Contents

Distributed Transactions

Junlong Gao

Atomicity, Consistency, Isolation Levels, and Durability (ACID), are the corner stones for the storage system programing interface: transactions [5]. In reality, these promises are harder and harder to keep when increasing the scale of the storage system [1].

Why distributed transactions? We need distribution in storage systems for mainly two purposes: scalability of performances and redundancy for reliability. In some systems, redundancy requires distributed transactions to ensure consistency across replicas, this includes some virtual SAN implementation and file systems [4]. The form of transactions in replication is somewhat simpler: their main purpose is for Durability and Atomicity.

In some other recent architectures, where storage system is implemented with two layers [2, 3, 6, 9, 11], replication is pushed down to the lower layer and can use other approaches like replicated logs or gossip for replication. In these cases, when the entire system storage is sharded for scalability, transactions involves multiple resources manager will require some form of two-phase commit to implement ACID semantics.

But the boundary of these use-cases is not black-and-white. Imagine implementing a virtual SAN system with RAID-6 erasure coding for data blocks. There are 4 data nodes and 2 syndrome nodes, and redundancy is achieved by contacting any 4 nodes out of the 6 to recover a data block [7]. In this case, is it replication, or sharding? Erasure coding works by generating 6 syndrome blocks from the 4 data blocks (one coding group), and two-phase commit is used to ensure these 6 pieces of data end up all-or-nothing durably on the 6 nodes. On the other handing, reading the 4 consecutive blocks in the same coding group can be distributed to any 4 of the 6 nodes, so read performance benefits from this distribution. There is no obvious replicated state machine approach the author know of to achieve this, hence two-phase commit is used.

The discussion gets even further into what kind of programming interface these distributed transactions offers. Is it key-value space, or Structured Query Language? In traditional MySQL implementation, SQL is done on a b-plus tree, which is effectively a key-value storage engine. Now it is interesting to see that when implementing a distributed SQL database, it is usually done over a thin SQL layer on a distributed key-value storage layer. There is a rabbit hole on how journaling should be done in key-value layer and b-plus tree layer [5, 8].

This article explores the distributed transactions and architectures entails in the two-layered approaches.

1 Physical and Logical Databases

In traditional centralized database designs, the system is already in 2 layers internally. SQL operations always generate key-value operations on the b-plus tree, and the b-plus tree is the physical representation of these tuples. The physical database, sometimes called "the storage engine", is physical in a sense that they are closer to disk (b-plus tree pages are literally disk pages), and relies on the underlying hardware for page atomicity (but no structured update). The logical database relies on the storage engine for storing the tuple, yet itself can have its own logging and in-memory state like cache. In traditional design, mutation of the database is expressed in the logical layer by translating SQL to key-value operations, and then translated to the physical structure mutation, and journaled for atomicity and durability.

For Google Spanner and Amazon Aurora, this relationship is turned "inside-out" [3,9], in the sense that logical mutations are first-class citizens in the data path instead of hidden beneath the internal structure. In fact, some open-source implementation like TiDB explicitly allows using the underlying logical database TiKV directly. In these cases, mutations must first be done in a logical manner: key-value operations are logged in the persistent storage, and picked up by the physical layer to replay onto the physical database, regardless which storage engine it uses. Popular implementations of LSM tree shows its value in write-intensive workloads.

2 Jounraling

The two-layered approach can benefit from the underlying key-value store being both highly available and highly scalable. Logical journaling allows this separation so that one can focus on solving data reliability and availability problems in the key-value layer. Also, journaling at the logical layer reduces write-amplification dramatically over the network [9]. Finally, logical journaling is also naturally suitable for consensus protocols implementing replicated logs like Raft, provided records in the logical journal are idempotent. In AWS Aurora, they have their own implementation of replicated log based on gossip and checkpoint progressing [10] (curiously, there is no notion of distributed transactions as there is no sharding in Aurora either).

Now it is worth mention the differences in Aurora and the rest of the world. Aurora is using a more involved form called Physiological Logging as this is what original MySQL uses [8]. The log records describes how the b-plus tree pages to be mutated in terms of logical key-value operations, without providing the entire post-image of the page. In this sense, the write path will still have to stick with the traditional database design where latches are used to guard pages dirtied without sharding to different nodes, and in this case Aurora can only support one writer. In this sense, the two layers in Aurora between the logical layer and the physical layer are still interleaved.

In Spanner however, transactions are done in the key-value and key-value form only, and there is no dependency on the underlying physical database. Writing is a matter of

logging the key-value pairs in the Paxos log, and no need to handle dirty page latches in a distributed manner. Then, sharding can be done in the key space and each shard has a Paxos file for its share of journaling. To make sure the update across shards are updated atomically and durability, an additional two-phase commit protocol is thus required.

3 Timestamps and Concurrency

[*TODO: Explain serailibility and linearizability and how timestamps can be used*]

References

[1] Peter Bailis, Aaron Davidson, Alan Fekete, Ali Ghodsi, Joseph M Hellerstein, and Ion Stoica. Highly available transactions: virtues and limitations (extended version). *arXiv preprint arXiv:1302.0309*, 2013.

[2] Wei Cao, Zhenjun Liu, Peng Wang, Sen Chen, Caifeng Zhu, Song Zheng, Yuhui Wang, and Guoqing Ma. Polarfs: an ultra-low latency and failure resilient distributed file system for shared storage cloud database. *Proceedings of the VLDB Endowment*, 11(12):1849–1862, 2018.

[3] Brian F. Cooper. Spanner. *Proceedings of the 6th International Systems and Storage Conference on - SYSTOR '13*, 2013.

[4] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 29–43, 2003.

[5] Jim Gray and Andreas Reuter. *Transaction processing: concepts and techniques*. Elsevier, 1992.

[6] Daniel Peng and Frank Dabek. Large-scale incremental processing using distributed transactions and notifications. 2010.

[7] James S Plank. Erasure codes for storage systems: A brief primer. *; login:: the magazine of USENIX & SAGE*, 38(6):44–50, 2013.

[8] Seppo Sippu and Eljas Soisalon-Soininen. *Transaction processing: Management of the logical database and its underlying physical structure*. Springer, 2015.

[9] Alexandre Verbitski, Anurag Gupta, Debanjan Saha, Murali Brahmadesam, Kamal Gupta, Raman Mittal, Sailesh Krishnamurthy, Sandor Maurice, Tengiz Kharatishvili, and Xiaofeng Bao. Amazon aurora: Design considerations for high throughput cloud-native relational databases. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1041–1052, 2017.

[10] Alexandre Verbitski, Anurag Gupta, Debanjan Saha, James Corey, Kamal Gupta, Murali Brahmadesam, Raman Mittal, Sailesh Krishnamurthy, Sandor Maurice, Tengiz Kharatishvilli, et al. Amazon aurora: On avoiding distributed consensus for i/os, commits, and membership changes. In *Proceedings of the 2018 International Conference on Management of Data*, pages 789–796, 2018.

[11] Sage A Weil, Scott A Brandt, Ethan L Miller, Darrell DE Long, and Carlos Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 307–320, 2006.