CS 131 Project: Twisted places proxy herd

Aditya Rao, 404434974
adityakatapadi@cs.ucla.edu

Abstract

In this paper I try to assess twisted as a potential networking framework for a Wikimedia style service designed for news. I explore twisted from a programmers perspective, with emphasis on implementation, protocols used and the ease of developing a prototype compared to other popular frameworks like Node.js.

1 Introduction

Wikimedia architecture is built on the LAMP stack. LAMP implements a Linux operating system, an Apache HTTP Server, a MySQL database and generally a PHP or Python framework for frontend. It is evident that the Apache server limits the type of connections to one protocol, HTTP. It can also be noted that the scalability of the server will depend on how well can a programmer deal with multi-threading/ concurrency. A programmer has to choose a break-even point between maintaining high performance and avoiding data races, which translates to an undesirable bottleneck at the server level. In this project I use the application server herd architecture to enable inter-server communication. This will allow servers to exchange client information, thus enabling mobile clients to access information irrespective of the server they query.

2 Twisted: An Overview

Twisted is an event-driven networking framework licensed under the open source MIT license. Event-driven means that it is characterized by event loops and call backs to trigger responses to events. In short, the program flow is determined by external events. Being event-driven also makes twisted circumvent the complexity associated with a multi-threaded approach. Hence it is not as complex as a multi-threaded approach. Deferred objects are used to keep the code asynchronous. Twisted has two main packages to help in the prototyping of any protocol: internet, protocols. These are explained in the subsequent sections.

3 Approach

3.1 Overview

The protocols are described in the assignment description[1]. I made use of three classes to encapsulate the role of the server, client and the protocol. The class ProxyServerHerdProtocol creates connections based off IAMAT commands by instantiating the protocol implementation. I made use of the reactor to listen to ports for any commands for the server/client. The class ProxyHerdServer is used to initialize the server and the classes ProxyHerdClient, ClientProtocol are used to instantiate the client. The functions for connections were taken from the twisted documentation[2].

3.2 Handling Protocols

The servers and clients communicate with each other using three main protocols: IAMAT, AT and WHATSAT.

- **The IAMAT protocol**

The IAMAT protocol is used by clients to send location updates to servers. The servers then relay the client information received to other servers by using an AT protocol. This works by the general idea of server flooding, with restrictions placed on the servers aa particular server can flood.

An IAMAT Command:

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

An IAMAT command has 4 fields. The command name, the client ID and the position in +latitude-longitude format and the time (POSIX format) while sending the message.

The parseIAMAT function: This function parses an IAMAT command. It has format checking ensuring that an erroneous IAMAT command is recorded the logger and not further processed. Currently it indicates whether a clock skew is present or not by appending a '+' when the time at which the server received the message is greater than the time in the IAMAT command. The parseIAMAT function records all IAMAT commands it receives and updates (through the **updateLocation function**) its neighbors on the status of the client received in the IAMAT message.

- **The AT protocol**

The AT protocol is used to communicate among servers. A server which receives an IAMAT command from a client, relays the client information to its neighbouring servers using the AT protocol.

An AT Command:

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

The AT command has 6 fields, the command name, the server which received the IAMAT command from the client, the time difference (between the sent time (client-dependent) and the received time(server-dependent)), the position of the client in +latitude-longitude format and the time at which the message was sent in POSIX format.

The parseAT function: The function parses an AT command. It enables a server to check if the information it received from another server is outdated or not by comparing the time of the message received to the corresponding record for the client in the server. The client record is updated if it is the latest update from the client. This function along with parseIAMAT implements the flooding algorithm[3].

- **The WHATSAT protocol**

The WHATSAT protocol is used to query the servers. A server searches its records to find information about the client specified in the WHATSAT command.

A WHATSAT Command:

```
WHATSAT kiwi.cs.ucla.edu 10 5
```

The WHATSAT command has 4 fields, the command name, the client about which information is needed, the radius and the number of results (json objects) to return. The server replies to a WHASAT command with an AT message with a json object appended to its end.

The parseWHATSAT function: This function makes a call the the Google Places API to get a json object about the client. The form of the result is mentioned in [1]. The server which receives a WHATSAT message checks its records for the specified client and returns with an AT message which has the information about the client. The position (latitude, longitude) is extracted from this and sent to google places API as as a url query. The query response waits for a deferred object by implementing a callback. The AT response message (server's reply to the WHATSAT command) is generated by the **parseGoogleResults function**.

4 Execution and Debugging

I have written a shell script to run all the servers. Debugging can be done through log files generated (server-"server name.log").

A shell script to start all servers:

```
for i in "Alford"
                "Bolden"
                "Parker"
                "Welsh"
                "Hamilton"
do
  echo Starting $i
  python serverfinal.py $i &
done
```

4.1 Python-based Approach vs. a Java-based Approach

Python uses reference counting to free objects as soon as they have no references i.e., most of the time everything is cleaned up by reference counting and the garbage collector never kicks in. There is also usually no need to tune various garbage collection parameters as in Java.

Object reference cycles might cause problems in a pure reference counting based garbage collector. In this project, I utilized a reference cycle in the following way to facilitate inter-server communication.

The server keeps a dictionary of the details of other servers and maps it with a twisted client object. The server maintains connections with the other servers through a client object. An obvious approach would be: a connection is terminated once the server sends the required AT message and a server attempts to reconnect when server to server connection is lost. But this causes significant overhead in addition to concurrency issues. In this project I maintain the server-to-server connection through the mapping created via the client object.

Moreover, Python has a reference counting based garbage collector with reference cycle detection, and nullifying one node in the cycle will not cause any memory leak.

5 Twisted: Ease of Prototyping

Twisted is well documented and I found it extremely feasible to come up with a prototype. In comparison to C's socket programming, twisted provides encapsulation for listen, read, write, connect operations. Twisted also exposes callbacks making it easier to prototype. Built-in functions like connectionMade, connectionLost also are quite helpful as in soket programming the programmer would have to manually handle those operation while also taking into account any I/O operations. Twisted also adopts a simplified development model. There are two main ways of deploying applications: TAC files and plugins. TAC files are simpler but less extensible, making them ideal for simple server deployments that want to take advantage of built-in features like logging and daemonization. Plugins have a higher initial development overhead but lay out a clear API for extending the application. Plugins are ideal for applications that need a stable interface for third-party developers or more control over plugin discovery and loading[4].

6 Twisted vs Node.js

Node.js is a JavaScript runtime that has gained significant momentum since the release of the V8 JavaScript engine. Node.JS installs a runtime into the OS, and can be executed from console. Node.js has features like non-blocking, event-driven IO model which is similar to twisted. For instance, the package "net" has support for listen, connect, read and write. Hence for the development of protocols there shouldn't be much problem as Node.js is as good as twisted, since both avoid thread-based networking, which is relatively inefficient and very difficult to use. The only difference between twisted and Node.js in this aspect, is perhaps the fact that Node.js treats the event loop as a language construct instead of a library. Node.js also has a support for concurrency for insistent programmers, through the cluster module of the child_process.fork() API[5].

Scalability of both are not yet fully tested with each claiming to be more scalable than the other. However since Node.js is evolving at a much faster pace than twisted, it might be the case that Node.js is more scalable for larger deployments in the immediate future.

If node.js is chosen as the language of choice for prototyping the server herd, the notion of class disappears (Since JavaScript has no notion of class.) So in place of deriving classes from libraries we should define prototypes of a client and a server. Other advantage that I see is the fact that functions exist as objects in JavaScript which makes the development of a prototype easier.

7 Conclusion

I would conclude that while twisted enables easy prototyping of custom protocols, node.js would also do a decent job at the same.

8 References

[1]http://web.cs.ucla.edu/classes/winter16/cs131/hw/pr.html
[2]http://twistedmatrix.com/trac/browser/tags/releases/twisted-15.5.0/twisted/internet/protocol.py#L496
[3]https://en.wikipedia.org/wiki/Flooding_%28computer_networking%29
[4]Twisted Network Programming Essentials 2nd Edition
[5]https://nodejs.org/en/about/