Scheduling and priorities

This page handle the scheduling policy of processes between the different CPUs available on a SEAPATH hypervisor

SEAPATH default CPU isolation

SEAPATH aims to host virtual machine with real time needs. To achieve that, process scheduling must be tuned in order to offer the best performance to the VM.

On SEAPATH, at configuration time with Ansible, a list of isolated CPUs is defined. On these CPUs, the kernel will avoid running

- system process
- user process
- IRQs

These CPUs will then be entirely dedicated to running real time processes like virtual machines.

> In the Ansible inventory of the hypervisors, these CPUs are defined by the `isolcpus` variables.

Tuned

The Debian version of SEAPATH uses tuned (https://github.com/redhat-performance/tuned)

This software configures different variables on the system in order to tune it for real-time virtualization performances. Among others :

- Add command line argument to isolate CPUs (as described above)
- Configure RT priority for kernel threads (ktimer,ksoft ...)
- Configure IRQs on isolated cores (with irqbalance)
- Configure sysfs workqueue on isolated cores

Find a list of all tuned scheduling modification at the end of this page.

On Yocto, tuned is not used. Instead, all these configurations are done at compile time.

Scheduling virtual machines

SEAPATH virtual machines are managed by Qemu.

On a hypervisor, when launching a VM, Qemu create different threads. They can be dispatched in two categories :

- Virtual CPUs threads : These threads emulate the CPUs of the virtual machines. Each CPU inside the virtual machine is handled by one vCPU thread on the hypervisor.
- Management threads : They are responsible for creating the VM, check if it crashes, manage the IO etc …

By default, all these threads will be managed by the Linux scheduler and thus run on the non isolated cores. But they can also be pinned to specific CPUs, what forced them to run on it.

Standard virtual machines

For a VM without any performance or real time needs, it is no use to handle any of the Qemu threads a particular way :

- All threads will inherit a default priority and scheduling type (TS 19)
- All threads will be handled by the Linux scheduler on the non isolated cores

Real time virtual machines

For a VM where performance and determinism is needed, here are our recommendations :

The vCPU threads are where the work of the VM will run, they then must have the most CPU time as possible. To achieve that, they should be

- Scheduled with a real time scheduler (FIFO) and real time priority (42)
- Put alone on an isolated core

> Each vCPU thread must be put **alone** on an isolated CPU. It is counterproductive to put two vCPU thread on the same core. This means you must have at least as much isolated cores as RT VM vCPUs.

The vCPU scheduler type as to be FIFO (FF). A Real Time priority of one is enough.

For more information read page Virtual machines on SEAPATH.

Finer control with cgroup (optional)

Implementation in SEAPATH

The Linux kernel uses cgroups in order to isolate processes. These cgroups work in a hierarchy where each layer restrains the resources a process can access too. Systemd also uses this mechanism by grouping his processes in slices.

By default, service and scope units are placed in the system slice, virtual machines and containers are found in the machine slice and user sessions in the user slice (more details here).

These slices can run on any cores of the hypervisor, but SEAPATH offers a way to restrict the CPUs where each of these slices can execute.

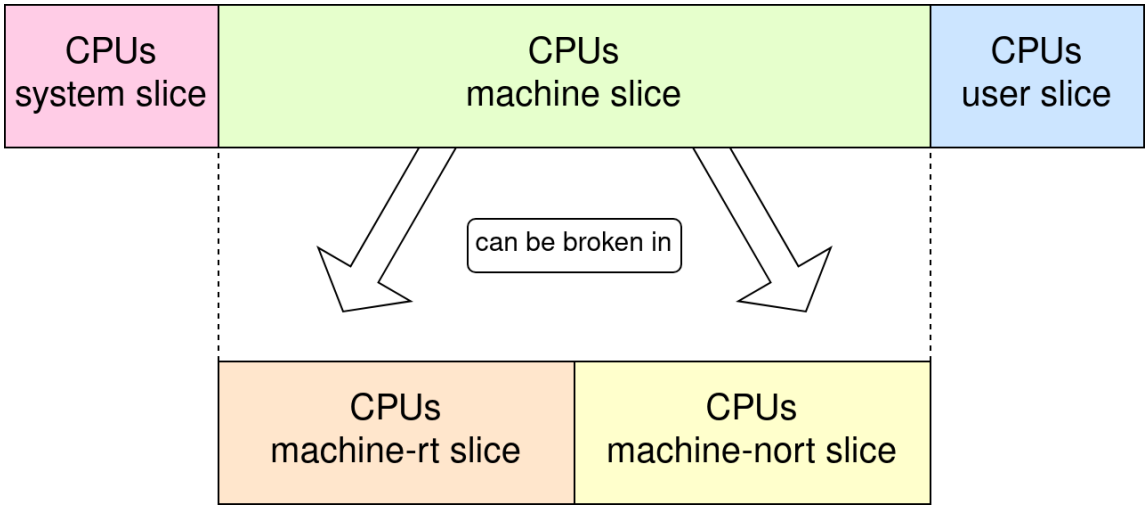
> This is configured using cpusystem, cpumachines and cpuuser in Ansible

The project also defines 3 others slices to separate the functionalities. These slices are optionals

- machine-nort: a subgroup of machine slice to run all virtual machines with the default scheduler.
- machine-rt: a subgroup of machine slice to run all virtual machines with real-time scheduler.
- ovs: a group to run OpenVSwitch processes.

> These are configured by cpumachinesnort, cpumachinesrt and cpuovs in Ansible.

> Because of the hierarchy, all CPUs used in machine-rt and machine-nort slices must also be used in the machine slice.



> Different slices can share the same CPUs. In a machine with few cores for example, it can be useful to put system, user and ovs slices on the same CPUs.

TODO : put the link to the inventories README once written

Utility of slices CPU isolation

Using these slices is useful to get a preset of CPU isolation for virtual machines. When placing a VM in either machine-rt or machine-nort slice it will be automatically scheduled on the CPUs of this slice.
It seems particularly useful when deploying many VMs at once.

One really important thing to have in mind when using these slices is that the Qemu management threads of the virtual machines will be part of the machine slice (resp machine-rt and machine-nort), and not the system slice. This means that this threads will not be executed in the CPUs associated with the Linux system, but on the CPU chosen for the machine slice.

Using this slices allows :

- Having one more step in VM isolation
- Avoiding CPU hardware attacks like Spectre or Meltdown

This new isolation layer protects from really advanced attacks. Because it has drawbacks (see below), the question remains open if you should or not activate this feature.

Drawbacks

By activating CPU isolation on the machine slice, the management threads of the VM will be scheduled on the allowed CPU list of the slice. This new mechanism implies two things :

- You must have one more CPU on the machine-rt slice. Because every vCPU thread needs to be scheduled on its own CPU, one more is needed to schedule management threads.
- You must carefully place the management threads. By default, they will be scheduled on the first allowed CPU of the slice. If this CPU also runs an RT vCPU threads, it will prevent the management thread to run and the VM will never boot.

The management thread scheduling is handled by the `emulatorpin` field in libivrt XML.

For more information, read page Virtual machines on SEAPATH.

Specific configurations

NUMA

NUMA (Non-Uniform Memory Access) refers to machines that have the ability to contain several CPU sockets. Each of these sockets has its own cache memory, which means that accessing memory from one socket to another is much slower than accessing memory on its own socket.

To know if a SEAPATH hypervisor has a NUMA architecture, use the command `virsh nodeinfo` . It gives, among others, the number of NUMA cells in the system.
To know the architecture of CPUs in the system, launch the command `virsh capabilities`. It provides a section “topology” with a description of the sockets (here called cells) and the CPUs each one contains. More information about these commands here.

If your system contains more than one NUMA cells, you must be careful to pin all the vCPU threads of one VM **on the same NUMA cell**. Otherwise, the data transfer between two cells will significantly slow down the VM.

Hyper-threading

Most of the modern CPUs support hyper-threading. This option can be enabled in the BIOS and double the number of CPUs available on the system. However, the newly created CPUs are not as fast and independent as classic ones.

Hyper-threading uses the concept of logical cores. A logical core is only the pipeline part of a CPU. It shares an arithmetic unit (ALU) and memory with another logical core to form a physical core.
When hyper-threading is disabled, every physical core runs only one logical core. It then has full access to the memory and the ALU. When hyper-threading is enabled, the two logical cores are enabled. The obvious drawback is that each logical core will influence its sibling.

When running real-time virtual machines, it is **highly recommended to disable hyper-threading.** However, on test systems or systems with fewer cores, it can be an interesting feature. In that case, the RT VM vCPU must be pinned to a logical core where its sibling doesn’t run any process. Otherwise, it will influence the vCPU thread, which will lose determinism.

To know the exact architecture of your CPUs, use the command `virsh capabilities` and watch the “topology” section. The siblings field describes which logical CPUs are grouped together.

On most systems, logical CPUs are grouped in numerical order (0 with 1, 2 with 3 …) but this is not always the case. Always refer to `virsh capabilities` to check the exact architecture.

Annex: list of tuned modifications

Below a list of all scheduling modifications done by tuned.

TODO : explain all ?

- /sys/module/kvm/parameters/halt_poll_ns = 0
  /sys/kernel/ktimer_lockless_check = 1
  /sys/kernel/mm/ksm/run = 2

- Kernel parameters :
  isolcpus=managed_irq,domain,{isolated_cores}
  intel_pstate=disable
  nosoftlockup
  tsc=reliable
  nohz=on
  nohz_full={isolated_cores}
  rcu_nocbs={isolated_cores}
  irqaffinity={non_isolated_cores}
  processor.max_cstate=1
  intel_idle.max_cstate=1
  cpufreq.default_governor=performance
  rcu_nocb_poll
- kernel thread priorities :
  group.ksoftirqd=0:f:2:*:^\[ksoftirqd
  group.ktimers=0:f:2:*:^\[ktimers
  group.rcuc=0:f:4:*:^\[rcuc
  group.rcub=0:f:4:*:^\[rcub
  group.ktimersoftd=0:f:3:*:^\[ktimersoftd
- configures irqbalance with isolated_cores list
- configures workqueue with isolated_cores list
- kernel.hung_task_timeout_secs = 600
  kernel.nmi_watchdog = 0
  kernel.sched_rt_runtime_us = -1
  vm.stat_interval = 10
  kernel.timer_migration = 0