Edwin Garcia

Lewis University

Distributed Computing Systems

Week 6 Assignment 1 – Distributed Processing Using Messages

CPSC-55500-02-FA16

Manoj Bhat

**Integrating ZeroMQ Distributed Messaging and Apache Avro RPC**

This is a brief narrative of the steps performed to integrate an Apache Avro application and perform messaging patterns based on ZeroMQ. This project is a derivative of Week 2 Assignment where an AVRO client/server application was created. This project expands into 2 Linux (Ubuntu 16.x) VM nodes where nodes perform either publish/subscribe and request/reply patterns (1.1).

|  |  |
| --- | --- |
| 1.1 Ubuntu Nodes Running | 1.2 Application files uploaded in the nodes |
| Macintosh HD:Users:engarcia:Dropbox:Screenshots:Screenshot 2016-10-12 00.20.24.png | Macintosh HD:Users:engarcia:Dropbox:Screenshots:Screenshot 2016-10-12 00.19.58.png |

The steps to install ZeroMQ is as follows:

1. sudo apt-get install cmake python-setuptools python-dev build-essential
2. sudo easy\_install pip
3. git clone https://github.com/zeromq/libzmq
4. cd libzmq
5. mkdir cmake-build && cd cmake-build

Once the nodes are created, the following files are uploaded (1.2):

1. avro-gradeserver.py – Contains a publish socket and a request socket
2. avro-gradeclient.py – Contains a subscribe socket
3. student\_write.py – Contains the AVRO application
4. student\_schema.avsc – AVRO schema file

The node, zmq, is designated as the server, with IP address 10.0.0.12. zmq will host the AVRO schema and data files as well as the zmq file, avro-gradeserver.py.

Avro-gradeserver.py is an application with 2 sockets and performs 2 tasks:

1. Port 5560 – Publishes to subscribers when a new student is added.
2. Port 5570 – Requests data when a student is created from student\_write.py application.

Avro-gradeclient.py contains a subscribe socket listening on port 5560.

**Operation**

To operate the messaging patterns, all applications must be executed in the shell. This is done by using ssh to the 2 nodes (2.1). student\_write.py and avro-gradeserver.py resides in node with IP address 10.0.0.12. avro-gradeclient.py resides in 10.0.0.17:

1. python ./app/py/student\_write.py – runs the AVRO application
2. python ./app/py/avro-gradeserver.py {2} – runs the PUB and REQ sockets. An optional argument sets the sleep interval in seconds.
3. python ./app/py/avro-gradeclient.py {Math} - runs the SUB socket. An optional argument sets the filter for the PUB/SUB pattern.

We create 2 instances of the subscriber application (avro-gradeclient.py) and set the arguments to “Hiking” and “Biking”. The purpose of these two different instance is to show how the subscriber sockets filter published messages (2.3).

|  |  |  |
| --- | --- | --- |
| 2.1 Initial command line | | |
| Macintosh HD:Users:engarcia:Dropbox:Screenshots:Screenshot 2016-10-12 00.13.19.png | | |
| 2.2 AVRO application menu | | 2.3 2 instances of avro-gradeclient.py with different filter arguments |
| Macintosh HD:Users:engarcia:Dropbox:Screenshots:Screenshot 2016-10-12 00.14.14.png | Macintosh HD:Users:engarcia:Dropbox:Screenshots:Screenshot 2016-10-12 00.14.40.png | |

We then create a student item using menu 2 in the AVRO application(3.1). Here we used “Hiking” as the course. This will trigger 2 actions:

1. Sent to the avro-gradeserver.py as a REP. Then the avro-gradeserver.py
2. Avro-gradeserver.py sends a PUB to the subscribers.
3. One of the SUB application will catch the published message from Avro-gradeserver.py

|  |
| --- |
| 3.1 Create New Student |
| Macintosh HD:Users:engarcia:Dropbox:Screenshots:Screenshot 2016-10-12 00.15.47.png |
| 3.2 Creating a message with course:Hiking |
| Macintosh HD:Users:engarcia:Dropbox:Screenshots:Screenshot 2016-10-12 00.16.02.png |

We can now see that the avro-gradeserver.py received the student object with course: Biking. It also published the message and the avro-gradeclient.py with “Biking” filter caught the message. The other avro-gradeclient.py filtered it out and did not log the student object (4.1).

|  |
| --- |
| 4.1 Creating a message with course:Biking |
| Macintosh HD:Users:engarcia:Dropbox:Screenshots:Screenshot 2016-10-12 00.17.14.png |

**Conclusion**

ZeroMQ is an eye opener for me. I ran into some errors that took me sometime to figure out. The key is to understand the basic concepts, particularly the way the patterns work. There are details in how the patterns work and you have to follow these concepts for it to work. Unfortunately, I did not find a viable debugger tool for the purpose. It is something I will need to explore.

I have also installed Apache ActiveMQ. It comes with a functional example that runs on node server. I elected to experiment with ZeroMQ as I feel it allows me to fully understand the concepts of how it works. ActiveMQ seems to wrap things into easier concepts but I prefer learning it the harder way. However, ZeroMQ is not a complicated concept. It is just something I am not familiar with and requires a little bit of learning curve.

I can see a few practical applications to ZeroMQ down the line. In the interim, I will continue to evaluate ZeroMQ and keep this in my toolbox.