**Design of Twisted Places Proxy Herd using Twisted in Python**

Bolun Hu

*University of California, Los Angeles*

# Abstract

In this paper I will discuss the feasibility of using Twisted, a web framework written in Python to implement a proxy herd. I will discuss the implementation of my server in detail, and will also compare Twisted with Node.js, Python with Java. In the end it turns out that Twisted is a suitable choice of implementing proxy herd with the advantage of better scalability and easier to build prototypes.

# Introduction

The framework I used in this project is called Twisted, which is a very robust event-driven network programming framework written in Python and licensed under MIT License. Twisted supports a wide range of network protocols, and in this project my prototypes use TCP to establish connections between servers and clients. My prototype code will simulate a multi-server behavior. When messages are sent to servers from client, servers will respond with proper format and at the same time “flood” the message to the neighborhood using TCP connection. For the response message of WHATSAT, I used the concept of deferreds: I first get the deferred object using getPage method, then I create a callback function named “print\_and\_stop” to print the response message to screen once the deferred object has the receives the data successfully.

# Implementation

In my implementation, all servers are assigned with a unique port number, and are initialized by the TCPServer constructor with proper parameters including port number and server factory. Therefore increasing and decreasing the number of server is easy.

Also, when a server wants to connect and share information with another server using flooding, it instantly establishes a temporary “client” to send the message containing the information. Compared with the method of using a constant connection, it saves resources and reduce the fail rate. The temporary client will cease to exist after the message is sent to the other side of the connection. So in the log file you will notice that every time a server tries to flood the AT message to its neighbors, those neighbor severs will receive client connection like “Bolden: Got new client: ('127.0.0.1', 45321)”; after the AT message is transferred, the temporary connection will terminate and neighbors will lost a client connection: “Bolden: Lost a client: ('127.0.0.1', 45321)”.

In my implementation, there are 3 message handlers for dealing with incoming messages, which are handle\_AT, handle\_WHATSAT, and handle\_IAMAT. The first one is used for handling AT messages between servers. Second one is used for searching nearby places around a certain client and the third one is used for receiving client location message. The response message of query WHATSAT is exactly in the same format as the Google Places original response message, with nice white spaces and newlines, making the response very human readable. I used Twisted’s getPage method to first obtain a deferred object that contain the response data; once the data is received by deferred object, it immediately call the callback function “print\_and\_stop” to print out the response string. However, my implementation is not perfect in the way that the response message will give client all places near a certain location, instead of a number limit specified in the command line. The reason I decided to print out everything to the screen and not to set up the limit is that, parsing the original nice formatted JSON response and print out only part of the results will make the output poorly formatted and hard for human to read. Thuy told me that a nice formatted response will be a very good feature. Therefore I decided to print out the original nice formatted string instead of parsing it and give out a hard-to-read response.

Flooding mechanism is a key part of my implementation, and I make it by creating a function called “flood\_update” with a simple idea: every time a server gets a new client side update, it will first check in its own database (user pool) to see if an update is necessary. If it’s not necessary, server simply stop. If update is necessary, which means the client’s message time is later than what is stored in the database, then server will send a response message first, then update its own database and finally flood the AT message to its neighbors. Once its neighbor servers receive the AT message, they will do the same checking procedure and act accordingly. Thus in my implementation servers will not flood forever.

The log file record.log is constantly updated whenever something happens with any one of the servers. I used logging module to write the log file. The format of the log file is easy to read: every line starts with the server name, then followed by information about the current action (got new client/lost client/forwarding messages to other servers, etc.). The records are ordered from first line to last line.

To run my server, please type “twistd -n -y server.py” to initiate servers; then you can connect multiple clients to the ports 12195 – 12199 on new sessions, and then type commands like “IAMAT kiwi.cs.ucla.edu +34.068930-118.445127 1400794645.392014450

”, “WHATSAT kiwi.cs.ucla.edu 10 5

” to see the responds.

# Python Pros & Cons

One of the most distinct features that Python has is duck typing. It provides programmers with benefits as well as potential problems. The good thing about duck typing is that it makes programmer’s coding faster and easier. Without defining the types of variables, programmers will write code more efficiently and worry less about the types of each variable; Python will infer the type of objects. However, the duck typing and dynamic property of Python will also introduce potentially more run time errors than static languages like Java. Also, the debugging process will also be more painful and time-consuming because unlike Java programmers who can easily get the variable types from the code directly, Python programmers will have to go back to the code base and understand the code more deeply to fix the type errors. In short, Python’s dynamic feature provide programmers with more flexibility and less difficulties in coding, but Java’s static type checking will on the other hand gives a better protection from generating errors and buggy code.

Another feature of Python is the deferred evaluation. Such feature is good in the way that it may speed up the programs and run computation only when needed. But such feature will also be more likely to introduce buggy programs because errors within some branches of instruction will only be detected when they are actually executed. Thus it will make some bugs less obvious to be noticed.

The memory management policy of Python may be a disadvantage here. Since Python uses reference counting as its mechanism for managing memory, it puts some burden on the user’s shoulder to reallocate memory and avoid unwanted situations like cyclic reference. Also, the reference counting method will not return memory to the OS; instead it will hold the memory for the purpose of future reuses. Such policy will potentially cause memory leaks. However, for the specific case like our current project, considering the scale of our program, I think memory management should not be an issue since it is quite cheap. Moreover, since in the Wikimedia type of application the real time garbage collection is not a necessity, I think both Java and Python will be sufficiently good in memory management.

Multithreading is another issue here. Python supports multithreading, but multithreading in Python does not always give a good performance gain. The main reason is that Python, CPython to be more specific, normally would only use single thread for processing tasks. Moreover, in the multithreading scenario, the “global interpreter lock” introduced in lecture, which is used by Python when dealing with multithreading memory management will also introduce a large overhead and weaken the performance, making the multithreading hardly to be achieved. In short, multithreading will only be a real benefit when dealing with asynchronous operations and with multi-core provided. In this way, Java would probably be a more suitable choice than Python based on its more powerful multithreading support.

# Twisted Suitability

Overall, Twisted is a very good candidate for implementing server herd. In the following paragraphs I will discuss the reason why it’s suitable for such type of application.

The first advantage of Twisted is the easiness in coding. Since the Twisted inherits most of the features of Python, including the dynamic type checking, during code I did not have to worry too much about the type definition. Also the dynamic type property makes it a lot easier for me to parse the command and saves me from writing extra lines of code.

Single-threadness is another issue. Twisted runs in single thread in most cases, thus making issues such as concurrency and data races easier to deal with. The alternative, running servers in multiple machines or processes will gives a better performance than single thread approach, especially for CPU intensive tasks. Also, multi-thread code will be more likely to cause concurrency problems including locking, data synchronization, and data redundancy. In my prototypes, data synchronization is still a problem because it is very hard to make all the messages updated atomically. One server may be responding the query about a certain client while another server trying to flooding a new update on that client, thus data inconsistency is still possible.

# Twisted vs. Node.js

Twisted and Node.js are both event driven frameworks. They are both heavily asynchronous as well. But there are still some differences between these two. Twisted supports a wide range of network protocols, but Node.js mainly supports HTTP. Therefore Twisted may be a better choice for developing applications that runs on multiple protocols and cross-platform environment. On the other hand, Node.js runs faster than Twisted. The main reason may be its concentration on lower level implementation. Thus Node.js definitely should be a qualified substitute for Twisted. Actually in recent years Node.js is much more popular than Twisted. I assume the reason may be the rising popularity of Javascript and better asynchronous IO performance.

# Reference

[1]http://docs.Python.org/2/reference/datamodel.html [2]http://en.wikipedia.org/wiki/Wikimedia\_Foundation

[3]http://twistedmatrix.com/documents/13.0.0/core/howto/defer.html

[4]http://twistedmatrix.com/documents/current/core/howto/servers.html

[5] http://twistedmatrix.com/trac/