Twisted Places Proxy Herd

Patrick Killian Jackson
University of California, Los Angeles

CS 131
March 9, 2016

Abstract: Twisted is an event-driven networking framework, running on Python, used for the development of internet applications. This paper examines the potential of using Twisted to implement a "server herd" as a possible replacement for a LAMP platform application. In doing so, this paper considers the implications of Python's type-checking, memory management, and multithreading with regard to Twisted as a candidate for this application and investigates how easy Twisted applications are to write, the maintainability and reliability of such applications, and how such applications might work in combination with others. Additionally, this paper compares Twisted with Node.js as a potential framework for this application.

1 Introduction

Wikipedia is based on the Wikimedia architecture which makes use of a LAMP platform based on GNU/Linux, Apache, MySQL, and PHP. This architecture uses a load-balancing virtual router along with numerous, redundant web servers. This redundancy results in increased performance and reliability. Additionally, this LAMP platform is very scalable with very good localization.

However, despite the advantages of the LAMP platform, difficulties arise when the application works with data that is being constantly and quickly modified, as updates need to be synchronized across all the redundant webservers. This performance issue encourages the consideration of an alternate implementation instead using even-driven programming in the form of a "server herd." In this implementation, servers will propagate data between each other using caches to increase performance. This paper examines such an implementation based on the Twisted framework demonstrated by a simple, parallelizable prototype proxy for the Google Places API.

2 Twisted

2.1 Language Notation

Twisted is implemented in Python and thus benefits from many of the languages features. Python is an interpretive language, sharing characteristics with both scripting and compiled languages, making it an incredibly powerful language despite its concise syntax. Python programs additionally benefit from dynamic type checking which allows the programmer to use variables without declaring type or size. With regard to memory management, Python automatically tracks the memory used by a program, allocating objects when necessary and performing automatic garbage collection when objects are no longer needed. While Python does support multithreading, Twisted uses a single thread implementation to make the platform more reliable and to avoid handling synchronization issues between threads. Largely due to Twisted using a single threaded design, our prototype proxy was relatively simple to implement.

2.2 Even-Driven Programming

The Twisted framework is implemented using even-driven programming, meaning that the program's execution is based off of specific events that trigger reactions instead of the program, for example, constantly checking if it should perform an action. In my proxy prototype server herd, servers send messages to each other and use a flooding algorithm to broadcast updated information

to neighboring servers in the server herd. Due to this flooding algorithm, the severs frequently connect to one another and transmit messages to neighboring servers containing updated information. Because of the high frequency of connections and data transmission, Twisted's event-driven design makes the application server herd more reliable and efficient as data across the server herd is quickly updated and synchronized across the herd when modified and work in order to handle specific events.

3 Google Places API

In addition to the Twisted framework, my application server herd also makes use of the Google Places API. This API provides vast location information that my servers use to query for a list of places near a given client location. In my implementation, clients obtain this information by making requests to the Google API, passing a client name, a radius, and a results limit. The client's name specifies for which client places information is being requested, the radius defines the area around the client from which the Google Places API will pull places results, and the results limit specifies how many results the Google Places API should serve. In order to do so, I created a Google Places key that authenticates my program when making requests to the API. My program makes these requests via a getPage function call, receives the results in JSON format, and then extracts and formats the information from the JSON. The Google Places API actually does not take a results limit parameter when making a request with getPage, so my program handles this parameter by taking a subset of the complete results returned from the Google Places API.

4 Prototype

4.1 Overview

My prototype implementation represents a server herd consisting of 5 servers which act as both clients and servers when communicating with each other. The implementation is based on 4 classes named herdServerProtocol, herdServerFactory, herdClientProtocol, and herdClientFactory. These classes take advantage of Python's class inheritance by inheriting from Twisted classes. The factory classes are used to create protocols and receive events while the protocol classed are used to define how the program responds to events. My implementation uses several Twisted libraries including reactor, protocol, and LineReceiver, to name a few, that allow the application to establish TCP connections with other servers, react to received messages, and utilize protocols.

4.2 herdServerFactory

This class is the top level representation of an individual server in the server herd. It creates the protocol this server will use and stores variables that track the server's name, its designated port number, and a list of clients it has received information about. Additionally, this class instantiates the logging for the server represented by this class. It creates a log file, named according to the server name, that tracks that individual server's activity.

4.3 herdServerProtocol

This class is instantiated by the herdServerFactory class and defines the protocol used by a server in the server herd. This protocol is an implementation of how a server responds to messages received from neighboring servers. This class includes the method definitions named connectionMade, lineReceived, funcIAMAT, funcWHATSAT, funcAT, printJSON, flood, and connectionLost. The connectionMade and connectionLost methods are simple methods used simply to enter logging information in the server's log to identify when a connection was made and ended, respectively. The funcIAMAT, funcWHATSAT, and funcAT

methods are used to handle each of the three types of messages a server may receive. The lineReceived method parses a message and determines which of the three method handler methods will be called. The printJSON method serves the purpose of extracting the desired information from the JSON-formatted result of a Google Places API request including applying the limit parameter describing how many results the program wants and then reformatting the results so they can be sent to neighboring servers. Lastly, the flood method allows my program to broadcast received messages to neighboring servers in order to propagate updated information throughout the server herd.

4.3.1 Handling IAMAT Messages

A message beginning with the IAMAT keyword signifies a message in which a client informing another server of its location. This message also includes a time stamp of the client's perception of when it sent the message. The funcIAMAT method parses this message and updates its record of the client's location. The server then responds to the client with an AT message and propagates this reply through the server herd by using the flood method to send the message to the server's neighbors. The funcIAMAT method documents this process in the corresponding server log file.

4.3.2 Handling AT Messages

Messages starting with the AT keyword are the messages sent from a server in response to a client's IAMAT message. The AT messages are propagated through the server herd via the flood method in order to update and synchronize the location information stored at each server. The funcAT method handles receiving these messages by updating the local location information according to the data in the AT message and then flooding the AT message to its neighboring servers. Since all servers that receive the AT message propagate the message to their respective neighbors, eventually all servers will have received the AT message. Thus, the funcAT method checks each AT message's timestamp and stops flooding when it detects a repeated message. The funcAT method documents this process in the corresponding server log file.

4.3.3 Handling WHATSAT Messages

Messages beginning with the WHATSAT keyword are sent from a client to a server querying for a specific number of places in a given radius around another client. The funcWHATSAT method uses the getPage method to send a query request to the Google Places API and receives a response from the API in JSON format. funcWHATSAT then calls the printJSON method to take a subset of the response according to the specified number of results requested and reply to the client with this subset of results appended to an AT message. The funcWHATSAT method documents this process in the corresponding server log file.

4.4 herdClientProtocol and herdClientFactory

The herdClientFactory class of my prototype invokes the herdClientProtocol class and maintains a value for the client's message. The herdClientProtocol class simply establishes a connection to a server and sends its message to that server. Following message transmission, the protocol kills the connection to the server. Doing so allows the server to handle a higher frequency of client connections.

4.5 Program Logs

As previously described, each server factory class instantiates a log file that tracks state of a server. In order to do so, I used Python's built-in logging library and Twisted's log library. The logs for each respective server

record interactions between servers and any errors detected.

4.6 Prototype Testing

In order to test my prototype I wrote two shell scripts. The first script, named startServerHerd.sh, loops through the five server names and starts each respective server by running the serverHerd.py program. After the servers are running, the test.sh script invokes a series of interactions between the servers to test the functionality of my server herd implementation. First, a client IAMAT message is sent from the "UCLA1" client to the Alford server resulting in Alford updating its location information for that client and propagating this information to its neighboring servers. After this first message, the script stops the Welsh server and sends another client IAMAT message from client "UCLA2" again to the Alford server to ensure that the message is still propagated throughout the server herd despite one inactive server. Next, the script stops the Parker server in order to cut off the Alford server from the rest of the server herd. The script then sends a client IAMAT message to the Alford server to test that this message is now not propagated to the rest of the server herd. Finally, a client WHATSAT query is sent to the Hamilton server to demonstrate a Google Places API request.

5 Twisted: Benefits and Drawbacks

In implementing my prototype server herd with the Twisted networking framework, I have identified several advantages to using the technology. As previously described, Twisted is coded in Python making it a very powerful framework that is presented in very concise source code. Python allows Twisted to be flexible as its components can be easily incorporated with code written in other languages. Twisted benefits from Python's dynamic type checking and automatic memory management including garbage collection both contributing to simpler code. Additionally, an application can easily implement classes that inherit from Twisted's classes in order to build upon the Twisted framework by customizing event-handling according to the programmer's specifications.

Despite the many advantages to using Twisted for a server herd type application, there are also some drawbacks. While dynamic typing makes writing Python code incredibly simpler for the programmer, it can make the code more difficult for others to interpret as types can help bring clarity to a program's implementation. Although Python does have automatic memory management, this can also be a drawback as it has the potential to fail with some applications using Twisted. In managing memory, Python does a lot of allocation of memory. When allocating space for larger objects, Python allocates the memory and then releases the memory when the object is no longer being used. However, when smaller objects are allocated, the memory they use is never freed with the hope that it will need to be used again. When running a server using Twisted, this can result in a lot of small sections of memory being allocated and never freed eventually rendering large sections of memory unusable.

6 Twisted vs. Node.js

Node.js is a JavaScript based platform used to create server-side web applications. Like Twisted, Node.js is event-driven and similarly can be used to build reliable and efficient web applications. Node.js is a newer platform, thus it does not have as extensive libraries as Twisted. Although, Node.js has gained popularity and its package library is rapidly growing, Twisted is much more mature. Additionally, Node.js has better performance, due to its JavaScript engine, than Twisted making it easily scalable for large applications. One drawback of Node.js is since it is based on JavaScript, it does not have access to any built-in class features.

Although it is lacking in built-in features, JavaScript and JSON pervade the internet, as demonstrated by the Google Places API, making Node.js the ideal choice to working this code and data. Both Twisted and Node.js are single threaded platforms so they suffer from the similar performance issues; however, Node.js especially suffers when there is an extensive server-side workload. Considering the advantages and disadvantages of both platforms, both are viable options for implementing web applications. Twisted is ideal if a large package library is desired and the implementation is simple. If the application in question is more complicated and has demanding scale requirements.

6 Conclusion

In creating the prototype server heard, I successfully demonstrated Twisted's viability as a potential candidate to replace the LAMP architecture. Due to Twisted's event-driven nature and its extensive package library it can be used to implement reliable and efficient web applications. Due to the many benefits of Twisted and Python it was relatively simple to create a prototype server heard. Although I did not have time to fully research Node.js and develop a prototype to compare with the Twisted prototype, Node.js is a growing platform that also serves as a viable replacement for the LAMP architecture.

8 References

[1] http://twistedmatrix.com/documents/12.1.0/core/howto/clients.html#auto3

[2] http://www.evanjones.ca/python-memory.html

[3] https://www.safaribooksonline.com/library/view/learning-python-5th/9781449355722/ch01s07.html