Proposal for the development of a benchmarking suite and adding storage backends to OpenStack Marconi

Introduction:

Openstack Marconi aims to provide a HTTP REST based messaging API and service that helps ease development of a variety of distributed application messaging patterns in an efficient, scalable and highly-available manner and work seamlessly with the OpenStack product portfolio. The trend among scalable web architectures is to design the entire product stack on a single open protocol such as HTTP based REST. Scalable Database technologies such as CouchDB allow REST based control over the storage layer. Marconi unifies the queue management protocol and enables scaling along with the rest of the OpenStack deployments.

Marconi currently supports WSGI (http) at the transport layer, and MongoDB and SQLAlchemy at the storage layer. A SQLite driver is supported for development and release. This project will help expand the storage backend options by adding Redis and Openstack Swift to the existing stack. Redis would help users in need of faster queuing operations by providing a completely in-memory object store. OpenStack Swift would help better integration of Marconi into the community's product portfolio. The two data stores fall in line with the community's long term vision of being able to shard queue operations based on the priority requirement of the incoming message. The project also involves developing a benchmarking suite using autobench/tsung to measure the performance of Marconi under various scenarios of transport, storage backends, and sharding options. The project scope includes development, testing, and documentation of the storage backends along with benchmarking and comparison of various backends for Marconi.

Objectives of the Project:

1. Source code development of Marconi backend drivers on Redis using py-redis (https://pypi.python.org/pypi/redis/) and OpenStack swift using swiftclient (http://docs.openstack.org/developer/python-swiftclient/swiftclient.html) as the client libraries on new github projects marconi-redis and marconi-swift respectively.

2. Testing the backend against version 1.0 and 1.1 of the WSGI API.

3. Functional, unit testing and documentation of the project based on the community defined guidelines.

4. A benchmarking suite using autobench/tsung to measure the performance of common pub-sub and passive auditing of producer-consumer workflows in Marconi under configurable loads and produce understandable and archivable output as graphs and csv/json files.

<u>Implementation</u>:

The project implementation comprises of three major tasks which are elaborated in the further sections.

In summary it involves:

Task 1. Development, testing and documentation of Redis storage backend for Marconi.

Task 2. Development, testing and documentation of Openstack Swift storage backend for Marconi.

Task 3. Developing a standard benchmarking suite to measure and compare various usage scenarios of Marconi.

<u>Task 1:</u>
The implementation of the Redis storage backend is broken into the following subtasks:

- ➔ <u>Task</u> <u>1.1:</u> Create a new github project with title marconi-redis and list marconi as a dependency. Stub out code for the storage driver under folders redis of the marconi/queues/storage folder. The task also involves understanding the oslo.config project configuration structure and setting up the appropriate parameters for Redis.

- ➔ <u>Task</u> <u>1.2:</u> Develop Queue and Message Controllers for Redis. The transport layer (WSGI) v1.0 and v1.1 will be integrated to use the storage layer for queue and message management. Validate if the corresponding test cases are working fine.

- ➔ <u>Task</u> <u>1.3</u>: Develop Claims , Shards and Catalogue Controllers for Redis. The transport layer (WSGI) v1.0 and v1.1 will be integrated to use the storage layer for claims ,shards and Catalogue management. Validate if the corresponding test cases are working fine.

- ➔ <u>Task</u> <u>1.4</u>: Develop test cases specific to Redis storage backends. This can used to verify common functionality against different versions of Redis.

- ➔ <u>Task</u> <u>1.5</u>: Perform functional testing and write the documentation according to the community specified guidelines.

<u>Task 2:</u>
The implementation of the Openstack swift storage backend is broken into the following subtasks:

➔ <u>Task</u> <u>2.1:</u> Create a new github project with the title marconi-swift and list marconi as a dependency. Stub out code for the storage driver under folder swift of the marconi/queues/storage folder. The task also involves understanding the oslo.config project configuration structure and setting up the appropriate parameters for the Swift storage backend.

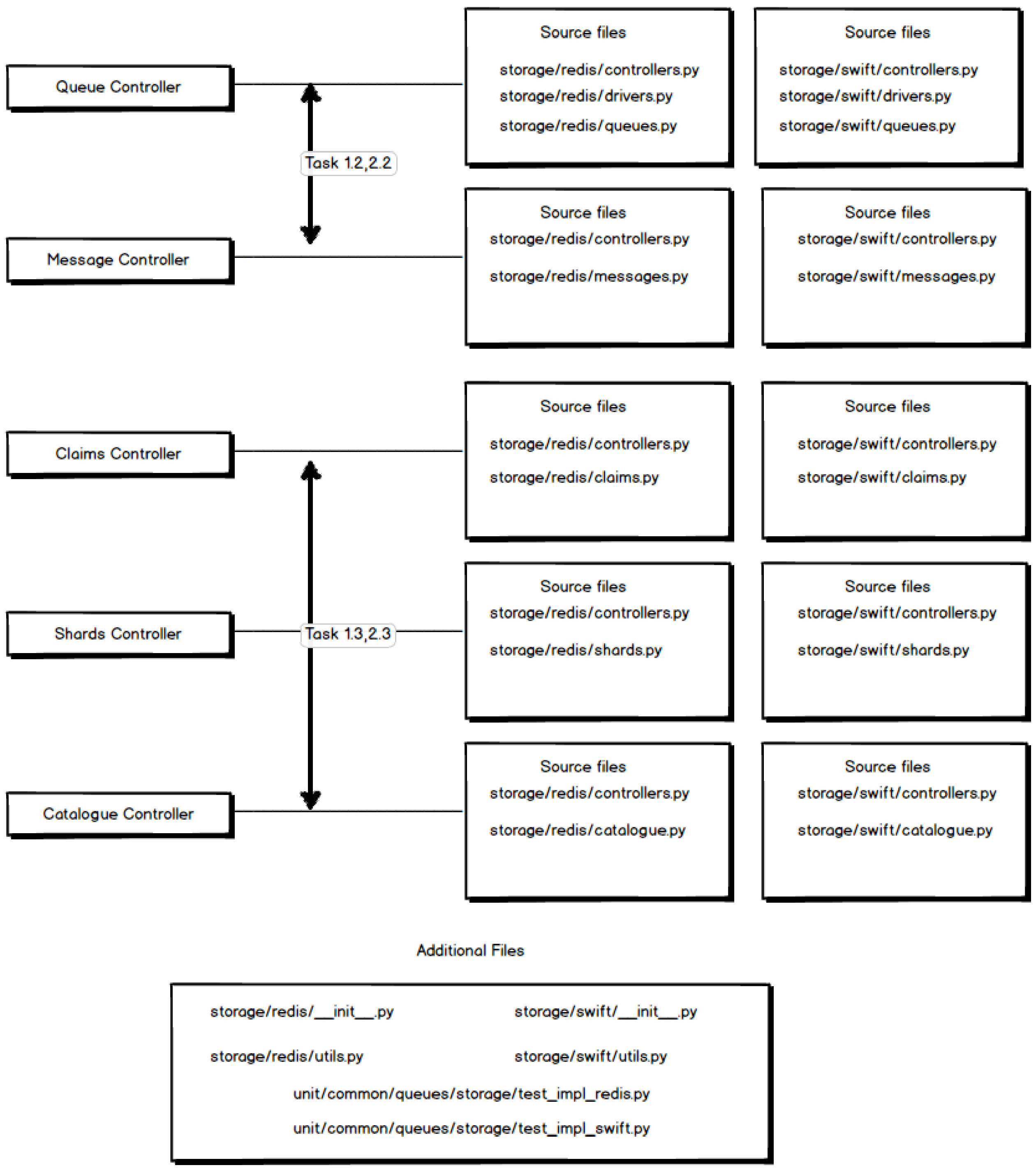
➔ <u>Task</u> <u>2.2:</u> Develop Queue and Message Controllers for Swift. The transport layer (WSGI) v1.0 and v1.1 will be integrated to use the storage layer for queue and message management. Validate if the corresponding test cases are working fine.

➔ <u>Task</u> <u>2.3</u>: Develop Claims , Shards and Catalogue Controllers for Swift. The transport layer (WSGI) v1.0 and v1.1 will be integrated to use the storage layer for claims ,shards and Catalogue management. Validate if the corresponding test cases are working fine.

➔ <u>Task</u> <u>2.4</u>: Develop test cases specific to the Swift storage backend. This can used to verify common functionality against different versions of Swift.

➔ <u>Task</u> <u>2.5</u>: Perform functional testing and write the documentation according to the community specified guidelines.

Feature - Task - Source code mapping:



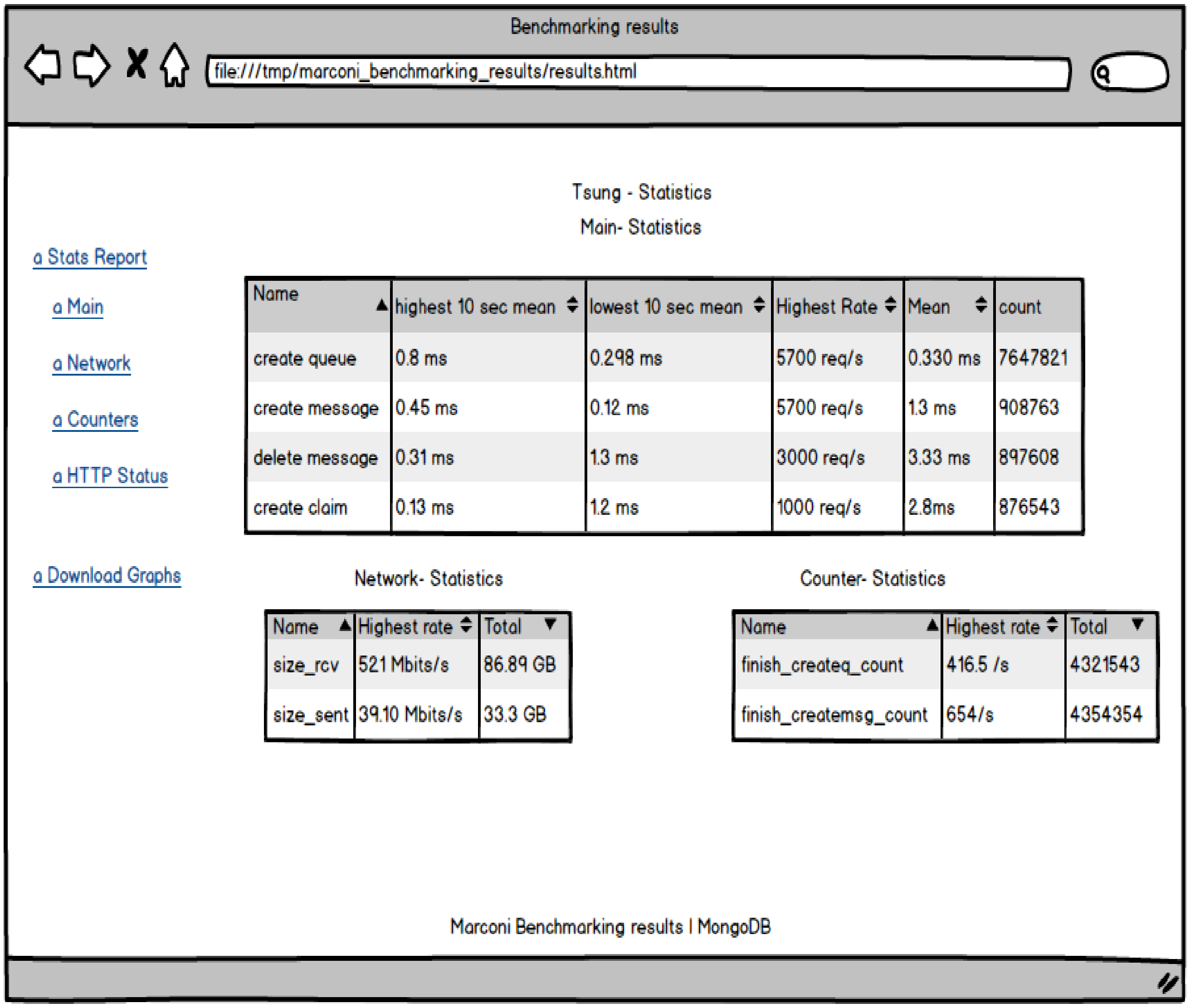
Queue Controller
Message Controller
Claims Controller
Shards Controller
Catalogue Controller
Task 1.2,2.2
Task 1.3,2.3
Source files
storage/redis/controllers.py
storage/redis/drivers.py
storage/redis/queues.py
Source files
storage/swift/controllers.py
storage/swift/drivers.py
storage/swift/queues.py
Source files
storage/redis/controllers.py
storage/redis/messages.py
Source files
storage/swift/controllers.py
storage/swift/messages.py
Source files
storage/redis/controllers.py
storage/redis/claims.py
Source files
storage/swift/controllers.py
storage/swift/claims.py
Source files
storage/redis/controllers.py
storage/redis/shards.py
Source files
storage/swift/controllers.py
storage/swift/shards.py
Source files
storage/redis/controllers.py
storage/redis/catalogue.py
Source files
storage/swift/controllers.py
storage/swift/catalogue.py
Additional Files
storage/redis/__init__.py
storage/swift/__init__.py
storage/redis/utils.py
storage/swift/utils.py
unit/common/queues/storage/test_impl_redis.py
unit/common/queues/storage/test_impl_swift.py

Fig 1 : Feature-Task-source file mapping.

Task 2:
The implementation of the benchmarking suite is broken down into the following subtasks:

➔ Task 2.1: Evaluate autobench, httpperf and tsung and choose one for setting up the benchmarks. Create an elaborate test plan for the evaluation of various pub-sub and passive auditing of producer-consumer workflows for all currently supported backends of Marconi.

➔ Task 2.2: Implement the benchmark tests cases from the test plan and extract reports for individual backends similar to the mockup shown below.



Tsung - Statistics

Main- Statistics

Name ▲	highest 10 sec mean ⇕	lowest 10 sec mean ⇕	Highest Rate ⇕	Mean ⇕	count
create queue	0.8 ms	0.298 ms	5700 req/s	0.330 ms	7647821
create message	0.45 ms	0.12 ms	5700 req/s	1.3 ms	908763
delete message	0.31 ms	1.3 ms	3000 req/s	3.33 ms	897608
create claim	0.13 ms	1.2 ms	1000 req/s	2.8ms	876543

Network- Statistics

Name ▲	Highest rate ⇕	Total ▼
size_rcv	521 Mbits/s	86.89 GB
size_sent	39.10 Mbits/s	33.3 GB

Counter- Statistics

Name ▲	Highest rate ⇕	Total ▼
finish_createq_count	416.5 /s	4321543
finish_createmsg_count	654/s	4354354

Marconi Benchmarking results | MongoDB

Fig 2 : Snapshot of marconi benchmarking results

➔ Task 2.3: Generate the benchmark comparison reports comparing different storage engines across API versions 1.0 and 1.1 . A mockup for one such report is shown below.

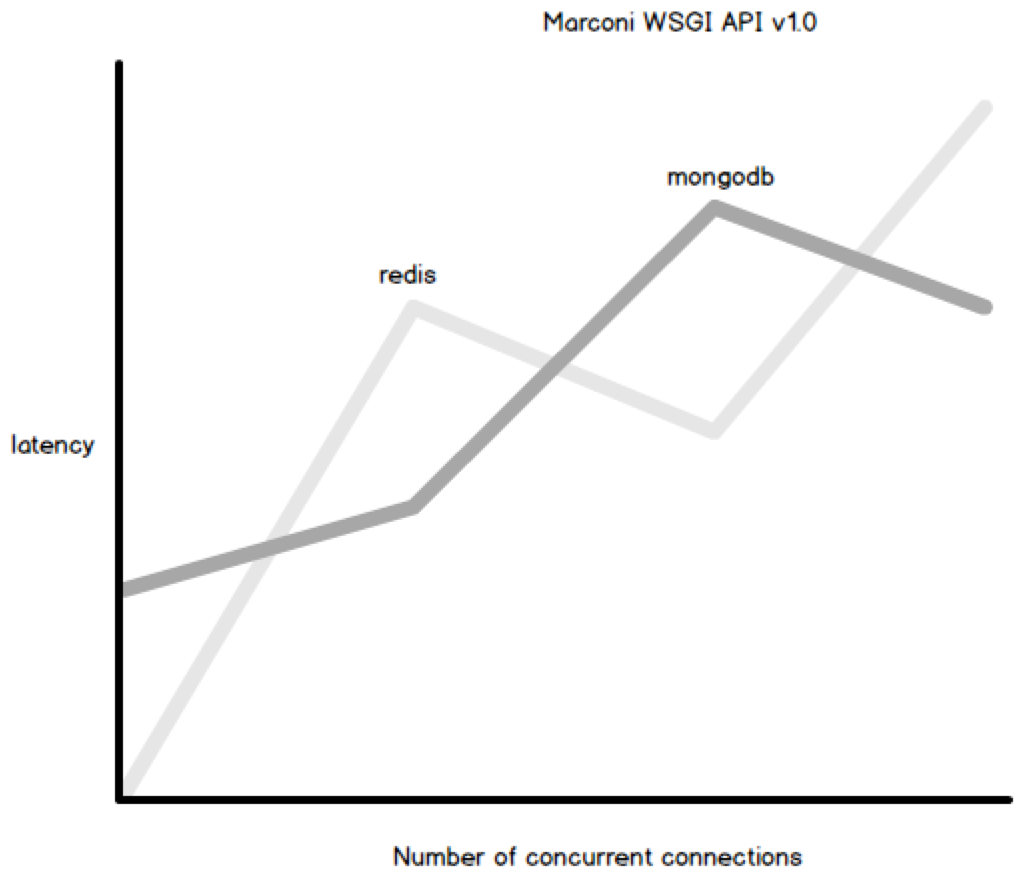


Fig 3: Performance comparison analysis of different storage backends

➔ Task 2.4: Automation and documentation of the entire process to enable single script benchmarking suite.

<u>Timeline:</u>

21st April - 4th May: Interact with the Marconi community and explore the marconi code base extensively. Implement task 1.1 & 2.1.

5th May - 12th May: Implementation of task 1.2.

13th May - 18th May : Implementation of task 2.2.

19th May - 26th May: Implementation of task 1.3.

27th May - 3rd June : Implementation of task 2.3.

4th June - 14th June: Testing features of task 1.2 & 1.3 against API versions 1.0 and 1.1

14th June - 25th June: Implementation of task 1.4 & task 2.4

27th June: Mid Term evaluation.

28th June - 7th July: Implementation of task 3.1 and create an elaborate test plan on the Openstack marconi wiki.

8th July - 20th July: Develop performance evaluation scripts and customize them to generate reports in line with the test plan. (Task 3.2)

21st July - 4th Aug: Performance evaluation and comparison of all storage backends for Marconi. Will get the results reviewed and add to the marconi wiki. (Task 3.3)

5th Aug - 11th Aug: Implementation of tasks 1.5 , 2.5 and 3.4.

11th Aug - 18th Aug: Code reviews and pending changes and final cleanup of the project source to ensure correctness.

22nd Aug: Final evaluation.

About me:

I am Prashanth Raghu and I am from Bangalore, India. I am currently based in Singapore and reading for my Masters by Computing at School of Computing at National University of Singapore. I completed my under graduation at PES Institute of Technology , Bangalore. My OpenStack personal page can be found at https://wiki.openstack.org/wiki/User:Prashanthr and my bug activity page can be found at https://review.openstack.org/#/dashboard/10812. I worked as a systems architect at Citrix R&D India for two years on REST based SOA services for supporting the software stack. During my undergraduation I worked on developing a scalable video sharing site developed on the OpenNebulla cloud orchestration platform using KVM , pylibvirt and Amazon EC2 at University of Applied Sciences(HEIG), Canton De Vaud, Switzerland.

My development skills are usually focussed towards python, javascript (client & server side) and Java. I contribute open content to wikipedia on the topics of Sports, Indian Politics , Indian culture , history and entertainment. My github profile can be found at: https://github.com/PrashanthRaghu. My mentor for this project will be Alejandro Cabrera and my project page can be found here.

Pre-Gsoc Preparation:
Used the pre gsoc time to learn about the general project structure of Open Stack on the Python Openstack and You guide https://github.com/cabrera/python-openstack-and-you. The first 2 weeks of March were used to learn python topics such as flask wsgi, py2 and py3 compatibility, itertools, metaclasses, functools, list comprehensions,generators, decorators among other python topics. I have also familiarized myself with the Gerrit code review process to submit patches to Openstack.

Community interaction:

Beginning from the end of February, I have been actively participating in the IRC channel and simultaneously begun exploration of the source code. I have submitted a patch for the bug (https://bugs.launchpad.net/marconi/+bug/1239834) the review for which can be found at https://review.openstack.org/#/c/80767/. I also was able to successfully reproduce the bug https://bugs.launchpad.net/marconi/+bug/1284995 for which I have provided the appropriate reproduction steps. I followed the test cases at all the layers of the stack and reported my findings to the community on the IRC channel. I shall continue working on these issues as a part of my GSOC preparation. I interacted with Alejandro Cabrera (also my mentor) and Fei Long Wang during the process.

I will try to make use of this opportunity to network with the entire community and share my results on the IRC channel meetings and the Marconi wiki at regular intervals to get timely feedback.

What the community can expect from me:

- ➔ An open source enthusiast with enterprise and open source development experience in developing scalable systems based on REST architectures and test driven development.
- ➔ A enthusiastic learner and an active contributor familiarized with openstack

community guidelines.

- ➔ On time delivery of project tasks and documentation.
- ➔ Clean code which adheres to community coding standards.
- ➔ Long term contribution to the community.

Experience with other python technologies:

- ➔ I have prior working experience with technologies such as OpenNebula, libvirt, libevent , django, memcached and their python libraries.

What I will learn during the summer:

- ➔ I will pick up additional skills on flask, pyredis and swiftclient , autobench and httperf/tsung during the course of the project.

My preferred methods of contact are through email (p.is.prashanth@gmail.com), Skype (prashanthraghu) and IRC (prashanthr_) on freenode.I microblog on tech, sports and politics regularly on Twitter with the handle @PrashanthRaghu.

I will be available full time during the entire duration of the project as I have no other commitments during the period.