beyond malloc efficiency to fleet efficiency

a hugepage-aware memory allocator

A.H. Hunter, Chris Kennelly, Paul Turner, Darryl Gove, Tipp Moseley, Partha Ranganathan



malloc can get slower, while the program gets faster.

normally reducing malloc cycles is good



Date

business logic changes, infrastructure does not

Indexing, Tensorflow model training, Gmail storage, web frontends, Youtube transcode, Bigtable, Colossus, ad model training, ad model serving, interns running mapreduces, streetview, front-end web servers, load balancers, Spanner, memcache, search leaves, search caches, search rankers, log aggregation, real time monitoring, deprecated products, build farms, web crawlers, ...

protobuffers, compression, RPCs, hashing, memcpy and memory allocation

"DATACENTER TAX"

malloc cycles do not matter.

\$\$\$ spent on hardware matter.

Search QPS matters.

Target QPS/(\$\$=core).

most cpu cycles do nothing



Figure 6: Top-level bottleneck breakdown. SPEC CPU2006 benchmarks do not exhibit the combination of low retirement rates and high front-end boundedness of WSC ones.

"Profiling a WSC", ibid.

pagetable walks are expensive (and cached)



Source: https://os.phil-opp.com/ page-tables/

hugepages cheapen the page table walk

- 4 level tree selects 4 KiB page -> 3 level tree selects 2 MiB page
- Shorter dependency chain, smaller physical table

hugepages make the TLB bigger!

0x1000 -> 0x42	22a7000 (4 KiB)	0x200000	->	0xbad6400000	(2
0x2000 -> 0x24	4acc000 (4 KiB)) MiB)			
0x3000 -> 0x32	2a07000 (4 KiB)	0x400000	->	0xcff0e00000	(2
0x4000 -> 0xdi	£38b000 (4 KiB)) MiB)			
0x5000 -> 0xat	£689000 (4 KiB)	0x600000	->	0xddf9000000	(2
0x6000 -> 0xb0)cd3000 (4 KiB)) MiB)			
		0x800000	->	0xd123800000	(2
		MiB)			
		0xa00000	->	0x23f3400000	(2
		MiB)			
		0xc00000	->	0x079a000000	(2
		MiB)			

actually, a hugepage aware allocator is trivial

- 1. Allocate memory in 2 MiB chunks.
- 2. Use transparent hugepages to back those chunks with 2 MiB
- 3. Never talk to the kernel again.

(This is practical--sometimes!)

space efficient hugepage aware allocators are hard



challenge #1: demand oscillates wildly



challenge #2: emptying + density -> binpacking



challenge #3: mistakes can live forever



tcmalloc structure

stacked caches



spans back everything



most allocations are small, and so are most spans



change nothing but the page heap

- New (n) allocates a span of N pages
- Delete(S) returns a New'd span to the allocator.
- Release (N) gives >= N unused pages
 back to the OS

- All called rarely -> we have time to think
- Singlethreaded (and serialized!)
- Most allocators have an interface like this!

Temeraire: the design

even more stacked caches



slack and donation



how does the HugeFiller make decisions?

Goals:

- max P(hugepage becomes free) (return memory to OS!)
- min fragmentation on page (each new allocation creates slack!)

Values:

- Nearly-empty hugepages are precious
- Long free ranges are precious
- (very long free range -> emptier!)

HugeFiller tracks metadata per hugepage

- L, longest free range in pages
- A, total number of allocations
- U, total number of pages used

These are only inputs to the decision problem!

Note: a request K requires L>=K!

Prioritizing high A or high U promotes using full pages

Prioritizing L promotes defragmentation!

A or U: which empties more pages?

Radioactive decay model says:

- Each object has independent halflife
- Dies with some probability p
- Allocation size is irrelevant
- (model is false, but highly useful)

L, longest free range in pages

A, total number of allocations

U, total number of pages used

Conclusion:

- p^5 << p
- A=1, U=10 is much more likely to empty than A=5, U=5

We should favor A (backed by experimentation)

we favor *fragmentation* over *fullness*



best-fit: not just expensive, also a bad idea

For allocation of size 2, this is better:

than this:



not a new theoretical result!

Still surprised us.

results

-

staged rollout

- application case studies
- Global A/B test
- full gradual rollout

saved ~1.3% of cycles



saving memory in the process



virtuous cycles: hugepage coverage

Note: with periodic release off, Temeraire can't do anything *locally*.



Systemwide effects:

- Less kernel-level fragmentation
- Aggressive return at hugepage level

thank you!

Thanks to: Tom Anderson, Atul Adya, Sanjay Ghemawat, Arvind Krishnamurthy, Martin Maas, Petros Maniatis, Phil Miller, and many others

Contact us:

andrewhhunter@gmail.com

ckennelly@google.com



Temeraire Hunter, 2010-2021