Proxy Herd Server Using Twisted

Ziheng Zhou
University of California, Los Angeles
May 29, 2015

Abstract

This report examines the features of Twisted framework, which is an event-driven networking framework using Python, in order to evaluate whether it can be a good alternative to Wikimedia web structure.

1 Introduction

The motivation of this paper comes from the concern that the old web architecture taken by Wikimedia, which is the classic LAMP, standing for Linux, Apache, MySQL, and PHP is not a good fit in the situation where updates becomes much more often and the access is not only limited to HTTP. Especially when clients tend to be more mobile, their location information is updated in real-time. LAMP structure is too slow for this situation, because for non-event-driven structure like LAMP, the way to boost the efficiency to handle large amount of requests after the use of load balancer and master/slave server structure, is to use multithread. But multithreading has its innate drawbacks that become another bottleneck: 1, the context switching among threads are very expensive – it takes a lot of I/O. 2, to prevent race-condition among threads it introduces locks, which is a huge performance bottleneck too.

2 Twisted

2.1 Overview

Twisted is an event-driven network engine that supports many different protocols. Most common protocols are supported so that people basically don't need to worry about protocol compatibility. At the core of Twisted is a reactor that responds to different events and take the time for waiting for any event (usually because of performing I/O) to do some other tasks, so that no task is blocking. It is achieved by pushing off the blocking calls to deferred objects, which is a placeholder for the future.

2.2 Language Features

Since Twisted in a framework built on Python, its syntax is that of Python's – concise and expressive. And it inherits all the features of Python language too. Python is an interpreted language instead of compiled. It uses indentation to define control flow, making it easy to read and fast to type. Python has a dynamic type system, so that programmers don't have to worry about what type to define, since it will just figure out itself. Also Python is an objective-oriented language in its blood. Everything is object – from class, function, to strings and even integers. This provides a lot of flexibility to pass it around and convert its type. Beside the features of Python itself, Twisted provides very high-level API interfaces too. It's very straightforward to call any functions you need. It takes very few lines of code to implement a simple server.

2.3 Pros and Cons

Twisted has many good features. As stated above, it inherits all the great things from Python. Programming in Python provides better experience than most other languages – I am talking from an objective perspective. And because Python is a high-level language, it can be immune to most common classes of security flaws in network software, particularly the notorious and hackers' favorite "buffer overflow". Also thanks to Python's error handling mechanisms, the Twisted server is extremely stable. It doesn't need much explanation on how much it worth to have server that doesn't crash. Another good side is that since Twisted's codebase is small and it contains many hooks for dynamic content, and with the source code and an open license, it gives users a great flexibility to add features that they want.

The bad side can be the steep learning curve on the notion of deferred objects and call back classes. There have been some examples of real life projects that switch out of Twisted because new beginners tend to make many mistakes because of this. Another weakness of Twisted nests on Python – it is a very slow lan-

guage. But the thing is, the usage of Twisted is never about low-level operations, so this much slowness doesn't matter too much here.

3 Server Herd Project

3.1 Overview

This project implements 5 servers, each being an instantiation of ServerFactory, which is a subclass of twisted.internet.protocol.Factory I customized. The ServerFactory instantiates the protocol I wrote every time a connection is made. The reactor listens on a specified port for TCP connections and then invokes ServerFactory.

3.2 Protocol

I write two protocols, the server protocol and the client protocol. The main one is server protocol, and the client protocol (and its corresponding factory) is only invoked when a server tries to send message to another server, since in this situation the sending server acts as a client to the receiving server. The client factory only sends a message then it will closes itself. Both protocols are built on top of LineReceiver that dispatches the LineReceived method for each line. For the server protocol, under this method, it checks the first token of the command received, which indicates the type of this command, and then it calls the corresponding handler to handle this command. If it's not a valid command, it will return the message unprocessed with a question mark.

3.2.1 IAMAT Handler

This handler first parses the line received into different tokens like user name, geo-location and timestamp. It will update this user information stored in this server (stored in the factory), and then propagates this information to its neighbors by acting as a client. Of course, it will give the command sender a response too.

3.2.2 AT Handler

This handler is for processing the message sent by other servers that is intent to update user's information. This handler parses the message and compares its timestamp with the last update time stored in its own server. If this updating message is newer than that stored in its own server, it will update its own data and keep propagating to other servers. If not, it will do nothing. In this way, it can always guarantee the newest data will be propagated to every server and will stop when every server is updated. And because the connection between servers is TCP, we don't need to worry about update message lost during the propagation. The only catch about this approach in comparison to implementing a minimum spanning tree that decides what server to send to before sending is that, for every server, it will have the neighbor-number amount of extra connections set up for each new update. But the good thing is, it's so simple.

3.2.3 WHATSAT Handler

This handler will process clients' requests for places information near the geo-location of some certain client specified by the name in the request message. It will parse the message, get the name of the demanded client, and use Google Places API to fetch the places information with the geo-location of this client stored in the server. The client request also specifies a maximum amount of places information it wants. Sometime the amount of places we get from Google will exceed this limit, and Google doesn't provide the ability to filter the extra info, so I have to implement a filter system by converting the raw info to JSON data, truncating extra items, and converting the JSON data back to strings that can be returned to client.

3.3 Logging

I use the logging module provided by Python standard library. It names the log file with the server name and the time the server started. And it logs the connection activities and input/output data of this server.

3.4 Run the server

We use a normal Python call to start the server. It takes the name of the server as its argument.

python myserver.py <server name>

3.5 Testing

I wrote a shell script start_server.sh to start all 5 servers. Then in test.sh, after started all the server, I send one IAMAT message to Alford to check if the message get forward to all servers without looping. Then it kills Powell server and sends a different IAMAT to Alford to check if Bolden still receives. And then it kills Parker, and send IAMAT message again. At this moment Bolden and Hamilton should not receive any message from other servers. Lastly it sends a WHATSAT message to

Hamilton to check how if it receives correct places info. Finally it kills all the servers.

4 Comparison to Node.js

Node.js has a package manager: npm, which makes adding a third party package a lot easier. And considering the community of Node.js is exploding, this is a big win. And also because Node.js is written by Javascript, this makes programming both backend and frontend in same language. It's a big advantage when you consider how many syntax mistakes people make when they switch programming backend in Python to frontend in Javascript. The simplicity saves developers a lot of troubles and cognitive efforts. It's a huge win actually in real world. Beside, people say Node.js is faster because of the JV8 engine from Google, but some people say it's not really the case.

On the otherside, Twisted is a much more mature framework, so it has better documentation, less bugs and less missing pieces, while Node.js is still at its youth age, rough and sloppy in some configurations. But youth will grow into a real man eventually. So it's just a matter of time now. So in general Twisted provides a much more reliable experience and more complete capacity.

5 References

[1] http://twistedmatrix.com/trac/wiki/TwistedAdvantage

[2] http://krondo.com/?p=1209

[3] https://lionfacelemonface.wordpress.com/2012/01/30/node-js-vs-twisted-a-scrum-tool-built-in-both-introductions-first-opnions/

[3] http://www.quora.com/What-are-the-benefits-of-developing-in-Node-js-versus-Python