Understanding the impact of Scheduler on your application

Atish Patra
Dhaval Giani
Linux Kernel Developer, Oracle
"You don't need to know how to operate an X-ray machine, but you do need to know that if you swallow a penny, an X-ray is an option."

- Brendan Gregg's friend
Program Agenda

1. Introduction to Scheduler
2. Load Balancing
3. Application Analysis
4. Scheduler Tunables
5. Conclusion
Introduction
Linux Scheduler

• Who gets to run when (and where)
Completely Fair Scheduler

• Two tasks
  • T1 runs
  • Followed by T2
  • Followed by T1
  • ...
How long?

• T1 has nice -1 and T2 has nice 0
  • T1 gets to run twice as long as T2!

• nice
• weight
• vruntime

\[ \text{vruntime} \propto \frac{1}{\text{weight}} \]
Group Scheduling

• Consider Apache spawns 1000 threads for every Oracle thread.
• Does that mean Apache gets to run 1000 times as often?
• Enter Group Scheduling
  • Apache Group has all the Apache threads
  • Oracle Group has all the Oracle threads
  • Oracle and Apache groups have their own nice values
Symmetric Multiprocessing

CPU 0

CPU 1

Last Level Cache (LLC)

Last Level Cache (LLC)

CPU 2

CPU 3
Load Balancing

• Consider
  • 4 CPUs (quad core laptops!)
  • 4 tasks (let’s consider same nice value)
  • Each CPU gets 1
• Now consider
  • 4 CPUs
  • 5 tasks (1 has nice value 1)
  • Now what?
Load Balancing with group scheduling

• Consider Apache with 1000 threads
• Consider Oracle with 4 threads
• Apache and Oracle have same weight
• 4 CPUs. Apache gets 2, Oracle gets 2
• Now what if we had a third group with 3 threads?
Non Uniform Memory Access
Load Balancing

• Problems:
  – Who runs where?
  – Who moves where?
  – How much movement?
  • Too much: tasks play ping pong!
  • Too less: cores go idle!
Scheduling domains/groups
Why scheduling domains?

• Issues
  – System wide task movement and search cost
  – Hardware properties

– Solution
  – A hierarchal search to reduce the cost
Scheduling Domains - Example
Scheduling Domains - SMT Domain

Last Level Cache

Last Level Cache

Last Level Cache

Last Level Cache

0 1

2 3

8 9

10 11

4 5

6 7

12 13

14 15
Scheduling Domains - MC Domain

Last Level Cache

Last Level Cache

Last Level Cache

Last Level Cache
Scheduling Domains - Die/CPU Domain

Last Level Cache

Last Level Cache

Last Level Cache

Last Level Cache
Scheduling Domains - NUMA Domain

Last Level Cache

Last Level Cache

Last Level Cache

Last Level Cache
Load Balancing (Pull)

• Periodic balancing
  – Happens at every scheduling tick
  – Trace domain to find imbalance
  – One CPU performs the search
  – Frequency reduces as we traverse up the hierarchy
Last Level Cache (LLC)

0 1
2 3
4 5
6 7

Last Level Cache (LLC)

8 9
10 11
12 13
14 15
Last Level Cache (LLC)
• New Idle balancing
  – average idle time > migration_cost
  – average idle time > domain search cost
  – Needs SD_BALANCE_NEWIDLE enabled
• New Idle balancing
  – average idle time > migration_cost
  – average idle time > domain search cost
  – Needs SD_BALANCE_NEWIDLE enabled
• Wakeup load balancing
  – cache affinity
  – waker-wakee relationship
  – search in the same LLC domain
• Wakeup load balancing
  – cache affinity
  – waker-wakee relationship
  – search in the same LLC domain
• Wakeup load balancing
  – cache affinity
  – waker-wakee relationship
  – search in the same LLC domain
• Wakeup load balancing
  – cache affinity
  – waker-wakee relationship
  – search in the same LLC domain
• Fork/Exec load balancing
  – Cache affinity might not matter
  – Find the least loaded Group
  – Find the least loaded CPU
  – Needs SD_BALANCE_FORK/EXEC enabled
• Fork/Exec load balancing
  – Cache affinity might not matter
  – Find the least loaded Group
  – Find the least loaded CPU
  – Needs SD_BALANCE_FORK/EXEC enabled
Application Analysis
Application Analysis - Tools (Our experience)

• Manual instrumentation
  – printk/pr_* : **NO**!
  – trace_printk(): Maybe!

• Issues
  – Kernel modification
  – Parsing difficulty
Application Analysis - perf

• Perf
  – call graphs
  
  ```
  # sudo perf record -a -g -e cycles perf bench sched \ 
  messaging -g 1000 -l 10000
  # sudo perf report --no-children -g \ 
  -s symbol > perf_callg.out
  ```

• Difficult to process in this format
Application Analysis - perf

• Flamegraph is immensely useful

$ sudo perf script |
./stackcollapse-perf.pl > out.perf-folded

$./flamegraph.pl out.perf-folded > perf-kernel.svg
Application Analysis - perf

• Use perf stat for migrations and cache-misses

```
$ sudo perf stat -a -e migrations -e cache-misses \  
   -e cache-references perf bench sched messaging \  
   -g 100 -l 1000

Performance counter stats for 'system wide':

7,847,249 migrations
20,950,769,373 cache-misses # 32.308 % of all cache refs (100.00%)
64,847,047,418 cache-references
```
Application Analysis - Ftrace

• trace-cmd or manual
• Able to trace each task wakeup/migration events

#trace-cmd record -e sched:sched_wakeup_new -e sched:sched_migrate_task perf bench sched messaging -g 100 -l 100

• trace-cmd spawns a process on each processor
• Might be tricky if task migrations/wakeup analysis
Application Analysis - Impact of Scheduling domains/groups

• `/proc/schedstat`
  – scheduling domains/groups follow hardware?
  – load balancing statistics
  – Too many new idle balances especially across NUMA?
• Try to reduce `relax_domain_level`
Application Analysis - mpstat

- `mpstat -P ALL -u 5`

```
11:12:05 AM CPU %usr %nice %sys %iowait %irq %soft %steal %guest %idle
11:12:16 AM all 81.36 0.07 8.49 0.00 0.00 0.67 0.00 0.00 9.40
11:12:16 AM 0 82.07 0.00 9.67 0.00 0.00 1.13 0.00 0.00 7.14
11:12:16 AM 1 81.30 0.00 11.03 0.00 0.00 0.19 0.00 0.00 7.48
11:12:16 AM 2 84.72 0.00 7.36 0.00 0.00 0.00 0.00 0.00 7.92
11:12:16 AM 3 77.87 0.00 8.00 0.00 0.00 7.06 0.00 0.00 7.06
11:12:16 AM 4 83.25 0.00 8.66 0.00 0.00 0.00 0.00 0.00 8.09
11:12:16 AM 5 83.49 0.00 8.44 0.00 0.00 0.00 0.00 0.00 8.07
11:12:16 AM 6 84.07 0.00 7.73 0.00 0.00 0.00 0.00 0.00 8.20
11:12:16 AM 7 83.37 0.00 8.32 0.00 0.00 0.00 0.00 0.00 8.32
11:12:16 AM 8 86.94 0.00 8.65 0.00 0.00 0.19 0.00 0.00 4.23
```
Application Analysis - eBPF

- New kid on the block
- Find cpu idleness due to scheduler

```
# ./cpuunclaimed.py
Sampling run queues... Output every 1 seconds. Hit Ctrl-C to end.
%CPU 12.63%, unclaimed idle 86.36%
%CPU 12.50%, unclaimed idle 87.50%
%CPU 12.63%, unclaimed idle 87.37%
%CPU 12.75%, unclaimed idle 87.25%
%CPU 12.50%, unclaimed idle 87.50%
%CPU 12.63%, unclaimed idle 87.37%
```
Application Analysis - eBPF

• Find run-queue length distribution
• https://github.com/iovisor/bcc/blob/master/tools/runqlen.py
Scheduler Tunables
Scheduler Tunables

- `sysctl -A | grep "sched" | grep -v "domain"

kernel.sched_cfs_bandwidth_slice_us = 5000
kernel.sched_child_runs_first = 0
kernel.sched_latency_ns = 24000000 (24ms)
kernel.sched_migration_cost_ns = 500000 (0.5ms)
kernel.sched_min_granularity_ns = 10000000 (10ms)
kernel.sched_nr_migrate = 32
kernel.sched_schedstats = 0
kernel.sched_time_avg_ms = 1000
kernel.sched_wakeup_granularity_ns = 15000000 (15ms)
Scheduler Tunables

• sched_migration_cost_ns: Is task cache-hot?

  Task bouncing  Cores go idle

• sched_wakeup_granularity_ns: To preempt or not to?

  Reduces scheduling delay  Less wakeup preemption
Scheduler Tunables

- `sched_latency_ns`: Period where each task guaranteed is to run
- `sched_min_granularity_ns`: Minimum guaranteed runtime for a task
- `sched_period = max(sched_latency, nr_tasks*sched_min_granularity)`

Reduces latency

Improves throughput

`sched_min_granularity_ns`
Conclusions

• Understand your application
  • How does it interact with the system?

• Use the tools at your disposal
  • Is it a scheduler issue?

• Experiment with different tunables
  • Tune one at a time

• Might be a kernel bug!
Thank You!

atish.patra@oracle.com
dhaval.giani@oracle.com