drm/cgroup memory management

Dave Airlie
Red Hat

GPU memory history

- Traditionally client devices
- Unified memory GPUs (UMA/APU)
  - Laptops/Desktops
  - Embedded devices
- Discrete memory GPUs
  - Laptops/Desktops
    - PCIe
  - VRAM - Device memory
- Uncached memory
- GPU virtual memory
  - No useful pagefaults

Developments from HPC/AI

- Server GPUs
  - 8 Discrete compute GPUs in a case
    - PCIe
    - Nvlink
    - Infinity Fabric
- APU
  - CPU/GPU Coherent memory architectures
  - CPU x86 or ARM
  - AMD MI300A
  - NVIDIA Grace Hopper / Grace Blackwell

Suddenly cgroups matter…

Worse

Suddenly NUMA matters…

DRM memory management

- GEM objects
  - Buffer object used by userspace
  - TTM backed for systems with discrete memory
  - TTM provides device/system RAM migration support
  - TTM provides cached/uncached allocations and pooling
- TTM bo shrinker - called from driver
  - Tries to free objects under memory pressure
  - Objects marked by userspace as discardable
  - Push objects to swap if needed/available

TTM Uncached memory

- Scenarios on x86-64 where GPU access over PCIe is unsnooped
- APUs where uncached memory has less overheads
- Moving pages to uncached means removing them from linear mapping
  - set_memory_uc/wc APIs
- This has overheads, so use pools to keep a set of these available.
- Pools are NUMA specific
- Pools get filled on object free
- TTM pool shrinker
  - Reclaim pools that just converts back to cached and frees them

What can cgroups do?

- Device memory allocations with a controller
  - dmem controller
- How do we track system allocations?
  - Use memcg
  - Sounds simple…

Problems/What if?

- Allocate a buffer object that can migrate (device/system)
  - Avoid double accounting
  - Object will mostly live in device RAM
  - Can be evicted at any time by another process
    - Who accounts for the eviction?
    - If process B causes process A object to be evicted?
      - What happens if process A has no system RAM budget left?
    - Process A will remigrate into VRAM when scheduled

Uncached/Cached pool overheads

- Uncached memory pool shared among all processes
- DRM sees this as a core kernel workaround
  - Kernel should support uncached pages in the allocators
- Scenario
  - cgroup A allocates uncached memory
  - Frees it into the shared pool
  - cgroup B allocates cached memory
    - System under memory pressure
    - B has to enter reclaim to empty pool and migrate pages back to cached
- Uncached page pool warmer - allocate 1GB of uncached and free
- Multiple process needing uncached pages for scanout

How to move forward?

- Start with accounting
  - Add memcg counters for GPU active and GPU reclaimable
    - GPU reclaim is the pooled memory stores
  - Feedback: no memcg counters without vm counters
  - Add VM counters for GPU active and GPU reclaim
  - Use these in TTM
  - Then add memcg counters for them and use them?

Not so fast…

- memcg has NUMA awareness
- TTM has some NUMA awareness

Two great tastes

- Port TTM to use list lru for the pools
- Make the pool shrinker NUMA aware
- Track max allocated pages per NUMA node
- Use the LRU/NUMA shrinker for TTM pool
- When allocating an uncached page if the cgroup doesn’t have any, check the parent cgroup
    - This lets uncached pages in a cgroup get handed to the parent for others to use.

Driver opt-in

- Drivers decide which GEM object should be memcg tracked
  - All objects controlled by userspace get tracked
- TTM page populating gets a memcg flag
- Pages are populated
  - First allocation of system objects
  - Eviction from device memory for dmem objects
- Currently only account on first allocation
- amdgpu + xe support on list

Future?

- Get the code out there into production and see what it breaks?
- Can we handle evictions, any ideas?
- Does the uncached stuff matter anymore?
    - Will the new code the lots of clients needing uncached pages regularly?