The Linux Scheduler, today and looking forward

Dhaval Giani
dhaval@linux.vnet.ibm.com
Agenda

- Introduction
- Old Scheduler
- Need for the new scheduler
- CFS
- Group scheduling
- Load Balancing
- Future
The O(1) Scheduler

- Known as the ultra scalable scheduler
- The typical scheduling operations were O(1)
  - enqueue
  - dequeue
- Used rotating priority arrays
- Basically a Weighted Round Robin scheduler
  - Used nice values for determining time slice
  - Used two arrays, active and expired.
    - Task finishes its timeslice and goes to the expired array
    - When active is empty, the arrays are exchanged and expired becomes active and active, expired
Need for a new scheduler

O(1) had problems
Need for a new scheduler

- O(1) had problems
  - Determinism
    - Was not
    - Erratic scheduling patterns
Need for a new scheduler

-O(1) had problems
  - Determinism
    - Was not
    - Erratic scheduling patterns
  - Runtime Accounting
    - Was statistical
    - Or too coarse
Need for a new scheduler

Led to problems
Need for a new scheduler

- Led to problems
  - Fair allocation
Need for a new scheduler

Led to problems

- Fair allocation
  - Did not provide equal bandwidth to tasks at same priority esp on SMP systems
Need for a new scheduler

Led to problems

- Fair allocation
  - Did not provide equal bandwidth to tasks at same priority esp on SMP systems
  - Similar workloads finish at varying times. Not good!
Need for a new scheduler

Led to problems

- Fair allocation
  - Did not provide equal bandwidth to tasks at same priority esp on SMP systems
  - Similar workloads finish at varying times. Not good!
- Desktop experience
Need for a new scheduler

Led to problems

- Fair allocation
  - Did not provide equal bandwidth to tasks at same priority esp on SMP systems
  - Similar workloads finish at varying times. Not good!

- Desktop experience
  - Desktop Applications,
    - Sleep long
    - Short time on CPU
Need for a new scheduler

Led to problems

✔ Fair allocation
  - Did not provide equal bandwidth to tasks at same priority esp on SMP systems
  - Similar workloads finish at varying times. Not good!

✔ Desktop experience
  - Desktop Applications,
    • Sleep long
    • Short time on CPU
  - Need to get CPU fast
    • Otherwise noticeable effects, for example, audio stutters
The Completely Fair Scheduler or just the CFS

Written by Ingo Molnar, and merged in v2.6.23 of the kernel
The Completely Fair Scheduler or just the CFS

- Written by Ingo Molnar, and merged in v2.6.23 of the kernel
- Moved from priority arrays to time based queues
The Completely Fair Scheduler or just the CFS

- Written by Ingo Molnar, and merged in v2.6.23 of the kernel
- Moved from priority arrays to time based queues
- Fairness
  - Equal bandwidth for same priority
  - Provided over __sched_period()
The Completely Fair Scheduler or just the CFS

- Written by Ingo Molnar, and merged in v2.6.23 of the kernel
- Moved from priority arrays to time based queues
- Fairness
  - Equal bandwidth for same priority
  - Provided over __sched_period()
- Uses an RB Tree to implement the queue
  - Uses vruntime as its index
  - vruntime is weight proportional runtime
    - That means heavier tasks run for longer and get charged lesser
The Completely Fair Scheduler or just the CFS

- Written by Ingo Molnar, and merged in v2.6.23 of the kernel
- Moved from priority arrays to time based queues
- Fairness
  - Equal bandwidth for same priority
  - Provided over __sched_period()
- Uses an RB Tree to implement the queue
  - Uses vruntime as its index
  - vruntime is weight proportional runtime
    - That means heavier tasks run for longer and get charged less
- Nice is now exponential and not linear
Interactivity

Interactivity improved, but how?
Interactivity

- Interactivity improved, but how?
- Two major “features”
Interactivity

Interactivity improved, but how?

Two major “features”

- Shorter time slices: On an average, the CFS has shorter time slices.
  - With the help of these, tasks which are further behind, get to run faster
Interactivity

Interactivity improved, but how?

Two major “features”

- Shorter time slices: On an average, the CFS has shorter time slices.
  - With the help of these, tasks which are further behind, get to run faster

- Wakeup behavior
  - Typical interactive task -> sleeps for long, and then has a short burst
  - Waiting for CPU, not good. Shows up as stutters in amarok
  - So we queue up a newly woken up task to the head of the queue
So how fair is the CFS?

• Tried out massive_intr.c
So how fair is the CFS?

- Tried out massive_intr.c
- Written by Satoru Takeuchi
So how fair is the CFS?

- Tried out massive_intr.c
- Written by Satoru Takeuchi
- Takes two arguments.
  - ✔ Number of threads
  - ✔ How long to run the program
So how fair is the CFS?

- Tried out massive_intr.c
- Written by Satoru Takeuchi
- Takes two arguments.
  - Number of threads
  - How long to run the program
- Very simple
  - Runs a thread for 8ms and then puts it to sleep for 1ms
  - At the end of the time, it kills all the threads, and prints out the time each thread got
Runtime for various threads in 2.6.22 using massive_interrupt.c
Runtime for various threads in 2.6.27-rc6 using massive_interrupt.c
Refining the CFS: Group Scheduling

Administrator finds it easier to control groups

- Database vs pids 4213, 4214, 4215...
Refining the CFS: Group Scheduling

- Administrator finds it easier to control groups
  - Database vs pids 4213, 4214, 4215...
- Control Groups provided the ability to group threads arbitrarily
  - So, group “blog”, could consist of webservers and database threads
Refining the CFS: Group Scheduling

- Administrator finds it easier to control groups
  - Database vs pids 4213,4214,4215...
- Control Groups provided the ability to group threads arbitrarily
  - So, group “blog”, *could* consist of webserver and database threads
- Srivatsa Vaddagiri extended the CFS to provide group scheduling, which would give control over groups such as “blog”
  - Merged in v2.6.24
Scheduler Entity

The CFS as well as the O(1) scheduler deal with just tasks
Scheduler Entity

- The CFS as well as the O(1) scheduler deal with just tasks
- Group scheduling requires scheduler to deal with "task groups"
Scheduler Entity

- The CFS as well as the O(1) scheduler deal with just tasks
- Group scheduling requires scheduler to deal with “task groups”
- Enter sched_entity
Scheduler Entity

- The CFS as well as the O(1) scheduler deal with just tasks
- Group scheduling requires scheduler to deal with “task groups”
- Enter sched_entity
  - Helped with reuse of the code
  - Can mean either a task or a task group. Basically something that can be “scheduled”
  - Keeps track of vital scheduling data, such vruntime
  - Scheduler core modified to work entities rather than tasks
Group Scheduling

❖ Take 1
  ✓ Scheduling a two step decision
Group Scheduling

Take 1

- Scheduling a two step decision
- First we choose which group to schedule in
Group Scheduling

Take 1

- Scheduling a two step decision
- First we choose which group to schedule in
- Then, which task in the group selected in the previous step gets to run
Group Scheduling

Take 1

✓ Scheduling a two step decision
✓ First we choose which group to schedule in
✓ Then, which task in the group selected in the previous step gets to run
✓ Limited to just one level of grouping
Group Scheduling

Take 1

- Scheduling a two step decision
- First we choose which group to schedule in
- Then, which task in the group selected in the previous step gets to run

- Limited to just one level of grouping
- Tasks in the “root” group are not really
- All tasks are grouped
Group Scheduling

- Take 1
  - Scheduling a two step decision
  - First we choose which group to schedule in
  - Then, which task in the group selected in the previous step gets to run

- Limited to just one level of grouping
- Tasks in the “root” group are not really
  - All tasks are grouped
  - Those which are not grouped, form a group :-)
Group Scheduling

Not really fair
Group Scheduling

- Not really fair
- Did not allow multiple levels of grouping
Group Scheduling

- Not really fair
- Did not allow multiple levels of grouping
- Take 2
  - Changed the definition of fairness
    - Remember, the root cgroup did not share bandwidth between the tasks and groups fairly
Group Scheduling

- Not really fair
- Did not allow multiple levels of grouping
- Take 2

  - Changed the definition of fairness
    - Remember, the root cgroup did not share bandwidth between the tasks and groups fairly
  - At every level we choose an entity
    - If it is a task, we run it
    - If it is a group, we choose another entity within it
Group Scheduling

- Not really fair
- Did not allow multiple levels of grouping
- Take 2
  - Changed the definition of fairness
    - Remember, the root cgroup did not share bandwidth between the tasks and groups fairly
  - At every level we choose an entity
    - If it is a task, we run it
    - If it is a group, we choose another entity within it
  - Available since v2.6.26
How # of threads in Group B affects fairness for Group A

- With Fair Group Scheduler
- Without Fair Group Scheduler
- Ideal Results

Allocated CPU Bandwidth to Group A vs. Number of threads in Group B
Real Time

- The highest priority scheduling class
Real Time

- The highest priority scheduling class
- Implements the POSIX standard
  - ✔ SCHED_FIFO
  - ✔ SCHED_RR
Real Time

- The highest priority scheduling class
- Implements the POSIX standard
  - SCHED_FIFO
  - SCHED_RR
- RT Hard limits
Real Time

- The highest priority scheduling class
- Implements the POSIX standard
  - SCHED_FIFO
  - SCHED_RR
- RT Hard limits
  - Introduced in v2.6.25
Real Time

- The highest priority scheduling class
- Implements the POSIX standard
  - SCHED_FIFO
  - SCHED_RR
- RT Hard limits
  - Introduced in v2.6.25
  - Trivial case for the group scheduler
Real Time

- The highest priority scheduling class
- Implements the POSIX standard
  - SCHED_FIFO
  - SCHED_RR
- RT Hard limits
  - Introduced in v2.6.25
  - Trivial case for the group scheduler
  - sched_rt_runtime_us -> Runtime Budget
  - sched_rt_period_us -> The refresh rate
Real Time

- The highest priority scheduling class
- Implements the POSIX standard
  - SCHED_FIFO
  - SCHED_RR
- RT Hard limits
  - Introduced in v2.6.25
  - Trivial case for the group scheduler
  - sched_rt_runtime_us -> Runtime Budget
  - sched_rt_period_us -> The refresh rate
- Prevents RT tasks from taking over the system
RT Group Scheduling

 sched_rt_entity introduced

 ✓ An abstraction similar to sched_entity
RT Group Scheduling

- `sched_rt_entity` introduced
  - An abstraction similar to `sched_entity`
- Two tunables
  - `rt_period_us`
  - `rt_runtime_us`
Load Balancing

“All what we talked about till now, was with UP in mind"
Load Balancing

- All what we talked about till now, was with UP in mind
- Linux can run on machines with 4096 CPUs
Load Balancing

- All what we talked about till now, was with UP in mind
- Linux can run on machines with 4096 CPUs
  - At least according to theory
Load Balancing

- All what we talked about till now, was with UP in mind
- Linux can run on machines with 4096 CPUs
  - At least according to theory
- Scheduler needs to be extended
Load Balancing

- All what we talked about till now, was with UP in mind
- Linux can run on machines with 4096 CPUs
  - At least according to theory
- Scheduler needs to be extended
  - Global Scheduling: N CPUs, 1 Runqueue
Load Balancing

• All what we talked about till now, was with UP in mind
• Linux can run on machines with 4096 CPUs
  ✔ At least according to theory
• Scheduler needs to be extended
  ✔ Global Scheduling: N CPUs, 1 Runqueue
  ✔ Partitioned Scheduling: N CPUs, N Runqueues, no interaction between runqueues
Load Balancing

- All what we talked about till now, was with UP in mind
- Linux can run on machines with 4096 CPUs
  - At least according to theory
- Scheduler needs to be extended
  - Global Scheduling: N CPUs, 1 Runqueue
  - Partitioned Scheduling: N CPUs, N Runqueues, no interaction between runqueues
  - Distributed Scheduling: N CPUs, N Runqueues, loosely coupled to approximate global scheduling
Load Balancing

- All what we talked about till now, was with UP in mind
- Linux can run on machines with 4096 CPUs
  - At least according to theory
- Schedulers needs to be extended
  - Global Scheduling: N CPUs, 1 Runqueue
  - Partitioned Scheduling: N CPUs, N Runqueues, no interaction between runqueues
  - Distributed Scheduling: N CPUs, N Runqueues, loosely coupled to approximate global scheduling
- Linux uses distributed scheduling
Scheduler Domains

Today's hardware

- Various sizes
- Various shapes
- In order to handle these, we build sched domains
Scheduler Domains

- Today's hardware
  - Various sizes
  - Various shapes
  - In order to handle these, we build sched domains
- Domains group processors
  - Based on various properties such as shared pipelines, shared caches
Scheduler Domains

- Today's hardware
  - Various sizes
  - Various shapes
  - In order to handle these, we build sched domains

- Domains group processors
  - Based on various properties such as shared pipelines, shared caches

- CPUsets allow the user to carve up CPUs into sets
  - Also used for load balancing decisions
Load Balancing in SCHED_OTHER

The basic idea is to balance any two runqueues
Load Balancing in SCHED_OTHER

- The basic idea is to balance any two runqueues
  - Converge to a global balance
Load Balancing in SCHED_OTHER

- The basic idea is to balance any two runqueues
  - Converge to a global balance
  - Balance “near” runqueues more frequently and slow down as we go up
Load Balancing in SCHED_OTHER

- The basic idea is to balance any two runqueues
  - Converge to a global balance
- Balance “near” runqueues more frequently and slow down as we go up
  - Done using sched domains
Load Balancing in SCHED_OTHER

- The basic idea is to balance any two runqueues
  - Converge to a global balance
- Balance “near” runqueues more frequently and slow down as we go up
  - Done using sched domains
  - Compensates for the fact that memory accesses and migration to “far” CPUs is more expensive
Load Balancing in SCHED_OTHER

- The basic idea is to balance any two runqueues
  - Converge to a global balance
- Balance “near” runqueues more frequently and slow down as we go up
  - Done using sched domains
  - Compensates for the fact that memory accesses and migration to “far” CPUs is more expensive
- Sched groups
  - Basically the child domains of a domain
Load Balancing in SCHED_OTHER

- The basic idea is to balance any two runqueues
  - Converge to a global balance
  - Balance “near” runqueues more frequently and slow down as we go up
  - Done using sched domains
  - Compensates for the fact that memory accesses and migration to “far” CPUs is more expensive

- Sched groups
  - Basically the child domains of a domain
  - Pick the busiest group and try to pull from there as long as we don't pull too much
SMP Nice

Introduced by Peter Williams, ~2.6.18
SMP Nice

- Introduced by Peter Williams, ~2.6.18
- Load Balancer introduced to the concept of nice values
SMP Nice

- Introduced by Peter Williams, ~2.6.18
- Load Balancer introduced to the concept of nice values
  - Fairness maintained across CPUs
  - Balance run queues based on weight and not number of tasks
SMP Nice

- Introduced by Peter Williams, ~2.6.18
- Load Balancer introduced to the concept of nice values
  - Fairness maintained across CPUs
  - Balance run queues based on weight and not number of tasks

Group SMP Nice
SMP Nice

- Introduced by Peter Williams, ~2.6.18
- Load Balancer introduced to the concept of nice values
  - Fairness maintained across CPUs
  - Balance run queues based on weight and not number of tasks
- Group SMP Nice
  - Significant complication
SMP Nice

- Introduced by Peter Williams, ~2.6.18
- Load Balancer introduced to the concept of nice values
  - Fairness maintained across CPUs
  - Balance run queues based on weight and not number of tasks

Group SMP Nice

- Significant complication
- Weight of a task became proportional to the weight of its supertask
SMP Nice

- Introduced by Peter Williams, ~2.6.18
- Load Balancer introduced to the concept of nice values
  - Fairness maintained across CPUs
  - Balance run queues based on weight and not number of tasks
- Group SMP Nice
  - Significant complication
  - Weight of a task became proportional to the weight of its *supertask*
  - But the supertask can be spread across multiple CPUs
SMP Nice

Introduced by Peter Williams, ~2.6.18
Load Balancer introduced to the concept of nice values
  ✓ Fairness maintained across CPUs
  ✓ Balance run queues based on weight and not number of tasks

Group SMP Nice

✓ Significant complication
✓ Weight of a task became proportional to the weight of its supertask
✓ But the supertask can be spread across multiple CPUs
✓ That means, the weight of a task is dependent on other runqueues.
SMP Nice

- Introduced by Peter Williams, ~2.6.18
- Load Balancer introduced to the concept of nice values
  - Fairness maintained across CPUs
  - Balance run queues based on weight and not number of tasks

Group SMP Nice

- Significant complication
- Weight of a task became proportional to the weight of its supertask
- But the supertask can be spread across multiple CPUs
- That means, the weight of a task is dependent on other runqueues.
- Bad for scalability
Distributed Group Balancing

Since we have a view of the tasks weight from the root group, we can balance on the weight of the root's runqueue itself.
Distributed Group Balancing

Since we have a view of the tasks weight from the root group, we can balance on the weight of the root's runqueue itself.

✓ Just take care that we account only the normalized weight of the task.
Distributed Group Balancing

Since we have a view of the tasks weight from the root group, we can balance on the weight of the root's runqueue itself

- Just take care that we account only the normalized weight of the task

Just one tiny problem

- How do we do it sanely?
Distributed Group Balancing

Since we have a view of the tasks weight from the root group, we can balance on the weight of the root's runqueue itself

- Just take care that we account only the normalized weight of the task

Just one tiny problem

- How do we do it sanely?
- That is, not touch other CPUs
Distributed Group Balancing

Since we have a view of the tasks weight from the root group, we can balance on the weight of the root's runqueue itself

- Just take care that we account only the normalized weight of the task

Just one tiny problem

- How do we do it sanely?
- That is, not touch other CPUs

Use sched domains
Distributed Group Balancing

Since we have a view of the tasks weight from the root group, we can balance on the weight of the root's runqueue itself

- Just take care that we account only the normalized weight of the task

Just one tiny problem

- How do we do it sanely?
- That is, not touch other CPUs

Use sched domains

- Re-compute shares as we walk up the tree.
Distributed Load Balancing

- Few corner cases
Distributed Load Balancing

- Few corner cases
  - Since we do only integer divisions, we can lose shares due to rounding errors
Distributed Load Balancing

Few corner cases

- ✓ Since we do only integer divisions, we can lose shares due to rounding errors
  - Solved by ensuring, the weight at the top domain will be the sum of the shares
Distributed Load Balancing

- Few corner cases
  - Since we do only integer divisions, we can lose shares due to rounding errors
    - Solved by ensuring, the weight at the top domain will be the sum of the shares
    - Will limit the loses
Distributed Load Balancing

Few corner cases

✔ Since we do only integer divisions, we can lose shares due to rounding errors
  – Solved by ensuring, the weight at the top domain will be the sum of the shares
  – Will limit the loses

✔ A boot strap problem – A group with no tasks
Distributed Load Balancing

Few corner cases

- Since we do only integer divisions, we can lose shares due to rounding errors
  - Solved by ensuring, the weight at the top domain will be the sum of the shares
  - Will limit the loses

- A boot strap problem – A group with no tasks
  - Needs instant recalculation on arrival of task, otherwise shares will remain zero till we redistribute shares
Distributed Load Balancing

- Few corner cases
  - Since we do only integer divisions, we can lose shares due to rounding errors
    - Solved by ensuring, the weight at the top domain will be the sum of the shares
    - Will limit the loses
  - A boot strap problem – A group with no tasks
    - Needs instant recalculation on arrival of task, otherwise shares will remain zero till we redistribute shares
    - Expensive. Arrival and departure of tasks is quite frequent
Distributed Load Balancing

❖ Few corner cases

✔ Since we do only integer divisions, we can lose shares due to rounding errors
  – Solved by ensuring, the weight at the top domain will be the sum of the shares
  – Will limit the loses

✔ A boot strap problem – A group with no tasks
  – Needs instant recalculation on arrival of task, otherwise shares will remain zero till we redistribute shares
  – Expensive. Arrival and departure of tasks is quite frequent
  – Never reduce shares to zero. Inflate shares of idle groups
Distributed Load Balancing

Few corner cases

✓ Since we do only integer divisions, we can lose shares due to rounding errors
  – Solved by ensuring, the weight at the top domain will be the sum of the shares
  – Will limit the loses

✓ A boot strap problem – A group with no tasks
  – Needs instant recalculation on arrival of task, otherwise shares will remain zero till we redistribute shares
  – Expensive. Arrival and departure of tasks is quite frequent
  – Never reduce shares to zero. Inflate shares of idle groups
  – Has some short term unfairness, but not more than what was already present, due to rebalancing
The Future

- The single runqueue approach
The Future

- The single runqueue approach
  - Group scheduling has a hierarchical task selection
The Future

The single runqueue approach

- Group scheduling has a hierarchical task selection
  - Not good for interactivity
The Future

- The single runqueue approach
  - Group scheduling has a hierarchical task selection
    - Not good for interactivity
  - Need to provide latency isolation
The Future

- The single runqueue approach
  - ✓ Group scheduling has a hierarchical task selection
    - Not good for interactivity
  - ✓ Need to provide latency isolation
- Faster convergence to fairness for group scheduling
The Future

- The single runqueue approach
  - Group scheduling has a hierarchical task selection
    - Not good for interactivity
  - Need to provide latency isolation
- Faster convergence to fairness for group scheduling
- Looking at RT scheduling function, independent of PI
The Future

- The single runqueue approach
  - Group scheduling has a hierarchical task selection
    - Not good for interactivity
  - Need to provide latency isolation
- Faster convergence to fairness for group scheduling
- Looking at RT scheduling function, independent of PI
  - Allows us to experiment with more advanced RT scheduling
  - Possibly allow us to extend PI for SCHED_OTHER
Thank You!

Questions?
Legal Statement

- This work represents the view of the authors and does not necessarily represent the view of IBM.
- IBM is a registered trademark of International Business Machines Corporation in the United States and/or other countries.
- Linux is a registered trademark of Linus Torvalds.
- Other company, product, and service names may be trademarks or service marks of others.
BACKUP
Scheduler Classes

Scheduler Classes, a definition,

An extensible hierarchy of scheduler modules which encapsulate scheduling policy details and are handled by the scheduler core without the core code assuming about them too much

Ingo Molnar

Essentially what he said, with a few custom changes :)

An extensible hierarchy of scheduler modules which encapsulate scheduling policy details and are handled by the scheduler core without the core code assuming about them too much
The CFS

More vruntime love

\[ vruntime = \frac{runtime \cdot runqueue\_weight}{weight} \]

Calculated as follows

- When a task forks, vruntime set so that it comes as the rightmost
  - Ensures that it does not affect the fairness promised to tasks already existing

- When a task runs, the it runs is normalized to its weight, and is added to its vruntime

- CFS tracks a variable known as cfs_rq->min_vruntime
Being nice

- CFS changed the definition of how nice worked
  - O(1) had liner values for nice
  - CFS has a exponential scale
  - $\text{Nice}_0 = 1024$
  - $\text{Nice}_{i-1} = 1.25 \times \text{Nice}_i$
- Time slice dependent on weight
- Weight dependent on nice
- Therefore, nice has a much stronger effect on time slices now.
Some basic definitions

- We have tasks Ti of weight wi running on CPU Pj such that its runqueue has weight
  \[ rw_j = \sum_{i|\tau_i \in P_j} w_i \]

  ✔ Each task gets \( w_i/rw_j \) runtime

- A task can be a supertask with weight \( w_i \) with subtasks spread across every CPU
  ✔ Gives rise to the concept of shares, which is per CPU weight of the supertask
  \[ w_i = \sum_j s_{i,j} \]
  \[ s_{i,j} = \frac{w_i rw_{i,j}}{rw_i} \]
Some basic definitions

另一位概念

✓ Task weight as viewed from the root group

\[ W_i = \prod_{\gamma} \frac{w_{k,\gamma}}{rw_{l,\gamma}} | k \in T_l, l_{\gamma} = k_{\gamma-1} \]

✓ Which gives rise to

\[ \sum_{i \mid i \in T_0} w_i = \sum_{k \mid k \in P_j, \! \! super(k)} W_k \]
Sched Features

- Some key features,
- NEW_FAIR_SLEEPERS: Provides a bonus to tasks that just wake up.
- NORMALIZED_SLEEPERS: Normalizes the aforementioned bonus
- START_DEBIT: Demotes a newly forked task to the right of the runqueue
Distributed Load Balancing

Wake Affine

- Requires precise re-calculation
  - Not good!
- We know,

\[ s_{i,j} = \frac{w_i r w_{i,j}}{r w_i} \]

- So we add in a delta

\[ s'_{i,j} = \frac{w_i (r w_{i,j} + \delta w)}{(r w_i + \delta w)} \]

- Express s'-s as a function of delta(w)