This document summarizes the common approaches for performance fine tuning with jemalloc (as of 5.1.0). The default configuration of jemalloc tends to work reasonably well in practice, and most applications should not have to tune any options. However, in order to cover a wide range of applications and avoid pathological cases, the default setting is sometimes kept conservative and suboptimal, even for many common workloads. When jemalloc is properly tuned for a specific application / workload, it is common to improve system level metrics by a few percent, or make favorable trade-offs.

Notable runtime options for performance tuning

Runtime options can be set via malloc_conf.

- background_thread

  Enabling jemalloc background threads generally improves the tail latency for application threads, since unused memory purging is shifted to the dedicated background threads. In addition, unintended purging delay caused by application inactivity is avoided with background threads.

  Suggested: `background_thread:true` when jemalloc managed threads can be allowed.

- metadata_thp

  Allowing jemalloc to utilize transparent huge pages for its internal metadata usually reduces TLB misses significantly, especially for programs with large memory footprint and frequent allocation / deallocation activities. Metadata memory usage may increase due to the use of huge pages.

  Suggested for allocation intensive programs: `metadata_thp:auto` or `metadata_thp:always`, which is expected to improve CPU utilization at a small memory cost.

- dirty_decay_ms and muzzy_decay_ms

  Decay time determines how fast jemalloc returns unused pages back to the operating system, and therefore provides a fairly straightforward trade-off between CPU and memory usage. Shorter decay time purges unused pages faster to reduces memory usage (usually at the cost of more CPU cycles spent on purging), and vice versa.

  Suggested: tune the values based on the desired trade-offs.

- narenas

  By default jemalloc uses multiple arenas to reduce internal lock contention. However high arena count may also increase overall memory fragmentation, since arenas manage memory independently. When high degree of parallelism is not expected at the allocator level, lower number of arenas often improves memory usage.

Suggested: if low parallelism is expected, try lower arena count while monitoring CPU and memory usage.

- percpu_arena

  Enable dynamic thread to arena association based on running CPU. This has the potential to improve locality, e.g. when thread to CPU affinity is present.

  Suggested: try `percpu_arena:percpu` or `percpu_arena:phycpu` if thread migration between processors is expected to be infrequent.

Examples:

- High resource consumption application, prioritizing CPU utilization:

  `background_thread:true,metadata_thp:auto` combined with relaxed decay time (increased `dirty_decay_ms` and / or `muzzy_decay_ms`, e.g. `dirty_decay_ms:30000,muzzy_decay_ms:30000`).

- High resource consumption application, prioritizing memory usage:

  `background_thread:true` combined with shorter decay time (decreased `dirty_decay_ms` and / or `muzzy_decay_ms`, e.g. `dirty_decay_ms:5000,muzzy_decay_ms:5000`), and lower arena count (e.g. number of CPUs).

- Low resource consumption application:

  `narenas:1,lg_tcache_max:13` combined with shorter decay time (decreased `dirty_decay_ms` and / or `muzzy_decay_ms`,e.g. `dirty_decay_ms:1000,muzzy_decay_ms:0`).

- Extremely conservative – minimize memory usage at all costs, only suitable when allocation activity is very rare:

  `narenas:1,tcache:false,dirty_decay_ms:0,muzzy_decay_ms:0`

Note that it is recommended to combine the options with `abort_conf:true` which aborts immediately on illegal options.

Beyond runtime options

In addition to the runtime options, there are a number of programmatic ways to improve application performance with jemalloc.

- Explicit arenas

  Manually created arenas can help performance in various ways, e.g. by managing locality and contention for specific usages. For example, applications can explicitly allocate frequently accessed objects from a dedicated arena with mallocx() to improve locality. In addition, explicit arenas often benefit from individually tuned options, e.g. relaxed decay time if frequent reuse is expected.

- Extent hooks

  Extent hooks allow customization for managing underlying memory. One use case for performance purpose is to utilize huge pages – for example, HHVM uses explicit arenas with customized extent hooks to manage 1GB huge pages for frequently accessed data, which reduces TLB misses significantly.

- Explicit thread-to-arena binding

  It is common for some threads in an application to have different memory access / allocation patterns. Threads with heavy workloads often benefit from explicit binding, e.g. binding very active threads to dedicated arenas may reduce contention at the allocator level.