

Twisted Places proxy Herd

Pedro Borges, UCLA

Abstract

This document explains the methods used to build a prototype of a TCP server using Python's Twisted with the idea of verifying whether it is suitable or not to a larger application. We provide the structure of the prototype, the lessons seen from it and make a comparison with Java and Node.js.

1. Introduction

The application server herd is our target application. In this application, multiple servers communicate between each other with the purpose of distributing the information without the need to access the database. The input of data for the servers comes from mobile clients. Specifically a user submit a position to a server using its GPS and might query for some information. When the client submit this information for the server, then the server avoids unnecessary accesses to the database and propagates the information to its connections using flooding.

In order to test the idea of this application, we use a framework called Twisted. It provides event-driven APIs such that IO interaction can be performed asynchronously to allow rapid processing and forwarding of information.

2. Implementation

This section describes the implementation of the prototype application

Receiving input

Each command was parsed first using regular expression. That already limits several possibilities of erroneous input. Then depending on the command (e.g. IAMAT, WHATSAT), each field of it is casted to a type and the bounds are assured. For instance radius limited from 0 to 50Km are tested.

Distributing information

The distribution of information is done by flooding. That way it is implemented is that, when a server receives a piece of information that is newer from what it currently has, it distributes to all its ports that are not from where it received the information.

Examples

In order to run the server, it is important that your Google Places API key is in the file called “config.py” under the variable “GOOGLE_KEY”.

Figures 1,2,3 and 4 represent respectively how to start the prototype, how to execute IAMAT, WHATSAT and the behavior for undefined or malformed commands.

On the figures, the terminal window on the left is the server and the terminal window on the right is the client.

In the figures we can also see the log for the server Bolden and how it reacts to the clients requests. All the logs are stored under the “log” folder.

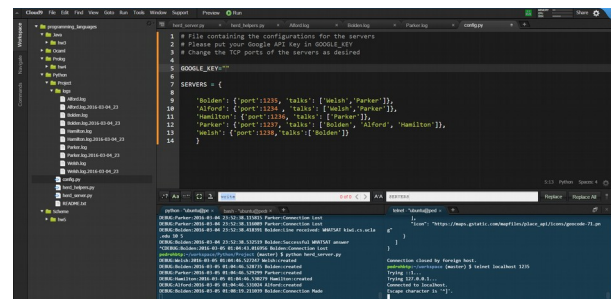


Figure 1: Starting the servers and communicating

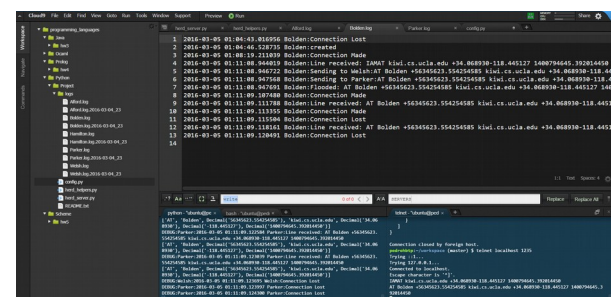


Figure 2: Behavior to an IAMAT command

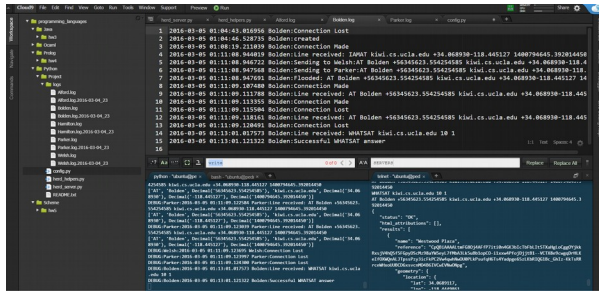


Figure 3: Behavior to a WHATSAT command

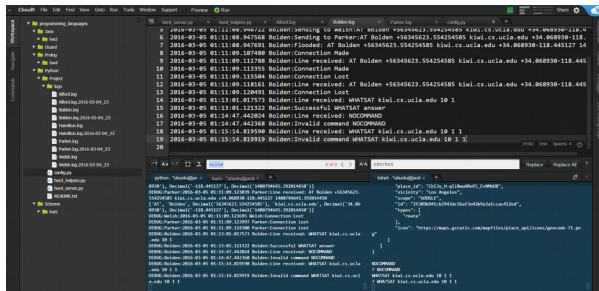


Figure 4: Behavior to an invalid command

3. Comparisons

We compare here twisted against two languages: Java and Node.

Twisted takes a very different approach with relation to multithreads in comparison to what normal Java does. Java has a really good support for synchronization of threads however, it is still programmer's duty to avoid race conditions. Twisted, on the other had, does not worry the programmer with the problem of synchronizing threads that want to access a same region. It instead uses only one thread that is responsible for the event loop. Meaning that IO calls are performed asynchronously and when they need to be processed by Twisted again, they are put into a queue so that the only thread that is active can process them in order.

The difference between Node.js and Twisted is more subtle. In Node.js' website it is said that it has the influence of other systems and Python's Twisted is among these influences. A key difference between them is the way they see the event loop. For twisted, you start your event loop by calling reactor.run(). That is because Python is not built with an event loop by default. Therefore Twisted goes around it implementing as a library. However, in Node.js, the event loop is built in the language by construction. You do not need to call any special method for it to take part on the code. By de-

fault, all IO interaction is already performed asynchronously.

3. Challenges

The greatest challenge in the implementation of the prototype was to get use to Twist's syntax. It escapes from orthodox python. It demands that you build specific classes with predefined methods so the server can take the proper actions in determined events. Specially when you have to create a set of servers with different attributes it might get tricky.

Another challenge was get over type conversion problems. For instance, converting the number "1400794645.392014450" might give problems if not performed with care. That means that if you are converting it to a string, you need to make sure there is no rounding and the string consider the number as a whole. If you are doing the other way around, which is converting its string representation to a float, you have also to make sure that there is no rounding. These problems are difficult to catch and require testing. To go around this problem, I decided to you the class called Decimal which did not present any conversion problems. For instance `str(decimal.Decimal('1400794645.392014450'))` returns the correct value.

4. Conclusions

We implemented an application that receives requests from clients, performs requests requests to Google Places and distributes information within servers. Python's Twisted proved to be a suitable framework for a larger project.

Python's dynamic type checking makes it really convenient to rapidly develop programs without bothering too much the programmer, but you also have to have certain caution with it so it does not bite back.

Python takes care of all memory management and twisted eliminates the necessity to deal with multithreaded code. Dealing with event loop frees the mind of the programmer by not having to worry about race conditions and thread synchronization.

As a final remark, Twisted can be used for the project. However, Node.js seems more natural for this kind of application. The way the event loop is introduced in Twisted and all the classes you have to build seems somewhat not very pythonic. Node.js has its event loop built-in the language and you don't have to write any big support classes to make it work. Then if possible, I would use Node.