Distributed PUB/SUB Exercise Report

Javier Rodríguez, 16/07/2018

Summary

I have implemented a C# producer with a class that generates random numbers and publish them by redis in a JSON format. A Node.js server is subscribed to the same redis channel and get those random numbers. This server communicates with Angular UI by websockets, so TCP connection doesn't finalize, allowing angular receiving in a non-stopped way that random numbers generated by the producer.

Only one instance (with two axis) of producer is invoked, so all the clients will have the same data. There has not been implemented any secure form, allowing any client to subscribe or publish in the channel. There has not been implemented any automated test.

I am aware about some bugs that it has when multiple clients connect and use the app at the same time, but I don't want to delay me more sending the task.

System

As it actually is my first C# project, I decided use Visual Studio as IDE, so I use Windows 10 as OS. Due Redis has no official support to Windows, I have use ubuntu bash for windows in order to install redis-server and use it, that in my opinion is more comfortable than third party supports. I have chosen JSON format for all messages.

Producer

The C# app has a class that implement the number generator and the main class.

The **number generator class** has as properties configurable variables (min and max range and an extra for time elapsed between generations), a name for console logging purposes, redis subscriber that should be pass by constructor and finally a cancellation token source and a status string for threading purposes.

The class has also three methods:

Start Method: If status is STOPPED, then it creates a new task (with a cancellation token) that will generate and send to redis channel numbers each time given by property Time until the token has a cancellation request. It starts that task and change the status property to STARTED.

Stop Method: If status is STARTED, then it requests a cancellation to the token. It also changes the status property to STOPPED.

Update Method: It update all configuration properties of the class given by a Dictionary, that will be the result of transform an incoming JSON.

The **main class** manage communications and the console logging. It creates the connection with redis server, the both axis instances of the number generator class, and subscribes to the channel of redis. Depending on the action type of the received message, it will do one or another action (update, start, stop or send information about any of the two axis).

Redis

I have use to channels to separate messages for the server and messages for the client. The producer subscribes to “ServerMessages” channel and pusblish in the “ClientMessages”, and the web server does in the other way.

All messages has JSON format, and all of them have an “action” property, that tells the controller (angular UI or C# producer) what is the purpose of the message.

WebServer

I have implemented express.js for the web server. This server is an intermediary between Angular UI and redis, and in order to maintain the real-time behavior, I have decided to implement websockets for the communication between this server and Angular UI. Redis, socket.io and express.js dependencies are very light, so this server is light too (~6MB with node_modules). It’s only function is publish on redis “ServerMessages” channel what it receives from websockets and send by websockets all received data from redis subscription on “ClientMessages”.

Angular UI

This is the heaviest part of the system, because angular has a lot of develop dependencies but the final dist folder is light too. I have added to these dependencies, socket.io for websockets, bootstrap for css and d3js for chart. I have use the angular cli tool to create the project, two components (one for each view) and two services (for websockets usage). This is probably the part that more time has needed, probably because I have never used before angular2+, typescript, and d3js charts. Plus, I had to integrate websockets on angular too.

Docker

I have use docker to deploy the application. I had to make some changes to server code because I didn’t configure some environment variables (just redis host in producer and webserver), but it could be easier fix that for future versions.

Deployed version is at http://itq-randomgraph.mooo.com/