Project Report

Xiao Yan, *UCLA*

1.Introduction

Python version: Enthought Canopy Python 2.7.11
Twisted version: 16.5.0

In this assignment, I implemented an event-driven networking framework twisted to be the application server. Twisted is engineered in Python and licensed under the open source MIT license. Twisted runs on both Python 2 and Python 3[1].

2.Design and Implementation

By using twisted package ClientFactory, Factory and Protocol, I implemented both server code and the client code in the same project.py. The server code has these main callback function:

In project.py:

For the server code:

connectionLost: If the server lost connection with the client, the server will log the error message into a server_id.log file

dataReceived: the core callback function of the server. This function will preprocess and split the message the server received.

If the message starts with "**IAMAT**", it will get the sent time, calculate the timediff by calling time.time(), added the timediff to the message, and stored the whole message in a global dictionary "locations", with the name of the query being the key. When storing the information, the function will check to see if the time of the incoming message is later than the message stored (that indicates that the incoming message contains the newest information). If a newer message comes, the server will broadcast the message to the nearby servers.

If the message starts with "**AT**", that means this message comes from other servers. If the sent time within the message is larger than the sent time stored in "locations" of this server, this function will update the "locations", and broadcast the message to the nearby servers (start the flooding algorithm). Otherwise, it will do nothing (stop the flooding algorithm).

If the message starts with "**WHATSAT**", that indicates the client is querying about the location, then the server will respond with the information stored in "locations". However, if the information was not found in "locations", the server will log an error message.

Besides the information, the server will connect to Google Places API to perform a nearby place search. And the data returned from Google Places is processed and sent back to the client.

If the message is not started with those mentioned above, or the length of the message is not as the spec says, or there are additional errors caused by the message, the server will return the message to the client staring with "? " and log the error message to the log file.

For the client code:

connectionMade: Once the client connected to the nearby servers, the client will send a message to the nearby servers. The intra-server message is stored in a global variable "intramsg".

connectionLost: If the connection is lost, the client will also log an error message to the same log file: server_id.log

For conf.py:

WESITE: the website for nearby search in Google Places API.
API_KEY: the key to use Google Places API. Based on the instructions on Piazza, I deleted the actually key and replaced with " ".
PROJ_TAG: set to "Fall 2016"
PORT_NUM: the port number that I used when testing the prototype code on my computer.
PARTNER: a dictionary that stores the nearby server relations. The key is the name/id of the server, while the value is a list of the port numbers for that server.

Running the code:

Make sure project.py and conf.py are in the same folder.

In different terminal sessions, type: python project.py server_id

For the client, in another terminal session, type: telnet localhost server_port

In the telnet session, one can communicate with server by:

Adding location information:
```
IAMAT kiwi.cs.ucla.edu +34.068930-
118.445127 1479413884.392014450
```

Query for information about places near other clients' locations:
```
WHATSAT kiwi.cs.ucla.edu 10 5
```

3. Discussion

3.1 Type checking:

Python uses dynamic type checking which gives more flexibility in passing the arguments/variables. On the other hand, static type checking requires the preassignment of variables.

In this situation, Python's dynamic type checking offers us more convenience because we are not quite certain the type of the data that will be passed between the servers.

3.2 Twisted vs. Node.js:

Both Twisted and Node.js are using the asynchronous event-driven network engine. Both of them are establishing an event loop and use callbacks to trigger actions when events happen. The APIs to use Twisted and Node.js are very similar. In my implementation, it should be easy to use either Twisted or Node.js.

However, there are certain differences between Twisted and Node.js:

Node.js presents an event loop as a runtime construct instead of as a library. That saves the time of using a blocking call to start the event loop and hide the event from the user. That design saves time and prevent unnecessary issues caused by the mistake from user. However, it also gives less adaptability to the user.

Node.js has not introduced the concept of objected oriented in the language. Twisted, implemented in Python, is using the benefit of OOP design and the change of the code is very efficient. On the other side, based on my understanding, it is a little difficult to apply the OOP in Node.js.

3.3 flooding algorithm:

In this application, one critical issue is the implementation of the flooding algorithm to synchronize between different servers. The algorithm I am using is that, upon receiving a new message, the server will broadcast to the adjacent servers (nodes that connected to this node). If a server receives the broadcast from another node, it will first check the message received with the message it stored. If the messages are the same, that means this node has been flooded, then it will not broadcast. However, if the messages are different, the server will update the message, and broadcast to its adjacent servers. The flood will stop when all the servers in the network has the same value.

There is a potential problem in my implementation. Since each server talks to two other servers and they form a closed circle, with this implementation, each message will be received twice in each server (as can be read in the log files attached in project.tgz). One comes from when the server received the message, and other from one of its adjacent nodes. This is not necessary and can be avoided by finding the server_id/address that sent the message originally, and prevent broadcasting to that server. Due to time issue, I have not implemented code for that purpose in my project.py.

3.4 Multi-threading vs asynchronous programming

In the situation of server implementation, asynchronous programming has greater advantage over multi-threading.

Multi-threading indicates that more than one thread will work on the same process and on the same set of data. It increases the chances that more than one thread will be in the race conditions. In that situation, we will need to implement locks to protect the integrity of data change. While one thread is accessing the core data (the “locations” library in our case), other threads will need to wait. That increases the time to run the program. Besides, more threads increase the chance of generating dead locks, which will greatly decrease the efficiency.

In the case of asynchronous programming, there will be only one thread running so there will be no race conditions and no lock is needed.

3.5 Memory management

In Python (if it is CPython), the garbage collection uses the reference counting mechanism in which if no more references pointing to the object the memory will be recycled[2].

While in Java, the memory heap is divided into old generation and young generation, the garbage collector first marks the unused objects, and then remove all the unused objects and reclaim the free space. Finally all survived objects will be moved together to save space[3].

In our situation where objects are frequently destroyed and constructed, the Python model is better because it Java does not have a free() function, so overtime, some of the unused objects will stay in the old generation and it takes longer time to clean these objects.

4. Conclusion

Overall, I believe twisted-based server implementation is a feasible and convenient method for server application and I generally recommend using this method.

5. Reference

[1] Node.js website https://nodejs.org/en/about/
[2] Python documentation http://deeplearning.net/software/theano/tutorial/python-memory-management.html
[3] Java memory management http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java