Overview of the Current Approaches to Enhance the Linux Scheduler

Preeti U. Murthy
preeti@linux.vnet.ibm.com
IBM Linux Technology Center
Agenda

• Introduction to Linux Scheduling

• Overview of the Power Aware Scheduler

• Usage of Per Entity Load Tracking Metric in Power Aware Scheduler

• Where does Power Aware Scheduler stand today?

• Experimental Evaluation of the Power Aware Scheduler

• Other Efforts towards Power Efficiency in Scheduling

• References
What does the Linux scheduler do?

- The Linux scheduler does two primary jobs:
  - Scheduling multiple runnable jobs fairly on a CPU
  - Scheduling jobs optimally across multiple CPUs in an SMP system
Scheduling multiple runnable jobs fairly on a CPU

**Requirement:** The CPU bandwidth needs to be “fairly” distributed among tasks

- Data Structure used is the Red Black Tree.
- The tasks are sorted out in this tree in the increasing order of CPU bandwidth received and the leftmost task is chosen to run on the CPU.
- A new task or a woken up task is positioned at the left most end of the tree.
- The CPU bandwidth consumed by a task is called the vruntime.
- vruntime and the above data structure ensure that all tasks receive a fair share of CPU bandwidth, which in-turn ensure better interactivity of workloads.
Scheduling jobs optimally across multiple CPUs in an SMP system

Goals of SMP scheduling:

- No CPU must have too many or too few tasks to run.
- A new task/ woken up task must find the right CPU to run on quickly.
Scheduling jobs optimally across multiple CPUs in an SMP system

Work Conserving Nature of scheduler:
If there is CPU bandwidth, schedule the task on it.

Queue of tasks to be scheduled

New Task arrives

CPU2 goes idle

Load Balancing

SMP Scheduling
Implementation schemes for SMP Scheduling

Implementing the goals of SMP Scheduling

- Efficient Load Balancing
- Scalable scheduling

Taking SMP Scheduling one step forward:

Consolidation of tasks to fewer CPUs belonging to fewer power domains for power efficiency through Power Aware Scheduling.
Power Aware Scheduling

• Goal:

Consolidation of load to fewer CPU belonging to fewer power domains, without degrading the power efficiency of the current scheduler.

Power Efficiency = Performance/Power
Scheduling domains and groups define the scheduling entities in a hierarchical fashion.
How does Power Aware Scheduling work?

- During forking of tasks:
  Schedule the new task on -
  $\text{idlest}_\text{cpu}(\text{group}\_\text{leader})$

- During waking of tasks:
  Schedule the woken up task on -
  $\text{leader}_\text{cpu}(\text{group}\_\text{leader})$
  if task\_is\_light\_weight()
Power Aware Scheduler against Race to Idle?

- No, because...

  - Power aware scheduler consolidates load within a power domain when it can afford to, i.e. when power domain is underutilized - idle CPUs exist. *Tasks will not be throttled.*

  - Power aware scheduler defaults to race to idle when it cannot afford to consolidate within a power domain, i.e. when power domain is overloaded.

---

**Group Leader**

**Group Min**

**Task Consolidation**

**Race To Idle**
Measuring the idleness of a CPU

- What determines if a CPU is busy and how much busy?
  If we were to measure the % utilization of a CPU, then

\[
\text{% utilization of a CPU} = \left( \frac{\text{time\_Runnable\_tasks\_exist}}{\text{total\_lifetime\_of\_cpu\_rq}} \right) \times 100
\]

at the time that the above measurement is made.
- If there is some task running on the CPU run queue all the time, then the CPU is 100% utilized.
- If there is no task to run on the CPU at some point, then the utilization of the CPU declines.

What is it then? 0 % or 0% < x < 100%? If it is 0%, why not use nr_running (number of tasks running) instead?
Measuring the idleness of a CPU

- Importance of choosing the right metric for measuring utilization of CPU

The above move ensures that idle CPUs have higher residency times in idle states. This can be ensured only with the %util metric and not nr_running.
Measuring the idleness of a CPU

- Behavior of tasks: How they affect the amount of idleness of a CPU

```
Never sleeps

100 % task when measured anytime within 1s

10 % task if measured every 10ms

10 % task if measured at the end of 1s
```

```
Task forks

awake 1ms sleeps 9ms awake 1ms sleeps 9ms

10ms 10ms

100 % task when measured within 1s

10 % task if measured every 10ms

10 % task if measured at the end of 1s
```
Measuring the idleness of a CPU

Importance of choosing the right metric for measuring utilization of CPU

- Currently in the scheduler, the following is how the idleness of a CPU is measured.

  - If we were to use the % utilization of task as a metric then:

  ![Diagram](image)

  
  #tasks on cpu0 < #tasks on cpu1
  
  Thus cpu0 is less idle than cpu1

  %util of cpu1 < %util of cpu0
  
  Thus cpu1 is less idle than cpu0

Thus %util identifies the busier CPU appropriately.
Per Entity Load Tracking Metric

- *Power aware scheduler makes use of the %utilization of CPU to measure how idle the CPU is.*

  Hence a group_leader is one in which the utilization of CPUs add up to very near to the group's capacity.

- How do we measure the utilization of a task?
  
  Here is where the Per Entity Load Tracking metric comes in.

- The Per Entity Load Tracking metric comes with many improvements over the current approach of tracking CPU loads.
Per Entity Load Tracking Metric

- Current Approach to calculating cpu->utilization
  
  \[
  \text{cpu->utilization} = \frac{\text{amount_of_time_cpu_rq_is_active}}{\text{amount_of_time_cpu_rq_is_(active+idle)}}
  \]

- Per Entity Load tracking approach to calculating cpu->utilization
  
  \[
  \text{cpu->utilization} = \frac{\text{amount_of_time_cpu_rq_is_active}}{\text{amount_of_time_cpu_rq_is_(active+idle)}}
  \]

  However

  All points in the history of the cpu->util is given equal importance

  Time-line of cpu->util

  • Coarse grained tracking of cpu->util compared to the PJT metric

  The history decays with time and more importance is given to the recent observation of cpu->utilization

  Time-line of cpu->util

  • Fine grained tracking of cpu->util
Per Entity Load Tracking Metric

- All the above points make the measurement of the utilization of a CPU more accurate and stable. But there are a few drawbacks to this metric:
  - Time taken to pick up cpu>util
  - Time taken to decay the cpu->util

which could lead to incorrect prediction of CPU utilization.
Per Entity Load Tracking Metric

- For example, when there are many tasks waking up on a CPU that is idle for very long, the cpu_rq->utilization will take time to catch up to 100% to reflect those many tasks.

- When the tasks leave the run queue, the cpu_rq->utilization will take time to decay so as to reflect no tasks.

- Hence at times we might need to fall back to the number_of_tasks_running on the CPU to measure its utilization in certain cases like bursty wakeups.

- Hence under stable load, per entity load tracking metric gives an accurate picture of the CPU utilization.

- However under steep increase or decrease of load, nr_running is a better measure of CPU utilization.
Where does the Power Aware Scheduler stand today?

- Alex Shi from Intel has posted out a series of patches implementing the Power Aware Scheduler as explained above.
- We, scheduler developers had discussions around:
  - The usage of the Per Entity Load Tracking Metric in the Power Aware Scheduler.
  - Areas where the metric fell short in evaluating the CPU utilization.
  - Scheduling policies of the Power Aware scheduler.
  - Implementation of the Power Aware Scheduler during fork and wake up of tasks and more implementation specifics.
- As a result, significant improvements in the code and design of Power Aware Scheduler has been achieved.
Where does the Power Aware Scheduler stand today?

- The **Per Entity Load Tracking** metric has been up-streamed into mainline.
- However the *usage of it in load balancing and power aware scheduling*-the recent efforts, is still under discussion.
- The reason being, any major changes in the Linux scheduler demands **significant positive experimental results** over its current implementation.
- There could be **points which we have missed** while discussing, which could show up during experimental evaluation.
- Thus the **following slides evaluate the Power Aware Scheduler** on two aspects:
  - Behavior: Consolidation of tasks under lightly loaded systems; Fallback to default scheduling otherwise.
  - Performance per Watt
Experimental Evaluation of Power Aware Scheduler

- Demonstration of the “behavior” of Power Aware Scheduler against today's scheduler: Load consolidation against Spreading of Load

Benchmark: Ebizzy
Platform: 2 socket, 8 cores per socket x86 machine
Experimental Evaluation of Power Aware Scheduler

- Demonstration of the “behavior” of Power Aware Scheduler against today's scheduler: Load Consolidation to fewer power domains under lightly loaded systems.

Benchmark: Kernbench
Platform: 2 socket, 8 cores per socket x86 machine
Experimental Evaluation of Power Aware Scheduler

- Demonstration of the “behavior” of Power Aware Scheduler against today's scheduler: 
  Load Consolidation to fewer power domains under lightly loaded systems.

Benchmark: Ebizzy
Platform: 2 socket, 8 cores per socket, SMT-4, Power machine

© 2013 IBM Corporation
Experimental Evaluation of Power Aware Scheduler

- Demonstration of the “behavior” of Power Aware Scheduler against today’s scheduler: Load Consolidation to fewer power domains under lightly loaded systems.

Benchmark: Kernbench
Platform: 2 socket, 8 cores per socket, SMT-4, Power machine
Experimental Evaluation of Power Aware Scheduler

- Demonstration of the impact of power aware scheduler relative to today's scheduler on the “performance per Watt” of benchmarks.

**Benchmark: Kernbench**
Platform: 2 socket, 8 cores per socket x86 machine

**Benchmark: Ebizzy**
Platform: 2 socket, 8 cores per socket x86 machine
Other Efforts towards Power Efficient Scheduling

Heterogeneous Platform Scheduling

Map nature of tasks to the CPU Power

Example: ARM's big.LITTLE CPUs

- Do not wake up higher power consuming CPUs – big CPUs if the tasks do not demand so much performance
- Schedule such tasks on LITTLE CPUs
- The nature of the task is known by the Per Entity Load Tracking measurement of the CPU utilization of the task
Other Efforts towards Power Efficient Scheduling

Packing Small Tasks

- Pack small tasks

- Ensure only selected power domains are responsible for handling light weight tasks so that the other power domains can go to idle state and save power.

- This will ensure that the CPUs of these power domains are not frequently woken up to handle small tasks and can have longer residency time in idle states.

- Identify light weight tasks using Per Entity Load Tracking metric.
References

Links to my discussions around the Per Entity Load Tracking Metric with the community:

• Initial efforts towards integrating the Per-Entity-Load-Tracking metric into the load balancer
  https://lwn.net/Articles/521272/

• Subsequent challenges of integrating the Per-Entity-Load-Tracking metric into the Normal Load Balancing of the Linux Scheduler
  https://lkml.org/lkml/2013/1/1/89

Links to the discussions around the Power Efficiency of the scheduler with the community:

• Alex Shi's patchset on the Power Aware Scheduler
  http://lwn.net/Articles/545910/

• Vincent Guittot's patchset on Packing small Tasks
  http://lwn.net/Articles/518834/

• Morten Rasmussens's patchset on Heterogeneous CPU scheduling
  http://lwn.net/Articles/544358/

• Per-Entity-Load-Tracking Metric patchset and discussions around it
  https://lkml.org/lkml/2012/6/27/644
QUESTIONS?
Legal Statement

• Copyright International Business Machines Corporation 2013.

• Permission to redistribute in accordance with Linux Foundation Collaboration Summit submission guidelines is granted; all other rights reserved.

• This work represents the view of the authors and does not necessarily represent the view of IBM

• IBM, IBM logo, ibm.com are trademarks of International Business Machines Corporation in the United States, other countries, or both.

• Intel is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

• Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

• Other company, product, and service names may be trademarks or service marks of others.

• References in this publication to IBM products or services do not imply that IBM intends to make them available in all countries in which IBM operates.

• INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you. This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.