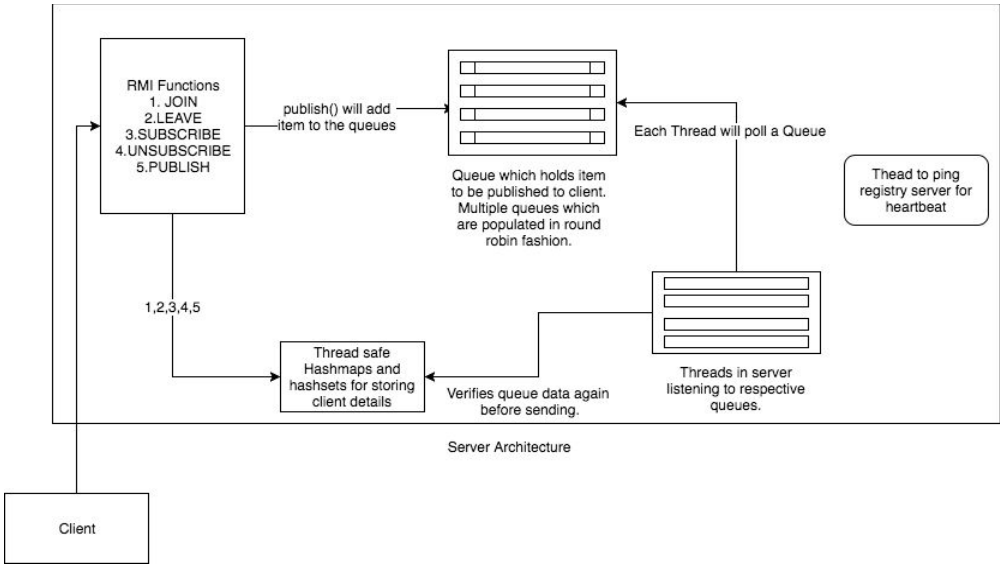
PubSub Using Java RMI

Implementation details:

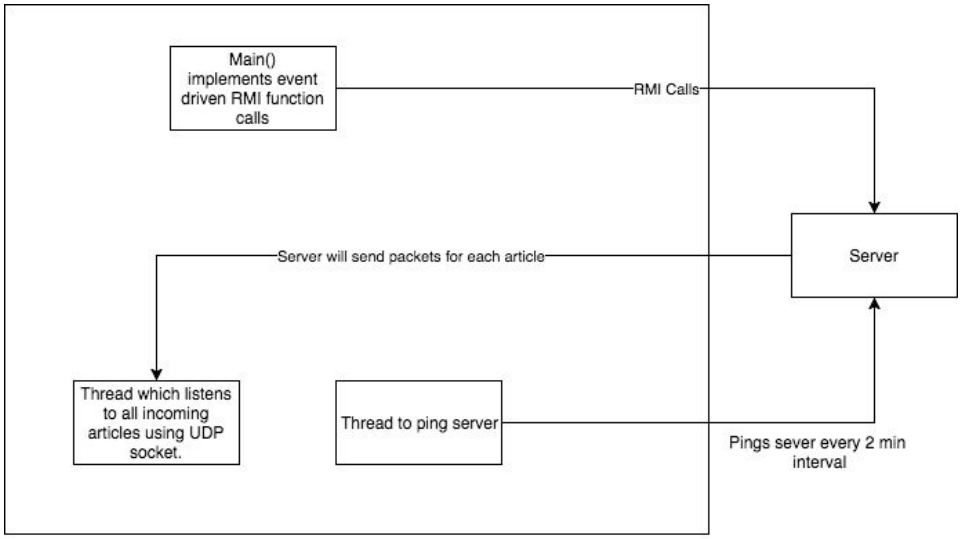
Server supports 5 RMI functions as shown in picture. We are using three hashmap<string,set<string>> (One for each type, originator, organization) for maintaining the user's subscription details. During publish, we perform SET OR operation to decide which all clients should get the published article. If an article is published as ;Sports;Someone;UMN;Article, then all the clients who have subscribed to Sports OR Someone OR UMN will get the message. Similarly we use OR operation while subscribing and unsubscribing.]



Click on the image to get high resolution picture

All functions except publish is constant time operations and therefore we are spawning thread only for publish operation. Publish depends on the number of subscribed clients and it may take a long time to broadcast these messages to the clients. For this reason, we have designed a queued architecture. On receiving a publish request from the client, we find out the clients who should get the messages and put them(Receiver address,article pair)in queues in a round robin fashion. We also have multiple threads who keep polling these queues. If any item is found in queue, thread will send the article to the intended receiver.

When the server starts, it connects to the registry server by doing a UDP call. Then it does a GetList and prints the list of server. We have dedicated thread to receive heartbeat message and keep sending those to the server. We have also tested the deregister functionality, but the code is currently commented out as we don't have a user flow supporting deregistering.



Client Architecture

Click on the image to get high resolution picture

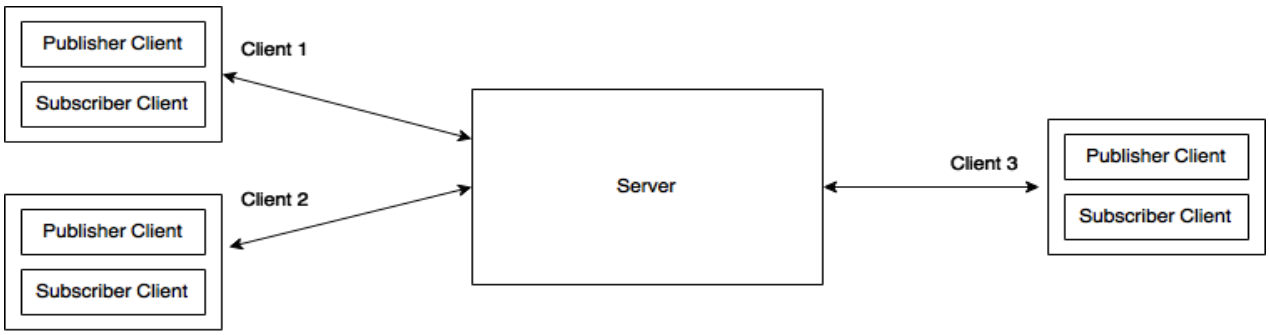
Client is an event driven command line application, which gives a menu that user can choose from (JOIN, LEAVE, PUBLISH, SUBSCRIBE, UNSUBSCRIBE). Based on the chosen event, the client calls appropriate RMI function in the server. After it joins the server it spawns two thread. One thread for receiving the incoming articles and another for pinging (ping() RMI) the server continuously. On ping failure, thread will print “Server Down Message”. Similarly, the other threads on receiving any article from the server prints it on command line.

Google PubSub

Implementation details:

As per the documentation for Google PubSub using python, client had to be either a publisher client or a subscriber client. In order to meet the requirements of our project, we have created a

publisher client object as well as subscriber client object for each of our clients so that the client can publish as well as subscribe to topics. Client makes a pull subscription to the server for a topic(say sports) and the server sends back messages related to that topic when some other client makes a publish to that topic. A subscription name has to be mentioned each time a subscription is made and the subscription name has to be unique across the project. Topic has to be created before we can subscribe or unsubscribe to that topic.



It was much simpler to implement Google pubsub compared to the pubsub system we created from scratch using Java RMI as we don't have to bother much about the communication between the server and the clients. We had some difficulties with the setup of the Google pubsub emulator. Once the setup was done, it was pretty easy to create clients which made calls to the emulator.

References :

- https://www.javatpoint.com/RMI
- https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/pubsub/cloud-client/publisher.py
- https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/pubsub/cloud-client/subscriber.py