CS 131 Project. Twisted Places Proxy Herd

Nilesh Gupta, 604489201
ngup@g.ucla.edu

Abstract

The goal of this project was to investigate the use of Twisted, an event-driven networking framework written for Python, to prototype an application server herd which consisted of 5 servers for the purposes of the assignment. Each server receives location updates and queries and is expected to efficiently relay this data to the rest of the network within a couple interserver transmissions. In this report, we look at the design of the simple network, the issues encountered with its implementation, and finally the effectiveness of the Twisted framework in handling the application server herd.

1. Introduction

Twisted is an event-driven networking engine written in Python under the MIT license. The framework runs on Python 2 with a considerable subset workable in Python 3 [1]. As opposed to the more traditional LAMP platform, Twisted offers a more flexible system applicable to multiple protocols. In addition, because of the event-driven asynchronous nature of the engine, Twisted provides much faster performance as it is less susceptible to bottlenecks.

By providing the base *ClientFactory*, *ServerFactory*, and *LineReceiver* classes as well as the *connectionMade* and *lineReceieved* protocols, Twisted allows prototyping of user's custom application needs. Much of the dirty work of implementing an event-driven network is hidden from the user since the built-in reactor manages all servers to this extent. This reactor represents the main event loop that handles all service requests to the network and distributes them to the correct event handlers, or servers.

Even with its enhanced performance, Twisted is based on a single-threaded implementation. It avoids multithreading since the questionable improvements in efficiency are outweighed by its issues such as context switches, increased complexity and concurrency issues which may cause slower performance overall.

The framework uses Python which makes the code condensed and readable. The user can implement a server and its protocol using less than 50 lines of code and an application server herd in approximately 200 as we'll see later. Twisted also does a good job of hiding many of the technicalities from the user just giving them the tools needed to setup a network with a complex but easy to understand structure.

2. Implementation

The bulk of the code lies in the ServerHerd.py class which instantiates the network on a per-connection basis. The prototype creates five servers, "Alford", "Ball", "Hamilton", "Welsh", and "Holiday", and creates a factory for each at a unique port. Each factory then invokes its protocol to allow for event-handling.

2.1. Protocol

We inherit our protocol from the LineReceiver class to allow for the lineReceived method. This checks a given line of input at its first token and calls the appropriate function. Any requests not starting with 'IAMAT', 'AT', or 'WHATSAT' are rendered invalid and returned in the form outlined in the spec.

2.2. IAMAT Handler

The IAMAT handler takes the user command and first parses it into tokens. It then formats the tokens into an 'AT' string which it transports to the user. Using the dictionary with timestamps, the protocol then checks if the current IAMAT update is the more recent than the

one it has on file. If so, it replaces the time of the message as well as the message itself for the current client in the dictionary, and also propagates this message to its neighbors which are stored in a global dictionary declared in the configuration file. To avoid an immediate send back of the message, the server name is added to the end of the string and checked for in the 'AT' handler. If the request is not more recent, then the current message must be outdated or a repeat of a message it has already received. As such, it returns. This prevents infinite flooding in so much as the same message does not keep bouncing between servers.

All of the work done in the method is set under a try umbrella whose except gets activated in case of any errors in the input string. The exception handling then calls the commandFailed function which returns the given input with a '?' in front.

2.3. AT Handler

The 'AT' handler manages interserver communication. Anytime this is called it means that the current request was propagated to the server in response to a client update. The handler first checks if this message is outdated before it propagates it to its neighbors using the talk function. As is the case above, the function body lies under a try umbrella to guard against errors.

2.4. WHATSAT Handler

Using the information about the client stored in server's dictionary as well as tokens in the query itself, handler fills in the necessary parts of the Google Places URL to initiate the asynchronous API call using the getPage method. To process the returned JSON data, a callback is added to filter out unnecessary results and format the input. This is then returned and, along with the original IAMAT message, transported to the user.

This again lies under a try block which discounts all invalid commands as well as the one valid command in which the network has no information about the queried client. The spec is unclear about what to do in this scenario so the program simply treats it like the rest of the failed commands.

2.5. Logging

We create a Python logging module during the initialization of the server factory to collect data about the program. This tracks all notable events and connections, such as requests handled, connections made, etc., made throughout the server herd for informational and debugging purposes.

2.6. Testing

The Program takes in a single parameter as the name of the server being created in the following format:

python serverherd.py <server_name>

A shell script is used to test the prototype which loads all five servers and sends an IAMAT request to Alford. The script then kills Welsh and ensures that cross network communication is still possible before taking down Ball and severing the network. It finally tests the WHATSAT functionality by sending a request to one of the remaining servers about the mentioned client.

3. Results

The ease of use of Python and the relatively straightforward implementation of Twisted make setting up an application server herd under the Twisted framework attractive. Despite the abstruse documentation, once one gets accustomed to the general organization of Twisted, it provides vast improves upon traditional designs in usability and clarity. Thus, Twisted presents a clean, concise solution to the problem posed in the spec. The only drawback is that anyone unfamiliar with server-side development (such as the author), especially that which involves APIs or callbacks, may have a harder time learning the framework which just means that it may not be the most beginner-friendly environment for back-end.

3.1. Node.js Comparison

Node.js is a Javascript platform for server-side and networking applications. It uses an event-driven, non-blocking model not far off from Twisted and shares

many similarities across its API. For the past few years, Node.js has been gaining a considerable amount of traction due to its syntactical elegance, like Twisted, as well as its tremendous package management. As a side effect of this popularity, the language is extensively documented and has arguably better learning resources than Twisted which is not to mention its minimal codebase and convenience. Powered by a strong V8 Javascript Engine, Node.js enjoys comparable if not better performance with Twisted though the scalability remains a question particularly due to its single-threaded nature though newer libraries improve on this aspect.

Some changes do exist between the two languages; namely, because of the absence of classes in Javascript and the treatment of functions as objects, Node.js provides a more prototypical approach to networks and some changes would be needed to make our code compatible with the JS system.

4. Conclusion

This project asked us to showcase the practicality of the Twisted framework in implementing an application server herd. As we saw, Twisted provides a fast, easy to use system for any server-side or network applications and with developments in both the Twisted and Python codebases coming out every day, the user can bask in the solid feature-filled foundation of Twisted. However, despite its relative immaturity, Node.js is a concise and even faster runtime that has the upper hand in certain areas over Twisted, and which many companies have adopted very quickly. Regardless, both Node.js and Twisted are excellent candidates to implement an application server herd and hold innumerable advantages over the traditional web server models.

5. References

[1] Twisted Website, https://twistedmatrix.com/trac/

[2] Fall 2016 CS 131 Project Description, http://web.cs.ucla.edu/classes/fall16/cs131/hw/pr.html

[3] "What are the use cases of Node.js vs. Twisted?" http://stackoverflow.com/questions/3461549/what-are-the-use-cases-of-node-js-vs-twisted (Accessed 2 Dec 2016)