Twisted Twitter proxy herd

Henry Yu
CS131, UCLA

Abstract

Twisted is a framework for writing asynchronous, event-driven network programs in Python. With its large utility library, and abstracting low-level system calls, it presents itself as a likely candidate for writing real-time web applications. We will build a prototype to investigate the effectiveness of the Twisted Framework. Analysis and tests including the analysis of the language it is written in reveal Twisted as a promising solution.

1. Introduction

Our goal is to find an appropriate platform for a web-based Wikipedia-style news service. The news service will have frequent updates to articles, access will be required via protocols and no just HTTP, and clients will tend to be more mobile. This will cause a bottleneck with the current implementation, and we desire to find an implementation that won't. It should have a fast enough response time to handle real-time updates.

With the number of connections in our application easily reaching thousands, a multithreaded approach will get complicated. Instead, we will use an event-driven network that manages the loop.

The Twisted framework is a possible solution. It uses 'Factory' class to allow multiple client connections without any extra effort in implementation. Its library supports a large number of protocols, including HTTP, IMAP, SSH, FTP, and many more. The server and client code are also very similar, making coding easier.

We will implement a simple prototype to test and gauge the suitability of the Twisted Framework. From this, we can access how well Twisted performs, how it can integrate, and whether language issues become a problem.

2. The Prototype

2.1 Description

Our prototype consists of proxy servers that can be easily parallelized. The server accepts incoming TCP connections from clients who send one line messages to either inform the server of its location (IAMAT command) or request tweets from the site, Twitter (WHATSAT command). There are a total of 5 servers, and if a client tells one server where its at, it will propagate that information to all the other servers via a flood algorithm. Each server only talks to certain servers so that if a server is down, there is a chance it will not propagate to all the servers.

Our sample application contains 5 servers. The servers' names are Blake, Bryant, Gasol, Howard and Mecca. The servers have a set rule on who they communicate to. Blake talks with everybody but Gasol and Metta. Gasol talks with Bryant and with Howard. Metta talks with Bryant. These relationships are two-way. When a message is received by a server, it uses the appropriate method to process the type of message with its Server Protocol.. For the "IAMAT" message, the server will store or update the client's location in its database. Afterwards, it will propagate the information to the servers that it talks to by using its Client Protocol. If the message is "WHATSAT" message, the server consults its database for the requested client's location, and then retrieves nearby tweets to send.

Queries sent to twitter are wrapped in a 'Deferred' object, allowing the system to proceed without having to wait for a response. When a response is received, the deferred object's callback is fired, processing the response in accordance.

2.2 Results

With the Twisted's built-in libraries and structure, this prototype becomes extremely simple to implement. The only thing we need to implement is the functionality specific to our application. In this case, it's a matter of writing methods of processing incoming messages. Using the library for both the Server protocols, and Client protocols, it becomes easy to make the-

se servers listen and send messages. With respect to writing code, this is as ideal as it gets.

In our sample prototype, the Twisted's LineReceiver class handles all the low-level calls for listening to port, and relaying a single line message for us to parse. It also gives us a Client Protocol that makes it easy to send a message. The built-in methods allow for us to overload the basic events, and program the functionality we want. This allows use to easily specify how to handle them.

In addition, the Twisted's 'Factory' class automatically declares Protocol objects for each successful connection. As a result, multiple clients can connect to a server on the same port without any additional efforts.

Another major advantage in Twisted's framework is the simplicity of using 'Deferred' objects to handle blocking functions. The headache of having to manage multiple threads, or the tolerance of dealing with inefficient use of system resources is eliminated.

With the general issues of asynchronous programming managed by the framework, we have to concentrate on the issues specific to our application. We must consider how the parallel servers are to share information with issues in storing, retrieving, and updating our data quickly and efficiently.

3. Further Discussion

The servers in our prototype should be a parallelizable proxy. This way, many servers can be utilized to load-balance a large number of clients. Currently our approach is to have each server maintain its own copy of the database. Each time a server receives a new or updated information, it tells and "floods" the information to all the servers. Another approach is to rely on a "Master Server" that retains the complete database. Then there are "Sub-servers" that would then query to this Master Server for data if it doesn't contain the requested information in its own local database.

There are major drawbacks to both of these approaches. In the approach that is not what we did with a Master Server, there is a high chance of additional delay in retrieving information since it may not be held locally. Additionally, the Master Server may be serving a large amounts of sub-servers, which would make a heavy burden for the server to update, maintain, and relay information.

Our approach is easy to implement but there are drawbacks in storage and speed. There is a potential for a large database to be replicated unnecessarily. The larger problem is the constant traffic to communicate to other servers. For each client update, there should be n - 1 connections made, where n is the number of servers in order to propagate the information through.

Between these two, we chose the one without a Master Server. The reasoning is that getting tweets is much faster. Also, the memory issue isn't a big issue with today's memory.

4. Twisted Performance

Twisted's structure and libraries are well-suited for our purpose. Its separation of the transport and application layers allow us to focus our efforts on the functionality of our application without having to worry about lower-level networking details. Running some simple tests with the prototype shows that most requests take less than one second.

(3pm)	2 max_tweets	100 max_tweets
6 concurrent requests	.387-.518	.406-.563
1 single request	0.468	0.538

(3am)	2 max_tweets	100 max_tweets
6 concurrent requests	.084-.132	.096.135
1 single request	0.099	0.104

These tests were small scale but there are some things to notice:

1. Network traffic due to time of day is a much larger factor in response time than increase size of data transfer, or an increase in concurrent users.

2. A large increase in data size accounts for a small increase in response time.

3. An increase in concurrent users does not appear to have a noticeable effect on response time.

With network traffic being the largest factor in response time, it looks promising that our application server is not a bottleneck.

With respect to data transfer size, these initial results seem as expected. We expect a small and reasonable increase in response time with relatively large increases in data size. With respect to a large number of concurrent users, this framework looks promising.

5. Language Issues

There are issues associated with Python that we should look into since Twisted is a framework written in Python for Python.

1.) Maintainability: Python presents a major advantage in this area. It is extremely easy to read and write. It is a object oriented, which provides modularity.

2.) Efficiency: This can be a disadvantage in choosing Python. Due to it being an interpreted language, dynamic type-checking, and other factors, it is widely accepted that on average, Python runs slower than C++.

3.) Multithreading: Multithreading in Python is generally safe due to its implementation of the Global Interpreter Lock.

Since Python has dynamic type checking, we see in increase in run time speeds. However, much of the slowdown is due to major data processing. In our application, we are not doing a lot of data processing, but transferring data. Slowdown due to this should then be minimal.

Another disadvantage to Python's dynamic type-checking is the need for extra code to catch exceptions. We saw this in our prototype code, such as the extra '+' sign in latitude/longitude input. This is however trivial in comparison to C pointers or multi-threading.

Another efficiency consideration involves the memory management. Python's build-in memory management provides safety at the cost of some memory overhead. This extra memory becomes less of a concern these days with cheap memory.

6. Conclusion

The twisted framework offers a multitude of important advantages. Although there are some minor drawbacks regarding processor speeds, our examination concludes that these effects should be relatively minimal. The advantages are the extreme ease and minimal cost for the development. Our investigation and analysis concludes that Twisted is a promising and elegant solution to our application.