Arvados - Story #10816

[AP] [spike] Implement permission lookups as a recursive graph query in Postgres instead of computing in Ruby

01/04/2017 08:39 PM - Tom Clegg

Status:	Resolved	**Start date:**	01/04/2017
Priority:	Normal	**Due date:**	
Assigned To:	Tom Clegg	**% Done:**	100%
Category:	API	**Estimated time:**	0.00 hour
Target version:	2017-01-18 sprint		

Description

Subtasks:

Task # 10820: Determine minimum PostgreSQL version	**Resolved**
Task # 10822: Implement recursive graph view	**Resolved**
Task # 10823: Review 10816-postgres-permissions	**Resolved**

Related issues:

Related to Arvados - Bug #8787: [API] repositories/get_all_permissions method...	**Resolved**	**03/24/2016**
Related to Arvados - Feature #10267: Permission-monitoring service keeps a fl...	**Rejected**	**10/12/2016**
Related to Arvados - Story #12032: [API] Allow projects to be deleted (ie pla...	**Resolved**	**08/08/2017**

Associated revisions

Revision 532be5dc - 01/11/2017 10:06 PM - Tom Clegg

Merge branch '10816-postgres-permissions'

refs #10816

History

#1 - 01/04/2017 08:39 PM - Tom Clegg

- Target version set to 2017-01-18 sprint

- Story points set to 2.0

#2 - 01/06/2017 08:02 PM - Tom Clegg

Findings:

- Postgres can do permission lookups on the fly using a single (large) recursive query. However, this is too slow: >500 ms† to check whether a single collection is readable.
- Building a flattened permission table (user_uuid, object_uuid, access_level) takes about two seconds, and makes permission lookups faster (<50ms to check a single object, <500ms to list all readable collections)
- Indexing the flattened permission table takes about 12 seconds using "create index concurrently", but permission lookups are much faster (<1ms to check a single object) afterwards.

† Times given are for a production system with 170K collections, 28K groups, 500 permission links.

Proposed approach:

- Use a flattened permission table to do permission lookups.
- Rebuild the table (build new and rename into place) whenever a group's owner_uuid changes or a permission link is created/updated/deleted.
- Return from the "write" operation as soon as the table is rebuilt.
- Rebuild the index in a background thread after returning. (While this is happening, some queries will use the unindexed version.)

Proposed implementation plan (naïve):

- In API server, rebuild the flattened permission table at startup and wherever we currently run "invalidate permission cache".
- Start a background thread to index the resulting table.

Proposed implementation plan (race-safe):

- Use database synchronization facilities to avoid re-building and re-indexing the permission table multiple times, or replacing new data with old data during races.

- In API server:
  - During each commit that requires permissions to be rebuilt, notify a PostgreSQL "perm_rebuild_needed" channel with a random string.
  - After the commit, wait for the same random string to appear on the "perm_rebuild_done" channel.
- In permission updater:
  - Lock the "perm_rebuild_worker" table (this ensures there is only one permission updater running at a time).
  - Let pending_rebuilds = [ ]
  - Start listening to the "perm_rebuild_needed" channel.
  - Rebuild the permission table.
  - Send each item in pending_rebuilds to the "perm_rebuild_done" channel.
  - If the perm_rebuild_needed channel has nothing new yet, start re-indexing the new permission table in a separate thread while waiting for the next rebuild.
  - Wait for the next notification on the "perm_rebuild_needed" channel.
  - Let pending_rebuilds = payloads from all notifications that are ready (possibly more than one).
  - Repeat from "rebuild the permission table".

#3 - 01/06/2017 08:25 PM - Tom Clegg

Build a flattened permission cache (~1.5s):

```
START TRANSACTION;
DROP TABLE IF EXISTS permission_cache;
CREATE TABLE permission_cache_new AS
WITH RECURSIVE
perm_value (name, val) AS (
     VALUES
     ('can_read', 0),
     ('can_login', 1),
     ('can_write', 2),
     ('can_manage', 3)
     ),
perm_edges (tail_uuid, head_uuid, val, follow) AS (
       SELECT links.tail_uuid, links.head_uuid,
              pv.val,
              (links.name = 'can_manage' OR groups.uuid IS NOT NULL) AS follow
              FROM links
              LEFT JOIN groups ON groups.uuid = links.head_uuid
              LEFT JOIN perm_value pv ON pv.name = links.name
              WHERE links.link_class = 'permission'
       UNION ALL
       SELECT owner_uuid, uuid, 999, true FROM groups
       ),
perm (val, follow, user_uuid, target_uuid) AS (
     SELECT 999, true, users.uuid, users.uuid FROM users
     UNION
     SELECT LEAST(perm.val, edges.val) AS val,
            edges.follow               AS follow,
            perm.user_uuid             AS user_uuid,
            edges.head_uuid            AS target_uuid
            FROM perm
            INNER JOIN perm_edges edges
            ON edges.tail_uuid = perm.target_uuid
            WHERE perm.follow
)
SELECT * FROM perm;
ALTER TABLE permission_cache_new RENAME TO permission_cache;
COMMIT;
```

Index the permission cache (~12s):

```
CREATE INDEX CONCURRENTLY permission_cache_idx ON permission_cache USING btree (user_uuid, target_uuid);
```

Use the permission cache to check read permission on a single item:

```
SELECT max(collections.uuid)
       FROM collections
       INNER JOIN permission_cache pc
       ON pc.target_uuid IN (collections.uuid, collections.owner_uuid)
       WHERE pc.user_uuid = 'su92l-tpzed-e3x3046g0tzruhk'
       AND collections.uuid = 'su92l-4zz18-zzzjtierdgrvw70'
       ;
```

Use the permission cache to count writable objects:

```
SELECT max(collections.uuid)
       FROM collections
```

```
      INNER JOIN permission_cache pc
      ON pc.target_uuid IN (collections.uuid, collections.owner_uuid)
      AND pc.val >= 2
      WHERE pc.user_uuid = 'su92l-tpzed-e3x3046g0tzruhk'
      ;
```

Approximate times

index ready?	count readable	count writable	test single object for readable or writable
no	500ms	260ms	50ms
yes	500ms	260ms	2ms

#4 - 01/09/2017 04:46 PM - Tom Clegg

- Status changed from New to In Progress

#5 - 01/09/2017 06:34 PM - Tom Clegg

10816-postgres-permissions @ f301a70fdcfda9872965835b26d1400a53d584a1

I used a temporary view, created in each session on first use (instead of a database migration) so we can fine-tune the view more easily and safely in future. We could switch to storing it permanently in the database, but the view is really just shorthand for a verbose query, and in that sense it seems better to keep it in the application instead of the database.

#6 - 01/11/2017 07:07 PM - Radhika Chippada

@ f301a70f

- Do we want to rescue from any errors during create_permission_view.sql or 'ROLLBACK TO SAVEPOINT' ?
- Wondering if 'ROLLBACK SAVEPOINT' should be invoked when the rescue block is executed as well?
- Can perms_for_val be created once and reused rather than creating for each invocation of calculate_group_permissions, (may be a static)?
- This is the first time I came across with exec_query usage and it took me a while to understand how [[nil, uuid]] works in the exec_query statement. May be a link to exec_query documentation and / or comment explaining nil is column and uuid is the value in the binds usage might be good for code maintenance

#7 - 01/11/2017 09:25 PM - Tom Clegg

Radhika Chippada wrote:

> - Do we want to rescue from any errors during create_permission_view.sql or 'ROLLBACK TO SAVEPOINT' ?

I don't think so... is there an error condition you have in mind that we could recover from gracefully?

> - Wondering if 'ROLLBACK SAVEPOINT' should be invoked when the rescue block is executed as well?

Ah, if you mean RELEASE, yes -- I was thinking ROLLBACK also released, but it doesn't. Changed the "else" to an "ensure" so RELEASE happens after rollback as well.

> - Can perms_for_val be created once and reused rather than creating for each invocation of calculate_group_permissions, (may be a static)?

Yes, moved this to a class constant.

> - This is the first time I came across with exec_query usage and it took me a while to understand how [[nil, uuid]] works in the exec_query statement. May be a link to exec_query documentation and / or comment explaining nil is column and uuid is the value in the binds usage might be good for code maintenance

Yeah, information about exec_query and binds was relatively hard to find. Updated:

```
    conn.exec_query('SELECT target_owner_uuid, max(perm_level)
                     FROM permission_view
                     WHERE user_uuid = $1
                     AND target_owner_uuid IS NOT NULL
                     GROUP BY target_owner_uuid',
                     # "name" arg is a query label that appears in logs:
                     "group_permissions for #{uuid}",
                     # "binds" arg is an array of [col_id, value] for '$1' vars:
```

```
                    [[nil, uuid]],
                    ).rows.each do |group_uuid, max_p_val|
```

Now at fe1b0b43931dcefbf9308dc7b0a3639a4410ca53

#8 - 01/11/2017 10:03 PM - Radhika Chippada

Thanks for the update and clarifications. LGTM

#9 - 01/18/2017 06:55 PM - Tom Clegg

- Status changed from In Progress to Resolved