Evaluation of Inter-Process Communication Techniques in Microservices Architecture

Kanishk Sharma, Uddashay Kumar Gupta, Aman Agarwal
Department of Computer Application, School of Information and Technology & Engineering
Vellore Institute of Technology Vellore – 632014, Tamil Nadu, India

Abstract

With the substantial growth of cloud computing over the past decade, microservices have gained significant popularity in the industry as a new architectural pattern. In the microservices architecture large applications are broken into a collection of small, independent, and distributed packages. Since microservices-based applications are distributed, one of the key challenges when designing an application is the choice of mechanism by which services communicate with each other. There are several approaches for implementing Interprocess communication (IPC) in microservices, such as synchronous and asynchronous mode of communication and each comes with different advantages and trade-offs. In the proposed paper we will take an experimental approach to compare and contrast common forms of IPC communications. In this thesis, IPC methods have been categorized into Synchronous and Asynchronous categories. The Synchronous type consists of REST API and Google gRPC, while the Asynchronous type is using a message broker known as RabbitMQ. For test environment Spring Boot will be used to create an web application based on the microservices architecture coupled with the MySQL database to store data. Tests will be conducted on both the aforementioned techniques to find the throughput of the system and also to check the availability of each systems by calculating their Mean Time to Failure (MTTF) and Mean Time to Recovery (MTTR).

Keywords: *Cloud Computing, Microservices, IPC, MTTF, MTTR*

Introduction

Cloud Computing is becoming more popular than ever and more businesses are considering migrating and running their infrastructure in the cloud. To fully leverage the potential that the cloud can offer, companies require a shift in the way they design and architect their software applications. But, it is also important to note that this approach could even result in higher operating costs in the long term. It is important for the new application to be designed with a cloud-first strategy in mind, as well as re-architecting legacy applications prior to moving to the cloud. One of the major changes in software architecture is to decompose a large monolithic application into a set of smaller, fine-grained, and independent services that work together in a distributed network.

Over the past few years, microservices architecture has earned enormous attention and gained popularity in the industry. Microservices architecture has helped large organizations such as Amazon and Netflix to serve millions of requests per minute. Microservices promotes loosely coupled systems and independently deployable components where each service can have its own stack of programming languages, frameworks and database models.

Problem Discussion

As of today, there are no concrete explanations or any standardized approach that can help to decide the right IPC method when designing microservices-based applications. Due to this reason, there is a lot of confusion as to which method and what are the trade-offs for choosing that method. It becomes particularly more challenging to decide as there is no right or wrong option, but rather more or less applicable based on the technical and functional requirements. So in this paper we aim to propose an experiment to compare and contrast between different architectures available for development of distributed applications.

Literature Survey

Evaluating the Impact of Inter-Process Communication in Microservices Architecture

(QuASoQ 2020 8th International Workshop on Quantitative Approaches to Software Quality)

In this paper, the author has simulated an E-Commerce based web application based on the microservices architecture to evaluate the performance of Inter-Process Communication (IPC) between different web services.

The author has conducted experiments on both synchronous and asynchronous communication techniques. REST API and gRPC are used to simulate synchronous communication while RabbitMQ which is based on the Advanced Message Queuing Protocol (AMQP) is used to simulate asynchronous communication.

Five different Microservices are used for the experiment which is built using NodeJs. Data is stored in MongoDB and MySQL and Docker is used to containerizing the microservices which are deployed on the Microsoft Azure Kubernetes Cluster Service.

Two different types of testing were done to evaluate the results.

In the first test, the number of virtual users was limited and the test duration lasted 180 seconds. Throughput was calculated within the specified duration of 180 seconds. The results of this test indicated that the gRPC performed better than both RabbitMQ and REST API which indicates that the synchronous form of communication can offer higher throughput when compared to the asynchronous form of communication when the load of the system is relatively low.

The number of virtual users in the second case has increased four times as compared to the first case. The outcome of the second testing experiment implies a considerable difference between the synchronous versus asynchronous forms of communication both in throughput and latency when the number of parallel requests increases. In this test, the asynchronous form of communication using RabbitMQ has outperformed the other two methods indicating that the asynchronous mode of communication works better when the load in the system is relatively high.

A Survey of Publish/Subscribe Middleware Systems for Microservice Communication
(2021 5th International Symposium on Multidisciplinary Studies and Innovative Technologies)

In this paper, the author has examined the Publisher/Subscriber communication model in the context of microservices. Publish/Subscribe communications, often known as pub/sub, is a powerful interprocess communication (IPC) mechanism that allows asynchronous service-to-service communication. It offers an alternative to the classic client-server architecture. A client communicates directly with an endpoint in the client-server model. The pub/sub model separates the client(s) who transmits the message (publisher) from the client(s) who receive the message (subscribers). Publishers and subscribers are completely unaware of the existence of the other. A third component called the broker manages the link between them. The broker's responsibility is to filter all incoming messages and properly distribute them to subscribers.

The author has performed a survey that showcases how pub/sub-middleware systems are used across different fields of computer science from Inter-Process Communication between web services to real-time applications such as the stock market and even in machine learning, image processing, and deep learning to process streams of data.

Similar to the last paper the author has reached the conclusion that for large-scale applications microservices are the way to go while monolithic architecture still being the go-to choice for small-scale applications.

Performance Evaluation of Microservices Architectures using Containers
(2015 IEEE 14th International Symposium on Network Computing and Applications)

In this paper, the author has analyzed the performance of microservices running in two different models: master-slave and nested container. The author has compared the performance of these two models with respect to CPU and network performance.

The author has performed five different tests to evaluate the CPU and the network performance and compares them with three different types of test environments - regular containers, nested containers, and virtual machines.

The CPU test is conducted using Sysbench which shows that the regular containers are always the fastest approach followed by nested containers and virtual machines.

The Network test is conducted using the Netperf benchmark to study the communication overhead of different technologies. It summarizes that the physical network becomes the bottleneck and there is no significant performance impact on comparing different combinations and configurations of network virtualization technologies.

Performance evaluation of RESTful web services and AMQP protocol
(2013 Fifth International Conference on Ubiquitous and Future Networks)

This paper presents a performance comparison study of RESTful Web services and the AMQP Protocol considering exchanging messages between client and server. The study is based on the average exchanged messages for a period of time.

The tests were performed by using 24 devices to continuously send messages to servers. The requests were sent continuously for 30 minutes each through HTTP protocol and the AMQP protocol. From the test it was found out that the average messages sent per second using HTTP was 125 while average messages sent using AMQP was 211. This clearly shows that asynchronous transmission can handle more requests when compared to synchronous transmission.

[1] has conducted a migration for a real-world mission-critical case study in the banking industry by transforming a monolithic-based system into microservices-based and observe how Availability and Reliability of the system changes as a result of the new architecture. The solution consists of de- composing several large components to which some of them requires to com- municate with third-party services. The services in the new architecture uses message-based Asynchronous communication as its IPC model to exchange data with each other. One of the motives behind using massed-based communication was to make services decoupled from each other. The author believes that aiming to have a simple and decouple integration between services and the following principle to handler failure will eventually lead to higher Reliability in microservice architecture. The author has argued that microservices-based architecture has given the targeted system to higher Availability as the new system is now broken into several components and decoupled from each other, which makes it possible to load-balance individual services as needed. This was particularly not possible in the legacy monolithic-based system. At the same time, the new architecture offers higher Reliability and can better cope with failures this is due to the fact that in the new system the communication relies on a message-broker that can be configured to ensure all messages get delivered eventually.

In [2], the author has compared REST API performance versus Advanced Message Queuing Protocol(AMQP)[3] protocol which is one of the protocols used in message-based communication that falls under Asynchronous category. The study has been done by measuring the averaged exchanged messages for a period of time using the REST API and AMQP. The authors performed the experiments by setting up two independent software instances that constantly receives messages for a 30 minutes period with an average 226 request per second. Each instance would process the received input message and store them into a persistent database. One of the instances was based using REST API communication, and the other one was set up using AMQP. After executing the experiments, the work has concluded that for scenarios where there is a need to receives and process-intensive amount of data, AMQP performs far better than REST API as it has a better mechanism for data loss prevention, better message organization and utilize lower hardware resources.

References:

[1] Nicola Dragoni et al. “Microservices: Migration of a mission critical system”. In: arXiv preprint arXiv:1704.04173 (2017).
[2] Joel L Fernandes et al. “Performance evaluation of RESTful web services and AMQP protocol”. In: 2013 Fifth International Conference on Ubiquitous and Future Networks (ICUFN). IEEE. 2013, pp. 810–815.
[3] Steve Vinoski. “Advanced message queuing protocol”. In: IEEE Internet Computing 10.6 (2006), pp. 87–89.