Analysis of the Twisted Networking Engine in proxy herd server applications

Anderson Huang
University of California, Los Angeles

Abstract

Twisted is a networking engine written in the Python programming language and. As an event-driven programming framework, Twisted supports an array of protocols - TCP, UDP, and HTTP to name a few. In this paper, we explore the feasibility of using the Twisted framework as the basis of an application server herd application, one in which servers actively communicate with each other in addition to caches and databases. More specifically, we dissect the effect of Python's type checking, memory management, and multithreading. We conclude by comparing Twisted's approach to a Java-based approach to a solution as well as comparing Twisted to Node.js, an event-driven framework written in Javascript.

1. Introduction

Wikipedia is based on the Wikimedia Architecture, which makes use of the LAMP platform, a combination of Linux, Apache, MySQL, and PHP. Although such a system works well in practice, we explore a scenario with three changes: more frequent updates, expansion to include networking protocols other than HTTP, and increased client mobility. Factoring in these issues, we strive to examine the benefit of using Twisted's framework in implementing an application server herd. In this kind of design, server to server intercommunication ensures propagation of changing data, whereas the database will house more stable data. For this project, we implemented a simple application revolving around geographical location updates between five servers and a variable number of clients. We also make use of the Google Places API in querying for nearby information. In high-level terms, clients may announce their locations to servers and query for nearby locations to other clients. Servers flood updates to each other, and reply to client queries. This is the basis for our project.

To do our analysis, we will start by examining the Python programming language, which Twisted is written in, proceed to compare Java-based and Node.js-based approaches to the same problem, and conclude by determining the feasibility of using Twisted in such an arena in practice.

2. Overview of language features in Python

Twisted's codebase is written in Python, and reaps many of the benefits of Python. It's codebase is small, and resulting applications are stable and secure. This section aims to examine several key features relevant to implementing an application server herd, and their related impacts on performance: dynamic type checking, memory management, and multi-threading.

2.1. Dynamic Type Checking

Python shuns static type checking, which checks for errors and bugs at compile time, before the program is run, for dynamic checking, which does so during runtime. In this way, Python is more flexible with inputs because the type is basically not known until the program is run. This makes the implementation a lot simpler and results in less lines of code. However, because the type of variable is not known yet, there is a new class of bugs we must check for with try and except blocks, lest we risk flooding our servers with invalid requests resulting from invalid inputs. Despite this limitation, dynamic type checking results in more flexibility.

2.2. Memory Management

Memory management in Python is done via a private heap by the Python memory manager[4]. The manager itself is split into components that deal with different management aspects like sharing segmentation, pre-allocation, or caching. Management of the Python heap is performed by the interpreter, so the user has no control over it. There is no notion of a nursery as there is in Java, which makes garbage collection a simpler process in Python. Having direct access to memory management has no added benefit to an application server herd, so there is no advantage or disadvantage in this respect.

2.3. Multi-threading

Per the official CPython docs, a global interpreter lock ensures that only one thread executes Python bytecode at a time. This does not prevent threading, per se, but the GIL prevents making use of more than one CPU core or separate CPUs to run threads in parallel. Twisted has functionality that adds on top of this, using deferreds, that makes asynchronous callbacks possible. This makes Python an excellent candidate for this type of application.

3. Twisted Library and Implementation

As previously stated, Twisted is a Python library designed for creating event-driven applications. It is open-source and thoroughly tested, with extensive API documentation and examples.

3.1. Overview of Core Library Features

Twisted derives its functionality from an event loop called the reactor. The reactor listens to events and dispatches them to callback functions that are designed to trigger upon completion of the event[2].

Twisted is designed for complete separation between logical protocols and physical transport layers. The connection between these two abstractions happens at the last possible moment. Another central feature of the Twisted model is the concept of a deferred, or future. Twisted revolves around these objects, which is an instance of a class designed to handle data that has not been calculated yet, which may be because it is an I/O operation or network fetch. Deferred objects can be passed around, much like regular objects. This process forms the callback chain, and allows flexibility in working with not-yet-calculated values, because it essentially lets the user operate on not-yet-ready data. This is in stark contrast to non-event driven systems that use threads. The operating system must schedule threads to run and stop in accordance with blocking calls.

Twiste does all this callback business with one thread. Although this reduces the capacity for parallelism, this ensures reliability. Using a thread as a deferred source, deferred objects return immediately. Callbacks are attached to the main thread, which eliminates complex locking mechanisms. Reliability and ease of implementation are two hallmarks of the Twisted framework.

3.2. Twisted Server Herd Implementation

To implement the client-server model, we inherit from various Twisted classes. On top of this, we develop a protocol for both servers and clients. Each server and client implements the protocol as part of its functionality.

3.2.1. Classes

Both client and server have two classes relating to them. One is to instantiate instance variables, and the other is to handle protocol specifics. This leads to the four classes TwistedHerdServer, TwistedHerdServerProtocol, TwistedHerdClient, and TwistedHerdClientProtocol. Twisted has client and server classes from which our custom server and clients are derived. For both Client and Server Protocols, we inherit from Twisted's LineReceiver protocol, which simply parses lines for us.

3.2.2. Client-side

The client-side implementation is relatively simple. All the client has to do is generate a message and forward it to the server when the connection has been made.

3.2.3. Server-side

The bulk of the implementation lies server-side. Specifically, five servers exist: Alford, Ball, Hamilton, Holiday, and Welsh. Each server is configured with the TwistedHerdServerProtocol. Each server also keeps track of the number of connections it has open, its client list, propagates client information to each other, handles callbacks, and finally handles AT, IAMAT, and WHATSAT messages. Specifically, the server protocol handles *lines received* by overriding the class method of the LineReceiver protocol.

The mechanism for updates in the server-side is derived from keeping UNIX timestamps from the client's perspective. This way, even if there is skew on the server-side, the correct data is replaced because all servers can agree on the client timestamp. Client information is stored in its entirety in a client to data mapping inside the respective server.

Clients are allowed to about nearby landmarks near other clients. Our servers consult the Google Places API. Here, we utilize the callback feature of Twisted. As a query to Google's database may take an extended amount of time, we use a deferred object to capture the future value. We then attach a

callback to log these values back to the client. In this way, we do not block the application.

Lastly, server's flood updates to each other using a propagation mechanism. As the number of servers here is small, a primitive flooding algorithm implemented here potentially sends duplicate messages to the original sender.

3.2.4. Logging

The server herd application also supports logging. Most functions on the server-side log an event when it happens, even trivial ones. This aids immensely in debugging and backtracing.

4. Comparison to approaches in Java and Node.js

In this section, we compare the Python language features discussed earlier with Java, and the overall Twisted framework with Javascript's Node.js framework.

4.1. Java language features

Java is similar in some respects to Python. Like Python, a garbage collector handles memory management for the programmer, and Java compiles to bytecode. Its syntax is reminiscent of C and C++.

Unlike Python, Java implements static type checking, which detects errors and bugs in syntax or types much earlier in the program execution process[3]. Implementing a perfect static type checker is very difficult; for this project, Python's dynamic type checking results in a more flexible implementation.

Java supports full-fledged multi-threading, whereas Python is restricted by the global interpreter lock. Java, being a lower-level language than Python, has somewhat of a speed advantage, then. Java also has asynchronous call mechanisms, but Twisted develops deferred objetcts to counteract this deficiency.

Overall, Python results in less lines of code, but there may be a performance advantage of going with Java and using multi-threading.

4.2. Twisted versus Node.js

Per the official Node.js website, Node.js is a Javascript runtime built on Google's V8 Javascript engine[1]. With an event-driven, non-blocking I/O model, its structure duly resembles Twisted. It is open-source, and operates on a single thread, much like Twisted. No thread overhead is incurred, but no vertical scaling is allowed. There is immense support for Node.js in industry. Big names like Netflix, Yahoo, and Walmart use Ndoe.js.

Twisted has been around for longer, as Node.js only came into fruition in 2009, but it is safe to say Node.js has a much larger following. Node.js also makes it easier on the programmer as Javascript can be used both on the front-end and back-end.

With the advent of Ecmascript 6, or ES6, Javascript has becoming more and more object-oriented. With full-fledged classes and arrow functions, Node.js is becoming more and more advanced. Node.js also used dynamic type checking like Python, so both very loosely check types, which may or may not be a source of bugs.

5. Project Difficulties

Having never worked with Twisted prior, the learning curve was steep. A lot of time was poured into reading the documentation and perusing the example code. However, as Python is generally an easy-to-use language, after grasping the intricacies of Twisted, the project gradually became more understandable. Server communication was also an issue, but since the number of servers is quite small, an extremely efficient flooding algorithm was not the core issue.

6. Conclusion

As far as scalability goes, we recommend that Node.js is the way to go. For this example, Python's Twisted provided a good starting point: simple syntax, small source code, easy to understand. For large numbers of server, it seems that Node.js is garnering more respect and support. Node.js also supports better performance, through Google's V8 engine and multiprocessing. So for small applications, and for programmers already experienced with Python, Twisted is the engine of choice. For general-purpose and for scalability, Node.js is the engine of choice.

7. References

[1] Foundation, Node.js. "Node.js." Node.js. N.p., n.d. Web. 01 Dec. 2016.

[2] Jacobs, Jonathan. "Downloads." Twisted. Twisted Matrix, n.d. Web. 01 Dec. 2016.

[3] Java 101 By Jeff Friesen |Follow About | A Beginner's Library for Learning about Essential Java Programming Concepts, Syntax, APIs, and P.

"Java 101: The Essential Java Language Features Tour, Part 1." JavaWorld. JavaWorld, n.d. Web. 01 Dec. 2016.

[4] "Memory Management." Memory Management — Python 2.7.12 Documentation. Python Software Foundation, 20 Sept. 2016. Web. 01 Dec. 2016.