SUSE

NAIST

KVM FORUM 2021

KubeVirt and the Cost of Containerizing VMs

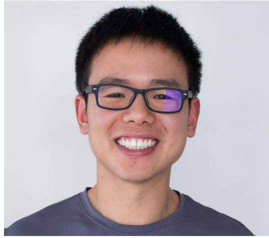
Guoqing Li, Nara Institute of Science and Technology, Japan

Dario Faggioli, SUSE, Italy

Vasiliy Ulyanov, SUSE, Germany

Self Introductions

Who we are, what we do...



Guoqing Li

Master's Student

Researching on container and lightweight VM technologies. Worked on SaltStack, Docker & K8s.

guoqing_li@pm.me
linkedin.com/in/gql

Dario Faggioli

Virtualization Software Engineer, SUSE

Worked on Linux scheduling, then Xen, now Xen & KVM

dfaggioli@suse.com
@DarioFaggioli

Vasiliy Ulyanov

Software Engineer, SUSE

Working on containers and VMs convergence technologies, K8s & KubeVirt

vulyanov@suse.de
linkedin.com/in/vulyanov

Today’s Topic

VM Performance Evaluation and Tuning with KVM and KubeVirt

We will see:

- What could be the effect of vCPU pinning and virtual topology on a VM’s performance
- What tuning facilities are available on KVM and on KubeVirt
- How tuning your VM for the best can lead you to … …
  … … *a quite significant performance* ***loss*** *!!!*

NAS ua
(lower == better)

KVM & KubeVirt

What they are

Traditional Virtualization

Referred to as *KVM*, in the rest of the talk

Open source virtualization solution built into Linux kernel which runs on x86 machines.

K8s Style Virtualization, with KubeVirt

Referred to as *KubeVirt*, in the rest of the talk

Kubernetes add-on that allows running and managing virtual machines on clusters alongside with containerized workloads.

KVM & KubeVirt

Pros and Cons

Traditional Virtualization referred as KVM

Advantages:

- Full control of tuning capability
- Full control of the hosts where the VM runs

Disadvantages:

- Tuning can be complex
- Managing hosts (e.g., allocating VMs on them, etc) might be complex

K8s Style Virtualization, with KubeVirt

Advantages:

- Equipped K8s capability to orchestrate VMs
- Unified management of VMs and containers
- Allows “running VMs on scale”
- Some VM configuration complexities are hidden behind a high-level yaml definition

Disadvantages:

- Does not allow to manually tweak all the available VM parameters
- May introduce additional overhead or limitations due to containerization

Experimental Setup

The Hardware

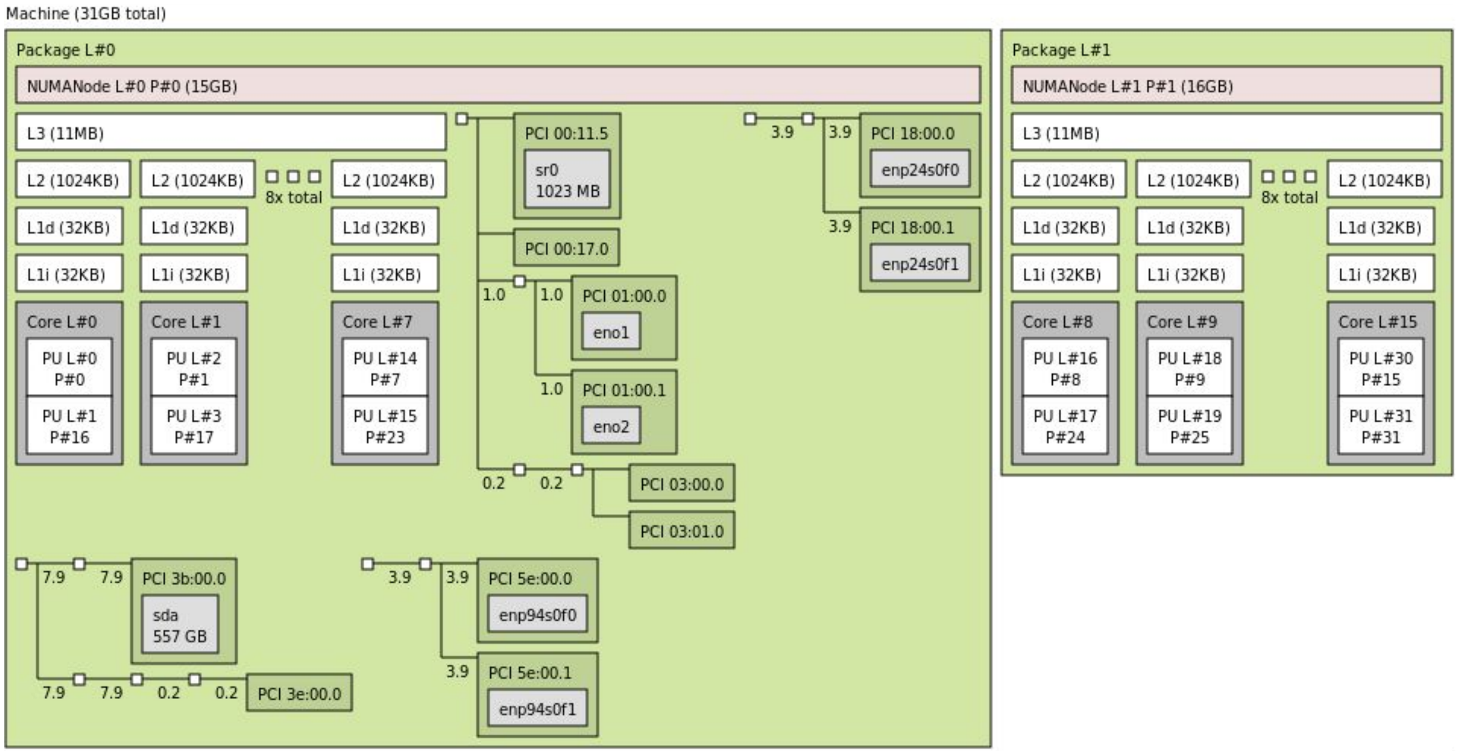
Both for the **KVM host** and for the **KubeVirt worker node**:

Intel(R) Xeon(R) Silver 4208 CPU @ 2.10GHz

– CPU(s):	32
– NUMA nodes (== sockets):	2
– Threads per core:	2
– Cores per socket:	8
– Family/Model/Stepping:	6 / 85 / 7
– MHz (min/max):	800 / 3200
– Cache L1 i & d / L2 / L3:	512 KB / 16 MB / 22 MiB
– Memory:	32 GB
– Node 0 / node 1:	16 GB / 16 GB
– Disk / Filesystem:	Rotational device (no SSD) / ext4

Experimental Setup

The Hardware



Experimental Setup

The Software - Host

Host OS

- Ubuntu 20.04.2 LTS, Kernel 5.4.0 (stock distro one)

KVM

QEMU

- Version 5.2.0 (built from sources)

Libvirt

- Version 7.0.0 (built from sources)

KubeVirt

K8s

- Version 1.21
- Cont. runtime: docker (stock distro one)

KubeVirt

- Version 0.44.0 (latest)
- includes QEMU 5.2.0 & Libvirt 7.0.0

Experimental Setup

The Software - Guest (both KVM & KubeVirt)

(Virtual) Hardware:

- 1 vCPU / 4 vCPUs
- 8 GB RAM
- File backed, raw-format, pre-allocated disk image

OS:

- openSUSE Leap 15.2, kernel 5.3.18 (stock distro one)

Benchmarking Suite:

- MMTests (see also: Scheduler benchmarking with MMTests)
- Benchmarks were running inside the VMs

Experimental Setup

The Benchmarks

Cyclictest

- 1 ms wakeups, FIFO priority, Hackbench in background as noise
- Runs: threads pinned to vCPUs, threads not pinned (unbound)

NASA Parallel Benchmark

- Parallelized with OpenMP, 2 threads (== half the nr. of vCPUs)
- Runs: various computational kernels (bt, cg, ep, ft, is, sp, ua)

STREAM

- Parallelized with OpenMP, 2 threads (== half the nr. of vCPUs)
- Runs: copy, scale, add, triadd

Experimental Setup

The Benchmarks

Hackbench

- Processes, communicating via pipes
- Runs: 2 thread groups (80 tasks), 4 thread groups (160 tasks)

Kernbench

- Building vmlinux, with defconfig
- Runs: make -j 1, make -j 2, make -j 4 (2 == half the nr. of vCPUs, 4 == nr. of vCPUs)

iozone

- Synchronous IO
- Write, rewrite, read, reread, random red, random write, backward read
- Runs: 1GB, 2GB, 4GB

Experimental Setup

Different Running Conditions

VM Size & Configuration

- 1 vCPU / 4 vCPUs
- Different combinations of vCPU pinning and VM virtual topology

Host conditions

1. Idle:
   - Nothing ⇒ Only our VM running
2. Loaded:
   - synthetic load (stress-ng):
     - Total host load ~ 1400% + our VM out of 3200%
     - E.g., simulating 7 other VMs (==> 8 VMs in total), 4 vCPUs, each 50% busy
3. Higly Loaded
   - synthetic load (stress-ng):
     - Total host load ~ 2800% + our VM out of 3200%
     - E.g., simulating 7 other VMs (==> 8 VMs in total), 4 vCPUs, each 100% busy

Icons by Roundicons @ www.flaticon.com

KVM Tuning

Let's Try to Improve Performance

- Transparent / 2MB / 1GB huge pages
- Memory pinning
- virtual CPU (vCPU) pinning
- Emulator threads pinning
- IO threads pinning
- Virtual topology
- Exposure/Availability of host CPU features
- Optimized spinlocks & vCPUs yielding/idling

(Semi-)Static resource allocation

- Less overhead ✔
- Fewer/No interference ✔
- More control ✔
- More difficult to manage ✘
- Less flexible ✘

vCPUs/IO/QEMU threads will only run on a specific subset of the host's physical CPUs (pCPUs)

Memory for the VM will be allocated on using specific pages size and on a specific host NUMA node

The VM vCPUs will be arranged in cores, threads, etc. The VM will use TSC as clocksource, etc. Check, e.g.: "Virtual Topology for Virtual Machines: Friend or Foe?"

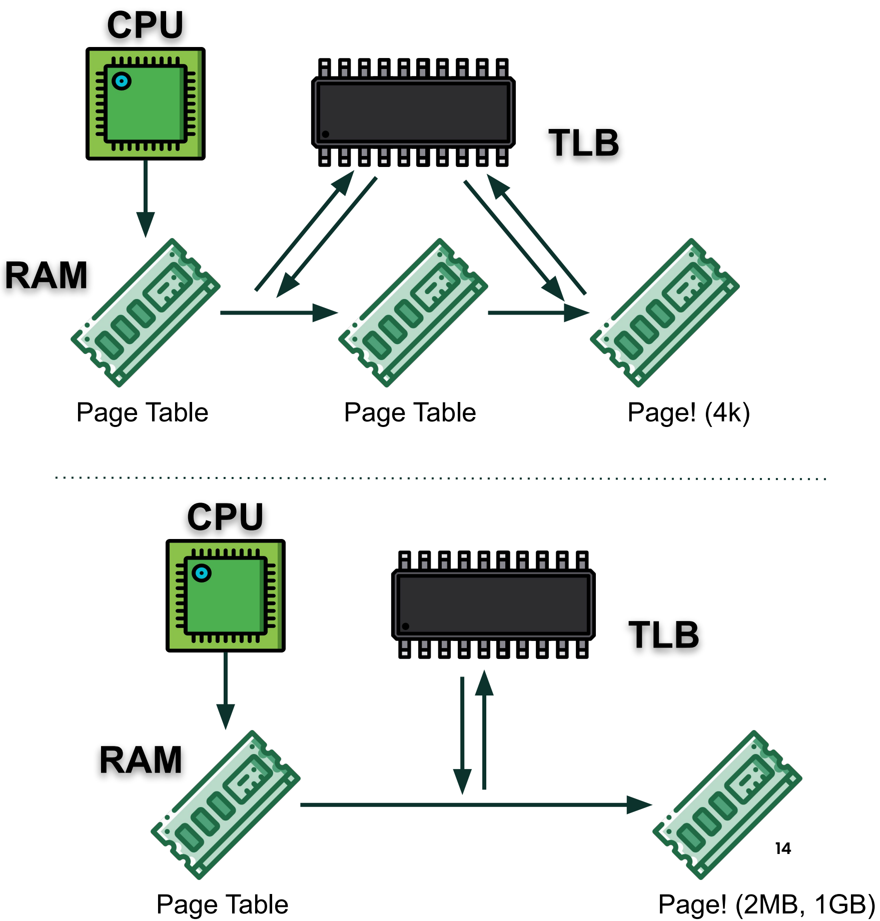
Disabling PV-Spinlocks and PLE, etc. Using cpuidle-haltpoll, etc. Check, e.g.: "No Slower than 10%!"

KVM Tuning

Huge Pages

Larger than 4k pages (2MB, 1GB):

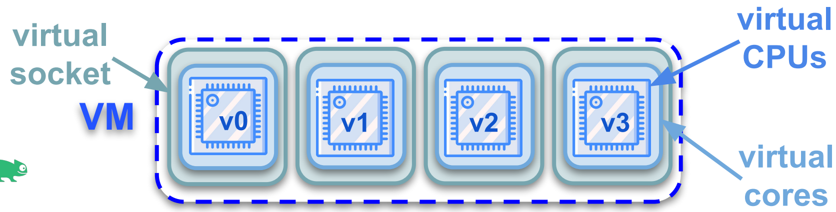
- Faster page walks
- Reduced TLB pressure
- Transparent
  - Use huge pages automatically, as much as possible
  - Dynamic online page merges/splits
    - overhead & fragmentation
- Pre-allocated
  - Less overhead
  - Smaller fragmentation
  - Less flexible
- Can be used both on host and in guest
  - Double the benefits!

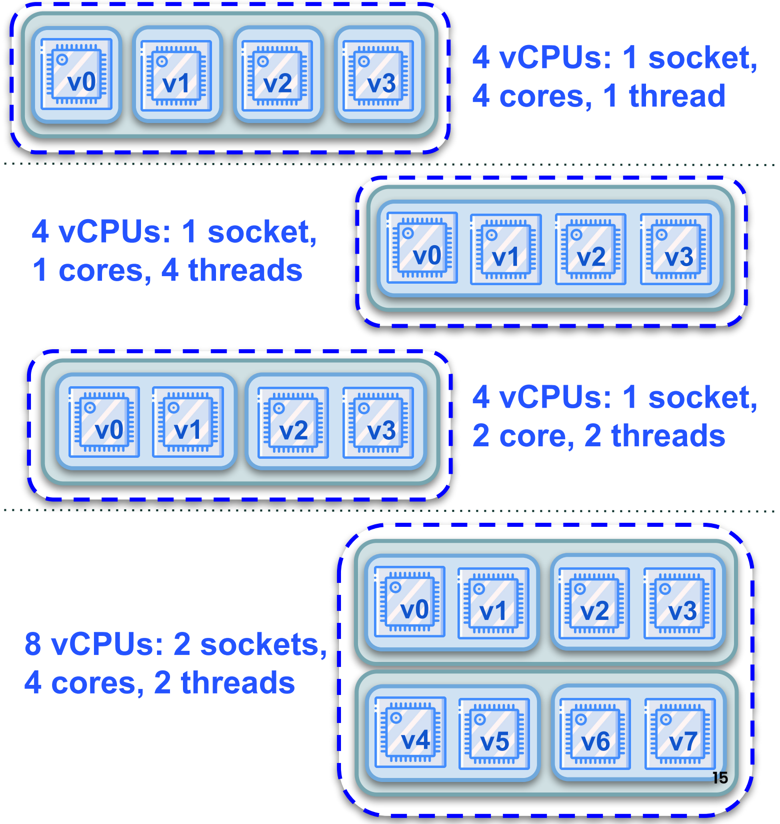


KVM Tuning

Virtual Topology

- Real HW has physical topology
  - NUMA nodes, sockets, cores, threads
  - Improved performance and scalability
- VMs (with > 1 vCPUs) can have virtual topology
  - virtual NUMA nodes, virtual sockets, virtual cores, virtual threads
- VM kernel and apps can make topology aware optimizations (e.g., scheduling)
- Default VM topology:
  - all vCPUs are sockets



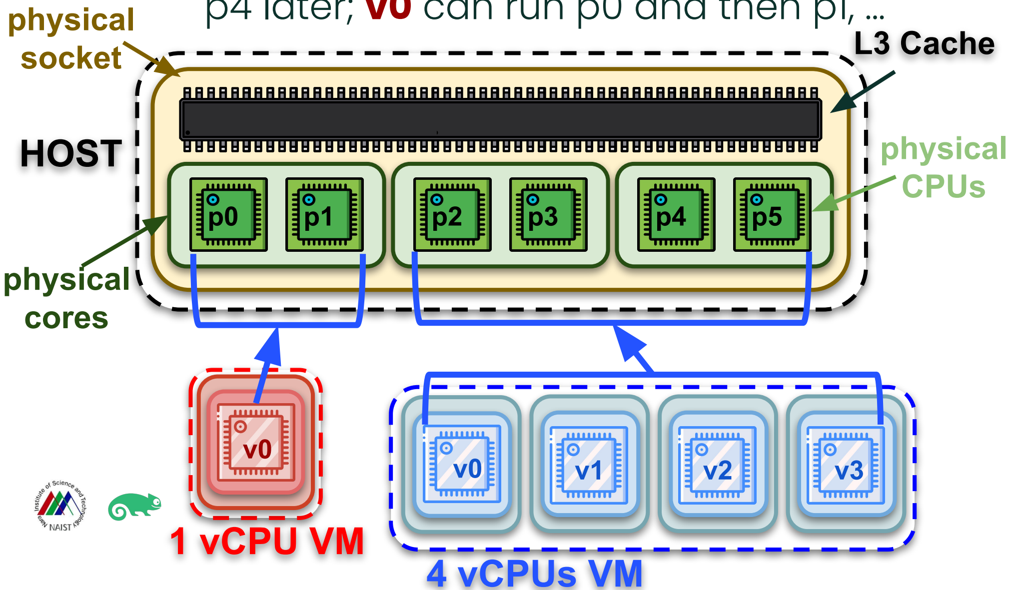


KVM Tuning

vCPU Pinning

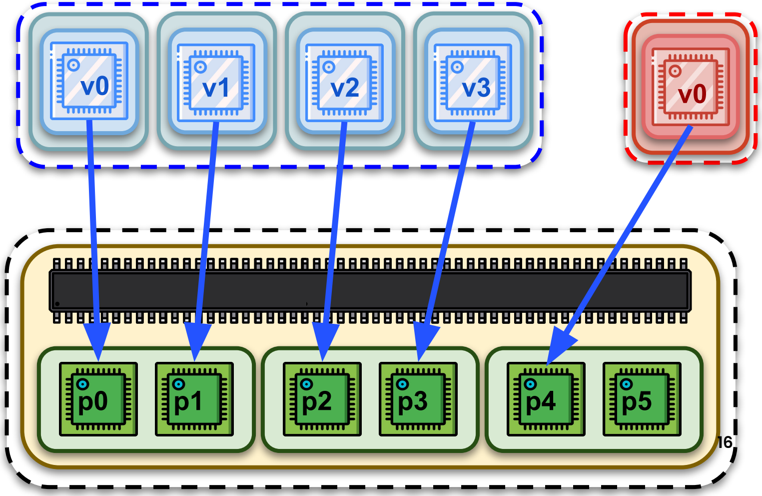
VM-wide vCPU Pinning:

- **v0, v1, v2, v3** will run on pCPUs p2, p3, p4, p5; **v0** will run on pCPUs p0, or p1
- e.g., **v0** can run on p2 now and on p4 later; **v0** can run p0 and then p1, ...



1-to-1 vCPU Pinning:

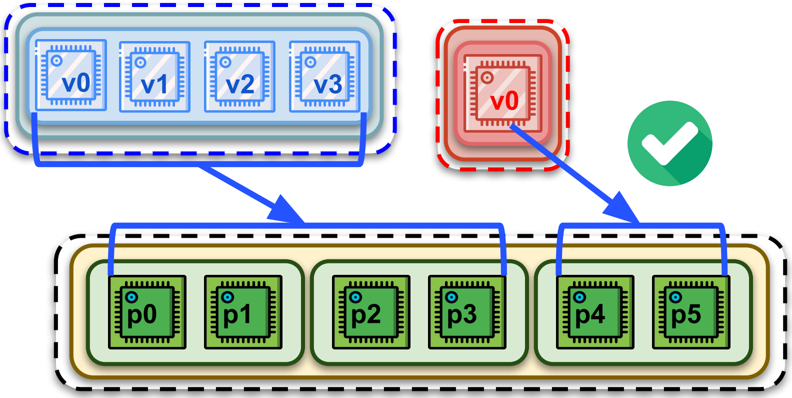
- Each vCPU will always run on a specific pCPU
- E.g., **v0** will always run on p0, **v1** on p1, **v2** on p2, **v3** on p3 and **v0** on p4

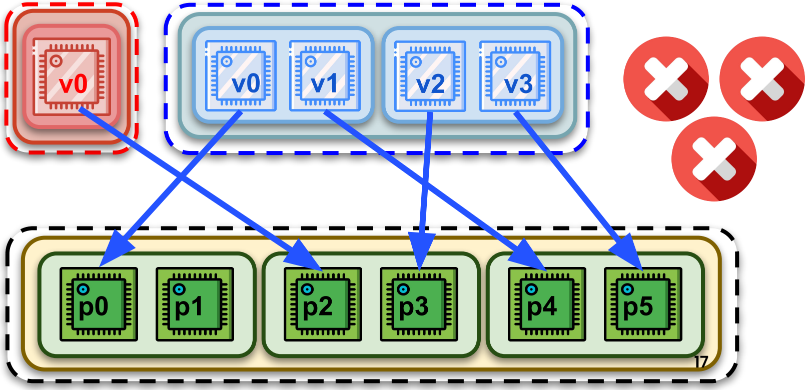


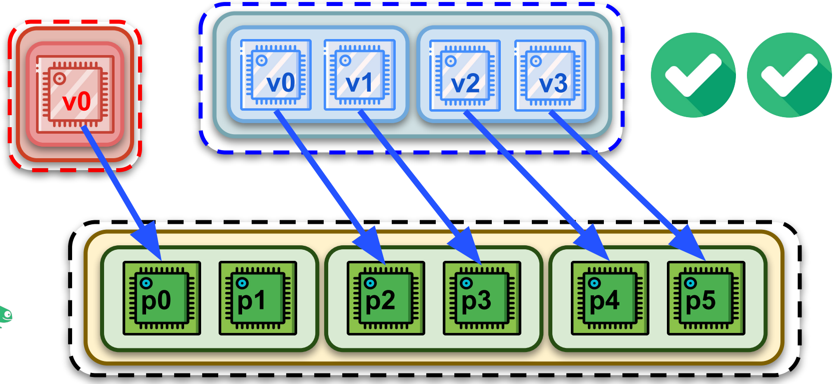
KVM Tuning

Virtual Topology + vCPU Pinning

- Mapping the virtual topology on the physical topology:
  - pin vCPUs of v-cores on pCPUs of p-cores, etc
- Topology aware optimizations in VM becomes really effective
  - Works best with 1-to-1 pinning
- Performance may get worse if done wrong!







KVM Tuning

CPU Model + Memory Pinning

Memory Pinning

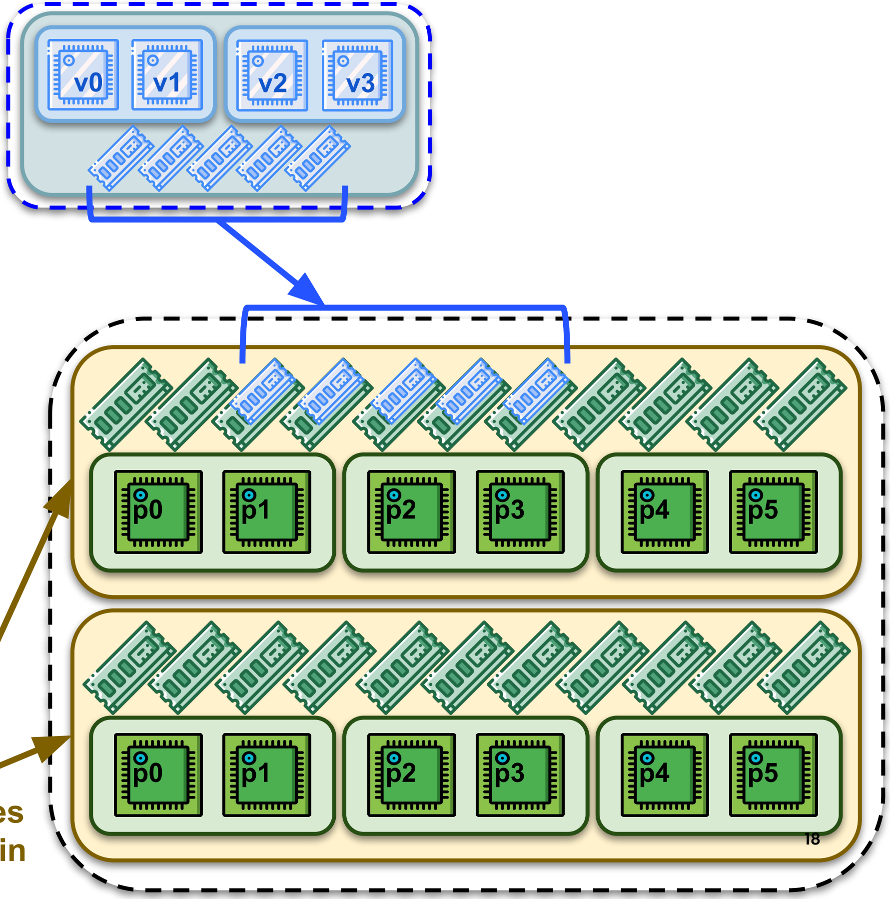
- All the memory for the VM allocated on one (if possible) NUMA node
- Works best together with vCPU pinning

"Passthrough" of the CPU Model

- Host pCPU features, special instruction sets, etc are available inside the VM
- We did it in our experiments

KVM `hint-dedicated` & `cpuidle-haltpoll`

- Further optimization when running on static partitioned host
- We don't use them in our experiments



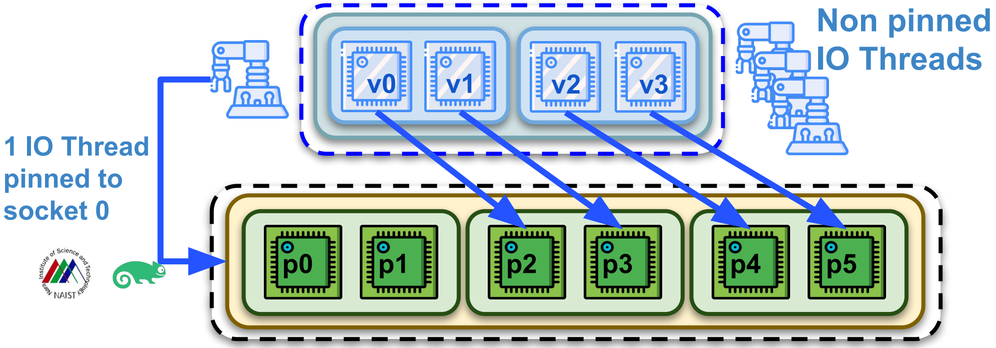
Host NUMA nodes (= host sockets, in this case)

KVM Tuning

Emulator and IO Threads Pinning

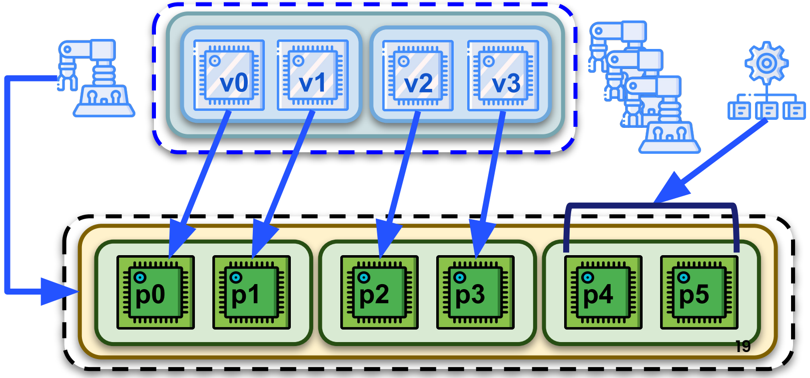
IO Threads

- Break down QEMU (IO) event handling
- Improved scalability:
  - Parallelizing work
  - Reduce lock contention
- Can have many IO Threads
  - E.g., 1 per block device
  - No more than nr. of pCPUs
- IO Threads may be pinned to pCPUs



Emulator threads

- Other QEMU threads (main event loop, SPICE, migration, …)
- May interfere with & “steal” resources from the vCPUs
  - Can be moved “out of the way” by pinning them on different pCPUs



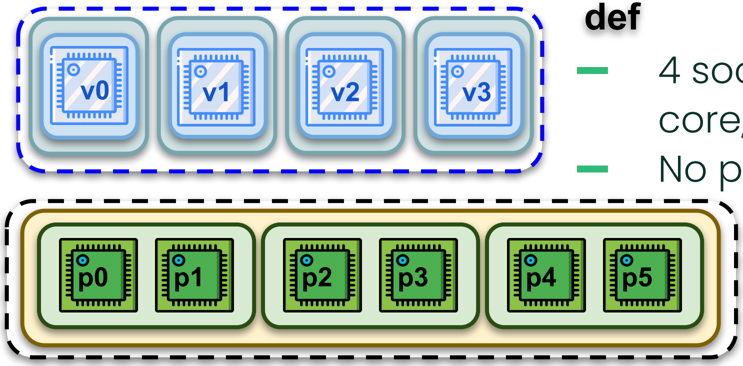
KVM Tuning

Disk IO Tuning

- Caching
  - `none` (see "Async IO Model" below)
- Async. IO Model
  - `threads` (default)
    - QEMU user-space thread pool
    - IOzone & kernbench lasting **a few hours...** *Not sure how many, killed before it finished!*
  - `native`
    - Linux kernel AIO
    - IOZone & kernbench, reasonable durations
  - `io_uring`
    - future investigations
  - Avoid trims (so image stays pre-allocated!)
- Multi-queueing
  - (if available)

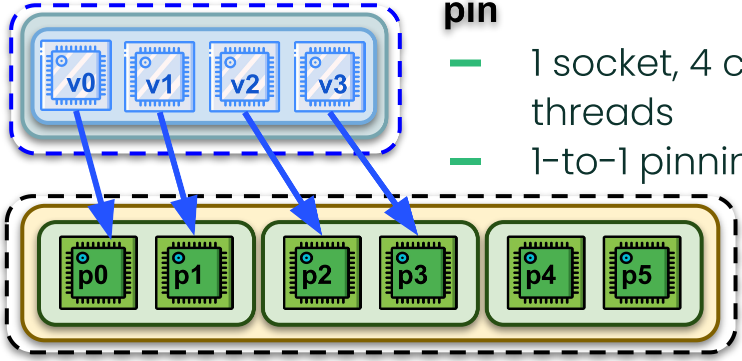
KVM Tuning

Experimented Pinning + Topology Configuration ⇐ Manually Crafted by Us



def

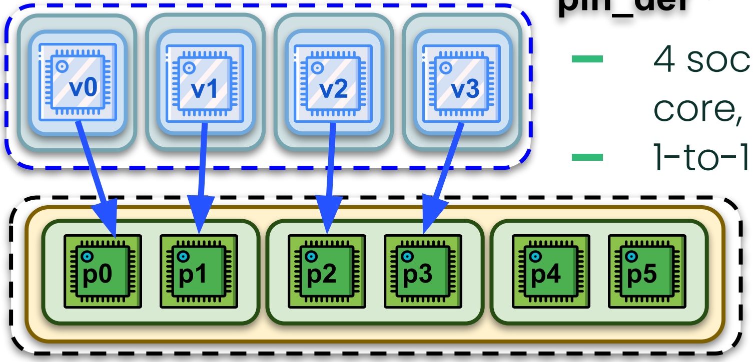
- 4 sockets, 1 core, 1 thread
- No pinning



pin

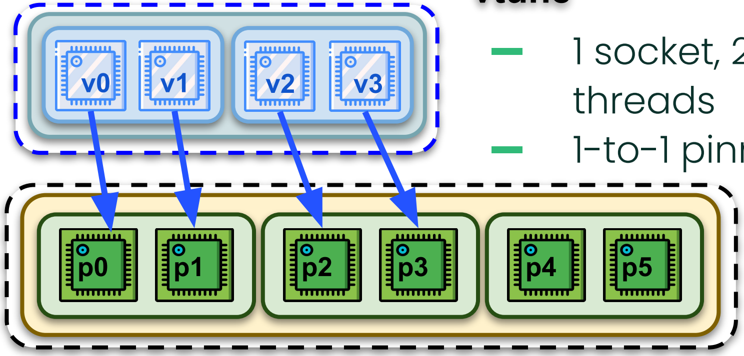
- 1 socket, 4 cores, 1 threads
- 1-to-1 pinning

Host Load was pinned as well!



pin_def

- 4 sockets, 1 core, 1 thread
- 1-to-1 pinning



vtune

- 1 socket, 2 cores, 2 threads
- 1-to-1 pinning

KVM Tuning

Experimented Pinning + Topology Configuration ⇐ Manually Crafted by Us

def

ets, 1
thread
ning

```
<vcpu placement='static'>4</vcpu>
<cpu mode='host-model' check='partial'/>
```

pin

, 1

```
<vcpu placement='static'>4</vcpu>
<cputune>
  <vcpupin vcpu='0' cpuset='1'/>
  <vcpupin vcpu='1' cpuset='17'/>
  <vcpupin vcpu='2' cpuset='2'/>
  <vcpupin vcpu='3' cpuset='18'/>
</cputune>
<cpu mode='host-passthrough' check='none'>
    <topology sockets='1' dies='1' cores='4' threads='1'/>
</cpu>
```

Host Load was pinned as well!

pin_def

ets, 1
thread
pinning

```
<vcpu placement='static'>4</vcpu>
<cputune>
  <vcpupin vcpu='0' cpuset='1'/>
  <vcpupin vcpu='1' cpuset='17'/>
  <vcpupin vcpu='2' cpuset='2'/>
  <vcpupin vcpu='3' cpuset='18'/>
</cputune>
<cpu mode='host-passthrough' check='none'>
```

vtune

es, 2

```
<vcpu placement='static'>4</vcpu>
<cputune>
  <vcpupin vcpu='0' cpuset='1'/>
  <vcpupin vcpu='1' cpuset='17'/>
  <vcpupin vcpu='2' cpuset='2'/>
  <vcpupin vcpu='3' cpuset='18'/>
</cputune>
<cpu mode='host-passthrough' check='none'>
    <topology sockets='1' dies='1' cores='2' threads='2'/>
</cpu>
```

KVM Tuning

Experimented Pinning + Topology Configuration ⇐ Manually Crafted by Us

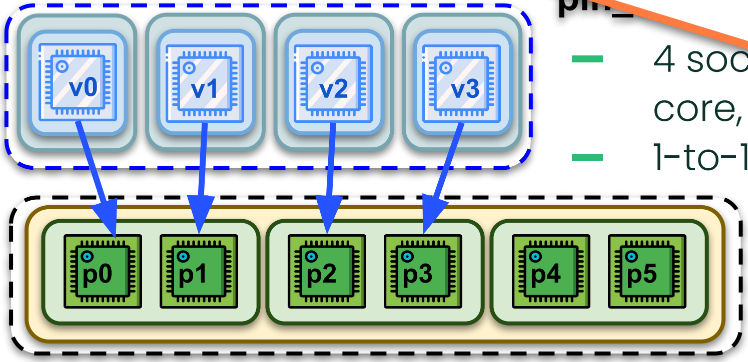
def

pin

Perfect match between virtual and physical topologies:

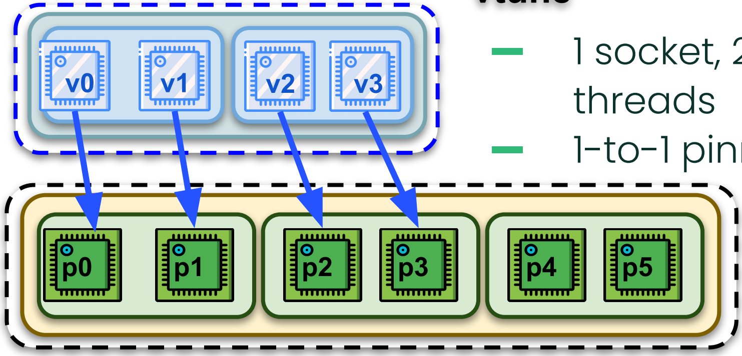
- Full virtual cores ⇒ Full physical cores
- v0 & v1: virtual hyperthreads ⇒ p0 & p1: physical hyperthreads

We expect **best** performance!



- 4 socket..., core, 1 thread
- 1-to-1 pinning

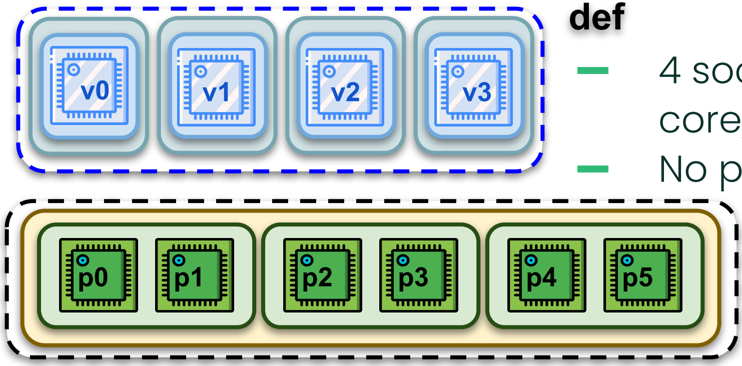
vtune



- 1 socket, 2 cores, 2 threads
- 1-to-1 pinning

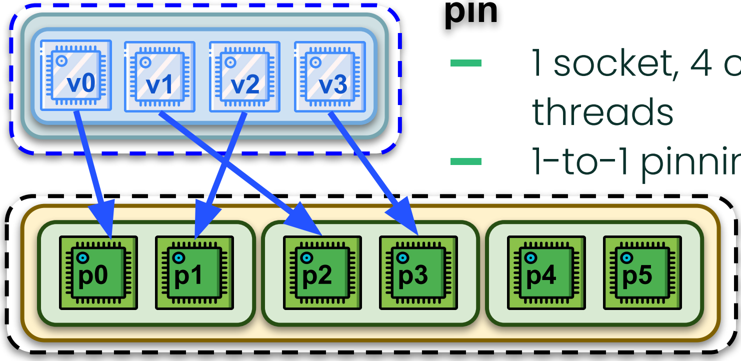
KubeVirt Tuning

Experimented Pinning + Topology Configuration ⇐ Automatically Done by KubeVirt



def

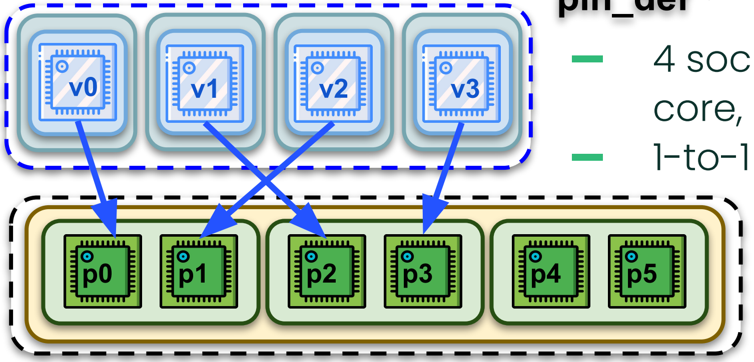
- 4 sockets, 1 core, 1 thread
- No pinning



pin

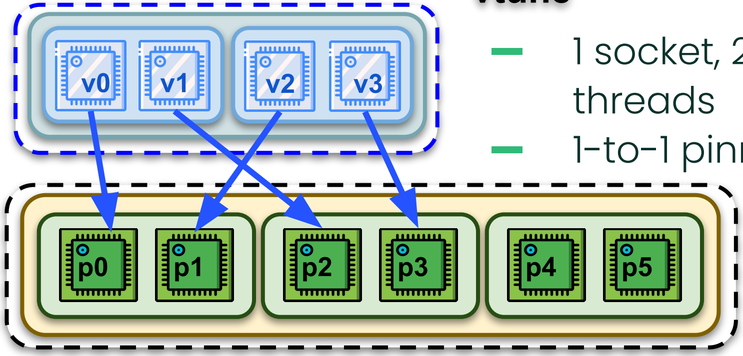
- 1 socket, 4 cores, 1 threads
- 1-to-1 pinning

Host Load was pinned as well!



pin_def

- 4 sockets, 1 core, 1 thread
- 1-to-1 pinning



vtune

- 1 socket, 2 cores, 2 threads
- 1-to-1 pinning

KubeVirt Tuning

Experimented Pinning + Topology Configuration ⇐ <u>Automatically Done by KubeVirt</u>

def

ets, 1
thread
ning

```yaml
spec:
  domain:
    resources:
      requests:
        cpu: 4
```

pin

ores, 1
g

```yaml
spec:
  domain:
    cpu:
      sockets: 1
      cores: 4
      threads: 1
      model: host-passthrough
      dedicatedCpuPlacement: true
```

Host Load was pinned as well!

pin_def

ets, 1
thread
pinning

```yaml
spec:
  domain:
    resources:
      requests:
        cpu: 4
    cpu:
      model: host-passthrough
      dedicatedCpuPlacement: true
```

vtune

cores, 2
ing

```yaml
spec:
  domain:
    cpu:
      sockets: 1
      cores: 2
      threads: 2
      model: host-passthrough
      dedicatedCpuPlacement: true
```

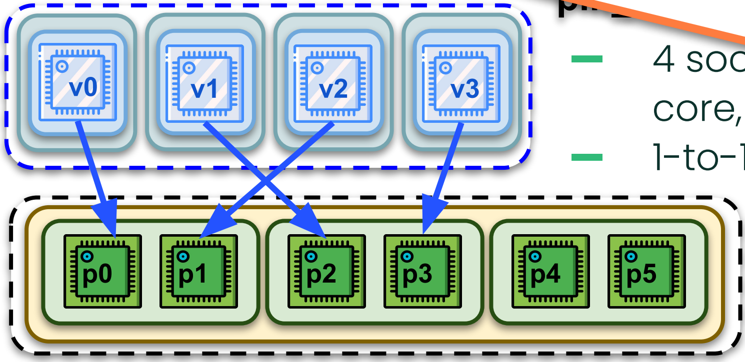
KubeVirt Tuning

Experimented Pinning + Topology Configuration ⇐ Automatically Done by KubeVirt

Wait… What ?!?

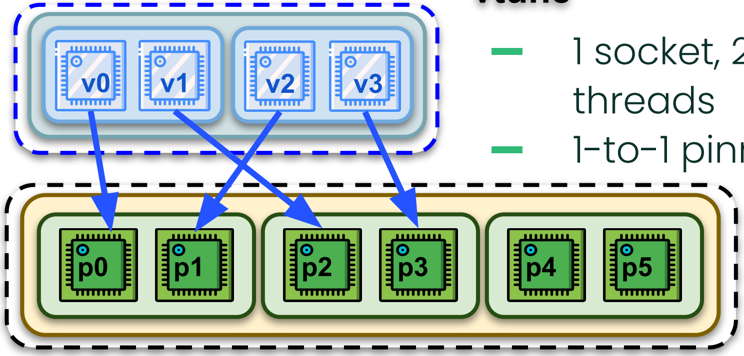
- Full virtual cores ⇒ Mixed & mismatched physical cores !!!
- v0 & v1: virtual hyperthreads
  - Pinned to p0 & p2 …
  - … but the real physical hyperthreads are p0 & p1 !!!

We expect ~~best~~ **???** performance!



- 4 sockets, core, 1 thread
- 1-to-1 pinning

vtune



- 1 socket, 2 cores, 2 threads
- 1-to-1 pinning

KVM vs. KubeVirt

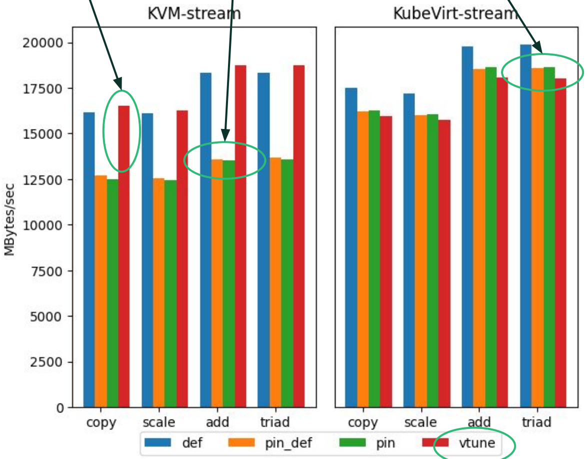
STREAM

Best performance Guarantees

2 vCPUs within same core competes

Idle

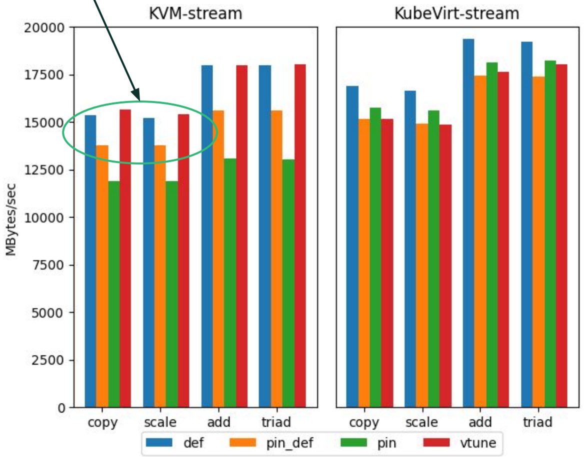
vCPU pinning is out of KubeVirt control



Matching host topology

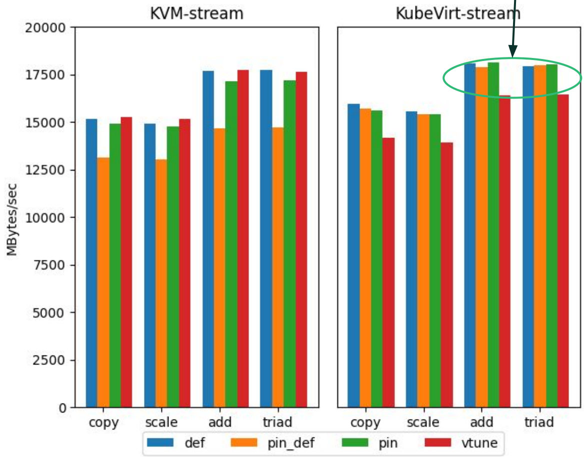
The gap became smaller

Loaded



Almost no difference

Highly loaded



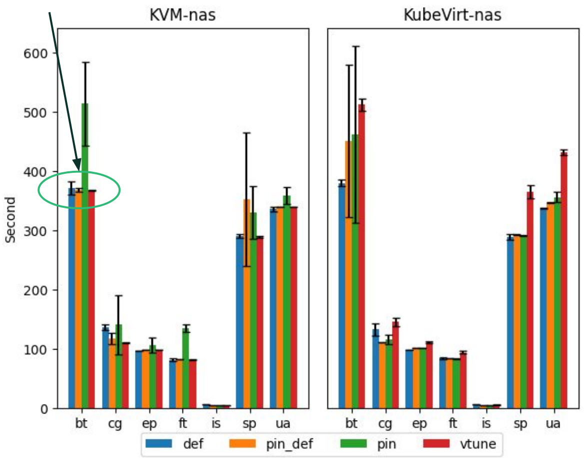
https://www.cs.virginia.edu/stream/

KVM vs. KubeVirt

NAS Parallel Benchmarks (with OpenMP)

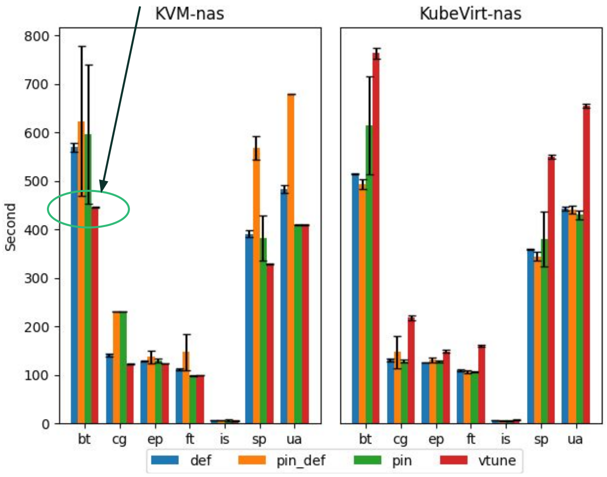
Difference is small

Idle

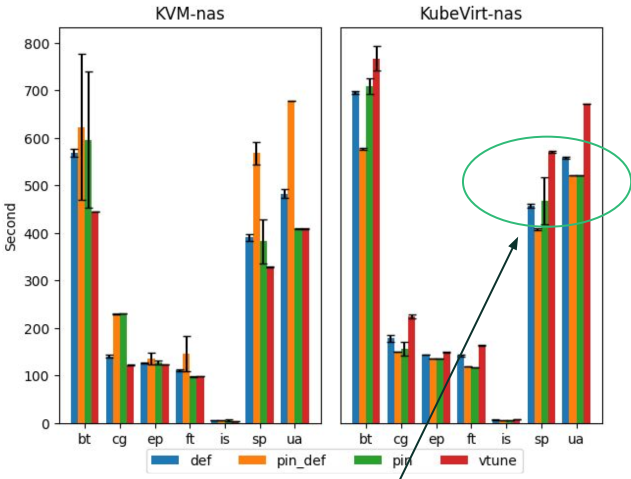


Obvious improvement with vtune

Loaded



Highly loaded



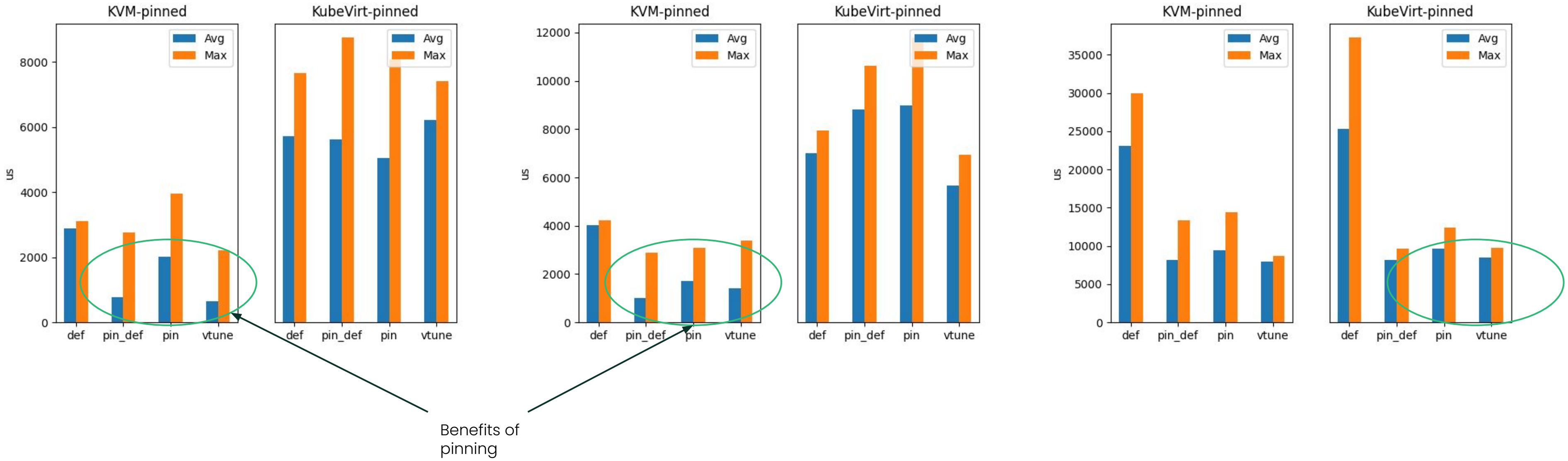
mismatched topology leads to disaster

https://www.nas.nasa.gov/software/npb.html

KVM vs. KubeVirt

Cyclictest (pinned threads)

Idle | Loaded | Highly loaded

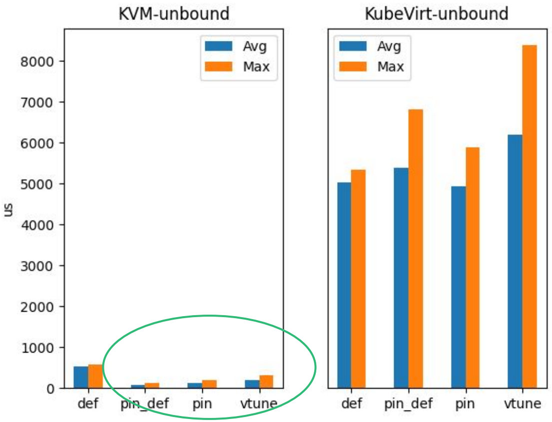


https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/cyclictest/start

KVM vs. KubeVirt

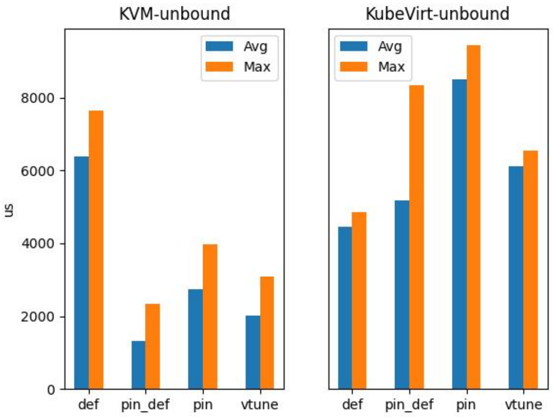
Cyclictest (unbound threads)

Idle

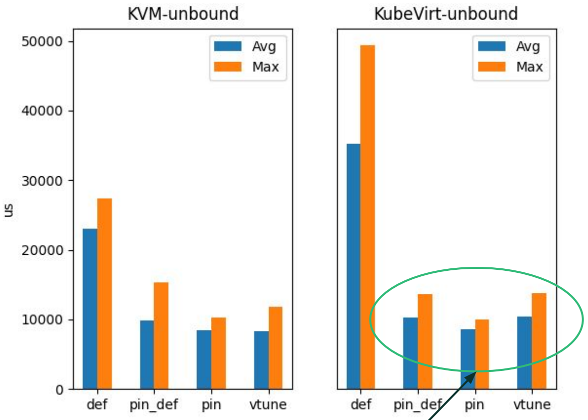


Something we
don't quite
understand

Loaded



Highly loaded



Larger latency
with high load

https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/cyclictest/start

KVM vs. KubeVirt

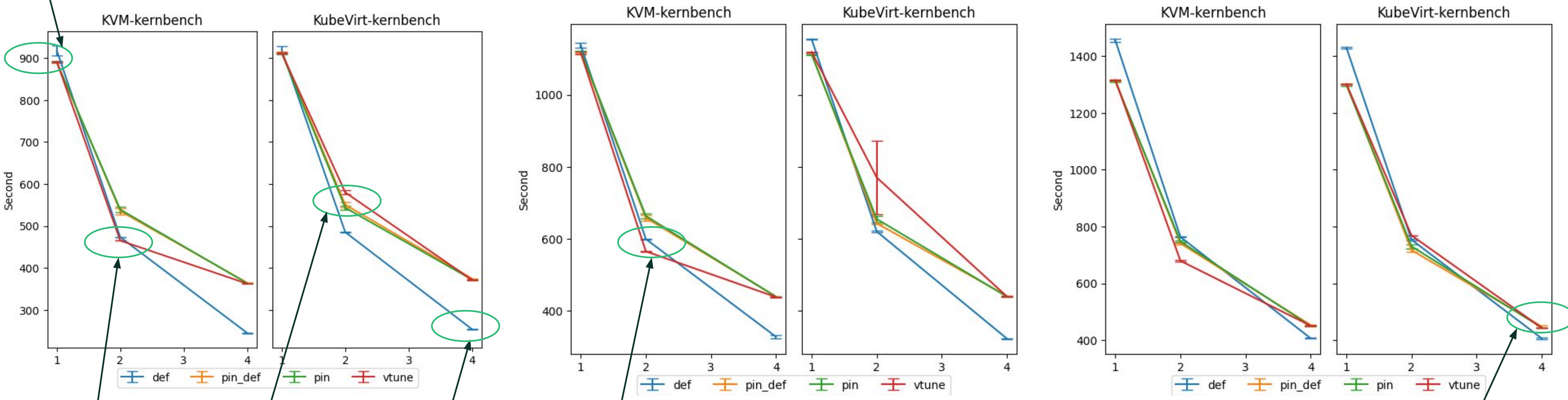
Kernbench

pinned, no cpu migration improves performance

Idle

Loaded

Highly loaded



vtunes beats def

Can't help with mismatched topology

Default beats vtune due to saturation

tuning became effect with load

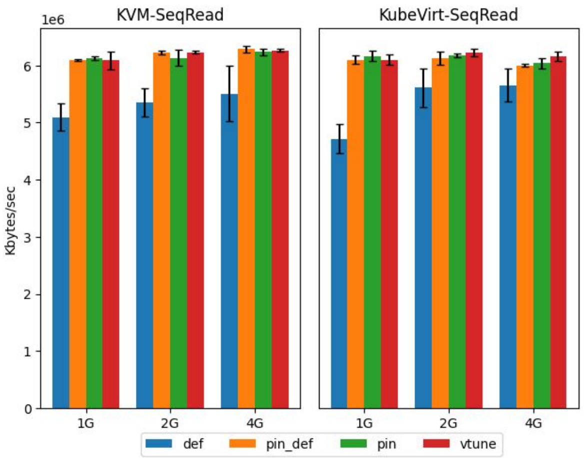
all vcpus are busy, topology doesn't really matter

http://ck.kolivas.org/apps/kernbench/kernbench-0.50/

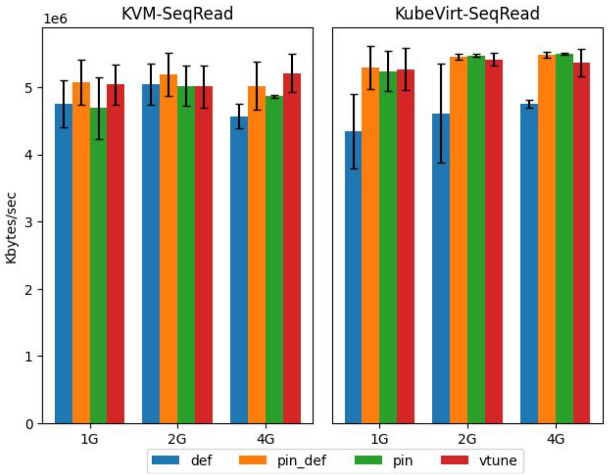
KVM vs. KubeVirt

IOzone – Sequential Read

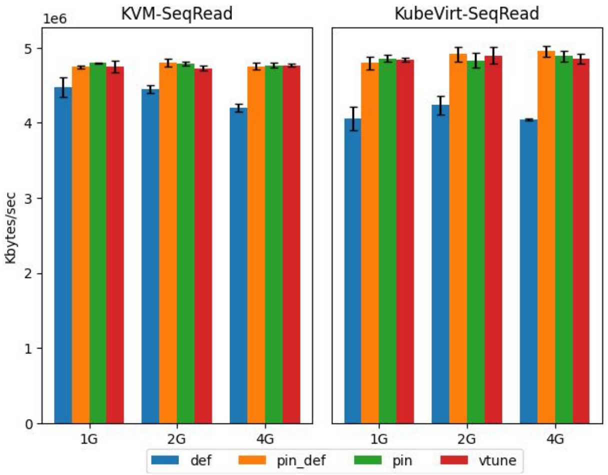
Idle



Loaded



Highly loaded

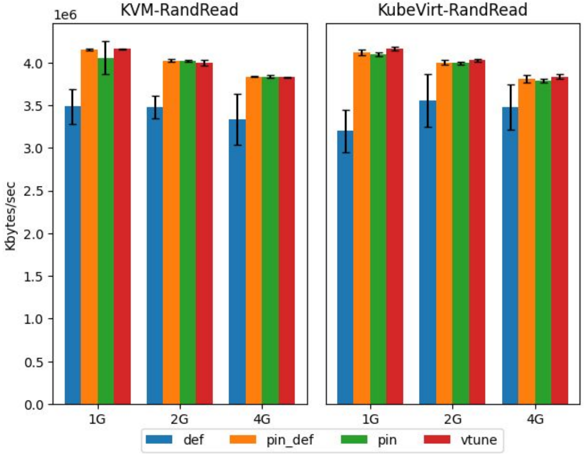


https://www.iozone.org/

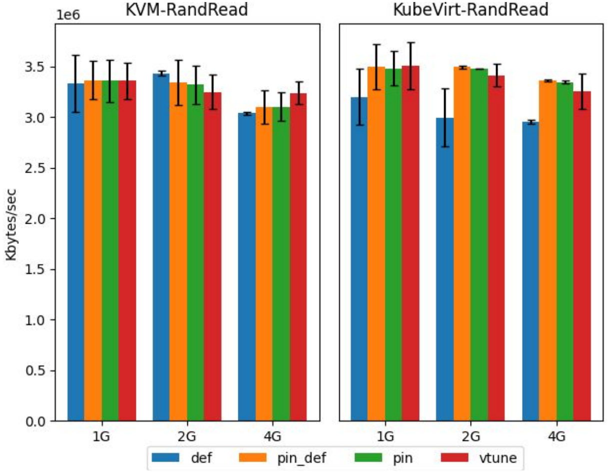
KVM vs. KubeVirt

IOzone – Random Read

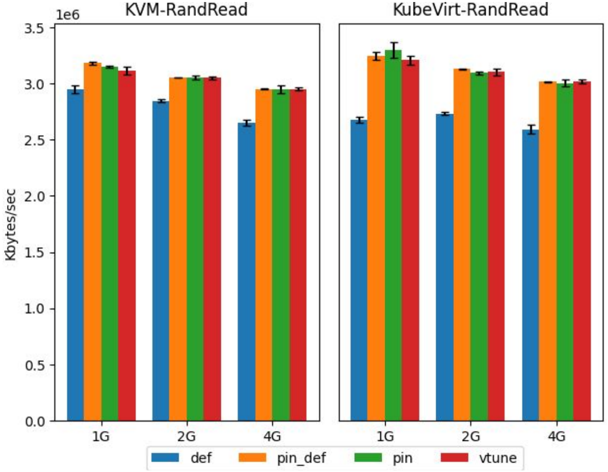
Idle



Loaded



Highly loaded

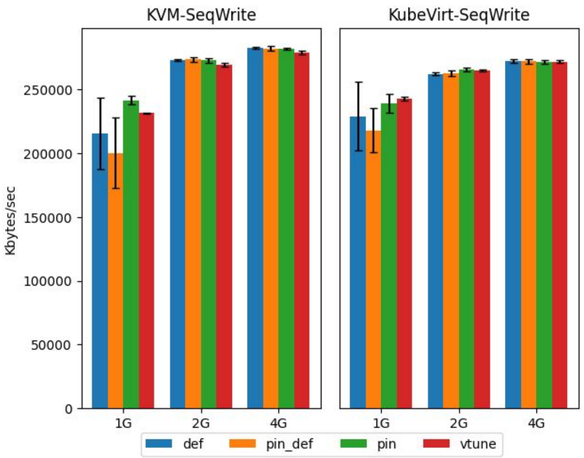


https://www.iozone.org/

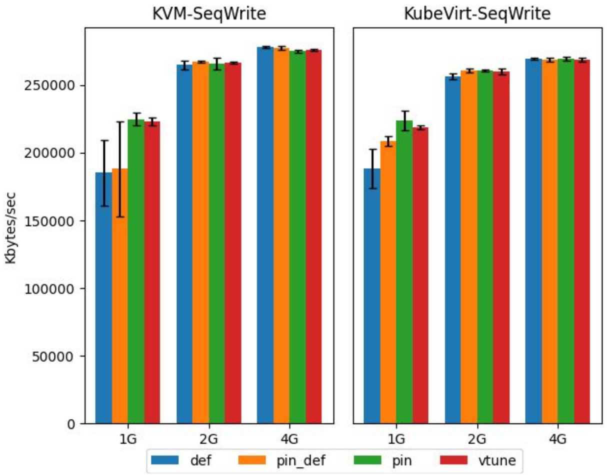
KVM vs. KubeVirt

IOzone – Sequential Write

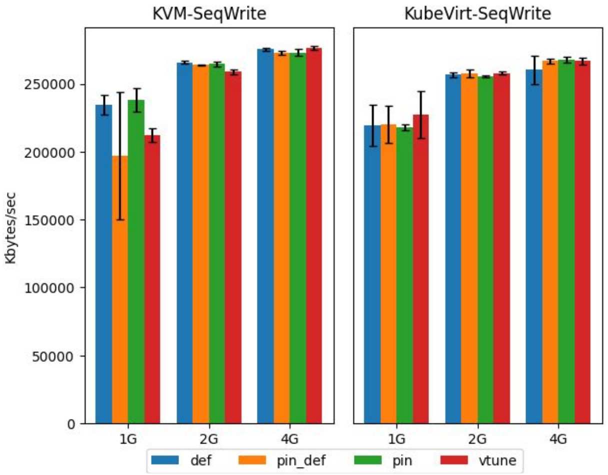
Idle



Loaded



Highly loaded

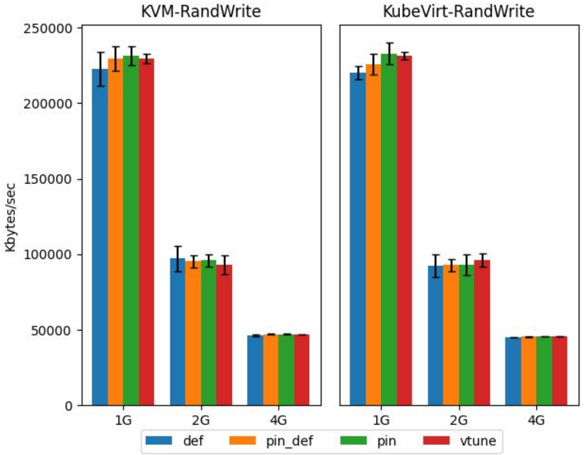


https://www.iozone.org/

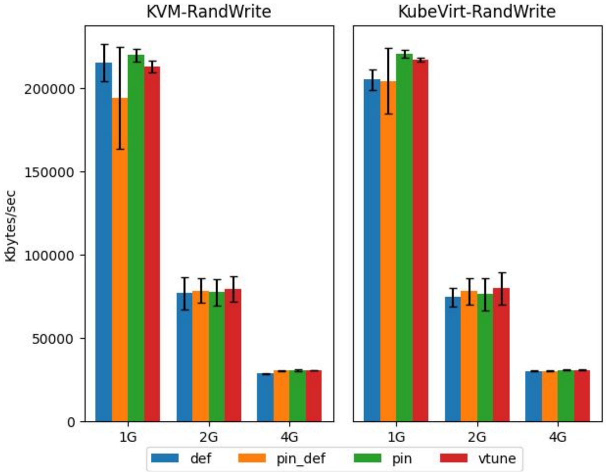
KVM vs. KubeVirt

IOzone - Random Write

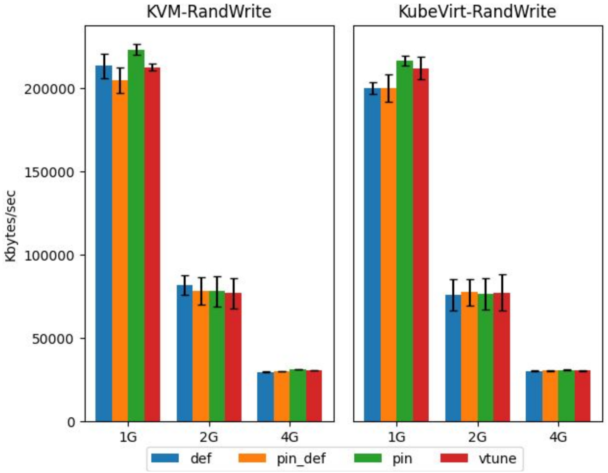
Idle



Loaded



Highly loaded



https://www.iozone.org/

Conclusions

- Matching host CPU topology guarantee good performance
- Host scheduler can manage well in default case if there is not much load
- Inherent limitation of Kubevirt with CPU pinning
  - CPU allocation is managed by CPU manager in K8s
  - default configuration works well in general
- KubeVirt can be improved to avoid mismatching cpu topology

SUSE

NAIST

Thank you

For more information, contact SUSE at:

+1 800 796 3700 (U.S./Canada)

Maxfeldstrasse 5

90409 Nuremberg

www.suse.com