Remote Procedure Call

Stephan Djurhuus, Pernille Lørup 11/12/2020

Abstract

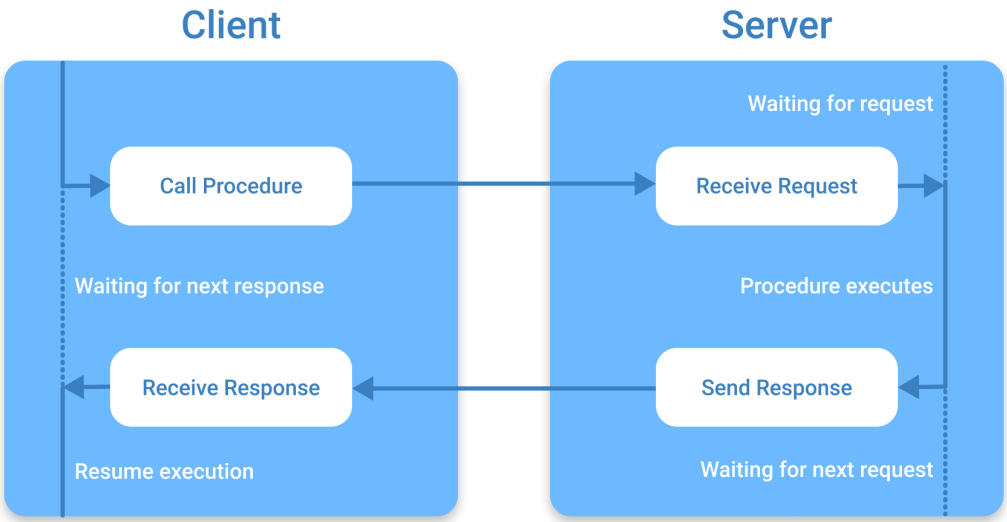
This article provides a simple introduction to Remote Procedure Call, also referred to as RPC. The topics of some of the advantages and disadvantages of RPC is presented alongside with some personal experiences dealing with the protocol and its alternatives. An analysis based upon the response times of RPC, REST and GraphQL is performed to display the difference between the three.

Introduction

Remote Procedure Call, also known as RPC, is a protocol which during runtime provides a connection between two or more systems. This protocol is a client-server model in which the client has knowledge of the server credentials. A connection between the two systems are obtained through a shared interface and the use of an RPC library. RPC offers the possibility to request a remote or local service while hiding the internal functionality from the user.

When making a remote procedure call it starts at the client environment where a request containing parameters if necessary is transferred through a network layer to the server. When the server receives the request, a method is executed in the remote environment that returns a response which is then transferred back to the client; this action is the procedure.

Figure 1: Procedure Flow



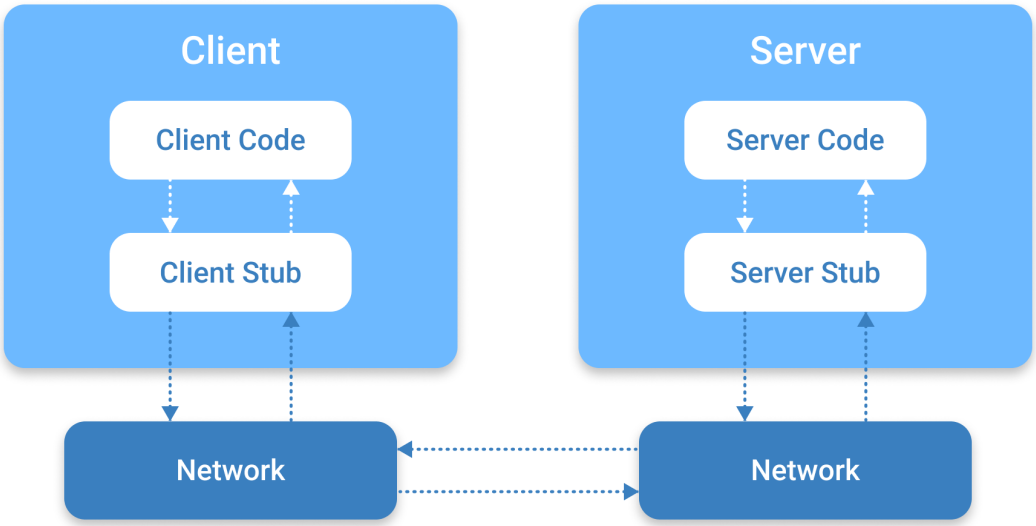
Functionalities

A contract between the two systems is established through the shared interface to give mutual constraints upon the requests and responses which is handled by a stub.

A stub is the result when the shared interface is provided to the RPC library. After providing the interface, the stub contains the implemented methods which can be called in the application. Calling the stub methods transforms the parameters to a suitable format for transportation, which is then received at the other system as a request. This formatting is also referred to as marshaling. When the request is received the system unmarshals it to make it readable. Marshaling and unmarshaling happens between both systems every time data is transferred.

Remote procedure call can be synchronous or asynchronous depending on the implementation. In most cases the procedures are synchronous which means the client system is suspended when making a request until a response is received.

Figure 2: RPC Model



Project References & Our Implementations

Our first assignment related to RPC was in the System Integration course where the main focus of the assignment was to get acquainted with and illustrate the use of RPC. We solved the assignment by making use of an RPC library for Node.js together with a strongly typed programming language, which we in our case decided would be TypeScript.

After getting acquainted with RPC we chose to use it as the foundation in our project from the Large System Development course. This new project gave us a broader understanding of the protocol and how to optimize the implementation. One of the key features when using RPC is the possibility to create two concurrent systems that work independently until production. Because the client and server share an interface we can implement a fake on the client-side to be

used during development. The fake substitutes the server with hardcoded implementations to test the client system's functionality.

Because the criteria of the Large System Development assignment was to build two systems from a contract, we decided to use the RPC protocol that is build around the common interface which in our case was the contract.

Advantages/ Disadvantages

Advantages

- **The possibility to develop to or more concurrent systems**
  By dividing the development into smaller systems, the possibility of distributing workload to multiple internal or external developers is obtained, which can result in an increased efficiency.
- **The contract creates full transparency**
  The contract provides a specified description of all methods and necessary parameters in the common interface. This gives all developers an understanding of their constraints and limitations regarding the connection between the systems. These constraints give the possibility to obtain a better collaboration from external developers.
- **Possibility for distributed subsystems**
  When distributing the development into smaller subsystems, each subsystem has the ability to use the desired amount of resources available to it. This can optimize CPU and memory consumption by expanding it over multiple machines an thereby increase the performance significantly.
- **Suitable for developing large systems**
  Because of the possibility of distributing multiple system features across different teams and servers, the use of RPC is convenient for developing large systems.
- **Underlying procedures are hidden from client**
  The client is only provided with methods from the shared interface which gives the developer knowledge of what response the client receives. Because of this, the developer doesn't have to be concerned about how the server's implementation handled the request, which will provide more focus and avoid a bloated system.
- **Subsystems are more maintainable**
  Smaller systems are easier to develop and maintain due to the better overview of the system, instead of navigating through a large system and waste time understanding the surrounded functionality. A subsystem limits the amount of functionalities and thereby increases the level of understanding so more time can be spend on developing.

Disadvantages:

- **Response time is dependant on network bandwidth**
  If the server and client are located on different machines, an outgoing network connection needs to be established. The user's response time will be increased in case of a low network bandwidth on one of the systems.
- **Distributed RPC systems are vulnerable to failure**
  These types of distributed systems are more vulnerable to failure in comparison to other systems where all implementation is gathered in one place. This is because the multiple distributed systems rely on each other in production and therefore if one of the systems where to disconnect, the others would be affected as well.
- **Not a flexible connection**
  The interaction between the server and client is limited to the contract methods. Only methods with the required parameters can be used and nothing else.
- **It isn't easy to update the contract.**
  The contract defines how the two systems can interact with each other. If the contract needs to be modified all implementations need to be implemented accordingly. Depending on the degree of the modification a significant amount of code can end up being rewritten.
- **Integration testing of the system is limited during development**
  The uncertainty of proper integration between systems throughout the development is difficult to test, because different systems can be developed in separation.

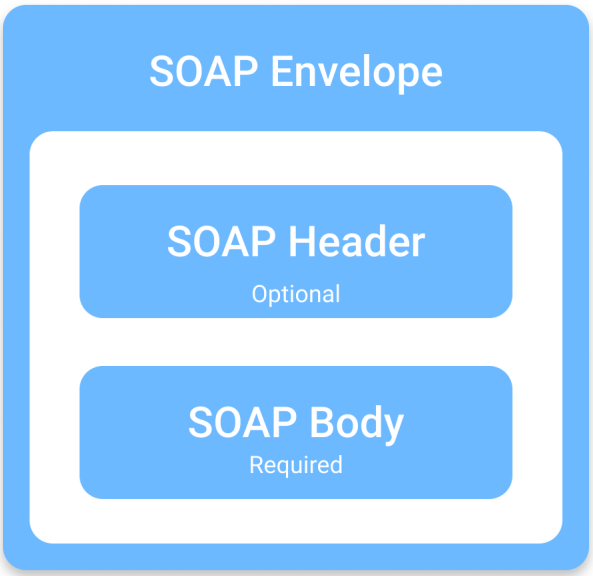
Alternatives

Simple Object Access Protocol

Simple Object Access Protocol also known as SOAP is a messaging protocol for exchanging data via XML. The message being exchanged is referred to as an envelope, which is mandatory because it indicates the beginning and the end of a message. The envelope consist of an optional header and a obligatory body.

It is based upon schemas that define the constraints and structure of the request. When a request is received on the server, the schema validates if the format is correct. If the request is accepted, it will be processed. However if the envelope body doesn't match the schemas' requirements, the request will be rejected.

Figure 3: SOAP Envelope Structure



Representational State Transfer

Representational State Transfer or REST is a broadly supported architectural style for client-server communication. REST sends information by using json or xml data structure via multiple predefined endpoints instead of sending information through methods. This gives the client a more flexible interaction to the server but is still constraint to the HTTP methods and headers together with the required body content and structure. REST doesn't use any interface which means the client needs to read the API documentation before using it.

Graph Query Language

Graph Query Language also called GraphQL is a newer query language standard for APIs. This implementation uses the client-server model where the server exposes a single endpoint to the client. Strongly typed schemas are used to define the possible structures of a query that a client can request to the server. These schemas, also referred to as contracts, gives the possibility to create advanced queries, consisting of the desired response data structure. Queries like these will be executed as a request to the server which sends a response only containing the requested data. One of the advantages of using these queries is to prevent under- and over-fetching which results in faster response time and reduced data consumption.

Analysis of Response Times

We created a simple Hello World example for RPC, REST and GraphQL to compare the different response times on a local machine. The results are based upon 500 iterations of 'Hello

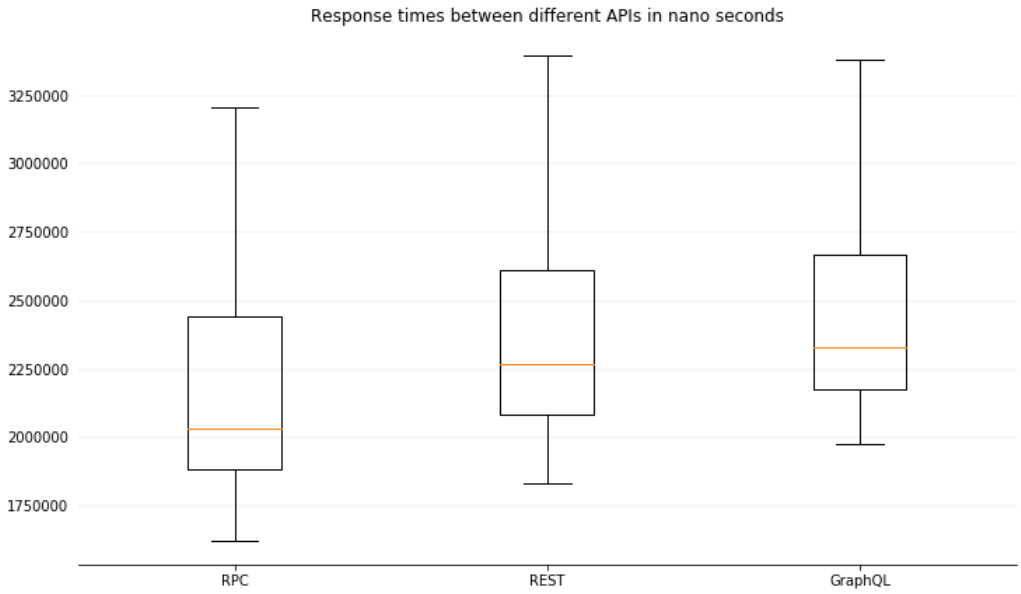
World' requests that responds with a simple string. The analysis resulted in RPC being the best performer, which is shown in the table below.

Table 1: Mean & Standard Deviation

	Mean	**Standard Deviation**
RPC	2.334ms	1.190ms
REST	2.529ms	1.576ms
GraphQL	2.636ms	2.520ms

We have plotted the results in a boxplot below, to show the descriptive statistics calculated throught the observed observations.

Figure 4: Response Times



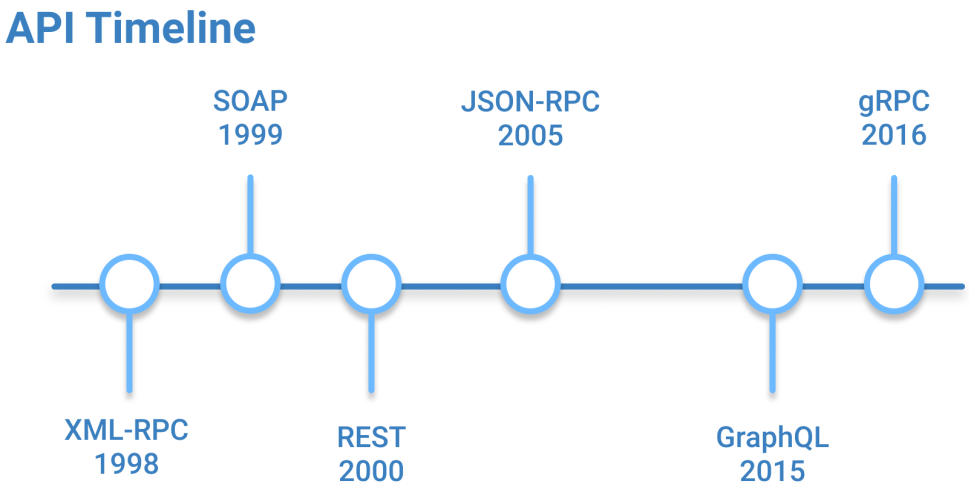
Specifications

The code was executed on a MacBook Pro (Retina, 13-inch, Early 2015) with a 2,9 GHz Dual-Core Intel Core i5 Processor. The MacBook has a memory of 8 GB 1867 MHz DDR3.
The code was executed with Java version 13.0.2.

Conclusion

Remote Procedure Call is a powerful protocol with many advantages. Working with it in multiple projects along with research and our previous knowledge of its alternatives in SOAP, REST and GraphQL, we have come to learn that RPC is a protocol suitable for developing large systems. If the contract is correctly specified between the systems it creates a strong base for development.

Figure 5: API Timeline



A perfect tool for everything doesn't exist. The compromise you give by developing a distributed RPC system is your response time which can be reduced while your operation performance can be increased. A system with heavy operations would prefer to use a distributed system like RPC, while a fast responding system wouldn't.