Designing Atomic Commit for Cloud DBMSs with Storage Disaggregation

Zhihan Guo
University of Wisconsin-Madison

Databases are migrating to the cloud because of desirable features such as elasticity, high availability, and cost competitiveness. Modern cloud-native databases feature a *storage-disaggregation* architecture where the storage is decoupled from computation as a standalone service. This architecture allows independent scaling and billing of computation and storage, which can improve resource utilization, reduce operational costs, and enable flexible cloud deployment with heterogeneous configurations. Many cloud-native database systems adopt such an architecture for both OLTP [18, 20, 15, 5] and OLAP [3, 2, 4, 17, 7, 10]. Nowadays, as storage services offer essential functions such as fault tolerance, scalability, and security at low cost, systems start to layer their designs on the existing disaggregated storage services [6, 8].

This work focuses on optimizing Two-Phase Commit (2PC), a widely-used atomic commit protocol, on existing storage services without requiring customized APIs. Therefore, a database can adopt existing highly optimized storage services and thereby avoid the expense of developing a new one, and can also allow the storage to adopt new mechanisms (e.g., new replication protocols) independently. Here we will discuss two parts: (1) how to optimize 2PC leveraging that the disaggregated storage service is highly-available and shared among compute nodes; (2) what is the minimal requirement from existing storage services to implement the optimized protocol.

We introduce **Cornus**, an optimized 2PC protocol specifically designed for the storage disaggregation architecture in cloud-native databases. Cornus makes two major changes to conventional 2PC. First, **it eliminates decision logging by the coordinator**, which significantly reduces the latency of 2PC. A transaction is committed as soon as all participants have written `VOTE-YES` in their log in response to prepare requests in phase one. This optimization is feasible because highly-available disaggregated storage ensures that after the log is written, it will not be lost. Second, Cornus denotes a *LogOnce()* API based on compare-and-swap functionality to **address the blocking problem.** The API allows multiple participants to read and update the same log file and ensures that only the first log append for a transaction can update this transaction's state. This feature allows any participant that is uncertain about the transaction's outcome to force an abort by writing a vote to an unresponsive participant's log.

To deploy Cornus on existing storage systems, Cornus requires an atomic operation so that we can leverage it to implement *LogOnce()* semantic in the compute layer. The atomic operation can be either a conditional write (write only if a condition holds) or an atomic append (two concurrent writes must have happen-before order). We demonstrate the specific setup and possible implementations on Microsoft Azure Storage [9] using Etag [1]. Azure Storage assigns an identifier to each object. The identifier is updated every time the object is updated and an HTTP GET request returns the identifier as part of the response using the Etag (entity tag) header defined within HTTP. A user updating an object can send in the original Etag with an "If-Match" conditional header so that the update will be performed only if the stored ETag matches the one passed in the request [1]. Thus, we can have *each partition/participant maintaining a corresponding log file as an object* in the storage. Each participant keeps track of the Etag in the compute layer and uses the conditional header to ensure *LogOnce()* semantic.

While Cornus directly runs on top of a highly-available storage service through a consensus-agnostic interface, many prior works [11, 14, 16, 19] manage the replication on their own and co-design 2PC with replication protocols for further optimizations. Although these protocols cannot be directly applied to existing storage services, we still evaluate them to show the potential optimization space when having different assumptions. We performed both theoretical and empirical evaluations on 2PC, Cornus, Paxos Commit [11] and MDCC [14]. For 2PC and Cornus, we assume the underlying storage runs a Multi-Paxos instance for each entity (coordinator/participant) since the others are based on Paxos and its variations. To briefly summarize, Cornus can save up to 2.5 RTT than 2PC without exposing replication details to compute nodes in this context. More complex protocols like MDCC and Paxos Commit that combine atomic commit with replication can achieve 1 RTT less than Cornus in latency, but loses flexibility since the storage layer and compute layer need to be co-designed. Moreover, these Paxos-based co-design protocols only depict the ideal latency when all network communications take an equal amount of time. It is not clear how much these protocols can gain in the cloud where network latency fluctuates and servers have jitter [13].

As a note, Cornus [12] is published and will be presented in VLDB 2023. In addition to the materials presented in the paper, we would like to share more details on the underlying storage services. We could elaborate on how to implement *LogOnce()* on existing storage services with atomic append APIs (while the paper focuses on storage with conditional write APIs), how it enables group commits, and how Cornus interacts with different consistency models and replication schemes in the storage.

1. REFERENCES

[1] Managing concurrency in microsoft azure storage. https://azure.microsoft.com/en-us/blog/managing-concurrency-in-microsoft-azure-storage-2/, 2014. (visited on 2022/03/01).

[2] Amazon Athena — Serverless Interactive Query Service. https://aws.amazon.com/athena, 2018. (visited on 2022/03/01).

[3] Amazon Redshift. https://aws.amazon.com/redshift, 2018. (visited on 2022/03/01).

[4] Presto. https://prestodb.io, 2018. (visited on 2022/03/01).

[5] P. Antonopoulos, A. Budovski, C. Diaconu, A. Hernandez Saenz, J. Hu, H. Kodavalla, D. Kossmann, S. Lingam, U. F. Minhas, N. Prakash, et al. Socrates: The new sql server in the cloud. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1743–1756, 2019.

[6] M. Armbrust, T. Das, L. Sun, B. Yavuz, S. Zhu, M. Murthy, J. Torres, H. van Hovell, A. Ionescu, A. Łuszczak, M. undefinedwitakowski, 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. volume 13, page 3411–3424. VLDB Endowment, aug 2020.

[7] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, et al. Spark SQL: Relational Data Processing in Spark. In *SIGMOD*, 2015.

[8] M. Brantner, D. Florescu, D. Graf, D. Kossmann, and T. Kraska. Building a database on s3. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, page 251–264, New York, NY, USA, 2008. Association for Computing Machinery.

[9] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, et al. Windows azure storage: a highly available cloud storage service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 143–157, 2011.

[10] B. Dageville, T. Cruanes, M. Zukowski, V. Antonov, A. Avanes, J. Bock, J. Claybaugh, D. Engovatov, M. Hentschel, J. Huang, et al. The Snowflake Elastic Data Warehouse. In *SIGMOD*, 2016.

[11] J. Gray and L. Lamport. Consensus on transaction commit. *ACM Trans. Database Syst.*, 31(1):133–160, Mar. 2006.

[12] Z. Guo, X. Zeng, K. Wu, W.-C. Hwang, Z. Ren, X. Yu, M. Balakrishnan, and P. A. Bernstein. Cornus: atomic commit for a cloud dbms with storage disaggregation. *Proceedings of the VLDB Endowment*, 16(2):379–392, 2022.

[13] P. Helland. Decoupled transactions: Low tail latency online transactions atop jittery servers. 2022.

[14] T. Kraska, G. Pang, M. J. Franklin, S. Madden, and A. Fekete. Mdcc: Multi-data center consistency. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 113–126, 2013.

[15] D. Malkhi and J.-P. Martin. Spanner's concurrency control. *ACM SIGACT News*, 44(3):73–77, 2013.

[16] R. Taft, I. Sharif, A. Matei, N. VanBenschoten, J. Lewis, T. Grieger, K. Niemi, A. Woods, A. Birzin, R. Poss, et al. Cockroachdb: The resilient geo-distributed sql database. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1493–1509, 2020.

[17] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy. Hive — A Petabyte Scale Data Warehouse Using Hadoop. In *ICDE*, 2010.

[18] A. Verbitski, A. Gupta, D. Saha, M. Brahmadesam, K. Gupta, R. Mittal, S. Krishnamurthy, S. Maurice, T. Kharatishvili, and X. 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.

[19] I. Zhang, N. K. Sharma, A. Szekeres, A. Krishnamurthy, and D. R. Ports. Building consistent transactions with inconsistent replication. *ACM Transactions on Computer Systems (TOCS)*, 35(4):1–37, 2018.

[20] J. Zhou, M. Xu, A. Shraer, B. Namasivayam, A. Miller, E. Tschannen, S. Atherton, A. J. Beamon, R. Sears, J. Leach, D. Rosenthal, X. Dong, W. Wilson, B. Collins, D. Scherer, A. Grieser, Y. Liu, A. Moore, B. Muppana, X. Su, and V. Yadav. *FoundationDB: A Distributed Unbundled Transactional Key Value Store*, page 2653–2666. Association for Computing Machinery, New York, NY, USA, 2021.