zalando

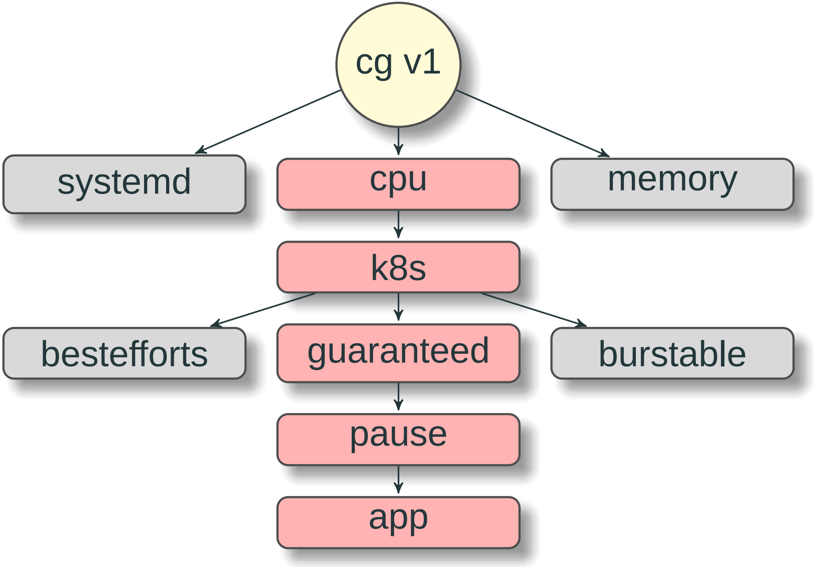
Kubernetes resources sharing

DMITRII DOLGOV

22-10-2018

- → CPU
- → CPU cache
- → Storage IO
- → Network
- → Memory
- → Hugetlb

container = cgroup + namespace


cg v1
systemd
cpu
memory
k8s
bestefforts
guaranteed
burstable
pause
app

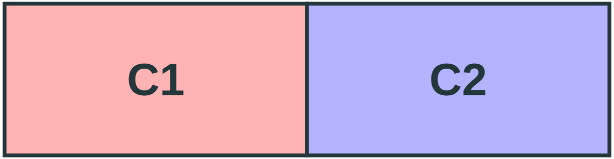
CPU

directly manageable

requests → cpu.share

limits → cpu.cfs_period & cpu.cfs_quota_us

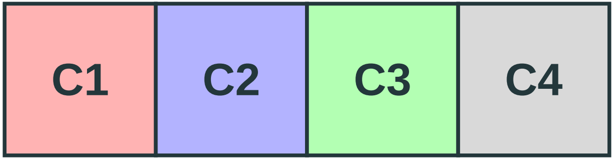
Share



C1

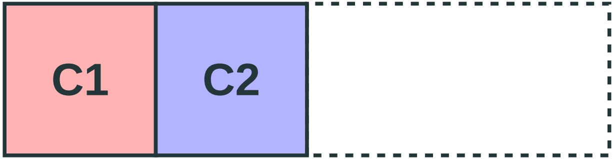
C2

Share

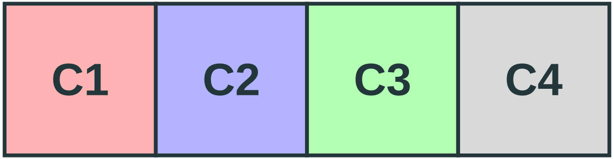


C1 C2 C3 C4

Bandwidth



Bandwidth



C1 C2 C3 C4

Bandwidth accounting

```
# from /proc/sys/kernel/
sched_cfs_bandwidth_slice_us
# default=5ms
```

Allocatable

```
Capacity:
 cpu:                          16
 hugepages-1Gi:                0
 hugepages-2Mi:                0
 memory:                       65947396Ki
Allocatable:
 cpu:                          15800m
 hugepages-1Gi:                0
 hugepages-2Mi:                0
 memory:                       65388292Ki
```

Exclusive CPU

cpu management policy

kube-reserved

guaranteed

integer quantity

cpuset.cpus

cpuset.cpuset.cpu_exclusive

cpuset.mems ?

Cache

```
# COS1 4 cache ways, COS 2 next 8 cache ways
=> pqos -e "llc:1=0x000f;llc:2=0x0ff0;"
SOCKET 0 L3CA COS1 => MASK 0xf
SOCKET 0 L3CA COS2 => MASK 0xff0
Allocation configuration altered.
```

Cache

```
=> pqos -s
L3CA COS definitions for Socket 0:
        L3CA COS0 => MASK 0xffff
        L3CA COS1 => MASK 0xf
        L3CA COS2 => MASK 0xff0
        L3CA COS3 => MASK 0xfff
Core information for socket 0:
        Core 0, L2ID 0, L3ID 0 => COS0
        Core 1, L2ID 1, L3ID 0 => COS0
        Core 2, L2ID 0, L3ID 0 => COS0
        Core 3, L2ID 1, L3ID 0 => COS0
```

Memory

directly manageable

requests $\not\rightarrow$ memory.soft_limit_in_bytes

limits $\rightarrow$ memory.limit_in_bytes (OOM)

memory.kmem.limit_in_bytes

best efforts (not everything is accounted)

Memory reclaim

```
# only under the memory pressure
root@k8s-node-2:/home/vagrant# ./page_reclaim.py
Attaching...
Listening...
Detaching...
[7382] postgres: 928K
[7138] postgres: 152K
[7136] postgres: 180K
[7468] postgres: 72M
[7464] postgres: 57M
[5451] postgres: 1M
```

Writeback (cgroup v1)

```
/* vmscan.c */
/* The normal page dirty throttling mechanism
 * in balance_dirty_pages() is completely broken
 * with the legacy memcg and direct stalling in
 * shrink_page_list() is used for throttling instead,
 * which lacks all the niceties such as fairness,
 * adaptive pausing, bandwidth proportional
 * allocation and configurability.
 */
static bool sane_reclaim(struct scan_control *sc)
```

Writeback monitoring

```
=> perf record -e writeback:writeback_written
kworker/u8:1 5816.288044: nr_pages=101429
kworker/u8:1 5816.288129: nr_pages=9223372036854775789
kworker/u8:3 5817.312319: nr_pages=101457
```

Writeback monitoring

```
# pgbench insert
=> ./io_timeouts.py -p bin/postgres
Attaching...
Listening...
Detaching...
[18335] END: MAX_SCHEDULE_TIMEOUT
[18333] END: MAX_SCHEDULE_TIMEOUT
[18331] END: MAX_SCHEDULE_TIMEOUT
[18318] truncate pgbench_history: MAX_SCHEDULE_TIMEOUT
```

Huge pages

directly manageable

transparent vs classic

isolation only per pod

no soft limits or reclaim (SIGBUS)

TLB misses are faster and less frequent

memory leaks (but PG is good)

Huge pages

```
# huge_pages on
Samples: 832K of event 'dTLB-load-misses'
Event count (approx.): 640614445 : ~19% less
Samples: 736K of event 'dTLB-store-misses'
Event count (approx.): 72447300 : ~29% less

# huge_pages off
Samples: 894K of event 'dTLB-load-misses'
Event count (approx.): 784439650
Samples: 822K of event 'dTLB-store-misses'
Event count (approx.): 101471557
```

Storage IO

blkio.weight

blkio.throttle.*

Not used in K8S

sane_behavior

cpuset.cpus

cpuset.cpuset.cpu_exclusive

cpuset.mems ?

IO scheduler

```
=> cat /sys/block/xvdcj/queue/scheduler
[mq-deadline] kyber bfq none
```

IO scheduler

BFQ distributes the bandwidth of of the device among all processes according to their weights, regardless of the device parameters and with any workload.

IO scheduler

The Kyber I/O scheduler is a low-overhead scheduler suitable for multiqueue and other fast devices. Given target latencies for reads and synchronous writes, it will self-tune queue depths to achieve that goal.

Network

not directly

network class

traffic control

Kernel noise

Futex

Compaction

Readahead

Filesystem

...

Questions?

github.com/erthalion

@erthalion

✉ dmitrii.dolgov at zalando dot de

✉ 9erthalion6 at gmail dot com