Twisted Places proxy herd

Muhammad Ali Gulzar
University of California, Los Angeles

Abstract

This paper presents an investigative report on using Twisted; an event driven networking engine and using it as a proxy herd for Wikimedia style architecture where server and clients communicate with each other to exchange information. Furthermore this paper covers the programming aspects of Twisted including the advantages and disadvantages of using Twisted for these kinds of applications rather than self-created Java or C application. I'll also cover the usability of Twisted, ease of implementing a particular design, language choice of Twisted and its ability to scale with the given scaling requirements these days.

1. Introduction

The basic idea of this work comes from the Wikimedia Architecture, which uses a LAMP platform based on GNU/Linux, Apache, MySQL and PHP. It comprised of several servers and an interface to help with load balancing for the sake of performance and reliability. With multiple servers at disposal, a failure in one of them is not bound to halt the entire operation. The task at hand was to create a service like Wikimedia with three fundamental requirements in mind. The first one is that the update to articles happened more often in this framework; second one is that it won't follow any specific protocol like HTTP and the last one is the mobility of the end users. If we follow the given Wikimedia Architecture, the trivial problem with design would be the overloading at the application server. Since the mobile clients and other sever have to communicate the front-end application server that would create a bottleneck on the whole response flow of the client request.

2. Approach

2.1. Application Server Herd

The approach we are following in this paper is "Application Server Herd" and to simulate the environment we designed a simple "Proxy Service for Google Places". This application server herd architecture consists of a set of servers that communicate with each other to exchange information they don't know and each of them also exchanges information with clients at the same time. In other words each server handles requests from both servers and clients. The server communication map is also defined at startup that restricts the servers' communications with each other to avoid any policy non-adherence. From the clients perspective they have a choice to connect to any of the server without facing any frontend application service that might be potential bottleneck. In the context of our example, clients send their GPS location to one of the server; the server they are connected to and server record that location in its database. Similarly the server responds to WHATSAT request by looking at the ID's location from the local memory and then asks Google Places for the places nearby. The server then filters the results form Google and send a filtered set of results back to client. In case where the requested ID is not handled by this server, an AT request is generated and sent to other neighboring servers who will then look for the ID in their database if found they will return the location and the queried server will handle the Google Places call and response to client. If even these servers could not find the ID they can further broadcast this AT message to other servers and so on.

This kind of Architecture eliminates the need of a front-end Application server that causes a bottleneck in the web service.

3. Implementation

3.1. Twisted: Event Driven Prog.

The basic idea behind twisted networking framework is event driven programming. This model differs a lot from the conventional client server model where each party had to perform a defined set of messaging to make other

do a specific task. But in case of event driven programming, a server can invoke an event on the client's side and make it perform that task.

A user usually implements those events at both client side and server side and at the time of registering they exchange the context of their event classes and each side can now use those contexts to execute certain events on the other side. The serve/client will handle these events with the same behavior as implemented by the user.

3.2. Network Configuration

In this project I used the same network configurations stated in the requirements. The project actually includes only one file *server.py* that can be run under twisted to perform the entire set of required task.

twistd –y server.py

As this script is invoked a network is set up with 5 servers; Alford, Hamilton, Powell, Parker and Bolden. The configuration of this network is defined as a Map Graph, which shows the allowed connections. One can changes this graph to configure the network. There is also a port-mapping object that assigns ports to a particular server that can too be configured if needed.

Once this script is invoked, all the servers start listening on the assigned port. Any new client can connect to any of the server and exchanges messages with that.

3.3. Server

A server uses the implemented *Factory* API of Twisted to create a server's clients pool. I have overridden the basic constructor of the class and a method called *buildProtocol*. When ever a client is trying to connect to the server the *buildProtocol* event is called on server that tells the server that a client is trying to connect. The server handles this by enrolling the client into its registered list and attaches a context of Client that implements the *LineReceiver* API. The server also keeps track of the number of clients and the name assigned to that client. Also each server logs the messages it has received and messages it has sent to a log file, named as *servers.log*

3.4. Client API

This class handles all the incoming messages from the client and neighboring servers. It has a constructor that sets up the local variables of the class and the *connectionMade* method that handles the event of connection being made and enrolls that particular client in the users list of that server. The *connectionLost* handles the lost connection event and delete the user from the registered users list. The most important handler is the *LineReceived* method then handles the incoming message event. This is where all the important things happen. For every incoming message, I first apply sanity checks to see if it complies with all the rules and parse it to see which category it belongs to.

3.5. IAMAT, WHATSAT and AT

There are only three sets of messages that I needed to take care of in this project. The first one is the IAMAT request from the client. This message reports the GPS coordinates location of that particular client. Along with latitudes and longitudes it also reports the time and ID of the client. Server upon receiving this message enrolls the data into its database and updates the location. In my implementation it also sends this information to the other neighboring servers using the AT protocol so that they also have the client's location that is not connected to them.

WHATSAT requests ask the server for top n Places with a radius r to a particular client's location given its ID. The server handles this request by looking up the coordinates in the database and then use them with radius to query Google Places API for the nearby places. The result is sent back to the client. Since we are broadcasting each client location at IAMAT I do not need to query another server.

AT request is just an informative message to other servers that tells them about the location of a particular client and it also used as an acknowledgement message to clients.

4. Evaluation

4.1. Twisted and Python

Since twisted is implemented in python it inherits all the advantages and disadvantages of python.

Python is maintainable. It is easy to write and read. We don't really need to take care of the types while writing code even list can have different types of elements in it. The forced indentation really helps with the understanding. Syntax is highly abstract and high level and has a lot of syntactic sugar. One thing I haves noticed in python is that the development in python is relatively fast then other languages, you don't need to take care of types or brackets and implementation is very compact as compare to other languages. Other advantage I see is the readily available documentation and large community.

Some of the disadvantage of python is also its dynamic typing that makes it slow at runtime where the interpreter needs to check every time that if the types are coherent. Being a scripting language makes is slower than compiled languages.

Using python in twisted really makes it easier to implement. The learning curve is not that steep. To work with twisted one need to understand the inheritance model, which looks intuitive if you have prior experience in object oriented programming.

4.2. Twisted Itself

I think of twisted as stable and highly intuitive framework for networking applications. The event driven architecture really makes sense in client server scenario. The implementation in twisted is really simple up to a certain point. In the context of this project it was extremely simple and easy to implement the server make in run and talk to clients. But in the larger perspective I don't think that twisted is good enough for the large scale cloud programs. Although Twisted take care of the multithreading by itself, but there are sometimes when you need to take care of that by yourself so that you are not limited to the design offered by framework. Lack of interference with threads makes it very easy to implement simple things but at the same time makes it hard to do the non-conventional things required by the application design.

One major point to consider while thinking in the context of large-scale applications is the performance. These kind of frameworks are very nice to maintain and handle and the code size is small that cater a large variety of features but all of these benefits comes at the cost of performance. Python is itself is slower than C/C++ and then there comes the overhead of the framework. With millions of requests to handle per second I think that Twisted is not capable enough. It hides the intricate details of the system, which is good, but sometimes these details need to be tweaked to get the appropriate performance.

Comparing Twisted to Nodejs, Nodejs is known to have great performance inherited from Javascript, which can scale but makes it single threaded only. The implementation in NodeJs might a bit tricky. Although the performance must better than Twisted but I would prefer Twisted for smaller scale applications because of the ease of implementation and C/C++ for large scale services so that we have more low level control over the system resources.

5. Conclusion

As said earlier I think Twisted is very much suitable for up to medium scale applications because of its easier implementation and maintainability, abstraction and event driven framework. But for large-scale applications where performance is crucial we can let go of ease of implementation and use low-level languages like C/C++ to have more control over the resources we have to achieve better performance.

Appendix

Server Log File

```
DEBUG:root:Alfred Initiated

DEBUG:root:Powell Initiated

DEBUG:root:Hamilton Initiated

DEBUG:root:Bolden Initiated

DEBUG:root:Parker Initiated

DEBUG:root:Alfred's number of Users: 0

DEBUG:root:Hamilton's number of Users: 0

DEBUG:root:Alfred Recieved => IAMAT kiwi.cs.ucla.edu +34.068930-118.445127 1400794645.392014450

DEBUG:root:Powell's number of Users: 0

DEBUG:root:Parker's number of Users: 0

DEBUG:root:Parker Recieved => AT Alfred 24829041.45713544 kiwi.cs.ucla.edu 34.068930 -118.445127
1400794645.39201450

DEBUG:root:Parker boradcasting location to Bolden

DEBUG:root:Parker boradcasting location to Hamilton

DEBUG:root:Powell Recieved => AT Alfred 24829041.45713544 kiwi.cs.ucla.edu 34.068930 -118.445127
1400794645.39201450

DEBUG:root:Powell boradcasting location to Bolden

DEBUG:root:Bolden's number of Users: 0

DEBUG:root:Hamilton's number of Users: 1

DEBUG:root:Bolden Recieved => AT Alfred 24829041.45713544 kiwi.cs.ucla.edu 34.068930 -118.445127
1400794645.39201450

DEBUG:root:Bolden boradcasting location to Parker

DEBUG:root:Bolden boradcasting location to Powell

DEBUG:root:Hamilton Recieved => AT Alfred 24829041.45713544 kiwi.cs.ucla.edu 34.068930 -
118.445127 1400794645.39201450

DEBUG:root:Hamilton boradcasting location to Parker

DEBUG:root:Bolden's number of Users: 1

DEBUG:root:Powell's number of Users: 1

DEBUG:root:Parker's number of Users: 1

DEBUG:root:Bolden Recieved => AT Alfred 24829041.45713544 kiwi.cs.ucla.edu 34.068930 -118.445127
1400794645.39201450

DEBUG:root:Parker's number of Users: 2

DEBUG:root:Parker Recieved => AT Alfred 24829041.45713544 kiwi.cs.ucla.edu 34.068930 -118.445127
1400794645.39201450

DEBUG:root:Powell Recieved => AT Alfred 24829041.45713544 kiwi.cs.ucla.edu 34.068930 -118.445127
1400794645.39201450

DEBUG:root:Parker Recieved => AT Alfred 24829041.45713544 kiwi.cs.ucla.edu 34.068930 -118.445127
1400794645.39201450

DEBUG:root:Hamilton Recieved => WHATSAT kiwi.cs.ucla.edu 10 5
```