Twisted Proxy Herd Server

Panos Karagiannis

University of California Los Angeles
panoskaragiannis.ucla@gmail.com

ABSTRACT
This paper explores whether Twisted, which is an open source event-driven networking engine written in Python, is an effective choice to implement a Proxy Server Herd as an alternative configuration to the Wikimedia style architecture.

1. INTRODUCTION
Wikimedia uses the LAMP platform, which is based on Linux as the operating system, Apache as the web server, MySQL as the RDBMS and PHP as the Object Oriented Scripting Language. Even though LAMP works well in most cases it seems it would be inefficient for news where updates happen too often and various protocols are used because of its non event-driven structure. As far as the software implementation is concerned adding new users would be difficult while from the system point of view constant searches would render the system slow. As a result, a Proxy Herd Server Application implemented using Twisted is considered as an alternative to LAMP.

2. Twisted
Since Twisted uses event-driven programming the core of this platform revolves around the reactor's event loop. The reactor waits and demultiplexes events dispatching them to event handlers and simplifying responding to events anywhere in the network stack. The event loop works by calling either an external or internal event provider which blocks until and event has arrived in order to avoid race conditions and then dispatches the event.

2.1 Advantages and Disadvantages
Twisted apart from being event driven also supports many protocols including TCP, HTTP, SSH etc. thus giving flexibility as far as the rules and conventions of network communication are concerned. A major advantage of Twisted is that it includes large amounts of functionality. For instance, server and client protocol implementations are included making it easier to produce applications. Also, its asynchronous and event-based architecture allows programmers to avoid the hustle of using multithreading and introducing painful race-condition errors. In addition Twisted is open source, thus allowing distribution of the Twisted source code as part of the application.

Nevertheless, a possible drawback originates from the nature of Twisted's event driven structure. Since each callback has to finish fast and return, keeping persistent state data inside the protocols is impossible. Moreover another disadvantage can be found in Python. Python is an interpreted language and therefore lacks the speed of other compiled languages. Therefore, Twisted can often be slow.

2.2 Python
Twisted is implemented in Python which is a widely purposed high-level language. Twisted exploits many of Python's advantages like its dynamic type system in order to simplify implementation. Twisted also uses Python's Object Oriented nature to implement various data structures. Also, since Python is an interpreted language it allows easy portability across platforms.

Moreover, part of the reason why Twisted is an event-driven platform is because of Python's difficulty to deal with multithreading. Since forking Python processes has many disadvantages, like Python's reference counting creating problems with copy-on-write and problems with shared state, it was suggested the best option was an event-driven framework.

3. Server Herd
To implement the Server Herd five servers are used named: Alford, Powell, Parker Bolden and Hamilton. Not every server can communicate with one another and this is why a flooding algorithm is implemented to allow data propagation along the network. Four main event handlers handle the events. The reactor listens on designated ports for TCP connections from clients. For this project a ServerFactory class and a ServerProtocol class deal with the behavior of the server at various events. In addition since data need to be propagated along the network ClientFactory and ClientProtocol classes were implemented in order to allow inter-server communication. ServerFactory and ClientFactory contain all persistent data

because the protocol classes are instantiated for every connection and go away when the connection terminates.

3.1 Usage

In order to boot a server the command:

```
python proxyHerd.py <Server Name>
```

is considered to be valid, where <Server Name> is one of the five valid names mentioned in the above section. If the number of arguments provided is not correct or if the server name does not correspond to one of the server names execution will terminate immediately printing an error message.

3.2 handle_IAMAT

This is the event handler that deals with the commands of the users. If the command given by the user starts with IAMAT then the specific handler is called which in turn breaks the query apart in order to calculate the various parts for the AT answer it provides to the user. After responding to the user it propagates this information to the other servers using an AT construct. In that case another handler is invoked which stores the information the server has received appropriately.

3.3 handle_AT

This event handler deals with propagated information. After each IAMAT command a user sends to the server, the server using the internal classes ClientFactory and in turn ClientProtocol, propagates the information to its peer servers (the ones it communicates with). The handle_AT handler handles this information and it is responsible to store it in a dictionary local to the server. After the server has received the data, it propagates it to its own peers. That way information from an IAMAT command reaches every server. This is the core of the flooding algorithm as described in the above sections.

3.4 handle_WHATSAT

This event handler gets called when a user asks what is around another user. The handler searches the local database of the server to locate the user and using the user's location it connects to the Google places API to retrieve the requested data. The Google places API returns a json file which after some manipulation is returned to the user accompanied by an AT message. If the user is not found in the Python dictionary then the response of a server to a WHATSAT command will be that no such user exists in the servers.

3.5 handle_ASK

This is the simplest event handler, called when a server is booting. Upon booting a server asks the other servers it communicates with to propagate the data of the users stored in their dictionaries. This is done in case the booting server was not online when some of the information was propagated along the network using the flooding algorithm. That way, even if a server comes online retrospectively it will still be informed of the users that were using the servers and thus it will be able to answer correctly WHATSAT commands.

3.6 handle_STORE

This is the only handler used in the ClientProtocol and its function is to store users to the server's dictionary upon booting. So, upon booting there is an ASK-STORE "dialogue" between the booting server and its peers. The booting servers "ASKS" for information and each peer responds with a STORE for each user it has stored in the dictionary. For each STORE received the server stores the information in its own local dictionary so that it can correctly respond to WHATSAT commands if needed in the future.

3.7 Logging information

Logging was done using the logging utility provided by the Python library. The log file is created so that is matches the server name plus a ".log" extension and it records all input, output as well as important aspects of the behavior of the server like propagating information to other servers or asking for information upon booting. After a server terminates and then is rebooted, a new log file is generated discarding all previous log files.

3.8 Testing

In order to test the program I manually set up the servers by calling them on the command line using:

```
python proxyHerd.py <Server Name>
```

Then I created telnet clients to connect to the servers and ask various commands. It is important to note that the API key to connect to Google places was stored on a separate file named project_config.py and which was included in the proxyHerd.py file.

4. Biggest Hurdles

4.1 Flooding Algorithm

The biggest hurdle in implementing the Proxy Herd Server was realizing how to implement the flooding algorithm. For instance, a naïve approach would be for each server to blindly propagate information to its peers, but that would lead to infinite loops since information would go back and forth between any two peers. In order to solve the problem I had to pass another parameter to the function responsible for the propagation of data in order to inform it which server did the propagation. In that way the server receiving the information would not send it back to the peer from where it received it.

4.2 Persistent Data

Also at first I was confused about how I should store persistent data for each server. I tried declaring a dictionary local to each protocol until I realized that this was the wrong approach since each protocol was instantiated with every new connection. As a result persistent data, such as the dictionary that stores the names and details of the users, is stored in the ServerFactory and is available for use throughout the lifetime of a server. This is how the WHATSAT_handler is able to respond to various commands concerning the location of users.

5. Node.js vs. Twisted

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine using an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js and Twisted are two different networking engines that can be used for a lot of similar applications. Nevertheless, they also bear subtle differences that would change the current implementations of our Proxy Server Herd.

5.1 Documentation

Node.js is much newer than Twisted and as a result its community is smaller. So far the packaging/sandboxing for Node.js is not as robust as that of Twisted.

5.2 Different Languages

Twisted uses Python while Node.js uses JavaScript. These are two different languages that would change the implementation of the Proxy Herd Servers. Python is a scripting language which can be used to do server side work. In contrast to that Javascript is mostly used to implement interactivity on web pages.

5.2.1 Numeric Types

JavaScript has only one numeric type without having a separate integer type. Therefore different numerical constructs would have to be used when calculating the time in the AT response to the user.

5.2.2 Statements and Declarations

In JavaScript omitting the var in the declaration var z=10 would cause the variable to be considered as a global object. In Python the statement z=10 would create a variable z, make it point to 10 and then would inscribe it in the symbol table. In Python, variables don't have types. As a result, using Node.js in the implementation for the Proxy Herd Server we would have a different syntax for every variable we declared.

5.3 Event Loop

Node.js implements its event loop as a language construct rather than a library as we see it in Twisted. The Node.js is written in low level C with JavaScript companions wrapped around it while Twisted event loop is all written in Python. Even though the two event loops behave in a similar way they are implemented using very different constructs.

6. Conclusion

In conclusion after having implemented a Proxy Server Herd Application in Twisted I could say that it is very maintainable and reliable. Its event-driven structure allows to eradicate mistakes related to multithreading thus saving a lot of time during implementation. Moreover in an application where more users are mobile and constantly broadcasting their GPS signal it is not feasible to have an arbitrarily large amount of threads blocking and waiting for remote method calls to return. Thus the event-driven nature of Twisted is again very useful in tackling this problem. As a result, judging by the conducted analysis as well as by actually implementing a simple application using Twisted I would say that this platform is a good alternative to the LAMP platform when it comes to implementing applications where network traffic is high and many protocols are used.

7. REFERENCES

[1] Lefkowitz, G. and Zadka, *The Twisted Network Framework,* `http://legacy.python.org/workshops/2002-02/papers/09/index.htm`

[2] Fettig, A. and McKellar, *Twisted Network Programming Essentials, 2nd Ed,* O'reilly, March, 2013, ISBN 978-1-449-32611-1

[3] Alex Martelli, *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`