Message Queue Based Email Archiver

Student Information

Name	:	Anirudh Dahiya
Email	:	anirudhdahiya9@gmail.com
Telephone	:	+91-9052041894
Time zone	:	Indian Standard Time (UTC +5:30)

Profiles

IRC (freenode)	:	spark
Gitlab username	:	anirudhdahiya9
Instant Messaging	:	anirudh.dahiya.1 (Facebook)
Alternative Email	:	anirudh.dahiya@research.iiit.ac.in
Personal Blog	:	codefullofsummer.blogspot.com

Educational Information

University	:	International Institute of Information Technology, Hyderabad
Major	:	Bachelors in Computer Science and MS by Research in Computational Linguistics
Degree	:	Bachelors in Technology, MS by Research
Current Year	:	2nd year
Expected to graduate	:	May 2019

Project Details

Proposal Title

Message queue based email archiver

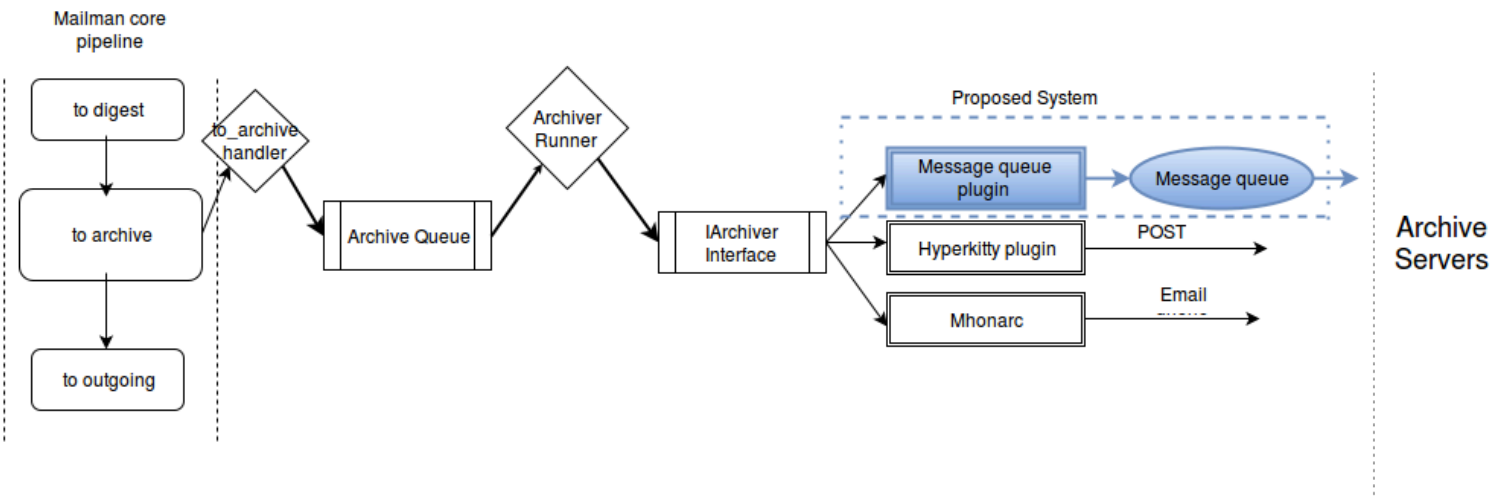
Background

Presently, the archiving system in Mailman core is based on the IArchiver interface from where the cleared mails are passed on to archivers via different archiver specific methods such as a simple POST request or an email directly to the archive servers .

Proposal Abstract

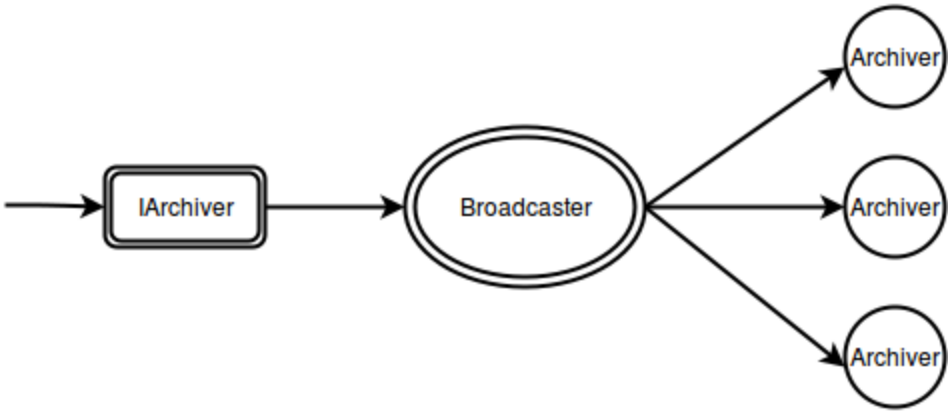
As part of the project, I intend to develop a message queue based system pluggable to the Mailman core archiving interface(IArchiver) which will enable **reliable**, **asynchronous** and **multi-client archiving**. This will also lead to a more robust distributed architecture while also expanding the scope of Mailman server allowing it to hook it up to a wide variety of web applications like static web page generators, event trackers or websockets servers.

Design Specification



From the archive queue, the mail is picked up by the archiver runner which uses the IArchiver interface to send the mails to archiver based on the archiver specific methods.

It is here at the IArchiver interface stage that I intend to plugin the message queue system.



An overview of the proposed pub/sub message queue system

The messages then will be pushed by IArchiver to the message queue system via plugins, where they would be handled by the broker/queuing device to which multiple applications can connect and receive messages **asynchronously**.

Proof of Concept

This is a small working model of the message queues implemented between a dummy server and client. Here we run multiple clients and differ the rate at which they can handle messages by using sleep functions. The clients irrespective of the rate recieve all the messages. Here I have also implemented selection based on topics as such filtering currently happens between mailing lists and archivers its mails are sent to. See Readme.md for more details.
https://github.com/anirudhdahiya9/prototype-message-queue

Deliverables

The primary deliverable of this project would be the message queue system with an interface to support multiple backends and message patterns. This can be broken down into the following parts -

- Minimum Viable Product :

  This would include a message system implemented with one suitable backend. This should be able to plugin to IArchiver interface and provide :

  I : Asynchronous messaging between Mailman server and clients

  II : Reliable transfer via acknowledgements or other suitable policies(mentioned below)
- Extension of mvp to support multiple backends :

  This would enable us to plugin multiple backends and message patterns to the mvp.
- Extendable message patterns :

  Use the above two to build the complete messaging system. This shall also include additional features such as persistence (discussed below)

Desirable features

- **Asynchronous message transfer**

  As the message queue decouples the archivers and the mailman server, archivers can accept messages asynchronously at their own pace as they connect.

- **Reliability**

  In case of network problems or client restarts, our system shall be designed to provide reliability.Though the exact methods are subject to the specific backend and further investigation, some policies in various backends to ensure reliability include -

  1. Keeping a specific time window before junking the messages
  2. Keeping a per subscriber queue with references to the relevant messages to be received by the subscriber.
  3. A protocol of sending acknowledgements upon receiving messages on the subscriber side

- **Persistence**

  In the event of broker restart or hardware failures, I intend to use persistence to disk feature provided by backends like Redis and RabbitMQ to store messages to disk at specified time intervals. The persisted messages can then be recovered when the broker restarts.

- **Support for multiple backends**

  The system plugin shall not be limited by the choice of backend

- **Support for multiple message patterns**

  The system plugin shall not be limited by the choice of specific message pattern design

Since our system further decouples the subscribers and Mailman core, it will go a long way in making Mailman a *robust distributed mailing list management system.*

A discussion on some backends investigated

Though the choice of initially included backends may change subject to further investigation and discussion during the community bonding period, a few backends (open source) which seem to fit the requirements of the project are -

- **RabbitMQ** - It also offers flexible and customizable routing options between publisher and message queues. This could be helpful to publish to select subset (based on mailing list) subscribers listening to our publisher. Also its reliability feature suits the requirements in our context as it provides both acknowledgements as well as personalised queues for each subscriber.
- **ZeroMQ** - Unlike others, it can run without a dedicated message broker. It basically provides web sockets which can be configured to customize message patterns, and can thus be used to suit specific personal needs. Though highly customizable, it might require explicit implementation of a few features.
- **Redis** - Though it started with some initial confusion about lack of any reliable way to message transfer after I approached the redis community for solutions, I subsequently found quite a nice solution in this blog post. It offers features like persistence to disk and message reference queues per subscriber, although no approach was found to provide retryability in case of failure. Thus , it is a lightweight broker system which can be used to provide reliability.

If the desired goals are met well in time and the timeline allows, the following tasks shall be attempted :

If the time allows, the idea of adding handlers to the Mailman chains so as to send list events to archive queue can be tinkered with. These events can then be transferred to the archivers via the message queue system or directly to archivers via interface IArchiver. This would open possibilities to developing an activity tracker application.

Future scope:

A variety of web applications which can also listen to mailinglist events such as list creation, deletion, policy change etc. can be developed and/or hooked up to the mailman. Applications such as static web page generators and websockets servers can can be hooked onto the message queue system. Since the system will be designed to be backend extensible, support for various backends implementing customized message queues can be implemented.

Timeline :

Till 10 May	Find out more about Message Patterns and understand Mailman and Archiver Interface code architecture. Get familiar with ‘tox’, the testing system.
10 May - 22 May	Find out more about the various backends available and corresponding message patterns supported. Discuss and finalise various architectural designs for messaging system with the mentors.
23 May - June 1	Implement plugin for IArchiver supporting multiple backends. Test the plugin with dummy messaging systems.
June 2 - June 12	Implement a message queue system for a viable backend. Finish with the minimum viable product.

June 13 - June 19	Write unit tests for the mvp. Ask for community review of the product. Catch up with any leftover tasks
June 20 - June 27	Code submitted for mid term evaluation. Write basic documentation for mvp. Reconsider the architecture for message queues and plugin.
June 28 -July 8	Improvise upon the message queue system based on the evaluation. Research about additional backends and message patterns viable for the job.
July 9 - July 25	Discuss and implement more message queuing patterns suitable. Implement support for suitable alternative backends. Attempt to incorporate list events into the messaging system.
July 26 - August 1	Refactoring Week. Seek community review for existing product and design architecture. Write integration tests.
Aug 2 - August 13	Complete any leftover work. Write documentation. Improvise upon reviews by mentors.Attempt to incorporate extensible metrics(as mentioned above). Write any tests left(system testing). Complete any unachieved milestone.
August 14 - August 20	Tidy up code. Complete documentation. Perform rigorous testing upon system. Submit for final submission.

Work Experience

Past projects:

I am well acquainted with python and C as programming languages and have done many projects based in these languages. I am also acquainted with websockets and have implemented an FTP application in python. I have primarily worked on Django as an MVC and have tried my hands at various aspects of it like authentication, generic views and customised forms, although I have working knowledge of web2py as well.
Particularly for this project, I've been through GNU Mailman architecture, listened to pycon talk by Barry Warsaw, played with the Mailman core codebase along with relevant portions in Hyperkitty and fixed issues in Mailman core and Postorius. I have also researched about message patterns (particularly message queues, pub/sub models and their variants), investigated various backends (RabbitMQ, ZeroMQ, and Redis) and complete relevant parts of tutorial for RabbitMQ and ZeroMQ.

Links to select projects :

FTP application implemented in python : https://github.com/anirudhdahiya9/File-transfer-app
Tic Tac Toe AI bot : https://github.com/anirudhdahiya9/Tic-Tac-Toe-bot-AI
Django Projects :
https://github.com/anirudhdahiya9/Web-developement-experiments-in-django
Open data Project (Semester project) : https://github.com/anirudhdahiya9/Open-data-projecy
Donkey Kong Game : https://github.com/anirudhdahiya9/donkeykong
Terminal Shell implemented in C : https://github.com/anirudhdahiya9/bash-shell
Cannon Ball Game (OpenGL GLFW) : https://github.com/anirudhdahiya9/2d-Cannon-Game
Github Profile for more

Interaction with the Mailman community :

I have been active with the GNU Mailman community for more than 2 months at the writing of this application, and feel confident about the codebase of Mailman core, Postorius and relevant portions in Hyperkitty. I have also been actively in touch with mentors on IRC. I am thankful for the warm support extended to me as a newbie into the open source community in the initial stages of my contribution.

Contributions and MR's :

A Filter for list index page -
https://gitlab.com/mailman/postorius/merge_requests/107

A fix to block duplicate subscription requests -
https://gitlab.com/mailman/mailman/merge_requests/114

Add already subscribed error to catch Mailman server exception to report meaningful error in Postorius -
https://gitlab.com/mailman/mailman/merge_requests/112

WORK SCHEDULE RELATED INFORMATION

Other work Commitments :
I do not have any work related commitments during the period.

Expected Work Hours :

I have 24/7 internet access and do not have any work commitments during the summer and thus , shall be able to devote at least 40 hours of work to the project per week. I shall usually be able to work between 12 pm to 12 am IST (UTC +5:30).

What days shall you be unable to meet the mentors?

Since I have no work related commitments, I shall usually be available all week, though I might take Sunday off at times with prior acknowledgement of the mentors and depending upon the workload and status of project. In case of unavailability, I can be contacted via email, instant messaging(Facebook) and the provided contact number.

Have I applied for any other project under any organisation ?

No.