Arvados - Bug #3400

[Workbench] Rails client library should offer an "all pages" mode for queries that return pages of lists

07/29/2014 11:02 AM - Peter Amstutz

Status:	Resolved	**Start date:**	11/05/2014
Priority:	Normal	**Due date:**	
Assigned To:	Peter Amstutz	**% Done:**	100%
Category:	SDKs	**Estimated time:**	0.00 hour
Target version:	2014-11-19 sprint		

Description

Background: If a developer writes something like Link.where(...).each do { ... } and the API call results in more than the result limit (default 100), the code will only iterate over the first 100 results. This is error prone and leads to subtle bugs where code works as expected when there are less than 100 results but suddenly starts losing results when there are more than 100 results. The client libraries should clearly distinguish between cases where paging is desired vs iterating over the entire result set. This problem currently exists in all client SDKs: Python, Go, Ruby, Java, Perl, command line, and Rails (Workbench).

Current priority:

- Add and document "all pages" feature in Rails/Workbench client library
- Document the limitation in other libraries. If possible, show an example of how to do it with the offset parameter, like the $newtask_results code in source:sdk/cli/bin/crunch-job#L1056.

Implementation:

"Get all pages automatically" should be the default behavior, as in ActiveRecord. This change will involve fixing all existing cases where Workbench relies on the "one page" behavior.

- Model.limit(100) sets limit=100 in the API call, *and* fetches multiple pages in order to reach 100 results (or EOF) even if the API server imposes a maximum page size smaller than the limit given.
- each(), select(), and collect() fetch subsequent pages as needed to get all matching results (an each loop that breaks in the first iteration will involve just one API call).
- The code that retrieves results will finally get extricated from where().
- to_ary will have to fetch all pages before returning. Therefore, it should be avoided where possible.
- Accept limit(:page) to signify "one page, using the API server's default limit" (i.e., don't send a limit parameter, and don't fetch additional pages).
- Use limit(1), limit(:page), etc. in Workbench where appropriate (e.g., retrieving a page of results for infinite-scroll content)

Subtasks:

Task # 4418: Write tests	**Resolved**
Task # 4417: ArvadosResourceList returns exactly @limit items unless there are fewer th...	**Resolved**
Task # 4441: Review 3400-workbench-all-items	**Resolved**
Task # 4381: ArvadosResourceList returns all items unless @limit is non-nil	**Resolved**

Associated revisions

Revision 2def2025 - 10/23/2014 05:53 PM - Peter Amstutz

Increase limit for collections controller querying same portable_data_hash as
workaround to fix tests, refs #4058, but really needs a proper fix, refs #3400

Revision 45bfc110 - 11/12/2014 06:42 PM - Peter Amstutz

Merge branch '3400-workbench-all-items' closes #3400

History

#1 - 07/29/2014 11:23 AM - Peter Amstutz

- Subject changed from Workbench API client doesn't handle paging. to [SDK] API clients don't handle paging.

- Description updated

- Category set to SDKs

#2 - 07/31/2014 05:24 PM - Tom Clegg

- Target version set to Arvados Future Sprints

#3 - 09/19/2014 06:17 PM - Peter Amstutz

- Story points set to 1.0

#4 - 10/24/2014 04:43 PM - Tom Clegg

- Subject changed from [SDK] API clients don't handle paging. to [SDKs] Client libraries should offer an "all pages" mode for queries that return pages of lists

- Description updated

#5 - 10/24/2014 06:18 PM - Tom Clegg

- Subject changed from [SDKs] Client libraries should offer an "all pages" mode for queries that return pages of lists to [Workbench] Rails client library should offer an "all pages" mode for queries that return pages of lists

#6 - 10/24/2014 06:19 PM - Tom Clegg

- Description updated

#7 - 10/27/2014 02:25 PM - Tom Clegg

- Description updated

#8 - 10/29/2014 06:07 PM - Ward Vandewege

- Target version changed from Arvados Future Sprints to 2014-11-19 sprint

#9 - 10/29/2014 07:14 PM - Ward Vandewege

- Assigned To set to Peter Amstutz

#10 - 11/05/2014 09:28 PM - Peter Amstutz

- Status changed from New to In Progress

#11 - 11/07/2014 05:44 AM - Tom Clegg

Reviewing 2361a1b

Instead of assert a < 201, say assert_operator a, :<, 201 -- that way, when the assertion fails, the test suite will report something like "expected 500 < 201", which is more helpful than "expected false to be true".

I suspect the addition use_token :active in users_controller_test.rb covers up an interesting bug which should be documented and eventually fixed. Consider:

- ```
  [redacted].current[:token] = 'foo'
  collections = Collection.where({})
  Thread.current[:token] = nil
  puts collections.first
  ```

- With the existing code, I expect the API call will be made with a nil token. This could produce surprising/confusing results when switching tokens (like using reader tokens for sharing links, using an "anonymous" token to do some things even when a user is logged in).
- We could stash Thread.current[:arvados_api_token] in the ArvadosResourceList object when it's instantiated, and use it in the API calls it makes later.
- We could change the "current api client" code so it stashes a client object that has its own token, rather than just a token, in Thread.current. Then we'd stash a reference to the *client object* in the ArvadosResourceList object. But this sounds like too much scope creep for now.
- Perhaps for now it is enough to document this, so the next developer who runs up against it doesn't have to do so much hair-pulling when writing another test case with the same subtle feature as this one?

ArvadosResourceList#all used to reset the where() criteria (but not filters, whoops) and fetch one page. Now, it leaves filter/where criteria alone and pre-fetches all pages. What anyone would actually expect from Model.where({foo:'bar'}).limit(10).all is a bit of a mystery, and I wonder if it would be best to remove that method entirely, like this: 8336977 ...?

Might want to raise an exception if not @limit.nil? or @limit.is_a? Integer rather than just ignoring it.

I find it a bit hard to follow the exact meaning of items_to_get as it succumbs to various conditions. It seems to have a harmless but strange feature that it fails to anticipate the end of the results when @offset>0 and @limit>items_available, and wastes one empty-results API call as a result. I think it would be much clearer to remove that variable and its arbitrary maximum, and instead deal with the two "enough pages" conditions separately after the yield:
  ```

```
-    end while @fetch_multiple_pages and item_count < items_to_get
+      break if @limit.is_a? Integer and @limit <= item_count
+      break if items.respond_to? :items_available and items.items_available <= offset
+
+    end while @fetch_multiple_pages
```

I notice #each_page will retrieve/yield more than the expected number of rows if (for example) the server responds to limit=150 by returning 100 rows. I've demonstrated this with bf63e93. One way to minimize fallout from the server change is

- Merge that commit
- Make the test pass
- Skip the test
- Change server max to something bigger (1000?) so old workbenches can still use their large limit hacks
- Merge this branch
- When workbenches are all deployed, change server max to something less than 200 and un-skip the workbench test.

Alternatively, each_page could accept a max_page_size argument. That way the test wouldn't have to depend on the API server's max_page_size feature, or on having more than that max_page_size fixtures. I suppose this could be a useful feature for real callers too, although I can't think of a good example offhand.

I suspect Group.find(...).contents.each_page() will do the wrong thing. Suggest:

- Make #each_page a protected method. (That seems to be the intent here anyway.)
- In the future, make it possible for API methods other than *.index to make use of the paging feature. (We definitely shouldn't hold up this branch for that, though.)

The significance of #result_links, #result_offset, and #result_limit during #each_page (or after calling #results on a multi-page set) seems a bit obscure. result_links doesn't seem to be used anywhere, and the rest of result_* seem to be used only by _paging.html.erb. Should we discourage/prevent clients from using those methods when #each_page is in play? This could just mean setting @result_limit = nil etc. in #each_page, rather than setting them to "offset of last page retrieved". Example test:

- [redacted]'result_offset makes sense after each_page' do
    use_token :admin
    all = Collection.all
    all.results
    assert_equal 0, all.result_offset # Failure: Expected: 0 Actual: 200
    assert_equal all.results.size, all.result_items_available
  end

#12 - 11/07/2014 07:04 PM - Peter Amstutz

Tom Clegg wrote:

> Reviewing 2361a1b
>
> Instead of assert a < 201, say assert_operator a, :<, 201 -- that way, when the assertion fails, the test suite will report something like "expected 500 < 201", which is more helpful than "expected false to be true".

Fixed.

> - We could stash Thread.current[:arvados_api_token] in the ArvadosResourceList object when it's instantiated, and use it in the API calls it makes later.

This is the solution I went with.

> ArvadosResourceList#all used to reset the where() criteria (but not filters, whoops) and fetch one page. Now, it leaves filter/where criteria alone and pre-fetches all pages. What anyone would actually expect from Model.where({foo:'bar'}).limit(10).all is a bit of a mystery, and I wonder if it would be best to remove that method entirely, like this: 8336977 ...?

I agree, #all is not necessary, I merged your commit.

> Might want to raise an exception if not @limit.nil? or @limit.is_a? Integer rather than just ignoring it.

Added a check to the #limit setter.

> I find it a bit hard to follow the exact meaning of items_to_get as it succumbs to various conditions. It seems to have a harmless but strange feature that it fails to anticipate the end of the results when @offset>0 and @limit>items_available, and wastes one empty-results API call as a result. I think it would be much clearer to remove that variable and its arbitrary maximum, and instead deal with the two "enough pages" conditions separately after the yield:

> [...]

Fixed.

> I notice #each_page will retrieve/yield more than the expected number of rows if (for example) the server responds to limit=150 by returning 100 rows.

I handled this on the client side by adjusting the limit to be the remaining number of items desired on subsequent calls, which fixes the problem without tinkering with the server, unless there's another problem I'm missing.

- Make #each_page a protected method. (That seems to be the intent here anyway.)

Done.

> The significance of #result_links, #result_offset, and #result_limit during #each_page (or after calling #results on a multi-page set) seems a bit obscure. result_links doesn't seem to be used anywhere, and the rest of result_* seem to be used only by _paging.html.erb. Should we discourage/prevent clients from using those methods when #each_page is in play? This could just mean setting @result_limit = nil etc. in #each_page, rather than setting them to "offset of last page retrieved". Example test:
>
> - [...]

#result_offset and #result_limit are used so that the paging code renders based on the actual pages returned by the server and not just what the client asked for. I added fetch_multiple_pages(false) to the server paging code so that #result_offset and #result_limit don't get set more than once.

result_links is used by #links_for and #links_for is used all over the place for name links, so pulling that out is a bigger job that's out of the scope of this task (although we could stub out #links_for so that it always returns nothing.)

#13 - 11/11/2014 11:10 PM - Tom Clegg

Looking at 0f9bca4...

Peter Amstutz wrote:

> I handled this on the client side by adjusting the limit to be the remaining number of items desired on subsequent calls, which fixes the problem without tinkering with the server, unless there's another problem I'm missing.

It's good to have that particular bug fixed, but it's still not covered by a test. We should have at least one test that goes through the #each_page loop more than once. That (more so than fixing this bug) was why I was suggesting all that max-page-size stuff.

> > The significance of #result_links, #result_offset, and #result_limit during #each_page (or after calling #results on a multi-page set) seems a bit obscure. result_links doesn't seem to be used anywhere, and the rest of result_* seem to be used only by _paging.html.erb. Should we discourage/prevent clients from using those methods when #each_page is in play? This could just mean setting @result_limit = nil etc. in #each_page, rather than setting them to "offset of last page retrieved". Example test:
> >
> > - [...]
>
> #result_offset and #result_limit are used so that the paging code renders based on the actual pages returned by the server and not just what the client asked for. I added fetch_multiple_pages(false) to the server paging code so that #result_offset and #result_limit don't get set more than once.

Ah, right, the loop in each_page happens even when @fetch_multiple_pages==false, so we do need to set them there in order for paging to work. However, the problem is that when @fetch_multiple_pages==true, #result_* will return strange things. Ideally, nobody would be looking at them anyway, but for the sake of future developer sanity I think it would be worth having them return nil (rather than numbers that look fine for small result sets but behave strangely in production) if @fetch_multiple_pages==true. Items_available should be reasonable/correct as it stands, but result_limit and result_offset not so much.

> result_links is used by #links_for and #links_for is used all over the place for name links, so pulling that out is a bigger job that's out of the scope of this task (although we could stub out #links_for so that it always returns nothing.)

Returning nothing sounds good. Then, in the unlikely event it actually has any effect, at least it will happen in plain view on page 1.

Couple other little tweaks with the new code

I think this line is superfluous (it isn't used before getting set again at the top of the loop):

- [redacted]`:limit] = @limit if @limit`

This could be if limit (like earlier in the function) instead of if limit.is_a? Integer:

- `[redacted]break if @limit.is_a? Integer and item_count >= @limit`

Other changes look good, thanks.

#14 - 11/12/2014 03:59 PM - Peter Amstutz

- Status changed from In Progress to New

Tom Clegg wrote:

> Looking at 0f9bca4...
>
> Peter Amstutz wrote:
>
> > I handled this on the client side by adjusting the limit to be the remaining number of items desired on subsequent calls, which fixes the problem without tinkering with the server, unless there's another problem I'm missing.
>
> It's good to have that particular bug fixed, but it's still not covered by a test. We should have at least one test that goes through the #each_page loop more than once. That (more so than fixing this bug) was why I was suggesting all that max-page-size stuff.

Well, I'm fairly certain that the 'get all items by default' is going through the #each_page loop more than once since the default page is 100 items and it yields 201 items. The 'get single page of items' test does check that the default page size is less than 201 items.

I added a 'get limited items more than default page size' test, but right now the API server is happy to return more than 100 items in a single result if you ask for it (actually, I'm not sure if the API server has any limit on limit?) so that doesn't actually test the behavior of returning the correct number of items when limit > server max page size. How much more time do you want me to spend tinkering with this?

> Ah, right, the loop in each_page happens even when @fetch_multiple_pages==false, so we do need to set them there in order for paging to work. However, the problem is that when @fetch_multiple_pages==true, #result_* will return strange things. Ideally, nobody would be looking at them anyway, but for the sake of future developer sanity I think it would be worth having them return nil (rather than numbers that look fine for small result sets but behave strangely in production) if @fetch_multiple_pages==true. Items_available should be reasonable/correct as it stands, but result_limit and result_offset not so much.

Fixed.

> Returning nothing sounds good. Then, in the unlikely event it actually has any effect, at least it will happen in plain view on page 1.

Fixed.

> Couple other little tweaks with the new code
>
> I think this line is superfluous (it isn't used before getting set again at the top of the loop):
>
> - [...]

Fixed.

> This could be if limit (like earlier in the function) instead of if limit.is_a? Integer:
>
> - [...]

Fixed.

#15 - 11/12/2014 05:16 PM - Tom Clegg

At 91f0b18...

Peter Amstutz wrote:

> Well, I'm fairly certain that the 'get all items by default' is going through the #each_page loop more than once since the default page is 100 items and it yields 201 items. The 'get single page of items' test does check that the default page size is less than 201 items.

That's true, I overstated that. The case we don't test is going through the loop more than once *when @limit is set*.

> I added a 'get limited items more than default page size' test, but right now the API server is happy to return more than 100 items in a single result if you ask for it (actually, I'm not sure if the API server has any limit on limit?) so that doesn't actually test the behavior of returning the correct number of items when limit > server max page size. How much more time do you want me to spend tinkering with this?

I thought what I proposed in note-11 would address this, and should only take a minute: Merge/cherry-pick bf63e93 where I added a server max page size and a test on the workbench side, make sure the test passes, then raise the limit on the API server side and put "skip" on the test until we feel like lowering the API limit. (Or perhaps I'm missing the reason you didn't like this solution?)

Other changes look good, thanks.

(It looks like you also spotted one more case where "fetch all pages" isn't desirable. Did you have a systematic way of finding such things, or should we just be on the lookout for stray "fetch all pages of all collections" incidents after merging this?)

#16 - 11/12/2014 06:12 PM - Peter Amstutz

Tom Clegg wrote:

> > I added a 'get limited items more than default page size' test, but right now the API server is happy to return more than 100 items in a single result if you ask for it (actually, I'm not sure if the API server has any limit on limit?) so that doesn't actually test the behavior of returning the correct number of items when limit > server max page size. How much more time do you want me to spend tinkering with this?
>
> I thought what I proposed in note-11 would address this, and should only take a minute: Merge/cherry-pick bf63e93 where I added a server max page size and a test on the workbench side, make sure the test passes, then raise the limit on the API server side and put "skip" on the test until we feel like lowering the API limit. (Or perhaps I'm missing the reason you didn't like this solution?)

Done. I even broke ArvadosResourceList on purpose (and fixed it again) to make sure the new test was testing what it was supposed to. Set server MAX_LIMIT to 1000.

> Other changes look good, thanks.
>
> (It looks like you also spotted one more case where "fetch all pages" isn't desirable. Did you have a systematic way of finding such things, or should we just be on the lookout for stray "fetch all pages of all collections" incidents after merging this?)

I found that after making #result_offset nil when yielding multiple pages, so good idea to tweak the API there.

#17 - 11/12/2014 06:41 PM - Tom Clegg

LGTM, thanks!

#18 - 11/12/2014 06:45 PM - Anonymous

- Status changed from New to Resolved

- % Done changed from 75 to 100

Applied in changeset arvados|commit:45bfc1104dd30fb97a586de5ff96d6b739f7bb2b.