Rosé: Flexible Replication With Strong Semantics For Partitioned Databases

Ioannis Zarkadas
Columbia University
iz2175@columbia.edu

Kelly Kostopoulou
Columbia University
kelkost@cs.columbia.edu

Thomas Graham
Columbia University
tg2833@columbia.edu

Junfeng Yang
Columbia University
junfeng@cs.columbia.edu

Philip A. Bernstein
Microsoft Research
phil.bernstein@microsoft.com

Asaf Cidon
Columbia University
asaf.cidon@columbia.edu

Tamer Eldeeb
Columbia University
tamer.eldeeb@columbia.edu

ABSTRACT

Asynchronous primary-backup database replication is popular because it strikes a desirable balance between write latency and durability. Unfortunately, it has significant downsides. In partitioned databases, each partition is typically replicated independently, which means that data loss during failover can leave the database in an undefined state that is hard for developers to reason about. In addition, replication lag can grow over time, expose users to stale data and create durability issues. Finally, time to recovery and performance after failover can suffer if backup partitions progress unevenly.

Rosé is a novel replication scheme to address the limitations of asynchronous primary backup replication in partitioned databases, by striking a balance between full synchronicity and asynchronicity. First, databases integrate their existing snapshotting mechanisms (e.g., real-time or epochs) with asynchronous replication to provide monotonic-prefix consistency semantics at the backup. Second, in order to bound replication lag, Rosé proposes push-based replication that can track the lag and apply backpressure at the primary, in a way that maintains high availability. Third, Rosé ensures fast recovery and full performance after failover by separating the replication of writes from their application to the backup partition's key-value store. We integrate Rosé with Chablis, a geo-distributed, multi-versioned transactional key-value store to preserve the benefit of fast single datacenter (DC) transactions while ensuring multi-DC durability.

1 INTRODUCTION

Replication is a ubiquitous technique used by database systems to improve durability and availability, as well as read latency by creating copies of the data geographically closer to the users. Asynchronous primary-backup replication is a widely used form of replication [3, 4, 8, 10, 13, 16] in which a designated primary replica-set executes and commits all transactions locally, and asynchronously sends the transaction writes to a set of backup replicas which then apply the writes to independently reconstruct the primary's state. Since writes need only be acknowledged by the primary (which is typically contained within a single zone or region), this technique achieves excellent write latency, but is prone to losing recent data if the primary replica fails (e.g. due to a datacenter or region-wide outage). The wide adoption of this technique in practice shows that many users and applications find that trade-off acceptable.

A desirable property in primary-backup replicated systems is *monotonic prefix consistency* [8], where each backup replica exposes a progressing sequence of the primary's recent states. This ensures that if failover has to happen, the backup replica that is promoted to become the new primary is going to be in a consistent state. In other words, while durability can be compromised during fail-over, all the other ACID properties would still be maintained, helping preserve application invariants. It also means the backups are able to serve consistent (albeit potentially stale) snapshots to read-only transactions.

Monotonic prefix consistency is relatively straightforward to provide in unpartitioned database systems with a single log. However, in partitioned systems each partition has its own log and is typically replicated independently. Thus, even if the replication of each partition preserves monotonic prefix consistency, the overall state of the backup can be inconsistent and undefined. Consider the following example to illustrate the issue: Suppose a committed transaction T wrote a key K_1 in partition P_1 and a key K_2 in partition P_2. It is possible, due to differences in replication pace, that the write for K_1 is replicated to the backup partition P_1 while the write for K_2 is not yet replicated to the backup partition P_2. In this case, an operation executing at the backup that reads K_1 and K_2 would only observe a part of T, breaking atomicity.

We know from experience and from speaking with many developer teams that dealing with fail-over during disaster recovery in such cases often requires performing complicated consistency checks and repairs on the database. Since the occurrence of such disasters is rare, regularly performing *drills* to exercise and validate the code and recovery processes is usually required. Fortunately, many modern systems that support global consistency across partitions also offer the ability to restore a cluster to a consistent point-in-time snapshot [2, 15] which simplifies this process. However, the process remains time consuming, and the systems offer no guarantees

 16th Annual Conference on Innovative Data Systems Research (CIDR '26). January 18-21, 2026, Chaminade, USA

on how far behind the consistent point-in-time snapshot on the asynchronously replicated backup can be. In addition, failover often causes degraded performance on the newly promoted the primary, due to the need to clean up data newer than the latest replicated snapshot.

Our work in this paper addresses the limitation of asynchronous primary backup replication in partitioned databases by designing a novel replication scheme that preserves the desirable properties of monotonic prefix consistency, while enabling a flexible, fast and efficient fail-over process, and bounds the replication lag to the backup as long as it is up. First, we observe that many distributed database systems already have support for globally consistent, serializable snapshot reads. Since these snapshot reads can span multiple partitions, the system must already have a notion of a global snapshot, which can be based on real time [5], hybrid logical clocks (HLC) [2, 15] or epochs [6, 7]. These snapshots can serve as a natural extension of monotonic prefix consistency to partitioned databases: the (externally visible) state of the backup should always be a globally consistent snapshot of the primary. Second, we propose a backpressure-based mechanism that effectively caps replication lag while maintaining high availability. Third, we tackle the core problem of degradation after failover by separating the replication of the write-ahead log (WAL) entries from their application in the partition's key-value store. WAL entries are replicated liberally but only applied in a coordinated fashion, up until the latest fully-replicated snapshot. Finally, we integrate Rosé with Chablis and evaluate several aspects of the protocol.

2 BACKGROUND AND RELATED WORK

In this section, we first provide a brief overview of replication techniques (§2.1) and how Rosé fits in this landscape. Then, we provide some minimal background of Chablis [6], in order to later describe its integration with Rosé (§2.2).

2.1 Replication

Distributed databases seek to support high-availability and durability, such that when some servers fail, data is not lost and the rest of the servers can continue providing the database functionality. These objectives are commonly satisfied using replication.

Consensus-based replication. Consensus protocols such as Paxos [9] or Raft [12] are commonly used in practice to replicate the database, such as in systems like Spanner [5], CockroachDB [15] and Yubabyte [2]. This approach is quorum-based, often using majority-based quorums in which data has to be replicated to a majority of replicas to be committed. With n replicas, the system achieves availability by tolerating up to n/2 - 1 replicas failing. However, it requires a relatively high number of replicas to achieve durability: to tolerate losing f replicas without permanent data loss, n must be 2f + 1.

Primary-backup replication. In primary-backup replication, a replica is designated the primary and is responsible for executing all client transactions. After all operations in a transaction execute, the transaction commits by writing to the primary's database and flushing a log of its changes to stable storage [8]. The primary then sends a copy of its log to (one or more) backup replicas, which apply it to their copy of the database to reach the same state as the primary (i.e. a form of state machine replication [14]). The log reflects a total order of the writes applied by the primary, determined by the primary's transaction commit order and the order of each transaction's operations. Primary-backup can be synchronous, if the primary waits for the replication to complete on the backup before acknowledging a transaction, or asynchronous otherwise.

Monotonic prefix consistency. In asynchronous primary-backup replication, it is inevitable that the backup's log lag behind the primary [8]. Helt et. al. [8] define monotonic prefix consistency of the backup's log relative relative to the primary's log of transactions as the following two invariants: First, the backup's state must reflect the changes of a contiguous prefix of transactions. Second, the sequence of states exposed to read-only transactions must reflect prefixes of monotonically increasing length.

Intuitively, this means that the backup should expose a progressing sequence of the primary's recent states. Many systems have this behavior [1, 3, 11, 17, 18], which has the desirable property that it maintains application invariants.

2.2 Chablis

Chablis is a scalable, geo-distributed, multi-versioned transactional key-value store that supports low-latency read-write transactions within a region and globally consistent, strictly-serializable, lock-free snapshot reads. Its architecture is shown in Figure 1 and consists of the following components:

- **RangeServer:** Each RangeServer is responsible for managing specific ranges or partitions of the key space and implements both 2-phase-locking and 2-phase-commit protocols while remaining mostly stateless. The RangeServer persists transaction data in a write-ahead log (offered by a separate WAL Service) through prepare, commit, and abort operations. In addition, it asynchronously applies these operations to a Key-Value Service, which fast read access, and occasionally trims the WAL. The WAL Service and Key-Value Service can either operate as separate services or be co-located on the same node depending on the deployment configuration.
- **Warden:** The Warden component is responsible for assigning ranges to individual RangeServers and continuously monitors the health of RangeServers through a heartbeat mechanism to ensure system reliability and proper load distribution.
- **Epochs:** Chablis achieves transaction serialization through the use of epochs, which provide a global ordering mechanism between transactions such that transactions in epoch e_i are guaranteed to have occurred before transactions in epoch e_j when $e_i < e_j$. While this epoch-based approach enables Chablis to deliver superior performance and strict serializability without requiring specialized hardware such as atomic clocks, it also means that global ordering is only well-defined at epoch boundaries. Transactions read the current epoch from regional epoch publishers during the 2-phase commit process, while a global epoch service periodically advances the epoch across all regions, thereby enabling fast regional writes and global strictly-serializable lock-free snapshot reads.

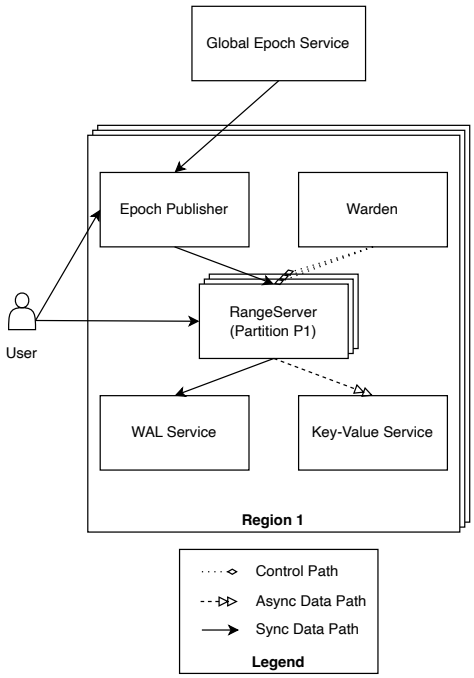


Figure 1: Chablis architecture.

3 CHALLENGES

Rosé addresses the following challenges arising from the problem of asynchronous replication in partitioned databases:

- **Bounding Replication lag**: Asynchronous replication acknowledges writes before remote replicas persist them, so lag is unavoidable; when that lag grows too large it exposes users to stale data and undermines failover.
- **Minimizing Time to Recovery**: In synchronous schemes, every replica already has a complete, identical log, enabling near-instant failover. With asynchronous replication, partitions may advance at different rates, which leads to accumulation of unusable data on failover. For example, if partitions p_1 and p_2 have replicated changes up to times t_1 and t_2 with $t_1 < t_2$, then a failover can rely only on state up to t_1 and discard all updates with $t > t_1$. Thus, uneven progress can inflate downtime and/or decrease performance after failover.

4 ROSÉ PROTOCOL

Rosé is an asynchronous primary-backup replication protocol for geo-distributed partitioned databases, shown in figure 2 It provides strong consistency guarantees (i.e. monotonic prefix consistency) at the secondary by integrating with a database's existing notions of time (§4.1). As with any asynchronous protocol, Rosé trades durability for increased performance during normal operations. However, Rosé takes a principled approach (§4.2 to cap replication lag, through a queue-based backpressure mechanism. This way, Rosé limits the maximum amount of data that could be lost in the event of an regional outage. Finally, Rosé tackles the problem of quick and performant failover by applying replicated entries in a coordinated fashion across the backup cluster, ensuring that all backup replicas make equal progress and don't accumulate potentially invalid data (§4.3).

4.1 Maintaining Monotonic Prefix Consistency

Many distributed database systems already provide globally consistent, serializable snapshot reads as a core feature. To enable cross-partition snapshot queries, these systems necessarily maintain a coherent notion of global snapshots, implemented through various ordering mechanisms including real-time ordering [5], hybrid logical clocks (HLC) [2, 15], or epoch-based approaches [6, 7]. This existing global snapshot infrastructure presents a natural foundation for extending monotonic prefix consistency to partitioned database environments: backup replicas should maintain externally visible state that corresponds to globally consistent snapshots of the primary system at all times. Since we are integrating Rosé with Chablis, we will use epochs as our measure of time. However, any of the aforementioned methods would work as well.

Based on this observation, we can construct a minimal asynchronous replication protocol that provides monotonic prefix consistency. Each primary partition sends transaction write-sets, in increasing order of epochs (WAL order), to the corresponding backup partition. Correspondingly, each backup partition receives committed transactions and applies them to reconstruct the primary's state. In the meantime, the component responsible for cluster management at the backup region tracks the latest fully applied epoch e_i at each partition P_i, which can be piggybacked on existing health checks. The most recent snapshot that can serve reads at the backup corresponds to the minimum epoch of all partitions, $e_{snapshot} = min(e_i)$. As the snapshot epoch $e_{snapshot}$ advances monotonically, the backup provides monotonic prefix consistency for reads.

4.2 Bounding the Replication Lag

4.2.1 Rosé's Mechanism. Replication lag, defined as the number of transactions that have not yet been replicated from the primary to the backup, represents a fundamental challenge in asynchronous replication systems. For example, a partition P_i where the primary operates at epoch $e_{primary,i}$ and the backup at epoch $e_{backup,i}$, the replication lag is $replication\_lag_i = e_{primary,i} - e_{backup,i}$. Excessive replication lag has proven to be a significant operational concern for asynchronous databases, causing both operational burden and system outages. In partitioned database environments, replication lag becomes particularly problematic because backup data remains useful only up to the snapshot epoch, as defined in §4.1. The effective replication lag across the system is thus $effective\_replication\_lag = \min(replication\_lag_i) = \min(e_{primary,i} - e_{backup,i})$. This formulation reveals a critical vulnerability: during failover scenarios, partitions that have progressed beyond the snapshot epoch contain data that becomes effectively useless and must be cleaned up before promotion to primary status.

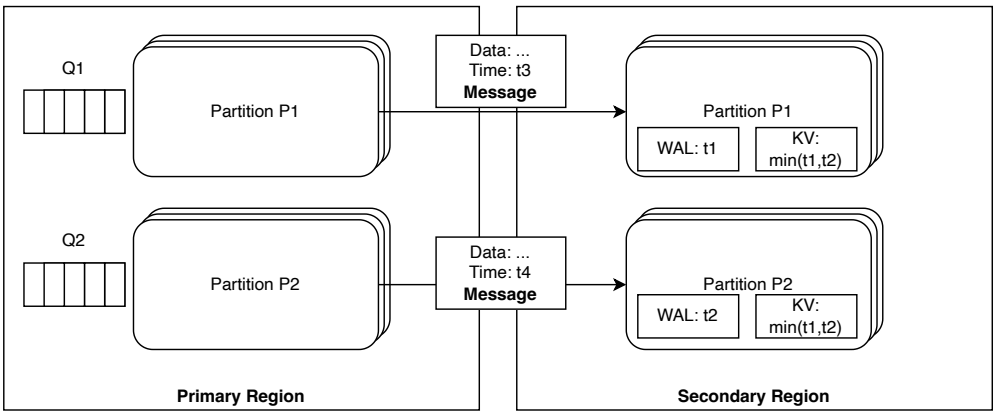


Figure 2: Rosé overview.

Consequently, the progress of all partitions becomes interdependent, as a single stalling partition can compromise the durability guarantees for the entire database.

Rosé proposes a backpressure-based mechanism that effectively caps replication lag in distributed partitioned databases, while maintaining availability. Rosé maintains a bounded queue of size L for each partition to track outstanding transactions, employing push-based replication where the primary actively propagates changes to backups while monitoring replication progress across all partitions. When any partition's queue reaches capacity, writes are throttled for that specific partition. This maintains a very desirable property: straggler partitions only affect themselves but at the same time avoid accumulating an ever-increasing backlog that keeps the backup constantly behind. Furthermore, this gives Rosé has an effective way to detect straggler partitions so it can take actions to mitigate them such as migrating them to faster or less loaded servers. We will show that Rosé provides enhanced availability during normal operation and at worst similar to synchronous availability under network outages.

4.2.2 Availability Proof. We will show that Rosé's availability is at worst as much as synchronous replication. Assume a distributed database with partitions $P_1, ...P_n$, queue sizes $B_1, ..., B_n$ and queue limit L under a primary-backup replication setup and transactions $T_1, T_2, ..., T_m$ issued in that order. First, we'll show that in the case of a single partition, the accepted transactions of Rosé are a strict superset of the accepted transactions for synchronous replication. Initially, any transaction accepted by synchronous replication must also be accepted by Rosé without accumulating backlog. Transaction that aren't accepted by the synchronous scheme are accepted by Rosé initially, while $B < L$. Once $B = L$ and a new transaction appears, there are two possible outcomes. If the transaction is rejected by the synchronous scheme, it is also rejected by Rosé. If it is accepted, it means the link to the secondary is up and a slot can be cleared in the queue, so that there is space for the new transaction. We account for this case by waiting for some timeout θ when the queue is full. Similarly, the same point holds for multiple partitions.

A key issue in this scheme is that replication lag can grow arbitrarily large if a partition is completely down. For example, a partition may accept a write at e_i and then go down for an arbitrarily large amount of time, preventing the snapshot epoch at the backup from advancing. Luckily, partitions are already replicated and highly available inside the region, so the odds of a partition being completely down for a long time is extremely unlikely. More specifically, assuming a standard replication factor $R = 3$ and considering that we care about read availability for replication,

$$\begin{aligned}\Pr[\text{single partition unavailable}] &= \Pr[\text{all replicas down}] \\ &= \Pr[\text{replica node down}]^R\end{aligned}$$

Assuming a 99.9% availability for a single node, any single replica will be available 99.9999999% of the time. Thus, the probability of a persisting stalled partition is practically infeasible.

4.2.3 Important Details. A robust and performant implementation is crucial to our backpressure design, so that writes aren't throttled just because the backup can't keep up. For this reason, we match the parallelism of the primary and backup using the C5 algorithm [8]. In addition, L is an important hyperparameter to tune, which will be different for each user. It should be high enough to utilize the full network bandwidth and also absorb normal load and temporary spikes, only triggering in genuine edge cases. At the same time, it should match the user's desired durability guarantees. Finally, in the event of a prolonged outage or loss of the backup region, we assume that an administrator or an external system monitoring uptime of the respective cloud will disengage the backpressure mechanism to restore availability if needed.

4.3 Minimizing Time to Recovery

As elaborate in §4.1, data at the secondary is only useful up to the snapshot epoch, which is the minimum replicated epoch across all partitions. This means that in the event of an outage, the backup cluster needs to restore every partition to that minimum epoch before it can be promoted to primary. So while a synchronously replicated backup would be immediately eligible for promotion,

failover in asynchronously replicated partitioned databases presents this significant challenge. A straightforward way of restoring the backup would be to delete all records that were written after the desired snapshot time, albeit at the cost of inflating time to recovery. To work around this limitation, databases like Yugabyte have designed complex recovery schemes that can start serving requests immediately, while deferring clean up to background operations. However, these schemes still suffer from degraded performance after failover, as we will explain below.

4.3.1 Yugabyte: An example from industry. Yugabyte is a distributed partitioned database that uses a modified version of RocksDB to store its data on each node. It supports consistent asynchronous primary-backup replication through xCluster replication, which is pull-based (i.e. the replica cluster pulls data from the primary cluster), unlike Rosé. It supports instant failover (at the coarse grained scope of the entire cluster) by deeply integrating with its underlying MVCC storage engine, RocksDB. RocksDB uses an LSM tree to organize key-timestamp-value data, as shown in figure 3. The LSM tree consists of many levels and each level contains several SST files, while each SST file contains a metadata block and several data blocks containing key-timestamp-value tuples. To support fast failover, Yugabyte extends RocksDB and stores an extra piece of information in each SST file's metadata block: the maximum timestamp contained in the file, shown as *max_ts*. To rewind the partition to an earlier timestamp, Yugabyte simply reads each file's metadata block to check if the maximum timestamp exceeds the desired snapshot timestamp. If it doesn't, the SST file is left as is. Otherwise, it records the desired snapshot timestamp in the file's metadata block, shown as *keep_ts*. The partition is now ready to serve requests. Writes work as usual, reads need to do extra work and potentially parse invalid entries, while compaction slowly cleans up the unused data. As we will see in the evaluation, even this elaborate scheme can result in degraded performance after failover.

4.3.2 Rosé Coordinated Apply. Rosé takes a different approach to address the fundamental issue of accumulating unusable data in asynchronous replication, by separating the *replication* of data from its *application* in the underlying MVCC store. We observe that most databases persist data in a similar fashion: by first making it durable to a fast write-ahead log (WAL) and periodically applying it to a structured key-value store for fast reads. Databases currently apply replicated records as soon as they arrive at the backup, incurring an expensive bulk-delete operation on failover or degrading their read performance. Instead, we notice that the WAL has a desirable property for this scenario. It keeps data in insertion order, thus enabling fast trimming of the log to a past offset.

Based on this observation, we propose coordinating the application of the data from the WAL to the key-value store, instead of doing it blindly. More specifically, the backup constantly keeps track of the current minimum replicated epoch across all partitions. Then, it notifies partitions to apply their WAL up to that epoch and no more. This way, on failover Rosé only needs to trim the WAL in order to clean up data, which is a very fast operation. At the same time, it preserves the full performance of the backup cluster after a failover.

A concern with this proposed approach is how it would affect replication under normal operating conditions. First of all,

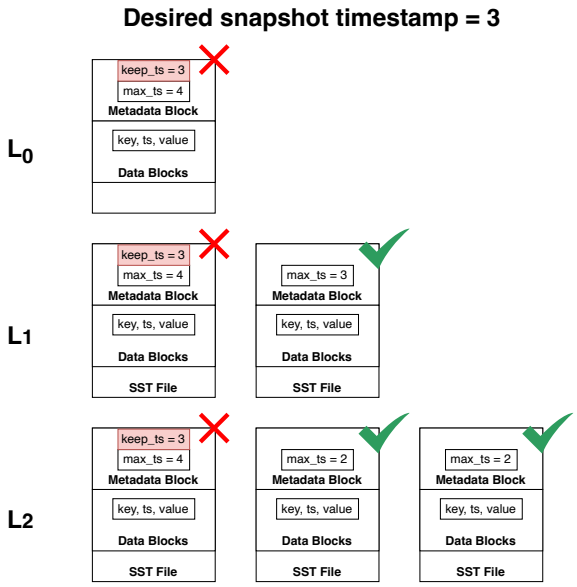


Figure 3: Yugabyte recovery of a single partition at t=3.

is our scheme optimizing for the worst case by sacrificing the usual case? For example, a single straggler secondary partition spells doom for this scheme, as it makes log application fall behind until it's resolved. Luckily, Rosé's mechanism for bounding replication lag mitigates these cases. If a partition straggles, writes to it will be throttled, allowing it to catch up. Second, all introducing spikes by advancing log application at epoch boundaries? Let's think about it through a two-partition example. Imagine we have two partitions, one hot, high write bandwidth w_{hot} and one cold, with low write bandwidth w_{cold}. The hot partition will replicate in $replication\_time_{hot} = RT_{hot} = \frac{w_{hot} * epoch\_duration}{network\_bw}$ seconds, while the cold will take $RT_{cold} = \frac{w_{cold} * epoch\_duration}{network\_bw}$ seconds. Thus, there will be an amount of "dead-time" for each partition, where log application could have started, but didn't, causing spikes at epoch boundaries. For the each partition i, this dead time is equal to:

$$\begin{aligned} \text{dead\_time}_i = DT_i &= \max(RT) - RT_i \\ &= \frac{(w_{\max} - w_i)\ \text{epoch\_duration}}{\text{network\_bw}} \end{aligned}$$

Minimizing this dead time is the key to minimizing the spikes. As we can see, epoch duration plays a big role. If the epochs are big, the dead time is big. In Rosé, epochs are configured to be a couple of milliseconds long, which should be sufficient to minimize spikes in the presence of hot partitions. In the absence of hot partitions, all the dead times should be roughly zero.

Finally, it should be noted that the snapshot epoch will advance the same in both scenarios. We illustrate this point in figure 4. While the second partition starts applying its log later under Rosé,

it should still finish before the first partition, which has a larger replication time and thus larger log.

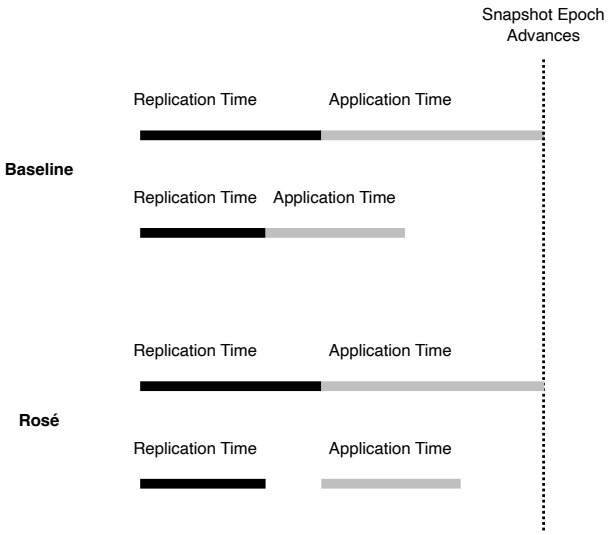


Figure 4: Log application and snapshot epoch advancement under Rosé.

5 EVALUATION

We evaluate Rosé's fast and performance recovery and ability to cap the replication lag. To do so, we integrate Rosé into Chablis [6], a geo-replicated, transactional key-value store.

In our evaluation, we aim to answer the following questions:

- **Q1:** Does Rosé help cap replication lag in asynchronous replication?
- **Q2:** Does Rosé improve performance after failover over existing techniques?

5.1 Setup

We simulate a multi-node setup, along with network faults, on a single Cloudlab c6525-25g machine. For experiments with Yugabyte, we used version 2.25.2.0-b359 and their manual instructions for failover [19].

5.2 Q1: Capping the replication lag

To show how backpressure allows Rosé to effectively cap replication lag, we track replication lag over time for a partition with and without backpressure enabled. As shown in figure 5, backpressure effectively caps the replication lag and allows the partition to keep up, at the cost of reduced performance. However, only the overloaded partition is affected.

5.3 Q2: Recovery with Rosé

To evaluate recovery, we run Chablis and Yugabyte in primary-backup configurations with two nodes in each region. We run uniform read-write transactions in the primary and then fail a link to one of the backup nodes. Yugabyte continues to apply writes in

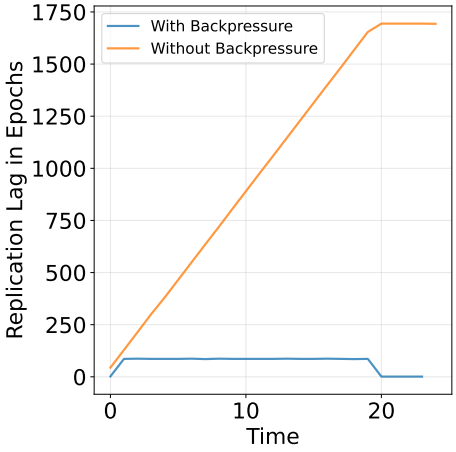
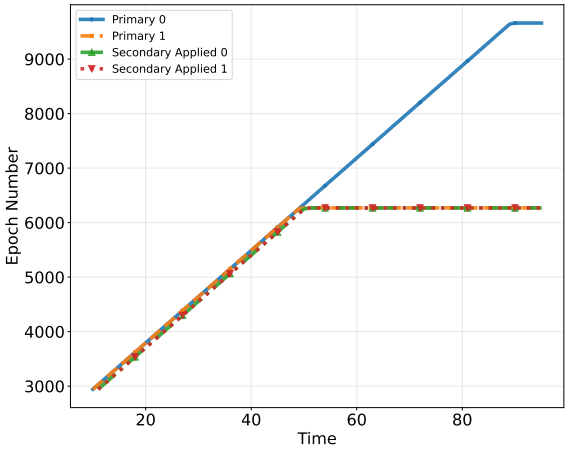


Figure 5: Rosé backpressure to cap replication lag.

the reachable node, while in Chablis, coordinated apply prevents that. In figure 6(a), we see coordinated apply in action. Up until time step 50, both backup nodes are reachable. In time step 50, one of the backup nodes becomes unreachable. Thanks to coordinated apply, the remaining node will not apply any changes after the last replicated epoch of the unavailable node.

Both databases can failover instantly, in under two seconds, but only Rosé can provide the same performance after failover, since its key-value store is clean. On the contrary, Yugabyte's performance for reads is degraded, with 22% less throughput and 15% higher P99 latency, as shown in figure 6(b).



(a) Coordinated apply in action.

Performance After Failover	Yugabyte	Rosé
Throughput Slowdown	22%	0%
P99 Latency Inflation	15%	0%

(b) Performance comparison after failover.

Figure 6: Coordinated apply and its impact on failover.

We do not compare absolute performance numbers of Rosé and Yugabyte, because that comparison would not be apples-to-apples, since Yugabyte is a production-grade system with full SQL support and many other features while Rosé is a key-value store prototype. Nonetheless, the relative performance of both systems in this experiment is instructive.

5.4 Conclusion

We presented Rosé, a novel replication protocol that addresses the typical shortcomings of asynchronous replication, while keeping many of the benefits. Thanks to Rosé, databases can provide low-latency regional writes while maintaining high durability and quick recovery in the face of an outage.

REFERENCES

[1] [n.d.]. PostgreSQL. https://www.postgresql.org/docs/18/index.html.

[2] [n.d.]. yugabyteDB. https://yugabyte.com/.

[3] Panagiotis Antonopoulos, Alex Budovski, Cristian Diaconu, Alejandro Hernandez Saenz, Jack Hu, Hanuma Kodavalla, Donald Kossmann, Sandeep Lingam, Umar Farooq Minhas, Naveen Prakash, Vijendra Purohit, Hugh Qu, Chaitanya Sreenivas Ravella, Krystyna Reisteter, Sheetal Shrotri, Dixin Tang, and Vikram Wakade. 2019. Socrates: The New SQL Server in the Cloud. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 1743–1756. https://doi.org/10.1145/3299869.3314047

[4] Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci, Jaidev Haridas, Chakravarthy Uddaraju, Hemal Khatri, Andrew Edwards, Vaman Bedekar, Shane Mainali, Rafay Abbasi, Arpit Agarwal, Mian Fahim ul Haq, Muhammad Ikram ul Haq, Deepali Bhardwaj, Sowmya Dayanand, Anitha Adusumilli, Marvin McNett, Sriram Sankaran, Kavitha Manivannan, and Leonidas Rigas. 2011. Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (Cascais, Portugal) *(SOSP '11)*. Association for Computing Machinery, New York, NY, USA, 143–157. https://doi.org/10.1145/2043556.2043571

[5] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. 2013. Spanner: Google's Globally Distributed Database. *ACM Trans. Comput. Syst.* 31, 3, Article 8 (aug 2013), 22 pages. https://doi.org/10.1145/2491245

[6] Tamer Eldeeb, Philip A. Bernstein, Asaf Cidon, and Junfeng Yang. 2024. Chablis: Fast and General Transactions in Geo-Distributed Systems. In *CIDR*.

[7] Tamer Eldeeb, Xincheng Xie, Philip A. Bernstein, Asaf Cidon, and Junfeng Yang. 2023. Chardonnay: Fast and General Datacenter Transactions for On-Disk Databases. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. USENIX Association, Boston, MA. https://www.usenix.org/conference/osdi23/presentation/eldeeb

[8] Jeffrey Helt, Abhinav Sharma, Daniel J. Abadi, Wyatt Lloyd, and Jose M. Faleiro. 2022. C5: cloned concurrency control that always keeps up. *Proc. VLDB Endow.* 16, 1 (sep 2022), 1–14. https://doi.org/10.14778/3561261.3561262

[9] Leslie Lamport. 1998. The Part-Time Parliament. *ACM Trans. Comput. Syst.* 16, 2 (may 1998), 133–169. https://doi.org/10.1145/279227.279229

[10] 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* (Monterey, California) *(SOSP '15)*. Association for Computing Machinery, New York, NY, USA, 295–310. https://doi.org/10.1145/2815400.2815426

[11] Yoshinori Matsunobu, Siying Dong, and Herman Lee. 2020. MyRocks: LSM-Tree Database Storage Engine Serving Facebook's Social Graph. *Proc. VLDB Endow.* 13, 12 (aug 2020), 3217–3230. https://doi.org/10.14778/3415478.3415546

[12] Diego Ongaro and John Ousterhout. 2014. In Search of an Understandable Consensus Algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. USENIX Association, Philadelphia, PA, 305–319. https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro

[13] Lin Qiao, Kapil Surlaker, Shirshanka Das, Tom Quiggle, Bob Schulman, Bhaskar Ghosh, Antony Curtis, Oliver Seeliger, Zhen Zhang, Aditya Auradar, Chris Beaver, Gregory Brandt, Mihir Gandhi, Kishore Gopalakrishna, Wai Ip, Swaroop Jgadish, Shi Lu, Alexander Pachev, Aditya Ramesh, Abraham Sebastian, Rupa Shanbhag, Subbu Subramaniam, Yun Sun, Sajid Topiwala, Cuong Tran, Jemiah Westerman, and David Zhang. 2013. On brewing fresh espresso: LinkedIn's distributed data serving platform *(SIGMOD '13)*. Association for Computing Machinery, New York, NY, USA, 1135–1146. https://doi.org/10.1145/2463676.2465298

[14] Fred B. Schneider. 1986. *The State Machine Approach: A Tutorial.* Technical Report. USA.

[15] Rebecca Taft, Irfan Sharif, Andrei Matei, Nathan VanBenschoten, Jordan Lewis, Tobias Grieger, Kai Niemi, Andy Woods, Anne Birzin, Raphael Poss, Paul Bardea, Amruta Ranade, Ben Darnell, Bram Gruneir, Justin Jaffray, Lucy Zhang, and Peter Mattis. 2020. CockroachDB: The Resilient Geo-Distributed SQL Database. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 1493–1509. https://doi.org/10.1145/3318464.3386134

[16] Alexandre Verbitski, Anurag Gupta, Debanjan Saha, Murali Brahmadesam, Kamal Gupta, Raman Mittal, Sailesh Krishnamurthy, Sandor Maurice, Tengiz Kharatishvili, and Xiaofeng Bao. 2017. Amazon Aurora: Design considerations for high throughput cloud-native relational databases. In *SIGMOD 2017*.

[17] Alexandre Verbitski, Anurag Gupta, Debanjan Saha, James Corey, Kamal K. Gupta, Murali Brahmadesam, Raman Mittal, Sailesh Krishnamurthy, Sandor Maurice, Tengiz Kharatishvili, and Xiaofeng Bao. 2018. Amazon Aurora: On Avoiding Distributed Consensus for I/Os, Commits, and Membership Changes. *Proceedings of the 2018 International Conference on Management of Data* (2018).

[18] Tianzheng Wang, Ryan Johnson, and Ippokratis Pandis. 2017. Query fresh: log shipping on steroids. *Proc. VLDB Endow.* 11, 4 (Dec. 2017), 406–419. https://doi.org/10.1145/3186728.3164137

[19] Yugabyte, Inc. 2025. *Transactional xCluster Replication Setup (Manual).* https://docs.yugabyte.com/preview/deploy/multi-dc/async-replication/async-transactional-setup-manual/ "Set up transactional xCluster replication (Manual mode)".