Ruby trunk - Bug #14490

MJIT slows down Rails applications

02/19/2018 01:36 AM - sam.saffron (Sam Saffron)

Status:	Closed		
Priority:	Normal		
Assignee:	k0kubun (Takashi Kokubun)		
Target version:			
ruby -v:		**Backport:**	2.3: UNKNOWN, 2.4: UNKNOWN, 2.5: UNKNOWN

Description

This was reported earlier but I would like to add some test methodology here: (tested using Discourse)

What I tested

1. Run ruby script/bench.rb to warm up profiling database
2. Run RUBYOPT='--jit --jit-verbose=1 --jit-max-cache=10000' RAILS_ENV=profile bin/puma -e production
    1. Increasing the cache size is an attempt to get higher hit rates
3. WAIT 5-15 or so minutes for all jitting to stop so we have no cross talk
4. Run ab -n 100 http://localhost:9292/
5. Wait for all new jitting to finish
6. Run ab -n 100 http://localhost:9292/

The absolute best I get is around **61ms** median for the test. With MJIT

This is MJIT results!

```
Percentage of the requests served within a certain time (ms)
  50%     61
  66%     63
  75%     70
  80%     71
  90%     73
  95%     83
  98%    106
  99%    109
 100%    109 (longest request)
```

Compared to w/o MJIT which is:

```
Percentage of the requests served within a certain time (ms)
  50%     47
  66%     48
  75%     50
  80%     51
  90%     60
  95%     62
  98%     75
  99%     77
 100%     77 (longest request)
```

Theory on why this happens

I suspect this is happening cause we are introducing new overhead to every single method dispatch (counting for mjit, and hash

lookup and so on). This overhead adds up.

Associated revisions

Revision 443f4d58 - 07/28/2018 04:14 PM - k0kubun (Takashi Kokubun)

mjit.c: introduce JIT compaction [experimental]

When all compilation finishes or the number of JIT-ed code reaches
--jit-max-cache, this compacts all generated code to a single .so file
and re-loads all methods from it.

In the future, it may trigger compaction more frequently and/or limit
the maximum times of compaction to prevent unlimited memory usage.
So the current behavior is experimental, but at least the performance
improvement in this commit won't be removed.

=== Benchmark ===
In this benchmark, I'll compare following four conditions:

- trunk: r64082
- trunk JIT: r64082 w/ --jit
- single-so JIT: This commit w/ --jit
- objfcn JIT: This branch https://github.com/k0kubun/ruby/tree/objfcn w/ --jit, which is shinh's objfcn https://github.com/shinh/ruby/tree/objfcn rebased from this commit

```
$ uname -a
Linux bionic 4.15.0-29-generic #31-Ubuntu SMP Tue Jul 17 15:39:52 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux
```

- Micro benchmark Using this script https://gist.github.com/k0kubun/10e6d3387c9ab1b134622b2c9d76ef51, calls some amount of different methods that just return nil. The following tables are its average duration seconds of 3 measurements.

Smaller is better.

** 1 method (seconds)
	trunk	trunk JIT	single-so JIT	objfcn JIT
Time	5.576067774333296	5.915551971666446	5.833641665666619	5.845915191666639
Ratio	1.00x	1.06x	1.05x	1.05x

** 50 methods (seconds)
	trunk	trunk JIT	single-so JIT	objfcn JIT
Time	3.1661167996666677	6.125825928333342	4.135432743666665	3.750358728333348
Ratio	1.00x	1.93x	1.31x	1.18x

** 1500 methods (seconds)
	trunk	trunk JIT	single-so JIT	objfcn JIT
Time	5.971650823666664	19.579182102999994	10.511108153999961	10.854653588999932
Ratio	1.00x	3.28x	1.76x	1.82x

- Discourse Using the same benchmark strategy as https://bugs.ruby-lang.org/issues/14490 with this branch https://github.com/k0kubun/discourse/commits/benchmark2 forked from discourse v1.8.11 to support running trunk.

1. Run ruby script/bench.rb to warm up profiling database
2. Run RUBYOPT='--jit-verbose=1 --jit-max-cache=10000' RAILS_ENV=profile bin/puma -e production
3. WAIT 5-15 or so minutes for all jitting to stop so we have no cross talk
4. Run ab -n 100 http://localhost:9292/
5. Wait for all new jitting to finish
6. Run ab -n 100 http://localhost:9292/

** Response time (ms)
Here is the response time milliseconds for each percentile.
Skipping 99%ile because it's the same as 100%ile in 100 calls.

	trunk	trunk JIT	single so JIT	objfcn JIT
50%	38	45	41	43
66%	39	50	44	44
75%	47	51	46	45
80%	49	52	47	47
90%	50	63	50	52
95%	60	79	52	55

| 98% | 91 | 114 | 91 | 91 |
|100% | 97 | 133 | 96 | 99 |

** Ratio (smaller is better)
Here is the response time increase ratio against no-JIT trunk's one.

| | trunk| trunk|single|objfcn|
		JIT	so JIT	JIT
50%	1.00x	1.18x	1.08x	1.13x
66%	1.00x	1.28x	1.13x	1.13x
75%	1.00x	1.09x	0.98x	0.96x
80%	1.00x	1.06x	0.96x	0.96x
90%	1.00x	1.26x	1.00x	1.04x
95%	1.00x	1.32x	0.87x	0.92x
98%	1.00x	1.25x	1.00x	1.00x
100%	1.00x	1.37x	0.99x	1.02x

While 50 and 60 %ile are still worse than no-JIT trunk, 75, 80, 90, 95,
98 and 100% are not slower than that.

So now it's a little harder to say "MJIT slows down Rails applications".
Probably I can close [Bug #14490] now. Let's start improving it.

Close https://github.com/ruby/ruby/pull/1921

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64094 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64094 - 07/28/2018 04:14 PM - k0kubun (Takashi Kokubun)

mjit.c: introduce JIT compaction [experimental]

When all compilation finishes or the number of JIT-ed code reaches
--jit-max-cache, this compacts all generated code to a single .so file
and re-loads all methods from it.

In the future, it may trigger compaction more frequently and/or limit
the maximum times of compaction to prevent unlimited memory usage.
So the current behavior is experimental, but at least the performance
improvement in this commit won't be removed.

=== Benchmark ===
In this benchmark, I'll compare following four conditions:

- trunk: r64082
- trunk JIT: r64082 w/ --jit
- single-so JIT: This commit w/ --jit
- objfcn JIT: This branch https://github.com/k0kubun/ruby/tree/objfcn w/ --jit, which is shinh's objfcn https://github.com/shinh/ruby/tree/objfcn rebased from this commit

```
$ uname -a
Linux bionic 4.15.0-29-generic #31-Ubuntu SMP Tue Jul 17 15:39:52 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux
```

- Micro benchmark Using this script https://gist.github.com/k0kubun/10e6d3387c9ab1b134622b2c9d76ef51, calls some amount of different methods that just return nil. The following tables are its average duration seconds of 3 measurements.

Smaller is better.

** 1 method (seconds)
	trunk	trunk JIT	single-so JIT	objfcn JIT
Time	5.576067774333296	5.915551971666446	5.833641665666619	5.845915191666639
Ratio	1.00x	1.06x	1.05x	1.05x

** 50 methods (seconds)
	trunk	trunk JIT	single-so JIT	objfcn JIT
Time	3.1661167996666677	6.125825928333342	4.135432743666665	3.750358728333348
Ratio	1.00x	1.93x	1.31x	1.18x

** 1500 methods (seconds)
	trunk	trunk JIT	single-so JIT	objfcn JIT
Time	5.971650823666664	19.579182102999994	10.511108153999961	10.854653588999932
Ratio	1.00x	3.28x	1.76x	1.82x

- Discourse Using the same benchmark strategy as https://bugs.ruby-lang.org/issues/14490 with this branch https://github.com/k0kubun/discourse/commits/benchmark2 forked from discourse v1.8.11 to support running trunk.

1. Run ruby script/bench.rb to warm up profiling database
2. Run RUBYOPT='--jit-verbose=1 --jit-max-cache=10000' RAILS_ENV=profile bin/puma -e production
3. WAIT 5-15 or so minutes for all jitting to stop so we have no cross talk
4. Run ab -n 100 http://localhost:9292/
5. Wait for all new jitting to finish
6. Run ab -n 100 http://localhost:9292/

** Response time (ms)
Here is the response time milliseconds for each percentile.
Skipping 99%ile because it's the same as 100%ile in 100 calls.

| | trunk| trunk|single|objfcn|
		JIT	so JIT	JIT
50%	38	45	41	43
66%	39	50	44	44
75%	47	51	46	45
80%	49	52	47	47
90%	50	63	50	52
95%	60	79	52	55
98%	91	114	91	91
100%	97	133	96	99

** Ratio (smaller is better)
Here is the response time increase ratio against no-JIT trunk's one.

| | trunk| trunk|single|objfcn|
		JIT	so JIT	JIT
50%	1.00x	1.18x	1.08x	1.13x
66%	1.00x	1.28x	1.13x	1.13x
75%	1.00x	1.09x	0.98x	0.96x
80%	1.00x	1.06x	0.96x	0.96x
90%	1.00x	1.26x	1.00x	1.04x
95%	1.00x	1.32x	0.87x	0.92x
98%	1.00x	1.25x	1.00x	1.00x
100%	1.00x	1.37x	0.99x	1.02x

While 50 and 60 %ile are still worse than no-JIT trunk, 75, 80, 90, 95,
98 and 100% are not slower than that.

So now it's a little harder to say "MJIT slows down Rails applications".
Probably I can close [Bug #14490] now. Let's start improving it.

Close https://github.com/ruby/ruby/pull/1921

Revision 64094 - 07/28/2018 04:14 PM - k0kubun (Takashi Kokubun)

mjit.c: introduce JIT compaction [experimental]

When all compilation finishes or the number of JIT-ed code reaches
--jit-max-cache, this compacts all generated code to a single .so file
and re-loads all methods from it.

In the future, it may trigger compaction more frequently and/or limit
the maximum times of compaction to prevent unlimited memory usage.
So the current behavior is experimental, but at least the performance
improvement in this commit won't be removed.

=== Benchmark ===
In this benchmark, I'll compare following four conditions:

- trunk: r64082
- trunk JIT: r64082 w/ --jit
- single-so JIT: This commit w/ --jit
- objfcn JIT: This branch https://github.com/k0kubun/ruby/tree/objfcn w/ --jit, which is shinh's objfcn https://github.com/shinh/ruby/tree/objfcn rebased from this commit

```
$ uname -a
Linux bionic 4.15.0-29-generic #31-Ubuntu SMP Tue Jul 17 15:39:52 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux
```

- Micro benchmark Using this script https://gist.github.com/k0kubun/10e6d3387c9ab1b134622b2c9d76ef51, calls some amount of different

methods that just return nil. The following tables are its average duration seconds of 3 measurements.

Smaller is better.

** 1 method (seconds)
	trunk	trunk JIT	single-so JIT	objfcn JIT
Time	5.576067774333296	5.915551971666446	5.833641665666619	5.845915191666639
Ratio	1.00x	1.06x	1.05x	1.05x

** 50 methods (seconds)
	trunk	trunk JIT	single-so JIT	objfcn JIT
Time	3.1661167996666677	6.125825928333342	4.135432743666665	3.750358728333348
Ratio	1.00x	1.93x	1.31x	1.18x

** 1500 methods (seconds)
	trunk	trunk JIT	single-so JIT	objfcn JIT
Time	5.971650823666664	19.579182102999994	10.511108153999961	10.854653588999932
Ratio	1.00x	3.28x	1.76x	1.82x

- Discourse Using the same benchmark strategy as https://bugs.ruby-lang.org/issues/14490 with this branch https://github.com/k0kubun/discourse/commits/benchmark2 forked from discourse v1.8.11 to support running trunk.

1. Run ruby script/bench.rb to warm up profiling database
2. Run RUBYOPT='--jit-verbose=1 --jit-max-cache=10000' RAILS_ENV=profile bin/puma -e production
3. WAIT 5-15 or so minutes for all jitting to stop so we have no cross talk
4. Run ab -n 100 http://localhost:9292/
5. Wait for all new jitting to finish
6. Run ab -n 100 http://localhost:9292/

** Response time (ms)
Here is the response time milliseconds for each percentile.
Skipping 99%ile because it's the same as 100%ile in 100 calls.

	trunk	trunk JIT	single so JIT	objfcn JIT
50%	38	45	41	43
66%	39	50	44	44
75%	47	51	46	45
80%	49	52	47	47
90%	50	63	50	52
95%	60	79	52	55
98%	91	114	91	91
100%	97	133	96	99

** Ratio (smaller is better)
Here is the response time increase ratio against no-JIT trunk's one.

	trunk	trunk JIT	single so JIT	objfcn JIT
50%	1.00x	1.18x	1.08x	1.13x
66%	1.00x	1.28x	1.13x	1.13x
75%	1.00x	1.09x	0.98x	0.96x
80%	1.00x	1.06x	0.96x	0.96x
90%	1.00x	1.26x	1.00x	1.04x
95%	1.00x	1.32x	0.87x	0.92x
98%	1.00x	1.25x	1.00x	1.00x
100%	1.00x	1.37x	0.99x	1.02x

While 50 and 60 %ile are still worse than no-JIT trunk, 75, 80, 90, 95,
98 and 100% are not slower than that.

So now it's a little harder to say "MJIT slows down Rails applications".
Probably I can close [Bug #14490] now. Let's start improving it.

Close https://github.com/ruby/ruby/pull/1921

History

#1 - 02/19/2018 01:56 AM - sam.saffron (Sam Saffron)

- Description updated

#2 - 02/19/2018 02:14 AM - k0kubun (Takashi Kokubun)

- Assignee set to k0kubun (Takashi Kokubun)

Yes, I recognize this problem and will investigate the cause for sure. I don't know a clear cause for this yet.

> I suspect this is happening cause we are introducing new overhead to every single method dispatch (counting for mjit, and hash lookup and so on). This overhead adds up.

My understanding of new overheads: JIT enqueue on 5 total calls, total calls count up (sometimes it happens twice for now), JIT-ed C function call instead continueing in the same VM, and additional VM invocation for not JIT-ed methods or JIT cancel. Some of those overheads might grow in Rails. I need investigation.

I'm not sure what's hash lookup.

#3 - 02/19/2018 02:38 AM - k0kubun (Takashi Kokubun)

> This was reported earlier but I would like to add some test methodology here: (tested using Discourse)

Ah, somehow I didn't read this part. Thank you for sharing! I'm using the similar way.

#4 - 07/28/2018 04:15 PM - k0kubun (Takashi Kokubun)

- Status changed from Open to Closed

Applied in changeset trunk|r64094.

---

mjit.c: introduce JIT compaction [experimental]

When all compilation finishes or the number of JIT-ed code reaches
--jit-max-cache, this compacts all generated code to a single .so file
and re-loads all methods from it.

In the future, it may trigger compaction more frequently and/or limit
the maximum times of compaction to prevent unlimited memory usage.
So the current behavior is experimental, but at least the performance
improvement in this commit won't be removed.

=== Benchmark ===
In this benchmark, I'll compare following four conditions:

- trunk: r64082
- trunk JIT: r64082 w/ --jit
- single-so JIT: This commit w/ --jit
- objfcn JIT: This branch https://github.com/k0kubun/ruby/tree/objfcn w/ --jit, which is shinh's objfcn https://github.com/shinh/ruby/tree/objfcn rebased from this commit

```
$ uname -a
Linux bionic 4.15.0-29-generic #31-Ubuntu SMP Tue Jul 17 15:39:52 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux
```

- Micro benchmark Using this script https://gist.github.com/k0kubun/10e6d3387c9ab1b134622b2c9d76ef51, calls some amount of different methods that just return nil. The following tables are its average duration seconds of 3 measurements.

Smaller is better.

** 1 method (seconds)

	trunk	trunk JIT	single-so JIT	objfcn JIT
Time	5.576067774333296	5.915551971666446	5.833641665666619	5.845915191666639
Ratio	1.00x	1.06x	1.05x	1.05x

** 50 methods (seconds)

	trunk	trunk JIT	single-so JIT	objfcn JIT
Time	3.1661167996666677	6.125825928333342	4.135432743666665	3.750358728333348
Ratio	1.00x	1.93x	1.31x	1.18x

** 1500 methods (seconds)

| | trunk | trunk JIT | single-so JIT | objfcn JIT |

|:------|:-----------------|:-----------------|:-----------------|:-----------------|
| Time | 5.971650823666664 | 19.579182102999994| 10.511108153999961| 10.854653588999932|
| Ratio | 1.00x | 3.28x | 1.76x | 1.82x |

- Discourse Using the same benchmark strategy as https://bugs.ruby-lang.org/issues/14490 with this branch https://github.com/k0kubun/discourse/commits/benchmark2 forked from discourse v1.8.11 to support running trunk.

1. Run ruby script/bench.rb to warm up profiling database
2. Run RUBYOPT='--jit-verbose=1 --jit-max-cache=10000' RAILS_ENV=profile bin/puma -e production
3. WAIT 5-15 or so minutes for all jitting to stop so we have no cross talk
4. Run ab -n 100 http://localhost:9292/
5. Wait for all new jitting to finish
6. Run ab -n 100 http://localhost:9292/

** Response time (ms)
Here is the response time milliseconds for each percentile.
Skipping 99%ile because it's the same as 100%ile in 100 calls.

| | trunk| trunk|single|objfcn|
		JIT	so JIT	JIT
50%	38	45	41	43
66%	39	50	44	44
75%	47	51	46	45
80%	49	52	47	47
90%	50	63	50	52
95%	60	79	52	55
98%	91	114	91	91
100%	97	133	96	99

** Ratio (smaller is better)
Here is the response time increase ratio against no-JIT trunk's one.

| | trunk| trunk|single|objfcn|
		JIT	so JIT	JIT
50%	1.00x	1.18x	1.08x	1.13x
66%	1.00x	1.28x	1.13x	1.13x
75%	1.00x	1.09x	0.98x	0.96x
80%	1.00x	1.06x	0.96x	0.96x
90%	1.00x	1.26x	1.00x	1.04x
95%	1.00x	1.32x	0.87x	0.92x
98%	1.00x	1.25x	1.00x	1.00x
100%	1.00x	1.37x	0.99x	1.02x

While 50 and 60 %ile are still worse than no-JIT trunk, 75, 80, 90, 95,
98 and 100% are not slower than that.

So now it's a little harder to say "MJIT slows down Rails applications".
Probably I can close [Bug #14490] now. Let's start improving it.

Close https://github.com/ruby/ruby/pull/1921