Ruby master - Feature #9113

Ship Ruby for Linux with jemalloc out-of-the-box

11/15/2013 12:08 PM - sam.saffron (Sam Saffron)

Status:	Closed
Priority:	Normal
Assignee:	
Target version:	

Description

libc's malloc is a problem, it fragments badly meaning forks share less memory and is slow compared to tcmalloc or jemalloc.

both jemalloc and tcmalloc are heavily battle tested and stable.

2 years ago redis picked up the jemalloc dependency see: http://oldblog.antirez.com/post/everything-about-redis-24.html

To quote antirez:

But an allocator is a serious thing. Since we introduced the specially encoded data types Redis started suffering from fragmentation. We tried different things to fix the problem, but basically the Linux default allocator in glibc sucks really, really hard.

I recently benched Discourse with tcmalloc / jemalloc and default and noticed 2 very important thing:

median request time reduce by up to 10% (under both)
PSS (proportional share size) is reduced by 10% under jemalloc and 8% under tcmalloc.

We can always use LD_PRELOAD to yank these in, but my concern is that standard distributions are using a far from optimal memory allocator. It would be awesome if the build, out-of-the-box, just checked if it was on Linux (eg: https://github.com/antirez/redis/blob/unstable/src/Makefile#L30-L34) and then used jemalloc instead.

Associated revisions

Revision 6ab08d2e - 06/05/2014 05:16 AM - nobu (Nobuyoshi Nakada)

configure.in, missing.h: jemalloc mangling

- configure.in (with-jemalloc): also check for header, for ABIs which JEMALLOC_MANGLE is needed, i.e., Mach-O and PE-COFF platforms. [ruby-core:62939] [Feature #9113]
- include/ruby/missing.h: include alternative malloc header to replace memory management functions.
- dln.c, io.c, parse.y, st.c: undef malloc family before re-definition to suppress warnings.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@46354 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 46354 - 06/05/2014 05:16 AM - nobu (Nobuyoshi Nakada)

configure.in, missing.h: jemalloc mangling

- configure.in (with-jemalloc): also check for header, for ABIs which JEMALLOC_MANGLE is needed, i.e., Mach-O and PE-COFF platforms. [ruby-core:62939] [Feature #9113]
- include/ruby/missing.h: include alternative malloc header to replace memory management functions.
- dln.c, io.c, parse.y, st.c: undef malloc family before re-definition to suppress warnings.

Revision 46354 - 06/05/2014 05:16 AM - nobu (Nobuyoshi Nakada)

configure.in, missing.h: jemalloc mangling

- configure.in (with-jemalloc): also check for header, for ABIs which JEMALLOC_MANGLE is needed, i.e., Mach-O and PE-COFF platforms. [ruby-core:62939] [Feature #9113]
- include/ruby/missing.h: include alternative malloc header to replace memory management functions.
- dln.c, io.c, parse.y, st.c: undef malloc family before re-definition to suppress warnings.

Revision 46354 - 06/05/2014 05:16 AM - nobu (Nobuyoshi Nakada)

configure.in, missing.h: jemalloc mangling

- configure.in (with-jemalloc): also check for header, for ABIs which JEMALLOC_MANGLE is needed, i.e., Mach-O and PE-COFF platforms.

[ruby-core:62939] [Feature #9113]
- include/ruby/missing.h: include alternative malloc header to replace memory management functions.
- dln.c, io.c, parse.y, st.c: undef malloc family before re-definition to suppress warnings.

Revision 46354 - 06/05/2014 05:16 AM - nobu (Nobuyoshi Nakada)

configure.in, missing.h: jemalloc mangling

- configure.in (with-jemalloc): also check for header, for ABIs which JEMALLOC_MANGLE is needed, i.e., Mach-O and PE-COFF platforms. [ruby-core:62939] [Feature #9113]
- include/ruby/missing.h: include alternative malloc header to replace memory management functions.
- dln.c, io.c, parse.y, st.c: undef malloc family before re-definition to suppress warnings.

Revision 46354 - 06/05/2014 05:16 AM - nobu (Nobuyoshi Nakada)

configure.in, missing.h: jemalloc mangling

- configure.in (with-jemalloc): also check for header, for ABIs which JEMALLOC_MANGLE is needed, i.e., Mach-O and PE-COFF platforms. [ruby-core:62939] [Feature #9113]
- include/ruby/missing.h: include alternative malloc header to replace memory management functions.
- dln.c, io.c, parse.y, st.c: undef malloc family before re-definition to suppress warnings.

Revision 46354 - 06/05/2014 05:16 AM - nobu (Nobuyoshi Nakada)

configure.in, missing.h: jemalloc mangling

- configure.in (with-jemalloc): also check for header, for ABIs which JEMALLOC_MANGLE is needed, i.e., Mach-O and PE-COFF platforms. [ruby-core:62939] [Feature #9113]
- include/ruby/missing.h: include alternative malloc header to replace memory management functions.
- dln.c, io.c, parse.y, st.c: undef malloc family before re-definition to suppress warnings.

Revision a4d7e428 - 06/05/2014 05:36 AM - nobu (Nobuyoshi Nakada)

version.c: show malloc_conf

- configure.in (jemalloc): check for the header regardless drop-in libjemalloc is found, for malloc_conf declaration.
- version.c (ruby_show_version): show malloc_conf if set. [Feature #9113]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@46355 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 46355 - 06/05/2014 05:36 AM - nobu (Nobuyoshi Nakada)

version.c: show malloc_conf

- configure.in (jemalloc): check for the header regardless drop-in libjemalloc is found, for malloc_conf declaration.
- version.c (ruby_show_version): show malloc_conf if set. [Feature #9113]

Revision 46355 - 06/05/2014 05:36 AM - nobu (Nobuyoshi Nakada)

version.c: show malloc_conf

- configure.in (jemalloc): check for the header regardless drop-in libjemalloc is found, for malloc_conf declaration.
- version.c (ruby_show_version): show malloc_conf if set. [Feature #9113]

Revision 46355 - 06/05/2014 05:36 AM - nobu (Nobuyoshi Nakada)

version.c: show malloc_conf

- configure.in (jemalloc): check for the header regardless drop-in libjemalloc is found, for malloc_conf declaration.
- version.c (ruby_show_version): show malloc_conf if set. [Feature #9113]

Revision 46355 - 06/05/2014 05:36 AM - nobu (Nobuyoshi Nakada)

version.c: show malloc_conf

- configure.in (jemalloc): check for the header regardless drop-in libjemalloc is found, for malloc_conf declaration.
- version.c (ruby_show_version): show malloc_conf if set. [Feature #9113]

Revision 46355 - 06/05/2014 05:36 AM - nobu (Nobuyoshi Nakada)

version.c: show malloc_conf

- configure.in (jemalloc): check for the header regardless drop-in libjemalloc is found, for malloc_conf declaration.
- version.c (ruby_show_version): show malloc_conf if set. [Feature #9113]

Revision 46355 - 06/05/2014 05:36 AM - nobu (Nobuyoshi Nakada)

version.c: show malloc_conf

- configure.in (jemalloc): check for the header regardless drop-in libjemalloc is found, for malloc_conf declaration.
- version.c (ruby_show_version): show malloc_conf if set. [Feature #9113]

History

#1 - 11/15/2013 02:45 PM - naruse (Yui NARUSE)

- Status changed from Open to Assigned

- Assignee set to kosaki (Motohiro KOSAKI)

Could you comment this?

#2 - 11/15/2013 02:49 PM - nobu (Nobuyoshi Nakada)

- Status changed from Assigned to Third Party's Issue

If system malloc is replaced with those newer libraries, ruby will use it.
Otherwise, configure with LIBS=-ljemalloc.

#3 - 11/15/2013 02:49 PM - nobu (Nobuyoshi Nakada)

- Category set to build

- Assignee deleted (kosaki (Motohiro KOSAKI))

#4 - 11/15/2013 03:23 PM - duerst (Martin Dürst)

On one level, this feels like a non-brainer. But then the question is why the standard memory allocator in libc hasn't been improved.

I can imagine all kinds of reasons, from "alternatives use too much memory" to "not invented here". Any background info?

Regards, Martin.

On 2013/11/15 12:08, sam.saffron (Sam Saffron) wrote:

> https://bugs.ruby-lang.org/issues/9113
>
> ---
>
> I recently benched Discourse with tcmalloc / jemalloc and default and noticed 2 very important thing:
>
> median request time reduce by up to 10% (under both)
> PSS (proportional share size) is reduced by 10% under jemalloc and 8% under tcmalloc.
>
> We can always use LD_PRELOAD to yank these in, but my concern is that standard distributions are using a far from optimal memory allocator. It would be awesome if the build, out-of-the-box, just checked if it was on Linux (eg: https://github.com/antirez/redis/blob/unstable/src/Makefile#L30-L34) and then used jemalloc instead.

#5 - 11/15/2013 04:16 PM - sam.saffron (Sam Saffron)

@martin this is a great "oldish" article by facebook about this
http://www.facebook.com/notes/facebook-engineering/scalable-memory-allocation-using-jemalloc/480222803919

nobu (Nobuyoshi Nakada) I guess my suggestion here is to include jemalloc source in the the repo, and compile on demand for linux (by default with an option to opt-out) that way everyone will pick this change up and it becomes "officially blessed" allocator.

At Github tmm1 (Aman Gupta) has been using tcmalloc for years now, fragmentation is less good than jemalloc (tmm1 said perf is better, but I think its time to re-test cause I found jemalloc to perform better)

Regardless, libc allocator is a problem and default compiles should not use it.

Firefox has been using jemalloc for years and years, it is safe for production http://glandium.org/blog/?p=2581 ..

#6 - 11/15/2013 09:53 PM - nobu (Nobuyoshi Nakada)

- Status changed from Third Party's Issue to Rejected

Then it is a task of package maintainers.

#7 - 11/22/2013 11:40 AM - kosaki (Motohiro KOSAKI)

duerst (Martin Dürst) It is not correct. I and glibc folks are working on several improvement about malloc. Moreover, each allocator has different pros/cons. jemalloc can retrieve some workload better and glibc allocator can retrieve some other workload. There is no single perfect allocator. That's our difficulty.

sam.saffron (Sam Saffron) The Facebook's page you pointed out is out of date. It compare glibc 2.5 vs jemalloc 2.1.0. But latest are glibc 2.18 and jemalloc 3.4.1. And, glibc malloc and jemalloc shares a lot of basic design. So, this documentation is completely useless. If you have several workload which glibc doesn't work well, please make and share benchmark instead of rumor. Then, we can improve several bottlenecks.

#8 - 11/22/2013 11:41 AM - kosaki (Motohiro KOSAKI)

It does NOT mean jemalloc has no chance. But we don't discuss performance issue if nobody has a number.

#9 - 11/22/2013 02:44 PM - naruse (Yui NARUSE)

- Status changed from Rejected to Feedback

#10 - 01/22/2014 10:19 PM - normalperson (Eric Wong)

Btw, jemalloc 3.5 includes an updated non-standard experimental API.

It looks like it has the ability to specify different arenas for
allocation (via MALLOCX_ARENA(a)). Perhaps could be used to
distinguish long/short-lived allocations.
Probably worth experimenting on some day...

#11 - 01/26/2014 02:11 AM - normalperson (Eric Wong)

I tried jemalloc 3.5.0 vs eglibc 2.13-38 (Debian x86_64)

http://80x24.org/bmlog-20140126-003136.7320.gz

Mostly close results, but I think our "make benchmark" suite is
incomplete and we need more fork/concurrency-intensive benchmarks of
large apps.

io_file_read and vm2_bigarray seem to be big losses because jemalloc
tends to release large allocations back to the kernel more aggressively
(and the kernel must zero that memory).

[1] I have applied two patches for improved benchmark consistency:
https://bugs.ruby-lang.org/issues/5985#change-44442
https://bugs.ruby-lang.org/issues/9430
(Note: I still don't trust the vm_thread* benchmarks too much,
they seem very inconsistent even with no modifications)

#12 - 02/18/2014 11:39 PM - sam.saffron (Sam Saffron)

I can confirm 2 findings.

When heaps are small you barely notice a different.
When heaps grow and general memory fragmentation grows, jemalloc is far better.

I see a 6% reduction of RSS running discourse bench on 2.1.0 https://github.com/discourse/discourse/blob/master/script/bench.rb

An artificial test is:

```
@retained = []

MAX_STRING_SIZE = 100

def stress(allocate_count, retain_count, chunk_size)
  chunk = []
  while retain_count > 0 || allocate_count > 0
    if retain_count == 0 || (Random.rand < 0.5 && allocate_count > 0)
      chunk << " " * (Random.rand * MAX_STRING_SIZE).to_i
      allocate_count -= 1
      if chunk.length > chunk_size
        chunk = []
      end
    else
```

```
      @retained << " " * (Random.rand * MAX_STRING_SIZE).to_i
      retain_count -= 1
    end
  end
end

start = Time.now
stress(1_000_000, 600_000, 200_000)
puts "Duration: #{(Time.now - start).to_f}"

puts `ps aux | grep #{Process.pid} | grep -v grep`
```

For glibc

```
sam@ubuntu ~ % time ruby stress_mem.rb
Duration: 0.705922489
sam      17397 73.0  2.5 185888 156884 pts/10  Sl+  10:37   0:00 ruby stress_mem.rb
ruby stress_mem.rb  0.78s user 0.08s system 100% cpu 0.855 total
```

For jemalloc 3.5.0

```
Duration: 0.676871705
sam      17428 70.0  2.3 186248 144800 pts/10  Sl+  10:37   0:00 ruby stress_mem.rb
LD_PRELOAD=/home/sam/Source/jemalloc-3.5.0/lib/libjemalloc.so ruby   0.68s user 0.09s system 100% cpu 0.771 to
tal
```

--

You can see the 8% or so better RSS with jemalloc

Note the more iterations you add the better jemalloc does. up allocations to 10 million

jemalloc 200mb rss vs glibc 230mb rss

glibc gets fragmented at a far faster rate than jemalloc

#13 - 02/18/2014 11:53 PM - sam.saffron (Sam Saffron)

Note, this pattern of

1. Retaining large number of objects
2. Allocating a big chunk of objects (and releasing)
3. Repeating (2)

Is very representative of web apps / rails apps. For our application requests will range between 20k allocations and 200k allocations.

It is very much a scenario we want to optimise for.

on another note Rust lang just picked jemalloc, golang uses a fork of tcmalloc http://golang.org/src/pkg/runtime/malloc.h?h=tcmalloc

#14 - 02/19/2014 12:00 AM - normalperson (Eric Wong)

sam.saffron@gmail.com wrote:

> An artificial test is:
>
> @retained = []
>
> MAX_STRING_SIZE = 100
>
> def stress(allocate_count, retain_count, chunk_size)

Note: I think we should seed the RNG to a constant to have
consistent data between runs

```
srand(123)
```

> chunk = []
> while retain_count > 0 || allocate_count > 0
> if retain_count == 0 || (Random.rand < 0.5 && allocate_count > 0)
> chunk << " " * (Random.rand * MAX_STRING_SIZE).to_i
> allocate_count -= 1
> if chunk.length > chunk_size

```
chunk = []
end
else
@retained << " " * (Random.rand * MAX_STRING_SIZE).to_i
retain_count -= 1
end
end
end
```

Sam: Thank you!

I think we should integrate this test into the mainline benchmark suite.
Perhaps even provide an option to run with all the existing tests with
the big @retained array.

ko1: what do you think?

#15 - 02/19/2014 12:05 AM - sam.saffron (Sam Saffron)

eric (Eric Anderson)

sure bench needs a bit more love to be totally representative of a rails request. Also this test will do ko1 lots of help improving the promotion to oldgen algorithm, we are talking about changing oldgen promotion to either use additional flags (as a counter) or only promote on major GC.

Either change will slash RSS in this test.

```
sam@ubuntu ~ % rbenv shell 2.1.0
sam@ubuntu ~ % ruby stress_mem.rb
Duration: 5.459891703
sam      17870  109  3.8 267076 238732 pts/10  Sl+  11:03   0:05 ruby stress_mem.rb
sam@ubuntu ~ % rbenv shell 2.0.0-p353
sam@ubuntu ~ % ruby stress_mem.rb
Duration: 7.616282557
sam      17986 95.6  2.0 151120 125684 pts/10  Sl+  11:04   0:07 ruby stress_mem.rb
sam@ubuntu ~ %
```

This is basically a repro of the memory growth under 2.1.0 people are seeing.

238mb in Ruby 2.1 vs 125mb in 2.0

#16 - 02/21/2014 12:57 AM - nobu (Nobuyoshi Nakada)

I'm absolutely against including external libraries into ruby repository itself, e.g., libyaml.
It may not be the worst idea to bundle them with the tarballs.

#17 - 02/22/2014 09:07 PM - sam.saffron (Sam Saffron)

@nobusan I think that would be a reasonable approach

eric (Eric Anderson) / ko1 (Koichi Sasada) / everyone

here are the results of running that script across every 5 builds in the last year (with seeded rand)

https://gist.github.com/SamSaffron/9162366

I think big jumps should be investigated

#18 - 02/25/2014 05:38 PM - ko1 (Koichi Sasada)

(2014/02/19 9:08), Eric Wong wrote:

> Btw, I also hope to experiment with a slab allocator since many internal
> objects are around the same size (like an OS kernel). This idea is
> originally from the Solaris kernel, but also in Linux and FreeBSD. One
> benefit with slab allocators over a general purpose malloc is malloc
> has too little context/information make some decisions:
>
> - long-lived vs short-lived (good for CoW)
> - shared between threads or not
> - future allocations of the same class
>
> Notes on slab: I don't think caching constructed objects like the
> reference Solaris implementation does is necessary (or even good),
> since it should be possible to transparently merge objects of different

> classes (like SLUB in Linux, I think).
>
> Anyways, I think jemalloc is a great general-purpose malloc for things that don't fit well into slabs. And it should be easy to let a slab implementation switch back to general-purpose malloc for testing/benching.

Recently I'm working around this topic.

(1) Life-time oriented, similar to Copying GC
(2) CoW frindly (read only) memories

More detail about (2):
The following figure shows the stacked memory usage (snapshot) collected by valgrind/massif, on discorse benchmark by @sam's help.
http://www.atdot.net/fp_store/f.69bk1n/file.copipa-temp-image.png

Interestingly, 50MB is consumed by iseq (iseq.c, compile.c). Most of data are read only, so it can be more CoW frindly. Now, we mixes read-only data and r/w data such as inline cahce.

There are several ideas. And I belive it is good topic to consider for Ruby 2.2.

--
// SASADA Koichi at atdot dot net

#19 - 02/25/2014 06:48 PM - normalperson (Eric Wong)

ko1@atdot.net wrote:

> (2014/02/19 9:08), Eric Wong wrote:
>
> > Btw, I also hope to experiment with a slab allocator since many internal objects are around the same size (like an OS kernel). This idea is originally from the Solaris kernel, but also in Linux and FreeBSD. One benefit with slab allocators over a general purpose malloc is malloc has too little context/information make some decisions:
> >
> > - long-lived vs short-lived (good for CoW)
> > - shared between threads or not
> > - future allocations of the same class
> >
> > Notes on slab: I don't think caching constructed objects like the reference Solaris implementation does is necessary (or even good), since it should be possible to transparently merge objects of different classes (like SLUB in Linux, I think).
> >
> > Anyways, I think jemalloc is a great general-purpose malloc for things that don't fit well into slabs. And it should be easy to let a slab implementation switch back to general-purpose malloc for testing/benching.
>
> Recently I'm working around this topic.
>
> (1) Life-time oriented, similar to Copying GC
> (2) CoW frindly (read only) memories

Yes. We should be able to do moving/defragmentation of long-lived internal allocations, even.

> Interestingly, 50MB is consumed by iseq (iseq.c, compile.c). Most of data are read only, so it can be more CoW frindly. Now, we mixes read-only data and r/w data such as inline cahce.

Yes, also the iseq struct is huge (300+ bytes on 64-bit). I think we can shrink it (like I did with struct vtm/time_object) and move r/w data off to a different area.

> There are several ideas. And I belive it is good topic to consider for Ruby 2.2.

OK; especially since this should have no public API breakage.

#20 - 02/26/2014 09:12 AM - normalperson (Eric Wong)

Sam: btw, if you have time, can you prepare a patch which integrates
jemalloc with the build/tarball dist?

We should also see if using the non-standard jemalloc API is worth it.
(with fallbacks to standard APIs on systems where jemalloc is unsupported).

#21 - 05/13/2014 02:58 AM - normalperson (Eric Wong)

- File 0001-configure.in-add-with-jemalloc-option.patch added

Attached patch to use jemalloc by default on GNU/Linux iff
jemalloc is already installed. Maybe we can integrate/force
it later, but this is a first step (and some distros like
Debian already ship jemalloc).

#22 - 05/13/2014 03:22 AM - nobu (Nobuyoshi Nakada)

"linux-gnu" in "target_os" is substituted by "linux", so your AS_CASE never match.
You should check for libjemalloc availability instead, I think.

```
diff --git a/configure.in b/configure.in
index cf317af..a9037d6 100644
--- a/configure.in
+++ b/configure.in
@@ -1170,6 +1170,17 @@ AS_IF([test "x$with_gmp" != xno],
    with_gmp="$ac_cv_lib_gmp___gmpz_init"
    AS_IF([test -z "$with_gmp"], [with_gmp=no])])

+dnl use jemalloc on GNU/Linux, assume non-GNU has a better malloc than glibc
+AC_ARG_WITH([jemalloc],
+  [AS_HELP_STRING([--with-jemalloc],
+    [use jemalloc allocator. Default is yes on GNU/Linux, no elsewhere])],
+  )
+AS_IF([test "x$with_jemalloc" != xno],[
+  AC_CHECK_LIB([jemalloc],[malloc_conf],
+    [LIBS="-ljemalloc $LIBS"],
+    [AS_IF([test "x$with_jemalloc" = xyes],
+      [AC_MSG_WARN([No jemalloc found, using system malloc])])])
+
 dnl check for large file stuff
 mv confdefs.h confdefs1.h
 : > confdefs.h
```

#23 - 05/13/2014 04:09 AM - normalperson (Eric Wong)

nobu@ruby-lang.org wrote:

> "linux-gnu" in "target_os" is substituted by "linux", so your AS_CASE never match.
> You should check for libjemalloc availability instead, I think.

I used $target, not $target_os. An early version of my patch used
$target_os, but I noticed it was a noop so I changed it to use $target.

I've tested this with bare "./configure" and also --without-jemalloc.
Both work as expected on my GNU/Linux systems.

I worry this leads to platform-specific problems on non-GNU/Linux
systems. It seems varnish/redis only tries jemalloc on Linux, too.

Also, I get one failure on both 32-bit and 64-bit:

[1/1] Test_StringModifyExpand#test_modify_expand_memory_leak = 0.02 s
1) Failure:
Test_StringModifyExpand#test_modify_expand_memory_leak

size: 14307328 => 31084544..
Expected 2.1726309762381906 to be < 1.5.

It looks like certain allocation patterns are worse with jemalloc, but

I think real-world apps use less memory.

#24 - 05/19/2014 08:09 PM - normalperson (Eric Wong)

normalperson@yhbt.net wrote:

> [1/1] Test_StringModifyExpand#test_modify_expand_memory_leak = 0.02 s
> 1) Failure:
> Test_StringModifyExpand#test_modify_expand_memory_leak
> [/home/ew/ruby/test/-ext-/string/test_modify_expand.rb:7]:
> rb_str_modify_expand().
> size: 14307328 => 31084544..
> Expected 2.1726309762381906 to be < 1.5.
>
> It looks like certain allocation patterns are worse with jemalloc, but
> I think real-world apps use less memory.

I can update this to limit=2.2 to workaround the failure and commit my
original patch soon. Any comment/objections?

Platform maintainers for non-GNU/Linux may enable jemalloc detection if
they feel comfortable, but I don't want to potentially break things
behind their backs. For example, I cannot test/fix OSX problems, but I
have read using non-standard malloc is a difficult problem on that
platform (but possible).

#25 - 05/19/2014 09:18 PM - ko1 (Koichi Sasada)

(2014/05/20 5:09), normalperson@yhbt.net wrote:

> I can update this to limit=2.2 to workaround the failure and commit my
> original patch soon. Any comment/objections?

which one?

--
// SASADA Koichi at atdot dot net

#26 - 05/19/2014 09:25 PM - ko1 (Koichi Sasada)

Sorry I found https://bugs.ruby-lang.org/attachments/download/4424/0001-configure.in-add-with-jemalloc-option.patch

I want to suggest only add "--enable-jemalloc" (not default).

I'm afraid that it makes performance prediction more difficult.
(someone use it and someone doesn't use)

#27 - 05/19/2014 10:09 PM - normalperson (Eric Wong)

ko1@atdot.net wrote:

> I want to suggest only add "--enable-jemalloc" (not default).

I am considering it, but I am not sure non-default is worth it over
nobu's original suggestion of using "./configure LIBS=-ljemalloc"

Sam: what do you think?

> I'm afraid that it makes performance prediction more difficult.
> (someone use it and someone doesn't use)

I understand. Maybe a big warning at the end of ./configure?

Fwiw, I am mildly against bundling jemalloc in the tarball. It is
more to maintain and keep up-to-date. Varnish no longer bundles
jemalloc for this reason, either.

I chose to detect + enable jemalloc by default since it can encourage
distributors and packagers to use and depend on it (like gmp).

#28 - 05/23/2014 09:59 PM - normalperson (Eric Wong)

Warning moved to the end of output where it may be more visible:
http://yhbt.net/ruby-9113-jemalloc-v2.patch
Comments?

#29 - 05/24/2014 04:07 AM - nobu (Nobuyoshi Nakada)

I'm still negative to enable it by default.

#30 - 05/24/2014 06:21 PM - kosaki (Motohiro KOSAKI)

On Sat, May 24, 2014 at 1:07 PM, nobu@ruby-lang.org wrote:

> Issue #9113 has been updated by Nobuyoshi Nakada.
>
> I'm still negative to enable it by default.

Do we have only one benchmark provided by Sam? I don't think it is enough much comparison to make decision.

Anyway, I don't think Linux distros enable jemalloc because of their packaging policy even if we enable by default.
So we need, at least

1. A way to disable it
2. Easy to detect which malloc is used from bug reports. For our maintenance.

#31 - 05/25/2014 01:48 AM - normalperson (Eric Wong)

KOSAKI Motohiro kosaki.motohiro@gmail.com wrote:

> Do we have only one benchmark provided by Sam? I don't think it is enough much comparison to make decision.

Empirical evidence on my 32-bit yahns server after running several days shows ~40M RSS w/ jemalloc 3.0/3.6 vs 60-80M RSS for eglibc malloc on Debian stable. I'll work on gathering more data for other systems.

> Anyway, I don't think Linux distros enable jemalloc because of their packaging policy even if we enable by default.

I know distros do not like bundling extra libs, but I'm not aware of policies against extra linkage for already packaged libraries.

Looking at Debian stable packages, both redis and varnish packages[1] use jemalloc on common architectures. For redis, Debian just favors the system jemalloc instead of the bundled one.

> So we need, at least
>
> 1. A way to disable it

My patch respects the --without-jemalloc option

> 1. Easy to detect which malloc is used from bug reports. For our maintenance.

rb_bug already shows dynamically loaded libraries. My patch only enables dynamic link by default.

[1] http://http.us.debian.org/debian/pool/main/r/redis/redis_2.4.14-1.dsc
http://http.us.debian.org/debian/pool/main/r/redis/redis_2.4.14.orig.tar.gz
http://http.us.debian.org/debian/pool/main/r/redis/redis_2.4.14-1.debian.tar.gz
http://http.us.debian.org/debian/pool/main/v/varnish/varnish_3.0.2-2+deb7u1.dsc
http://http.us.debian.org/debian/pool/main/v/varnish/varnish_3.0.2.orig.tar.gz
http://http.us.debian.org/debian/pool/main/v/varnish/varnish_3.0.2-2+deb7u1.debian.tar.gz

#32 - 06/01/2014 07:27 PM - hansdegraaff (Hans de Graaff)

Eric Wong wrote:

> ko1@atdot.net wrote:

> > I want to suggest only add "--enable-jemalloc" (not default).
>
> I am considering it, but I am not sure non-default is worth it over
> nobu's original suggestion of using "./configure LIBS=-ljemalloc"

Speaking from the Gentoo perspective: with "--enable-jemalloc" I could give my Gentoo users a choice of enabling jemalloc via our "USE flag" mechanism. I'd rather not do that with the LIBS mechanism since then I cannot be sure that this is a maintained and supported configuration.

> Fwiw, I am mildly against bundling jemalloc in the tarball. It is
> more to maintain and keep up-to-date. Varnish no longer bundles
> jemalloc for this reason, either.

In Gentoo we would always remove or ignore the bundled version and build against our system version.

#33 - 06/02/2014 02:56 AM - nobu (Nobuyoshi Nakada)

Hans de Graaff wrote:

> I'd rather not do that with the LIBS mechanism since then I cannot be sure that this is a maintained and supported configuration.

It's a common method for autoconfisticated programs.
Did you have troubles with it?

#34 - 06/02/2014 05:11 PM - hansdegraaff (Hans de Graaff)

Nobuyoshi Nakada wrote:

> It's a common method for autoconfisticated programs.
> Did you have troubles with it?

I have not tried. My point was that, being one of the Gentoo ruby maintainers, I would not want to use this mechanism to officially support jemalloc or tmalloc for the ruby packages that we offer in Gentoo.

#35 - 06/03/2014 12:42 AM - kosaki (Motohiro KOSAKI)

On Sat, May 24, 2014 at 9:48 PM, normalperson@yhbt.net wrote:

> Issue #9113 has been updated by Eric Wong.
>
> KOSAKI Motohiro kosaki.motohiro@gmail.com wrote:
>
> > Do we have only one benchmark provided by Sam? I don't think it is
> > enough much comparison to make decision.
>
> Empirical evidence on my 32-bit yahns server after running several days
> shows ~40M RSS w/ jemalloc 3.0/3.6 vs 60-80M RSS for eglibc malloc on
> Debian stable. I'll work on gathering more data for other systems.

So, I'd suggest two phase action.

1. Commit your patch, but disable by default.
2. Gather more use-case and performance data. Note: They should be reproducible. We need measure them again and again when libs updated.
3. Change the default when the community is convinced the benefit.

#36 - 06/03/2014 01:10 AM - nobu (Nobuyoshi Nakada)

Hans de Graaff wrote:

> My point was that, being one of the Gentoo ruby maintainers, I would not want to use this mechanism to officially support jemalloc or tmalloc for the ruby packages that we offer in Gentoo.

```
$ ./configure --help
(snip)
Some influential environment variables:
  CC          C compiler command
  CFLAGS      C compiler flags
  LDFLAGS     linker flags, e.g. -L<lib dir> if you have libraries in a
```

```
              nonstandard directory <lib dir>
  LIBS        libraries to pass to the linker, e.g. -l<library>
  CPPFLAGS    (Objective) C/C++ preprocessor flags, e.g. -I<include dir> if
              you have headers in a nonstandard directory <include dir>
  CXX         C++ compiler command
  CXXFLAGS    C++ compiler flags
  CPP         C preprocessor

Use these variables to override the choices made by `configure' or to help
it to find libraries and programs with nonstandard names/locations.
```

#37 - 06/04/2014 07:51 PM - normalperson (Eric Wong)

KOSAKI Motohiro kosaki.motohiro@gmail.com wrote:

> So, I'd suggest two phase action.
>
> 1. Commit your patch, but disable by default.

OK, r46349

> 1. Gather more use-case and performance data. Note: They should be reproducible. We need measure them again and again when libs updated.
> 2. Change the default when the community is convinced the benefit.

I wouldn't mind changing the default. However small applications
take up more memory with jemalloc. Using MALLOC_CONF=lg_chunk:21
or smaller numbers can reduce it (at performance cost).

Users may also try the Lockless Inc allocator. However we
cannot ship it with Ruby since Ruby is not GPLv3.
http://locklessinc.com/downloads/

#38 - 06/05/2014 01:06 AM - hsbt (Hiroshi SHIBATA)

I tried to build with --with-jemalloc option.

- OS X 10.9.4 pre-release
- jemalloc-3.6.0 via homebrew
- r46350

but it got build failure.

```
checking for malloc_conf in -ljemalloc... no
configure: error: jemalloc requested but not found
```

libjemalloc.dylib is following location:

```
% ls -l /usr/local/lib/libjemalloc.dylib
lrwxr-xr-x 1 hsbt admin 46  6  5 09:18 /usr/local/lib/libjemalloc.dylib -> ../Cellar/jemalloc/3.6.0/lib/libjem
alloc.dylib
```

config.log: https://gist.github.com/hsbt/87f2748dd4117d3cd139

#39 - 06/05/2014 01:58 AM - nobu (Nobuyoshi Nakada)

jemalloc prefixes "je_" to each public symbols at Mach-O (Mac OS X) and PE-COFF (Windows), so it can't be drop-in by default.

https://github.com/jemalloc/jemalloc/blob/dev/configure.ac#L494L498

Or you have to install it with --with-jemalloc-prefix option unless compiling with "jemalloc/jemalloc.h".

#40 - 06/05/2014 05:16 AM - nobu (Nobuyoshi Nakada)

- Status changed from Feedback to Closed

- % Done changed from 0 to 100

Applied in changeset r46354.

---

configure.in, missing.h: jemalloc mangling

- configure.in (with-jemalloc): also check for header, for ABIs which JEMALLOC_MANGLE is needed, i.e., Mach-O and PE-COFF platforms. [ruby-core:62939] [Feature #9113]
- include/ruby/missing.h: include alternative malloc header to replace memory management functions.
- dln.c, io.c, parse.y, st.c: undef malloc family before re-definition to suppress warnings.

#41 - 08/20/2014 07:48 PM - normalperson (Eric Wong)

SASADA Koichi ko1@atdot.net wrote:

> Recently I'm working around this topic.
>
> (1) Life-time oriented, similar to Copying GC
> (2) CoW frindly (read only) memories
>
> More detail about (2):
> The following figure shows the stacked memory usage (snapshot) collected
> by valgrind/massif, on discorse benchmark by @sam's help.
> http://www.atdot.net/fp_store/f.69bk1n/file.copipa-temp-image.png
>
> Interestingly, 50MB is consumed by iseq (iseq.c, compile.c). Most of
> data are read only, so it can be more CoW frindly. Now, we mixes
> read-only data and r/w data such as inline cahce.
>
> There are several ideas. And I belive it is good topic to consider for
> Ruby 2.2.

ko1: any progress on this front? I may use dlmalloc mspace API[1] to
make a special CoW-friendly heap for read-only parts of the iseq
structure (and probably other read-only data such as frozen pathnames).

And continue using regular malloc (e.g. jemalloc) for r/w heap.

[1] ftp://gee.cs.oswego.edu/pub/misc/malloc.c

Files

0001-configure.in-add-with-jemalloc-option.patch	1.29 KB	05/13/2014	normalperson (Eric Wong)