Finding more DRAM
Mobile devices to Data centers

Suren Baghdasaryan, Shakeel Butt, Yu Zhao
Google
Why more DRAM?

● Mobile Devices
  ○ Application RAM demand keep increasing
  ○ Limited form factor
  ○ No way to add more physical RAM

● Datacenter
  ○ DRAM cost is a major factor of the total datacenter cost
  ○ Over-provision and under-utilized
  ○ Expensive ECC RAM
Current status

- Kill background apps (Android)
  - More cold starts -> slow and power hungry
- Overcommit memory (data centers)
  - Global memory pressure -> Direct reclaim -> No performance isolation
  - High refault cost due to slow storage devices
Solution: Proactively reclaim memory

- Reclaim **unneeded** memory proactively which is **very cheap** to refault.
- Approaches
  - Userspace driven proactive reclaim (**Android**)
    - Unneeded: memory of background apps
    - Cheap refaults: in-memory compression (zram)
  - Kernel driven proactive reclaim (**Google datacenter kernel**)
    - Unneeded: maintains idle age of whole memory
    - Cheaper: in-memory compression (zswap)
Android story

Cold starts are slow and power hungry

Can we keep more background apps alive while not affecting interactive ones?

Let’s shrink them!
Where all the memory is being used?
Userspace driven proactive reclaim

Idea:
keep more RAM available for interactive applications by hinting kernel about processes unlikely to be used in the near future

Solution:
proactively reclaim application memory after its transition into non-interactive state

Implementation:
new process_madvise() APIs:
• MADV_COLD and MADV_PAGEOUT merged into mm tree
• process_madvise() syscall under development

decreased number of kills under memory pressure

Results:
• 15% less kills from the dogfood population
• up to 30% less kills while running stress tests
Experiment #1: proactively reclaim all pages

<table>
<thead>
<tr>
<th>App cycle order:</th>
<th>ABCABC</th>
<th>ABCBA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Launch time</td>
<td>No change</td>
<td>8% regression</td>
</tr>
<tr>
<td># of kills</td>
<td>50% less kills</td>
<td>19% more kills</td>
</tr>
<tr>
<td>notes</td>
<td>major faults for file-mapped pages increased by 6%</td>
<td></td>
</tr>
</tbody>
</table>

Observation:
file LRU is very small  
memory spikes result in more kills

Conclusion: proactively reclaiming file-backed pages is not a good idea.

How about deactivating file-backed pages instead?
Experiment #2: reclaim anonymous and deactivate file-backed pages

<table>
<thead>
<tr>
<th>App cycle order:</th>
<th>ABCABC</th>
<th>ABCBA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Launch time</td>
<td>29% decrease</td>
<td>15% decrease</td>
</tr>
<tr>
<td># of kills</td>
<td>21% less kills</td>
<td>40% more kills</td>
</tr>
</tbody>
</table>

**Observation:** app-compaction is trying to allocate more memory while system is under heavy memory pressure, making the situation worse and increasing lmk kills

**Conclusion:** userspace should avoid doing app-compaction while the system is under heavy memory pressure
Results

When app-compaction is timed correctly we get 15% less kills from the dogfood population and up to 30% less kills while running stress tests

A popular game: ~1.8GB in game, 1.7GB in background, ~700MB compacted

No noticeable penalty on warm start observed for high-end devices

Reduced number of cold starts
Implementation: process_madvise()

Use process_madvise() with MADV_COLD and MADV_PAGEOUT to hint the kernel that process won’t be used in the near future.

- MADV_COLD deactivates active pages speeding up their reclaim in case of memory pressure
- MADV_PAGEOUT reclaims private pages immediately

Status: MADV_COLD and MADV_PAGEOUT options for madvise() are in MM tree, process_madvise() syscall is under development
Proactive reclaim for Data Centers

- Finding memory to reclaim proactively
- Reclaim memory
Google data center Memory Overcommit Model

- Replace part of DRAM with cheap slow memory (or far memory)
- Memory Provisioning
  - Quota request
  - Translates to
    - Cheap slow memory is completely **transparent** to the users
      - Examples: ZSWAP, PMEM(slower+cheaper), swap (remote/local).
- Size of cheap slow memory == Idle Memory Estimation
Idle Memory in across Google's Datacenters

**Opportunity:**
32% of memory usage is idle

**Challenge:**
Frequent accesses to idle memory
Existing mechanisms

- Why not kswapd
  - Reclaim based on watermarks
  - Aims to balance nodes
  - Too many complicated heuristics

- Page Idle Tracking
  - High CPU overhead
  - High memory overhead

```python
for pf in Machine:
    flag   = read("/proc/kpageflags")
    cgroup = read("/proc/kpagecgroup")
    idle   = read("/sys/kernel/mm/page_idle(bitmap")
    // Track idle pages for each page and their cgroups.
```
Our Approach

- **kstaled** (in-kernel page idle tracking)
  - No memory overhead (by storing age in page flags)
  - **CPU overhead** is similar to Page Idle Tracking

- **kreclaimd**
  - A kernel thread scanning all PFNs and reclaiming idle pages

- **Per-memcg knobs**
  - Idle age threshold for reclaiming anon and file pages
  - Page idle age histograms
kstaled CPU overhead

- CPU usage increases linearly with the RAM size
  - Spends 100% of a CPU for 512 GiB
- CPU usage increases linearly with the scan frequency
  - Spends 80% of a CPU for 60 sec cycle
- 50% time spent in rmap_walk
kstaled optimizations

- Remove rmap_walk from kstaled
  - Link all PMD pages into per-node linked list
  - Traverse PMD linked list to extract access bit
  - Traverse PFNs to age all the pages
- 3.5x CPU usage reduction by kstaled
  - Allows to scan larger systems or with higher frequency
- Remove PFN scanning from kreclaimd
  - kstaled passes idle pages to kreclaimd through a queue
- 1.5x CPU usage reduction by kreclaimd
Upstream concerns (during LSFMM'19)

- Bypassing existing LRUs
- High CPU cost
- Potential way forward
  - Decouple ageing from memory pressure
  - Use ages to sort LRUs
  - User controlled trigger for ageing and LRU sorting
Discussion points

1. Avoid direct reclaim at all cost.
2. `process_madvise()` parameters - vector (efficiency) vs a single VMA (simple error handling).
3. VMAs might change between reading `/proc/maps` and `process_madvise()` and hint might be applied to a wrong VMA. Possible solutions:
   a. Limit `process_madvise()` to non-destructive hints only to minimize the price of an unlikely mistake
   b. Introduce a new syscall to get MM snapshot ID and pass it to `process_madvise()` to detect possible VMA changes