Multi-Consistency Transactional Support for Function-as-a-Service

Rafael Soares
INESC-ID, Instituto Superior Tecnico, Universidade de Lisboa
joao.rafael.pinto.soares@tecnico.ulisboa.pt

ABSTRACT

We propose to design and implement a framework to support the concurrent execution of transactions with different consistency levels in Function-as-a-Service (FaaS) environments. The goal is allow functionalities that have weak consistency requirements to execute efficiently, with minimal coordination, while, at the same time, allow functionalities that have strong consistency requirements to access the same data. We will leverage previous works, such as SALT, that combine both weak and strong transactional consistency while adding support for additional transactional levels that have been suggested for FaaS, such as Transactional Causal Consistency.

1 INTRODUCTION

Function-as-a-Service (FaaS), also known as Serverless Computing, has emerged as a key paradigm to support the execution of applications in the cloud. A significant advantage of FaaS is that users are not required to reserve resources explicitly: these are automatically provisioned by the cloud provider as needed. This paradigm requires programmers to develop their applications in the form of compositions of stateless functions that can be organized into directed acyclic execution graphs (DAGs) to implement complex functionalities. In run-time, the provider assigns the computational nodes required to execute these functions: different functions, or even different instances of the same function, can be executed in different nodes.

Functions cannot preserve state across invocations or share state in memory. To preserve and share state, functions need to store data in some external storage service. Even if the external storage service is able to offer strong guarantees to each individual function, it may be hard to offer strong consistency for a DAG composition, because functions from the same DAG may be executed by different nodes and, therefore, can be observed as different clients (that are not part of the same “session") by the storage service. For instance, it may not be trivial to ensure the atomicity of the writes performed by a DAG, if parts of the write-set are persistent by different functions. Also, when reading from the persistent store, functions from a given DAG can read version that belong to different snapshots. In most cases, functions within a DAG only have Eventual Consistency (EC) [1] guarantees.

Due to these limitations, a number of recent works have proposed to extend FaaS frameworks with support for consistent access to persistent store, including different forms of transactional guarantees. Examples of supported models include Transactional Causal Consistency (TCC) [3] and Strict Serializability [6]. To the best of our knowledge, all of these system assume that all DAGs run under the same consistency level: the one provided by the middleware. This can be overly restrictive, and impose a performance penalty of DAGs that can operate under weak consistency. For example, a social media application may require Strict Serializability for login, TCC for adding new contacts, and EC for post creation, while these functionalities may be using the same functions or key-space in different transactional contexts. DAGs that only require EC should not have their performance penalized by others that require Strict Serializibility.

2 RESEARCH OBJECTIVES

We propose to design and implement a framework to support the concurrent execution of transactions with different consistency levels in Function-as-a-Service (FaaS) environments. The goal is to allow DAGs that have weak consistency requirements to execute efficiently, with minimal coordination, while, at the same time, allowing DAGs that have strong consistency requirements to access the same data.

3 APPROACH

To achieve our goal, we will use coordination mechanisms to synchronize multiple transactional levels. There have been past approaches that mix weak and strong consistency levels at the operation granularity [2]. However, such coarse grained levels will affect the system scalability, as different functions that would interact on different key-spaces could execute concurrently. We also wish to allow the possibility of the same function being executed in different transactional levels. As such, we will base our approach on SALT [5], an isolation level that allows the concurrent execution of strongly consistent ACID transactions with weakly consistent BASE transactions at the object granularity. We aim at studying techniques to implement the SALT ideas in a FaaS

	ACID-R	ACID-W	alka-R	alka-W	causal-R	causal-W	saline-R	saline-W
causal-R	✓	✗	✓	✗	✓	✗	✓	✗
causal-W	✗	✗	✗	✗	✗	✓	✗	✓

Table 1: Conflict Table for TCC transactions. Causal represents TCC, ACID and Alka represent Strict Serializability and Saline represent EC.

environment and incorporate additional consistency criteria, such as TCC in the SALT framework

4 CHALLENGES

Multiple challenges arise when porting SALT to a FaaS environment and adding new transactional levels to it. First, SALT was designed without considering Multi-Versioning Concurrency Control (MVCC), which greatly affects the performance of transactional levels that are lenient in their snapshot choice, like TCC and Snapshot Isolation (SI). Second, SALT isolation is based on a set of pessimistic locks, dictating which transactions are allowed to be executed concurrently (weakly consistent transactions) and which transactions must be blocked and executed in complete isolation (strongly consistent transactions). When adding new weakly consistent levels like TCC, we wish to maintain the maximum concurrency between TCC and EC. However, it is not a trivial task, as EC reads must respect the atomicity and causal snapshot requirements of TCC. Finally, the coordination mechanism must be lightweight, as it may become a bottleneck for FaaS environments in terms of scalability.

5 RESEARCH DIRECTIONS

I will now present some research directions to tackle the above challenges.

- Regarding the locking system, we wish to improve upon the concurrency control design of SALT. We will introduce multi-versioning concurrency control (MVCC) to the locking design of SALT, allowing a smoother transition of optimistic approaches for TCC and SI implementations to the pessimistic locking of SALT.
- Regarding the addition of new transactional levels, an analysis of their interactions must be made to decide which transactional levels should be allowed to execute concurrently and what limitations exist between each other. Interactions with TCC are of special interest here, as translating causal information to other consistency levels is not trivial.
- Regarding the coordination mechanism, we will implement an additional layer to FaaS to provide the SALT isolation to all functions.

6 PRELIMINARY RESULTS

A first step for this work was the prototype of FaaSSI [4], a system that provides SI support for FaaS environment relying on an eventual consistent key-value store. We used an intermediate layer between the computational and storage layer to ensure system correctness, comprised of a set of servers responsible for conflict detection, known as conflict managers. We will further level our previous work on FaaSTCC [3] to add a caching system and a TCC storage to reduce some of the workload on the conflict manager nodes on read transactions. This intermediate layer will become the basis for the SALT implementation on FaaS.

Regarding interactions between TCC and other consistency levels, Table 1 shows our current design for TCC transactions in the SALT framework, without considering MVCC. The most surprising constraints for TCC transactions come in the form of Read-Write conflicts. Because TCC must be read from a causal snapshot, transactions may not read concurrently with writes, as they may break transaction atomicity. However, Write-Write transactions for weaker consistency levels are allowed, as TCC allows concurrent writes as long as values converge. Note that in a MVCC scenario, the above scenarios would be allowed, as long as additional metadata checks for causal dependencies are added.

Acknowledgments: This work was supervised by professor Luis Rodrigues in INESC-ID. This work was partially funded by Fundação para a Ciência e Tecnologia (FCT) under grant UI/BD/153590/2022 and via project UIDB/50021/2020 and DACOMICO (via OE withref. PTDC/CCI-COM/2156/2021).

REFERENCES

[1] Tabby Ward (AWS). 2020. *AWS Step Function transaction Saga Pattern.* Retrieved April 29, 2022 from https://docs.aws.amazon.com/prescriptive-guidance/latest/patterns/implement-the-serverless-saga-pattern-by-using-aws-step-functions.html

[2] C. Li, D. Porto, A. Clement, J. Gehrke, N. Preguiça, and R. Rodrigues. 2012. Making Geo-Replicated Systems Fast as Possible, Consistent when Necessary *(OSDI 12).* Hollywood, CA.

[3] T. Lykhenko, R. Soares, and L. Rodrigues. 2021. FaaSTCC: Efficient Transactional Causal Consistency for Serverless Computing *(Middleware '21).* Virtual Event, Canada.

[4] R. Soares. 2021. *An Architecture to Offer Transactional Strong Consistency for FaaS Applications.* Master's thesis. Instituto Superior Tecnico, Universidade de Lisboa.

[5] C. Xie, C. Su, M. Kapritsos, Y. Wang, N. Yaghmazadeh, L. Alvisi, and P. Mahajan. 2014. Salt: Combining ACID and BASE in a Distributed Database *(OSDI'14).* Broomfield, CO.

[6] H. Zhang, A. Cardoza, P. Baile Chen, S. Angel, and V. Liu. 2020. Fault-tolerant and transactional stateful serverless workflows *(OSDI 20).* Banff, Canada.