Design Document - Project 1

Aditya Pakki, Tanmay Mehta

February 15, 2018

1 System Design

We try to provide a complete picture of our thought process in coming up with the design choices.

- **Work division and skill set:** The from Scratch PubSub (called fS-pubsub hereafter) was implemented in Java with the communication using UDP on RMI. Google based simple PubSub (g-pubsub hereafter) was implemented using GO as described in the spec sheet. Both of these were tested on the CSELab machines. Tanmay has a 8/10 proficiency in Java and 0/10 in Golang and hence implemented the client and server communication as well as data structures necessary to publish the articles. Aditya has 6/10 proficiency in Java and 5/10 in GoLang. He has implemented the Registry server - server communications & g-pubsub. This document was prepared by Aditya to compare and contrast the experiences from implementing and designing both the implementations.

- **Assumptions:** We faced a couple of problems which lead us to assume the following. (1) The heartbeat server on dio.cs is erratic. Listening for heatbeat on 5104 localhost was successful but there were problems with the 5105 production server. The project assumes a server deregistered with the registry server might still serve the client. Also connection to dio.cs.umn.edu seem to work only within the University network.(2) ”As per the Tanmay’s conversation with the TA, we got to know that for RMI calls a thread gets spawned for each remote client making the call. Hence we did not spawn any new thread except for dispatching the UDP packets in publish method” (3) There is no saving of the state. We followed a push based architecture design to let the subscribers know that an article has been published by the client.

  (4) Clients all follow a similar structure of registering with the server and performing a set of publishes and leave the server. This is to minimize the testing load and work on getting the basic infrastructure working. (5) Hardcoding of the ports and IP address was done at few places. The RMI registry is started on the server at port 5000. The registry server is listening on 5104/5105 as given by the project and the UDP port to communicate with the registry server is flexible and not hardcoded.

  (6) The g-pubsub project followed the document to the T. We implemented the simple API for client as requested. The client design could substantially be improved by storing the state of Client as well as the context in a struct instead of passing as arguments to all the functions. (7) A comparison of the g-pubsub and fS-pubsub may not be fair in our project as we had to learn the API by searching for examples on the Internet and then testing our code to make sure it doesn’t crash. The internet deserves a lot of credit for our g-pubsub implementation.

- **File Organization:** At the root directory are two folders - src containing the fS-pubsub code and SPubSub containing the simple client implementation of PubSub. The src file has a Makefile and the testable registry_server. The src files include

  i ArticleInfo.java - The three tuple information fields of identifying the article.

  ii Client.java - A default list of test statements which call the server functions.

  iii ClientPingThread.java - A Thread class which maintatins communication with the server

  iv ClientReceiveSubscription.java - A Thread class spawned by Client to pull the messages

  v DispatchArticles.java - A Thread class spawned by Server to publish articles to subscribers

  vi GroupServer.java - Remote Interface to be implemented by the GroupServerImpl.java

  vii GroupServerImpl.java - The heart of the project which contains all the functions of the server and client communication as well as Registry Server and server communication

  viii IP_and_Port.java - A set of IP and Port pair program used across the project

  ix StartServer.java - A stub to test the server implementation and start the process

  x /SPubSub/test.go - A go file which calls the client functions in simplepubsub package

  xi /SPubSub/simplepubsub/gcpsClient.go - A go client file containing client API definitions

2 Component API of (a) fS-pubsub and (b) g-pubsub

i **Method:** GroupServerImpl.java - boolean join(String ip, int port)
Functionality: Client calls to join a server

ii **Method:** GroupServerImpl.java - boolean leave(String ip, int port)
Functionality: Client calls to leave a server

iii **Method:** GroupServerImpl.java - boolean subscribe(String ip, int port, String article)
Functionality: Client subscribes to an article

iv **Method:** GroupServerImpl.java - boolean unsubscribe(String ip, int port, String article)
Functionality: Client unsubscribes from server regarding an article

v **Method:** GroupServerImpl.java - boolean publish(String article, String ip, int port)
Functionality: Server pushes the articles to subscribers

vi **Method:** GroupServerImpl.java - String ping()
Functionality: returns a constant Hi as a ping message.

vii **Method:** GroupServerImpl.java - void register(InetAddress rsHost, int rsPort)
Functionality: server registers with the registry server listening on rsPort

viii **Method:** GroupServerImpl.java - void deregister(InetAddress rsHost, int rsPort)
Functionality: server deregisters from the registry server listening on rsPort

ix **Method:** GroupServerImpl.java - void getList(InetAddress rsHost, int rsPort)
Functionality: servers calls getList asking registry server to send list of servers

x **Method:** GroupServerImpl.java - void listenHeartBeat()
Functionality: internal function which maintains a heartbeat with the registry server.

xi **Method:** gcpsclient.go - client Client func Join() (err error)
Functionality: New Client joins the server

xii **Method:** gcpsclient.go - func (client *Client)CreateTopic(topicName string) (topic *pubsub.Topic, err error)
Functionality: Client creates a new topic on the server if not available

xiii **Method:** gcpsclient.go - func (client *Client)Subscribe(topic *pubsub.Topic) (sub *pubsub.Subscription,err error)
Functionality: Client subscribes to a topic if it exists on server

xiv **Method:** gcpsclient.go - func (client *Client)Unsubscribe(topic *pubsub.Topic) (err error)
Functionality: Cilent unsubscribes from a topic if it exists on the server

xv **Method:** gcpsclient.go - func (client *Client)Publish(topic *pubsub.Topic, msg string) (err error)
Functionality: clients sends a message to a topic as a publisher

xvi **Method:** gcpsclient.go - func (client *Client)Consume(sub *pubsub.Subscription) (err error)
Functionality: Internal function to check if the published messages are sent to the subscribers

3 Comparison of fS-pubsub with g-pubsub implementation

In a rather skewed comparison between the two implementations - we agree that the correctness of the g-pubsub implementation was easy to verify. The time to implement the client after understanding the API was 5 hours for a single developer as compared to over 3 days for two of us to implement client - server communication in the fS-pubsub project. We found that abstracting the task of concurrency is obviated by the use of existing infrastructure. Using the g-pubsub interface and coding the API was easy, fun, and guaranteed to work. However, treating the g-pubsub API as a blackbox wouldn't help when the program fails and debugging the error is similar to finding a needle in the haystack.

4 How to test the programs

- **fS-pubsub:** In the src folder run javac *.java to compile all the classes. Connect to the registry server by calling registry_server_test in the window W1. Start the server in new window W2 by calling 'java StartServer' in a new terminal inside src folder.
  To run a single client: Run the client program using 'java Client ip client-port'. The IP argument is the IP of the server where our rmi-registry is running. The client-port argument can be any random port number. The client-port is the port where the client is listening for incoming UDP messages from the group-server.
  To run multi client: run 'java ClientDriver ip' where the ip argument is the same as mentioned above.

- **g-pubsub:** copy the contents inside the folder SPubSub to $GOPATH/src/. The test.go program should be at the same level as the simplepubsub interface.
  Start the server emulator in a new terminal.
  Use 'go build' command at the $GOPATH/src folder.
  Run the test program using 'go run test.go' and observe the results.