Redmine - Defect #4547

git: Very high CPU usage for a long time with large repository

2010-01-11 14:47 - Philip Hofstetter

Status:	Closed	**Start date:**	2010-01-11
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:	SCM	**Estimated time:**	0.00 hour
Target version:	0.9.3		
Resolution:	Fixed	**Affected version:**	0.9.0

Description

This is a redmine-instance updated from 0.7.x to 0.8.x and now to 0.9.0. It's connected to a git repository which has about 9000 commits over various branches.

Whenever I'm hitting the repistory-tab and the repository has changed, redmine now is shelling out to git with the command line

```
sh -c git --git-dir '/var/git/repositories/xxx.git/' log --find-copies-harder --raw --date=iso --pretty=fuller --all
```

which - with a repository this size - causes git to work at 100% cpu usage for ~ 5 minutes.

It feels like a shortcoming in redmine to search through the whole repository each time something changes.

Or is this an artefact of updating from 0.8?

Related issues:

Related to Redmine - Defect #4773: Redmine+Git+PostgresSQL 8.4 fails with lin...	**Closed**	**2010-02-09**
Related to Redmine - Defect #8564: Git: `git mv file file2` shows up as delet...	**Closed**	**2011-06-08**
Related to Redmine - Defect #8857: Git: Too long in fetching repositories aft...	**Closed**	**2011-07-20**
Related to Redmine - Feature #1406: Browse through Git branches	**Closed**	**2008-06-09**
Has duplicate Redmine - Defect #4876: Problem with fetch git-changesets after...	**Closed**	**2010-02-18**

Associated revisions

Revision 3394 - 2010-02-07 16:17 - Jean-Philippe Lang

Do not parse the entire git log to fetch new commits (takes several minutes for a few thousands commits), but only 1 week before the last known commit (#4547, #4716).

Revision 3468 - 2010-02-21 15:37 - Jean-Philippe Lang

Removes --find-copies-harder git option to retrieve commits which was way to slow (#4547).

Revision 5644 - 2011-05-05 01:38 - Toshi MARUYAMA

scm: git: add comments of revision order in fetch_changesets().

Related issues.
#5357, #6013, #7146, #4773, #4547, #1406, #3449, #3567.

Revision 8156 - 2011-12-10 12:55 - Toshi MARUYAMA

scm: git: remove unused --since option (#4547, #4716, #7146, #6013)

History

#1 - 2010-01-11 15:39 - Philip Hofstetter

I had a look at this.

In app/models/repository/git.rb, there's this comment:

```
With SCM's that have a sequential commit numbering, redmine is able to be
clever and only fetch changesets going forward from the most recent one
it knows about.  However, with git, you never know if people have merged
commits into the middle of the repository history, so we always have to
```

`parse the entire log.`

and while I agree about the general points of this, I think we'll need some additional intelligence here if redmine's git support was to be useful for larger repositories.

I was thinking. If we have revision deadbeef as our last revision (for any particular branch) in the database and we look at the repository and the HEAD points to cafebabe, so obviously, something has changed. Now if we see that deadbeef is somewhere in the list of parents of cafebabe, then, it's save to just

`git log deadbeef...`

In this case, the full scan of the repository could be avoided.

Because rebases should not happen too often on public repositories (which is what redmine is usually used for), this optimization seems to be valid in most of the cases.

Agreed?

If so, I'll try and have a look into this (no promises though - I'm still a rails noob waiting for a good opportunity to get my feet wet).

#2 - 2010-01-11 20:30 - Philip Hofstetter

Philip Hofstetter wrote:

> I was thinking. If we have revision deadbeef as our last revision (for any particular branch) in the database and we look at the repository and the HEAD points to cafebabe, so obviously, something has changed. Now if we see that deadbeef is somewhere in the list of parents of cafebabe, then, it's save to just

The plan would be to git-rev-list HEAD and search for the latest revision that redmine has stored.

Of course, this gets quite a lot more difficult once branches come into play - especially considering that redmine does not track any branching information in its changeset tables. Especially when one wants to import revisions in bulk as it is redmine's current design. If it was possible to fetch revisions as the need arises, I could come up with a solution, but if the current possibility of having a cronjob fetching information is to remain in-place, there really is no way around this but to fetch all revisions.

On the other hand: Why is redmine doing `log --find-copies-harder` instead of just "rev-list --all" and importing all revisions (which is incredibly fast even with larger repositories)?

While investigating this, I came across another issue with just storing changesets and no heads in redmine: The "view all revisions" link (/repository/revisions) is broken and intermingles branches. I'll report a second issue on that.

Bug back to this problem: To fix this, I see a few options:

1. Use git-rev-list --all and dumbly import everything (why is this not done?)
2. Extend redmine's schema to track heads (and parents - you can't use the date to determine chronological order). This is ugly though as not all SCM systems know about different heads. Also it would replicate a lot of git's functionality.
3. Don't use the changesets table at all and shell out for all operations
4. Keep the current system, but import revisions as we determine them to be missing. This won't solve the "all revisions" issue but at least would remove the slowdowns. Make the bulk update use the old behavior.

Ideas? Comments?

#3 - 2010-01-11 20:35 - Philip Hofstetter

Philip Hofstetter wrote:

> While investigating this, I came across another issue with just storing changesets and no heads in redmine: The "view all revisions" link (/repository/revisions) is broken and intermingles branches. I'll report a second issue on that.

issue reported as #4549

#4 - 2010-01-11 20:58 - Felix Schäfer

While I haven't looked at the code, some points here:

Philip Hofstetter wrote:

1. Use git-rev-list --all and dumbly import everything (why is this not done?)
2. Extend redmine's schema to track heads (and parents - you can't use the date to determine chronological order). This is ugly though as not all SCM systems know about different heads. Also it would replicate a lot of git's functionality.
3. Don't use the changesets table at all and shell out for all operations

Wouldn't be a problem for git because IIRC it requires a local copy, but everything going over the wire (think e.g. svn with http or ssh) would be horridly slow. Making it a two-tiered solution where some SCM would get cached in the db, others not is also a lot of trouble. Lastly, I think you can search over the commit log, not practically feasible if not in the db. I wouldn't count on this one.

1. Keep the current system, but import revisions as we determine them to be missing. This won't solve the "all revisions" issue but at least would remove the slowdowns. Make the bulk update use the old behavior.

Again, could work for "local" copies (i.e. stuff that is on the same server and gets notified of changes itself, which then can notify redmine about the changes), but how do you notify redmine about changes in an off-server repo? I myself use a post-commit hook on our svn repositories to notify redmine about a change in a repo, and then make redmine update only this repo, which takes away the need for a cronjob and the need to walk over all repositories.

You could make this work for distant repos for which you can implement some sort of post-commit hook too by making some action available in redmine, e.g. http://your.server/projects/yourproject/repository/update_me, but again, you have to have some level of control over the repo. For repos you only have read access or a very limited access, you won't be able to use anything but the cron or the automatic method on display of the repo tab.

I'm not a git "repo" specialist in any way, neither do I really know what redmine does internally, but maybe a solution here would be to make a post-commit hook for git that would say "hey, I'm a new commit $SOME_HASH and my parent is $SOME_OTHER_HASH", that way redmine could look up its table and see if it already has $SOME_OTHER_HASH, in which case it would only need update its internal information about $SOME_HASH. This would only work if the copy on your redmine server is the "primary" repo, or if it's kept in sync good enough to be of some use though (don't know how it's done currently, so that may be an already existing problem).

Anyhow, maybe this would be a great improvement to redmine to work only with post-commit (or whatever) scripts for local repositories, which would tell redmine "hey, the repo for $SOME_PROJECT has received a new commit $SOME_COMMIT with parent $SOME_PARENT", so that redmine could only the info needed, and leave the bulk cronjob updater as it is. As far as I can see it, most people use it with local repos anyway, so a tightening of the coupling wouldn't be anything bad here.

And I've just lost my motivation to proof-read this, so I'll just hope it makes enough sense :-) </ramblings>

#5 - 2010-01-11 21:12 - Philip Hofstetter

Felix Schäfer wrote:

> Wouldn't be a problem for git because IIRC it requires a local copy, but everything going over the wire (think e.g. svn with http or ssh) would be horridly slow. Making it a two-tiered solution where some SCM would get cached in the db, others not is also a lot of trouble. Lastly, I think you can search over the commit log, not practically feasible if not in the db. I wouldn't count on this one.

well... git log --grep is extremely fast. Considering that without any fulltext indexing going on, the database would have to do a full table scan anyways (if searching without start-anchor) I don't see this as a problem.

> 1. Keep the current system, but import revisions as we determine them to be missing. This won't solve the "all revisions" issue but at least would remove the slowdowns. Make the bulk update use the old behavior.

> Again, could work for "local" copies (i.e. stuff that is on the same server and gets notified of changes itself, which then can notify redmine about the changes), but how do you notify redmine about changes in an off-server repo? I myself use a post-commit hook on our svn repositories to notify redmine about a change in a repo, and then make redmine update only this repo, which takes away the need for a cronjob and the need to walk over all repositories.

I see the point about the remote repositories. But as it stands now, even on a local repository, I have to wait up to 5 minutes whenever the repository changes at all. That's even less useful than fetching revisions on-demand.

> I'm not a git "repo" specialist in any way, neither do I really know what redmine does internally, but maybe a solution here would be to make a post-commit hook for git that would say "hey, I'm a new commit $SOME_HASH and my parent is $SOME_OTHER_HASH", that way redmine could look up its table and see if it already has $SOME_OTHER_HASH, in which case it would only need update its internal information about $SOME_HASH. This would only work if the copy on your redmine server is the "primary" repo, or if it's kept in sync good enough to be of some use though (don't know how it's done currently, so that may be an already existing problem).

Also, it would hardly work with remote repositories either. Unless you add some additional configuration settings where you tell the git adapter where to find the status files you have written. Also, see my point below about redmine not tracking parent/child relations.

> Anyhow, maybe this would be a great improvement to redmine to work only with post-commit (or whatever) scripts for local repositories, which would tell redmine "hey, the repo for $SOME_PROJECT has received a new commit $SOME_COMMIT with parent $SOME_PARENT", so that redmine could only the info needed, and leave the bulk cronjob updater as it is. As far as I can see it, most people use it with local repos anyway, so a tightening of the coupling wouldn't be anything bad here.

Yeah. Especially considering that cloning a git repository is simple. In fact I see no advantage in having a remote git repository and trying to fetch that remotely over just cloning that remote repository and looking at the local data (which is the same as the remote one by definition).

But one of the problems with the $SOME_PARENT approach is the fact that redmine does not in fact track parent/child relations of commits in its local store, so it would be quite interesting to find out which revisions to delete from the local store if a new commit came in which is a child of $PARENT which is already in the cache. Is this another branch? Is this replacing an existing revision?

#6 - 2010-01-12 09:09 - Philip Hofstetter

ok. After sleeping over this issue, here's what I think I'm going to do:

1. the local (see below) git repository is always authoritative and will be queried for all operations dealing with fetching changesets.
2. Each commit we come across that we don't already have in the changesets table is imported. This is because there are many places where redmine is using revision.changesets and various ActiveRecord methods on that, so emulating all this will be very error-prone.
3. fetch_changesets will also clean this "cache" and regularely remove revisions in the changesets table but not any longer in git (git-rev-list --all)
4. Remote support is provided by cloning the remote repository. fetch_changesets does a "git fetch origin" after which point 1) and 2) will come into place.

This solution not only solves the inconsitencies around, it would also give us support for remote git repositories which currently isn't available. At the cost of disk space of course.

IMHO, this is the only viable solution without extending redmine's changesets schema to track parent/child relationships between revisions (which still would be a pain to deal with considering we are using sql - to find the n-th parent, we'd have to issue n queries).

I'll work on a couple of patches to do above things.

#7 - 2010-01-12 20:10 - Adam Soltys

Hi Philip, I appreciate you looking at this. I'm responsible for the current solution which came in under #1406. My knowledge of git is fairly limited so any improvements you can make to the shell commands to improve performance are more than welcome.

A very quick fix that should speed things up quite a bit with the current solution would be to simply remove the --find-copies-harder flag. I originally wanted to have a redmine setting to enable/disable that flag but never got around to it. Removing it should be fine as long as you don't mind redmine reporting file copies as separate "delete" and "add" operations.

Anyways, sounds like you know what you're talking about more than I do when it comes to git so I'm interested to see what solution you come up with.

#8 - 2010-01-12 22:26 - Philip Hofstetter

Hi Adam,

Adam Soltys wrote:

> Hi Philip, I appreciate you looking at this. I'm responsible for the current solution which came in under #1406. My knowledge of git is fairly limited so any improvements you can make to the shell commands to improve performance are more than welcome.

I'm trying my best. While I have quite some git experience under my belt, my ruby skills are lacking. But hey - I feel much more productive with my first ruby-steps than I did with any other language before that, so there yet is hope.

> A very quick fix that should speed things up quite a bit with the current solution would be to simply remove the --find-copies-harder flag. I originally wanted to have a redmine setting to enable/disable that flag but never got around to it. Removing it should be fine as long as you don't mind redmine reporting file copies as separate "delete" and "add" operations.

Yeah. That takes care of 99.9999% (rough estimate **grin**) of time spent in git. The ruby and database part of course still takes ages, so I'm determined to go on with the plan I outlined earlier.

On http://github.com/pilif/redmine/tree/git-fix you'll see my progress on the matter. Note that while I should not happen, I reserve the right to rebase and force-push the branch as I make changes (what you see has already gone through "add -p" and some rebasing action.

Because I'm very noobish in matters of Ruby, I'm probably creating too small commits and spending too much effort writing the commit messages, but I want people to understand and follow my reasoning.

The branch is nowhere near complete, but I wanted to inform watchers of this bug of my progress. So far, I stopped importing anything in fetch_changesets and I'm lazily storing revisions as I come across them in Repository::Git#entries and Repository::Git#latest_changesets.

Please take a couple of minutes to have a look at my ruby - as I said, I'm a complete and utter newbie (I DO really enjoy this work though)

#9 - 2010-01-13 14:09 - Dominik Wild

Well, a repository configured in redmine usually is shared. And for shared repositories it's not really a common use case to delete revisions or rewrite the history. Hence I think we could just rely on the reverse chronological order of git-rev-list or git-log and import the missing revisions. Or am I missing something?

#10 - 2010-01-13 16:03 - Philip Hofstetter

Dominik Wild wrote:

> Well, a repository configured in redmine usually is shared. And for shared repositories it's not really a common use case to delete revisions or rewrite the history. Hence I think we could just rely on the reverse chronological order of git-rev-list or git-log and import the missing revisions. Or am I missing something?

Well... and /what if/ the repository has been rebased? Wrong data feels like a very wrong thing to have.

On the other hand, you gave me an idea: While I still like to pursue my original approach (in order to fix the "view all revisions" function), I could easily just delete from changesets what is not in rev-list --all and add to changesets what is in rev-list --all but not in changesets, giving us the bulk-import functionality back.

#11 - 2010-01-13 16:12 - Dominik Wild

> Well... and /what if/ the repository has been rebased? Wrong data feels like a very wrong thing to have

Rebasing (I mentioned it as rewriting in the previous post) a shared repository is a bad idea. This will cause problems for anyone who already has a copy of the branch.

#12 - 2010-01-13 17:56 - Dominik Wild

Oh, I think I overlooked the emphasising of "what if". In that case we indeed have a problem.

> I could easily just delete from changesets what is not in rev-list --all and add to changesets what is in rev-list --all but not in changesets

Isn't that what fetch_changesets is doing now?

#13 - 2010-01-13 21:59 - Philip Hofstetter

Dominik Wild wrote:

> Oh, I think I overlooked the emphasising of "what if". In that case we indeed have a problem.

nope. not really. fetch_changesets will do the right thing, because when a rebased branch gets pushed, our git log --all will not list the now unreachable commits which will then get removed.

By the way: Having force-pushed rebases isn't such an uncommon thing. The repository I'm working with has some really volatile throwaway branches that get force-pushed and to fix a couple of mistakes, I have even pushed a rebased master - of course after informing the team and picking a convenient time to do so.

> > I could easily just delete from changesets what is not in rev-list --all and add to changesets what is in rev-list --all but not in changesets
>
> Isn't that what fetch_changesets is doing now?

yes. the problem was just with the --find-copies-harder.

I just sent a pull request for a small patch that removes that option which solves this bug (in my test repo, without the flag, git takes less than a second, whereas with the flag it's a matter of five minutes).

I will still investiate into my earlier solution of using the changesets table as a simple cache (saving huge amounts of memory - right now, fetch_changesets will read all commits plus messages into memory) plus the ability to have support for remote repositories (using a local clone), but that's work for past 0.9 and past the scope of this bug.

#14 - 2010-01-13 23:31 - Dominik Wild

> > Oh, I think I overlooked the emphasising of "what if". In that case we indeed have a problem.
>
> nope. not really. fetch_changesets will do the right thing, because when a rebased branch gets pushed, our git log --all will not list the now unreachable commits which will then get removed.

Yes of course, I was referring to the "delta" fetching idea. Because assuming there are no rewrites it would be feasible to just get the last imported revision of the database and fetch the revisions starting from there.

> By the way: Having force-pushed rebases isn't such an uncommon thing. The repository I'm working with has some really volatile throwaway branches that get force-pushed and to fix a couple of mistakes, I have even pushed a rebased master - of course after informing the team and

> picking a convenient time to do so.

But it's nothing you do in a daily workflow. And for the daily workflow the "delta" mechanism would do the job quite okay.

> I will still investiate into my earlier solution of using the changesets table as a simple cache (saving huge amounts of memory - right now, fetch_changesets will read all commits plus messages into memory) plus the ability to have support for remote repositories (using a local clone), but that's work for past 0.9 and past the scope of this bug.

Yes, the memory wasting is obvious.
But in my opinon syncing the repository using a cron job is the more elegant solution, escpecially when combining that step with Repository.fetch_changesets.

#15 - 2010-01-13 23:39 - Philip Hofstetter

Dominik Wild wrote:

> > Oh, I think I overlooked the emphasising of "what if". In that case we indeed have a problem.
>
> nope. not really. fetch_changesets will do the right thing, because when a rebased branch gets pushed, our git log --all will not list the now unreachable commits which will then get removed.
>
> Yes of course, I was referring to the "delta" fetching idea. Because assuming there are no rewrites it would be feasible to just get the last imported revision of the database and fetch the revisions starting from there.

true. Unfortunately, it's impossible to detect whether a rewrite happened or not without changing redmine's schema, which I consider to be too much hassle, especially in light of the fact that no other SCM would require this kind of infrastructure. Still. Aside of the memory consumption problem, just doing what fetch_changesets currently does (and without the --find-copies-harder) is perfect and catches the rewrites.

> I will still investiate into my earlier solution of using the changesets table as a simple cache (saving huge amounts of memory - right now, fetch_changesets will read all commits plus messages into memory) plus the ability to have support for remote repositories (using a local clone), but that's work for past 0.9 and past the scope of this bug.
>
> Yes, the memory wasting is obvious.
> But in my opinon syncing the repository using a cron job is the more elegant solution, escpecially when combining that step with Repository.fetch_changesets.

It could be, but you see, the code is already a hybrid: It reads all revisions. True. But it's shelling out much more often than any other SCM adapter already. And as long as redmine does not track branching- and parent/child relationships, there's no way around shelling out to determine which revisions belong to which branch - information which is undoubtedly useful.

So the idea is: If we shell out often, why not always shell out and only copy the revisions we really need? Redmine's current schema is not up to the task to handle a git repository (aside of just providing metadata for a single commit).

#16 - 2010-02-28 11:15 - Jean-Philippe Lang

- Status changed from New to Closed

- Target version set to 0.9.3

- Resolution set to Fixed

A few commits were merged in 0.9-stable including r3394 and r3468.
This brings back fetching git changesets to a decent speed.