ACIDBRAIN: Maintaining data consistency in microservices

Nana Pang*, Wode "Nimo" Ni †, and Xin Chen ‡
Columbia University
Email: *np2630@columbia.edu, †wn2155@columbia.edu, ‡xc2409@columbia.edu

Abstract

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. Microservice architecture emerged as a popular software architecture where developers divide a 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. Our project targets maintaining data consistency among distributed microservices. To maintain data consistency, an event driven model called **Saga**[1] is often used to keep track of the data sources of each transaction. In this project, we investigate two predominant event sourcing frameworks: *Eventuate* and *Axon*, from an **end-users perspective**. We experiment with both of them by building data consistency guarantees on a dummy microservice system, and analyze the effectiveness of the two frameworks both quantitatively and qualitatively.

I. INTRODUCTION

Large software systems have been growing rapidly and becoming more distributed. As a result, the traditional monolithic software architecture, where modules of different functionalities are closely coupled and developed by the same group of developers, is now challenged by a new architectural pattern: **Microservices**. A software system that employs microservice architecture comprises of a suite of clearly defined small services, each running in its own process [2]. As a result, each service can be developed by completely separated teams and using distinct technology stack, thereby decoupling the modules and parallelizing development.

In a monolithic system, all modules typically share the same backend database and enjoy the ACID (Atomicity, Consistency, Isolation, Durability) properties provided by the database [3]. Under microservice architecture, however, each module would use a separate database to ensure good isolation among service modules, and each database might be using an entirely different technology. Therefore, large transactions that involves multiple modules, which used to run in the same database, now span across multiple databases with drastically different implementations. In this case, maintaining data consistency becomes a more complex task. **Two-phase-commit (2PC)** protocol [4] is a popular solution, where a coordinator process ensures all databases have the requested resource available before commiting the transaction. While 2PC ensures the correctness of distributed transactions, it requires all resources to be exclusively locked until the transaction finishes.

Unfortunately, in real world applications, many distributed transactions are long-lived, meaning they take much longer to finish. For example, an e-commerce application may have ordering, billing, and shipping modules, and a complete transaction of purchasing an item would involve all components and would not complete until the item is shipped. In the case of these long-lived transactions, 2PC does not scale well because it locks all databases involved in the transaction and there can be many other transactions occurring in the system simultaneously.

Among the existing solutions to maintain data consistency in distributed databases of microservice systems, we chose to look at the **Saga pattern** [1]. There are a few arguments for Sagas over traditional consensus protocols such as 2PC. Firstly, the Saga pattern does not require synchronization of all databases in a transaction, making it more suitable for long-lived transactions. Also, Saga tends to be easier to implement than 2PC, whose logic is relatively complex.

II. CONCLUSION

REFERENCES

[1] Hector Garcia-Molina and Kenneth Salem. *Sagas*, volume 16. ACM, 1987.

[2] James Lewis and Martin Fowler. Microservices: a definition of this new architectural term. *MartinFowler. com*, 25, 2014.
[3] Jim Gray et al. The transaction concept: Virtues and limitations. In *VLDB*, volume 81, pages 144–154. Citeseer, 1981.
[4] Philip A Bernstein, Vassos Hadzilacos, and Nathan Goodman. Concurrency control and recovery in database systems. 1987.