Using Twisted Framework to Implement Wikimedia-like Platform

Akila Welihinda
UCLA

Abstract

In this paper, we analyze the feasibility and benefits of using the Twisted Framework to implement a Wikimedia-like Platform. Twisted is a Python event-driven networking framework and allows programmers to implement an architecture known as the "application server herd". This architecture allows multiple application servers to communicate with each other. At the end of this paper, we will compare pros and cons of the Twisted Framework against Node.JS, which is an event-driven framework built on top of Javascript.

1 Introduction

In this paper, we analyze whether the Twisted Framework will allow us to implement the Wikimedia-like Platform more efficiently. Traditionally, the Wikimedia Platform is implemented using LAMP stack (Linux, Apache, MySQL, and PHP). However, in our Wikimedia-like service also needs to support the following: (1) ability frequently update articles (2) allow access via various protocols, not just HTTP, and (3) efficiently support mobile clients (which will be the majority of the clients). These different requirements suggest that we explore other options instead of limiting ourselves to the LAMP stack. In this paper, we examine how well the Twisted Framework will support our specific requirements.

2 Implementation using Twisted

2.1. General Overview

There are 4 major classes in our prototype implementation: server_as_client_protocol, server_protocol, server_as_client, and server. Whenever we want to create a new server, we instantiate an object of type server. This instantiation results in creating an object of type server_protocol. The server_as_client and server classes are both subclasses of a factory super-class. The server_as_client_protocol and server_protocol are both subclasses of the LineReceiver superclass. We send messages between servers by using the server_as_client and server_as_client_protocol classes. We create our logging information by using Python's built in logging utilities. We respond to client's requests by using the transport.write function, which is a function defined in LineReceiever. All of the server's caches were implemented using a dictionary.

2.2. LineReceived Function

ln the server_protocol class, the LineReceived function takes a request and routes it to the proper handler. If the request is formatted incorrectly, the LineReceived function will not route the request to any handler and will instead return a message to the client specifying that a bad request was sent.

2.3. IAMAT Request

An IAMAT request is routed to the iamat_request function. This function generates an AT response message to the IAMAT request and sends this message to the client. The iamat_request function also stores the client's location, time, and response message in server's cache and then forwards the AT response message to the server's neighbors in order to update all the server's caches appropriately.

2.4. WHATSAT Request

A WHATSAT request is routed to the whatsat_request function. The whatsat_request function takes the request and returns the Google Places API JSON data, which contains all the nearby places to the client in the whatsat_request. The whatsat_request function creates a query for the Google Places API by using information provided in the WHATSAT request and the information of the client in question that is contained in the server's cache. The call to the Google Places API is an asynchronous, non-blocking call, so we add a callback function to the query that will execute after the query finishes. Our callback function logs the results of the query and sends the query results to our client.

2.5. AT Request

An AT request is routed to the at_request function. Note that the AT request is purely for internal use only; it is only used to implement the flooding algorithm and allows servers to update each other about the different client's locations. The clients of our application do not have the AT request specified their interface. Upon receiving the AT request, the at_request function checks to see if the AT request contains any new information that is not in the server's cache. If so, then the at_request function updates the server's cache and forwards the at_request to the server's neighbors. If the AT request does not contain any new information, the server simply ignores the AT request in order to avoid infinite cycles.

2.6. Flood Function

The flood function contains the logic which spreads the AT request between adjacent nodes in the server graph. It simply creates a TCP connection between the current server and all of its neighbors and sends the AT request to each of the current server's neighbors. The flood function makes use of the server_as_client class because in the TCP connection, the current server is acting as a client.

2.6. Logging

Logging is done using the built-in Python logger. Logging is setup with the logging level as the logging.INFO level. The name of each log is simply the name of the server.

3 Testing

I tested my prototype by creating some simple yet somewhat comprehensive test cases. The test cases can be run by running the "simple_tests.sh" shell script. This test script initializes all 5 servers and then sends some requests to different servers to test different aspects of the implementation. Looking at the logs generated after running the test script verifies that the prototype behaves as expected. This test script tests all request types (IAMAT, AT, WHATSAT), flooding, logging, and disconnected servers.

4 Pros and Cons of Twisted

Twisted is definitely an improvement for our Wikimedia-like application. The Twisted framework allows us to implement our application using event-driven, non-blocking code. This allows us to make great use of multiple threads without the hassle of dealing with race-conditions and synchronization of different threads. Twisted also supports tons of protocols, which is exactly what our application requires. Since Twisted is event-driven by nature, we only need our classes to inherit from some Twisted super-classes and our application will then automatically become an event-driven application. This allows to quickly and easily implement the "application server herd" architecture, which is the architecture we are aiming for.

However, there are some downsides to Twisted that I encountered. Since Twisted is event-driven by nature, it poses as a difficulty to programmers who have never encountered the event-driven programming paradigm. Another downside to using Twisted is that the programmer must become familiar with the Twisted API, which takes time. Since Python is a scripting language that is not compiled, it may be the case that our application will crash due to an uncaught syntax error or type error. Therefore, it is absolutely necessary to create test cases that exercise every line of code if we are using Twisted to implement our application. If we used a compiled language, such as Java, we would not have this issue.

5 Challenges

I faced multiple challenges in implementing this prototype. Understanding how to use the Twisted interface and how to design the class hierarchy took me a significant amount of time. I also spent a great deal of time figuring out how to create automated test cases for my

prototype using Bash. I also found myself irritated by the fact that syntax errors I made would not be caught until runtime due to Python being a scripting language.

6 Twisted v.s. Node.JS

Both Twisted and Node.js are event-driven languages. Twisted has been around longer than Node.js has, so naturally Twisted has more features, documentation, and tutorials. However, although Node.js is relatively new, it is actually gaining popularity very quickly. There are a couple of reasons why Node.js is very popular. Node.js has very good support for asynchronous programming because it's built natively to handle asynchronous I/O. Since Node.js is a Javascript framework that serves as the back-end logic, it allows developers who use Javascript on the front-end to develop in a familiar environment when they do back-end development. Another advantage Node.js has is that since it is newer, it is able to make use of the latest and greatest features and tools in the software industry. For example, Node.js has a very good package-manager and runs on top of a very fast V8 Javascript Engine created by Google. Overall, Node.js would be a good alternative to Twisted. Although both languages have similar paradigms, each language has its own benefits that will be useful for different applications.

7 References

[1]https://nodejs.org/api/

[2]http://blog.modulus.io/top-10-reasons-to-use-node

[3]https://www.quora.com/What-are-the-benefits-of-developing-in-Node-js-versus-Python

[4]http://krondo.com/blog/?page_id=1327