Towards Local-First Distributed Property Graphs

Ayush Pandey
Sorbonne Université (LIP6)
France
ayush.pandey@lip6.fr

Stefania Dumbrava
ENSIIE & Télécom SudParis
France
stefania.dumbrava@ensiie.fr

Marc Shapiro
Sorbonne Université (LIP6) & Inria
France
marc.shapiro@acm.org

Carla Ferreira
NOVA University Lisbon
Portugal
carla.ferreira@fct.unl.pt

Mário Pereira
NOVA University Lisbon
Portugal
mjp.pereira@fct.unl.pt

Nuno Preguiça
NOVA University Lisbon
Portugal
nuno.preguica@fct.unl.pt

Abstract

A graph database system (GDBMS) is designed to efficiently manage highly interconnected and diverse data. Compared to a traditional database system, a GDBMS has more stringent requirements due to long running transactions which also have a large footprint as they access a large number of vertices, as well as specific constraints and invariants related to graphs, such as key and connectivity constraints. Hence, existing sharding and georeplication techniques for scalability and availability are not well-suited for GDBMSes. Strong isolation levels suffer from a high probability of spurious conflict and unavailability, while techniques for highly available isolation, such as Replicated Data Types (RDTs), are well studied for traditional key-value stores but largely unexplored for GDBMSes. We analyze the interplay between availability, consistency, and constraints and propose studying RDTs and consistency levels tailored for GDBMSes.

CCS Concepts: • **Information systems → Parallel and distributed DBMSs**; **Graph-based database models**.

Keywords: local-first, graph databases, consistency

ACM Reference Format:
Ayush Pandey, Stefania Dumbrava, Marc Shapiro, Carla Ferreira, Mário Pereira, and Nuno Preguiça. 2025. Towards Local-First Distributed Property Graphs. In *12th Workshop on Principles and Practice of Consistency for Distributed Data (PaPoC '25), March 31, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3721473.3722139

PaPoC '25, March 30-April 3 2025, Rotterdam, Netherlands

ACM ISBN 979-8-4007-1558-7/2025/03
https://doi.org/10.1145/3721473.3722139

1 Introduction

Graphs have emerged as a cornerstone for managing and querying complex, interconnected datasets across domains such as social networks, knowledge graphs, and recommendation systems. During the pandemic, initiatives like EU Datathon [41], COVID-19 Biomedical Knowledge Miner [13], CovidGraph [16], *collaboratively* curated graph datasets that integrate multi-omics data from all over the world for genomic analyses and contact tracing. Graph data is also collaboratively produced in other critical areas, as finance [38], telecommunications [21], journalism [20], transportation [29], challenging traditional RDBMSes. Often, it holds sensitive information and requires retaining ownership and control.

Local-first systems [22] prioritize local autonomy, performance, and usability in environments where network connectivity is unreliable or inconsistent. However, despite their potential, local-first graph databases remain underexplored in academic research and industry practice. This paper unites the concept of local-first computing, a paradigm that prioritizes offline-first capabilities with graph models characterized by connected data. Local-first graph systems aim to address the critical trade-offs between consistency, performance, and usability by enabling users to work locally and synchronize seamlessly across peers. We present a vision for how such systems can transform applications in fields ranging from collaborative knowledge management to edge computing in smart cities. We also outline the technical challenges and research directions necessary to realize this.

Scalable and efficient processing of large graphs while also providing desirable properties, such as availability under network partition, fast response time, and consistency, is challenging [35]. The CAP theorem shows that a system prone to partitions cannot guarantee strong consistency and availability [14] at the same time. Similarly, a high network latency implies either a slow response (under strong consistency) or weaker guarantees (under availability). This trade-off is inherent: a geo-distributed application that requires strong consistency (e.g., in security-sensitive areas) must accept the resulting slow response time; one that requires availability and fast response (e.g., in a local-first setting [22]) must make do with less stringent consistency guarantees.

2 Background

Collaboratively curated graphs are widely used in various fields such as healthcare [30, 40], scientific repositories [18], enterprise data management [34], financial fraud information [12], and trend/pattern analysis [31]. These are typically large; for instance, the COVID-19 Knowledge Graph has more than 36M vertices and 60M edges and can be sensitive, e.g., containing personal medical data, requiring data ownership for privacy. At a foundational level, the most expressive graph data model is the label property graph model [2] (PG), i.e., a multi-labeled directed multi-graph with key-value properties on nodes and edges.

We illustrate the principles of a local-first architecture in a use case modeled on the FMKe benchmark [39] based on an actual application: the Danish National Joint Medicine Card. FMKe involves handling healthcare data, including sensitive information such as patient social security numbers (SSN), the information of their designated family doctor, prescriptions, and treatments they undergo. This type of information requires special care to ensure security and privacy.

In edge environments, healthcare institutions can deploy such applications on edge nodes, enabling local transaction processing. Operations such as delivering prescriptions, scheduling appointments, and managing lab reports are often performed on-premises and then broadcast to other replicas. Often, some operations need to be completed even when network partitions occur. For example, urgent healthcare cannot be denied when remote nodes are unreachable. In such scenarios, the operations performed are synchronized with remote replicas after the fact to ensure that the global application constraints are not violated.

3 PGs for Local-First Systems

While local-first systems offer significant benefits in terms of responsiveness and offline functionality, they introduce several unique challenges when applied to graph databases. A key difficulty is the inherently interconnected nature of graph data, where relationships between vertices are as meaningful as the vertices themselves. This complexity complicates the synchronization and conflict resolution processes, particularly when multiple replicas make concurrent updates in a distributed environment. Ensuring eventual consistency across highly interconnected graphs while retaining performance remains a significant technical challenge.

Graph databases are evolving beyond schema-less designs, with more advanced applications increasingly requiring explicit schemas and constraints to ensure data integrity, validate relationships, and maintain the correct structure of the graph. In centralized systems, schemas and constraints can be easily enforced globally. However, in local-first environments, where updates happen independently, and synchronizations occur asynchronously, ensuring that all vertices adhere to the same schema and constraints becomes complex. Conflicting updates may violate integrity constraints, requiring sophisticated mechanisms to reconcile these conflicts without undermining the structure or validity of a graph.

Expressing and enforcing schemas and constraints is particularly important for applications with strict data governance, such as financial services, healthcare, and enterprise systems. In these domains, maintaining data integrity is non-negotiable, and violations of schema or business rules can have significant repercussions. In local-first systems, this necessitates the development of new techniques for enforcing schema and constraint consistency during synchronization while still allowing local nodes to operate autonomously. Without such mechanisms, local-first graph databases risk introducing inconsistencies that could degrade the reliability and trustworthiness of the data over time.

To our knowledge, local-first property graphs are an unexplored area. Two challenges need to be addressed: graph-invariant specification and graph-structure synchronization.

Graph Invariant Specification. The recent PG-Schema language allows one to express typing (PG-Types), constraints for keys (PG-Key), and cardinality restrictions on portions of the graph. We illustrate a simplified PG-Schema for the property graph database instances in Figure 1.

```
CREATE GRAPH TYPE healthcareGraphType STRICT {
(doctorType : Doctor {name STRING, OPTIONAL kind STRING}),
(patientType : Patient {SSN INT32}),
(labType : Lab {kind STRING}),
(dptType : Department {size ("small"|"medium"|"large")}),
(:doctorType)-[:works {yr:DATE}]->(:dptType),
(:doctorType)-[:manages]->(:dptType),
(:doctorType)-[:refers]->(:doctorType),
(:doctorType)-[:endorses]->(:labType),
(:patientType)-[:visits]->(:doctorType|labType)
FOR (p:Patient) EXCLUSIVE MANDATORY SINGLETON p.SSN, ...}
```

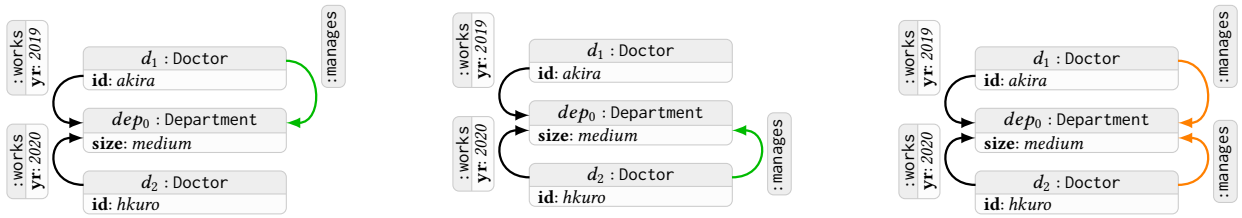
Constraints. With graphs replicated in multiple nodes, several violations can occur due to disconnected local operations on the data. We discuss the anomalies caused by key and cardinality constraint violations.

1. EXCLUSIVE constraints two objects in the key scope from sharing a key value. Consider the constraint: *if a department has a manager of type doctor, it is unique.*

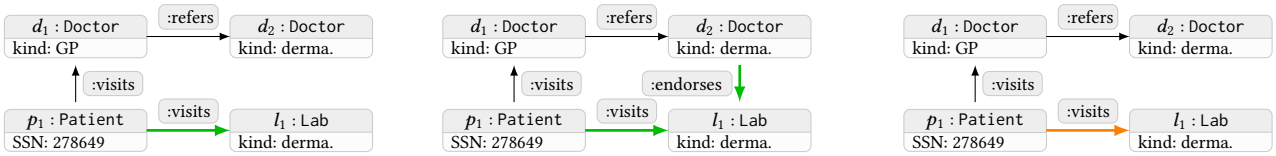
```
FOR (d:Department) EXCLUSIVE e
WITHIN (e:Doctor)-[:manages]->(d)
```

If replicas R1 and R2 add doctor d_1 as manager of dep_0; without a defined strategy, the merged state violates this constraint, given that two edges have the `:manages` label.

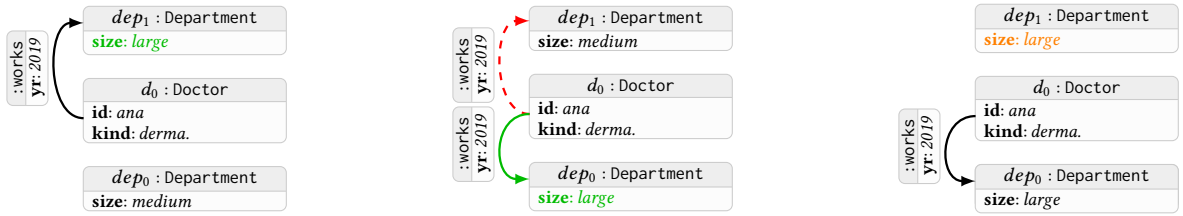
2. MANDATORY constraints enforce that every object must have a key, imposing a *total reference scheme*, i.e., every graph object should be referred to by at least one key. Consider the following constraint: *a doctor works in a department when a* `:works` *edge exists between the department and doctor vertices.*



(a) State of replica R1 (left), replica R2 (center), and merged state with SINGLETON constraint violated (right)



(b) State of replica R1 (left), replica R2 (center), and merged state with path pattern violation (right)



(c) State of replica R1 (left), replica R2 (center), and merged state with cardinality constraint violation (right)

Figure 1. A property graph RDT with concurrent updates: edge insertions (green), node deletion (red), conflicts (orange)

```
FOR (doc:Doctor) MANDATORY e
WITHIN (doc)-[:works]->(d:Department)
```

If a department dep_1 is deleted on replica R1, while an edge is concurrently added to dep_1 in R2. The merged state may contain a dangling edge leading to a MANDATORY violation.

3. SINGLETON constraints enforce that every graph can have at most one key within a scope. Ensuring that if two objects have the same key, then they are identical. Consider the constraint: *a department has a unique department head.*

```
FOR (d:Department) SINGLETON m
WITHIN (:Doctor)-[m:manages]->(d)
```

This translates to a node of type `Department` having a unique `:manages` edge from a node of type `Doctor`. In Figure 1a, a doctor is assigned as the department head. Replicas R1 and R2 concurrently assign different doctors as heads and are locally correct. However, the constraint is violated when merging remote updates without conflict resolution.

4. IDENTIFIER constraints combine EXCLUSIVE, MANDATORY and SINGLETON. Consider the corresponding PG-Key path constraint: *a patient's visit to a lab is valid only if the lab is endorsed by a doctor of the same specialty.* The visit edge e is defined in the scope of the key constraint (first WITHIN clause), and the endorses edge is specified in the descriptor part (second WITHIN clause), as part of a path.

```
FOR e WITHIN (:Person)-[e:visits]->(lb:Lab)
IDENTIFIER p, e WITHIN
(d:Doctor)-[p:endorses]->(lb:Lab)<-[e:visits]-(:Person)
WHERE d.kind = lb.kind
```

In Figure 1b, R1 adds an edge denoting a visit by the patient to the lab l_1. This update is broadcast and observed by all replicas. Currently, the doctor endorses lab l_1 in replica R2, but a missing update causes a violation of the path pattern in the merged state.

5. Cardinality constraints enforce bounds on property values based on the graph topology. Consider the department size constraint that *a department with the property size* `:large` *has at least 10* `:works` *edges from doctors*, enforced in Figure 1c. Replica R1 updates the value of the property `size` of department dep_1 to 'large', based on the number of doctors, while Replica R2 moves a doctor from dep_1 to dep_0 and changes the value of the property `size` of dep_0 to 'large'. Locally, replicas observe correct states and do not violate the constraint; upon merging, dep_1 is still a 'large' department, even though a doctor has been removed from it.

Graph Structure Synchronization. In a local-first local graph database, maintaining consistency across all nodes is essential for preserving data integrity. When multiple nodes

are involved, synchronization protocols ensure that updates are propagated correctly and conflicts from concurrent modifications are resolved. Given the complex interconnections between vertices and edges, these protocols are especially critical for graph data. Unlike linear data models, where updates to individual records are straightforward, graph structures require careful handling due to their intricate relationships. For example, conflicting updates can occur if different users make simultaneous changes to various parts of a graph. These conflicts are not always easy to resolve, as path modifications may lead to inconsistent graph states, such as creating duplicate or missing edges. Ensuring these changes do not violate the graph's integrity demands careful coordination. Moreover, certain operations, such as updating multiple edges or vertices together, must be made atomically visible to maintain consistency. This becomes particularly challenging in local-first systems, where decentralized architecture complicates the implementation of distributed transactions and makes it challenging to ensure that all updates are applied consistently across nodes.

In local-first systems, achieving atomicity with partial replication is problematic since updates may be incomplete or only partially applied. Furthermore, large graphs present additional hurdles with their dense network of interconnected elements. Efficient communication protocols are necessary to synchronize the graph across multiple nodes while minimizing data transfer. The challenge is propagating only the portions of the graph that have changed, avoiding transmitting redundant information. At the same time, the system must preserve the integrity of the graph, ensuring that no part of it is left in an inconsistent or invalid state.

4 Local-First Connectivity for PGs

Local-first systems are built with an offline-first approach, where data is stored locally on the user's device and synchronized with other devices when connectivity is available. This approach offers several advantages: improved responsiveness, offline access, and reduced network dependency. To achieve this, a system needs a few key features. First, it must support local data storage and processing, enabling users to interact with the application even when offline. This allows users to perform operations like querying, modifying vertices and edges, or adding new properties without requiring immediate synchronization with a remote database. Technologies such as SQLite, IndexedDB, or other local databases offer local storage and querying solutions, but cannot handle graph data without an expensive translation layer. This makes existing client-side RDBMSes or NoSQL databases unsuitable for local-first graph databases.

Synchronization is a critical feature of local-first systems. After a period of disconnection, a client device must propagate local changes to a central server or peers and merge updates from others to maintain data consistency. This can be achieved through various synchronization mechanisms, such as operational transform, state-based synchronization, or delta-based synchronization. These methods track changes in the graph's structure and properties, ensuring these updates are merged after reconnecting.

In an ideal distributed system, concurrent operations on a shared graph would never conflict. However, achieving this in practice is fundamentally constrained: it requires a central authority to enforce a global order of operations or restrictions on the types of permissible graph modifications. Centralized coordination undermines the scalability and fault tolerance benefits of distributed systems, while overly restrictive operation rules limit the graph's utility. Consequently, in decentralized environments, conflicts are unavoidable. Consider a scenario where two users, Alice and Bob, work on copies of the same graph. Alice operates in a remote, intermittently connected edge environment, while Bob interacts with an on-premise instance. Suppose Alice adds an edge between vertices V1 and V2, while Bob, unaware of Alice's pending update, deletes V1 entirely. When Alice reconnects, the system must reconcile these conflicting actions to ensure both users converge on a consistent graph state. Without a deterministic resolution policy, divergent outcomes could emerge, violating consistency guarantees.

To address this, conflict resolution strategies must prioritize clarity and predictability. In addition, when considering property graphs, one could imagine defining such conflict resolution policies at different levels of granularity, e.g., at the level of nodes, edges, or the properties thereof, leading to the need to study their interplay. An example of such a policy is the Last-Writer-Wins (LWW) approach, which relies on timestamps to prioritize the most recent update. While LWW simplifies ordering in systems with loosely synchronized clocks, it risks data loss in concurrent scenarios.

Alternative strategies, such as Add-Wins or Remove-Wins, prioritize operation semantics over timestamps. In Remove-Wins, Bob's deletion of V1 would invalidate Alice's edge addition, as the vertex no longer exists. Conversely, Add-Wins might preserve Alice's edge by marking the deleted vertex with a tombstone and allowing the edge to persist in a degraded state. However, neither approach is universally optimal. Remove-Wins risks undoing valid work, while Add-Wins could leave the graph in an inconsistent state (e.g., edges referencing nonexistent vertices).

Advanced systems may employ operational transformation (OT) or Conflict-Free Replicated Data Types (CRDTs) to enable semantically rich merges. For example, a CRDT-based graph could track dependencies using metadata that captures the causality of operations. If Alice's edge addition is causally independent of Bob's deletion, the system might flag the conflict for manual resolution or apply domain-specific heuristics (e.g., preserving the edge only if both vertices still exist post-merge). Such strategies demand greater complexity but align outcomes with user intent. Ultimately, the choice of

EC: *Eventual Consistency*, **TCC**: *Transactional Causal Consistency*, **SI**: *Snapshot Isolation*, **SSER**: *Strict Serializability*

Figure 2. Graph database operation and constraint *scopes* vs. the weakest *consistency* level required to enforce them.

policy depends on the system's consistency requirements, tolerance for data loss, and the graph's role in broader workflows. A hybrid approach, as prescribed by [36], involves combining automated resolution for common cases with escalation mechanisms for ambiguities to strike a balance between usability and correctness.

5 Research Directions

Given that local-first systems are designed to operate in disconnected or intermittently connected environments—and the challenges thereof—we outline key research directions.

Research Challenge 1: Finding the right consistency level to support graph database requirements.

Local-first connectivity indicates an environment with high availability with the possibility of disconnections. In such environments, choosing an appropriate consistency level is crucial to ensure that the graph database can operate effectively. One way to guarantee correctness is to utilize methods to systematically detect anomalies, which are based heavily on an exhaustive correctness analysis of applications [9]. Approaches such as CISE [15], CALM [1], or Hamsaz [19] propose a static analysis. These approaches are not readily available to ordinary programmers. Alongside, such analyses cannot be performed when the queries are not defined a priori. The LDBC SNB and BI workloads show examples of queries for which the execution plan changes as the underlying graph data evolves. In lieu of theoretically verified correctness, database developers rely on well-known consistency models. Most consistency levels can be characterized by the ordering guarantees of operations. We list popular consistency levels and how they affect graph operations and invariants, based on the work of Bailis et al. [4]. The two extremes of consistency are eventual consistency (EC) and strict serializability (SSER). EC assumes no operation ordering and is hence insufficient for GDBMSes.

SSER *totally* orders all operations across all geo-distributed replicas and prevents anomalies, ensuring that every replica executes operations in the same order and does not miss remote updates. A system that guarantees total visibility of operations between replicas foregoes availability [8]. For local-first graph databases, this is too big a sacrifice. Some invariants, such as global cardinality, are too strong to be maintained without sacrificing availability. For example, "*the number of edges between two vertices should be less than 3*" is especially difficult to maintain with concurrent operations without all replicas agreeing on a global execution order.

Without a global total order, consistency of schema updates and cardinality constraint preservation is not guaranteed.

Transactional causal consistency (TCC) eliminates some anomalies by ordering causally related operations but allows replicas to disagree on the order of concurrent operations, achieving a global partial operation order [24]. This makes TCC the strongest level of consistency that is always available [3]. It should, by definition, allow an application to maintain PG-key constraints in a local-first application as presented in Figure 2. Concurrent operations that cannot be ordered are resolved by assigning them priority, e.g., between *addEdge* and *deleteNode*, one could give priority to *addEdge*, ignoring a concurrent *deleteNode* operation.

However, after two replicas of a graph have diverged significantly, merging operations from the two to reach a consistent state requires solving graph isomorphism which remains quasi-polynomial at best [17]. For graphs with millions of vertices and edges, this can become very expensive very quickly. Alternatively, one could maintain parallel snapshots with multiple temporal versions of vertices and edges in the same graph instance. This further increases the already high cost of distributed pattern matching queries [7].

There is a need to comprehensively study the consistency levels that facilitate common graph operations in a local-first setting. A solution might involve dynamically switching consistency models such that strong constraints like schema and cardinality operate under strong consistency models while trivial operations like vertex updates and reads can occur under weaker consistency levels.

Research Challenge 2: Designing graph databases that can operate in a disconnected environment.

Most commercial GDBMSs are designed to operate in a connected environment. Both native graph databases, like Neo4j, and non-native ones, like CosmosDB, rely on a central server for coordination, making them unfit for local-first connectivity. As shown in Table 1, native graph databases are

mainly fully replicated and sacrifice availability for consistency, adopting a leader-follower approach. AllegroGraph allows multi-leader reads/writes and improves availability, but does not guarantee consistency. Non-native graph databases, such as JanusGraph, use an external storage layer, e.g., HBase, Cassandra, ScyllaDB, for availability and consistency [26]. Others, e.g., CosmosDB, DB2 Graph, and Oracle Graph, rely on the guarantees of their underlying relational database.

In a local-first environment, operations performed on the client side are typically agnostic to the server-side capabilities. With the bulk of computation being local and remote collaboration being an afterthought, at the implementation level, more robust and varied synchronization methods are needed. For example, clients working in close vicinity should be able to synchronize via a local communication channel, even if their states cannot be synchronized with remote clients. This precludes the assumption that a globally consistent state can always be achieved via some form of synchronization that guarantees a total order of operations. There is a need to study how to support peer-to-peer synchronization between replicas of PG-RDTs-based graph databases, allowing collaboration among clients without resorting to central infrastructure. For example, two clients on a stable, fast, local network should be able to synchronize their states without connecting to the data-center in the cloud.

Database	Txn Model	Consistency	Scalability
Native Graph Databases			
Neo4J	ACID	TCC	FR, LF
TigerGraph	ACID	SSER	FR, Consensus
AllegroGraph	ACID	None	FR, Multi-leader
OrientDB	ACID	SI	FR, Quorum
Non-native Graph Databases			
JanusGraph	Non ACID	None	Partitioned
ArangoDB	Non ACID	TCC	Partitioned, FR, LF
CosmosDB	ACID	EC - SSER	Partitioned, FR, LF
DB2 Graph	ACID	EC - SSER	Provided by DB2
Oracle Graph	ACID	EC - SSER	Provided by Oracle DB

EC: *Eventual Consistency*, SI: *Snapshot Isolation*, TCC: *Transactional Causal Consistency*, SSER: *Strict Serializability*, FR: *Fully Replicated*, LF: *Leader-Follower*

Table 1. Popular graph database feature support

For GDBMSes, there is a need for custom solutions to unify the expressiveness of the property graph model with the guaranteed conflict resolution of RDTs. We call such replicated data types PG-RDTs. Existing RDTs can support the building blocks of property graphs, i.e., strings for labels, maps for properties, and sets for vertices and edges. RDTs such as Sequences [28], Replicated Growable Arrays [33], and Treedoc [32] provide modifiable string types that can be used as labels. Also, Map RDTs can define properties, and Set RDTs can store vertices and edges. The challenge lies in unifying these replicated data types into a usable data structure which, not only, provides a policy for resolving conflicts but also the ability to express constraints without having to manage them within the application. For example, reciprocal consistency is a simple constraint which ensures bidirectional traversability of an edge between vertices that are on two separate shards [43]. Graph databases listed in Table 1 rely on underlying RDBMSes for constraints or do not guarantee constraints at all. To our knowledge, no native graph database handles even trivial constraints.

Research Challenge 3: Ensuring that a PG-RDT framework preserves constraints and invariants.

Wang et al. [42] propose replication-aware linearizability as a criterion to verify the correctness of RDT. The authors manually encode RDTs in Boogie [5] to obtain automatic correctness proofs for various implementations. However, they do not consider graph-based structures. Recently, Soundarapandian et al. [37] proposed to build and verify MRDTs using F* and SMT solvers. The proof rule parameterized by weakly consistent models [27] is proposed to automatically check RDT convergence. Nagar and Jagannathan [27] designs a custom verification strategy that takes advantage of first-order logic specifications and the Z3 solver [11]. While basic graph RDTs (2P2P-Graph and Graph-with-ORSet) are supported, correct RDT implementations may be rejected due to imprecise specifications. The work by Waudby et al. [44] is the first attempt to test constraint compliance for anomaly detection in graph databases. *To the best of our knowledge, there is no off-the-shelf tool that can be directly for reliable, invariant-preserving property graph replication.*

6 Conclusion

Despite growing demands for graph database scalability [6], comparatively less attention has been paid to decentralized GDBMSes. Also, while many recent works focus on performance [10, 23, 25], guaranteed correctness in this setting is yet to be investigated. While systems like Oracle Graph or DB2 Graph leverage existing research on invariant preservation for traditional RDBMSes, there is a need for principled solutions to preserve constraints and invariants against the practical limitations of synchronization brought forth by decentralization. We argue that proposing such solutions requires carefully analyzing the consistency requirements and designing novel replicated types to uphold these invariants.

Acknowledgments

This work was supported by the grants ANR-24-CE25-1109 (Dumbrava), PHC Pessoa 2024 50959YJ (Dumbrava and Ferreira), UID/04516/NOVA LINCS FCT.IP, EU Horizon Europe under Grant Agreement no. 101093006 (Ferreira, Pereira, and Preguiça) and ANR-23-PECL-0004 under the France 2030 program (Julien Sopena).

References

[1] Peter Alvaro, Neil Conway, Joe Hellerstein, and William Marczak. 2011. Consistency Analysis in Bloom: a CALM and Collected Approach. In *Biennial Conf. on Innovative DataSystems Research (CIDR)*. Asilomar, CA, USA. http://www.cidrdb.org/cidr2011/

[2] Renzo Angles. 2018. The Property Graph Database Model. In *AMW (CEUR Workshop Proceedings, Vol. 2100)*. CEUR-WS.org.

[3] Hagit Attiya, Faith Ellen, and Adam Morrison. 2017. Limitations of Highly-Available Eventually-Consistent Data Stores. *IEEE Trans. on Parallel and Dist. Sys. (TPDS)* 28, 1 (Jan. 2017), 141–155. https://doi.org/10.1109/TPDS.2016.2556669

[4] Peter Bailis, Aaron Davidson, Alan D. Fekete, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. 2013. Highly Available Transactions: Virtues and Limitations. *Proc. VLDB Endow.* 7, 3 (2013), 181–192. https://doi.org/10.14778/2732232.2732237

[5] Michael Barnett, Bor-Yuh Evan Chang, Robert DeLine, Bart Jacobs, and K. Rustan M. Leino. 2005. Boogie: A Modular Reusable Verifier for Object-Oriented Programs. In *FMCO (Lecture Notes in Computer Science, Vol. 4111)*. Springer, 364–387. https://doi.org/10.1007/11804192_17

[6] Maciej Besta, Robert Gerstenberger, Emanuel Peter, Marc Fischer, Michal Podstawski, Claude Barthels, Gustavo Alonso, and Torsten Hoefler. 2024. Demystifying Graph Databases: Analysis and Taxonomy of Data Organization, System Designs, and Graph Queries. *ACM Comput. Surv.* 56, 2 (2024), 31:1–31:40. https://doi.org/10.1145/3604932

[7] Sarra Bouhenni, Saïd Yahiaoui, Nadia Nouali-Taboudjemat, and Hamamache Kheddouci. 2021. A Survey on Distributed Graph Pattern Matching in Massive Graphs. *ACM Comput. Surv.* 54, 2, Article 36 (Feb. 2021), 35 pages. https://doi.org/10.1145/3439724

[8] Andrea Cerone, Giovanni Bernardi, and Alexey Gotsman. 2015. A Framework for Transactional Consistency Models with Atomic Visibility. In *Int. Conf. on Concurrency Theory (CONCUR) (Leibniz Int. Proc. in Informatics (LIPIcs), Vol. 42)*, Luca Aceto and David de Frutos Escrig (Eds.). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 58–71. https://doi.org/10.4230/LIPIcs.CONCUR.2015.58

[9] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM Comput. Surv.* 41, 3, Article 15 (jul 2009), 58 pages. https://doi.org/10.1145/1541880.1541882

[10] Marek Ciglan, Alex Averbuch, and Ladislav Hluchý. 2012. Benchmarking Traversal Operations over Graph Databases. In *Workshops Proceedings of the IEEE 28th International Conference on Data Engineering, ICDE 2012, Arlington, VA, USA, April 1-5, 2012*, Anastasios Kementsietsidis and Marcos Antonio Vaz Salles (Eds.). IEEE Computer Society, 186–189. https://doi.org/10.1109/ICDEW.2012.47

[11] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. 2008. Z3: An Efficient SMT Solver. In *TACAS (Lecture Notes in Computer Science, Vol. 4963)*. Springer, 337–340. https://doi.org/10.1007/978-3-540-78800-3_24

[12] Emil Eifrem. 2016. Graph databases: the key to foolproof fraud detection? *Computer Fraud & Security* 2016, 3 (2016), 5–8. https://doi.org/10.1016/S1361-3723(16)30024-0

[13] Fraunhofer SCAI. 2020. The Biomedical Knowledge Miner. https://bikmi.covid19-knowledgespace.de/

[14] Seth Gilbert and Nancy Lynch. 2002. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News* 33, 2 (2002), 51–59. https://doi.org/10.1145/564585.564601

[15] Alexey Gotsman, Hongseok Yang, Carla Ferreira, Mahsa Najafzadeh, and Marc Shapiro. 2016. 'Cause I'm Strong Enough: Reasoning about Consistency Choices in Distributed Systems. In *Symp. on Principles of Prog. Lang. (POPL)*. Assoc. for Computing Machinery, St. Petersburg, FL, USA, 371–384. https://doi.org/10.1145/2837614.2837625

[16] HealthECCO. 2020. CovidGraph. https://healthecco.org/

[17] Harald Andrés Helfgott, Jitendra Bajpai, and Daniele Dona. 2017. Graph isomorphisms in quasi-polynomial time. arXiv:1710.04574 [math.GR] https://arxiv.org/abs/1710.04574

[18] Samar El Helou, Shinji Kobayashi, Goshiro Yamamoto, Naoto Kume, Eiji Kondoh, Shusuke Hiragi, Kazuya Okamoto, Hiroshi Tamura, and Tomohiro Kuroda. 2019. Graph databases for openEHR clinical repositories. *Int. J. Comput. Sci. Eng.* 20, 3 (2019), 281–298. https://doi.org/10.1504/IJCSE.2019.103955

[19] Farzin Houshmand and Mohsen Lesani. 2019. Hamsaz: Replication Coordination Analysis and Synthesis. In *Symp. on Principles of Prog. Lang. (POPL) (Proc. ACM Program. Lang., Vol. 3)*. Assoc. for Computing Machinery, Cascais, Portugal, 74:1–74:32. https://doi.org/10.1145/3290387

[20] ICIJ. 2022. *Panama Papers Case Study*. https://guides.neo4j.com/sandbox/icij-panama-papers/datashape.html.

[21] Neo4j Inc. 2021. Graph Database Use Cases for Financial Services Companies. https://neo4j.com/use-cases/telecom/

[22] Martin Kleppmann, Adam Wiggins, Peter van Hardenberg, and Mark McGranaghan. 2019. Local-First Software: You Own Your Data, in spite of the Cloud. In *Int. Symp. on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!* Assoc. for Computing Machinery Special Interest Group on Pg. Lang. (SIGPLAN), Assoc. for Computing Machinery, Athens, Greece, 154–178. https://doi.org/10.1145/3359591.3359737

[23] Matteo Lissandrini, Martin Brugnara, and Yannis Velegrakis. 2018. Beyond Macrobenchmarks: Microbenchmark-based Graph Database Evaluation. *Proc. VLDB Endow.* 12, 4 (2018), 390–403. https://doi.org/10.14778/3297753.3297759

[24] Benoît Martin, Laurent Prosperi, and Marc Shapiro. 2023. Transactional-Turn Causal Consistency. In *Euro. Conf. on Parallel and Dist. Comp. (Euro-Par) (Lecture Notes in Comp. Sc. (LNCS), Vol. LNCS 14100)*, José Cano, Marios D. Dikaiakos, George A. Papadopoulos, Miquel Pericàs, and Rizos Sakellariou (Eds.). Springer-Verlag, Limassol, Cyprus, 578–591. https://doi.org/10.1007/978-3-031-39698-4_39

[25] Robert Campbell McColl, David Ediger, Jason Poovey, Dan Campbell, and David A. Bader. 2014. A performance evaluation of open source graph databases. In *Proceedings of the First Workshop on Parallel Programming for Analytics Applications* (Orlando, Florida, USA) *(PPAA '14)*. Association for Computing Machinery, New York, NY, USA, 11–18. https://doi.org/10.1145/2567634.2567638

[26] Jéssica Monteiro, Filipe Sá, and Jorge Bernardino. 2023. Graph Databases Assessment: JanusGraph, Neo4j, and TigerGraph. In *Perspectives and Trends in Education and Technology*, Anabela Mesquita, António Abreu, João Vidal Carvalho, and Cristina Helena Pinto de Mello (Eds.). Springer Nature Singapore, Singapore, 655–665.

[27] Kartik Nagar and Suresh Jagannathan. 2019. Automated Parameterized Verification of CRDTs. In *CAV (2) (Lecture Notes in Computer Science, Vol. 11562)*. Springer, 459–477. https://doi.org/10.1007/978-3-030-25543-5_26

[28] Brice Nédelec, Pascal Molli, Achour Mostéfaoui, and Emmanuel Desmontils. 2013. LSEQ: an adaptive structure for sequences in distributed collaborative editing. In *ACM Symposium on Document Engineering 2013, DocEng '13, Florence, Italy, September 10-13, 2013*, Simone Marinai and Kim Marriott (Eds.). ACM, 37–46. https://doi.org/10.1145/2494266.2494278

[29] Inc. Neo4j. 2022. *Transport for London Case Study*. https://neo4j.com/case-studies/transport-for-london/.

[30] Yubin Park, Mallikarjun Shankar, Byung-Hoon Park, and Joydeep Ghosh. 2014. Graph databases for large-scale healthcare systems: A framework for efficient data management and data services. In *Workshops Proceedings of the 30th International Conference on Data Engineering Workshops, ICDE 2014, Chicago, IL, USA, March 31 - April 4, 2014*. IEEE Computer Society, 12–19. https://doi.org/10.1109/ICDEW.2014.6818295

[31] Nataliia Pobiedina, Stefan Rümmele, Sebastian Skritek, and Hannes Werthner. 2014. Benchmarking Database Systems for Graph Pattern

Matching. In *Database and Expert Systems Applications - 25th International Conference, DEXA 2014, Munich, Germany, September 1-4, 2014. Proceedings, Part I (Lecture Notes in Computer Science, Vol. 8644)*, Hendrik Decker, Lenka Lhotská, Sebastian Link, Marcus Spies, and Roland R. Wagner (Eds.). Springer, 226–241. https://doi.org/10.1007/978-3-319-10073-9_18

[32] Nuno Preguiça, Joan Manuel Marquès, Marc Shapiro, and Mihai Leţia. 2009. A commutative replicated data type for cooperative editing. In *Int. Conf. on Distributed Comp. Sys. (ICDCS)*. Montréal, Canada, 395–403. https://doi.org/10.1109/ICDCS.2009.20

[33] Hyun-Gul Roh, Myeongjae Jeon, Jinsoo Kim, and Joonwon Lee. 2011. Replicated abstract data types: Building blocks for collaborative applications. *J. Parallel Distributed Comput.* 71, 3 (2011), 354–368. https://doi.org/10.1016/J.JPDC.2010.12.006

[34] Michael Rudolf, Marcus Paradies, Christof Bornhövd, and Wolfgang Lehner. 2013. The Graph Story of the SAP HANA Database. In *Datenbanksysteme für Business, Technologie und Web (BTW), 15. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS), 11.-15.3.2013 in Magdeburg, Germany. Proceedings (LNI, Vol. P-214)*, Volker Markl, Gunter Saake, Kai-Uwe Sattler, Gregor Hackenbroich, Bernhard Mitschang, Theo Härder, and Veit Köppen (Eds.). GI, 403–420. https://dl.gi.de/handle/20.500.12116/17334

[35] Sherif Sakr, Angela Bonifati, Hannes Voigt, Alexandru Iosup, Khaled Ammar, Renzo Angles, Walid G. Aref, Marcelo Arenas, Maciej Besta, Peter A. Boncz, Khuzaima Daudjee, Emanuele Della Valle, Stefania Dumbrava, Olaf Hartig, Bernhard Haslhofer, Tim Hegeman, Jan Hidders, Katja Hose, Adriana Iamnitchi, Vasiliki Kalavri, Hugo Kapp, Wim Martens, M. Tamer Özsu, Eric Peukert, Stefan Plantikow, Mohamed Ragab, Matei Ripeanu, Semih Salihoglu, Christian Schulz, Petra Selmer, Juan F. Sequeda, Joshua Shinavier, Gábor Szárnyas, Riccardo Tommasini, Antonino Tumeo, Alexandru Uta, Ana Lucia Varbanescu, Hsiang-Yun Wu, Nikolay Yakovets, Da Yan, and Eiko Yoneki. 2021. The future is big graphs: a community view on graph processing systems. *Commun. ACM* 64, 9 (2021), 62–71.

[36] Marc Shapiro, Annette Bieniusa, Nuno Preguiça, Valter Balegas, and Christopher Meiklejohn. 2018. *Just-Right Consistency: reconciling availability and safety.* Rapport de Recherche 9145. Inria Paris; Sorbonne Universités; Tech. U. Kaiserslautern; U. Nova de Lisboa; U. Catholique de Louvain, Paris, France.

[37] Vimala Soundarapandian, Adharsh Kamath, Kartik Nagar, and K. C. Sivaramakrishnan. 2022. Certified mergeable replicated data types. In *PLDI*. ACM, 332–347. https://doi.org/10.1145/3519939.3523735

[38] Sherry Tiao. 2021. Graph Database Use Cases for Financial Services Companies. https://blogs.oracle.com/database/post/graph-database-use-cases-for-financial-services-companies

[39] Gonçalo Tomás, Peter Zeller, Valter Balegas, Deepthi Devaki Akkoorath, Annette Bieniusa, João Leitão, and Nuno M. Preguiça. 2017. FMKe: a Real-World Benchmark for Key-Value Data Stores. In *Proceedings of the 3rd International Workshop on Principles and Practice of Consistency for Distributed Data, PaPoC@EuroSys 2017, Belgrade, Serbia, April 23 - 26, 2017*, Annette Bieniusa and Alexey Gotsman (Eds.). ACM, 7:1–7:4. https://doi.org/10.1145/3064889.3064897

[40] Sultan N. Turhan. 2023. Leveraging Graph Databases for Enhanced Healthcare Data Management: A Performance Comparison Study. In *IEEE International Conference on Big Data, BigData 2023, Sorrento, Italy, December 15-18, 2023*, Jingrui He, Themis Palpanas, Xiaohua Hu, Alfredo Cuzzocrea, Dejing Dou, Dominik Slezak, Wei Wang, Aleksandra Gruca, Jerry Chun-Wei Lin, and Rakesh Agrawal (Eds.). IEEE, 5007–5013. https://doi.org/10.1109/BIGDATA59044.2023.10386297

[41] European Union. 2020. COVID-19 data as linked data. https://op.europa.eu/en/web/eudatathon/covid-19-linked-data

[42] Chao Wang, Constantin Enea, Suha Orhun Mutluergil, and Gustavo Petri. 2019. Replication-aware linearizability. In *PLDI*. ACM, 980–993. https://doi.org/10.1145/3314221.3314617

[43] Jack Waudby, Paul Ezhilchelvan, Jim Webber, and Isi Mitrani. 2020. Preserving reciprocal consistency in distributed graph databases. In *Proceedings of the 7th Workshop on Principles and Practice of Consistency for Distributed Data* (Heraklion, Greece) *(PaPoC '20)*. Association for Computing Machinery, New York, NY, USA, Article 2, 7 pages. https://doi.org/10.1145/3380787.3393675

[44] Jack Waudby, Benjamin A. Steer, Karim Karimov, József Marton, Peter Boncz, and Gábor Szárnyas. 2021. Towards Testing ACID Compliance in the LDBC Social Network Benchmark. In *Performance Evaluation and Benchmarking*, Raghunath Nambiar and Meikel Poess (Eds.). Springer International Publishing, Cham, 1–17.