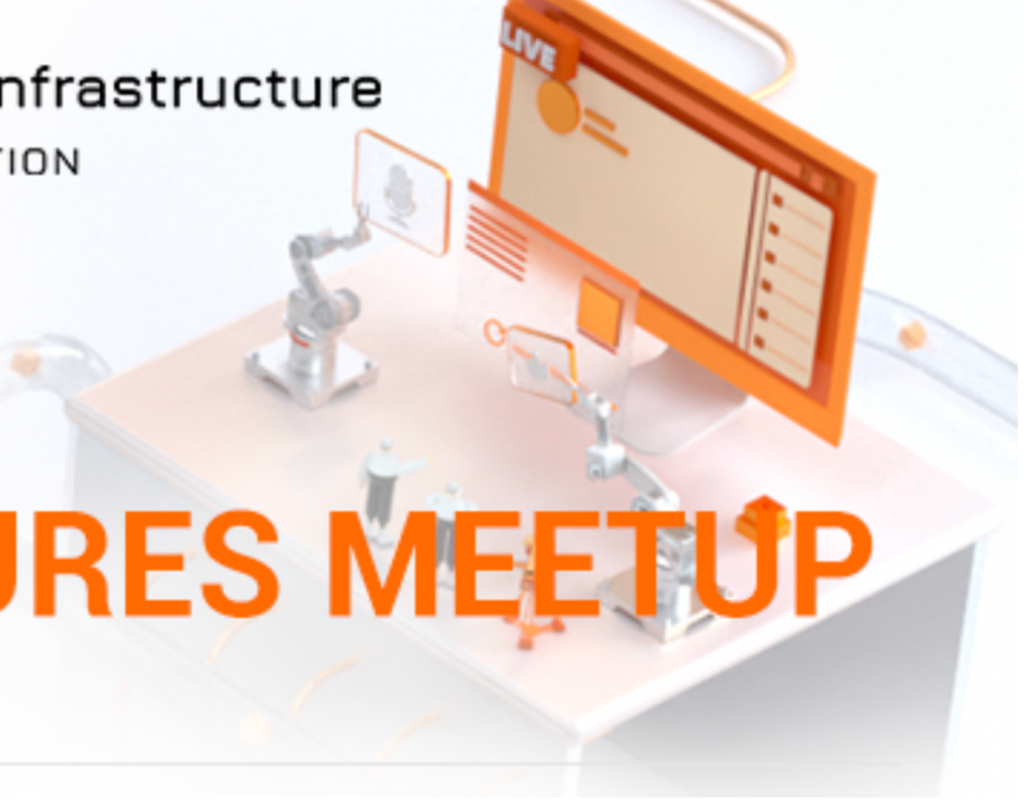
Open Anolis | Open Infrastructure FOUNDATION



CLOUD NATIVE INFRASTRUCTURES MEETUP

活动交流群

关注OpenAnolis.org

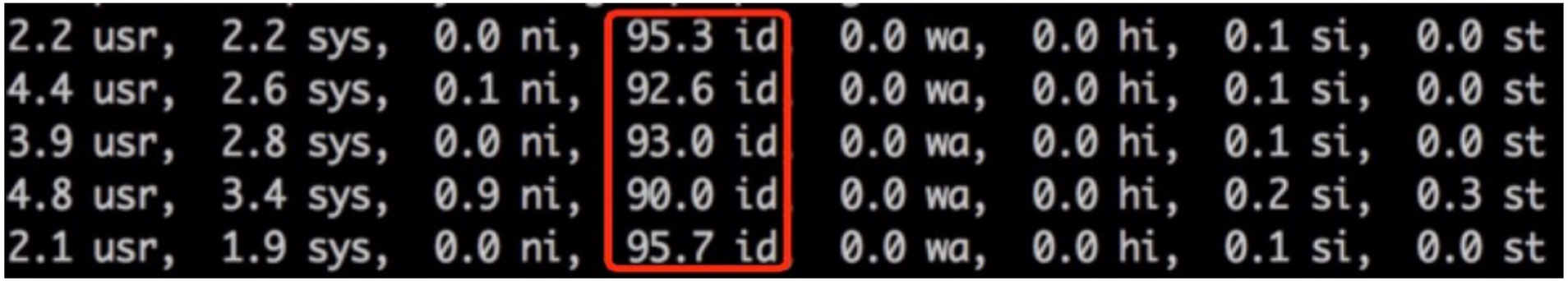
阿里云 | 奥运会全球指定云服务商

Alibaba Cloud Linux
内核资源隔离及混部实践

阿里云智能 操作系统研发
庞训磊 2020.12

背景

线上某业务容器的日常CPU水位：

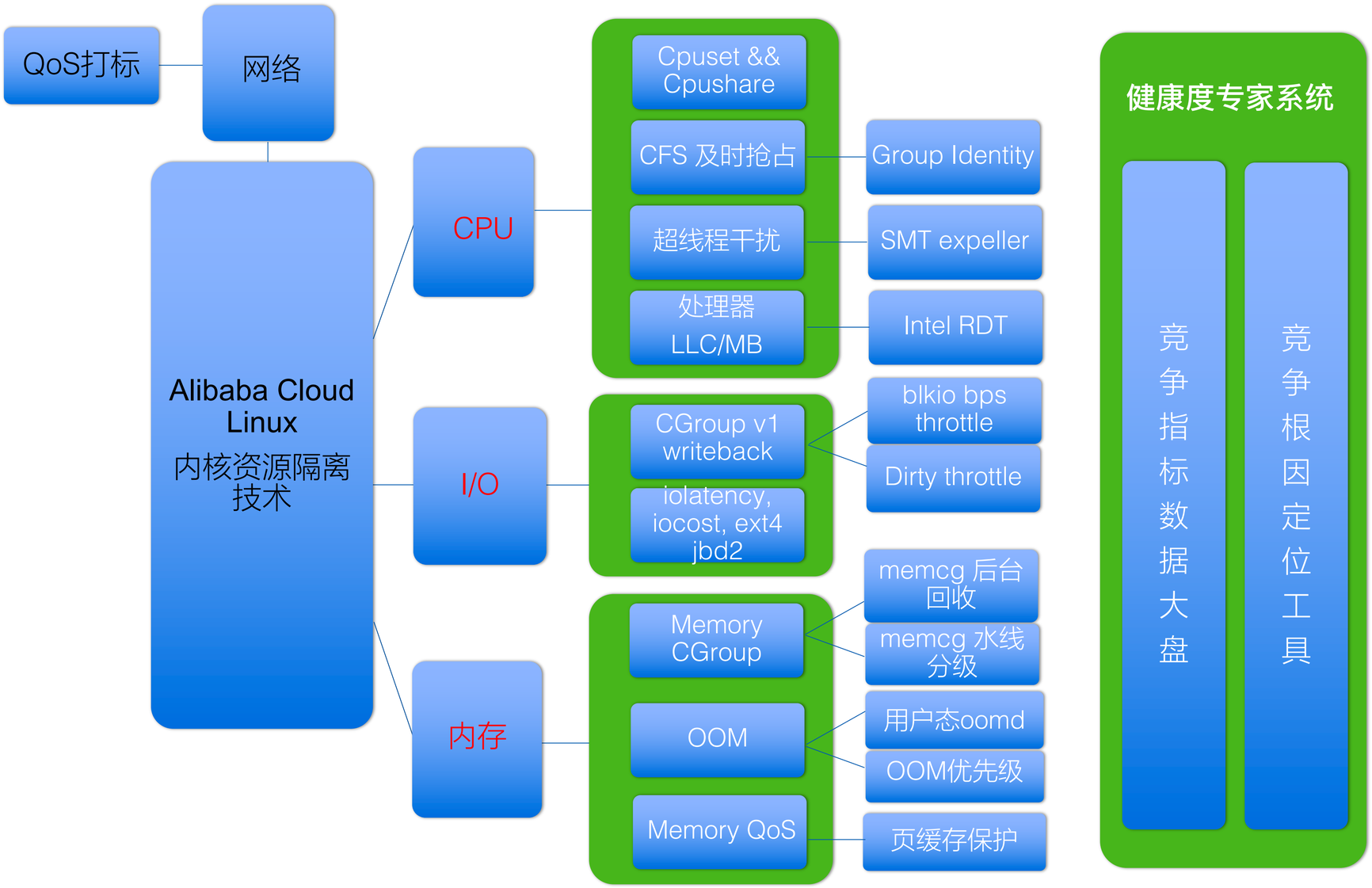


其它类型的资源使用水位也很低

混部的价值：提升资源利用率

混部方案强依赖于内核各个子系统的资源隔离技术(based on Linux CGroup): CPU, 内存, I/O等。

Alibaba Cloud Linux 资源隔离技术方案

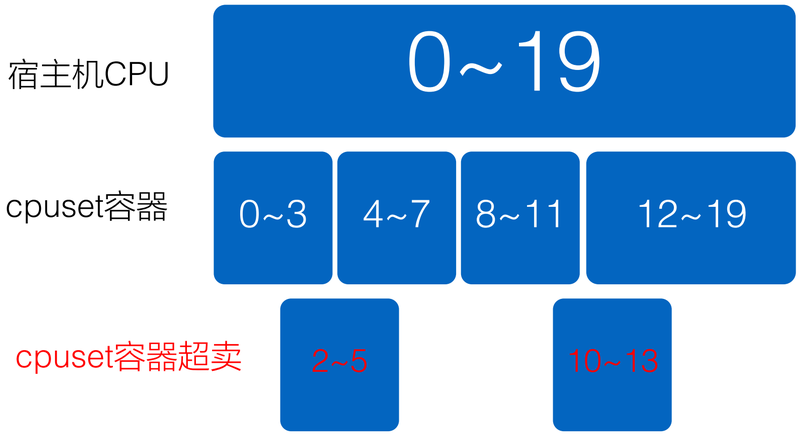


CPU

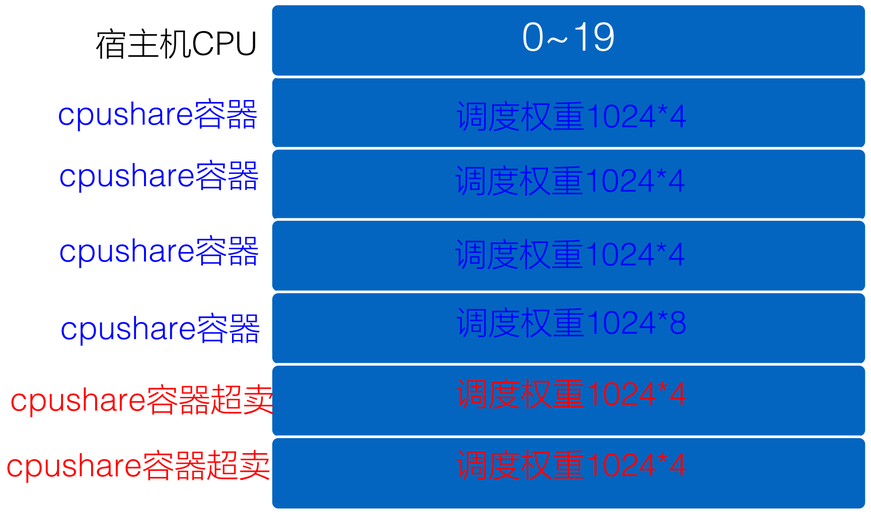
- Cpuset and Cpushare
- CPU Quota
- CGroup Identify
- SMT expeller
- Intel RDT

Cpuset and Cpushare

Cpuset方式



Cpushare方式



技术对比	**优　点**	**缺　点**
Cpuset	- 部署简单 - 调度隔离性好 - Cache亲和性好	- 资源管理复杂，资源池机器资源碎片化加重 - 规格粒度粗，资源利用率低 - 动态扩容和动态缩容困难 - 超卖后的长尾时延差
Cpushare	- CPU资源共享，资源利用率高 - 可以提升有突发请求的业务性能 - 超卖方案灵活，扩展性好 - 容器调度粒度控制精细化	- 应用并发度增加，容易失控 - 调度争抢加剧 - Cache污染

CGroup Identify

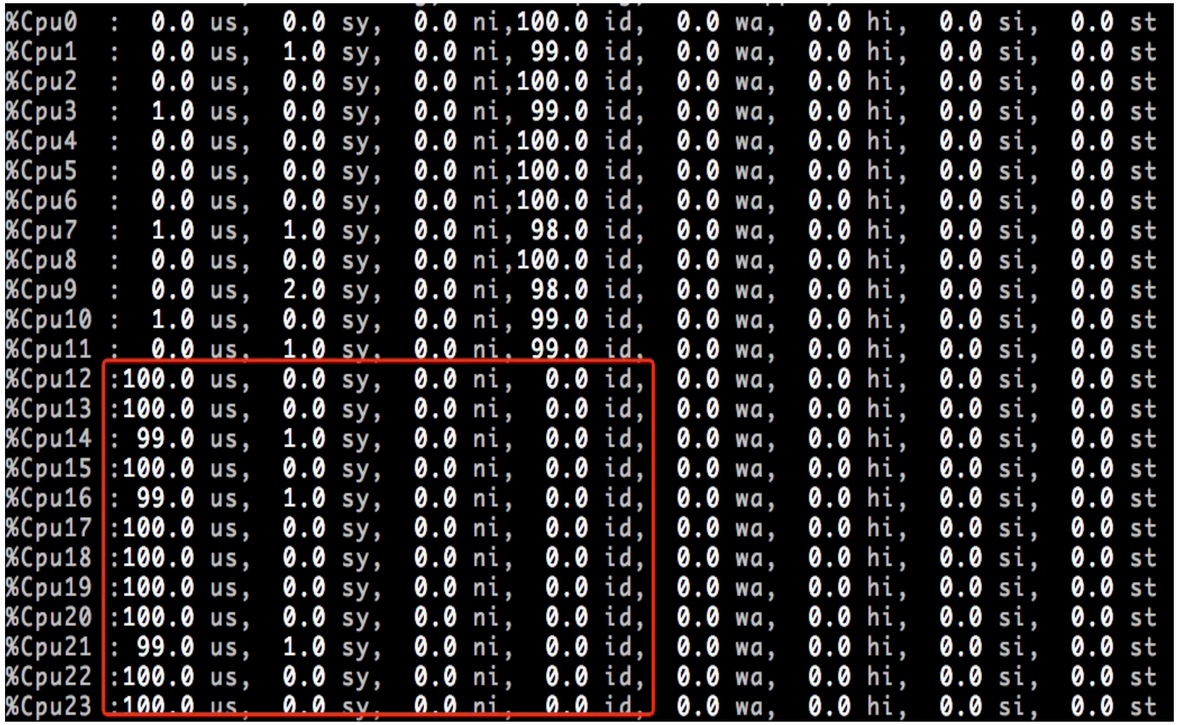
以CPU CGroup组为单位实现调度特殊优先级，提升高优先级组的及时抢占能力：

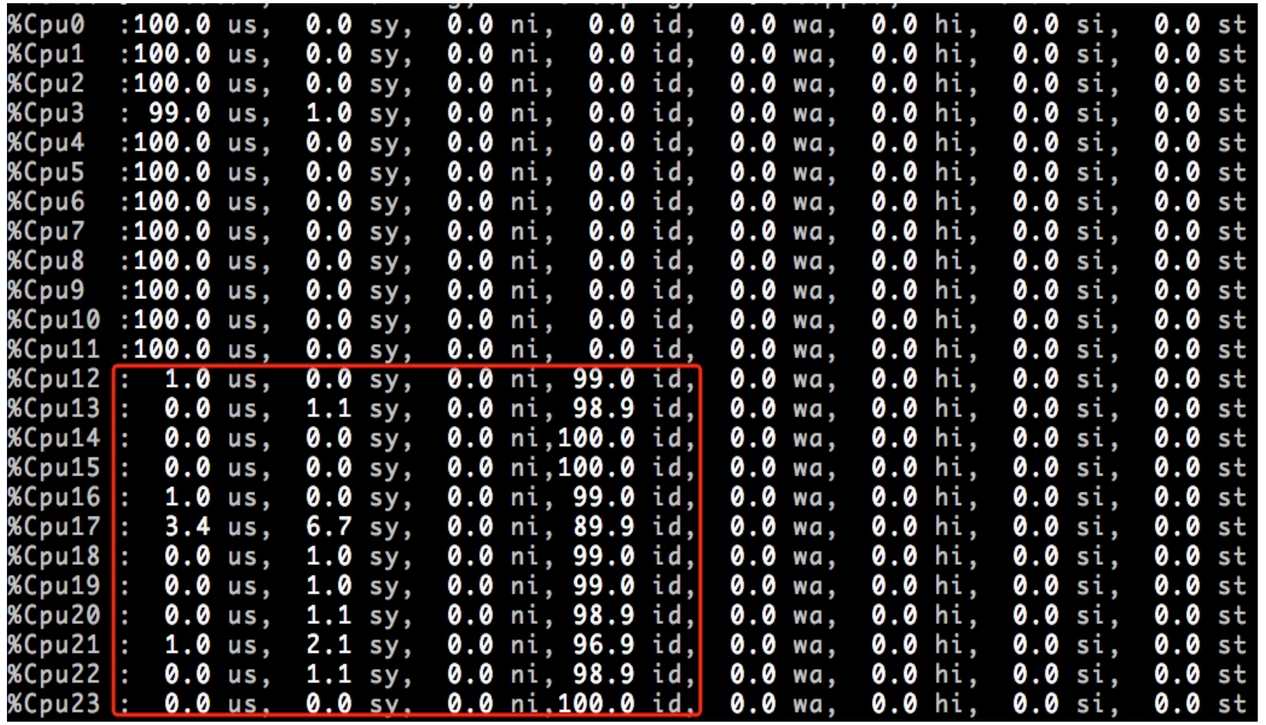
- 基于Linux CFS调度器实现，代码易维护
- 惩罚低优先级组的vruntime
- 增强高优先级组的抢占条件

测试数据：
使能本功能后，混部集群的电商业务RT(Response Time)可以下降15%，基本恢复至非混部集群的水平。

SMT expeller

关于SMT超线程资源的干扰，我们实测过某容器的QPS下降45%，RT增加60%。





CPU0~11分别同CPU12~23互为SMT siblings
- 左图，在CPU12~23上跑满离线作业。
- 右图，接下来在CPU0~11上启动在线作业，可以看到离线作业被抑制住。

Intel RDT

适用于RT时延比较敏感的业务场景:
- CAT, MBA
- RDT只能静态划分资源，resctl接口。
- 动态RDT需求，类似于Cpushare方式
- Intel新处理器的HWDRC(Hardware Dynamic Resource Controller)

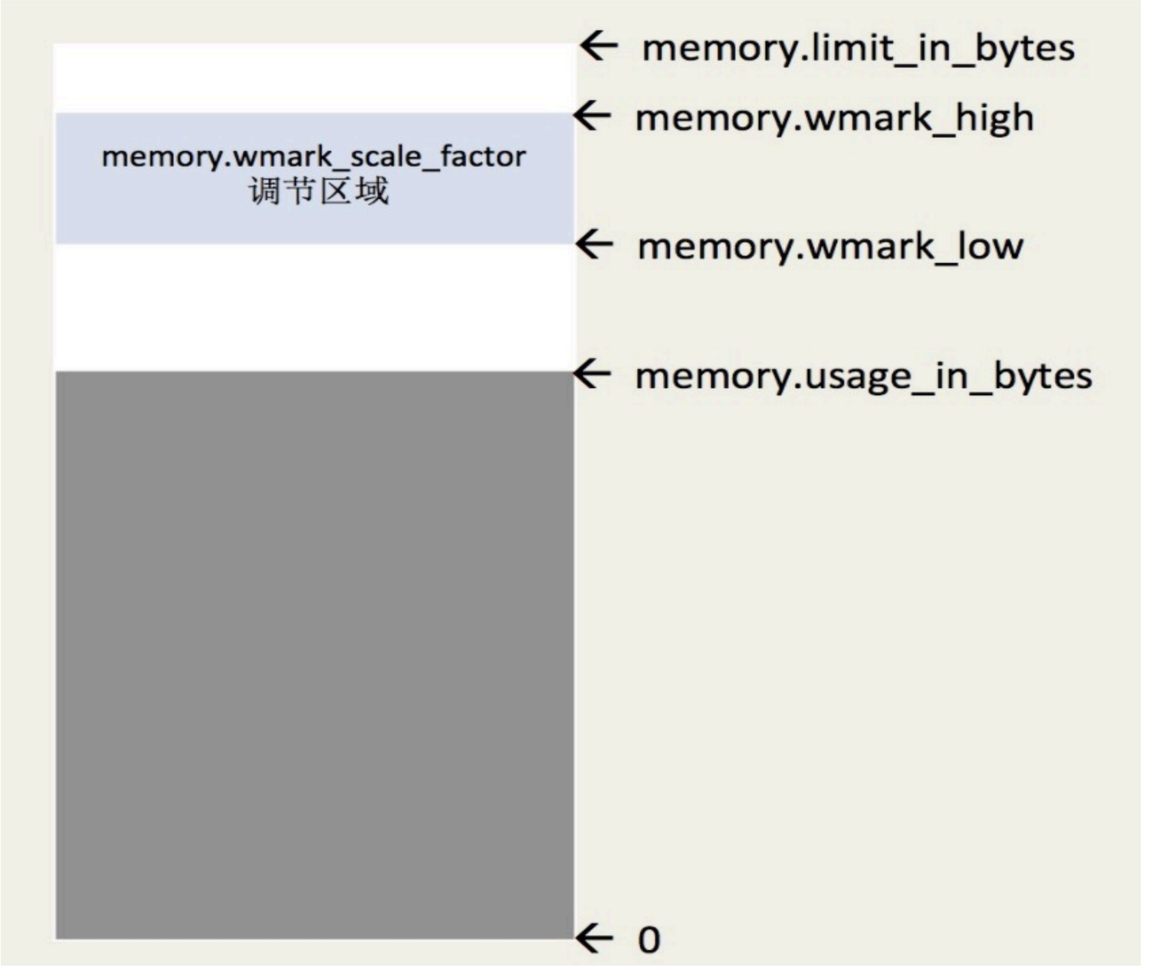
内存

- memcg 后台回收
- memcg 水线分级
- QoS: CGroup v1 memory.min and memory.low, unevictable process
- memcg OOM 优先级
- user-space oomd based on PSI

memcg 后台回收

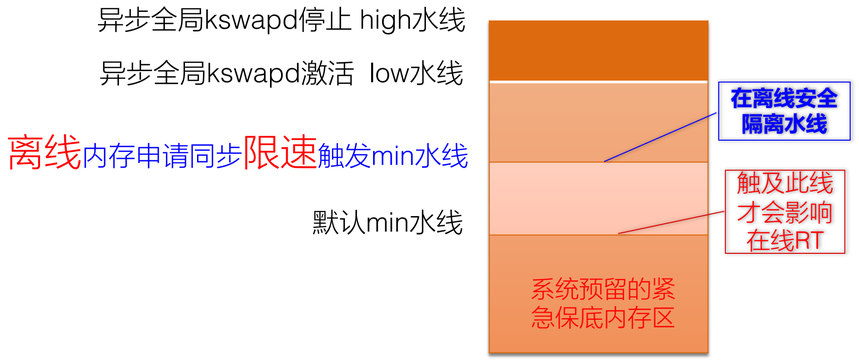
内存慢速路径上的直接回收(direct reclaim)会严重影响业务性能和RT(Response Time):

- global direct reclaim 和 memcg direct reclaim
- 在memcg维度实现异步内存回收功能
- 引入memcg的usage high 和 low水线
- 类似kswapd内核线程的方式进行后台异步回收
- 缺点：异步回收的开销不好归属至本memcg

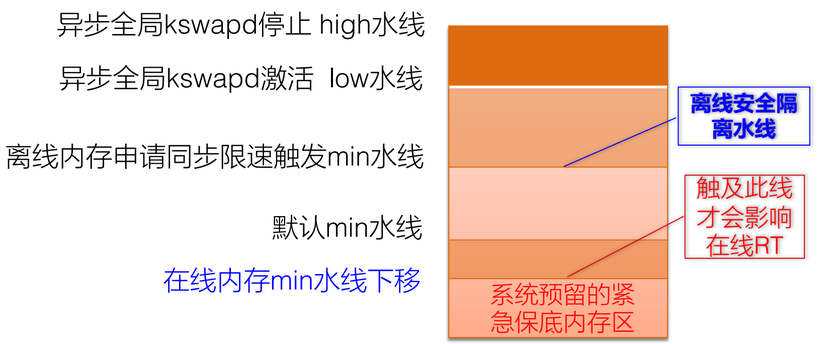


memcg 水线分级 – QoS保障

混部水线参数调优后保障了大盘，但仍然存在着局部的随机抖动需要解决。



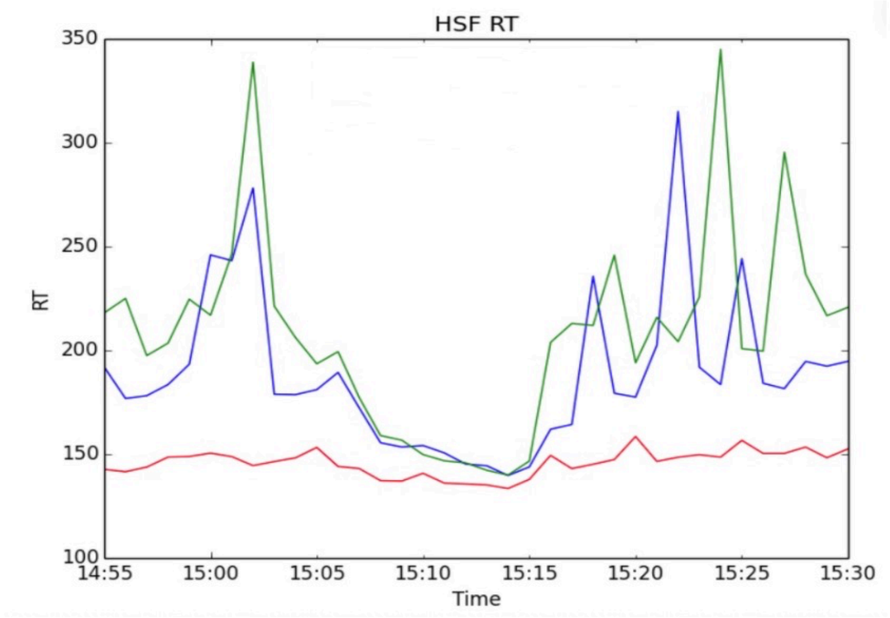
低优先作业的全局min水线上移
触及水线后施加限速



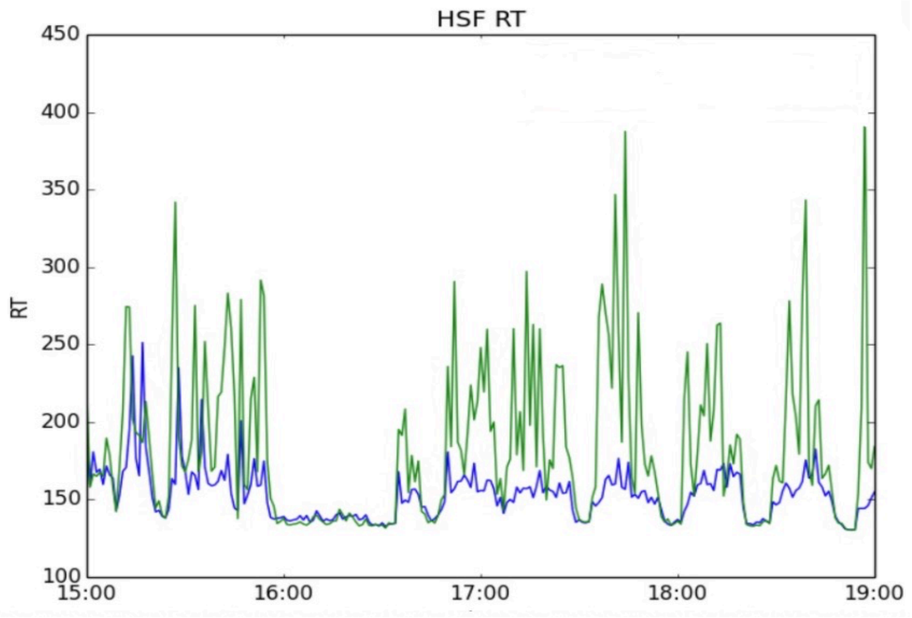
高优先级的全局min水线下移

memcg 水线分级 – 功能效果

新接口memory.wmark_min_adj，范围为[-25, 50]。
- 配合较高的全局min水线（例如4GB~8GB)，通过vm.min_free_kbytes设置
  -25 means WMARK_MIN is "WMARK_MIN + (WMARK_MIN - 0) * (-25%)”
  50 means WMARK_MIN is "WMARK_MIN + (WMARK_LOW - WMARK_MIN) * 50%”
- 负值代表min水线下移，代表的是对内存QoS要求高的memcg组(例如在线组)。
- 正值代表min水线上移，代表的是对内存QoS要求低的memcg组(例如离线组)。



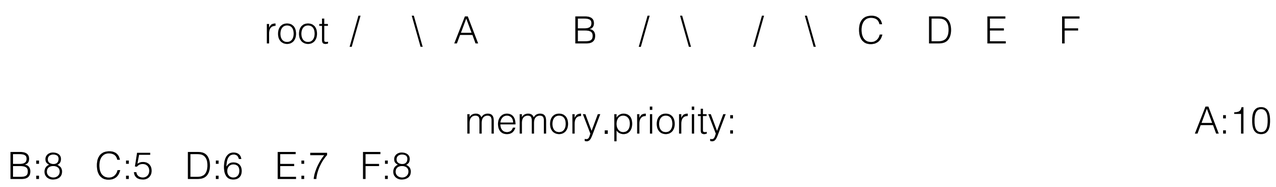
- 红线 为无任何内存压力的在线HSF RT曲线，平稳在150附近
- 绿线 为全局大内存压力下的未部署方案时在线HSF RT曲线，产生了较大波动
- 蓝线 为水线等优化手段后的在线HSF RT曲线，依然有大波动



- **绿线** 为未部署任何方案时在线HSF RT变化曲线
- 蓝线 为整套memcg 水线分级方案下在线HSF RT变化曲线

memcg OOM优先级

- k8s利用oom_score_adj设计Guaranteed/Burstable/Best-Effort不同类型OOM优先级方案:
  https://github.com/kubernetes/community/blob/master/contributors/design-proposals/node/resource-qos.md
- 内核态设计OOM优先级，支持Global OOM和CGroup OOM，相比k8s更灵活高效。
- 新接口memory.priority，范围[0, 12]，数值越大优先级越高。
- 优先级的生效范围为同级memcg之间。



memory.priority: A:10
B:8 C:5 D:6 E:7 F:8

当发生Global OOM时，从根组(root)开始按每个层级数据最小的原则遍历路径
root->B->E，最终E被选出来作为被杀对象。

oomd

基于Memory CGroup PSI在用户态实现oom killer:
https://engineering.fb.com/2018/07/19/production-engineering/oomd/

关键难点：
- PSI数据的准确性和及时性
- Bad case的安全兜底，试想一下极端并发匿名页申请导致瞬间OOM(一秒以内)

IO

主要为了解决共享盘/ext4文件系统的场景

- CGroup v1 writeback: buffered writes blkio throttle
- CGroup v1 dirty throttle
- 高版本移植 iolatency, iocost
- Disk quota (容器下的root权限的disk quota hardlimit受限)
- ext4 jbd2 调优

CGroup v1 writeback

Blkio-throttle用来解决共享磁盘带宽的场景，但buffered写是由kthread后台周期性提交IO落盘，
标准CGroup v1无法获得blkio的限流配置。

- 基于cgroup v1接口，实现per-cgroup的writeback
- 将memcg和blkcg两CGroup信息桥接起来，inode bdi记录memcg和blkcg信息，这样就可以在kthread上下文获取相应的blkcg限流配置
- 实现异步写per-blkio的buffered IO throttle
- 接口：blkio.throttle.write_bps_device, blkio.throttle.write_iops_device

配置示例

```
echo 8:32 104857600 > /sys/fs/cgroup/blkio/test/blkio.throttle.write_bps_device
echo 10080 > /sys/fs/cgroup/blkio/test/tasks
echo 10080 > /sys/fs/cgroup/memory/test/tasks
```

其中，8:32为设备的主次设备号，104857600为bps即100MiB/s，10080为业务进程pid。
注：需要将业务进程同时配置到对应的memcg和blkcg中。

CGroup v1 dirty throttle

有了CGroup v1 wirteback之后，就可以进一步在更高层(VFS)实现per-memcg dirty throttle，
极大地缓解共享文件系统(以及blkio-throttle)导致的“IO优先级反转”难题:

- memory.dirty_bytes{ratio}，类似于vm.dirty_bytes{ratio}
- memory.dirty_background_bytes{ratio} ，类似于vm.dirty_background_bytes{ratio}
- 主要逻辑实现位于balance_dirty_pages()

ext4 jbd2 调优

- 调小 nr_requests
- remount参数: lazytime, nobarrier等
- 异步checkpoint
- Upstream ext4 jbd2 fast commit:
  https://lwn.net/Articles/834483/

容器资源视图隔离

top, free, ps, uptime 容器视图:
- /proc/cpuinfo
- /proc/stat
- /proc/<pid>/stat
- /proc/loadavg
- /proc/uptime
- /proc/meminfo
- /sys/devices/system/cpu/online
- …

相比用户态lxcfs方案，实现傻瓜式开箱即用，定制化高级指标。

容器资源视图隔离 – 示例

某个部署在物理服务器上的4c8g规格容器:

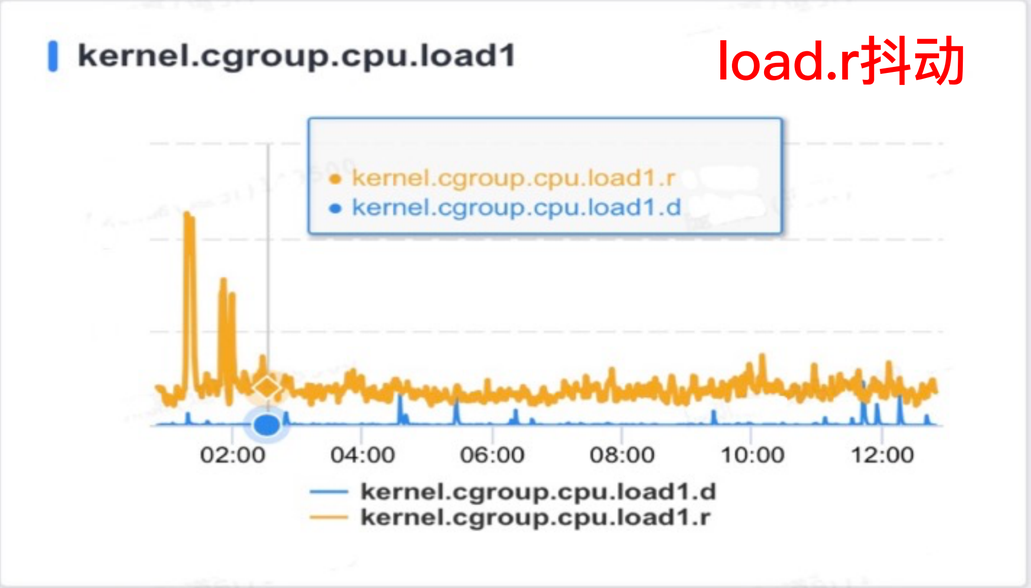
```
top - 16:04:33 up 14 days,  2:17,  1 user,  load average: 0.86, 0.79, 0.65
Tasks:  19 total,   1 running,  18 sleeping,   0 stopped,   0 zombie
%Cpu0  : 14.9 us,  5.9 sy,  0.0 ni, 79.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  : 13.7 us,  5.9 sy,  0.0 ni, 79.4 id,  0.0 wa,  0.0 hi,  0.0 si,  1.0 st
%Cpu2  : 13.9 us,  5.9 sy,  0.0 ni, 80.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  : 13.9 us,  5.9 sy,  0.0 ni, 79.2 id,  0.0 wa,  0.0 hi,  0.0 si,  1.0 st
KiB Mem :  8388608 total,   554484 free,  3319112 used,  4515012 buff/cache
KiB Swap:        0 total,        0 free,        0 used.   554484 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 1280 admin     20   0 7382840   2.8g  14340 S  78.2 34.5   9341:20 java
```

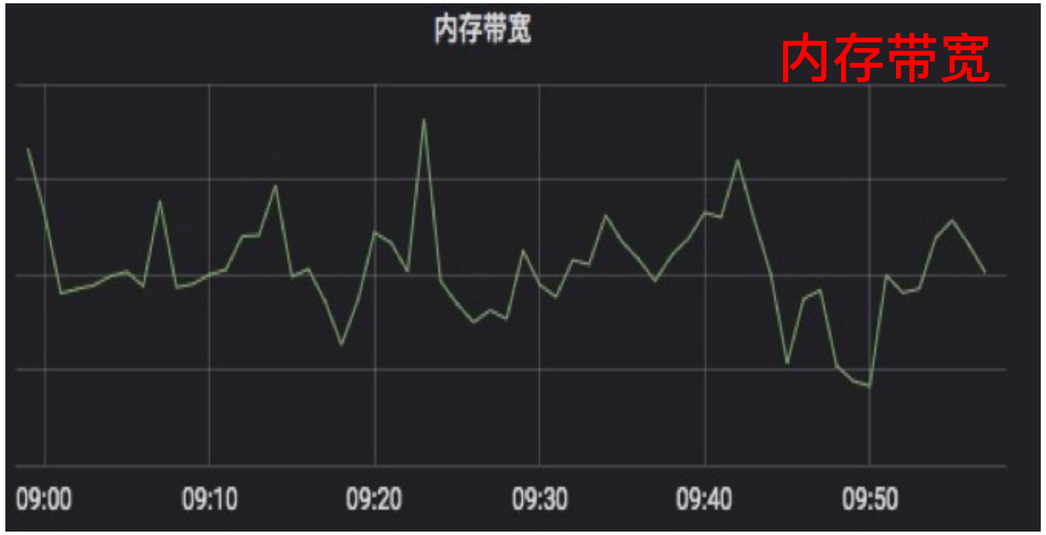
如果没有容器资源视图，看到的将是宿主机的100-CPU的top数据

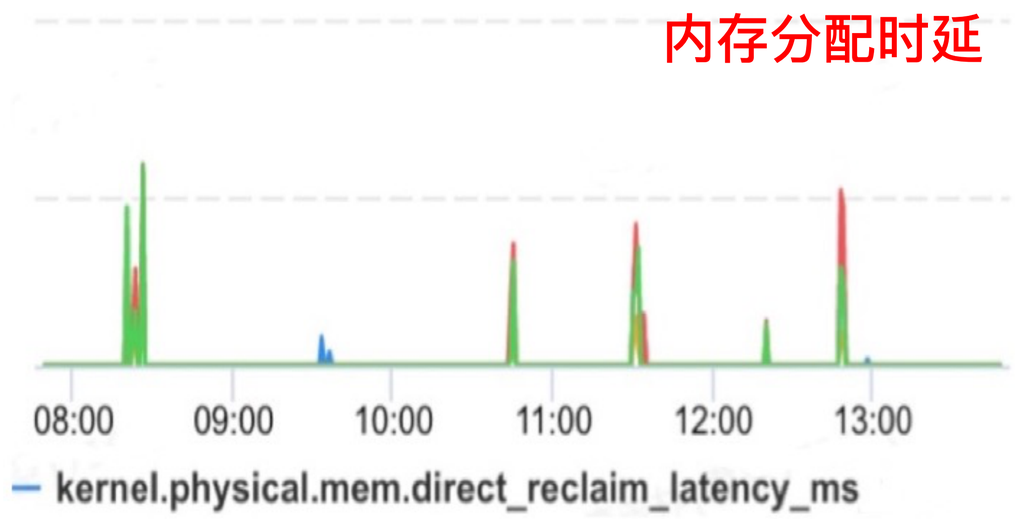
标准化SLI – Global and CGroup

General	CPU	Memory	IO/FS	Network
per_cgroup /proc/stat per_cgroup load.d/load.r softlockup, hung task, rcu stall (proc方式) 整机sys(识别有害sys) CGroup数目监控 微架构PMU资源竞争 (IPC, LLC, 内存带宽)	PSI cpu pressure queue other/queue sibling per-cgroup steal CPU Quota throttle wait/block/ioblock histograms	PSI memory pressure direct reclaim/compaction/swap histograms workingset refault/activate/restore writeback/dirty Slab/Sreclaimable/SUnreclaim Global/CGroup OOM memory leak detection kswapd NUMA info	PSI io pressure disk avgqu-sz/await/util disk io hung ext4 jbd2 transactions (/proc/fs/jbd2/<disk>/stats) blkio iops/bps/throttle	RX/TX bps/pps TCP retrans rate

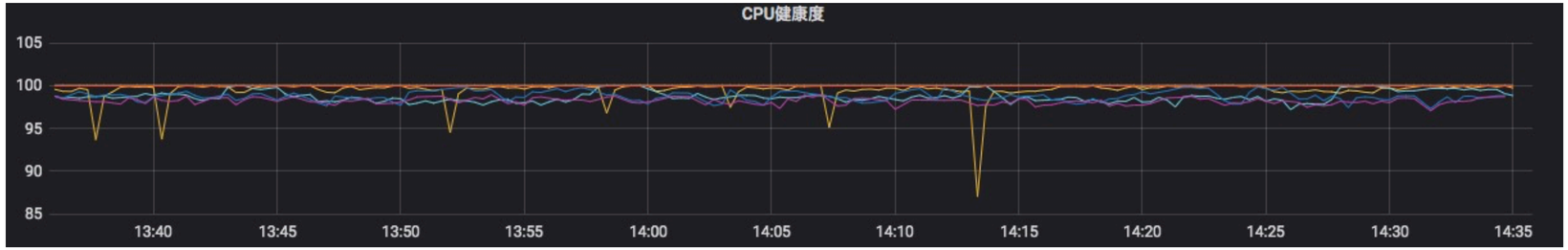
SLI 示例 – 常见监控抖动

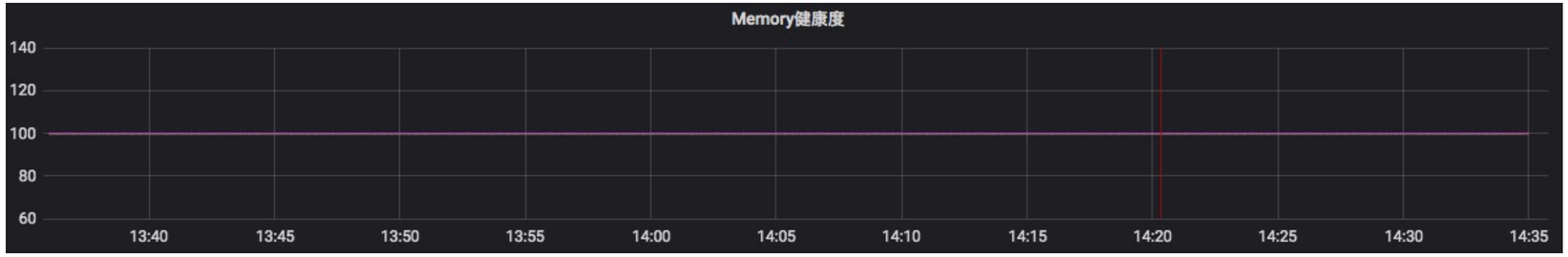


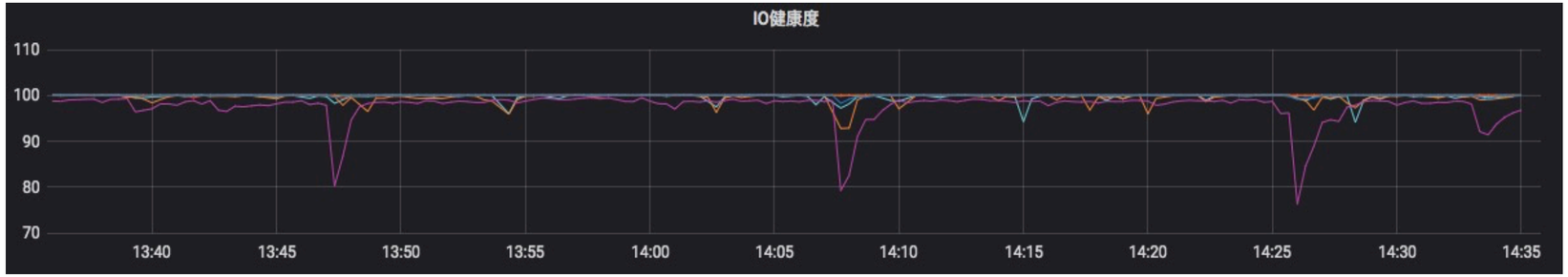




SLI 示例 – PSI标准指标







注：
针对CPU健康度指标，除了调度时延外，还加入了调度阻塞的数据聚合

标准化SLI的数据化价值

数据的价值是无限的

1. 量化出SLO
2. 分析当前集群的问题症结: CPU, Memory, I/O等
3. 集群聚合数据分析，业务资源Profiling，大盘告警等
4. 数据驱动集群调度(k8s)决策: 流量调度，驱逐策略，资源调优
5. … ...

混部集群稳定性 – 随机抖动难题

异常表现

- LOAD 高
- SYS 高
- iowait/disk await 高
- TCP重传高
- Memory CGroup 综合症(cgroup_mutex大锁，memory.stat时延，memcg kmem问题，memcg残留等)

因素太多

CFS算法时延，内核关抢占，全局锁竞争(spinlock, mutex, semaphore)，中断、软中断，关中断，调度阻塞，调度抑制，内存回收，内存碎片，大内存释放，ext4/jbd2 wait transaction, io hung等。

工具定位根因

- 轻量级资源消耗, 不影响性能，能够稳定常态化部署
- 设定指标异常门限，抓取异常现场CallTrace
- in-kernel blackbox triggers, ali-diagnose等工具

```
[2019-06-23 16:41:14.443264346]
CPU: 6 PID: 124002 state 2 comm: dbench load.d=53 Call Trace:
io_schedule+0x2a/0x80
wait_on_page_bit+0x82/0xa0
__filemap_fdatawait_range+0xff/0x170
filemap_fdatawait_range+0x14/0x30
filemap_write_and_wait_range+0x57/0x90
ext4_sync_file+0xe8/0x3a0 [ext4]
vfs_fsync_range+0x4b/0x160
do_fsync+0x3d/0x70
SyS_fsync+0x10/0x20
do_syscall_64+0x74/0x180
entry_SYSCALL_64_after_swapgs+0x58/0xc6
0xffffffffffffffff

[2019-06-23 16:41:14.443270180]
CPU: 22 PID: 124003 state 2 comm: dbench load.d=53 Call Trace:
io_schedule+0x2a/0x80
do_get_write_access+0x1f4/0x450 [jbd2]
jbd2_journal_get_write_access+0x53/0x70 [jbd2]
```

```
===detect                                suspicious long single run 1208ms
stacktrace: ext4_es_lru_add+0x57/0x90 [ext4] ext4_ext_map_blocks+0x76e/0xed0 [ext4]
ext4_da_get_block_prep+0x390/0x4c0 [ext4] __block_write_begin+0x1a7/0x490 ext4_da_write_begin+0x15f/0x350 [ext4]
generic_file_buffered_write+0x11d/0x290 __generic_file_aio_write+0x1d5/0x3e0 generic_file_aio_write+0x5d/0xc0
ext4_file_write+0xb8/0x460 [ext4] do_sync_write+0x8d/0xd0

=== [redacted] blocked 770ms, waker interrupt
wakee stacktrace: sleep_on_shadow_bh+0xe/0x20 [jbd2] do_get_write_access+0x2dd/0x4e0 [jbd2]
jbd2_journal_get_write_access+0x27/0x40 [jbd2] __ext4_journal_get_write_access+0x3b/0x80 [ext4]
ext4_reserve_inode_write+0x70/0xa0 [ext4] ext4_mark_inode_dirty+0x53/0x220 [ext4]
ext4_dirty_inode+0x48/0x70 [ext4] __mark_inode_dirty+0x16f/0x330 generic_update_time+0x87/0xe0
ext4_ext_update_time+0x2b/0xc8 [ext4] file_update_time+0xbd/0x120 __generic_file_aio_write+0x198/0x3e0
generic_file_aio_write+0x5d/0xc0 ext4_file_write+0xb8/0x460 [ext4] do_sync_write+0x8d/0xd0
vfs_write+0xbd/0x1e0 SyS_write+0x7f/0xe0 system_call_fastpath+0x16/0x1b 0xffffffffffffffff          (null)

===detect                                suspicious long single run 100ms
stacktrace: memcg_stat_show+0x1ff/0x330 cgroup_seqfile_show+0x73/0x80
seq_read+0xfa/0x3a0 vfs_read+0x9c/0x170 SyS_read+0x7f/0xe0
system_call_fastpath+0x16/0x1b 0xffffffffffffffff
```

深水区混部

>=50%的整机CPU水位

1. 资源超卖
   - CPU资源超卖(相对容易)
   - 内存资源超卖
   - 不要高估NVMe盘性能(随机写更慢，<200MiB)，不要低估共享文件系统的挑战：“做IO分盘(独立云盘)”
   - 网络QoS，独立的虚拟网卡
2. 安全兜底方案，能够及时处理“bad case”
3. 用户无感

内存资源超卖

1. 基于业务资源画像的方式
   - 业务Profiling数据驱动
   - 适用于业务场景相对稳定

2. 基于打标的内存安全回收技术
   - 实现相对通用的内存超卖方案
   - 支持 page cache, mlocked, anonymous, slab四种类型
   - 轻量级安全回收

3. 内存带宽瓶颈的影响非常大
   - 常态化内存带宽监控，识别/优化/抑制捣蛋作业。
   - Dynamic RDT: Intel HWDRC (icelake)
   - 用更先进的硬件(更高DDR带宽, 加速器offload)

我们的内存超卖方案已落地部署数个集群

两种混部方案对比

混 部 方 案	主 要 优 点	主 要 缺 点
基于容器	1. 性能好，资源利用率高 2. 资源弹性和超卖能力强 3. 资源共享灵活	1. 安全能力差 2. 故障隔离差(宕机、配置等) 3. 消除集群随机抖动要求很高
基于虚拟化	1. 满足业务安全需求 2. 实现内存隔离 3. 消除host全局锁竞争 4. 内核宕机、环境配置等隔离	1. 性能损失(5%~10%) 2. 资源弹性和超卖能力弱 3. 资源共享性能差 4. 二级调度加剧guest全局锁 5. guest+runtime资源内耗 6. 内存大页碎片化问题

混部方案落地情况

普通混部:
(有部分CPU超卖，并且实际上也存在由于根组/监控/管控等导致的隐式内存超卖)
已在集团内部落地十几个集群规模，大促态和日常态 均常态化运行。

深水区混部：
(全面的显式CPU/内存资源超卖)
已在集团内部落地数个集群规模，大促态和日常态 均常态化开启内存资源超卖。

未来工作

1. 虚拟化场景下的云产品资源安全超卖和高密部署
2. 虚拟化场景下的云产品资源弹性
3. 内存大页的反碎片化

Thank you!

Questions?

欢迎加入共建

Alibaba Cloud Linux 钉钉群

Alibaba Cloud Linux 资源隔离SIG：
https://openanolis.org/sig/resources-co-location/overview/

阿里云

奥运会全球指定云服务商

WWW.ALIYUN.COM