A Comprehensive analysis on IPC (Inter process communication) mechanisms and improvements in SAP IoT.

By

ABIN BABU

2020SP93094

ACKNOWLEDGEMENT

I would like to thank my supervisor **Joshy Mon A M** for his guidance and wholehearted support. His valuable comments have been immensely helpful in enhancing the quality of work.

Special thanks to my examiner **Sri Prashant Bhandare** for his support and guidance throughout my project. Their selfless nature to help has resulted in successful completion of this project on time and their valuable remarks have led this project to come out this great.

My sincere regards and thanks to BITS examiner **Dr. Lohith J J** for his valuable feedback along the way.

I am also thankful to **SAP Labs India** for giving me this opportunity and making available all the resources required for this dissertation. Also, my gratitude to **BITS, Pilani** and the Training department for all their efforts in organizing this course.

ABIN BABU
(202SP93094)

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
SECOND SEMESTER 2020-21

SESAP ZG629T DISSERTATION

Dissertation Title : A Comprehensive analysis on IPC (Inter process communication) mechanisms and improvements in SAP IoT.

Name of Supervisor : Joshy Mon A M

Name of Student : Abin Babu

ID No. of Student : 2020SP93094

ABSTRACT

Distributed computing is significant for any highly scalable service. Designing these services as different small microservices is the go-to architectural choice in the current scenario. It breaks down enormous applications into small, decoupled, and disseminated bundles.

Since microservices-based applications are widely used, one of the key challenges while planning an application is the decision of the system by which services speak with one another i.e., Inter-process communication of these microservices. There are a few standard methodologies for implementing IPC in microservices landscapes and each of them comes with various benefits and compromises.
In this project, we are analyzing different Synchronous and Asynchronous IPC strategies. The Synchronous kind comprises REST API and Google gRPC, while the Asynchronous kind is utilizing a message Queue known as RabbitMQ. Further, the Performance analysis of different strategies is measured and compared.

Keywords: IPC (Interprocess communication), REST API, gRPC,RabbitMQ

LIST OF TABLES

LIST OF FIGURES

TABLE OF CONTENTS

INTRODUCTION & BACKGROUND

SAP Leonardo IoT is a collection of tools and applications built on the SAP Cloud Platform that can be used to create an administrative environment for managing and monitoring sensor data generated by technical objects that are part of the Internet of Things (IoT). The raw data obtained by the sensors are put into the business object context and then we use query models, rules, events, and actions to leverage the real-time analytics or transactions for business applications.

Throughout recent years, microservices design has acquired tremendous consideration and gained enormous adoption from the industry. While it is challenging to define microservices, a few attributes can be seen in common such as,

- Decoupled and easily deployable independent modules.
- Communication between these light components to achieve a higher business functionality
- Each module might be having different tech stack and databases and communicate over a standard protocol.

IoT uses a plethora of micro-services with over 50 micro-services in the current portfolio. Performance is pivotal in IoT Scenario hence a small improvement in terms of the latency of these inter-process communications can amplify the overall performance. In this research, we focus to find out different approaches & improvements that we can introduce in terms of faster service to service calls. This research can serve as a base/ reference material for future research and improvements

.

PROBLEM STATEMENT

What impact does the IPC choice for any microservice-based system have on non-functional needs such as performance, availability, scalability, and complexity, and what are the benefits and downsides of such a strategy?

This research aims to conduct a detailed analysis and comparison of various IPC communication protocols to determine the benefits and trade-offs. This research can be used as a foundation for making architectural decisions when developing SAP IoT apps in the future.

OBJECTIVE

The main objectives of this project are:

- Analyze the different IPC strategies
- Experimenting and understanding different available solutions
- Analyzing how these can be integrated into our apps
- Understanding the benefits and tradeoffs with each approach
- Obtaining quantitative data with load testing

UNIQUENESS OF THE PROJECT

There are currently no well-defined explanations or methods for choosing the ideal IPC strategy while developing or designing microservices-based applications. As a result, there is a great deal of confusion regarding when to use which approach and what the trade-offs are when adopting a strategy. Over the course of time, reaching a decision becomes even more difficult. There are no right or wrong solutions; rather, they are more or less appropriate depending on the circumstance. The objectives of this work are to learn more about the different IPC strategies and evaluate them in order to arrive at a quick decision while building and selecting the best communication method. This project can serve as the foundation for any future SAP IoT architecture decisions.

BENEFIT TO THE ORGANIZATION

The significant proportion of SAP products are already cloud-native or on their way to becoming so, and the business is putting a lot of effort into developing a strong cloud portfolio. Due to the intricacy of the services being delivered, microservices are the go-to architectural consideration, as the solutions deal with vast amounts of data and extensive functionality. Given the frequency with which these things are consumed by customers, any performance improvements will be significant. This project can be used as a model for any architectural design, and it will help to determine what information should be shared between microservices based on specific use cases.

SCOPE OF WORK

The scope of this dissertation project is as follows:

- Analyzing and understanding different IPC communication approaches
- Synchronous communication with gRPC and REST
- Asynchronous communication with RabbitMQ
- Tradeoffs and Benefits of different approaches
- Performance analysis and benchmarking

IMPLEMENTAION DETAILS

One of the most crucial and essential considerations to be made when implementing a system based on a microservices architecture is how microservices will communicate with one another. The main challenge is determining the best option for inter-process communication. In contrast to monolithic architecture, where modules can access and invoke each other at the language level, microservices structure the system as a set of independent distributed services with the potential for each service to run on a different host than the other. This means that IPC plays a much more important role in IPC architecture than it does in monolithic architecture. The interaction between microservices whether synchronous or asynchronous is a crucial factor to consider when choosing an IPC mechanism.

Different types of microservice interaction:

- Asynchronous
- Synchronous

Synchronous Communication

This form of communication is often regarded as a request/response interaction style. In this mode, one microservice makes a request to another service, and wait for that services to process the result and send a response back. In this mode, it is common that the requester blocks it's operation while waiting for a response from the remote server.

HTTP-based REST API and gRPC4 are the two most common type of Synchronous communication when building microservices.

- REST API

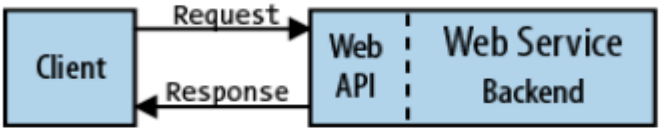

Figure 1: REST API directly listens and response to request

Regardless of the software architecture of the two systems, data communication over the REST API is one of the most popular methods. In this approach, client program uses interface with the web services provider using APIs over the HTTP protocol. Each service often has a web server running on a certain port, such 8080 or 443, and each service exposes a set of endpoints to enable interactions with other microservices and the exchange of information between them in a system that uses REST API for its IPC communication.

Solution Architecture

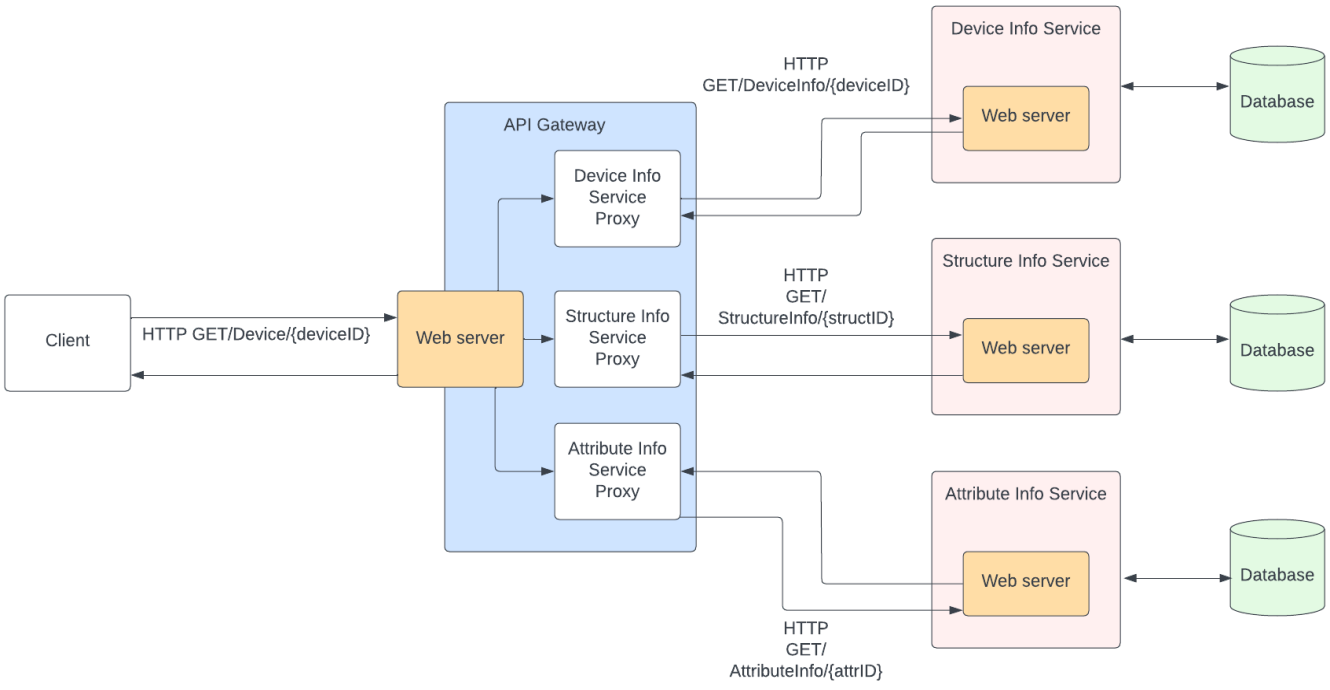

Figure 2: Communication using REST API

Demonstrates how the IPC mechanism takes place when microservices communicate with each other via REST API. The API Gateway get queried from the client app which with the deviceId. The gateway then performs a REST API call to each microservice, passing the device Id/StructureId/AttributeId as parameters waits for each service response to come back. In this model, as the communication takes place using REST API over HTTP protocol, each microservice must run a web server to be able to handle HTTP queries.

- gRPC

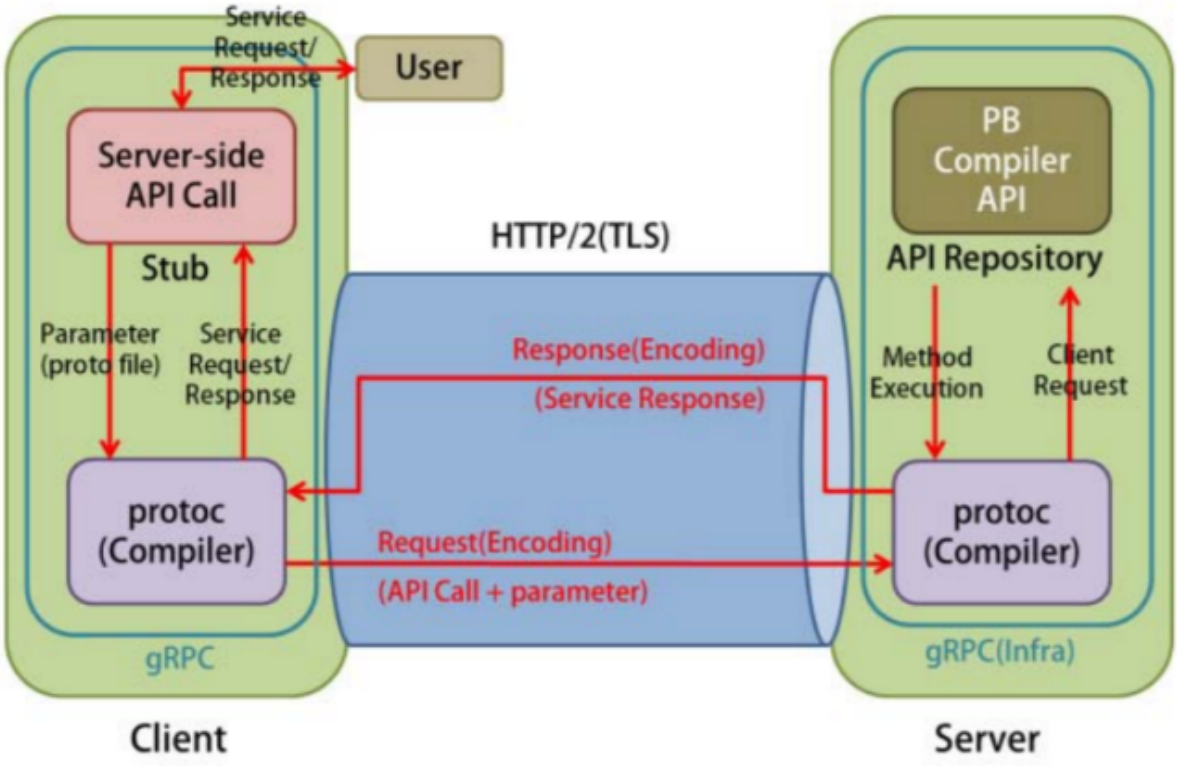

Figure 3: Operation process between client/server in gRPC

gRPC is a modern RPC based protocol developed by Google for the purpose of creating cross-language client and server. Many distributed applications employ RPC as a technique to support interprocess communication. Message exchange between two processes is made possible by the RPC protocol, which was first implemented by. It is known for its low overload, simplicity, and transparency. By default, when a client makes a request to a server, the operation is stopped and the client waits for the response. Therefore, a remote procedure call is viewed as a synchronous kind of communication. gRPC runs via HTTP/2.0 as opposed to REST API, which uses HTTP/1.1 as the default transport protocol, giving gRPC significant benefits in terms of performance and security.

Solution Architecture

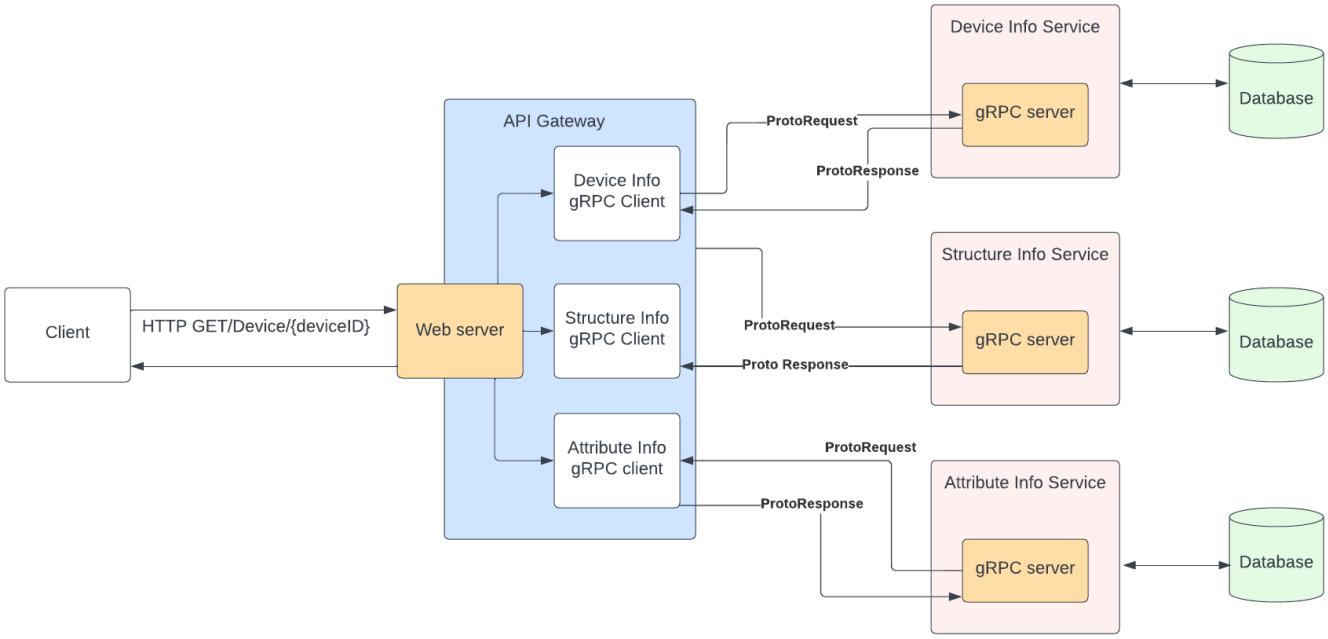

Figure 4: Communication using gRPC

In gRPC, each microservice runs a gRPC server on a specified port, and the API gateway runs as a client of each gRPC server, establishing a connection with each microservice during initial setup. The API Gateway gets the device request from the client over HTTP, then takes that request and calls all microservices by supplying the appropriate arguments and waiting for their responses, similar to the prior approach. The results sent from microservices to API Gateway are in bytes since gRPC utilises a binary-based data serialisation standard. As a result, before returning those bytes to the user, the API Gateway translates and transforms them into JSON format.

Asynchronous communication

Synchronous communication can be implemented when services exchange messages with one another using a messaging pattern. This type of IPC uses a message broker between the services to coordinate requests and responses on behalf of the microservices. One of the key distinctions between synchronous and asynchronous communication is that with asynchronous communication, the client no longer places a direct call to the server and anticipates receiving a prompt response. The request will be received from the broker and handled by one or more services before being returned to the broker with the results. In other words, the interaction between microservices in the asynchronous type of communication is controlled by intermediary service known as service broker.

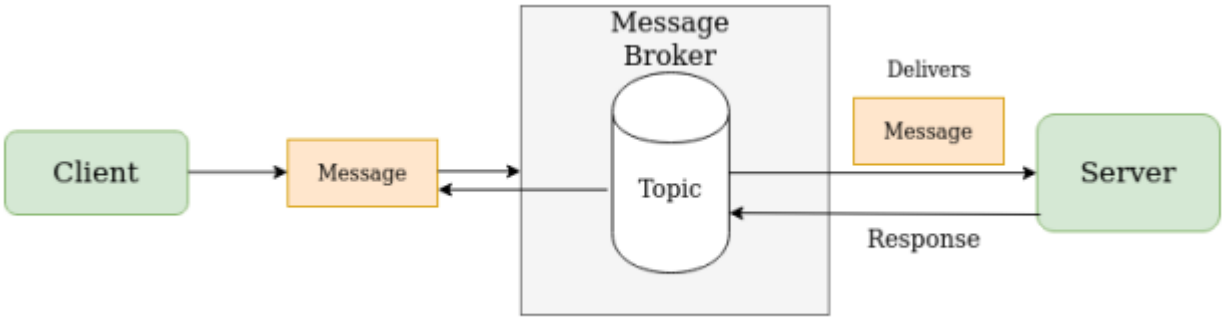

Figure 5: Asynchronous communication using message broker

- RabbitMQ

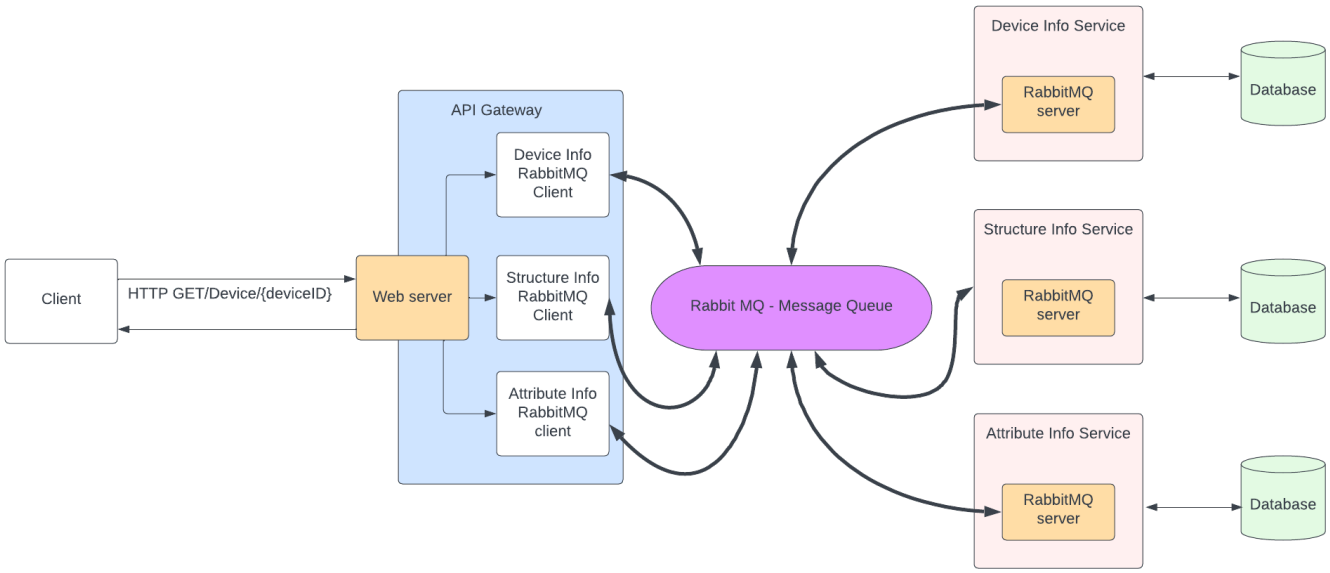

Figure 6: Communication using RabbitMQ

RabbitMQ has been chosen as the message broker for asynchronous communication using the AMQP protocol. Unlike the Synchronous technique, the API GateWay does not have direct connection with any of the microservices in this architecture; instead, they communicate through a message broker. The Gateway sends a request to the broker with the required parameters using this technique. That request would be picked up by other microservices, which would process it and return the result to the broker. The Gateway will pick up the response after all microservices responses are available.

RESULTS AND EVALUATION

Analyzing Performance

Apache JMeter has been used to create and run three test cases. Each IPC method's latency and throughput are evaluated in each test case. The two key factors for determining performance efficiency are latency and throughput, as described in the first chapter. While the number of concurrent virtual users that constantly send queries to the system and wait for responses has fluctuated in all three test tests, the length has remained consistent at 180 seconds. The first case had 50 virtual users operating concurrently, the second case had 100, and the third case had 200 virtual users operating concurrently. Understanding how each IPC method responds differently as concurrent requests and system traffic rise or decrease is the goal of making the test time a constant variable and the number of virtual users a controlled variable.

Every test case's throughput is determined by how many requests and responses the method was able to complete in the allotted 180 seconds; the more requests, the greater the throughput and the better the method. In contrast, the amount of time required to process each request is how latency is calculated. The average response time is being assessed in this experiment. By adding up the response times for all of the requests made by the technique and dividing the result by the total number of requests, the average response time is obtained. Lower Latency suggests a lower average response time, which is advantageous for the given approach.

IPC Method	Test Duration	Virtual Users	Average Response Time	Total Request/ Response
REST	180s	50	0.23s	4261
gRPC	180s	50	0.23s	4304
Rabbit MQ	180s	50	0.23s	4157

Table 1: TestCase-1 Result

In the first testcase, the data shows that gRPC outperformed REST API and RabbitMQ, processing 43 more requests than REST API and 104 more requests than RabbitMQ. This shows that synchronous forms of communication can provide higher throughput than asynchronous ones in circumstances where the load on the system is relatively low. When the number of concurrent threads in the system is modest, the first case's outcome also shows that synchronous communication may process requests a little bit quicker than asynchronous communication and therefore, has reduced latency.

IPC Method	Test Duration	Virtual Users	Average Response Time	Total Request/ Response
REST	180s	100	3.5s	4373
gRPC	180s	100	3.3s	4453
Rabbit MQ	180s	100	3.5s	4398

Table 2: TestCase-2 Result

In comparison to the first scenario, there are twice as many virtual users in the second case. The number of concurrent requests in the system grows as the number of virtual users increases, extending processing time. By handling more requests than RabbitMQ and REST API, the same data suggest that gRPC has the greatest throughput. In this test,

gRPC outperformed REST API and RabbitMQ by 200 milliseconds in terms of average response time. In this round, RabbitMQ and REST API both processed requests in the same amount of time, however RabbitMQ was able to handle an additional 25 requests than its Synchronous competitor.

IPC Method	**Test Duration**	**Virtual Users**	**Average Response Time**	**Total Request/ Response**
REST	180s	200	7.8s	4373
gRPC	180s	200	7.2s	4453
Rabbit MQ	180s	200	6.4s	4398

Table 3: TestCase-3 Result

In the third scenario, there are four times as many virtual users as there were in the first. The third testing experiment's findings suggest that when the number of concurrent requests rises, there will be a noticeable difference between synchronous and asynchronous forms of communication in terms of throughput and latency. In this test, asynchronous communication via RabbitMQ surpassed the other two ways by handling a total of 4480 requests within the allotted time, whereas gRPC was only able to handle 132 requests less than RabbitMQ and REST API handled 146 less requests than RabbitMQ.

CONCLUSION

One of the major concerns with microservices design is interprocess communication, which might influence the system due to various non-functional needs. To demonstrate how the choice of IPC might affect the system's non-functional needs, extensive testing has been conducted against each technique. The review provides compelling evidence to support the claim that asynchronous communication is preferable to synchronous communication because it delivers superior performance efficiency, availability, and scalability, even if it increases code complexity and demands more development work. Additional advice on how to choose between asynchronous and synchronous form for various circumstances have been given in the discussion part of the sixth chapter. In conclusion, because it is so important to a microservices design, the IPC method selection must be carefully considered. There are circumstances in which one kind of communication is preferable to the other. Because of this, in a perfect system, both synchronous and asynchronous types must be used in accordance with the functional and non-functional needs of the individual components.

RESOURCES NEEDED FOR THE PROJECT

Software:

1. RabbitMQ
2. Java, Spring Boot
3. SAP BTP
4. IntelliJ IDE
5. MongoDB

Hardware:

1. Laptop (16.0 GB RAM)
2. Internet Connection

People:

1. Supervisor

PROJECT PLAN AND DELIVERABLES

Serial Number of Task/Phases	**Tasks or subtasks to be done** (be precise and specific)	Planned duration in weeks	Specific Deliverable in terms of the project	Status
1	Literature Survey- Understanding the problem, use cases and related research	2	Document	Done
2	Feasibility Study	2	Document	Done
3	Understanding different IPC approaches	2	Document	Done
4	Implementation using gRPC	3	Backend service	Done
5	Implementation using RabbitMQ	3	Backend service	Done
6	Performance testing and analysis	2	Document	Done
7	Solution Architecture	2	Document	Done

Table 4: Detailed Plan of Work

KEY CHALLENGES FACED DURING THE PROJECT

- There hasn't been any prior study that outlines the various IPC techniques and their tradeoffs.
- There isn't enough information on how to use gRPC and RabbitMQ as IPC solutions, or how to compare and contrast them methodologically.
- Within SAP, there aren't many teams that employ gRPC-based or message queues-based IPC techniques there was some uncertainty about the project's feasibility

REFERENCES

- Shafabakhsh, Benyamin, Robert Lagerström, and Simon Hacks. "Evaluating the Impact of Inter Process Communication in Microservice Architectures." *QuASoQ@ APSEC*. 2020.
- Wang, Xingwei, Hong Zhao, and Jiakeng Zhu. "GRPC: A communication cooperation mechanism in distributed systems." *ACM SIGOPS Operating Systems Review* 27.3 (1993): 75-86.
- Pahl, Claus, and Pooyan Jamshidi. "Microservices: a systematic mapping study." *CLOSER (1)* (2016): 137-146.
- https://ieeexplore.ieee.org/document/4012603
- https://www.baeldung.com/grpc-introduction
- RabbitMQ Tutorials — RabbitMQ