Implementing Application Server Herds in Python with Twisted

Abstract

Application servers may be able to communicate more efficiently through an application server herd where multiple servers talk directly to each other. Traditionally servers would store data into a database or some single data store where other servers would go to for their data. But in this experiment, we are eliminating that need by allowing servers to communicate directly though an event-driven approach taken by Twisted (a Python library).

Objective:

The goal of this project was to see if using Twisted is a good choice to implement an application server herd. The idea behind inter-server communication is to reduce the amount of database calls and network connections which can become a huge bottleneck in any application.

Approach:

A prototype was built to simulate frequent inter-server communication in the form of multiple clients declaring their latitude and longitude to a server. The client information would propagate to all other servers which the original server is allowed to communicate to. The various functionalities allowed included IAMAT (client declares location), AT (server propagation of client information), and WHATISAT (API call about a specific location).

Implementation:

A server would be instantiated and connect to some port number based off of predefined server names and could run in parallel. Their implementation extended off of Twisted's, protocol.ServerFactory which took care of connection handling. Thus when a client-server connection is made or lost, the programmer is able to implement additional functionality to those events. A client was extended off Twisted's protocol.ClientFactory. A client connection was made by sending a message from a client to a certain server's port. Because all servers were listening on their ports, they would receive any messages clients send to them and could react accordingly. In my implementation, clients sent messages through the telnet command. Through the command prompt, I would form a client-server connection by running telnet localhost <port_num> which would allow a long-lasting connection between the client and the server with that port_num.

Pros of Using Twisted:

Twisted is an event-driven library that allows a user to extend off many protocols relating to server communication. It can handle thousands of connections on a single thread. Twisted allows for continuous client connections to a server which can be beneficial for geolocation based applications where location updates frequently. This is good so that an application does not have to frequently form and end connections and can maintain a long polling connection/socket. Twisted does not block the single main thread it runs on by implementing underlying protocols with callbacks and asynchronous calls on background threads.

Another positive feature of using Twisted is that it requires the use of Python which

is a very easy language to prototype and quickly develop with if a product wants to be launched as soon as possible. Python has a lot of external libraries to support developers' needs. Thus it was not too difficult to prototype the application server herd and have all the networking issues abstracted away.

Python also has the benefit of including garbage collection so that developers don't have to manually allocate and free memory. It will depend on reference count to determine if memory will be freed, so this is a positive to save on developer time.

Cons of Using Twisted:

A problem with using Twisted is that is runs on a single thread, so altering the implementation of our program could require much more effort to make it run in parallel.

Another problem with using Twisted is that Python itself has dynamic typing. A team of developers could run into trouble if using Python because types of not clear and could be misused and even changed during runtime. A Java implementation would not run into this problem since Java is a statically typed language. But this type confusion could be fixed by clear variable naming conventions. Another issue with Python is that its garbage collection can cause some overhead in freeing and allocating memory. If the machine's memory is nearing it's full capacity then the Python Virtual Machine may free a large queued up chunk of memory that is no longer used, which may stall the program execution momentarily.

Another last consequence of using Python is that Python is a slow language. Compared to Java, it is further from machine code and is an interpreted language thus its runtime is slower than that of Java's.

Conclusion:

Overall, Twisted is a good choice for our application server herd. It easily allows us to extend off the client-server communication protocol and to use external APIs in our program. Although there are some cons in using Python as our language of choice, those can be overcome if developers adhere to good naming practices and clean code. Although Python is slow, this tradeoff is balanced out by how quick and easy it is to develop applications compared to if other more difficult languages were used such as Java or C++. Thus Twisted and Python is a good framework for our application's needs.

Alternative Approaches:

Node.js is another very popular framework to handle event-driven applications. Twisted's approach is quite similar to node.js in that it can detect client connections and abstract the client-server connection. A significant pro that node.js has over the Pythonic approach is that since JavaScript is used, it is meant for the web and can be easily put on a website for nontechnical users to interface with. Node.js is a specialized web architecture for this client-server connection whereas Python also allows for this connection but is not meant to be hosted online. Node.js is a good alternative to using Twisted, especially if you want to host the application on the internet.