Twisted Places Proxy Herd Report

Thomas Lutton
University of California, Los Angeles
CS 131, Fall 2015

Abstract

Twisted is an open source event-driven networking engine written in Python. This project explores Twisted as a potential replacement for Wikimedia's LAMP architecture. This paper will describe a simple Twisted implementation for a proxy herd and will discuss whether Twisted is well-suited as a replacement. Finally, Node.js and Twisted will be compared and contrasted to make a final ruling.

1 Introduction

The Wikimedia Foundation uses a LAMP model for its web services. It appears to serve its purpose well, but with the outside world consistently shifting from desktop to mobile, Wikimedia could see its method become obsolete. If articles were updated more frequently and the clients were more mobile and accessed with a wider variety of protocols, information retrieval could grind to a halt due to LAMP's blocking I/O protocols. Since Twisted is event-driven and uses non-blocking I/O with its reactor model, it is seen a potential replacement if LAMP were to become obsolete.

2 Twisted

2.1 Twisted Overview

Twisted is an event-driven framework that mirrors multi-threaded performance while maintaining the simplicity of a single-threaded network server. Twisted allows easy customization of an application level protocol by encapsulating a line receiver as well as client and server factories. Programmers derive their own implementations from these classes and implement their own callback functions to create a customized protocol.

A key concept for Twisted applications is the use of deferred or future objects. A deferred is an instance of a class that receives a process and enqueues it in a queueing buffer without performing any operations. Each deferred object manages its own callback function in a FIFO manner.

2.2 Single and Multi-Threaded Servers

Single threaded servers are simple because there are no threads to organize which means that deadlock and race conditions are impossible. However, they have the potential to be very slow if a single client has a bug or simply has very long requests.

Multi-threaded servers could be faster if done carefully and correctly, but they add much more complexity. If clients don't connect for long, creating and killing threads as well as context switching can become a performance hog. If clients have long gaps in their communications, their threads will be idle and waste resources.

2.3 Twisted Server

Twisted deferreds solve these problems and behave in a multi-threaded way while still being single-threaded. Each deferred has a callback chain that can make it possible to operate on the result of a function before its value is even available. This works by creating a class that waits for its value to arrive. The asynchronous Twisted reactor processes these callbacks as they occur. When the event of data arrives, the reactor does a callback and finishes any unprocessed portions of the deferred.

3 Implementation

3.1 Implementation Overview

The prototype involves a five server herd that has some pre-selected neighbors as determined by the project specification:

```
HERD = {
       "Alford" : ["Parker", "Powell"],
       "Bolden" : ["Parker", "Powell"],
       "Hamilton" : ["Parker"],
       "Parker" : ["Alford", "Bolden",
       "Hamilton"],
       "Powell" : ["Alford", "Bolden"]
}
```

For example, Parker can talk to Alford, Bolden and Hamilton, but not Powell. Messages are thereby exchanged between these servers with TCP connections.

Each server is assigned a port value. I decided to use ports 13000 through 13004.

```
PORTS = {
    "Alford" : 13000,
    "Bolden" : 13001,
    "Hamilton" : 13002,
    "Parker" : 13003,
    "Powell" : 13004
}
```

Each server has the capability to send three messages: IAMAT, WHATSAT, AT. Clients can use IAMAT to send its location to the server. These IAMAT messages contain the client's ID, location via latitude and longitude as well as the client's perceived time that it sent the message at. The server responds to IAMAT messages with an AT message. ATs contain the server ID, the difference in time from when the IAMAT message was sent and the server processed the message, and then a mirror image of the IAMAT's data fields. WHATSAT messages contain the name of another client, a radius (in KM) and an upper bound. The radius must be at most 50km to limitations to the Google Places API. The server then responds to WHATSAT messages in a similar way to IAMATs. The server will send an AT message as well as a JSON-format message in the same format that Google Places uses for a "Nearby Search". Here are some example calls and responses, excluding the large JSON format message:

```
IAMAT kiwi.cs.ucla.edu +34.068930-118.445127
1400794645.392014450

AT Alford +0.563873386 kiwi.cs.ucla.edu
+34.068930-118.445127 1400794699.108893381

WHATSAT kiwi.cs.ucla.edu 10 5

AT Alford +0.563873386 kiwi.cs.ucla.edu
+34.068930-118.445127 1400794699.108893381
```

3.2 MyServerFactory

MyServerFactory is a ServerFactory derived class that is the backbone of this project. Each of the five required servers, Alford, Bolden, Hamilton, Parker, and Powell, are instances of this class. The class contains basic information that each server needs to operate; such as its name, clients/neighbors, port number, and number of connections. The client information that is stored is the same information that would need to be accessed when sending an AT response. This information is important to store because it can be used to determine whether or not a message contains new or old information.

It is also responsible for implementing the creation of its log file which will contain any information about input, output, new/dropped, connections as well as errors.

3.3 My(Server)Protocol

MyServerProtocol receives all messages and dispatches all event handlers. It derives from LineReceiver and implements lineReceived, connectionMade, connectionLost, and all of the message handlers and their helper functions. The protocol class also handles all logging to the file generated by the ServerFactory.

3.4 IAMAT Handling

An IAMAT handler is called whenever the an IAMAT message is detected by lineReceived. In this function, an AT response is built and client

information is saved for further reference. Once the AT response is generated, it is sent to the client and also to its neighbors. This is called flooding and is how neighbors stay up to date. By giving each server a copy of the information that its neighbors know, it eliminates the need for a central server to store the information as well as the bottleneck that this server would create.

The flood itself works by a server sending a message to its neighbors and the neighbors take it upon themselves to determine whether or not the information is new and worthy of storing. If it is, it will pass the information to its neighbors as well. The worth of information is determined by looking at the timestamps of the data. If the timestamp is newer than the copy of the data that the server currently holds, it will replace its own data with the new data. If the time stamp is the same or older, it simply discards the data and doesn't propagate it any further.

3.5 WHATSAT Handling

WHATSAT handling is done whenever a WHATSAT command is detected by the lineReceiver. The handler first validates the received information and then makes an API call to Google Places. The handler validates first because API calls, when done in large volume can become expensive and there is no point to making a call when the radius exceeds the limit of the Google API itself.

The handler then constructs a response and uses the getPage method execute the request. Once this is done, the result must be packaged and sent to the server's neighbors.

3.6 AT Handling

AT handling is done whenever the lineReceiver determines that an AT command has been received. This handler checks the clientID's "new" information with the information that it currently has on record. If the new information has a later timestamp than the current information, the information is updated and the information is propagated further via flooding.

3.6 Logging

Logging is performed whenever a server receives or sends an I/O. New and dropped connections are also added to the log. Further, I added error logging as well.

3.7 Challenges

My only previous experience with Python was in CS 35L two years ago, but picking it back up did not pose much of a problem syntactically. My main problem lied in finding and applying the proper libraries to efficiently accomplish what this project required.

Figuring out how to do the flooding was also a difficulty since I am inexperienced in Twisted and flooding in general. It was a new challenge but was straightforward after combing through the Twisted documentation and implementing a LineSender.

3.8 Analysis

Since Twisted has become so widespread and well documented it is easy to work with once you get your feet wet. The Twisted protocol is very straight forward and is aided by the fact that Python as a language is very straight forward. Python has numerous features like its syntax and garbage collection that make it easy to pick up on the fly.

4 Python vs. Java

As requested in the specification, the following section is a brief discussion of the main differences of the implementations of Python and Java.

Python and Java have very different approaches to multithreading. Java gives the user much more control to multithread than Python which actually only allows one thread to run at a time. If your boss wanted you to implement a multi-threaded server, Python would not do the job. However, for this project we use Twisted which utilizes deferred objects in a single-threaded

manner so this lack of concurrency is not a problem.

Since Python is mainly an interpreted language, it does type checking dynamically. Java, on the other hand, is statically typed. This poses both advantages and disadvantages to Python. The advantage is that programmers are given a very simple syntax that is not cluttered by type definitions like many imperatively styled static typed languages like Java or C++. However, this can lead to runtime errors that would have been caught by the Java compiler. This is a tradeoff in simplicity versus reliability with no clear frontrunner. The argument can be made that a fundamentally sound programmer would run into runtime errors rather infrequently, so the difference becomes somewhat inconsequential.

Both Java and Python use garbage collection for memory management. However, the two approaches of garbage collection couldn't be more different. Java uses the mark and sweep method that we discussed in class. This method marks items for deletion and occasionally sweeps through and reclaims memory. This can be left up to the garbage collector to decide when to run, or it can be invoked by a programmer using a finalize call. Mark and sweep is not very good for the performance of a program, but benefits from simplicity because garbage collection is not always a necessity. Python employs a reference count method that reclaims memory whenever a reference count for an object reaches zero. If a Python program is running low on memory it will then use mark and sweep like Java to reclaim some more memory. The Python method of reference counts comes with one fatal flaw in the form of circular references. It is possible to create memory leaks by programming carelessly.

5 Node.js

5.1 What is Node.js

Node.js is an open-source, cross-platform runtime environment for developing server-side web applications. These applications are written mainly in javascript and provide an event-driven architecture for non-blocking I/O. Many large companies such as IBM, Microsoft, and Yahoo! Use Node.js.

5.2 Comparison to Twisted

Node.js and Twisted are both asynchronous event-driven network programming frameworks. Node.js is much newer than Twisted and has been built atop Google Chrome's V8 JavaScript engine which gives it better performance than Twisted. Node.js was built specifically for the purpose of handling asynchronous I/O whereas Twisted was built on top of a Python platform that had a broader purpose in mind. This gives the advantage to Node.js because of performance. However, Node.js is still relatively young, and although it has gained massive traction in both large companies and startups alike, it does not have the code base that Twisted has been able to develop over its life span. Since Node.js has been developed for such a specific purpose, any customization beyond that purpose would probably result in the user having to rely on third party libraries instead of the extensive libraries that the Python language itself provides.

As far as the implementation changing, it would probably change more along the lines of the requirements of JavaScript vs Python as opposed to Node vs Twisted (since Node and Twisted are very similar). JavaScript is largely prototypical and does not contain classes. So we would not be able to derive classes for factories and protocols like we did in Twisted. However, this is a very small hindrance when we look at it from the perspective of JavaScript being the dominant web programming language for front-end and by employing Node.js we can avoid programmers having to learn JS for the front end and then Python to do the back.

Due to these reasons, Twisted seems like a feasible solution to the problem posed in the specification, but Node.js may actually be a stronger solution, but it is still young and more research should be done before making a firm conclusion on the subject.

6 Conclusion

This project was to implement a prototype proxy server using Twisted in Python. This report outlined what Twisted is, how the project was implemented, a discussion on Python versus Java, as well as an exploration of Node.js and its potential as a solution to the proposed Wikimedia problem. Overall, Twisted seems like it is a suitable solution to this problem, however, with more research, Node.js may become the frontrunner.

7 References

"Developer Guides." Developer Guides — Twisted 15.0.0 Documentation, http://twistedmatrix.com/documents/15.0.0/core/howto/index.html.

Eggert, Paul. "Class Lectures." UCLA. Lecture.

Fall 2015 CS 131 project description, http://web.cs.ucla.edu/classes/fall15/cs131/hw/pr.html.

"Global Interpreter Lock." Python. 13 Jan. 2015.

"What are the benefits to developing in Node.js versus Python?" https://www.quora.com/What-are-the-benefits-of-developing-in-Node-js-versus-Python.