Request for Test: jemalloc tuning

As one of our retainer clients who is also running `jemalloc`, I've been looking into and developing a set of `jemalloc` settings on your behalf. I think I've come up with something that could reduce process RSS usage at some amount of CPU utilization cost.

I would say that these settings are currently widely applied throughout the community, but with very little (basically zero) empirical basis of their effects. This test will hopefully change that.

Summary: What To Try

I'm going to start with the goodies.

This assumes you are running jemalloc 5.2+. If you are running any other jemalloc version, contact me. If you are running the Heroku buildpack, and if you do not have the `JEMALLOC_VERSION` env var set, you are using 5.3. If you are installing via Dockerfile, you should check `dpkg -l | grep jemalloc`, but in general Debian bookworm has v5.3 installed, bullseye has 5.2. If you're on `buster`, contact me. If you're on Ubuntu 20+, you should have 5.2 or better.

After extensive customer experience, we believe jemalloc 5.2+ has no worse RSS performance than the jemalloc 3.x series, so we no longer recommend installing jemalloc 3.

One-Step Strategy If you would like to roll this all out at once, follow this approach.

You need to set `MALLOC_CONF` to the following:

```
export MALLOC_CONF="\
background_thread:true,\
dirty_decay_ms:1000,\
narenas:2,\
lg_tcache_max=12,\
lg_extent_max_active_fit:4"
```

… if you're using a Dockerfile and want to use ENV:

```
ENV MALLOC_CONF="\
background_thread:true,\
dirty_decay_ms:1000,\
narenas:2,\
lg_tcache_max=12,\
lg_extent_max_active_fit:4"
```

Since there are approximately a million ways to set ENV variables and everyone does it differently, I will not be opening a PR to this effect, you'll have to do this on your own.

Multi-Step Rollout If you would like to try several increasingly aggressive "stages", each stage gradually increasing settings *away* from the default, you can use these settings:

Stage 1

```
export MALLOC_CONF="\
background_thread:true,\
dirty_decay_ms:8000,\
narenas:16"
```

Stage 2

```
export MALLOC_CONF="\
background_thread:true,\
dirty_decay_ms:7500,\
narenas:8,\
lg_tcache_max=14"
```

Stage 3

```
export MALLOC_CONF="\
background_thread:true,\
dirty_decay_ms:2500,\
narenas:4,\
lg_tcache_max=13,\
lg_extent_max_active_fit:5"
```

Stage 4 (Final)

```
export MALLOC_CONF="\
background_thread:true,\
dirty_decay_ms:1000,\
narenas:2,\
lg_tcache_max=12,\
lg_extent_max_active_fit:4"
```

You should let each "stage" live in production for at least 24 hours before proceeding to the next stage.

If you experience high CPU usage... If you see a response time degradation or unacceptably high CPU use, try the multi-step rollout, or omit the `lg_tcache_max` and `lg_extent_max_active_fit` options entirely, as those options are the ones we have the least confidence in having any payoff re: RSS.

Monitoring Instructions

Monitor the following metrics for **exactly 3 days**:

1. Process memory (RSS)
    - Expected: 1-15% reduction
    - Watch both average and max values
2. CPU Usage
    - Expected: 0-5% increase
3. Response Times
    - Monitor median, p95 and p99
    - Web requests and background jobs separately

You should be monitoring all of these already, so no additional monitoring will be required.

Rollback Criteria:

- No RSS improvement within 3 days. This is important, don't leave these settings just sitting around if they don't improve anything.
- CPU increase > 5%
- Response time degradation > 5%

In-Depth

In general, what these settings are doing is trying to free memory to the OS more aggressively at the cost of increased allocator CPU usage, and potentially wall clock time.

We believe there is an 80% chance these settings do nothing for most apps. We're trying to figure out what apps do see big changes and which don't.

Per-setting explanation

Each setting targets specific jemalloc behaviors:

1. `background_thread:true` (default: false)
    - Enables background memory purging
    - jemalloc issue tracker reports this may be required for other settings to have any effect
    - Slight CPU cost for background operations
2. `dirty_decay_ms:1000` (default: 10000)
    - How long to wait before returning memory to OS
    - Higher values retain more memory
3. `narenas:2` (default: 4xCPU count)
    - Reduces memory fragmentation
    - Improves cross-thread memory reuse

   - May increase lock contention slightly, increasing wall clock time to allocate

4. `lg_tcache_max:12` (default: 15)
   - Maximum size for thread cache (2^15)
   - Reduces per-thread memory overhead
   - May increase allocation costs slightly, but reduce fragmentation in the thread cache

5. `lg_extent_max_active_fit:4` (default: 6)
   - Lower values may reduce fragmentation

What We Didn't Set

Common settings we explicitly avoided:

1. `tcache:false`
   - Too aggressive
   - Serious performance impact

2. `metadata_thp:auto`
   - The amount of memory in the jemalloc metadata is too low to really matter
   - Can cause issues on some platforms

3. `thp:never`
   - Not required in Ruby 2.6+

References

1. jemalloc manpages
2. This gist by jjb
3. jemalloc issue tracker
4. Gitlab seeing no effect of MALLOC_CONF