Augmented Page Reclaim

We would like to share a work with you and see if there is enough interest to warrant a run for the mainline. This work is a part of result from a decade of research and experimentation in memory overcommit at Google: an augmented page reclaim that, in our experience, is performant, versatile and, more importantly, simple.

Performance

On Android, our most advanced simulation that generates memory pressure from realistic user behavior shows 18% fewer low-memory kills, which in turn reduces cold starts by 16%. This is on top of per-process reclaim, a predecessor of `MADV_COLD` and `MADV_PAGEOUT`, against background apps.

On Borg (warehouse-scale computers), a similar approach enables us to identify jobs that underutilize their memory and downsize them considerably without compromising any of our service level indicators. Our findings are published in the papers listed below, e.g., 32% of memory usage on Borg has been idle for at least 2 minutes.

On Chrome OS, our field telemetry reports 96% fewer low-memory tab discards and 59% fewer OOM kills from fully-utilized devices and no UX regressions from underutilized devices. Our real-world benchmark that browses popular websites in multiple tabs demonstrates 51% less CPU usage from `kswapd` and 45% (some) and 52% (full) less PSI on v5.11-rc6 built from the tree below.

Versatility

Userspace can trigger aging and eviction independently via the `debugfs` interface [note] for working set estimation, proactive reclaim, far memory tiering, NUMA-aware job scheduling, etc. The metrics from the interface are easily interpretable, which allows intuitive provisioning and discoveries like the Borg example above. For a warehouse-scale computer, the interface is intended to be a building block of a closed-loop control system, with a machine learning algorithm being the controller.

Simplicity

The workflow [note] is well defined and each step in it has a clear meaning. There are no magic numbers or heuristics involved but a few basic data structures that have negligible memory footprint. This simplicity has served us well as the scale and the diversity of our workloads constantly grow.

Repo

git pull https://linux-mm.googlesource.com/page-reclaim refs/changes/80/1080/1

Gerrit: https://linux-mm-review.googlesource.com/c/page-reclaim/+/1080

FAQ

What is the motivation for this work?

In our case, DRAM is a major factor in total cost of ownership, and improving memory overcommit brings a high return on investment. Moreover, Google-Wide Profiling has been observing the high CPU overhead [note] from page reclaim.

Why not try to improve the existing code?

We have tried but concluded the two limiting factors [note] in the existing code are fundamental, and therefore changes made atop them will not result in substantial gains on any of the aspects above.

What particular workloads does it help?

This work optimizes page reclaim for workloads that are not IO bound, because we find they are the norm on servers and clients in the cloud era. It would most likely help any workloads that share the common characteristics [note] we observed.

How would it benefit the community?

Google is committed to promoting sustainable development of the community. We hope successful adoptions of this work will steadily climb over time. To that end, we would be happy to learn your workloads and work with you case by case, and we will do our best to keep the repo fully maintained. For those whose workloads rely on the existing code, we will make sure you will not be affected in any way.

References

1. Long-term SLOs for reclaimed cloud computing resources
2. Profiling a warehouse-scale computer
3. Evaluation of NUMA-Aware Scheduling in Warehouse-Scale Clusters
4. Software-defined far memory in warehouse-scale computers
5. Borg: the Next Generation

---

note See `Documentation/vm/multigen-lru.rst` in the tree.