openQA Project - action #93086

unstable test in openQA master t/10-jobs.t exceeding runtime of 280s

2021-05-25 12:26 - okurz

Status:	Resolved	**Start date:**	2021-05-25
Priority:	High	**Due date:**	2021-06-23
Assignee:	kraih	**% Done:**	0%
Category:	Concrete Bugs	**Estimated time:**	0.00 hour
Target version:	Ready		
Difficulty:			

Description

Observation

https://app.circleci.com/pipelines/github/os-autoinst/openQA/6526/workflows/3caccede-e153-4757-b795-d97046d58228/jobs/61350

shows

```
RETRY=0 timeout -s SIGINT -k 5 -v $((25 * (0 + 1) ))m tools/retry prove -l --harness TAP::Harness::JUnit --timer t/10-jobs.t t/14-grutasks.t t/32-openqa_client.t t/44-scripts.t
[11:40:36] t/10-jobs.t ..... 22/? Bailout called.  Further testing stopped:  test exceeds runtime limit of '280' seconds
```

Acceptance criteria

- **AC1:** t/10-jobs.t takes significantly lower runtime of 280s in circleCI (with coverage analysis enabled)

Suggestions

- Run t/10-jobs.t locally with and without coverage analysis and check runtime for regression and optimization hotspots

Related issues:

Copied to openQA Project - coordination #93883: [epic] Speedup openQA coverag...	**Resolved**	**2021-07-07**

History

#1 - 2021-05-28 10:32 - kraih

- Assignee set to kraih

I'll run a profiler against the test to see what's going on.

#2 - 2021-05-28 10:38 - kraih

There might actually be a bug somewhere, because the test is not heavy at all. Locally it takes 4 seconds to finish.

```
All tests successful.
Files=1, Tests=41,  4 wallclock secs ( 0.06 usr  0.01 sys +  3.40 cusr  0.45 csys =  3.92 CPU)
Result: PASS
```

#3 - 2021-05-28 13:45 - mkittler

Did you test with coverage enabled?

#4 - 2021-05-28 13:58 - kraih

Ok, i think i can now replicate locally what is happening on Circle CI.

```
$ time TEST_PG='DBI:Pg:dbname=openqa_test;host=/dev/shm/tpg' HEAVY=1 DEVEL_COVER_DB_FORMAT=JSON perl -Ilib -MDevel::Cover=-select_re,'^/lib',+ignore_re,lib/perlcritic/Perl/Critic/Policy,-coverage,statement,-db,cover_db t/10-jobs.t
...
real    2m12.795s
user    2m5.298s
sys     0m5.640s
```

I will do a full profiling with NYTProf, to make sure, but i'm afraid so far it does look like the overhead is simply Devel::Cover processing its data and writing to the database in all the spawned Minion jobs. And if i disable the Devel::Cover::report() call for the Minion jobs the number drops sharply.

```
real    0m13.368s
user    0m11.653s
sys     0m0.894s
```

Perhaps there's ways to speed up Devel::Cover.

#5 - 2021-05-28 14:34 - kraih

So, there's 10 forked processes, most look like this:

```
Stmts   Exclusive Time Reports Source File
8813074 10.6s   line    B/Deparse.pm
3206105 3.54s   line    Devel/Cover.pm (including 3 string evals)
4570491 3.25s   line    B.pm
302536  530ms   line    Data/Dumper.pm
178862  269ms   line    Devel/Cover/DB/Structure.pm
209533  178ms   line    Devel/Cover/DB.pm
318697  129ms   line    Devel/Cover/Criterion.pm
90654   81.5ms  line    Devel/Cover/Statement.pm
10190   41.5ms  line    Devel/Cover/DB/IO/Base.pm
7197    39.6ms  line    Devel/Cover/DB/IO/JSON.pm
35925   33.9ms  line    warnings.pm
3689    22.4ms  line    Devel/Cover/DB/IO.pm (including 2 string evals)
29650   18.3ms  line    Devel/Cover/DB/File.pm
5830    9.64ms  line    DBIx/Class/Storage/DBIHacks.pm
619 7.74ms  line    DBIx/Class/Storage/DBI.pm
2635    6.41ms  line    JSON/MaybeXS.pm
189 5.32ms  line    Mojo/Pg/Database.pm
104 4.93ms  line    DBD/Pg.pm
2606    4.67ms  line    DBIx/Class/ResultSet.pm
2478    4.28ms  line    SQL/Abstract/Classic.pm
1002    2.67ms  line    DBIx/Class/ResultSource.pm
1245    1.88ms  line    DBIx/Class/SQLMaker/ClassicExtensions.pm
363 1.55ms  line    DBI.pm (including 2 string evals)
```

And some like this (i assume they have a no_cover setting active):

```
Stmts   Exclusive Time Reports Source File
155 989µs  line    IPC/Run.pm (including 1 string eval)
84  462µs  line    Devel/Cover.pm (including 3 string evals)
96  209µs  line    IPC/Run/Debug.pm (including 1 string eval)
0   0s  line    YAML/XS/LibYAML.pm
0   0s  line    YAML/XS.pm
0   0s  line    YAML/PP/Writer/File.pm
```

And then the main test process:

```
Stmts   Exclusive Time Reports Source File
94  130s    line    Minion/Job.pm
8811266 10.2s   line    B/Deparse.pm
4313728 4.77s   line    Devel/Cover.pm (including 3 string evals)
4745489 3.33s   line    B.pm
3079    882ms   line    IPC/Run.pm (including 1 string eval)
216646  551ms   line    DBIx/Class/Storage/DBI.pm
473804  510ms   line    Archive/Extract.pm
289556  499ms   line    SQL/SplitStatement.pm
302586  493ms   line    Data/Dumper.pm
335375  452ms   line    SQL/Abstract/Classic.pm
321298  404ms   line    DBIx/Class/ResultSet.pm
186789  295ms   line    Devel/Cover/DB/Structure.pm
172707  294ms   line    Class/Accessor/Grouped.pm (including 182 string evals)
188547  286ms   line    DBIx/Class/ResultSource.pm
233287  256ms   line    DBIx/Class/Storage/DBIHacks.pm
149411  202ms   line    DBIx/Class/Row.pm
164689  187ms   line    Try/Tiny.pm
209658  165ms   line    Devel/Cover/DB.pm
109961  155ms   line    DateTime.pm
```

And looking at the actual lines, there's also nothing too unexpected:

```
Calls   P   F   Exclusive Time  Inclusive Time  Subroutine
6   1   1   130s    130s         Minion::Job::CORE:waitpid (opcode)
```

```
1077415 29  2   1.61s   2.15s                 B::class
95217   92  2   1.39s   12.4s       Devel::Cover::deparse (recurses: max depth 22, inclusive time 30.2s)
2410    2   1   1.24s   2.21s         B::Deparse::populate_curcvlex
64472   3   1   849ms   1.41s         B::Deparse::_pessimise_walk (recurses: max depth 31, inclusive time 7.30
s)
12  1   1   840ms   840ms              POSIX::read (xsub)
```

#6 - 2021-05-28 16:09 - kraih

We are using $minion->perform_jobs to run all jobs currently in the queue sequentially. That could also be done in parallel with a custom worker. On my local dev machine that did cut runtime in half, but of course it depends very much on the available CPU, and might therefore also be very inconsistent on Circle CI.

```
real    1m0.666s
user    1m47.174s
sys     0m7.287s
```

#7 - 2021-05-29 21:05 - okurz

ok, I think then I understand how that could be a regression because over time we moved more functionality to minion jobs. We could try to not collect coverage information from minion subprocesses at all and see if we actually loose significant coverage. Or circumvent the minions in separate processes completely and execute the according code segments directly within the main test process. Theoretically this might be slower but makes tests deterministic hence also easier to debug and does not trigger the coverage collection in other processes which causes the slowdown.

#8 - 2021-05-31 16:19 - kraih

I did run a bunch of Circle CI tests with my hacked up support for parallel Minion job processing, and the results ended up fairly underwhelming unfortunately. The first result in each of these three runs is always the one with parallel jobs, and the second without.

```
[13:03:31] t/10-jobs.t ........... ok   153668 ms ( 0.08 usr  0.00 sys + 250.87 cusr  5.38 csys = 256.33 CPU)
[13:16:46] t/10-jobs.t ........... ok   173135 ms ( 0.07 usr  0.00 sys + 166.85 cusr  4.42 csys = 171.34 CPU)

[14:01:59] t/10-jobs.t ........... ok   151535 ms ( 0.08 usr  0.00 sys + 241.61 cusr  5.61 csys = 247.30 CPU)
[14:15:16] t/10-jobs.t ........... ok   162337 ms ( 0.08 usr  0.00 sys + 156.56 cusr  4.60 csys = 161.24 CPU)

[14:32:42] t/10-jobs.t ........... ok   143575 ms ( 0.06 usr  0.00 sys + 231.69 cusr  5.24 csys = 236.99 CPU)
[15:02:31] t/10-jobs.t ........... ok   165410 ms ( 0.07 usr  0.00 sys + 160.33 cusr  4.07 csys = 164.47 CPU)
```

For the three runs above the parallel job limit was 10, i also did another run with 4 parallel jobs, and the results improved a little bit again (so there's probbaly a sweet spot where we use Circle CI resources most efficiently).

```
[16:03:02] t/10-jobs.t ........... ok   120680 ms ( 0.07 usr  0.01 sys + 188.07 cusr  4.51 csys = 192.66 CPU)
```

My hack to make it work is not really production quality, but i can offer to make it a proper upstream feature in Minion ($t->app->minion->perform_jobs({jobs => 4}) or so). Not sure it's really worth it though.

#9 - 2021-06-01 14:52 - mkittler

A cheap workaround is already present in t/42-df-based-cleanup.t but I know you don't like giving up the forking.

Not sure whether parallelization would gain us much considering that the test likely just spawns one job at a time and then calls perform_minion_jobs.

> Perhaps there's ways to speed up Devel::Cover.

That would of course be very good because it would also help with many other tests slowed down by this (also within os-autoinst).

#10 - 2021-06-07 15:34 - kraih

mkittler wrote:

> A cheap workaround is already present in t/42-df-based-cleanup.t but I know you don't like giving up the forking.

Not sure why i didn't think of just doing that. :) That is a very good idea and i'll probably add a cleaner implementation of it to OpenQA::Test::Utils.

#11 - 2021-06-08 12:10 - kraih

That does look like a success on Circle CI.

```
[15:39:48] t/10-jobs.t ........... ok    19263 ms ( 0.07 usr  0.01 sys + 18.16 cusr  0.66 csys = 18.90 CPU)
```

#12 - 2021-06-08 12:24 - kraih

Opened a PR. https://github.com/os-autoinst/openQA/pull/3934

#13 - 2021-06-08 12:24 - kraih

- Status changed from Workable to In Progress

#14 - 2021-06-09 04:29 - openqa_review

- Due date set to 2021-06-23

Setting due date based on mean cycle time of SUSE QE Tools

#15 - 2021-06-11 09:03 - kraih

- Status changed from In Progress to Resolved

PR has been merged. The feature will also be available upstream from Minion soon.
https://github.com/mojolicious/minion/compare/de1057fa4607...b8bb86e119c2

#16 - 2021-06-11 09:39 - okurz

- Copied to coordination #93883: [epic] Speedup openQA coverage tests with running minion jobs synchronously using new upstream "perform_jobs_in_foreground" mojo function added