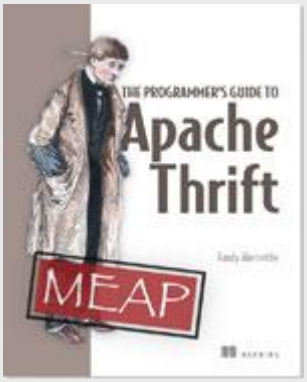

Features of Apache Thrift

By Randy Abernethy

In this article, excerpted from The Programmer's Guide to Apache Thrift, we'll discuss the key features of Apache Thrift.

There are several key benefits associated with using Apache Thrift to develop network services or perform cross language serialization tasks.

- Full SOA Implementation - Apache Thrift supplies a complete SOA solution
- Modularity - Apache Thrift supports plug-in serialization protocols and transports
- Performance - Apache Thrift is fast and efficient
- Reach - Apache Thrift supports a wide range of languages and platforms
- Flexibility - Apache Thrift supports interface evolution

Let's take a look at each of these features in turn.

Service Implementation

Services are modular application components that provide interfaces accessible over a network. Service interfaces are described in Apache Thrift using Interface Definition Language (IDL) (see Listing 1). The IDL can be compiled to generate stub code used to connect clients and servers in a wide range of languages.

For example, imagine you have a C++ module in a GUI application that tracks and computes sailing team statistics for the America's Cup. As it happens, your company's web development team would like to use the sail stats module to enhance a client facing web application, but the web site is written in PHP. To provide the sail stats features to the web dev team the sail stats module can be deployed as a network service.

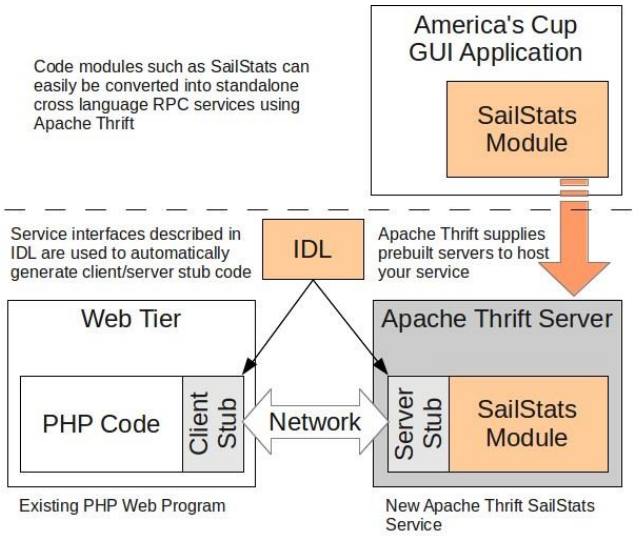

Figure 1 - Converting a module from a monolithic application (above dotted line) into a network service for a distributed application (below dotted line)

Microservices And Service Oriented Architecture (SOA)

The microservices and SOA approaches to distributed application design break applications down into services, which are remotely accessible autonomous modules composed of a set of closely related functions. SOA based systems generally provide their features over language agnostic interfaces, allowing clients to be constructed in the most appropriate language and on the most appropriate platform, independent of the service implementation. SOA services are typically stateless and loosely coupled, communicating with clients through a formal interface contract. SOA services may be internal to an organization or support clients across business boundaries.

Encapsulating the SailStats module in a SOA style service will make it easy for any part of the company's enterprise to access the service. There are several common ways to build SOA services using web-oriented technologies. However many of these would require the installation of web or application servers, possibly a material amount of additional coding, the use of HTTP communications schemes and text based data formats, which are broadly supported but not famous for being fast or compact.

Apache Thrift offers a compelling alternative. Using Apache Thrift IDL, we can define a service interface with the functions we want to expose. We can then use the Apache Thrift compiler to generate RPC code for our SailStats service in PHP and C++ (and most other commercially viable languages). The web team can now use code generated in their language

of choice to call the functions offered by the SailStats service, exactly as if the functions were defined locally (see Figure 1).

Apache Thrift also supplies a complete library of RPC servers. This means that you can use one of the powerful multithreaded servers provided by Apache Thrift to handle all of the server RPC processing and concurrency matters. Apache Thrift RPC servers are not only fast but they also have a much smaller footprint than most web application servers, making them suitable for many embedded systems.

Listing 1

```
service SailStats {
    double GetSailorRating(1: string SailorName)
    double GetTeamRating(1: string TeamName)
    double GetBoatRating(1: i64 BoatSerialNumber)
    list<string> GetSailorsOnTeam(1: string TeamName)
    list<string> GetSailorsRatedBetween(1: double MinRating,
                                        2: double MaxRating)
    string GetTeamCaptain(1: string TeamName)
}
```

In summary, to turn a code library or module into a high performance RPC service with Apache Thrift, all we need do is:

1. Define the service interface in IDL
2. Compile the IDL to generate client and server RPC stub code in the desired languages
3. On the client side call the remote functions as if they were local using the client stubs
4. On the Server side connect the server stubs to the desired functionality
5. Choose one of the prebuilt Apache Thrift servers to host the service

In exchange for a fairly small amount of work, we can turn almost any set of existing functions into a high performance Apache Thrift service, accessible from a broad range of client languages.

Modular Serialization

To make a function call from a client to a server, both client and server must agree on the representation of data exchanged. The typical approach to solving this problem is to select an interchange format and then to transform all data to be exchanged into this interchange format. The process of transforming data to and from an interchange format is called serialization.

The Apache Thrift framework provides a complete, modular, cross language serialization layer which supports RPC and stand alone serialization. Serialization frameworks make it easy to store data to disk for later retrieval by another application. For example, a service written in C that captures live earthquake data in a C struct could serialize this data to disk using Apache Thrift (see figure 3). The serialization process converts the C struct into a generic Apache Thrift serialized object. At a later time, a Ruby earthquake analysis application could use Apache Thrift to restore the serialized object. The serialization layer takes care of the various differences in data representation between the languages automatically.

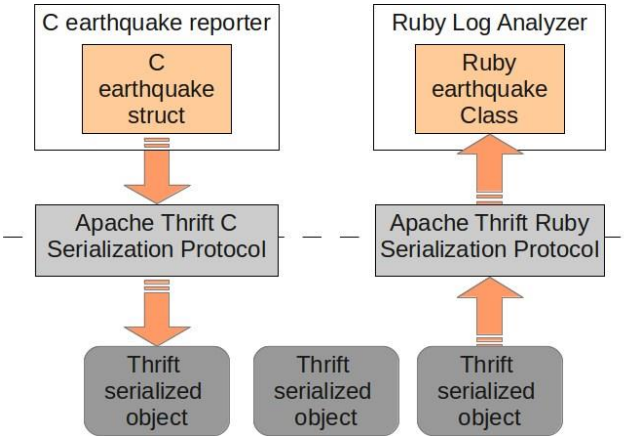

Figure 2 - Apache Thrift serialization protocols enable different programming languages to share abstract data types

A fairly unique feature of the Apache Thrift serialization framework is that it is not hard-wired to a single serialization protocol. The serialization layer provided by Apache Thrift is modular, making it possible to choose from an assortment of serialization protocols, or even to create custom serialization protocols. Out of the box, Apache Thrift supports an efficient binary serialization protocol, a compact protocol that reduces the size of serialized objects and a JSON protocol which provides broad interoperability with JavaScript and the web. A ZLib layer can also be added to provide high ratio compretion in some languages.

Performance

Apache Thrift is a good fit in many distributed computing settings, however it excels in the area of high performance backend services. The choice of prebuilt and custom protocols for serialization allows the application designer to choose the most appropriate serialization protocol for the needs of the application, balancing transmission size, speed, portability and human readability.

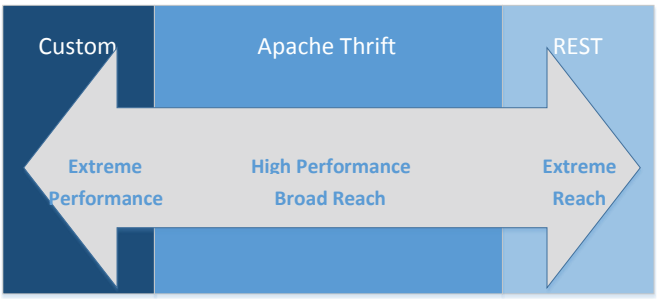

Figure 3 – Apache Thrift balances performance with reach and flexibility.

Apache Thrift supports compiled languages such as C, C++, Java and C#, which generally have a performance edge over interpreted languages. This allows performance-critical services to be built in the appropriate language while still providing interoperability with highly productive front end development languages.

Apache Thrift RPC servers are lightweight, performing only the task of hosting Apache Thrift services. A selection of servers is available in various languages giving application designers the flexibility to choose a concurrency model well suited to their application requirements. These servers are easy to deploy and load balance as standalone processes or within virtual machines or containers.

Apache Thrift covers a wide range of performance requirements in the spectrum between custom communications development on one end and REST on the other (see figure 3). The lightweight nature of Apache Thrift combined with a choice of efficient serialization protocols allows Apache Thrift to meet demanding performance requirements while offering support for an impressive breadth of languages and platforms.

Reach

The Apache Thrift framework supports a number of programming languages, operating systems and hardward platforms in both serialization and service capacities. Companies that are growing and changing rapidly need solutions that give teams the flexibility to integrate with new languages and platforms rapidly and with low friction. Apache Thrift can be a significant business advantage in such settings. Figure 4 illustrates the broad scope of environments within which Apache Thrift is often found.

Figure 4 - Apache Thrift is an effective solution in embedded, enterprise and web technology environments.

The table below provides a list of the languages currently supported directly by Apache Thrift. Note that support for C# enables other .Net/CLR languages, such as F#, VisualBasic and IronPython. By the same token, support for Java enables most JVM based languages to interoperate with Apache Thrift, including Scala, Clojure and Groovy. JavaScript support is provided for browser based applications and Node.js. Other projects found on the web expand this list further.

Table 1 - Languages supported by Apache Thrift

C	C++	C#	D
Delphi	Erlang	Go	Haskell
Haxe	Java	JavaScript	Lua
Objective-C	OCaml	Perl	PHP
Python	Ruby	Smalltalk	TypeScript

Apache Thrift supports these languages on a range of platforms including Windows, iOS, OS X, Linux, Android and many other Unix-like systems. Because Apache Thrift is compact and supports C/C++ and JavaME, it is often appropriate for embedded systems. Apache Thrift also supports HTTP[S], Webscoket and an array of web tech languages, including Perl, PHP, Python, Ruby and JavaScript, making it viable in web oriented environments. Few frameworks can supply the breadth of reach in languages and platforms offered by Apache Thrift.

Interface Evolution

Interface evolution is the process of changing the elements of an interface gradually over time. Modern IDL based systems like Apache Thrift make it possible to evolve interfaces without breaking interoperability with modules built around older versions of the interface.

For example, consider the previously described earthquake application where a C language program writes a C language struct to disk each time a tremor is reported. Let's assume that the earthquake struct contains fields for the date, time, position and magnitude. The interface evolution features of Apache Thrift allow new fields, say the earthquake's nearest city and state, to be added to the earthquake struct without breaking other applications reading the serialized data. The Ruby reporting program will continue to read old and new earthquake files, simply ignoring fields it does not recognize. Should the Ruby programmers require the new fields they may add support for them at their leisure, using default values when old files without the new fields are read.

Early RPC systems like SunRPC, DCE RPC, CORBA and MSRPC supplied little or no support for interface evolution. As platforms grow and requirements change, rigid interfaces can make it hard to extend and maintain RPC based services. Modern RPC systems such as Apache Thrift provide a number of features which allow interfaces to evolve over time without breaking compatibility with existing systems. Functions can be extended with new parameters, old parameters can be removed, and default values can be supplied. Properly applied these changes can be made without impacting peers using older versions of the interface.

Modern engineering sensibilities such as Microservices, Continuous Integration (CI) and Continuous Delivery (CD) require systems to support incremental improvements without impacting the rest of the platform. Systems that do not supply some form of interface evolution tend to "break the world" when changed. In such systems changing an interface means that all of the clients and servers using that interface must be rewritten and/or recompiled, then redeployed in a big bang. Apache Thrift interface evolution features allow multiple interface versions to coexist, making incremental updates simple and natural.