Reduce Once, Verify Many: Verifying Isolation Guarantees via Hierarchical Abstractions

Shabnam Ghasemirad
ETH Zurich, Switzerland
shabnam.ghasemirad@inf.ethz.ch

Christoph Sprenger
ETH Zurich, Switzerland
sprenger@inf.ethz.ch

Si Liu
Texas A&M University, USA
si.liu@tamu.edu

David Basin
ETH Zurich, Switzerland
basin@inf.ethz.ch

Abstract

We present a mathematically rigorous, systematic approach for the verification of database isolation guarantees, which (i) supports a spectrum of seven isolation levels, (ii) uncovers a fundamental dichotomy among isolation levels: stronger levels can be verified via refinement alone, whereas weaker levels additionally require reduction, and (iii) provides a hierarchy of abstract models that substantially simplifies proofs by factoring out their most labor-intensive parts. In particular, we eliminate the need for per-protocol reduction proofs for the weaker class of isolation levels by performing a once-and-for-all reduction at a high level of abstraction in our hierarchy. To achieve this, we develop and apply a generic theory of reduction, which is also of more general interest. Overall, our approach minimizes the user's proof effort to a single, simpler refinement of the most concrete model in our hierarchy. All our results are formalized in Isabelle/HOL.

1 Introduction

Modern databases trade strong isolation guarantees, such as serializability, for performance, offered by *weak* isolation levels [3, 4], including read atomicity, causal consistency, snapshot isolation, and more (cf. Figure 1). Weaker isolation levels permit more complex concurrent behavior, which makes the concurrency control protocols at those levels harder to get right. Indeed, weak-isolation bugs, stemming especially from design-level defects, have repeatedly surfaced in carefully designed and extensively tested production databases [13, 17].

Over the past decades, sustained efforts have been made to develop reliable databases that support weakly isolated distributed transactions. Reasoning about isolation guarantees in these systems requires a formal semantics for isolation levels and a methodology for their formal verification.

Although the semantics have been extensively studied, including dependency graphs [5], abstract executions [8], and operational semantics [23], tool support for verifying them remains rare. Our recent VerIso [10] framework takes an important step toward closing this gap by mechanizing the operational semantics of [23] in Isabelle/HOL [22]. Its core abstraction, namely the centralized isolation-level model

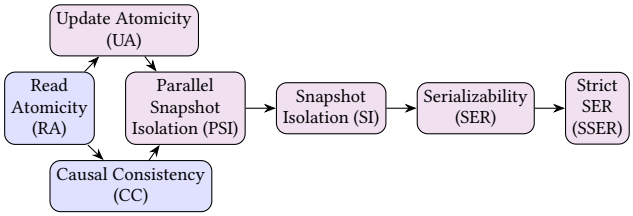

Figure 1. Isolation levels, weaker (⟶) stronger. Blue levels (IL ≤ CC) require a reduction step in addition to refinement. Violet levels (IL ≥ UA) directly refine the abstract model.

(*CIL*), captures a multi-versioned key–value store and different client views. Parameterized by isolation levels, *CIL* can be instantiated with a wide range of them shown in Figure 1.

In VerIso, one verifies isolation guarantees by showing that a concurrency control protocol *refines* the abstract isolation-level model *CIL* instantiated to the target level. However, such proofs are labor-intensive and do not scale for two reasons: (1) there is a substantial abstraction gap between the abstract *CIL* model and concrete protocol models, which requires rather involved refinement mappings linking the fine-grained protocol steps to the one-shot commit event in the *CIL* model, and (2) refinement alone is sometimes insufficient (since there is no trace inclusion) and must be complemented by a Lipton-style *reduction* [16]. These reduction proofs are often done ad-hoc and on a per-protocol basis, which requires considerable additional effort.

In this work, we address these problems in three ways. First, we introduce a general Lipton-style [16] *reduction* framework, whose application substantially reduces the effort needed for reduction proofs. It is parameterized on an event system *ES* (e.g., distributed transaction systems), an event dependency relation D (e.g., write-read dependencies [1]), and a desired (good) event order G (e.g., the order induced by transaction timestamps). Assuming that these relations satisfy three general conditions, including the linearity of G and the commutativity of independent events, we prove that any execution of *ES* is transformable into a G-ordered execution by commuting a finite number of independent event pairs (w.r.t. D) while preserving the reachable states. To apply our result, one only needs to define the three

parameters and discharge the three conditions. Notably, this general result is not limited to the database context.

Second, to gain deeper insight into why reduction is needed, we examine isolation levels and the associated protocol correctness proofs that require reduction (e.g., [11]). We observe that write conflicts in protocols cause a mismatch between the logical order of events (e.g., according to timestamps) and their execution order, thereby necessitating a reduction proof to correct the mismatch. Since update atomicity (UA) [8] is the weakest level preventing write conflicts, direct refinement alone suffices to prove correctness for this and all stronger levels. Accordingly, we classify the isolation levels in our lattice from Figure 1 into two groups. The levels IL ≥ UA can be verified without reduction, whereas the verification of the two weaker levels IL ≤ CC, namely RA and CC, requires a combination of reduction and refinement.

Third, based on this insight, we construct a hierarchy of new abstract models with the aim of replacing per-protocol reduction proofs by a once-and-for-all reduction proof at a high abstraction level. Therefore, we construct a new abstract base model, TXM, based on transaction maps, which explicitly models timestamps and refines *CIL*. For this refinement to hold, we must restrict TXM such that the timestamp and execution order coincide. From TXM, we then derive two more specialized abstract models, $\text{TXM}_{UA\uparrow}$ and $\text{TXM}_{CC\downarrow}$, one for each IL group (Figure 2). We use our reduction framework to prove that $\text{TXM}_{CC\downarrow}$ reduces to TXM by reordering commits to respect the timestamp order. We also show that $\text{TXM}_{UA\uparrow}$ directly refines TXM. As a result, to prove a protocol's correctness, a refinement of the respective abstract model, instantiated to the desired IL, suffices.

Moreover, to factor out the verification effort of the shift from centralized to distributed settings, we introduce an abstract distributed protocol model (DTXM) parameterized over IL, which further refines the previous two models and captures the client/server two-phase commit (2PC) architecture shared by most modern distributed transaction systems. This concludes our hierarchy of intermediate models, which extends VerIso and reduces the verification of a new protocol to a single refinement of the distributed model (see Figure 2).

Contributions. To summarize, this work makes the following contributions:

1. A reduction framework, parameterized on an event system, a dependency relation, and a desired event order, for verifying distributed transaction protocols with weak isolation guarantees (Section 2).
2. A taxonomy of isolation levels ordered by whether their correctness proof follows from direct refinement of the abstract model (IL ≥ UA) or requires a *reduction step* in addition (IL ≤ CC), which we factor out and prove using our reduction framework (Section 4).

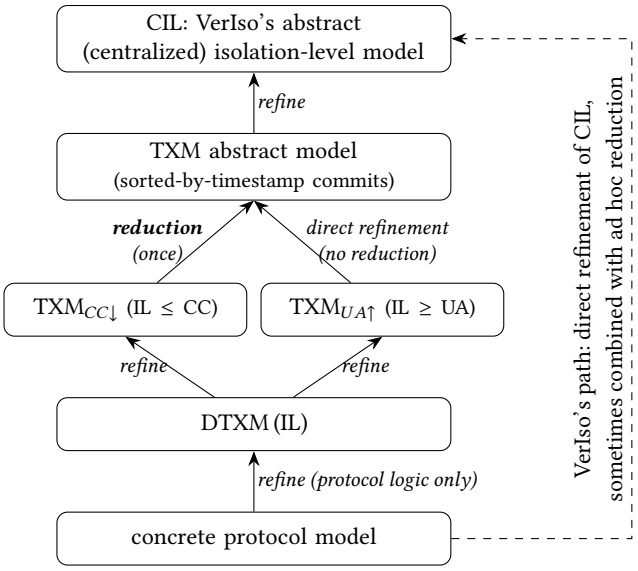

Figure 2. A hierarchy of models showing a transitive *inclusion of reachable states* (via refinement or reduction) of a concrete protocol in the states permitted by its target IL.

3. A hierarchy of abstract intermediate models that shrink the abstraction gap between VerIso's abstract model and concrete protocols (Sections 3 to 5).

The reduction framework, models' hierarchy, refinements, and the single reduction are all mechanized in Isabelle/HOL.

This work also complements two lines of research. First, while prior work has primarily viewed isolation levels through the lenses of system performance [2, 3, 18, 20] and black-box checking complexity [6, 7, 21], we study them from the perspective of protocol verification and uncover a related dichotomy. Second, while prior reduction frameworks based on Lipton's movers target concurrent *programs*, typically by inferring atomic blocks [9, 12, 14, 15], we provide a reduction framework for distributed systems that instead commutes protocol events into a certain desired order.

2 A Reduction Framework in Isabelle/HOL

In this section, we introduce our general reduction framework, mechanized in Isabelle/HOL. We model database protocols and abstract transaction systems as labeled transition (event) systems. Our framework is based on the idea that two independent events in an execution *commute*, i.e., swapping them disables neither event and yields the same final state.

Lipton in [16] uses this idea to group program statements into atomic blocks, to simplify reasoning about concurrent systems. We generalize this concept by commuting events into a given *desired* order, while preserving reachable states.

Our framework is parameterized on an event system *ES*, an event dependency relation *D*, and a partial function *G* that specifies the desired (good) event order. *G* assigns a linearly ordered label (e.g. a timestamp) to a subset of events.

We identify a small set of assumptions on these parameters, which is required for the reduction to work:

1. $\forall \tau \in traces\ ES. \tau = \alpha \cdot e_1 \cdot e_2 \cdot \beta \ \wedge \ (e_1, e_2) \notin D \implies (e_2, e_1) \notin D$, i.e., if two events on the trace are independent, commuting them preserves their independence.
2. $\forall \tau \in traces\ ES.\ \tau = \alpha \cdot e_2 \cdot \beta \cdot e_1 \cdot \gamma \wedge G(e_1) < G(e_2) \implies (e_2, e_1) \notin D$, i.e., "badly ordered" events are independent.
3. $\forall \tau \in traces\ ES.\ \tau = \alpha \cdot e_1 \cdot \beta \cdot e_2 \cdot \gamma \wedge (e_1, e_2) \notin D \implies commute\ ES\ e_1\ e_2$, i.e., independent events commute.

Theorem 2.1 (Reduction Soundness)**.** *Let ES be an event system with an event dependency relation D and a desired event order G, satisfying the framework's obligations. Then every reachable state of ES is reachable by an execution where the event order is consistent with G (a G-ordered execution).*

The proof reorders an arbitrary execution into its sorted form by repeatedly swapping events. Then, a well-founded measure that strictly decreases under commuting events, marking progress towards a G-sorted execution, guarantees termination. Verifying an arbitrary-interleaving system thus reduces to verifying only its sorted executions.

3 Abstract Base Model: Transaction Maps

We introduce Transaction Maps (TXM), a new parameterized abstract model whose instantiations capture a range of well-known and prevalent isolation levels (Figure 1). The target isolation level IL is specified by two model parameters:

- R_{IL}, a relation on transaction IDs, and
- $vShift_{IL}$, a predicate on allowed client view updates.

We model TXM by defining its states and events as follows.

TXM states. The TXM model's states (M, U) consist of a *transaction map* M from transaction IDs to per-transaction states, and a *client views map* U from client IDs to their views, i.e., the set of transactions they see. Unlike VerIso's *CIL* model, which uses key-value version lists, this model stores information *per transaction*, comprising (i) a read map, (ii) a write map, and (iii) a transaction timestamp.

This change reduces the abstraction gap to concrete protocols because one works directly with *transaction IDs*, which serve both as the main way to access transaction states and as a replacement for list indices in client views. This approach can be seen as an operational version of abstract executions [8], as it defines a visibility relation between transactions via client views, and assigns timestamps to transactions, resembling the arbitration relation in abstract executions. TXM also offers view atomicity for free, i.e., clients see either all or none of other transactions' writes, and concrete protocols *inherit* the atomic view by refining this model.

TXM events. TXM has two events: a view extension event and an (atomic) commit event. The view extension event extends one client's view and preserves M. The commit event changes both M and the committing client's view u (in U), and requires that all of the following conditions hold for M, u, and their updates M' and u' resulting from the commit.

1. u is *backward closed* under R_{IL} (a.k.a. $canCommit_{IL}$).
2. The $vShift_{IL}(M, u, M', u')$ predicate holds.
3. The client reads the latest version in its view (*last-write-wins* policy).
4. The transaction ID and timestamp are fresh.
5. For each involved write key, the timestamp is larger than all previous write transaction timestamps on that key (sorted-timestamps condition).

The last condition restricts this model's executions to those in which the commits on each key are sorted by timestamp.

Finally, we show that this model refines VerIso's *CIL* model. So, by transitivity, any model that refines TXM also refines *CIL*. (The reverse refinement, which would establish equivalence, is work in progress.)

4 Isolation Levels Taxonomy, Refinement, and Once-and-for-All Reduction

In this section, we first discuss general proof techniques for transaction isolation verification. Then, we propose a new characterization of isolation levels and the associated proof techniques. In order to capture recurring proof patterns, we introduce two new models in our model hierarchy, namely $\text{TXM}_{CC\downarrow}$ and $\text{TXM}_{UA\uparrow}$ (see also Figure 2) and generalize those patterns as a once-and-for-all effort.

Refinement is a standard, commonly used technique to show that concurrency control protocols satisfy their isolation guarantees. A refinement is defined between two event systems at different abstraction levels, using a refinement mapping pair (r, π) that maps the states and events of one system to those of the other, respectively. We say an event system ES_1 refines event system ES_2, if (i) r preserves initial states, and (ii) (r, π) maps ES_1 transitions to ES_2 transitions, i.e., if $s \xrightarrow{e}_1 s'$, then $r(s) \xrightarrow{\pi(e)}_2 r(s')$.

Refinement guarantees inclusion of reachable states modulo r, i.e., $r(reach(ES_1)) \subseteq reach(ES_2)$, with r applied pointwise. For database protocols, showing that the concrete concurrency control protocol refines the desired isolation level's event system proves that the protocol's reachable states are included in those of the isolation level, thereby establishing satisfaction of that level's guarantees.

Note, however, that for such correctness proofs, a *direct* refinement of the abstract isolation-level model (whether of *CIL* or base TXM model) does not always work. We can observe this, for example, in the Eiger-PORT+[11] protocol. In this protocol, a read transaction always reads the version with the *highest timestamp* below a certain global safe time (which determines a subset of already committed transactions). In contrast, both abstract models *CIL* and TXM require that a client always reads the *latest* committed version in its view, which is determined by the *execution order* of commit events. However, the execution order of commits and the

order of the associated commit timestamps may not coincide. We call such commits *inverted commits*.

Inverted commits in an execution cannot simulate the behavior of the abstract isolation-level models *CIL* or TXM. In *CIL*, simulating such commits requires *inserting* a key's new version to its version list rather than *appending* it, which is not allowed. Similarly, in TXM, which refines *CIL*, such commits are excluded by the sorted-timestamps condition. We solve this problem by combining reduction and refinement. Reduction orders the commit events by their timestamps, resulting in the inversion-free (a.k.a. restricted/sorted) model TXM, and TXM subsequently refines the *CIL* abstract model.

To pinpoint the root cause of this problem, we examined a range of isolation levels and observed that if a level disallows *write conflicts*, then this kind of timestamp inversion is avoided. Based on this observation, we classify isolation levels into two groups, as shown in Figure 1.

$\text{TXM}_{\text{UA}\uparrow}$ *and direct refinement.* For levels stronger than update atomicity, **IL ≥ UA** (UA, PSI, SI, SER, SSER), we introduce the $\text{TXM}_{UA\uparrow}$ intermediate model, which differs from the TXM abstract model in two ways.

First, IL ≥ UA entails that IL's commit condition implies the UA commit condition ($canCommit_{UA}$). This condition requires the client view to be backward closed under the relation $R_{UA} = \bigcup_{(k,v)\in write\_map} WW_M^{-1}(k)$, meaning it should include *all* versions of the keys in the transaction's write map (because of closure on write-write dependency), essentially avoiding write conflicts.

Second, the model replaces the restrictive *sorted timestamps* condition of TXM with a much weaker timestamp condition, namely requiring a client to choose a timestamp greater than the transactions' timestamps *in its view*. This condition is respected by timestamped protocols, since the timestamp always increases locally for a client, and its choice depends only on the client's view, rather than on *all* committed transactions of the involved write keys.

Combining these two conditions, we prove that for all keys a transaction commits to, the timestamp will be greater than those of the existing transactions in the client's view, which must, in turn, include *all* versions of those keys. Therefore, this model directly refines the TXM abstract model, and *no* reduction is needed to put commits in the correct order.

$\text{TXM}_{\text{CC}\downarrow}$ *and the once-and-for-all reduction.* For levels **IL ≤ CC** (RA and CC), we introduce the $\text{TXM}_{CC\downarrow}$ intermediate model, which removes the *sorted timestamps* condition of TXM in exchange for a few weaker timestamp conditions similar to $\text{TXM}_{UA\uparrow}$. Here, however, the instantiated levels are weaker and therefore permit more complex concurrent behavior. This breaks trace inclusion between this model and TXM, making a direct refinement of TXM infeasible.

Therefore, to prove the inclusion of the set of reachable states for the IL ≤ CC region, represented by the $\text{TXM}_{CC\downarrow}$ model, we instantiate the framework from Section 2 a *single* time, with an event dependency relation D_{CC} and the commit timestamp order G_{CC}, as the desired good order.

D_{CC} defines two events e_1 and e_2 as dependent in two cases: (1) if both events are from the same client, or (2) if e_1 is a commit, and its transaction ID is in e_2's client view (capturing write-read dependency).

G_{CC} maps commit events to their timestamps, thereby indicating the timestamp order as the desired order of commits.

The instantiation yields three proof obligations (see Section 2), which we prove hold for $\text{TXM}_{CC\downarrow}$, using D_{CC} and G_{CC}. As a result, $\text{TXM}_{CC\downarrow}$ is proven reducible to the TXM model, in which the commits are ordered by their timestamps. Thus, every concrete protocol that refines $\text{TXM}_{CC\downarrow}$ automatically satisfies the reduced semantics and the level guarantee, with no further reduction reasoning. The reduction argument, the hardest part of the proof, is paid for exactly once.

5 Distributed Transaction Maps

Distribution in most concurrency control protocols is structured as clients and servers running a two-phase commit protocol. We capture this in the Distributed Transaction Maps (DTXM) model, which bridges the centralized TXM models to distributed protocol models.

DTXM's global state includes client and server configurations, and a map of transactions to timestamps. A client configuration stores the client's view, its current transaction's ID, timestamp, and state machine *init*→*ops*→*prepared*→*committed*/*aborted*. A server configuration stores, for each (t, k) transaction ID and key pair, a state machine *idle*→*read*/*prepared*→*committed*/*aborted*. DTXM events form a full 2PC protocol: read request and response, prepare, commit, and abort.

We show that DTXM refines previous TXM models by proving that its client commit event refines the *one-shot atomic commit* of TXM models. Therefore, a future protocol correctness proof requires *only a refinement* of DTXM.

6 Conclusions and Future Work

We have introduced a new approach to transaction isolation verification and factored out its labor-intensive proof steps by developing a generic reduction framework and a hierarchy of abstract models. We have also shed new light on isolation levels from a protocol verification perspective by distinguishing those that can be verified by refinement alone from those that additionally require reduction.

In order to validate our approach at both ends of the isolation-level lattice, we are currently verifying two concurrency control protocols: for the IL ≤ CC class, we reverify Eiger-PORT+[11] as causally consistent, and for the IL ≥ UA class, we verify that ROLA [19] satisfies UA. The reverification highlights the reduced proof effort as a result of using our reduce-once model hierarchy, while the second case study witnesses the generality of our approach.

References

[1] Atul Adya. 1999. *Weak consistency: a generalized theory and optimistic implementations for distributed transactions.* Ph. D. Dissertation. Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science.

[2] Karolos Antoniadis, Diego Didona, Rachid Guerraoui, and Willy Zwaenepoel. 2020. The Impossibility of Fast Transactions. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 1143–1154. doi:10.1109/IPDPS47924.2020.00120

[3] 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. doi:10.14778/2732232.2732237

[4] Hal Berenson, Phil Bernstein, Jim Gray, Jim Melton, Elizabeth O'Neil, and Patrick O'Neil. 1995. A critique of ANSI SQL isolation levels. *SIGMOD Rec.* 24, 2 (May 1995), 1–10. doi:10.1145/568271.223785

[5] Philip A Bernstein and Nathan Goodman. 1981. Concurrency control in distributed database systems. *ACM Computing Surveys (CSUR)* 13, 2 (1981), 185–221. doi:10.1145/356842.356846

[6] Ranadeep Biswas and Constantin Enea. 2019. On the Complexity of Checking Transactional Consistency. In *Proc. ACM Program. Lang. (OOPSLA)*, Vol. 3. 165:1–165:28. doi:10.1145/3360591

[7] Ahmed Bouajjani, Constantin Enea, and Enrique Román-Calvo. 2025. On the Complexity of Checking Mixed Isolation Levels for SQL Transactions. In *Computer Aided Verification: 37th International Conference, CAV 2025, Zagreb, Croatia, July 23-25, 2025, Proceedings, Part IV*. Springer-Verlag, 315–337. doi:10.1007/978-3-031-98685-7_15

[8] Andrea Cerone, Giovanni Bernardi, and Alexey Gotsman. 2015. A Framework for Transactional Consistency Models with Atomic Visibility. In *26th International Conference on Concurrency Theory, CONCUR 2015 (LIPIcs, Vol. 42)*, Luca Aceto and David de Frutos-Escrig (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 58–71. doi:10.4230/LIPICS.CONCUR.2015.58

[9] Namratha Gangamreddypalli, Constantin Enea, and Shaz Qadeer. 2026. Reduction for Structured Concurrent Programs. In *Programming Languages and Systems: 35th European Symposium on Programming, ESOP 2026, Held as Part of the International Joint Conferences on Theory and Practice of Software, ETAPS 2026, Turin, Italy, April 11–16, 2026, Proceedings, Part I* (Turin, Italy). Springer-Verlag, Berlin, Heidelberg, 252–282. doi:10.1007/978-3-032-22720-1_10

[10] Shabnam Ghasemirad, Si Liu, Christoph Sprenger, Luca Multazzu, and David Basin. 2025. VerIso: Verifiable Isolation Guarantees for Database Transactions. *Proc. VLDB Endow.* 18, 5 (2025), 1362–1375. doi:10.14778/3718057.3718065

[11] Shabnam Ghasemirad, Christoph Sprenger, Si Liu, Luca Multazzu, and David Basin. 2025. Pushing the Limit: Verified Performance-Optimal Causally-Consistent Database Transactions. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS), ETAPS (LNCS, Vol. 15698)*. 43–62. doi:10.1007/978-3-031-90660-2_3

[12] Chris Hawblitzel, Erez Petrank, Shaz Qadeer, and Serdar Tasiran. 2015. Automated and Modular Refinement Reasoning for Concurrent Programs. In *Computer Aided Verification (LNCS, Vol. 9207)*. 449–465. doi:10.1007/978-3-319-21668-3_26

[13] Jepsen. 2026. Jepsen Analyses. Retrieved June 1, 2026 from https://jepsen.io/analyses

[14] Bernhard Kragl and Shaz Qadeer. 2018. Layered Concurrent Programs. In *Computer Aided Verification (LNCS, Vol. 10981)*. 79–102. doi:10.1007/978-3-319-96145-3_5

[15] Bernhard Kragl, Shaz Qadeer, and Thomas A. Henzinger. 2020. Refinement for Structured Concurrent Programs. In *Computer Aided Verification (LNCS, Vol. 12224)*. 275–298. doi:10.1007/978-3-030-53288-8_14

[16] Richard J. Lipton. 1975. Reduction: A Method of Proving Properties of Parallel Programs. *Commun. ACM* 18, 12 (1975), 717–721. doi:10.1145/361227.361234

[17] Si Liu, Long Gu, Hengfeng Wei, and David Basin. 2024. Plume: Efficient and Complete Black-Box Checking of Weak Isolation Levels. *Proc. ACM Program. Lang.* 8, OOPSLA (2024), 876–904. doi:10.1145/3689742

[18] Si Liu, Luca Multazzu, Hengfeng Wei, and David A. Basin. 2024. NOCNOC: Towards Performance-optimal Distributed Transactions. *Proc. ACM Manag. Data* 2, 1, Article 9 (March 2024), 25 pages. doi:10.1145/3639264

[19] Si Liu, Peter Csaba Ölveczky, Qi Wang, Indranil Gupta, and José Meseguer. 2019. Read atomic transactions with prevention of lost updates: ROLA and its formal analysis. *Formal Aspects Comput.* 31, 5 (2019), 503–540. doi:10.1007/S00165-019-00489-W

[20] Haonan Lu, Siddhartha Sen, and Wyatt Lloyd. 2020. Performance-Optimal Read-Only Transactions. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 333–349. https://www.usenix.org/conference/osdi20/presentation/lu

[21] Lasse Møldrup and Andreas Pavlogiannis. 2025. AWDIT: An Optimal Weak Database Isolation Tester. *Proc. ACM Program. Lang.* 9, PLDI, Article 209 (June 2025), 25 pages. doi:10.1145/3742465

[22] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. 2002. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*. Lecture Notes in Computer Science, Vol. 2283. Springer. doi:10.1007/3-540-45949-9

[23] Shale Xiong, Andrea Cerone, Azalea Raad, and Philippa Gardner. 2020. Data Consistency in Transactional Storage Systems: A Centralised Semantics. In *34th European Conference on Object-Oriented Programming, ECOOP 2020 (LIPIcs, Vol. 166)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 21:1–21:31. doi:10.4230/LIPICS.ECOOP.2020.21