ACIDBrain: Maintaining ACID properties in microservices architecture

Xin Chen: xc2409
Nana Pang: np2630
Wode “Nimo” Ni: wn2155

How do you create an ACID-compliant system given a microservices architecture ?

Introduction

As software systems become more distributed and larger in scale nowadays, there is a growing need for novel architectures that are modularized, flexible, and scalable. Microservices architecture emerged as a popular software architecture where developers divide large application into small, self-contained components (“services”). Although the separation significantly increases flexibility and reduces difficulty in development, maintaining data consistency across these isolated databases can still be be challenging. Normally, to ensure good isolation among service modules, each module would use an isolated database. While smaller transactions within a service can still benefit from the **ACID (Atomicity, Consistency, Isolation, Durability)** properties that most traditional databases provide, there are common, larger transactions that span over multiple services. For example, a large e-commerce system will process a customer purchase by invoking multiple services (e.g. billing services, inventory, customer feedback, shipping services). When executing multiple instructions across multiple distributed databases, can we still maintain ACID at a *global* level?

This project aims to study and implement various existing design patterns that address the problem of data consistency in distributed database transactions. We would like to find out what, or what combination of these patterns yields to most efficient systems both in terms of usability and performance.

Possible Solution

- **Rollback Mechanism & Two Phase Commits Protocol:** The two-phase commit protocol (2PC) is a type of atomic commitment protocol (ACP) that coordinates all the processes. Two phase commits protocol maintain distributed atomic transaction by whether to commit or abort (rollback) the transaction (it is a specialized type of consensus protocol). The protocol achieves its goal even in many cases of temporary system failure (involving either process, network node, communication, etc. failures), and is thus widely used.
- **SAGA Event Driven Model & CQRS Views:** For most applications, the way to make microservices work and to manage distributed data successfully is to use SAGA and Command Query Responsibility Segregation (CQRS) views. In such an architecture, services communicate asynchronously using domain events, and

command/reply messages. SAGA is a sequence of local transactions. Each local transaction updates the data in one service and sends a message/event that triggers the next transaction. CQRS view is a replica of data from one or more services that is optimized for a particular set of queries. The service that maintains the view does so by subscribing to domain events. Whenever a service, updates its data it publishes a domain event.

Prototype

Financial services do this kind of thing all the time. In the context of e-commerce where multiple services and platforms involve, one microservice will execute commands while updating others if necessary. The whole process is broken down into different services under microservice architecture. As discussed, larger transactions can require data dependence across multiple components. If a user wanted to make a purchase on Amazon, the inventory system will firstly whether the sufficient amount of items are on stock, when the user checks out, the billing system needs to draft the money out of the bank account, then notifies the shipping system. If any steps in the chain fails, then all proceeding operations should rollback. Another case is that when the user returns a purchased item, if she wants to move money from my bank to your bank, there is no single transaction like you'd have in a DB. You don't know what systems either bank is running, so must effectively treat each like your microservices. In this case, my bank would move my money from my account to a holding account and then tell your bank they have some money, if that send fails, my bank will refund my account with the money they tried to send.

Experiment

First, we need to build several bank system services on cloud server such as AWS. These services have RESTful API to access information. Also, we need an API gateway to handle all the requests from clients.
Second, we will build the ACID manager for data consistency in different services. We are planning to use strategy such as rollback mechanism & two phase commits, SAGA models & CQRS Views.
Third, we will evaluate the performance based on response time and error hit rates. Also we will add our control manager in different position of Microservices architecture to evaluate the best location with high performance and low complexity.

Personal Statement

Nana Pang: I would like to use SAGA models and CQRS Views to implement the distributed data management in our prototype and test its performance in different position in Microservices. The evaluation of this method is based on the response time and error hit rates. I want to compare SAGA models with Two Phase Commit Protocol to find out the better solution of maintaining ACID in Microservices architecture.

Wode “Nimo” Ni: Software architecture plays an increasingly significant role in software development due to the larger scale of systems and the more rapid and collaborative mode of development nowadays. I think taking a higher-level look at software design can greatly inform modern programming language design, which I have particular interest in. Specifically in this project, I would like to take a closer look at the popular microservices frameworks such as Kong, and design a library/API to enforce data consistency. I am also interested in making such feature more user friendly and principally designed.

Xin Chen: Implementing a data consistent distributed system sounds interesting and pratical. Since I have learned semaphore for interprocess communication and deadlock avoidance, the solution to deadlock in multi-database should be pretty much similar. Microservices is a rather new concept to me, we need to preserve ACID properties in the course of implementation. The goal of the project is to compare the performance discrepancy between SAGA, 2 phase commit protocol and/or other possible solutions that may emerge in the future.