Towards Efficient Trustworthy Data Systems

Muhammad El-Hindi

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Towards Efficient Trustworthy Data Systems

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Doctoral thesis by
Muhammad El-Hindi, M.Sc.

submitted in fulfillment of the requirements for the
degree of *Doctor rerum naturalium (Dr. rer. nat.)*

Reviewers
Prof. Dr. rer. nat. Carsten Binnig
Prof. Amr El Abbadi, Ph.D. (UC Santa Barbara, USA)

Department of Computer Science
Technical University of Darmstadt

Darmstadt, 2023

Muhammad El-Hindi: *Towards Efficient Trustworthy Data Systems*
Darmstadt, Technical University of Darmstadt
Year thesis published in TUprints: 2023
URN: urn:nbn:de:tuda-tuprints-244808
URL: https://tuprints.ulb.tu-darmstadt.de/24480
Date of the viva voce: 24.08.2023

Erklärung laut Promotionsordnung

§8 Abs. 1 lit. c PromO

Ich versichere hiermit, dass die elektronische Version meiner Dissertation mit der schriftlichen Version übereinstimmt.

§8 Abs. 1 lit. d PromO

Ich versichere hiermit, dass zu einem vorherigen Zeitpunkt noch keine Promotion versucht wurde. In diesem Fall sind nähere Angaben über Zeitpunkt, Hochschule, Dissertationsthema und Ergebnis dieses Versuchs mitzuteilen.

§9 Abs. 1 PromO

Ich versichere hiermit, dass die vorliegende Dissertation selbstständig und nur unter Verwendung der angegebenen Quellen verfasst wurde.

§9 Abs. 2 PromO

Die Arbeit hat bisher noch nicht zu Prüfungszwecken gedient.

Darmstadt, July 12, 2023

_____

Muhammad El-Hindi

Abstract

Modern trends like digitization and data ecosystems, accelerated by recent events such as COVID-19, necessitate a shift from isolated data management in silos to more open models in which organizations process and share data across organizational boundaries. This transition, however, spawns interdependencies among organizations and generates unique challenges for data management, including data integrity, auditability, and regulatory compliance.

Addressing these novel requirements presents significant challenges for traditional data systems such as database management systems. These systems were designed under the assumptions of a single organization owning and managing data, not considering shared data access by multiple parties. Hence, this dissertation explores the concept and development of trustworthy data systems designed to address the unique demands of managing data across multiple organizations. Nevertheless, creating efficient, trustworthy data systems poses several challenges, which are examined through the lens of three main dimensions: data storage, processing, and benchmarking. In this thesis, we provide an overarching analysis of the requirements of data systems in these areas and dive deep into fundamental building blocks from a performance-centric perspective.

The concept of trustworthy data storage is investigated within our novel system, BlockchainDB. It addresses the requirements of data integrity and auditability by leveraging blockchains as a storage backend. However, to mitigate the performance limitations of blockchains and facilitate a user-friendly data interaction, we introduce an additional database layer that utilizes techniques like sharding. To advance the efficiency of trustworthy data storage, we also inspect the performance limitations of Merkle Trees, a key data structure for integrity in many systems such as blockchains. We find that Merkle Trees suffer from significant performance limitations when data is frequently updated and propose techniques to improve both throughput and scalability.

Addressing the novel requirements of trustworthy data processing, this work presents the system TrustDBle. By integrating blockchains and secure hardware such as Intel's Software Guard Extensions (SGX), this system efficiently ensures policy adherence and

computational integrity. Recognizing the constraints SGX faces with large data volumes, we propose incorporating only critical components within an enclave, thereby balancing efficiency and integrity. Additionally, the dissertation explores the capabilities and limitations of Intel's second-generation SGX technology (SGXv2) in supporting data-intensive applications. By doing so, we find that SGXv2 improves upon its predecessor and can handle larger data volumes more efficiently, but new issues like remote NUMA access need attention. This research also extends beyond traditional database workloads and explores trustworthy data processing within a federated learning context, proposing a decentralized parameter server architecture that provides robust privacy protection.

In the context of benchmarking trustworthy data systems, this dissertation provides a two-fold contribution. In analogy to the ACID properties, particularly isolation levels, it advocates for declarative expressions of system properties like verifiability, enhancing user understanding and implementation flexibility. Secondly, it introduces a holistic benchmark design for these systems, incorporating traditional performance metrics and novel elements like verifiability and auditability.

The research encapsulated in this dissertation paves the way for efficient, trustworthy data management across organizations. The presented insights and techniques create promising opportunities for adoption by the broader data management community and other systems for cross-organizational collaboration.

Zusammenfassung

Moderne Trends wie Digitalisierung und Datenökosysteme, beschleunigt durch jüngste Ereignisse wie COVID-19, erfordern eine Abkehr von isolierter Datenverwaltung in Silos hin zu offeneren Modellen, in denen Organisationen Daten über Organisationsgrenzen hinweg verarbeiten und teilen. Dieser Ansatz erzeugt jedoch Abhängigkeiten zwischen den Organisationen und führt zu besonderen Herausforderungen für das Datenmanagement, einschließlich Datenintegrität, Prüfbarkeit und regulatorischer Konformität.

Die Bewältigung dieser neuen Anforderungen stellt herkömmliche Datensysteme wie zum Beispiel Datenbankmanagementsysteme vor erhebliche Herausforderungen. Diese Systeme wurden unter der Annahme entwickelt, dass eine einzige Organisation Daten besitzt und verwaltet, ohne die gemeinsame Datennutzung durch mehrere Parteien zu berücksichtigen. In dieser Dissertation wird daher das Konzept und die Entwicklung vertrauenswürdiger Datensysteme untersucht, die auf die besonderen Anforderungen der Datenverwaltung in mehreren Organisationen zugeschnitten sind. Die Entwicklung effizienter, vertrauenswürdiger Datensysteme birgt jedoch einige Herausforderungen, die anhand von drei Hauptdimensionen untersucht werden: Datenspeicherung, Datenverarbeitung und Benchmarking. In dieser Arbeit wird eine übergreifende Analyse der Anforderungen an Datensysteme in diesen Bereichen vorgenommen und die grundlegenden Bausteine aus einer leistungsorientierten Perspektive beleuchtet.

Das Konzept der vertrauenswürdigen Datenspeicherung wird in unserem neuartigen System, BlockchainDB, untersucht. Es erfüllt die Anforderungen an Datenintegrität und Nachvollziehbarkeit, indem es Blockchains als Speicher-Backend nutzt. Um jedoch die Leistungseinschränkungen von Blockchains zu mildern und eine benutzerfreundliche Dateninteraktion zu ermöglichen, führen wir eine zusätzliche Datenbankschicht ein, die Datenbank-Techniken wie Sharding nutzt. Um die Effizienz der vertrauenswürdigen Datenspeicherung zu verbessern, untersuchen wir auch die Leistungseinschränkungen von Merkle-Bäumen, einer Schlüssel-Datenstruktur für Datenintegrität in vielen Systemen wie Blockchains. Wir stellen fest, dass Merkle-Bäume erhebliche Leistungseinschränkungen aufweisen, wenn Daten häufig aktualisiert werden und schlagen Techniken zur Verbesserung von Durchsatz und Skalierbarkeit vor.

Um die neuen Anforderungen an vertrauenswürdige Datenverarbeitung zu adressieren, stellt diese Arbeit das System TrustDBle vor. Durch die Integration von Blockchains und sicherer Hardware wie den Software Guard Extensions (SGX) von Intel gewährleistet dieses System effizient die Einhaltung von Richtlinien und die Integrität von Berechnungen. Aufgrund der Kapazitätseinschränkungen von SGX integrieren wir in unserem Ansatz nur

kritische Komponenten in einer Enklave, um so Effizienz und Integrität auszubalancieren. Zusätzlich untersucht die Dissertation die Fähigkeiten und Grenzen der SGX-Technologie der zweiten Generation (SGXv2) von Intel bei der Unterstützung datenintensiver Anwendungen. Dabei stellen wir fest, dass SGXv2 wesentliche Verbesserungen einführt und größere Datenmengen effizienter verarbeiten kann, aber neue Herausforderungen wie der remote NUMA-Zugriff Aufmerksamkeit erfordern. Diese Forschung geht auch über traditionelle Datenbank-Workloads hinaus und erforscht die vertrauenswürdige Datenverarbeitung im Kontext des föderierten maschinellen Lernens. Hierbei erarbeiten wir eine dezentralisierte Parameterserver-Architektur, die einen robusten Schutz der Privatsphäre bietet.

Im Zusammenhang mit dem Benchmarking vertrauenswürdiger Datensysteme leistet diese Dissertation zwei Beiträge. In Analogie zu den ACID-Eigenschaften, insbesondere den Isolationsstufen, schlagen wir ein deklarative Angabe von Systemeigenschaften wie Nachprüfbarkeit vor, um das Benutzerverständnis und die Implementierungsflexibilität zu verbessern. Zweitens führen wir ein ganzheitliches Benchmark-Design für diese Systeme ein, das traditionelle Leistungsmetriken und neuartige Elemente wie Nachprüfbarkeit und Prüfbarkeit einbezieht.

Die in dieser Dissertation vorgestellten Forschungsergebnisse ebnen den Weg für effizientes, vertrauenswürdiges Datenmanagement über Organisationen hinweg. Die vorgestellten Erkenntnisse und Techniken bieten vielversprechende Möglichkeiten für deren Anwendung in der breiteren Datenverwaltungslandschaft und andere Systeme für die organisationsübergreifende Zusammenarbeit.

Publications

The following peer-reviewed publications are part of this cumulative dissertation. Their content is printed in Part II, Chapters 8 - 15.

[1] Muhammad El-Hindi, Martin Heyden, Carsten Binnig, Ravi Ramamurthy, Arvind Arasu, and Donald Kossmann. "BlockchainDB - Towards a Shared Database on Blockchains." In: *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*. Ed. by Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska. ACM, 2019, pp. 1905–1908. DOI: 10.1145/3299869.3320237. URL: https://doi.org/10.1145/3299869.3320237.

[2] Muhammad El-Hindi, Carsten Binnig, Arvind Arasu, Donald Kossmann, and Ravi Ramamurthy. "BlockchainDB - A Shared Database on Blockchains." In: *Proc. VLDB Endow.* 12.11 (2019), pp. 1597–1609. DOI: 10.14778/3342263.3342636. URL: http://www.vldb.org/pvldb/vol12/p1597-el-hindi.pdf.

[3] Muhammad El-Hindi, Simon Karrer, Gloria Doci, and Carsten Binnig. "TrustDBle: Towards Trustable Shared Databases." en. In: *3rd International Symposium on Foundations and Applications of Blockchain*. 2020. URL: https://scfab.github.io/2020/FAB2020_p7.pdf.

[4] Muhammad El-Hindi, Zheguang Zhao, and Carsten Binnig. "ACID-V: Towards a New Class of DBMSs for Data Sharing." In: *Heterogeneous Data Management, Polystores, and Analytics for Healthcare - VLDB Workshops, Poly 2021 and DMAH 2021, Virtual Event, August 20, 2021, Revised Selected Papers*. Ed. by El Kindi Rezig, Vijay Gadepally, Timothy G. Mattson, Michael Stonebraker, Tim Kraska, Fusheng Wang, Gang Luo, Jun Kong, and Alevtina Dubovitskaya. Vol. 12921. Lecture Notes in Computer Science. Springer, 2021, pp. 60–64. DOI: 10.1007/978-3-030-93663-1_5. URL: https://doi.org/10.1007/978-3-030-93663-1_5.

[5] Muhammad El-Hindi, Tobias Ziegler, Matthias Heinrich, Adrian Lutsch, Zheguang Zhao, and Carsten Binnig. "Benchmarking the Second Generation of Intel SGX Hardware." In: *International Conference on Management of Data, DaMoN 2022, Philadelphia, PA, USA, 13 June 2022*. Ed. by Spyros Blanas and Norman May. ACM, 2022, 5:1–5:8. DOI: 10.1145/3533737.3535098. URL: https://doi.org/10.1145/3533737.3535098.

[6] Muhammad El-Hindi, Zheguang Zhao, and Carsten Binnig. "Towards Decentralized Parameter Servers for Secure Federated Learning." In: *Proceedings of the 11th International Conference on Data Science , Technology and Applications, DATA 2022, Lisbon, Portugal, July 11-13, 2022*. Ed. by Alfredo Cuzzocrea, Oleg Gusikhin, Wil M. P. van der Aalst, and Slimane Hammoudi. SCITEPRESS, 2022, pp. 257–269. DOI: 10.5220/0011146300003269. URL: https://doi.org/10.5220/0011146300003269.

[7] Muhammad El-Hindi, Ashwin Arora, Simon Karrer, and Carsten Binnig. "Towards a Benchmark for Shared Databases [Vision Paper]." In: *Datenbank-Spektrum* 22.3 (2022), pp. 227–239. DOI: 10.1007/s13222-022-00429-8. URL: https://doi.org/10.1007/s13222-022-00429-8.

[8] Muhammad El-Hindi, Tobias Ziegler, and Carsten Binnig. "Towards Merkle Trees for High-Performance Data Systems." In: *Proceedings of the 1st Workshop on Verifiable Database Systems, VDBS '23, Seattle, WA, USA, June 23, 2023*. Ed. by Tien Tuan Anh Dinh, Beng Chin Ooi, and Xinying Yang. ACM, 2023, pp. 28–33. DOI: 10.1145/3595647.3595651. URL: https://doi.org/10.1145/3595647.3595651.

Due to the nature of the synopsis and for better readability, selected paragraphs from these publications were transferred verbatim throughout the synopsis without explicit labeling as suggested in the department regulations "Kumulative Dissertation und Eigenzitate in Dissertationen" (21.09.2021) §1.

Acknowledgments

In the name of God, the Most Gracious and the Most Merciful. I thank God Almighty for all His graces and blessings during my doctoral journey.

First and foremost, I owe a debt of gratitude to Prof. Dr. Carsten Binnig for his exceptional support and guidance throughout the various stages of my research. His mentorship has continually pushed me to evolve both personally and academically. I am indebted for his unwavering availability whenever I sought academic or personal advice. His wisdom, technical knowledge, and visionary outlook not only guided my research but also inspired my decision to pursue this Ph.D. I feel fortunate to have had him as my advisor.

I extend my sincere thanks to my co-assessor, Prof. Amr El Abbadi, for generously dedicating his time and effort to review this dissertation and serve on my defense committee.

My journey at the Systems Group at TU Darmstadt was richly augmented by the invaluable contributions of my colleagues. The past years have been both enjoyable and productive, thanks to the supportive environment, the exchange of innovative ideas, engaging discussions, and the feedback that inspired growth. To each one of you – thank you. A special acknowledgment goes to my office comrades – Tobi, Matthias, Lasse, and Nils – who have been the best coworkers one could wish for. Extra special thanks are reserved for Mona, whose invaluable support – both professional and personal – helped me navigate through various challenges, administrative and otherwise, over the years.

I am fortunate to have collaborated with exceptional individuals on various projects. Working with them has been a privilege, and their expertise and guidance have significantly contributed to my academic growth. Thank you all for the great collaboration and team efforts!

Last but certainly not least, my deepest gratitude goes to my parents, my wife Asma, and my siblings, who have provided unyielding support and shown remarkable patience throughout this long journey.

Contents

Acronyms

2PC two-phase-commit

BC blockchain

CCPA California Consumer Privacy Act

CDPA Consumer Data Protection Act

DB database

DBMS database management system

DLT distributed ledger technology

EPC Enclave Page Cache

EU European Union

GDPR General Data Protection Regulation

MEE Memory Encryption Engine

ML Machine Learning

NN neural network

NUMA Non-Uniform Memory Access

OLAP Online Analytical Processing

OLTP Online Transaction Processing

PRM Processor Reserved Memory

PS parameter server

RBAC role-based access control

SGD Stochastic Gradient Descent

SGX Software Guard Extensions

SUT system under test

SW Software

TEE Trusted Execution Environment

TDX Trust Domain Extensions

TME Total Memory Encryption

TPM Trusted Platform Module

TX transaction

UCE UPI Crypto Engine

UPI Ultra Path Interconnect

Part I

Synopsis

1 Introduction

1.1 Context & Motivation

Cross-organizational data management. Recent events, such as the COVID-19 pandemic, have catalyzed a rapid digital transformation across various sectors, including public and private organizations [7, 150]. This trend has been a major driving force for organizations to process and share data across organizational boundaries and can be observed in two major developments.

First, there has been a shift in how we operate and use data management systems, specifically database management systems (DBMSs). In the past, organizations deployed these systems within their own local data centers. Now, there is a growing demand[1] for cloud-based DBMSs, managed by third-party providers in offsite data centers [41].

Second, there is a move away from data silos, which kept data locked within one organization. Instead, initiatives like the European Data Strategy [61] advocate for sharing data across organizations to form integrated data ecosystems. This cross-organizational data integration, evident in industries like manufacturing, allows organizations to directly access and utilize data from other entities, optimizing business processes [133, Chapter 1, p. 3]. While manufacturing is a prominent example, the benefits of inter-organizational data management extend to sectors like supply chain [32, 83, 138, 142], healthcare [78] or finance [37, 62]. This represents a significant shift towards a new era in data management, characterized by integrated data storage and processing to enhance existing services or develop novel ones [70].

The need for trustworthy data management. However, when data is processed and shared across boundaries, concerns over the influence of external parties handling the shared data arise. This interdependency among organizations creates new requirements for regulating and governing the data management process. A prominent example is the General Data Protection Regulation (GDPR) of the European Union [60] that regulates

[1] https://blogs.gartner.com/merv-adrian/2022/04/16/dbms-market-transformation-2021-the-big-picture/

how the data of European citizens have to be processed. These regulations impose strict transparency and accountability requirements, ensuring that policy violations or data corruption can be identified, and the party responsible can be held accountable. Especially when multiple parties are involved in data processing, ensuring the accountability and liability of all involved parties is a vital concern [64].

Upholding these regulations and ensuring the required transparency present significant challenges for traditional DBMSs. These systems were designed with the assumption that data is owned and managed by a single organization, not considering shared data access by multiple parties. In fact, many DBMSs tend to "lock in" the data, making it complicated to extract and share[2].

This sets the context for this work, which introduces the concept of *trustworthy data management systems*. In contrast to traditional DBMSs, these novel systems are designed to process and share data across the boundaries of different organizations in a trustworthy manner. They establish trust by confirming compliance with agreed-upon policies and offering transparency into all data interactions, enabling detection of policy breaches or data corruption. As mentioned earlier, incorporating these new requirements in DBMSs is not a straightforward task and introduces several issues, which will be discussed in the subsequent section.

1.2 Problem Statement & Challenges

In the past, the focus in industry and academia has been on storing and processing large amounts of data efficiently. However, with the rise of cross-organizational data management, new requirements for the trustworthy management of data have emerged. Consequently, this thesis re-evaluates several aspects of traditional data systems in the pursuit of developing trustworthy data management practices. Thereby, several challenges that center around the trustworthy *storage* and *processing* of data arise:

- **Ensuring trustworthiness of data storage:** A main issue of trustworthy data management is to protect the *integrity of the data* itself. This guarantee implies safeguarding data from any unlawful modification. For example, even privileged administrators should be unable to modify data directly, even if they have physical access to the underlying storage device.

[2] https://bryteflow.com/sap-s-4-hana-overview-and-5-ways-to-extract-s-4-erp-data/

Additionally, trustworthy data storage plays a major role in addressing the *auditability and transparency* requirements of trustworthy data management. These two properties are important in settings where multiple organizations process data collaborative since they ensure the required accountability and liability. For example, whenever data is updated it is important to track by whom this change was performed. A major challenge in this context is storing all relevant information (metadata) so that all accesses or changes of the data can be audited and any misbehavior can be traced to the responsible party.

- **Ensuring trustworthiness of data processing:** When data management spans organizational boundaries, it is important for organizations to stay in control over the access, manipulation, and use of their data. This control is in some cases even mandated by legal regulations, such as for the processing of health data, where many legal regulations exist concerning who is allowed to access which data. To ensure compliance with such regulations or company internal standards, organizations need to be able to define *policies* which ensure that data can only be processed according to a pre-defined logic that even system administrators cannot alter.

  Facilitating this policy based processing in data systems not only demands research on how the policies are expressed and processed. It further requires to ensure the *integrity of computation*, i.e., confirming that the agreed-upon logic is correctly executed, preventing breaches that might involve deviation or manipulation of the processing logic. As we will discuss in more detail in Chapter 2, such strong guarantees were not applicable in classical data systems, since data was only processed within a single organization.

Addressing these challenges efficiently is a challenging task, especially when attempting to maintain high performance and minimize any degradation in system speed or responsiveness. For example, ensuring the integrity of data involves using cryptographic data structures such as Merkle Trees [121] that introduce new performance limitations. Hence, it is a goal of this thesis is to identify such bottlenecks and devise *efficient* paradigms and techniques for *trustworthy* data management across multiple organizations. In the following, we briefly summarize the contributions made by this work and defer a more detailed discussion of the challenges to the subsequent chapters.

1.3 Contributions

This thesis aims to contribute towards building novel data systems for efficient and trustworthy data management. To achieve this objective, several contributions have been made that revolve around high-performance trustworthy data storage and processing. Additionally, since trustworthy data systems come with unprecedented requirements in addition to performance, we propose concepts for benchmarking these systems more holistically. This initiative facilitates the evaluation and comparison of different trustworthy data processing systems.

1.3.1 Trustworthy Data Storage

As we will discuss further in Chapter 4, the core requisites for trustworthy data storage are data integrity protection and auditability. The former ensures that data modifications are only permissible through direct interactions with the data processing system, effectively safeguarding against any unauthorized alterations. Auditability, on the other hand, enables a system to render a transparent account of all data interactions such that any data change can be traced to the responsible party.

Using blockchains as trustworthy storage. Our first contribution explores the potential of blockchain technology to fulfill the prerequisites of trustworthy data storage. Although blockchain technology inherently offers robust data integrity protection and auditability, its usage is limited due to significant performance and scalability constraints.

To bridge this performance gap, we propose an enhancement to the conventional blockchain architecture by incorporating an additional database layer. This layer leverages traditional database technologies, such as sharding[3], to enhance the transaction processing capability of blockchains. Sharding distributes the storage load across multiple subdivisions or "shards", each representing an independent blockchain, to improve overall system performance while retaining the data integrity protection characteristics of traditional blockchains. Since our sharding approach circumvents the full replication of data common to blockchains, it requires additional verification mechanisms that we introduce in our work to achieve the same level of trustworthiness.

High-performance Merkle Trees for trustworthy storage. Diving deeper into the performance bottleneck of blockchains, we identify that Merkle Trees, a fundamental data structure used in blockchains, face considerable performance limitations on modern multi-core machines. In our second contribution, we tackle these performance limitations and

[3] In this thesis we use *sharding* and *partitioning* as synonyms.

enhance the performance of Merkle Trees. We introduce novel techniques to construct high-performance Merkle Trees, including a new latching scheme that improves concurrency for update workloads.

1.3.2 Trustworthy Data Processing

Trustworthy data processing brings to light two major issues: *policy based processing* and *ensuring computational integrity*. In Chapter 5, we introduce three main contributions addressing these issues in the context of both database and Machine Learning (ML) workloads.

Hybrid architecture for trustworthy processing. Building on our contributions towards trustworthy data storage, we introduce TrustDBle, a novel system for trustworthy data processing. This system amalgamates the strengths of secure hardware and blockchain technology. Secure hardware, like Intel Software Guard Extensions (SGX), provides isolated execution environments, or *enclaves*, to perform sensitive computations. Our system uses this technology to ensure computational integrity and enforce data access policies, while blockchain technology provides a trustworthy storage environment. However, as Intel SGX faces capacity limitations for processing large volumes of data, we propose to only place the most critical components for ensuring the transactional guarantees inside the Trusted Execution Environment (TEE).

Secure hardware for trustworthy processing. Secure hardware and enclaves play an instrumental role in trustworthy processing as they protect data and code from external interference. However, the first version of Intel SGX has known performance issues with real-world workloads, such as database transactions. Recognizing this, we conduct an in-depth performance evaluation of the most recent Intel SGX version (SGXv2), particularly in the context of database workloads.

Trustworthy processing of ML workloads. Beyond the classical scope of database workloads, we extend our exploration of trustworthy data processing to non-database workloads, such as machine learning tasks. We demonstrate how sharding can be utilized to implement trustworthy data processing techniques, mitigating potential threats from a curious parameter server in federated machine learning settings.

1.3.3 Benchmarking Trustworthy Data Systems

The growing emphasis on trustworthy data management has given rise to numerous innovative systems designed to address this concern, as we will discuss in Chapter 6. However, these emerging systems vary significantly in their implementation of the novel

requirements and the provided guarantees. For example, we observe differences across systems in how clients can *verify* that essential guarantees, such as data integrity, are being upheld.

Incorporating verification to ACID. Recognizing this variability, we propose ACID-V, a new transactional model designed to embed verification into the traditional ACID (Atomicity, Consistency, Isolation, Durability) transaction model of DBMSs. The ACID model ensures consistent processing of database transactions, and the added *Verification* component in ACID-V allows users to verify whether a transaction was executed in accordance with the ACID properties and any defined policies. By establishing this model, we offer a generic framework that can be utilized to compare the verifiability levels offered by different systems.

Benchmarking trustworthy DBMSs. Further enhancing the comparison of trustworthy data management systems, we introduce a unique benchmark design that encompasses the distinct attributes of these systems. Classic benchmarks often prioritize the performance of data systems, and hence fall short when evaluating novel aspects such as verifying data integrity. Motivated by this observation, our benchmark design introduces new metrics and workloads as an extension to existing performance-oriented benchmarks. These metrics and workloads evaluate both the performance and trustworthiness of data systems, providing a more comprehensive evaluation of these systems. Therefore, our proposed benchmark design sets the foundation for a more holistic appraisal of systems built for trustworthy data management.

1.4 Outline

The upcoming chapters delve more deeply into the contributions in each of the above areas. Before diving into these detailed examinations, we first provide a brief review of the related work in Chapter 2. This discussion highlights the previous landscape of data systems and the pressing need for trustworthy data management solutions.

In Chapter 3, we detail the research methodology that guided this dissertation's inquiries, thereby establishing a clear path for the ensuing discussions.

We start by discussing trustworthy data storage in Chapter 4, followed by trustworthy data processing in Chapter 5 and benchmarking trustworthy data systems in Chapter 6. The synopsis concludes with a summary of the results and an outlook of future research directions in Chapter 7.

2 Related Work

The protection of data and its governance – particularly in terms of storage and processing – has remained an integral aspect of data systems since their very beginnings. Yet, in recent decades, the landscape has significantly evolved, presenting shifts in the underlying security and trust assumptions that distinguish the unique research featured in this dissertation. In this chapter, we categorize relevant work broadly based on the underlying trust assumptions and briefly sketch the techniques used to protect and govern data in these systems. A more in-depth discussion of these works will be conducted in the individual contributions of Part II of this dissertation.

2.1 Trust Assumptions in Data Systems

As illustrated in Figure 2.1, databases (DBs) were initially designed for use by a single, trusted organization (left). However, the shift toward outsourced DB settings (middle) moved the database outside off an organization's controlled environment and introduced the database provider as an external entity responsible for database system administration. This trend of introducing more external entities that interact with a data system, continues with trustworthy data systems (right) which now confront trust issues among multiple data owners.

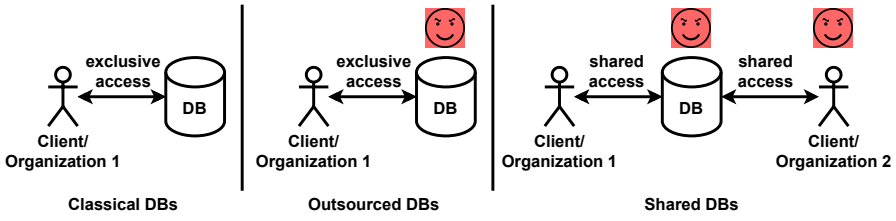


Figure 2.1: The evolution of trust assumptions leads to a spectrum along which related work can be positioned.

The evolving landscape of data systems forms a spectrum for classifying a diverse range of related works, techniques, and systems.

Classical data management systems. As mentioned earlier, classical data management systems assumed that they are run and operated in a controlled and secure environment. By simply running them within an organization's secure perimeter, it was possible to isolate these systems from most external threads (e.g., access from unauthorized external parties).

The main concern in these system was thus to control how users of a single organization can access and interact with the data and the database system. For that purpose, concepts like role-based access control (RBAC) [21, 147] emerged to manage trust issues within an organization. RBAC assigns access rights to specific roles, and users are assigned to these roles. This approach confines trust within organizational boundaries, controlling data access based on roles. Nevertheless, it does not inherently address the complex issues associated with cross-organizational data sharing and processing as discussed in the previous chapter.

Federated data management systems. The concept of sharing data across multiple DBMSs was introduced in the mid-1980s with federated database systems[153]. These systems logically integrate multiple autonomous databases into a single virtual system.

Nonetheless, like classical data management systems, federated databases operate under the assumption that all connected systems are trusted entities. A prime example for their use case are different business units that store data in different relational and non-relational data platforms, but need to maintain a global view over the data of the organization. In other words, they aim to create an abstraction for managing and processing data of heterogeneous systems in a globally consistent manner rather than directly addressing cross-organizational trust issues.

Outsourced databases and data management. The advent of cloud computing led to a surge in outsourcing database management to third-party organizations, responsible for managing the necessary infrastructure and software components [2]. This shift potentially exposed data to unauthorized access by rogue administrators without the data owner's consent. Despite this, administrators were still entrusted to operate databases and process transactions correctly.

This setup led to the development of encrypted outsourced databases [141]. These systems aim to protect the privacy of client data, especially from untrusted database operators. This is accomplished by encrypting data at rest, in transit, and during processing [12]. While these systems primarily focus on data protection from administrators,

they continue to operate under the assumption that a single organization exclusively uses the hosted database.

Blockchain systems. The rise of blockchain technology and distributed ledger systems brought about a radical shift in trust assumptions. Systems like Bitcoin [126] or Ethereum [29] propose a decentralized setup in which multiple entities cooperate, each with potentially varying interests. Blockchain systems, unlike outsourced DBMSs, eliminate the need for a single entity to host and manage data centrally.

Moreover, the decentralized setup of these systems enables entities with potentially conflicting interests to collaborate under challenging failure models, including malicious behavior. As such, they have been instrumental in the evolution of trustworthy data management systems, particularly regarding security and potential threats.

However, the primary focus of blockchain systems remains on purely decentralized setups. In contrast, the scope of trustworthy data systems extends to include centralized systems such as the recently developed ledger databases [11, 184, 185] or verifiable databases [68, 188]. Although these systems are hosted by a single provider on a centralized platform, they still facilitate data access by multiple organizations and implement several guarantees of trustworthy data systems such as data integrity and auditability.

Additionally, the research on blockchain systems predominantly centers around security-related aspects. Trustworthy data systems also prioritize maintaining trust and security but explore additional efficiency aspects, such as performance, ease of use, and integration with existing data management standards and interfaces, like SQL. Hence, in Chapter 4 we investigate how to combine blockchain technology with common database techniques to get the best of both words.

2.2 Security Techniques in Data Systems

As the previous discussion shows, the scope of enhancing data management systems with additional data protection and governance guarantee has been evolving over the past years. Similarly, the application of security techniques has been explored in various settings.

Encryption. For instance, numerous strategies for ensuring data privacy have been examined in the context of outsourced database systems. These range from methods that merely encrypt data at rest [154] to those using homomorphic encryption [71] to maintain encryption even during processing. Although these works lay a strong foundation for

ensuring data privacy within trustworthy data systems, data privacy protection is not the primary focus of this dissertation.

Differential privacy. The concept of differential privacy [46] is another significant addition to privacy-preserving techniques in the database domain. It provides a statistical approach to privacy, ensuring that the release of statistical information about a database does not compromise the privacy of individual records within that database. This technique is particularly useful in scenarios where aggregate data needs to be shared without revealing sensitive details about individual records [86].

Cryptographic data structures. Cryptographic data structures such as Merkle Trees [121] or cryptographic accumulators [134] have also found their use in data systems. Rather than protecting data from unwanted manipulation, the early use of Merkle Trees and similar structures focused on detecting divergence and consistency issues among multiple nodes of a distributed system. However, with the advent of outsourced databases several of these techniques were adopted in the data management context [174, 191].

Secure Processors. FPGA-based query coprocessor have also been explored in prior work [15, 49], requiring a custom and complex design to achieve the expected guarantees. The introduction of TEE and secure hardware such as Intel SGX is a recent development, which we will discuss in our work on trustworthy data processing in Chapter 5.

While several security techniques exist, their usage in databases or data systems often results in high performance overheads. Hence, researching techniques to overcome these performance limitations in the data management context is a major concern in this thesis.

In summary, this chapter situates the research in this dissertation within the context of related work, underscoring the unique settings and challenges that trustworthy data systems, as introduced in this dissertation, entail. The following chapter will present the methodology followed in this thesis and highlight the common research patterns.

3 Methodology

Managing data across organizational boundaries and adapting to the corresponding trust assumptions, as discussed in the previous chapter, introduces novel requirements for building data management systems. To address the complexity resulting from this paradigm shift, we structure our work into several areas, adopting both a broad and detailed research perspective for each.

3.1 Focus Areas

Trustworthy data management penetrates all layers of data systems, ranging from the foundational aspects of data storage to the higher-level considerations of processing and access. To capture the challenges associated with each layer of a data management system, we dissect the data management stack into three core focus areas:

- Trustworthy Data Storage
- Trustworthy Data Processing
- Benchmarking Trustworthy Data Systems

These focus areas not only reflect the fundamental challenges in trustworthy data management but also mirror the contribution areas of this dissertation. Figure 3.1 effectively illustrates this by mapping the dissertation's contributions and publications to each focus area.

For each focus area, our initial exploration is broad before we delve into a specific fundamental building block with a performance-centric focus. We will elaborate on this research pattern in the subsequent section.

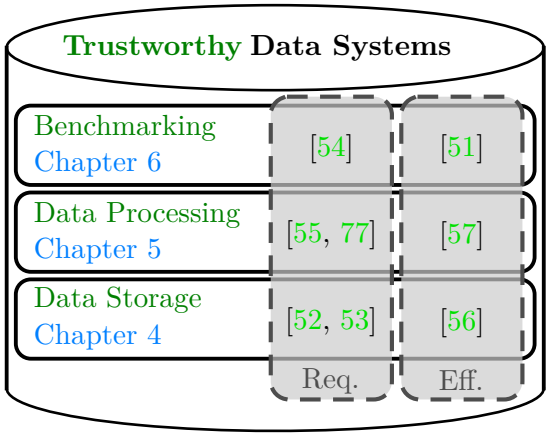


Figure 3.1: Mapping of contributions to focus areas (Storage, Processing, Benchmarking) and research perspectives (Requirements & Efficiency).

3.2 Research Perspectives

This dissertation advances the exploration of trustworthy data systems for cross-organizational data management. Thus, it is crucial to define the unique requirements and properties of this emerging class of systems, as well as ensure their efficient implementation. To this end, we adopt two research perspectives in each area.

Requirements and properties of trustworthy data systems. Initially, we adopt a high-level perspective in each focus area, proposing a system to meet the unique requirements and properties. In the context of trustworthy data storage, we explore the needs and requirements through our proposed system, BlockchainDB. This system integrates blockchain and database technology to provide a trustworthy storage solution.

Efficient implementations and performance aspects. Next, we dive deeper into the technical details of each focus area, emphasizing efficiency. For example, we recognize the significant role of Merkle Trees in maintaining data integrity within trustworthy data storage. We also identify their potential as performance bottlenecks for update-intensive workloads. Hence, we conduct an in-depth analysis of these performance issues and suggest solutions.

Example. In line with this, we observe that the performance of enclaves can be a limiting factor when considering TrustDBle, our comprehensive system for trustworthy data processing. This insight motivates us to conduct a detailed technical investigation into the performance of Intel's hardware enclave technology.

4 Trustworthy Data Storage

This chapter summarizes the contributions of this dissertation towards realizing the requirements for efficient trustworthy data storage. These contributions span over the following peer-reviewed publications[1]:

- **"BlockchainDB - A Shared Database on Blockchains"**, published in *Proc. VLDB Endow.* 12.11 (2019) [52], (cf. Chapter 8).

  Contributions of the author: Muhammad El-Hindi is the leading author and was thus responsible for the proposed design of BlockchainDB, implementation, evaluation, and the manuscript. The co-authors Carsten Binnig, Arvind Arasu, Donald Kossmann, and Ravi Ramamurthy contributed invaluable feedback. All authors agree with the use of the publication for this dissertation.

- **"BlockchainDB - Towards a Shared Database on Blockchains"**, published in the *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019* [53], (cf. Chapter 9).

  Contributions of the author: Muhammad El-Hindi is the leading author and was thus responsible for the design of the BlockchainDB system and the proposed demonstration as well as for the implementation of both. The co-author Martin Heyden contributed to initial ideas for possible demonstration scenarios that were not included in the publication. The remaining authors Carsten Binnig, Ravi Ramamurthy, Arvind Arasu, and Donald Kossmann contributed invaluable feedback. All authors agree with the use of the publication for this dissertation.

- **"Towards Merkle Trees for High-Performance Data Systems"**, published in the *Proceedings of the 1st Workshop on Verifiable Database Systems, VDBS '23, Seattle, WA, USA, June 23, 2023* [56], (cf. Chapter 10).

  Contributions of the author: Muhammad El-Hindi is the leading author and was thus responsible for the proposed design space analysis, evaluation and the manuscript. The co-author Tobias Ziegler contributed to the initial implementation and evaluation design. The co-author Carsten Binnig contributed invaluable feedback. All authors agree with the use of the publication for this dissertation.

[1] Several passages in this chapter were transferred verbatim from these publications.

4.1 Motivation & Requirements

As discussed in Chapter 1, trustworthy data management introduces several unprecedented requirements for data storage. As an essential contribution of this dissertation, we motivate and distill these requirements in the following. The subsequent section will delve deeper into further contributions to achieve the properties of trustworthy data storage efficiently.

Motivating example. To understand the unique properties of trustworthy data storage, consider the data sharing scenario depicted in Figure 4.1. The example showcases a typical supplier scenario with WholeFoods as the customer, Lindt as the supplier, and FedEx as the shipping company.

In this scenario, WholeFoods first places a new order by inserting two new entries into the shared database consisting of two tables ①. After the order is placed, Lindt processes the new order ②. To keep track of the order, Lindt updates the status from *new* to *ready* as shown in ③. Then, FedEx begins its operation ④ and updates the status to *delivered* after shipping the order ⑤.

A naïve way of implementing such a scenario would be that one of the parties is hosting the shared database; e.g., say WholeFoods hosts the shared database as a service for all their suppliers. However, this gives WholeFoods the potential to manipulate the shared data and return false values about the order status without the other parties being able to verify the actions. For example, WholeFoods could claim that the order was lost during transit by returning a spurious (i.e., false) order status to Lindt as shown in ⑥ to trigger that a replacement is sent (without paying for it). Even worse, WholeFoods could actually delete the order or not store it in the database in the first place. In the case of a lawsuit, no evidence could thus be found that WholeFoods (or any other party) was actually acting maliciously.

This example highlights two fundamental requirements for trustworthy data storage:

Data integrity. In trustworthy data management, data systems should protect the data from unintended or unauthorized modifications. This means even privileged users with physical access to the storage device should not be able to manipulate data secretly. In the above example, for instance, even WholeFoods as an operator of the database with physical access, should not be able to manipulate the data without being detected. Any change in data should only be made through interactions with the data processing system, safeguarding against any hidden alterations. Besides, the system should allow other participants to verify the integrity of retrieved data and ensure that it has not been manipulated.

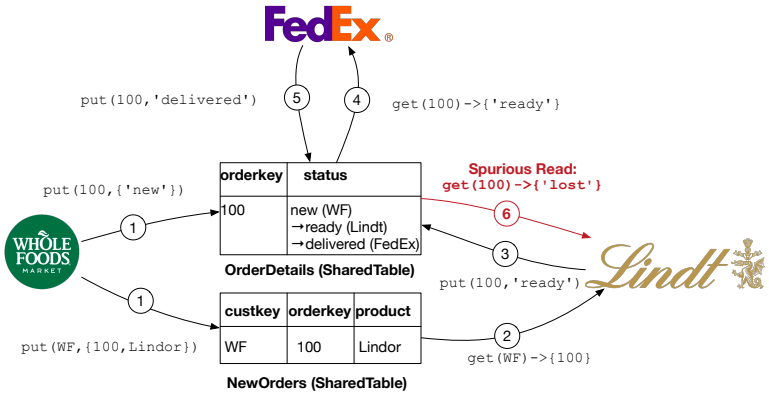


Figure 4.1: A typical data sharing scenario motivating trustworthy data management.

Auditability & transparency. Ensuring auditability and transparency of data access can further protect data. Auditability ensures that all accesses and modifications to the data are recorded and traceable. Transparency, on the other hand, allows participants to verify the current state of the data, assuring that it matches expectations. Both properties play instrumental roles in establishing accountability and liability among parties. In the example above, auditability and transparency would allow all parties involved to verify the actual and expected state of an order. The challenge here is to store all relevant information so that all data interactions can be audited and any misbehavior traced back to the responsible entity.

4.2 Findings & Contributions

Addressing these challenges, this dissertation presents the following two key contributions to advance the field of trustworthy data storage.

4.2.1 Blockchains as Trustworthy Storage

The first contribution is the development of BlockchainDB, a platform that combines blockchain and database techniques to provide the guarantees of trustworthy data storage. An in-depth discussion and a demonstration of BlockchainDB are provided in [52] and [53] respectively (also refer to Chapter 8 and Chapter 9).

While blockchain technology has its origins in cryptocurrencies, its potential extends far beyond this domain, finding applications across governmental, healthcare, and IoT scenarios [16, 20, 169]. A common thread running through these diverse applications is

the utilization of blockchains to facilitate shared data access among distrusting entities. This is made possible due to two inherent properties of blockchain technology. First, it retains a tamper-evident, append-only ledger, encapsulating a full history of data modifications, thereby enabling auditability and traceability. Second, it obviates the need for a central trust authority, functioning in a reliable, decentralized manner.

However, despite these merits, the practical adoption of blockchains is inhibited by significant performance and scalability constraints. For instance, a typical blockchain platform can only process several hundred to thousands of transactions per second, while traditional databases can handle millions of transactions per second [44]. Additionally, blockchains lack the convenience and ease-of-use offered by conventional data management systems, including a straightforward query interface and well-defined consistency levels.

BlockchainDB addresses these limitations by leveraging blockchains as an underlying tamper-proof storage layer and implementing a database layer on top to provide user-friendly data access. The database layer of BlockchainDB imparts a set of functions:

- *Partitioning and Partial Replication:* A major performance bottleneck of blockchains today is that all peers of a blockchain network hold a full copy of the state and still only provide (limited) sharding capabilities. In the database layer of BlockchainDB, we allow applications to define how data is replicated and partitioned across all available peers. Thus, applications can trade performance and security guarantees in a declarative manner.
- *Query Interface and Consistency:* In the DB layer, BlockchainDB additionally provides shared tables as easy-to-use abstractions including different consistency protocols (e.g., eventual and sequential consistency) as well as a simple key/value interface to read/write data without knowing the internals of a blockchain system.

BlockchainDB also facilitates an *off-chain verification procedure* allowing peers to verify the read- and write-set of their own clients. This is crucial as not all peers maintain a full database copy, and a remote peer could potentially drop updates or return spurious reads (i.e., values not previously persisted in the database).

By deploying a database layer on top of existing blockchains, BlockchainDB not only simplifies the adoption process for organizations, but also enhances performance and scalability. Among others, we illustrate this by evaluating the performance of BlockchainDB against an increasing network size (i.e., as more peers join the network). While traditional blockchains experience a degradation in performance due to increased storage and consensus overhead [44], BlockchainDB mitigates this problem as shown in Figure 4.2.

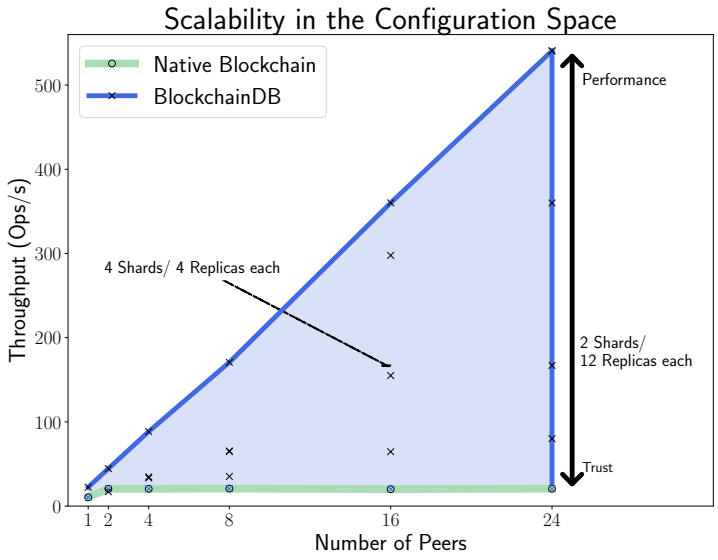


Figure 4.2: Trust vs. performance trade-off in BLOCKCHAINDB. By choosing different sharding and replication strategies BLOCKCHAINDB allows users to navigate the configuration space.

In our experiment, we use a fixed number of peers as shown on the x-axis (i.e., a fixed amount of compute and storage resources) and vary the used sharding configuration. For instance, with a fixed number of 16 peers, we tested configurations ranging from 1 shard with 16 replicas to 16 shards, each with 1 replica. This was repeated for different setups with lower/higher numbers of peers (ranging from $1 - 24$ as indicated on the x-axis). We use a dataset with $64,000$ tuples that are distributed across different shards and peers as given by the configuration. On the y-axis, we report the throughput in ops/sec for a YCSB-like write-only workload, as reported by the different marks in the figure.

Figure 4.2 shows how BLOCKCHAINDB enables applications to customize partitioning and replication strategies, optimizing performance and trust characteristics based on requirements. The two extremes *high trust* and *high performance* are highlighted by the green and blue line, respectively. For the green line, BLOCKCHAINDB only uses one shard to store the data and each newly added peer stores yet another replica of this shard and participates in its corresponding blockchain. Hence, this configuration represents a classical blockchain configuration and shows similarly worse scalability. In the other extreme, *high performance*, every peer that is added also adds a new shard to the network. This configuration is comparable to a classical distributed database, in which the parallelism and throughput of the system is increased with every new node. Yet, since only one replica exists per shard, the application does not get any trust guarantees.

We mitigate this trust issue with the help of our additional verification protocols that allow peers to verify the integrity of data stored in a replica. Using this approach, BlockchainDB also offers a variety of configurations (light blue space) that strike a balance between trust and performance, offering flexibility to applications. For instance, a configuration with 4 shards and 4 replicas each for 16 peers provides a 7× speedup over the full-replicated baseline while still ensuring trustworthiness through our additional verification protocols. The feasibility of such configurations underscores the unique advantages of BlockchainDB.

Clearly, the additional verification protocols also come with their own overheads. An evaluation of the overheads and a more comprehensive discussion of BlockchainDB can be found in Chapter 8 and Chapter 9.

4.2.2 High-Performance Merkle Trees for Trustworthy Storage

In our quest for efficient, trustworthy data storage, we examine the performance limitations of Merkle Trees [121] more closely. This fundamental data structure ensures data integrity in numerous systems, including blockchains and secure outsourced data systems, by assembling a tree of cryptographic hashes derived from the stored data.

Compared to other approaches (such as cryptographic accumulators [167] or signature aggregation [136]), Merkle Trees efficiently support all relevant operations including updates and the verification thereof. This makes them the go-to choice for update-intensive applications, such as secure outsourced key-value stores (e.g., [13, 14, 17, 156]). Using a Merkle Tree as an authenticated data structure, these systems enable clients to store and update data on an untrusted server while guaranteeing the authenticity and integrity of the data. However, they suffer from significant performance limitations when data is frequently updated.

Contrary to classical data structures like B-Trees, which facilitate millions of operations per second, Merkle Trees typically yield only tens of thousands ops/sec [13]. The reasons for this inferior performance are two-fold. First, data updates in Merkle Trees necessitate CPU-intensive operations, such as cryptographic hash computations or digital signatures, imposing a throughput constraint dictated by the CPU speed. Second, enhancing performance through concurrent updates by multiple threads is challenging due to potential conflicts at the root level requiring synchronization, resulting in high contention and performance degradation for highly-parallel modern multi-core systems [13, 14, 17, 156]. Unfortunately, addressing these two issues is not trivial for Merkle Trees.

At first glance, these performance limitations seem inherent to the data structure and thus unavoidable. Hence, previous work mainly proposed alternative data structures [14] or employed secure hardware [13] to circumvent the performance limitations of Merkle Trees. Although their properties make performance optimizations hard, we observe that Merkle Trees still provide unrealized performance potential. For example, while synchronization for classical tree structures [100] is a known research area, efficient and scalable synchronization for Merkle Trees is largely unexplored. Further, while it is not yet possible to make cryptographic operations orders of magnitudes faster, we can exploit workload characteristics to adapt the Merkle Tree to significantly reduce the overhead and thus increase performance.

In this contribution we explore different techniques for building high-performance Merkle Trees. As a first step, we focus on efficient synchronization schemes and propose a new *reverse latch-coupling* scheme to improve concurrency in Merkle Trees. For brevity, we will only summarize the studied synchronization techniques and defer their detailed discussion to Chapter 10.

In addition to our novel reverse latch-coupling strategy, we study the following two latching strategies:

- **Global Latch:** This is the most prevalent approach for synchronizing Merkle Trees today. It is to use a global latch[2] that is maintained for the entire leaf-to-root traversal to prevent conflicts [14, 89]. As only one thread can acquire the exclusive latch required for updating the node, this approach serializes all threads and prevents concurrent operations.

- **Level Latch:** An alternative latching scheme can be derived by increasing the granularity of latches to a latch per level in the Merkle Tree. By utilizing traditional latching coupling [158], this latching scheme creates a chain of threads that traverse the tree level-by-level and allows threads to run in parallel if they are in different tree levels.

Our scheme increases the granularity of latches further and combines fine-grained node latches with latch coupling (aka hand-over-hand latching) [27, 99, 100, 158] to avoid deadlocks, as observed by previous work [89]. The twist of our latching scheme is that it latches the node's parent (exclusively) to update a node. This backward-directed latching effect (hence the name *reverse latching*) effectively also latches the node's sibling(s). The underlying observation is that the inherent data dependency in a Merkle Tree enables

[2]The systems community usually uses the term lock to describe the same mechanism.

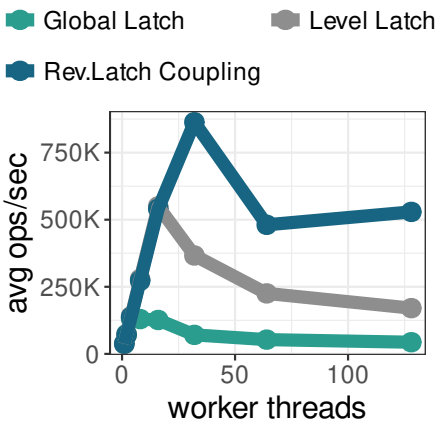


Figure 4.3: Evaluation of synchronization schemes. Reverse latch coupling increases both scalability and performance by around 2× compared to level latching.

us to simultaneously synchronize the (two) children of a node using their parent. I.e., instead of acquiring a latch for each child separately, we can protect the consistency of both nodes by latching their common ancestor first.

We evaluate our novel latching scheme against the previously mentioned alternatives on an update-only workload to demonstrate that an enhanced synchronization technique can indeed improve the concurrency and performance of Merkle Trees. In our benchmark, we execute updates for 8 byte records uniformly on a Merkle Tree with 2^{24} records for 10 seconds and report the system throughput as an average over 5 repetitions of our experiment.

The results, illustrated in Figure 4.3, show that reverse latch coupling improves scalability and performance by an order of magnitude when compared to the global latch strategy. In comparison to the level latch strategy, it increases both scalability and performance by approximately 2×. However, performance degradation occurs after 32 worker threads due to remaining contention effects.

To further reduce the contention and the cryptographic overhead, we propose the concept of *splitting* that enables a lightweight and dynamic sharding of Merkle Trees. We will discuss this concept in more detail in Chapter 10.

5 Trustworthy Data Processing

This chapter summarizes the contributions of this dissertation towards realizing the requirements for efficient trustworthy data processing. These contributions span over the following peer-reviewed publications[1]:

- **"TrustDBle: Towards Trustable Shared Databases"**, published in the *3rd International Symposium on Foundations and Applications of Blockchain* [77], (cf. Chapter 11).

  Contributions of the author: Muhammad El-Hindi is the leading author and was thus responsible for the proposed architecture and concepts as well as the manuscript. The co-author Simon Karrer contributed to the first implementations of the trusted Lock Manager and provided initial experiments. The remaining authors Gloria Doci, and Carsten Binnig contributed invaluable feedback. All authors agree with the use of the publication for this dissertation.

- **"Benchmarking the Second Generation of Intel SGX Hardware"**, published in the *International Conference on Management of Data, DaMoN 2022, Philadelphia, PA, USA, 13 June 2022* [57], (cf. Chapter 12).

  Contributions of the author: Muhammad El-Hindi is the leading author and was thus responsible for the benchmark design, evaluation of SGXv2, and the manuscript. The co-author Tobias Ziegler contributed to the implementation of the B-Tree experiments and provided invaluable feedback on the manuscript. The remaining authors Matthias Heinrich, Adrian Lutsch, Zheguang Zhao, and Carsten Binnig contributed invaluable feedback. All authors agree with the use of the publication for this dissertation.

- **"Towards Decentralized Parameter Servers for Secure Federated Learning"**, published in the *Proceedings of the 11th International Conference on Data Science , Technology and Applications, DATA 2022, Lisbon, Portugal, July 11-13, 2022* [55], (cf. Chapter 13).

[1] Several passages in this chapter were transferred verbatim from these publications.

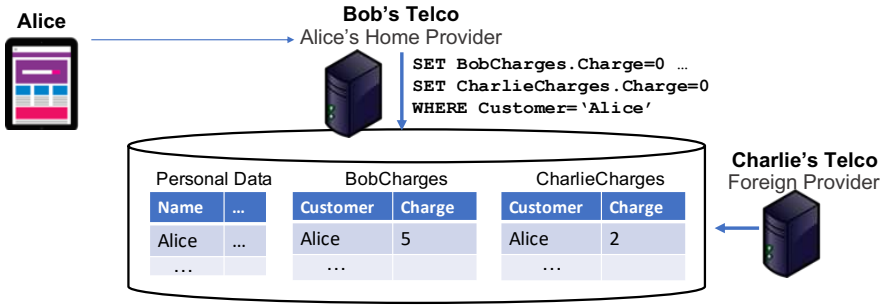


Figure 5.1: Shared DB of two Telco providers motivating the need for trustworthy data processing.

Contributions of the author: Muhammad El-Hindi is the leading author and was thus responsible for the proposed design of the decentralized parameter server, the privacy-preserving configurations, the corresponding implementation, evaluation, and manuscript. The co-authors Zheguang Zhao, and Carsten Binnig contributed invaluable feedback. All authors agree with the use of the publication for this dissertation.

5.1 Motivation & Requirements

Similar to the previous chapter, we first analyze the new prerequisites for trustworthy data processing, followed by a detailed discussion of our contributions to efficiently achieving these properties.

Motivating example. Let us consider a typical telecommunication scenario, where mobile phone users frequently connect to networks outside their home provider's range. Even though users incur charges in these other networks, payments are made solely to their home provider, who settles any outstanding amounts with the secondary provider. An illustration of this scenario is presented in Figure 5.1, where Alice (a user) primarily uses Bob's (home provider) services but also connects to Charlie's (secondary provider) network. By using the abstraction of a shared database, Alice could not only settle her charges with Bob but also Bob could clear the debts of Alice with Charlie using transactions on the shared database.

In such a scenario, it is crucial for the used database management system (DBMS) to ensure that data can only be accessed and modified based on terms agreed by all parties. For instance, when Alice clears her charges with Bob, Bob must settle Alice's debts with Charlie according to the stipulated mobile contract terms.

To provide such guarantees, a trustworthy data system must fulfill the following properties:

Data policies & data sovereignty. As data traverses organizational boundaries, it is crucial that the owning entity retains control over its access, modification, and use. In our example, while multiple telecommunication providers may use the same shared database, Alice should always control who can access her personal data. This sovereignty enables organizations to manage interactions with their data, ensuring intellectual property protection and compliance with legal requirements. For instance, only Alice's home provider should be able to read information such as her personal address.

Integrity of computation. Also referred to as *computation integrity* or *verifiable processing*, this property necessitates that the execution of data management logic, as agreed upon in the policies, is performed accurately, preventing potential breaches due to manipulated transaction logic. This includes the guarantee that all parties correctly execute transactions verifiably. For example, in the given scenario, computation integrity implies that Alice's settlement transaction is accurately executed on both Bob's and Charlie's charge tables. Furthermore, correct and verifiable execution also means that a DBMS can prove that all involved parties executed a transaction in an ACID-compliant way. Clearly, these properties are specific to database transactions, but application-specific requirements also exist in other settings.

5.2 Findings & Contributions

Following our common research pattern, we first adopt a high-level approach to designing a system that implements the guarantees of trustworthy data processing. Then, we proceed with a technical deep dive into a specific building block, in this case, Intel's SGX, to address efficiency and performance concerns. While our initial focus is database workloads, we broaden our scope to include non-database workloads, particularly those related to collaborative machine learning settings, e.g., federated learning.

5.2.1 Hybrid Architecture for Trustworthy Processing

In addressing the requirements for trustworthy data processing, this contribution introduces a system called TrustDBle. TrustDBle aims to satisfy trustworthy data processing by supporting data policies and guaranteeing computation integrity. It extends our previous work on using blockchains as trustworthy data storage by supporting SQL transactions and ACID properties.

Figure 5.2 shows an overview of TrustDBle's architecture. Similar to our previous system, BlockchainDB, we build a database layer on top of blockchains as a storage layer. Thereby, TrustDBle also uses sharding in the storage layer to enable scalability. We also implement further optimization, such as caching in the storage layer to speed up data access. In the following, we focus on extending the database layer with a secure execution engine enabling data policies and ensuring the integrity of computation.

A traditional approach to guaranteeing the integrity of computation is to define smart contracts for the transaction processing logic, e.g., as done in [44]. However, this approach is hampered by the inherent scalability limitations of blockchains and the added complexity of re-implementing new transaction logic as smart contracts. To address this, we propose a secure transaction processing engine outside the blockchain, leveraging a Trusted Execution Environment (TEE) provisioned with Intel Software Guard Extensions (SGX).

Intel SGX establishes TEEs as so-called hardware enclaves protected by the CPU. The CPU provides an enclave with a special address space that is only accessible by the trusted code inside the enclave.

One major limitation of SGX, however, is that it only provides a small portion of protected memory. To overcome this challenge, TrustDBle does not place all transaction execution components in the trusted environment. Instead, we place only those components inside the TEE, which are required to verify that the ACID properties have been fulfilled. For example, to provide verifiable isolation, we only implement the lock manager (LockMgr) of our database layer inside the trusted environment (green box in Figure 5.2).

We use two key criteria to decide which component to implement inside the TEE. First, the component should only maintain a small state inside the protected memory region. In the case of the lock manager, we only need to maintain the lock table of the lock manager in the TEE. Second, verifiable processing must depend on the correct behavior of a node. For other components, such as the transaction coordinator (TxCo), we use verification protocols as done in BlockchainDB to confirm their correct behavior (e.g., attain verifiable atomicity).

TrustDBle's verifiable transaction processing engine currently offers verifiable isolation and atomicity (i.e., A and I of ACID) based on a secure locking scheme and a verifiable two-phase-commit (2PC) protocol for cross-shard transactions.

Our secure locking scheme operates via sharded lock managers in the TEE of a TrustDBle node. Transactions can only access or modify data with a valid lock signed by the LockMgr. Sharding the lock manager enables us to prevent the lock manager from

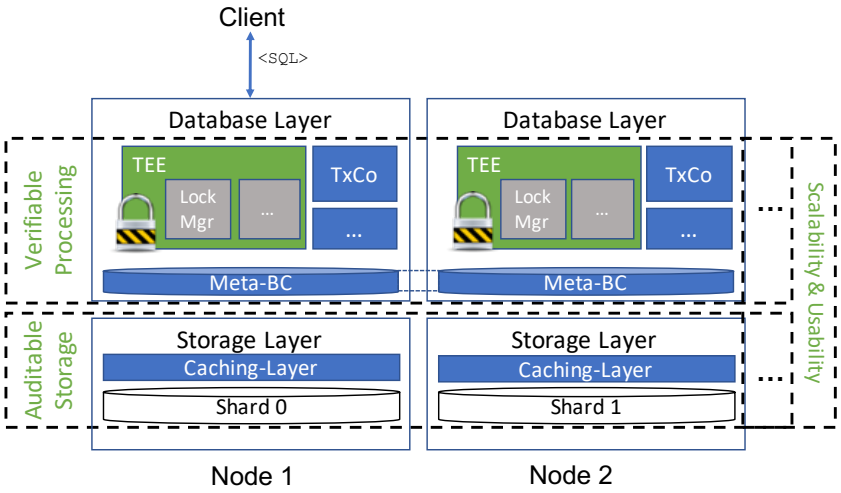


Figure 5.2: TrustDBle Architecture. A hybrid architecture combining TEEs (green box) for verifiable processing and blockchains as auditable storage allows to overcome the capacity limitations of TEEs and the performance overheads of blockchains.

becoming a bottleneck. Availability is achieved by persisting the state of lock managers to auditable storage. This way, other nodes in TrustDBle can recover from a lock manager fault. Moreover, our locking scheme supports the execution of transactions under different isolation levels. To that end, the trusted lock managers enforce that locks are acquired and released according to the specified isolation level.

Similarly, we utilize the auditable storage to log 2PC messages and detect or recover from a faulty transaction coordinator. Thereby, local transaction managers apply a *trust, but verify* strategy with respect to a transaction coordinator: While the transaction coordinator is responsible for collecting and forwarding 2PC messages and decisions to local TxMgrs, each local TxMgr also logs messages to a shared meta-blockchain which all TxMgr can access. Messages in this meta-blockchain are used to verify the decisions of the transaction coordinator. Similarly to our previous work in BlockchainDB, we use deferred verification techniques to mask verification in the case of no failures.

Since TrustDBle combines database, blockchain (BC) technology, and TEEs to overcome the limitations of blockchains and Intel SGX, we evaluate our system with a focus on scalability. Our experiments using the Smallbank OLTP benchmark [30] (see Chapter 11) show that our verifiable processing approach can execute cross-shard transactions in a scalable manner.

5.2.2 Secure Hardware for Trustworthy Processing

Until not too long ago, SGX was exclusively available for consumer-grade CPUs, which are typically less powerful than server-grade CPUs. Coupled with the low core count inherent in these CPUs, SGX enclaves also had other significant technical constraints, such as a restricted memory capacity of up to 256 MB and substantial performance overheads. These capacity restrictions especially limited the application of Intel SGX for DBMSs. In consequence, researchers in the database community started to explore different ways to overcome these limitations, e.g., by employing hybrid architectures as done in TrustDBle or by only placing certain DBMS components inside an enclave [173] or designing enclave-native engines [93, 159].

With the availability of the new server-grade Intel Ice Lake processors [79], Intel promised to address several limitations of SGX enclaves. The latest implementation of Intel SGX on these processors (in the following referred to as SGXv2), not only reduced the overhead of memory protection but also increased the capacity of the protected memory region. This region which is also referred to as Enclave Page Cache (EPC) can be up to 512 GB per socket large, depending on the CPU model [87]. Furthermore, the newly introduced scalability improvements now permit DBMSs using Intel SGXv2 to scale across multiple sockets of high-end servers.

Following our research pattern of diving deeper into certain components to study efficiency and performance issues, we ask whether the previous solutions to overcome SGX limitations for DBMSs are still relevant and if the new generation of SGX processors can deliver on their promise to secure data without compromising performance. To answer this question, we systematically study the performance of Intel SGXv2 and compare it to its predecessor (SGXv1). Specifically, we evaluate SGXv2's performance with typical database workloads while examining multi-core scalability and Non-Uniform Memory Access (NUMA) effects as well as the performance with increasing data set sizes. Furthermore, we present several findings concerning the use of multiple concurrent enclaves in SGXv2 that may arise from running different DBMS instances on the same hardware. As a further contribution, we discuss lessons learned for building the next generation of high-performance enclave-based DBMSs on SGXv2. For brevity of this synopsis, we focus on our findings regarding data scalability, referring to Chapter 12 for more detailed information on other experiments and learned lessons.

In order to evaluate to which extent SGXv2 supports larger amounts of data, we implement a B-Tree index structure as a trusted library inside SGX. We use it to run a YCSB-like workload with 20% inserts and 80% reads to gradually increase the size of

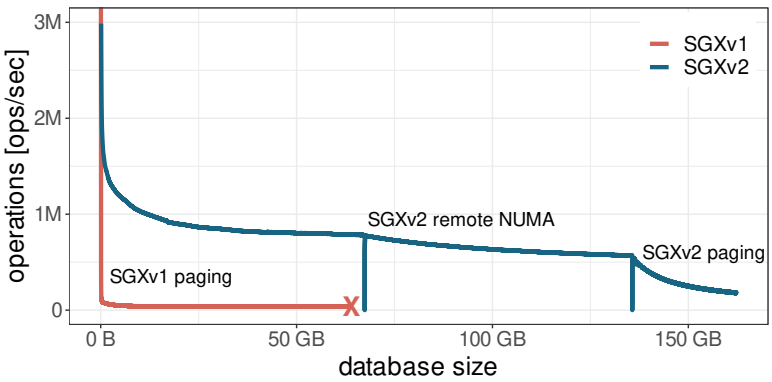


Figure 5.3: Throughput of a 80% read / 20% insert workload in a B-Tree. A significant performance improvement is observable for larger database sizes in SGXv2.

the database. The results of this experiment are shown in Figure 5.3. As expected, the performance of the B-Tree on SGXv1 (red line) drops very quickly (after 128 MB) and only supports a total database size of up to 64 GB due to the corresponding limit on enclave sizes in SGXv1. In contrast, the B-Tree in the SGXv2 enclave (blue line) shows orders of magnitude better performance for much larger database sizes.

Moreover, with SGXv2, we observe two significant reductions in performance as the data size increases: The first performance drop is at around 64 GB, when the data size reaches the capacity limit of the EPC on our first NUMA node. The second performance drop happens at around 128 GB where the capacity of the EPC on the second NUMA node is reached. At this point, similar to SGXv1, we can observe a drastic performance penalty due to paging.

As this and other results in Chapter 12 show, SGXv2 delivers on its promise to increase enclave capacity. Depending on the chosen CPU, its enclave capacity is two to three orders of magnitude larger than SGXv1. In our setup, for example, an in-memory B-Tree can scale up to about 120 GB of data, providing high performance with only around 25% overhead compared to a pure in-memory B-Tree. Conversely, this workload performs 25× worse in SGXv1 due to its limited capacity and the involved paging. The server-grade CPU support provided by SGXv2 also offers additional advantages, including larger cache sizes, increased core counts, and improved scalability across NUMA regions.

However, despite its benefits, using Intel SGX technology still comes with several potential challenges, as detailed in Chapter 12. For instance, memory management needs careful design to accommodate the extra overhead for remote NUMA access. Secondly, different trusted disk I/O implementations can lead to significant performance variations,

requiring a careful choice of libraries and functions. Lastly, SGXv2 does not eliminate certain fundamental performance penalties, like paging, but rather shifts them to larger memory sizes.

5.2.3 Trustworthy Processing of Machine Learning Workloads

Thus far, our focus has primarily been on database workloads and access patterns. Nonetheless, it is crucial to recognize that the principle of trustworthy data processing extends to diverse workloads, such as those encountered in machine learning.

Federated learning (FL) [118] is one instance where this principle is relevant. It provides a collaborative framework for organizations to learn predictive models. Often, individual organizations lack the volume of training data required to train deep models, and FL allows them to pool their resources without the data leaving organizational boundaries [88, 103]. This is particularly useful when the data contains private information that must be safeguarded, such as in the healthcare sector, where hospitals need to share data while complying with stringent legal and privacy regulations [26].

The predominant architecture for federated learning today is to use a central parameter server (PS) that collects the model updates from all participants and aggregates them. In this setup, the data owners participate in the training process by sending their locally computed model updates to the central server, which combines these updates into a global model [118]. The original assumption was that such an approach is able to protect the privacy of each participant's training data, since only model updates and not the training data itself is exchanged with the server. Yet, recent works [193, 196] have shown that privacy attacks exist that allow an attacker to successfully extract information about the training data by observing the model updates (i.e., exchanged gradients). More surprisingly, these attacks showed that it is possible to successfully reconstruct individual training examples with high accuracy (e.g., a full picture used for training) [193, 196]. Even worse, these attacks are also applicable for different model architectures [69, 177]. Therefore, trustworthy processing in this context should prevent such attacks and ensure servers process updates following the agreed logic.

Existing approaches remain limited in preventing privacy attacks in federated learning [1, 23, 69, 139, 196] and thus do not meet the requirements of trustworthy data processing. Most strategies either significantly reduce the learning accuracy (e.g., using noisy gradients can result in 30% less accuracy [196]) or have other limitations such as assuming a trusted central PS, or being incompatible with widely used model architectures (e.g., secure aggregation [23]). Additionally, generic cryptographic primitives, such as homomorphic

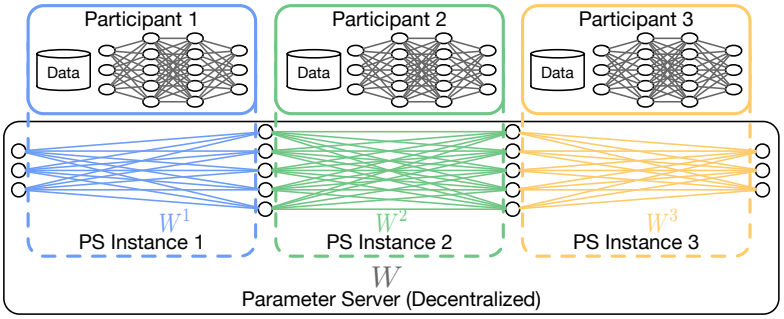


Figure 5.4: Decentralized Parameter Server Architecture for an example with 3 shards (W^1, W^2, W^3). The parameter space W is partitioned across several independent PS instances managed by different participants. In the example, the PS instances are co-located with each participant, but they can also be hosted by independent physical servers.

encryption, increase training times significantly, often up to 100× [139], despite not affecting accuracy.

In this contribution, we take a different, system-driven approach by modifying the federated learning framework. We propose to architect the training system around decentralized parameter servers that follows the concept of trustworthy data processing, as shown in Figure 5.4.

Our approach replaces the centralized PS, a single point of security vulnerability, with a decentralized one. Instead of pooling all model parameters on one PS, we distribute the model – or a *shard* of it – among several independent parameter server instances. Our `P2Sharding` framework thus reduces the potential for privacy invasions by preventing PS instances from having a full view of the gradients or model parameters – both essential for reconstruction attacks.

In our framework, we consider a set of K clients who participate in training a model on their joint data (e.g., $K = 3$ in Figure 5.4). The model W is partitioned into M shards $W^1, \cdots, W^M$ using a configurable strategy, and each shard W^m is hosted on a separate parameter server instance. Each server instance W^m receives the corresponding gradients for its shard from each client k. Each client downloads the full model and iterates $W = (W^1, \cdots, W^M)$ by sending requests (i.e., polling) to the M server instances.

The `P2Sharding` framework assumes all parties to be *semi-honest*, that is, each client and server follow their prescribed protocol and only attempts to extract more information

from the other client's data[2]. Moreover, at least one client and one shard are assumed to be non-colluding with the other parties (i.e., we have at least two non-colluding shards). Otherwise, the security reduces to that of a centralized server.

We also anticipate scenarios involving collusion between a subset of PS instances or between client and server instances. In these cases, the `P2Sharding` framework provides configurations designed to mitigate the risk of both PS-side and client-side attacks. We propose three privacy-preserving techniques to mitigate these attacks: model sharding, asynchronous updates, and polling intervals on stale parameters.

Our model sharding technique distributes the model parameters across all PS instances to minimize the possibility of server-side reconstruction attacks. Configuring the size of shards allows for adjustments to the privacy level in the face of potential collusion attacks. Our framework also includes a parameter for asynchronous updates designed to reduce the risk of collusion between malicious clients and servers. Lastly, we use polling intervals to introduce an element of uncertainty to model iterations for added privacy. A more detailed explanation and discussion of these techniques is provided in Chapter 13.

The evaluation of our framework on the CIFAR10 and MNIST datasets (cf. Chapter 13) demonstrates its effectiveness in thwarting gradient-based reconstruction attacks on deep learning models. These configurations reduce the attack outcome to near-random noise, illustrating the potential of our system-driven approach in ensuring trustworthy data processing in machine learning workloads.

[2]Protocols that assume a semi-honest setup prevent inadvertent leakage of information between parties, and are thus useful if this is the concern. In addition, protocols in the semi-honest model are quite efficient and are often an important first step for achieving higher levels of security.

6 Benchmarking Trustworthy Data Systems

This chapter summarizes the contributions of this dissertation for benchmarking trustworthy data systems more holistically. These contributions span over the following peer-reviewed publications[1]:

- **"ACID-V: Towards a New Class of DBMSs for Data Sharing"**, published in *Heterogeneous Data Management, Polystores, and Analytics for Healthcare - VLDB Workshops, Poly 2021 and DMAH 2021, Virtual Event, August 20, 2021, Revised Selected Papers* [54], (cf. Chapter 14).

  Contributions of the author: Muhammad El-Hindi is the leading author and was thus responsible for developing the proposed transaction model, verification levels and the manuscript. The co-authors Zheguang Zhao, and Carsten Binnig contributed invaluable feedback. All authors agree with the use of the publication for this dissertation.

- **"Towards a Benchmark for Shared Databases [Vision Paper]"**, published in *Datenbank-Spektrum* 22.3 (2022) [51], (cf. Chapter 15).

  Contributions of the author: Muhammad El-Hindi is the leading author and was thus responsible for the analysis of traditional database benchmarks and requirements of shared databases as well as for the proposed benchmark design and manuscript. The co-author Ashwin Arora provided invaluable feedback on the manuscript and contributed to an initial of the benchmark that was not included in the publication. The remaining authors Simon Karrer, and Carsten Binnig contributed invaluable feedback. All authors agree with the use of the publication for this dissertation.

6.1 Motivation & Requirements

In recent years, shared and collaborative data management use cases have gained increasing traction, leading to a new classification of systems to which our proposed systems,

[1] Several passages in this chapter were transferred verbatim from these publications.

BlockchainDB and TrustDBle, belong. Other systems in this category include Veritas [68], FalconDB [137], Fides [111], and Spitz [188]. These systems, distinct from classical database management systems designed for single-party usage, empower multiple parties to manage a shared database. They, therefore, need to uphold additional guarantees, such as data and computation integrity, as discussed in prior chapters.

Common abstractions. However, let us examine how these existing systems provide novel guarantees of trustworthy data management. We can observe that these systems typically take a very implementation-centric approach and often do not integrate well with the ACID guarantees of properties DBMSs. Moreover, these systems' concrete guarantees are very different from system to system and often hard-baked into their execution model. For instance, FalconDB is a blockchain-based system that incentivizes nodes to verify the execution of queries asynchronously, mitigating the high verification cost. As a result, potentially unverified queries from malicious servers stay undetected. In contrast, transactions (updates) in FalconDB are always verified synchronously by an entire network.

Benchmarking frameworks. Further, each new trustworthy data system introduces its own architectural choices and custom guarantees, which make it hard for users to navigate the plethora of shared database systems. While standard benchmarks like the TPC-C database benchmark have been a well-established tool to compare and analyze traditional database systems, they are unsuited to evaluate shared database systems. This is because these benchmarks do not take the novel properties and assumptions of trustworthy data systems into account. For instance, while classical databases were built with an isolated single-owner setting in mind, shared databases assume a multi-owner setting. This observation is also evidenced by the common practice in several of the above-mentioned shared database systems, that all implement (additional) custom benchmarks and evaluation frameworks.

6.2 Findings & Contributions

To improve the comparability and evaluation of future trustworthy data systems, this thesis provides two key contributions, which we explore subsequently.

6.2.1 Abstraction for Verifiability

Recognizing the diversity in guarantees and their implementations across different trustworthy data systems, we propose a more principled and database-centric approach toward

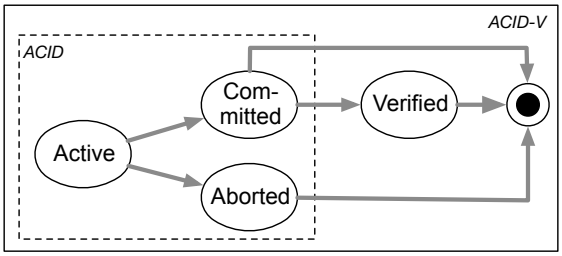


Figure 6.1: Simplified state model for ACID-V. The classical transaction state model is extended with a *Verified* state.

ensuring trustworthy guarantees such as transaction verifiability. Similar to the other components in ACID, such as the well-known isolation property, we suggest specifying the guarantees of verifiability in a declarative manner using different verification levels (i.e., strict or looser).

For instance, in the context of isolation in ACID, users can specify the isolation level (e.g., read committed, serializable) under which a transaction should operate. The database system ensures this isolation level through its concurrency control scheme (e.g., optimistic vs. pessimistic). Similarly, we propose to add a new *verifiability* property that the user can specify declaratively, and database systems can implement differently. This abstract perspective on verifiability enables users to reason about the guarantees provided by a system, independent of its implementation specifics.

To incorporate verifiability into ACID, we introduce a *verified* state into the classical transaction state model of ACID-compliant DBMSs, as illustrated in Figure 6.1. The simplified figure only considers scenarios where all nodes in a shared DBMS act honestly. However, we also discuss some aspects of malicious behavior in Chapter 14. As we can observe in the state model, a transaction can only attain the *verified* state after reaching the *committed* state.

Modeling *verified* as a state that follows the *committed* state has several advantages. First, since verification is typically an expensive step, the model leaves some freedom when the transition from *committed* to *verified* happens (i.e., directly after the commit or if it can be deferred). Moreover, it enables the user to declare which state is allowed to be read by other transactions (e.g., if committed but unverified can be read, or if all states must be verified before becoming visible). Second, as the *verified* state is optional, not all *committed* transactions need to be verified, thereby reducing overhead and allowing partial verification of transactions.

As elaborated in Chapter 14, using this model, we propose three distinct verification levels (strict, unstrict-full, unstrict-partial) that describe how systems verify transactions.

We also consider the potential effects of malicious behavior by individual peers (i.e., in the event of incorrect transaction execution). Our detailed discussion in Chapter 14 reveals that such a standardized model, shared by all trustworthy data systems, can ease the comparison of different systems and assist users in understanding the burgeoning landscape of new systems for trustworthy data systems.

6.2.2 Data Sharing Benchmark

While traditional database benchmarks like TPC-C [168], Smallbank [5] or YCSB [38] could serve as a starting point for evaluating trustworthy data systems, we argue that these benchmarks do not cover all important dimensions for shared databases as discussed above.

Recognizing this issue, we propose a standardized benchmark tailored to shared database systems based on the following contributions:

1. Analysis of unique characteristics distinguishing shared databases
2. Evaluation of the limitations in traditional benchmarks for evaluating shared database systems
3. Proposal for a novel benchmark design specifically for shared databases

We provide an overview of the proposed benchmark design (3.) here and refer to Chapter 15 for a detailed requirement analysis and comparison with traditional database benchmarks (1. and 2.).

Our benchmark design integrates three distinct workload categories — application, verification, and auditing — to effectively capture the unique requirements and properties of trustworthy data systems, as illustrated in Figure 6.2.

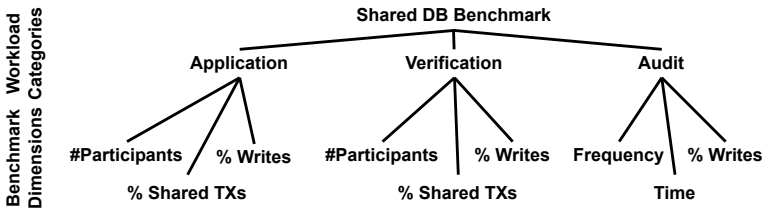


Figure 6.2: High-level benchmark design. Our benchmark defines three main workload types (Application, Verification, Audit) with novel benchmark dimensions.

Application workloads usually model a real-world use case and corresponding database queries that are executed by clients (also called terminals). We believe that a classical

benchmark such as TPC-C and its workload patterns (i.e., the transaction mix and data access patterns) is actually a good starting point. However, they can clearly not be used out of the box without any modifications for data sharing since, e.g., the TPC-C benchmark models the activities of one organization only — a wholesale supplier who accepts product orders at and for different warehouses.

The verification workload aims to reveal the overhead of different verification schemes implemented in data-sharing systems (e.g., online vs. offline verification). The auditing workload models an auditor's access patterns, which is read-heavy and scan-oriented since it needs to review a long history of data updates to see where potential issues (e.g., illegal data modifications) occurred. Overall, we envision that a benchmark execution has to include the application workload, while the other two categories, verification and auditing, are optional. This (modular) approach of workload categories enables the usage of the benchmark for systems that do not offer a certain capability (e.g., auditing).

Our benchmark reports classical performance metrics such as latency and throughput. We deliberately omit the quantification of security or trust levels provided by a system, acknowledging the inherent difficulty in measuring these concepts due to the potential for unknown attacks [128]. However, security-relevant parameters like verification strategy and data usage policies, which can impact overall performance, are included in the benchmark report. Future work may leverage these parameters to enable a classification-based evaluation of system security/trust as suggested in [128].

As illustrated in Figure 6.2, our proposal includes unique benchmark dimensions for each workload category. In the ensuing discussion, we will briefly outline the application workload dimensions, participant scalability, fraction of shared transactions, and read-/write ratio. These dimensions are varied in our benchmark to better understand the performance characteristics of a shared database system.

- **Participant Scalability:** Scaling the number of participants in a shared DB is different from simply adding additional database clients (terminals). This stems from the fact that scaling participants does not necessarily correlate with the number of clients generating transactions. For instance, in TPC-C, each warehouse could represent an independent organization, and adding more participants could correspond to increasing the number of warehouses. Note that, although it is the case for TPC-C, increasing the number of participants does not necessarily mean that the amount of data needs to be scaled.

- **Fraction of Shared Transactions:** This dimension aims to understand the impact on performance as the proportion of shared transactions increases. Note,

however, that traditional application workloads must be modified to incorporate shared data and transactions. For instance, in TPC-C, `new-order` transactions involving remote order-lines could be modeled as shared transactions.

- **Read/Write Ratio:** This benchmark dimension aims to unveil differences in how systems handle verification for read and write operations. By adjusting the workload mix, we can vary the proportion of writes in the workload, thereby observing the effect of verifying more writes in the system. For example, instead of the fixed transaction mix with mostly write-heavy transactions in TPC-C, we propose varying the transaction mix to gradually increase the proportion of write-heavy transactions.

For a more detailed discussion of these benchmark dimensions and a deeper exploration of the verification and audit workload categories, refer to Chapter 15. In summary, our novel benchmark design, integrating traditional performance metrics and emerging aspects like verifiability and auditability, lays the groundwork for a holistic appraisal of systems dedicated to trustworthy data management.

7 Conclusion

Our work encompasses multiple contributions towards the development of efficient and trustworthy data systems, providing valuable insights and a unique perspective to data management across multiple organizations. The subsequent sections delve into a detailed reflection on these contributions and outline potential directions for future research.

7.1 Summary

This thesis advocates for the development of *trustworthy data systems* as specialized platforms to accommodate the unique demands of cross-organizational data management. In line with the discussions laid out in Chapter 1 and Chapter 2, modern trends like digitization and data ecosystems necessitate a shift from isolated, localized data systems to more open models, thereby engendering new dependencies and challenges in data protection and governance. As explained in Chapter 3, we address these new requirements through two main strands of inquiry: *trustworthy data storage* and *trustworthy data processing*, and suggest a holistic benchmarking framework for assessing these systems.

Chapter 4 delves into the development of efficient trustworthy data storage. Our proposed system, BlockchainDB, combines blockchain and database technologies to meet the novel requirements of *data integrity* and *auditability*. Although blockchains are ideal for ensuring these properties, they are inherently limited in terms of performance and usability. To address this issue, we introduce an additional database layer on top of blockchains as a storage backend. This enhancement provides users with an *easy to use key value interface* to interact with the data and leverages database techniques such as partitioning to *parallelize read/write operations* across multiple blockchain networks. However, since data is now only stored in a certain partition instead of being replicated on all participants, BlockchainDB implements *additional verification protocols* to ensure data integrity on all partitions. Our experiments show that BlockchainDB can provide

up to two-orders of magnitude higher throughput than native blockchains and allows to better scale-out with the number of participants.

In our quest for efficient trustworthy data storage, we also examine the performance limitations of Merkle Trees more closely. Merkle Trees are a fundamental data structure in systems like blockchains for guaranteeing data integrity. However, they suffer from significant performance limitations when data is frequently updated. To improve the update performance of Merkle Trees, we investigate various latching techniques and introduce a novel method, *Reverse Latch Coupling*. Our innovative latching scheme significantly increases concurrency in Merkle Trees, leading to an *order of magnitude increase in throughput* and considerably *improved scalability*.

In Chapter 5, we explore achieving trustworthy data processing while maintaining high performance. We discuss how the novel requirements of *policy adherence* and *computational integrity* are implemented in our system, TrustDBle, by employing a *hybrid architecture*, utilizing blockchains and Trusted Execution Environments. Our architecture acknowledges that TEEs, like Intel SGX, are more efficient than blockchains for executing policies, e.g., a pre-defined transaction logic, but face capacity limitations for processing large volumes of data. To address this limitation, we propose placing only the most critical components inside the TEE to process database transactions correctly. For example, we suggest to implement a *trusted lock manager* inside the TEE that ensures that the isolation property of transactions is ensured. Our results confirm that combining TEEs, such as Intel SGX, with blockchains indeed enables a more efficient processing of database transactions compared to a baseline solely based on blockchains.

Addressing the limitations of the first version of Intel's SGX technology, we perform a *database centric evaluation* of the newly released second version (SGXv2). We aim to evaluate the efficiency of conventional techniques such as placing only a subset of components in the TEE and determine if SGXv2 can better support data-intensive applications. Our rigorous experiments using a *variety of typical database workloads* and access patterns confirm that SGXv2 can handle larger data volumes more efficiently. For instance, we find that with SGXv2 an in-memory B-Tree can scale up to about 120 GB of data and still provide high performance compared to a pure in-memory B-Tree. In contrast, this workload performs 25× worse in SGXv1 due to the limited capacity and involved overheads such as paging. However, we also find that SGXv2 still comes with several pitfalls, since capacity limitations remain and new challenges like remote NUMA access need to be addressed.

As trustworthy data processing is not limited to database workloads, we also show how it can be applied to *machine learning workloads*. In particular, we apply trustworthy data

processing to the collaborative setting of federated machine learning, in which a central parameter server is used to aggregate model updates computed by multiple organizations locally. We show how trustworthy data processing can be applied to this setting to prevent the PS from reconstructing private training data based on the received model updates. Our approach proposes a *decentralized parameter server architecture*, that splits the central parameter server into multiple, independent PS instances. This architecture includes three privacy-preserving configurations on how to partition, serve and update the model parameters for better privacy. Empirical evidence on commonly used ML data sets show noticeably stronger resilience against gradient-based data reconstruction attacks.

Finally, in Chapter 6, we make two significant contributions to enhance the comparability and evaluation of future trustworthy data systems. First, we advocate for expressing common properties of trustworthy data systems, such as verifiability, in a declarative manner. This approach facilitates users' understanding of the system's guarantees while allowing flexibility in implementation. For instance, in analogy to isolation levels, we define a set of *verification levels* that users can employ to describe how strict trustworthy processing properties must be checked. Second, we introduce a comprehensive benchmark for trustworthy data systems, introducing three workload categories: application, verification, and audit. This holistic approach allows a more thorough evaluation of these systems, capturing traditional performance metrics as well as new aspects like verifiability and auditability.

In a nutshell, our work has covered various aspects of trustworthy data systems, focusing on novel ways to manage data across organizational boundaries. We believe that the developed techniques and concepts can find adoption in other domains paving the way for several future research directions.

7.2 Future Research Directions

In today's digital climate, data collaboration across organizations is not only increasing in importance but also becoming more regulated. As such, DBMSs, which are the focus of this dissertation, represent only one facet of the diverse platforms used to store and process data. Consequently, the concepts and methodologies proposed within this dissertation pave the way for a multitude of future explorations:

Embracing trustworthy techniques. Organizations rely on an array of platforms to store and process data, ranging from basic file systems to more contemporary platforms

such as lakehouses [186]. The level of support these systems provide for data protection and governance varies considerably. While many platforms incorporate features like data encryption at rest, comprehensive auditability guarantees for trustworthy data storage are often lacking.

This dissertation has primarily focused on integrating trustworthy data management requirements into DBMSs. However, we contend that many of the insights gleaned from our work have applicability beyond databases. For instance, an optimized Merkle Tree data structure as proposed by our work could bolster the efficient implementation of data integrity guarantees in lakehouses. Similarly, our findings on high-performance, trustworthy data processing using TEEs, like Intel SGX, could be beneficial in other contexts, such as machine learning [180], ensuring data is processed following a predefined logic. While not all of our approaches may be directly applicable, they provide a solid foundation for investigating alternative strategies.

Commoditizing trustworthy data management. While this dissertation proposes the concept of trustworthy data systems as a specialized platform for cross-organizational data management, we foresee the wider adoption of trustworthy data management properties in data systems. Just as data encryption at rest is now commonplace, our work lays the groundwork for broader support of guarantees such as data integrity and auditability in general-purpose DBMSs. The reason for this is that these properties not only facilitate cross-organizational data management, but they also help organizations comply with regulations such as GDPR, which demand auditability and transparency.

The process of integrating trustworthy data management techniques into existing systems, as opposed to creating new systems from scratch, leverages many of the building blocks used in our research. For example, while using a standard blockchain as a storage backend (as in BlockchainDB) may not be practical for established systems, the cryptographic principles and data structures like Merkle Trees that underlie blockchains remain pertinent. While the idea of integrating cryptographic techniques and ideas from blockchains in databases is already being explored in related work [191], these approaches still constitute additional engines that are not tightly integrated into the standard database stack. Instead we envision a shift towards replacing common index structures like B-Trees with alternatives that support data integrity (e.g., MB-Trees [101]) without compromising performance.

Exploration of other technologies and approaches. As we anticipate the commoditization of trustworthy data management, we recognize the need to research alternative strategies to meet its requirements. We have only explored a subset of the security and cryptographic techniques applicable in this context as a reference point.

For instance, recently released alternative TEE technologies, such as Intel Trust Domain Extensions (TDX) [82] or AWS Nitro [8], could potentially revolutionize the implementation of trustworthy data management. Likewise, maturing cryptographic primitives like cryptographic accumulators [134] present viable alternatives to Merkle Trees for ensuring data integrity in data management systems. With the rapid evolution of security and cryptographic practices, keeping pace with these developments and efficiently integrating them into data systems is an ongoing challenge and necessity for enhancing trustworthy data management in general.

Part II

Peer-Reviewed Publications

8 BlockchainDB - A Shared Database on Blockchains

Abstract

In this chapter we present BlockchainDB, which leverages blockchains as a storage layer and introduces a database layer on top that extends blockchains by classical data management techniques (e.g., sharding) as well as a standardized query interface to facilitate the adoption of blockchains for data sharing use cases. We show that by introducing the additional database layer, we are able to improve the performance and scalability when using blockchains for data sharing and also massively decrease the complexity for organizations intending to use blockchains for data sharing.

Bibliographic Information

The content of this chapter was previously published in the peer-reviewed work Muhammad El-Hindi, Carsten Binnig, Arvind Arasu, Donald Kossmann, and Ravi Ramamurthy. "BlockchainDB - A Shared Database on Blockchains." In: *Proc. VLDB Endow.* 12.11 (2019), pp. 1597–1609. DOI: 10.14778/3342263.3342636. URL: http://www.vldb.org/pvldb/vol12/p1597-el-hindi.pdf.

8.1 Introduction

Motivation. Blockchain (BC) technology emerged as the basis for crypto-currencies, like Bitcoin [126] or Ethereum [29], allowing parties that do not trust each other to exchange funds and agree on a common view of their balances. With the advent of smart contracts, blockchain platforms are being used for many other use cases beyond crypto-currencies and include applications in domains such as governmental, healthcare and IoT scenarios [16, 20, 169].

An important aspect that many scenarios have in common, is that blockchains are being used to provide shared data access for parties that do not trust each other. For example, one use case is that the blockchain is used for tracking goods in a supply chain where independent parties log the location of individual goods. What makes blockchains attractive in those scenarios are two main characteristics: First, blockchains store data in an immutable append-only ledger that contains the history of all data modifications. That way, blockchains enable auditability and traceability in order to detect potential malicious operations on the shared data. Second, blockchains can be operated reliably in a decentralized manner without the need to involve a central trusted instance which often does not exist in data sharing. However, while there is a lot of excitement around blockchains in industry, they are still not being used as a shared DB in many real-world scenarios. This has different reasons: First and foremost, a major obstacle is their limited scalability and performance. Recent benchmarks [44] have shown that state-of-the-art blockchain systems such as Ethereum or Hyperledger, that can be used for building general applications on top, can only achieve $10's$ or maximally $100's$ of transactions per second, which is often way below the requirements of modern applications. Second, blockchains lack easy-to-use abstractions known from data management systems such as a simple query interface as well as other guarantees like well-defined consistency levels that guarantee when/how updates become visible. Instead, blockchains often come with proprietary programming interfaces and require applications to know about the internals of a blockchain to decide on the visibility of updates.

Contribution. In this paper, we present BlockchainDB that tackles the beforementioned issues. The main idea is that BlockchainDB leverages blockchains as the native storage layer and implements an additional database layer on top to enable access to data stored in shared tables. That way, existing blockchain systems can be used (without modification) as a tamper-proof and de-centralized storage. On top of the storage layer, BlockchainDB implements a database layer with the following functions:

- *Partitioning and Partial Replication:* A major performance bottleneck of blockchains today is that all peers hold a full copy of the state and still only provide (limited) sharding capabilities. In the database layer of BlockchainDB, we allow applications to define how data is replicated and partitioned across all available peers. Thus, applications can trade performance and security guarantees in a declarative manner.

- *Query Interface and Consistency:* In the DB layer, BlockchainDB additionally provides shared tables as easy-to-use abstractions including different consistency protocols (e.g., eventual and sequential consistency) as well as a simple key/value interface to read/write data without knowing the internals of a blockchain system. In future, we want to extend the query interface to shared tables to support SQL with full transactional semantics.

In addition to these functions, the database layer of BlockchainDB comes with an *off-chain verification procedure* in which peers can easily verify the read- and write-set of their own clients. The idea of the verification procedure is that peers can detect other potentially misbehaving peers in the BlockchainDB network. This is needed since not all BlockchainDB peers hold the full copy of the database and the storage layer of a remote peer could potentially drop puts or return a spurious value for a read operation (i.e., a value that was not persisted in the database).

By introducing a database layer on top of an existing blockchain, BlockchainDB is not only able to provide higher performance, but also to decrease the complexity for organizations intending to use blockchains for data sharing. While the concept of moving certain functions out of the blockchain into additional application layers has been studied previously (e.g., [34, 47, 48, 125]), to the best of our knowledge, BlockchainDB is the first system to provide a fully functional DB layer on top of blockchains. Our experiments show that BlockchainDB allows to increase the performance significantly to support many real-world applications.

Outline. The remainder of this paper is organized as follows: First, in Section 8.2 we give an overview of what functionality and security guarantees BlockchainDB provides for data sharing. Afterwards, in Section 8.3 we present the BlockchainDB architecture and discuss the trust assumptions as well as potential attacks. Then, in Section 8.4 and Section 8.5 we discuss the details of the database layer and how blockchains are being used as the storage layer. Section 8.6 afterwards outlines our off-chain verification protocol. The results of our evaluations with the YCSB benchmark are then presented

in Section 8.7. Finally, we conclude with related work in Section 8.8 and a summary in Section 8.9.

8.2 Overview & Guarantees

As explained already in the introduction, there are many different applications where untrusted parties need to have shared access to the same database. To enable such a shared access to data, BlockchainDB provides so-called *shared tables*. For accessing a shared table, clients can use the put/get interface of BlockchainDB to access tuples in the shared table based on their primary key. In order to understand the functionality and guarantees that BlockchainDB provides for untrusted parties to access data via shared tables, we will introduce a short motivating scenario and use this scenario also to outline the guarantees that applications get when using BlockchainDB for data sharing.

Scenario. Figure 8.1 shows an example scenario for data sharing where three untrusted parties (WholeFoods, FedEx, and Lindt) share access to the same database. The scenario describes a typical supplier scenario where WholeFoods acts as customer, Lindt as supplier, and FedEx as shipping company. In this scenario, WholeFoods first places a new order by inserting two new entries into the shared database consisting of two shared tables ①. After the order is placed, Lindt processes the new order ②. To keep track of the order, Lindt updates the status from *new* to *ready* as shown in ③. Once the order is ready, FedEx starts its operation ④. After the order has been shipped to the customer, FedEx updates the status of the order to *delivered* as shown in ⑤.

A naïve way of implementing such a scenario would be that one of the parties is hosting the shared database; e.g., say WholeFoods hosts the shared database as a service for all their suppliers. In this setup, however, WholeFoods could easily leverage the fact that it can manipulate the shared data or return false values to the other parties about the order status, without any chance for the parties to verify that WholeFoods was in fact acting in a malicious manner. For example, WholeFoods could claim that the order was lost during transit by returning a spurious (i.e., false) order status to Lindt as shown in ⑥ to trigger that a replacement is sent (without paying for it). Even worse, WholeFoods could actually delete the order or not store it in the database in the first place. In case of a lawsuit, no evidence could thus be found that WholeFoods (or any other party) was actually acting maliciously.

Guarantees. In order avoid these problems, a blockchain network such as Ethereum or Hyperledger could be used to implement a shared database. The benefit of using

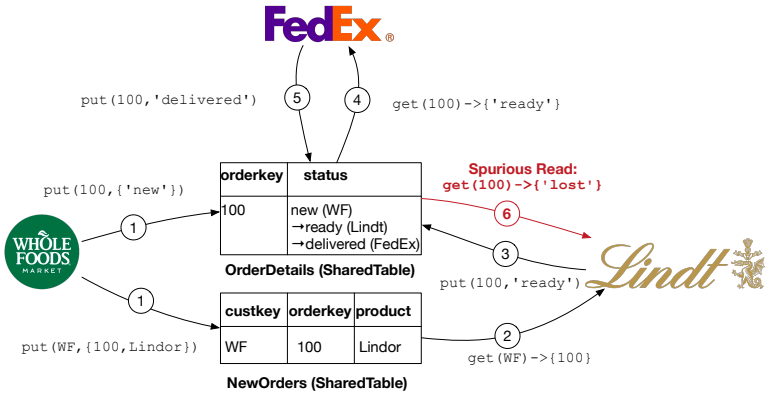


Figure 8.1: A Data Sharing Scenario

a blockchain is that every party (WholeFoods, Lindt, and FedEx) keeps a full copy of the database and the majority of parties needs to agree (based on the blockchain-based consensus protocol) on every update before its effect becomes visible. As a consequence of using blockchains, we get the following important guarantees for data sharing: First, the data in each peer is stored in a tamper-proof log. That way, any data modification of the log by any potentially malicious party could be detected since the cryptographic hashes used in the blockchain would not be valid anymore. Second, all parties read only state from their local copy of the database. That way, spurious reads (that are a problem if data needs to be read form a remote party) can be avoided.

However, as discussed in the introduction, using blockchains directly for data sharing comes with significant problems (e.g., w.r.t. performance) and complexities due to missing abstractions. The idea of BlockchainDB is thus to provide the same security guarantees as blockchains — (1) a tamper-proof (auditable) log as well as (2) verifiable reads and writes. At the same time, BlockchainDB enables high performance and provides an easy-to-use query interface.

8.3 System Architecture

8.3.1 Architecture Overview

The main idea of BlockchainDB is that it implements a database layer on top of an existing blockchain. The database layer provides clients with a simple-to-use abstraction (called a shared table) with a put/get interface and stores all data in its storage layer that relies on blockchains as discussed before. Figure 8.2 shows a possible deployment of

BlockchainDB across four different BlockchainDB peers (i.e., untrusted parties) to enable access to shared tables as discussed in the scenario before.

The key idea of BlockchainDB is that data is not replicated to all peers to avoid the high overhead of blockchain consensus. Instead, shared tables are partitioned (i.e., sharded), thereby each shard is implemented as a separate blockchain network. Moreover, shards are only replicated to a limited number of peers instead of replicating the data to all peers. For example, in the scenario explained in the previous section, both shared tables (*NewOrders* and *OrderDetails*) can be partitioned using the *OrderKey* as partitioning key and replicated only to a subset of the peers (WholeFoods, Lindt, and FedEx).

As a direct consequence of sharding to speed-up the performance not all peers store all data locally. When accessing the shared table, the database layer thus needs to redirect the request either to the local or a remote storage depending on the requested key. While storing data in a blockchain still gives us a tamper-proof log, the remote peer can drop a put or return a spurious value for a read. To verify all remote reads/writes an additional verification procedure is thus provided by BlockchainDB.

In order to participate in a BlockchainDB network and allow clients of a party to read/write data into a shared database, a peer in BlockchainDB can be either deployed as a *full peer* which hosts a database and a replica of at least one shard or as a *thin peer* which only connects to other remote peers to access data in a shard (i.e., the peer does not store a copy of a shard). Having thin peers enables parties with only limited resources to participate in a BlockchainDB network and access the shared tables (such as a small supermarket, called *SmallMarket* in our example).

Finally, similar to permissioned blockchains, BlockchainDB assumes that the parties who want to share data are previously authenticated and known to each other. However, parties do not need to trust each other (since they might have contrary goals). More details about our security assumptions will be provided in Section 8.3.3.

8.3.2 System Components

Next, we explain how clients interact with BlockchainDB as well as the functionality of each component.

Clients. Clients interact with a shared table via their own BlockchainDB peer (which they trust). Thus, instead of interacting with a blockchain network directly, in BlockchainDB clients interact with their peer (i.e., the database layer) through a simple `put/get` interface to read/write shared data. Furthermore, clients can use the `verify` method of the database layer to trigger an off-chain verification procedure for the

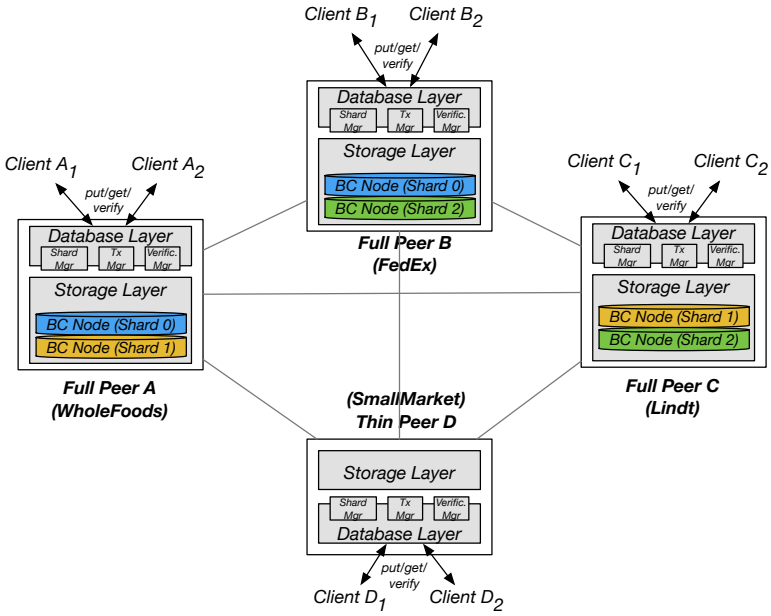


Figure 8.2: A typical BlockchainDB Network

online verification of the last read/write-operation of that client. This allows the client to detect potential misbehaving peers in a BlockchainDB network (e.g., to detect if another BlockchainDB peer — more precisely its storage layer — returned a spurious value for a `get`) and is needed in BlockchainDB since not all peers keep a full copy of the database state. So, all reads/writes that are being redirected to a remote storage layer must be additionally verified to get the same guarantees as local reads/writes. We additionally support (a deferred) offline verification procedure that is called from the database layer for batches of reads/writes from all clients. The deferred verification procedure provides a higher throughput than the online verification, since it performs the verification while new reads/writes are being executed. However, clients might work (for a limited amount of time) on an unverified database state. More details about the verification procedure(s) are discussed in Section 8.6.

Database Layer. The database layer in BlockchainDB is mainly responsible to execute the `put/get` calls from the clients. If a `put/get` call comes in, the database layer uses the *Shard Manager* to decide to which shard of a table the operation should be directed to. The shard can be either stored in its local storage or remotely in another BlockchainDB peer depending on the partitioning scheme of the shared table. Currently, BlockchainDB implements a hash-based sharding approach, in which the user defines the number of shards and their allocation to BlockchainDB peers when creating

a shared table. Another major difference of BlockchainDB and a pure blockchain network is that the database layer of BlockchainDB implements a *Transaction Manager* that provides well-known consistency levels (i.e., eventual consistency and sequential consistency). That way, clients get a defined behavior for concurrent puts/gets without the need to know the internals of blockchains. Additionally, the database layer can re-order/batch puts/gets depending on the chosen consistency level to optimally leverage the underlying blockchain and thus further improve the performance. Details about the database layer are discussed in Section 8.4.

Storage Layer. The storage layer serves as a persistent, auditable storage backend of BlockchainDB and is based on existing blockchain systems. As depicted in Figure 8.3, the storage layer of BlockchainDB is able to parallelize data processing across different shards whereas each shard is implemented as a separate blockchain network where the data in a shard is replicated to multiple (but not necessarily all) peers using the internal blockchain consensus protocols. For example, in Figure 8.2 the BlockchainDB network uses in total three different blockchain networks (one for each shard) with a replication factor of two. Details about the storage layer are discussed in Section 8.4.

8.3.3 Trust Assumptions & Threat Model

As mentioned previously, we assume a permissioned setting in which the set of participants is known at the beginning. For simplicity, we assume the set of participants is fixed; extending our techniques to a dynamic set and incorporating more complex consortium rules is orthogonal to our work. Further, since we use an off-the-shelf blockchain such as Ethereum for storing data, we inherit several security characteristics and guarantees of the underlying blockchain system, e.g., w.r.t. peer authentication using public/private keys, replay protection, number of tolerable malicious nodes.

Moreover, the need for an off-chain verification procedure stems from the fact that a BlockchainDB peer might need to read/write data from/to a remote peer; i.e., the local peer does not participate in the blockchain network for that shard. In particular, *Thin Peers* need to run the verification procedure for all read/write operations. For our threat model we thus make the following assumptions:

- Clients that connect to a BlockchainDB peer trust their local database and storage layer.
- This allows a BlockchainDB peer (i.e., the database layer) to perform verification on behalf of all locally connected clients.

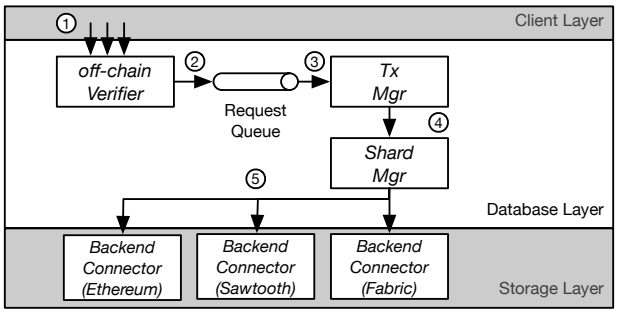


Figure 8.3: Database and Storage Layer

- Moreover, a BlockchainDB peer can trust the data that is written to or read from a local shard. Those operations thus do not need to be verified.
- If the majority of the peers that keep a copy of a shard is not malicious, then a client can trust all puts/gets once verified. Thereby, the number of peers that can form a majority depends on the security assumptions of the used blockchain system. For example, some blockchains might require $\frac{2}{3}$ of the nodes to be trusted while others have different properties.

In consequence, whenever a client accesses data that is stored on or written to a remote shard on another peer, the local peer will run an additional verification protocol to verify the operation and mitigate the following attacks:(1) The drop of a `put` operation that needs to write data to a remote peer will be detected. (2) Spurious/fake data returned for any `get` operation that needs to read data from a remote peer will also be detected. Details about the off-chain verification procedure will be explained in Section 8.6.

8.4 Database Layer

In this section, we describe the `put/get` interface of BlockchainDB and how these operations are being executed by the database layer to implement different consistency levels on top of blockchains as a storage layer.

8.4.1 Query Interface

As mentioned before, BlockchainDB provides shared tables as main abstraction. Each table has multiple columns (attributes), whereas one is the dedicated primary key that can be used to access the table. More details about the data model and table creation

is discussed in Section 8.5. The query interface of BlockchainDB enables clients to execute the following three operations on shared tables:

- `get(t, k)` → `v`: This call returns all attributes of the row in table t that has the key k.
- `put(t, k, v)` → `void`: This call inserts a new row in table t with key k. All attribute values are encoded in v similar to what document stores do. In case a row with key k is already in table t, the row is updated with the new values in v. For simplicity, we assume that values for all attributes are given in v.
- `verify()` → `bool`: This call is for online verification; i.e., a client can call `verify` immediately after any `get/put` call and gets back `true` or `false` to indicate whether or not the verification was successful. For a `put`, it means whether or not the `put` was actually committed in the storage layer and for a `get` it is checked whether or not the returned value was correct or a spurious value.

Next, we explain how `get/put` methods are implemented. The verification (online/offline) is explained in Section 8.6.

8.4.2 Query Execution

In the following we explain the execution process and discuss the involved components as depicted in Figure 8.3.

For accessing the table, clients send put/get requests to their local BlockchainDB peer using the query interface discussed before. Requests arriving from the client in the database layer are first received by the off-chain verifier ① that records the unverified reads/writes (i.e., all remote reads/writes) of a client. This information is used for the online/offline verification. The verifier then forwards the request to an internal *RequestQueue* ②. Next, the *TransactionManager* (Tx.Mgr)[1] polls the requests from the queue and processes them according to a specified consistency level[2] ③. Currently, we support sequential and eventual consistency and a version of eventual consistency (called bounded staleness) that guarantees a limited staleness of the accessed data as discussed in the next section.

The different consistency levels differ in how quickly the database layer can process operations. For example, for eventual consistency a get-operation is immediately processed;

[1] We call this component transaction manager since it translates every put/get of a client into a blockchain transaction.

[2] The consistency level can be specified for each table individually.

however, no guarantee is given that a potential outstanding put-operation has already been committed to the blockchain. In sequential consistency, the database layer needs to execute put/get-requests in a global order and thus, potentially blocks a get-request until an outstanding put-request has been committed to the blockchain.

Once a put/get-request is ready to be sent to the storage layer, it is forwarded to the *ShardManager* ④. The *ShardManager* service has two main purposes: First, it determines the correct shard for a given key, and second, it is responsible for sending requests as read/write operations in parallel to the table shards. To access data in blockchains, different *BackendConnectors* can be used ⑤ that allow BlockchainDB to read/write data from/into the underlying blockchain network. The details about the *BackendConnectors* are discussed in Section 8.5.

8.4.3 Consistency Levels

As mentioned before, BlockchainDB provides well defined consistency levels on top of blockchains. In order to understand how different consistency levels can be implemented on top of blockchains, we first discuss how blockchains make updates visible to clients. Afterwards, we explain how sequential consistency is implemented and how eventual consistency is supported. Moreover, a version of eventual consistency that guarantees a limited staleness is introduced.

Processing Model in Blockchains. In general, blockchain networks (i.e., in our case all data that is stored in one shard) agree on a global order of writes (i.e., blockchain transactions) in all replicas in which they are appended to their log. Thus, a naïve way to implement sequential consistency in the database layer of BlockchainDB on top of a blockchain network would be to wait after every write (i.e. put) until the blockchain transaction is committed. However, as shown in [44] the latency until a blockchain transaction is committed can take from seconds to minutes and would severely limit the throughput of write-intensive workloads significantly. Another challenge of blockchains is that some transactions might end up in a fork (e.g., if proof-of-work is used as in Ethereum). These transactions must be re-executed which further increases latency under the blocking execution model discussed before.

Sequential Consistency. In BlockchainDB, we thus follow a different approach. Instead of waiting after each write (in an eager manner), we monitor all pending writes in the database layer of BlockchainDB to enable lazy waiting. This means, only in case a read operation comes in for a pending write (i.e., read and write share the same key), we wait for that write, and all other pending writes that have been issued

before, to be committed to the blockchain. Reads can only be executed once the write is committed to the blockchain. This enables that clients not only read their own writes but defines a global order of writes (for all clients) connected to the same database layer. Moreover, since the blockchain network (which is used to implement a shard table in BlockchainDB) orders all writes sequentially, we get a global order of all writes and thus even clients connected to different peers in BlockchainDB see the same global order of write operations.

Eventual Consistency and Bounded Staleness. Providing eventual consistency on top of the sequential model of blockchains is simple. Instead of waiting for pending writes, we execute each incoming read operation of a client (i.e., a get) immediately. This, however, could lead to two issues: First, the pending-write queue might grow quickly for write-intensive workloads. Second, for reads (i.e., get operations of clients), BlockchainDB might return stale values (without any bound on the staleness) since the time until a blockchain transaction is committed can take up to minutes (as mentioned before). In order to mitigate these issues, a user can define a maximum staleness-factor in BlockchainDB (which defines the maximum number of writes in the pending-write queue). That way, applications can control staleness and latency; i.e., with a longer queue the staleness will increase but the latency of reads decreases (as we will also show in our experiments). We call this version of eventual consistency, bounded staleness which is similar to the ideas discussed in [164].

8.5 Storage Layer

The storage layer of BlockchainDB is responsible for all interactions with the blockchain networks that are used to store shared tables. In the following, we first explain the creation of shared tables as well as their data model. Afterwards, we discuss the interface that is exposed by the storage layer to the database layer for executing reads/writes on shared tables. Finally, we exemplary discuss how the methods of the storage interface are implemented in so called backend connectors that allow BlockchainDB to access the data in a blockchain. Currently, we provide connectors for Ethereum, Hyperledger Sawtooth and Hyperleder Fabric.

8.5.1 Shared Tables

As a first step of a data sharing scenario, a new shared table has to be created in BlockchainDB. A new table in BlockchainDB is defined by its schema and its sharding configuration.

In BlockchainDB, the schema of a new table is defined using a key/value data model where the key is the primary key and the value represents the payload of a tuple. The sharding information of a new table contains values for the parameters such as the number of shards or the replication factor and the allocation. These parameters not only have an influence on the overall performance but more importantly on the trust guarantees a new table provides. For example, the number of replicas directly dictates how many malicious peers can be tolerated; e.g., most blockchain networks (such as Ethereum) tolerate if less than half of the peers are malicious.

For creating a new shared table in BlockchainDB, one of the clients involved in the data sharing application proposes a new table and submits the information about the table name and its schema as well as the sharding information to its local peer. The local peer then coordinates the table creation process with all other peers on behalf of the initiating client. The main steps of the process are discussed below.

The first step of the table creation process is implemented as a smart contract which takes the information about the new table including the sharding parameters as input and updates the BlockchainDB catalog (i.e., its metadata). The metadata of BlockchainDB is stored in a dedicated blockchain that is replicated to *all* peers. By storing the metadata in a dedicated blockchain network that is fully replicated and governed by a smart contract, we can not only guarantee that all peers have the same view on the metadata but also that no peer can tamper with the metadata. Fully replicating the metadata is not a performance problem since metadata is typically small and updated less frequently.

As a second step of the table creation process, and once the metadata is updated successfully by the smart contract, the peer which coordinates the table creation process signals all other peers to deploy the shards for the new table. For each new shard that should be stored on a peer, a new blockchain node is started by the peer and connected to the other blockchain nodes, thus forming a new blockchain network for the shard. For finding out which shards need to be deployed for a new table and to which other peers the shard should be connected to, each peer uses its local trusted copy of the metadata.

One important question of the table creation process is, why clients of the other peers should trust the new table. One could think that the table creation process opens up

a possible attack since the client and the local peer who coordinates the table creation could be already malicious at the time of table creation and thus could decide to create a new shared table with low trust guarantees that in the extreme case has only one shard consisting of one replica (that might even be assigned to the local peer). In this case, the new table would not provide any trust guarantees to clients of other peers since the peer which created the table stores the only copy and thus could drop puts and return spurious values for get operations without the possibility for the other clients or thin peers to verify their operations.

Thus as a last step of the table creation process, all BlockchainDB peers have to confirm that they agree with the table (i.e., in particular the sharding information) proposed by the coordinating peer. The key for the confirmation step is that BlockchainDB provides trusted and replicated metadata across peers. That way, all other peers can check whether they agree in a trusted manner with the sharding information before using that table for data sharing. Once the trust guarantees for the new table are confirmed by all peers in BlockchainDB, they update the metadata (i.e., by incrementing a confirmation counter). Only once all peers confirmed the new table, it can be used by clients of any peer for actual data sharing by executing put/gets on it.

A last point we want to mention is that BlockchainDB assumes a permissioned setup where only authenticated peers can participate in a network. The peers can work together while they do not necessarily need to trust each other.

8.5.2 Storage Interface

As shown in Figure 8.3, the database layer uses so called backend connectors in the storage layer to access data in a shared table (i.e., a blockchain network). The idea of the backend connector is to provide a stable interface to the database layer to access the data independent of which blockchain is being used as backend. The main methods of the storage interface are:

- `read(s, k)` → `v`: This method allows the database layer to read a value v (i.e., the tuple) for a given shard s (which is just a global unique identifier in BlockchainDB) and a key k.

- `write-async(s, k, v)` → `tx-id`: This method allows the database layer to write a value v (representing a tuple) with a key k into shard s. Important is that the write is an asynchronous operation and just returns an identifier `tx-id` of the blockchain transaction that was created for that write.

- `check-tx-status(s, tx-id)) → TX-STATUS`:
  In order to check if a `write-async` operation has been successfully committed, this operation can be called. This method takes a shard identifier s and a transaction identifier `tx-id`, and returns the status of the blockchain transaction in that shard. The status can either be `COMMITED` if the write was successful, `ABORTED` if the write failed (e.g., due to failed validation in the blockchain), or `PENDING` if the transaction is submitted but not yet added to a valid block in the blockchain.
- `get-writeset(s, e)) → ws`: Returns all writes that were executed on shard s in epoch e. This method is used for offline verification, which verifies the workload of all clients connected to one BlockchainDB peer in epochs as discussed in Section 8.6.

The first three methods of the storage interface are called by the database layer in order to implement different consistency levels. For example, under sequential consistency, the `write-async` method is called when a `put(t, k, v)` (for a table t, key k, and value v) from a client is processed by the database layer. The returned transaction identifier `tx-id` is put together with the table t and key k into a pending-write queue in the database layer. If a `get(t, k, v)` operation for the same key is coming in from a client after the `put`-call, the database layer has to check the pending-write queue, and if a pending write is found, it needs to call `check-tx-status(tx-id)` to see whether the transaction committed or not. If not, the database layer has to block until the transaction status changes to `COMMITED`.

8.5.3 Backend Connector

The methods discussed before need to be implemented by each backend connector for different blockchain systems. In the following, we discuss the implementation of those methods for Ethereum as an example.

It is important to note that the backend connector stores the connection information for each shard identifier and uses a native blockchain client (such as *geth* for Ethereum) to access a blockchain network which stores the data of a shard. The connection information contains the list of IP-addresses and ports of all BlockchainDB peers which host a copy of the shard (i.e., the peers which participate in the blockchain network that store the data of the shard). For executing operations, the local IP-address is used if it exists in the connection information (which means that a shard copy is stored on the local peer). Otherwise, one of the remote IP-addresses is selected in a random manner to load balance the execution across different peers.

For accessing shared data in a blockchain network, BlockchainDB needs to install a minimal smart contract definition which provides a simple *read/write* interface for each shard. These smart contracts are then called by the connector to implement the interface methods presented before. In the following, we show an extract of the smart contract code installed for an Ethereum network.

```
contract KVContract {
  // state variables and constructor omitted

  // read method in contract
  function read-blockchain(bytes memory key)
  public view returns(bytes memory value){
    return data[key];
  }

  // write method in contract
  function write-blockchain(bytes memory key, bytes memory val)
  public returns(bool success){
    data[key] = val; return true;
  }
}
```

The first method of the backend connector is the `write-async` method. This method takes the incoming tuple (i.e., a key/value pair) as well as the shard identifier s. Afterwards, the connection information is looked up for this shard and the key as well as the value is converted into a byte representation before sending it to the blockchain network for processing (or more precisely to the smart contract of the blockchain as discussed above). The byte-data is then sent to the `write-blockchain` method of the `KVContract` using *geth* as client for Ethereum. We found that representing the data in a byte-format before storing it in a blockchain network not only leads to decreased storage cost, but also allows for more efficient processing of the transaction on the blockchain. The unique transaction identifier `tx-id` returned by *geth* client is returned to the database layer as a return value. This identifier can be used to check the status of the transaction from the database layer.

The second method implemented in a backend connector is the `read` method. This method takes a shard s and a key k as input. For the execution, the connection information is looked up for this shard and the backend controller then converts the key into a byte representation and sends the data to the `read-blockchain` method of the `KVContract`. Different from the `write-async` method, the backend connector does not use a blockchain transaction to execute the `read-blockchain` contract but it uses a call (which is a read-only operation in Ethereum). The benefit of this method is that it does

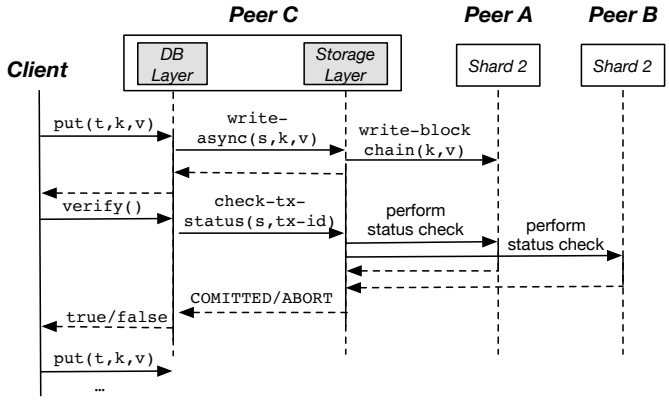


Figure 8.4: Online Verification

not require the heavy-weight processing of a blockchain transaction and usually only needs a few milliseconds to be executed. The result of the call is the byte representation of the value (i.e., tuple). Before sending the value back to the database layer, the byte representation is converted into the original data type of the table.

Finally, the third method implemented in a backend connector is the `check-tx-status` method. This method takes a shard s and a transaction identifier `tx-id` as input and returns the transaction status to the database layer. To check the status of a transaction, Ethereum provides different options: First, the storage layer can regularly poll the blockchain for the latest status of the transaction using *geth*. Second, the storage layer can subscribe to events and be notified when, e.g., a new block has been created. When notified, the storage layer can query the blockchain for details of the new block and determine the status of the transaction. However, in order to detect failed/rejected transactions, the storage layer still has to poll the blockchain regularly for the transaction status.

8.6 Off-Chain Verification

In this section we describe the details of our off-chain verification protocol. The main goal of the verification procedures is to prevent (1) dropped puts and (2) spurious reads if a peer needs to read/write data from/to a remote shard.

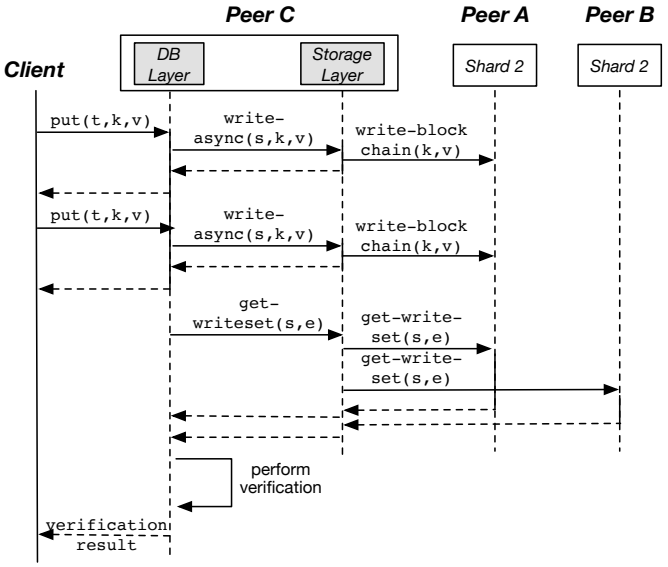


Figure 8.5: Offline Verification

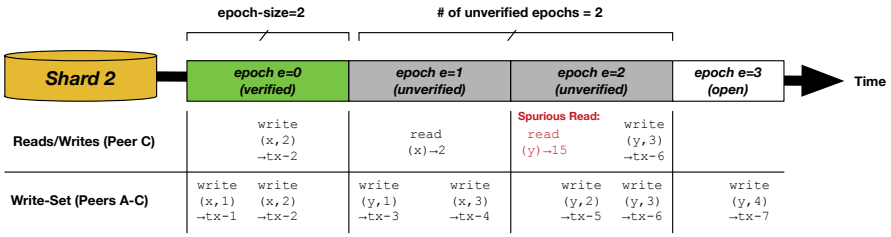


Figure 8.6: Epoch-based Offline Verification

8.6.1 Online Verification

Overview. In online verification every operation issued by a client is subsequently verified if the client calls the `verify` operation as shown in Figure 8.4. While all blockchain systems internally make use of Merkle-Trees and similar structures to store data in a verifiable way, only few blockchain systems expose an interface to clients that allows them to retrieve data along with a verifiable proof. Hence, in order to verify the result of an operation, a BlockchainDB peer needs to contact the majority of the blockchain network to verify that the retrieved result is valid. In the following, we explain how `get`- and `put`-operations can be verified.

Verification Procedure. In order to verify the value v returned by a `get`-operation, we extend the storage interface to return the block number from which the value was read

for the given shard. Afterwards, we read that block from the majority of peers which hold a copy of the shard and then verify if the value v for key k in those blocks matches the returned value. If the returned value does not match with that of the majority, a manipulated read was detected.

`Put`-Operations are verified differently, since they are executed as transactions on the blockchain as discussed before. Consequently, they are mined as part of blocks. Similar to get operations, `put`-operations can be verified by querying the majority of the blockchain for the latest transaction status. If the transaction is not recorded as committed on the majority of peers, a dropped write was detected. We do not need to check the validity of the block content, since transactions are signed by clients. Thus, a manipulation of the transaction content by a remote peer is not possible.

8.6.2 Offline Verification

Overview. While online verification guarantees the validity of an operation right away, it has several inefficiencies and drawbacks. First, online verification is a blocking action that prevents any other operation to be executed by a client. Second, since transactions are grouped into blocks and thus mined in batches by blockchains, system throughput can be improved by verifying transactions in batches.

The basic idea of offline verification is shown in Figure 8.5. Instead of calling the `verify`-method after every `put`- or `get`-call, offline verification defers verification. Deferring verification allows us to batch multiple `put`- and `get`-calls together and verify multiple operations at once instead of separately. In the following, we describe the procedure for offline verification and its main parameters.

Verification Procedure. The offline verification procedure is executed per shard in batches (called epochs). An epoch of a shard in BlockchainDB is defined by a fixed number of writes (called epoch-size $|e|$) that can be executed in one epoch in one shard. Once the maximum number of writes is executed in one epoch, the epoch of the shard is closed.

The main idea of offline verification is that all operations that are executed (by one peer) in one epoch are verified together in a batch. To do so, we extend the smart contracts shown in the previous section to keep a global counter for the epoch. Moreover, all writes are added in a separate variable that keeps a separate list per epoch. The epoch counter is increased automatically by the `write-blockchain` method in the smart contract every $|e|$-th write call.

One additional parameter of offline verification is that an epoch does not need to be immediately verified by a peer once it is full (i.e., once it is closed). Instead, a peer can have a maximal number of closed but not-yet verified epochs per shard (called number of unverified epochs Δe). In case that the number of unverified epochs for a peer is growing larger than Δe, it calls the `halt(s)` operation implemented as a method in the smart contract on the storage layer which blocks all other peers from further writing into the shard s (i.e., no more new epochs are created) until the peer caught up with verifying the shard. Once the number of unverified epochs is smaller than Δe, the peer calls `continue(s)` implemented as a smart contract on shard s.

One could argue that the `halt(s)` method opens up a backdoor for malicious peers to block other peers. However, since the blockchain will keep track of those calls other peers can detect this behavior (as part of a potential audit). Further, both parameters ($|e|$ and Δe) are important as they allow to optimally tune the verification to the workload characteristics of a given application as discussed later.

In Figure 8.6, we show an example for offline verification. The example shows the write-sets of all peers over different epochs of one shard (Shard 2), as well as the read/write-set of peer C that should be verified. All epochs have an epoch size of $|e| = 2$ and we have four epochs in total. The first epoch $e = 0$ has been already successfully verified, while epoch $e = 1$ and $e = 2$ are closed but not yet verified (i.e., $\Delta e = 2$). The current epoch $e = 3$ is still open since only one write was executed so far.

For verifying the next epoch in a shard, the database layer of a peer runs a verifier thread that runs in continuous intervals and keeps track of the last verified epoch for that peer (e.g., $e = 0$ in the example). The verifier thread calls the `get-writeset` method for a shard using the last unverified (and closed) epoch as a parameter. In our example, peer C calls `get-writeset(s=2, e=1)` since $e = 1$ is the oldest not-yet verified epoch. The call returns the write-set (`write(y,1)`→ `tx-3`, (`write(x,3)`→ `tx-4`). In order to make sure we have the correct write-set, peer C needs to read it from the majority of peers (not shown in the example), which store that shard. For verifying its own read-/write-set, peer C compares its own read- and write-set of the same epoch ((`read(x)` → `2`) against the write-set of epoch $e = 1$ and all previous epochs; i.e., the write-set of epoch $e = 0$ in our example ((`write(x,1)`→ `tx-1`, (`write(x,2)`→ `tx-2`)).

The verifier thread then checks, if the read-calls in the read-set of peer C match the value of the last committed write-operation in the global read-/write-set (to avoid spurious reads) and if all write-calls are found in the global read-/write-set (to avoid dropped writes). In order to enable an efficient offline verification, a system peer caches (parts) of the write-sets that it reads for verification in the past. What exactly needs

to be cached depends on the isolation level. Under sequential consistency, only the last write to a key needs to be cached while under eventual consistency all pending writes for a key plus the latest committed write to the same key are cached. Older committed writes can be evicted from the cache.

Finally, if the checks fail, the database layer sets a flag for that shard to indicate that it is in a corrupted state. The application on top then has to decide how to react. One idea is that the client calls the operation `halt(s)` which means that all further operations (from all peers) on the shard are blocked since the database is in a corrupted state and the tamper-proof log of the shard must be audited to see what went wrong. A way to restore the shard to a non-corrupted state is to reset the shard to the last verified epoch. Restoring the database from a blockchain is possible since the blockchain keeps all writes. Discussing possible consequences and other variants for reactions is beyond the scope of this paper though.

Discussion. In the following we first discuss how to set the parameters $|e|$ as well as Δe for offline verification and then discuss what influence the parameters have on the performance that BlockchainDB can provide.

For setting $|e|$, we found out that the epoch size should be set at least to the number of transactions that fit into a block of the underlying blockchain (which allows to verify all transaction of a block in one epoch). This information can be retrieved from many blockchain systems and thus be used to configure $|e|$. Setting $|e|$ to a smaller value typically decreases the throughput since new generated blocks can not be completely filled with transactions.

Setting the second parameter, Δe, has a different effect. If $\Delta e = 0$, a peer will not accept any new write-operation once an epoch is closed (i.e., all peers must verify the last epoch first before new writes are accepted). Hence, Δe can be used to overlap the actual writes and verification. Moreover, Δe can be also used for mitigating issues resulting from skew between different peers. For example, a straggling peer of one party could block a faster peer of another party just because it needs more time for executing the offline verification procedure. This is an issue in blockchains where multiple parties that come with different hardware characteristics participate in the network. Thus setting Δe to higher values helps to mitigate the skeweness of different peers in the system.

For understanding the performance impacts of these two parameters, it is important to note that only the verification procedure itself is batched while the actual operations (i.e., the puts/gets of clients) are executed without batching as described in Section 8.4. To that end, offline verification does not increase the latency of individual put/get-operations but it can have a negative impact on the overall throughput if $|e|$ and Δe are set to a

too low value. However, when setting $|e|$ and Δe to a too high value clients will operate for a longer period of time on an *unverified* state of a shared table and the time before a problem (i.e., a dropped put-operation or a spurious get-operation) can be detected increases. Thus, tuning those parameters is extremely important.

In Section 8.7.5, we show how to set these parameters in an optimal manner for a given workload and setup of peers.

Table 8.1: Parameters of Evaluation

Parameter	*Description*
shardCount	Number of shards in a table.
repFactor	Number of replicas that are stored for each shard, each replica is stored on a separate (full) peer.
consistency	Consistency level configured for a shared table.
numPeers	Total number of (full) peers that participate in the BlockchainDB network.
numClients	Total number of clients sending put/get operations.
workload	The ratio and distribution of put/get operations.
opsCount	Total number of operations issued in total.

8.7 Experimental Evaluation

In order to evaluate the different characteristics of BlockchainDB, we executed multiple experiments with the YCSB Benchmark [38] that provides workloads with different read/write characteristics. The main goal of the experiments is to study the effects of the different techniques implemented in BlockchainDB on the performance, and also to show that BlockchainDB allows applications with its configuration parameters to trade performance over trust.

8.7.1 Setup & Workloads

For the evaluation, we implemented a BlockchainDB prototype in *Java 8*. All experiments used an Ethereum backend (with Geth/v1.8.23-stable). Furthermore, we ran all experiments in Azure on virtual machines with 16 vcpus and 32 GB memory, running Ubuntu 16.04 LTS. To configure the blockchain network for a new shard in BlockchainDB, we used similar genesis parameters as reported in [44].

In every experiment, we varied different system parameters, which are briefly described in Table 8.1. We will explain the parameters values, we have used for each experiment separately. Furthermore, in our experiments, we first concentrate on the performance characteristics of BlockchainDB. To isolate the effect of verification, we turned verifi-

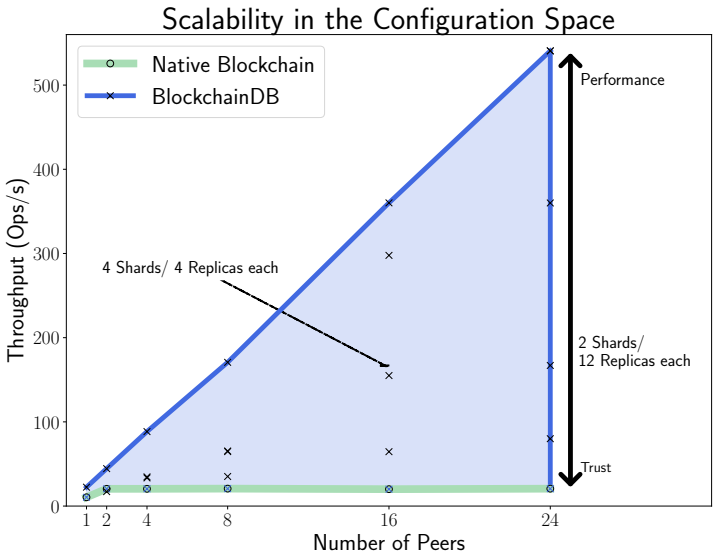


Figure 8.7: Scalability of BlockchainDB

cation off in these experiments. In the second set of experiments, we then studied the overhead of verification as well as the different verification strategies in detail.

8.7.2 Exp. 1: Scalability with Peers

In this experiment we evaluate the performance of BlockchainDB when the size of the network is increased (i.e., more and more peers join a BlockchainDB network to share data). In classical blockchains the performance of the system heavily degrades, since more peers increase the storage and consensus overhead of the network as shown in[44]. We also made a similar observation in our experiments shown in Figure 8.7 as we discuss below.

To make a fair comparison between different configurations of BlockchainDB, we use a fixed number of n peers (i.e., a fixed amount compute and storage resources) and vary the sharding configuration of one table. For example, for a setup with a fixed number of 16 peers (shown as 16 on the x-axis of Figure 8.7), we evaluated the different configurations shown as individual points above the x-tic 16. The configurations used for 16 peers are: (1 shard with 16 replicas), (2 shards, each 8 replicas), (4 shards, each 4 replicas), ..., (16 shards, each 1 replica). We repeated the experiment for other setups with a lower/higher number of peers (ranging from $1 - 24$). The setups (i.e., number of peers used) are shown as different x-tics/x-labels in Figure 8.7 and all throughput

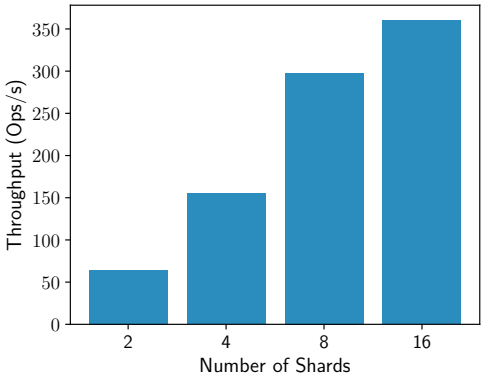


Figure 8.8: Effect of Sharding on Throughput

resulting from using different sharding configurations for one setup are shown as points on the vertical line above the corresponding x-tic.

For running the experiment, we created a shared table with the number of shards and distributed them to the different peers as given in the configurations and inserted into each shard initially 4,000 tuples. This results in the fact that in total the same number of tuples has to be stored in BlockchainDB for the different configurations used for a given number of peers. For example, for a fixed number of 16 peers, 64,000 tuples are being stored in total in the table for any sharding configuration — when using 16 shards/1 replica on the one extreme but also when using 1 shard/16 replica the other extreme.

In Figure 8.7, we show the resulting throughput of BlockchainDB for the different setups each using a fixed number of peers (x-axis) and for each setup using different configurations (as indicated by the different points along the y-axis) as discussed before. The number of clients used in this experiment is equal to the number of peers (i.e. for every new peer a new client is added; i.e., *numPeers=numClients*). Furthermore, we use *consistency= sequential* and set *opsCount=1000* $\cdot$ *numClients* using *workload= 100% write/ 0% read*; i.e., every client sends a total of $1k$ put-operations to the database and waits until these put operations have been committed (e.g. by sending one `get` at the end to force that the writes are committed under sequential consistency). As explained above, the replication factor was set to $repFactor = \frac{numPeers}{shardCount}$.

Figure 8.7 shows the results of the experiment. Since BlockchainDB allows applications to apply different partitioning and replication strategies for shared tables, we can better trade performance and trust characteristics depending on the applications requirements. The two extremes *high trust* and *high performance* are highlighted by

the green and blue line, respectively. For the green line, BlockchainDB only uses one shard to store the data and each newly added peer stores yet another replica of this shard and participate in its corresponding blockchain. Hence, this configuration represents a classical blockchain configuration and shows similarly worse scalability. In the other extreme, *high performance*, every peer that is added also adds a new shard to the network. This configuration is comparable to a classical distributed database, in which the parallelism and throughput of the system is increased with every new node. Yet, since only one replica exists per shard, the application does not get any trust guarantees.

An interesting aspect that is also shown in Figure 8.7 is that BlockchainDB can provide other configurations "in the middle" that provide a trade-off between trust and performance (shown as the area shaded in light-blue). For example, when using a configuration with 4 shards and 4 replicas each for 16 peers (shown as one point in the Figure 8.7) we can provide a 7× speedup over the full-replicated baseline and still provide some trust guarantees.

8.7.3 Exp. 2: Effect of Sharding

In this experiment, we show the effects of sharding where we fixed the number of peers in the network to 16 and varied the number of shards per table (with a fixed replication factor of 4 per shard). Different from the experiment before, we want to show that sharding provides a speed-up even for a setup with fixed trust guarantees.

For running the experiment, we created a shared table and increased the number of shards from 1 to 16 where we filled each shard initially with 4,000 tuples per shard (i.e., 64,000 tuples are in the table in total for 16 shards) to simulate a setup of BlockchainDB with fixed trust-guarantees. Furthermore, we used a constant replication factor of 4 (as mentioned before). The number of clients in this experiment is fixed to *numClients=16*. We used the same workload as in the previous experiment (*workload= 100% write/ 0% read*) but each client sends 2000 operations (i.e. *opsCount=32,000*) using *consistency=sequential*.

Figure 8.8 shows the result of this experiment. As we can see, the throughput of BlockchainDB increases linearly with the number of shards. This is because the degree of overall parallelism is increased.

8.7.4 Exp. 3: Effect of Consistency Levels

In this experiment, we show the effect of using different consistency levels for clients. For this experiment, we used the parameters shown in Table 8.2.

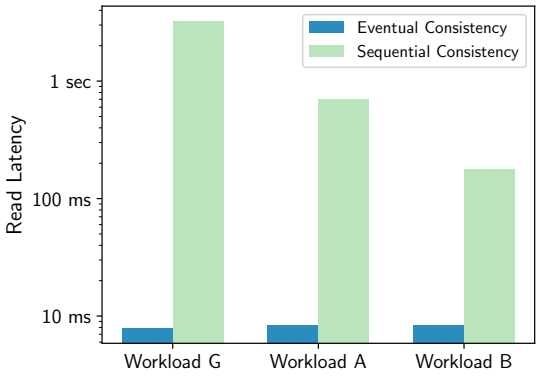


Figure 8.9: Effect of Consistency on Latency

Furthermore, in this experiment we are using different workload mixes: workload G (new mix/not in YCSB) - a write-intensive workload (*95% write/ 5% read*), workload A (same as in YCSB) - a workload with same amount of reads and writes (*50% write/ 50% read*), and workload B (same as in YCSB) - a read-intensive workload (*5% write/ 95% read*). In order to show the effect of the two main consistency levels (eventual and sequential), we measured the read-latency on a client for the different workloads.

Table 8.2: Parameters for Exp. 3

parameter	*value*
shardCount	2
repFactor	2
numPeers	4
numClients	4
opsCount	8000

Exp 3a: Sequential vs. Eventual Consistency.

Figure 8.9 shows the resulting latencies for the different workloads. As we can see, the read latency under eventual consistency is not affected by the change in workload at all (which we also expected). For, sequential consistency, however, we can see that the performance increases with a higher number of read-operations. This is a direct consequence of the fact that with a lower number of writes we also have a lower frequency of blocking read-operations. For a read intensive workload, we see a decrease in latency for sequential consistency by two orders of magnitude compared to the write-intensive workload since there are only a few write-operations that could potentially block the execution of subsequent read-operations on the same key.

Table 8.3: Parameters for Exp. 4

parameter	*value*
shardCount	2
repFactor	2
numPeers	4
numClients	4
consistency	sequential
opsCount	4000

Exp 3b: Bounded Staleness.

While sequential consistency guarantees a client to see fresh values, it has a much higher latency than eventual consistency since it forces a client to wait until pending writes for a key are committed. In contrast, eventual consistency has as significantly lower latency, but does not provide any guarantee on the staleness of retrieved values. As described in Section 8.4, eventual consistency with bounded staleness allows a client to trade off staleness for improved read latency.

We therefore repeated the experiment above but used bounded staleness. In this experiment, we set the size of the write-queue to different values ranging from 0 to 900. The resulting effects on latency are shown in Figure 8.10. We see that the average read latency decreases as we increase the staleness. For example, while waiting until *all* pending transactions have been committed (i.e., staleness is 0) causes a maximal delay of about 35s, tolerating around 900 pending puts can improve the latency to around 20s.

In the extreme case, bounded staleness 0 gives us the same guarantees as sequential consistency. However, we see that it results in a higher latency as for sequential consistency as shown in Figure 8.9 since sequential consistency blocks lazily if a read for a pending write arrives.

8.7.5 Exp. 4: Verification Overhead

To measure the overhead of verification, we performed two experiments. First, we compared the overhead of online and offline verification. Second, we show the effects of the different parameters for offline verification.

Exp. 4a: Online vs. Offline Verification.

In this experiment we evaluated the performance of the different verification strategies supported by BlockchainDB. Table 8.3 indicates the parameters used in this experiment.

For showing the overhead of verification, we report the throughput based on the time it takes to commit and verify all 4,000 operations. As a baseline, we show a configuration without any verification (called no-verification).

As can be seen in Figure 8.11, *online verification* achieves the lowest throughput, since it directly waits until a put-operation (i.e., a transaction executing a put) is committed to verify the operation, which prevents other put requests from being executed. With the help of offline verification the throughput can be increased as more transactions are added into one block. As explained earlier the overall throughput depends on the two parameters of the offline verification: epoch-size $|e|$ and number of unverified epochs Δe.

In this experiment we use an optimal configuration and set $|e| = 100$ and $\Delta e = 10$ which results in a throughput of about approx. 40 put operations per second. As can be seen this value is similar to the throughput of no-verification. This is because the verification can be performed once a new block is mined for all operations in the block. This is similar to the time it takes to commit a transaction to the DB plus a small overhead for verifying the retrieved read/write-sets (which is negligible since reading the read/write-sets is fast).

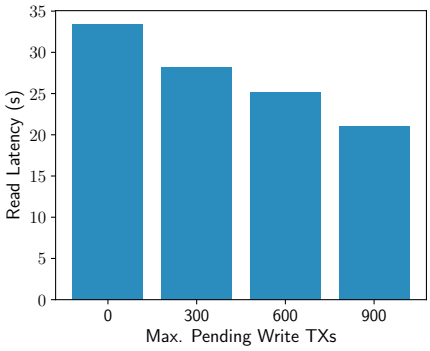


Figure 8.10: Effect of Staleness on Latency

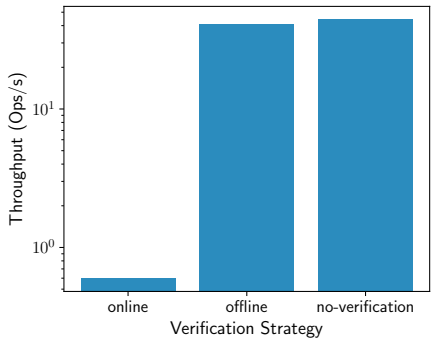


Figure 8.11: Online vs. Offline Verification

Exp. 4b: Offline Verification Parameters.

In the second experiment we fixed the verification strategy to *offline verification* and varied the two parameters $|e|$ and Δe.

In a first micro-benchmark, we evaluated how $|e|$ effects the throughput for a client and set $\Delta e = 0$, i.e., only one epoch can be unverified at a time. For running the micro-benchmark we use a table with a single shard to show the effects of setting $|e|$ on an isolated shard. Further, we varied the replication factor (i.e., number of peers a shard is replicated to) to study its influence on the overall throughput.

As we can see in Figure 8.12, when using a replication factor of one (1 Peer, green line) the throughput increases significantly with increasing epoch size until $|e| = 70$ is reached, which corresponds to the maximum number of transactions that fit into one block (i.e., the block size) of the underlying blockchain. When increasing $|e|$ further, the throughput increases much slower (and almost stagnates). Furthermore, if multiple peers participate in a shard the throughput is higher in total. We can see this effect in Figure 8.12 by the second line (2 Peers, blue line). The reason here is that the verification overhead is distributed across two peers and thus each peer only needs to verify half of the put-operations of an epoch on average. This leads to an overall higher throughput since the total elapsed time for verification is shorter.

In a second micro-benchmark we wanted to show the effect of the Δe parameter. As explained previously, the Δe parameter determines the number of unverified epochs a peer tolerates. In this experiment, we thus execute the same benchmark as before with two peers but this time one fast and one slow peer in order to analyze the effect of potential resource skew (e.g., a slower network or less computational power for one). A $skew = x$ means that the slower peer is only able to verify transactions with a lag of x blocks on average behind the faster peer. In order to show the sensitivity of the overall throughput on Δe, we set $|e| = 70$ (i.e., the optimal $|e|$ of the previous experiment) and vary the Δe parameter from 1 to 20.

In Figure 8.13, we show the throughput (including its standard deviation for 10 runs) for different skew factors where the slower peer has a lag of 4, 8, and 14 blocks on average. As we can see, the throughput improves with increasing Δe whereas for a higher skew a higher Δe is required. As our experiments show, Δe should be set according to the lag of the slower peer; e.g., the maximal throughput for a lag of 14 is achieved with $\Delta e = 14$ whereas smaller lags (skew=4 and skew=8) can also tolerate a smaller Δe. In general, we see that if Δe is set to a value smaller than the lag, then the faster peer is always slowed down by the slower peer which decreases the overall throughput.

8.8 Related Work

Related work spans three major areas, namely, Verifiable Databases, Scalable Blockchains, and Distributed Databases. For Distributed Databases, there is a long line of work that covers relevant topics such as replication, sharding and peer-to-peer approaches. Due to space constraints, we omit a detailed discussion and refer to [135] for an overview.

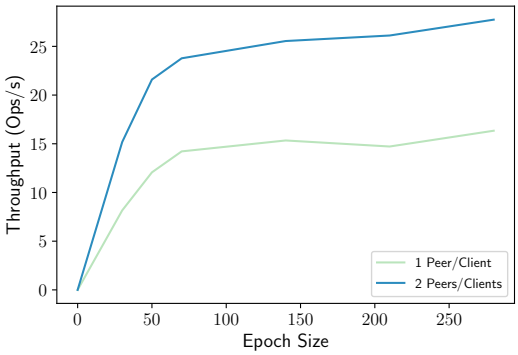


Figure 8.12: Offline Verification with varying $|e|$

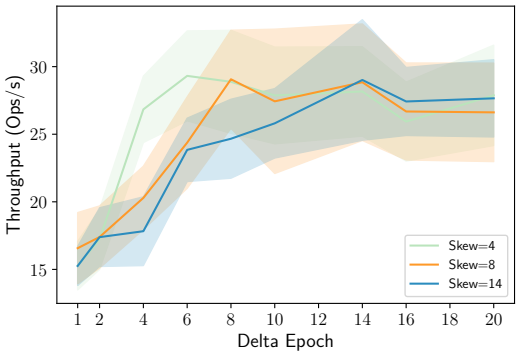


Figure 8.13: Offline Verification with varying Δe

Verifiable Databases. The closest work to BlockchainDB is work done by Allen et al. in [68]. In [68], the authors propose the idea of "Databases and tables that can be shared and verified". While their work makes use of the same abstractions of *shared tables* they differ in how they implement these abstractions. While [68] also uses blockchains to implement a voting/consensus schema they store the actual data in a traditional database on every peer and only a digest in the blockchain. In contrast, in BlockchainDB we store all data in blockchains, such that BlockchainDB not only uses the consensus and verification features provided by the blockchain but also the capability of having all data in a tamper-proof ledger that allows us to audit all changes to the database. Another major difference is that BlockchainDB allows applications to navigate the trade-off between trust and scalability when using blockchains as a shared database.

BlockchainDB also relates to previous work done on verifiable databases in the context of outsourcing [14, 18, 19, 84, 175, 190, 191]. This body of research addresses the question of how to securely delegate the management of data to untrusted third parties, such that the third party cannot manipulate the data or the result of queries on that data. In BlockchainDB peers face the same challenges when accessing data on shards stored at remote peers. However, in BlockchainDB we do not aim to modify the underlying blockchain and thus build our verification protocol on top.

Scalable Blockchains. The second area of related work is in the context of blockchain systems, where various proposals have been made to improve the scalability and performance of a blockchain. A good overview of the bottlenecks and approaches to scale blockchains are discussed in [40]. In the following, we discuss recent results not covered in [40].

For example, several new protocols have been developed to make use of sharding as part of the blockchain consensus protocol [4, 94, 109, 145, 146, 187] to address the scalability challenge. BlockchainDB differs from this line of work as it does not propose a new consensus protocol, but implements sharding on top of existing blockchains. Hence, these new blockchain systems could be also used by BlockchainDB as a backend and improve the overall performance of the database. A further difference is that BlockchainDB acts as an abstraction layer for clients, such that normal users do not need to deal with new interfaces or programming models that might be introduced by a new blockchain system.

Other proposals to overcome the scalability problem of blockchains discuss the usage of off-chain computation [34, 47, 48, 125]. While BlockchainDB shares the concepts of moving certain functions out of the blockchain, it does that on top of the blockchain layer and not as part of it.

Another direction of work is that systems aim to add blockchain-like functions to existing distributed and replicated databases. A prominent representative for this line of work is BigChainDB [115] which builds on MongoDB. While BigChainDB shows that it can provide a higher performance than native blockchain systems, it is being constantly under critique to not provide the same trust guarantees and fault-tolerance model as native blockchains. Some of the original shortcomings have been recently addressed in a newer version by using Tendermint to achieve Byzantine fault tolerance. Different from BigChainDB, BlockchainDB hence has chosen another route and instead builds directly on top of blockchain systems and their trusted execution model.

Finally, recent papers [151] also looked into blockchains with a database angle and add database techniques into the blockchain (e.g., re-ordering transactions for higher throughput). Same as before, BlockchainDB does all its optimizations on top of the blockchain layer and not as part of it.

8.9 Conclusion & Future Work

In this paper, we presented BlockchainDB, which introduces a database layer on top of blockchains to participate in data sharing scenarios. Our experiments show that BlockchainDB can provide up to two-orders of magnitude higher throughput than native blockchains and allows to better scale-out with the number of peers. At the moment, BlockchainDB only provides a key/value interface on top of shared tables.

In future, we thus want to extend the query interface to shared tables to full SQL with verifiable transactional semantics.

8.10 Acknowledgements

This research work has been funded by the German Federal Ministery of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity.

9 BlockchainDB - Towards a Shared Database on Blockchains

Abstract

In this demo we present BlockchainDB, which leverages blockchains as storage layer and introduces a database layer on top that extends blockchains by classical data management techniques (e.g., sharding). Further, BlockchainDB provides a standardized key/value-based query interface to facilitate the adoption of blockchains for data sharing use cases. With BlockchainDB we can thus not only improve the performance and scalability of blockchains for data sharing but also decrease the complexity for organizations intending to use blockchains for this use case.

Bibliographic Information

The content of this chapter was previously published in the peer-reviewed work Muhammad El-Hindi, Martin Heyden, Carsten Binnig, Ravi Ramamurthy, Arvind Arasu, and Donald Kossmann. "BlockchainDB - Towards a Shared Database on Blockchains." In: *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*. Ed. by Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska. ACM, 2019, pp. 1905–1908. DOI: 10.1145/3299869.3320237. URL: https://doi.org/10.1145/3299869.3320237.

9.1 Introduction

Motivation. Blockchain (BC) technology emerged as the basis for crypto-currencies, like Bitcoin [126] or Ethereum [29], allowing parties that do not trust each other to exchange funds and agree on a common view of their balances. With the advent of smart contracts, blockchain platforms are being used for many other use cases beyond crypto-currencies and include applications in domains such as governmental, healthcare and IoT scenarios [16, 20, 169].

An important aspect that many scenarios have in common is that blockchains are being used to implement shared data access of parties that do not trust each other. For example, one use case is that the blockchain is used for tracking of goods in a supply chain. What makes blockchains attractive in those scenarios are two main characteristics: First, blockchains store data in an immutable append-only ledger that contains the history of all data modifications. That way, blockchains enable auditability and traceability in order to detect potential malicious operations on the shared state. Second, blockchains can be operated reliably (tolerating byzantine failures) in a completely decentralized manner without the need to involve a central trusted instance which often does not exist in the use cases mentioned before.

However, while there is a lot of excitement around blockchains in industry, they are still not being used as a shared DB in many real-world scenarios. This has different reasons: First and foremost, a major obstacle is their limited scalability and performance. Recent benchmarks [44] have shown that state-of-the-art blockchain systems such as Ethereum or Hyperledger that can be used for general applications can only achieve $10's$ or maximally $100's$ of transactions per second which is often way below the requirements of modern applications. Second, blockchains lack easy-to-use abstractions known from data management systems such as a simple query interface as well as other guarantees such as well-defined consistency levels that guarantee when/how updates become visible. Instead, blockchains often come with proprietary programming interfaces and require applications to know about the internals of the blockchain system to decide on the visibility of updates. For example, when using Ethereum, clients have to manually check if their updates have successfully been committed to the main branch by waiting a certain number of blocks.

Contributions. In this demo, we present BlockchainDB to tackle the before-mentioned issues. The main idea is that BlockchainDB leverages blockchains as storage layer. That way, existing blockchain systems can be used (without modification) as a

temper-proof and de-centralized storage. On top of the storage layer, BlockchainDB adds a database layer that implements the following functions:

- *Partitioning and Replication*: A major performance bottleneck of blockchains today is that all peers hold a full copy of the state and still only provide (limited) sharding capabilities. In BlockchainDB, we allow applications to define in the database layer how data is replicated and partitioned across all available peers. Thus, applications can trade performance and security guarantees in a declarative manner.

- *Query Interface and Consistency*: In the database layer, BlockchainDB additionally provides easy-to-use abstractions including different consistency protocols (e.g., eventual and sequential consistency) as well as a simple key/value interface to read/write data without knowing the internals of a blockchain system.

In addition to these two functions, BlockchainDB comes with an *off-chain verification procedure*. The idea of the verification procedure is that clients can detect potential misbehaving peers in the BlockchainDB network. This is needed since not all BlockchainDB peers hold the full state and the storage layer of a remote peer could potentially drop put-operations or return spurious values for get-operations.

By introducing a database layer on top of an existing blockchain as storage layer, BlockchainDB is able to not only provide higher performance, but also to massively decrease the complexity for organizations, that intend to use blockchains for data sharing. While the concept of moving certain functions out of the blockchain into additional application layers has been studied previously (e.g., [34, 47, 48, 125]), to the best of our knowledge, BlockchainDB is the first system to provide a fully functional database layer on top of blockchains.

Outline. The remainder of this paper is organized as follows: First, in Section 9.2 we provide an overview of the BlockchainDB architecture as well as the provided trust guarantees. Then, in Section 9.3 and 9.4 we discuss the details of BlockchainDB's database and storage layer. Finally, in Section 9.5 we conclude with a description of our demo scenario that we intend to present to the audience.

9.2 Overview

In the following, we first give an overview of the architecture of BlockchainDB and then discuss which attacks can be mitigated by our off-chain verification.

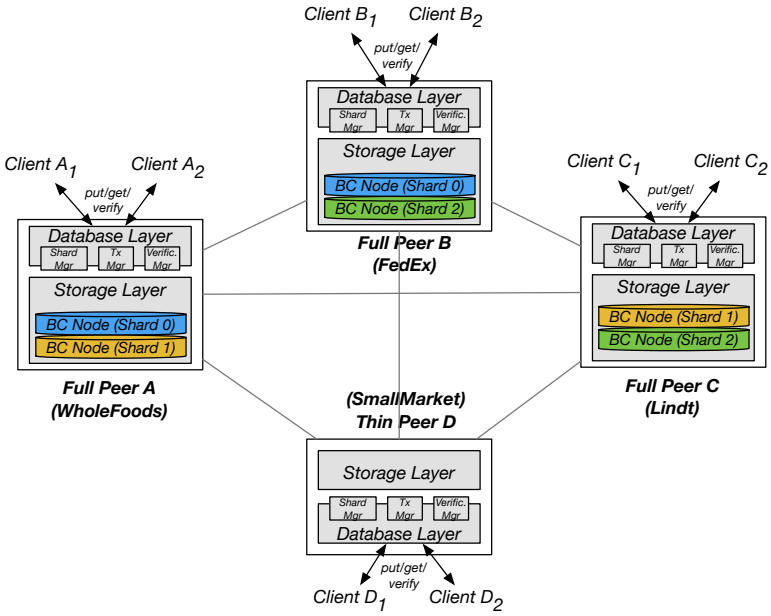


Figure 9.1: A typical BlockchainDB Network

9.2.1 System Architecture

Figure 9.1 shows a typical deployment of BlockchainDB across a network of four different BlockchainDB peers (i.e., untrusted parties) that access the same shared data. Similar to permissioned blockchains, in BlockchainDB we assume that the parties who want to share data are previously known to each other.

In order to participate in a BlockchainDB network and allow clients of a party to read/write data into a shared database/table, a peer in BlockchainDB can be either deployed as a *full peer* which hosts data and participates in a shard with its storage layer or as a *thin peer* which only hosts a database layer (i.e., the peer's storage layer does not store data locally). Having thin peers enables parties with only limited resources to participate in a BlockchainDB network and access the shared database or table. In the following, we explain how clients interact with a BlockchainDB as well as the functionality of each layer.

Clients. Instead of interacting with a blockchain peer directly, in BlockchainDB clients interact with the database layer through a simple `put/get` interface to read from or write to the shared state. Furthermore, clients can use the `verify` method of the database layer to trigger an off-chain verification procedure for the read/write-set of that client (since the last verification call) to detect potential misbehaving peers in a BlockchainDB network (e.g., to detect if another BlockchainDB peer — more

precisely its storage layer — returns a spurious read or drops a put). An example to intuitively explain verification is discussed in the next section.

Database Layer. The database layer in BlockchainDB is mainly responsible to execute the `put/get` calls from the clients. If a `put/get` call comes in, the database layer uses the *Shard Manager* to decide to which (copy of a) shard the operation should be directed to. The shard can be either stored in its local storage layer or remotely in another BlockchainDB peer depending on a pre-defined distribution scheme. Currently, BlockchainDB implements a hash-based sharding approach, in which the user defines the number of shards and their allocation to BlockchainDB peers when creating a new shared table. Another major difference of BlockchainDB and a pure blockchain network is that the database layer of BlockchainDB implements a *Transaction Manager* that implements well-known consistency levels (e.g., eventual consistency and sequential consistency). That way, clients not only get a defined behavior for concurrent read/writes without the need to know the internals of blockchains, but the database layer can additionally re-order/batch reads/writes depending on the chosen consistency level to optimally leverage the underlying blockchain-based storage layer and thus further improve the performance.

Storage Layer. The storage layer serves as a persistent, auditable storage backend of BlockchainDB using existing blockchain systems as temper-proof storage. As depicted, the storage layer of BlockchainDB is able to parallelize data processing across different shards whereas each shard is implemented using a separate blockchain network which replicates its state to multiple (but not necessarily all) peers using the blockchain internal consensus protocols. For example, in Figure 9.1 BlockchainDB uses in total three different blockchain networks (one for each shard) while each shard is only replicated to two peers.

9.2.2 Attacks & Trust Guarantees

In BlockchainDB, we aim to detect two categories of attacks.

First, a common problem in a shared data scenario is that one party might change the shared state to its advantage. For example, in a supplier-scenario the party who delivers the products might update the price of the products after an order has been placed from the other party. In BlockchainDB such a modification-after-the-fact could be detected since the storage layer uses blockchains to store the history of all updates in a temper-proof ledger. In consequence, any later modification of data will be transparent to all parties. Clearly, the logic to verify the validity of updates has to be implemented

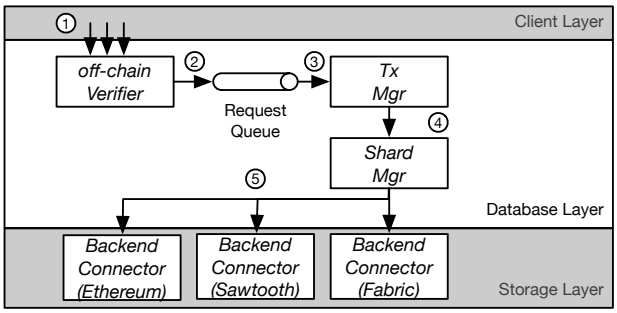


Figure 9.2: Database Layer

in the application layer on top of BlockchainDB, but the important fact is that the database guarantees the audibility.

Second, individual BlockchainDB peers do not hold the full database state and thus redirect a subset of reads/writes to the storage layer of other peers. However, these peers might be potentially malicious and drop puts or return spurious values for get-operations. To detect this type of attack, BlockchainDB tracks all issued operations in the blockchain and provides clients with an off-chain verification procedure that checks whether the read/write-set has been correct.

The off-chain verification procedure is implemented in the database layer of and comes in two flavors: In online verification, a client can call a separate verify method after a put/get-operation that checks the validity of the last operation. For example, to verify a get-operation the value for a given key is retrieved from the majority of peers. However, online verification comes with significant additional overhead. We therefore additionally support (a deferred) offline verification procedure which verifies reads/writes in batches. The deferred verification procedure provides less overhead than the online verification procedure. However, clients might work (for a limited amount of time) on an unverified database state.

9.3 Database Layer

In the following we explain the database layer and briefly discuss the involved components as depicted in Figure 9.2.

As mentioned before, when a new shared table is created in BlockchainDB, the user needs to specify the number of shards and their sizes (i.e., to define how many replicas of a shard exist). The database layer of BlockchainDB can setup the required shards using

a `ShardController` component (not shown in Figure 9.2) that automatically creates a new blockchain network. As shown by Dinh et al. [44], a higher number of peers in a blockchain network decreases the throughput, however, it offers higher security since the data is protected by a larger network.

For accessing a shared table, clients send requests to their local BlockchainDB peer using a simple key/value interface. Requests arriving from the client in the database layer are first received by the off-chain verifier ① that records the unverified reads/writes of a client for online/offline verification (as discussed before). The verifier then forwards the request to an internal *RequestQueue* ②. Next, the `TransactionManager` polls the requests from the queue and processes them according to a specified consistency level[1] ③. Currently, we support sequential and eventual consistency. The different consistency levels differ in how quickly the database layer can process operations. For example, for eventual consistency a get-operation is immediately answered, however, no guarantee is given that a potential outstanding put-operation has already been committed to the blockchain. In sequential consistency, the database layer needs to execute put/get-operations in order and thus, potentially blocks a get-operation until an outstanding put-operation has been committed to the blockchain. Once a put/get-operation is ready to be sent to the storage layer, it is forwarded to the `ShardManager` ④. The `ShardManager` service has two main purposes: First, it determines the correct shard for a given key and second, it is responsible for sending the request in parallel to the storage shards. To do this multiple `BackendConnectors` are being used ⑤ that allow BlockchainDB to write data into the underlying storage layer as discussed in the next section.

9.4 Storage Layer

A shared table in BlockchainDB has a key/value data model where key is the primary key and value can represent the payload of the table. For keys and values, the database layer provides different data types that an application can choose from when creating a new table (such as `INT`, `DECIMAL`, `STRING`). For values, BlockchainDB also supports complex `JSON`-based data types that are comprised of lists and nested structures.

For storing shards of a table in the storage layer, the database layer converts the key/value payload into a byte representation. We found out that representing the data in a byte-format before storing it in a blockchain backend not only leads to decreased storage cost, but also allows for more efficient processing on the blockchain. In the future,

[1] The consistency level can be specified for each shared table.

we plan to study the application of lightweight compression techniques in the database layer to further optimize writing and retrieving data from the storage layer.

In order to support a new blockchain backend to store shards, BlockchainDB requires a minimal smart contract definition which provides a simple `put/get` access to the blockchain state via a so called `BackendConnector`. Different `BackendConnectors` are currently available for Ethereum, Hyperledger Sawtooth as well as Hyperleder Fabric. In the following, we show a simplified extract of the smart contract code that is used by our Ethereum `BackendConnector`.

```
contract KVContract {
  // state variables and constructor omitted
  function get(bytes memory key)
  public view returns(bytes memory value){
    return data[key];}
  function put(bytes memory key, bytes memory value)
  public returns(bool success) {
    data[key] = value; return true;}
}
```

9.5 Demonstration Scenario

In our demo we show-case how BlockchainDB can support data sharing in various scenarios by optimizing the system for a given workload. To this end, our demo provides two interfaces, an *administration* and a *simulation* interface.

Administration Interface. This interface (left side in Figure 9.3) shows the current system status and performance (in terms of throughput/latency) for different workloads (read-heavy vs. write-heavy). For the demo, we setup BlockchainDB networks with up to 24 BlockchainDB peers. Each peer represents a participant in the network that shares data with other peers. The administration interface enables the user to change system parameters of BlockchainDB like the replication factor per shard (1-24), number of shards per table (1-24) or the consistency-level (eventual or sequential consistency) and see how the system performance is affected.

Simulation Interface. This interface allows the user to simulate attacks and verify that they were successfully detected by BlockchainDB. While our simulation is generic, the simulation components can be mapped to any use case that benefits from data sharing, such as supply chain, health networks, etc. With the help of the simulation interface (right in Figure 9.3), we present three scenarios in our demo application: (1) An attacker

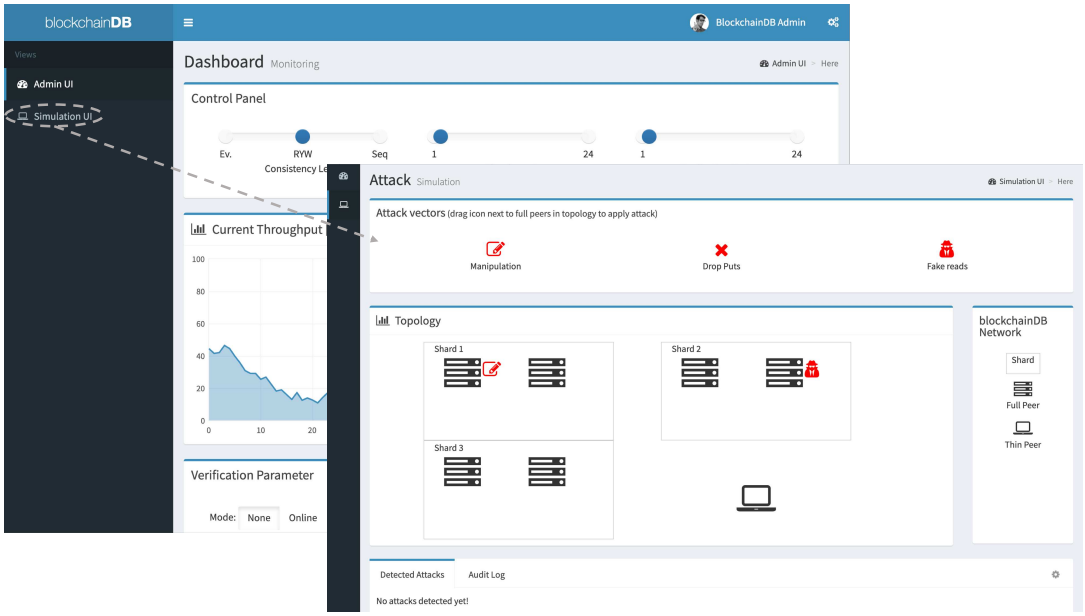


Figure 9.3: Admin (left) and Simulation Interface (right)

wrote manipulated data into the shared table, which was detected by a participant in the network. An auditor wants to account which participant wrote the responsible value into the shared table. One can think of BlockchainDB in this context, as a black box that reliably records all writes and enables a later audition of all actions. (2) A malicious peer tries to prevent other peers from writing to the shared table by dropping their writes. Users can observe, that the attack has been detected and logged in the simulation interface. (3) A malicious peer returns spurious values (not actually stored in a shared table). As such attacks are detected, they will be also logged in the simulation interface.

10 Towards Merkle Trees for High-Performance Data Systems

Abstract

Merkle Trees (MTs) (and their variants) are widely used for building secure outsourced data systems. The adoption of MTs for high-performance data systems, however, uncovered major performance challenges. First and unlike classical data structures, Merkle Trees involve expensive cryptographic operations and are thus CPU-bound. Second, they are not well suited for modern multi-core CPUs because they introduce a single point of contention making MTs hard to parallelize. While recent work aimed at replacing Merkle Trees to circumvent their performance problem, we suggest new techniques to speed-up this ubiquitous data structure and achieve high-performance. In this paper, we present initial results showing that in contrast to common wisdom it is indeed possible to build high-performance MTs with orders of magnitude performance improvements.

Bibliographic Information

The content of this chapter was previously published in the peer-reviewed work Muhammad El-Hindi, Tobias Ziegler, and Carsten Binnig. "Towards Merkle Trees for High-Performance Data Systems." In: *Proceedings of the 1st Workshop on Verifiable Database Systems, VDBS '23, Seattle, WA, USA, June 23, 2023*. Ed. by Tien Tuan Anh Dinh, Beng Chin Ooi, and Xinying Yang. ACM, 2023, pp. 28–33. DOI: 10.1145/3595647.3595651. URL: https://doi.org/10.1145/3595647.3595651.

10.1 Introduction

The Pervasiveness of Merkle Trees. Merkle Trees (MTs) [121] (and their variants) are widely used for building secure outsourced data systems. Compared to other approaches (such as cryptographic accumulators [167] or signature aggregation [136]), they efficiently support all relevant operations including updates and the verification thereof. This makes MTs the go-to choice for update-intensive applications, such as secure outsourced key-value stores (e.g., [13, 14, 17, 156]). Using a Merkle Tree as an authenticated data structure, these systems enable clients to store and update data on an untrusted server while guaranteeing the authenticity and integrity of the data.

Data Authentication with Merkle Trees. Merkle Trees provide these guarantees by constructing a tree of cryptographic hashes from the stored data, where parent nodes are constructed by combining the hashes of their child nodes. In a binary tree, for example, a parent node is constructed by concatenating the cryptographic hashes of its two children and applying a cryptographic hash function to the result. This process is repeated until a single root hash remains, which is signed by the owner of the data (e.g., the user of an outsourced key-value store). Valid data updates result in a new root hash that is similarly authenticated by the owner. Due to the recursive structure of Merkle Trees, any unintended change or data corruption also leads to a changed root hash. As we will detail later, this property allows the data owner to check the integrity of the entire database or the validity of query results only by maintaining the authenticated root hash (instead of the entire database).

Recognizing the Performance Constraints of Merkle Trees. The adoption of Merkle Trees in high-performance database systems such as main memory key-value stores running on modern hardware platforms, uncovered major performance challenges of Merkle Trees. In contrast to classical data structures (e.g., B-Trees) that achieve millions of operations per second, Merkle Trees only provide tens of thousands ops/sec [13]. The reasons for this inferior performance are two-fold. First, unlike classical data structures, updating data in Merkle Trees involve CPU-intensive operations, such as computing cryptographic hash functions or digital signatures. This bounds the performance by the speed of the CPU and limits the maximum achievable throughput. Second, improving performance by utilizing multiple threads that concurrently update the data structure is challenging. Since the root hash of a Merkle Tree is re-computed on every update, threads will conflict at the root and must be synchronized. This synchronization causes high contention and degrades the performance of a Merkle Tree for highly-parallel modern

multi-core systems [13, 14, 17, 156]. Unfortunately, addressing these two issues is not trivial for Merkle Trees.

Unrealized Performance Potential. At a first glance, these performance limitations seem inherent to the data structure and thus unavoidable. Hence, previous work mainly proposed alternative data structures [14] or employed secure hardware [13] to circumvent the performance limitations of Merkle Trees. Although their properties make performance optimizations hard, we observe that Merkle Trees still provide unrealized performance potential. For example, while synchronization for classical tree structures [100] is a known research area, efficient and scalable synchronization for Merkle Trees is largely unexplored. Further, while it is not yet possible to make cryptographic operations orders of magnitudes faster, we can exploit workload characteristics to adapt the Merke Tree to significantly reduce the overhead and thus increase performance.

Contributions and Outline. In this work we present initial results to show that building high-performance Merkle Trees is indeed possible. As a first contribution, we focus on an efficient synchronization schemes for Merkle Trees and propose a new *reverse latch-coupling* scheme to improve concurrency in Merkle Trees. Moreover, we suggest the concept of *splitting* to further reduce the contention and the cryptographic overhead at the same time. In the remainder of this paper, we first briefly review the basics of Merkle Trees before introducing our approaches for increasing their performance.

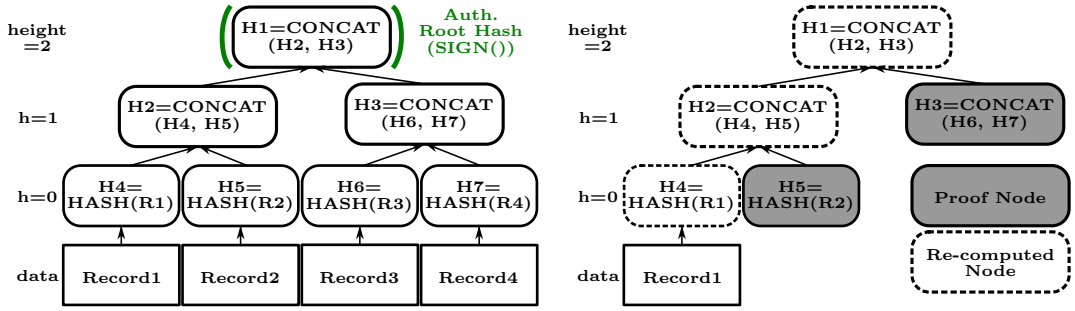
10.2 Merkle Tree Basics & Challenges

Intuitively, Merkle Trees protect data authenticity by recursively computing a cryptographic digest from the stored data. This section overviews how to construct, use and update this digest.

10.2.1 Construction

Classical Merkle Trees are binary trees in which each parent node is the hash of the concatenation of its two child nodes. To obtain a single root hash as a digest, we repeatedly construct $h = log_2(N)$ upper levels, where N is the number of data records.

General Process. The construction starts on the leaf level with height 0. There, a cryptographic one-way hash function $HASH()$ is applied to compute the hash of each data item. On the next level, two child nodes (leaf hashes on the lowest level) are concatenated, and the resulting concatenation is hashed to retrieve a corresponding parent node. We refer to this concatenate-and-hash operation simply as $CONCAT$, i.e.,



(a) Merkle Tree Structure (b) Merkle Tree Proof

Figure 10.1: Hierachical structure of the Merkle Tree (a) and how it is used to construct an inclusion proof (b). The $CONCAT$ operation refers to hashing the concatenation of two child nodes, i.e., $CONCAT := HASH(HASH_{left}||HASH_{right})$

$CONCAT := HASH(HASH_{left}||HASH_{right})$. This process is repeated until a single root node at height h remains, as shown in Figure 10.1a. The figure shows that the leaf with $H4$ is constructed by hashing the data record $R1$, i.e., $H4 = HASH(R1)$. The corresponding parent on the next level is created by concatenating both children hashes ($H4$ and $H5$) and computing the hash of their concatenation ($H2 = CONCAT(H4, H5)$). The final hash is then authenticated using a digital signature algorithm $SIGN()$ or storing the root hash in a trusted location.

10.2.2 Verification Procedure

Merkle Trees enable verifying the integrity of a data item without traversing the entire tree. This verification is done using an *inclusion proof*, a minimal set of nodes proving that the item is part of tree.

Proof Construction. To generate an inclusion proof, we start at the hash of the data item we want to authenticate. We then traverse the tree from the leaf to the root, collecting the sibling nodes along the way. Those sibling nodes are referred to as *proof nodes* in Figure 10.1b (marked grey). This leaf-to-root path containing the sibling hashes is called the *co-path* and serves as the proof of inclusion.

Proof Verification. To verify the inclusion of a data item, we start by computing the hash of the data we retrieved from the server, e.g., $H4 = HASH(R1)$. We then use the proof nodes to recompute all parent hashes up to the root as depicted by the dotted nodes in Figure 10.1b. For instance, we concatenate the previously recomputed hash $H4$ with the proof node $H5$ and hash the result to determine $H2$. If the final root hash

matches the authenticated root hash, we can confirm the presence of the data item in the Merkle Tree.

While verification is an essential feature of Merkle Trees, it is often not critical for the overall system performance since verification typically happens asynchronously on client machines [85]. However, efficiently performing updates that are vital for modern high-performance systems, represents a major source of overhead.

10.2.3 Update Procedure

One reason Merkle Trees are widely used in the context of outsourced data systems, such as secure key-value stores, is that they can be efficiently updated in an incremental fashion.

General Process. When a data item is updated, it is not required to re-compute the entire tree. Instead, the inner nodes of the tree are stored on the server, and only the direct parents along the root-to-leaf path need to be updated. For example, in Figure 10.1a, if $R1$ is updated, resulting in $H4^* = HASH(R1^*)$, all corresponding parent nodes must be updated since they are recursively computed. Updating a single data record thus only requires computing $h = log_2(N)$ new hashes. Finally, the updated root hash must be authenticated using a digital signature from the owner.

Performance Challenges. Merkle Tree updates are challenging in the context of high throughput scenarios because of two reasons: (1) They involve compute-intensive cryptographic operations such as hashing and signing. In contrast to simple memory accesses or non-cryptographic hash functions used in classical data structures, these operations cost thousands of CPU cycles. Consequently, any additional hash operation (e.g., due to an increased tree height) directly translates to thousands of cycles of overhead. (2) In order to leverage the parallelism of multi-core machines, multiple updates need to be executed concurrently. However, Merkle Tree updates are hard to parallelize since operations typically conflict for recomputing parent nodes and in particular the root node. However, naive synchronization, e.g., using a global latch of the Merkle Tree, introduces contention and limits concurrency since only one thread can modify the data structure at a time. Moreover, more fine-granular latching schemes known from B-trees do not easily translate to Merkle Trees as we discuss next. In the next sections, we thus discuss novel ideas to tackle these performance challenges, starting with the concurrency limitations.

10.3 Enhancing Concurrency

Efficient synchronization to improve the concurrency of classical tree-based index structures has been a prominent area of research in the database community [24, 74, 99, 100]. However, Merkle Trees possess several unique properties that justify a principled discussion of synchronization techniques.

Unique Properties of Merkle Trees. In contrast to Merkle Trees, (pure) updates in conventional tree structures (e.g., B+-Tree) typically do not propagate to the root node. Hence, updates usually involve a (read-only) top-down traversal to locate the record, followed by a local update on the leaf. This way, concurrently running threads (especially for uniform workloads) are dispersed across different leaf nodes, allowing each thread to apply its update directly. In contrast, Merkle Trees updates are propagated to the root node bottom-up so that all operations eventually converge at the root node. Although threads may update different leaves, collisions at inner levels (ultimately at the root) necessitate waiting for other threads to perform parent node updates. This contention not only limits concurrency but also degrades performance. Given that Merkle Trees are CPU-bound, contention for updates wastes precious cycles that could be devoted to actual work, such as hashing or signing. Hence, designing efficient synchronization techniques for Merkle Trees is of paramount importance.

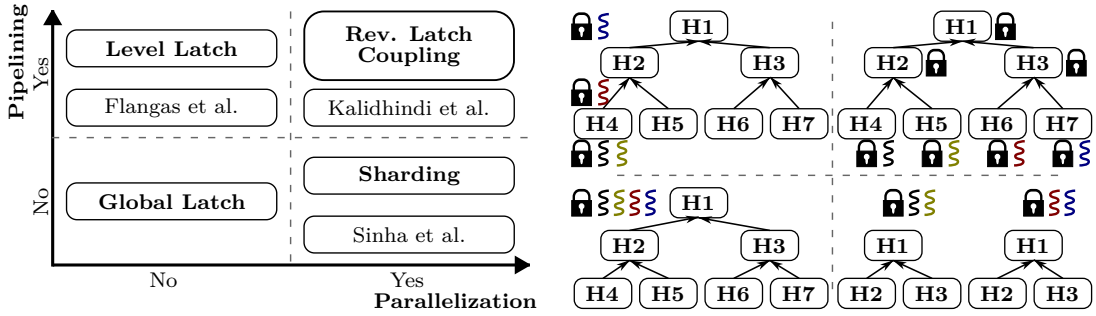
10.3.1 A Design Space for Concurrency

We identify two fundamental strategies for improving concurrency in Merkle Trees: *Pipelining (PI)* and *Parallelization (PA)*.

Pipelining overlaps the work of distinct threads by permitting updates to different tree levels, while parallelization allows threads to work concurrently within separate tree regions. Both approaches aim to minimize thread idle time by permitting updates to tree regions no longer needed by a previous thread. Based on these strategies, we can construct a two-dimensional design space, as shown in Figure 10.2a.

10.3.1.1 Four Concurrency Quadrants

The design space comprises four quadrants that characterize various synchronization schemes. Examining the quadrants enables us to understand why previous approaches fail to fully realize the performance potential of Merkle Trees. For evaluation purposes, we also devise a corresponding latching strategy for each quadrant.



(a) Design Space for Concurrency (b) Corresponding Latching Appr.

Figure 10.2: Pipelining and Parallelization define a two-dimensional design space for concurrency enhancements. Corresponding latching approaches (bold) and related work can be placed in its different quadrants (a). Subfigure (b) visualizes the corresponding latching approach of each quadrant.

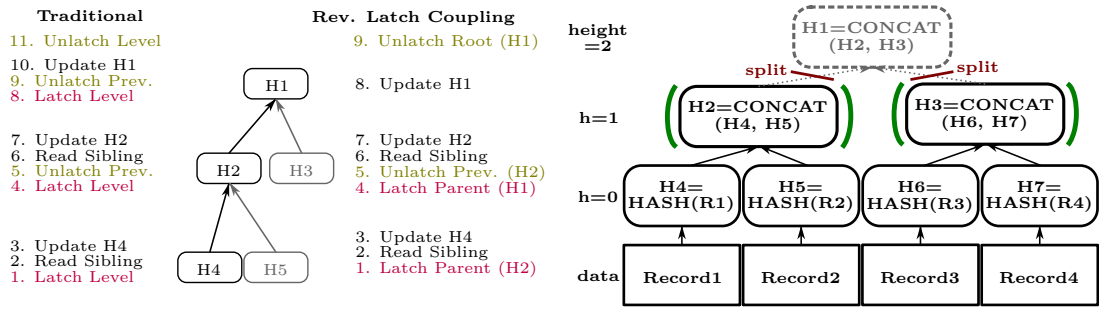
No-Pipelining/No-Parallelization. The most prevalent approach for synchronizing Merkle Trees today is situated in the bottom left quadrant. It uses a global latch[1] that is held for the entire leaf-to-root traversal to prevent conflicts [14, 89]. As illustrated in the bottom left quadrant of Figure 10.2b, in this approach, multiple threads queue for a central latch. As only one thread can acquire the exclusive latch required for updating the node, this approach serializes all threads and prevents concurrent operations.

Pipelining/No-Parallelization. Only a few existing approaches, such as [66], can be placed in the upper left quadrant (PI, No PA). For instance, [66] utilizes OpenMP[2] to implement a parallelized Merkle Tree that pipelines threads through all tree levels. This is done by traversing the tree level-by-level using OpenMP's `for ordered` construct.

An alternative latch-based scheme can be derived by increasing the granularity of latches to a latch per level, as shown in the upper left quadrant of Figure 10.2b. By utilizing traditional latching coupling [158] (see Section 10.3.2), this latching scheme creates a chain of threads that traverse the tree level-by-level.

No-Pipelining/Parallelization. While the previous approaches limited parallelism, approaches in the bottom right quadrant (No PI, PA) limit the pipelining effect. Several existing approaches (e.g., [13, 156]) realize the approach of this quadrant by sharding the data and building a dedicated Merkle Tree for each shard/partition[3]. Then, either a single thread is assigned to each shard or a global latch per shard is used to synchronize multiple threads (as depicted in the bottom right of Figure 10.2b). This scheme enables

[1] The systems community usually uses the term lock to describe the same mechanism.
[2] https://www.openmp.org/
[3] We use sharding/partitioning as synonyms



(a) Reverse Latch Coupling (b) Splitting of a MT

Figure 10.3: Comparison of traditional latch coupling (e.g., as used for level latching) and reverse latch coupling (a). The concept of splitting (b) yields sub-trees with lower height, reducing the overhead of CPU-intensive operations. New root nodes after the splits are marked with a bracket.

threads on different shards to process updates in parallel, but it limits the concurrency within a shard since threads contend for the shard latch (no pipelining).

Pipelining/Parallelization. Surprisingly, there is only one existing approach that falls into the last quadrant (PI, PA). The work in [89] uses fine-grained node latches in combination with a binary encoding scheme to detect the location of conflicts in a batch of updates. For instance, in the example shown in the upper right of Figure 10.2b, we can determine that conflicts will occur at $h = 1$ and $h = 2$. After determining the conflict positions, we can execute non-conflicting updates in parallel (e.g., all updates on the leaf level) and schedule conflicting updates for serial execution.

Unlike this work, we propose a general latching scheme based on fine-grained latches that does not require any encoding and pre-computation of conflict locations. Nevertheless, it still combines the benefits of pipelining and parallelization.

10.3.2 Reverse Latch Coupling

Naively applying node latches to synchronize updates in Merkle Trees can lead to deadlocks, as observed by previous work [89].

Key Idea. Hence, our scheme combines fine-grained node latches with latch coupling (aka hand-over-hand latching) [27, 99, 100, 158] to avoid deadlocks. The twist of our latching scheme is that it latches the node's parent (exclusively) to update a node (the root node is a special case). This backward-directed latching effect (hence the name *reverse latching*) effectively also latches the node's sibling(s). The underlying observation is that the inherent data dependency in a Merkle Tree enables us to simultaneously

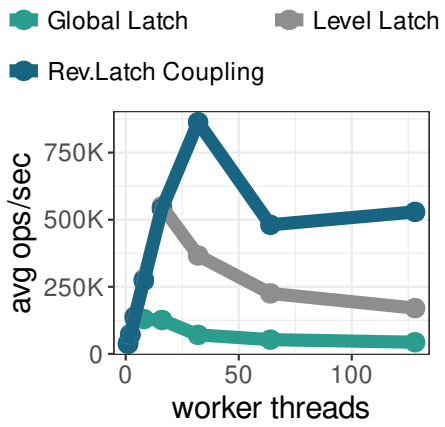
synchronize the (two) children of a node using their parent. I.e., instead of acquiring a latch for each child separately, we can protect the consistency of both nodes by latching their common ancestor first. We will discuss our latching in more detail in the following.

Latching Scheme. Figure 10.3a outlines our latching scheme and compares it to traditional latch coupling as used in the level latch scheme. In the level latch scheme, we latch the same level that includes the node subject to update. Afterward, we proceed by first latching the next level before releasing the latch of the previous level (= latch coupling). This traditional latching is possible since acquiring the latch of a level also protects the node's sibling(s). Consistently accessing the node's sibling(s) is necessary to compute the update of the next node correctly. However, since the level latch protects all nodes of a level, i.e., even a node's cousins, this scheme limits the parallelism within a level, as discussed before.

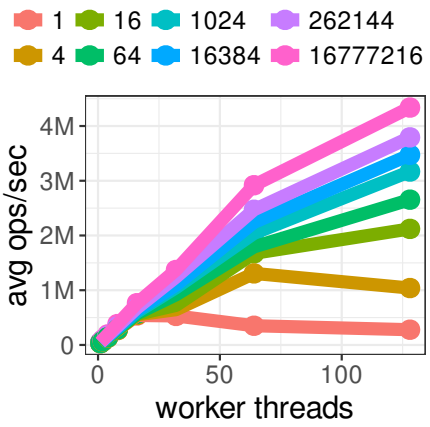
We use a reversed latching scheme to update a node and access its sibling consistently while avoiding deadlocks and enabling parallelism. As shown in Figure 10.3a, this means that we latch a node's parent exclusively to manipulate a given node. This scheme avoids deadlocks with concurrent threads trying to update a sibling node since both threads synchronize on a common latch instead of multiple latches, as in the naive schemes discussed by Kalidhindi et al.. Propagating updates upwards proceeds similarly by utilizing latch coupling. I.e., to update the parent, we first latch the next parent node (= grandparent) before unlatching the previously acquired latch. Clearly, the root node constitutes a special case since it does not possess a parent node. Hence, its latch is only released after updating its hash (step $8 - 9$).

Concurrent Read Operations. Concurrent read operations must follow the same reverse latching protocol to ensure consistency, albeit using shared latches. This is because both updates and reads follow the same access pattern and are funneled bottom-up to the root node. Nevertheless, an open question for future research is whether we can relax the need for acquiring exclusive latches for parent nodes during updates. The idea of this optimization is to allow for more concurrent readers, as done in previous research in the context of B-Trees [27, 99, 100, 158].

Insert Operations. Inserts might involve more complex structural modifications in a Merkle Tree. This is especially the case for balanced trees sorted on the key-order. However, we observe that reverse latch coupling can be applied without modifications if the Merkle Tree is only used for append-only purposes (i.e., sorted on insertion order) or without any sorting or balancing requirements. In these cases, inserts follow the same access pattern as updates, i.e., they are funneled to the root node. Hence, following the same reverse latch coupling scheme preserves consistency for both operations.



(a) Evaluation of Synchronization Schemes



(b) Evaluation of Splitting (colors represent number of sub-trees)

Figure 10.4: Initial evaluation results with a Merkle Tree containing 2^{24} records of size 8 Byte. Reverse latch coupling scheme significantly improves performance up to 32 threads (a). Leveraging the concept of splitting achieves better scalability and another order of magnitude speed up (b).

10.3.3 Evaluation & Insights

We next evaluate our novel latching scheme on an update-only workload to demonstrate that an enhanced synchronization technique can indeed improve the concurrency and performance of Merkle Trees. More extensive experiments using other workloads are planned for future work.

Experiment Setup. We implemented the different latching schemes mentioned above in a custom Merkle Tree implementation in C++. Our implementation uses the `BLAKE2b` hash algorithm and the `Ed25519` signature algorithm from the libsodium library [4]. The code is compiled using g++ version 11.1.0 on Ubuntu 18.04.1 LTS with optimization enabled (O3). We run our benchmark on a server with four Intel Xeon Platinum 8268 (2.90/3.90GHz, 35.75MB LLC) CPUs (192 threads in total) while setting the CPU governor to performance and pinning our process threads using the taskset utility. We execute updates for 8 byte records uniformly on a Merkle Tree with 2^{24} records for 10 seconds and report the system throughput as an average over 5 repetitions of our experiment.

The experiment results in Figure 10.4a illustrate that reverse level latching can scale to an order of magnitude more cores and improve performance to several hundred thousand

[4] https://libsodium.org/, Version 1.0.18-stable

operations per second. Nonetheless, performance degradation occurs after 32 worker threads. This effect can be attributed to contention, an issue we address subsequently.

10.4 Splitting Merkle Trees

While improved latching schemes successfully increase concurrency and reduce wasted cycles spent on contention, they cannot eliminate contention at the root node or decrease the computational overhead of Merkle Trees. Thus, we started to explore the concept of *splitting*. The intuition behind our approach is to enable a lightweight and more dynamic way of sharding Merkle Trees. By doing so, we aim to leverage the benefits of sharding while avoiding the downsides of the classical approach.

Benefits of Sharding. As discussed previously, classical Merkle Tree sharding partitions the data to create multiple physical trees. The resulting trees (cf. Figure 10.2b bottom right) are of lower height, reducing the number of required hash computations and thus the computation overhead for updates. Moreover, introducing multiple root nodes supports a higher degree of parallelism, as discussed earlier.

Downsides of Classical Sharding. Unfortunately, the classical sharding approach is static and thus sensitive to changing workloads or skewed access patterns [14]. For instance, while partitioning a Merkle Tree into two partitions might be sufficient for one workload, creating four partitions might be more beneficial for another workload. However, changing the sharding configuration to accommodate the changed workload usually requires expensive re-partitioning as it is tight to the data partitioning.

How Splitting Works. To avoid the downsides of classical sharding, splitting modifies the internal structure of a tree to create logical sub-trees instead of physical trees. As shown in Figure 10.3b, we accomplish this by *cutting the edge* between a given node and its parent (red line). This *split* promotes the detached node (green brackets) to become a new root representing the newly created sub-tree. Consequently, instead of representing the entire data set using a single root hash, we allow a tree to contain multiple roots associated with the split sub-trees. Further, unlike classical approaches that limit parallelism within a shard (cf. Section 10.3.1), we use our reverse latching scheme to improve concurrency within a shard.

Symmetric Splitting & Initial Results. We can adopt a simple symmetric splitting strategy to assess the impact of splitting on performance. This approach involves splitting child nodes from the topmost parent node to create two new sub-trees with individual root nodes. For instance, we can split the children of the initial Merkle root to create two

new trees with a reduced height of $h - 1$, as depicted in Figure 10.3b. The symmetric splits can be applied until we reach the leaf level of the tree. In this case, splitting creates a sub-tree with one (root) node for each data item. This configuration resembles the signature [127, 136] aggregation approach and suggests an interesting relationship between Merkle Trees and signature aggregation.

In Figure 10.4b, we present the results of using the symmetric splitting strategy for the same experiment as in Section 10.3.3. This experiment aims to assess the benefits of splitting for improving concurrency and performance in Merkle Trees. To this end, we study the update performance of different splitting configurations (i.e., the number of sub-trees) when increasing the number of worker threads. As seen in Figure 10.4b, the throughput increases with more sub-trees, and the scalability w.r.t the number of worker threads improves.

The outcomes demonstrate that splitting can significantly enhance Merkle Trees' performance in multi-core environments by an order of magnitude, resulting in over 4M operations per second in the optimal setup. We can attribute this improvement to two factors: First, splitting shortens the root-to-leaf path, reducing the number of hash computations and the CPU burden. Second, it eliminates the central contention point of the global tree and enables multiple threads to work on different regions of the tree concurrently.

Challenges & Open Questions. Our results suggest that splitting is a promising technique for improving performance. Nevertheless, several challenges and open questions remain that we plan to address in future work:

While radically splitting the tree to the leaf level offered the best performance in our experiment, previous work on signature aggregation shows that this also comes with clear downsides. For example, the verification overhead for range queries increases since multiple digests (instead of one) are required to verify a query result. Further, especially when many splits (e.g., 16M) are performed, the associated overhead for storing and maintaining all authenticated root hashes might increase dramatically. This observation raises the question of how many splits are beneficial and what trade-offs are involved.

Further, our symmetric scheme always splits the tree at the top. Although this is a rational choice for a uniform workload, splitting at different locations in the tree might be appealing for other workloads.

Finally, an essential benefit of splitting is its ability to adapt to changing workloads. Addressing workload shifts, however, raises additional questions concerning when to perform splits and how to control splitting decisions.

10.5 Conclusion & Outlook

In this paper, we presented initial results for improving the performance of Merkle Trees for high-performance data systems. To address the concurrency issues of Merkle Trees, we presented a novel synchronization technique that exploits pipelining and parallelization to increase the degree of concurrency. Further, we introduced *splitting* as a lightweight technique to shard Merkle Trees. Splitting reduces the CPU-cost by lowering a tree's height and reducing the contention problem. While several open questions remain, e.g., w.r.t to the involved trade-offs, our initial results suggest that it is indeed possible to improve the performance of Merkle Trees by orders of magnitude.

In future work, we want to analyze both approaches in more detail (e.g., w.r.t. skewed workloads) and study their involved trade-offs and limitations.

10.6 Acknowledgements

This research work was supported by the National Research Center for Applied Cybersecurity ATHENE. It also received funding from the Federal Ministry of Education and Research (BMBF) under Grant Agreements No 2WDG017A as well as from the Federal Ministry for Economic Affairs and Climate Action (BMWK) under Grant Agreement No 01MK21002K.

11 TrustDBle: Towards Trustable Shared Databases

Abstract

In this chapter, we present TrustDBle ['trʌstəbəl], a trustable DBMS which can be used for shared access by multiple parties. TrustDBle is based on our previous work on BlockchainDB which uses blockchains as an auditable storage to guarantee transparency and auditability of any data change. TrustDBle extends this work with a secure OLTP engine that implements verifiable ACID-compliant transaction execution on shared data while preserving scalability. In this work we discuss the main design choices and considerations for building trustable data management systems and show first results from our work on TrustDBle.

Bibliographic Information

The content of this chapter was previously published in the peer-reviewed work Muhammad El-Hindi, Simon Karrer, Gloria Doci, and Carsten Binnig. "TrustDBle: Towards Trustable Shared Databases." en. In: *3rd International Symposium on Foundations and Applications of Blockchain*. 2020. URL: https://scfab.github.io/2020/FAB2020_p7.pdf.

11.1 Introduction

Motivation. Databases (DBs) and database management systems (DBMSs) are a proven technology to efficiently store and query data. They scale very well to support a large number of nodes and amounts of data. Further, they offer standardized interfaces, such as SQL, and their use is well established among developers and operation teams. However, DBMSs are centralized solutions that assume a single database owner who solely can query and modify the data. Yet, in many use cases the data might belong to several different entities. For example, in typical supply chain scenarios the information about goods along the supply chain needs to be tracked by many different companies.

The traditional solution for shared data access is that each company keeps a local copy of the database and uses custom-developed data integration solutions to synchronize state between the different instances. However, this way of data sharing comes with many different drawbacks. For example, once data leaves the database of the data producer it is not transparent for this data producer to whom the data will be made available or how the data might be altered. This opens up the door for data misuse in different directions. For instance, in a food supply chain, the best-before date might be "faked" to resell products even after they expired.

A solution to this problem can be achieved with BCs (or distributed ledgers in general) that can be used as shared databases. First, BCs record all updates in an immutable manner and thus provide an auditable storage that can be used to find out how data was changed over time by whom. Second, BCs require consensus of the participants before a data update can be committed. While BCs thus seem to address the aforementioned issues, they lack the performance and scalability required for many use cases. For example, BCs offer transaction rates of $100's$ or $1000's$ transactions per second, while databases can achieve $100,000's$ transactions per second. Further, BCs introduce new interfaces and programming models that many organizations are not familiar with. Due to the lack of standards almost every blockchain platform even uses different interfaces, which increases the complexity and risk for adopting this technology in enterprises.

Contribution. This paper presents TrustDBle ['trʌstəbəl], a trustable DBMS which can be used for shared access by multiple parties. TrustDBle is based on our previous work [52] that shows how a scalable storage manager can be built on top of blockchains to provide high performance and audibility of all data changes at the same time. However, the shared storage manager in [52] is limited to simple put/get-operations. TrustDBle thus extends our previous work [52] with a secure execution engine that implements verifiable transaction execution on shared databases while preserving the scalability of

[52]. Verifiable transaction execution means that the DBMS engine itself guarantees a correct (i.e., ACID-compliant) execution of transactions. For example, to achieve isolation TrustDBle introduces a lock manager that resides in a trusted execution environment and thus guarantees correct serializable execution of multi-statement transactions from different parties. In this work, we discuss the main design choices of all involved components when building a trustable DBMS and show first results from our work on TrustDBle. This includes the aforementioned verifiable transaction execution engine, but also other components needed to establish trust. For example, in many data sharing scenarios it is important for a data owner/producer to track who accessed data during which queries.

Outline. The remainder of this paper is organized as follows: The next section discusses the guarantees and requirements that a trustable data management system should provide. Then, we present our architecture of TrustDBle and the main design choices to establish trust in DBMSs. Afterwards, initial results of running a OLTP benchmark (SmallBank) simulating a typical data sharing scenario is shown in our evaluation of TrustDBle. Finally, we conclude with the discussion of related work and a summary.

11.2 Properties of a Trustable DBMS

In telco scenarios, mobile phones of users are often being used outside the range of their home networks and can also connect to an available cell network of another provider. While this causes charges for users in different cell networks, a user still pays her charges only to the telco provider of her home network, who clears the debts of the user with the second provider. Figure 11.1 depicts such a scenario where Alice has a home provider (Bob) but was also using the network of another provider (Charlie). Through the abstraction of the shared database, Alice could not only settle her charges with Bob, but also Bob could clear the debts of Alice with Charlie using transactions on the shared database.

However, in a setup in which the involved parties do not trust each other, it is required that the DBMS providing the shared database abstraction guarantees that data can only be accessed and manipulated as agreed by all parties (e.g., if Alice clears her charges with Bob, Bob has to clear the debts with Charlie based on the agreed terms of the mobile contracts). Further, any data modification (this includes malicious modifications) should be recorded in a tamper-proof manner. For example, if Charlie manipulates the phone charges of Alice to his own favor (e.g., by increasing his charge state for Alice and

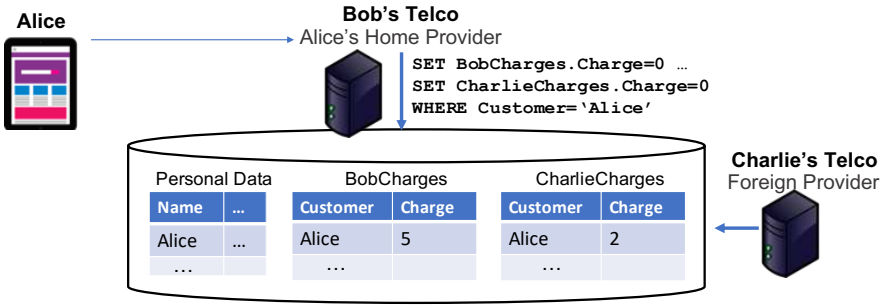


Figure 11.1: Shared DB of Two Telco Providers

decreasing Bob's) it should be possible for Alice to detect this malicious change using the tamper-proof history of all changes in the DBMS.

To achieve such guarantees the following three criteria must be met by a trustable DBMS:

Requirement 1 - Data Sovereignty. While multiple Telco providers could use the same shared database, Alice should be able at any time to control who has access to her (personal) data. For example, only the home provider can read information such as the personal address.

Requirement 2 - Verifiable Processing. As shown in the example, transactions are used to execute complex data manipulations that consist of multiple operations and access multiple records. Especially, in a collaborative setting where multiple parties control different parts of the data, it must be guaranteed that all parties execute transactions in a correct and verifiable way. In this context, correct and verifiable execution means that a DBMS is able to prove that a transaction was executed in an ACID-compliant way by all involved parties. In the above scenario, for example, this means that Alice's settlement transaction is correctly executed on both Bob's and Charlie's charge tables.

Requirement 3 - Auditable Storage. The existence of verifiable processing as described above, is often not sufficient if one could potentially tamper with or remove data from the DBMS after processing. For example, as mentioned earlier, Charlie might tamper with the data to manipulate the current state of charges to his advantage. Hence, auditability requires that all information needed to validate how data was updated over time must be tamper-proof. Further, data management systems must provide new interfaces to make databases easily auditable. This could be done by users directly or by additional applications or services.

11.3 Architecture & Key Concepts

TrustDBle is a distributed database that is build around the key requirements of a trustable DBMS as discusseed before. Similar to a traditional database, it offers users a standard SQL-Interface and provides ACID properties. However, it also guarantees sovereignty, verifiable processing and auditable storage.

Figure 11.2 shows an overview of TrustDBle's architecture. Similar to our work in [52], we build a database layer on top of blockchains as a storage layer. Thereby, TrustDBle also makes use of sharding in the storage layer to enable scalability. In this work, we extend the database layer with a secure execution engine that enables verifiable processing and sovereignty.

In the following, we provide more details on TrustDBle's different layers and explain how the previously mentioned properties of a trustable DBMS are achieved.

11.3.1 Auditable Storage

TrustDBle's storage layer utilizes blockchains as a persistent, auditable storage backend. Blockchains enable us to store data in a tamper-proof way and record modifications of the data on the blockchain. Thereby, as shown in [52], database techniques such as sharding help TrustDBle to overcome the scalability limitations of blockchains.

Despite sharding, we also aim to implement further optimization such as caching in the storage layer to speed up data access. Caching will enable TrustDBle to treat the auditable storage as another layer in the memory hierarchy of the database. With the help of *verifiable processing* it can be guaranteed that data is maintained in a tamper-proof way in memory or on disk until it has been persisted to the auditable storage.

11.3.2 Verifiable Processing & Data Sovereignty

On top of the auditable storage, we offer components for verifiable transaction processing and sovereignty. In the following, we mainly focus on verifiable transaction processing and only briefly touch on sovereignty.

Verifiable Processing. With verifiable transaction processing we refer to two aspects. First, all parties involved in a shared DBMS can only execute transactions which all parties agreed on. Second, when transactions are executed, parties can make sure that those transactions where executed correctly (i.e., following the ACID properties a non-shared DBMS would give).

One approach to achieve verifiable transaction processing is to use smart contracts to implement the transaction processing logic, e.g., as done in [44]. However, we found that this approach suffers from the same scalability limitations as blockchains themselves and adds additional complexity since new transactions logic needs to be re-implemented as smart contracts.

To overcome these issues, we implement a secure transaction processing engine outside the blockchain (i.e., on top of the auditable storage) with the help of a TEE (green box in Figure 11.2). In our current prototype we use Intel Software Guard Extensions (SGX) to provision the TEE. Intel SGX establishes TEEs as so called hardware enclaves, that are protected by the CPU. Thereby, the CPU provides the enclave with a special address space that is only accessible by the trusted code inside the enclave.

One major limitation of SGX, however, is that it only provides a small portion of protected memory. To overcome this challenge, TrustDBle does not place all transaction execution components in the trusted environment. Instead, we place only those components inside the TEE which are required to verify that the ACID properties have been fulfilled. For example, to provide verifiable isolation, we only implement the lock manager (LockMgr) of our database layer inside the trusted environment.

We use two key criteria to decide which component to implement inside the TEE. First, the component should only maintain a small state inside the protected memory region. In the case of the lock manager, for instance, we only need to maintain the lock table of the lock manager in the TEE. Second, verifiable processing must depend on the correct behaviour of a node. As an alternative approach, we employ verification protocols similar to [52] to verify the correct behaviour of a component. This is for example used to verify the correct behaviour of the transaction coordinator (TxCo) component and achieve verifiable atomicity.

Data Sovereignty. Furthermore, we plan to use TEEs to guarantee sovereignty. The key idea is that TrustDBle encrypts data inside the TEE and only provides authorized parties access to the encryption key.

11.3.3 Scalability & Usability

Despite trust achievement being the core of TrustDBle, we do not want to scarify scalability and usability. TrustDBle achieves this with the help of two main ideas:
DB optimizations & interfaces: Scalability is achieved by making use of classical database techniques, such as sharding, distributed locking and caching. While these techniques

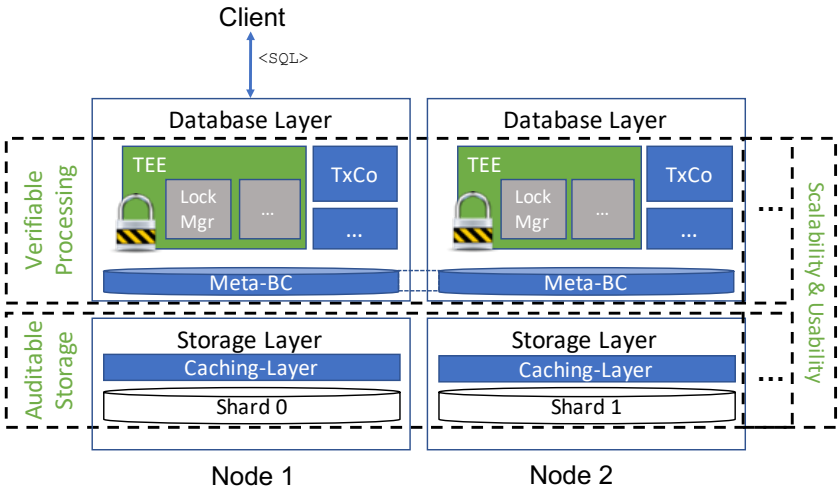


Figure 11.2: TrustDBle Architecture

introduce new challenges such as cross-shard transactions and cache-coherence, we want to address them without compromising trust.

Further, we want to provide users with familiar database interfaces and abstractions (e.g., ACID transactions and isolation levels). This way, TrustDBle can be integrated in existing enterprise architectures and serve as drop-in replacement for traditional databases that do not provide trust guarantees.

Adaptivity & Flexibility: Also, we want to achieve usability through adaptivity. As mentioned previously, TrustDBle supports different auditable storage backends. Therefore, TrustDBle can be used with any blockchain network that users might already be familiar with. Moreover, we plan to support connecting to local databases via an ODBC interface. This way, users can run queries that target local and TrustDBle's data.

11.4 Current State of TrustDBle

In the following, we report on the current state of TrustDBle and show our initial performance results of running a simple OLTP benchmark on TrustDBle.

11.4.1 Implementation Details

TrustDBle is still in its early stage and we mainly focused on verifiable transaction processing and not on data sovereignty so far. Our verifiable transaction processing

engine currently provides verifiable isolation and atomicity guarantees (i.e., the A and I of ACID) which are based on a secure locking scheme and an auditable 2PC protocol for cross-shard transactions.

The secure locking scheme is implemented via sharded lock managers that run inside the TEE of a TrustDBle node. Only, transaction managers (TxMgrs) with a valid lock (i.e., signed by the LockMgr) are allowed to modify or access data. Sharding the lock manager enables us to prevent the lock manager from becoming a bottleneck. Availability, is achieved by persisting the state of lock managers to auditable storage. This way, other nodes in TrustDBle can recover from a lock manager fault. Moreover, our locking scheme supports the execution of transactions under different isolation levels. To that end, the trusted lock managers enforce that locks are acquired and released according to the specified isolation level.

Similarly, we utilize the auditable storage to log 2PC messages and detect or recover from a faulty transaction coordinator. Thereby, local transaction managers apply a *trust, but verify* strategy with respect to a transaction coordinator: While the transaction coordinator is responsible to collect and forward 2PC messages and decisions to local TxMgrs, each local TxMgr also logs messages to a shared meta-blockchain which all TxMgr can access. Messages in this meta-blockchain are used to verify decisions of the transaction coordinator. Similarly to our previous work [52] we make use of deferred verification techniques to mask verification in the case of no failures.

11.4.2 Initial Results

Since TrustDBle combines database, BC technology and TEEs to overcome the limitations of blockchains, the focus of our evaluation is currently mainly on scalability. In particular, we want to show that our approach to verifiable processing can execute cross-shard transactions in a scalable way. To that end, we implement the Smallbank OLTP benchmark [30] and measure the performance of TrustDBle in different settings.

Figure 11.3 shows an experiment in which we increased the number of participants in a TrustDBle network. Thereby, the number of participants matches the number of shards, since each node was responsible for a separate shard. We further scale the number of clients from 16 to 256. Each client runs an update heavy workload of 1000s of transactions per shard under a read committed isolation level. All nodes are running as virtual machines in Microsoft Azure with 16 vcpus, 32 GB memory and Ubuntu 16.04 LTS as operating system. Hyperledger Sawtooth 1.04 is used as blockchain platform in the storage layer. This experiment shows that TrustDBle can execute cross-shard

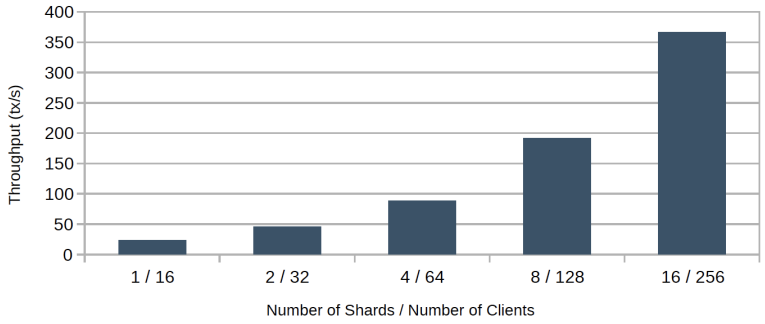


Figure 11.3: Scalability with the number of shards

transactions in a scalable manner for a network with up to 16 nodes and multiple shards. In future, we plan to perform a more detailed evaluation and, e.g., study TrustDBle's behaviour for larger network sizes. Moreover, we will analyze other scalability aspects (such as scalability with data size) and evaluate TrustDBle's behaviour under attacks.

11.5 Related Work

Several other work makes use of TEEs inside databases. EnclaveDB [143], for example, implements an entire signlenode database with the help of a TEE. TrustDBle is a distributed database, and extends the use of TEEs to a distributed setup. Further, EnclaveDB focuses on privacy and integrity, but not on a collaborative setup in which multiple parties share access to data. Further, all sensitive data is stored inside the TEE and therefore EnclaveDB requires support for large TEEs with hundred gigabytes of memory. In contrast, TrustDBle uses TEE only to secure few critical components of the system like the LockMgr.

Another area of related work are hybrid approaches that combine TEEs with blockchains to address the performance limitations of blockchains. Ekiden [35] uses a hybrid approach to get confidentiality and use TEEs to improve smart contract computation and scalability of the underlying blockchain. It provides privacy, but lacks usability, since it implements a new blockchain platform with custom interfaces and programming abstractions. TrustDBle is rather a database than a blockchain and provides standard database interfaces (e.g., SQL) and abstractions (e.g., ACID transactions and isolation levels). Another system using a hybrid approach is Hyperledger Avlon[165]. It combines trusted execution environments, such as Intel SGX, and blockchains. Howerever,

ACID guarantees are not provided and instead it uses trusted oracles to offload certain computations from a blockchain.

Further, Blockchain databases like Veritas [68] as well as BigchainDB [115] aim to provide a similar abstraction of a shared database as TrustDBle. However, they differ in how they implement this abstractions. TrustDBle focuses on how the complexity of existing blockchains can be overcome with the help of an additional database layer, while relying on the auditability and tamper-proofness characteristics of blockchains.

11.6 Conclusion

We presented TrustDBle, which combines database, blockchain and secure hardware technology to implement a trustable data management system. It extends our previous work with a secure OLTP engine that implements verifiable ACID-compliant transaction execution on shared data while preserving scalability. In this paper, we discussed the properties that are required to build trustable DBMSs and explained how these guarantees are implemented in TrustDBle. In contrast to native blockchains, TrustDBle is able to fulfill these guarantees without scarifying scalability and usability. Our initial experiments show that TrustDBle successfully supports cross-shard transactions and allows us to scale the performance of the system by increasing the number of shards.

11.7 Acknowledgements

This research work has been funded by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE. We also thank Microsoft Research for providing us with Azure resources for our evaluation.

12 Benchmarking the Second Generation of Intel SGX Hardware

Abstract

In recent years, Trusted Execution Environments (TEEs) such as Intel Software Guard Extensions (SGX) have gained a lot of attention in the database community. This is because TEEs provide an interesting platform for building trusted databases in the cloud. However, until recently SGX was only available on low-end single socket servers built on the Intel Xeon E3 processor generation and came with many restrictions for building DBMSs. With the availability of the new Ice Lake processors, Intel provides a new implementation of the SGX technology that supports high-end multi-socket servers. With this new implementation, which we refer to as SGXv2 in this paper, Intel promises to address several limitations of SGX enclaves. This raises the question whether previous efforts to overcome the limitations of SGX for DBMSs are still applicable and if the new generation of SGX can truly deliver on the promise to secure data without compromising on performance. To answer this question, in this paper we conduct a first systematic performance study of Intel SGXv2 and compare it to the previous generation of SGX.

Bibliographic Information

The content of this chapter was previously published in the peer-reviewed work Muhammad El-Hindi, Tobias Ziegler, Matthias Heinrich, Adrian Lutsch, Zheguang Zhao, and Carsten Binnig. "Benchmarking the Second Generation of Intel SGX Hardware." In: *International Conference on Management of Data, DaMoN 2022, Philadelphia, PA, USA, 13 June 2022*. Ed. by Spyros Blanas and Norman May. ACM, 2022, 5:1–5:8. DOI: 10.1145/3533737.3535098. URL: https://doi.org/10.1145/3533737.3535098.

12.1 Introduction

Trusted Execution Environments (TEEs). Trusted Execution Environments have gained a lot of attention [10, 25, 59, 67, 73, 93, 129, 143, 159, 170, 195] in recent years because they enable trusted processing of private data. As such, TEEs can be used to build trusted databases [143, 149] that guarantee the confidentiality and integrity of data even when hosted by a third party. This is particularly interesting when storing company data in public cloud environments rather than on-site. As TEEs fully protect data during processing, this secures data from manipulation by even privileged users such as administrators.

Intel SGX as important TEE technology. Today, different implementations of TEEs exist such as Intel SGX [117], ARM TrustZone [6] and several others [39, 91]. Among those, Intel SGX has already seen wide adoption for many different use cases including DBMSs. To enable secure execution, Intel SGX provides special CPU instructions for defining private memory regions in which the security of data is guaranteed by the hardware [75, 117, 171]. Even the operating system has no access to these so-called enclaves.

SGX has many limitations for DBMSs. However, until recently SGX was only available in consumer-grade CPUs based on the previous Intel Xeon E3 processor generation. In addition to the low core count of these CPUs, SGX enclaves came with drastic technical limitations, such as a limited memory capacity of up to 256 MB and significant performance overheads. The capacity restrictions especially limited the application of Intel SGX for DBMSs. In consequence, researchers in the database community started to explore different ways to overcome these limitations, e.g., by only placing certain DBMS components inside an enclave [173] or designing enclave-native engines for this restricted environment [93, 159].

SGXv2 lifts the major limitations. With the availability of the new Intel Ice Lake processors [79], Intel promises to address several limitations of SGX enclaves. The latest implementation of Intel SGX on these processors (in the following referred to as SGXv2), not only reduced the overhead of memory protection, but also increased the capacity of the protected memory region to up to 512 GB per socket (depending on the CPU model) [87]. This allows DBMSs to hold even large data sets fully in the enclaves. In addition to that, the new scalability enhancements now allow DBMSs that use Intel SGXv2 to scale across multiple sockets of high-end servers.

The need to benchmark SGXv2 for DBMSs. This raises the question whether previous efforts to overcome the limitations of SGX for DBMSs are still applicable and if

the new generation of SGX processors can truly deliver on the promise to secure data without compromising on performance. To answer this question, as a first contribution, we perform a systematic performance study of Intel SGXv2 and compare it to the previous generation of SGX (referred to as SGXv1 in this paper). In particular, we evaluate the performance of Intel SGXv2 for typical data access patterns of OLTP and OLAP workloads. Moreover, as a second contribution we discuss lessons learned for building the next generation of high-performance enclave-based DBMSs on SGXv2. While the new SGXv2 enhancements also cover several new security aspects, we will solely focus on performance characteristics in this paper.

In the following, we first briefly review the basics of Intel's SGX (SGXv1 and SGXv2). Afterwards, we present the results of our evaluation study and the findings of our performed benchmarks.

12.2 Background

In the following we discuss the most important building blocks of the Intel SGX technology and the latest enhancements that were introduced by the second generation of Intel SGX.

12.2.1 Intel SGX Overview

Intel SGX is a hardware-based TEE technology. It introduces new platform extensions such as a Memory Encryption Engine (MEE) and new CPU instructions (`SGX1` and `SGX2` [116, 182]) to enable applications to create private memory regions protected from privileged software. That means that even the operating system or hypervisor are not allowed to access these regions. To create these so-called *enclaves*, SGX reserves a portion of the memory called the Processor Reserved Memory (PRM), as depicted in Figure 12.1. The size of the PRM is configured in the BIOS and will not be available to the operating system. As shown in Figure 12.1, inside the PRM Intel SGX maintains the EPC that stores code and data of enclaves in 4 KB memory pages. These pages are encrypted and their data is only decrypted when it is loaded into the CPU cache for processing. Furthermore, EPC pages are integrity protected to prevent unnoticed manipulation of the data. [171]

Software (SW) (i.e., applications) that runs inside an enclave is referred to as *trusted SW* or *trusted code*. Intel SGX guarantees that only code from within the same enclave has access to the EPC pages of that enclave. Any code that is running outside the PRM, i.e., in the untrusted memory region, is prevented from accessing pages in the PRM.

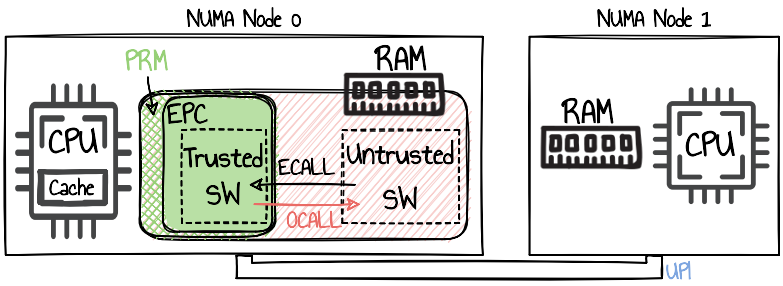


Figure 12.1: Intel SGX multi-socket architecture

This is achieved with the help of hardware address translation by making sure that the virtual-to-physical address mapping is only accessible to the owning enclave. Hardware address translation also enables enclave developers to assign a heap size larger than EPC to their application (by setting a `HeapMaxSize` config parameter) [81]. However, exceeding EPC capacity does not come without cost as explained next.

Since enclave memory is a finite resource, Intel SGX comes with a mechanism to swap the encrypted EPC pages out to unprotected memory (and vice versa). This mechanism is also referred to as *paging*. In order to protect the integrity of pages and to ensure that only the latest version of an evicted EPC page can be loaded back, SGX stores version information in a version array that is stored in EPC pages. The additional integrity protection as well as the required context switch and data transfer makes EPC paging very expensive as already discussed by previous work on SGXv1 [131, 162].

The first step towards using and interacting with an enclave is enclave creation and initialization. During enclave creation the initial code and data of the enclave is loaded by untrusted system software. As part of this process, the system software copies data from outside the PRM into the EPC and assigns the EPC pages to the created enclave. Thereby, the entire content of the enclave plus some metadata is cryptographically hashed by the CPU. This hash is called the *measurement hash* and can be used to attest that the expected code is running inside the enclave [171].

Trusted and untrusted SW interact through so-called `E-` and `OCALL`s. In order to transfer the control flow to code inside the EPC, untrusted software performs `ECALL`s (enclave calls). Similarly, trusted software can transfer the control flow back to code outside the enclave by means of `OCALL`s. These interactions not only represent a context switch in the CPU, but they involve additional steps to preserve confidentiality of enclave data, such as flushing CPU caches and the address translation cache (TLB) [131, 166, 179].

In our evaluation, we do not focus on the overhead of the interaction between trusted and untrusted parts of an application, but instead we concentrate on the overhead that comes from using data within the EPC. The reason is that SGXv2, which we discuss next, provides much higher EPC capacities which potentially allows a DBMS to hold all data structures fully inside the enclave (i.e., much less interactions with untrusted code are needed).

12.2.2 Second Generation Intel SGX

In this paper, we focus on SGXv2 that was introduced with the launch of the new Intel Ice Lake processors. With SGXv2, Intel introduced several enhancements to the SGX technology such as an increased EPC capacity (from previously 256 MB to up to 512 GB per socket). Further, Intel SGX now also supports running enclaves on multi-socket server systems. However, to the best of our knowledge, little is known about the implementation details of these enhancements. Hence, in the following, we will only give a brief overview of the new enhancements based on [87] and then focus on empirically evaluating these enhancements in Section 12.3.

Increased EPC capacity. In order to increase the memory capacity available to enclaves, Intel SGX reworked its encryption and integrity protection mechanism. Instead of the MEE, SGX now uses Intel's Total Memory Encryption (TME) technology to protect the confidentiality of data. While Intel mentions that TME relies on AES-XTS [114] as a block cipher mode, no detailed information about the integration of Intel SGX with TME is available. Yet, Intel TME is undoubtedly one major building block for supporting larger EPC sizes. At the same time, new ways are used to guarantee data integrity and protect against replay attacks which might introduce additional overheads. For instance, error correction codes are used to detect manipulations of data from outside the enclave. Similarly, a trusted firewall has been introduced to prevent manipulations of data from a malicious enclave inside the EPC.

Multi-socket support. Previously, Intel SGX was only available on single-socket servers. With the introduction of SGXv2, however, Intel SGX is also available on multi-socket systems that utilize a NUMA memory architecture. For this, two major challenges had to be addressed: First, in SGXv1 encryption keys and certificates were derived from per CPU socket secrets. However, in case of multiple CPUs it is required to share a common key across sockets to allow CPUs to access EPC pages that were encrypted by a different CPU [87]. Second, and more relevant in our context, protected memory that is allocated

on one socket needs to be securely accessible to the remote CPU. In the following, we elaborate on this challenge in more detail.

In a NUMA architecture, local memory is provided for each processor as depicted in Figure 12.1. Each CPU resides on a separate *NUMA node* (i.e. socket) which are connected via a fast interconnect (the Ultra Path Interconnect (UPI) on Intel platforms) to support memory access from remote CPUs. To securely access data of EPC pages on a remote NUMA node, SGXv2 introduces an additional UPI Crypto Engine (UCE) that protects the confidentiality of data transferred over UPI.

To avoid overhead for untrusted Software, SGXv2 extends the memory coherence architecture as follows: When a core on one socket (e.g., the left one) incurs a cache miss, the referenced memory address is passed on to a local caching agent. The agent knows which physical memory addresses are attached to which socket and is in charge of forwarding the requests via UPI to the remote socket if necessary. When protected memory is requested, the request is sent with a so-called secure attribute to the UCE for encryption. In contrast, requests for unprotected memory are passed on in plain text. The receiving side will forward the request to its local Caching Agent. If a request references data in protected memory, the agent will check if the secure attribute has been set and only then forward the request to its memory controller. Subsequently, the retrieved cache line is forwarded to the UCE for protected transmission of the cache line to the requesting socket.

The example indicates an additional overhead for enclaves when accessing memory on remote NUMA nodes. That means, that despite the regular overhead of cross NUMA communication, Intel SGX introduces even further overhead, e.g., because of the additional encryption. Due to the recent availability of this hardware, however, no empirical evaluation of this overhead has been performed yet. This motivated us to include experiments for analyzing the NUMA effects on SGXv2 in our evaluation.

12.3 Experimental Evaluation

In the following, we perform several experiments to understand the characteristics and pitfalls of the second generation of Intel SGX hardware. Where applicable, we also compare and relate to known characteristics of the first generation of SGX.

In the first set of experiments we take a look at the performance for growing data set sizes. Afterwards, we study the performance of SGXv2 for basic OLTP and OLAP workload patterns. In the last group of experiments, we present several findings with

Table 12.1: Properties of experiment hardware

	SGXv1	**SGXv2**
Architecture	Cascade Lake	Ice Lake
#Sockets	1	2
CPU	Xeon E-2288G	Xeon Gold 6326
Core Count	8	16
CPU frequency (max)	5.0 GHz	3.5 GHz
L1 d/i cache	256/256 KiB	1.5/1.0 MiB
L2 cache	2 MiB	40 MiB
LLC cache	16 MiB	48 MiB
EPC (per socket)	128 MB	64 GB
DRAM (per socket)	128 GB	256 GB

regard to the usage of multiple concurrent enclaves in SGXv2 that can result from running different DBMS instances on the same hardware.

Setup. For our study, we use one SGXv1 and one SGXv2 server. The hardware characteristics of both systems are shown in Table 12.1. To make our experiments reproducible over several runs, we configured both servers to always use their maximum CPU frequency. Note that, while SGXv2 could in principle support 512 GB of EPC, our CPU model only support a maximum EPC capacity of 64 GB per socket (i.e., 128 GB in total).

The comparison of the plain hardware characteristics as shown in Table 12.1 indicates a significant improvement for applications running on SGXv2. At the same time, the hardware differences make it difficult to perform a direct comparison of SGXv1 and SGXv2. Hence, in the following, we mainly study the characteristics of SGXv2 and only include results for SGXv1 when appropriate.

12.3.1 Data Scalability

In the first set of experiments, we evaluate SGXv2 with regard to growing data set sizes and compare it to SGXv1. Moreover, since SGXv2 supports multiple sockets, it allows enclaves to grow across NUMA boundaries. Therefore, we also study typical NUMA effects that play an important role in modern DBMSs [65, 92].

Performance for growing data sizes. In order to evaluate to which extent SGXv2 supports larger amounts of data, we implemented a B-Tree index structure as a trusted library inside SGX. We use it to execute a YCSB-like workload with 20% inserts and 80% reads to gradually increase the size of the database. The results of this experiment are shown in Figure 12.2. As expected, the performance of the B-Tree on SGXv1 (red

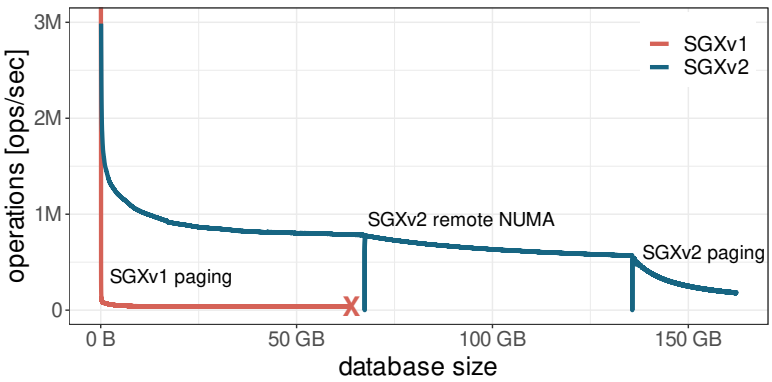


Figure 12.2: Throughput of a 80% read / 20% insert workload in a B-Tree. A significant performance improvement is observable for larger database sizes in SGXv2.

line) drops very quickly (after 128 MB) and only supports a total database size of up to 64 GB due to the corresponding limit on enclave sizes in SGXv1. In contrast to that, the B-Tree in the SGXv2 enclave (blue line) shows orders of magnitude better performance for much larger database sizes.

Moreover, in the case of SGXv2 we can observe two performance drops when scaling the data size: The first performance drop is at around 64 GB, when the data size reaches the capacity limit of the EPC on the first NUMA node. The second performance drop happens at around 128 GB where the capacity of the EPC on the second NUMA node is reached. At this point, similar to SGXv1, we can observe a drastic performance penalty due to paging. In the following, we will have a closer look at these two effects.

Cost of crossing NUMA boundaries. In Figure 12.2, we can observe that the performance of our B-Tree decreases for the first time when the data size exceeds the EPC capacity of the NUMA node to which the process was pinned. In consequence, new data must be stored in EPC pages that are allocated on the remote NUMA node. While the importance of NUMA awareness to overall system performance is a known issue [65, 92], it is unclear what overhead is added by SGXv2 and its additional encryption of UPI traffic.

Hence, in the next experiment, we measure the latencies for accessing memory on the local and remote NUMA node in CPU cycles. For this, we pin the process to the first NUMA node using `numactl`. Since the `libnuma` library is not available for use inside the enclave, we apply the following approach for allocating memory on the remote NUMA node: Since memory is first allocated on the NUMA node to which a process is pinned to, we pre-allocate a large chunk of 70 GB inside the enclave to consume more than the

Table 12.2: Local and cross NUMA access latencies [CPU cycles]

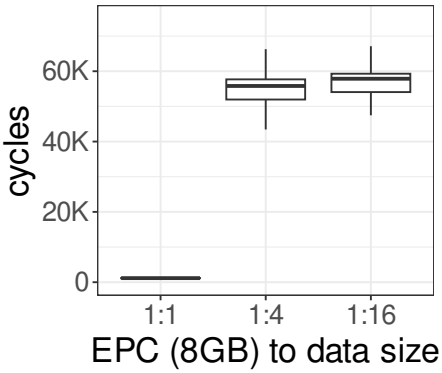
	Untrusted	**Trusted**	
	Latency	Latency	Overhead
Local NUMA	367.7646	486.1015	32.2%
Cross NUMA	568.6215	832.1615	46.3%
Rel. NUMA penalty	54.6%	71.1%	

EPC capacity on the first NUMA node. This forces any subsequent memory allocation to happen on the remote NUMA node.

To measure the memory access latencies that involve the remote NUMA node, we allocate another 2 GB memory buffer (which is then allocated in the second NUMA node). The data in this region is organized as a randomly linked list. This allows us to perform a (random) scan over the data while avoiding that the out-of-order execution schedules memory loads in parallel, which would distort the memory access latencies. For NUMA local access measurements, we of course skip the previous pre-allocation step to make sure that the buffer is fully located on the first NUMA node. For the untrusted baseline we use the `numactl` tool to pin the process to the first NUMA node while binding the memory allocation to the second node and thus enforcing cross NUMA accesses.

Table 12.2 shows the average latency for local NUMA and remote NUMA memory accesses. We start discussing the table row-by-row. Comparing untrusted to trusted local NUMA access, we can see that local memory access in the enclave has a clear latency overhead of around 30 %. This overhead is most likely caused by the necessary decryption of EPC pages when loading data into the CPU cache [75, 171]. When looking at cross NUMA access latencies, we can observe that accessing remote memory in an enclave has an even higher overhead compared to the untrusted baseline. To quantify the penalty of accessing a remote NUMA compared to local NUMA, we now look at the last row of Table 12.2. Thereby, we view the untrusted and trusted case in isolation. While it is evident from Table 12.2 that cross NUMA accesses are expensive even for the untrusted baseline, the penalty for trusted code is disproportionately high with around 71%. Based on the few information provided by Intel in [87], we speculate that this overhead can be attributed to the additional encryption of UPI traffic for trusted memory accesses.

Cost of Paging. In SGXv2, paging is similarly expensive as in SGXv1 as evidenced by the second performance drop in Figure 12.2. However, it is unclear how the overhead changes for growing data set sizes in SGXv2. To show the impact of paging on memory access latencies in SGXv2, we varied the data set size. To enforce paging, we reduced



ratio	99.9th perc.
1:1	1645.177
1:2	68790.278
1:4	101569.258
1:8	155954.108
1:16	218224.092

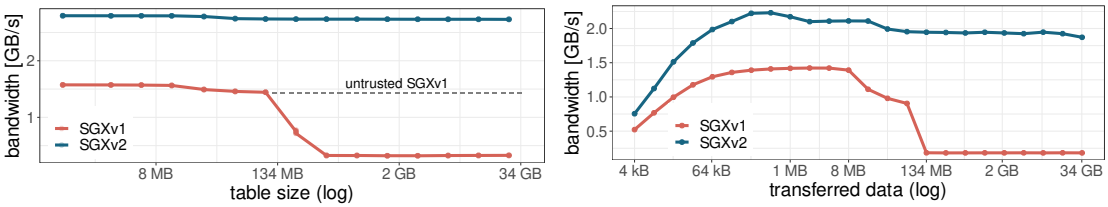
Figure 12.3: Paging overhead in CPU cycles. To enforce paging, we use a data set size that is a multiple of the EPC (e.g., 1:4 means 4× the EPC capacity of 8 GB(=32 GB)).

the PRM in the BIOS to 8 GB and use data set sizes that are multiples of the EPC, i.e., 4× and 16× (shown as ratios 1:4 and 1:16 in the figure). As a baseline, we use a configuration which uses data of the same size as the EPC, ie. 8 GB (shown as 1:1). As before, we ensure that we can measure the latency of page faults by traversing a randomly linked list of memory-aligned nodes of 4 KB. The results are shown in Figure 12.3.

We can observe that paging increases the latency by around two orders of magnitude when comparing 1:1 and 1:4. Surprisingly, when increasing the ratio even further to 1:16, the latencies seem to be affected only minimally. Yet, when looking at the tail latencies (right side of Figure 12.3), the impact becomes more apparent. As shown in the table, the tail latency doubles when comparing ratio 1:4 and 1:16. Obviously, these higher tail latencies have a tremendous impact on the DBMS performance (in particular for OLTP).

Another interesting question is to which extent the paging overhead influences typical OLAP-style workloads. To show this, we further evaluate the effect of paging on sequentially scanning data. Figure 12.4a shows that SGXv2 can maintain a steady performance for sequentially scanning data while the performance of SGXv1 quickly drops after reaching the EPC limit due to paging.

In the previous experiments we allocated the data directly inside the enclave. In the next experiment, we study the effect of paging on copying data that needs to be transferred into the enclave. Intel SGX provides two mechanisms to give enclaves access to data residing outside the enclave. In our experiment we use the `in/out` mode which allocates EPC pages inside the enclave and copies the requested data from untrusted memory. This is in contrast to the `user_check` mode which provides direct (i.e. without copying), but unprotected access to untrusted memory. As can be seen in Figure 12.4b,



(a) Sequential scan of varying data sizes

(b) Data copy to an enclave for various sizes

Figure 12.4: Effect of paging on a sequential scan (a) and on copying data securely to an enclave (b). In contrast to SGXv1, SGXv2 is not affected by paging due to its larger EPC capacity.

for small amounts of data the performance pattern of SGXv1 and SGXv2 is comparable since the context switch overhead dominates. Once this overhead has been amortized, the variable cost of data copying becomes the dominant factor. Further, the bandwidth of SGXv1 is severely reduced for small data sizes once paging kicks in. This is not the case for SGXv2 due to the increased EPC capacity.

Summary. With regard to data scalability, we observed that the increased EPC of SGXv2 enables database engines running inside enclaves to use orders of magnitude more memory. This allows to store more data inside an enclave and to copy larger chunks of data with one enclave call, which makes SGX more practical for building database engines. While the new NUMA support allows enclaves to span multiple NUMA nodes, developers should pay attention to the additional overhead of memory accesses across NUMA regions and to the lack of a fine-grained control of NUMA allocation due to the missing `libnuma` library inside SGX. Finally, although the amount of memory usable by enclaves has been increased significantly, the cost of paging is still tremendous and must be considered when data exceeds the total EPC capacity.

12.3.2 Trusted SGXv2 vs. Untrusted

In the previous experiment we identified two critical factors which impact performance, i.e., remote NUMA access and paging. In this experiment we instead shed light on the overhead of trusted SGXv2 vs. untrusted execution. To that end, we execute a YCSB-like benchmark with varying read/write (update) ratios where all data fits into the available EPC capacity. This avoids effects like paging which we analyzed before. Moreover, we use the same B-Tree as in the first experiment, however, we populate it with 10 M key-value pairs (using 8 B for keys and 128 B for values).

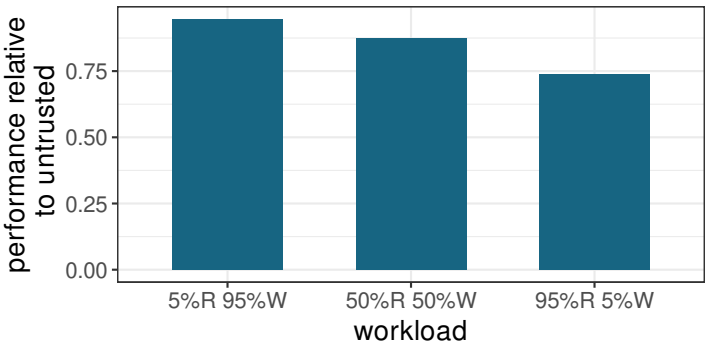


Figure 12.5: Relative performance of various YCSB-like workloads in SGXv2 compared to an untrusted baseline. SGXv2 has only a relatively small overhead across all workloads.

Figure 12.5 shows the performance of SGXv2 relative to the untrusted baseline. We can observe that the read-heavy workload appears to have a higher overhead than the write-heavy workload. This effect can be explained by comparing both workloads in depth. As discussed in Section 12.3.1, the main overhead of SGXv2 stems from higher memory access latencies. The read-heavy workload is dominated by memory loads and therefore highly impacted by these increased latencies. In contrast, the write-heavy workload contains more CPU-intensive operations because it generates random 128 byte values as updates. Since CPU-intensive operations have similar to identical performance inside and outside enclaves, they mask the impact of memory access latencies overall.

Summary. Comparing the performance of our data-intensive workload between untrusted and trusted execution, we saw a maximum performance reduction of approximately 25 % when all the data fits into the EPC. We conclude that for many use cases requiring the security of a TEE, this is probably a good trade off.

12.3.3 Overhead of Trusted I/O

In addition to accessing data in memory, DBMSs also need to access and store data on disk for purposes like recovery. Therefore, in this experiment we look at two different flavors of enabling trusted I/O in SGXv2: `sgx_fwrite` and `sgx_seal_data`. With `sgx_fwrite` the Intel Protected File System Library [80] provides the trusted equivalent to `fwrite` for writing binary streams. Using this function, data is encrypted and integrity protected before being written to an untrusted disk. In contrast to `sgx_fwrite`, `sgx_seal_data` does not perform file I/O, but only encrypts and integrity protects the data. Hence, an additional interaction with untrusted code to perform file I/O is required (`OCALL`).

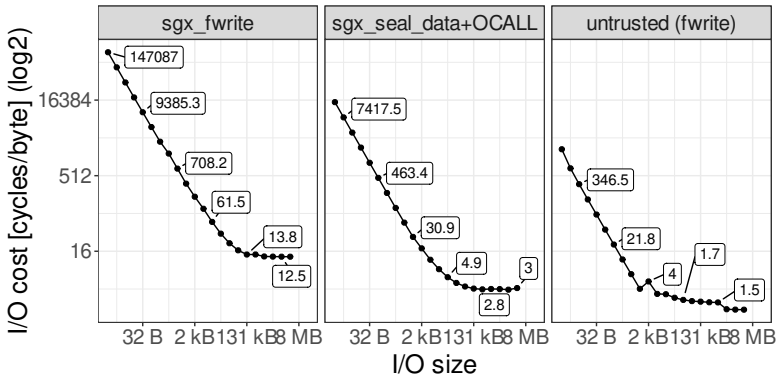


Figure 12.6: Effect of I/O on CPU utilization in cycles/byte of the two trusted I/O variants (`sgx_fwrite` and `sgx_seal_data`) vs. untrusted I/O (`fwrite`).

To ensure that data is written to disk, we use `fflush` and its trusted counterpart `sgx_fflush`.

In Figure 12.6 we show the effect of various I/O sizes on the normalized CPU utilization (cycles/byte). We compare the two secure I/O variants with untrusted file I/O (we use `fwrite` because it resembles `sgx_fwrite`). All three variants have in common that larger I/O sizes significantly reduce CPU utilization because most overhead is per function call, not per byte. Surprisingly, `sgx_fwrite` is much more expensive than `sgx_seal_data`. In fact, when writing only 2 bytes `sgx_fwrite` spends 20× more cycles than `sgx_seal_data` (note the log-scale on the y-axis). This initial function call overhead of `sgx_fwrite` can be partially amortized by writing larger I/O sizes at once. However, even with large batch writes `sgx_fwrite` is at best 4× more expensive than `sgx_seal_data`. We regard a more in-depth analysis of the differences between both protected file I/O strategies as future work.

Summary. Our key insight is that the choice of the protected I/O library can impact performance heavily. Moreover, regardless of the I/O strategy, we can see that larger I/O sizes burn less cycles per byte. For DBMS designs based on SGXv2 this means that optimizations such as group commit are important to collect enough log entries before flushing.

12.3.4 Other Effects of Enclaves

The increased EPC capacity offered by SGXv2 not only suggests that more data can be stored inside an enclave, but it also provides the possibility to run multiple (larger) enclaves on a SGXv2 capable system at the same time. This is especially interesting for

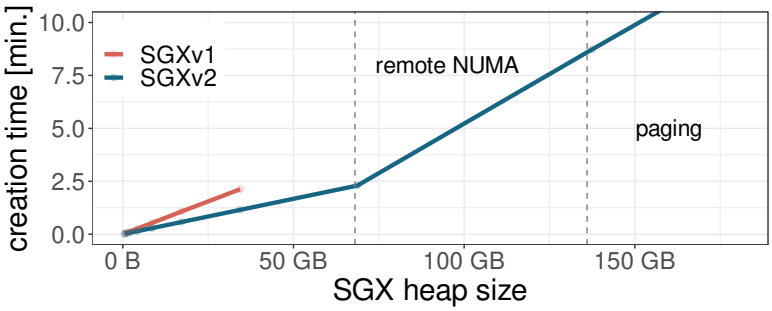


Figure 12.7: Duration of enclave creation for varying heap sizes. As soon as the heap size exceeds the EPC (≈ 64 GB in SGXv2), a significant increase in creation time can be observed.

cloud settings since not only larger data sizes can be kept in the enclave, but also several DBMS instances could be run on the same SGXv2 server.

Creation Time of Enclaves. In this experiment, we investigate the trade-off between the creation time and the size of the enclave (`HeapMaxSize`). Choosing an appropriate `HeapMaxSize` for the application is important because when the `HeapMaxSize` is reached the application crashes on the next memory allocation (as shown in Figure 12.2). Therefore, we vary the `HeapMaxSize` and measure the time it takes to create the enclave in SGXv2. We fixed the value for `HeapMinSize` to 65536 byte. Due to the support for dynamic memory allocation in Intel SGXv2, one would expect that the duration of enclave creation is stable regardless of the `HeapMaxSize` setting. However, as can be seen in Figure 12.7, we observe that the enclave creation time increases significantly the more heap size is configured via the `HeapMaxSize` setting. More importantly, as the configured heap size exceeds the capacity of the local NUMA node, we can observe an even stronger slowdown of enclave creation. This suggests that in cases where an on-the-fly creation of enclaves is required for DBMSs, extra caution should be applied. Especially for larger enclave sizes it seems undesirable to create short running on-the-fly enclaves.

Performance of Concurrent Enclaves. In this experiment, we show the effect of running multiple DBMS instances in parallel on the same server. For this experiment we create three enclaves with a `HeapMaxSize` setting of 64 GB, i.e., each enclave is guaranteed to occupy the total EPC of a NUMA node. All enclaves are started concurrently and run the same B-Tree insertion workload as introduced in Section 12.3.1. This means that the enclaves initially utilize only a small fraction of their heap size, but their memory consumption increases as time passes. All three enclaves are pinned to the first NUMA node using the `numactl` tool. The results of this experiment are depicted in Figure 12.8.

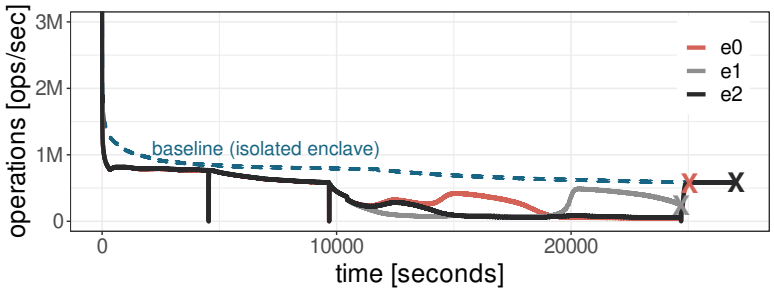


Figure 12.8: Performance of 3 concurrently running enclaves executing the same workload as in Figure 12.2. When the aggregated size of the enclaves exceeds the total EPC, the performance of all enclaves starts to become unpredictable.

For comparison, we additionally show the performance of only a single enclave running the same workload in isolation (blue dashed line).

Based on Figure 12.8 we can make several interesting observations. First, although all three enclaves are small in the beginning and fit into the EPC of the first NUMA node, the performance of the enclaves is lower than the baseline; i.e., concurrently running enclaves affect each others performance. Second, as long as all enclaves fit into the EPC (regardless of the NUMA node), their performance behaves similarly. This indicates that all three enclaves are treated equally in terms of scheduling and memory allocation (the BIOS setting for SGX QoS is OFF). Third, due to the aggregated enclave sizes, the performance drops caused by remote NUMA access or paging occur earlier. Fourth, once paging sets in, the performance of the different enclaves becomes unpredictable. Only after the first enclave finishes execution (depicted by an `X` in the figure), the performance of the other enclaves recovers due to more memory being available. Moreover, once the two enclaves *e0* and *e1* finish their execution, the performance of the last enclave *e2* increases to the same level as the baseline.

Summary. Based on our experiments we observed that the creation time of an enclave is not only affected by the size of the enclave, but is also influenced by NUMA and paging effects. Further, we showed that concurrently running enclaves decreases the performance of the enclaves and could even lead to unpredictable performance when EPC capacity is exceeded. Both observations are important for scaling out databases (by spinning up additional instances) or supporting concurrent customer workloads.

12.4 Related Work

To the best of our knowledge, most related work revolves around SGXv1. These works cover several different areas from within and outside of the database community. In the following, we focus on work that addresses the limitations of SGXv1, builds secure DBMSs for SGXv1 or studies the performance properties of SGXv1.

Overcoming SGX limitations. Several previous work from the system community looked at how the overhead of context switches in SGXv1 (i.e., `ECALL/OCALL`) [131, 166, 179] or paging [107, 162] can be overcome. While these works present an in-depth view of the corresponding limitation, our work followed a broader focus to identify new challenges for SGXv2 such as NUMA effects.

Building enclave-enabled DBMSs. Studying Intel SGX from a data management perspective is another area that has attracted several research efforts. Besides works that discuss how Intel SGX and TEEs in general can be used for trusted data processing [9, 73], recent work looked at how DBMS can be engineered to make use of the capabilities of SGXv1 [10, 59, 67, 93, 143, 159, 195]. Our work instead focused on SGXv2 and showed that SGXv2 includes several improvements that open up new opportunities for database design and necessitate a re-evaluation of previous assumptions.

Analyzing performance properties. To gain more detailed insights into the performance of SGXv1 applications, several works propose tools [96, 110, 161, 178] that can be used to measure different performance aspects such as page faults and enclave transitions [96, 178]. Moreover, there has been work on studying the performance of SGXv1 in different settings [3, 45, 194] such as virtualized environments [45] or comparing Intel SGX to other TEE technologies [123]. Most related to our paper are the works by Harnik et al. [76] and Maliszwski et al. [112]. While the former looks at the performance of SGXv1 for different data encryption settings and related access patterns, the latter studies the performance of SGXv1 for join algorithms.

12.5 Conclusions & Future Directions

In this paper, we benchmarked the second generation of Intel SGX (SGXv2) focusing on the question how this new generation might change the design of future secure DBMSs.

Main Findings. As a first finding, we showed that SGXv2 delivers on its promise to improve the capacity of enclaves. Compared to SGXv1 the capacity is two to three orders of magnitude larger depending on the configuration. For example, in our setup

we showed that an in-memory B-Tree can scale up to about 120 GB of data and still provide high performance and that SGXv2 has only around 25 % overhead compared to a pure in-memory B-Tree. In contrast, this workload performs 25× worse in SGXv1 due to the limited capacity and the involved paging. Moreover, the support of SGXv2 in server-grade CPUs brings additional benefits like larger caches, higher core counts, and scaling across NUMA regions. Therefore, we belief that SGXv2 is a huge step towards building high performance in-memory databases completely inside an SGX enclave.

However, using the Intel SGX technology still comes with several pitfalls. First, memory management needs to be designed carefully to take the additional overhead for, e.g., remote NUMA access into account. Second, our experiments indicate that different implementations of trusted disk I/O can have performance differences in orders of magnitude, requiring a careful choice of libraries and functions. Finally, SGXv2 does not eliminate the performance penalty of paging, but rather shifts it to larger memory sizes.

Future Work. While this paper provides a first systematic study on the basic aspects of SGXv2 as discussed before, it is only a first step in the direction of designing efficient, reliable and secure database engines given the potentials of SGXv2. Prior work on secure enclave databases mainly focused on the limitations of SGXv1 by, e.g., only placing certain DBMS components inside the enclave. In contrast to that, we believe that the whole DBMS and its data can be secured inside the enclave by using the advancements of SGXv2. As such, the design of all DBMS core components needs to be revisited to take the unique characteristics of SGXv2 into account, e.g., with regard to memory management (NUMA effects) and disk I/O (logging).

Finally, we plan to refine the results of our study in several directions to provide a more comprehensive view about designing DBMSs for SGXv2. For instance, studying the effects of multi-threading, taking a closer look at the effect of cache misses and analyzing the performance impact of enclave calls in more detail are important extensions of our work. Further, not only looking at concurrently running enclaves, but also studying the effect of untrusted code running in parallel to an enclave is another interesting direction for future work.

12.6 Acknowledgements

This work was partially funded by the National Research Center ATHENE, the BMWK project SafeFBDC (01MK21002K) and the BMBF project TrustDBle (16KIS1267). We also thank the SAP HANA team for the support.

13 Towards Decentralized Parameter Servers for Secure Federated Learning

Abstract

Federated learning aims to protect the privacy of data owners in a collaborative machine learning setup since training data does not need to be revealed to any other participant involved in the training process. This is achieved by only requiring participants to share locally computed model updates (i.e., gradients), instead of the training data, with a centralized parameter server. However, recent papers have shown that privacy attacks exist which allow this server to reconstruct the training data of individual data owners only from the received gradients. To mitigate this attack, in this paper, we propose a new federated learning framework that decentralizes the parameter server. As part of this contribution, we investigate the configuration space of such a decentralized federated learning framework. Moreover, we propose three promising privacy-preserving techniques, namely model sharding, asynchronous updates and polling intervals for stale parameters. In our evaluation, we observe on different data sets that these techniques can effectively thwart the gradient-based reconstruction attacks on deep learning models, both from the client side and the server side, by reducing the attack results close to random noise.

Bibliographic Information

The content of this chapter was previously published in the peer-reviewed work Muhammad El-Hindi, Zheguang Zhao, and Carsten Binnig. "Towards Decentralized Parameter Servers for Secure Federated Learning." In: *Proceedings of the 11th International Conference on Data Science , Technology and Applications, DATA 2022, Lisbon, Portugal, July 11-13, 2022*. Ed. by Alfredo Cuzzocrea, Oleg Gusikhin, Wil M. P. van der Aalst, and Slimane Hammoudi. SCITEPRESS, 2022, pp. 257–269. DOI: 10.5220/0011146300003269. URL: https://doi.org/10.5220/0011146300003269.

13.1 Introduction

Motivation. Federated learning (FL) [118] enables organizations to learn predictive models in a collaborative way. There are several reasons for using federated learning. One is that each individual organization has too little training data and thus data of multiple organizations is needed to train a deep model. Moreover, FL is interesting since the training data does not leave organizational boundaries [88, 103]. This is especially helpful if the data contains private information that needs to be protected. A prime example for FL is healthcare. There, multiple hospitals want to learn a model over their joint data for the classification of a new disease, but need to keep patient data local due to legal and privacy regulations [26].

The predominant architecture for federated learning today is to use a central PS that collects the model updates from all participants and aggregates them. In this setup, the data owners participate in the training process by sending their locally computed model updates to the central server, which combines these updates into a global model [118]. The original assumption was that such an approach is able to protect the privacy of each participant's training data, since only model updates and not the training data itself is exchanged with the server. Yet, recent works [193, 196] have shown that privacy attacks exist that allow an attacker to successfully extract information about the training data by observing the model updates (i.e., exchanged gradients). More surprisingly, these attacks showed that it is possible to successfully reconstruct individual training examples with high accuracy (e.g., a full picture used for training) [193, 196]. Even worse, these attacks are also applicable for different model architectures [69, 177].

Meanwhile existing defense strategies remain limited in preventing privacy attacks in federated learning [1, 23, 69, 139, 196]. Most strategies either significantly reduce the learning accuracy (e.g., using noisy gradients can result in 30% less accuracy [196]) or have other limitations such as assuming a trusted central PS, or being incompatible with widely used model architectures (e.g., secure aggregation [23]). Generic cryptographic primitives such as homomorphic encryption, although not affecting accuracy, typically incur significant overhead resulting in longer training times (e.g., by 100x [139]).

Contributions. In this work we take a different, system-driven approach by modifying the federated learning framework. We propose to architect the training system around decentralize parameter servers. Further, we initiate the study of the configuration space of such a decentralized FL framework, called `P2Sharding`, w.r.t. its security against client-side and server-side attacks. We propose three promising privacy-preserving techniques, namely model sharding, asynchronous updates and polling intervals on stale parameters.

Our evaluation on the CIFAR10 and MNIST datasets shows that these configurations can effectively thwart the gradient-based reconstruction attacks on deep learning models by reducing the attack outcome close to random noise.

Outline. The remainder of this paper is organized as follows. Section 13.2 gives an overview of federated learning and the common basis for existing privacy attacks. We present our privacy-preserving sharding framework based on a decentralized parameter server architecture in Section 13.3. We evaluate these configurations in Section 13.4 and conclude with related work and a summary.

13.2 Background

In the following, we briefly discuss the basics of federated learning and the typical structure of privacy attacks on federated learning. Finally, we review existing defense strategies when using a central parameter server and their limitations.

13.2.1 Federated Learning

Federated learning is a collaborative learning setting in which multiple parties jointly train a machine learning model. To coordinate the learning process, federated learning typically uses a central parameter server to initialize a global model, and interacts with a set of participants (clients) to collect updates to the model. One distinct aspect of this setting is that participants never upload their data to the server, and the only information the server collects is model updates computed on privately held data [22, 88, 118]. The de facto class of training algorithms deployed in the federated setting for deep learning is stochastic gradient descent (SGD). SGD updates are gradients of model weights towards minimizing a loss function computed on batches of the training data [95]. Training data is possibly iterated through multiple times locally before sending the final update to be averaged to the server [118]. The parameter server aggregates updates from each client, and applies changes to the model parameters either synchronously [33] (accepting one update per client in a round) or asynchronously [42] (allowing clients to progress independently).

Figure 13.1 illustrates the basic steps. Each client k downloads the parameters W, computes the gradient $\nabla_W L(W; x_k, y_k)$ for the loss function L on its local data (x_k, y_k) (denoted as $\nabla W(x_k, y_k)$) . The server collects each gradient and aggregates it into the global model $W \leftarrow W - \eta \sum_{k=1}^{K} \beta_k \nabla W(x_k, y_k)$ with the weights β_k and the learning rate η. The interactive process iterates until convergence.

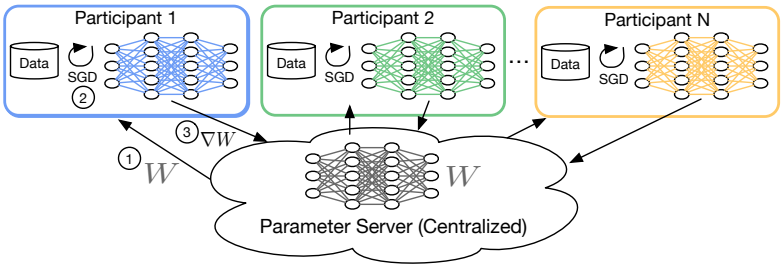


Figure 13.1: Federated learning with a centralized parameter server using stochastic gradient descent (SGD). Each participant ① downloads global model parameters W from the server; ② computes local model updates ∇W based on privately held data; ③ sends local updates to the server. The server aggregates the local updates to global parameters.

13.2.2 Privacy Attacks

Since its inception, federated learning has emerged as a common paradigm to train on real-world privacy sensitive data. It was commonly believed that the information transmitted over the network in federated settings contains only minimal updates for improving the model, and therefore reveals much less information about the private training data [95, 118]. However, this conceived advantage has been cast into doubt by recent work on privacy attacks that revealed that even the model updates contain enough information to compromise the data privacy [69, 120, 196].

The authors [120] show that gradients carry valuable information that can be leveraged by attackers to leak unintended knowledge about the private data. The authors of [196] were the first to show that it is even possible to reconstruct full images and text data with high precision from gradients sent by clients and thus breach the privacy of the learning process. This class of reconstruction attacks only requires access to the exposed model updates (i.e., the gradients ∇W), plus the parameters W and is therefore applicable to most federated settings.

The common basis for these attacks is the following optimization problem: Find some estimated data x' (e.g., an image) and its label y' (e.g., the classification of the image) such that its gradient $\nabla_W L(x', y')$ is closest to the transmitted client gradient ∇W for its private input x and label y. In other words, the distance of the two gradients with regard to a distance function $\mathbb{D}()$ is minimized:

$$\arg\min_{x',y'} \mathbb{D}(\nabla W, \nabla_W L(W, x', y')) \tag{13.1}$$

The original attack [196] used an L-BFGS solver [106] with randomly initialized (x', y') to optimize Eq. (13.1) based on the euclidean distance for a training batch size of 1. The attack accuracy was improved by [193] in which the private labels y are recovered analytically from the direction of the gradients. More recently, [69] used cosine similarity in Eq. (13.1) to yield a stronger attack for larger batch sizes.

13.2.3 Existing Defense Strategies

The key ingredients for the reconstruction attacks in Eq. (13.1) are: (a) the access to the entire model parameters W, and (b) the view of the entire gradient ∇W. Several countermeasures based on differential privacy [196], gradient compression [105, 196] and cryptography [23, 139] were proposed. Yet, they have several limitations:

Differential privacy approach. The differential privacy-based approach in [196] adds Lapacian and Gaussian noise to the local model updates before transmission, but larger noise is often necessary for enough privacy protection, which tends to degrade the training accuracy significantly.

ML-based approach. Gradient compression [105] by dropping out small gradient components has shown to be effective only when the sparsity of the gradient exceeds 20% [196]. Yet, this method does not prevent a corrupt server from inverting the gradients from observed model iterates, a crucial step for reconstruction (cf. Sec. 13.3.2.2).

Cryptographic approach. Some cryptographic protocols are still ineffective against the attack in [69], e.g., secure aggregation [23] or, in the case of Homomorphic encryption [139], incur prohibitive overhead and are limited to integer fields. A multi-party computation-based approach [72] also has large overhead when scaling to more than two servers or clients or is not directly applicable to the federated setting as is the case for the approach in [124].

13.3 Decentralized Secure FL

In the following, we present an alternative, system-driven approach for enhancing privacy in federated learning without the drawbacks seen in the above mentioned approaches. We first give an overview of our decentralized parameter server that partitions the model into shards. We then discuss how these model shards are created and updated on different server instances in order to enhance data privacy.

13.3.1 Decentralized Parameter Server

Centralized parameter server presents a single point of security vulnerability because a corrupt server can see all updates from all clients and launch the gradient-based reconstruction attack as is. By contrast, we proposed to base our defense mechanism on decentralizing the parameter server.

Instead of congregating the model parameters on a centralized PS, our framework, called `P2Sharding`, distributes trust among several independent parameter server instances, each hosting a fraction of the model, called a *shard*. No single server instance holds the entire model W, nor receives the entire gradient ∇W. Thus just by design, sharding enhances data privacy by preventing the adversary from having a consistent view of the entire gradients and the model parameters, both the key elements in the reconstruction attacks.

Federated Settings. We consider a set of K clients who participate to train a model on their joint data. The model W is partitioned into M shards $W^1, \cdots, W^M$ using a configurable strategy, and each shard W^m is hosted on a separate parameter server instance. Each server instance W^m receives the gradient shard ∇W^m from each client k. Each client downloads the full model iterates $W = (W^1, \cdots, W^M)$ by sending requests (i.e., polling) to the M server instances.

Note that the amount of exchanged data between clients and the parameter server shards is similar to the classical central parameter server setting. In both cases, clients need to download the full model W or send all gradients ∇W to a remote location. The only additional overhead introduced by our framework are additional messages/connections since clients have to communicate with multiple remote endpoints.

The framework can adapt to several federated settings using different configurations. For example, for enterprise clients where each has enough computation resource, each of the M server instances can be co-located with a client. An example is shown in Figure 13.2. In contrast, a server-aided model can be used to outsource the M server instances onto M independent physical servers, which is more suitable for resource-constrained clients running on edge devices such as mobile phones.

Security Model. The `P2Sharding` framework assumes all parties to be *semi-honest*, that is each client and server follow their prescribed protocol and only attempts to extract more information from the other client's data[1]. Moreover, at least one client and one

[1]Protocols that assume a semi-honest setup prevent inadvertent leakage of information between parties, and are thus useful if this is the concern. In addition, protocols in the semi-honest model are quite efficient, and are often an important first step for achieving higher levels of security.

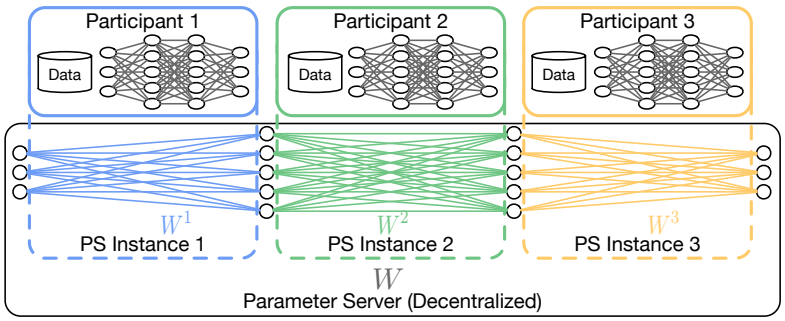


Figure 13.2: Decentralized Parameter Server Architecture for an example with 3 shards (W^1, W^2, W^3). *The parameter space W is partitioned across several independent PS instances managed by different participants. In the example, the PS instances are co-located with each participant, but they can also be hosted by independent physical servers.*

shard are assumed to be non-colluding with the other parties (i.e., we have at least two non-colluding shards). Otherwise, the security reduces to that of a centralized server.

We also consider cases where (1) a subset of PS instances or (2) client and server instances collude (e.g., due to being co-located as shown in Figure 13.2). In the first case, the goal of the colluding PSs is to extend their knowledge about exchanged gradients. In the second case, the attacker only wants to get access to the full model iterates $W^1, \cdots, W^M$ since the client regularly receives the latest global model. This provides crucial information for reconstruction because the attacker can estimate the full gradients from the *history* of the received model iterates.

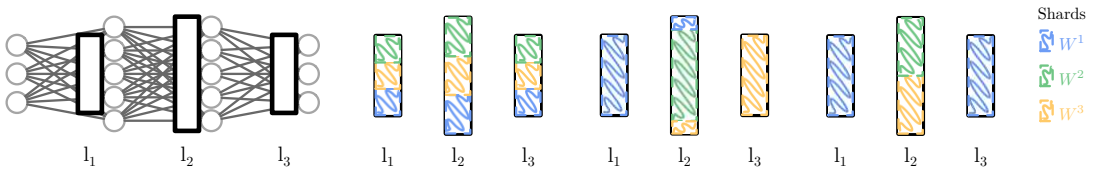
In `P2Sharding`, we provide several configurations to help reduce the risk of both PS-side and client-side attacks as explained next.

13.3.2 Privacy-Preserving Configurations

We turn to investigate several configurations within the `P2Sharding` framework that can enhance privacy. These configurations are rooted in system designs, and therefore present a different tool for protecting private training data than differential privacy or end-to-end cryptographic approaches.

13.3.2.1 Model Sharding

Since a shard contains only a partition of the model, the information leakage to a corrupt server is limited to its hosted shard. Which data can be reconstructed from a shard depends on how the model parameters are distributed. For example, if the penultimate



(a) Layers of model parameters (b) Uniform (c) Slicing (d) Boundary-aware

Figure 13.3: *Partitioning Strategies for a neural network (NN) model architecture with four layers (a); i.e., three layers of parameters since the input layer has no parameters. (b) The* Uniform *strategy creates equally sized partitions that span across all layers. (c) The* Slicing *strategy divides the parameters into equally sized consecutive slices (only some of which span layers). (d) The* Boundary-aware *strategy guarantees that while a shard may posses parameters from multiple layers, these layers are not adjacent. This strategy might create shards of different sizes.*

layer in a feed-forward, softmax-output neural network were allocated to the same shard, its corrupt server may learn the training label [193]. In the following, we describe three sharding strategies that provide strong privacy in our evaluation, and we show how to vary the shard size to increase resilience against collusion of multiple shards.

Uniform Sharding. The idea of the *Uniform*-strategy is to create S similar-sized shards that store the same fraction of parameters from each layer, as depicted in Figure 13.3b. As shown in the figure, all three shards have the same size and equally span all layers.

In order to achieve this partitioning, the strategy uniformly assigns the parameters p_i to the shards W^j. This can be expressed by selecting a shard for a parameter at index i using a shared hash function $\mathcal{P}$:

$$\mathscr{S}(i) = (\mathcal{P}(i) \bmod S) + 1 \tag{13.2}$$

Uniform sharding creates equally-sized partitions. However, it is completely oblivious of the different layers and hence a participant might possess parameters from all layers.

Slicing Sharding. Another model-oblivious technique is the *Slicing*-strategy. In this strategy, the parameter space is contiguously divided into S equally-sized partitions, as depicted in Figure 13.3c. More formally, this strategy can be described by the following function:

$$\mathscr{S}(i) = \left\lceil \frac{i}{\frac{|W|}{S}} \right\rceil \tag{13.3}$$

Intuitively, we can linearly iterate over all parameters in W and assign the first $\frac{1}{S}$ parameters to the first shard (blue) and the second $\frac{1}{S}$ to the second (green) shard and so on.

In terms of shard size, this strategy creates equally sized shards as shown in Figure 13.3c.Moreover, shards span only a few layers of a NN as depicted in the example. There, the blue and yellow shard span multiple layers but the green shard is limited to a single parameter-layer. Hence, this strategy differs from the *Uniform*-strategy since it reduces the number of layers from which a participant holds parameters.

Boundary-aware Sharding. In contrast to the previous strategies, this strategy guarantees that a partition does not hold any parameters of adjacent parameter-layers, e.g., the blue shard in Figure 13.3d does not hold any parameters from layer l_2. Further, this strategy might create partitions with a different number of parameters in order to prevent a shard from storing parameters from adjacent layers. This situation can be seen in Figure 13.3d since the blue shard is bigger than the remaining two shards.

Such a mapping can be expressed as follows:

$$\mathscr{S}(i,l) = \begin{cases} (\mathcal{P}(i) \bmod \lfloor \frac{S}{2} \rfloor) + 1, & l \bmod 2 = 1 \\ (\mathcal{P}(i) \bmod \lceil \frac{S}{2} \rceil) + \lceil \frac{S}{2} \rceil, & l \bmod 2 = 0 \end{cases} \tag{13.4}$$

That is, we distinguish two sets of shards, one is responsible for odd layers ($l \bmod 2 == 1$) while the other stores the parameters of the even layers ($l \bmod 2 == 0$). We create these two sets by splitting the shards in two halves ($\lfloor \frac{S}{2} \rfloor$). For example, we can observe in Figure 13.3d that shard W^1 is responsible for the odd layers, while shard W^2 and W^3 are responsible for the even layer. Moreover, within every set of shards, the parameters are uniformly distributed with the help of a uniform shared hash function $\mathcal{P}(i)$. Note that, we have decided to assign more shards to the even layers (i.e., using $\lfloor \rfloor$ as first bound), since even layers tend to contain more parameters (cf. Figure 13.3d) than, e.g., the odd in-/output layers. While the *Boundary-aware*-strategy does not create equally-sized partitions, it takes the layer boundaries of the model architecture into account and avoids that shards receive gradients of two adjacent layers.

Shard Sizes. The `P2Sharding` framework provides the shard size configuration parameter for adjusting the privacy of sharding in light of collusion attacks. The effect of a collusion attack is the same as if several shards were combined to form a larger subset of the model. Hence, the shard size is privacy-sensitive in that it can be used to control for the expected number of collusions. Intuitively, if C server instances collude, their knowledge is the union of their shards. If the number of colluding shards becomes large

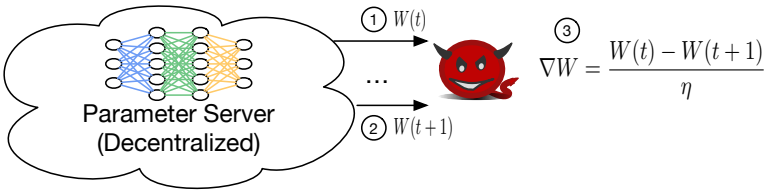


Figure 13.4: Attackers recovering full gradients from model iterates. *The estimation takes three steps: Downloading all parameters from all shards repeatedly (① and ②) and assembling the global model at t and $t+1$. Finally, both models are used to recompute the gradients that were used to update W_t to W_{t+1} ③. For the Stochastic Gradient Descent (SGD) algorithm, the gradients can be recomputed with the shown formula.*

enough the reconstruction attack becomes easier. So by reducing the maximum shard size S, `P2Sharding` can control the amount of leakage during C shard-collusion to be $\leq C \cdot S$ gradient components.

13.3.2.2 Asynchronous Updates

This configuration aims to reduce the risk of malicious client and server collusion.

Leakage from Synchronous Updates. Under synchronous training, with the help of a colluding server, a malicious client can retrieve model snapshots of different iterations for a targeted victim, then uncover the full gradient to launch the gradient-based attack. As depicted in Figure 13.4 an attacker could continuously retrieve (i.e., poll) the latest model from all parameter servers to determine $W(t)$ ① and $W(t+1)$ ② and re-compute the gradients ∇W based on the retrieved models ③. This observation especially holds true for standard SGD which uses the update rule $W(t+1) = W(t) - \eta * \nabla W$ with learning rate η (i.e. $\nabla W = \frac{W(t)-W(t+1)}{\eta}$) for computing the new model parameters. In other SGD variants, gradients may only be estimated from the full history of model iterates $\{W(t)\}_t$.

Perturbation by Asynchronous Updates. Our approach to mitigate the aforementioned gradient recovery is to use *asynchronous federated updates*. In this setting, concurrent client updates to the same model shard are applied without a global order, which may lead to the effect of different clients overwriting each other's model updates, or some clients update some shards more often than others. As a result, data reconstruction becomes harder as shown in our evaluation.

Intuitively, if multiple updates to each shard are incorporated asynchronously in parallel, then the full model iterates may contain out-of-sync values. This leads to non-uniform

directional perturbation in the estimated full gradients. Following the model iterate analysis for asynchronous SGD [113], such perturbation can be modeled as essentially injecting random noises $\mathcal{E}(t)$ on the model iterates to the adversary's view g:

$$\nabla\widehat{W} = g(\{W(t) + \mathcal{E}(t)\}_t) \tag{13.5}$$

where $\mathcal{E}(t)$ is a vector that describes perturbation per shard j at time t. Thus the attacker only sees a gradient with noisy rotation and stretch. Since it is the angle of the gradient that contains most information about the data [69], our evaluation shows that perturbed gradients indeed make data reconstruction harder while maintaining model accuracy high.

13.3.2.3 Polling Intervals

Asynchronous updates can lead to degraded model quality for some models and datasets [113]. For these cases where asynchronous updates are not applicable, we introduce *polling intervals* as another means to perturb model iterates for privacy. The idea is to create stale parameters, while still allowing for *synchronous* updates to the entire model for more stable training.

More specifically, each server instance j can independently implement a manual random delay τ_j, called the polling interval, for each connecting client. For each request, the server instance checks the client's identity, and answers with an outdated model shard $W^j(t - \tau_j)$. All such artificially out-of-sync model shards form a client's view on the entire model, which crucially still contains inconsistent values at all time. As such this setting can also resist the reconstruction attack. Specifically, the difference between the latest but hidden model shard $W^j(t)$ and the polled model shard $W^j(t - \tau_j)$ can be viewed as the source for the perturbation $\mathcal{E}(t)$ in Eq. (13.5) at a particular time t. As such, polling intervals provide a deterministic way for inducing perturbation through staleness.

An important point to note here is that each parameter server instance is assigned to a separate trust domain with its internal states hidden from the environment, that is, it only exposes its interface to the other parties. Hence, each PS instance can actually implement different strategies of which data can be read from the local shard. That way, each server instance in `P2Sharding` can control parameter staleness local to its shard by keeping track of the updated model versions (i.e., one version per iteration) and return a model with a definable staleness to the client.

Finally, another interesting observation is that model iterates with the same increasing polling interval actually also produce a similar effect of increasing the batch size, since it results in accumulated updates across multiple training examples. As shown in [196] and in [69] an increased batch size makes privacy attacks harder. We validated the effectiveness of polling interval in Section 13.4 for both privacy and performance.

13.4 Experiments

Overall the goal of the evaluation is to analyze the effects of the different configurations of `P2Sharding` on privacy protection. We first demonstrate that the privacy of the training data can be significantly strengthened with the help of `P2Sharding`'s decentralized setting when compared to a centralized parameter server. We then decompose `P2Sharding` into each of its configurations. We show that for varying data and model complexities `P2Sharding` is able to provide configurations under which the attack results remained unrecognizable, i.e., close to random noise.

13.4.1 Setup & Metrics

In our evaluation we used a similar setup as the original reconstruction attack [69, 196], because our goal is to show that the range of configurations in `P2Sharding` can mitigate such attacks successfully. More details on the setup are provided below:

Dataset Complexities. We used the `MNIST` [98] and `CIFAR10` [97] datasets as representatives for less and more complex datasets. `MNIST` is considered less complex than `CIFAR10` for it contains only black-and-white images of handwritten digits, whereas `CIFAR10` contains colored images of more complex objects.

Model complexities. As models, we used `LeNetZhu` as in [196] and `ConvNet` as in [69] to represent lower and higher model complexities. Both models are widely-used convolutional neural networks. However, one noticeable difference is their depth, or number of layers, since `ConvNet` is deeper than `LeNetZhu`. Another important difference from a reconstruction attack perspective is the used activation function. `LeNetZhu` uses a Sigmoid function which makes the network inherently twice-differentiable, resulting in a smoother optimization problem in attack formulation (Eq. 13.1). `ConvNet` by contrast uses ReLU which is non-smooth and non-differentiable at 0.

Attack Implementation. For the attacks, we used the code in [69] (cosine similarity based reconstruction attack) and adapted their hyper-parameters for the actual learning process (i.e., SGD with learning rate $\eta = 0.1$). We integrated our partitioning strategies

by implementing a separate module that limits the gradients that are accessible to the attacker (i.e., a PS instance).

In the following, we always execute 5 attacks on randomly selected training batches. Thereby, we set the batch size to 1, as done in [196], since this represents the worst-case that `P2Sharding` has to defend against — as mentioned in [196], a bigger batch size makes the attack harder. Further, we executed each experiment three times.

Privacy Metrics. In order to evaluate the privacy against reconstruction attacks, we measured the *structural dissimilarity* (`DSSIM`) of a reconstructed image from its targeted private image. The `DSSIM` is derived from the *structural similarity* (`SSIM`) [176] as $DSSIM(x, y) = \frac{1-SSIM(x,y)}{2}$. We used this measure instead of the *mean squared error* (`MSE`) as in [196] or the `MSE`-based peak noise-to-signal ratio as in [69], because `DSSIM` not only captures pixel-local deviation but more importantly the structural differences. Hence, it has been shown to be a superior measure for signal fidelity such as perceptual distortion or *recognizability* [28]. For example, a color-inverted `MNIST` image (a black-and-white digit) will have extremely high `MSE`, but it does not correlate with the privacy of the image as the structure of the digit remains obviously the same. In contrast, the `DSSIM` value for the same image will be kept low to reflect the structural similarity. In general, a higher `DSSIM` suggests the reconstructed image deviates more from the private image and may even be unrecognizable. We show a sample of images from `MNIST` and `CIFAR10` with varying `DSSIM` in Figure 13.5. Three reconstructed images were randomly selected to represent different `DSSIM` intervals. It can be observed that when the `DSSIM` reaches a value > 0.45 the reconstruction become unrecognizable, which coincides with the `DSSIM` of random noise.

In our evaluation, we recorded the empirical distribution of `DSSIM` over `MNIST` and `CIFAR10` under repeated attacks, and reported the average and minimum `DSSIM` as average and worst-case performance.

Baselines. In the following we compared `P2Sharding` to two baselines. As the first baseline, we compared against a centralized parameter server with no sharding. This setting represents an insecure setup and we show that `P2Sharding` configurations achieve much stronger privacy. This baseline will be shown as red dashed line in the experiment plots. On the other hand, the best privacy against reconstruction is to have attack results close to random noise. Hence, we also compared against a randomly generated picture in which each pixel is sampled i.i.d. from a uniform range (shown as a green dotted line in the following figures).

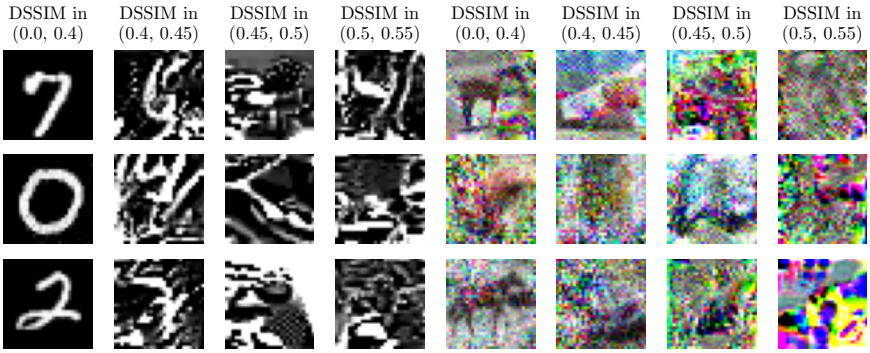


Figure 13.5: Visualization of different DSSIM ranges for attacks on the MNIST (left) and CIFAR10 (right) dataset. *A higher* DSSIM *value means that an attack was unsuccessful and privacy is protected. For* MNIST *and* CIFAR10 *privacy is protected beginning with a* DSSIM > 0.45.

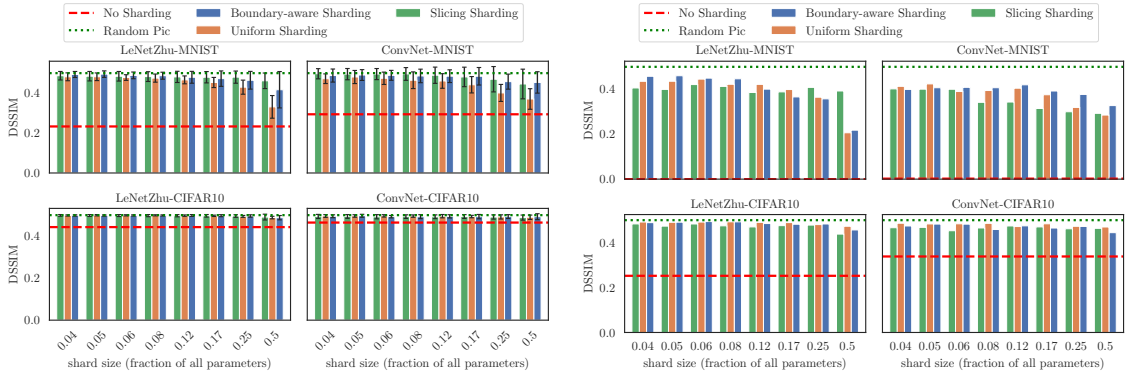
13.4.2 Decentralized vs. Centralized PS

In the first experiment we evaluate the privacy gain from sharding the model parameters in the decentralized parameter server setting, as compared to the centralized parameter server without sharding.

To that end, we used P2Sharding to partition the parameter space using the three proposed sharding techniques. Additionally, we varied the number of shards used to partition the model parameters in order to capture the effect of varying shard sizes ($size = \frac{1}{\#shards}$). For instance, a shard size of 0.5 refers to the fact that the model was split into 2 partitions. We then randomly selected a resulting partition and measured the DSSIM when executing reconstruction attacks across all datasets and model architectures.

The results of this experiment can be seen in Figure 13.6. All sharding techniques show a clear improvement over the centralized parameter server baseline (dashed red line). In particular, the benefit of sharding is most visible when looking at the success of the best attack (called worst-case scenario) in Figure 13.6b. Without sharding the centralized setting is consistently able to recover some private image. However, with P2Sharding we were able to find configurations (e.g., shard size = 0.12) that result in DSSIM values close to random noise even in the worst-case.

While Figure 13.6 illustrates that all partitioning strategies improve privacy clearly when compared to the baseline without sharding, it also reveals subtle differences among the different strategies, i.e., not all strategies get close to the upper bound of a random picture (green dotted line) for all settings. In the following, we analyze these difference in more detail.



(a) Average Performance

(b) Worst-case Performance

Figure 13.6: Evaluation of the proposed sharding techniques in terms of (a) the average success and (b) the success of the best attack (called worst-case scenario). *All sharding techniques show a clear improvement over the baseline (dashed red line).*

13.4.3 Effects of Model Sharding

`P2Sharding` provides configurations for different sharding strategies. Each sharding strategy determines which parameters are stored in the same shard. Since an attacker sees at least one shard of gradients, it is important to understand the privacy impact of different ways of sharding. In the following, we shed more light on when to use which sharding strategy.

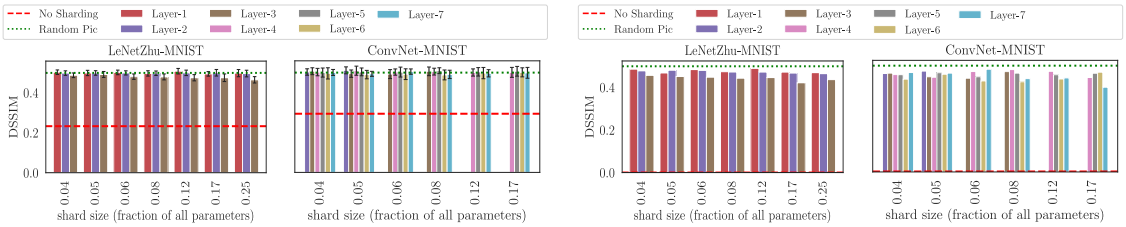
In Figure 13.6, we compared the three proposed sharding strategies across data and model characteristics. The first important observation is that the attack becomes consistently harder on more complex datasets (i.e., `CIFAR10` bottom row). This observation is in line with what was reported in previous work [69, 193]. Therefore, we mainly used the simpler `MNIST` dataset to differentiate the privacy impact of the sharding strategies.

In the case of the `MNIST` data (upper row) we make the following observations. For the simple `LeNetZhu` model, both the `uniform` and the `boundary-aware` strategy result in lower `DSSIM` values than the `slicing` sharding technique. This can be observed in terms of both the average (Figure 13.6a) and the worst-case performance (Figure 13.6b). With decreasing shard size (ie., higher number of shards) both strategies improve and eventually achieve similar or slightly better (in terms of the worst-case) privacy protection than the `slicing` strategy.

For the more complex `ConvNet` model, the opposite effect is noticeable: Initially the `uniform` and the `boundary-aware` strategy show a better (worst-case) performance

than the `slicing` strategy (see Figure 13.7b). Yet, with decreasing shard size again all strategies provide comparable performance.

We found that this effect can be explained by the boundary-awareness of the different strategies, as will be outlined next.



(a) Average Performance

(b) Worst-case Performance

Figure 13.7: Relative Importance of Layers. An attack based on the later layers of a model seem to have a higher chance for a successful attack (lower `DSSIM` value). This effect is most noticeable for simple data as well as models and when looking at the worst-case performance (Figure 13.7b).

Figure 13.7 reveals two key observations: First, limiting the gradients of a shard to only one layer yields (as in the `slicing` strategy) a relatively consistent performance across different shard sizes. Secondly, in the case of the `MNIST` dataset the last layer (`Layer-3`) enables more successful attacks (lower `DSSIM`) even for smaller shard sizes (e.g., 0.06). This is the case for both average as well as the worst-case performance shown in Figure 13.7b. For the `ConvNet` model, however, this effect is not as significant and is only partially observable in the worst-case performance.

This experiment highlights that different layers of a deep learning model can carry more information than others. Further, and even more importantly, limiting the information of a shard to one layer provides a more robust privacy protection compared to the `uniform` and `boundary-aware` strategy (Figure 13.6).

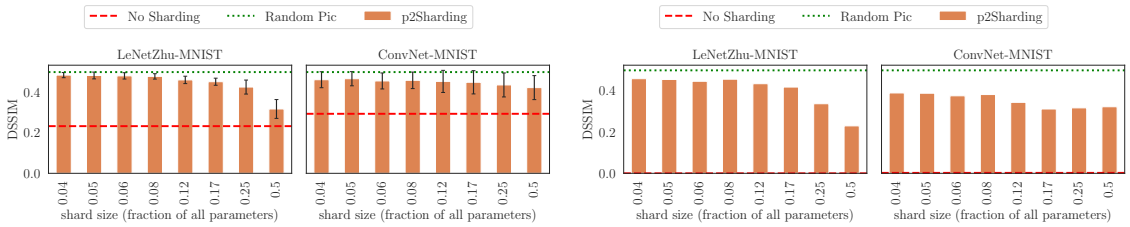
Yet, as shown next, the ability of `P2Sharding` to control the shard size helps to improve the privacy protection of sharding strategies.

13.4.3.1 Reducing Shard Sizes

The main idea of the `P2Sharding` framework is to partition the global model into several shards hosted on independent parameter server instances. Intuitively, with smaller shard sizes a corrupted parameter server learns less about the gradients of the model, which increases the resistance to data reconstruction.

To evaluate this concept, we used the `uniform` strategy to create shards of different sizes and measure the success of reconstruction attacks for different model complexities on the `MNIST` dataset. We used the `uniform` strategy, since out of the three proposed strategies it had the most sensitivity to changing shard sizes. Hence, using this setup we show that reducing the shard size is another effective measure to make the privacy for sharding more robust.

Figure 13.8 shows the result of this experiment for various shard sizes on the x-axis. We can see, that by reducing the shard size the privacy protection is improved significantly. Starting from a shard size below 0.12, `P2Sharding` is able to achieve a privacy protection that was otherwise only reached by `boundary-aware` sharding.



(a) Average Performance

(b) Worst case Performance

Figure 13.8: Influence of the shard size on privacy attacks (zoom-in into Figure 13.6). *Decreasing the shard size is an effective measure to prevent attacks. This is shown by the increasing* `DSSIM` *(less successful attack) for small shard sizes (e.g.,* 0.12*).*

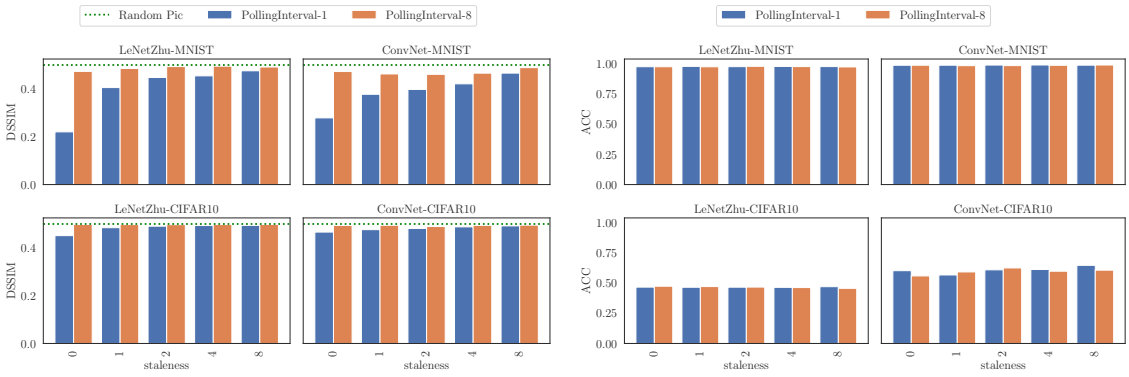
13.4.4 Effects of Asynchronous Updates

If `P2Sharding` is used for asynchronous updates the different parameter shards can be updated at different points in time. In general, with more clients concurrently training and updating the model shards, the more staleness on average one can observe, and the larger such variance across shards becomes [113].

In the experiment shown in Figure 13.9, we simulated the effect of concurrent activities and stale shards (i.e., delayed parameter updates) by randomly delaying the incorporation of an update in a shard. We studied the effects of staleness with no additional polling (i.e., polling interval = 1) and with a high polling interval.

In our setup, already with an average staleness per shard increased to 2, we observed that all data and models started to have increased resistance to reconstruction. That is, the average `DSSIM` (Figure 13.9a) increases. With more concurrency, such as when the average staleness reached 8, the privacy becomes close to the ideal privacy of random

noise, even for a simple model and dataset. This effect was even stronger when a polling interval of 8 was used in addition, which suggests that polling intervals contribute significantly towards an increased privacy protection. Hence, we will study the effect of polling intervals in more detail in the next section.



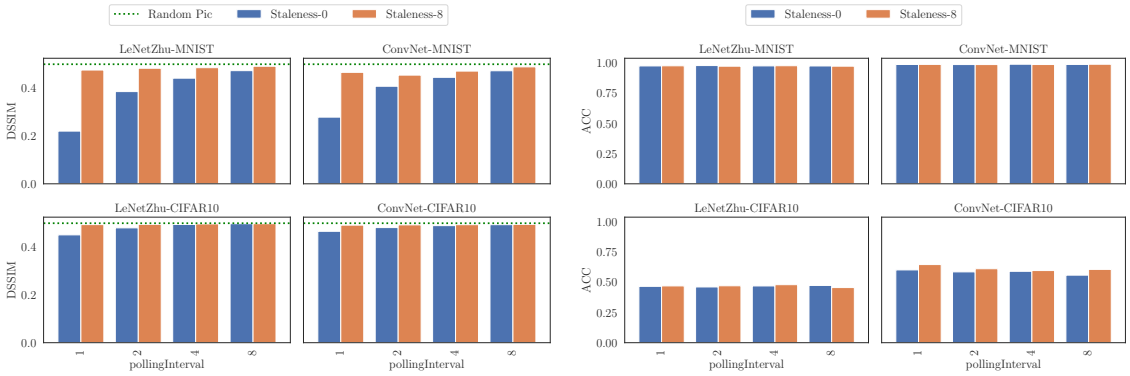
(a) Average Performance (b) Accuracy

Figure 13.9: Influence of staleness on privacy attacks. Staleness makes the model iterates inconsistent across all shards as in a typical asynchronous learning setup since the shards incorporate client updates with a delay. An increased staleness helps to improve the privacy protection, i.e., increase the `DSSIM` (13.9a) while not affecting the resulting model accuracy (13.9b).

13.4.5 Effects of Polling Intervals

Similar to asynchronous updates, polling intervals aim at preventing client-side attacker from uncovering the full gradient, a crucial step in reconstruction. However, polling intervals rely on serving parameters with randomly varying staleness to perturbe the attacker's view on the entire model. The added benefit is that polling intervals can work with synchronous updates to all the model shards such that the training becomes more stable.

In the experiment of Figure 13.10, we observed that with increasing polling intervals, the resistance to construction also became stronger. This effect was most obvious when the staleness due to asynchronous updates was controlled for and set to 0 (i.e., no staleness created by asynchronous updates, blue bar). We note that this setting is essentially polling interval combined with *synchronous updates*. With the only effect left due to polling intervals, we observed that a higher average interval at 8 consistently achieved higher resistance to attacks than without the polling interval (i.e., 1) across all data and model complexities (cf. Figure 13.10a).



(a) Average Performance (b) Accuracy

Figure 13.10: Influence of polling intervals on privacy attacks. Polling intervals enable `P2Sharding` to introduce asynchronous effects even to synchronous training. As for staleness, an increased polling interval improves the privacy protection, i.e., increases the `DSSIM` (13.10a) while not affecting the resulting model accuracy (13.10b).

On a final note, in both experiments of the asynchronous training and polling intervals we observed consistently high test accuracy as reported in Figure 13.9b and Figure 13.10b. This finding is consistent with previous work on asynchronous learning that showed that asynchronous updates achieve robust training quality [42, 113].

13.5 Related Work

Privacy Preserving Machine Learning. While Federated Machine Learning represents a recent technique to protect the privacy of training data, preserving the privacy in machine learning is a much broader area of research [1, 23, 124, 139, 140, 155]. In particular, [124] considers a non-federated setting where users upload secret shares of their data on two non-colluding servers. [155] uses a central parameter server to host the up-to-date model. It allows clients to train on the latest model while only sending selective gradients. However, it cannot prevent gradient-based attacks from colluding clients or a corrupt server.

Privacy in Federated ML. There have already been several existing approaches for enhancing privacy in Federated ML. In fact, cryptographic techniques such as differential privacy [119], homomorphic encryption [108] or secure multi-party computation [23] have also been proposed to improve the privacy in the context of federated learning.

However, as mentioned previously these techniques show several limitations such as limited compatibility with different model architectures or an increased learning overhead.

Recently, papers also explored non-cryptographic techniques to protect privacy in Federated ML, such as gradient compression [105, 196]. In this technique, gradients with small values are pruned to zero such that the number of useful gradients that are sent to the server are limited. While this technique also limits the amount of information available to an attacker, it depends heavily on whether gradients can be pruned or not and can influence the training process negatively.

Decentralized Architectures and Sharding. Moreover, there has been a lot of work in the context of decentralized machine learning [104, 132]. This works mainly focus on how to enable ML without using any central (parameter) server component. In contrast, our approach does not eliminate the parameter server, but decentralizes this component itself. The most similar approach with this regard is the work in [58]. However, compared to their architecture, which assumes a full replication of parameters across all server instances to make the overall training robust against potentially misbehaving parameter servers and thus to tolerate Byzantine failures, our approach utilizes sharding to distribute parameters across server instances to achieve privacy.

Lastly, sharding or partitioning in general has widely been used in both database systems [43] and ML systems [36, 42, 102, 183] to improve scalability and performance or reduce communication costs [192]. However, to the best of our knowledge, looking at sharding from a privacy perspective is a new proposal.

13.6 Conclusion & Future Work

The security of federated learning was recently called into question by works on gradient-based attacks to reconstruct private trainnig data. In this work, we initiated the study of secure FL based on a different, decentralized parameter server architecture called `P2Sharding`. We proposed three configurations on how to partition, serve and update the model parameters for better privacy. Empirical evidence on CIFAR10 and MNIST showed noticeably stronger resilience against gradient-based data reconstruction attacks by limiting the attack outcome close to random noise. In future work, we aim to further establish the formal security analysis of our FL framework.

Several areas for other future work exist. Our framework can be extended with differential privacy or cryptographic tools to further strengthen the security such as against malicious adversaries or support secure synchronous non-stochastic optimization.

Another work is to develop an automatic mechanism to optimally configure our framework given a wider range of models and datasets. Lastly our framework may also be extended for other security concerns in federated learning such as data and model poisoning.

13.7 Acknowledgements

This work was partially funded by the National Research Center ATHENE, the BMWK project SafeFBDC (01MK21002K), and the BMBF project TrustDBle (16KIS1267). We also thank hessian.AI for the support.

14 ACID-V: Towards a New Class of DBMSs for Data Sharing

Abstract

Recently, a new class of systems for shared and collaborative data management has gained more and more traction. Different from classical DBMSs, systems for shared data need to provide additional guarantees to ensure the integrity of data and transaction execution. In this paper, we propose to extend the ACID properties used by classical DBMSs with a new Verifiability component to enable users to specify the required guarantees of verifiability in a declarative manner.

Bibliographic Information

The content of this chapter was previously published in the peer-reviewed work Muhammad El-Hindi, Zheguang Zhao, and Carsten Binnig. "ACID-V: Towards a New Class of DBMSs for Data Sharing." In: *Heterogeneous Data Management, Polystores, and Analytics for Healthcare - VLDB Workshops, Poly 2021 and DMAH 2021, Virtual Event, August 20, 2021, Revised Selected Papers*. Ed. by El Kindi Rezig, Vijay Gadepally, Timothy G. Mattson, Michael Stonebraker, Tim Kraska, Fusheng Wang, Gang Luo, Jun Kong, and Alevtina Dubovitskaya. Vol. 12921. Lecture Notes in Computer Science. Springer, 2021, pp. 60–64. DOI: 10.1007/978-3-030-93663-1_5. URL: https://doi.org/10.1007/978-3-030-93663-1_5.

14.1 Introduction

Motivation. Recently, a new class of systems for shared and collaborative data management has gained more and more traction. Examples of such systems include Veritas [68], BlockchainDB [52], FalconDB [137], Fides [111] and Spitz [188]. Compared to classical DBMSs that are designed for being used by a single party, these systems enable multiple parties to manage a shared database (DB) in a collaborative manner. For example, think of a shared database for medical patient records. Here, hospitals and doctors would be able to directly share and modify patient data to keep track of diagnoses and treatments a patient received. Clearly, shared DBs provide many opportunities not only in the medical domain such as for large-scale epidemic studies [157], but also for many other fields where access to a shared DB enables more effective collaboration or new use cases (e.g., financial domain [163] or supply chains [90]).

However, different from classical DBMSs, systems for shared data need to provide additional guarantees to ensure the integrity of data and transaction execution (called *verifiability* guarantees in the following). The main reason for this is that when manipulating a shared database in a collaborative manner there is often some mutual distrust between the different parties that jointly access the shared database since they often have different interests (e.g., think of an insurance company and a hospital that use a shared database for medical records). Hence, the goal of the verifiability guarantees is to govern the shared database; i.e., to guarantee that the shared database is only modified based on a predefined and agreed upon set of transactions that every party adheres to and that none of the parties can tamper with the data in a different manner.

If we now look at how existing systems for shared data (such as those mentioned at the beginning) provide verifiability, we can observe that these systems typically take a very implementation-centric approach and often do not integrate well with the ACID guarantees of classical DBMSs. Moreover, the concrete guarantees that such systems provide are very different from system to system and often hard-baked into their execution model. FalconDB, for instance, is based on blockchains to implement verifiability and uses an incentive-based scheme where nodes are encouraged to verify the execution of queries asynchronously to hide the high verification cost. As a result, however, potentially unverified queries from malicious servers stay undetected. In contrast to that, updates are always verified synchronously for the entire network.

Vision. In this paper, we propose to take a more principled and more database-centric approach to provide verifiability for shared data systems. The main idea is to extend the ACID properties used by classical DBMSs with a new *Verifiability* component which

results in the *ACID-V* properties. To be more precise, similar to the other components in ACID such as the well-known isolation property, we propose to specify the guarantees of verifiability in a declarative manner using different verification levels (i.e., strict or more loose). Moreover, we believe that the integration of verification with the ACID properties not only is a natural fit and gives applications well-defined guarantees but it enables a new class of shared DBMSs that decide based on the verification level what optimizations and concrete execution strategies are best suited to meet the desired guarantees.

14.2 From ACID to ACID-V

14.2.1 Adding the V to ACID

In classical databases, transactions are governed by the ACID properties. As mentioned before, the concrete properties that should be satisfied can be defined declaratively and are implemented by databases in various ways. For example, for the I(solation) in ACID, a user can declare the specific isolation level (e.g., read committed, serializable) that a transaction should run under. This isolation level is then guaranteed by a database through its concurrency control scheme (e.g., optimistic vs. pessimistic). Similarly, we propose to add a new Verifiability property that user can specify declaratively and that database systems can implement in different ways. Further, looking at Verifiability from a conceptional perspective enables users to reason about the guarantees a system provides independent from implementation details.

To add the V to ACID, we extend the classical transaction state model of ACID-compliant DBMSs by a *verified* state. For simplicity, Figure 14.1 visualizes the extended state model for ACID-V for the case in which all nodes in a shared DBMS act honestly. We will briefly discuss some aspects of malicious behavior later in Section 14.2.3. As we can see, in our state model a transaction can only reach the *verified* state after it reached the *committed* state.

Modeling *verified* as a state that follows the *committed* state has several advantages. First, since verification is typically an expensive step the model leaves some freedom when the transition from *committed* to *verified* happens (i.e., directly after the commit or if it can be deferred). Moreover, it enables the user to declare which state is allowed to be read by other transactions (e.g., if committed but unverified can be read or if all state must be verified before becoming visible). Second, the *verified* state is an optional state as shown in Figure 14.1, i.e., not all *committed* transactions need to be verified, which allows partial verification to reduce the overhead.

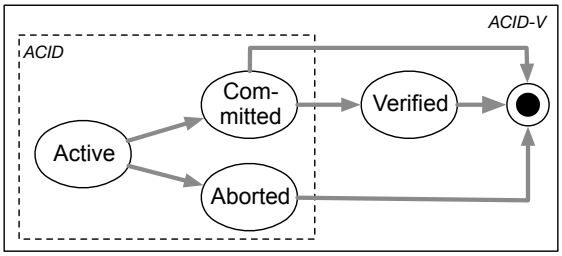


Figure 14.1: Simplified state model for ACID-V. The classical transaction state model is extended with a *Verified* state.

14.2.2 Verification Levels

While a formal definition of ACID-V and a more complete discussion of possible verification levels are out of scope for this paper, in the following we show how a first set of different verification levels can be defined based on the state model we introduced before. Based on this we will discuss what implications different levels can have on the integrity of data/execution and a system's performance.

Strict Verification (SV) This verification level requires that all transactions need to be verified. Moreover, all transactions are allowed to read only verified state. A similar guarantee can be provided by the online verification schemes of existing systems such as Veritas and BlockchainDB which guarantee that the result of a transaction (or database operation) is verified before becoming visible to other transactions. For the actual execution of transactions, this level implies that transactions should transition as fast as possible from the *committed* state to *verified* since otherwise (i.e., if there are too many committed but unverified transactions) this can lead to low performance or in worst case starvation. However, clearly strict verification thus has a high overhead and might lead to inferior performance when compared to more relaxed levels that we discuss below.

Unstrict Verification / full (UV-f) Compared to the previous level, this is a more relaxed verification level since it allows transactions to read from committed but not yet verified state. That is, even if the verification of a transaction is still pending, other transactions can access its committed state. However, all transactions are still being verified (hence it is called *full*) and unsuccessful verification in case of malicious behavior needs to be handled as we discuss below in Section 14.2.3. In contrast to the *SV* level, though, this makes room for different optimizations. Most importantly, transactions are not blocked by potentially expensive verification protocols since verification can be executed in batches and in a deferred manner. This is similar to deferred verification schemes that are available in existing systems (e.g., [188] or [52]). But still, verification

should not lag behind too much. This can be controlled by setting an additional parameter that specifies how many committed but unverified transactions are allowed.

Unstrict Verification / partial (UV-p) This verification level relaxes the guarantees of the previous level (UV-f) even further. In UV-f, transactions are allowed to access committed, but unverified state. However, unlike UV-f in *partial unstrict verification* (UV-p) we do not enforce that all transactions need to be verified. Consequently, this verification level assumes that *verified* is an optional state of a transaction. In this level, a user can thus explicitly request to verify only a subset of transactions. Hence, UV-p could be used to limit the verification overhead to some (e.g., important) transactions or to provide probabilistic guarantees by verifying only a sample of all transactions.

14.2.3 Handling Malicious Behavior

As mentioned before, in ACID-V it is important to take the effects of potentially malicious behavior of individual peers into account (i.e., in case they do not execute transactions in a correct manner). That is, if the verification fails for a particular transaction (e.g., due to incorrect execution by a malicious peer) all dependent subsequent transactions need to be rolled back in order to guarantee a correctly verified state of the database as specified in the verification level. For strict verification levels, this is less of a problem since no other transaction can read committed but unverified state from other transactions and hence only the effects of the transactions where the verification failed need to be reverted. However, for unstrict verification handling malicious behavior is more difficult since transactions can read from committed and not yet verified transactions and thus erroneous state can propagate across multiple dependent transactions.

14.3 Future Directions

In this paper we presented our vision for ACID-V compliant DBMSs to enable data sharing. As a core contribution, with ACID-V we propose to specify the guarantees of verifiability in a declarative manner and let the DBMS decide on what optimizations and concrete execution strategies are best suited to meet the guarantees of a particular verification level. In the future, we think that this model of ACID-V compliant DBMSs can trigger many follow-up work. First, the verification levels proposed in this paper are just an initial direction and we think that this requires a more profound discussion of what levels data sharing applications actually require. Second, similar to isolation levels that have triggered different implementation strategies (optimistic vs. pessimistic), we

think ACID-V will also enable a wide variety of different implementation strategies (e.g., beyond using blockchains) to implement the desired guarantees of verification.

14.4 Acknowledgments

This work partly grew out of discussions within and support of the National Research Center ATHENE, the BMWi project SafeFBDC (01MK21002K), and the BMBF project TrustDBle (16KIS1267).

15 Towards a Benchmark for Shared Databases

Abstract

Traditionally, data has been held in silos and was rarely shared with other organizations. However, recently data sharing across organizations is becoming more and more important as evidenced by governmental and industrial initiatives such as the EU data strategy. As a result, both academia and industry have been proposing new systems for shared databases, that allow multiple organizations to collaboratively insert and manage data in a common database. Yet, each new system seems to come with its own architectural choices and custom guarantees that make it hard for users to navigate the plethora of shared database systems. While standard benchmarks like the TPC-C database benchmark have been a well-established tool to compare and analyze traditional database systems, they seem to be unsuited to evaluate shared database systems. This is because these systems are built with fundamentally different assumptions in mind, such as a different threat/trust model since multiple (untrusted) parties access and modify the same data. In this chapter, we present a vision and initial ideas for a new benchmark to evaluate shared databases and capture their unique characteristics.

Bibliographic Information

The content of this chapter was previously published in the peer-reviewed work Muhammad El-Hindi, Ashwin Arora, Simon Karrer, and Carsten Binnig. "Towards a Benchmark for Shared Databases [Vision Paper]." In: *Datenbank-Spektrum* 22.3 (2022), pp. 227–239. DOI: 10.1007/s13222-022-00429-8. URL: https://doi.org/10.1007/s13222-022-00429-8.

15.1 Introduction

Traditionally, data has been held in silos and was rarely shared with other organizations. However recently, data sharing across organizations is becoming more and more important. Industry and governments alike are launching various initiatives to support and encourage data sharing [63, 122, 144] in different areas such as supply chain [32, 83, 138, 142], healthcare [78] or finance [37, 62]. At the same time, dealing with data is becoming more and more regulated by legislations such as the European Union (EU)'s GDPR or the California Consumer Privacy Act (CCPA) and Consumer Data Protection Act (CDPA) in the United States. Such regulations are even more important when data is shared and additional complexity is introduced in terms of governance and compliance [50].

While data sharing is seeing more and more adoption for different use cases and settings, the focus of this paper is on the setting of *shared* databases where the same DB is accessed by different parties (i.e., owners and consumers). In this paper, we differentiate between OLTP-style (collaborative) and OLAP-style shared databases. Figure 15.1 shows the setup for an OLTP-style (collaborative) shared database which is also in the focus of this paper. In this setting, two main characteristics are important: First, it involves that two or more organizations share ownership of the same database (i.e., multi-owner). Second, both organizations execute read and write operations on the shared database, i.e., the workload can rather be classified as an Online Transaction Processing (OLTP) workload. Such multi-owner scenarios have recently become more and more relevant since they enable a broad set of different use cases. For instance, medical data sharing where doctors and hospitals collaboratively work on the patients' data but also other use cases such as tracking information along a supply chain can be mapped to such a collaborative setting where multiple parties read/write to the same DB.

In contrast to the multi-owner (collaborate) setting, other data sharing scenarios with shared databases exist that target more Online Analytical Processing (OLAP) style sharing. In such scenarios, an owner provides DB-access to a consumer for read-only queries. In this setting, only the owning organization is allowed to update the data, while the other organization is limited to read workloads. This read-only data sharing mechanism was recently introduced, e.g., in Snowflake with its Secure Data Sharing feature. Common to both settings (i.e., OLTP- and OLAP-style data sharing), however, is the assumption that the partnering organizations do not fully trust each other, e.g., due to conflicts of interests or malicious behavior of potential inside attackers. Hence, additional technical measures must be provided to prevent or detect incorrect behavior of any sharing partner.

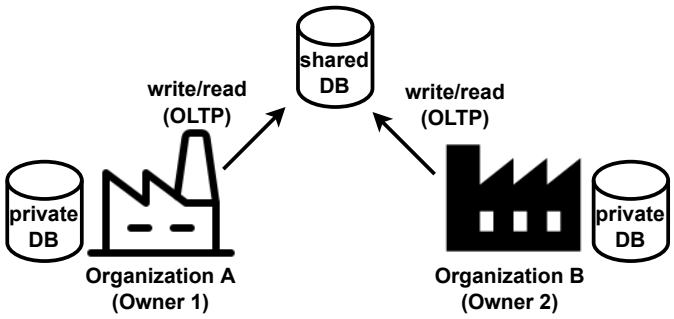


Figure 15.1: Shared database concept. Shared DBs enable multiple database owners to collaboratively write and read data to a common database — this workload can be classified as an OLTP workload.

With the advent of new technologies such as TEEs and distributed ledger technology (DLT) various systems using different architectures and approaches have been proposed to implement OLTP-style shared databases (e.g., [52, 68, 77, 137, 148]). However, different from established non-shared databases that provide well-known interfaces and guarantees, each new shared databases system seems to come with its own architectural choices and custom guarantees that make it hard for users to navigate the plethora of shared database systems. In this paper, we argue that traditional database benchmarks (like the TPC benchmarks) are not sufficient for analyzing the shared databases. Moreover, we present initial ideas for a new benchmark that helps us to better understand this wide space of different shared database systems and the impact of their architectural and technical choices on the system performance. While we think that traditional database benchmarks like TPC-C [168], Smallbank [5] or YCSB [38] are still a good starting point, we argue that these benchmarks do not cover all important dimensions for shared databases. For instance, while classical databases were built with an isolated single-owner setting in mind, shared databases assume a multi-owner setting. This observation is also evidenced by the common practice in several of the above-mentioned systems, that all implement (additional) custom benchmarks and evaluation frameworks. Naturally, this makes it hard to compare those systems with each other.

To address this issue we thus propose that a new standardized benchmark for evaluating shared database systems is needed. As a main contribution, we discuss an initial design for such a benchmark where we consider the unique characteristics of shared databases and address the challenges of designing systems for shared databases. To achieve this goal we provide the following contributions in this paper:

- We first analyze the unique characteristics of shared databases in contrast to traditional (non-shared) databases (Section 15.2).
- Based on that, we then discuss in detail the shortcomings of traditional benchmarks (Section 15.3).
- Finally, we propose our vision and initial ideas for such a new benchmark design for shared databases (Section 15.4). In this vision paper, we set the main focus on a benchmark design for shared OLTP databases while an extension towards shared OLAP databases is an interesting future avenue.

15.2 Shared Databases

In this section, we first analyze the unique characteristics and capabilities of shared database systems. Afterwards, in Section 15.3, we present why existing benchmarks are insufficient for evaluating shared databases.

15.2.1 Fundamental Paradigm Shifts

Shared databases for data sharing across organizations are based on fundamentally different assumptions than traditional data management solutions: First, obviously, the very traditional assumption that data is managed and accessed by a single organization is not true anymore. Instead, data sharing and shared databases involve multiple organizations in different roles. For instance, *data owners* write to and update data in a database, while *data consumers* only require read access. Further, external entities such as regulators or government institutions are often involved as *auditors* or overseeing participants that need access to meta-data and histories of the database. As such, in contrast to the traditional setup of classical data management, shared databases require that a different number of participants (e.g., multiple data owners) that do not necessarily trust each other have access to the same data. It is important to note that this should not be confused with multi-tenancy in traditional DBMSs which is about handling multiple isolated databases (i.e., access is restricted to a single organization).

Second, shared databases and cross-organizational sharing come with a new threat/trust model. This is necessary since DBMSs now not only need to manage the access of multiple users with potentially different roles, but they also need to manage the access of different legal entities (i.e., organizations). As shown in Figure 15.2, classical databases were built for use by or within a single trusted organization (left). Even with the introduction of

cloud computing and outsourced databases (middle), this fundamental assumption did not change. While organizations started to be concerned about the trustworthiness of the service provider (i.e., the organization operating the DB) and the correct operation of the database, they still assumed that their database is isolated from other organizations and will only be accessed by their own organization. In the context of shared databases, this assumption changed. Now, multiple organizations with potentially conflicting interests can access or even update the same database (cf. Figure 15.2 right). As a consequence, additional security and compliance guarantees became first-class citizens in shared DBs to make sure or at least be able to monitor that none of the involved organizations manipulate the shared database.

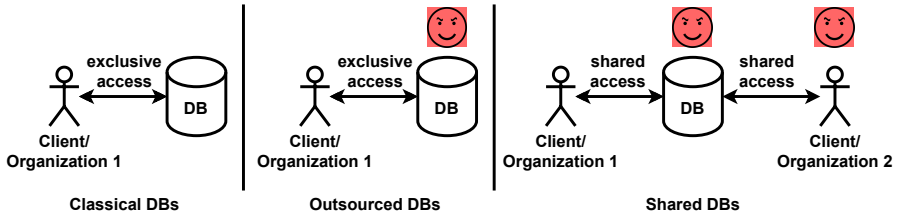


Figure 15.2: Shift in trust assumptions in data sharing. In the past, DBs were designed for use by a single, trusted organization (left). While the database or the database operator might still be untrusted as in the outsourced DB setting (middle), the additional concern of data sharing is to address trust issues among database owners (right).

These fundamental paradigm shifts motivated the development of new capabilities and abstractions that are essential in DBMSs for shared databases. Despite their importance, however, current benchmarks do not evaluate how well new systems cover these capabilities or what influence these capabilities have on a system's performance. In the following, we will first discuss the new capabilities of shared DBs in more detail and outline the shortcomings of existing benchmarks in the next section.

15.2.2 New Capabilities

In summary, shared databases come with four new abstractions and capabilities that are essential to allow multiple organizations to collaborate on data: *shared tables*, *verifiability*, *data usage & compliance* as well as *auditability*.

Thereby, shared tables provide a new abstraction for organizations to access shared data in a relational DBMS. Verifiability and data usage & compliance allow monitoring and controlling of how other organizations interact with the shared data. Lastly, auditability

enables an external organization to check if all involved parties have been behaving correctly. We will describe each new capability in the following.

Shared Tables. In the context of relational databases, *shared tables* are a common abstraction to allow the user to access both private and shared data transparently, while still taking the special characteristic of shared data into account. Different from private data, shared data can be maintained (i.e., added and updated) by multiple organizations. For instance, organization A can write orders for a product to a shared orders table that is also accessible to organization B, which will fulfill the order. Thereby, organization A accesses the shared orders table in the same way it would access any other private table. At the same time, the same instance of this table is also accessible to organization B, which can also add and update records to the shared orders table.

Shared tables as such are only an abstraction and do not demand a specific implementation. For example, one DBMS might choose to implement the shared table abstraction in a centralized manner, while another system might actually maintain two copies of the table at different locations and keep the copies consistent. Yet, these different implementation strategies might come with various performance implications that are necessary to account for when evaluating a database system. Further, to make sure that no participant manipulates or processes the data incorrectly, shared tables are also backed by additional data structures and algorithms that are used to implement the other capabilities of shared DBs (i.e., variability, compliance, auditability). However, these additional security measures come with additional overheads that current database benchmarks do not cover sufficiently.

Verifiability. Shared tables are only meaningful for organizations if they truly provide a trustworthy and consistent view of the data across all organizations. In a setting with highly fluctuating prices, for instance, it is critical for an organization A to prove that it placed an order before B changed the price for the ordered product. Similarly, it should not be possible for an organization to tamper with the data without being noticed by the other parties.

To achieve this trustworthy and consistent view, Allen et. al. introduce the abstraction of *shared verifiable tables* [68]. Verifiability refers to the ability of the system to provide proofs to any of the involved organizations about the state and the behavior of the database. In the above examples, for instance, a shared verifiable table would enable organization A to prove that it placed the order with the latest price. At the same time, organization B will not be able to prove that its claimed price existed at any point in time in the shared table.

These examples show the need for shared verifiable tables to enable organizations to check the integrity of the data (i.e., that no organization can tamper with the data without the other noticing it). However, the concept of verifiability is not limited to data integrity. Rather, verifiability is also critical for computation or query/transaction execution (integrity of execution). For example, while organization B might execute a transaction (TX) to increase the price of a product by 10%, organization A might change the logic of the transaction to decrease the price instead. Similar manipulations are also possible in the case of read-only queries that could, e.g., be manipulated to only return incomplete or false results and thus lead an organization that consumes data from another one to wrong decisions. To address these situations, verifiability provides a mechanism to prove that a transaction or query was executed correctly and resulted in the expected outcome.

Last but not least, there are also non-functional execution aspects that can be subject to verification. For instance, in [181] the authors show how the adherence of a system to isolation levels, in particular serializable, can be verified. Another important non-functional aspect is if a system conforms with policies and regulations for data sharing. For example, when the deletion of a record is requested, e.g., in the context of regulations such as GDPR, a shared database needs to prove that all instances of a record have been deleted. This is especially challenging in a shared database setting since the database is controlled by multiple organizations.

To support verifiability, shared database systems implement several new routines (e.g., proof generation) and provide new interfaces (e.g., a new `verify()` interface) that allow participants to make use of the verifiability capability. Further, as we will discuss later, shared DB systems might use different verification strategies with different performance characteristics. We believe that a new benchmark for shared DBs is required to take the new interface(s) into account and be able to assess the different verification strategies properly.

Data Usage & Compliance. In traditional databases RBAC is used to control access to data. Data sharing systems and shared DBs, however, are not only concerned about access, but also about usage. This means it is not only sufficient to limit who can access which data. In addition, we need to control how the entities accessing the data are allowed to use the data. For instance, in RBAC we can only specify that a certain user has read-access to a single column *age*, but we cannot control which queries the user executes on the data, e.g., only allowing aggregates queries that include a certain minimum population.

Furthermore, there are also other factors that are important for organizations that are related to data usage control A major dimension is the ability of systems to enable users to specify data usage requirements regarding compliance. For example, certain use cases might require that data is not allowed to leave the premises of an organization, which is a common requirement in the financial industry. In this case, the DBMS must store data locally and is not allowed to, e.g., replicate the data to another organization or location.

Supporting such more fine-grained usage controls clearly comes with an additional cost that needs to be evaluated when benchmarking shared databases.

Auditability. As mentioned earlier, besides data owners that collaborate on a shared database, sometimes additional external organizations, e.g. auditors, require occasional or recurring access to the meta-data and the history of the database. However, since such external entities are not regular users of the system, they do not have access to the shared state or the transaction history for example. Hence, different from verifiability, auditability describes the capability of the system to allow external organizations to efficiently check the correct state and behavior of the system. Further, while verifiability focuses on the timely validation of interactions and queries, auditability has a more retrospective view. Hence, for auditability, it is required to keep the previous state around to enable external parties to audit actions that happen in the past. Moreover, in contrast to verification which needs to be fast and efficient, auditing can be a resource-intensive task that is executed out-of-band from normal database operations.

Similar to verifiability, auditability comes with additional overheads and interfaces that were not required in traditional database systems. Hence, as will discuss in the next section, current benchmarks do not sufficiently evaluate the effects of these aspects on the system performance.

15.3 Shortcomings of Existing Benchmarks

Traditional benchmarks for data management systems test a DBMS end-to-end from the perspective of an actual end-user. However, these benchmarks are designed with the classical assumptions in mind that there is only one organization accessing the data. Therefore, traditional benchmarks are unsuitable for evaluating shared database systems.

Moreover, also more recent benchmark proposals like LEDGERBENCH [189] that target data sharing systems, do not consider all data sharing requirements. For example, while aspects to cover verification and auditing are included, these benchmarks lack important benchmark dimensions, such as the number of participants, that are required

to evaluate systems holistically. Moreover, the workloads are often rather simplistic and do not really reflect the complexity we see in many real-world scenarios. In the following, we will discuss these limitations of existing benchmarks with regard to shared databases in more detail. As mentioned before, in this vision paper we set the main focus on a benchmark design for shared OLTP databases while an extension towards shared OLAP databases is an interesting future avenue.

15.3.1 New Workload Requirements

Classical benchmarks were built with the premise of a single database user and a single workload in mind. As such, they only define a main application workload from the perspective of a single organization that users execute, e.g., either an OLAP or OTLP workload. In contrast to that, data sharing involves multiple different organizations and roles that require a system to handle different types of workloads. As such, there is a need to adapt the benchmark workloads accordingly. First, in contrast to traditional benchmarks, the application workload of the benchmark needs to include shared tables as well as transactions accessing these shared tables. Moreover, workloads of data sharing systems should test other aspects than traditional workloads. For example, since the overhead of data sharing typically increases if more organizations are involved in data sharing, testing the scalability with the number of participants is an important aspect in addition to testing the scalability with the size of data which traditional benchmarks typically focus on as we discuss next. Finally, in addition to the application workload, other aspects such as verifiability and auditability must be considered with new dedicated workloads. For example, external auditors must also be considered as a special form of clients that can request more intensive proof generation and checking.

15.3.2 Other Forms of Scalability

Data and workload scalability characterize the behavior of a system when the amount of data or the number of requests are increased. All of the major database benchmarks consider these aspects. However, data sharing additionally introduces *participant scalability* that describes the behavior of a system under test (SUT) when the number of data sharing participants varies. Due to the dynamic nature of data sharing, some tables might be shared with only a few partners while other tables might have many participating organizations. Depending on the use case, the fluctuation of partners can be high or rather low, leading to shorter or longer-lasting relationships.

An interesting observation in this context is that due to different possible architectures of shared DB systems, adding a new data sharing partner might involve different aspects depending on the system. For instance, in a centralized shared database adding a new organization might simply involve granting access to a new account of the platform. However, for some systems (e.g., decentralized systems) adding a participant might require spinning up a new node and replicating the entire data to that node. To the best of our knowledge, none of the existing data management benchmarks (including LEDGERBENCH) takes this aspect of data sharing into account.

15.3.3 Private & Shared Data

As discussed before, a new benchmark for shared databases needs to include shared tables as well as transactions accessing these shared tables. This means that when defining the data model, we must distinguish between data that is shared with other organizations and private data that will not be exposed to others. Such a distinction is important since a system usually needs to build additional data structures for the shared tables, e.g., to be able to guarantee data integrity. Similarly, querying or executing transactions on this data might involve more computational overhead to prove the validity of a transaction or check data usage controls, for instance. We refer to such transactions touching shared data as *shared transactions*. Due to the additional overhead that is involved in the execution of shared TXs compared to private transactions, we believe that the amount of executed shared transactions is another important dimension that is currently not considered in existing benchmarks.

15.3.4 New Transaction Model

As mentioned previously, shared transactions incur more overhead during transaction processing. Among other reasons, this is mainly because of the additional verifiability requirement (cf. Section 15.2.2) that allows data sharing participants to check whether a system executed a given transaction correctly. As discussed in previous work [54], several data sharing systems incorporate such verification checks in their transaction model and require that all or a majority of participants (i.e., data owners) agree on the outcome of a shared transaction before a transaction is committed to the shared DB.

This is achieved using some form of conensus protocol such as Raft[130] or PBFT[31]. In the Veritas system, for example, a transaction is only committed once all nodes approve it. This is done by shipping transaction log records periodically to all other Veritas nodes. The other nodes then apply the log and verify the to-be-committed

transactions. Afterward, they broadcast their votes to all other nodes and commit or abort the transaction based on the outcome of a Cesar consensus [68].

Important to note is that existing shared DB systems differ in how they incorporate verifiability in their transaction model. In general, we can distinguish two approaches: *Online verification* follows a synchronous approach, in which a shared transaction can only be committed to the shared database if the verification of the transaction succeeds (e.g., the majority of participants vote to commit). *Offline verification* (or deferred verification) is an asynchronous approach in which a transaction can be committed similar to the traditional execution model without verification. Yet, the systems allow participants to trigger the verification process after some delay to verify multiple TXs at once and amortize the verification overhead. Moreover, some system like FalconDB [137] even implement different models for write and read transactions. That is, while write transactions that update the state of the shared DB are verified synchronously, FalconDB employs offline verification for read-only queries.

While verification thus plays a crucial role in shared databases, existing benchmarks are not able to determine the overhead involved in verification. This is because traditional benchmarks only measure the time it takes to commit a transaction and are not aware of additional verification steps that potentially need to be triggered asynchronously. To address this issue and be able to compare the performance and cost of verification in detail we propose to include an additional workload category (i.e., a verification workload) in our benchmark. This workload category analyses the overhead of verification under different setups (i.e., online vs. offline), as will be explained in the next section.

15.4 A New Benchmark Design

To address the unique characteristics and capabilities of shared databases discussed before, we envision a new standardized benchmark that evaluates shared database systems end-to-end. In the following, we will first give an overview of our proposed benchmark design before we present initial concrete ideas on how to realize such a benchmark.

15.4.1 Overview

As we discussed previously, shared databases come with new capabilities (cf. Section 15.2.2) that are essential to support the different participants (i.e., data owner, auditor) of the system. In order to include these aspects in our benchmark design, we suggest introducing three different workload categories (application, verification, and

auditing) as shown in Figure 15.3. While the application workload models the core (OLTP-)workload of organizations involved in data sharing, the verification and auditing workload focus on more specific aspects: The verification workload aims to reveal the overhead of different verification schemes implemented in data sharing systems (e.g., online vs. offline verification). The auditing workload models the access patterns of an auditor which is much more read-heavy and scan oriented since it needs to go over a long history of data updates to see where potential issues (e.g., illegal data modifications) occurred. Overall, we envision that a benchmark execution has to include the application workload, while the other two categories are optional. This (modular) approach of workload categories enables the usage of the benchmark for systems that do not offer a certain capability (e.g., auditing).

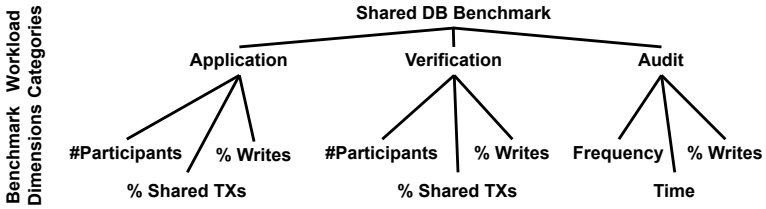


Figure 15.3: High-level benchmark design. Our benchmark defines three main workload types (Application, Verification, Audit) with novel benchmark dimensions.

15.4.2 Systems Under Test

Besides studying the effects of the paradigm shifts, our new benchmark allows us to analyze different existing architectures for shared databases that come with very different overheads. To reveal these overheads, our benchmark defines three important dimensions (i.e., the number of participants, the fraction of shared transactions, as well as the read/write ratio) that we evaluate for each workload category as shown in Figure 15.3. As we discuss later, these dimensions have a significant impact on the performance of a data sharing system. For example, the number of participants can have a severe impact on the overall system performance in terms of throughput and latency). To make the importance of our dimensions more clear, in the following sections we consider two architecture stereotypes for the systems under test (i.e., the shared database systems) depicted in Figure 15.4 as examples[1]:

[1]This is not meant to be an exhaustive list. In fact, the principled analysis of possible architectures for shared databases is an interesting area for future work.

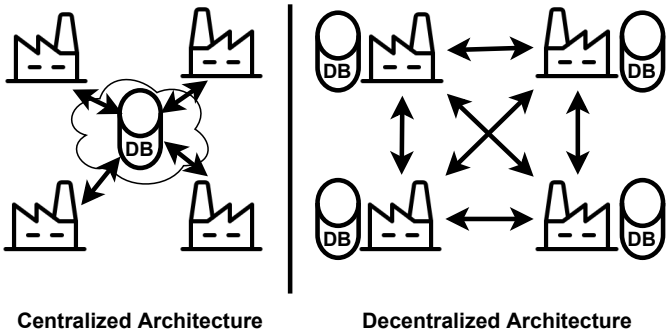


Figure 15.4: Architecture stereotypes used as examples. In the centralized architecture (left) all participants access the same shared DB platform. The decentralized approach (right) removes the need for a central platform. Instead, every organization operates its own DBMS node and additional protocol are in place to keep all nodes in sync. Depending on the chosen architecture we expect systems to exhibit different behaviors in the benchmark.

Centralized Architecture. In this architecture, multiple organizations access the same shared database platform. The central DBMS usually provides additional verifiability and auditability guarantees to establish trust in the platform and among the participants. Examples for such an architecture are Microsoft's SQL Ledger [11] or Alibaba's LedgerDB [184].

Decentralized Architecture. This architecture eliminates the requirement for a central entity that provides the shared database platform. Instead, every participating organization is in charge of operating its own DBMS node that stores a copy of the shared data as is the case in many of the proposed hybrid-blockchain-database systems. Verifiability and auditability guarantees are provided via additional protocols, such as consensus protocols. Recent examples for this architecture are systems like Veritas [68] or FalconDB [137].

Both of these architectures might show differences and commonalities when evaluated with a specialized benchmark for shared DBs. Lastly, note that due to the fundamental shift in the trust model, shared databases come with new security and compliance challenges (e.g., in the context of data usage). However, "benchmarking" security is known to be hard or even impossible [128]. Hence, in the following, we will focus on evaluating the performance characteristic and regard defining suitable security evaluation frameworks for shared DBs as an important area of future work.

15.4.3 Workload & Data Definition

As mentioned before, our benchmark design proposes three workload categories to evaluate the performance of a shared database; i.e., the system under test. The categories address the different capabilities of shared database systems and include *application*, *verification* and *audit workloads*. In the following, we will first discuss the application workload — the mandatory part of our benchmark. For the application workload, we will also discuss how the novel benchmark dimensions (e.g., participant scalability) can help to analyze shared DB systems. After that, we will focus on the main characteristics of the remaining two workload categories. However, for these workloads, we will provide fewer details.

15.4.3.1 Application Workloads

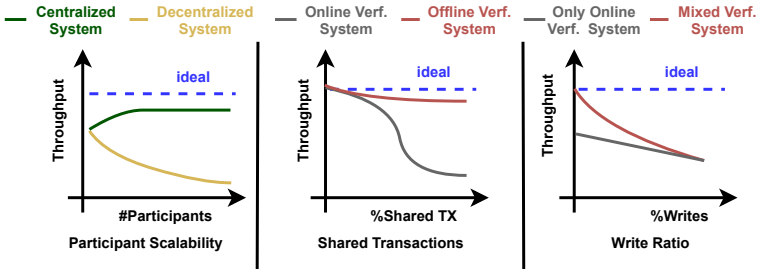


Figure 15.5: Application workload to measure the end-to-end system performance. We propose to use the TPC-C benchmark as starting point and re-use its performance metrics (e.g., throughput). However, we introduce new benchmark dimensions (Participant Scalability, Shared Transactions, Write Ratio) to help uncover differences in the performance of different shared DB systems.

Application workloads usually model a real-world use case and corresponding database queries that are executed by clients (also called terminals). Since we focus on OLTP-style workloads for shared databases, we believe that a classical benchmark such as TPC-C and its workload patterns (i.e., the transaction mix and data access patterns) is actually a good starting point[2]. However, it can clearly not be used out of the box without any modifications for data sharing since the TPC-C benchmark models the activities of one organization only — a wholesale supplier, who accepts product orders at and for different warehouses. In the following, we thus explain how the application workload of TPC-C can be adapted to model multiple organizations as well as which metrics we aim to report.

[2]We use TPC-C as a concrete example in this paper, but we envision that other benchmarks can and will be used similarly.

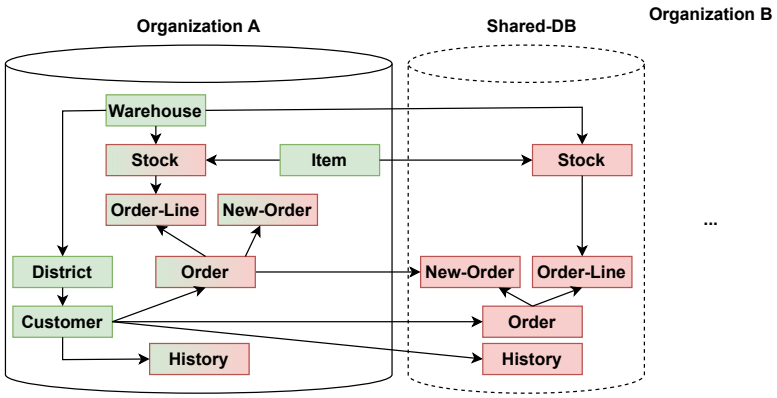


Figure 15.6: Adapted TPC-C data model. In our data model, it is assumed that every warehouse belongs to a different organization. Therefore, we modify the TPC-C data model to distinguish between private data (green) and shared data (red). To support private-only transactions, i.e. orders that do not include remote order-lines, every organization has a private partition of the initially shared tables (green/red).

Data Model and Workload Mix. In order to model an OLTP-workload for shared databases, we need to define multiple data owners in the TPC-C workload. We think that this comes naturally for TPC-C since the data model is already partitioned by warehouse. As such, to model different owners and be able to scale participants at the same time, we assign each warehouse to a different data owner (i.e., each warehouse belongs to a different organization). To reflect this change in the data model, we suggest extending the TPC-C data model to use private and shared tables as shown in Figure 15.6.

The figure illustrates that we adopt the classical TPC-C data model and additionally partition the data into shared and private tables (i.e., data). For example, the four green relations in Organization A's database (`warehouse, district, customer, item`) are only written to by Organization A — they represent private data. To support certain transaction types (e.g., `new-order` that queries a price of an item) read-access on specific columns and rows can be allowed for other organizations. However, the five red relations (`stock, order-line, new-order, order` and `history`) are written to by multiple owners to support the concept of remote order-lines in TPC-C. A remote order-line is an item that is supplied by a different warehouse, resulting in, e.g., a corresponding stock update for that other warehouse. To support this stock update in a remote warehouse (of another organization), we define a shared `stock` table that contains the stock information of any shared item. This corresponds to a partitioning of the `stock` table to private stock-information for items that are never ordered by other

warehouses (represented by the green/red `stock` table in organization A's DB) and shared stock-information for items that can be part of a remote-order line (represented by the red `stock` table in the shared DB).

To fully support, e.g., the `new-order` transaction profile, we similarly partition the other involved tables (`order-line, new-order, order`) into a private and shared table. Transactions that do not include any remote order-lines can be simply executed using only private tables. Any `new-order` transaction, however, that contains at least one remote order-line will be executed as a shared transaction. This involves writing to the shared `new-order, order` and `order-line` tables. Note that slight modifications to the transaction profiles are required for certain TXs (e.g., the payment TX) to avoid access to private information in the `warehouse` table for example. However, a detailed discussion of necessary changes to the transaction profiles is out of scope for this paper.

In addition to the data model, we need to specify a workload mix to include such shared transactions. Here, we again propose to rely on the TPC-C transaction mix with its five transaction types and the ratio of remote order-lines that is defined in the benchmark. However, as discussed later, the main difference is that we propose to change the ratio deliberately to vary the fraction of shared TXs in a workload mix as one important dimension of our benchmark.

Performance Metrics. We suggest reporting the classical performance metrics like latency and throughput as benchmark metrics. As mentioned earlier, we decided not to attempt to include the level of security or trust that a system provides as a measurable metric. The reason for this is that it is inherently hard to measure "security" or trust as discussed in previous work [128] since these concepts require measuring the effect of potentially *unknown* attacks. However, security-relevant system parameters like the used verification strategy or certain data usage policies can impact the end-to-end performance. Hence, we suggest reporting such security- and trust-related properties as part of the benchmark report. In the future, this might also enable a classification-based evaluation of a system's security/trust as suggested in [128].

In the following, we now discuss in more depth the new benchmark dimensions (participant scalability, fraction of shared transactions, and read/write ratio). We propose that these dimensions are varied in our benchmark to reveal the performance characteristics of a shared database. In the following, we discuss the dimensions for the application workload but the same dimensions can also be varied for the verification and auditing workload.

Participant Scalability. Scaling the number of participants in a shared DB (Figure 15.5 left) is different from simply adding additional database clients (terminals). This is because the number of participants can be scaled independently from the number of clients that generate the transactions. As described earlier, in TPC-C we can model each warehouse as an independent organization. Adding more participants would then mean increasing the number of warehouses. Note that, although it is the case for TPC-C, increasing the number of participants does not necessarily mean that the amount of data needs to be scaled.

Moreover, in contrast to scaling the number of clients, increasing the number of participants (i.e., data owners) can actually have a negative performance effect on some systems as shown in Figure 15.5 (left). The figure shows that in the case of a centralized architecture (green line) the system might first benefit from adding participants since they, e.g., might first improve the utilization of the platform. In contrast to this, joining a new organization in a decentralized system like Veritas [68], for instance, requires adding a new node to the network which has been shown to reduce the throughput of the system significantly [68] (yellow line).

Fraction of Shared Transactions. As discussed previously, we can adapt the data model of the TPC-C workload to easily incorporate shared data and transactions in the workload. More precisely, we can model, e.g., `new-order` transactions that involve remote order-lines as shared transactions since they access the shared tables `new-order, order, order-line` and `stock`.

The goal of the new *shared transactions* dimension is to investigate the performance overhead that a system incurs when more and more shared transactions are executed. In our TPC-C based benchmark, we can control this by gradually increasing the ratio of remote order-lines in a `new-order` transaction. As shown in Figure 15.5 (middle), when we do not include any shared transactions (i.e., 0% shared TXs), a shared database system should reach the performance of a traditional database, in the best case. Yet, with an increasing amount of shared TXs, the performance might vary depending on the design choices of a system. For example, in a system with an online verification scheme, a performance drop will be observable as soon as shared transactions dominate transaction execution (due to high verification costs). This is represented by the gray line in Figure 15.5 (middle). In contrast to that, a system using an offline verification scheme (red line), will first show a performance drop but later stabilize. This is because the costly verification runs asynchronously after transaction commit and hence does not affect the commit throughput which is measured by application workloads.

Read/Write Ratio. With the last benchmark dimension, we plan to uncover differences in how systems handle verification for reads and writes. To do that, in the case of TPC-C, for example, we propose to change the workload mix to vary the ratio of writes in the workload. That is, the classical TPC-C benchmark defines a fixed transaction mix with mostly write-heavy transactions (44.5% `new-order` and 43.1% `payment`). For the write-ratio benchmark dimension, we modify the transaction mix to gradually increase the ratio of write-heavy transactions.

As shown in Figure 15.5 (right), in a system that uses online verification for both read and write-heavy transactions, we expect to see a verification overhead even with a low write-ratio. With an increasing write-ratio, however, the performance might be further reduced due to a potentially more costly verification of writes. In a system that uses a mixed verification scheme (e.g. online verification for writes, offline verification for reads), we can expect that the system performance is initially high because of the low write-ratio. However, as soon as the write-ratio increases performance will drop significantly due to the high verification cost of writes.

15.4.3.2 Verification Workloads

As mentioned earlier, traditional database benchmarks only measure the throughput of a system until the commit/abort of a transaction and ignore any further verification steps that can run asynchronously. As a consequence, when we look at the performance of systems with an offline/deferred verification scheme in the application workloads category, we will see that those systems usually provide noticeably better performance than online verification systems. The reason for this is that online verification schemes verify transactions before the commit of a transaction, while offline verification schemes can commit without waiting for the outcome of the asynchronously triggered verification.

To address this issue and shed light on the verification performance of such systems, we propose including additional verification workloads when benchmarking shared databases. Verification workloads take the new transaction model of shared databases into account and use the verification interfaces of these systems to measure both the TX-execution and TX-verification performance for a given transaction (type). Thereby, the same transaction types of the application workloads can be used.

For instance, for TPC-C, we envision that the same workload mix as in the application workload category is executed. However, to measure the verification performance, we assume that the benchmark client is extended to use the additional `verify()` interfaces that shared database systems provide. That is, the benchmark runner not only sends the

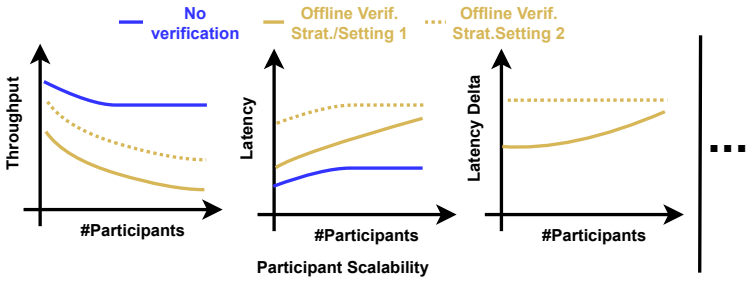


Figure 15.7: Verification workloads measure verification effects that are not visible in application workloads. For example, they help to detect differences in the used verification strategies. While the workloads in this category re-use traditional performance metrics (throughput/latency), we make use of our previously described benchmark dimensions (e.g., participant scalability) to uncover performance differences of different systems and verification strategies.

transaction to the database but additionally calls the `verify()` interface to retrieve the verification result from the system. As shown by the multiple yellow lines in Figure 15.7, the benchmark includes repeating this measurement for multiple verification settings or strategies depending on the SUT (e.g., different batching/delay settings). To reveal this overhead, we propose to also measure the performance without calling the `verify()` interface (blue line in Figure 15.7).

Depending on the given system, the performance of TX-execution might be significantly affected by TX-verification (e.g., if both processes contend on the same data structures) and other factors (e.g., the number of participants). Therefore, we suggest using the previous benchmark dimensions in this workload category, as exemplified by the *participant scalability* dimension in Figure 15.7.

Note that analyzing the throughput and latency of verification is important since many offline verification schemes employ batching to amortize the verification overhead. While this can improve the throughput, it can also deteriorate the latency. In fact, in some shared database systems, the transaction latency increases significantly from milliseconds to seconds when verification is involved. Because of such effects, we thus propose to explicitly consider the latency (in addition to throughput) as a metric. This additional metric can help to reveal the difference between verification and the commit latency (called latency delta). This metric can be easily computed from the measured latencies and is a practical way to visualize whether verification or commit latency is affected more severely by, e.g., an increasing number of data owners.

15.4.3.3 Audit Workloads

Audit workloads are designed to assess the auditability capability of a shared database. Recap that auditability allows an external auditor to check the correct behavior of the shared DB system. Compared to verification, auditing can be resource-intensive and require checking long histories or more complicated proofs. Hence, some systems assume a dedicated auditor component exists to perform this task. This component is usually not involved in regular TX-processing and hence does not have any previous state information. Therefore, the auditing process involves retrieving previous TX log entries from the system and applying them to verify the data integrity and correct execution of transactions.

To implement this procedure, we again envision extending the benchmark runner to encompass or mimic the role of the audit component. Systmes for shared DBs, e.g., [184] or [188], ususally do not offer dedicated `audit()` interfaces. Instead, they provide interfaces to retrieve a verifiable part of the system's transaction log that an auditor can replay to compare it with the claimed state of the database. Some systems, e.g., [11] also allow the use of the `verify()` interface for auditing purposes, i.e., verifying historical state and data.

Traditional auditing-related benchmarks or workloads focus on the performance of the auditing process itself, e.g., the auditing latency on the auditor component. While this is an important aspect, we propose to focus on how auditing affects the performance of the shared DB system.

To this end, we envision the following benchmarks in the audit workload category (cf. Figure 15.8).

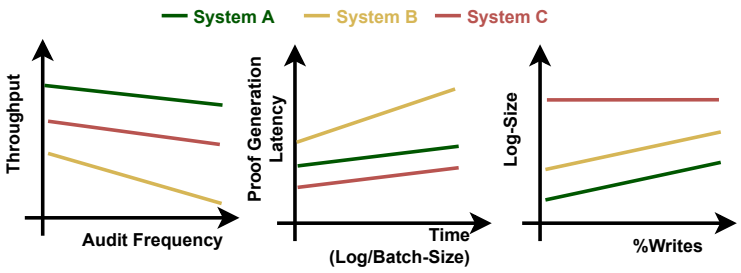


Figure 15.8: Instead of measuring the audit performance of the auditor, we propose to analyze the effect of auditing on the system performance. Audit workloads consider different metrics from the perspective of the shared DB, e.g., how transaction throughput is affected by an increasing audit frequency (left) or how proof generation latency (middle) or log-size (right) are influenced.

Audit Frequency. As shown in the first plot of Figure 15.8, we suggest measuring the system throughput (e.g., using an application workload from before) while changing the frequency of audit requests. In the case of TPC-C, for example, we propose running the standard workload mix and initiating additional audit operations in parallel. However, we do not necessarily focus on the audit performance as suggested by other benchmarks. Instead, this benchmark aims to investigate how auditing operations affect system performance. We expect an influence on most systems' performance because auditing involves requesting proofs from the DBMS, which has to generate proofs while serving regular database transactions. These proofs can have different forms ranging from historical transaction logs to more concise cryptographic proofs. Depending on the proof generation overhead, we expect some systems to show a more significant throughput degradation than others, as exemplified by System B (yellow line).

We believe this dimension also has practical implications for determining when and how often to schedule audits. For example, in a system with a high impact on performance, the benchmark helps to determine that it is best to schedule audits less frequently, e.g., only during the night, to avoid performance degradation.

Proof Generation. The aforementioned proof generation overhead is analyzed in more detail by a dedicated experiment. We suggest measuring the proof generation latency, i.e., the time an auditor waits to get the requested proof back, for a varying time duration. A longer time duration corresponds to a longer history which has to be audited. However, often longer histories involve longer proofs, which cause an increase in proof generation latency of the systems (cf. middle plot in Figure 15.8). For example, compared to classical verification operations, proof generation can take up to several hours [137]. We believe that discovering and understanding these overheads can help engineers to determine bottlenecks and optimize the proof generation in a system.

Log Size. While it is possible to investigate the influence of all previously suggested benchmark dimensions, we believe that the most relevant dimension for the audit workloads is the write ratio. As previously, the reason is that the write ratio has a direct impact on the number of log entries that are written. However, some systems also log read operations to, e.g., be able to audit data usage controls. To be able to capture such differences, we propose the experiment that is sketched in the left plot of Figure 15.8.

The plot measures the log size after running a workload for a fixed duration while varying the percentage of executed write transactions. As before, in the case of TPC-C, this can be achieved by varying the workload mix to include more or less write-heavy transactions. Figure 15.8 (right) visualizes that System C logs both read and write operations and, hence, has a constant log size. In contrast to that the other systems

only log write operations in the audit log. This leads to an increasing log size the more write transactions are executed. That log size can be a critical factor for shared databases is indicated by the recent discussion about blockchain-based systems. There, high storage requirements hinder the addition of new nodes (i.e., participants) that are resource-restricted to the network. Hence, investigating the log-size and similar space amplifications are important to support participant scalability.

15.4.3.4 End-to-end Comparison

For a better end-to-end comparison of various systems, we additionally envision an extension of the application workloads to include specific verification and audit requirements. This extension is different from the previous verification and audit workload categories that allow an assessment of varying verification/audit schemes within one system.

Similar to the rationale behind TPC-C, we believe real-world usage scenarios should ideally drive this extension. The specific verification and audit requirements could, for example, be derived from particular industry practices or legislations that mandate, e.g., the frequency of audits or the timeliness of verification (which controls the applicability of offline verification). We believe such requirements might already exist in regulated industries, such as healthcare or finance. However, in-depth industry know-how is required to formulate realistic verification and audit requirements for an end-to-end scenario. We believe our current verification and audit workload categories can aid future discussions with industry experts to define these requirements.

15.5 Related Work

Custom Benchmarks. As mentioned in the introduction, so far most academic works introduced custom benchmarks to evaluate and compare their proposed shared DB system. Both Blockchain Relational Database and LedgerDB use handcrafted benchmarks with custom workloads due to their specific interfaces. Spitz [188], BlockchainDB [52], Veritas [68] use a custom YCSB-like key-value benchmark to evaluate their system end-to-end without considering verification and audit workloads in particular. FalconDB [137] uses the YCSB while ChainfyDB [148] and Basil [160] use the Smallbank benchmark to investigate the end-to-end performance of the system. These systems additionally include custom benchmarks to assess verification effects. The most extensive evaluation so far has been done in GlassDB [185] which defines the new YCSB workloads `workload-X` and `workload-Y` to test verifiability interfaces. Further, they extend the five types

of transactions in TPC-C to verified versions and add a new transaction type called `VerifiedWarehouseBalance` which determines the last 10 versions of the year-to-date balance of a warehouse. Their work also includes additional microbenchmarks to evaluate verification or storage overheads. Our work differentiates from the above-mentioned efforts by proposing a standard approach to benchmarking shared DB systems.

Specialized Benchmarks. More similar to our work in this regard are recent papers that introduce new specialized benchmarks. LEDGERBENCH [189], for instance, is a specialized benchmark for Ledger Databases. It uses Smallbank [5] and a custom range-experiment as macro benchmarks. Further, they define a set of micro benchmarks to evaluate the verification, audit, and storage overhead of ledger databases. In contrast to them, we follow a more modular benchmark design which allows us to incorporate other application workloads. Moreover, we introduce novel benchmark dimensions, e.g., participant scalability, that are critical for evaluating systems in a shared database setting. Further, we propose to perform audit-related benchmarks with a stronger focus on the system instead of an auditor perspective. BLOCKBENCH [44] is a specialized benchmark that targets private blockchains. While private blockchains can be used as a shared database system, they only represent a single possible architecture. Further, they do not distinguish between local and shared transactions since all data and workloads on a blockchain are shared. Another specialized benchmark is GDPRBench [152] which defines workloads that correspond to the core entities of GDPR: controller, customer, processor, and regulator. This is similar to our approach of defining workload categories that evaluate the system based on different capabilities. However, a fundamental difference is that GDPRBench focuses on assessing GDPR-related features of database management systems, while our focus is on the performance of shared DBs.

15.6 Conclusions & Future Work

In this paper, we presented our vision and ideas for a novel benchmark design to evaluate shared database systems. Our benchmark design takes the new paradigm shifts introduced by shared DBs into account and also considers the unique capabilities of those systems.

This is done by defining three categories of workloads, namely application, verification, and audit workloads. Application workloads represent existing database benchmarks like TPC-C or YCSB and are used to evaluate the overall performance of a system. Verification and audit workloads zoom in to the two new characteristics of shared

databases, verifiability and auditability. They are designed to evaluate the overhead of verification and auditing on the overall system to enable a holistic evaluation.

Further, we introduce new benchmark dimensions that are used in our workload categories to assess the effects of typical shared database overheads. These dimensions include participant scalability, shared transactions, write ratio, and audit frequency among others.

For the future, we envision two main areas that are worth exploring in more detail. First, we plan to realize a reference implementation for our proposed benchmark that enables the evaluation of different shared DB systems. Second, as mentioned earlier, we did not focus on data usage and compliance capabilities of shared DB systems, since it is hard to benchmark security in the classical sense. However, we believe that developing frameworks for classifying and assessing the security properties (similar to the approach followed in [172]) is one important area of future work.

15.7 Declarations

- Funding: The research leading to these results received funding from the Federal Ministry of Education and Research (BMBF) under Grant Agreements No 16KIS1267, 2WDG017A and its Research Institute ATHENE as well as from the Federal Ministry for Economic Affairs and Climate Action (BMWK) under Grant Agreement No 01MK21002K.

- Competing interests: All authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript.

Bibliography

[1] Martín Abadi, Andy Chu, Ian J. Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. "Deep Learning with Differential Privacy." In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. ACM, 2016, pp. 308–318. DOI: 10.1145/2976749.2978318. URL: https://doi.org/10.1145/2976749.2978318.

[2] Divyakant Agrawal, Sudipto Das, and Amr El Abbadi. "Big Data and Cloud Computing: New Wine or just New Bottles?" In: *Proc. VLDB Endow.* 3.2 (2010), pp. 1647–1648. DOI: 10.14778/1920841.1921063. URL: http://www.vldb.org/pvldb/vldb2010/pvldb%5C_vol3/T02.pdf.

[3] Ayaz Akram, Anna Giannakou, Venkatesh Akella, Jason Lowe-Power, and Sean Peisert. "Performance Analysis of Scientific Computing Workloads on General Purpose TEEs." In: *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. Portland, OR, USA: IEEE, 2021, pp. 1066–1076. DOI: 10.1109/IPDPS49936.2021.00115.

[4] Mustafa Al-Bassam, Alberto Sonnino, Shehar Bano, Dave Hrycyszyn, and George Danezis. "Chainspace: A Sharded Smart Contracts Platform." In: *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018. URL: http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2018/02/ndss2018%20%5C%5C_09-2%5C%5C_Al-Bassam%5C%5C_paper.pdf.

[5] Mohammad Alomari, Michael J. Cahill, Alan D. Fekete, and Uwe R öhm. "The Cost of Serializability on Platforms That Use Snapshot Isolation." In: *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, Mexico*. Ed. by Gustavo Alonso, José A. Blakeley, and Arbee L. P. Chen.

IEEE Computer Society, 2008, pp. 576–585. DOI: 10.1109/ICDE.2008.4497466. URL: https://doi.org/10.1109/ICDE.2008.4497466.

[6] Thaynara Alves and D. Felton. "Trustzone: Integrated Hardware and Software Security." In: *Information Quarterly* 3.4 (Jan. 2004).

[7] Joseph Amankwah-Amoah, Zaheer Khan, Geoffrey Wood, and Gary Knight. "COVID-19 and digitalization: The great acceleration." In: *Journal of Business Research* 136 (2021), pp. 602–611. ISSN: 0148-2963. DOI: https://doi.org/10.1016/j.jbusres.2021.08.011. URL: https://www.sciencedirect.com/science/article/pii/S0148296321005725.

[8] Amazon Web Services. *The Security Design of the AWS Nitro System*. Nov. 2022. URL: https://docs.aws.amazon.com/pdfs/whitepapers/latest/security-design-of-aws-nitro-system/security-design-of-aws-nitro-system.pdf#security-design-of-aws-nitro-system (visited on 07/12/2023).

[9] Nicolas Anciaux, Philippe Bonnet, Luc Bouganim, Benjamin Nguyen, Philippe Pucheral, Iulian Sandu Popa, and Guillaume Scerri. "Personal Data Management Systems: The security and functionality standpoint." en. In: *Information Systems* 80 (Feb. 2019), pp. 13–35. ISSN: 0306-4379. DOI: 10.1016/j.is.2018.09.002. URL: https://www.sciencedirect.com/science/article/pii/S0306437918304022 (visited on 03/13/2022).

[10] Panagiotis Antonopoulos, Arvind Arasu, Kunal D. Singh, Ken Eguro, Nitish Gupta, Rajat Jain, Raghav Kaushik, Hanuma Kodavalla, Donald Kossmann, Nikolas Ogg, Ravi Ramamurthy, Jakub Szymaszek, Jeffrey Trimmer, Kapil Vaswani, Ramarathnam Venkatesan, and Mike Zwilling. "Azure SQL Database Always Encrypted." en. In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. Portland OR USA: ACM, June 2020, pp. 1511–1525. ISBN: 978-1-4503-6735-6. DOI: 10.1145/3318464.3386141. URL: https://dl.acm.org/doi/10.1145/3318464.3386141 (visited on 11/03/2020).

[11] Panagiotis Antonopoulos, Raghav Kaushik, Hanuma Kodavalla, Sergio Rosales Aceves, Reilly Wong, Jason Anderson, and Jakub Szymaszek. "SQL Ledger: Cryptographically Verifiable Data in Azure SQL Database." In: *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*. Ed. by Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava. ACM, 2021, pp. 2437–2449. DOI: 10.1145/3448016.3457558. URL: https://doi.org/10.1145/3448016.3457558.

[12] Arvind Arasu, Spyros Blanas, Ken Eguro, Manas Joglekar, Raghav Kaushik, Donald Kossmann, Ravishankar Ramamurthy, Prasang Upadhyaya, and Ramarathnam Venkatesan. "Engineering Security and Performance with Cipherbase." In: *IEEE Data Eng. Bull.* 35.4 (2012), pp. 65–72. URL: http://sites.computer.org/debull/A12dec/cipher.pdf.

[13] Arvind Arasu, Badrish Chandramouli, Johannes Gehrke, Esha Ghosh, Donald Kossmann, Jonathan Protzenko, Ravi Ramamurthy, Tahina Ramananandro, Aseem Rastogi, Srinath T. V. Setty, Nikhil Swamy, Alexander van Renen, and Min Xu. "FastVer: Making Data Integrity a Commodity." In: *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*. Ed. by Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava. ACM, 2021, pp. 89–101. DOI: 10.1145/3448016.3457312. URL: https://doi.org/10.1145/3448016.3457312.

[14] Arvind Arasu, Ken Eguro, Raghav Kaushik, Donald Kossmann, Pingfan Meng, Vineet Pandey, and Ravi Ramamurthy. "Concerto: A High Concurrency Key-Value Store with Integrity." In: *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*. Ed. by Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu. ACM, 2017, pp. 251–266. DOI: 10.1145/3035918.3064030. URL: https://doi.org/10.1145/3035918.3064030.

[15] Arvind Arasu, Ken Eguro, Raghav Kaushik, Donald Kossmann, Ravi Ramamurthy, and Ramarathnam Venkatesan. "A secure coprocessor for database applications." In: *23rd International Conference on Field programmable Logic and Applications, FPL 2013, Porto, Portugal, September 2-4, 2013*. IEEE, 2013, pp. 1–8. DOI: 10.1109/FPL.2013.6645524. URL: https://doi.org/10.1109/FPL.2013.6645524.

[16] Asaph Azaria, Ariel Ekblaw, Thiago Vieira, and Andrew Lippman. "MedRec: Using Blockchain for Medical Data Access and Permission Management." In: *2nd International Conference on Open and Big Data, OBD 2016, Vienna, Austria, August 22-24, 2016*. Ed. by Irfan Awan and Muhammad Younas. IEEE Computer Society, 2016, pp. 25–30. DOI: 10.1109/OBD.2016.11. URL: https://doi.org/10.1109/OBD.2016.11.

[17] Sumeet Bajaj, Anrin Chakraborti, and Radu Sion. "ConcurDB: Concurrent Query Authentication for Outsourced Databases." In: *IEEE Trans. Knowl. Data Eng.* 33.4 (2021), pp. 1401–1412. DOI: 10.1109/TKDE.2019.2943557. URL: https://doi.org/10.1109/TKDE.2019.2943557.

[18] Sumeet Bajaj and Radu Sion. "CorrectDB: SQL Engine with Practical Query Authentication." In: *Proc. VLDB Endow.* 6.7 (2013), pp. 529–540. DOI: 10.14778/2536349.2536353. URL: http://www.vldb.org/pvldb/vol6/p529-bajaj.pdf.

[19] Michael Benedikt, Timothy Griffin, and Leonid Libkin. "Verifiable Properties of Database Transactions." In: *Inf. Comput.* 147.1 (1998), pp. 57–88. DOI: 10.1006/inco.1998.2731. URL: https://doi.org/10.1006/inco.1998.2731.

[20] Christian Berger, Birgit Penzenstadler, and Olaf Drögehorn. "On using blockchains for safety-critical systems." In: *Proceedings of the 4th International Workshop on Software Engineering for Smart Cyber-Physical Systems, ICSE 2018, Gothenburg, Sweden, May 27, 2018*. Ed. by Tomás Bures, John S. Fitzgerald, Bradley R. Schmerl, and Danny Weyns. ACM, 2018, pp. 30–36. DOI: 10.1145/3196478.3196480. URL: https://doi.org/10.1145/3196478.3196480.

[21] Elisa Bertino and Ravi S. Sandhu. "Database Security-Concepts, Approaches, and Challenges." In: *IEEE Trans. Dependable Secur. Comput.* 2.1 (2005), pp. 2–19. DOI: 10.1109/TDSC.2005.9. URL: https://doi.org/10.1109/TDSC.2005.9.

[22] Kallista A. Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konený, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. "Towards Federated Learning at Scale: System Design." In: *Proceedings of Machine Learning and Systems 2019, MLSys 2019, Stanford, CA, USA, March 31 - April 2, 2019*. Ed. by Ameet Talwalkar, Virginia Smith, and Matei Zaharia. mlsys.org, 2019. URL: https://proceedings.mlsys.org/book/271.pdf.

[23] Kallista A. Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. "Practical Secure Aggregation for Privacy-Preserving Machine Learning." In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. Ed. by Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM, 2017, pp. 1175–1191. DOI: 10.1145/3133956.3133982. URL: https://doi.org/10.1145/3133956.3133982.

[24] Jan Böttcher, Viktor Leis, Jana Giceva, Thomas Neumann, and Alfons Kemper. "Scalable and robust latches for database systems." In: *16th International Workshop on Data Management on New Hardware, DaMoN 2020, Portland, Oregon, USA,*

June 15, 2020. Ed. by Danica Porobic and Thomas Neumann. ACM, 2020, 2:1–2:8. DOI: 10.1145/3399666.3399908. URL: https://doi.org/10.1145/3399666.3399908.

[25] Stefan Brenner, Tobias Hundt, Giovanni Mazzeo, and Rüdiger Kapitza. "Secure Cloud Micro Services Using Intel SGX." In: *Distributed Applications and Interoperable Systems*. Ed. by Lydia Y. Chen and Hans P. Reiser. Cham: Springer International Publishing, 2017, pp. 177–191. ISBN: 978-3-319-59665-5.

[26] Theodora S. Brisimi, Ruidi Chen, Theofanie Mela, Alex Olshevsky, Ioannis Ch. Paschalidis, and Wei Shi. "Federated learning of predictive models from federated Electronic Health Records." In: *Int. J. Medical Informatics* 112 (2018), pp. 59–67. DOI: 10.1016/j.ijmedinf.2018.01.007. URL: https://doi.org/10.1016/j.ijmedinf.2018.01.007.

[27] Nathan Grasso Bronson, Jared Casper, Hassan Chafi, and Kunle Olukotun. "A practical concurrent binary search tree." In: *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP 2010, Bangalore, India, January 9-14, 2010*. Ed. by R. Govindarajan, David A. Padua, and Mary W. Hall. ACM, 2010, pp. 257–268. DOI: 10.1145/1693453.1693488. URL: https://doi.org/10.1145/1693453.1693488.

[28] Dominique Brunet, Edward R. Vrscay, and Zhou Wang. "On the Mathematical Properties of the Structural Similarity Index." In: *IEEE Trans. Image Process.* 21.4 (2012), pp. 1488–1499. DOI: 10.1109/TIP.2011.2173206. URL: https://doi.org/10.1109/TIP.2011.2173206.

[29] Vitalik Buterin. "A next-generation smart contract and decentralized application platform." In: *white paper* (2014).

[30] Michael J. Cahill. "Serializable Isolation for Snapshot Databases." PhD thesis. University of Sydney, Australia, 2009. URL: https://hdl.handle.net/2123/5353.

[31] Miguel Oom Temudo de Castro. "Practical Byzantine fault tolerance." PhD thesis. Massachusetts Institute of Technology, Cambridge, MA, USA, 2000. URL: https://hdl.handle.net/1721.1/86581.

[32] Catena-X Automotive Network e.V. *Catena-X Homepage*. Ed. by Catena-X Automotive Network e.V. Accessed 19 August 2022. 2022. URL: https://catena-x.net/en/ (visited on 06/29/2022).

[33] Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Jó zefowicz. "Revisiting Distributed Synchronous SGD." In: *CoRR* abs/1604.00981 (2016). arXiv: 1604.00981. URL: http://arxiv.org/abs/1604.00981.

[34] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah M. Johnson, Ari Juels, Andrew Miller, and Dawn Song. "Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contract Execution." In: *CoRR* abs/1804.05141 (2018). arXiv: 1804.05141. URL: http://arxiv.org/abs/1804.05141.

[35] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah M. Johnson, Ari Juels, Andrew Miller, and Dawn Song. "Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contracts." In: *IEEE European Symposium on Security and Privacy, EuroS&P 2019, Stockholm, Sweden, June 17-19, 2019*. IEEE, 2019, pp. 185–200. DOI: 10.1109/EuroSP.2019.00023. URL: https://doi.org/10.1109/EuroSP.2019.00023.

[36] Trishul M. Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. "Project Adam: Building an Efficient and Scalable Deep Learning Training System." In: *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA, October 6-8, 2014*. Ed. by Jason Flinn and Hank Levy. USENIX Association, 2014, pp. 571–582. URL: https://www.usenix.org/conference/osdi14/technical-sessions/presentation/chilimbi.

[37] Consumer Financial Protection Bureau. *CFPB Outlines Principles For Consumer-Authorized Financial Data Sharing and Aggregation*. en. Ed. by Consumer Financial Protection Bureau. Accessed 19 August 2022. URL: https://www.consumerfinance.gov/about-us/newsroom/cfpb-outlines-principles-consumer-authorized-financial-data-sharing-and-aggregation/ (visited on 06/29/2022).

[38] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. "Benchmarking cloud serving systems with YCSB." In: *Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC 2010, Indianapolis, Indiana, USA, June 10-11, 2010*. Ed. by Joseph M. Hellerstein, Surajit Chaudhuri, and Mendel Rosenblum. ACM, 2010, pp. 143–154. DOI: 10.1145/1807128.1807152. URL: https://doi.org/10.1145/1807128.1807152.

[39] Victor Costan, Ilia Lebedev, and Srinivas Devadas. "Sanctum: Minimal Hardware Extensions for Strong Software Isolation." In: *25th USENIX Security Symposium*

(USENIX Security 16). Austin, TX: USENIX Association, Aug. 2016, pp. 857–874. ISBN: 978-1-931971-32-4. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/costan.

[40] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed E. Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, and Roger Wattenhofer. "On Scaling Decentralized Blockchains - (A Position Paper)." In: *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church , Barbados, February 26, 2016, Revised Selected Papers*. Ed. by Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff. Vol. 9604. Lecture Notes in Computer Science. Springer, 2016, pp. 106–125. DOI: 10.1007/978-3-662-53357-4_8. URL: https://doi.org/10.1007/978-3-662-53357-4%5C%5C_8.

[41] Benoit Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huang, Allison W. Lee, Ashish Motivala, Abdul Q. Munir, Steven Pelley, Peter Povinec, Greg Rahn, Spyridon Triantafyllis, and Philipp Unterbrunner. "The Snowflake Elastic Data Warehouse." In: *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*. Ed. by Fatma Özcan, Georgia Koutrika, and Sam Madden. ACM, 2016, pp. 215–226. DOI: 10.1145/2882903.2903741. URL: https://doi.org/10.1145/2882903.2903741.

[42] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew W. Senior, Paul A. Tucker, Ke Yang, and Andrew Y. Ng. "Large Scale Distributed Deep Networks." In: *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*. Ed. by Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger. 2012, pp. 1232–1240. URL: https://proceedings.neurips.cc/paper/2012/hash/6aca97005c68f1206823815f66102863-Abstract.html.

[43] David J. DeWitt and Jim Gray. "Parallel Database Systems: The Future of High Performance Database Systems." In: *Commun. ACM* 35.6 (1992), pp. 85–98. DOI: 10.1145/129888.129894. URL: https://doi.org/10.1145/129888.129894.

[44] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. "BLOCKBENCH: A Framework for Analyzing Private Blockchains."

In: *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*. Ed. by Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu. ACM, 2017, pp. 1085–1100. DOI: 10.1145/3035918.3064033. URL: https://doi.org/10.1145/3035918.3064033.

[45] Tu Dinh Ngoc, Bao Bui, Stella Bitchebe, Alain Tchana, Valerio Schiavoni, Pascal Felber, and Daniel Hagimont. "Everything You Should Know About Intel SGX Performance on Virtualized Systems." In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 3.1 (Mar. 2019), 5:1–5:21. DOI: 10.1145/3322205.3311076. URL: https://doi.org/10.1145/3322205.3311076 (visited on 01/07/2021).

[46] Cynthia Dwork. "Differential Privacy." In: *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*. Ed. by Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener. Vol. 4052. Lecture Notes in Computer Science. Springer, 2006, pp. 1–12. DOI: 10.1007/11787006_1. URL: https://doi.org/10.1007/11787006%5C_1.

[47] Jacob Eberhardt and Jonathan Heiss. "Off-chaining Models and Approaches to Off-chain Computations." In: *Proceedings of the 2nd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers, SERIAL@Middleware 2018, Rennes, France, December 10, 2018*. ACM, 2018, pp. 7–12. DOI: 10.1145/3284764.3284766. URL: https://doi.org/10.1145/3284764.3284766.

[48] Jacob Eberhardt and Stefan Tai. "On or Off the Blockchain? Insights on Off-Chaining Computation and Data." In: *Service-Oriented and Cloud Computing - 6th IFIP WG 2.14 European Conference, ESOCC 2017, Oslo, Norway, September 27-29, 2017, Proceedings*. Ed. by Flavio De Paoli, Stefan Schulte, and Einar Broch Johnsen. Vol. 10465. Lecture Notes in Computer Science. Springer, 2017, pp. 3–15. DOI: 10.1007/978-3-319-67262-5_1. URL: https://doi.org/10.1007/978-3-319-67262-5%5C%5C_1.

[49] Ken Eguro and Ramarathnam Venkatesan. "FPGAs for trusted cloud computing." In: *22nd International Conference on Field Programmable Logic and Applications (FPL), Oslo, Norway, August 29-31, 2012*. Ed. by Dirk Koch, Satnam Singh, and Jim Tørresen. IEEE, 2012, pp. 63–70. DOI: 10.1109/FPL.2012.6339242. URL: https://doi.org/10.1109/FPL.2012.6339242.

[50] Robert Eiss. "Confusion over Europe's data-protection law is stalling scientific progress." en. In: *Nature* 584.7822 (Aug. 2020). Accessed 19 August 2022, pp. 498–498. DOI: 10.1038/d41586-020-02454-7. URL: https://www.nature.com/articles/d41586-020-02454-7 (visited on 06/29/2022).

[51] Muhammad El-Hindi, Ashwin Arora, Simon Karrer, and Carsten Binnig. "Towards a Benchmark for Shared Databases [Vision Paper]." In: *Datenbank-Spektrum* 22.3 (2022), pp. 227–239. DOI: 10.1007/s13222-022-00429-8. URL: https://doi.org/10.1007/s13222-022-00429-8.

[52] Muhammad El-Hindi, Carsten Binnig, Arvind Arasu, Donald Kossmann, and Ravi Ramamurthy. "BlockchainDB - A Shared Database on Blockchains." In: *Proc. VLDB Endow.* 12.11 (2019), pp. 1597–1609. DOI: 10.14778/3342263.3342636. URL: http://www.vldb.org/pvldb/vol12/p1597-el-hindi.pdf.

[53] Muhammad El-Hindi, Martin Heyden, Carsten Binnig, Ravi Ramamurthy, Arvind Arasu, and Donald Kossmann. "BlockchainDB - Towards a Shared Database on Blockchains." In: *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*. Ed. by Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska. ACM, 2019, pp. 1905–1908. DOI: 10.1145/3299869.3320237. URL: https://doi.org/10.1145/3299869.3320237.

[54] Muhammad El-Hindi, Zheguang Zhao, and Carsten Binnig. "ACID-V: Towards a New Class of DBMSs for Data Sharing." In: *Heterogeneous Data Management, Polystores, and Analytics for Healthcare - VLDB Workshops, Poly 2021 and DMAH 2021, Virtual Event, August 20, 2021, Revised Selected Papers*. Ed. by El Kindi Rezig, Vijay Gadepally, Timothy G. Mattson, Michael Stonebraker, Tim Kraska, Fusheng Wang, Gang Luo, Jun Kong, and Alevtina Dubovitskaya. Vol. 12921. Lecture Notes in Computer Science. Springer, 2021, pp. 60–64. DOI: 10.1007/978-3-030-93663-1_5. URL: https://doi.org/10.1007/978-3-030-93663-1_5.

[55] Muhammad El-Hindi, Zheguang Zhao, and Carsten Binnig. "Towards Decentralized Parameter Servers for Secure Federated Learning." In: *Proceedings of the 11th International Conference on Data Science , Technology and Applications, DATA 2022, Lisbon, Portugal, July 11-13, 2022*. Ed. by Alfredo Cuzzocrea, Oleg Gusikhin, Wil M. P. van der Aalst, and Slimane Hammoudi. SCITEPRESS, 2022, pp. 257–269. DOI: 10.5220/0011146300003269. URL: https://doi.org/10.5220/0011146300003269.

[56] Muhammad El-Hindi, Tobias Ziegler, and Carsten Binnig. "Towards Merkle Trees for High-Performance Data Systems." In: *Proceedings of the 1st Workshop on Verifiable Database Systems, VDBS '23, Seattle, WA, USA, June 23, 2023*. Ed. by Tien Tuan Anh Dinh, Beng Chin Ooi, and Xinying Yang. ACM, 2023, pp. 28–33. DOI: 10.1145/3595647.3595651. URL: https://doi.org/10.1145/3595647.3595651.

[57] Muhammad El-Hindi, Tobias Ziegler, Matthias Heinrich, Adrian Lutsch, Zheguang Zhao, and Carsten Binnig. "Benchmarking the Second Generation of Intel SGX Hardware." In: *International Conference on Management of Data, DaMoN 2022, Philadelphia, PA, USA, 13 June 2022*. Ed. by Spyros Blanas and Norman May. ACM, 2022, 5:1–5:8. DOI: 10.1145/3533737.3535098. URL: https://doi.org/10.1145/3533737.3535098.

[58] El-Mahdi El-Mhamdi, Rachid Guerraoui, Arsany Guirguis, Lê-Nguyên Hoang, and Sébastien Rouault. "Genuinely distributed Byzantine machine learning." In: *Distributed Comput.* 35.4 (2022), pp. 305–331. DOI: 10.1007/s00446-022-00427-9. URL: https://doi.org/10.1007/s00446-022-00427-9.

[59] Saba Eskandarian and Matei Zaharia. "ObliDB: oblivious query processing for secure databases." In: *Proceedings of the VLDB Endowment* 13.2 (Oct. 2019), pp. 169–183. ISSN: 2150-8097. DOI: 10.14778/3364324.3364331. URL: https://doi.org/10.14778/3364324.3364331 (visited on 01/07/2021).

[60] EU. "Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)." In: *Official Journal of the European Union* OJ L 119, 4.5.2016 (May 2016), pp. 1–88. URL: https://eur-lex.europa.eu/eli/reg/2016/679/oj.

[61] European Commission. *A European strategy for data*. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52020DC0066. Feb. 2020. (Visited on 02/27/2020).

[62] European Commission. *COMMUNICATION FROM THE COMMISSION TO THE EUROPEAN PARLIAMENT, THE COUNCIL, THE EUROPEAN ECONOMIC AND SOCIAL COMMITTEE AND THE COMMITTEE OF THE REGIONS on a Digital Finance Strategy for the EU*. en. Ed. by European Commission.

Accessed 19 August 2022. Sept. 2020. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52020DC0591 (visited on 06/29/2022).

[63] European Commission. *Proposal for a REGULATION OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL on European data governance (Data Governance Act)*. en. Ed. by European Commission. Accessed 19 August 2022. Nov. 2020. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%5C%%203A52020PC0767 (visited on 06/29/2022).

[64] European Data Protection Board. *Guidelines 07/2020 on the concepts of controller and processor in the GDPR, Version 2.1*. https://edpb.europa.eu/system/files/2021-07/eppb_guidelines_202007_controllerprocessor_final_en.pdf. July 2020. (Visited on 07/07/2021).

[65] Frans Faerber, Alfons Kemper, Per-Ãke Larson, Justin Levandoski, Tjomas Neumann, and Andrew Pavlo. "Main Memory Database Systems." en. In: *Foundations and TrendsÂő in Databases* 8.1-2 (2017), pp. 1–130. ISSN: 1931-7883, 1931-7891. DOI: 10.1561/1900000058. URL: http://www.nowpublishers.com/article/Details/DBS-058 (visited on 03/13/2022).

[66] Andrew Flangas, Autumn Cuellar, Michael Reyes, and Frederick C. Harris. "Parallelized C++ Implementation of a Merkle Tree." en. In: *ITNG 2021 18th International Conference on Information Technology-New Generations*. Ed. by Shahram Latifi. Advances in Intelligent Systems and Computing. Cham: Springer International Publishing, 2021, pp. 107–114. ISBN: 978-3-030-70416-2. DOI: 10.1007/978-3-030-70416-2_13.

[67] Benny Fuhry, H A Jayanth Jain, and Florian Kerschbaum. "EncDBDB: Searchable Encrypted, Fast, Compressed, In-Memory Database Using Enclaves." In: *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. Taipei, Taiwan: IEEE, 2021, pp. 438–450. DOI: 10.1109/DSN48987.2021.00054.

[68] Johannes Gehrke, Lindsay Allen, Panagiotis Antonopoulos, Arvind Arasu, Joachim Hammer, James Hunter, Raghav Kaushik, Donald Kossmann, Ravi Ramamurthy, Srinath T. V. Setty, Jakub Szymaszek, Alexander van Renen, Jonathan Lee, and Ramarathnam Venkatesan. "Veritas: Shared Verifiable Databases and Tables in the Cloud." In: *9th Biennial Conference on Innovative Data Systems Research, CIDR 2019, Asilomar, CA, USA, January 13-16, 2019, Online Proceed-*

ings. www.cidrdb.org, 2019. URL: http://cidrdb.org/cidr2019/papers/p111-gehrke-cidr19.pdf.

[69] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. "Inverting Gradients - How easy is it to break privacy in federated learning?" In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin. 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/c4ede56bbd98819ae6112b20ac6bf145-Abstract.html.

[70] Joshua Gelhaar and Boris Otto. "Challenges in the Emergence of Data Ecosystems." In: *24th Pacific Asia Conference on Information Systems, PACIS 2020, Dubai, UAE, June 22-24, 2020*. Ed. by Doug Vogel, Kathy Ning Shen, Pan Shan Ling, Carol Hsu, James Y. L. Thong, Marco De Marco, Moez Limayem, and Sean Xin Xu. 2020, p. 175. URL: https://aisel.aisnet.org/pacis2020/175.

[71] Craig Gentry. "Fully homomorphic encryption using ideal lattices." In: *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*. Ed. by Michael Mitzenmacher. ACM, 2009, pp. 169–178. DOI: 10.1145/1536414.1536440. URL: https://doi.org/10.1145/1536414.1536440.

[72] Oded Goldreich. "Secure multi-party computation." In: *Manuscript. Preliminary version* 78 (1998).

[73] Javier GonzÃąlez and Philippe Bonnet. "Towards an Open Framework Leveraging a Trusted Execution Environment." en. In: *Cyberspace Safety and Security*. Ed. by Guojun Wang, Indrakshi Ray, Dengguo Feng, and Muttukrishnan Rajarajan. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2013, pp. 458–467. ISBN: 978-3-319-03584-0. DOI: 10.1007/978-3-319-03584-0_35.

[74] Goetz Graefe. "A survey of B-tree locking techniques." In: *ACM Trans. Database Syst.* 35.3 (2010), 16:1–16:26. DOI: 10.1145/1806907.1806908. URL: https://doi.org/10.1145/1806907.1806908.

[75] Shay Gueron. *A Memory Encryption Engine Suitable for General Purpose Processors*. Tech. rep. 204. Intel, 2016. URL: http://eprint.iacr.org/2016/204 (visited on 02/24/2022).

[76] Danny Harnik, Eliad Tsfadia, Doron Chen, and Ronen Kat. *Securing the Storage Data Path with SGX Enclaves*. arXiv: 1806.10883. June 2018. arXiv: 1806.10883 [cs.CR]. URL: http://arxiv.org/abs/1806.10883 (visited on 01/07/2021).

[77] Muhammad El-Hindi, Simon Karrer, Gloria Doci, and Carsten Binnig. "TrustDBle: Towards Trustable Shared Databases." en. In: *3rd International Symposium on Foundations and Applications of Blockchain*. 2020. URL: https://scfab.github.io/2020/FAB2020_p7.pdf.

[78] Tim Hulsen. "Sharing Is Caring: Data Sharing Initiatives in Healthcare." In: *International Journal of Environmental Research and Public Health* 17.9 (May 2020), p. 3046. ISSN: 1661-7827. DOI: 10.3390/ijerph17093046. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7246891/ (visited on 06/29/2022).

[79] Intel. *3rd Gen Intel Xeon Scalable Processors Brief*. en. 2021. URL: https://www.intel.com/content/www/us/en/products/docs/processors/xeon/3rd-gen-xeon-scalable-processors-brief.html (visited on 02/28/2022).

[80] Intel. *Intel Protected File System Library*. Dec. 2016. URL: https://www.intel.com/content/dam/develop/external/us/en/documents/overviewofintelprotectedfilesystemlibrary.pdf (visited on 03/12/2022).

[81] Intel. *Intel Software Guard Extensions SDK Developer Reference*. Nov. 2021. URL: https://download.01.org/intel-sgx/sgx-linux/2.15.1/docs/Intel_SGX_Developer_Reference_Linux_2.15.1_Open_Source.pdf (visited on 02/24/2022).

[82] Intel. *Intel Trust Domain Extensions*. Feb. 2023. URL: https://cdrdv2.intel.com/v1/dl/getContent/690419 (visited on 07/12/2023).

[83] International Air Transport Association. *ONE Record Homepage*. en. Ed. by International Air Transport Association. Accessed 19 August 2022. URL: https://www.iata.org/one-record/ (visited on 06/29/2022).

[84] Rohit Jain and Sunil Prabhakar. "Trustworthy data from untrusted databases." In: *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*. Ed. by Christian S. Jensen, Christopher M. Jermaine, and Xiaofang Zhou. IEEE Computer Society, 2013, pp. 529–540. DOI: 10.1109/ICDE.2013.6544853. URL: https://doi.org/10.1109/ICDE.2013.6544853.

[85] Rohit Jain and Sunil Prabhakar. "Trustworthy data from untrusted databases." In: *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*. Ed. by Christian S. Jensen, Christopher M. Jermaine, and Xiaofang Zhou. IEEE Computer Society, 2013, pp. 529–540. DOI: 10.1109/ICDE.2013.6544853. URL: https://doi.org/10.1109/ICDE.2013.6544853.

[86] Noah M. Johnson, Joseph P. Near, and Dawn Song. "Towards Practical Differential Privacy for SQL Queries." In: *Proc. VLDB Endow.* 11.5 (2018), pp. 526–539. DOI: 10.1145/3187009.3177733. URL: http://www.vldb.org/pvldb/vol11/p526-johnson.pdf.

[87] Simon Johnson, Raghunandan Makaram, Amy Santoni, and Vinnie Scarlata. *Supporting intel sgx on multi-socket platforms*. English. 2021. URL: https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/supporting-intel-sgx-on-mulit-socket-platforms.pdf.

[88] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista A. Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Kone ný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Hang Qi, Daniel Ramage, Ramesh Raskar, Mariana Raykova, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tram` er, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. "Advances and Open Problems in Federated Learning." In: *Found. Trends Mach. Learn.* 14.1-2 (2021), pp. 1–210. DOI: 10.1561/2200000083. URL: https://doi.org/10.1561/2200000083.

[89] Janakirama Kalidhindi, Alex Kazorian, Aneesh Khera, and Cibi Pari. *Angela: A Sparse, Distributed, and Highly Concurrent Merkle Tree*. en. Berkeley: UC Berkeley, 2018, p. 11.

[90] Andreas Kamilaris, Agusti Fonts, and Francesc X. Prenafeta-Boldu. "The Rise of Blockchain Technology in Agriculture and Food Supply Chains." In: *CoRR* abs/1908.07391 (2019). arXiv: 1908.07391. URL: http://arxiv.org/abs/1908.07391.

[91] David Kaplan, Jeremy Powell, and Tom Woller. *AMD memory encryption*. AMD. 2016. URL: http://developer.amd.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf.

[92] Tim Kiefer, Benjamin Schlegel, and Wolfgang Lehner. "Experimental evaluation of NUMA effects on database management systems." In: *Datenbanksysteme für Business, Technologie und Web (BTW) 2025*. Ed. by Volker Markl, Gunter Saake, Kai-Uwe Sattler, Gregor Hackenbroich, Bernhard Mitschang, Theo Härder, and Veit Köppen. Bonn: Gesellschaft für Informatik e.V., 2013, pp. 185–204.

[93] Taehoon Kim, Joongun Park, Jaewook Woo, Seungheun Jeon, and Jaehyuk Huh. "ShieldStore: Shielded In-memory Key-value Storage with SGX." In: *Proceedings of the Fourteenth EuroSys Conference 2019*. EuroSys '19. New York, NY, USA: Association for Computing Machinery, Mar. 2019, pp. 1–15. ISBN: 978-1-4503-6281-8. DOI: 10.1145/3302424.3303951. URL: https://doi.org/10.1145/3302424.3303951 (visited on 02/16/2022).

[94] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. "OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding." In: *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. IEEE Computer Society, 2018, pp. 583–598. DOI: 10.1109/SP.2018.000-5. URL: https://doi.org/10.1109/SP.2018.000-5.

[95] Jakub Konený, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. "Federated Learning: Strategies for Improving Communication Efficiency." In: (2016), pp. 1–10. arXiv: 1610.05492. URL: http://arxiv.org/abs/1610.05492.

[96] Robert Krahn, Donald Dragoti, Franz Gregor, Do Le Quoc, Valerio Schiavoni, Pascal Felber, Clenimar Souza, Andrey Brito, and Christof Fetzer. "TEEMon: A continuous performance monitoring framework for TEEs." In: *Proceedings of the 21st International Middleware Conference*. Middleware '20. New York, NY, USA: Association for Computing Machinery, Dec. 2020, pp. 178–192. ISBN: 978-1-4503-8153-6. DOI: 10.1145/3423211.3425677. URL: https://doi.org/10.1145/3423211.3425677 (visited on 02/20/2022).

[97] Alex Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. University of Toronto, 2009.

[98] Yann LeCun. *The MNIST database of handwritten digits*. Courant Institute, NYU. 1998.

[99] Viktor Leis, Michael Haubenschild, and Thomas Neumann. "Optimistic Lock Coupling: A Scalable and Efficient General-Purpose Synchronization Method." In: *IEEE Data Eng. Bull.* 42.1 (2019), pp. 73–84. URL: http://sites.computer.org/debull/A19mar/p73.pdf.

[100] Viktor Leis, Florian Scheibner, Alfons Kemper, and Thomas Neumann. "The ART of practical synchronization." In: *Proceedings of the 12th International Workshop on Data Management on New Hardware, DaMoN 2016, San Francisco, CA, USA, June 27, 2016*. ACM, 2016, 3:1–3:8. DOI: 10.1145/2933349.2933352. URL: https://doi.org/10.1145/2933349.2933352.

[101] Feifei Li, Marios Hadjieleftheriou, George Kollios, and Leonid Reyzin. "Dynamic authenticated index structures for outsourced databases." In: *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*. Ed. by Surajit Chaudhuri, Vagelis Hristidis, and Neoklis Polyzotis. ACM, 2006, pp. 121–132. DOI: 10.1145/1142473.1142488. URL: https://doi.org/10.1145/1142473.1142488.

[102] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. "Scaling Distributed Machine Learning with the Parameter Server." In: *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA, October 6-8, 2014*. Ed. by Jason Flinn and Hank Levy. USENIX Association, 2014, pp. 583–598. URL: https://www.usenix.org/conference/osdi14/technical-sessions/presentation/li%20%5C%5C_mu.

[103] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. "Federated Learning: Challenges, Methods, and Future Directions." In: *IEEE Signal Process. Mag.* 37.3 (2020), pp. 50–60. DOI: 10.1109/MSP.2020.2975749. URL: https://doi.org/10.1109/MSP.2020.2975749.

[104] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. "Can Decentralized Algorithms Outperform Centralized Algorithms? A Case Study for Decentralized Parallel Stochastic Gradient Descent." In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob

Fergus, S. V. N. Vishwanathan, and Roman Garnett. 2017, pp. 5330–5340. URL: https://proceedings.neurips.cc/paper/2017/hash/f75526659f31040afeb61cb7133e4e6d-Abstract.html.

[105] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and Bill Dally. "Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training." In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL: https://openreview.net/forum?id=SkhQHMW0W.

[106] Dong C. Liu and Jorge Nocedal. "On the limited memory BFGS method for large scale optimization." In: *Math. Program.* 45.1-3 (1989), pp. 503–528. DOI: 10.1007/BF01589116. URL: https://doi.org/10.1007/BF01589116.

[107] Ximing Liu, Wenwen Wang, Lizhi Wang, Xiaoli Gong, Ziyi Zhao, and Pen-Chung Yew. "Regaining Lost Seconds: Efficient Page Preloading for SGX Enclaves." In: *Proceedings of the 21st International Middleware Conference*. Middleware '20. New York, NY, USA: Association for Computing Machinery, Dec. 2020, pp. 326–340. ISBN: 978-1-4503-8153-6. DOI: 10.1145/3423211.3425673. URL: https://doi.org/10.1145/3423211.3425673 (visited on 03/12/2022).

[108] Yang Liu, Yan Kang, Chaoping Xing, Tianjian Chen, and Qiang Yang. "A Secure Federated Transfer Learning Framework." In: *IEEE Intell. Syst.* 35.4 (2020), pp. 70–82. DOI: 10.1109/MIS.2020.2988525. URL: https://doi.org/10.1109/MIS.2020.2988525.

[109] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. "A Secure Sharding Protocol For Open Blockchains." In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. ACM, 2016, pp. 17–30. DOI: 10.1145/2976749.2978389. URL: https://doi.org/10.1145/2976749.2978389.

[110] Mohammad Mahhouk, Nico Weichbrodt, and RÃijdiger Kapitza. "SGXoMeter: Open and Modular Benchmarking for Intel SGX." In: *Proceedings of the 14th European Workshop on Systems Security*. EuroSec '21. New York, NY, USA: Association for Computing Machinery, Apr. 2021, pp. 55–61. ISBN: 978-1-4503-8337-0. DOI: 10.1145/3447852.3458722. URL: https://doi.org/10.1145/3447852.3458722 (visited on 02/15/2022).

[111] Sujaya Maiyya, Danny Hyun Bum Cho, Divyakant Agrawal, and Amr El Abbadi. "Fides: Managing Data on Untrusted Infrastructure." In: *40th IEEE International Conference on Distributed Computing Systems, ICDCS 2020, Singapore, November 29 - December 1, 2020*. IEEE, 2020, pp. 344–354. DOI: 10.1109/ICDCS47774.2020.00053. URL: https://doi.org/10.1109/ICDCS47774.2020.00053.

[112] Kajetan Maliszewski, Jorge-Arnulfo QuianÃĺ-Ruiz, Jonas Traub, and Volker Markl. "What is the price for joining securely?: benchmarking equi-joins in trusted execution environments." en. In: *Proceedings of the VLDB Endowment* 15.3 (Nov. 2021), pp. 659–672. ISSN: 2150-8097. DOI: 10.14778/3494124.3494146. URL: https://dl.acm.org/doi/10.14778/3494124.3494146 (visited on 02/16/2022).

[113] Horia Mania, Xinghao Pan, Dimitris S. Papailiopoulos, Benjamin Recht, Kannan Ramchandran, and Michael I. Jordan. "Perturbed Iterate Analysis for Asynchronous Stochastic Optimization." In: *SIAM J. Optim.* 27.4 (2017), pp. 2202–2229. DOI: 10.1137/16M1057000. URL: https://doi.org/10.1137/16M1057000.

[114] Luther Martin. "XTS: A Mode of AES for Encrypting Hard Disks." In: *IEEE Security Privacy* 8.3 (2010), pp. 68–69. DOI: 10.1109/MSP.2010.111.

[115] Trent McConaghy et al. "BigchainDB: a scalable blockchain database." In: *white paper, BigChainDB* (2016).

[116] Frank McKeen, Ilya Alexandrovich, Ittai Anati, Dror Caspi, Simon Johnson, Rebekah Leslie-Hurd, and Carlos Rozas. "Intel Software Guard Extensions (Intel SGX) Support for Dynamic Memory Management Inside an Enclave." In: *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*. HASP 2016. New York, NY, USA: Association for Computing Machinery, June 2016, pp. 1–9. ISBN: 978-1-4503-4769-3. DOI: 10.1145/2948618.2954331. URL: https://doi.org/10.1145/2948618.2954331 (visited on 02/25/2022).

[117] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. "Innovative instructions and software model for isolated execution." en. In: *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy - HASP '13*. Tel-Aviv, Israel: ACM Press, 2013, pp. 1–1. ISBN: 978-1-4503-2118-1. DOI: 10.1145/2487726.2488368. URL: http://dl.acm.org/citation.cfm?doid=2487726.2488368 (visited on 11/03/2020).

[118] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. "Communication-Efficient Learning of Deep Networks from Decentralized Data." In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*. Ed. by Aarti Singh and Xiaojin (Jerry) Zhu. Vol. 54. Proceedings of Machine Learning Research. PMLR, 2017, pp. 1273–1282. URL: http://proceedings.mlr.press/v54/mcmahan17a.html.

[119] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. "Learning Differentially Private Recurrent Language Models." In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL: https://openreview.net/forum?id=BJ0hF1Z0b.

[120] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. "Inference Attacks Against Collaborative Learning." In: *CoRR* abs/1805.04049 (2018). arXiv: 1805.04049. URL: http://arxiv.org/abs/1805.04049.

[121] Ralph C. Merkle. "A Digital Signature Based on a Conventional Encryption Function." In: *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*. Ed. by Carl Pomerance. Vol. 293. Lecture Notes in Computer Science. Springer, 1987, pp. 369–378. DOI: 10.1007/3-540-48184-2_32. URL: https://doi.org/10.1007/3-540-48184-2%5C%5C_32.

[122] Microsoft News Center. *Adobe, Microsoft and SAP announce the Open Data Initiative to empower a new generation of customer experiences*. en-US. Ed. by Microsoft News Center. Accessed 19 August 2022. Sept. 2018. URL: https://news.microsoft.com/2018/09/24/adobe-microsoft-and-sap-announce-the-open-data-initiative-to-empower-a-new-generation-of-customer-experiences/ (visited on 06/29/2022).

[123] Saeid Mofrad, Fengwei Zhang, Shiyong Lu, and Weidong Shi. "A comparison study of intel SGX and AMD memory encryption technology." In: *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy*. HASP '18. New York, NY, USA: Association for Computing Machinery, June 2018, pp. 1–8. ISBN: 978-1-4503-6500-0. DOI: 10.1145/3214292.3214301. URL: https://doi.org/10.1145/3214292.3214301 (visited on 10/06/2020).

[124] Payman Mohassel and Yupeng Zhang. "SecureML: A System for Scalable Privacy-Preserving Machine Learning." In: *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 19–38. DOI: 10.1109/SP.2017.12. URL: https://doi.org/10.1109/SP.2017.12.

[125] Carlos Molina-Jiménez, Ioannis Sfyrakis, Ellis Solaiman, Irene C. L. Ng, Meng Weng Wong, Alexis Chun, and Jon Crowcroft. "Implementation of Smart Contracts Using Hybrid Architectures with On and Off-Blockchain Components." In: *8th IEEE International Symposium on Cloud and Service Computing, SC2 2018, Paris, France, November 18-21, 2018*. IEEE, 2018, pp. 83–90. DOI: 10.1109/SC2.2018.00018. URL: https://doi.org/10.1109/SC2.2018.00018.

[126] Satoshi Nakamoto. "Bitcoin: A Peer-to-Peer Electronic Cash System." In: (2008).

[127] Maithili Narasimha and Gene Tsudik. "Authentication of Outsourced Databases Using Signature Aggregation and Chaining." In: *Database Systems for Advanced Applications, 11th International Conference, DASFAA 2006, Singapore, April 12-15, 2006, Proceedings*. Ed. by Mong-Li Lee, Kian-Lee Tan, and Vilas Wuwongse. Vol. 3882. Lecture Notes in Computer Science. Springer, 2006, pp. 420–436. DOI: 10.1007/11733836_30. URL: https://doi.org/10.1007/11733836%5C%5C_30.

[128] Afonso Araújo Neto and Marco Vieira. "TO BEnchmark or NOT TO BEnchmark security: That is the question." In: *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W 2011), Hong Kong, China, June 27-30, 2011*. IEEE Computer Society, 2011, pp. 182–187. DOI: 10.1109/DSNW.2011.5958810. URL: https://doi.org/10.1109/DSNW.2011.5958810.

[129] Olga Ohrimenko, Felix Schuster, Cedric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. "Oblivious Multi-Party Machine Learning on Trusted Processors." In: *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 619–636. ISBN: 978-1-931971-32-4. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/ohrimenko.

[130] Diego Ongaro and John K. Ousterhout. "In Search of an Understandable Consensus Algorithm." In: *2014 USENIX Annual Technical Conference, USENIX ATC '14, Philadelphia, PA, USA, June 19-20, 2014*. Ed. by Garth Gibson and Nickolai

Zeldovich. USENIX Association, 2014, pp. 305–319. URL: https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro.

[131] Meni Orenbach, Pavel Lifshits, Marina Minkin, and Mark Silberstein. "Eleos: ExitLess OS Services for SGX Enclaves." In: *Proceedings of the Twelfth European Conference on Computer Systems*. EuroSys '17. New York, NY, USA: Association for Computing Machinery, Apr. 2017, pp. 238–253. ISBN: 978-1-4503-4938-3. DOI: 10.1145/3064176.3064219. URL: https://doi.org/10.1145/3064176.3064219 (visited on 02/28/2022).

[132] Róbert Ormándi, István Hegedüs, and Márk Jelasity. "Gossip learning with linear models on fully distributed data." In: *Concurr. Comput. Pract. Exp.* 25.4 (2013), pp. 556–571. DOI: 10.1002/cpe.2858. URL: https://doi.org/10.1002/cpe.2858.

[133] Boris Otto, Michael ten Hompel, and Stefan Wrobel, eds. *Designing Data Spaces: The Ecosystem Approach to Competitive Advantage*. Springer, 2022. ISBN: 978-3-030-93975-5. DOI: 10.1007/978-3-030-93975-5. URL: https://doi.org/10.1007/978-3-030-93975-5.

[134] Ilker Özçelik, Sai Medury, Justin T. Broaddus, and Anthony Skjellum. "An Overview of Cryptographic Accumulators." In: *Proceedings of the 7th International Conference on Information Systems Security and Privacy, ICISSP 2021, Online Streaming, February 11-13, 2021*. Ed. by Paolo Mori, Gabriele Lenzini, and Steven Furnell. SCITEPRESS, 2021, pp. 661–669. DOI: 10.5220/0010337806610669. URL: https://doi.org/10.5220/0010337806610669.

[135] M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems, Third Edition*. Springer, 2011. ISBN: 978-1-4419-8833-1. DOI: 10.1007/978-1-4419-8834-8. URL: https://doi.org/10.1007/978-1-4419-8834-8.

[136] HweeHwa Pang, Jilian Zhang, and Kyriakos Mouratidis. "Scalable Verification for Outsourced Dynamic Databases." In: *Proc. VLDB Endow.* 2.1 (2009), pp. 802–813. DOI: 10.14778/1687627.1687718. URL: http://www.vldb.org/pvldb/vol2/vldb09-625.pdf.

[137] Yanqing Peng, Min Du, Feifei Li, Raymond Cheng, and Dawn Song. "FalconDB: Blockchain-based Collaborative Database." In: *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. Ed. by David Maier, Rachel Pottinger, AnHai Doan, Wang- Chiew Tan, Abdussalam Alawini, and Hung Q.

Ngo. ACM, 2020, pp. 637–652. DOI: 10.1145/3318464.3380594. URL: https://doi.org/10.1145/3318464.3380594.

[138] Peter Ottaviano. *National Freight Data Portal One Step Closer to Reality*. en. Ed. by Peter Ottaviano. Accessed 19 August 2022. Mar. 2022. URL: https://www.businesswire.com/news/home/20220310005691/en/National-Freight-Data-Portal-One-Step-Closer-to-Reality (visited on 06/29/2022).

[139] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. "Privacy-Preserving Deep Learning via Additively Homomorphic Encryption." In: *IEEE Trans. Inf. Forensics Secur.* 13.5 (2018), pp. 1333–1345. DOI: 10.1109/TIFS.2017.2787987. URL: https://doi.org/10.1109/TIFS.2017.2787987.

[140] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. "Privacy-Preserving Deep Learning: Revisited and Enhanced." In: *Applications and Techniques in Information Security - 8th International Conference, ATIS 2017, Auckland, New Zealand, July 6-7, 2017, Proceedings*. Ed. by Lynn Batten, Dong Seong Kim, Xuyun Zhang, and Gang Li. Vol. 719. Communications in Computer and Information Science. Springer, 2017, pp. 100–110. DOI: 10.1007/978-981-10-5421-1_9. URL: https://doi.org/10.1007/978-981-10-5421-1%5C%5C_9.

[141] Raluca A. Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. "CryptDB: protecting confidentiality with encrypted query processing." In: *Proceedings of the 23rd ACM Symposium on Operating Systems Principles 2011, SOSP 2011, Cascais, Portugal, October 23-26, 2011*. Ed. by Ted Wobber and Peter Druschel. ACM, 2011, pp. 85–100. DOI: 10.1145/2043556.2043566. URL: https://doi.org/10.1145/2043556.2043566.

[142] Port of Long Beach. *West Coast Ports Support 'Supply Chain Information Highway'*. en. Ed. by Port of Long Beach. Accessed 19 August 2022. 2022. URL: https://polb.com/port-info/news-and-press/west-coast-ports-support-supply-chain-information-highway-03-03-2022/ (visited on 06/29/2022).

[143] Christian Priebe, Kapil Vaswani, and Manuel Costa. "EnclaveDB: A Secure Database Using SGX." In: *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. IEEE Computer Society, 2018, pp. 264–278. DOI: 10.1109/SP.2018.00025. URL: https://doi.org/10.1109/SP.2018.00025.

[144] Publications Office of the European Union. *European Commission launched the Support Centre for Data Sharing! | data.europa.eu*. Ed. by Publications Office of the European Union. Accessed 19 August 2022. URL: https://data.europa.eu/en/news/european-commission-launched-support-centre-data-sharing (visited on 06/29/2022).

[145] Ivan Puddu, Alexandra Dmitrienko, and Srdjan Capkun. "μchain: How to Forget without Hard Forks." In: *IACR Cryptology ePrint Archive* 2017 (2017), p. 106. URL: http://eprint.iacr.org/2017/106.

[146] Zhijie Ren, Kelong Cong, Johan Pouwelse, and Zekeriya Erkin. "Implicit Consensus: Blockchain with Unbounded Throughput." In: *CoRR* abs/1705.11046 (2017). arXiv: 1705.11046. URL: http://arxiv.org/abs/1705.11046.

[147] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. "Role-Based Access Control Models." In: *Computer* 29.2 (1996), pp. 38–47. DOI: 10.1109/2.485845. URL: https://doi.org/10.1109/2.485845.

[148] Felix Martin Schuhknecht, Ankur Sharma, Jens Dittrich, and Divya Agrawal. "chainifyDB: How to get rid of your Blockchain and use your DBMS instead." In: *11th Conference on Innovative Data Systems Research, CIDR 2021 , Virtual Event, January 11-15, 2021, Online Proceedings*. www.cidrdb.org, 2021. URL: http://cidrdb.org/cidr2021/papers/cidr2021%5C%5C_paper04.pdf.

[149] Felix Schuster, Manuel Costa, CÃľdric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. "VC3: Trustworthy Data Analytics in the Cloud Using SGX." In: *Proceedings of the 2015 IEEE Symposium on Security and Privacy*. SP '15. USA: IEEE Computer Society, May 2015, pp. 38–54. ISBN: 978-1-4673-6949-7. DOI: 10.1109/SP.2015.10. URL: https://doi.org/10.1109/SP.2015.10 (visited on 03/13/2022).

[150] Priya Seetharaman. "Business models shifts: Impact of Covid-19." In: *International Journal of Information Management* 54 (2020), p. 102173. ISSN: 0268-4012. DOI: https://doi.org/10.1016/j.ijinfomgt.2020.102173. URL: https://www.sciencedirect.com/science/article/pii/S0268401220309890.

[151] Ankur Sharma, Felix Martin Schuhknecht, Divya Agrawal, and Jens Dittrich. "Blurring the Lines between Blockchains and Database Systems: the Case of Hyperledger Fabric." In: *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*. Ed. by Peter A. Boncz, Stefan Manegold, Anastasia

Ailamaki, Amol Deshpande, and Tim Kraska. ACM, 2019, pp. 105–122. DOI: 10.1145/3299869.3319883. URL: https://doi.org/10.1145/3299869.3319883.

[152] Supreeth Shastri, Vinay Banakar, Melissa Wasserman, Arun Kumar, and Vijay Chidambaram. "Understanding and Benchmarking the Impact of GDPR on Database Systems." In: *Proc. VLDB Endow.* 13.7 (2020), pp. 1064–1077. DOI: 10.14778/3384345.3384354. URL: http://www.vldb.org/pvldb/vol13/p1064-shastri.pdf.

[153] Amit P. Sheth and James A. Larson. "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases." In: *ACM Comput. Surv.* 22.3 (1990), pp. 183–236. DOI: 10.1145/96602.96604. URL: https://doi.org/10.1145/96602.96604.

[154] Erez Shmueli, Ronen Vaisenberg, Yuval Elovici, and Chanan Glezer. "Database encryption: an overview of contemporary challenges and design considerations." In: *SIGMOD Rec.* 38.3 (2009), pp. 29–34. DOI: 10.1145/1815933.1815940. URL: https://doi.org/10.1145/1815933.1815940.

[155] Reza Shokri and Vitaly Shmatikov. "Privacy-preserving deep learning." In: *53rd Annual Allerton Conference on Communication, Control, and Computing, Allerton 2015, Allerton Park & Retreat Center, Monticello, IL, USA, September 29 - October 2, 2015*. IEEE, 2015, pp. 909–910. DOI: 10.1109/ALLERTON.2015.7447103. URL: https://doi.org/10.1109/ALLERTON.2015.7447103.

[156] Rohit Sinha and Mihai Christodorescu. "VeritasDB: High Throughput Key-Value Store with Integrity." In: *IACR Cryptol. ePrint Arch.* (2018), p. 251. URL: http://eprint.iacr.org/2018/251.

[157] Emily R Smith and Valerie J Flaherman. "Why you should share your data during a pandemic." In: *BMJ Global Health* 6.3 (2021).

[158] V. Srinivasan and Michael J. Carey. "Performance of B+ tree concurrency control algorithms." en. In: *The VLDB Journal* 2.4 (Oct. 1993), pp. 361–406. ISSN: 0949-877X. DOI: 10.1007/BF01263046.

[159] Yuanyuan Sun, Sheng Wang, Huorong Li, and Feifei Li. "Building enclave-native storage engines for practical encrypted databases." In: *Proceedings of the VLDB Endowment* 14.6 (Feb. 2021), pp. 1019–1032. ISSN: 2150-8097. DOI: 10.14778/3447689.3447705. URL: https://doi.org/10.14778/3447689.3447705 (visited on 02/16/2022).

[160] Florian Suri-Payer, Matthew Burke, Zheng Wang, Yunhao Zhang, Lorenzo Alvisi, and Natacha Crooks. "Basil: Breaking up BFT with ACID (transactions)." In: *SOSP '21: ACM SIGOPS 28th Symposium on Operating Systems Principles, Virtual Event / Koblenz, Germany, October 26-29, 2021*. Ed. by Robbert van Renesse and Nickolai Zeldovich. ACM, 2021, pp. 1–17. DOI: 10.1145/3477132.3483552. URL: https://doi.org/10.1145/3477132.3483552.

[161] Kuniyasu Suzaki, Kenta Nakajima, Tsukasa Oi, and Akira Tsukamoto. "TS-Perf: General Performance Measurement of Trusted Execution Environment and Rich Execution Environment on Intel SGX, Arm TrustZone, and RISC-V Keystone." In: *IEEE Access* 9 (2021). Conference Name: IEEE Access, pp. 133520–133530. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3112202.

[162] Meysam Taassori, Ali Shafiee, and Rajeev Balasubramonian. "VAULT: Reducing Paging Overheads in SGX with Efficient Integrity Verification Structures." en. In: *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. Williamsburg VA USA: ACM, Mar. 2018, pp. 665–678. ISBN: 978-1-4503-4911-6. DOI: 10.1145/3173162.3177155. URL: https://dl.acm.org/doi/10.1145/3173162.3177155 (visited on 02/28/2022).

[163] Don Tapscott and Alex Tapscott. *Blockchain revolution: how the technology behind bitcoin is changing money, business, and the world*. Penguin, 2016.

[164] Doug Terry. "Replicated data consistency explained through baseball." In: *Commun. ACM* 56.12 (2013), pp. 82–89. DOI: 10.1145/2500500. URL: https://doi.org/10.1145/2500500.

[165] The Linux Foundation. *Hyperledger Avalon*. Hyperledger Avalon. https://www.hyperledger.org/projects/avalon. 2020.

[166] Hongliang Tian, Qiong Zhang, Shoumeng Yan, Alex Rudnitsky, Liron Shacham, Ron Yariv, and Noam Milshten. "Switchless Calls Made Practical in Intel SGX." In: *Proceedings of the 3rd Workshop on System Software for Trusted Execution*. SysTEX '18. New York, NY, USA: Association for Computing Machinery, Jan. 2018, pp. 22–27. ISBN: 978-1-4503-5998-6. DOI: 10.1145/3268935.3268942. URL: https://doi.org/10.1145/3268935.3268942 (visited on 03/13/2022).

[167] Alin Tomescu, Vivek Bhupatiraju, Dimitrios Papadopoulos, Charalampos Papamanthou, Nikos Triandopoulos, and Srinivas Devadas. "Transparency Logs via Append-Only Authenticated Dictionaries." In: *Proceedings of the 2019 ACM*

SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019. Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. ACM, 2019, pp. 1299–1316. DOI: 10.1145/3319535.3345652. URL: https://doi.org/10.1145/3319535.3345652.

[168] Transaction Processing Performance Council (TPC). *TPC BENCHMARK C - Standard specification - Revision 5.11*. en. Ed. by Transaction Processing Performance Council (TPC). Accessed 19 August 2022. Feb. 2010. URL: https://www.tpc.org/tpc%5C_documents%5C_current%5C_versions/pdf/tpc-c%5C_v%205.11.0.pdf (visited on 08/14/2022).

[169] Sarah Underwood. "Blockchain beyond bitcoin." In: *Commun. ACM* 59.11 (2016), pp. 15–17. DOI: 10.1145/2994581. URL: https://doi.org/10.1145/2994581.

[170] Sébastien Vaucher, Rafael Pires, Pascal Felber, Marcelo Pasin, Valerio Schiavoni, and Christof Fetzer. "SGX-Aware Container Orchestration for Heterogeneous Clusters." In: *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. Vienna, Austria: IEEE, 2018, pp. 730–741. DOI: 10.1109/ICDCS.2018.00076.

[171] Srinivas Devadas Victor Costan. *Intel SGX Explained*. Tech. rep. 086. MIT, 2016. URL: http://eprint.iacr.org/2016/086 (visited on 01/07/2021).

[172] Marco Vieira and Henrique Madeira. "Towards a Security Benchmark for Database Management Systems." In: *2005 International Conference on Dependable Systems and Networks (DSN 2005), 28 June - 1 July 2005, Yokohama, Japan, Proceedings*. IEEE Computer Society, 2005, pp. 592–601. DOI: 10.1109/DSN.2005.93. URL: https://doi.org/10.1109/DSN.2005.93.

[173] Dhinakaran Vinayagamurthy, Alexey Gribov, and Sergey Gorbunov. "StealthDB: a Scalable Encrypted Database with Full SQL Query Support." en. In: *Proceedings on Privacy Enhancing Technologies* 2019.3 (July 2019). Publisher: Sciendo Section: Proceedings on Privacy Enhancing Technologies, pp. 370–388. DOI: 10.2478/popets-2019-0052. URL: https://content.sciendo.com/view/journals/popets/2019/3/article-p370.xml (visited on 12/14/2020).

[174] Jianfeng Wang and Xiaofeng Chen. "Efficient and Secure Storage for Outsourced Data: A Survey." In: *Data Sci. Eng.* 1.3 (2016), pp. 178–188. DOI: 10.1007/s41019-016-0018-9. URL: https://doi.org/10.1007/s41019-016-0018-9.

[175] Jianfeng Wang, Xiaofeng Chen, Xinyi Huang, Ilsun You, and Yang Xiang. "Verifiable Auditing for Outsourced Database in Cloud Computing." In: *IEEE Trans. Computers* 64.11 (2015), pp. 3293–3303. DOI: 10.1109/TC.2015.2401036. URL: https://doi.org/10.1109/TC.2015.2401036.

[176] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. "Image quality assessment: from error visibility to structural similarity." In: *IEEE Trans. Image Process.* 13.4 (2004), pp. 600–612. DOI: 10.1109/TIP.2003.819861. URL: https://doi.org/10.1109/TIP.2003.819861.

[177] Wenqi Wei, Ling Liu, Margaret Loper, Ka Ho Chow, Mehmet Emre Gursoy, Stacey Truex, and Yanzhao Wu. "A Framework for Evaluating Gradient Leakage Attacks in Federated Learning." In: *CoRR* abs/2004.10397 (2020). arXiv: 2004.10397. URL: https://arxiv.org/abs/2004.10397.

[178] Nico Weichbrodt, Pierre-Louis Aublin, and RÃijdiger Kapitza. "sgx-perf: A Performance Analysis Tool for Intel SGX Enclaves." In: *Proceedings of the 19th International Middleware Conference*. Middleware '18. New York, NY, USA: Association for Computing Machinery, Nov. 2018, pp. 201–213. ISBN: 978-1-4503-5702-9. DOI: 10.1145/3274808.3274824. URL: https://doi.org/10.1145/3274808.3274824 (visited on 02/20/2022).

[179] Ofir Weisse, Valeria Bertacco, and Todd Austin. "Regaining Lost Cycles with HotCalls: A Fast Interface for SGX Secure Enclaves." In: *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ISCA '17. New York, NY, USA: Association for Computing Machinery, June 2017, pp. 81–93. ISBN: 978-1-4503-4892-8. DOI: 10.1145/3079856.3080208. URL: https://doi.org/10.1145/3079856.3080208 (visited on 03/13/2022).

[180] Newton C. Will and Carlos A. Maziero. "Intel Software Guard Extensions Applications: A Survey." In: *ACM Comput. Surv.* (Apr. 2023). Just Accepted. ISSN: 0360-0300. DOI: 10.1145/3593021. URL: https://doi.org/10.1145/3593021.

[181] Yu Xia, Xiangyao Yu, Matthew Butrovich, Andrew Pavlo, and Srinivas Devadas. "Litmus: Towards a Practical Database Management System with Verifiable ACID Properties and Transaction Correctness." In: *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*. Ed. by Zachary G. Ives, Angela Bonifati, and Amr El Abbadi. ACM, 2022, pp. 1478–1492. DOI: 10.1145/3514221.3517851. URL: https://doi.org/10.1145/3514221.3517851.

[182] Bin Cedric Xing, Mark Shanahan, and Rebekah Leslie-Hurd. "IntelÂő Software Guard Extensions (Intel SGX) Software Support for Dynamic Memory Allocation inside an Enclave." en. In: *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016 on - HASP 2016*. Seoul, Republic of Korea: ACM Press, 2016, pp. 1–9. ISBN: 978-1-4503-4769-3. DOI: 10.1145/2948618.2954330. URL: http://dl.acm.org/citation.cfm?doid=2948618.2954330 (visited on 02/25/2022).

[183] Eric P. Xing, Qirong Ho, Wei Dai, Jin Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. "Petuum: A New Platform for Distributed Machine Learning on Big Data." In: *IEEE Trans. Big Data* 1.2 (2015), pp. 49–67. DOI: 10.1109/TBDATA.2015.2472014. URL: https://doi.org/10.1109/TBDATA.2015.2472014.

[184] Xinying Yang, Yuan Zhang, Sheng Wang, Benquan Yu, Feifei Li, Yize Li, and Wenyuan Yan. "LedgerDB: A Centralized Ledger Database for Universal Audit and Verification." In: *Proc. VLDB Endow.* 13.12 (2020), pp. 3138–3151. DOI: 10.14778/3415478.3415540. URL: http://www.vldb.org/pvldb/vol13/p3138-yang.pdf.

[185] Cong Yue, Tien Tuan Anh Dinh, Zhongle Xie, Meihui Zhang, Gang Chen, Beng Chin Ooi, and Xiaokui Xiao. "GlassDB: Practical Verifiable Ledger Database Through Transparency." In: *CoRR* abs/2207.00944 (2022). DOI: 10.48550/arXiv.2207.00944. arXiv: 2207.00944. URL: https://doi.org/10.48550/arXiv.2207.00944.

[186] Matei Zaharia, Ali Ghodsi, Reynold Xin, and Michael Armbrust. "Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics." In: *11th Conference on Innovative Data Systems Research, CIDR 2021 , Virtual Event, January 11-15, 2021, Online Proceedings*. www.cidrdb.org, 2021. URL: http://cidrdb.org/cidr2021/papers/cidr2021%5C_paper17.pdf.

[187] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. "RapidChain: Scaling Blockchain via Full Sharding." In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. Ed. by David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang. ACM, 2018, pp. 931–948. DOI: 10.1145/3243734.3243853. URL: https://doi.org/10.1145/3243734.3243853.

[188] Meihui Zhang, Zhongle Xie, Cong Yue, and Ziyue Zhong. "Spitz: A Verifiable Database System." In: *Proc. VLDB Endow.* 13.12 (2020), pp. 3449–3460. DOI: 10.14778/3415478.3415567. URL: http://www.vldb.org/pvldb/vol13/p3449-zhang.pdf.

[189] Meihui Zhang, Cong Yue, Changhao Zhu, and Ziyue Zhong. "LEDGERBENCH: A Framework for Benchmarking Ledger Databases." In: *IEEE Data Eng. Bull.* 45.2 (2022), pp. 59–69. URL: http://sites.computer.org/debull/A22june/p59.pdf.

[190] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. "vSQL: Verifying Arbitrary SQL Queries over Dynamic Outsourced Databases." In: *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 863–880. DOI: 10.1109/SP.2017.43. URL: https://doi.org/10.1109/SP.2017.43.

[191] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. "IntegriDB: Verifiable SQL for Outsourced Databases." In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel. ACM, 2015, pp. 1480–1491. DOI: 10.1145/2810103.2813711. URL: https://doi.org/10.1145/2810103.2813711.

[192] Zhipeng Zhang, Wentao Wu, Jiawei Jiang, Lele Yu, Bin Cui, and Ce Zhang. "ColumnSGD: A Column-oriented Framework for Distributed Stochastic Gradient Descent." In: *IEEE ICDE 2020*. IEEE, 2020, pp. 1513–1524. DOI: 10.1109/ICDE48307.2020.00134. URL: https://doi.org/10.1109/ICDE48307.2020.00134.

[193] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. "iDLG: Improved Deep Leakage from Gradients." In: *CoRR* abs/2001.02610 (2020). arXiv: 2001.02610. URL: http://arxiv.org/abs/2001.02610.

[194] C. Zhao, D. Saifuding, H. Tian, Y. Zhang, and C. Xing. "On the Performance of Intel SGX." In: *2016 13th Web Information Systems and Applications Conference (WISA)*. Wuhan, China: IEEE, Sept. 2016, pp. 184–187. DOI: 10.1109/WISA.2016.45.

[195] Jinwei Zhu, Kun Cheng, Jiayang Liu, and Liang Guo. "Full encryption: an end to end encryption mechanism in GaussDB." In: *Proceedings of the VLDB Endowment* 14.12 (July 2021), pp. 2811–2814. ISSN: 2150-8097. DOI: 10.14778/3476311.3476351. URL: https://doi.org/10.14778/3476311.3476351 (visited on 02/16/2022).

[196] Ligeng Zhu and Song Han. "Deep Leakage from Gradients." In: *Federated Learning - Privacy and Incentive*. Ed. by Qiang Yang, Lixin Fan, and Han Yu. Vol. 12500. Lecture Notes in Computer Science. Springer, 2020, pp. 17–31. DOI: 10.1007/978-3-030-63076-8_2. URL: https://doi.org/10.1007/978-3-030-63076-8%5C%5C_2.