



TOKYO, JAPAN / DECEMBER 11-13, 2025

Memory Allocation Profiling upcoming features



TOKYO, JAPAN / DEC. 11-13, 2025

New features being considered / in development

- Allocation context capture
- MEMCG support (two options)
- NUMA awareness
- Efficient data querying and filtering via IOCTL API
- Userspace tooling (grouping allocations by categories, issuing ioctl commands, etc)



TOKYO, JAPAN / DEC. 11-13, 2025

Allocation context capture (testing)

Allocation tags get a new flag bit indicating context capture is enabled

New ioctl interface to set this flag for a specific allocation

Allocation hooks are modified to check the flag and generate a trace event when it's set

BPF program can be attached to the tracepoint to capture:

- pid of the caller
- timestamp
- callstack



The diagram consists of 12 colored boxes arranged in a grid-like pattern. The boxes are colored green, orange, or blue. The text inside the boxes is as follows:

- Row 1: Green box with 'd...', Green box with 'e...'.
- Row 2: Green box with 'd...', Green box with 'd...'.
- Row 3: Green box with 'd...', Green box with 'd...'.
- Row 4: Green box with 'd...', Green box with 'd...'.
- Row 5: Green box with 'd...', Orange box with 'b...', Blue box with 'e...'.
- Row 6: Orange box with 'b...', Blue box with 'e...', Green box with 'd...'.
- Row 7: Blue box with 'e...', Blue box with 's...', Orange box with 'b...'.

There are also some boxes with multiple lines of text, such as the first green box with 'd...' and 'e...', and the second green box with 'd...' and 'd...'.

s...	l...	e...	entry_SYSCALL_64_after_...	...
l...	l...	s...	do_syscall_64	... e...
padze...	l...		__x64_sys_clone	d... d...
clear_user			__se_sys_clone	d... ..
__clear_us...			__do_sys_clone	c... ..

asm_exc_page...	kernel_clone			g...	...
exc_page_fault	copy_process	a...	g...	d...	...
_exc_page_f...	copy_mm	e...	...	d...	...
handle_page_...	dup_mm	b...	...
do_user_addr...	dup_mmap	h...	...	e...	...

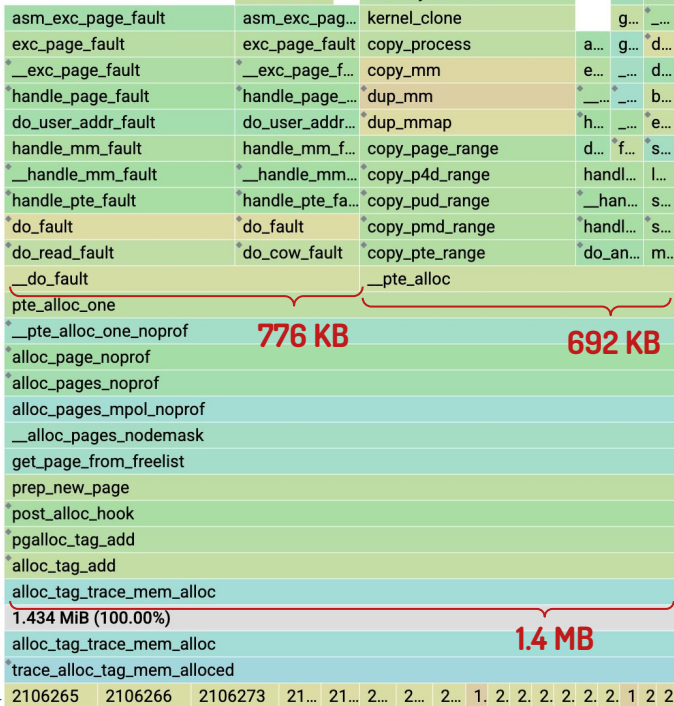
handle_mm_f...	copy_page_range	d...	f...	s...
__handle_mm...	copy_p4d_range	handl...	l...	
handle_pte_fa...	copy_pud_range	__han...	s...	
do_fault	copy_pmd_range	handl...	s...	
do_cow_fault	copy_pte_range	do_an...	m...	

A diagram illustrating memory layout. A horizontal bar is divided into two segments. The left segment is labeled '776 KB' in red text. The right segment is labeled '692 KB' in red text. Above the right segment, the text '_pte_alloc' is written in black. A red bracket spans the entire length of the bar, indicating the total size of 1468 KB.

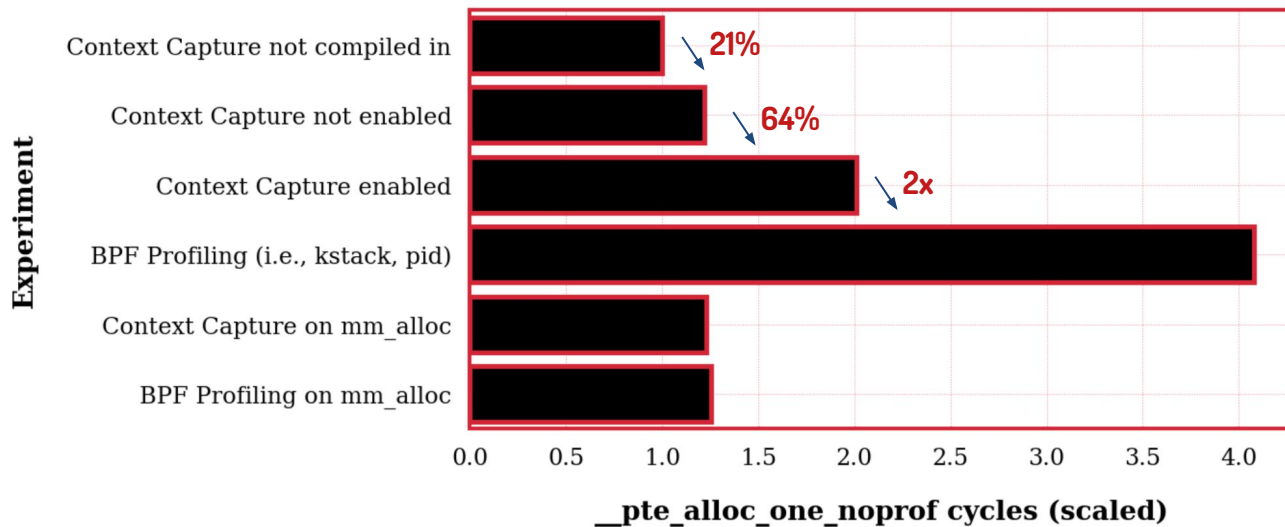
1.4 MB

d

6273	21...	21...	2...	2...	2...	1.	2.	2.	2.	2.	2.	2.	1	2	2
------	-------	-------	------	------	------	----	----	----	----	----	----	----	---	---	---



Profiling Overhead



__pte_alloc_one_noprof	
alloc_page_noprof	
alloc_pages_noprof	
alloc_pages_mpol_noprof	
__alloc_pages_nodemask	
get_page_from_freelist	
prep_new_page	
post_alloc_hook	
pgalloc_tag_add	
alloc_tag_add 1,497,276 (0.006%)	
alloc_tag_trace_mem_alloc	
trace_alloc_tag_mem_allocated	
trace_event_raw_event_alloc_tag_mem_allocated	
trace_event_buffer_reserve	
trace_event_buffer_lock_reserve	
__trace_buffer_lock_reserve	
ring_buffer_lock_reserve	



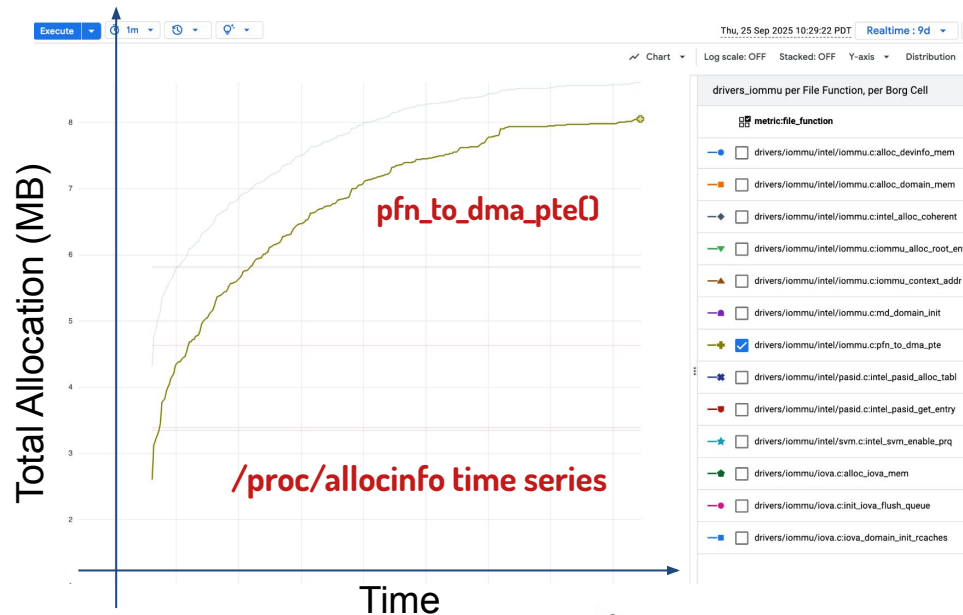
Function	Cycle Count (approximate)
__pte_alloc_one_noprof	18.5%
alloc_page_noprof	18.5%
alloc_pages_noprof	18.5%
alloc_pages_mpol_noprof	1.5%
__alloc_pages_nodemask	1.5%
get_page_from_freelist	1.5%
prep_new_page	1.5%
*post_alloc_hook	1.5%
*pgalloc_tag_add	1.5%
*alloc_tag_add	1.5%
alloc_tag_trace_mem_alloc	1.5%
*trace_alloc_tag_mem_allocated	1.5%
__traceiter_alloc_tag_mem_allocated	1.5%
bpf_trace_run3	1.5%
*__bpf_trace_run	1.5%
bpf_prog_run	1.5%
*__bpf_prog_run	1.5%
*bpf_dispatcher_nop_func	1.5%
[vfat.ko]	1.5%
bpf_get_stackid_raw_tp	1.5%
*__bpf_get_stackid_raw_tp	1.5%
bpf_get_stackid	1.5%
*__bpf_get_stackid	1.5%
get_perf_callchain	1.5%
perf_callchain_kernel	1.5%
unwind_next_frame	1.5%
orc_find	1.5%
*__orc_find	1.5%
*orc_ip	1.5%

> 80% cycles

< 8% cycles (hashtable ops)



Context Capture in action



Context Capture

```
scsi_queue_rq
*scsi_dispatch_cmd
ata_scsi_queuecmd
__ata_scsi_queuecmd
*ata_scsi_translate
ata_qc_issue
ata_sg_setup
dma_map_sg_attrs
intel_map_sg
*domain_sg_mapping
domain_mapping
__domain_mapping
pfn_to_dma_pte
iommu_alloc_page_node_noprof
iommu_alloc_pages_node_noprof
*_iommu_alloc_pages_node_noprof
alloc_pages_node_noprof
__alloc_pages_node_noprof
*_alloc_pages_noprof
__alloc_pages_nodemask
```


Context Capture

scsi_queue_rq
*scsi_dispatch_cmd
ata_scsi_queuecmd
__ata_scsi_queuecmd
*ata_scsi_translate
ata_qc_issue
*ata_sg_setup
dma_map_sg_attrs
intel_map_sg
*domain_sg_mapping
domain_mapping
__domain_mapping
pfn_to_dma_pte
iommu_alloc_page_node_noprof
*iommu_alloc_pages_node_noprof
*__iommu_alloc_pages_node_noprof
*alloc_pages_node_noprof
*__alloc_pages_node_noprof
*__alloc_pages_noprof
__alloc_pages_nodemask

```
perf record -e intel_iommu:map_sg  
✗ Cancels Out  
perf record -e intel_iommu:unmap_sg  
  
// No further investigation needed
```

trace_mm_page_free

*ata_sg_clean
intel_unmap_sg
intel_unmap
queue_iova
*fq_ring_free
iova_entry_free
*iommu_free_pages_list
*__iommu_free_page
*__iommu_free_pages
free_unref_page
*free_unref_page_prepare
*free_pcp_prepare
*free_pages_prepare

Focus investigation here!



東京 2025
LINUX
PLUMBERS CONFERENCE

TOKYO, JAPAN / DEC. 11-13, 2025

MEMCG support option 1: single group tracking

Pros:

Enabled for a single cgroup only

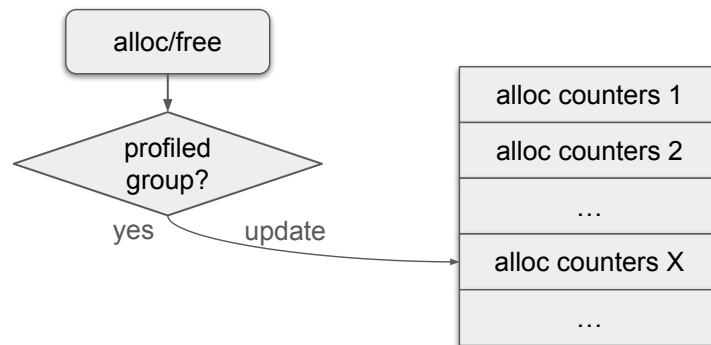
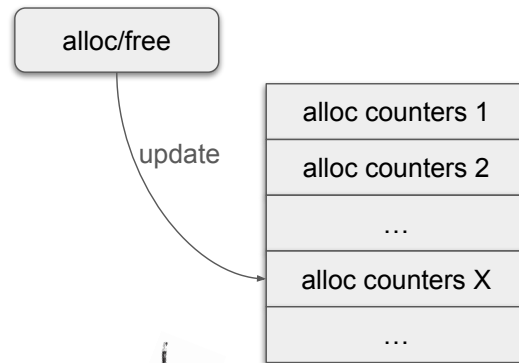
No memory and minimal performance overhead

Simple implementation

Cons:

Limited to a single cgroup at a time

No global profiling when cgroup is profiled



MEMCG support option 2: multi-group tracking

Pros:

Can be enabled for multiple cgroups

Global profiling is not affected

Cons:

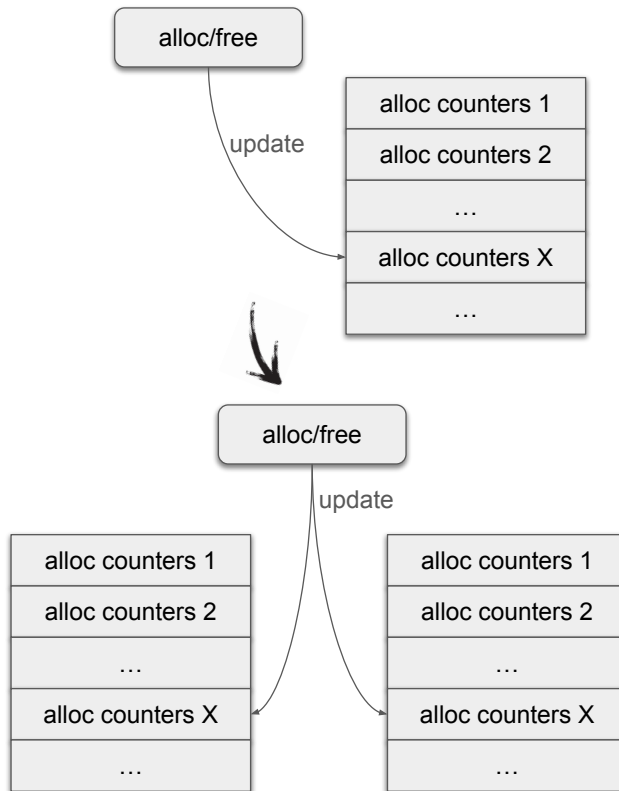
Memory overhead is linear to the number of profiled groups

(each selected memcg gets a new set of per-cpu counters)

Performance profiling overhead for profiled groups roughly doubles

(two sets of counters need to be updated)

Complex implementation



NUMA awareness (proposal)

Proposed at:

<https://lore.kernel.org/all/20250711002322.1303421-1-cachen@purestorage.com/>

Each tag gets per-NUMA node counter

Concerns:

- High memory overhead (linear to the number of nodes)
- Output format changes (unless queried via IOCTL)

Is this useful enough?

Efficient data querying and filtering via IOCTL API

Enabling context capture

Filtering

- by size
- by allocation type (page/slab/per-cpu)

Reporting extra allocation data

- gfp_flags
- unaccounted memory size (early boot or no extension to account)
- per-NUMA information



Userspace tooling (alloctop)

Supported basic features:

- Limiting the number or lines in the output
- Sorting by allocations size, call count, tag name
- Continuous polling

Planned features:

- Grouping allocations by categories (mm/net/drivers/etc.)
- Command-line support for IOCTL commands





Thank you!