An overview of Apache THRIFT and its architecture

KARTHIK ANBAZHAGAN[1]

[1] *School of Informatics and Computing, Bloomington, IN 47408, U.S.A.*
[*] *Corresponding authors: kartanba@iu.edu*

April 4, 2017

Thrift [1] is a software framework developed at Facebook to expedite the development and implementation of efficient and scalable cross-language development services [2]. Its primary goal is to enable efficient and reliable communication across programming languages by abstracting the portions of each language that tend to require the most customization into a common library that is implemented in each language. This paper speculates the feasibility of using a such a system and looks deeper into the different ways you can use it and show how Apache Thrift can fulfill any needs with the flexibility it provides in choosing the different layers of the architecture separately. © 2017 https://creativecommons.org/licenses/. The authors verify that the text is not plagiarized.

Keywords: Cloud, Apache Thrift, cross-language, I524

https://github.com/cloudmesh/sp17-i524/tree/master/paper2/S17-IR-2008/report.pdf

This review document is provided for you to achieve your best. We have listed a number of obvious opportunities for improvement. When improving it, please keep this copy untouched and instead focus on improving report.tex. The review does not include all possible improvement suggestions and for each comment you may want to check if it applies elsewhere in the document.

The references in the abstract should be moved to the main text of the paper. Only use citations in the abstract if they are necessary for a specific statement that is crucial to the abstract. Neither reference included is like that.

Abstract: You say "This paper speculates the feasibility..." Your paper shouldn't speculate. It needs to look at the available information and research about Thrift and summarize it. There is no speculation involved.

The statement *...looks deeper into the different ways you can use it and show how Apache Thrift can fulfill any needs with the flexibility it provides in choosing the different layers of the architecture separately.* is too general. You can replace *Apache Thrift* with any other technology and the statement would probably still hold. Please rewrite and be more specific or simply omit.

Overall, you've done a good job of describing Thrift. However, you need to update the Architecture section since it includes some details that are unnecessary for an overview paper. In addition, there are parts of the paper that need to be rewritten to not use bullet points. Finally, if you want to make the claim that Thrift outperforms other similar systems you need to provide more evidence in a separate section that examines more comparisons and benchmarks.

Assessment: Changes required. Please take a look at the comments below for more detail.

CONTENTS

Don't use ToC. In this case it's not helpful since there aren't that many sections and everything is pretty clear at a glance. It only bloats the paper.

INTRODUCTION

Apache Thrift is an Interface Definition Language [3] (IDL) used to define and create services between numerous languages as a Remote Procedure Call (RPC). Its lightweight framework and support for cross-language communication makes

Grammar

it more robust and efficient compared to other RPC frameworks like SOA [4] (REST/SOAP).

You are using a relatively weak source to make a very strong statement. A blog entry is not peer-reviewed and represents someone's opinion. The blog entry you cite is well-written but shouldn't be used to make a blanket statement that Thrift "is more robust and efficient compared to other RPC frameworks". The job of your paper is to critically examine the evidence for the usefulness and robustness of Thrift.

It allows you to create services that are usable by numerous languages through a simple and straightforward IDL. It combines a software stack with a code generation engine to build services that works efficiently and seamlessly between $C++$, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C, Cocoa, JavaScript, Node.js, Smalltalk, and OCaml. In addition to interoperability, Thrift can be very efficient because of a serialization mechanism [5] which can save both space and time. In other words, Apache Thrift lets you create a service to send/receive data between two or more softwares that are written in completely different languages/platforms.

Try to use language that lessreader-centric, e.g. "allows you", "lets you", etc., to more Thrift-centric, e.g. "Thrift facilitates the creation of services...", "Thrift enables the creation of a service to..." You did this in many places already, but there are some sentences you can change.

Thrift was originally developed at Facebook and is one of the "core parts of their infrastructure".

This doesn't need to be quoted. It's a short and generic statement and can be used without quotes (or slightly paraphrased).

The choice of programming language at Facebook was based on what language was best suited for the task at hand. This flexibility resulted in difficulties when these applications needed to call one another and Facebook needed an application that could meet their needs of interoperability, transport efficiency, and simplicity. Out of this need, they developed efficient protocols and a service infrastructure which became Thrift. Facebook decided to make Thrift an Open Source and finally contributed it to Apache Software Foundation (ASF) in April 2007 in order to increase usage and development. Thrift was later released under the Apache 2.0 license.

Please include a reference for the statements in this paragraph. These are specific statements about Facebook's processes.

ARCHITECTURE

Figure. 1

Please refer to figures by their labels, rather than hard-code the figure number. That way if the order of the figures changes, you don't need to change any text.

shows the architecture of a model for using the Thrift Stack. It is very

Avoid using adverbs like "very"

essential to understand every component of the architecture to understand how Apache Thrift works. This section describes in brief about the components that are part of the Thrift architecture.

This is an unnecessary statement. Focus on the content rather than the paper.

It includes a complete stack for creating clients and servers. The top portion of the stack is the user generated code from the Thrift Client-Server definition file. The next layer of the framework are the Thrift generate client and processor codes which also comprises of data structures. The next two important layers are the protocol and transport layers which are part of the Thrift run-time libraries. This provides Thrift the freedom to define a service and change the protocol and transport without regenerating any code. Thrift includes a server infrastructure to tie the protocols and transports together. There are blocking, non-blocking, single and multi-threaded servers. The 'Physical' portion of the stack varies from stack to stack based on the language. For example, for Java and Python network I/O, the built-in libraries are leveraged by the Thrift library, while the C++ implementation uses its own custom implementation. Thrift allows users to choose independently between protocol, transport and server. With Thrift being originally developed in C++, Thrift has the greatest variation among these in the C++ implementation [6].

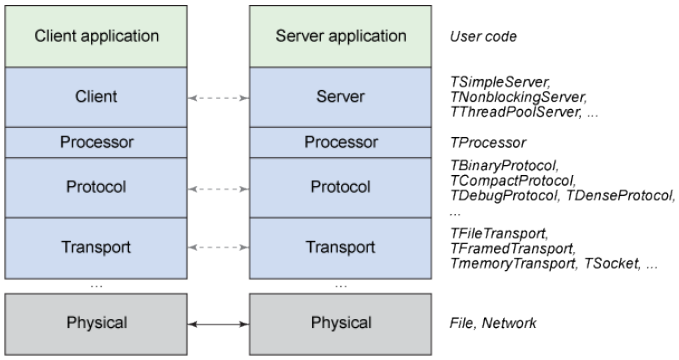

Fig. 1. Architecture of Apache Thrift [7]

Transport Layer

The transport layer provides simple abstraction for read/write to/from the network. Each language must have a common interface to transport bidirectional raw data. The transport layer describes how the data is transmitted. This layer seperates the underlying transport from the rest of the system, exposing only the following interface:

1. open: Opens the tranpsort
2. close: Closes the tranport
3. isOpen: Indicates whether the transport is open
4. read: Reads from the transport
5. write: Writes to the transport

6. flush: Forces any pending writes

This level of detail – listing specific functions – is not necessary for an overview paper like this. Please remove. Instead, just summarize what kind of operations the API support without listing them exhaustively.

There are multiple transports supported by Thrift:

1. **TSocket**: The TSocket class is implemented across all target languages. It provides a common, simple interface to a TCP/IP stream socket and uses blocking socket I/O for transport.
2. **TFramedTransport**: The TFramedTransport class transmits data with frame size headers for chunking optimization or non-blocking operation
3. **TFileTransport**: The TFileTransport is an abstraction of an on-disk file to a data stream. It can be used to write out a set of incoming Thrift requests to a file on disk
4. **TMemoryTransport**: Uses memory for I/O operations. For example, The Java implementation uses a simple ByteArrayOutput stream
5. **TZlibTransport**: Performs compression using zlib. It should be used in conjunction with another transport

Like above, don't list each transport, just describe in general terms what supported transport do.

Protocol Layer

The second major abstraction in Thrift is the separation of data structure from transport representation. While transporting the data, Thrift enforces a certain messaging structure. That is, it does not matter what method the data encoding is in, as long as the data supports a fixed set of operations that allows it to be read and written by generated code. The Thrift Protocol interface is very straightforward, it supports two things: bidirectional sequenced messaging, and encoding of base types, containers, and structs.

Thrift supports both text and binary protocols. The binary protocols almost always outperforms text protocols, but sometimes text protocols may prove to be useful in cases of debugging. The Protocols available for the majority of the Thrift-supported languages are:

1. **TBinaryProtocol**: A straightforward binary format encoding takes numeric values as binary, rather than converting to text
2. **TCompactProtocol**: Very efficient and dense encoding of data. This protocol writes numeric tags for each piece of data. The recipient is expected to properly match these tags with the data
3. **TDenseProtocol**: It's similar to TCompactProtocol but strips off the meta information from what is transmitted and adds it back at the receiver side
4. **TJSONProtocol**: Uses JSON for data encoding
5. **TSimpleJSONProtocol**: A write-only protocol using JSON. Suitable for parsing by scripting languages.
6. **TDebugProtocol**: Sends data in the form of human-readable text format. It can be well used in debugging applications involving Thrift.

Like before, out of scope to list all protocols.

Processor Layer

A processor encapsulates the ability to read data from input streams and write to output streams. The processor layer is the simplest layer. The input and output streams are represented by protocol objects. Service-specific processor implementations are generated by the Thrift compiler and these generated codes make the Process Layer of the architecture stack. The processor essentially reads data from the wire (using the input protocol), delegates processing to the handler (implemented by the user), and writes the response over the wire (using the output protocol).

Server Layer

A server pulls together all the various functionalities to complete the Thrift server layer. First, it creates a transport, then specifies input/output protocols for the transport. It then creates a processor based on the I/O protocols and waits for incoming connections. When a connection is made, it hands them off to the processor to handle the processing. Thrift provides a number of servers:

1. **TSimpleServer**: A single-threaded server using standard blocking I/O socket. Mainly used for testing purposes
2. **TThreadPoolServer**: A multi-threaded server with N worker threads using standard blocking I/O. It generally creates five minimum threads in the pool if not specified otherwise
3. **TNonBlockingServer**: A multi-threaded server using non-blocking I/O
4. **THttpServer**: A HTTP server (for JS clients)
5. **TForkingServer**: Forks a process for each request to server

Out of scope.

In general, focus a little less on enumerating different types of objects and a little more on how the different layers work together.

ADVANTAGES OF THRIFT

A few reasons where Thrift is robust and efficient compared to other RPC frameworks are:

- Thrift leverages the cross-language serialization with lower overhead than alternatives such as SOAP due to use of binary format
- Thrift generates the client and server code completely [8], including the data structures passed by the user, leaving the user with the only task of writing the handlers and invoking the client. Everything including the parameters and returns are automatically validated and parsed
- Thrift is more compact than HTTP and can easily be extended to support things like encryption, compression, non blocking IO, etc

- Thrift supports persistent connections and avoids the continuous TCP and HTTP handshakes that HTTP incurs

- Because Protocol Buffers [9] are implemented in a variety of languages, they make interoperability between polyglot applications simpler

LIMITATIONS OF THRIFT

A few reasons where Thrift is limited compared to other RPC frameworks are:

- Thrift [10] is limited to only one service per server

- There can be no cyclic structs. Structs can only contain structs that have been declared before it. A struct also cannot contain itself

- Important OOP concepts like inheritance and polymorphism are not supported

- Null cannot be returned by a server. Instead a wrapper struct or value is expected

- No out-of-the-box authentication service available between server and client

- No Bi-Directional messaging is available

Relying on bullet points so heavily is appropriate for a presentation but not a paper. Please rewrite the Advantages and Limitations sections (possibly merging them) in paragraph form and elaborate a little more on the items with examples.

CONCLUSION

This paper has given an overview of Thrift [10] and its architecture and the different ways you can use it.

This sentence is unnecessary. In the concusion, focus on Thrift, not on what the paper did.

Apache Thrift can fulfill any needs

"any needs" is very unspecific

with the flexibility it provides in choosing the different layers of the architecture separately. It outmatches any other similar technology available

You haven't shown strong evidence for this. If you want to make this point, you need to expand the paper and report on comparisons and benchmarks to other technologies beyond the blog post you cited earlier.

by providing more language support and flexibility. Thrift has enabled Facebook to build scalable back-end services efficiently. It has been employed in a wide variety of applications at Facebook, including search, logging, mobile, ads, and the developer platform. Application developers can focus on application code without worrying about the sockets layer.

REFERENCES

[1] Wikipedia, "Apache thrift," Web Page, March 2017, accessed: 2017-03-21. [Online]. Available: https://en.wikipedia.org/wiki/Apache_Thrift

[2] M. Slee, A. Agarwal, and M. Kwiatkowski, "Thrift: Scalable cross-language services implementation," 2013. [Online]. Available: https://thrift.apache.org/static/files/thrift-20070401.pdf

[3] Apache, "Thrift interface description language," Web Page, 2016. [Online]. Available: https://thrift.apache.org/docs/idl

[4] K. Sandoval, "Microservice showdown – rest vs soap vs apache thrift," May 2015, accessed: 2015-05-19. [Online]. Available: http://nordicapis.com/microservice-showdown-rest-vs-soap-vs-apache-thrift-and-why-it-matters/

[5] P. Kloe, "Benchmarks serializers," Web Page, July 2016, accessed: 2016-07-09. [Online]. Available: https://github.com/eishay/jvm-serializers/wiki

[6] A. Prunicki, "Apache thrift," Web Page, June 2009, accessed: 2009-06-01. [Online]. Available: https://objectcomputing.com/resources/publications/sett/june-2009-apache-thrift/

[7] C. Sun, "Understanding how thrift rpc works," Web Page, March 2015, accessed: 2015-09-22. [Online]. Available: http://sunchao.github.io/posts/2015-09-22-understanding-how-thrift-works.html

[8] S. Dimopoulos, "Generating code with thrift," Web Page, 2013, accessed: 2013-01-01. [Online]. Available: http://thrift-tutorial.readthedocs.io/en/latest/usage-example.html

[9] Google, "Protocol buffers - google's data interchange format," Web Page, 2015, accessed: 2017-03-01. [Online]. Available: https://developers.google.com/protocol-buffers/

[10] C. Maheshwari, "Apache thrift: A much needed tutorial," Web Page, August 2013, accessed: 2013-08-01. [Online]. Available: digital-madness.in/blog/wp-content/uploads/2012/11/BSD_08_2013.8-18.pdf