Scheduling in Xen: Past, Present and Future

Dario Faggioli
dario.faggioli@citrix.com

Seattle, WA – 18th of August, 2015
Introduction
Welcome

- Hello, my name is Dario
- Working on Xen (from within Citrix) since 2011
- Mostly hypervisor stuff, but also toolstack: NUMA, cpupools, scheduling
Scheduling & Scheduling in Xen
Scheduling in General

Is scheduling really important, e.g., in OSes?
Is scheduling really important, e.g., in OSes? *Well...*
Is scheduling really important, e.g., in OSes? Well...

- most of the time, there isn’t even **any CPU overbooking**
- when there’s overbooking, **not** everything is equally important
- **I/O** is more important!
Scheduling in Virtualization

Is scheduling really important on a virtualized host?

- There is **pretty much always** CPU overbooking
- All activities (i.e., all VMs) are (potentially) **equally important**
Scheduling in Virtualization

Is scheduling really important on a virtualized host?

- There is **pretty much always** CPU overbooking
- All activities (i.e., all VMs) are (potentially) **equally important**

Scheduler hacker: ≪Hey, this sounds pretty cool...≫
Is scheduling really important on a virtualized host?

- There is **pretty much always** CPU overbooking
- All activities (i.e., all VMs) are (potentially) **equally important**

Scheduler hacker: «Hey, this sounds pretty cool...»

Well, it’s not! I/O is **still** more important!

Oh come on!!!
Wait a Second...

And we, on Xen, are even more special:
And we, on Xen, are even more **special**:

- How does Xen handle I/O:

```
Control Domain
NetBSD or Linux

device model (qemu)
toolstack

Hardware
Drivers
netback
blkback

Driver Domain
Paravirtualized (PV)
Domain: NetBSD or Linux
netback
blkback

Paravirtualized (PV)
Domain: NetBSD or Linux
netfront
blkfront

Xen Hypervisor

I/O Devices
CPU
Memory

Hardware

Control Domain
NetBSD or Linux

device model (qemu)
toolstack

Hardware
Drivers
netback
blkback

Driver Domain
Paravirtualized (PV)
Domain: NetBSD or Linux
netfront
blkfront

Paravirtualized (PV)
Domain: NetBSD or Linux
netfront
blkfront

Fully Virtualized (HVM)
Domain: Windows, FreeBSD...

Xen Hypervisor

I/O Devices
CPU
Memory

Hardware
```
Wait a Second... (cont. II)

So, I/O, turns out you actually need scheduling, eh?

LIKE A BOSS
History of Xen Scheduling
(Past)
I Mean... Honestly...

[Some info at http://wiki.xen.org/wiki/Xen Project]

▶ we just killed SEDF in 4.6 (yay!!)
check RTDS for something similar (but better!)
▶ that's it
I Mean... Honestly...

WHO CARES ?!?!
I Mean... Honestly...

WHO CARES ?!?!?

No, seriously:
I Mean... Honestly...

WHO CARES ?!?!?

No, seriously:

I Mean... Honestly...

WHO CARES ?!?!?

No, seriously:

- we just killed SEDF in 4.6 (yay!!)
  check RTDS for something similar (but better!)
I Mean... Honestly...

WHO CARES ?!?!?

No, seriously:

- we just killed SEDF in 4.6 (yay!!)
  check RTDS for something similar (but better!)
- that’s it
Xen’s Scheduler Features
(Preset)
Hard and Soft Affinity

- pinning: you can run there and only there!
  `# xl vcpu-pin vm1 0 2`

- hard affinity: you can’t run outside of that spot
  `# xl vcpu-pin vm1 all 8-12`

- soft affinity: you can’t run outside of that spot and, preferably, you should run there
  `# xl vcpu-pin vm1 all - 10,11`

Same achieved with `cpus=` and `cpus_soft=` in config file.

`cpus=` or `cpus_soft=` in config file control where memory is allocated
**Hard and Soft Affinity (cont.)**

<table>
<thead>
<tr>
<th>Name</th>
<th>ID</th>
<th>VCPU</th>
<th>CPU State</th>
<th>Time(s)</th>
<th>Affinity (Hard / Soft)</th>
</tr>
</thead>
<tbody>
<tr>
<td>vml</td>
<td>5</td>
<td>0</td>
<td>2</td>
<td>1.5</td>
<td>2 / all</td>
</tr>
<tr>
<td>vml</td>
<td>5</td>
<td>1</td>
<td>15</td>
<td>1.9</td>
<td>all / all</td>
</tr>
<tr>
<td>vml</td>
<td>5</td>
<td>2</td>
<td>13</td>
<td>0.8</td>
<td>all / all</td>
</tr>
<tr>
<td>vml</td>
<td>5</td>
<td>3</td>
<td>23</td>
<td>0.8</td>
<td>all / all</td>
</tr>
<tr>
<td>vml</td>
<td>5</td>
<td>0</td>
<td>8</td>
<td>1.5</td>
<td>8-12 / all</td>
</tr>
<tr>
<td>vml</td>
<td>5</td>
<td>1</td>
<td>9</td>
<td>1.9</td>
<td>8-12 / all</td>
</tr>
<tr>
<td>vml</td>
<td>5</td>
<td>2</td>
<td>9</td>
<td>0.8</td>
<td>8-12 / all</td>
</tr>
<tr>
<td>vml</td>
<td>5</td>
<td>3</td>
<td>8</td>
<td>0.8</td>
<td>8-12 / all</td>
</tr>
<tr>
<td>vml</td>
<td>5</td>
<td>0</td>
<td>11</td>
<td>1.5</td>
<td>8-12 / 10-11</td>
</tr>
<tr>
<td>vml</td>
<td>5</td>
<td>1</td>
<td>10</td>
<td>1.9</td>
<td>8-12 / 10-11</td>
</tr>
<tr>
<td>vml</td>
<td>5</td>
<td>2</td>
<td>11</td>
<td>0.8</td>
<td>8-12 / 10-11</td>
</tr>
<tr>
<td>vml</td>
<td>5</td>
<td>3</td>
<td>10</td>
<td>0.8</td>
<td>8-12 / 10-11</td>
</tr>
</tbody>
</table>
What about NUMA

Automatic placement policy in libxl (since Xen 4.2)

- acts at domain creation time, important for memory allocation!
- easy to tweak (at libxl build time, for now) heuristics:
  - use the smallest possible set of nodes (ideally, just one)
  - use the (set of) node(s) with fewer vCPUs bound to it ([will] consider both hard and soft affinity)
  - use the (set of) node(s) with the most free RAM (mimics the “worst fit” algorithm)
What about NUMA

Automatic placement policy in libxl (since Xen 4.2)

- acts at domain creation time, important for **memory allocation**!
- easy to tweak (at libxl build time, for now) heuristics:
  - use the smallest possible set of nodes (ideally, just one)
  - use the (set of) node(s) with fewer vCPUs bound to it ([will] consider both hard and soft affinity)
  - use the (set of) node(s) with the most free RAM (mimics the “worst fit” algorithm)

Example:
4 GB VM, 6 GB free on node0, 8 GB free on node1
What about NUMA

Automatic placement policy in libxl (since Xen 4.2)

- acts at domain creation time, important for memory allocation!
- easy to tweak (at libxl build time, for now) heuristics:
  - use the smallest possible set of nodes (ideally, just one)
  - use the (set of) node(s) with fewer vCPUs bound to it ([will] consider both hard and soft affinity)
  - use the (set of) node(s) with the most free RAM (mimics the “worst fit” algorithm)

Example:
4 GB VM, 6 GB free on node0, 8 GB free on node1
Bound vCPUs: 12 to node0, 16 to node1 $\implies$ use node0
What about NUMA

Automatic placement policy in libxl (since Xen 4.2)

- acts at domain creation time, important for **memory allocation**!
- easy to tweak (at libxl build time, for now) heuristics:
  - use the smallest possible set of nodes (ideally, just one)
  - use the (set of) node(s) with fewer vCPUs bound to it ([will] consider both hard and soft affinity)
  - use the (set of) node(s) with the most free RAM (mimics the “worst fit” algorithm)

Example:

4 GB VM, 6 GB free on node0, 8 GB free on node1

Bound vCPUs: 12 to node0, 16 to node1 $\implies$ use node0

Bound vCPUs: 8 to node0, 8 to node1 $\implies$ use node1
What about NUMA

Automatic placement policy in libxl (since Xen 4.2)

▶ acts at domain creation time, important for memory allocation!

▶ easy to tweak (at libxl build time, for now) heuristics:
  ▶ use the smallest possible set of nodes (ideally, just one)
  ▶ use the (set of) node(s) with fewer vCPUs bound to it ([will] consider both hard and soft affinity)
  ▶ use the (set of) node(s) with the most free RAM (mimics the “worst fit” algorithm)

Example:
4 GB VM, 6 GB free on node0, 8 GB free on node1
Bound vCPUs: 12 to node0, 16 to node1 \(\rightarrow\) use node0
Bound vCPUs: 8 to node0, 8 to node1 \(\rightarrow\) use node1

Coming: node distances, IONUMA, vNUMA
What about NUMA (cont.)

```
root@Zhaman:~# xl list -n
Name       ID  Mem  VCPUs  State  Time(s)  NODE  Affinity
Domain-0   0    503   4      r-----  3631.6  0      0
test       1   4096   8      r-----  2377.9  1      1
```
For Dom0

- `dom0_max_vcpu`: makes sense
- `dom0_vcpus_pin`: bleah!!
- `dom0_nodes`: new parameter. Place Dom0’s vCPUs and memory on one or more nodes
  - `strict` (default) uses hard affinity
  - `relaxed` uses soft affinity
For Dom0 (cont.)
Workin’ On
(Future)
(Better) Exploiting Intel PSR

Intel Platform Shared Resource Monitoring (PSR):
- Cache Monitoring Technology (CMT)
- Cache Allocation Technology (CAT)
(Better) Exploiting Intel PSR

Intel Platform Shared Resource Monitoring (PSR):

- Cache Monitoring Technology (CMT)
- Cache Allocation Technology (CAT)

CMT: gather information useful for mid-level load balancing decisions (e.g., at vCPU wakeup, or across sockets/NUMA nodes):

- how 'cache hungry' is a given pCPU?
- how much free cache is there on a given socket/NUMA node?
(Better) Exploiting Intel PSR

Intel Platform Shared Resource Monitoring (PSR):
- Cache Monitoring Technology (CMT)
- Cache Allocation Technology (CAT)

CMT: gather information useful for mid-level load balancing decisions (e.g., at vCPU wakeup, or across sockets/NUMA nodes):
- how ‘cache hungry’ is a given pCPU?
- how much free cache is there on a given socket/NUMA node?

CAT: when setting affinity, set CAT accordingly (and vice versa?):
- if you’re going to run on pCPUs #x, #y and #z...
- ...you better have some cache space reserved there!
Interactions With Guests’ (Linux’s) Scheduler

Scheduler paravirtualization?
Interactions With Guests’ (Linux’s) Scheduler

Scheduler paravirtualization?

**SHHSH!!** *There’s Linux people in the other room, what it they hear!?!?! :‐O*
Interactions With Guests’ (Linux’s) Scheduler

Scheduler paravirtualization?

**SHHSH!!**  *There’s Linux people in the other room, what it they hear?!?!?! :-O*

An example: scheduling domains:

- scalability measure (for the Linux scheduler)
- balance the load: often among SMT threads, less often between NUMA nodes!
- based on CPU topology
In Xen:

- vCPUs **wander** among pCPUs
- vCPU #0 and #1 of a guest can be SMT siblings or not... **at different times!!**
In Xen:

- vCPUs *wander* among pCPUs
- vCPU #0 and #1 of a guest can be SMT siblings or not... *at different times!!*
- it doesn’t make sense, *in the guest*, to try to optimize scheduling basing on something that varies
In Xen:

- vCPUs **wander** among pCPUs
- vCPU #0 and #1 of a guest can be SMT siblings or not... **at different times!!**
- it doesn’t make sense, *in the guest*, to try to optimize scheduling basing on something that varies

How to avoid this?

- flat layout for the scheduling domains → just one domain with all the pCPUs
Credit2 scheduler, authored by George, is still in experimental status.
Credit2 scheduler, authored by George, is still in experimental status.

**Take it out from there!!**
Credit2 scheduler, authored by George, is still in experimental status.

**Take it out from there!!**

Why? In general, more advanced, a lot of potential:

- historical load based load balancing
- runqueue kept in order of credit (instead than Round-Robin as in Credit1)
- configurable runqueue arrangement
Credit2: Why?

Complexity:
Credit2: Why?

Complexity:
- in Credit we have:
  - credits and weights
  - 2 priorities
  - oh, actually, it’s 3
  - active and non-active state of vCPUs
  - flipping between active/non-active means flipping between burning/non-burning credits, which in turns means wandering around among priorities

Seattle, WA – 18th of August, 2015
Credit2: Why?

Complexity:
- in Credit we have:
  - credits and weights
  - 2 priorities
  - oh, actually, it’s 3
  - active and non active state of vCPUs
  - flipping between active/non-active means flipping between burning/non-burning credits, which in turns means wandering around among priorities

- in Credit2 we have:
  - credits burned basing on weights
Credit2: Why?

Complexity:

▶ in Credit we have:
  ▶ credits and weights
  ▶ 2 priorities
  ▶ oh, actually, it’s 3
  ▶ active and non active state of vCPUs
  ▶ flipping between active/non-active means flipping between burning/non-burning credits, which in turns means wandering around among priorities

▶ in Credit2 we have:
  ▶ credits burned basing on weights
  ▶ that’s it
Credit2: Why?

Complexity:

- in Credit we have:
  - credits and weights
  - 2 priorities
  - oh, actually, it’s 3
  - active and non active state of vCPUs
  - flipping between active/non-active means flipping between burning/non-burning credits, which in turns means wandering around among priorities

- in Credit2 we have:
  - credits burned basing on weights
  - that’s it
  - no, really, it’s that simple! :-)}
Credit2: Why? (cont.)

Scalability:
Credit2: Why? (cont.)

Scalability:

▶ in Credit
  ▶ periodic runqueue sorting. *Freezes* a runqueue
  ▶ periodic accounting. *Freezes* the whole scheduler!
Credit2: Why? (cont.)

Scalability:

- in Credit
  - periodic runqueue sorting. *Freezes* a runqueue
  - periodic accounting. *Freezes* the whole scheduler!

- in Credit2 we have:
  - “global” lock only for load balancing
    (looking at improving it)
Credit2: What’s Missing?

- SMT awareness (done, missing final touches)
- hard and soft affinity support (someone working on it)
- tweaks and optimization in the load balancer (someone looking at it)
- cap and reservation (!!!)
Credit2: What’s Missing?

- SMT awareness (done, missing final touches)
- hard and soft affinity support (someone working on it)
- tweaks and optimization in the load balancer (someone looking at it)
- cap and reservation (!!!)

Plan: mark it as experimental for 4.6
Credit 2: What’s Missing?

- SMT awareness (done, missing final touches)
- hard and soft affinity support (someone working on it)
- tweaks and optimization in the load balancer (someone looking at it)
- cap and reservation (!!!)

Plan: mark it as !experimantal for 4.6
**New Plan:** mark it as !experimental for 4.7 :-P
Credit2: Performance Evaluation

Time for building Xen:
Credit2: Performance Evaluation

Time for building Xen:
- 16 pCPUs host
- 8 vCPUs guest
Credit2: Performance Evaluation

Time for building Xen:

- 16 pCPUs host
- 8 vCPUs guest
- varying # of build jobs → # of vCPUs busy in the guest
  - -j4, -j8, -j (unlimited)
Credit2: Performance Evaluation

Time for building Xen:

- 16 pCPUs host
- 8 vCPUs guest
- varying # of build jobs → # of vCPUs busy in the guest
  - -j4, -j8, -j (unlimited)
- varying the interference load → # of vCPUs active in Dom0
  - nothing, 4 CPU hog tasks, 12 CPU hog tasks
Credit2: Performance Evaluation

Time for building Xen:

- 16 pCPUs host
- 8 vCPUs guest
- varying # of build jobs $\rightarrow$ # of vCPUs busy in the guest
  - -j4, -j8, -j (unlimited)
- varying the interference load $\rightarrow$ # of vCPUs active in Dom0
  - nothing, 4 CPU hog tasks, 12 CPU hog tasks

Anticipation:
Credit2: Performance Evaluation

Time for building Xen:
- 16 pCPUs host
- 8 vCPUs guest
- varying # of build jobs → # of vCPUs busy in the guest
  - -j4, -j8, -j (unlimited)
- varying the interferece load → # of vCPUs active in Dom0
  - nothing, 4 CPU hog tasks, 12 CPU hog tasks

Anticipation:
- Some tweaks still missing, but really promising!
Credit2: Performance Evaluation (cont.)
Credit 2 with HyperThreading support FTW (a.k.a: we’re done here, let’s go have beers!! ;-P)
Credit2: Performance Evaluation (cont. II)

Xen Build Time (low interference)
8 vCPUs guest, 16 pCPUs host

- "j4"
- "j8"
- "j"

# of build jobs

seconds

Credit1
Credit2
Credit2 - HT

Almost good, apart from "-j4". It should be consistent with "-j4" of "no interf" (a.k.a.: ok, maybe just one beer?)
Almost good, apart from "-j4". It should be consistent with "-j4" of "no interf" (a.k.a.: ok, maybe just one beer?)
Credit2: Performance Evaluation (cont. III)

Xen Build Time (high interference)
8 vCPUs guest, 16 pCPUs host

- Credit1
- Credit2
- Credit2 - HT

seconds
60 50 40 30 20 10 0

"j4"  "j8"  "j"

# of build jobs

Really good against Credit1, but "-j4" still has issues (a.k.a.: ok, I got it. Let's grab some coffee and get back to work!)

Seattle, WA – 18th of August, 2015
Really good against Credit1, but "-j4" still has issues (a.k.a.: ok, I got it. Let’s grab some coffee and get back to work!)
Conclusions
Conclusions

Having our scheduler and being able to tweak it to our needs is very important:
Having our scheduler and being able to tweak it to our needs is very important:

- we should continuously try assess where we stand (performance wise, scalability wise, etc.)
Having our scheduler and being able to tweak it to our needs is very important:

- we should continuously try assess where we stand (performance wise, scalability wise, etc.)
- we should always strive to do better!
Thanks again,

Questions?
Spare Slides
(Better) Exploiting Intel CMT

Biggest limitations:

- num. of activities that can be monitored is limited
- applies to L3 cache only, for now
(Better) Exploiting Intel CMT

Biggest limitations:

▷ num. of activities that can be monitored is limited
▷ applies to L3 cache only, for now

What would be desirable:

▷ per-vCPU granularity ⇒ No! Too few monitoring IDs
▷ L2 occupancy/bandwidth stats, for helping intra-socket scheduling decisions ⇒ No! Only L3
(Better) Exploiting Intel CMT

Biggest limitations:
- num. of activities that can be monitored is limited
- applies to L3 cache only, for now

What would be desirable:
- per-vCPU granularity $\Rightarrow$ No! Too few monitoring IDs
- L2 occupancy/bandwidth stats, for helping intra-socket scheduling decisions $\Rightarrow$ No! Only L3

What I’m thinking to:
- use one monitoring ID per pCPU. This gives:
  - how ‘cache hungry’ a pCPU is being
  - how much free cache there is on each socket/NUMA node
- sample periodically and use for mid-level load balancing decisions
- ... ideas welcome!!
(Better) Exploiting Intel CAT

Basic idea:
(Better) Exploiting Intel CAT

Basic idea:

▶ if you’re going to run on pCPUs #x, #y and #z...
▶ ...you better have some cache space reserved there!

Implementation:
(Better) Exploiting Intel CAT

Basic idea:

- if you’re going to run on pCPUs #x, #y and #z...
- ...you better have some cache space reserved there!

Implementation:

- when setting affinity, set CAT accordingly

Open ”issues”:
(Better) Exploiting Intel CAT

Basic idea:
- if you’re going to run on pCPUs #x, #y and #z...
- ...you better have some cache space reserved there!

Implementation:
- when setting affinity, set CAT accordingly

Open ”issues”:
- always?
- for hard affinity, or soft affinity, or both?
- with what mask (i.e., how to split the cache)?
- what about the vice-versa?
Driver Domain Aware Scheduling

Suppose:

- vCPU x is top priority (higher credits, whatever)
- vCPU x issues an I/O operation. It has some remaining timeslice (or credit, or whatever) available, but it blocks waiting for results
- some other domains’ vCPUs y, w and z have higher priority than I/O’s vCPUs (Dom0 or driver domain)
Suppose:

- vCPU $x$ is top priority (higher credits, whatever)
- vCPU $x$ issues an I/O operation. It **has some remaining timeslice (or credit, or whatever) available**, but it blocks waiting for results
- some other domains’ vCPUs $y$, $w$, and $z$ have higher priority than I/O’s vCPUs (Dom0 or driver domain)

Schedule: $v_x$, $v_y$, $v_w$, $v_z$, $v_{drv\_dom}$ $\rightarrow$ only now $v_x$ can resume
Driver Domain Aware Scheduling (cont.)

What if, $v_x$ could *donate* its timeslice to the entity that is blocking it?
Driver Domain Aware Scheduling (cont.)

What if, $v_x$ could donate its timeslice to the entity that is blocking it?

Schedule: $v_x, v_{drv-dom}, v_x, v_w, v_z \rightarrow v_x$ unblocks right away (this, assuming servicing I/O to be quick, and does not even exhaust $v_x$ timeslice)
What if, $v_x$ could donate its timeslice to the entity that is blocking it?

Schedule: $v_x, v_{drv\_dom}, v_x, v_w, v_z \rightarrow v_x$ unblocks right away (this, assuming servicing I/O to be quick, and does not even exhaust $v_x$ timeslice)

- avoids priority inversion (no, we’re not the Mars Pathfinder, but still...)
- makes $v_x$ sort of “pay”, from the CPU load it generates with its I/O requests (fairness++)