Verified Key-Value Stores satisfying Snapshot Isolation

Arnaud Daby-Seesaram
arnaud.daby-seesaram@cs.au.dk
Aarhus University
Aarhus, Denmark

Léon Ducruet
leon.ducruet@cs.au.dk
Aarhus University
Aarhus, Denmark

Lars Birkedal
birkedal@cs.au.dk
Aarhus University
Aarhus, Denmark

Amin Timany
timany@cs.au.dk
Aarhus University
Aarhus, Denmark

Isolation Levels constrain the behaviour of concurrent transactions in a database. Among them, Snapshot Isolation is particularly important and widely used since it provides strong guarantees while allowing more efficient implementations than Serialisability. However, isolation bugs are common [1]. Also, clients making incorrect assumptions about the isolation guarantees can lead to security vulnerabilities [12]. Testing techniques can detect bugs in both databases [8, 9, 13] and their clients [2], but cannot rule out bugs altogether.

In this work we use separation logic, a flavour of Hoare logic, to prove correctness of both an implementation of Snapshot Isolation, called SIrIS, and its clients. To this end, we use the Iris [7] separation logic framework and its Rocq mechanisation which we have used to mechanise all our results. Separation logic, and Iris in particular has already been proven useful for verifying databases and their clients [3, 5, 10]. Recent work by Mathiasen et al. [10] has shown how to specify several database isolation levels in separation logic, including Snapshot Isolation. However, their implementation is inefficient; it uses a global lock to protect the whole database. In contrast, SIrIS is an efficient in-memory key-value store library satisfying Snapshot Isolation. It is implemented based on a lock-free variant of Reed's multi-version concurrency control algorithm [11], and a verified garbage collector that periodically prunes stale versions.

Specs. We give specs for the following database operations: `init_kvs`, `start`, `read`, `scan`, `write`, `delete`, `abort`, and `commit`. Our specifications are based on so-called logically atomic triples which in addition to ordinary pre- and post-conditions, also specify the state of the database immediately before and after the linearisation point [6] of the operation. For instance, for the `commit` operation, the spec, given in Figure 1, specifies that:

Precondition: The transaction being committed is a valid transaction of the database db with snapshot S, written $\mathrm{IsTransaction}(tr, db, S)$.
Immediately before the lin. point: The transaction is in state (C, U), written $\mathrm{TransState}(tr, C, U)$, where C maps all keys accessed by tr to their values, and U is the set of keys

$$\{\mathrm{IsTransaction}(tr, db, S)\}$$
$$\left\langle \exists M,\ C,\ U.\ \mathrm{IsKvs}(db, M) * \mathrm{TransState}(tr, C, U) * \mathop{\Large *}_{k \in U} k \rightarrowtail^{db} M(k) \right\rangle$$
$$\texttt{commit}\ tr$$
$$\left\langle \begin{array}{l} b.\ b = \mathrm{CanCommit}(M, S, U)\ * \\ \quad \textbf{if } b \textbf{ then } \mathrm{IsKvs}(db, \mathrm{update}(M, C, U)) * \mathop{\Large *}_{(k,v) \in C \wedge k \in U} k \rightarrowtail^{db} v :: M(k) \\ \quad \textbf{else } \mathrm{IsKvs}(db, M) * \mathop{\Large *}_{k \in U} k \rightarrowtail^{db} M(k) \end{array} \right\rangle$$
$$\{\mathrm{True}\}$$

Figure 1. Specification of `commit`

whose value is overwritten or inserted during tr. The database is in state M ($\mathrm{IsKvs}(db, M)$). And, we require the exclusive ownership of all keys that have been *overwritten or inserted*, written $k \rightarrowtail^{db} M(k)$.[1] (Here, $*$ is the separating conjunction of separation logic.)
Immediately after the lin. point: The result of the operation is the boolean $b = \mathrm{CanCommit}(M, S, U)$ where the CanCommit function (a pure Rocq function) returns `true` if and only if the overwritten keys in U have the same version history in snapshot S as they do in the current state M, *i.e.*, if there was no write-write conflicts.
Postcondition: needs not carry any further information than the condition immediately after the linearisation point.

A simple client. In our specs $\mathrm{CanCommit}(M, S, U)$ ensures that `commit` does not fail more transactions than strictly necessary. This allows to *prove* the success of commit operations in client code. In particular, we can leverage separation logic's reasoning principles about fractional ownership to prove that concurrent transactions can succeed, as illustrated in Figure 2. Since T2 keeps half of the permission to overwrite key x (represented by $x \rightarrowtail^{db}_{1/2} [0]$), it can *prove* that no concurrent transaction can overwrite key x and thus, that it can successfully commit.

Evaluation. We ran the Yahoo! Cloud Serving Benchmarks (YCSB) [4] on SIrIS, which we have implemented in OCaml. SIrIS provided throughput in the neighbourhood of around 1 million transactions per second for workloads A–C (using 18 threads on a virtual machine with 20 cores).

[1] Database assertions keep track of the complete value history of keys. This simplifies the definition of CanCommit.

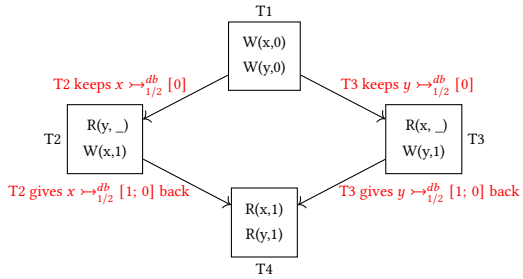

Figure 2. Provably Successful Transactions in Client-Code

Future work. Our proofs only ensure safety and functional correctness of SIrIS and its (verified) clients, *i.e.*, that they do not crash, and that they produce correct result when they terminate. However, we have not proven any liveness properties, *e.g.*, that there are no deadlocks or live-locks. We leave this as future work.

References

[1] 2026. Jepsen Analyses. https://jepsen.io/analyses. Accessed: 2026-03-12.

[2] Ranadeep Biswas, Diptanshu Kakwani, Jyothi Vedurada, Constantin Enea, and Akash Lal. 2021. MonkeyDB: effectively testing correctness under weak isolation levels. *Proc. ACM Program. Lang.* 5, OOPSLA (2021), 1–27. doi:10.1145/3485546

[3] Yun-Sheng Chang, Ralf Jung, Upamanyu Sharma, Joseph Tassarotti, M. Frans Kaashoek, and Nickolai Zeldovich. 2023. Verifying vMVCC, a high-performance transaction library using multi-version concurrency control. In *17th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2023, Boston, MA, USA, July 10-12, 2023*, Roxana Geambasu and Ed Nightingale (Eds.). USENIX Association, 871–886. https://www.usenix.org/conference/osdi23/presentation/chang

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

[5] Léon Gondelman, Simon Oddershede Gregersen, Abel Nieto, Amin Timany, and Lars Birkedal. 2021. Distributed causal memory: modular specification and verification in higher-order distributed separation logic. *Proc. ACM Program. Lang.* 5, POPL (2021), 1–29. doi:10.1145/3434323

[6] Maurice Herlihy and Jeannette M. Wing. 1990. Linearizability: A Correctness Condition for Concurrent Objects. *ACM Trans. Program. Lang. Syst.* 12, 3 (1990), 463–492. doi:10.1145/78969.78972

[7] Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Ales Bizjak, Lars Birkedal, and Derek Dreyer. 2018. Iris from the ground up: A modular foundation for higher-order concurrent separation logic. *J. Funct. Program.* 28 (2018), e20. doi:10.1017/S0956796818000151

[8] Kyle Kingsbury and Peter Alvaro. 2020. Elle: inferring isolation anomalies from experimental observations. *Proc. VLDB Endow.* 14, 3 (Nov. 2020), 268–280. doi:10.14778/3430915.3430918

[9] Si Liu, Long Gu, Hengfeng Wei, and David A. Basin. 2024. Plume: Efficient and Complete Black-Box Checking of Weak Isolation Levels. *Proc. ACM Program. Lang.* 8, OOPSLA2 (2024), 876–904. doi:10.1145/3689742

[10] Anders Alnor Mathiasen, Léon Gondelman, Léon Ducruet, Amin Timany, and Lars Birkedal. 2025. Reasoning about Weak Isolation Levels in Separation Logic. *Proc. ACM Program. Lang.* 9, ICFP (2025), 306–340. doi:10.1145/3747515

[11] David P. Reed. 1983. Implementing Atomic Actions on Decentralized Data. *ACM Trans. Comput. Syst.* 1, 1 (1983), 3–23. doi:10.1145/357353.357355

[12] Todd Warszawski and Peter Bailis. 2017. ACIDRain: Concurrency-Related Attacks on Database-Backed Web Applications. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu (Eds.). ACM, 5–20. doi:10.1145/3035918.3064037

[13] Jian Zhang, Ye Ji, Shuai Mu, and Cheng Tan. 2023. Viper: A Fast Snapshot Isolation Checker. In *Proceedings of the Eighteenth European Conference on Computer Systems, EuroSys 2023, Rome, Italy, May 8-12, 2023*, Giuseppe Antonio Di Luna, Leonardo Querzoni, Alexandra Fedorova, and Dushyanth Narayanan (Eds.). ACM, 654–671. doi:10.1145/3552326.3567492