gRPC vs REST: Milliseconds matter and why?

Abstract:

From a layman's perspective web services are similar to coffee machines. The customer simply inputs his preferences and the coffee machine does the rest. Web services are similar in a way that the programmer simply defines a use case. That is a task that needs to be performed. The web service handles the actual execution of the task. A web service is a service that can be called by an application. These web services are separate programs that are independent from other applications and can be run on different machines.

A web service exposes an API (Application Programming Interface) that enables users to communicate over the internet, most often over HTTP (Hyper Text Transfer Protocol). HTTP is a web-client and web-server communication protocol. Other apps can access the functionalities of a web service using a port [5].

Since web services can be run on different machines, they require some mechanism to communicate with the application. This communication is usually handled by a Service API. The Service API can be implemented in many ways. The most popular ones are REST (Representational State Transfer) and gRPC (Google Remote Procedure Calls) [5].

This paper aims to examine how gRPC performs as opposed to its more popular counterpart REST. A series of tests consisting of varying payloads and operations was carried out to compare the performance of REST and gRPC. The results were compared, and it was found that gRPC tends to perform faster in most scenarios, save one (gRPC streaming).

Introduction

REST is a popular resource-based technique using GET, POST, PUT DELETE methods. Resources are data entities on which you perform actions, like the GET, POST, PUT and DELETE methods. REST's communication often includes sending a JSON and can run over HTTP/1.1 or HTTP/2 [5].

gRPC is an open-source RPC framework that is created and used by Google. It is built upon HTTP/2.0 which makes bi-directional communication possible. gRPC communicates using binary data using protocol buffers by default for serializing structured data. gRPC servers allow cross-language unary calls or stream calls [5].

There are a number of differences between gRPC and REST which make it suitable in different scenarios. REST is more widely used because of two primary reasons.

- REST APIs are desirable when a system needs high speed iteration and standardization of HTTP protocol (runs on HTTP 1.1).
- When system is highly reliant on third party tools. Since REST is more widely used it is likely that application integration will be easier as there is widespread support for REST.

Since gRPC is not as widely used, most third-party tools lack in built features for gRPC compatibility. Therefore, gRPC is mainly used to build internal systems. i.e. systems that are closed to the external world. The following scenarios would be ideal to use gRPC.

- Microservices connections: Because of its low latency and high throughput connectivity, gRPC is ideal for linking systems made up of lightweight microservices where message transmission efficiency is critical.
- Multi-language systems: gRPC is ideal for handling connections in a multilingual context since it supports native code generation for a broad range of programming languages.

- Real time streaming: When real-time communication is required, gRPC's ability to manage bidirectional streaming allows your system to send and receive messages without having to wait for Unary client-response communication.
- Low bandwidth networks: gRPC's use of serialized protobuf messages provides lightweight communications, increased efficiency, and speed for low-power, low bandwidth networks (especially when compared to JSON). The Internet of Things (IoT) is an example of a network that might benefit from gRPC API [3].

Literature Review

I found various articles on the internet arguing the benefits and drawbacks of REST over gRPC. On paper gRPC out performs REST in most scenarios. Various companies like Ably and Yonego have successfully transitioned from REST to gRPC. However, on the other hand C. L. Chamas, D. Cordeiro and M. M. Eler have analyzed the effects of the type of communication protocol on energy consumption [1]. Although this does not relate directly to the situation under test, it provides insight into why developers and companies are hesitant to move away from REST. gRPC has not yet been widely adopted by major companies because of which the research carried out in the domain is very application specific. Most articles/papers present niche areas where gRPC might prove to be a good option. The aim of this experiment and analysis aims to verify the hypothesis that gRPC is indeed faster than REST.

Goals of the Paper

To compare the performance of gRPC and REST testing in on different kinds of payloads. Three specific scenarios are considered to be tested. Receiving small payload, receiving large payload, and sending large payload. In order to test the streaming capability of gRPC another method GetLargePayload is implemented. The performance is compared, to verify the claim that gRPC is indeed faster than REST.

Technical Implementation

System used for testing: Dell Latitude 5410. Intel i5 VPro (10th Gen) Quad Core CPU (8 Logical cores) [8 GB RAM.]

Constraints:

There were a few assumptions and decisions that were implemented when bench-marking the REST vs. gRPC calls [6].

- Test pure communication throughput, not logical operations were carried out.
- No connection to database, static hard coded values are used. Thus, eliminating caching issues and random execution time differences
- Turn off logging so that it doesn't interfere with the test runs. If not, it might have resulted in very erroneous values.

Components:

- **RestAPI**: Contains the WebAPI, which exposes three methods for receiving a small payload, receiving a big payload, and transmitting a large payload in three different contexts [6].
- **GrpcAPI:** This folder contains the Service Implementation as well as some initialization code for the gRPC Server. I wanted to see how gRPC streaming stacked up against REST, so I added an extra method called "GetLargePayload" that iterates and streams one data point at a time [6].

- **RESTvsGRPC**: Includes the Benchmark harness, which runs all of these methods in two batches to eliminate measurement mistakes caused by short execution durations. As a result, the benchmark execution times shown below have been adjusted accordingly [6].

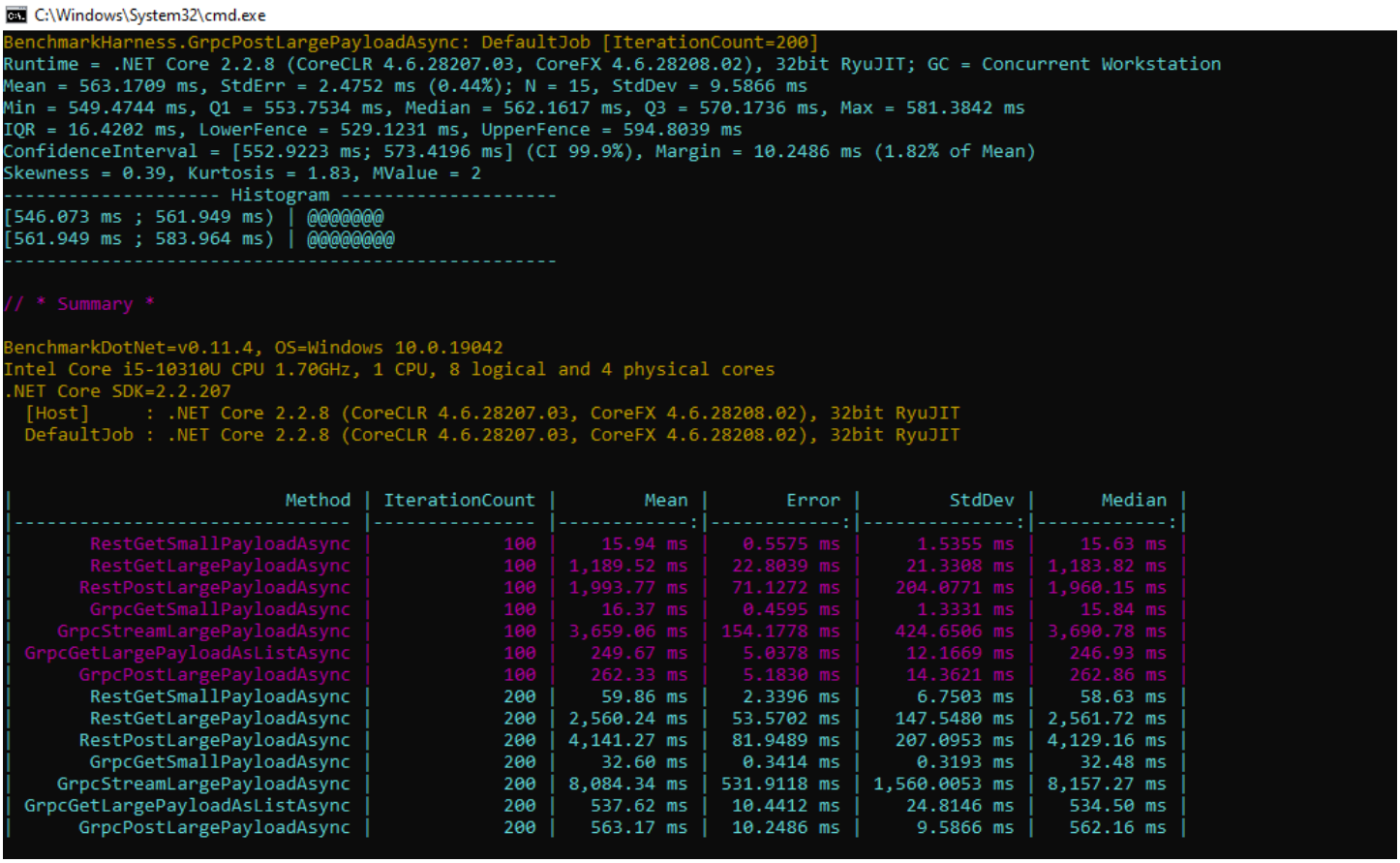

Figure 1. Performance Comparison (gRPC vsREST)

Anomaly in results: The hypothesis that gRPC is roughly 7-10 times faster than REST has been verified in most cases. gRPC streaming performs worse than REST. I decided to delve a little deeper to understand why does gRPC streaming perform slower.

There are two sorts of request models in gRPC:
Unary — simple request-responses that are mapped onto HTTP2 request-responses.
Streamed — a long-lived HTTP2 stream, which can be unidirectional or bidirectional, is used to exchange many requests and answers.

HTTP2 multiplexes streams across a long-lived TCP connection, avoiding TCP connection costs for new requests. Using HTTP2 framing, a single TCP packet can carry many gRPC messages. For long-lived connections, streamed queries should provide the optimum performance per message. Unary requests necessitate the construction of a new HTTP2 stream and the sending of additional header frames across the network [3].

Analysis of the discrepancy:

Various articles indicated changing the size of the response message in the protobuf message. Complex messages might affect the speed of gRPC streaming.

```
BenchmarkDotNet=v0.11.4, OS=Windows 10.0.19042
Intel Core i5-10310U CPU 1.70GHz, 1 CPU, 8 logical and 4 physical cores
.NET Core SDK=2.2.207
  [Host]     : .NET Core 2.2.8 (CoreCLR 4.6.28207.03, CoreFX 4.6.28208.02), 32bit RyuJIT
  DefaultJob : .NET Core 2.2.8 (CoreCLR 4.6.28207.03, CoreFX 4.6.28208.02), 32bit RyuJIT


|                          Method | IterationCount |       Mean |      Error |     StdDev |     Median |
|-------------------------------- |--------------- |-----------:|-----------:|-----------:|-----------:|
|       RestGetSmallPayloadAsync |            100 |    23.99 ms |   2.4501 ms |   7.1081 ms |    22.08 ms |
|       RestGetLargePayloadAsync |            100 | 1,766.55 ms |  94.7045 ms | 274.7549 ms | 1,728.65 ms |
|      RestPostLargePayloadAsync |            100 | 2,691.21 ms |  77.3372 ms | 220.6473 ms | 2,631.28 ms |
|       GrpcGetSmallPayloadAsync |            100 |    19.47 ms |   0.3840 ms |   0.8184 ms |    19.39 ms |
|    GrpcStreamLargePayloadAsync |            100 | 3,754.88 ms |  72.1742 ms | 193.8912 ms | 3,721.00 ms |
| GrpcGetLargePayloadAsListAsync |            100 |   156.73 ms |   3.2352 ms |   9.1776 ms |   157.20 ms |
|      GrpcPostLargePayloadAsync |            100 |   158.30 ms |   3.1560 ms |   8.8498 ms |   158.54 ms |
|       RestGetSmallPayloadAsync |            200 |    40.32 ms |   1.7277 ms |   4.9572 ms |    38.54 ms |
|       RestGetLargePayloadAsync |            200 | 2,706.00 ms |  54.0257 ms | 137.5127 ms | 2,706.48 ms |
|      RestPostLargePayloadAsync |            200 | 4,041.80 ms | 184.8423 ms | 545.0118 ms | 4,028.46 ms |
|       GrpcGetSmallPayloadAsync |            200 |    28.55 ms |   0.3212 ms |   0.2847 ms |    28.59 ms |
|    GrpcStreamLargePayloadAsync |            200 | 5,185.97 ms | 101.3369 ms | 182.7314 ms | 5,172.12 ms |
| GrpcGetLargePayloadAsListAsync |            200 |   201.47 ms |   3.9612 ms |   3.7053 ms |   201.47 ms |
|      GrpcPostLargePayloadAsync |            200 |   206.02 ms |   4.0764 ms |   7.8539 ms |   206.20 ms |
```

Figure 2. Performance Comparison (gRPC vs REST) [Smaller message format in protobuf]

As it is evident from the result GrpcStreamLargePayload performance improved (8084ms to 5185ms) This leads to the question, “What exactly is the reason for the discrepancy in performance?”. The answer to the question lies in the analysis of TCP packets.

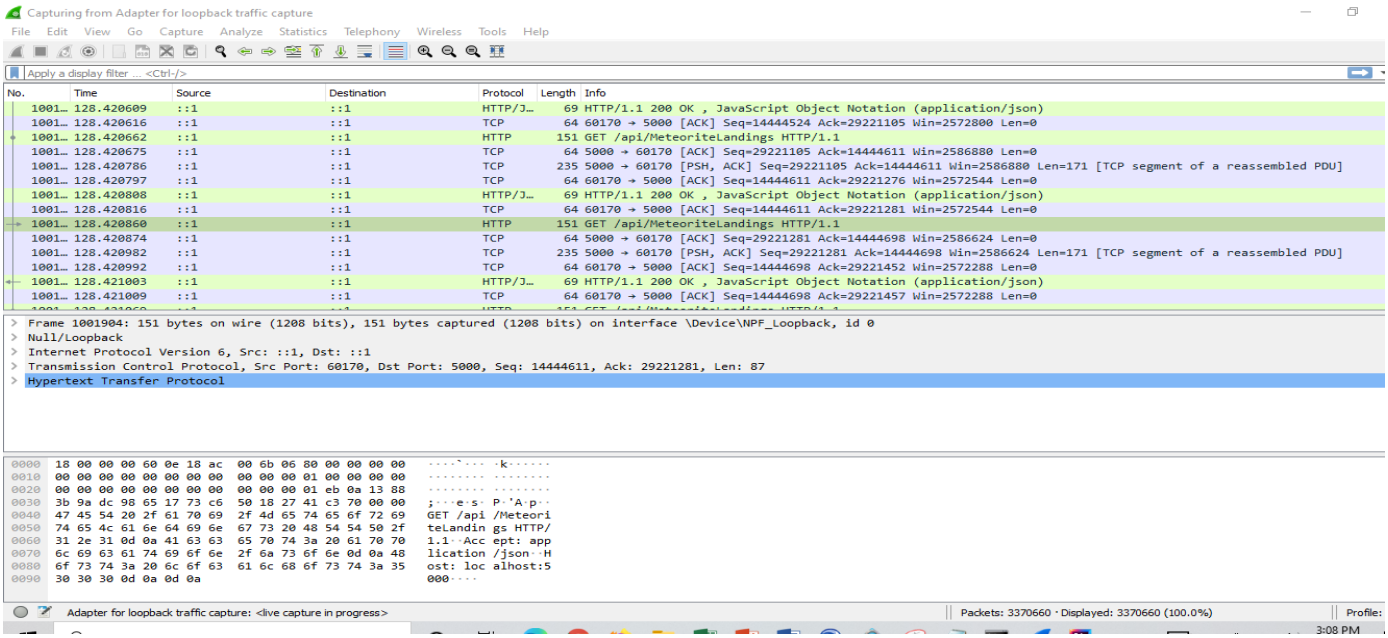

Figure 3. Wireshark to try to analyze TCP packets

There are 2 major factors which causes unpredictability in gRPC performance. Although the data processed in each frame is somewhat from (64 in our case), the amount of work performed in setting up the initial connection is more. Complex messages require more data to be transmitted. How the data is packed in the data frames and transmitted is not always consistent. This inconsistency causes more delay. This can sometimes lead to less data being transmitted. Adding randomized delays at time of connection establishment might lead to more uniform outcomes.

Protobuf files help make gRPC easier, however the simplicity has a disadvantage in that it hides what's going on in the TCP connection. At high message rates, this becomes an issue since you need to see what's going on, but

gRPC doesn't make it simple. As a result, reasoning about and forecasting network performance is difficult, generating at the very least a headache and perhaps severe downstream repercussions [3].

Conclusion

When receiving data, gRPC is around 7 times quicker than REST, and when sending data, gRPC is roughly 10 times faster than REST. This is mostly due to gRPC's usage of HTTP/2 and the tight packing of Protocol Buffers. Further research is required to help understand the unusual behavior of gRPC streaming. Still, the benefits of gRPC outweigh its disadvantages, if so "Why hasn't it been widely adopted?". The answer, REST is supported universally and virtually every microservices-based infrastructure depends entirely on REST APIs as the glue that holds them together.

It took me a lot more time to implement and understand gRPC as compared to the traditional WebAPI approach. This is mainly due to REST becoming mainstream a long time back, and most major frameworks having built in support to quickly build up such services. At present, gRPC although more beneficial should be avoided in large scale applications unless the use case of the application is very specific.

References:

[1]	C. L. Chamas, D. Cordeiro and M. M. Eler, "Comparing REST, SOAP, Socket and gRPC in computation offloading of mobile applications: An energy cost analysis," *2017 IEEE 9th Latin-American Conference on Communications (LATINCOM)*, 2017, pp. 1-6, doi: 10.1109/LATINCOM.2017.8240185.
[2]	Jeremy H, *gRPC vs. REST: How Does gRPC Compare with Traditional REST APIs?* Dream Factory, Sep. 2020. Accessed on: Nov 10, 2021. [Online]. Available: https://blog.dreamfactory.com/grpc-vs-rest-how-does-grpc-compare-with-traditional-rest-apis/
[3]	P. Cruickshank, *The Mysterious Gotcha of gRPC Stream Performance*, Ably, Oct. 2021. Accessed on: Nov 09, 2021. [Online]. Available: https://ably.com/blog/grpc-stream-performance
[4]	M. Berga, A. Santos, *gRPC Vs Rest: Comparing API Architectural Styles*, Imaginary Cloud, Jun. 2021. Accessed on: Nov 09, 2021. [Online]. Available: https://www.imaginarycloud.com/blog/grpc-vs-rest/
[5]	W. Toeman, *Why milliseconds matter*, Yonego, Oct. 2021. Accessed on: Nov 09, 2021. [Online]. Available: https://www.yonego.com/nl/blogs/why-milliseconds-matter/
[6]	R. Fernando, *Evaluating Performance of REST vs. gRPC*, Medium, Apr. 2019. Accessed on: Nov 09, 2021. [Online]. Available: https://medium.com/@EmperorRXF/evaluating-performance-of-rest-vs-grpc-1b8bdf0b22da