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 13, 2021

Current Behaviour of CPU Manager with static policy

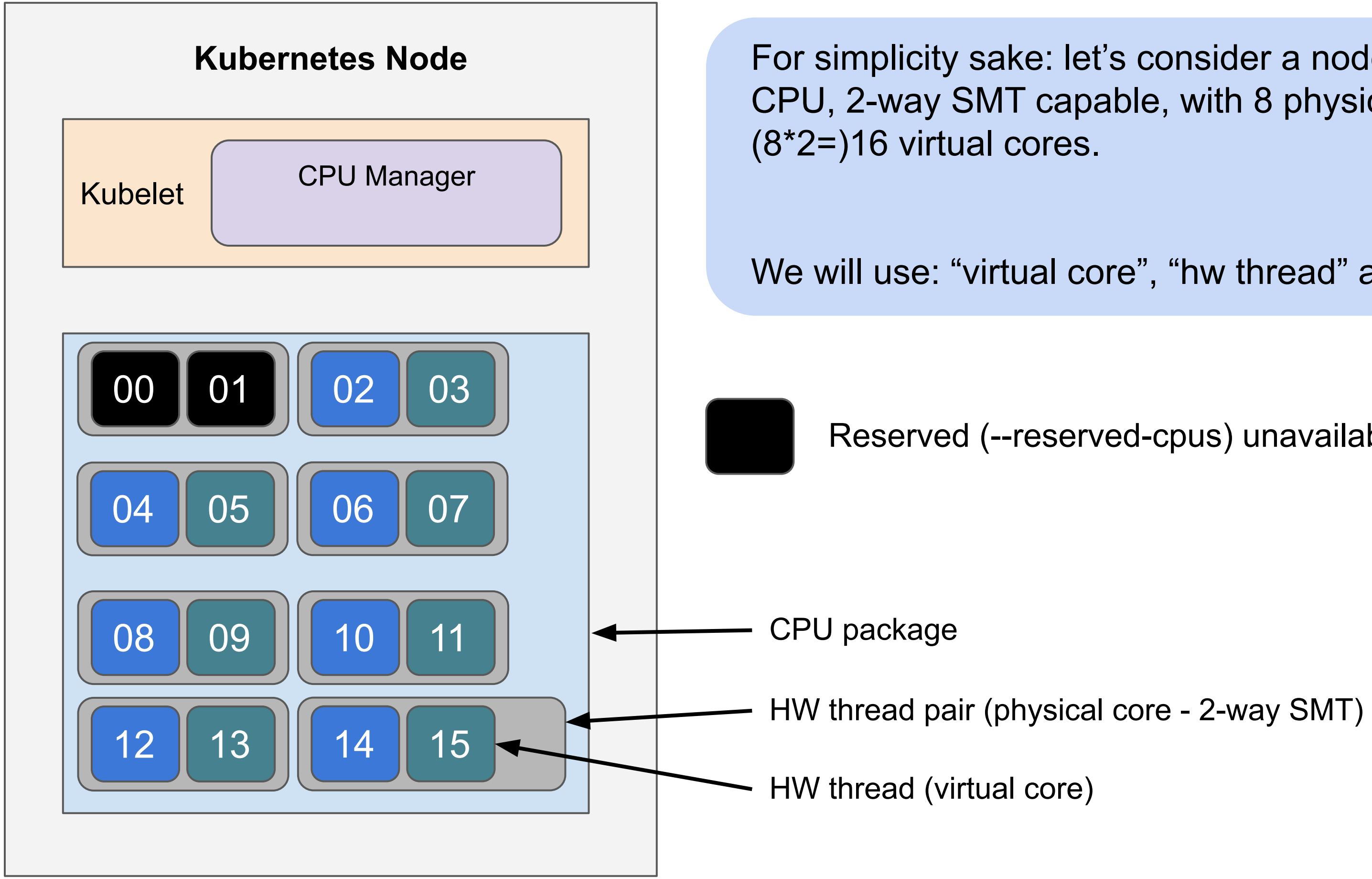
- Containers have access to exclusive CPUs on the node
- On SMT-enabled systems, this means virtual CPUs - aka HW threads
- CPU manager performs topology-aware best fit, allocates:
  - Full sockets, full cores, individual cpus (=HW threads)

Thread allocation control

- Some applications require more isolation, at HW thread level
  - Latency-sensitive applications (DPDK, RT)
  - Mitigate cache-based side channel attacks
- 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



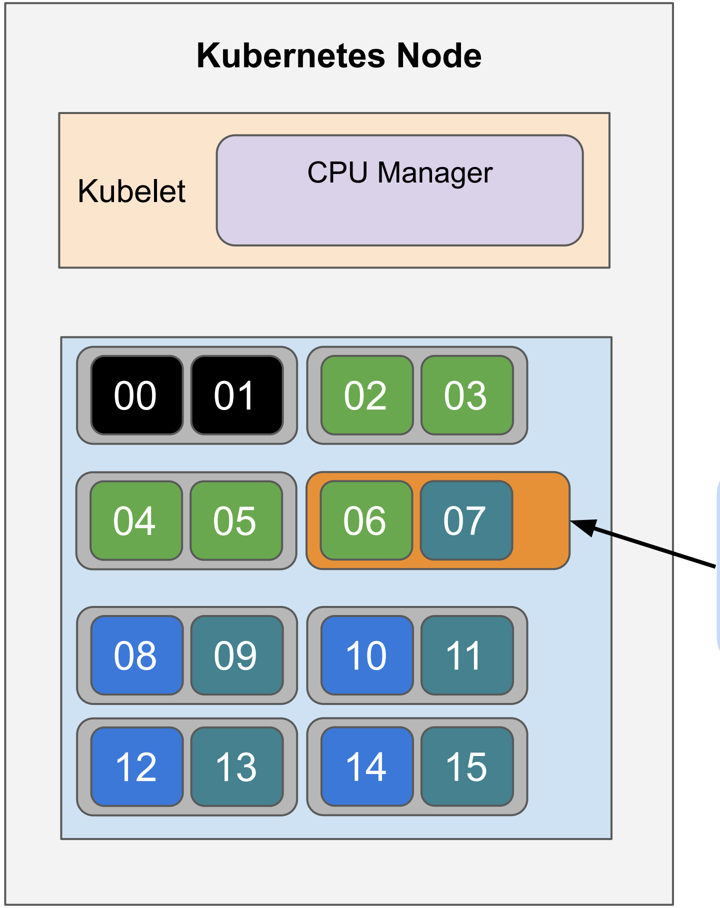
■ Reserved (--reserved-cpus) unavailable for workloads

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

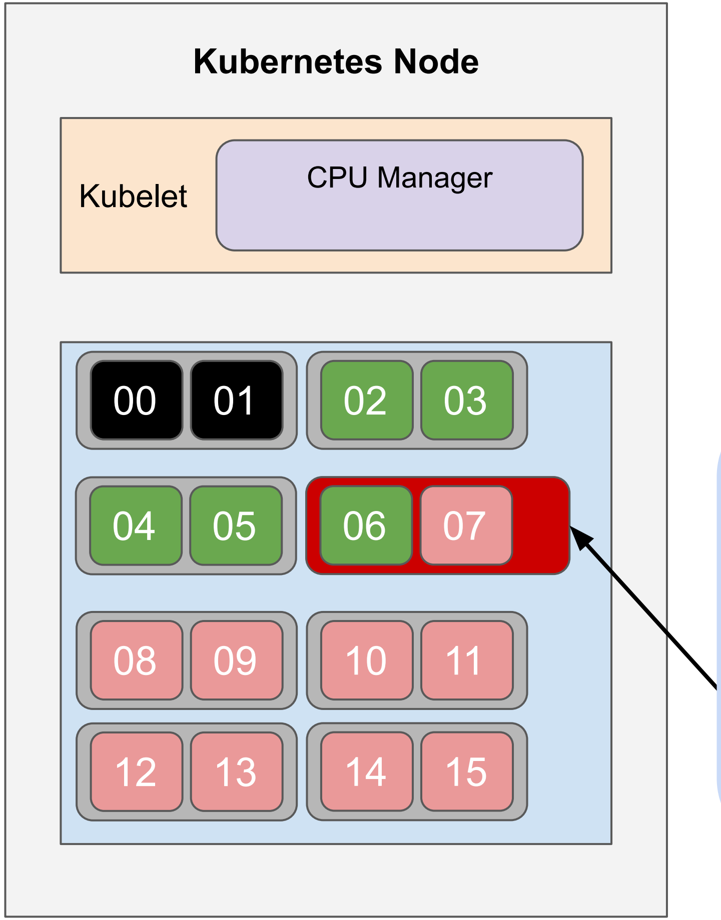
🟩 Virtual Core allocated to Pod1

Potential for noisy neighbour!

When different containers run on the same physical cpu

```
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.

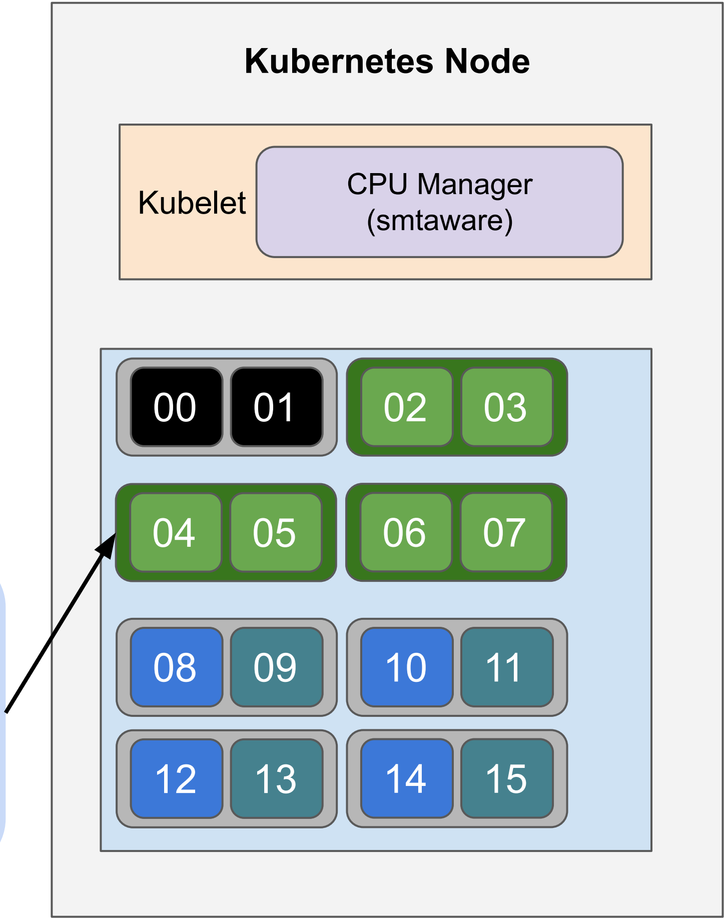
Proposal: add two new CPU Manager Policies to make it more SMT-aware

1. **smtaware** - new policy with minimal changes, to prevent noisy neighbours. Works best with some workload cooperation.

2. **smtisolate** - new policy to emulate no-smt on smt-enabled machines. Works transparently with any workload.

smtaware policy

```
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

▩ **Physical** Core allocated to Pod1

Ask for 5 cores
Get 6 **virtual** cores

We need to reconcile the resource accounting

Always allocate full physical cores

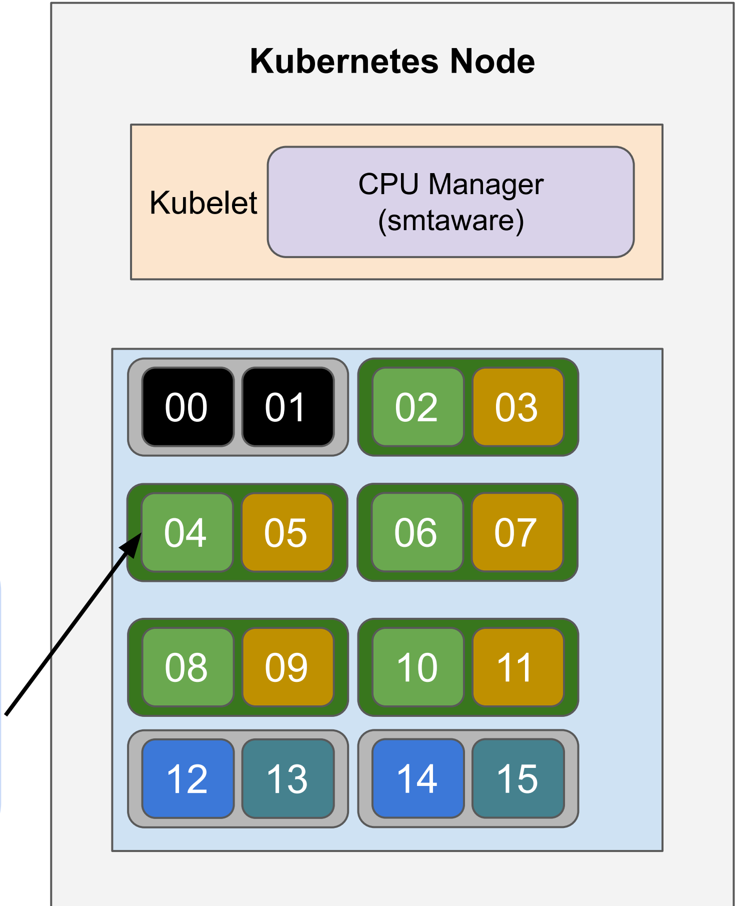
Round up allocated physical cores to prevent any possible noisy neighbours

smtisolate policy

```
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
Get 10 **virtual** cores

We 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

Guaranteed QoS

We would very much like to make sure pods still are in the Guaranteed QoS Class, because:

1. Consistency
2. Principle of least surprise

So we need to have the container specify their "cpu" resource.

Challenge: resource accounting

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

Allocation >= 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

Possible solutions:

Do nothing! Trust the system reconciliation process

1. Different reconciliation frequency between nodes and pods - nodes updated less frequently.
2. Not ideal: window for bad scheduling decisions due to stale/inconsistent data

Challenge: resource accounting

Possible solutions (cont.):
Add a new (extended) resource to represent cores (physical cpus)

1. Confusing relationship with existing “cpu” resource
2. Not ideal: users need to specify two cpu-related resources? Confusing, error prone.

Challenge: how to implement new policies

1. In tree
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