Opportunistic Transparent Huge Pages (Part 1)

Shashank Singh (17887) | Advised by Arkaprava Basu
Computer Science and Automation, Indian Institute of Science

BACKGROUND

Increasing memory consumption
+ Deep page tables (5 levels in x86)
+ Fundamental limitations on TLB growth
= **Address Translation Wall**

Approximately 20% of cycles stall on TLB misses on workloads running on warehouse-scale-computers

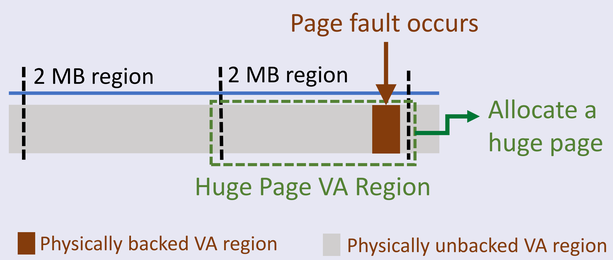
Large Pages → Higher TLB coverage → Fewer TLB misses → Faster Translation → Large Pages

	Relink/ Recompile	Reservation of Memory	Fine-grained Memory Management
hugetlbfs	Yes	Yes	No
syscalls	Yes	Yes	No
THP	**No**	**No**	**Yes**

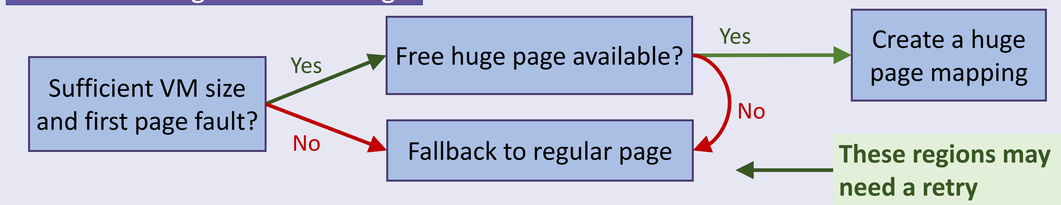
THP in Linux

- THP transparently creates and destroys large page mappings
- Creates large page mappings through following:
  - First page fault
  - Using a background kernel thread called *khugepaged*

Allocation at Page Faults



Need of background scanning



Working of *khugepaged*

- Periodic scanning in FCFS order
- Locks the memory area descriptor *(mmap_lock)*
- Requests free huge page and calls Linux's memory compaction *(kcompactd)* if needed
- Blocks access to the 2 MB virtual memory region while promoting

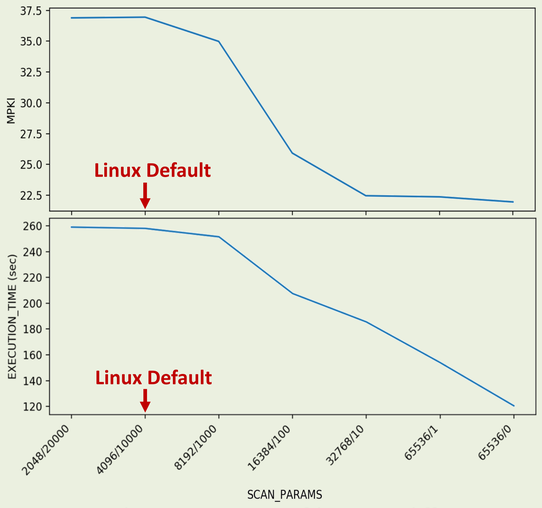
Too many tunables

Tunable Name	Default Value	Description
alloc_sleep_millisecs	60000	Wait time on huge page allocation failure
defrag	1	Whether to run compaction synchronously on allocation
max_ptes_none	511	Limit on number of unmapped regular pages allowed
max_ptes_shared	256	Limit on number of shared regular pages allowed
max_ptes_swap	64	Limit on number of pages that can be brought in from swap
pages_to_scan	4096	Number of pages to scan in each pass
scan_sleep_millisecs	10000	Sleep time between subsequent passes

We will be focusing on these two tunable parameters

MOTIVATION

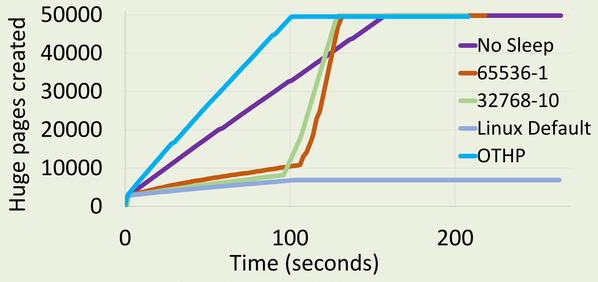
Tuning the tunables



MPKI and execution time of *btree* on different THP settings

- To get the most out of THP, the parameters need to be tuned

Delayed reaction to promotion opportunities



Huge pages created by *btree* on different THP settings wrt time

- There will always be an indefinite delay in any scan-and-promote approach

Unnecessary Background Scanning

- Scanning involves inspecting the full process VA space
- Any change in address space of a process leads to the scanning of the **entire** address space
- Requires holding *mmap_lock* in read mode causing contention if the process vigorously alters its address space

Different application behaviour on different parameter settings

- Scanning approach implicitly assumes all processes behave identically
- However, different applications behave differently on different parameter settings
- Hence, there is a risk of being too conservative and lose out on performance or too aggressive and keep on scanning processes

GOALS & DESIGN PHILOSOPHY

- Shortening the reaction time to a promotion opportunity
- Elimination of background scanning
- Getting rid of the need to **tune the tunables**
- Dynamically adjust the rate of promotion according to application behavior
- Instead of periodic scanning,
  - we latch onto **events of interest** that create opportunities and save them.
  - Then, attempt promotion in a separate execution context
- To realize these goals, we propose an event-driven approach : **Opportunistic Transparent Huge Pages (OTHP)**

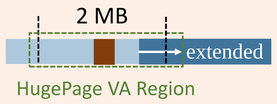
Opportunistic Transparent Huge Pages (Part 2)

Tanmay Sule (17797) | Advised by Arkaprava Basu
Computer Science and Automation, Indian Institute of Science

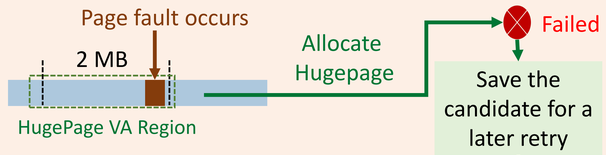
IMPLEMENTATION

Events of Interest

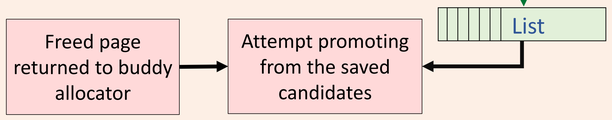
Event 1 Memory Allocation Request

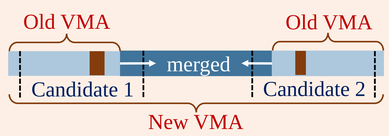


Event 2 A Page Fault



Event 3 Freeing of a page





Physically backed region | Region before vma extension | New region after vma extension

Promotion Mechanism

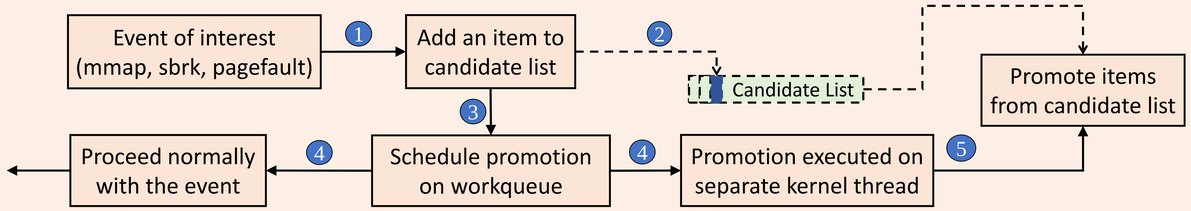
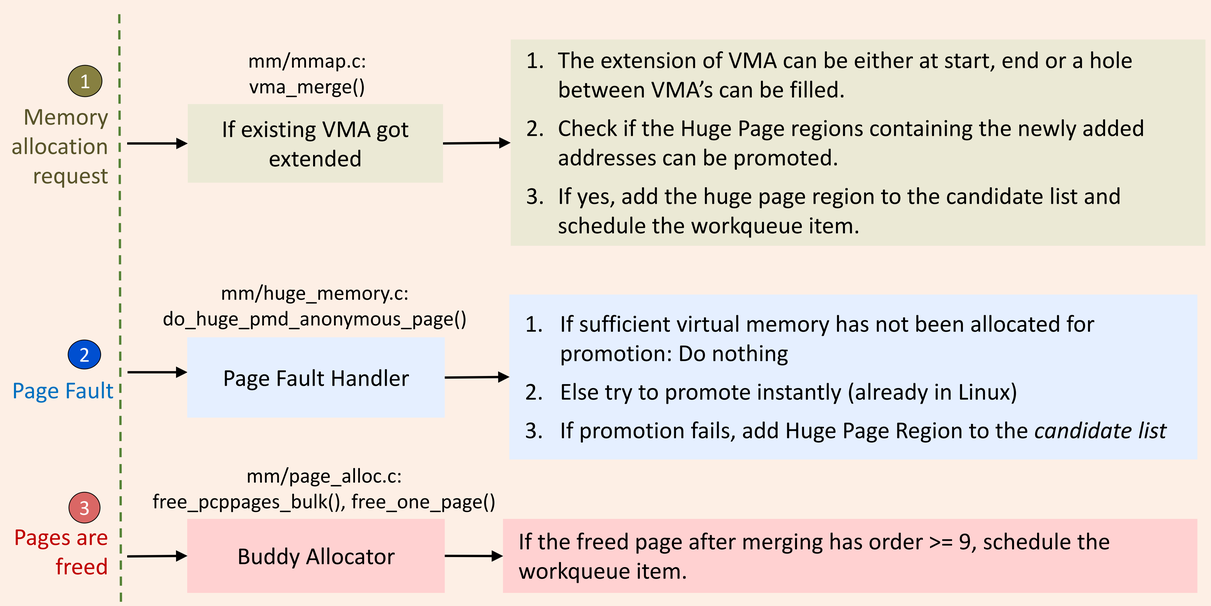


Illustration of Events and resulting actions



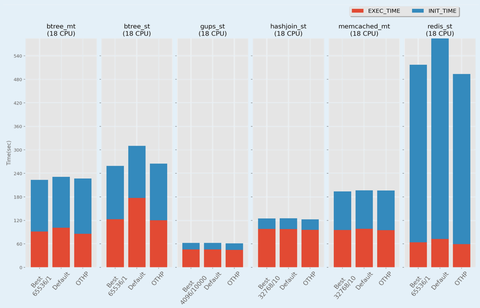
EVALUATION

Methodology

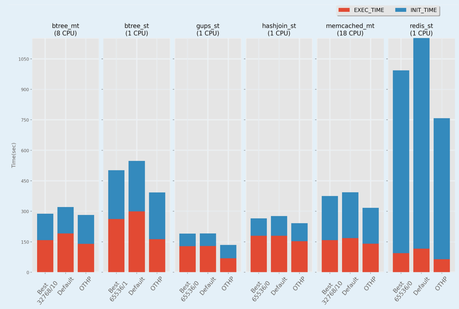
- Creation of huge page mappings largely depends on available number of huge pages in the system and CPU availability to move data.
- **CPU availability**
  - We used CPU hotplug to turn off cores to vary CPU availability
  - The goal is to make *khugepaged* contend for CPU
- **Memory Fragmentation**
  - There are two well defined metrics to measure the degree of fragmentation- usability index and fragmentation index.
  - We use usability index to describe fragmentation as it is equally relevant, more deterministic, and easy to control.

$$Usability\ Index = \frac{Total\ Usuable\ Memory}{Total\ Free\ Memory}$$

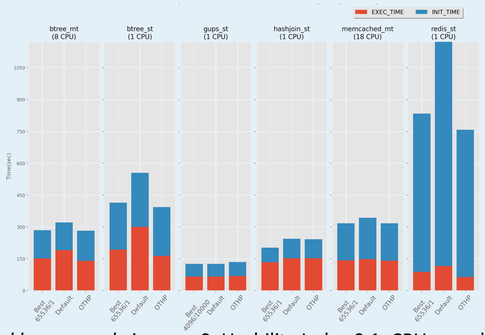
Evaluation on benchmarks



Vanilla Run



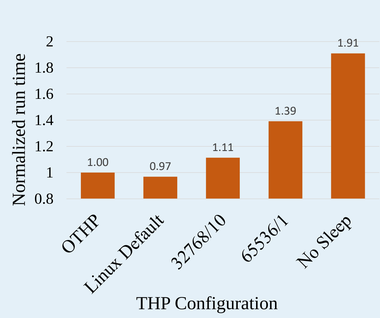
Usability Index 0.1 and CPU crunch

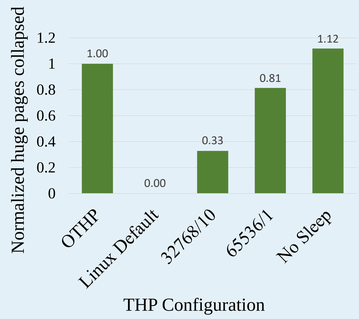


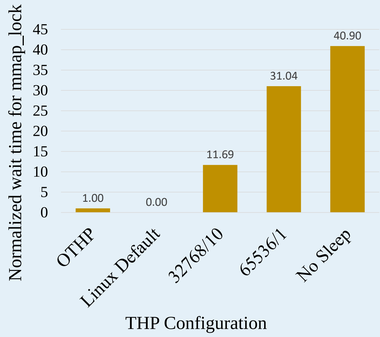
khugepaged niceness 0, Usability Index 0.1, CPU crunch

- OTHP performs near to optimal THP setting when sufficient CPUs and Huge Pages are available
- OTHP performs better than ideal THP setting when the memory is fragmented and CPU is scarce
- Even after setting nice value of *khugepaged* to 0, OTHP performs at par with *khugepaged*.

Evaluation on microbenchmark







- OTHP matches the optimal THP setting while promoting nearly same number of huge pages.
- The gain in performance is due to less *mmap_lock* contention in OTHP