TECHNOLOGY BRIEF

# Smarter CPU Pinning in OpenStack* Nova

## Introduction

Virtualization promotes high resource/equipment usage. Virtualization allows for the kind of fast provisioning that large-scale applications need. Virtualization improves reliability immeasurably. I think we can all agree that virtualization is great. Unfortunately, virtualization-based solutions tend toward lower performance than equivalent bare metal-based solutions. Technologies such as Intel® Virtualization Technology for IA-32, Intel® 64, and Intel® Architecture (Intel® VT-x) and Intel® Virtualization Technology for Directed I/O (Intel® VT-d), can reduce this performance penalty, but bare metal will often outperform virtualization. Without careful deployment configurations, virtualization-based solutions may also tend toward non-determinism: one can ask for something to be done and it will be done; but one cannot generally say when it will be done. For most workloads, the benefits that virtualization brings in areas of improved scale and lowered cost significantly outweigh any performance impact. Similarly, any potential indeterminism is likely to go unnoticed. However, it can impact concepts like Network Functions Virtualization (NFV) where virtualized network functions running on industry-standard, high-volume servers are expected to replace proprietary hardware. For such latency-sensitive applications, the recommendations outlined later should be considered.

## Enhanced Platform Awareness

There are many ways to deliver high-performance, deterministic virtualized solutions. Many modern Intel® processor-based server platforms include hardware capabilities to improve the performance of virtualization, along with specific tasks such as crypto or networking. Exposing these features in a virtualization-based environment, such as cloud deployments, is an important first step in addressing performance and latency concerns. To this end, Intel and the community have been working on driving “Enhanced Platform Awareness” extensions in OpenStack* to facilitate higher performing, more efficient workloads in virtualized deployments. These extensions work by exposing information about host functionality to hypervisors and virtualized guests alike. However, there is far more that can be done.

## NUMA Awareness

Modern, multi-socket x86 systems use a shared memory architecture that describes the placement of the main memory modules with respect to processors in a multiprocessor system. In a NUMA-based system, each processor has its own local memory controller that it can access directly with a distinct performance advantage. At the same time, it can also access memory belonging to any other processor using a shared bus (or some other type of interconnect), but with different performance characteristics. This topology information is generally available to host operating systems and other host applications, which can use it to take advantage of the performance improvements “NUMA alignment” offers. However, until recently this information was not available to the controller and therefore could not be considered when scheduling OpenStack instances. As part of the Juno and Kilo releases, work was undertaken in Nova to allow the controller to “understand” NUMA and thus allow it to allocate resources more efficiently.

## CPU Pinning

Awareness of the underlying architecture is a necessity for implementing features such as "CPU pinning." Support for the "CPU Pinning" functionality was also added to OpenStack Nova in the Juno release and was further refined in the Kilo release. In short, the feature allows a user to take the "virtual CPUs (vCPUs)" used by a guest and tie them to the real "physical CPUs (pCPUs)" of the host. This allows the controller to direct the host to dedicate some of the many cores in the Symmetric Multi-Processing (SMP) enabled host to a guest, preventing resource contention with other guest instances and host processes. As a result, CPU pinning can dramatically improve the performance of guests and the applications they run. However, while CPU pinning implicitly provides NUMA topology awareness to the guest, it does not provide any awareness of another technology that can impact performance: Simultaneous Multi-Threading (SMT), or Intel® Hyper-Threading Technology on Intel® platforms.


**Figure 1**. Without CPU Pinning. vCPUs are "floating" across host cores.


**Figure 2**. With CPU Pinning: vCPUs are tied to pCPUs.

## CPU Thread Pinning

The thread pinning feature provides SMT-awareness, in addition to the existing NUMA topology-awareness. SMT is different from SMP in that SMT "cores" share a number of components with their "thread sibling(s)" (SMP cores share only buses and some memory). CPU Thread Pinning differs from CPU Pinning in how pCPU-vCPU bindings are chosen. With the latter, pCPUs are chosen randomly or linearly. No consideration is given to the differences in performance between SMP cores and SMT thread siblings. Thread Pinning adds this functionality and allows a user to choose a host with or without this feature.

CPU Thread Pinning provides awareness thread siblings to the scheduler. There are four possible configurations that cater to different requirements and workloads. As stated above, so-called "thread siblings" includes a number of components. In some cases, this sharing may not matter and may even benefit performance, thus thread siblings are used (the *require* case). Other workloads may see performance impacts due to this contention so non-thread siblings are used (the *separate* case). Yet more workloads may want to not only avoid the use of thread siblings but also prevent other guests (or workloads) from using the siblings (the *isolate* case). Finally, some workloads may want to avoid any host that supports SMT entirely (the *avoid* case).


**Figure 3**. The "require" case.

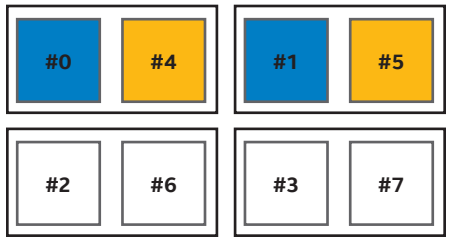


**Figure 4**. The "separate" case.

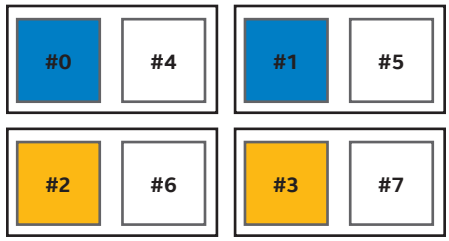


**Figure 5**. The "isolate" case.

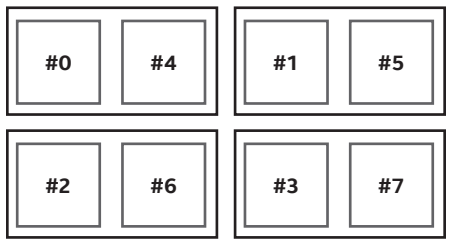


**Figure 6**. The "avoid" case. Note how no guests are scheduled on this SMT-enabled system.

This feature was part of the original CPU pinning blueprint but was not implemented. As a result, a new blueprint was proposed specifically to fill this gap, and code was later submitted to implement this. This implementation works by utilizing a combination of modifications to Nova filters and modifications to the current libvirt-scheduling code in Nova. The former are used to provide per-host checks for SMT support (or lack thereof): the *require* case ostensibly requires SMT on the host, while the *avoid* case forbids it. Filters are used to ensure that unsuitable hosts are quickly ruled out. The latter forms the bulk of the change, implementing "fitting algorithms" that fit a requested guest topology into a host topology (which comes replete with information on the various host cores and their siblings). This code was submitted for the Mitaka release.

## How to Use It?

The CPU thread pinning feature is not part of the Liberty release. As a result, the code cannot yet be checked out from master or one of the stable branches. Instead, the code should be checked out directly from Gerrit. Once done, enabling CPU thread pinning requires enabling CPU pinning and one additional command. As such, many of the commands below will be similar to those described in the original CPU pinning blueprint.

To validate this feature, an SMT-capable node is required. The instructions below validate using a single, "all-in-one" node but can be adjusted for different configurations. SMT must be enabled on the node(s) to test the *separate, isolate,* and *prefer* cases; it should be disabled for the *avoid* case. This can be confirmed by running `lscpu` on the node.

```
$ lscpu | grep -i -E
"^CPU\(s\):|core|socket"

CPU(s):                8
Thread(s) per core     2
Core(s) per socket:    4
Socket(s):             1
```

In this example, the platform uses a quad-core (four cores per socket), single-socket, SMT-enabled (two threads per core) processor. Given that this requirement has been satisfied, the next step is to create a new flavor. This flavor should be used for all VMs that you want to be pinned. A single new flavor is created below, but it is possible to modify existing flavors or create multiple new flavors as desired.

```
$ nova openstack flavor create
--ram 2048 --vcpu 4 pinned
```

This flavor should be modified to include the required metadata for both the CPU policy and CPU threads policy. The *require* case is demonstrated here, but it is possible to experiment with other cases.

```
$ nova flavor-key pinned set
hw:cpu_policy=dedicated

$ nova flavor-key pinned set
hw:cpu_threads_policy=require
```

Finally, an instance should be booted using this flavor. Adjust the image name as appropriate.

```
$ openstack server create
--image cirros-0.3.2-x86_64-uec
--flavor pinned test_pinned_
vm_a --wait
```

Let this instance boot. Once booted, you can use `virsh` on the node to validate that the placement has occurred correctly.

```
$ virsh list
Id    Name                State
----------------------------
 1    instance-00000001   running
$ virsh dumpxml 1
  ...
  <cputune>
   <vcpupin vcpu='0' cpuset='0'/>
   <vcpupin vcpu='1' cpuset='4'/>
   <emulatorpin cpuset='0,4'/>
  </cputune>
  ...
```

In this case, libvirt has not only pinned the vCPUs to pCPUs, but it also has pinned these vCPUs on thread siblings. This is the expected behavior. Attempting to boot a second instance will result in a similar output but with different cores used.

```
$ openstack server create
--image cirros-0.3.2-x86_64-uec
--flavor pinned test_pinned_
vm_a –wait
$ virsh dumpxml 2
  ...
  <cputune>
   <vcpupin vcpu='0' cpuset='1'/>
   <vcpupin vcpu='1' cpuset='5'/>
   <emulatorpin cpuset='1,5'/>
  </cputune>
  ...
```

There is no overprovisioning for pinned instances, therefore it is possible to boot two more instances on this particular platform before consuming all available resources. Note that unpinned instances will not respect pinning and may utilize these CPUs. Host aggregates should be used to isolate the "high performance" hosts used for pinned instances from the "general use" hosts used for unpinned instances.

## Summary

The CPU thread pinning feature is available for anyone to check out and experiment with right now. Comments and other feedback are welcome. This feature should further help users deploying high-performance, highly scalable workloads on OpenStack-powered clouds. Stay tuned for more information.

 1115/JL/PDF 333517-001US