Introduction

From the developers:

“Collabbit is a web-based communication tool which helps coordinate and organize relief efforts in times of disaster and recovery. It aims to eliminate burdensome conference calls between the various parties involved, replacing them with quick and easy updates online, as well as provide a written record of the progress of the event.”

Collabbit allows different organizations and groups to post emergency information online. Collabbit can also send the posted information to users’ email and phone via text message.

Because of the potentially large number of users, when sending email and text alerts to many users, the system could become overwhelmed. A queue needs to be implemented where tasks such as email and text sending can be queued and run at a later time.

Message Queue using DelayedJob

In this project, we used collectiveidea’s DelayedJob which is a queuing system that can run queued tasks in the background.

collectiveidea delayed_job (v.2.0.3)
http://github.com/collectiveidea/delayed_job/tree/v2.0.3

Installation

For Linux,

1. Download and extract source code
2. Install MySQL
   You will need to install “mysql-server” and “libmysqlclient-dev”
3. Setup MySQL
   If you are prompted to setup password, leave it blank.
   Run `mysqladmin -u root create collabbit-dev` to create the database
4. Run `rake gems:install` to install the necessary gems
5. Run `rake db:redo` to reset the database tables
6. Modify /etc/hosts
   Add the following line “127.0.0.1 collabbit.dev demo.collabbit.dev”
7. Run `script/server` to run the server
8. Run `script/delayed_job run` to run the delayed_job daemon
9. If the environment is Production, make sure the root password is entered in the database.yml file for Production.

Notes:

Run `script/delayed_job` stop to stop all daemons, or type `script/delayed_job` to see list of available commands.

If you do not run the delayed_job daemon, tasks will still be queued, however, they will not execute until delalyed_job daemon is run.

Run `rake jobs:clear` to delete all jobs in the queue.

We modified the STMP settings in config/settings/smtp.yml to a Gmail account we created for testing purposes. Users who run the server, should receive email from collabbit.test@gmail.com

How to Receive Email/Text alerts from Collabbit

After server and delayed_job daemon are up and running, go to
http://demo.collabbit.dev:3000

1. Click on “Create an account”
2. Enter your name and email/phone information.
   Make sure to check “Receive Email Alerts” and/or “Receive Text Alerts”.
3. After an account has been created, the admin must approve.
   -Log out, and sign in as:
   Email: test1@collabbit.org
   Password: test
   -Click on the “Groups” tab
   -You should see a message “There are new users pending.”
   -Click on “Go to them.”
   -Click on “Approve User”
   -You should receive an email confirmation.
   -To activate, click on the link in the email.
   -Type in your desired password, and submit.
4. Log in with your new account.
5. Join a group.
   -Click on “Groups” tab. Pick and group and click on “Join”
6. Create a new incident.
   -Click on “Incidents” tab. Click on “New Incident”
   -Fill out the information, and submit.
7. Add an update
   -Click on “Add Update”
   -Fill out the information. Make sure to select your own group under “Relevant Groups”. Click on “Create Update” to submit.

You and the group members you selected in “Relevant Groups” will receive alerts about this

update.

Trying creating multiple updates without running the delayed_job daemon. You will see that the update alert messages will not be sent, but they are queued. Then activate the delayed_job daemon, and the queued messages will be sent.

Added/Modified files

README - delayed_job

Documentation on overview, installation, use, and changelog

config/environment.rb

added delayed_job gem to config.gem list
(testing) comment out line 37: “if(rails[ENV] == production)”
(testing) uncomment line 8:RAILS_GEM_VERSION and set the version to 2.3.4

Rakefile

added require statement

config/initializers/delayed_job_config.rb

Worker initialization parameters

config/settings/smtp.yml
(tesing)

Changed SMTP settings

Controllers

users_controller.rb
- resend_activation()
- reset_password()

Model

user_observer.rb
- after_create()
- before_update()

user.rb
- activate!()

update_observer.rb
- find_and_send_alerts()

Prioritization

delayed_job supports prioritization of the tasks sent to the queue. The table set up by delayed_job has a priority property that allows higher priority tasks to jump the the front of the

queue.

```
create_table :delayed_jobs, :force => true do |table|
  table.integer  :priority, :default => 0      # Allows some jobs to
jump to the front of the queue
```

Unfortunately, using the `send_later` function, the priority cannot be set. For example, the email delivery in /app/model/update_observer.rb was changed from

```
UserMailer.deliver_email_alert(user, feed, update, action)
```

to

```
UserMailer.send_later :deliver_email_alert, user, feed, update, action
```

to send the delivery task to delayed_job. We tried to append a priority parameter to the call as shown below, however, this does not work.

```
UserMailer.send_later :deliver_email_alert, user, feed, update,
action, :priority => 20
```

We checked the definition of `send_later` in <gems_directory>/delayed_job-2.0.3/lib/delayed/message_sending.rb and discovered that the definition appears as follows.

```
def send_later(method, *args)
     Delayed::Job.enqueue Delayed::PerformableMethod.new(self,
method.to_sym, args)
end
```

The `send_later` method expects a method symbol, then arguments to that method. When `:priority => 20` is append to the argument list, the method thinks that this an argument and method will fail to run.

Work-arounds

delayed_job supports another way of sending task to the queue. An entire method definition can be sent to the queue as shown below, using the `handle_asynchronously` keyword.

```
def some_method
    # Some other code
end
handle_asynchronously :some_method, :priority => 20
```

For example, the entire `find_and_send_alerts(update, action)` method

in /apps/models/update_observer.rb can be sent to the queue by appending the `handle_asynchronously` keyword.

However, this could have potentially dangerous behavior because there is code the in the `find_and_send_alerts(update, action)` method that searches for database entries which could out-of-date by the time delayed_job actually executes the method. Other methods like `resend_activation` in /app/controllers/users_controller.rb have real-time events such as `redirect_to :back` which probably should not be put into delayed_job queue, otherwise the redirection could occur at a later time.

The `handle_asynchronously` could also be appended to just the `UserMailer.deliver*` method, however, this would require modifying the UserMailer gem code.

Another way to add prioritization is to add a new function in the delayed_job gem. Specifically add a new function to the file <gems_directory>/delayed_job-2.0.3/lib/delayed/message_sending.rb

```
def send_later_priority(method, priority, *args)
     Delayed::Job.enqueue(Delayed::PerformableMethod.new(self,
     method.to_sym, args), priority)
end
```

This `send_later_priority` method allows us to pass in a priority parameter. Password related tasks have higher priority than updates, therefore we changed the `reset_password` method in /app/controllers/user_controller.rb to

```
UserMailer.send_later_priority :deliver_password_reset, 0, @user, pass
```

and changed `find_and_send_alerts` in /app/models/update_observer.rb to

```
UserMailer.send_later_priority :deliver_email_alert, 20, user, feed,
update, action
```

Now password reset has higher priority than the updates. **Lower number priority means higher priority.** So a priority of 0 is top priority. By default, when `send_later` is used, priority is 0 for all tasks.

This work-around requires modification to the delayed_job gem. If prioritization is absolutely needed, this may be a solution.

Note that we implemented the non-prioritized send_later function in our code.

Setting Worker Parameters

It is probably necessary to set the proper parameters for the workers. Please refer to the section “Gory Details” at http://github.com/collectiveidea/delayed_job/tree/v2.0.3

We have created a config/initializers/delayed_job_config.rb file which sets the parameters for the worker.

We set the following parameters:

```
Delayed::Worker.destroy_failed_jobs = false
Delayed::Worker.sleep_delay = 60
Delayed::Worker.max_attempts = 3
Delayed::Worker.max_run_time = 1.minutes
```

We set the max_run_time to 1 minute, since the task of sending an email should not take longer than this, especially when there are several emails waiting to be sent. DelayedJob will attempt 3 times to send email and if tasks fails, the :failed_at parameter will be non-null, and task will remain in the delayed_job table.