Arvados - Bug #8998

[API] Memory overflow when dumping a 25TB collection as JSON

04/15/2016 09:10 AM - Peter Grandi

Status:	Resolved	**Start date:**	04/15/2016
Priority:	Normal	**Due date:**	
Assigned To:	Peter Amstutz	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	2016-05-11 sprint		

Description

When uploading a collection of nearly 25TiB in a bit over 3,000 files with arv-put the outcome was:

```
librarian@rockall$ arv-put --replication 1 --no-resume --project-uuid gcam1-j7d0g-k25rlhe6ig8p9na
 --name DDD_WGS_EGAD00001001114 DDDP*
25956153M / 25956153M 100.0%
arv-put: Error creating Collection on project: <HttpError 422 when requesting https://gcam1.exampl
e.com/arvados/v1/collections?ensure_unique_name=true&alt=json returned "#<NoMemoryError: failed to
 allocate memory>">.
Traceback (most recent call last):
 File "/usr/local/bin/arv-put", line 4, in <module>
   main()
 File "/usr/local/lib/python2.7/dist-packages/arvados/commands/put.py", line 533, in main
   stdout.write(output)
UnboundLocalError: local variable 'output' referenced before assignment
```

From logs:

```
Oh... fiddlesticks.

An error occurred when Workbench sent a request to the Arvados API server. Try reloading this page
. If the problem is temporary, your request might go through next time. If that doesn't work, the
information below can help system administrators track down the problem.

API request URL
    https://gcam1.camdc.genomicsplc.com/arvados/v1/collections/gcam1-4zz18-i4nlpovriwdxu6j
API response

    {
      ":errors":[
        "#<NoMemoryError: failed to allocate memory>"
      ],
      ":error_token":"1457687649+b12feaf3"
    }
```

and I have attached the longer backtrace from a previous log.

A 25TB upload should result in a 15MB manifest, large but it should not overflow the API server that has 4GiB of memory.

Anyhow we can allocate more GiB of memory, but it would be nice to have a guideline as to how many are needed in relationship to largest collection size.

Perhaps 25TB collections are too large, especially considering the resulting manifest size, and my understanding that any access to a file in a collection results in the latency of a download of the full manifest.

But I have been told that we have a requirement for arbitrary naming conventions, where it is not acceptable to split large sets of data (many small files or fewer large files) into separate collections, like "data-subset-1", "data-subset-2", "data-subset-3", ... solely because of storage system limitations.

Subtasks:

Task # 9094: Testing	**Resolved**
Task # 9093: Review 8998-optimize-decode-www-form-component	**Resolved**
Task # 9092: Update JSON library usage	**Closed**

Associated revisions

Revision 8b097ea8 - 05/03/2016 01:20 PM - Peter Amstutz

Merge branch '8998-optimize-decode-www-form-component' closes #8998

History

#1 - 04/15/2016 09:13 AM - Peter Grandi

- File 160311_arvOutOfMemBt.txt added

Attached backtrace of the likely overflow.

#2 - 04/15/2016 02:21 PM - Brett Smith

- Subject changed from Keep: memory overflow on upload of 25TB collection to [API] Memory overflow when dumping a 25TB collection as JSON

Peter,

Thanks for attaching the backtrace. It's sort of a good news/bad news situation. The good news is, the API server got through saving your collection just fine, so your work is saved in the database and nothing is lost. It ran out of RAM when trying to render the collection back out as JSON. You should be able to confirm this yourself by getting a list of collections, from the CLI or Workbench. Collection listings don't include the collections' manifest_text by default, so that should avoid taxing the JSON parser too much and let you verify that the new collection exists. It would be a good double-check if you can try that and let us know if that's consistent with what you're seeing.

If you do see the new collection in a listing, the next check would be to try to get that specific collection from the API. That should similarly fail with an out of memory error, because it's the same code path.

We use the Oj Ruby Gem to render JSON, because in previous benchmarks we saw it had the best time performance. We may need to do another round of comparisons based on RAM use. The backtrace starts inside one of Oj's methods.

#3 - 04/15/2016 04:03 PM - Brett Smith

- File ojram.rb added

Wrote the attached script to take a quick glance at Oj's memory use when dumping a simple string. At the kinds of input sizes we're looking at (multiple tens of megabytes), the peak VM size of the process ends up being about 52 times the length of the input string.

It also seems to allocate RAM in a single large MMAP:

```
% strace -e mmap ruby ojram.rb 20m
[...]
mmap(NULL, 1027616768, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fa2cb2d8000  # Almost 1G

% strace -e mmap ruby ojram.rb 40m
[...]
mmap(NULL, 2055221248, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f410842f000  # Almost 2G
```

I can easily believe that Linux would've refused an mmap on this order of magnitude for the kind of manifest we're talking about, depending on how much swap is available, the relevant VM settings, and what else was running on the server at the time.

#4 - 04/15/2016 07:35 PM - Brett Smith

- File yajlram.rb added

Doing the same basic test on yajl-ruby showed it using RAM about five times the size of the input string. It's also written in C, so it should be fast enough. I think we should consider a switch?

#5 - 04/15/2016 07:38 PM - Brett Smith

(03:36:02 PM) Me: When you did JSON gem evaluations, did you look at Yajl?
(03:36:15 PM) tomclegg: not iirc
(03:36:33 PM) Me: Would you be open to switching, based on those RAM numbers I got?
(03:36:36 PM) tomclegg: 5x sounds better than 52x
(03:36:54 PM) tomclegg: Yes, certainly.

#6 - 04/18/2016 01:44 PM - Peter Grandi

Sorry, but this memory overflow happened many weeks ago, so most state is no longer available. Logs and backtraces had been saved though.

I can't remember exactly, but I suspect that the collection itself was not registered, or not completely.

#7 - 04/18/2016 01:47 PM - Peter Grandi

Also note that the specific error message printed by arv-put says "arv-put: Error creating Collection on project:", but if the collection has actually been created then the message could be changed.

#8 - 04/19/2016 02:48 PM - Brett Smith

Peter Grandi wrote:

> Sorry, but this memory overflow happened many weeks ago, so most state is no longer available. Logs and backtraces had been saved though.

Assuming your API server wasn't super-busy at the time, the timestamps in the logs should be sufficient for you to search for the collection that should exist. Something like:

```
arv collection list -f '[["created_at", ">", "2016-03-11T07:11:30Z"], ["created_at", "<", "2016-03-11T07:11:35Z"]]'
```

> I can't remember exactly, but I suspect that the collection itself was not registered, or not completely.

This would be really good to double-check using the techniques I suggested. Because if you're right that the collection was not registered correctly in the database, that suggests that the solution we're currently considering is barking up the wrong tree, and wouldn't provide a permanent resolution to this problem.

> Also note that the specific error message printed by arv-put says "arv-put: Error creating Collection on project:", but if the collection has actually been created then the message could be changed.

Yes, but that's going to require some support from the API server too. Right now it doesn't distinguish "failed to create collection" from "failed to render created collection" in its response, so the client has no way to distinguish the two cases.

#9 - 04/27/2016 08:16 PM - Peter Amstutz

- Assigned To set to Peter Amstutz

- Target version set to 2016-05-11 sprint

- Story points set to 1.0

#10 - 04/27/2016 08:21 PM - Peter Amstutz

- Story points changed from 1.0 to 2.0

#11 - 04/29/2016 12:58 PM - Peter Amstutz

Test case creating a collection with a 35 MiB manifest (represents about 25 TiB).

Master:

```
Begin post VmHWM: 334872 kB
End post VmHWM:  1770316 kB
23.95 s = .
```

With decode_www_form_component fix:

```
Begin post VmHWM: 335260 kB
End post VmHWM:   854340 kB
22.60 s = .
```

#12 - 04/29/2016 01:32 PM - Peter Amstutz

With Oj:

```
Begin post VmPeak: 2284180 kB
Begin post VmHWM:   335132 kB
End post VmPeak:   2793520 kB
End post VmHWM:     835044 kB
```

With Yajl:

```
Begin post VmPeak:  568084 kB
Begin post VmHWM:   371548 kB
```

```
End post VmPeak:   1113728 kB
End post VmHWM:     870880 kB
```

#13 - 04/29/2016 07:36 PM - Peter Amstutz

So it looks like this is a Ruby standard library bug:

```
Rack uses the standard library method URI.decode_www_form_component to process
    parameters.  This method first validates the string with a regular expression,
    and then decodes it using another regular expression.  The bug is in the
    validation; the regular expression that is used generates many backtracking
    points, which results in exponential memory growth when matching large strings.
    The fix is to tweak the validation regex to use "posessive" matching (?>)
    and (.*+) which eliminates backtracking.  The optimized regex requires minimal
    memory and is around 50% faster.
```

#14 - 04/29/2016 08:09 PM - Tom Clegg

8998-optimize-decode-www-form-component @ d59fa34

Nice find.

Looks like you're not the first -- stdlib 2.2 and 2.3 contain a simpler solution (see http://ruby-doc.org/stdlib-2.2.0/libdoc/uri/rdoc/URI.html#method-c-decode_www_form_component)...

```
def self.decode_www_form_component(str, enc=Encoding::UTF_8)
  raise ArgumentError, "invalid %-encoding (#{str})" if /%(?!\h\h)/ =~ str
  str.b.gsub(/\+|%\h\h/, TBLDECWWWCOMP_).force_encoding(enc)
end
```

Suggest wrapping this patch in something like "Gem::Version.new(RUBY_VERSION) < Gem::Version.new('2.2')". Also, putting it in middlewares/arvados_api_token.rb seems a little mysterious. Maybe something like initializers/fix_ruby21_decode.rb?

Indentation got weird somehow in the vmpeak func and the new test.

nit: test "test memory usage" can be just test "memory usage" ...

#15 - 05/02/2016 05:25 PM - Tom Clegg

From IRC discussion:

- Yes, Oj mmaps a lot of virtual memory
- Oj *uses* less real memory (the kind you can run out of) than Yajl
- Ignoring VM mapping, Oj is better than Yajl in both memory economy and speed, therefore we should just stick with Oj

The current theory is the "out of memory" bug here was caused solely by decode_www_form_component, and has nothing to do with the fact that Oj harmlessly maps a lot of memory.

#16 - 05/02/2016 08:33 PM - Tom Clegg

LGTM @ a829e7e

#17 - 05/03/2016 01:25 PM - Peter Amstutz

- Status changed from New to Resolved

- % Done changed from 0 to 100

Applied in changeset arvados|commit:8b097ea832516b3f5104c62a40a6d9cf4826232a.

Files

160311_arvOutOfMemBt.txt	11.1 KB	04/15/2016	Peter Grandi
ojram.rb	525 Bytes	04/15/2016	Brett Smith
yajlram.rb	525 Bytes	04/15/2016	Brett Smith