从内核的角度压榨内存

李泽帆
字节跳动

内存优化的重要性

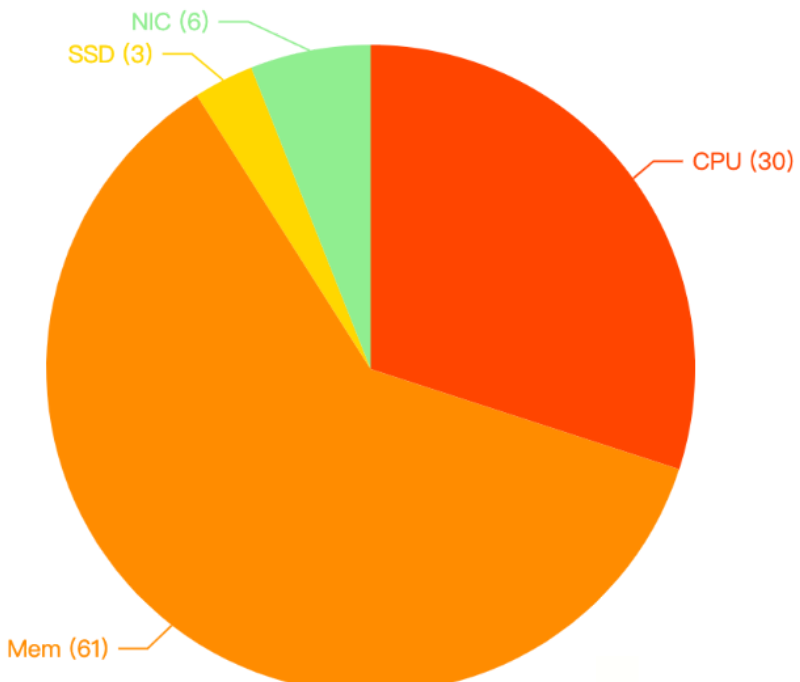
内存是服务器成本的大头

新机型的核存比问题

内存紧张带来的稳定性问题：

业务延迟抖动 -> OOM -> livelock

优化的困难之处：内存是一种弹性很差的资源



注意内核内存泄漏

ipmi内存泄漏：ipmi发送命令后等待硬件响应，如果硬件无响应，引用计数不会减1，大量的消息堆积造成内存泄漏。

```
VmallocChunk:          0 kB
Percpu:            20954624 kB
HardwareCorrupted:     0 kB
```

```
Slab:              15976640 kB
SReclaimable:        465456 kB
SUnreclaim:        15511184 kB
KernelStack:          22352 kB
```

```
  OBJS ACTIVE  USE OBJ SIZE  SLABS OBJ/SLAB CACHE SIZE NAME
2026159 2025881   0%    4.00K 295976        8   9471232K kmalloc-4096
4196396 4175038   0%    1.00K 135641       32   4340512K kmalloc-1024
5476528 4206450   0%    0.50K 171290       32   2740640K kmalloc-512
```

2023-04-12	ipmi: fix SSIF not responding under certain cond.
2022-10-17	ipmi: fix msg stack when IPMI is disconnected
2022-10-17	ipmi: fix memleak when unload ipmi driver
2022-10-17	ipmi: fix long wait in unload when IPMI disconnect

无用数据占用大量cache - negative dentry

negative dentry：查找不存在的文件产生negative dentry，在没有内存压力的情况下可以一直堆积，占用大量内存，并且遍历dentry可能导致soft lockup

类似问题：cgroup/pids/system.slice/system-bpfd-xxx目录下dentry太多，在删除system-bpfd-xxx这个目录的时候需要扫描很多dentry耗时太长，导致其他进程在访问system.slice这个目录的时候D住

```
crash> list -h 0xffff9029771d0510 | wc -l
203607
crash> list dentry.d_child -s dentry.d_name.name -H ffff9031942e3060
ffff9029771d0480
  d_name.name = 0xffff9029771ccad0 "bpfd-acl-vendor-file-save@tc.2.service"
ffff902971397680
  d_name.name = 0xffff902972571650 "bpfd-acl-vendor-file-save@tc.1.service"
ffff9019e0172e40
  d_name.name = 0xffff90118c1dca90 "bpfd-acl-vendor-file-save@tc.service"
ffff902978cf18c0
  d_name.name = 0xffff902972917e90 "bpfd-acl-vendor-file-save@tc.2.service"
ffff8ff4edfa1080
  d_name.name = 0xffff8fe17c36e6d0 "bpfd-acl-vendor-file-save@tc.1.service"
ffff8ff56ee69ec0
```

无用数据占用大量cache - zombie memcg

删除cgroup后，很多时候memcg不会被释放，而memcg结构体非常庞大（主要是里面的一些per-cpu成员），长期下来积累大量zombie memcg

曾经kmem charging导致的zombie memcg，已经被解决：

- [PATCH v3 00/19] The new cgroup slab memory controller
- [PATCH v5 0/7] Use obj_cgroup APIs to charge kmem pages

page cache的问题，怎么办：

- 主动回收内存后，memcg引用计数被释放
- [PATCH v6 00/11] Use obj_cgroup APIs to charge the LRU pages

还有shared memory的问题……

```
#subsys_name    hierarchy       num_cg
cpuset  5       168     1
cpu     6       1082    1
cpuacct 6       1082    1
blkio   2       1052    1
memory  3       83531   1
devices 11      1012    1
freezer 7       142     1
net_cls 8       142     1
perf_event      10      142     1
net_prio        8       142     1
hugetlb 9       141     1
pids    4       1054    1
```

```
MemTotal:       399551072 kB
MemFree:          1537912 kB
MemAvailable:     2008832 kB
Buffers:           450556 kB
......
VmallocUsed:       731772 kB
VmallocChunk:           0 kB
Percpu:         24569088 kB
HardwareCorrupted:      0 kB
```

LRU链表的内存优化

每个挂载点都使用两个LRU链表分别跟踪inode和dentry，这个LRU链表与memcg数量成正比：

num_numa_node * memcg_nr_cache_ids * sizeof(kmalloc-32)

在某机器上：

4 * 24574 * 32 bytes ~= 3MB

该机器有952个super block，因此LRU总消耗内存：

952 * 2 * 3MB ~= 5.6GB

→

解决方案：对于一个super block，不需要跟踪所有的memcg

- [PATCH v5 00/16] Optimize list lru memory consumption

To highlight the significance of this patchset, we ran a simple test on a 32-cpu Intel Xen machine with 188G of RAM. We sequentially created and deleted 400 lxc containers.

kmalloc-32 object count

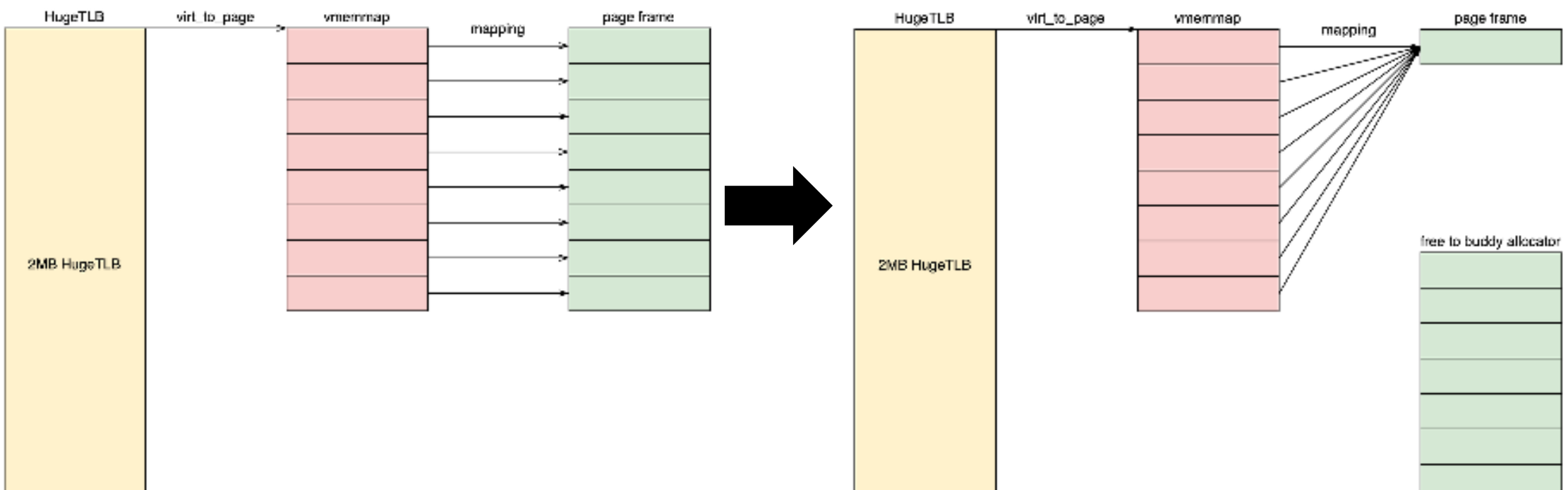
	Prior to test start	During test	After test
UEK7 (prior to this patchset)	23,424	7,236,608	583,296
UEK7 (with this patchset)	11,392	82,944	82,080

For the above example, this patchset reduced the number of kmalloc-32 objects from 7.2 million down to approximately 83,000, a savings of ~227 MB and roughly 87x fewer kmalloc-32 objects!

HugeTLB Vmemmap Optimization

HVO：

- 使得一个大页只需要一个struct page表示，而不是每4K一个struct page
- 对于1GB HugeTLB，内存收益HugeTLB总内存的~1.6%。对于2MB HugeTLB，收益为~ 1.4%。



HugeTLB Vmemmap Optimization

HVO的缺点：只能用于HugeTLB，而HugeTLB的使用场景很有限 -> 虚拟机

THP是不是也能支持HVO?

THP也有缺点：运行时overhead，内存碎片化

Flexible THP：struct page -> struct folio

PTE页表回收

案例：进程虚拟地址空间达到55T，实际使用物理内存仅590G，页表占用内存达到100G

原因：业务使用jemalloc/tcmalloc，后者通过madivse(MADV_DONTNEED)而不是munmap释放内存

方案：[PATCH v1 0/7] synchronously scan and reclaim empty user PTE pages

```
VmData: 189532484 kB
VmStk:       132 kB
VmExe:     43076 kB
VmLib:      4912 kB
VmPTE:    310764 kB
VmSwap:        0 kB
```

页表共享

Oracle发现在他们的数据库场景，有大量重复的页表，极端情况PTE可以占用878GB+内存

开源数据库呢？基于多进程的数据库都有类似问题，例如PostgreSQL、PolarDB

On a database server with 300GB SGA [Oracle system global area], a system crash was seen with out-of-memory condition when 1500+ clients tried to share this SGA even though the system had 512GB of memory. On this server, in the worst case scenario of all 1500 processes mapping every page from SGA would have required 878GB+ for just the PTEs. If these PTEs could be shared, the amount of memory saved is very significant.

用户态内存：冷内存卸载

16年：Google：Software-defined far memory in warehouse-scale computers

18年：阿里，冷内存项目

21年：Meta，TMO（Transparent Memory Offloading）

用户态内存：冷内存卸载（字节）

基于Meta的TMO方案，结合字节的业务和线上问题做二次设计和优化：

- 除了PSI，也通过page refaults、filemap等等信息判断当前业务的内存压力
- 实现多级卸载：ZRAM内存长时间冷则二次卸载到磁盘TMO多级卸载之zswap，进一步释放内存
- 内存冷热依据是LRU链表，有时会出现过度回收的情况
  - 使用memory.min对业务内存进行保护
  - 屏蔽对可执行文件页的回收
  - 计划采用MGLRU
- 减少内存扫描压力
  - 为了回收匿名页冷内存，需开启swap。问题：内存即将到low水位会进行kswapd全局回收，达到min水位后进程会进行主动的内存回收，开启swap导致扫描的链表变长，系统进入不稳定不稳定状态甚至长时间无法无响应的情况大幅增加
- ……

用户态内存：冷内存卸载（字节）

线上~24W服务器，内存节省~9PB

持续上线中……

用户态内存：虚拟机场景下的冷内存卸载

虚拟机内的冷内存，如何回收给host?

用户态内存：KSM

KSM at Meta

- instagram业务在老机器上的内存和cpu压力都很大
- 业务由一个controller和32个以上的worker进程组成，每个worker都将interpreter加载到内存，此外还有很多其他相同的数据结构
- 结果：在64GB的机器上节省6GB内存

问题

- 安全风险
- overhead，对线上业务的影响
- 适用的普适性

Metric	Value
pages_shared	73,670
pages_sharing	2,110,000
pages_unshared	1,390,000
pages_volatile	7,070,000
Parameter	**Value**
pages_to_scan	var
sleep_millisecs	20

CXL能否派上用场

当前：扩充内存，缓解cpu核多、内存少的问题？
未来：内存池化？

业界的一些尝试：

- Meta：TPP: Transparent Page Placement for CXL-Enabled Tiered-Memory
- Microsoft：Pond: CXL-Based Memory Pooling Systems for Cloud Platforms

Thanks!