redmineorg-copy202205 - Vote #63388

When downloading a file from the repository : don't fill the memory with bytes

2022/05/09 13:26 - Admin Redmine

ステータス:	New	**開始日:**	2008/06/09
優先度:	通常	**期日:**	
担当者:		**進捗率:**	0%
カテゴリ:	SCM_3	**予定工数:**	0.00時間
対象バージョン:		**作業時間:**	0.00時間
Redmineorg_URL:	https://www.redmine.org/issues/1410	**status_id:**	1
category_id:	3	**tracker_id:**	2
version_id:	0	**plus1:**	1
issue_org_id:	1410	**affected_version:**	
author_id:	809	**closed_on:**	
assigned_to_id:	1	**affected_version_id:**	
comments:	16		

説明

Current procedure for downloading a file from the repository implies dumping the bytes (all of them) into a string through method "cat", then serving the content of the string. Although it works well for small files, some problems are raised when using large files :

dumping a file into the RAM is a bit risky, as it is quiet common now to have files larger than the memory size. And we don't want servers to swap :)

it locks a ruby thread during the download time, which is a waste of ressources

Point 1) can probably be adressed using streams, but this won't fix point 2)
Point 1) and 2) can be fixed by dumping file content into a temporary file in the filesystem (RAILS_ROOT/tmp for example) and asking the webserver (apache, or any other) to serve the content as a static file. That would free the ruby process for some other work.

What do you think ?

journals

Here is a patch that tries to solve the problem of downloading a file larger than the memory size. I had to implement a buffered IO with a is_binary_data? that encapsulate an IO instance reading the result of the cat command ("svn cat" for example). This buffered IO uses 4Mo maximum of ram to read the beginning of the file and do the is_binary_data? test from String class. It can maybe be reduced without influencing the test accuracy...

I tested the patch only with subversion adapter for the moment.

This code doesn't work as expected. I have to work a bit more on it...

Here is a patch that seems to work correctly. I included some unit tests.

I tried your patch on my machine, which has 512mb RAM and I need to transfer files of 400mb size. Unfortunately this does not work, the machine runs out of memory and does not respond anymore. If I try smaller files (50mb) it works, but the memory is not freed by redmine after the transfer completed. I think this is a serious problem and it is sad to see that it was taken out of 0.8 as target.

I also tried the latest patch. It works much better than the current code with with files around 100MB.
But it doesn't handle a 700MB file (I've got 3GB of RAM):

```
NoMemoryError (failed to allocate memory):
        /app/controllers/repositories_controller.rb:135:in   `write'
        /app/controllers/repositories_controller.rb:135:in   `entry'
        d:/dev/ruby/lib/ruby/gems/1.8/gems/actionpack-2.1.2/lib/action_controller/cgi_process.rb:212:in   `call'
```

What is strange is that I get the same error when trying to stream a file located on the disk using the standard send_file Rails method in streaming mode: http://api.rubyonrails.org/classes/ActionController/Streaming.html#M000266.

Pierre, do you have this problem with similar files?

I only tested that all I give to the response is bufferized. It's the case, as when using the send_file method. The problem yet with memory usage is caused by Mongrel or Webrick that uses a StringIO to buffer the response.

It seems that Rails+Mongrel is unable to stream a response ! The send_file problem is a known problem. Mongrel need as much memory as the file size. The memory is freed by the garbage collector a long time after the response has been transmitted. This is exactly the same problem with the patch I purposed.

An alternative could be x-send-file : http://wiki.rubyonrails.org/rails/pages/HowtoSendFilesFast.

As a workaround, could you add the possibility to add links to files in the files component that can then be handled as arbitrary http-requests by apache?

Issues 502 and 2205 requested having links to files as well.

+ 1

+1000

We've been running into this a lot at "Planio Redmine Hosting":http://plan.io/redmine-hosting recently.

Our approach to solving this (works perfectly in production with 20,000+ Redmines) is piping the output of the repository cat operation into a tempfile and serving this using @send_file@.This works great by itself since it uses buffered streaming internally.

Our finding was that the real problem with Redmine's current implementation is loading the content into a single ruby variable, not so much the locking of a ruby process during the time the file is streamed.

Here's a rundown as an example using git:

repositories_controller.rb:

```
def  entry

    #  ...

    if  'raw'  ==  params[:format]  ||
              (@entry.size  &&  @entry.size  >  Setting.file_max_size_displayed.to_i.kilobyte)  ||
              !  ((@content  =  @repository.cat(@path,  @rev))  &&  is_entry_text_data?(@content,  @path))

        #  Force  the  download
        send_opt  =  {  :filename  =>  filename_for_content_disposition(@path.split('/').last)  }
        send_type  =  Redmine::MimeType.of(@path)
        send_opt[:type]  =  send_type.to_s  if  send_type

        #  if  we  have  the  @content  already  use  it  -  otherwise  use  a  tempfile
        if  @content
            send_data  @content,  send_opt
        else
            send_file  @repository.cat(@path,  @rev,  true),  send_opt
        end
    else
        #  ...
    end
end
```

repository.rb:

```
def cat(path, identifier=nil, use_tempfile=nil)
  if use_tempfile
    File.join(Dir.mktmpdir, 'repository.tempfile').tap do |tempfile|
      scm.cat(path, identifier, tempfile)
      Thread.new do
        sleep 1 ; File.unlink(tempfile) ; Dir.unlink(File.dirname(tempfile))
      end
    end
  else
    scm.cat(path, identifier, nil)
  end
end
```

git_adapter.rb

```
def cat(path, identifier=nil, tempfile=nil)
  # ...
  git_cmd(cmd_args, :tempfile => tempfile) do |io|
    io.binmode
    cat = io.read
  end
  cat
  # ...
end
```

abstract_adapter.rb

```
def self.shellout(cmd, options = {}, &block)
  # ...
  if options[:tempfile].present?
    # write stdout to tempfile if given
    cmd = "#{cmd} > #{shell_quote(options[:tempfile])}"
  end
  # ...
end
```

As I said, this works great already. Two apparent downsides:

- We're spawning a thread.
- For deployment, you have to have a @/tmp@ dir that has enough space for large files in your repo.

You can take this a step further even, depending on what you're using as a frontend. With Nginx (Apache should work, too), we were even able to free the Ruby process from streaming entirely. A simple @send_opt.merge(:x_sendfile => true)@ and a little tweak in the nginx config allow us to use @X-Accel-Redirect@ for this and stream the tempfile without ruby having to read a single byte.

I have a patch against 1.4 that patches all repository types (but needs testing with repos other than git & svn) and I would be willing to prep that for trunk, but before we start working on this I wanted to get a contributors opinion, so that we're sure the patch won't end up unused.

Looking forward to your feedback.

Writing the file to disk before sending it is indeed the way to go IMO, but:

- I don't get the thread thing: you're trying to delete the file after 1 sec, is that right? But we just can't know when sending will be finished
- Writing to disk small files seems to be less efficient, there should be some kind of threshold on the file size that triggers the use of a temp file
- If we're trying to adress the problem of big files, a better option would be to checkout the file only once and serve it multiple times
- No need to patch with @:x_sendfile => true@, you just need to set @config.action_dispatch.x_sendfile_header@ appropriately

I started working on this some time ago, I'll see what I have.

Jean-Philippe Lang wrote:

> Writing the file to disk before sending it is indeed the way to go IMO, but:
>
> - I don't get the thread thing: you're trying to delete the file after 1 sec, is that right? But we just can't know when sending will be finished

Yes, it gets unlinked after 1 sec. Actually, you just need to be sure that streaming has started (not finished) before the file is unlinked. If the file has been opened, the process has a handle and can finish streaming it even if it gets unlinked.

> - Writing to disk small files seems to be less efficient, there should be some kind of threshold on the file size that triggers the use of a temp file

I'm doing this already. Check this out:

```
if 'raw' == params[:format] ||
     (@entry.size && @entry.size > Setting.file_max_size_displayed.to_i.kilobyte) ||
     ! ((@content = @repository.cat(@path, @rev)) && is_entry_text_data?(@content, @path))
```

If the file is small (i.e. @@entry.size > Setting.file_max_size_displayed.to_i.kilobyte@ evaluates to @false@), lazy evaluation will execute @@content = @repository.cat(@path, @rev)@. Note, that the third argument (@use_tempfile@) is not set. Then, later on:

```
# if we have the @content already use it - otherwise use a tempfile
if @content
  send_data @content, send_opt
else
  send_file @repository.cat(@path, @rev, true), send_opt
end
```

If we have @@content@ already, we just use @send_data@. If @@content@ is @nil@ (because is wasn't set earlier), we use the tempfile thing. Note that here the third argument is set to @true@.

> - If we're trying to adress the problem of big files, a better option would be to checkout the file only once and serve it multiple times

True, but then you would need something like a cronjob to erase files according to some LRU algorithm.

> - No need to patch with @:x_sendfile => true@, you just need to set @config.action_dispatch.x_sendfile_header@ appropriately

You're right, I think for Redmine 2.x/Rails 3.x @:x_sendfile => true@ is not needed anymore...

> I started working on this some time ago, I'll see what I have.

If that helps I can supply my raw patch against 1.4... Let me know.

Would it be useful to stream data?

Something like:

```
class IOStreamer

  def initialize(input)
    @in = input
  end

  def each
    while data = @in.read(4096)
      yield data
    end
    @in.close
    end
end

self.response_body = IOStreamer.new(scm_stdout)
```

Any plans to solve this issue?

---

related_issues

relates,New,29250,Problem with high RAM usage
duplicates,Closed,11275,Large file download

履歴

#1 - 2022/05/10 17:28 - Admin Redmine

- カテゴリ を SCM_3 にセット