Orion: Time Estimated Causally Consistent Key-Value Store

Diptanshu Kakwani
IIT Madras, India
dipk@cse.iitm.ac.in

Rupesh Nasre
IIT Madras, India
rupesh@cse.iitm.ac.in

Abstract

This paper presents a causally consistent key-value store Orion, which uses a novel protocol for Read Only Transactions (ROT). Unlike most of the existing protocols, Orion uses only one round of communication in the best case, and not more than two rounds in the worst case. We provide a theoretical bound on its communication complexity and qualitatively compare it with recent ROT protocols. We also quantitatively compare Orion with state-of-the-art protocol CausalSpartanX and illustrate that Orion achieves up to 1.7× higher throughput and generates 10× fewer messages on widely-used YCSB workload.

CCS Concepts. • **Computer systems organization → Distributed architectures**; • **Software and its engineering → Consistency**.

Keywords. Causal consistency, distributed consistency, key value stores, geo-replication

ACM Reference Format:
Diptanshu Kakwani and Rupesh Nasre. 2020. Orion: Time Estimated Causally Consistent Key-Value Store. In *PaPoC'20: 7th Workshop on Principles and Practice of Consistency for Distributed Data, April 27, 2020, Heraklion, Crete, Greece.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/1122445.1122456

1 Introduction

Distributed key-value stores have become a *de facto* component of modern web-scale applications. These data stores provide low latency operations to clients across the world. Many of these data stores support eventual consistency, which is the weakest form of consistency and offers the best performance. However, eventual consistency does not provide any practically useful guarantees, except that once there are no new updates to a data item, it will eventually become consistent. Causal consistency, on the other hand, has caught up attention in recent years, as it lies in between the two extremes of eventual consistency and strong consistency. It is also the strongest form of consistency that can satisfy all the three properties of CAP [10, 15], which makes it a good candidate for distributed applications.

In the past few years, various protocols for enforcing causal consistency have been proposed. A majority of these protocols focuses on optimizing *read* operations since real-world workloads are often read-heavy [3, 14]. In particular, most protocols support Read Only Transactions (ROT) [11, 12], which allows a client to read multiple keys from a causally consistent snapshot. In this work, we present a novel ROT protocol, named Orion, to reduce communication in the underlying distributed network. A salient feature of Orion is that it uses only one round of communication in the best case, unlike the usual, fixed two rounds in other protocols. Reduction in rounds is achieved by carefully *predicting* the stable vector of data items. We implemented this protocol in DKVF key-value store framework [18], and observe that our proposal outperforms the state-of-the-art protocols on YCSB workload [6].

1.1 Contributions

This paper makes the following contributions[1]:

- A novel ROT protocol for causally consistent key-value store that uses only one round of communication in the best case, and not more than two rounds in the worst case.
- Theoretical comparison of Orion with the recent causally consistent key-value stores.
- Benchmarking Orion against the existing state-of-the-art system CausalSpartanX [17] on YCSB workload, and illustrating that it achieves upto 1.7× throughput improvement.

1.2 Outline

Section 2 describes the system model. Section 3 describes the proposed Orion protocol. Section 4 quantitatively evaluates Orion and compares it against CausalSpartanX protocol. Section 5 compares and contrasts with the related work, and Section 6 concludes with directions for future work.

[1]Source code of Orion is available at https://github.com/dipkakwani/Orion

PaPoC'20, April 27, 2020, Heraklion, Crete, Greece

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

2 System Model

In our system, we consider a distributed multi-version key-value store with full replication across all data centers (DCs). Keys are partitioned using a hash function, and each partition is replicated in all the data centers. The data store has n partitions and m data centers. Partition i located in data center j is denoted by p_i^j. Each partition periodically sends a heartbeat to its replicas in other data centers. We assume that each client is *sticky* to one data center, to ensure highly available transactions with causal consistency [4].

A client can read and write a key k using $GET(k)$ and $PUT(k, value)$ operations respectively. In addition, a client can perform read-only transaction as defined in [11, 12]. $ROT(K = \{k_1, k_2, k_3, ...\})$ returns causally consistent values of all the keys in the set K.

3 Orion Protocol

Orion protocol extends the CausalSpartan protocol [16] to support ROT. Similar to CausalSpartan, Orion uses Hybrid Logical Clock (HLC), which assigns timestamps to events using both logical and physical clocks. In case of concurrent updates, Orion uses *last-writer-win* [23] strategy to resolve conflicts, *i.e.*, write with the higher timestamp overwrites a lower timestamp write. If two writes have the same timestamp, the tie is broken using DC id. Each partition maintains a Version Vector VV[2], which has m timestamps, one for each data center. $VV[j]$ stores the timestamp of the latest update received from the j^{th} data center. Partitions within the same data center exchange their VVs periodically and compute Data center Stable Vector (DSV) by finding entry-wise minimum values. A timestamp t in $DSV[j]$ indicates that the local data center has received all the updates happened till clock value t at the j^{th} data center. Client also maintains DSV_c, which is the entry-wise maximum DSV value seen by the client since it may contact different partitions within the same data center.

DSV alone is insufficient to enforce causality. It captures only stable timestamps across data centers, whereas the most recent update within a partition can have a higher timestamp. Each client also maintains a Dependency Set DS_c, which contains tuples of $\langle i, t \rangle$, where t is the highest timestamp of the versions read, which were originally written at data center i. Each version x_i of a key also has Dependency Set $x_i.DS$, which is computed using DS_c at the time of the creation of the version.

Figure 1 shows an example of Orion ROT protocol. The client issues $ROT(a, b)$ at DC_1 with the predicted snapshot vector $SV = [15, 23, 12]$. We explain how client makes this prediction in the next section. Keys a and b reside in partitions P_i^1 and P_j^1 respectively. Since $SV <= DSV_i^1$ and $SV <= DSV_j^1$, the SV is valid in both the partitions. Partition then finds the latest version which satisfies the constraint $x_i.DS <= SV$. The underlined versions in Figure 1 satisfy the constraint and are returned to the client.

[2]We use capital letters for vectors, tuples and sets, and lower case letters for scalar values.

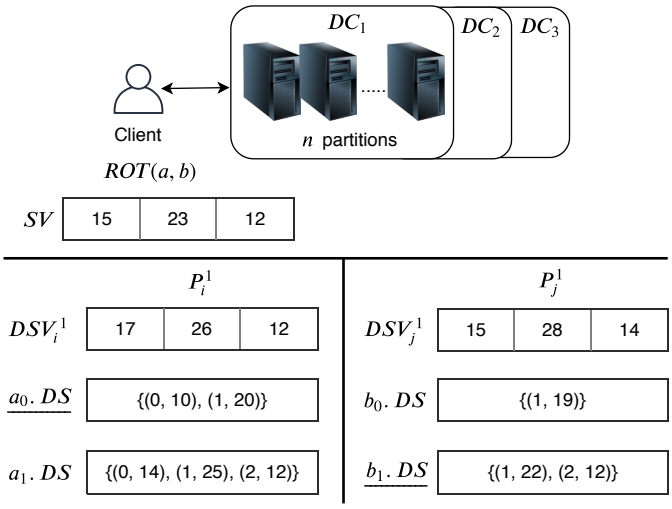

Figure 1. Example of Orion ROT protocol

Algorithm 1 Client side algorithm for ROT

```
1:  function ROT(K)
2:      ct = currentTime()
3:      V = {}
4:      SV = DSV + ct − (τ_h + τ_dsv + LAT + t)
5:      K_p = {k ∈ K | findPartition(k) = p} ∀p ∈ {1..n}
6:      t = currentTime()                    ▷ Update last ROT time
7:      for K_i ∈ K_p ≠ φ do
8:          send ⟨ROT K_i, SV⟩
9:          receive ⟨ROTReply V_i, DSV, DS⟩
10:         if V_i = DSV = DS = {} then
11:             ▷ Retry ROT with old DSV value
12:             SV = DSV
13:             RETRY
14:         DSV_c = max(DSV_c, DSV)
15:         DS_c = maxDS(DS_c, DS)
16:         V = V ∪ V_i
17:     return V
```

3.1 ROT Client Protocol

Algorithm 1 presents the client-side ROT processing. In this protocol, the client *predicts* a SV, instead of relying on a coordinating server to find out a valid snapshot. Such a prediction crucially avoids an extra step of communication. The prediction uses latency between the servers as one of the parameters. Sovran et al. [19] experimentally showed that updates replicated across data centers take time uniformly distributed in the interval $[RTT_{max}, 2 \times RTT_{max}]$ where RTT_{max} is the maximum round-trip latency between different data centers.

The prediction can be made in multiple ways. In our implementation, as shown in line 4, the client performs the prediction using these parameters: previous DSV, current time at the client ct, heartbeat period of servers τ_h, DSV computation interval of partitions τ_{dsv}, latency between servers LAT, and the timestamp of the client in the last ROT. In words, the formula adds the time elapsed since the last ROT while taking into account the delay with which any update is propagated between servers. Note that we are assuming that the network latency between the servers is known a priori to the client. For simplicity, we use the maximum latency between any two servers in our implementation. This formula can be explained by tracing the exact time it takes for a timestamp at one server to reach another server. The clock increments after every heartbeat interval τ_h, and then it is sent to other replicas. The time taken for this message to reach other replicas is proportional to the latency LAT between the servers. Once it reaches a replica, it is propagated to other partitions in the same DC as part of DSV computation τ_{dsv}. A possible alternative formulation can consider using the frequency with which PUT operations are performed instead of relying on heartbeat interval.

The client combines keys from the same partition and sends those as one ROT message, as shown in Line 5. We tried multiple ways to send ROT messages: sending messages in parallel, sending one key per message, etc. However, sending messages asynchronously with all the keys per partition achieved the best performance. If the client receives an empty response (Line 10), the client retries with a safe value of SV, without any prediction. This would ensure a valid result but may result in stale values. In our experiments, we have found that 99 percent of the predictions had a small Mean Absolute Error (MAE) of less than or equal to 76ms (explained in Section 4).

3.2 ROT Server Protocol

Algorithm 2 presents the server-side ROT processing in Orion. It checks the validity of SV received from the client. The client may have mispredicted and sent an invalid SV, if at least one of the SV entries is greater than the corresponding entry in DSV (Line 2). In such a case, the server gives a second chance to the client. It waits until a pre-configured timeout and rechecks for validity of SV. If the check fails again, then the server aborts the request and sends an empty result to the client. The client must now retry with a lower SV value. Such a second-chance allows the protocol to keep a smaller timeout (to improve latency), and still reduce the number of aborts (to improve throughput).

The timeout is a user-defined parameter, which puts a hard limit on how long to wait. It can make the protocol non-blocking or blocking. If SV is valid, the server performs read operation on each key using SV, which returns the latest version of the key visible in the snapshot. The server then returns the resulting value set V, along with the DSV and the maximum DS to the client.

Algorithm 2 Server side algorithm for ROT

```
1:  Upon receive ⟨ROT K, SV⟩
2:      if isValidSV(SV, DSV) = FALSE then
3:          wait(timeout)
4:          if isValidSV(SV, DSV) = FALSE then
5:              send ⟨ROTReply {}, {}, {}⟩
6:              Exit
7:      V = {}
8:      DS_max = {}
9:      for k ∈ K do
10:         V_k, DS = read(k, SV)
11:         V = V ∪ V_k
12:         DS_max = maxDS(DS_max, DS)
13:     send ⟨ROTReply V, DSV, DS_max⟩
14: function isValidSV(SV, DSV)
15:     if SV_i <= DSV_i, ∀i ∈ {1..m} then
16:         return true
17:     else
18:         return false
```

4 Experimental Evaluation

Orion is implemented on top of DKVF [18][3]. We compare Orion against CausalSpartanX [17] which is a state-of-the-art distributed implementation of causally consistent data store. The experiments were performed on Google Cloud Platform (GCP) using N1-Standard-2 VMs, each having 2 vCPUs and 7.5 GB RAM. We use 3 DCs: Iowa (US-Central), Finland (Europe-North), Taiwan (Asia-East) with each DC having 4 VMs and 4 partitions. We populate the setup with YCSB workload [6].

4.1 Communication Improvement

Having single-round processing allows Orion to improve overall communication substantially. To quantify the effect, we measure the number of messages generated. Figure 2 compares the total number of messages generated by both the systems in ROT transactions. CausalSpartanX involves server to server communication and generates slice request messages to get values of remote keys. As the number of keys in ROT increases, we see a linear increase in the number of messages. In Orion, in contrast, the number of messages increases both slowly and linearly, as it involves only client to server communication. This clearly illustrates the performance benefit of Orion.

4.2 Prediction Accuracy

A crucial aspect of Orion is client-side prediction, and to be effective, the prediction accuracy must be low. We calculate

[3] https://github.com/roohitavaf/DKVF

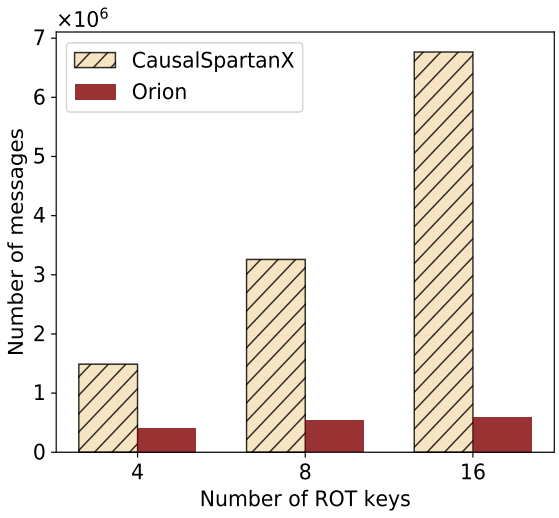

Figure 2. Comparison of number of messages

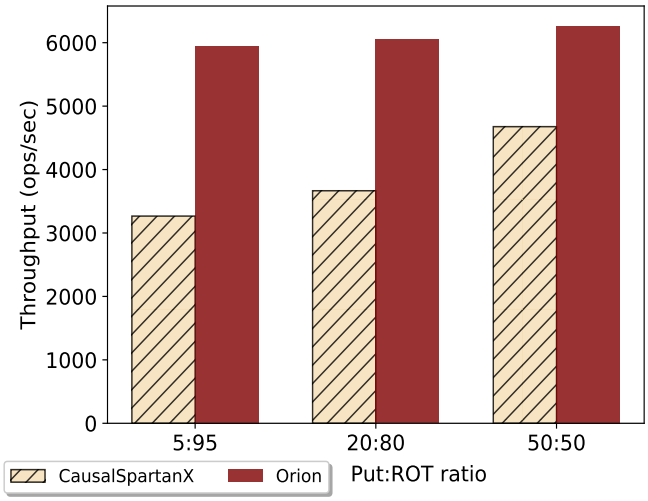

Figure 4. Throughput with varying put:ROT ratio

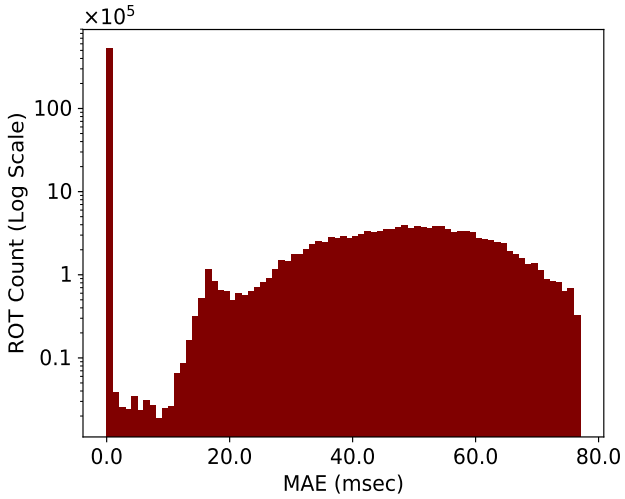

Figure 3. Prediction accuracy of 99 percentile ROTs

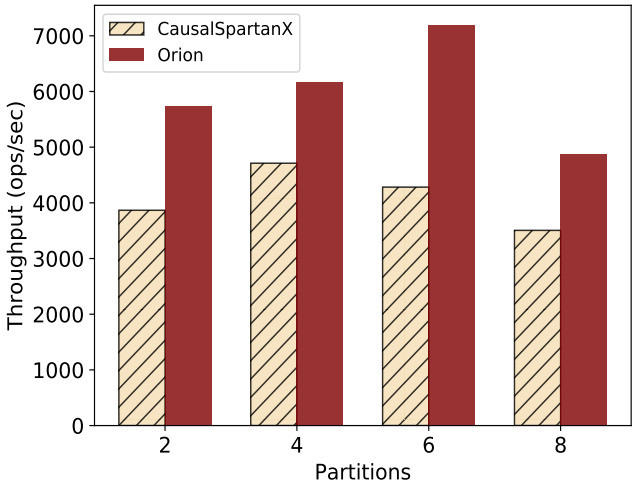

Figure 5. Throughput with varying number of partitions

the prediction accuracy using Mean Absolute Error (*MAE*) between *SV* (predicted *DSV*) and the actual *DSV*, $MAE = \frac{1}{m}\sum_{i=1}^{m}|SV_i - DSV_i|$. Figure 3 shows the histogram plot of MAE value and count of ROT transactions. We limit to 99 percentile of MAE values, to discount the initial 0 prediction made by the clients. A large majority of the predictions (~80%) is exact and has no error. We observe that there are very few predictions with error marginally above zero. This is because our prediction is most likely to be off by the factors included in the equation in Algorithm 1, line 4. We also observe the local maxima around 50, which is also the *DSV* computation interval used in the experiment. Overall, less than 1% of transactions have *invalid* predictions. With the timeout interval set to 200 ms, none of the transactions timed out.

4.3 Effect of Put:ROT Ratio

Figure 4 shows the overall throughput (number of operations per second) with varying put:ROT ratio. Since ROT operation is relatively slower than put (10 keys by default in one ROT operation), we see an increase in throughput in both the systems as we decrease the ROT proportion in the workload. However, the throughput increases relatively faster in CausalSpartanX as ROTs are much slower than put operation in CausalSpartanX compared to those in Orion.

4.4 Effect of Number of Partitions

The number of partitions is an important factor that affects the overall system performance. Figure 5 shows the throughput with varying number of partitions. Overall, the throughput increases as we increase the number of partitions. However, with 8 partitions, the hardware becomes the bottleneck as the underlying VM has 2 cores, and each of the 4 VMs runs 2 processes and an extra client process runs on one of the VMs. Therefore, we see a drop in throughput for 8 partitions in both the systems. Interestingly, even with 6 partitions CausalSpartanX throughput drops. A factor which affects this behavior is increase in communication between partitions, as run time of ROT in CausalSpartanX is dependent on communication between *all* pairs of partitions.

System	NB[1]	#R[2]	Comm[3]		C[4]
			c↔s	s↔s	
POCC[20]	✗	2	$K+M$	$K \times M + K$	P
CausalSpartanX[17]	✓	2	$K+M$	$K \times M + K$	H
Wren[21]	✓	2	$K+2$	$K \times 2 + K$	H
Contrarian[7]	✓	2	$K+M$	$K \times M + K$	H
PaRiS[22]	✓	2	$K+1$	$K+K$	H
Orion	✓	≤ 2	$K \times M + K$	0	H

[1] Nonblocking
[2] Number of rounds
[3] Communication complexity (including metadata) of ROT with K keys and M DCs
[4] Clock - Hybrid (H) or Physical (P)

Table 1. Comparison of Orion with recent causally consistent key-value stores.

5 Related Work

Earlier systems such as COPS [11], Eiger [12], Bolt-on causal consistency [5] and COPS-SNOW [13] used explicit dependency tracking to enforce causal consistency, which caused extra overhead in these systems. ChainReaction [2], Orbe [8], GentleRain [9], Cure [1] and more recent systems such as POCC [20], Wren [21], CausalSpartanX [17], Contrarian [7] and PaRiS [22], use variants of vector clocks instead of explicit dependency tracking. These systems use either physical clock, logical clock, or a combination of these two clocks. Depending on the clock, ROT can either be blocking or nonblocking. Also, all these systems use a coordinating server in ROT protocol which causes an extra round of communication among the servers.

Table 1 shows the ROT properties of the recent systems and Orion. The basic structure of the ROT protocol of these systems is as follows. Client sends a key set along with the metadata to the coordinating server. The coordinating server calculates a stable snapshot timestamp and sends query to the respective partitions based on the keys in ROT, along with the snapshot timestamp. Once it receives replies from other partitions, it sends the result back to the client. As shown in Table 1, the client to server communication is dependent on the number of keys K and size of the metadata. On the other hand, the server to server communication is dependent on the number of partitions communicated, which is the number of keys in the worst case, and size of the metadata. Note that even though the metadata-size required for ROT in some systems is smaller, that cost is transferred to periodic metadata exchange messages between the partitions. For example, in PaRiS, the partitions exchange *DSV* with all other partitions across all data centers, including the local data center. We chose CausalSpartanX for comparison because it makes a reasonable trade-off between metadata communication and the size of the transaction. Orion requires only client to server communication since the client predicts the stable snapshot timestamp. Orion can take 2 rounds of communication if the client performs a wrong prediction, but in the best case, it takes only 1 round. As discussed in Section 4, less than 1% of transactions had invalid predictions in our setup.

6 Conclusion and Future Work

In this paper, we presented a novel protocol for causally consistent ROTs and showed both theoretically and empirically that it outperforms the previous state-of-the-art protocols. We used a prediction technique to generate a snapshot vector at the client-side, instead of relying on servers for generating the vector. As part of future work, we would like to explore various other prediction techniques that might perform better and do not explicitly depend on server configuration parameters. For example, instead of relying on the client to provide latency parameters between servers, the servers compute it during run time and share it with the clients as part of the initial handshake.

Acknowledgments

We thank K. C. Sivaramakrishnan for his valuable inputs in our discussions. This work is partially supported by MEITY project CS/19-20/1123/MEIT/008606.

References

[1] Deepthi Devaki Akkoorath, Alejandro Z Tomsic, Manuel Bravo, Zhongmiao Li, Tyler Crain, Annette Bieniusa, Nuno Preguiça, and Marc Shapiro. 2016. Cure: Strong semantics meets high availability and low latency. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 405–414.

[2] Sérgio Almeida, João Leitão, and Luís Rodrigues. 2013. ChainReaction: a causal+ consistent datastore based on chain replication. In *Proceedings of the 8th ACM European Conference on Computer Systems*. 85–98.

[3] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. 2012. Workload analysis of a large-scale key-value store. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on Measurement and Modeling of Computer Systems*. 53–64.

[4] Peter Bailis, Aaron Davidson, Alan Fekete, Ali Ghodsi, Joseph M Hellerstein, and Ion Stoica. 2013. Highly Available Transactions: Virtues and

Limitations. *Proceedings of the VLDB Endowment* 7, 3 (2013).

[5] Peter Bailis, Ali Ghodsi, Joseph M Hellerstein, and Ion Stoica. 2013. Bolt-on causal consistency. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. 761–772.

[6] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM symposium on Cloud computing*. 143–154.

[7] Diego Didona, Rachid Guerraoui, Jingjing Wang, and Willy Zwaenepoel. 2018. Causal consistency and latency optimality: friend or foe? *Proceedings of the VLDB Endowment* 11, 11 (2018), 1618–1632.

[8] Jiaqing Du, Sameh Elnikety, Amitabha Roy, and Willy Zwaenepoel. 2013. Orbe: Scalable causal consistency using dependency matrices and physical clocks. In *Proceedings of the 4th annual Symposium on Cloud Computing*. 1–14.

[9] Jiaqing Du, Călin Iorgulescu, Amitabha Roy, and Willy Zwaenepoel. 2014. Gentlerain: Cheap and scalable causal consistency with physical clocks. In *Proceedings of the ACM Symposium on Cloud Computing*. 1–13.

[10] Seth Gilbert and Nancy Lynch. 2002. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *Acm Sigact News* 33, 2 (2002), 51–59.

[11] Wyatt Lloyd, Michael J Freedman, Michael Kaminsky, and David G Andersen. 2011. Don't settle for eventual: scalable causal consistency for wide-area storage with COPS. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. 401–416.

[12] Wyatt Lloyd, Michael J Freedman, Michael Kaminsky, and David G Andersen. 2013. Stronger semantics for low-latency geo-replicated storage. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. 313–328.

[13] Haonan Lu, Christopher Hodsdon, Khiem Ngo, Shuai Mu, and Wyatt Lloyd. 2016. The SNOW Theorem and Latency-Optimal Read-Only Transactions. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 135–150.

[14] Haonan Lu, Kaushik Veeraraghavan, Philippe Ajoux, Jim Hunt, Yee Jiun Song, Wendy Tobagus, Sanjeev Kumar, and Wyatt Lloyd. 2015. Existential consistency: measuring and understanding consistency at Facebook. In *Proceedings of the 25th Symposium on Operating Systems Principles*. 295–310.

[15] Prince Mahajan, Lorenzo Alvisi, and Mike Dahlin. 2011. *Consistency, availability, convergence.* Technical Report.

[16] Mohammad Roohitavaf, Murat Demirbas, and Sandeep Kulkarni. 2017. Causalspartan: Causal consistency for distributed data stores using hybrid logical clocks. In *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 184–193.

[17] Mohammad Roohitavaf, Murat Demirbas, and Sandeep Kulkarni. 2018. CausalSpartanX: Causal Consistency and Non-Blocking Read-Only Transactions. *arXiv preprint arXiv:1812.07123* (2018).

[18] Mohammad Roohitavaf and Sandeep Kulkarni. 2018. DKVF: a framework for rapid prototyping and evaluating distributed key-value stores. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 912–915.

[19] Yair Sovran, Russell Power, Marcos K Aguilera, and Jinyang Li. 2011. Transactional storage for geo-replicated systems. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. 385–400.

[20] Kristina Spirovska, Diego Didona, and Willy Zwaenepoel. 2017. Optimistic causal consistency for geo-replicated key-value stores. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2626–2629.

[21] Kristina Spirovska, Diego Didona, and Willy Zwaenepoel. 2018. Wren: Nonblocking reads in a partitioned transactional causally consistent data store. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 1–12.

[22] Kristina Spirovska, Diego Didona, and Willy Zwaenepoel. 2019. PaRiS: Causally Consistent Transactions with Non-blocking Reads and Partial Replication. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 304–316.

[23] Robert H Thomas. 1979. A majority consensus approach to concurrency control for multiple copy databases. *ACM Transactions on Database Systems (TODS)* 4, 2 (1979), 180–209.