# Global Views on Partially Geo-Replicated Data

André Rijo
NOVA LINCS, FCT,
Universidade NOVA de Lisboa

Carla Ferreira
NOVA LINCS, FCT,
Universidade NOVA de Lisboa

Nuno Preguiça
NOVA LINCS, FCT,
Universidade NOVA de Lisboa

## ABSTRACT

bla
bla
bla
bla
bla
bla
bla
bla

**PVLDB Reference Format:**
. . *PVLDB*, 12(xxx): xxxx-yyyy, 2020.
DOI: https://doi.org/TBD

## 1. INTRODUCTION

The increasing reliance on web service in many domains of activity, from e-commerce to business applications and entertainment, leads to stringent requirements regarding latency, availability and fault tolerance [7, 4]. To address these requirements, cloud platforms have been adding new data centers at different geographic locations. By allowing users to access a service by contacting the closest data center, a global service can provide low latency to users spread across the globe. The increasing number of data centers also contributes for proving high availability and fault tolerance, by allowing a user to access the service by accessing any available data center.

The database is a key component of any web service, storing the service's data. For supporting global services running at multiple geographic locations, it is necessary to rely on a geo-replicated database [3], which maintains replicas of the data at the data centers where the service is running. A number of geo-replicated database have been proposed, providing different consistency semantics. Databases that provide strong consistency [2, **?**, 5] intend to give the illusion that a single replica exists, requiring coordination among multiple replicas for executing (update) operations. This leads to high latency and may compromise availability int he presence of network partitions. Database that provide weak consistency [8, 3, 6] allow any replica to process a client request, leading to lower latency and high availability. As a consequence, these databases expose temporary state divergence to clients, making it more difficult to program a system.

In either case, geo-replicated databases typically rely on a full replication model, where each data center replicates the full database, with data being sharded across multiple partitions in each data center. As both the data managed by these systems increases in size and the number of data centers increases, this approach leads to a number of problems. First, storing all data in all data centers imposes a large overhead in terms of storage. Furthermore, storing some data in all data centers may be unnecessary, as data is only needed at some geographic locations. Second, increasing the number of data centers makes the replications process more complex and costly, as each update needs to be propagated to all other data centers.

For addressing these problems, partial replication is an attractive approach, with each data center replicating only a subset of the data. A number of works have been addressing the challenges of partial replication, for example by proposing algorithms to manage partially replicated data [**?**, **?**, **?**] and to decide which data is replicated in which replica [].

In this paper we address the problem of querying data in a weakly consistent partially geo-replicated database, focusing on recurrent queries for which a programmer would want to generate a (materialized) view. For example, consider an e-commerce system with users from multiple geographic locations. In this case, the data pertaining users of a given location do not need to be replicated in all data centers (but only in a few for fault tolerance). The same applies to other information, such as data on orders and warehouses. Other data, such as information on products would be replicated in the regions where the product is available. Under this data placement, obtaining the list of best seller products is challenging, as it requires accessing data that is located at multiple data centers.

Several possible solutions exist for this problem. First, it is possible to have a data center that replicates all data, and forward these queries to such data center. Doing this imposes a latency penalty and requires a data center to host all data and execute all queries of this type. Second, it is possible to execute the query by accessing multiple locations, by using, for example, a distributed processing system with support for geo-partitioned data [**?**, **?**]. This approach requires running an additional external service and poses challenges for the consistency of the results returned and the data observed by users.

We propose a different approach: to maintain materialized views, as commonly available in relational databases. Implementing such feature efficiently in a partially geo-replicated database requires addressing two main challenge challenges. First, it is necessary to guarantee consistency between the base data available in a replica and the relevant materialized views. To achieve this, we designed


*Proceedings of the VLDB Endowment,* Vol. 12, No. xxx
ISSN 2150-8097.
DOI: https://doi.org/TBD

a replication mechanism where updates to the base data and views are made visible atomically in each replica.

Second, it is necessary to efficiently support views with limits, used for example to support *top-k* queries. To achieve this, we build on the concept of non-uniform replication [1], in which the state of different replicas may be different, given that the observable state is (eventually) the same. This allows each replica to propagate only the updates that might be relevant to the observable state. Providing support for views required us to extend non-uniform replication from simple data types to more complex structures that could support a view with multiple columns. **nuno:** can we support updates to the views? why not?

We present the design and implementation of PotionDB, a geo-replicated key-value store with support for partial replication and materialized views. PotionDB provides weak consistency, for improved latency and availability, and support for highly available transactions [**?**]. To our knowledge, our work is the first to address the problem of maintaining materialized views in such setting.

We have evaluated our system using micro-benchmarks and TPC-H queries [] **nuno:** TPC-H, certo?. The results show that our algorithms for maintaining materialized views impose low overhead when executing and asynchronously replicating transactions, particularly for views with limits. **nuno:** deviamos ter uns micro-benchmarks que comparassem o overhead com limites e sem limites Additionally, the results show that executing queries by relying on the materialized views is much more efficient than using alternative mechanisms. Furthermore, our algorithms for maintaining materialized views in a decentralized way perform better that alternative approaches where the view is computed in a single data center, while being able to keep consistency between the base and view data in every replica.

In this paper we make the following contributions:

- the design of a geo-replicated key-value store with support for partial replication and views over partially replicated data;
- replication algorithms for efficiently maintaining consistent materialized views over partially replicated data;
- an implementation and evaluation of the proposed approach with micro-benchmarks and TPC-H.

The remainder of the paper is organized as follows. Section ...

## 2. SYSTEM OVERVIEW

- System model
  - Replicação parcial
- System API
  - ”Create table”
  - ”Create view”
    - CRDT não uniforme
    - put numa table $\Longrightarrow$ puts nas várias views
      - consistência das views face aos dados - in sync
- System description
  - CRDT não uniforme
  - Implementação de queries?

## 3. IMPLEMENTATION

## 4. EVALUATION

## 5. RELATED WORK

## 6. CONCLUSIONS

## 7. REFERENCES


[1] G. Cabrita and N. Preguiça. Non-uniform Replication. In *Proceedings of the 11th International Conference on Principles of Distributed Systems (OPODIS 2017)*, 2017.

[2] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford. Spanner: Google's Globally-distributed Database. In *Proceedings 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 251–264, Hollywood, USA, 2012. USENIX Association.

[3] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's Highly Available Key-value Store. In *Proceedings of the 21st ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 205–220, Stevenson, Washington, USA, 2007. ACM.

[4] Gomez. Why web performance matters: Is your site driving customers away?, 2018. Accessed May/2018.

[5] T. Kraska, G. Pang, M. J. Franklin, S. Madden, and A. Fekete. MDCC: Multi-data Center Consistency. In *Proceedings 8th ACM European Conference on Computer Systems*, EuroSys '13, pages 113–126, Prague, Czech Republic, 2013. ACM.

[6] 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 *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*, SOSP '11, pages 401–416, Cascais, Portugal, 2011. ACM.

[7] E. Schurman and J. Brutlag. Performance related changes and their user impact. Presented at velocity web performance and operations conference. `http://slideplayer.com/slide/1402419/`, 2009.

[8] W. Vogels. Eventually consistent. *Communications of the ACM*, 52(1):40–44, Jan. 2009.


## 8. SYSTEM OVERVIEW

Possiveis pontos mais detalhados?

## 8.1 System model

- Network assumptions
- Client-server interaction (refer key-value store interface? Maybe refer this instead in System API?)
- Server-server interaction? (is it needed? We'll already touch this in Replication.)
- System guarantees
  - CRDTs
  - Consistency level
- Replication
- Async
- Op-based
- Maintains consistency, i.e., transaction level based.
- Partial (system admin defined, each server only has a subset of the data based on topics. Potencially some data can be replicated everywhere)

## 8.2 System API

- Basically how can we translate a problem to sql-like operations
- Create table
- Create view
- Updates (incluir problema de consistência de views/dados)
- Queries (incluir aqui problema de os CRDTs não uniformes precisarem de mais dados? Ou na zona da view?)

## 8.3 System description

- Structure? Maybe that's for implementation? How much detail?
  - Internal partitioning vs external partitioning? Capaz de não ser boa ideia...
- CRDTs and non-uniform CRDTs?

# 9. IMPLEMENTATION

- Go
- Transactions (TM/Mat?)
- Replication (RabbitMQ and other stuff?)
- Communication (protobufs. Also worth noticing the compability with existing AntidoteDB clients)
- CRDTs (version management at least)