Fine-grained Transaction Scheduling in Replicated Databases via Symbolic Execution

Pedro Raminhas
INESC-ID/Instituto Superior Técnico
University of Lisbon
pedro.raminhas@tecnico.ulisboa.pt

Miguel Matos
INESC-ID/Instituto Superior Técnico
University of Lisbon
mm@gsd.inesc-id.pt

Paolo Romano
INESC-ID/Instituto Superior Técnico
University of Lisbon
romano@inesc-id.pt

ABSTRACT

Nowadays, the majority of modern Internet services use databases to store relevant data. These services tend to have strong scalability, high availability and fault tolerance requirements that created a strong urge for designing highly efficient database replication techniques. However, state of the art replication schemes that offer strong consistency guarantees introduces non-negligible costs to ensure consistent state among all the replicas, which greatly limits systems' parallelism due to a coarse grained transaction scheduling. Our goal is to leverage Symbolic Execution to obtain in an automatic and fine-grained way the transactions' data access patterns to deterministically and efficiently schedule transactions.

KEYWORDS

Database Replication, Transaction Scheduling, State Machine Replication, Symbolic Execution, SAT Solver

1 INTRODUCTION

Nowadays, databases are the norm to store relevant data, like an inventory of an online store. These services tend to have strong requirements for what concerns scalability, high availability and fault tolerance, which created a strong urge in the database community to design efficient database replication techniques. In an ideal scenario, replication allows tolerating crashes of individual replicas while increasing the perceived system availability by placing multiple copies of applications' data across failure-independents machines. However, state of the art replication schemes still incurs non-negligible costs in order to ensure that the state maintained by the various replicas is properly synchronized. These overheads are even more exacerbated in strong consistency scenarios, i.e. where applications must guarantee a consistent state among all replicas.

A typical approach to ensure consistency in replicated system is the so-called *Two-Phase Commit* (2PC) [1]. 2PC encompasses, as the name suggests, two phases: an agreement phase where the coordinator sends the set of locks to be acquired by each replica, followed by a confirmation phase where the coordinator sends an acknowledgement/rollback message according to the successful replicas' lock acquisition. However, 2PC is notably known to incur distributed deadlocks or in case of timeout usage, transaction starvation.

State Machine Replication (SMR) was introduced to overcome the aforementioned limitations. In a nutshell, SMR is based on an order-then-execute approach. In each round, replicas first reach an agreement using some consensus protocol, on a totally ordered set of (deterministic) operations to be executed at all replicas. Next, the set of operations are executed independently at each replica in an order that is consistent with the total order established during the agreement phase.

Since the actual order of execution may differ between two replicas due to a myriad of causes, e.g. network latency, different hardware or different scheduling policies, replicas' internal state may end up diverging. Thus, to avoid inconsistency, the system must ensure that transactions executing at different replicas are serialized in the same order.

Calvin [13] ensures the same serialization order by estimating transactions' data access patterns through a *reconaissance-phase*, i.e. a single-threaded execution of the application before actually executing it. Using the estimation of the data access patterns, Calvin acquires the required locks before executing transactions. Thus, guaranteeing deterministic state changes in all replicas. However, the *reconaissance-phase* implies that the application must be run twice and does not provide any correctness guarantees: values previously read may have changed meanwhile, which makes transactions to rollback.

An alternative approach to overcome the aforementioned limitations is the usage of the concept of conflict classes, as proposed by Kemme et. al [6]. This way, transactions that conflict, i.e. belong to the same conflict class, are serialized in the same order, whereas transactions that belong to different conflict classes can execute in parallel, thus delivering maximum throughput while maintaining consistency among replicas. Unfortunately, current schemes for devising conflict classes are far from optimal. They either rely on the programmer input, which is both not optimal and hard to do; or in automatic schemes that provide coarse-grained classes by typically acquiring locks whenever a table is accessed. Thus leading to either incorrect programs or poor parallelism degree.

This work addresses the aforementioned limitations of SMR-based solutions by providing a fine-grained estimation of transactions' conflict classes in an autonomous and accurate way without the need for executing them (unlike Calvin [13]) or any apriori knowledge on the transactions conflict classes (unlike the solution proposed by [6]). This is done recurring to a Symbolic Execution (SE) engine [2, 4], such as Java Path Finder [4]. With the fine-grained information provided by SE , we are investigating mechanisms to efficiently schedule transactions' relying on both locking and deterministically scheduling mechanisms.

EuroDW'18, April 2018, Porto, Portugal
2018.

2 RELATED WORK

As this work targets Database Replication, specifically SMR, a well-known technique to build *fault-tolerant* systems. It clearly has relations to works [7, 10] that totally-order conflicting requests and avoid synchronization costs for commutative operations. Also, works [5, 9] that optimistically deliver and execute operations before the final order provided by the consensus arrive, which reduces latency if the final order and optimistic order match.

This work is also included in a set of works that investigate how to avoid conflicts between operations and parallelize non-conflicting operations. Clements et. al proposed *COMMUTER* [3], a commutative rule for OS Kernels that allows developers to define an interface which states which operations do conflict, and it provides an operation order that maximizes software scalability. Shapiro et. al [12] defined CRDT, a data structure which can be replicated across multiple computers in a network where the replicas can be updated independently and concurrently without any coordination. However, if conflicts do arise, conflicts are solved recurring to costly merging mathematical operations. Finally, *SYMPLE* [11], a system for performing MapReduce-style group by-aggregate queries that automatically parallelizes Users Defined Aggregations using Symbolic Execution. The unresolved dependencies are treated as symbolic values and evaluated at run-time using the concrete input.

3 APPROACH

Our goal is to fill an important gap in the literature: enhancing the parallelism of SMR-based solutions through the usage of fine-grained conflict classes while guaranteeing a deterministic transaction schedule among all the replicas.

This is done recurring to SE, a technique originally developed for software testing (but used also in other domains, like security), which allows for determining every possible execution branch of a code block, based on the value of its input parameters. This is an idea, to the best of our knowledge, still unexplored in the literature: to use SE techniques to have a fine-grained information about the conflict classes accessed. The information extracted, offline, via SE will be used to build an on-line deterministic transaction scheduler, which can be exploited in SMR-based replication schemes to ensure that replicas execute transactions according to the same serialization order while maximizing parallelism of non-conflicting transactions.

Figure 1 depicts the architecture of the system that we are currently developing. Instead of having a single master that uses replicas as its failover mechanism, we target a SMR system where each replica maintains a complete copy of the data.

In a nutshell, our system is composed by 3 modules: SE engine, Scheduler and Contention Manager. The SE engine is responsible by, at compile-time, to analyze the application's stored procedures and retrieve the possible execution paths according to possible inputs. The Scheduler is responsible to choose the best scheduling policy for a given batch of transactions by consulting the Contention Manager module, which is responsible to determine the existence of conflicts based on the information of transactions currently executing.

We gather the information of the possible execution paths and their read and write-set recurring to a module that we developed

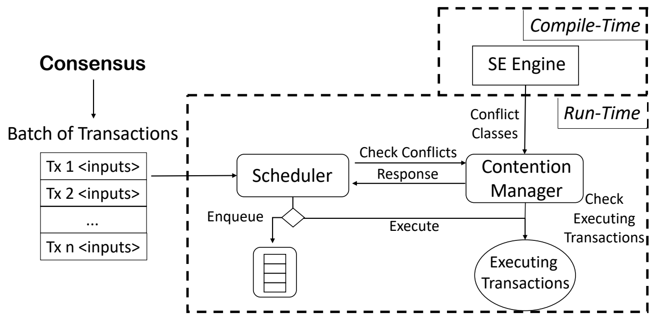

Figure 1: Architecture of System

on top of the SE engine. Its responsibility is to perform an one-time offline analysis of the application code and assign variables of operations that retrieve or change the underlying store, such as a Key-Value Store (KV), as symbolic variables. Finally, still at compile-time, the SE engine symbolically analyzes the application code abstracting these operations by just gathering the associated symbolic variables. The final result is that for each possible execution path, the SE engine retrieves the associated path constraints alongside the possible reads and writes performed based on the transactions' inputs.

Since, the number of possible paths can be very large, and moreover, there could be situations where paths can be compressed due to the fact of being sub-parts of other paths or leading to the same read/write-set. We are investigating how to model the information provided by SE to: 1) eliminate duplicate paths or paths that lead to the same read/write-set (in practice representing the same transactional access) and 2) accelerate path exploration by modeling it as a structure, e.g. a graph, that allows to efficiently explore paths without re-reading previously explored paths.

Then, at run-time, all the replicas reach an agreement via a consensus algorithm, typically a Paxos variant, in the order of transactions to be executed by all the replicas. Then, the Scheduler module deterministically picks a transaction from the batch of transactions retrieved from consensus and asks the Contention Manager module if the transaction can execute with a determined set of inputs.

Currently, we are investigating how we can develop a lightweight efficient Contention Manager module recurring two different mechanisms, namely:

1) Solver - By using a SAT solver, e.g. Microsoft Z3, we aim to first, transform the information provided by the SE engine into boolean conditions that encompass each transaction's path conditions as well as the duration of each transaction. Then, at run-time, the batch of transactions received is deterministically scheduled by the SAT solver in order to provide the maximum parallelism possible. The Solver-variant should use run-time information to predict transactions' duration and use it internally to efficiently schedule transactions. However, we envision that different replicas can perceive different transactions' duration, thus leading to different serialization orders. To ensure that all the replicas use the same transaction duration, we are investigating the implications of replicas having to engage periodically in a consensus protocol to determine transactions' duration.

2) Lock Table - By using a *Lock Table*, we aim to leverage the information provided by the SE engine to check before a transaction executes if it can acquire all the necessary locks. Thus, efficiently and deterministically execute transactions from the batch with the goal of maximizing the parallelism. To achieve this, the information provided by the SE Engine must be manipulated in a way that the respective path conditions and the read and write-set are converted to read and write locks in the underlying module.

We are also investigating the idea of, at run-time, use the information of the currently held locks and the transactions that are present in the batch of transactions to deterministically choose transactions that maximize parallelism. We will have in account not only the necessary locks to be acquired but also the dependencies between transactions, an idea recently proposed by Tian et. al [14].

We expect that the Solver-based solution will incur larger overheads than the Lock Table due to the fact that solving boolean expressions is well known to be costly. However, we are investigating how to leverage the Solver approach to deterministically analyze and devise the best order among transactions in a batch. We are also investigating the use of scope saving features to reduce the time taken to solve a set of constraints.

Finally, although one expects that the Lock Table will be faster than the Solver-based solution, we envision that in a scenario where transactions have to acquire a large set of locks this might not be true. We intend to further evaluate this premise in the future.

4 CONCLUSION AND FUTURE DIRECTION

This paper briefly introduces an ongoing work where we are developing a scheduling algorithm for replicated databases that leverages SE to infer the set of path constraints and read and write-set associated with each execution. We already integrated our module with Java Path Finder [4], thus we already are able to perform static analysis and infer the constraints associated to each path, as well as the read and write-sets performed according to a given set of inputs. Currently, we are developing a prototype of the Contention Manager based on the Solver approach, which allows to have fine-grained conflict classes, at the cost of solving costly boolean constraints. We are further investigating techniques that allow solving constraints in a lightweight and deterministic fashion. We intend to compare it to the Lock Table approach, which we are also currently developing, to infer the most efficient scheme.

Finally, further extension of this work seeks to understand if this technique can be extended also to relational databases. More precisely, we intend to investigate how to leverage SE performed at the code level, at the Schema level or even recurring to a Symbolic Execution engine that tests SQL code [8] to achieve a fine-grained conflict detection than the one present in today's systems, i.e., at the table level.

ACKNOWLEDGMENTS

This work was supported by Portuguese funds through Fundação para a Ciência e Tecnologia via project PTDC/EEISCR/1743/2014.

REFERENCES

[1] P. A. Bernstein, V. Hadzilacos, and N. Goodman. 1987. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[2] C. Cadar, D. Dunbar, and D. Engler. 2008. KLEE: Unassisted and Automatic Generation of High-coverage Tests for Complex Systems Programs *(OSDI'08).*

[3] Austin T. Clements, M. Frans Kaashoek, N. Zeldovich, Robert T. Morris, and E. Kohler. 2013. The Scalable Commutativity Rule: Designing Scalable Software for Multicore Processors. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13).*

[4] K. Havelund and T. Pressburger. 1999. Model Checking Java Programs Using Java PathFinder. 2 (10 1999).

[5] S. Hirve, R. Palmieri, and B. Ravindran. 2014. Archie: A Speculative Replicated Transactional System. In *Proceedings of the 15th International Middleware Conference (Middleware '14).*

[6] B. Kemme, F. Pedone, G. Alonso, and A. Schiper. 1999. Processing Transactions over Optimistic Atomic Broadcast Protocols. In *Proceedings of the 19th International Conference on Distributed Computing Systems, Austin, TX, USA, May 31 - June 4, 1999.* 424–431.

[7] Leslie Lamport. 2005. *Generalized Consensus and Paxos.* Technical Report.

[8] M. Marcozzi, W. Vanhoof, and J. Hainaut. 2015. A Direct Symbolic Execution of SQL Code for Testing of Data-Oriented Applications. *CoRR* (2015).

[9] R. Palmieri, F. Quaglia, and P. Romano. 2011. OSARE: Opportunistic Speculation in Actively REplicated Transactional Systems. In *Proceedings of the 2011 IEEE 30th International Symposium on Reliable Distributed Systems (SRDS '11).*

[10] Fernando Pedone and André Schiper. 2002. Handling message semantics with Generic Broadcast protocols. *Distributed Computing* 15, 2 (2002), 97–107.

[11] V. Raychev, M. Musuvathi, and T. Mytkowicz. 2015. Parallelizing User-defined Aggregations Using Symbolic Execution. In *Proceedings of the 25th Symposium on Operating Systems Principles (SOSP '15).*

[12] S. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. 2011. Conflict-free Replicated Data Types. In *Proceedings of the 13th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS).*

[13] A. Thomson, T. Diamond, S. Weng, K. Ren, P. Shao, and D. Abadi. 2012. Calvin: Fast Distributed Transactions for Partitioned Database Systems. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD '12).*

[14] B. Tian, J. Huang, B. Mozafari, and G. Schoenebeck. 2018. Contention-Aware Lock Scheduling for Transactional Databases *(VLDB).*