Redmine - Patch #13301
Performance: avoid querying all memberships in User#roles_for_project

2013-02-27 01:07 - Jean-Baptiste Barth

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Jean-Baptiste Barth	**% Done:**	0%
Category:	Permissions and roles	**Estimated time:**	0.00 hour
Target version:	2.3.0		

Description

Already discussed a bit in #11827

Problem

On my instance at work the average staff user has 150 private projects. User#roles_for_project retrieves all memberships then selects only the one needed for a specific project. This is not problematic when done for current user (for whom memberships need to be retrieved for the project jump box) but it can lead to substantial performance loss when applied to many other users, for instance when generating the list of recipients for email notifications.

Proposal

Replace app/models/user.rb line 428 :

```
membership = memberships.detect {|m| m.project_id == project.id}
```

with :

```
membership = memberships.where(:project_id => project.id).first
```

Results

On a very slow machine I noticed a 70% gain when generating this list for a specific issue (13s=>4s). On a decent server you will probably see a smaller gap. Some notes :

- some 50 users on project or so, private project
- takes ~50ms to retrieve all memberships per user, + object instantiation
- take less than 5ms to retrieve only needed memberships per user
- a global count shows 50 memberships for this project, 7000+ memberships in the worst case retrieved without the patch

Of course the whole test suite is green before and after the modification (trunk, redmine 1.9.3 on sqlite without scm tests, but no reason to doubt it'd be different with an other setup).

What's next?

First, let me know what you think. I think this patch won't change anything for little redmine installations but can be great for redmine instances with hundreds of projects and dozens of users per project.

Second, performance is hard to track in a complex app such as Redmine. This method is central for determining object visibility, and a simple test showed it's called multiple times for different projects on the same User instance in a few functional/integration test cases, so it's probably the case in real world usage. I'll patch manually my own instance at work and tell you in a few days the benefits and if it doesn't degrade performance dramatically elsewhere.

Third, object visibility code and permissions may deserve a *bigger rewrite* later, this change is just a quick win and a way to begin the

discussion here if you want.

Related issues:

Related to Redmine - Patch # 11827: Avoid retrieving memberships for projects...	**Closed**	

Associated revisions

Revision 11508 - 2013-03-01 11:24 - Jean-Philippe Lang

Performance: avoid querying all memberships in User#roles_for_project (#13301).

Revision 11516 - 2013-03-01 17:22 - Jean-Philippe Lang

Merged r11508 from trunk (#13301).

Revision 11605 - 2013-03-12 18:08 - Jean-Philippe Lang

Prevent one query per User#member_of? call after r11508 (#13301).

Revision 11606 - 2013-03-12 18:09 - Jean-Philippe Lang

Merged r11605 from trunk (#13301).

History

#1 - 2013-02-27 11:53 - Etienne Massip

- Target version set to Candidate for next major release

Looks fine and very useful to me, can't see any drawback neither in functionality, performance nor DB stress overhead.

#2 - 2013-02-27 12:58 - Jean-Baptiste Barth

On my production server:

- before the patch, IssuesController#create average response time was 5.8s
- after applying the patch, average response time is 1.6s, yay :)
- nearly the same for IssuesController#update

For now no problem on other actions.

#3 - 2013-02-27 14:33 - Etienne Massip

Jean-Baptiste Barth wrote:

> *On my production server: - before the patch, IssuesController#create average response time was 5.8s*
> *- after applying the patch, average response time is 1.6s, yay :)*
> *- nearly the same for IssuesController#update*
>
> *For now no problem on other actions.*

FWIW: - if you were sending asynchronous notifications you would'nt notice the bloat, would you?

- 1.6s is still slow :p

#4 - 2013-02-27 22:45 - Jean-Baptiste Barth

Etienne Massip wrote:

> *FWIW: - if you were sending asynchronous notifications you would'nt notice the bloat, would you?*

I already use async_smtp settings. My SMTP relay is near me in the datacenter, pretty fast, it doesn't change much anyway.

> - *1.6s is still slow :p*

Yep I know :/ There are many factors (mainly related to our redmine usage which is not the most "mainstream" one). I don't have the fastest stack too but there are good reasons behind (passenger, ruby 1.9, standard postgres on debian, etc.). Still digging into quick wins like this one ;-)

#5 - 2013-02-28 09:04 - Etienne Massip

Jean-Baptiste Barth wrote:

> *Etienne Massip wrote:*
> > *FWIW: - if you were sending asynchronous notifications you would'nt notice the bloat, would you?*
>
> *I already use async_smtp settings. My SMTP relay is near me in the datacenter, pretty fast, it doesn't change much anyway.*

True, my remark was stupid since the recipient list is created sync.

Rails 4 adds a pooled async mailer where the mail composition is async too. If we .dup every model object involved in the mail creation and pass them through the async mailer task pool it should work but I don't know how much pricy switching to this behavior can be.

#6 - 2013-02-28 10:14 - Jean-Baptiste Barth

I'm currently in the process of rewriting the whole notification process in a plugin so that I can add a lot more flexibility, exceptions, etc. Here's the plugin repo, I can show you some screenshots of the admin interface later if you want.

The only real problem to do everything asynchronously is the dirty attributes used in Mailer. Passing <Model>#id and a flat <Model>#changes hash everywhere is good enough I think. For the sake of simplicity I'll rely on the sucker_punch gem (a wrapper around Celluloid) for asynchronicity.

#7 - 2013-02-28 19:57 - Jean-Philippe Lang

It generates lots of activerecord queries for non admin users. They hit the AR cache but still, I'd prefer doing something like this:

```
Index: app/models/user.rb
===================================================================
--- app/models/user.rb    (revision 11497)
+++ app/models/user.rb    (working copy)
@@ -132,6 +132,7 @@
   def reload(*args)
```

```
     @name = nil
     @projects_by_role = nil
+    @membership_by_project_id = nil
     base_reload(*args)
   end

@@ -419,6 +420,12 @@
     !logged?
   end

+  def membership_by_project_id
+    @membership_by_project_id ||= Hash.new {|h, project_id|
+      h[project_id] = memberships.where(:project_id => project_id).first
+    }
+  end
+
   # Return user's roles for project
   def roles_for_project(project)
     roles = []
@@ -426,7 +433,7 @@
     return roles if project.nil? || project.archived?
     if logged?
       # Find project membership
-      membership = memberships.detect {|m| m.project_id == project.id}
+      membership = membership_by_project_id[project.id]
       if membership
         roles = membership.roles
       else
```

#8 - 2013-02-28 20:59 - Etienne Massip

You're basically substituting a hash cache to the AR cache, I'm curious to know the performance gain for Jean-Baptiste but if the AR cache is good enough it should not be much?

#9 - 2013-02-28 22:18 - Jean-Philippe Lang

With AR cache on:

```
Benchmark.ms { 200.times {user.memberships.where(:project_id => 1).first }}
=> 452.026 ms

Benchmark.ms { 200.times {user.membership_by_project_id[1] }}
=> 2.0 ms
```

Any questions?

#10 - 2013-02-28 22:24 - Etienne Massip

No.

#11 - 2013-03-01 07:03 - Jean-Baptiste Barth

Nice catch, it's even better on my test platform with your version Jean-Philippe (+30-40% faster).

#12 - 2013-03-01 11:28 - Jean-Philippe Lang

- Status changed from New to Closed

- Target version changed from Candidate for next major release to 2.3.0

Fix committed in r11508. I've made a quick test with 50 users that all belong to 200 projects. Creating an issue with a notification to the 50 users was ~2500ms and is now ~400ms.