SMT-aware CPU Manager

or: thread allocation policies for cpumanager
or: how do we extend cpumanager?

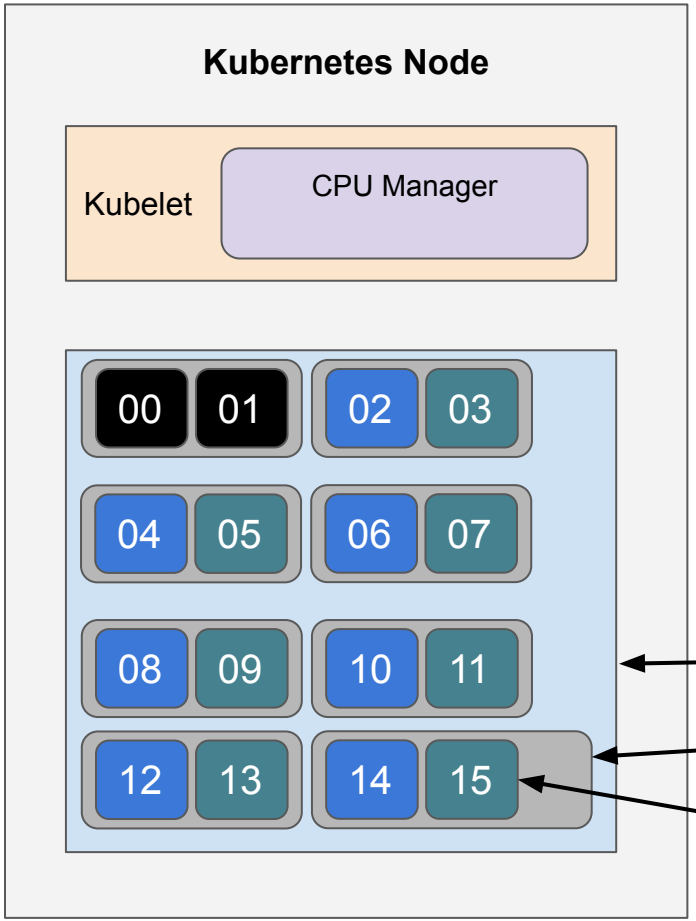
@fromani (fromani@redhat.com) @swatisehgal (swsehgal@redhat.com)

Sig-node weekly, April 20, 2021

Agenda

Thread allocation control

- Some applications require more isolation, at HW thread level
  - Latency-sensitive applications (DPDK, RT)
  - Mitigate cache-based side channel attacks
- Enabling finer-grained thread allocation enables better container density
  - Easier and safer for RT/non-RT (infra) pods to coexist
- Similar capabilities are already present in OpenStack
- Also implemented by the cpu-pooler



For simplicity sake: let's consider a node, with 1 CPU, 2-way SMT capable, with 8 physical cores, and (8*2=)16 virtual cores.

We will use: "virtual core", "hw thread" as synonyms

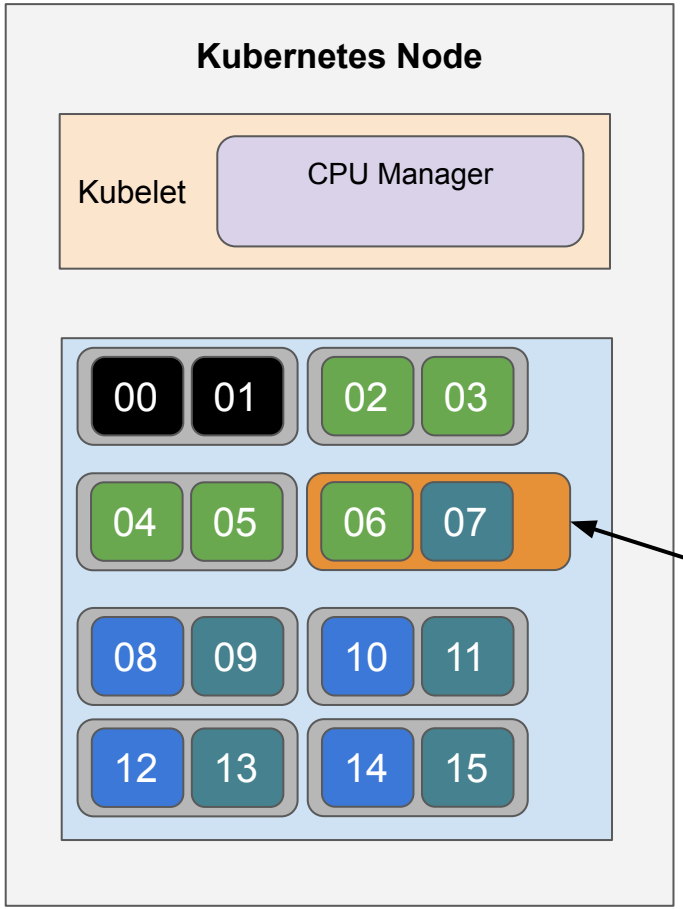
Reserved (--reserved-cpus) unavailable for workloads

CPU package

HW thread pair (physical core - 2-way SMT)

HW thread (virtual core)

```
apiVersion: v1
kind: Pod1
Spec:
  containers:
  - name: nginx
image: nginx
resources:
        limits:
          memory: "256Mi"
          cpu: "5"
        requests:
          memory: "256Mi"
          cpu: "5"
```



Reserved core

Virtual Core allocated to Pod1

Potential for noisy neighbour!

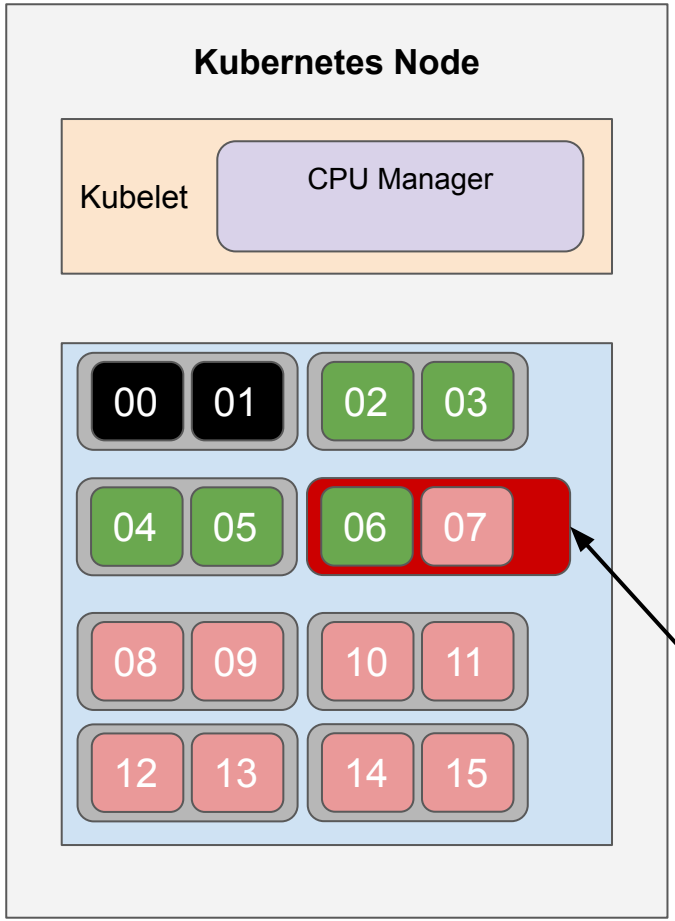
When different containers run on the same physical cpu

HW thread share some silicon (part of execution units, L2 cache...) - so even **non malicious** containers interfer to each other.

Red Hat

```
apiVersion: v1
kind: Pod1
Spec:
  containers:
  - name: nginx
image: nginx
resources:
        limits:
          memory: "256Mi"
          cpu: "5"
        requests:
          memory: "256Mi"
          cpu: "5"
```

```
apiVersion: v1
kind: Pod2
Spec:
  containers:
  - name: nginx
image: nginx
resources:
        limits:
          memory: "256Mi"
          cpu: "9"
        requests:
          memory: "256Mi"
          cpu: "9"
```



Reserved core

Virtual Core allocated to Pod1

Virtual Core allocated to Pod2

Actual noisy neighbour!
Two container share the same physical core!

HW thread share some silicon (part of execution units, L2 cache...) - so even **non malicious** containers interfer to each other.

Red Hat

Proposal: add a new CPU Manager Policy to make it more SMT-aware

smtaware - new policy with minimal changes, to prevent noisy neighbours.

Equivalent approach: extend the existing static policy
But the workload needs a clear way to opt-in (just a new kubelet option?)

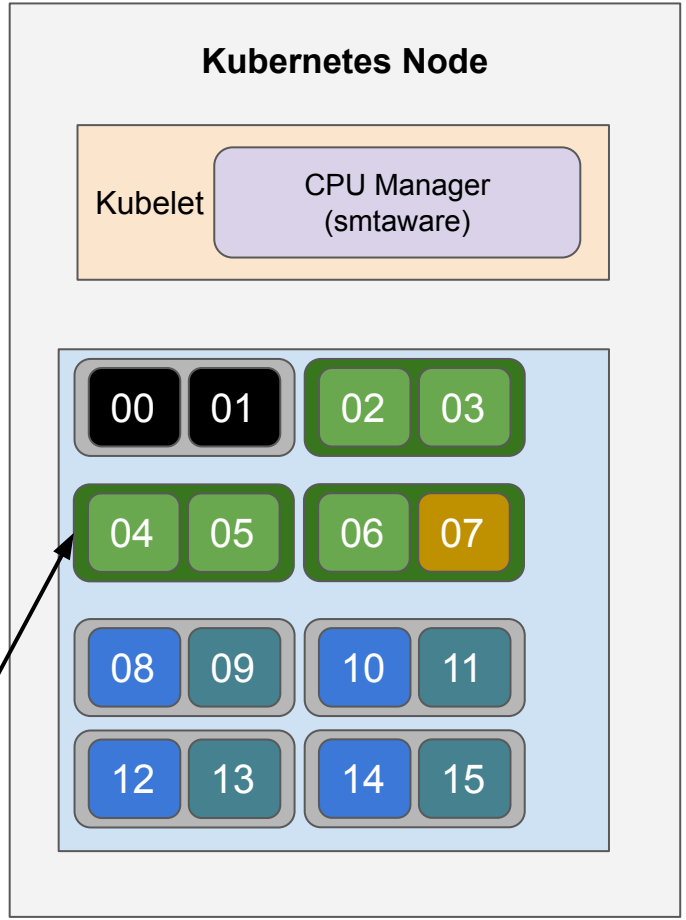
Coming next: another policy to emulate non-SMT on SMT machines - not a priority for us in this cycle.

smtaware policy: desired behaviour

```
apiVersion: v1
kind: Pod1
Spec:
  containers:
  - name: nginx
image: nginx
resources:
        limits:
          memory: "256Mi"
          cpu: "5"
        requests:
          memory: "256Mi"
          cpu: "5"
```

Ask for 5 cores

Need to reconcile the resource accounting



Reserved core

Virtual Core allocated to Pod1

Physical Core allocated to Pod1

Virtual core accounted to Pod1, but not usable

Always allocate full physical cores

Red Hat

Challenge: resource accounting

For both the new proposed policies, the container will get more virtual cores than what is requesting for its resources

ActualAllocation > Request

Example (2-way SMT, 8 physical cores, 16 cpus):

- Smtaware policy
- Request 5 cpus = round_up(5.0/2.0) = 3 cores
- Kubelet allocates 3 cores = 3 * 2 = 6 cpus

Challenge: resource accounting

Proposed solution: add an admission handler, add and enforce resource requests constraints

1. Admit containers whose cpu resource request is a multiple of SMT level (e.g. multiple of 2 with HT).
   a. Returns **SMTAlignmentError** in case CPU request is not a multiple of SMT level
2. With this requirement enforced, the cpumanager allocation algorithm will guarantee avoidance of physical core sharing
3. **NOTE**: Pods might have to overallocate resources.
4. **Pros**:
   a. Conforms to the design pattern followed by Topology Manager

Challenge: resource accounting

Alternative solution 1: add a new (extended) resource to represent cores (physical cpus)

1. Confusing relationship with existing “cpu” resource
2. **Cons**:
   a. users need to specify two cpu-related resources? Error prone.
   b. pods should be able to opt in the Guaranteed QoS Class, so we need to have the container specify their “cpu” resource.

Challenge: resource accounting

Alternative solution 2: add a new unit to allow to express physical cpus in resource requests

1. Builds on the fact that a physical cpu corresponds to one or more virtual cpus
2. **Cons**:
   a. the relationship between “physical” and “virtual” cpu is not fixed (depends on HW implementation, settings)
3. Such unit would make sense only for CPUs!

First things first: how do we extend cpumanager?

The initial feedback request on sig-node ML spawned a lively discussion about the best way to extend cpumanager.

Challenge: how to implement new policies

1. In tree
   a. Make changes to static policy with an additional flag
   b. Add additional CPU Manager Policy
2. Enable external policies!
   a. Cpumanager plugins?
   b. Cpumanager as device plugin?

Revamped interest in enabling external policies

Much more flexible and extendible approach, aligns nicely with other ongoing initiatives.

(External) policies implementation talking points

1. Resources API - consistent interface, accounting
2. Reconciliation - loop ownership, possible conflicts with built-in cpumanager
3. State ownership (cpu_manager_state) - which component holds it? Just move into the plugins?
4. Cgroups ownership - which component manages them?
5. API - is Device Plugin API good enough?

Discussion thread ongoing on kubernetes-sig-node

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make
Red Hat a trusted adviser to the Fortune 500.


linkedin.com/company/red-hat


youtube.com/user/RedHatVideos


facebook.com/redhatinc


twitter.com/RedHat

Red Hat

Backup/Extra slides

Extended proposal: add a new CPU Manager Policy to emulate non-SMT

smtisolate - new policy to emulate non-SMT behaviour on SMT-enabled machines.

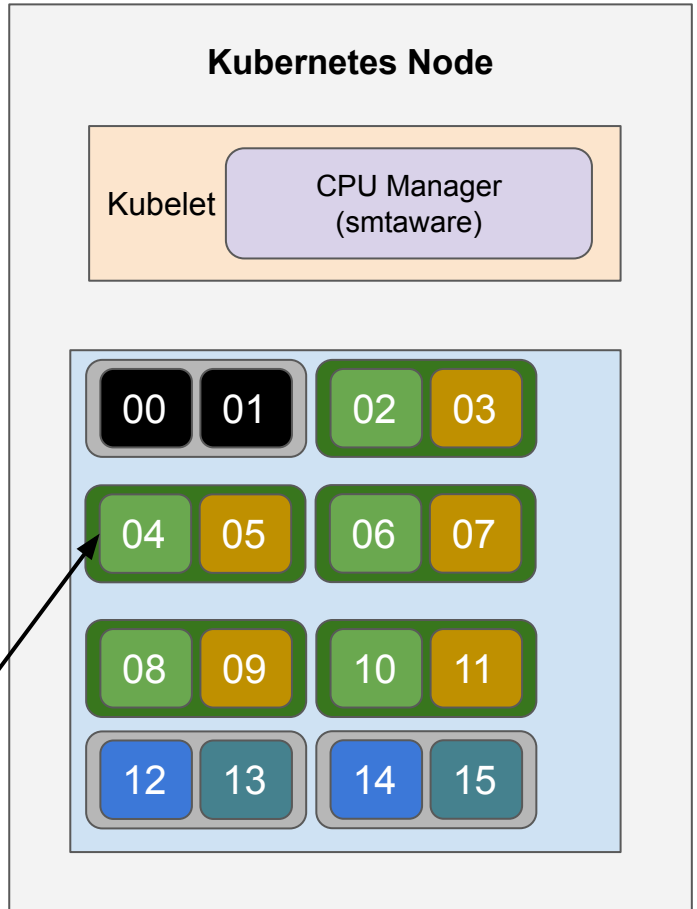
Each cpu granted to containers is guaranteed to be on a different physical cpu.

smtisolate policy: desired behaviour

```
apiVersion: v1
kind: Pod1
Spec:
  containers:
  - name: nginx
image: nginx
resources:
        limits:
          memory: "256Mi"
          cpu: "5"
        requests:
          memory: "256Mi"
          cpu: "5"
```

Ask for 5 cores

Need to reconcile the resource accounting



Reserved core

Virtual Core allocated to Pod1

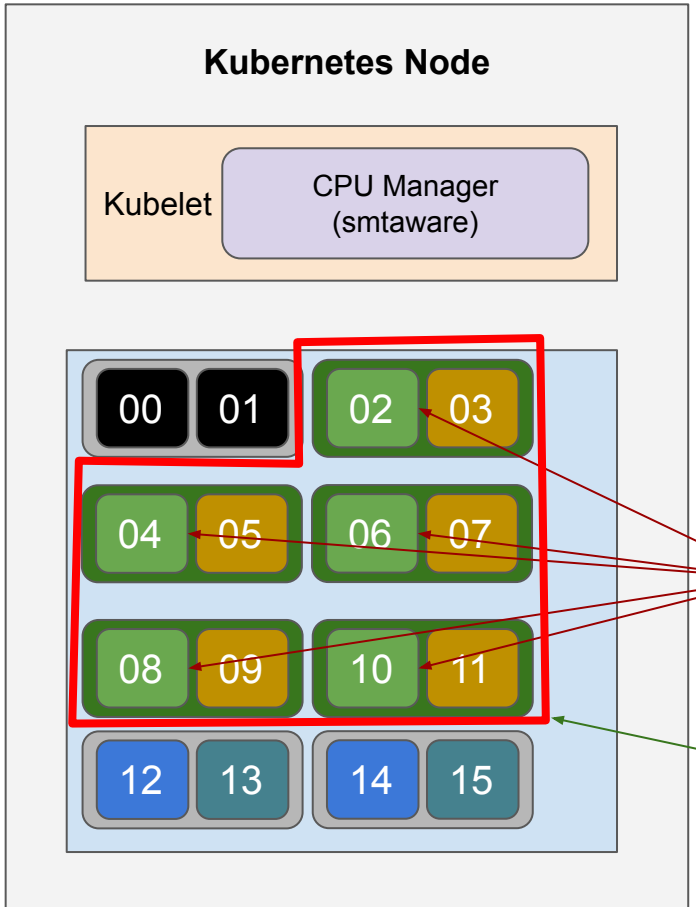
Physical Core allocated to Pod1

Virtual core accounted to Pod1, but not usable

Allocate the requested amount of physical (no longer virtual) cores

smtisolate policy implementation

```
apiVersion: v1
kind: Pod1
Spec:
  containers:
  - name: nginx
image: nginx
resources:
        limits:
          memory: "256Mi"
          cpu: "10"
        requests:
          memory: "256Mi"
          cpu: "10"
```



Reserved core

Virtual Core allocated to Pod1

Physical Core allocated to Pod1

Virtual core accounted to Pod1, but not usable

Previously: a single set of cpus to be assigned to the container and to be removed from the available set

Now: two separate sets

Set#1: Cpus added to the container cgroup (5 cpus)

Set#2: Cpus removed from the available set (10 cpus)