

Twisted as a framework for a Server-Herd

Atibhav Mittal – University of California, Los Angeles

Abstract: The purpose of this project is to identify whether Twisted would be a good choice for a framework to build a server herd. Due to its event-driven nature and asynchronous code, it might seem like a good choice initially. However, since it is written in Python, this report addresses concerns such as Memory Management, Type-Checking, Asynchronous programming vs. Multithreading; issues that arise due to the way the Python interpreter works. I built a working prototype of a server herd that interacts with clients by doing research on the stated issues. Twisted is an excellent framework for building a server herd.

Introduction

Twisted is an event driven networking framework. The aim of this project was to examine Twisted, an event driven engine, as a substitute for LAMP Architecture. The project consists of building a server herd to rapidly send client locations to all other servers and querying what places are near other users locations using the Google Places API. Due to the event-driven nature of Twisted, the location of clients can be rapidly forwarded to other servers. The project consists of finding the problems with Twisted 16.5.0 (written in Python), especially with memory management and type checking and comparing its performance to a multithreaded application (which would be written in Java).

Design

Overview

Twisted is written in Python. Although it works with Python3, we use Python 2 in our application. Python is a strong, dynamically typed language. Since there is no explicit declaration of variables (like in more conventional languages like C++, Java), it makes development both faster and concise.

Code Detail

For this application, I designed a server class (TwistdServer) and a client class (TwistdClient), both of which inherit from the LineReceiver class from the Twisted.protocols.basic package. I decided to inherit from this class as both the Servers and the Clients receive string input. I referred to the example chatserver.py, which was given in the Twisted Documentation (1). Since we need each server and client to maintain some persistent information, we define a factory class for the server and the client.

To start a twisted application, we require a reactor to be run. The reactor is the “core of the event loop in the Twisted framework, and provides services like network communications,

threading, and event dispatching.” (1) To make the Google Places API Call more efficient, we use asynchronous programming.

Propagation of information

Each server maintains a local cache, a dictionary, from the client’s name to the client’s location, server where it was received, time at which it was sent and time difference from when it was received. Whenever any new information is sent, the server finds out all of its neighbors, connects to its ports via the reactor, and sends a message. The message is similar to the message a client sends, except that it includes the information about which server the location was sent to and at what time difference it was sent.

Flooding Algorithm and Preventing the Infinite Loop

For our application to work as described in the spec, we want each server to have the same cache. To do this, we implement a flooding algorithm. A possible source of error using this algorithm is having an infinite loop where data is being sent in a cycle. The way my code prevents that is by checking if the server cache (clients dictionary) has the same location stored for the client. If this is true, then the information has already proceeded along this route so we stop it from going again, thereby preventing the infinite loop. This however includes sending the data back one extra time to the source.

Google Places API Call

A client has the option to find what places are close to another client. This is done via the Google Places API. The object returned by the getPage() function is a “Deferred Object”. A callback function is added to it, which is executed when the getPage() function finishes executing. To limit the results to what the client specified, I used the json module in Python. I extracted the “results” from the json object and limited it to the upper bound that the client specified. I then modified the “results” in the

json object to hold these. To pretty print the json object, I called json objects with a couple of parameters, which were specified by the json Python module documentation.

Execution of Code

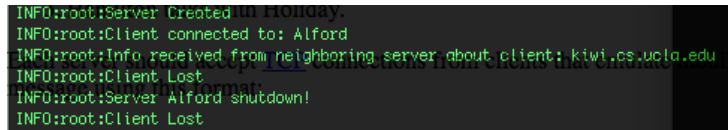
To run the code, we need multiple terminal windows. Each terminal window acts as either a server or as a client. To setup the server, we execute the following command “python server.py <server name>”. To connect to this server from a different terminal window, we need to know the port number of a running server, which is given in the conf.py file. The command “telnet localhost <port num>” is executed to connect to the server.

Twisted as a Framework for our Application Memory Management

Memory Management in Python is done via a garbage collector. The way the Garbage Collector works depends on the implementation of the Python interpreter being used. For instance, CPython uses reference counts whereas Jython uses JVM’s garbage collector (2). The presence of a Garbage Collector adds a significant overhead. For each server, we are storing the clients’ information in a local cache. Since this is a persistent property of the Server Factory, the garbage collector does not collect the dictionary in the class until the server stops running. Due to the large information that could be sent to this server, and since each server maintains its own copy of the cache, an extremely large chunk of memory could be possibly used.

Since Python manages its memory by keeping reference counts, our Client class design might have another problem. Each TwistedClient has a reference to a TwistedClientFactory. By looking under the hood of how Factories maintain persistent properties is by having a protocol variable in the Factory class. Instead of having a buildProtocol function, a protocol variable is used under the hood, which stores the protocol. Thus we have a cyclic reference between the Client and its ClientFactory. When the client is in use this is not a problem since it has a reference to the Server. However, when the client is shut down, we want to send this to the Garbage Collector. Python can detect this cyclic reference in the Garbage Collection though. (3) However, if the application were to run slower, then this is a possible source of error. To deal with this, we could use weak references in

Python to help the Garbage Collector deal with cyclic references.



```
INFO:root:Server Created in Monday.
INFO:root:Client connected to: Alford
INFO:root:Info received from neighboring server about client: kiwi.cs.ucla.edu
INFO:root:Client Lost
INFO:root:Server Alford shutdown!
INFO:root:Client Lost
```

The above picture shows how python is dealing with garbage collection. I shutdown the server before exiting the client window. As a result the server was moved to garbage collection and the line “Server Alford shutdown” was displayed. However since the client and the client factory have references to each other, the Garbage Collector didn’t collect them. Because the Python interpreter has a way of detecting cyclic references the clients were eventually lost and collected by the Garbage Collector.

Type Checking

Python is a dynamic, strong typed language. Since Python is a dynamic language, there is one way that the code could be broken down. My code assumes that the parameters, originally passed in as a line of type string, could be converted to their required format. However, if I try to send the command: IAMAT kiwi.cs.ucla.edu UCLA ROCKS the code breaks due to a ValueError. This is because the last parameter “ROCKS” cannot be converted to a type float, which is required for the time comparison. If we had a static language, we could have defined parameters that were checked at the compile. This would have prevented our code from breaking down. However, we could use exceptions in Python to deal with this, but that makes the code verbose and adds complexity to it.

Multithreading

Twisted uses an asynchronous event-driven framework (4). Since the execution is asynchronous, the application would still be responsive when making slow I/O interactions or API calls. In our application, we use asynchronous programming to make API calls to the Google Places API. So when we make the call to the API, Twisted returns a Deferred Object at the moment. A callback function is added to this Object and is executed when the call returns with either data, if it were successful, or an error. Note that the application is still responsive and running while an API call is made. To do the same with multithreading, we

would have to create another thread to execute the call to the API, then switch back to the current thread and then check whether the other thread is done executing. In this sense, context-switching with asynchronous programming is easier and faster since creating another thread adds a significant overhead (5).

My take on using Twisted

Developing the server herd with Twisted was an achievable task. Due to the modularity of the Python Twisted framework, it is easy to implement the prototype and also to add additional features to it later on. For instance, I started out with a client and a server in which the server echoes the input of the client. This was an example in the Twisted Documentation. Building on top of this, I was able to generate my own customized Server and Client class without much trouble. Furthermore, since Twisted is coded in Python, it makes the development faster as well as more concise. Twisted also supports asynchronous programming which as discussed above is a better approach for our application versus multithreading. Twisted has a great API for asynchronous programming as compared to dealing with the complexity with multithreading that involves writing synchronous code.

Node.js as a Replacement to Twisted

"Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine." (6) Node.js, like Twisted, is event-driven and supports asynchronous I/O. Node.js could be a better replacement to the Twisted framework. The compilation and execution of code in node.js is "lightning-fast" (7) due to Chrome's V8 engine. One of the main reasons to use Node.js would be that many other database and web services are written in JavaScript and would be easier to implement in Node than in Twisted. These include services like a database (MongoDB written in JavaScript) or if we were to implement a web UI. Developing in the same language would make the code faster and easier to develop since we wouldn't have to deal with syntax issues. For a person that already has knowledge of JavaScript, Node.js would definitely be a better option. Personally, however, I prefer Python to JavaScript and would use Twisted for this server herd.

Problems I faced

The main problems that I faced with Twisted and building the prototype were first getting a hang

of how to use Twisted. I have never taken a networking class, and thus did not have any knowledge of building servers and clients. One of the silliest problems I faced was that I tried connecting to a port from different Linux servers on SEASNet. Since they were different servers, I could not understand why the client was unable to connect to the server. Another issue I had was with "Deferred objects". Twisted uses asynchronous programming for web page calls like getPage(). I did not know how to deal with Deferred objects. However, after posting on Piazza about it and reading some more in the Twisted documentation, I found out that a callback is triggered when the Deferred object finishes its web page call.

Conclusion

Twisted is a great application to use to build a server herd. Written in Python, it makes development faster. However it introduces vulnerabilities with Memory Management and Type Checking, places where the code could break down. However, for most of the application Twisted is an easy-to-use framework that makes execution faster due to asynchronous programming. I conclude that Twisted would be an excellent choice for building the server herd.

Bibliography

1. "Reactor Overview." Twisted Documentation: Reactor Overview. N.p., n.d. Web. 28 Nov. 2016.
2. "How Does Python Manage memory?" How Does Python Manage memory? N.p., n.d. Web. 28 Nov. 2016. (<http://effbot.org/pyfaq/how-does-python-manage-memory.htm>)
3. Engineering, Hearsay Social. "Hearsay Social Engineering." Sitewide ATOM. N.p., n.d. Web. 28 Nov. 2016. (<http://engineering.hearsaysocial.com/2013/06/16/circular-references-in-python/>)
4. (http://proquest.safaribooksonline.com/book/programming/python/9781449326104/preface/twisted_adn_preface_2_sect_1_html)
5. Humrich, Nick. "Asynchronous Python." Hacker Noon. N.p., 23 Sept. 2016. Web. 28 Nov. 2016. (<https://hackernoon.com/asynchronous-python-45df84b82434>)
6. Foundation, Node.js. "Node.js." Node.js. N.p., n.d. Web. 28 Nov. 2016. (<https://nodejs.org/en/>)
7. "Top 10 Reasons To Use Node.js." Modulus Blog RSS. N.p., n.d. Web. 28 Nov. 2016. (<http://blog.modulus.io/top-10-reasons-to-use-node>)