Evaluation of Python and the Twisted Framework

Jun Zhou, 804179311

Abstract:

The goal of this project was to prototype a proxy server herd to query an API and send data back to clients. We used Twisted, a framework that provides event-driven networking in Python. This report will evaluate Twisted 16.5.0 as a framework, and Python 2.7.12 as a language, for use in this type of application, including a brief comparison with Node.js. Special attention will be paid to issues like ease of implementation, reliability, type checking, memory management, etc.

Intro:

Our final application will be a news service application where the concerns are 1) frequent article updates, 2) access via various protocols not limited to HTTP, and 3) mobile clients. As an alternative to the Wikimedia architecture running on LAMP (Linux, Apache, MySQL, PHP), we are considering an application server herd, which we think will better suit these needs. The initial prototype described in this report involves a server herd of 5 servers, with a flooding algorithm to update each other on most recent client locations. The prototype uses Twisted, along with the Google Places API, to test out the desired functionalities and evaluate whether we should use Twisted in our final application.

Twisted:

Twisted is an event driven framework which abstracts away a lot of low level networking and TCP complexity. This makes it quick to write, and great for prototyping. The main classes are the Factory and the Protocol, along with some basic derived classes for more specific use cases. Users can build upon these classes to implement their own functionality.

Classes

A Factory holds persistent server information, including the server name/ID and stored client information. I also used the factory to open up a log file based on the serverID, with name serverID + ".log" for each server. Since the factory is persistent, the file pointer to the log is stored here.

A Protocol defines what happens each time a connection is made. It is destroyed once the connection closes, so it is not good for holding data. My implementation derives from the LineReceiver protocol class, which grabs a line at a time from the incoming connection and processes it. The lineReceived function parses the line. For lines starting with "IAMAT," the server recognizes a client location update. It stores the new client info in the factory.clientInfo dictionary if the timestamp is more recent than any previous info received from the client. It then sends back an AT response and propagates the response to its neighboring servers. The neighbors are known from a global dictionary of ALL_NEIGHBORS.

Creating the connection to the neighboring servers was the only slightly clunky part of the implementation. Using Twisted, to make a server which can also receive data like a client, I had to include another ClientFactory class and a ClientProtocol. To send data to the neighbor's port from the current server in the propagate function, the current server needed to create a ClientFactory with a ClientProtocol representing the connection, then invoke connectTCP to connect. Meanwhile, the neighbor would have to be running its own ServerFactory and ServerProtocol in another instance to recieve the data.

On receiving a WHATSAT query, the server would parse the client name being queried, look up the client's most recent location in its factory's clientInfo dictionary, and query the Google Places API for that location and the radius specified. Results would be returned in a json format.

On receiving an AT from another server, the server would have to check whether the location information was more recent than the location it already had, based on timestamp. If so, update the clientInfo dictionary and continue propagating the AT message to neighbors. Otherwise, ignore the message and stop propagating. This way, the flooding algorithm can stop properly once information becomes out of date.

Reactor

Calling connectTCP connects a client to a port. Calling listenTCP connects a protocol factory to a listening port. Calling reactor.run() starts Twisted's main event loop. The reactor listens for certain events and triggers callback functions which will handle the events. Events include attempted connections, building protocols, and handling lines received.

The main reactor handles all connections on a single thread. Thus, it needs to be nonblocking and event-driven.

Some drawbacks of a single threaded event-driven approach are the inability to use recursion and the need for a state machine to. Also, callbacks should be finished quickly, persistent state cannot be kept in function local variables. Much of this is simply handled in Twisted's code, so the programmer using Twisted doesn't need to worry about these aspects, with the exception of the callback model. Though Twisted abstracts away a lot of these issues, the programmer should try to understand the callback approach and avoid using outside function calls that will block.

Specifically, in my implementation, the call to Google Places API goes through Twisted's getPage(), which returns a deferred object. This means that the program will continue executing until the object is complete, ie the API call has returned, rather than halting execution to wait for a potentially slow API call. Once it is done, my callback function handlejson will be run to process the json returned by Google and return the response to the client.

Despite these complications, the idea behind Twisted's approach is that an event-driven framework is still simpler to write and use (and often more robust) compared to a multithreaded one. The use of callbacks also reduces the wait cost of using a single thread, allowing the program to continue running until a deferred object is ready to be processed.

Evaluation

Overall, Twisted makes it simple to write a quick prototype via abstraction. It implements a lot of low level functionality, and the only thing the programmer needs to do is define some event handlers in classes. This is much easier than socket programming in C, for instance. The ease of use also contributes to reliability and robustness.

Returning to the three considerations of the application: 1) frequent article updates, 2) access via various protocols not limited to HTTP, and 3) mobile clients, it also seems that Twisted is well suited to address the second issue. It includes support for HTTP, TCP, FTP, SSH, DNS, and others. Overall, it seems to be a robust and well-developed framework which already implements many protocols and can be adapted to different needs.

Prototype Testing:

Testing results can be found in the logs for each server as well as the test script. I started up each server with 'python2 server.py <servername>' in the start.sh script and sent an initial client message to Alford in the test.sh script. Alford stored this information and sent a reply back, and propagated the reply to its neighbors. A WHATSAT query to Ball returned a json, showing that the information propagated. The flooding algorithm also correctly stopped when passing in client info with an older timestamp. I then tested nodes going down by shutting down Holiday and sending a location update to Hamilton. This time, a query to Hamilton returned updated results, but all the other servers still had the old data.

Overall, it was feasible to get a prototype up and running in Twisted to implement a flooding algorithm.

Python:

More broadly, it's also important to consider whether Python is a good choice for the application. An advantage of Python is ease of use. Code is concise readable, and maintainable, especially when combined with Twisted's abstractions. Furthermore, CPython's garbage collector reduces memory management concerns for the programmer. However, it does not actually detect circular references which have a __del__ method, since objects may refer to each other or other objects within the __del__. Programmers should still deal with these objects manually, possibly by using weak references, which allow memory to be reclaiming even if references remain, as long as all remaining references are weak.

On the other hand, Python's dynamic type checking may have drawbacks. Though dynamic type checking allows for more concise code, the code must be thoroughly tested to catch every error, since errors may not be caught until attempting to execute a piece of code. Furthermore, runtime checks slow down execution. Although it is easier to write a networking application on top of Twisted with Python, compared to sockets in C, the C program will generally run faster.

Comparison with Node.js:

Node.js is a framework for Javascript providing similar functionality as Twisted. It is also single threaded and event driven, but written in low level C/C++ and runs on the Chrome V8 engine. Due to these details, Node may be faster than Twisted, which is written in Python for Python.

However, Node.js is newer and therefore possibly less robust. Twisted has been well tested, has decent documentation, and includes implementations for a lot of different protocols, not limited to HTTP, though Node.js also includes several libraries for HTTP, SSL, TCP, etc.

Since they are used for different languages, I think ultimately the decision may come down to language preference and speed requirements, depending on the application. However, since Python is not too difficult to learn and efficient to write, it may still be worth trying to use Twisted with Python.

Conclusion and Recommendation:

I think Twisted is a good choice for the application. It's easy to implement, supports many protocols, and feasible for implementing a proxy server herd to handle client location updates. In terms of speed, it may not be the absolute optimal choice, but the tradeoff is development time and maintainability. Depending on language preference, Node.js is also a viable option.

References:

[1] *Circular References in Python*, http://engineering.hearsaysocial.com/2013/06/16/circular-references-in-python/

[2] *The Twisted Network Framework*, http://legacy.python.org/workshops/2002-02/papers/09/index.htm

[3] *Twisted*, http://www.aosabook.org/en/twisted.html

[4] *Twisted Docs*, https://twistedmatrix.com/documents/16.5.0/index.html

[5] *What is Node.js?*, http://radar.oreilly.com/2011/07/what-is-node.html

[6] *Why I'm Switching From Python To NodeJS*, https://blog.geekforbrains.com/why-im-switching-from-python-to-nodejs-1fbc17dc797a#.48toj6qq1