CS 131 Project: Twisted places proxy herd

Abstract: Twisted is an event-driven networking framework for Python. It is used in this project to prototype a proxy server herd application. A proxy server herd of 5 servers is created to handle location updates and query what is at a given location. The behavior of the server herd is also logged. This report will discuss the design and implementation using Twisted and compare it to Node.js.

1. Introduction

The LAMP stack uses multiple redundant web servers behind a load balancing virtual router to improve performance and reliability. However, in the case were we might be making frequency updates, access to various protocols other than HTTP , and when clients are mobile the application server will be a bottleneck. An application server herd has multiple application servers that communicate directly to each other and have a core database and cache which could perform better in this case. Twisted is an event-driven networking framework that we will use to implement a proxy server herd in Python and see how feasible this idea is in comparison to the Java based approach.

2. Design

Twisted enables us to use its ClientFactory, ServerFactory and LineReciever classes to build up the server herd. There are 5 servers that make up the proxy server herd. We also utilize callback functions to support the asynchronous behavior. Clients can tell the servers where they are via the IAMAT command, servers respond and communicate with each other telling what is at a position via the AT command, and the client can query what is near a location via the WHATSAT command.

2.1 Nodes in server herd

Each node in the server herd is a server. We have 5 of them: Alford, Ball, Hamilton, Holiday and Welsh. They talk to each other as follows:

- Alford talks to Hamilton and Welsh
- Ball talks to Welsh and Holiday
- Hamilton talks to Alford and Holiday
- Holiday talks to Hamilton and Ball
- Welsh talks to Alford and Ball

Whenever a server receives a location update it floods the entire herd by propagating the message to its neighboring servers. The message is only propagated if the client's timestamp is larger than the one stored in the server. Servers do not propagate place information only; if a server wants place information it should call the Google Places API. The herd continues to work even if one or more of the servers are disconnected/shutdown.

2.2 Protocol

The LineReceiver protocol was used to handle input to the server. The input line was passed on to another function that checked if the input had valid syntax and then called the appropriate command handler. If the command was incorrect a question mark followed by a space and then the invalid command would be returned.

2.3 IAMAT

The IAMAT command is used by the client to tell the server where it is. Its operands are a client ID, latitude and longitude in decimal degrees (ISO 6709 notation) and the time the client sent the message (POSIX). The server responds with an AT command. Example:
IAMAT kiwi.cs.ucla.edu +34.068930-118.445127 1479413884.392014450

2.4 AT

The AT command is used by the server to respond to IAMAT commands. Its operands are the ID of the server that responded to the client, the time difference between when the server got the message and the client's time stamp, and the remaining fields are a copy of the IAMAT data.

Example:
AT Alford +0.263873386 kiwi.cs.ucla.edu
+34.068930-118.445127 1479413884.392014450

2.5 WHATSAT

Clients can query for information about places near a location with this command. The arguments are the name of a client, a radius from the location (in km) and an upper bound on how many locations to see. The Google Places API is used to query for the places at the location. Therefore the radius has an upper limit of 50km and max of 20 places that can be returned.
Example:
WHATSAT kiwi.cs.ucla.edu 10 1

The server responds with an AT message followed by a JSON-format message containing the places of interest followed by two newlines. The format of the JSON is exactly the same as that of the Google Places API response.
Example (some details replaced by "..."):
AT Alford +806008.632277 kiwi.cs.ucla.edu
+34.068930-118.445127 1479413884.392014450

```
{
   "status": "OK",
   "html_attributions": [],
   "results": [
      {
         "name": "Westwood Plaza",
         "reference": "CmRbAAAA_U... ",
         "geometry": {
            "location": {
               "lat": 34.0689117,
               "lng": -118.4449861
            }
         },
         ...
      }
   ]
}
```

2.6 Logging

The python logging module is used to track the behavior of the servers. All major events such as new and dropped connections, sent messages, received messages, and errors are logged in output files called servername.log (where servername is the name of the server).

2.7 Testing

A shell script was written (test.sh) that starts up all 5 servers. It behaves as follows:

1. IAMAT message is sent to Alford
2. Welsh is killed
3. IAMAT message is sent to Alford
4. Holiday is killed
5. IAMAT message sent to Ball
6. WHATSAT message is sent to Hamilton
7. The rest of the servers are killed

3 Discussion

3.1 Feasibility of Using Twisted to Implement An Application Server Herd

It is very feasible to prototype a server herd using Twisted in Python. The classes supplied by Twisted, such as ClientFactory, ServerFactory and LineReciever, make it easy to create the servers and the server herd. These functions also make it easy to understand and break up the code according to its functionality. Furthermore, the callback functions enable us to properly execute our desired behavior asynchronously.

3.2 Python vs. Java

Type Checking:

- Java uses static type checking, forcing the user to define the type of a variable when declaring it. This is helpful in catching errors.
- Python uses dynamic typing which enables the programmer to just use variables without caring for what its type is. This duck typing makes code easier to write by reducing work. Thus prototyping would be easier to write quickly.

Memory Management & Garbage Collection:

- Java uses a heap and the nursery and generation method. It also uses a mark and sweep algorithm when doing garbage collection.
- It depends on implementation (Cpython, pypy, iron python, jython), but new versions of python use reference counting, and if that fails, they fall back on mark and sweep.
    - Circular references can occur (if it depends on stack methods) but Python can handle this by looking at outer references

Therefore Python's form of garbace collection may be preffered, but this depends on which implementation of Python is being used. Java will work too but is not preferred for real-time systems as there will be a performance hit. Therefore for the server herd prototype Python is a better choice in this regard.

Multi-threading->context-switching issues:

- Java processes can use multiple cores, multithreading, and hyper threading but there are no safety guarantees in the language itself. However, Java has numerous libraries with thread safe data structures.
- Python behavior depends on the implementation, some have the Global Interpreter Lock (GIL) while others may not
    - CPython has the GIL and uses only one CPU level thread for python code. However, code that calls thread safe libraries release the GIL and allow it to execute code that runs on multiple cores
    - Python does have queues, locks, semaphores, and other objects that can be used when writing multithreaded code.

Therefore Java could potentially run faster than Python (depending on implementation and what we are trying to do). Thus it might be better to use Java rather than python from this viewpoint as Java is more straightforward.

3.2 Twisted vs. Node.js (Python vs. JavaScript)

It may be of similar to difficulty to prototype the serverherd in Node.js as it was using Twisted. Node.js is also event-driven and non-blocking like Twisted. The APIs also seem to be similar as Node.js has functions to listen for input, handle connection and disconnecting, and also has callbacks. Python has PIP while Node.js has NPM which many developers seem to prefer. Some developers claim to also find more up to date modules on NPM. However, there seems to be no standard for handling callbacks, promises, and errors in Node.js. Development seems to be easy for smaller scale programs in Node.js but gets complex with larger programs, although others claim otherwise. Python seems to be a bit easier to code with and many developers are probably more familiar with it than Node.js but again this isn't necessarily true as some programmers may find Node.js easier to use for their purposes and are more familiar with it. However, Node.js is leaner than Python.

Node.js is prototypal and classes are traditionally not in it, thus the Twisted classes like ServerFactory would have to be replaced with a prototype of it. Thus the style of it would differ from using Twisted in Python. I am unable to tell which of these two would be better for the purpose of creating a server herd. Most of the research I have found seems to be opinionated and I won't be able to truly see which is better unless I prototyped in Node.js. I also don't have much prior knowledge of Node.js so personally it may be more difficult than using Twisted.

4 Conclusion

Twisted in Python is a feasible method to prototype server herds. It is easy to write code to do this and it works successfully. We could potentially use Java but overall Python seems to be easier to write code for and works well for our purposes. Multithreading can also be accomplished in Python if the right implementation of it is used and there are libraries that support multithreaded behavior. Node.js is a possible alternative to Twisted and theoretically it

may be faster than Twisted. However, a prototype would have to be built using Node.js to get some conclusive results. Thus, overall Twisted in Python is a good solution for implementing a server herd and will definitely work much better than the typical web server model described in the spec such as the Wikimedia Architecture.

5 References

1. Project. Twisted Places proxy herd, http://web.cs.ucla.edu/classes/fall16/cs131/hw/pr.html
2. Understanding Memory Management, https://docs.oracle.com/cd/E13150_01/jrockit_jvm/jrockit/geninfo/diagnos/garbage_collect.html
3. Python Garbage Collection, http://www.digi.com/wiki/developer/index.php/Python_Garbage_Collection
4. Multi-threading in Python vs Java, https://mail.python.org/pipermail/python-list/2013-October/657460.html
5. Why I'm Switching From Python To Node.js,https://blog.geekforbrains.com/why-im-switching-from-python-to-nodejs-1fbc17dc797a#.cqrwcffdh
6. After A Year Of Using Node.js In Production, https://blog.geekforbrains.com/after-a-year-of-using-nodejs-in-production-78eecef1f65a#.dso3xeu4b
7. Node.js v7.2.0 Documentation, https://nodejs.org/api/net.html
8. Prototypal Inheritance, https://howtonode.org/prototypical-inheritance