Ruby trunk - Feature #9508

Add method coverage and branch coverage metrics

02/10/2014 06:21 PM - srawlins (Sam Rawlins)

Status:	Assigned
Priority:	Normal
Assignee:	mame (Yusuke Endoh)
Target version:	

Description

Since the Coverage extension was introduced in Ruby 1.9, Ruby has had built-in line code coverage. Ruby should support more of the basic code coverage metrics [1]. I have a pull request on GitHub (https://github.com/ruby/ruby/pull/511) to add Method Coverage (Function Coverage) and Branch Coverage. I'd love feedback to improve it.

Currently, with this feature, Coverage.result would look like:

```
{"/Users/sam/code/ruby/cov_method.rb" => {
  lines: [1, 2, 2, 20, nil, nil, 2, 2, 2, nil, 0, nil, nil, nil, 1, 0, nil, nil, 1, 1, nil, nil, 1
],
  methods: {1=>2, 15=>0, 19=>1},
  branches: {8=>2, 11=>0}
}}
```

which includes

- the current Ruby line coverage report,
- as well as a method report (The method defined on line 1 was called 2 times; the method on line 15 was called 0 times; ...),
- and a branch report (the branch on line 8 was called 2 times; the branch on line 11 was called 0 times).

Branches

Branches include the bodies of if, elsif, else, unless, and when statements, which are all tracked with this new feature. However, this feature is not aware of void bodies, for example:

```
if foo
  :ok
end
```

will report that only one branch exists in the file. It would be better to declare that there is a branch body on line 2, and a void branch body on line 3, or perhaps line 1. This would require the keys of the [:branch] Hash to be something other than line numbers. Perhaps label_no? Perhaps nd_type(node) paired with line or label_no?

More Coverage

I think that Statement Coverage, and Condition Coverage could be added to this feature, using the same techniques.

Caveats

I was not very clear on the bit-arrays used in ruby.h, and just used values for the new macros that seemed to work.

Also, I would much rather use Ranges to identify a branch, so that a Coverage analyzer like SimpleCov won't need any kind of Ruby parser to identify and highlight a full chunk of code as a tested branch, or a not tested branch. I'm trying to find how that could be implemented...

[1] Wikipedia has good definitions: http://en.wikipedia.org/wiki/Code_coverage

History

#1 - 02/10/2014 06:27 PM - srawlins (Sam Rawlins)

- File pull-request-511.patch added

Here's the pull request as a patch.

#2 - 02/10/2014 10:01 PM - normalperson (Eric Wong)

I suggest using a coverage struct pointer in rb_iseq_struct to hold all
3 coverage VALUEs. rb_iseq_struct is gigantic already, so adding 2 more
VALUEs for infrequent coverage use can lead to even more bloat.

#3 - 02/21/2014 05:03 AM - srawlins (Sam Rawlins)

- File pull-request-511.patch added

Good call Eric. I've carried out your suggestion [1], and attached the cumulative patch.

[1] this commit: https://github.com/srawlins/ruby/commit/cc50eab44f5ce0a4febdc05bdd99a09708e78b7e

#4 - 02/21/2014 07:58 AM - normalperson (Eric Wong)

sam.rawlins@gmail.com wrote:

> Good call Eric. I've carried out your suggestion [1], and attached the cumulative patch.
>
> [1] this commit: https://github.com/srawlins/ruby/commit/cc50eab44f5ce0a4febdc05bdd99a09708e78b7e

Thanks Sam! I didn't check very closely, but in places where before
where you checked for "iseq->coverage" being true, now jumps straight
to "iseq->coverage->(methods|branches|lines)".

Wouldn't that crash if iseq->coverage isn't set at all?

Unless I'm missing another check elsewhere, perhaps checking:

```
(iseq->coverage && iseq->coverage->FOO)
```

is safer.

#5 - 02/26/2014 03:22 AM - srawlins (Sam Rawlins)

- File pull-request-511.patch added

I completely rewrote and rebranded "Branch" coverage into "Decision" coverage after reading Steve Cornett's paper [1] closer. The resultant metric is much more robust, tracking void elses, and one-line if/else combos much better. Now every conditional statement, like:

```
:yes if 2+2 == 4
```

will have a count of truthy decisions, and falsey decisions to come out of the statement. This code here would register 1 truthy decision, and 0 falsey decisions, resulting in incomplete decision coverage.

Patch is attached. Also, I forgot to mention that this feature is well-tested, with the updated and new tests in test/coverage/

[1] http://www.bullseye.com/coverage.html

#6 - 02/26/2014 03:23 AM - srawlins (Sam Rawlins)

Hi Eric, thanks so much for reviewing this.

I took the same safety precautions that appeared around the new code. So, for example in iseq.c, I setup isec->coverage in the same place that I see iseq-> compile_data is set (lines 117 and 124). The other spot you might be talking about is thread.c, inside the new update_method_coverage() and update_decision_coverage(). I added more precaution to the update_decision_coverage() method, and could do the same to update_method_coverage().

Are there examples you had other than those two spots?

#7 - 02/26/2014 09:00 AM - normalperson (Eric Wong)

sam.rawlins@gmail.com wrote:

> Are there examples you had other than those two spots?

The macros in compile.c, but then I noticed. something I missed
earlier: rb_coverage_struct is always allocated.

The reason I wanted rb_coverage_struct to be separate pointer is to save
memory since coverage is not a common case, so avoid allocating memory

unless coverage is enabled.

So what I mean was to do something like this:
http://yhbt.net/iseq-coverage.diff

#8 - 02/26/2014 09:28 AM - srawlins (Sam Rawlins)

Ah, I see. You're right; it's much better to allocate it only when tracking coverage. I've implemented that in a new commit [1]; I only had to add more checks to the _TRACE macros at the top of compile.c.

I also added decision coverage-tracking for while and until.

[1] https://github.com/srawlins/ruby/commit/eaadf820633e74350404d009a1c251f6319454aa

#9 - 03/01/2014 10:42 PM - normalperson (Eric Wong)

I finally tried commit eaadf820633e74350404d009a1c251f6319454aa
and it segfaults right away:
ruby -rcoverage -e 'Coverage.start; require "tempfile"'
Backtrace here: http://yhbt.net/feature-9508.bt.txt

#10 - 03/02/2014 08:02 AM - srawlins (Sam Rawlins)

Oh, sorry, Eric. eaadf820633e74350404d009a1c251f6319454aa was just the last commit I made to tweak when coverage is initialized. The entire patch would be my cumulative pull request:

https://github.com/ruby/ruby/pull/511.patch

I can compile ruby and run all coverage tests:

```
$ ../ruby-bin/bin/ruby test/coverage/test_coverage.rb
Run options:

# Running tests:

Finished tests in 0.059334s, 67.4147 tests/s, 269.6587 assertions/s.
4 tests, 16 assertions, 0 failures, 0 errors, 0 skips

ruby -v: ruby 2.2.0dev (2014-02-23) [x86_64-linux]
$ ../ruby-bin/bin/ruby test/coverage/test_method_coverage.rb
Run options:

# Running tests:

Finished tests in 0.004020s, 248.7709 tests/s, 746.3128 assertions/s.
1 tests, 3 assertions, 0 failures, 0 errors, 0 skips

ruby -v: ruby 2.2.0dev (2014-02-23) [x86_64-linux]
$ ../ruby-bin/bin/ruby test/coverage/test_decision_coverage.rb
Run options:

# Running tests:

Finished tests in 0.006942s, 720.2504 tests/s, 3601.2521 assertions/s.
5 tests, 25 assertions, 0 failures, 0 errors, 0 skips

ruby -v: ruby 2.2.0dev (2014-02-23) [x86_64-linux]
```

#11 - 03/02/2014 09:21 AM - normalperson (Eric Wong)

sam.rawlins@gmail.com wrote:

> Oh, sorry, Eric. eaadf820633e74350404d009a1c251f6319454aa was just the last commit I made to tweak when coverage is initialized. The entire patch would be my cumulative pull request:
>
> https://github.com/ruby/ruby/pull/511.patch

Yes, and your patch results in tree 7a51f93796a28bcbf9cb8fa25c6b752202c517ae
(which is your commit eaadf820633e74350404d009a1c251f6319454aa)

I also tried applying your series on top of current trunk, and got the
same result.

> I can compile ruby and run all coverage tests:

"make check" passed for me, too; but did you try my failing command?
ruby -rcoverage -e 'Coverage.start; require "tempfile"'

#12 - 03/03/2014 04:17 AM - srawlins (Sam Rawlins)

Hi Eric, great find! It turns out the bug here was when requiring Shared Objects (etc.so in this case). I've fixed that in the last commit. Cumulative patch available at the pull request:

https://github.com/ruby/ruby/pull/511.patch

#13 - 03/03/2014 07:02 AM - normalperson (Eric Wong)

sam.rawlins@gmail.com wrote:

> Hi Eric, great find! It turns out the bug here was when requiring Shared Objects (etc.so in this case). I've fixed that in the last commit. Cumulative patch available at the pull request:
>
> https://github.com/ruby/ruby/pull/511.patch

Thanks. I just tried it with dtas[1] and realized it's not compatible with the existing Coverage.result.

I guess this breaks existing, widely-used coverage tools like simplecov, too?

How about keeping Coverage.result the same, and allowing Coverage.result2 or maybe Coverage.result(:all)?

I'm excited about this feature, but we should try to not break existing tools.

Another thing I noticed with your latest fix:
You call rb_hash_lookup(coverages, path) 3 times more than you need to. I don't know if there's a measurable speed difference, but it's still ugly (and yeah, I just spent a fair amount of time trying to eek out the last bit of hash lookup performance in #9425).

[1] git clone git://80x24.org/dtas
cd dtas
cat test/helper.rb
cat test/covshow.rb
make coverage # (needs sox/flac/mp3gain installed, at least)

#14 - 03/03/2014 08:06 AM - srawlins (Sam Rawlins)

Hi Eric,

I'd actually like to keep the format of Coverage.result as the new format (Coverage.result values are each a Hash with :lines, :methods, and :decisions keys), rather than the existing Ruby 2.1.0 format, for two reasons:

1) Currently, the call to Coverage.result is very destructive: it immediately freezes the Coverage results, and clears the line coverage arrays. I don't think we can cleanly retrieve the line coverage, then maybe later the method coverage, always leaving the collective coverage results in an indeterminate "some cleared and some not cleared" state... and not really remember (or even let be discoverable) which results have been wiped and which haven't. [1]

2a) The very top of the Coverage documentation reads "Coverage provides coverage measurement feature for Ruby. This feature is experimental, so these APIs may be changed in future." so tools should be ready for a change.

2b) This change should be very easy to sniff out. For example, in SimpleCov, only one small change is needed in simple_cov/result.rb [2] in order to remain compatible with old Coverage and new:

```
   def initialize(original_result)
     @original_result = original_result.freeze
     @files = SimpleCov::FileList.new(original_result.map do |filename, coverage|
-      SimpleCov::SourceFile.new(filename, coverage) if File.file?(filename)
+      if coverage.is_a? Array  # Ruby < 2.2.0
+        SimpleCov::SourceFile.new(filename, coverage) if File.file?(filename)
+      else                     # Ruby >= 2.2.0
+        SimpleCov::SourceFile.new(filename, coverage[:lines]) if File.file?(filename)
+      end
     end.compact.sort_by(&:filename))
     filter!
   end
```

Of course the decision isn't up to me, but multiple methods like Coverage.result2 or Coverage.result(metric = :lines) or something feels bad... maybe there is another solution.

I'll amend my code with fewer rb_hash_lookups. My code there is super ugly... your suggestion will be much better. And #9425 is looking very promising!

[1] footnote: I'm not a huge fan of the Coverage.result methodology... but I think it was written this way so that the returned Coverage hash of results is now eligible for garbage collection. I'd rather a method called maybe Coverage.end that ends coverage (and neatly pairs with Coverage.start) and returns results that it does today. AND that those results can again be retrieved later, at will, by calling Coverage.result. Those results would be valid and unchanged until Coverage.start is called again, at which point the results are cleared (rather than when Coverage.end is called). But this would be for another ticket, if its even worth opening soon.
[2] https://github.com/colszowka/simplecov/blob/master/lib/simplecov/result.rb#L26

#15 - 03/03/2014 08:51 AM - normalperson (Eric Wong)

Fair enough on the changes (I cannot make decisions on API changes).
We shall wait for others (mame?) to respond.

I found another failure, this time with fork:

```
ruby -rcoverage -e 'Coverage.start; fork {}'
```

backtrace: http://yhbt.net/feature-9508-2.bt.txt

#16 - 03/03/2014 11:05 AM - mame (Yusuke Endoh)

Hello, Sam and Eric

Eric Wong wrote:

> Fair enough on the changes (I cannot make decisions on API changes).
> We shall wait for others (mame?) to respond.

Yes, I'm the original author and current maintainer of ext/coverage.

I'm positive for Branch coverage itself! I don't care the API so much because it is a "backend" library that casual users should not use directly. However, I don't think that it is a good idea to break compatibility with no strong reason.

I also thank you for providing a patch. But sorry, at the moment, I can't afford the time to review it. For a time, could you address the problems that Eric pointed?

Eric, thank you for your reviewing!

--
Yusuke Endoh mame@tsg.ne.jp

#17 - 03/03/2014 10:06 PM - srawlins (Sam Rawlins)

Eric, I could not recreate your fork failure (I'm on OS X 10.6, compiling with gcc 4.2.1...). However, I made update_method_coverage safer, thanks to your backtrace. I've updated the pull request with that fix, and fewer rb_hash_lookup calls:

https://github.com/ruby/ruby/pull/511.patch

#18 - 03/03/2014 10:28 PM - srawlins (Sam Rawlins)

Endoh-san and Eric, thanks both for considering these changes. I'm in favor of changing the format of Coverage.result because of the reasons I outline in above, but I can give more evidence that it should be a very safe change:

After a code search on GitHub [1], and after looking at the Code Metrics section of The Ruby Toolbox [2], it is clear that the Ruby community in general (at least open source) rely almost 100% on the SimpleCov gem [3]. Many of the other metrics tools found in the Ruby Toolbox use SimpleCov (coveralls, flog, rails_best_practices, and more) and never call Coverage.result manually. I can find no other libraries currently maintained that execute "Coverage.result".

And I still just don't like the idea of some Coverage results being duped/frozen/cleared, while others are not.

Another idea was suggested to me, where we introduce an optional parameter to Coverage.start, which defaults to :lines. Old libraries can continue to call Coverage.start, which will cause Coverage.result to return a Hash of Ruby files mapped to line counts. New libraries (or updated SimpleCov) can call Coverage.start(:all), which will cause Coverage.result to return the new, more complicated format.

This solution, however, makes it harder for SimpleCov to be compatible with old AND new format: the library must figure out whether Coverage.start takes an optional argument or not (or use something like RUBY_VERSION), track that, and parse Coverage.result differently based on the answer. I would again just prefer upgrading the format to this solution, allowing SimpleCov to just test coverage.is_a? Array.

[1] https://github.com/search?l=ruby&q=%22Coverage.start%22&ref=searchresults&type=Code

[2] https://www.ruby-toolbox.com/categories/code_metrics
[3] https://github.com/colszowka/simplecov

#19 - 03/04/2014 08:09 AM - normalperson (Eric Wong)

I'm still hitting it. Can you try adding waitpid2 to reproduce it?
ruby -rcoverage -e 'Coverage.start; Process.waitpid2(fork {})'
Thanks.

#20 - 03/06/2014 08:05 AM - srawlins (Sam Rawlins)

Hi Eric, thanks for that test. It revealed my poor choice for #define RUBY_EVENT_MCOVERAGE 0x040000 (RUBY_INTERNAL_EVENT_SWITCH is the same). I've updated my pull request with better choices in include/ruby/ruby.h: https://github.com/ruby/ruby/pull/511.patch

Another design question: I think that block calls should probably also be considered part of "Method Coverage" (which would then be renamed to "Function Coverage", tracking both method and block coverage). One big design problem here is that it is common to chain several blocks in one line, like iterators. Should I put in the work to track block coverage now, or wait until after this gets merged?

#21 - 03/07/2014 10:52 AM - normalperson (Eric Wong)

sam.rawlins@gmail.com wrote:

> Hi Eric, thanks for that test. It revealed my poor choice for #define RUBY_EVENT_MCOVERAGE 0x040000 (RUBY_INTERNAL_EVENT_SWITCH is the same). I've updated my pull request with better choices in include/ruby/ruby.h: https://github.com/ruby/ruby/pull/511.patch

Thanks Sam, works for me! I'll try to play with it more next week.
(and maybe extract the dtas/yahns coverage hack into its own gem for
GUI-phobes like myself)

I'll let mame answer your design question.

#22 - 05/03/2014 05:01 PM - mame (Yusuke Endoh)

- Status changed from Open to Feedback

Sorry for the very late response. I tried and read through your patch.
However, at first, I'd like to discuss the proposal itself.

Demand

In fact, I think we can virtually implement this feature by using a Ruby code parser, such as ripper.

For example, method coverage is usually identical to the execution count of the first line of each method.
(Of course, to make it precise, there might be many annoying cases, such as a method defined in one line.)

In similar way, you can measure decision coverage by parsing if/then/else and case/when statements.

If there is a great demand for this feature, I'm not against embedding it to the core. But, is it really needed?

Use case

Is it fully-clarified what type of visualization and analysis you want to do? Does the proposed API give you enough information for your use case?
If not, we will have to extend the API repeatedly, or even worse, the API will turn out not to be unusable after it is released.

For example, method coverage does not include method name. Decision coverage does not include lineno of "else" statement. Are they okay?

The current API is designed for an apparent use case: to visualize the execution count of each line.

Performance

I'm afraid if it is heavy to measure decision coverage because the patch calls rb_hash_lookup in each branch. I consulted the following micro benchmark:

https://gist.github.com/mame/2c1100664d452bff133a

It takes longer two times than the current. Doesn't it matter?
Just one idea: it would be good to allow a user to specify what s/he want to measure, like:

```
Coverage.start(line: true, method: false, decision: false) # default?
Coverage.start(line: true, method: true, decision: true)   # all
```

Please let me know if you have a benchmark in practical case. Though I did "make test-all" with coverage measurement, it caused core dump:

```
$ time make test-all RUN_OPTS="--disable-gems -r./sample/coverage.rb"
*snip*
[ 3559/15034] TestDir#test_close*** Error in `./test/runner.rb': munmap_chunk(): invalid pointer: 0x00002ae91f
a42770 ***
Aborted (core dumped)
```

However, I guess this is not a problem of your patch but a pre-existing GC bug that is triggered by your patch.

Review comments

Hereinafter, I describe review comments for your patch.
I think there is no big problem except ruby.h.

include/ruby/ruby.h

```
#define RUBY_EVENT_DEFN      0x0090
#define RUBY_EVENT_DECISION_TRUE  0x00a0
#define RUBY_EVENT_DECISION_FALSE 0x00b0
```

I think we don't have to declare these three constants here since they are used only in compile.c.

```
#define RUBY_EVENT_MCOVERAGE              0x080000
```

Seems like this event is identical to RUBY_EVENT_CALL, i.e., RUBY_EVENT_MCOVERAGE is fired if and only if RUBY_EVENT_CALL is fired. If so, this constant is not needed.

```
#define RUBY_EVENT_DCOVERAGE_TRUE        0x2000000
#define RUBY_EVENT_DCOVERAGE_FALSE       0x4000000
```

These two are needed for decision coverage.
But ko1 hesitates to add a new type of event unless it is really needed.
We must persuade ko1.

parse.y

```
VALUE rb_file_coverage = rb_hash_new();
VALUE methods = rb_hash_new();
VALUE decisions = rb_hash_new();
```

By using ObjectSpace.each_object, a user can get a reference to these objects and destroy them. You should use RBASIC_CLEAR_CLASS to make them invisible for users. (But invisible objects may cause another problem. As I recall, rb_hash_lookup might not be used for an invisible object.)

thread.c

clear_coverage deletes the coverage information measured so far. I think it also should delete method and decision coverage.
When Kernel#fork is called, this function is used in the child process, because the parent and the child has the same coverage information which may lead to duplicated measurement.

compile.c

```
#define ADD_METHOD_COVERAGE_TRACE(seq, line, event, end_line)
```

end_line is not used.

sample/coverage.rb

This file must be updated because of the API change. I'm still unsure if the API change is not harmful, though.

Thank you,

#23 - 05/09/2014 05:04 PM - srawlins (Sam Rawlins)

Hi Yusuke, thanks for the comments! I want to first defend the Demand and Use Case. And thank you for the Review comments; I'll apply them ASAP.

Demand in Ruby Core

I think that Ripper is inadequate for these new metrics for the exact reason you mention: if/else code and methods that are all defined within one line, as well as implicit "else". Code like this of course exists everywhere, so a tool would be greatly inadequate if it could not give metrics regarding lines like "return if x.nil?" or "def foo; bar.baz; end".

Visualization and Analysis

I was largely inspired by the visualization and analysis of Istanbul [1], the standard Javascript coverage library. Here is a great example of a coverage report: http://gotwarlost.github.io/istanbul/public/coverage/lcov-report/istanbul/lib/report/html.js.html

- On line 79, the report shows that the "else" branch is not taken.
- On line 171, the report shows that the "if" branch is not taken, and this if/else is all one line! Line coverage shows that the line is executed, because at a minimum, the line is reached, and the condition is evaluated.
- On line 522, the report shows that an implicit "else" branch is not taken. This is important because the line coverage looks fine, but the user is unaware, without Decision Coverage, that no test exercises a false condition in that "if." I think this is perhaps the most exciting and useful example of decision coverage.

I'm going to look into your performance notes. Thanks for benchmarking!

[1] https://github.com/gotwarlost/istanbul

#24 - 05/10/2014 12:41 AM - mame (Yusuke Endoh)

- Category set to core

- Status changed from Feedback to Assigned

- Assignee set to mame (Yusuke Endoh)

- Target version set to 2.2.0

Sam Rawlins wrote:

> I was largely inspired by the visualization and analysis of Istanbul [1], the standard Javascript coverage library. Here is a great example of a coverage report: http://gotwarlost.github.io/istanbul/public/coverage/lcov-report/istanbul/lib/report/html.js.html

Thank you for the explanation, I understood well what you want to do.

A minor question: Istanbul ignores ternary operators, but the proposed API can not distinguish if/else branches and ternary operators. Is that okay?

I think that the last major problem for this proposal is that ko1 is not very keen to add a new type of events. I'll talk with him.

Thank you,

--
Yusuke Endoh mame@tsg.ne.jp

#25 - 05/22/2014 02:29 AM - srawlins (Sam Rawlins)

Hi Yusuke, I looked into the performance issue:

- I used the mail gem specs as a slightly longer performance test. Without Coverage, the specs take 10.5 seconds. The current Coverage library increases that by 14%. My proposed changes instead increase by 57% (6 seconds total). Ouch!
- I used the jekyll gem specs as a much longer performance test. Without Coverage, the specs take 112 seconds. The current Coverage library increases that by 4%. My proposed changes instead increase by 11% (12 seconds total). Not too bad...
- results here: https://gist.github.com/srawlins/5b0fe367cd3a412e6925

Maybe this proposal (with Hashes) isn't too bad. However, I wrote a patch to my proposed changes (changing the :methods and :decisions values to be Arrays instead of Hashes), which decreases the slowdown by some, but not much (instead of increasing mail specs by 57%, this patch only increases them 38%; 4 seconds).

Another possible solution is to allow the user to specify what should be tracked, with something like

```
Coverage.start  # track everything?
Coverage.start(:lines)  # track lines
Coverage.start(:lines, :methods)  # track lines and methods
```

This would be a large change, and I would prefer writing it in a different feature if we want to do it...

What do you think of all of this?

#26 - 01/05/2018 09:00 PM - naruse (Yui NARUSE)

- Target version deleted (2.2.0)

Files

pull-request-511.patch	26.7 KB	02/10/2014	srawlins (Sam Rawlins)
pull-request-511.patch	38.5 KB	02/21/2014	srawlins (Sam Rawlins)