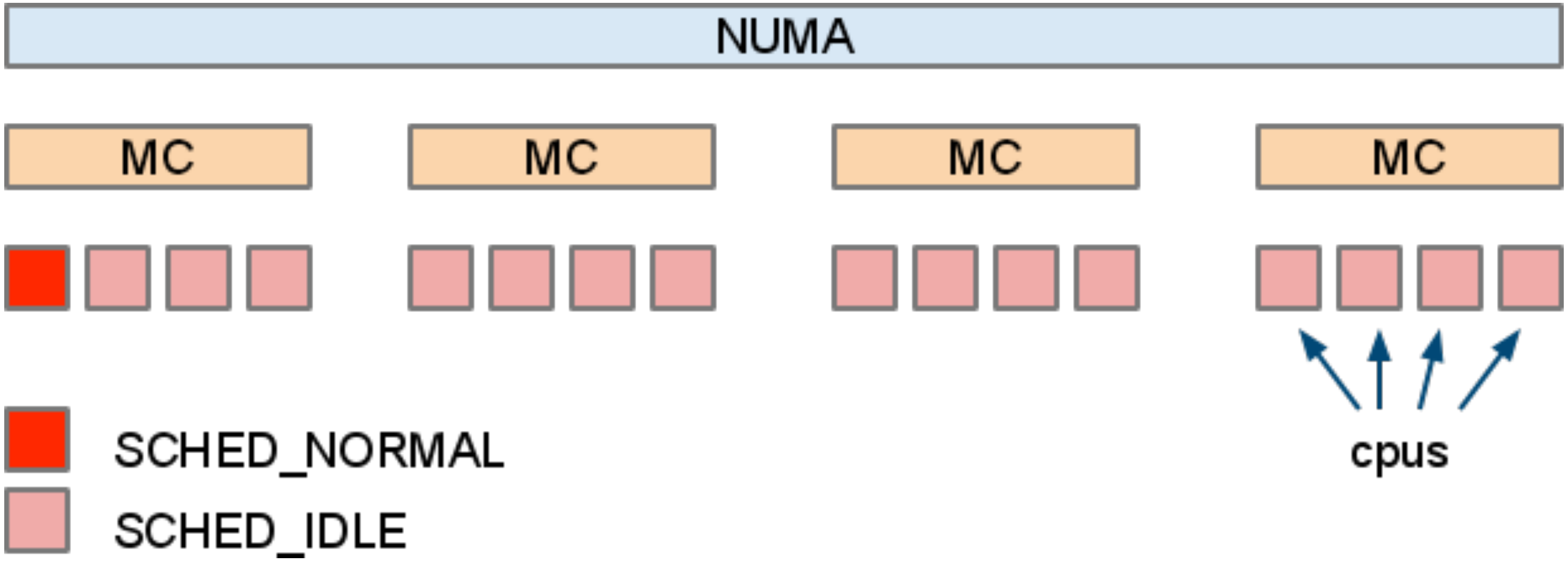
#1: Sub-optimal Utilization Example

Test Setup

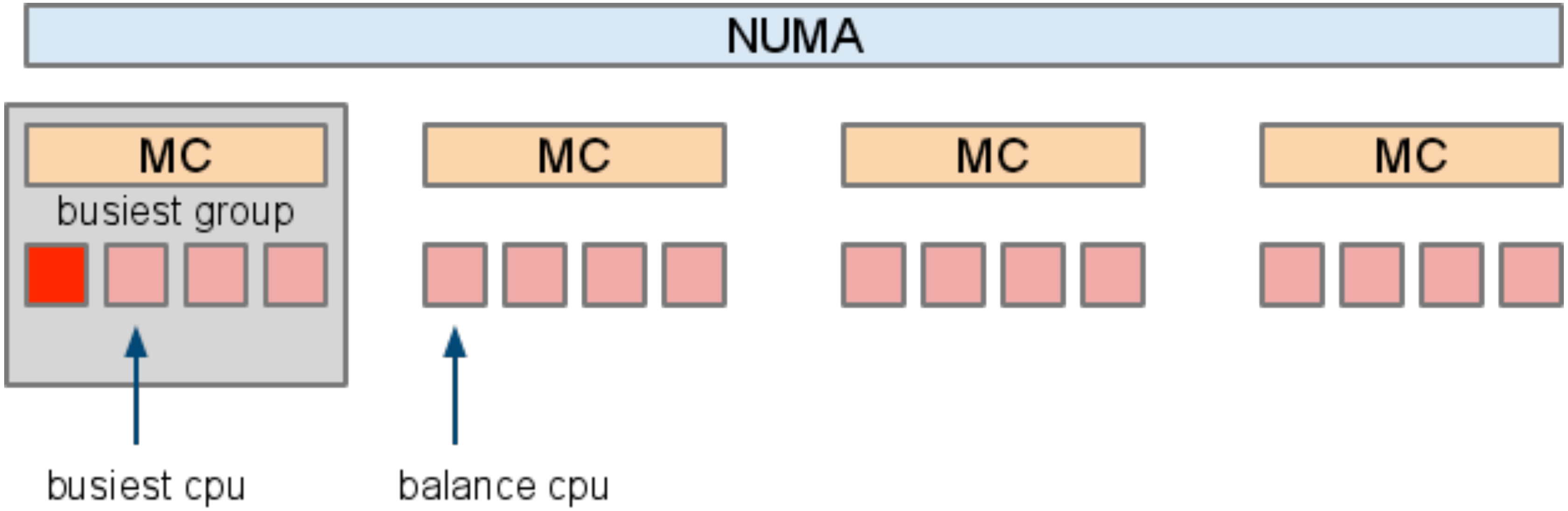
- 16 cpu test machine (quad-socket, quad-core)
- 15 SCHED_IDLE soaker tasks (load wt = 3)
- 1 SCHED_NORMAL soaker task (nice 0, load wt = 1024)
- 2.6.36 kernel

Result

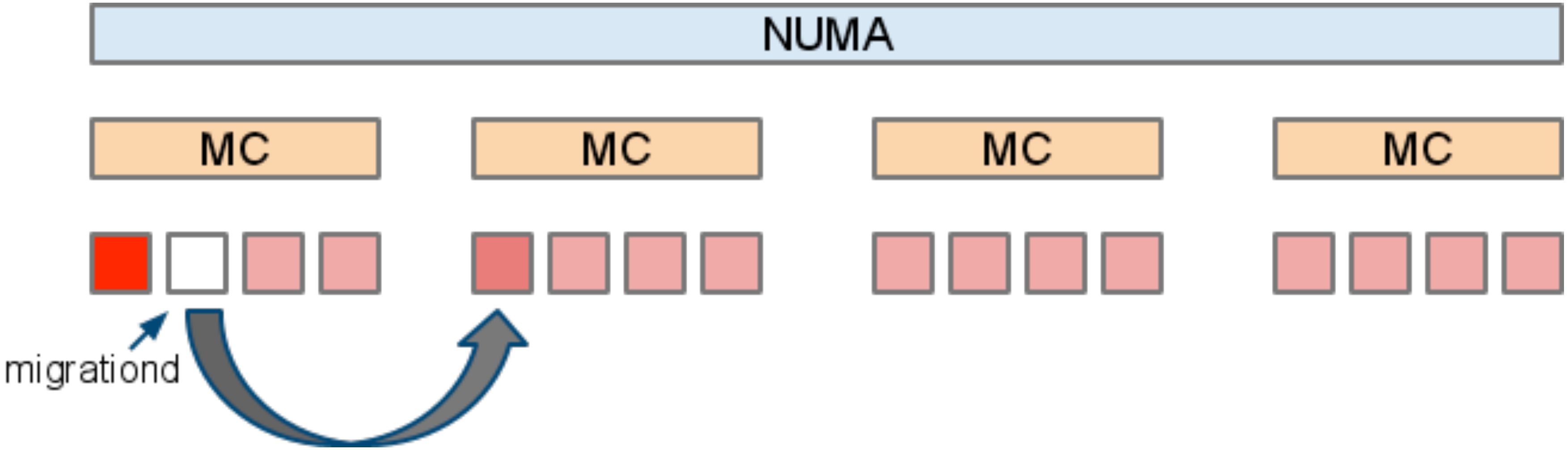
```
04:58:46 PM  CPU   %user   %nice    %sys %iowait    %irq   %soft  %steal   %idle    intr/s
04:58:47 PM  all   81.47    0.00    0.25    0.00    0.00    0.00    0.00   18.28  13796.00
04:58:48 PM  all   81.20    0.00    0.25    0.00    0.00    0.00    0.00   18.55  13816.00
04:58:49 PM  all   80.93    0.19    0.25    0.00    0.00    0.06    0.00   18.57  13965.00
04:58:50 PM  all   81.40    0.00    0.25    0.00    0.00    0.00    0.00   18.35  13837.37
04:58:51 PM  all   81.19    0.00    0.31    0.00    0.00    0.00    0.00   18.50  13592.08
04:58:52 PM  all   81.25    0.00    0.25    0.00    0.00    0.00    0.00   18.50  13721.00
04:58:53 PM  all   81.19    0.00    0.25    0.00    0.00    0.00    0.00   18.56  13764.00
04:58:54 PM  all   81.25    0.00    0.25    0.00    0.00    0.00    0.00   18.50  13841.41
04:58:55 PM  all   80.30    0.00    1.19    0.00    0.00    0.00    0.00   18.51  14989.11
04:58:56 PM  all   80.77    0.00    0.50    0.00    0.00    0.00    0.00   18.73  13964.65
Average:     all   81.09    0.02    0.37    0.00    0.00    0.01    0.00   18.51  13929.53
```



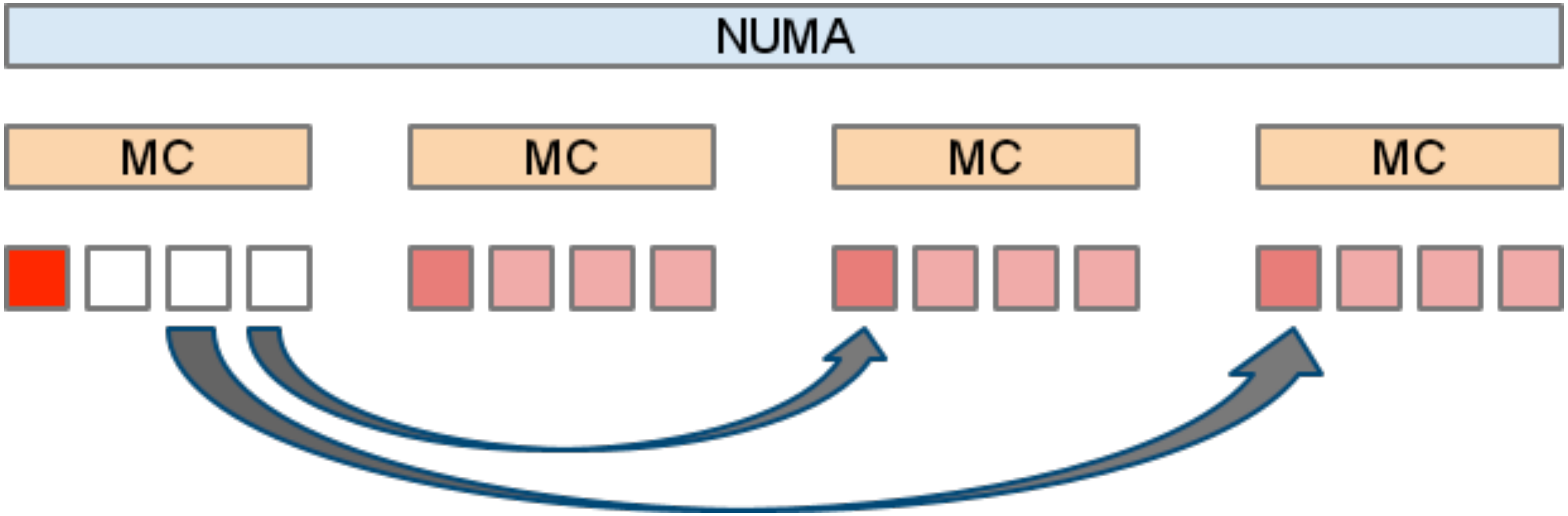
- Two scheduling domain levels, MC and NUMA
- cpu0 has the SCHED_NORMAL task
- SCHED_IDLE tasks distributed equally on cpus 1-15



- Consider balancing decisions at the NUMA domain
- Sched group weights = { 1033, 12, 12, 12 }
- f_b_g() picks group0 as busiest group
- f_b_q() picks cpu1, 2 or 3 as busiest queue
  - Never picks cpu0 because weight > imbalance

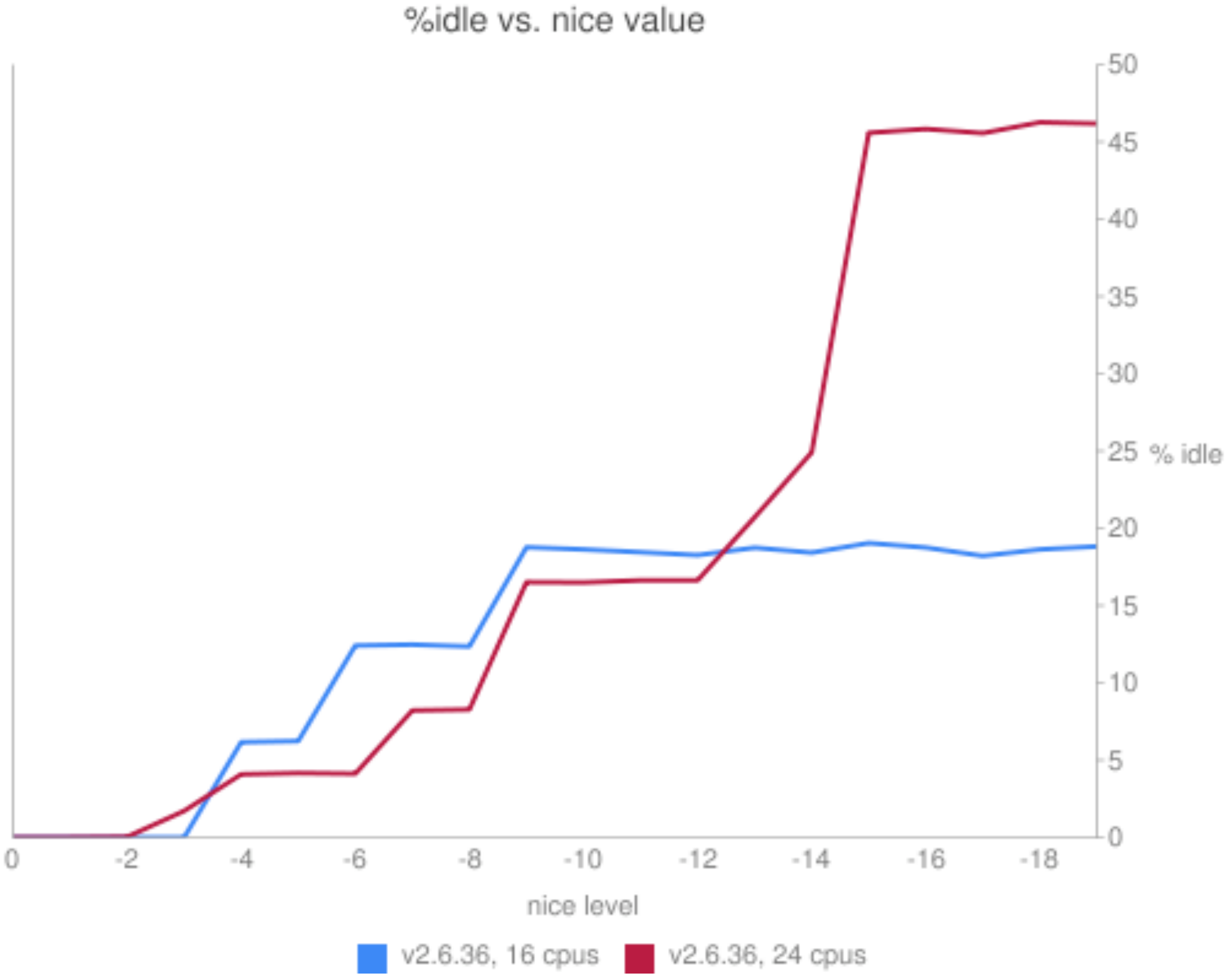


- Load balancer pulls tasks until one task per runqueue
- Balancing operations fail (unable to pull running task)
- Active migration kicks in after 5 failures
- Pushes SCHED_IDLE task from the sched group



- Active migrations kick off all SCHED_IDLE tasks
- Idle cpus unable to pull load back
    - load balancer does not find any busy group

#1: Experiment with niced task



Increasing task priority (via nice) leads to sub-optimal utilization!

#1: Load Balancer Fixes

- Introduced notion of extra group capacity
    - extra capacity => nr_running < group_capacity

- Set group_imb only if max_nr_running > 1

- Group capacity fixes when SD_PREFER_SIBLING enabled on child domain

- Force balancing if local group has extra capacity

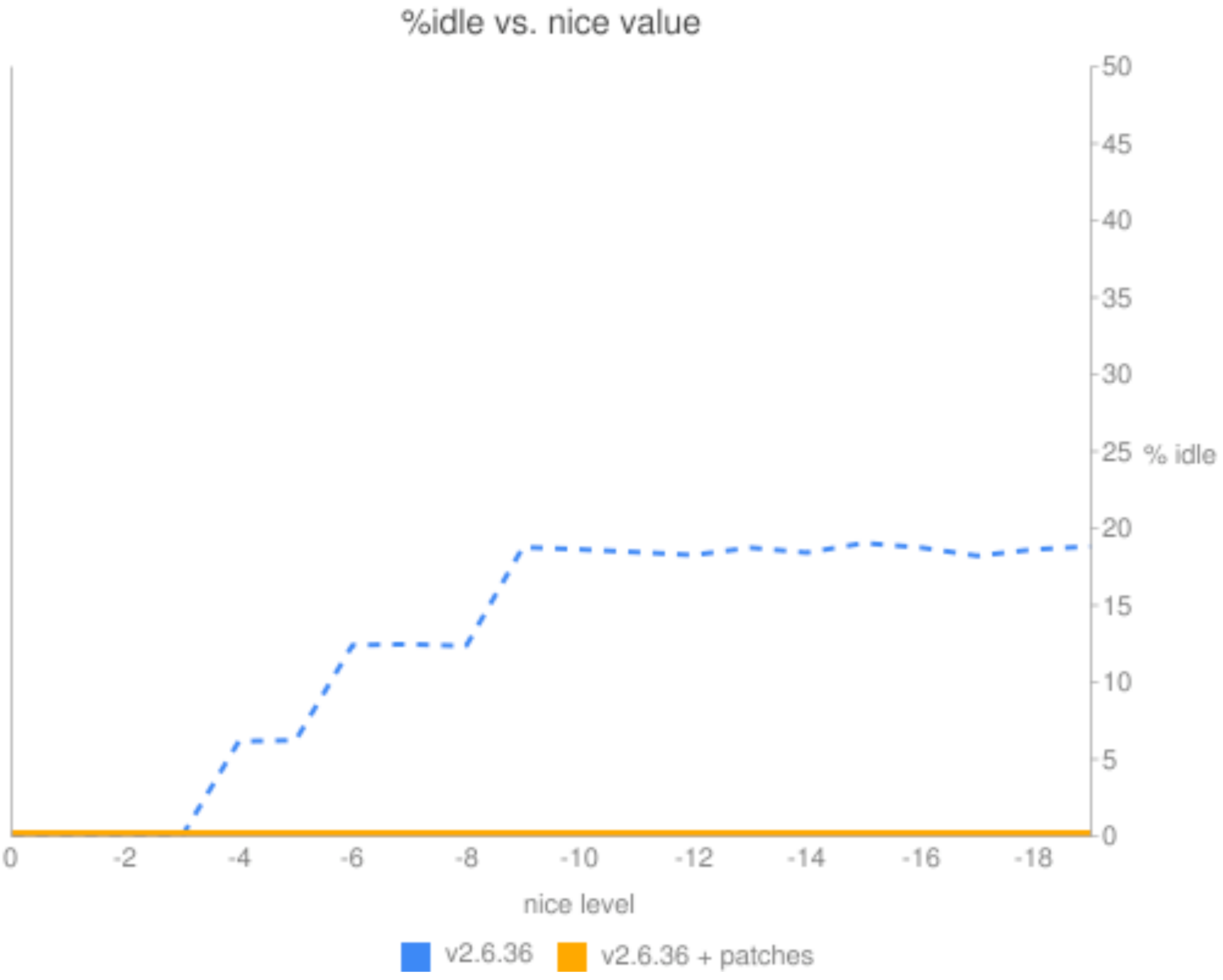
#1: Results after fixes

- v2.6.36 + patches
- 15 SCHED_IDLE tasks, 1 SCHED_NORMAL task
- Improved utilization!

```
12:58:29 PM  CPU   %user   %nice    %sys %iowait    %irq   %soft  %steal   %idle    intr/s
12:58:30 PM  all   99.81    0.00    0.19    0.00    0.00    0.00    0.00    0.00  16384.00
12:58:31 PM  all   99.75    0.00    0.25    0.00    0.00    0.00    0.00    0.00  16428.00
12:58:32 PM  all   99.81    0.00    0.19    0.00    0.00    0.00    0.00    0.00  16345.00
12:58:33 PM  all   99.75    0.00    0.25    0.00    0.00    0.00    0.00    0.00  16383.00
12:58:34 PM  all   99.75    0.00    0.19    0.00    0.00    0.06    0.00    0.00  16333.00
12:58:35 PM  all   99.81    0.00    0.19    0.00    0.00    0.00    0.00    0.00  16359.00
12:58:36 PM  all   99.75    0.00    0.25    0.00    0.00    0.00    0.00    0.00  16523.23
12:58:37 PM  all   99.75    0.00    0.25    0.00    0.00    0.00    0.00    0.00  16352.00
12:58:38 PM  all   98.75    0.00    1.25    0.00    0.00    0.00    0.00    0.00  17128.00
12:58:39 PM  all   99.31    0.06    0.62    0.00    0.00    0.00    0.00    0.00  16757.00
Average:     all   99.63    0.01    0.36    0.00    0.00    0.01    0.00    0.00  16499.20
```

#1: Results after fixes

- Experiment with niced task, 16 cpu machine



#2: Insufficient granularity

- Load balancing math breaks down with low weight groups

- Consider hierarchal load factor calculations in tg_load_down() for static batch task group (shares = 2)
    - *h_load = parent->h_load * shares / parent->cfs_rq->wt + 1*
    - h_load = 1 for /batch task group
    - h_load = 0 for any task group under /batch

- Setting h_load = 0 leads to a couple of issues:
    - Herd migrations
    - Loss of fairness between batch tasks

#2: Herd Migrations

- *Herd migrations:* mass migration of batch tasks from the busiest cpu to the balancing cpu to satisfy large imbalance

- Insufficient granularity of low weight task groups result in h_load being estimated as 0

- Small imbalance is greatly exaggerated
  - For example, imbalance of 10 with 5 tasks on busiest cpu results translates to rem_move of ~51K
  - Enough to migrate all tasks except running task

- Incorrect accounting after migration
  - moved_load = moved_load * h_load / (weight + 1) = 0!
  - Failed migration!

#2: Example of Herd Migration

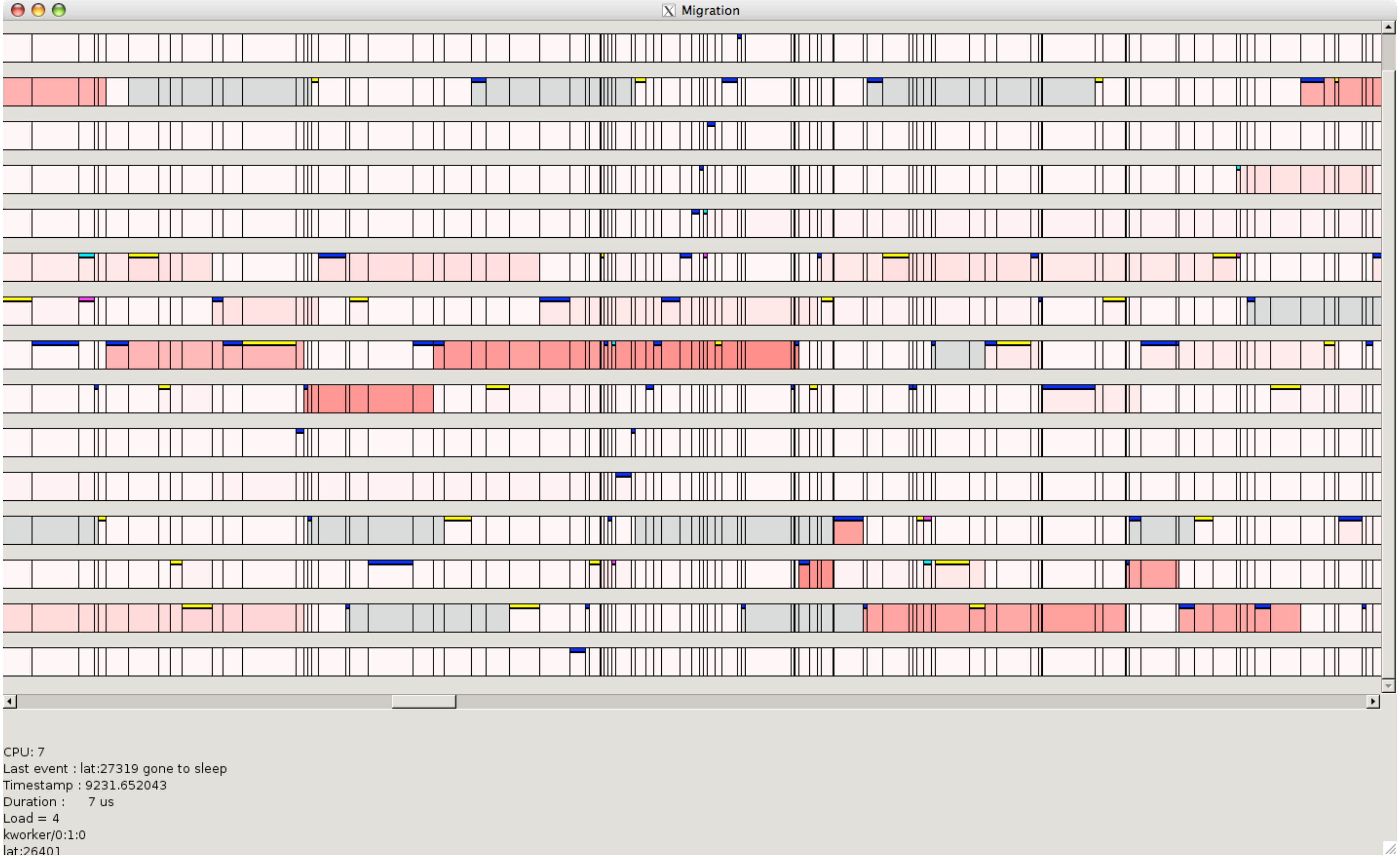
Test Setup

- 16 cpu test machine (quad-socket, quad-core)
- Create a batch task group under /batch with shares = 1024
- Spawn 48 tasks with random sleep/busy pattern (100ms)

Result

```
03:04:24 PM  CPU   %user   %nice    %sys %iowait    %irq   %soft  %steal %idle    intr/s
03:04:25 PM  all   91.72    0.00    0.31    0.00    0.00    0.00    0.00  7.97  15607.00
03:04:26 PM  all   93.50    0.00    0.37    0.00    0.00    0.00    0.00  6.12  15749.50
03:04:27 PM  all   94.62    0.00    0.31    0.00    0.00    0.00    0.00  5.06  16045.00
03:04:28 PM  all   94.69    0.00    0.31    0.00    0.00    0.00    0.00  4.99  16311.11
03:04:29 PM  all   93.95    0.00    0.37    0.00    0.00    0.00    0.00  5.68  16037.00
03:04:30 PM  all   94.07    0.00    0.37    0.00    0.00    0.00    0.00  5.56  15843.56
03:04:31 PM  all   94.93    0.00    0.31    0.00    0.00    0.00    0.00  4.75  16081.00
03:04:32 PM  all   95.19    0.00    0.38    0.00    0.00    0.00    0.00  4.44  16157.00
03:04:33 PM  all   95.75    0.00    0.37    0.00    0.00    0.00    0.00  3.87  16030.69
03:04:34 PM  all   95.69    0.00    0.31    0.00    0.00    0.00    0.00  4.00  16184.00
Average:     all   94.41    0.00    0.34    0.00    0.00    0.00    0.00  5.25  16003.89
```

#2: Example of Herd Migration



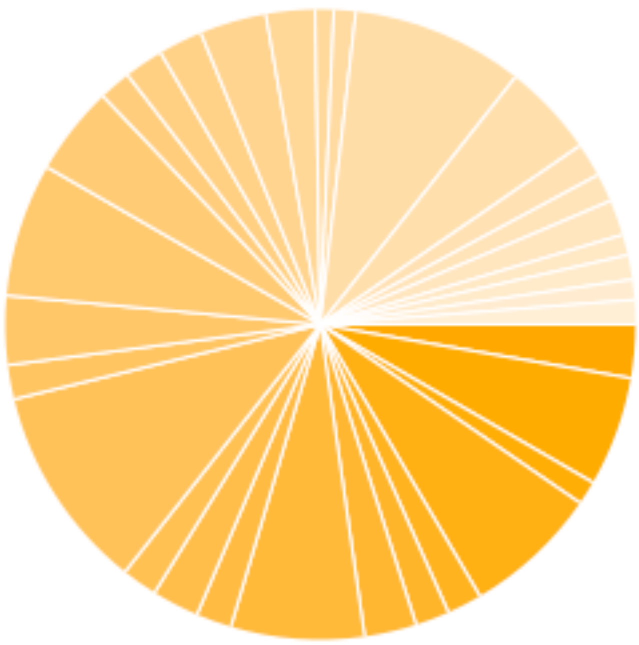
```
CPU: 7
Last event : lat:27319 gone to sleep
Timestamp : 9231.652043
Duration :      7 us
Load = 4
kworker/0:1:0
lat:26401
```

#2: Lack of Fairness (batch tasks)

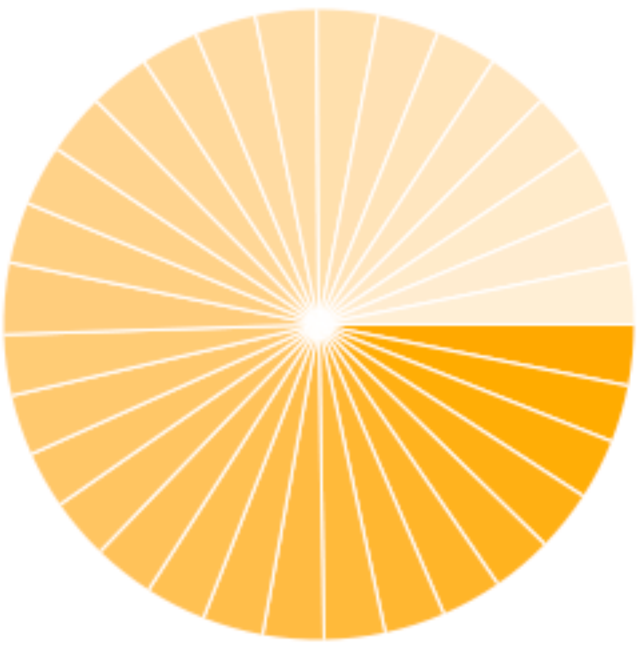
Test Setup:

- 16 cpu machine (quad-core, quad-socket)
- 48 task groups with one soaker each
- Compare cpuacct.usage for each task group

Result:



Distribution of runtime between batch tasks



Distribution of runtime between latency sensitive tasks

#2: Ideas to improve granularity

- Scale up shares by a constant
    - Update MIN_SHARES, MAX_SHARES
    - Update nice to weight ratios
    - Loss of accuracy in update_curr()
        - Scale down weights before calling update_curr()
        - Can we do 128-bit math?

- Scale load weights before balancing operations
    - Scale h_load by a factor of 1024 in load_balance_fair()

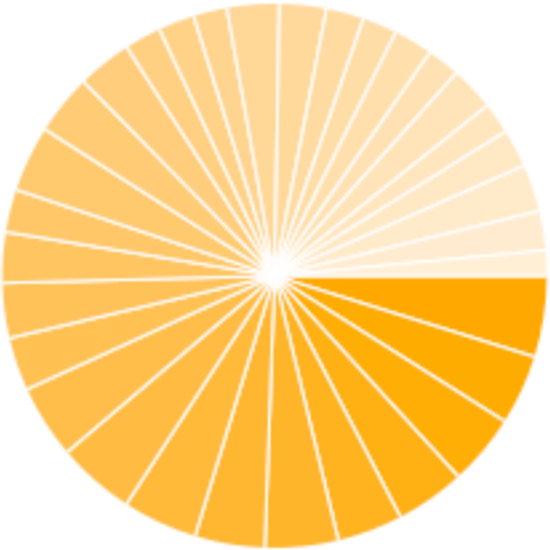
#2: Results of scaling up shares

- Reduces herd migrations, improves utilization!

```
02:43:04 PM  CPU   %user   %nice    %sys %iowait    %irq   %soft  %steal %idle    intr/s
02:43:05 PM  all   99.56    0.00    0.31    0.00    0.00    0.00    0.00  0.12  16772.00
02:43:06 PM  all   98.94    0.00    1.00    0.00    0.00    0.00    0.00  0.06  17031.00
02:43:07 PM  all   98.94    0.00    0.75    0.00    0.00    0.06    0.00  0.25  17002.97
02:43:08 PM  all   98.81    0.00    1.00    0.00    0.00    0.00    0.00  0.19  16930.30
02:43:09 PM  all   98.75    0.00    1.00    0.00    0.00    0.00    0.00  0.25  16792.00
02:43:10 PM  all   99.56    0.00    0.31    0.00    0.00    0.00    0.00  0.12  16785.00
02:43:11 PM  all   99.44    0.00    0.37    0.00    0.00    0.00    0.00  0.19  16923.00
02:43:12 PM  all   98.69    0.00    0.37    0.00    0.00    0.06    0.00  0.87  16806.00
02:43:13 PM  all   99.25    0.00    0.37    0.00    0.00    0.00    0.00  0.37  16760.00
02:43:14 PM  all   99.50    0.00    0.37    0.00    0.00    0.00    0.00  0.12  16515.84
Average:     all   99.14    0.00    0.59    0.00    0.00    0.01    0.00  0.26  16831.57
```

- Improved fairness

fairness distribution of batch tasks with scaled shares



Future Work

- Avoid batch task preempting latency sensitive task
  - Extend "SCHED_IDLE" concept to group entities
  - Change preemption model for group entities
  - Tasks in to idle groups do not preempt non-idle tasks

- Support more complex task group hierarchies
  - Arbitrary nesting of task groups
  - Guarantee fairness to tasks in this hierarchy
  - Maximize overall system utilization

- Further improve fairness between batch tasks

Thank you!

Nikhil Rao
ncrao@google.com