ChiliProject - Feature # 317: Improved performance with large git repositories

Status:	Open	**Priority:**	Normal
Author:	David Kowis	**Category:**	SCM
Created:	2011-04-05	**Assignee:**	
Updated:	2011-07-07	**Due date:**	
Remote issue URL:			
Affected version:			

Description: Performance of a large git repo is pretty horrible, at least in the Web Repository front-end.

I know this is redmine, but at this current point in time, I don't think the code bases will have diverged sufficiently to be the effect.

http://scm.shlrm.org/redmine/projects/smgl-test-grimoire

The repository link on there takes about 13 seconds to render the page. Makes lots of git calls and is generally very slow.

I'm going to look into seeing what kind of performance improvements I can make on a fork in github. Making this issue so that my fork has a link back to the project.

History

2011-04-05 06:56 pm - David Kowis

-https://github.com/dkowis/chiliproject/tree/feature-317-

Maybe by linking this here I'll actually do something about it ;)

2011-04-05 09:10 pm - David Kowis

This is quickly turning into a massive refactor of the git adapter... I will need assistance in improving the tests to figure out exactly what the behavior is supposed to be, since I think the tests cover like maybe 20% of the functionality.

2011-04-05 09:14 pm - David Kowis

I'm replacing the existing functionality with grit https://github.com/mojombo/grit

it appears to be maintained, and it does everything that the git adaptor is currently doing, without all the parsing. Hopefully by abstracting that to someone else's well tested git library, one can worry less about parsing CLI output and more about making it work well.

2011-04-05 10:11 pm - David Kowis

So Grit doesn't deal well with files that have spaces ending or leading. How often is this encountered in the wild?

I would say drop that support in favor of having a more maintainable git adapter, and we'd do better getting one patch into grit rather than trying to maintain the currently scary git-adaptor.

I filed this issue: https://github.com/mojombo/grit/issues#issue/59 on grit, and I looked into how to test it, but I haven't figured out a solution yet.

2011-04-06 02:17 pm - David Kowis

https://github.com/mojombo/grit/pull/60

This fixes the trailing and leading spaces stuff with Grit. Hopefully the pull request is taken.

2011-04-06 10:13 pm - Eric Davis

- Category set to SCM

When branch support was added to the git adapter someone did a quick performance test of grit and found that grit was actually slower than shelling out to @git@. Things might have changed but you might be able to dig up some information about that.

I'm also going to be pulling in some of Toshi MARUYAMA's SCM commits soon. There are a few git changes and more test support if I recall.

2011-04-06 10:28 pm - David Kowis

http://www.redmine.org/issues/6566 Yeah I just found that :|

Maybe it'd be good to use libgit2 stuff anyway, but I don't know how stable that is.

```
<pre>
$ irb
ruby-1.9.2-p180 :001 > require 'grit'
 => true
ruby-1.9.2-p180 :002 > include Grit
 => Object
ruby-1.9.2-p180 :003 > repo_path = '/home/dkowis/gitwork/personal/chiliproject'
 => "/home/dkowis/gitwork/personal/chiliproject"
ruby-1.9.2-p180 :004 > grit = Repo.new(repo_path)
 => #<Grit::Repo "/home/dkowis/gitwork/personal/chiliproject/.git">
ruby-1.9.2-p180 :005 > recent_list = grit.commits_since(start = 'master', since = '2010-10-10', extra_options = {})
SystemStackError: stack level too deep
	from /home/dkowis/.rvm/rubies/ruby-1.9.2-p180/lib/ruby/1.9.1/irb/workspace.rb:80
Maybe IRB bug!!
</pre>
```

Yeah, so that's no good at all, heh. Welp, I might commit the work anyway, and then start converting it to gitlib2, because it's somewhat similar. BLARG.

2011-04-06 10:28 pm - David Kowis

Eric Davis wrote:

> I'm also going to be pulling in some of Toshi MARUYAMA's SCM commits soon. There are a few git changes and more test support if I recall.

Perhaps it'd be best if I wait for that to show up, to avoid duplication of work and conflicts.

2011-04-06 10:49 pm - David Kowis

Okay, so I have pushed my grit changes into a grit-fail branch, because grit is fail :(Perhaps tomorrow I'll look at what the effort will be to use the native git.

Perhaps tomorrow I'll have some time to investigate the libgit2 stuff.

2011-04-08 01:49 pm - Andy Bolstridge

I'm interested in this as I did a patch to improve performance for large subversion repositories.

http://www.redmine.org/issues/7528

This pulls down a limited number of revisions from the repo to populate the database, as I have 300,000+ revisions in my repo, it used to take a very long time to read in ancient revisions that no-one is interested in. This patch was (rightly) rejected as it didn't cope with git revision handling (I used rev numbers). I mean to update it and resubmit it, but if you're modifying the git adaptor, I'll wait until you're done. If you want to keep the way I changed redmine to fix the issue, then that would be great too. FYI I figured on refactoring the adaptor to use date ranges instead.

2011-04-08 02:01 pm - David Kowis

Andy Bolstridge wrote:
> I'm interested in this as I did a patch to improve performance for large subversion repositories.
>
> http://www.redmine.org/issues/7528
>
> This pulls down a limited number of revisions from the repo to populate the database, as I have 300,000+ revisions in my repo, it used to take a very long time to read in ancient revisions that no-one is interested in. This patch was (rightly) rejected as it didn't cope with git revision handling (I used rev numbers). I mean to update it and resubmit it, but if you're modifying the git adaptor, I'll wait until you're done. If you want to keep the way I changed redmine to fix the issue, then that would be great too. FYI I figured on refactoring the adaptor to use date ranges instead.

The initial load isn't so bad, it takes forever on my git repo as well. I can run the initial load asynchronously so it doesn't affect the web interface, the issues I'm trying to solve revolve around the performance of the git repo web interface. Unfortunately, the work I did modifying the git adaptor failed miserably, because I neglected to make sure grit didn't suck. Unfortunately grit tanks when trying to access any length of history.

The ajax thing mentioned earlier might be an optimal solution, as it allows the history data for each file to be queried individually, and the server can get back to us eventually. I'm thinking of adding a patch with an option to turn off the per file history stuff just to speed up large repos as a temporary fix, maybe I can do it in a plugin.

2011-04-08 02:40 pm - Toshi MARUYAMA
David Kowis wrote:
> I'm thinking of adding a patch with an option to turn off the per file history stuff just to speed up large repos as a temporary fix, maybe I can do it in a plugin.

Please see http://www.redmine.org/projects/redmine/repository/revisions/4945 .

2011-04-08 02:44 pm - Toshi MARUYAMA
And please see http://www.redmine.org/issues/4455#note-73 .

2011-04-08 03:52 pm - David Kowis
Toshi MARUYAMA wrote:
> David Kowis wrote:
> > I'm thinking of adding a patch with an option to turn off the per file history stuff just to speed up large repos as a temporary fix, maybe I can do it in a plugin.
>
> Please see http://www.redmine.org/projects/redmine/repository/revisions/4945 .

Yeah I can do that, I was thinking of adding it as a plugin and monkeypatching the existing code instead of patching it. But patching it is probably the easiest thing to do. Thanks.

Toshi MARUYAMA wrote:
> And please see http://www.redmine.org/issues/4455#note-73 .

Yeah, that one there might be solved using libgit2 bindings. I think you can get a list of commits you don't have on an update when doing a fetch. If so, you can then only update the things that you're fetching. Of course, this would require doing some things differently with git entirely. I guess I should write up a potential long term plan and some details on doing it. Unfortunately I don't know if I'll have time to do these things...

2011-04-08 05:15 pm - Andy Bolstridge
turning off the initial load is already an option (in RM at least), but that's not optimal. Fetching a small set of recent changes on initial load is much better - I fetch 200 and only get more if the user requests them. Changing this to the last week/10 days/configurable option would be better, but requires a small tweak to the methods used by the adapters.

My repo didn't go slowly BTW, it would time out completely. That makes it a problem :)

2011-04-11 04:44 pm - David Kowis

Andy Bolstridge wrote:

> turning off the initial load is already an option (in RM at least), but that's not optimal. Fetching a small set of recent changes on initial load is much better - I fetch 200 and only get more if the user requests them. Changing this to the last week/10 days/configurable option would be better, but requires a small tweak to the methods used by the adapters.
>
> My repo didn't go slowly BTW, it would time out completely. That makes it a problem :)

Perhaps for subversion, but this has nothing to do with git. Doing a background initial import is no big deal.

2011-06-09 06:59 pm - David Kowis

Okay, so what I'm thinking needs to be done here is a significant rewrite of how repositories are handled. I'm going to approach it from the git aspect, as that's the SCM I know the best, but I think most of it can translate to the other SCMs with minor changes.

Primarily, the interface to the SCM should be through a native library, or as close as possible. Perhaps use Ruby-FFI for things that don't have maintained and tested ruby bindings. The other benefit to using Ruby-FFI is that it will work with JRuby.

Secondarily, and almost as important, will be redoing the repository controller so that things can happen asynchronously. It would have to be a requirement that JavaScript be supported for the asynchronous rendering of information, unfortunately.

Doing it this way, we can take advantage of rails caching to cache the generation of JSON responses, unless the file has been updated. This should dramatically improve loading times past the first one, it's even possible that we could precache it on import of new SCM data. We'd be making it work similarly to github, in that it will make concurrent requests to the back-end to render the SCM data on the page, rather than shelling out each time to make it happen, as that's terribly slow. With better knowledge of the updates coming in from the repo, we can better clean the caches and actually use caching to keep the performance high, rather than hitting the SCM library every time.

Another issue would be how we keep SCMs up to date. An algorithm would have to be built for each SCM that we would use to poll data. Using git, and an anonymous read only access (or maybe even an authenticated access) we could get the information regarding what packs are being fetched, and then inspect those packs and determine which files are being touched by the packs. This is a change to the existing setup. Instead of specifying the path to a .git bare repository, we could have the user enter a url to a git repository, whether remote or file://. Then upon repository creation, chili will go ahead and clone a repository the way it wants one to be and maintain control of updating branches and such. Far more efficient than the current setup, which has no clue about things being updated. This might necessitate some cron, or a mechanism that would check to see if a time-to-live has passed before firing off a new request for update. Perhaps another option would be a background job server like Delayed Job or something like that to ensure that things that are long running are handled correctly. Github uses their own thing, resque, open sourced, but may be suitable if we can properly integrate it into the way we start up Chili. It would certainly be useful if we opt to do more things in the background.

CVS/Subversion already supports a nice way of determining what new changes exist, since it's just a sequential bit of numbers. Getting further information from Mercurial and other distributed SCMs might be tricky, but I'm sure can be accomplished somehow. Optimizing will likely be the hardest part, getting native ruby interfaces to the stuff instead of shelling out.

As a side note, the various graphics don't work very well when there's a large number of committers or when there's lots of activity. The graphics tend to get all smorshed and difficult to look at. Perhaps revamping those, or taking them out for the first iteration of the new SCM thing and then implementing different graphs.

So this ultimately will result in a rewrite of most of the SCM logic to behave completely differently whilst still presenting a similar interface to the user. It might be best to hold off on it until we get to the Rails 3, as the inherent support for unobtrusive JavaScript will make the JavaScript authoring much cleaner and keep the views clean. As for JavaScript degradation... Maybe we simply won't display the advanced information you'd get from the JavaScript interface. but you'd still get to navigate the source tree and look at diffs and whatnot. I don't think it's unreasonable to require JavaScript support, however.

2011-06-10 04:19 pm - Eric Davis

David Kowis:

tl;dr: I like the ideas and would encourage someone to start developing this.

I had a similar idea back in 2008 while attending RailsConf and seeing how @grit@ worked. There are 3 points that I see:

# Should rewrite the SCM adapters to use interfaces closer to the metal (e.g. native or FFI libraries) instead of parsing shell commands
# Should make the web view asynchronous
# Should provide some way to auto update the data when a repo is changed.

I'll try to quickly address each of these:

h3. Should rewrite the SCM adapters to use interfaces closer to the metal (e.g. native or FFI libraries) instead of parsing shell commands

95% agree. Shelling out is really slow and parsing the information is really complex. There was a prototype of using a Ruby gem for the git adapter and from what I heard:

* it was slower
* but the code was clearer

Personally, I'd rather have clearer code than speed. It's easier to make slow code faster if you can understand the code.

The only problem is that a rewrite of the adapters would be a large undertaking. I think if someone wanted to start porting them one at a time we could start to incorporate them as they are finished.

h3. Should make the web view asynchronous

100% agree. I actually have a quick solution to this:

# When the repo page is loaded, show the page and send an asyc request to server to "Update repo data"
# The user can browse around parts while the data is loading
# When the data is loaded on the backend, a message could appear asking the user to refresh the page to see the latest data

Then later on we can start breaking out chunks of the page (e.g. just the repo browser, revisions list, file commits..) In an *ideal* world, I'd like to add a RESTful API to the Repositories and then we use our own API in the view. That way others could use the API also.

h3. Should provide some way to auto update the data when a repo is changed.

I agree but this could be started separately from the core code. Using a plugin someone could write the "updaters" for each SCM and then when it's done we could add it to the project.

Summary: I'd be happy to have someone make all of these changes. But I don't use the repository module that much myself, partially because these (and other) limitations so it isn't a top priority for me. If someone else wants to take over the SCM module and build a team to build these features, I'd be extremely happy.

2011-06-10 04:32 pm - Andy Bolstridge
Well..

Shelling out might be less performant than an API call, but by how much? I think you'll find that the cost of shelling is insignificant compared to the cost of pulling the data from the repo. The cost of parsing the output is likely to remain, either directly in chili code, or in the API adapter for the SCM so that's not really going to make it perform better.

Providing an API is the best option for these to work on, as then whatever mechanism you use to update the view with new commits will have a nice

API to work with - a hook or script can just call the API and you're job's pretty much done.

the best 'bang for buck' however is to load minimal repo views - currently you load everything and this is slow. If you maintained the earliest received change, you can optimise the fetch of the original import of history. Then, you'll find things go faster anyway because theres less data in the DB to run through - data that no-one is necessarily going to view anyway (ie no-one really cares that chili hasn't loaded history from 10 years ago). A background import for any SCM doesn't take any time as far as the user is concerned (unless they add the repo, and immediately go to view it... but who'd do that :)), but even if the repo imports overnight... you've still filled your DB.

One thing Redmine users were clamouring for was multiple repository links for each project. I think this is a significantly good thing to have, and would recommend adding it to the list of changes you want to make, if you're updating the way Chili interacts with the SCM.

2011-06-10 04:36 pm - David Kowis

Eric Davis wrote:
> h3. Should rewrite the SCM adapters to use interfaces closer to the metal (e.g. native or FFI libraries) instead of parsing shell commands
>
> 95% agree. Shelling out is really slow and parsing the information is really complex. There was a prototype of using a Ruby gem for the git adapter and from what I heard:
>
> * it was slower
> * but the code was clearer
>
> Personally, I'd rather have clearer code than speed. It's easier to make slow code faster if you can understand the code.

Ruby-FFI is reasonably easy to understand, as it's all ruby code. Use that only if there's not native ruby bindings for the other SCM tools. There's libgit2 for git, specifically, which has ruby bindings already: https://github.com/libgit2/rugged From a cursory inspection, the code wouldn't be much more complex than doing the shell out stuff anyway.

>
> The only problem is that a rewrite of the adapters would be a large undertaking. I think if someone wanted to start porting them one at a time we could start to incorporate them as they are finished.

Yeah, I was thinking that we'd leave the old functionality in place until the at least one of the newer ones is functional, as once the framework is in place, building the SCM adaptors should be easy.

>
> h3. Should make the web view asynchronous
>
> 100% agree. I actually have a quick solution to this:
>
> # When the repo page is loaded, show the page and send an asyc request to server to "Update repo data"
> # The user can browse around parts while the data is loading
> # When the data is loaded on the backend, a message could appear asking the user to refresh the page to see the latest data

This would certainly solve the problem of importing new data into the repo, but for repos with lots of files in a directory, the current system is unbearably slow. It would only solve one of the problems, not the one that happens every time someone loads a page (as it works right now.)

> Then later on we can start breaking out chunks of the page (e.g. just the repo browser, revisions list, file commits..) In an *ideal* world, I'd like to add a RESTful API to the Repositories and then we use our own API in the view. That way others could use the API also.

Yeah I would rather focus directly on breaking out chunks of the page. It can be done via JSON and some javascript to incorporate it into the main view. Nice and asyncronous. Also gives us the advantage of being able to use caching for some of the more intensive spots, pending better knowledge of updating the SCMs. Upon updates to the SCM, the caches would be swept for the changed items.

>
> h3. Should provide some way to auto update the data when a repo is changed.
>
> I agree but this could be started separately from the core code. Using a plugin someone could write the "updaters" for each SCM and then when it's done we could add it to the project.

I'm not completely sure I agree with this. Maybe we're thinking about it differently. The update logic needs to be part of the core to take advantage of the cache sweeping and other such things. Running it in the background could certainly be a plugin, as that doesn't matter so much. But updating the repo should be part of the adapter that is built, as it's necessary, especially for distributed SCMs, to figure out what to update in redmine (loading commit history into the database, etc.)

>
> *Summary*: I'd be happy to have someone make all of these changes. But I don't use the repository module that much myself, partially because these (and other) limitations so it isn't a top priority for me. If someone else wants to take over the SCM module and build a team to build these features, I'd be extremely happy.

That was one of the things I was fishing for. It needs a targeted release time, especially with things being updated to Rails 3. It would break existing functionality until things were completed.

One task is building a new repository controller, setting it up to handle more AJAX and JSON requests, maybe even a separate controller to do the web calls into repository data. And then building a new web page combining those things to present them to the user.

The other task would be redoing the SCM adaptors to be faster and more efficient. They should be reasonably divorced from the presentation, except where things cannot be: Branches in git are a differently handled beast than branches in subversion, and other differences. It doesn't have to be implemented using native code, it would just be faster in most cases. Do what makes sense with the SCM. If subversion isn't any faster with native, but is way more complex, then there's no reason to force it into a native implementation. Git is terribly slow with the current shell parsing mechanism, so a native implementation should improve things dramatically.

So minimum acceptance criteria for the project?
# All SCMs supported that are currently supported
# Supports running on Windows

Another question would be, what's the best way to start building this? Probably a branch off of unstable, kept in sync with unstable, and then pull it into a release branch when it's ready?

2011-06-10 04:39 pm - David Kowis
Andy Bolstridge wrote:
> Well..
>
> Shelling out might be less performant than an API call, but by how much? I think you'll find that the cost of shelling is insignificant compared to the cost of pulling the data from the repo. The cost of parsing the output is likely to remain, either directly in chili code, or in the API adapter for the SCM so that's not really going to make it perform better.

In terms of git, the shell calls is what makes it slow, because there's lots of files visible in the main repo view, not even history. In the tests I've ran, with my git repository, not shelling out would significantly improve performance, as would caching so we wouldn't even have to hit the repo at all to get the data out, just serve up static JSON, or HTML or whatever.

>
> Providing an API is the best option for these to work on, as then whatever mechanism you use to update the view with new commits will have a nice API to work with - a hook or script can just call the API and you're job's pretty much done.
>

> the best 'bang for buck' however is to load minimal repo views - currently you load everything and this is slow. If you maintained the earliest received change, you can optimise the fetch of the original import of history. Then, you'll find things go faster anyway because theres less data in the DB to run through - data that no-one is necessarily going to view anyway (ie no-one really cares that chili hasn't loaded history from 10 years ago). A background import for any SCM doesn't take any time as far as the user is concerned (unless they add the repo, and immediately go to view it... but who'd do that :)), but even if the repo imports overnight... you've still filled your DB.

The initial import is slow, but that's not a big deal, we can improve that. I do want to do smarter loads regarding the SCM imports of commit data.

>
> One thing Redmine users were clamouring for was multiple repository links for each project. I think this is a significantly good thing to have, and would recommend adding it to the list of changes you want to make, if you're updating the way Chili interacts with the SCM.

That would be nice, and probably doable. Especially since most of the loading and presentation code would be rewritten. I don't think it'd be too much more to make repositories many-to-one with a project.

2011-06-10 04:48 pm - Andy Bolstridge
ok. I can say that using a native API for SVN would probably not gain you much at all - the shell command line is just a thin wrapper over an internal API anyway, so I don't think you'd see any performance difference. just FYI.

There is a patch for mutl-repos in redmine (ticket #779) that works (with a couple of niggles). However,it uses a parameter to determine which repo you're looking at as you browse through the project, and I don't think it was implemented as well as it could have been - the repo id is the index into the project's list of repositories. But apart from that, it works great and could be a starting point for someone prepared to really refactor a fair chunk of the methods. It does affect a lot of chili though - eg. a ticket that is fixed in a commit needs to know which repo so it can show the correct revision, currently it stores and passes the project id.

That one has only been open for 3 years :)

2011-06-10 04:55 pm - Eric Davis
David Kowis wrote:
> So minimum acceptance criteria for the project?
> # All SCMs supported that are currently supported
> # Supports running on Windows

Yes and:

# Throughly tested code
# Code level documentation for people like me that don't have a clue what the SCM code is actually doing :)

> Another question would be, what's the best way to start building this? Probably a branch off of unstable, kept in sync with unstable, and then pull it into a release branch when it's ready?

# Branch from unstable
# Sync with unstable over time
# Send pull requests when feedback is wanted and/or the feature is complete

Following our [[Code Standards]] and [[Code Review]] standards.

Andy Bolstridge wrote:
> Shelling out might be less performant than an API call, but by how much? I think you'll find that the cost of shelling is insignificant compared to the cost of pulling the data from the repo. The cost of parsing the output is likely to remain, either directly in chili code, or in the API adapter for the SCM so

that's not really going to make it perform better.

That's something we wouldn't know until a prototype was done. For me, even if performance was the same or a bit slower: having easier to read (and change) code on our side and pushing some of the maintainable to the libraries/adapters would be an improvement.

Now that I think about it, I think we support using libsvn or something directly already. I think it showed a decent performance improvement itself.

2011-06-10 05:51 pm - David Kowis

Andy Bolstridge wrote:

> That one has only been open for 3 years :)

Heh, I might have you beat (nope, not quite): http://www.redmine.org/issues/1984

Perhaps I'll work on that one when I get a chance as well :)

2011-07-07 06:39 pm - David Kowis

Toshi MARUYAMA wrote:

> David Kowis wrote:
> > I'm thinking of adding a patch with an option to turn off the per file history stuff just to speed up large repos as a temporary fix, maybe I can do it in a plugin.
>
> Please see http://www.redmine.org/projects/redmine/repository/revisions/4945 .

So I'm going to note that this change is in the 2.0.0 branch. Just so that I don't forget about it :D Yay for temporary fixes :D