Server Herd Management with Twisted in Python

Jules Robins — University of California Los Angeles

ABSTRACT

A server herd allows an application to handle larger volumes of client communication by enabling such optimizations as allowing load balancing between multiple servers and cutting down on latency by having clients access more local servers. All the while data is shared amongst the herd in order to ensure the same behavior as one would get from a single server.

This paper analyzes the applicability of the Twisted framework in Python to this sort of application, and briefly compares it with the possibility of implementing the same setup using NodeJS. The primary focus is upon language-specific issues such as Python's handling of type checking, multithreading, and memory management.

Introduction

A server herd functions by taking client connections to any one of a number of different servers. The servers maintain homogeneous data structures by sending messages to one another via telnet or a similar technology, or through a more direct local connection. This approach generally requires each server to be associated with a database that it keeps up to date with new information from clients and from other servers.

In order to share this data the servers need a way to determine when to send data to one another. Which server can connect to which other server is likely to be a factor outside of the developer's control, based on factors such as different service providers managing different servers or divisions between more open front-facing servers and more secure backend servers. Therefore any solution should be able to accommodate any group of servers and connections that can be represented as a connected graph.

Flooding Algorithms

The simplest solution to this conundrum is what's known as a flooding algorithm. The core idea is to have each server relay the changes to its database to each other server its connected to [1].

Of course, any actual implementation requires additional restrictions because otherwise every message will persist in the system indefinitely as any two connected servers will keep sending the message back and forth. Including data on the identity of the originating server might avert some instances of the problem, but still allows for cycles down the line.

For that reason the implementation accompanying this write-up includes time stamps on outgoing messages and each server only updates its database and forwards messages with newer timestamps than the previous data it had for a given user (as well as the first time it receives data on a new user.

Twisted

The Twisted framework is an event-driven networking engine coded in the Python programming language [2]. That is to say, Twisted is asynchronous and therefore avoids hanging while waiting for a process, but it's still possible to ensure that parts of the code execute at the appropriate time because the built-in functions are called upon detection of an event, for example, when a connection is lost.

This is highly advantageous for an application such as managing a server heard because each server needs to remain responsive while handling multiple clients and updating the other servers its connected to. Asynchronicity is key here because otherwise the application would be bottlenecked by which server was handling synchronization, but synchronicity within specific events can be key.

For instance, upon receiving new information about a client's location, the server must verify that the data is newer than the client's last stored check-in before storing the new one. An arbitrarily asynchronous system might result in a server receiving data, doing a check, then receiving another packet of data, doing that check, updating its database, and then finally updating its database with the first packet received even if it was less recent than the second packet received. However all of these operations are bundled into a single event and so are handled synchronously relative to one another. With Twisted, whatever order the packets are received in, the end result will be that storage contains the newer data.

Using Twisted effectively requires an understanding of this reactor loop model (pictured in Figure 1). Twisted maintains an event queue, pushing each new event into the queue and handling the next event by popping off the oldest item in the queue. This model allows inter-server communication to be handled as just one more event and affords that process the same necessary assurances that the client-facing processes get.

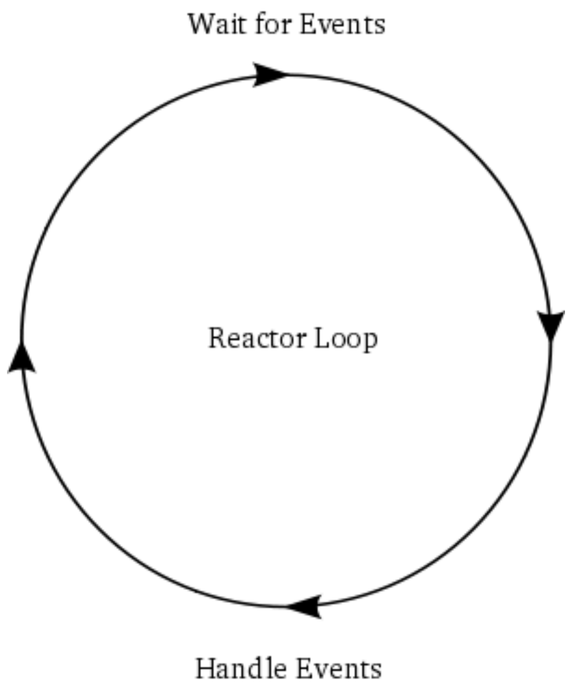

Figure 1: The Twisted Reactor Loop [3].

Clearly, Twisted itself has desirable features for such an application, but no piece of an implementation stands in isolation. Twisted is set up to handle a multitude of protocols from Telnet to DNS to RPC [1], but it locks the development team into using Python, an object oriented-interpreted language. In order to determine whether Twisted is a good technology choice for this application, one must determine whether Python is the right language to write it in.

Python

Python brings a few issues to the table, but by far the most salient is one of performance. How does Python handle multithreading? The short answer is: it doesn't.

Python is an interpreted language, so rather than being compiled beforehand like Java, the code is fed into an interpreter as it runs. And the Python interpreter is globally locked. Simply put, no matter how many threads a Python function spawns, or how many cores are idling, all of the threads will execute sequentially. The CPU can switch back and forth between them, but it won't achieve true parallelization [4].

However this limitation isn't quite the death knell it sounds like because Python *can* parallelize multiple processes using the multiprocessing library [5]; this is a somewhat more limited capability, but for systems such as UNIX on which it is very cheap to spawn a new process, this implementation can result in similar performance gains to Java's multithreading [5]. Twisted is particularly well adapted to this approach because each item in the event queue can be implemented as a separate process since they are, by definition, self contained.

Python fares somewhat better in other respects. Its memory management model uses reference counting alongside less frequent generational garbage collection [6], which allows the system to keep sessions around only so long as they are being used rather than having to manually monitor the inter-server connections in case one goes out. Of course, Java does garbage collection as well, but because it forgoes reference counting, Java always has to use the more resource-intensive true garbage collection and thus has worse performance on large enough domains.

That leaves one more major point of comparison to Java: type checking. Java does static type checking at compile time, whereas Python checks at runtime. This makes Java less prone to having programmer errors manifest in production, but also requires prior compilation which makes it less feasible to release regular small patches to the application.

NodeJS

NodeJS is built with many of the same principles as Twisted, but in JavaScript rather than Python. It's asynchronous, event-driven, and uses multiple cores by spawning child processes rather than multithreading. That's not to say they developed with no underlying philosophical differences; for instance, where Twisted requires the programmer to start the event loop it's all built on top of, functionality from the NodeJS library automatically begins the loop and exits it when all of the callbacks in the program have been executed. Nevertheless, the two frameworks have enough common ground that the biggest difference between them is the languages they're built upon.

JavaScript is the purview of web browsers, full of native interaction with HTML elements and built with the server-client relationship at its core. That's fantastic for potential extensions of this sort of application, such as building out a graphical user interface in a browser, but it adds a lot of bulk to the syntax which makes JavaScript for back end applications considerably harder to read than its Python equivalent. Or, at least, it often is.

JavaScript doesn't fit as neatly into a box as most programming languages. Is it compiled or interpreted? Well, Mozilla touts it as an interpreted language [7], but is often used with just-in-time compilation [8]. Similarly, while it's advertised as being object oriented, JavaScript is also built out to support functional and imperative programming styles (though the line becomes fuzzy once one notes that JavaScript functions are objects unto themselves with their own attributes and methods). And really even the "object oriented" style doesn't mean the same thing that it would in C++ or Java. JavaScript objects aren't defined statically, but rather start empty and can be assigned additional attributes during execution.

JavaScript runs into the same pitfalls and boons that appear in Python from dynamic typing, most notably the added vulnerability to human error.

Conclusion

Ultimately Twisted and NodeJS overlap in most of their essential functionality. Either can be used effectively to maintain a server herd with multiple client faces, but they are not identical.

An implementation in Twisted will be easier to maintain. Python's syntax is much more readable and generally less verbose, but moreover the syntax makes not just individual statements more readable, but also blocks of code. There's no need to match parentheses when automatic indentation determines scope.

However, that concern often has to take a back seat to efficiency. Python simply wasn't designed to run quickly, and it doesn't. That would be fine for a system that's not under heavy load, but the fact that this application is being evaluated for a server herd in the first place means that there are too many clients for one server to handle efficiently. Ultimately the question becomes one of whether to prioritize minimizing the number of servers in the system, or the development team's time. In a highly successful venture, the latter has to win out, and Twisted is the best option available for that purpose.

References

[1] *Flooding (computer networking)*, http://en.wikipedia.org/wiki/Flooding_%28computer_networking%29

[2] McKellar, Jessica and Abe Fettig, *Twisted network programming essentials*, 2nd edition, O'Reilly (2013), ISBN 978-1-4493-2611-1.

[3] jathanism, *Twisted Python getPage*, http://stackoverflow.com/questions/2671780/twisted-python-getpage

[4] Knupp, Jeff. *Python's Hardest Problem*, http://www.jeffknupp.com/blog/2012/03/31/pythons-hardest-problem/

[5] Python Software Foundation, *multiprocessing — Process-based "threading" interface*, https://docs.python.org/2/library/multiprocessing.html#module-multiprocessing

[6] *Python Garbage Collection*, http://www.digi.com/wiki/developer/index.php/Python_Garbage_Collection

[7] *JavaScript*, https://developer.mozilla.org/en-US/docs/Web/JavaScript

[8] *JavaScript*, http://en.wikipedia.org/wiki/JavaScript