Towards Efficient Client-Side Transactions for Heterogeneous Cloud Data Stores

Pedro A. Sousa
U. Minho and INESCTEC
Braga, Portugal
pedro.a.sousa@inesctec.pt

Nuno Faria
U. Minho and INESCTEC
Braga, Portugal
nuno.f.faria@inesctec.pt

José Pereira
U. Minho and INESCTEC
Braga, Portugal
jop@di.uminho.pt

Ana Nunes Alonso
U. Minho and INESCTEC
Braga, Portugal
ana.n.alonso@inesctec.pt

***Abstract*—Data intensive applications increasingly make use of multiple data stores in the cloud, providing a diversity of data and query models, as well as durability and scale trade-offs. However, this has a severe impact on reliability, as the key fault-tolerance mechanism for database systems, *i.e.* ACID transactions, is no longer available. Although it is possible to implement transactions without changes to the database servers, this either requires a proxy server, which compromises scale and availability, or a client-side layer that changes the data schema, excludes legacy applications, and adds significant overhead. We address this challenge with a proposal to delegate functionality from a client-side transactional layer to a server-side query engine such that compatibility with legacy applications is restored. We implemented a proof-of-concept and show that it significantly improves performance for analytical applications.**

***Index Terms*—Transactions, cloud computing, database systems, replication.**

I. Introduction

Cloud computing has profoundly changed the way database systems are built and used by data-intensive applications. In contrast to using a traditional SQL database system, there is now a diversity of data storage options, and applications often use a combination of them [1]. This has a profound impact on dependability, as transactional ACID guarantees in both standalone and replicated database servers have long been the foundation for recoverability and high availability [2]. This is particularly worrisome, as cloud databases are increasingly sought for data-centric applications even in critical infrastructures, such as smart grids and self-driving vehicles, where data inconsistency cannot be tolerated.

Initial proposals for cloud native database systems that focused on scalability and performance did not provide ACID transactions [3]. This made it harder to develop reliable general applications and even to operate analytical applications [4]. Eventually, new cloud-native database systems that offer ACID transactions were proposed [5]. However, this does not solve the problem for applications that use multiple data stores for different purposes, as these systems do not provide the standard API for two-phase commit distributed transactions [6].

There have been several proposals to address this problem (see Section II). On the one hand, the Cherry Garcia protocol [7], used in ScalarDB [8], provides multi-database atomic transactions over systems that have only single-item atomicity. However, it severely limits direct usage of the underlying database systems, namely, for analytical workloads. On the other hand, TiQuE [9], [10] is compatible with the original applications and analytical workloads, but requires server-side SQL support and a centralized proxy, which limits scalability and availability. This is important, as there is an increasing demand for Hybrid Transactional Analytical Processing (HTAP) [11], [12].

We address this challenge with a proposal for a cloud transactional architecture that combines the advantages of both approaches (Section III). It allows client-side transactions over a diversity of systems, even if they do not natively support transactions, while at the same time exploiting server-side SQL support where available to improve the performance and compatibility of analytical applications. We implement a proof-of-concept of this approach in ScalarDB [8] and evaluate it experimentally with simple transactional and analytical workloads (Section IV). Section V discusses the lessons learned towards better support for transactions in multi-database cloud applications.

II. Background

Add-on transactional systems for cloud-based database services take different approaches, resulting in different feature and performance trade-offs. We consider only systems where data in each store are independent, *i.e.*, there is no replication among them that introduces consistency requirements.

A. Client-side two-phase commit

The Cherry Garcia approach [7] proposes a client-side library that abstracts all transaction logic, allowing programmers to use transactions in their applications over existing cloud-native database services. It supports basic key-value put and get operations and requires only that the underlying systems are able to execute atomic read and test-and-set operations, and annotate each record with additional data, specifically, transactional meta-data and past versions. Moreover, one of the databases holds a coordinator table containing the state of all transactions and must be accessible to all clients.

The client-side transaction works as follows. When started by a client, it locally selects a globally unique identifier and a timestamp. An update to an item is stored in a local transaction cache. To read an item, the transaction first checks its cache. If it is present, then it is returned; otherwise it must be read

from the data store and needs to be verified. This is necessary because this value could be invalid if it has been updated by an unfinished transaction. In this case, the start timestamp from the transaction that wrote this record is used to validate the record. The validation is done by checking the transaction state in the coordinator table.

If the transaction is in the finished state, the value is accepted; otherwise, the starting timestamp is used to determine the transaction state. Each transaction has a maximum amount of time to execute, and, using the start timestamp, we are able to determine if a transaction which wrote a record has already exceeded said time limit. In that case, the record is reverted to its previous version stored in the record's meta-data. In case the limit has not been exceeded, the read fails and the transaction which made the read is aborted.

Transaction commit uses a two-phase commit protocol. The main function of the first step, the prepare step, is to check if the transaction can commit without conflicting with other transactions. This step is done by first marking all records in the write cache of a transaction with the transaction id, the preparing timestamp, and the preparing state. Then, each record is written in the data store using the test-and-set operation. This is done sequentially using a global order, to reduce the chance of two transactions conflicting. If even one of these operations fails, the transaction is aborted. If all operations are successful, then the preparation phase ends, and the commit phase begins. In the commit step, the transaction id is written in the coordinator table with the committed state, marking it as committed. Then, all records written during the prepare phase are updated with the new state.

Lastly, a transaction can be aborted directly, by invoking the abort method, or indirectly, due to failed read or prepare. Depending on when it was called, the abort has two different behaviors. If the abort was called before the transaction was preparing, then we only need to clear the cache. If the abort was called during the prepare phase, then all written records until the abort call must be reverted to their previous version.

ScalarDB [8] is an open source and commercially supported implementation of the Cherry Garcia protocol with several improvements. The first one is parallelizing the writes in the prepare step. The second is enabling serializable transactions, by using an extra-read or extra-write after the prepare step.

Finally, ScalarDB also provides a PostgreSQL Foreign Data Wrapper (FDW) [13] that can scan tables using the client-side transactional API and thus support SQL queries. Since the protocol supports only simple put and get operations, this allows complex analytical queries to be executed.

B. Transactions in the query engine

The TiQuE approach [9] proposes a server-side implementation of ACID transactions for SQL systems that support only single-item atomicity. The key feature of this approach is that it is achieved with views and rules, allowing applications to use all SQL operations, including complex queries, in their transactions, and achieving transparency. Briefly, each table is replaced by a view that computes the current transactional snapshot from a backing-store table that keeps multiple versions of data items and transactional meta-information. Write operations on the view, such as inserts, updates, and deletes, are captured by rules, annotated with transactional meta-data, and redirected to the underlying table.

When a new transaction starts, an id and a starting timestamp are acquired and written in the log table. Then, any operation can be issued during this transaction. The views and rules previously created will ensure that operations performed by a transaction will only affect said transaction. Finally, the transaction will be committed or aborted. First, the log will be checked for any conflicting transaction, *i.e.*, any transaction that was committed after the one being prepared started, having both updated the same value. If a conflict is detected, the transaction is aborted and the respective log entry is marked as such. If not, the transaction commits and its log entry is updated with a commit timestamp.

This approach can be applied to the multi-database cloud scenario using a SQL server as a proxy that manages transactional meta-data locally [10]. This allows complex SQL statements to be executed efficiently while pushing parts of the query to underlying data stores, if their query processing capabilities allow. For key-value stores such as Cassandra, this reduces to simple put and get operations. For systems with query capabilities, such as MongoDB, this also includes aggregations.

III. Hybrid transactions

Existing approaches for client-side transactions propose different trade-offs suiting different applications. It is thus interesting to combine them so that one can execute atomic transactions across a variety of database systems while retaining support for efficient analytical and hybrid transactional-analytical processing.

A. Challenges

The main challenge is how to delegate transactional functionality from the client-side middleware layer to the server, in SQL, as in TiQuE, in a way that it interoperates with other operations, within the same transaction, that are directed at different database servers and being managed directly at the client-side. In fact, ScalarDB imposes its own schema on database systems to store meta-data and multiple versions for each item.

The second challenge is that ScalarDB and the Cherry Garcia approach inherently support only elementary read and write operations, *i.e.*, a simple key-value data store. Therefore, even if we are able to delegate part of the transactional functionality to the server-side, as done by TiQuE, we need to provide additional interfaces to allow expressing analytical queries (*i.e.*, complex SQL SELECT statements). Moreover, such an interface would not make sense for native key-value store systems, which would not be able to interpret such queries.

B. Approach

Starting with the baseline implementation of ScalarDB, we create a new driver that supports databases with a subset of TiQuE. Instead of directly accessing the meta-data in the underlying database, this driver translates the operation to its equivalent in the TiQuE schema. To make this possible, the key change to TiQuE is the implementation of two-phase commit, thus separating prepare and commit phases and exposing them as methods. The driver then calls the begin, prepare, and commit methods in the modified TiQuE. Finally, we need to create a new operation that allows us to send complex queries to TiQuE.

The TiQuE driver for ScalarDB contains all standard methods from ScalarDB, such as put, get, and delete, as well as the required methods to begin, prepare, commit, and abort a transaction. It also contains a map linking a transaction in ScalarDB to the respective transaction in TiQuE. This allows, when delegating operations, to execute them in the correct context.

With this driver, when a put is done during a ScalarDB transaction on a TiQuE database, it is applied to the database instead of keeping it in the transaction cache. This allows complex select operations in the context of the same transaction to observe their own updates. In this step, we implicitly start a TiQuE transaction in case one has not yet been started within the ScalarDB transaction. This is done lazily to maintain maximum compatibility between these systems.

When the commit is requested by the client application, the two phases are translated to TiQuE as follows. For the prepare step, instead of only performing test-and-set operations on each individual item, we call the prepare method of TiQuE. In the commit step, we do everything like the original ScalarDB with the main exception that we execute the commit method in TiQuE that updates the state of the records. In addition, we also changed ScalarDB's abort procedure to call the abort method in TiQuE, in addition to its default behavior.

Finally, we created a new operation in ScalarDB to execute a complex query operation. This operation allows us to send a SQL query directly to the underlying TiQuE database. With these three major changes, we can use a TiQuE database with ScalarDB.

As an example, consider an application that executes a transaction in which it writes an item to each of two cloud database systems and issues an analytical operation. One of them, $DB1$, is using the new TiQuE driver, while the other, $DB2$, uses a standard ScalarDB driver. It works as follows:

1) The application calls the middleware layer to begin a transaction, which initializes the local state with the global transaction id.
2) When the application writes an item to $DB1$, the operation is sent to the driver. The driver then starts a TiQuE transaction in $DB1$, collecting a TiQuE transaction id that is associated with the global transaction.
3) The write operation is now forwarded to $DB1$, where it is inserted in the cache table.

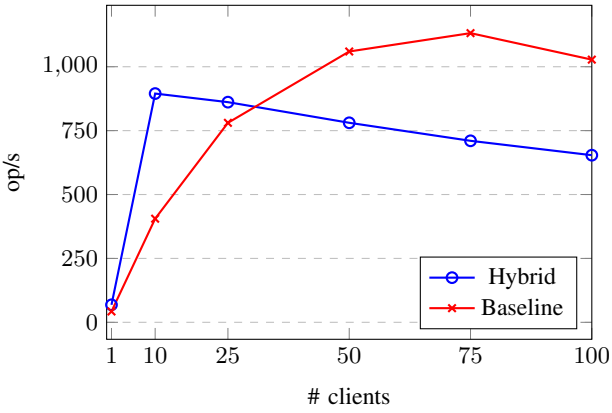

Fig. 1. Throughput with transactional workload.

4) When the application writes an item to $DB2$, the operation is simply cached locally in the middleware.
5) The application now issues a complex analytical operation. This is routed to the TiQuE driver and then to $DB1$. It is executed there making use of views to observe a current snapshot, including the previously written item in the context of the same transaction.
6) When the application requests that the transaction commits, we execute the following in parallel:
   a) The TiQuE driver requests $DB1$ to execute the prepare method.
   b) The original driver issues each of the pending writes to $DB2$ and collects responses.
7) Assuming that all responses have been positive, the coordinator table is updated, and, asynchronously, both databases are notified of the commit.

IV. Preliminary Evaluation

To assess the performance impact of a hybrid approach, we compared it experimentally with the client-side approach. All tests use two servers: One runs a PostgreSQL 12 database server and the other runs the workload generator. The servers are configured with an i3-4170 CPU with 8GB of RAM and 500GB of HDD storage, using Ubuntu 20.04.

A. OLTP workload

To assess the impact on transactional performance, we use ScalarDB's version of the YCSB benchmark [14], which is adapted for multi-operation transactions and uses ScalarDB's client API. The benchmark is configured with workload F, *i.e.*, a read-modify workload. The workload is configured to perform two read-modify operations per transaction, meaning each transaction will do a read followed by write two times. The database is populated with 1000 records and each test runs for 30 seconds, with 5 repetitions for each configuration. We benchmark considers 1, 10, 25, 50, 75, and 100 clients. The baseline ScalarDB is configured for read-committed isolation, as serializability has a significant impact on performance, and the hybrid solution provides snapshot isolation.

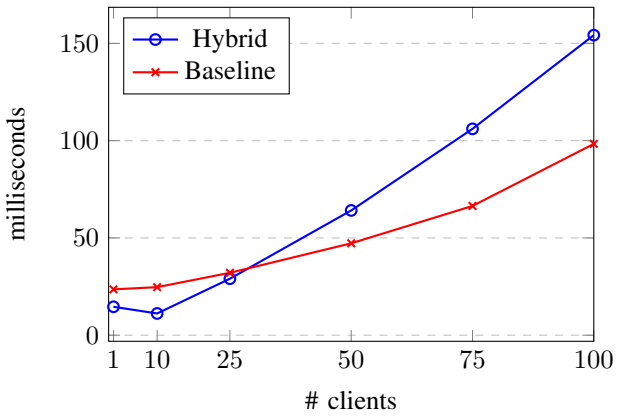

Fig. 2. Response time with the transactional workload.

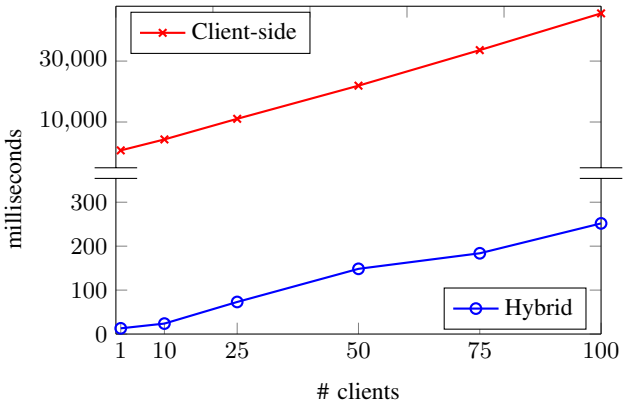

Fig. 3. Response time with the analytical workload.

The results are shown in Figures 1 and 2. They show that, on the one hand, the hybrid approach achieves higher throughput with a smaller number of clients, as it reduces the number of round-trips to the database server for committing transactions. On the other hand, the baseline ScalarDB achieves a maximum higher throughput with a larger number of clients, as TiQuE adds some overhead to the transaction, with the added cost of executing the methods of begin, prepare and commit. Of these, the commit adds the biggest overhead because of locking. In short, the hybrid and baseline approaches for transactional workloads offer slightly different trade-offs with similar performance.

B. Simple OLAP workload

To assess the impact on analytical performance, we populate the database with 10000 items containing 6 integer and 6 text columns, as commonly found in fact tables for analytical applications. Then, we measure the response time for running an aggregation query (*i.e.*, a sum of one column). Again, we use 5 repetitions for each configuration and run this benchmark for 1, 10, 25, 50, 75, and 100 clients.

The results are shown in Figure 3. Note that there is a discontinuity in the y-axis as there is now a very large difference in the performance of the two alternatives. The reason for this is that the baseline ScalarDB implementation provides only a key-value interface and has to download all the data to the query engine for each execution. On the other hand, the hybrid approach keeps a transparent representation of transactional meta-data at the server and can thus upload the query, which executes natively.

C. Complex OLAP workload

To further test that the proposed approach can execute complex analytical queries, we resorted to CH-BenCHmark [15], which proposes a set of analytical queries similar to the TPC-H analytical benchmark adapted to run on the TPC-C schema. These queries include joins over multiple tables, aggregations, and sorting operations that greatly benefit from the ability to fully optimize them. Namely, taking advantage of pushing down filtering operations to reduce the amount of data being processed. Moreover, they also take advantage of index ordering to improve the performance of joins and aggregations. For these tests, we use the same hardware and software configuration as the previous tests, populating the database with data for one warehouse.

As an example, our proposal improves the performance of Query 1, which is a simple query using only one table, from 2.9 to 0.9 s. However, with the more complex Query 4 that includes a subquery, the improvement is from 648.6 to 1.0 s.

V. Conclusions

In this paper, we address the challenge of providing a client-side transactional layer that is efficient for both operational and analytical workloads, as well as compatible with a diversity of systems. We achieve this by combining two existing approaches, ScalarDB and TiQuE.

A preliminary evaluation shows that this has a small impact on transactional performance. However, since with ScalarDB and TiQuE we can send analytic queries directly to the data store instead of using a client-side query engine like ScalarDB does, response time for analytical workloads decreases significantly.

Although the current delegation of analytical queries requires them to be sent entirely to a single database system, it would be interesting as future work to further improve the client-side query engine such that complex queries can span different database systems, while parts of the query that target the one supporting TiQuE be still accelerated.

Acknowledgment

This work is co-financed by Component 5 – Capitalization and Business Innovation, integrated in the Resilience Dimension of the Recovery and Resilience Plan within the scope of the Recovery and Resilience Mechanism (MRR) of the European Union (EU), framed in the Next Generation EU, for the period 2021–2026, within project ATE, with reference 56.

REFERENCES

[1] M. Stonebraker and U. Cetintemel, "One size fits all: an idea whose time has come and gone," in *21st International Conference on Data Engineering (ICDE'05)*. IEEE, 2005, pp. 2–11. [Online]. Available: https://ieeexplore.ieee.org/document/1410100

[2] G. Weikum and G. Vossen, *Transactional Information Systems*. San Francisco: Morgan Kaufmann, 2002.

[3] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," *Oper. Syst. Rev.*, vol. 41, no. 6, pp. 205–220, Oct. 2007. [Online]. Available: https://doi.org/10.1145/1323293.1294281

[4] M. Armbrust, T. Das, L. Sun, B. Yavuz, S. Zhu, M. Murthy, J. Torres, H. van Hovell, A. Ionescu, A. Łuszczak, M. Świtakowski, M. Szafrański, X. Li, T. Ueshin, M. Mokhtar, P. Boncz, A. Ghodsi, S. Paranjpye, P. Senster, R. Xin, and M. Zaharia, "Delta lake: high-performance ACID table storage over cloud object stores," *Proceedings VLDB Endowment*, vol. 13, no. 12, pp. 3411–3424, Aug. 2020. [Online]. Available: https://doi.org/10.14778/3415478.3415560

[5] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford, "Spanner: Google's globally distributed database," *ACM Trans. Comput. Syst.*, vol. 31, no. 3, pp. 1–22, Aug. 2013. [Online]. Available: https://doi.org/10.1145/2491245

[6] D. Processing, "The XA specification," *X/Open Company Ltd*, 1991.

[7] A. Dey, A. Fekete, and U. Rohm, "Scalable distributed transactions across heterogeneous stores," in *2015 IEEE 31st International Conference on Data Engineering*. IEEE, Apr. 2015, pp. 125–136. [Online]. Available: https://ieeexplore.ieee.org/document/7113278

[8] H. Yamada, T. Suzuki, Y. Ito, and J. Nemoto, "ScalarDB: Universal transaction manager for polystores," *Proceedings VLDB Endowment*, 2023. [Online]. Available: https://www.vldb.org/pvldb/vol16/p3768-yamada.pdf

[9] N. Faria, J. Pereira, A. Nunes Alonso, R. Vilaça, Y. Koning, and N. Nes, "Tique: Improving the transactional performance of analytical systems for true hybrid workloads," *Proceedings of the VLDB Endowment*, vol. 16, no. 9, pp. 2274–2288, 2023.

[10] N. Faria, J. Pereira, A. N. Alonso, and R. Vilaça, "Towards generic fine-grained transaction isolation in polystores," in *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*, ser. Lecture notes in computer science. Cham: Springer International Publishing, 2021, pp. 29–42. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-93663-1_3

[11] G. Li and C. Zhang, "HTAP databases: What is new and what is next," in *Proceedings of the 2022 International Conference on Management of Data*. New York, NY, USA: ACM, Jun. 2022. [Online]. Available: https://dl.acm.org/doi/10.1145/3514221.3522565

[12] A. Prout, S.-P. Wang, J. Victor, Z. Sun, Y. Li, J. Chen, E. Bergeron, E. Hanson, R. Walzer, R. Gomes, and N. Shamgunov, "Cloud-native transactions and analytics in SingleStore," in *Proceedings of the 2022 International Conference on Management of Data*, ser. SIGMOD '22. New York, NY, USA: Association for Computing Machinery, Jun. 2022, pp. 2340–2352. [Online]. Available: https://doi.org/10.1145/3514221.3526055

[13] "PostgreSQL documentation – 5.12. foreign data," The PostgreSQL Global Development Group, 2020. [Online]. Available: https://www.postgresql.org/docs/13/ddl-foreign-data.html

[14] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proceedings of the 1st ACM symposium on Cloud computing*, 2010, pp. 143–154.

[15] R. Cole, F. Funke, L. Giakoumakis, W. Guy, A. Kemper, S. Krompass, H. Kuno, R. Nambiar, T. Neumann, M. Poess, K.-U. Sattler, M. Seibold, E. Simon, and F. Waas, "The mixed workload CH-benCHmark," in *Proceedings of the Fourth International Workshop on Testing Database Systems*, ser. DBTest '11. New York, NY, USA: Association for Computing Machinery, 2011. [Online]. Available: https://doi.org/10.1145/1988842.1988850