Analysis of Using Twisted as a Framework of Application Server Herd

Jun Ma
University of California, Los Angeles

Abstract

The goal of this paper is to discuss the feasibility of Twisted as a candidate for Application Server Herd. After evaluate Twisted from type checking, memory management, and comparing to multi-thread approach and Node.js, I come to the conclusion that Twisted is one of the best choice for Application Server Herd.

1. Introduction

The motivation of the paper is to look into a different architecture called an "application server herd", where the multiple application servers communicate directly to each other as well as via the core database and caches. The idea is that if a user posts its GPS location to any one of the application servers, then other servers will learn of the location after one or two inter-server transmissions, without having to talk to database.

Since commonly speaking, listen on a specific port and data I/O are block operation, so in order to support parallel connection (i.e. multiple clients connect to the server at same time or the server communicate with other servers at same time), there are two basic approaches. One is to use multi-threading strategy, the other is to use asynchronous programming (usually single thread, event driven). Twisted framework is one representation of event-driven programming, and a good candidate for Application Server Herd.

1.1. Python

Before talk about Twisted, I'll start with something about Python, because Twisted is implemented by Python. The merits that we can benefit from Python for this project is that Python is a dynamic type (no need to declare before use), dynamic type checking (type check is done at runtime), strong type (no type conversions) script language. Because of this, I can assign values easily by just using following syntax:

```
host, port = ("localhost", 10234)
```

It also makes string parsing become super easy. Python is also objective oriented. I can easily inherent class from Twisted package and override functions to satisfy my unique requirement. In terms of this application, I can easily create new protocol to satisfy my need benefit by inheriting helper protocol in Twisted.

1.2. Twisted

Twisted is a robust, cross-platform implementation of Reactor Pattern. Reactor Pattern is an asynchronous approach that use a loop which waits for events to happen, and then handles events using callback functions. The new Twisted approach to handle callbacks is using deferred object, which is an encapsulation of a chain of callback functions that you'd use after receiving a result. The way we understand the result should be "an asynchronous result" or more general, "a result has not yet come". I'll go through in details how I use deferred object to solve communication between servers later.

2. Twisted Approach to Server Herd

There are three request methods I need to handle for Server Herd protocol: IAMAT, WHATSAT and AT. The communication pattern is following:

> Farmar talks with everybody but Gasol and Hill.
> Gasol talks with Meeks and with Young.
> Hill talks with Meeks.

I'll explain my implementation of server protocol and server communication separately. I implement a BaseServer.py file, and other 5 server files in each folder, which import everything from BaseServer. To run the server, simply using "twistd -n -y Gasol.py".

2.1. Protocol Implementation

There are two parts of my Server Protocol. One is major protocol class, which inherits LineReceiver class. The other is Server Protocol Factory, which contains data structures and basic parameters that my server protocol

will use, including server name, log file name, dictionary to save clients information and peers information.

For IAMAT request, I'll treat each one server received as new request and update clients' information dictionary. Then flood the information to peers.

For WHATSAT request, each server will check their client dictionary to see if it has the required client ID. If the server have the client's information, it will response tweets. Since I failed to figure out how to get tweets in required format (I actually can got the tweets using TwitterSearch package, but this package cannot set the limitation number of results), I decided to hard code the tweets and reply to client.

For AT request, I firstly check if the client information contains in the request is new or not (based on sending time). If it is new, then I'll record it and flood it to peers. This strategy effectively prevent flooding overhead since only new information will be flooded.

2.2. Communication Implementation

In terms of communication between server herd, especially when trying to reconnect, I heavily depend on using deferred. To set up a connection, I use TCP4ClientEndpoint to generate an endpoint of destination, and then call connect() function, which return a deferred object. Then I add a callback function which send data to the target endpoint. If destination server is not available, then the callback function will return an error. Then I add an errback() function to redo the connection again. In this way, I can keep trying to reconnect to target server. This approach will not cost much resource, since the most valuable thing about asynchronous programming is that it will only do things when there are resources available (I actually monitor the CPU and memory cost).

In previous version of Twisted, this reconnection policy can be done by creating a new protocol which inherits Reconnection class, and then using reactor to set up the connection and reconnection. But this deferred approach is super cool that all things can be done in 5 lines. Deferred wins.

3. Analysis and Comparison

In this part I will analysis the reliability of Twisted as framework of Application Server Herd, specifically in type checking, memory management and multi-threading. Then I'll make a brief comparison with Node.js approach.

3.1. Type Checking

As mentioned before, Python is dynamic type checking, which means it checks type error at run time. This is not good for servers because once the server is running, you may not want it down unexpectedly. But this downside can be overcome by careful programming and using try-catch block.

3.2. Memory Management

Python handle memory automatically, it has own garbage collection implemented. Python does a lot of allocations and deallocations. All objects, including "simple" types like integers and floats, are stored on the heap. Python interpreter uses pymalloc, which designed specifically to handle large numbers of small allocations. Any object that is smaller than 256 bytes uses this allocator, while anything larger uses the system's malloc. This implementation never returns memory to the operating system. Instead, it holds on to it in case it is needed again. This is efficient when it is used again in a short time, but is wasteful if a long time passes before it is needed.

To some extends, this will cause problem when running Twisted server for a long time, because small object will keep consuming memory and eventually it will cause memory leak. But for Application Server Herd, this will not be a problem. The feature of Server Herd is that it flood information, that is, even if one server is down, it won't cause big trouble because data can still transmit along other path to destination. And when the server is reconnect, it can receive missed data based on my reconnection implementation. So the simple solution to memory management downside is to restart servers one by one in specific order, that is, don't shut down all servers in the same time.

3.3. Multi-threading

Twisted approach to asynchronous programming is handling event loop using callback functions but not multi-threading. One drawback about Twisted or generally event loop approaching is that it consumes CPU resource all the time, while multi-threading approach would only create new thread when there is new connection. The other is that if there is a fatal error happened within event loop, the whole server will hangs, while this would only affect a single thread in multi-threading approach. But those drawbacks should not be an issue for Server Herd, because each server is light-weighted and distributed, it won't consume much re-

source, and even if one server is down, the whole system will still function well.

One important thing that event-driven approach beats multi-threading approach is that event-driven approach is easy to use. You don't need to bother shared resource, synchronous I/O and scheduling which are must be thoroughly cared in multi-threading programming.

3.4. Twisted vs. Node.js

Twisted and Node.js are all using event-driven approach, they are similar in many ways. But after doing some research about node.js, I found following differences. 1. Node.js is said faster and more flexible than Python Twisted. The reason is that Node.js focus on low level event-driven implementation, just like the reason C is faster than Java. 2. Python is easier to understand and maintain than Javascript, because Javascript is designed for simple task at very beginning. 3. Python is more mature and has more other library to use, like beautifulsoup, one of my favorate library to analysis html file.

4. Conclusion

To summary above facts, I claim that event-driven programming is the best fit for Application Server Herd, and Python Twisted is easier to learn and use than Node.js. So Twisted is the best choice for this project.

5. References

[1]http://twistedsphinx.funsize.net/projects/core/howto/deferredindepth.html

[2]http://krondo.com/?page_id=1327

[3]http://stackoverflow.com/questions/3461549/what-are-the-use-cases-of-node-js-vs-twisted