TAO: THP Allocator Optimizations

Yu Zhao <yuzhao@google.com>

Fun facts

- Some CPU vendor is planning to remove 4KB page size support in the next decade.
- macOS on Apple silicon uses 16KB base page size.
- Windows on IA-64 used 8KB base page size.

At Google:

- Our POWER9 servers used 64KB base page size.
- We are experimenting with 16KB base page size on Android.

Proposition

4KB page size is suboptimal for modern userspace:

- Some archs, e.g., x86_64, doesn't support other base page sizes.
- Switching to a larger base page size, e.g., on arm64., breaks ABI.

A forward-looking OS would offer the opportunity to favor larger logical pages over 4KB base pages:

- Such an OS would be able to treat 4KB page size as legacy, but
- It would not require larger base page size support from h/w or break ABI.

Image source: https://en.wikipedia.org/wiki/I386

Rationale

Favoring THPs over 4KB pages brings:

- Better **overall** performance, and
- Less metadata overhead by, e.g., HVO or **memdesc**.

4KB page allocations are **fairly** penalized because:

- They are the source of fragmentation, and
- Defragmentation comes with a price, e.g., reclaim and/or compaction.

Fragmentation can become **irreversible** unless it’s contained.

A different economy

The aforementioned proposition is the cornerstone of TAO, i.e., it explores the opposite of the existing economy:

		Existing	**TAO**
Allocation	**4KB page**	Relatively cheap	Relatively expensive
	THP	Relatively expensive	Relatively cheap
Conversion	**THP -> 4KB pages**	Relatively cheap	Relatively expensive
	4KB pages -> THP	Relatively expensive	Relatively cheap

Fungibility

Fungibility makes the conversion between 4KB pages and THPs flexible. Rather than biases against THPs, TAO biases against 4KB pages.

	4KB pages -> THP	**THP -> 4KB pages**
In place (when possible)	Recover (TAO)	Split (existing)
Out of place (when possible)	Collapse (existing)	Shatter (TAO)

Recovering is mainly designed for **1GB THPs**, since copying 1GB data is obviously unaffordable for many potential use cases.

Allocation-time hints

__GFP_MOVABLE is **stronger** than __GFP_COMP because the former can make the latter possible (by compaction) but not the vice versa.

	Order 0	**Compound** (weight = 1)
Movable (weight = 2)	2	1 + 2 = 3
Unmovable	0	1

Ideal fallback order:

A. Movable compound (weight = 3, i.e., most desirable)
B. Movable
C. Unmovable compound
D. Unmovable (weight = 0, i.e., least desirable)

Runtime hints

Lifetimes of unmovable allocations can be statistically estimated by sampling, e.g.,

- Sample an allocation site by recording PFN and timestamp.
- Calculate the lifetime of the sample when freeing the page.
- Group unmovable allocations by their estimated lifetimes.

This is another ongoing **research** project (Tetris) at Google.

Interoperability with userspace

Fragmentation is only observable (or measured) systemwide:

- A low priority task can make a high priority task suffer.
- Per-memcg observability (or ideally limit) would be very helpful.

THP fungibility needs to be a cooperation between the userspace and kernel:

- To account for additional runtime behaviors like hotness and lifetime of allocations by, e.g., Profile-Guided Heap Optimization (PGHO).
- To better utilize physical contiguity in a system, e.g., making 1GB THPs possible by additional madvise() flags.

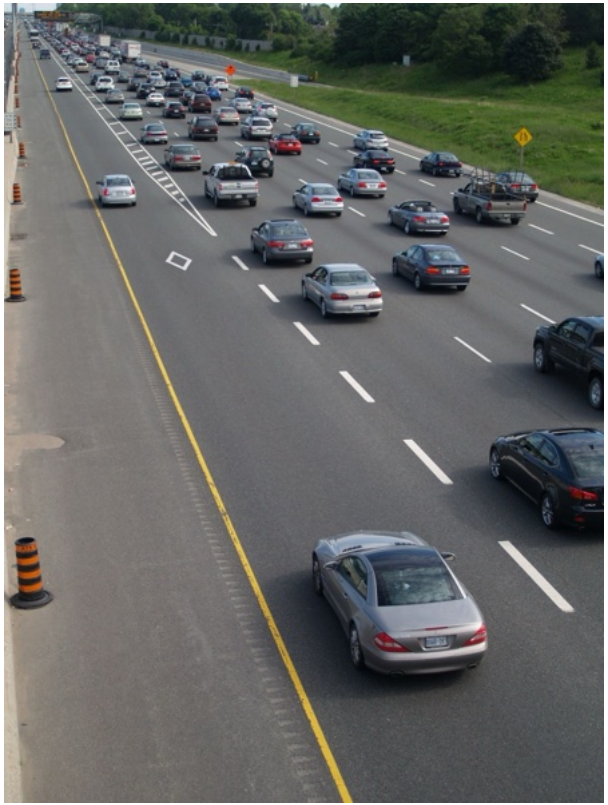
Memory partitioning

Containing (by hardwalling) 4KB pages and THPs in two separate partitions can:

- Provide **guaranteed** THP coverage, and
- Apply differential pressure, i.e., higher pressure to the 4KB page partition.

	Memory utilization	System throughput
Mix (existing)	High	Low
Separate (TAO)	Low	High

Image source: https://en.wikipedia.org/wiki/High-occupancy_vehicle_lane



Sizing and resizing

The sizes of the 4KB page and THP partitions can be based on:

- Global min/max (new knobs) of the THP partition, and
- Per-memcg min/max (new knobs) of the **allotted** THP partition.

The per-memcg min/max prevent priority inversion, i.e., a low priority task consuming more THPs than a high priority task.

Resizing relies on hot removal/plug:

- Shrinking the THP partition (enlarging the 4KB page partition) is theoretically guaranteed.
- The opposite direction is best effort, but still likely to succeed (by Tetris).

Auto resizing and OOM kills

Auto resizing can be done based on memory pressure from respective partitions:

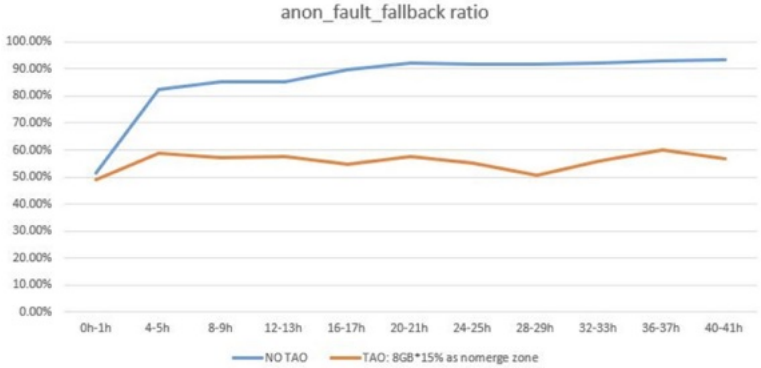
- Differential pressure is required to counteract fragmentation.
- Memory pressure in the 4KB page partition can **optionally** invoke the OOM killer even if the THP partition experiences no pressure.

For some major platforms, (userspace-managed) OOM kills are not only affordable; oftentimes, they are preferred:

- Android LMKs of background apps.
- ChromeOS tab discard of background tabs.
- In Cloud (containerized), realtime jobs preempting batch jobs.

Preliminary results

Android (OPPO)



ChromeOS (Google)

	4KB	**32KB**	**32KB + TAO=30%**
Tab switch time (ms)	111	105 (baseline)	95 (-10%)

Development history

- First attempted to reduce systemwide fragmentation – bunch of random hacks.
- Then attempted to isolate fragmentation between memcg – DEFRAG.
- Arrived at the aforementioned proposition – TAO.
- Still doing the research on estimating lifetimes of unmovable allocations – Tetris.

Related works

1. CMU/Meta's zone-based Contiguitas.
2. OPPO's pageblock-based 64KB large folio pool + dual LRU.
3. Google's pageblock-based DEFRAG (DEcoupled Fragmentation-Resistant Allocation Groups) – basically a rewrite of the page allocator.
4. Google's Tetris (estimate lifetimes of unmovable allocations) – a research project.