Description
A user reported that creating many projects at once resulted in extreme slowdown and eventually crashing the API server unless the create requests were throttled.

Creating a project (or any other group) also refreshes the permissions graph in the same transaction.

This makes the API call slow, but we intend/expect that other API calls can proceed while the update is in progress, so other (non-permission-modifying) API calls aren't expected to be slowed down.

The approach would be:
- Accept an async param to permission-changing calls that would delay the permission graph update for a server configurable number of seconds in the future.
- When updating the permission graph asynchronously, use the CONCURRENTLY parameter to avoid blocking SELECT while it's being updated. (PostgreSQL 9.4 feature, check API server's dependency requirements).
- Use status 202 (Accepted) on async call responses.
- Make a performance analysis comparison to make sure the issue is solved.
- Update documentation.

Subtasks:
Task # 14895: Review 13593-async-perm-graph-update

Related issues:
- Related to Arvados - Bug #14966: [API] Fix hanging test - suspect permission ...
- Blocks Arvados - Feature #13688: [API] optional asynchronous flag for create/...
# Updates

- New configuration knob: async_permissions_update_interval to hold the number of seconds a permission update will be deferred (default: 60)
- When passing async: true to a group's update or create call, a new thread is created that waits the number of seconds before doing a permission graph update.
- This deferred graph update is de-duplicated, so in case many async calls are made within the configure time, only one thread is created.
- The async calls return 202 Accepted on success
- API docs updated

The CONCURRENTLY parameter wasn't added because the permission view needs an additional index.

## Performance analysis

The test environment consisted on a newly created arvbox instance on a 4GB RAM Virtualbox VM.
The measurements were done by subsequently creating new groups (as a non-admin user) in batches of 1000 groups.

- First, group create calls using the current API server:
• 1st try: 0.158 secs/group
• 2nd try: 0.171 secs/group
• 3rd try: 0.191 secs/group
• 4th try: 0.218 secs/group
• 5th try: 0.225 secs/group (timings continue to rise as more groups are created)

  Then, group create calls using the new async parameter (without removing previous groups):
  • 1st try: 0.065 secs/group
  • 2nd try: 0.059 secs/group
  • 3rd try: 0.06 secs/group
  • 4th try: 0.063 secs/group

This difference (almost 4x) may appear not so significant, but we should take into account the fact that is an almost empty instance.

The script used to do the benchmarks:

```ruby
require 'arvados'

group_name = (0...8).map { (65 + rand(26)).chr }.join

begin
  puts "Starting creation of #{group_qty} groups..."

  start_t = Time.now.to_i

  (0...group_qty).each do |group_num|
    api.group.create({name: "#{group_name}#{group_num}"}, async: true)
  end

  end_t = Time.now.to_i

  puts "Finished! Avg group creation time: #{(end_t-start_t)/group_qty} secs"
rescue => e
  Rails.logger.error "Updating permission view: #{e} \n#{e.backtrace.join("\n\t")}"  
ensure
  ActiveRecord::Base.connection.close
end
```

#19 - 03/12/2019 02:27 PM - Peter Amstutz

I feel like there's a potential race condition here if AsyncRefreshPermissionView does not expire before the do_refresh_permission_view starts.

I can kind of convince myself that shouldn't happen, that AsyncRefreshPermissionView should always expire a few microseconds before sleep() returns, but with timing it is always hard to be sure (for example, what if the system clock is changed?). Maybe add Rails.cache.delete('AsyncRefreshPermissionView') between sleep(exp) and do_refresh_permission_view?

I feel like 60 seconds is a long time for the refresh period. A default of 15 or 20 seconds feels better. The ideal delay is probably be slightly longer than it takes to complete one permission table update, since the real goal here is to avoid delaying API calls and loading the database by stacking up a bunch of mostly redundant permission updates.

I noticed this:

```ruby
def self.invalidate_permissions_cache(timestamp=nil)
  if Rails.configuration.async_permissions_update
    timestamp = DbCurrentTime::db_current_time.to_i if timestamp.nil?
    connection.execute "NOTIFY invalidate_permissions_cache, '#{timestamp}'"
  else
    refresh_permission_view
  end
end
```

I checked and async_permissions_update and the invalidate_permissions_cache message are from #8886 which was a previous attempt at asynchronous permission updates that predates both the materialized permission view and (obviously) this branch. I think that code can be removed entirely so that invalidate_permissions_cache just calls refresh_permission_view.
It would be good to have a test that confirms the permission update actually happened. Suggest a test that:

1. sets permission update period to 5s
2. does an async update
3. tests permission (shouldn't be updated yet)
4. waits 5 seconds
5. tests permission again (should be updated now)

Updates at 7618631a4
Test run: https://ci.curoverse.com/job/developer-run-tests/1115/
- As suggested, changed from 60 to 20 seconds the default interval config
- Removed old code related to experimental async permissions update
- Enhanced test to not only check for status 202 Accepted, but also check that permission updates are actually being deferred.

This LGTM

Updates at 7618631a4
Test run: https://ci.curoverse.com/job/developer-run-tests/1115/
- As suggested, changed from 60 to 20 seconds the default interval config
- Removed old code related to experimental async permissions update
- Enhanced test to not only check for status 202 Accepted, but also check that permission updates are actually being deferred.

This LGTM