View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Infoscience - École polytechnique fédérale de Lausanne

The design of Wren, a Fast and Scalable Transactional Causally Consistent Geo-Replicated Key-Value Store

Diego Didona, Kristina Spirovska and Willy Zwaenepoel
École polytechnique fédérale de Lausanne

Abstract

This paper presents the design of Wren, a new geo-replicated key-value store that achieves Transactional Causal Consistency. Wren leverages two design choices to achieve higher performance and better scalability than existing systems.

First, Wren uses hybrid logical physical/clocks to timestamp data items. Hybrid clocks allow Wren to achieve low response times, by avoiding the latencies that existing systems based on physical clocks incur to cope with clock skew.

Second, Wren relies on a novel dependency tracking and stabilization protocol, called Hybrid Stable Time (HST). HST uses only two scalar values per update regardless of the number of data centers and nodes within a data center. HST achieves high resource efficiency and scalability at the cost of a slight increase in remote update visibility latency.

We discuss why Wren achieves higher performance and better scalability than state-of-the-art approaches.

1. Target Model

Consistency model. Wren is a new geo-replicated key-value store that achieves Transactional Causal Consistency (TCC) [3]. CC [2] is the strongest consistency model that can be achieved in an always available fashion for individual operations [8]. TCC extends causal consistency with the abstraction of interactive read-write transactions. As such, TCC achieves the strongest semantics compatible with high-availability and low-latency [3]. In addition, Wren can achieve convergence [8] in different fashions, e.g., using the last-writer-wins rule [10], user-defined commutative and associative functions [7] or CRDTs [3, 9].

System model. We assume a distributed key-value store whose data-set is split into N partitions. Each key is deterministically assigned to one partition according to a hash function. Each partition is replicated at M different data centers. Hence, a full copy of the data is stored at each site. We further assume nodes communicate through point-to-point lossless FIFO channels. Each server is equipped with a physical clock that advances monotonically. Such clocks are loosely synchronized by a time synchronization protocol such as NTP [1]. The correctness of Wren does not depend on the synchronization precision.

2. The design of Wren

Clients establish a session towards a node in their local data center. This node acts as coordinator of a client's transactions and as proxy towards other nodes in the data center. Clients manipulate data by means of transactions. A client starts a transaction, performs some reads and writes, and then commits the transaction.

Upon starting, a transaction T is assigned a snapshot S by the coordinator node. S represents a causal snapshot, i.e., it includes the effects of all transactions that causally precede T. S also includes all the items produced by remote transactions whose effects are visible in the local data center. The visibility of a remote transaction is determined by a stabilization protocol, as we shall describe later. S is represented by a local timestamp and a remote one. They define the upper bound on the commit time of local, resp. remote, transactions whose effects (i.e., updates) are visible to T.

Upon committing, T gets a commit timestamp, which is used as update time of the items updated by T. This timestamp reflects causality among transactions: if T has a timestamp t, it means that T potentially depends on all the transactions committed in the local data center with timestamp smaller than t.

The commit timestamp of a transaction T is determined by means of a 2PC protocol. Every node that stores an item written by T proposes a timestamp and the maximum is chosen as commit timestamp. Every update produced by T also carries the remote entry of the snapshot S assigned to T at the beginning of T's execution. The combination of commit time and remote snapshot timestamp determines the snapshot to which T and the items written by T belong to.

Once T is committed, each server p involved in T appends T and the set of modifications to be applied on p to a commit list. The effects of transactions are applied in increasing commit timestamp order. The updates of a transaction are also replicated to remote data centers upon being applied on the local server.

Hybrid Logical/Physical Clocks (HLC). To ensure correctness, transactions in Wren are applied in increasing commit timestamp order. Assume a transaction T writes x on node p and y on node p'. Assume that p proposes t as commit timestamp and p' proposes $t' > t$. Then, by picking the maximum

of the proposed values, T is timestamped with t'. To enforce correctness, the effect of T cannot be applied on p until p is sure that it will never commit another transaction that writes x and that has a lower timestamp than t'.

Assume that t and t' are timestamps taken from the servers' physical clocks, as it is the case in GentleRain [5] and Cure [3], two state-of-the-art approaches to (T)CC. Then, it is possible that the clock on p will catch up with t' in a long time because of clocks skew. During this time, p stalls any transaction that wants to access the snapshot of the data store that includes the version of x written by T.

Wren overcomes this limitation by using HLC [6] to generate timestamps. A HLC has a physical part $HLC.p$ and a logical part $HLC.l$. Likewise, a hybrid timestamp t is a pair $t = \langle t.p, t.l \rangle$. A hybrid timestamp with logical part equal to 0 corresponds to a physical timestamp. The physical part of a HLC on a node p is always greater than or equal to the value of the physical clock of p. It can, in fact, be moved forward with respect to the local physical clock value to reflect causality among events. The logical part is, similarly, increased when p needs to generate a timestamp that is higher than any previous timestamp generated on p and the physical clock of p is lower than the physical component of the current HLC on p.

With reference to the previous example, as soon as p receives the (hybrid) commit timestamp t' for T, p (whose $HLC.p$ is smaller than $t'.p$) moves its HLC to t'. Because the HLC is used to generate the proposed commit timestamps, p is guaranteed to not commit any later transaction with timestamp smaller than t': the smallest timestamp that p can now generate is $\langle t'.p, t'.l+1 \rangle$. Hence, p is able to make promptly visible to later transactions the version of x generated by T, without stalling them because of clock skew.

Recent work shows that clock skew severely hinders the performance of transactional operations in causally consistent systems [3, 4]. We argue that, by means of HLC, Wren will be able to achieve better performance than state-of-the-art approaches to TCC. Our claim is also backed by our recent successful experience in using HLC to achieve (non-Transactional) CC incurring a latency up to 3.5x lower than the one achieved by a design based on physical clocks [4].

Hybrid Stable Time (HST). HST is the novel stabilization protocol employed by Wren to determine when the updates generated by a transaction can become visible to clients in a data center. As with most causally consistent systems, HST allows updates originating in a data center DC to become visible immediately in DC. A remote update d, instead, becomes *stable*, i.e., visible to clients in DC when all the causal dependencies of d have been received in DC.

The key feature of HST is that every data item tracks dependencies by means of only two scalar timestamps, regardless of the scale of the system. One entry tracks the dependencies on local items. One entry summarizes the dependencies on remote items.

HST hits a sweet spot in the meta-data size vs performance spectrum. On one side, the use of only two scalar values enables high scalability and throughput in Wren. This contrast with the design of Cure, in which the dependency meta-data grows linearly with the number of data centers [3].

On the other side, decoupling local and remote dependencies ensures that a transaction can determine the visibility of a local item without the need to synchronize with remote data centers. This is, for example, the case of GentleRain [5]. In GentleRain, dependencies are tracked by means of a single timestamp. Likewise, the snapshot visible to a transaction is determined by a single timestamp t. Then, to be sure that a node has installed all the items belonging to the snapshot, a transaction is stalled until the local data center has received *all* the updates with timestamp smaller than t coming from *all* the data centers . This synchronization step can lead to a latency that is in the same order of magnitude of the communication delay between data centers [4].

HST works as follows. Each server p_n^m maintains a version vector VV_n^m. n is the id of the data partition replicated by p_n^m, and m the id of its data center (DC_m). $VV_n^m[m]$ is the commit timestamp of the latest local transaction applied by p_n^m. $VV_n^m[i],\ i \neq m$, is the timestamp of the latest transaction applied by p_n^m that comes from the i-th replica of the n-th partition.

Periodically, servers within DC_m compute the minimum timestamp across the remote entries of all the version vectors in DC_m. This value becomes the Remote Stable Time (RST) and indicates that all the transactions coming from all the remote data centers with commit timestamp $\leq RST$ have been applied in DC_m.

When a transaction T starts, the remote entry of the corresponding snapshot S is set to be RST. The local entry, instead, is set as the maximum between the latest commit timestamp seen by the client and the hybrid clock of the server responsible for coordinating T. In this way, the freshness of S is maximized, and S also includes any previous snapshot manipulated by the client. T uses the timestamps in S to determine the freshest version of a data item that falls within the visible snapshot.

HST enables high scalability and performance at the expense of a slight increase in the time that it takes for an update to become visible in remote data centers (the so called *remote updates visibility latency*). This stems from the fact that HST only tracks a lower bound on replicated transactions coming from all the remote data centers. This effectively makes the visibility latency of an update in a remote data center DC_r dependent on the communication latency between DC_r and its furthest data center.

We believe that the performance benefits brought by HST largely outweigh the slight increase in remote update visibility latency.

3. Conclusion

We have presented the design of Wren, a new geo-replicated key-value stores that achieves TCC. Wren uses HLC and a new stabilization protocol to achieve high performance and scalability. We are implementing Wren and we plan to compare it with state-of-the-art approaches to (T)CC.

References

[1] NTP: The network time protocol. http://www.ntp.org.

[2] M. Ahamad, G. Neiger, J. E. Burns, P. Kohli, and P. W. Hutto. Causal memory: Definitions, implementation, and programming. *Distributed Computing*, 9(1):37–49, 1995.

[3] D. D. Akkoorath, A. Tomsic, M. Bravo, Z. Li, T. Crain, A. Bieniusa, N. Preguiça, and M. Shapiro. Cure: Strong semantics meets high availability and low latency. In *Proc. of ICDCS*, 2016.

[4] D. Didona, K. Spirovska, and W. Zwaenepoel. Okapi: Causally Consistent Geo-Replication Made Faster, Cheaper and More Available. *ArXiv e-prints, https://arxiv.org/abs/1702.04263*, Feb. 2017.

[5] J. Du, C. Iorgulescu, A. Roy, and W. Zwaenepoel. Gentlerain: Cheap and scalable causal consistency with physical clocks. In *Proc. of SoCC*, 2014.

[6] S. S. Kulkarni, M. Demirbas, D. Madappa, B. Avva, and M. Leone. Logical physical clocks. In *Proc. of OPODIS*, 2014.

[7] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. Don't settle for eventual: Scalable causal consistency for wide-area storage with cops. In *Proc. of SOSP*, 2011.

[8] P. Mahajan, L. Alvisi, and M. Dahlin. Consistency, availability, convergence. Technical Report TR-11-22, Computer Science Department, University of Texas at Austin, May 2011.

[9] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. Conflict-free replicated data types. In *Proc. of SSS*, 2011.

[10] R. H. Thomas. A majority consensus approach to concurrency control for multiple copy databases. *ACM Trans. Database Syst.*, 4(2):180–209, June 1979.