Datenbank Spektrum (2022) 22:227–239
https://doi.org/10.1007/s13222-022-00429-8

SCHWERPUNKTBEITRAG

Towards a Benchmark for Shared Databases [Vision Paper]

Muhammad El-Hindi[1] · Ashwin Arora[1] · Simon Karrer[1] · Carsten Binnig[1]

Received: 24 August 2022 / Accepted: 24 October 2022 / Published online: 6 December 2022

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 paper, we present a vision and initial ideas for a new benchmark to evaluate shared databases and capture their unique characteristics.

Keywords Shared Databases · DBMS for Data Sharing · Data Sharing · Benchmarking · TPC-C

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 [1–3] in different areas such as supply chain [4–7], healthcare [8] or finance [9, 10]. At the same time, dealing with data is becoming more and more regulated by legislations such as the European Union (EU)'s General Data Protection Regulation (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 [11].

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 database (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. Fig. 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

✉ Muhammad El-Hindi
muhammad.el-hindi@cs.tu-darmstadt.de

Ashwin Arora
ashwin.arora@stud.tu-darmstadt.de

Simon Karrer
skarrer@trustdble.com

Carsten Binnig
carsten.binnig@cs.tu-darmstadt.de

[1] Technical University of Darmstadt, Darmstadt, Germany

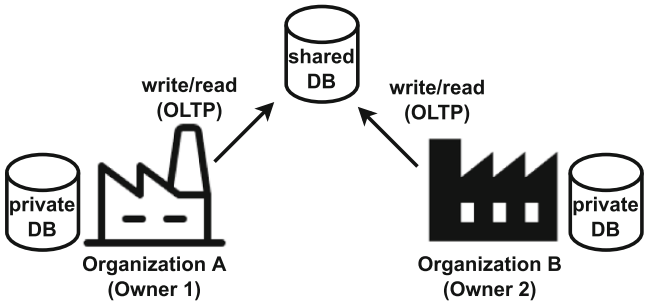

Fig. 1 Shared database concept. Shared DBs enable multiple database owners to collaboratively write and read data to a common database – this workload can be classified as an OLTP workload

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.

With the advent of new technologies such as trusted execution environments (TEE)s and distributed ledger technology (DLT) various systems using different architectures and approaches have been proposed to implement OLTP-style shared databases (e.g., [12–16]). 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 [17], Smallbank [18] or YCSB [19] 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 (Sect. 2).
- Based on that, we then discuss in detail the shortcomings of traditional benchmarks (Sect. 3).
- Finally, we propose our vision and initial ideas for such a new benchmark design for shared databases (Sect. 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.

2 Shared Databases

In this section, we first analyze the unique characteristics and capabilities of shared database systems. Afterwards, in Sect. 3, we present why existing benchmarks are insufficient for evaluating shared databases.

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 database management system (DBMS)s which is about handling multiple isolated databases (i.e., access is restricted to a single organization).

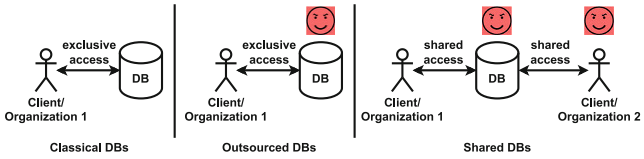

Fig. 2 Shift in trust assumptions in data sharing. In the past, DBs were designed for use by a single, trusted organization (*left*). While the database or the database operator might still be untrusted as in the outsourced DB setting (*middle*), the additional concern of data sharing is to address trust issues among database owners (*right*)

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 Fig. 2, classical databases were built for use by or within a single trusted organization (left). Even with the introduction of cloud computing and outsourced database (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. Fig. 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.

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.

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.

2.2.1 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.

2.2.2 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* [12]. 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 [20] 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.

2.2.3 Data Usage & Compliance

In traditional databases role-based access control (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.

2.2.4 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.

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 [21] 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.

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.

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.

3.3 Private and 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 transaction, we believe that the amount of executed shared transactions is another important dimension that is currently not considered in existing benchmarks.

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. Sect. 2.2) that allows data sharing participants to check whether a system executed a given transaction correctly. As discussed in previous work [22], 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[23] or PBFT[24]. 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 [12].

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 [15] 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.

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.

4.1 Overview

As we discussed previously, shared databases come with new capabilities (cf. Sect. 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 Fig. 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).

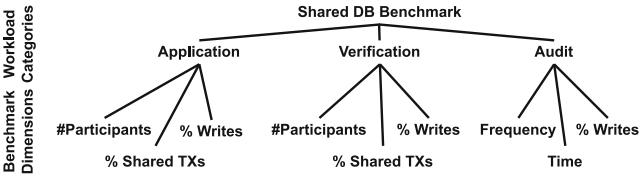

Fig. 3 High-level benchmark design. Our benchmark defines three main workload types (Application, Verification, Audit) with novel benchmark dimensions

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 Fig. 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 Fig. 4 as examples[1]:

Centralized Architecture. In this architecture, multiple organizations access the same shared database platform.

[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.

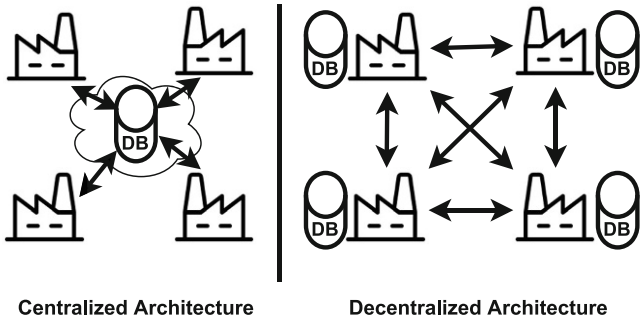

Fig. 4 Architecture stereotypes used as examples. In the centralized architecture (*left*) all participants access the same shared DB platform. The decentralized approach (*right*) removes the need for a central platform. Instead, every organization operates its own DBMS node and additional protocol are in place to keep all nodes in sync. Depending on the chosen architecture we expect systems to exhibit different behaviors in the benchmark

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 [25] or Alibaba's LedgerDB [26].

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 [12] or FalconDB [15].

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 [27]. 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.

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.

4.3.1 Application Workloads

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.

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 Fig. 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

[2] We use TPC-C as a concrete example in this paper, but we envision that other benchmarks can and will be used similarly.

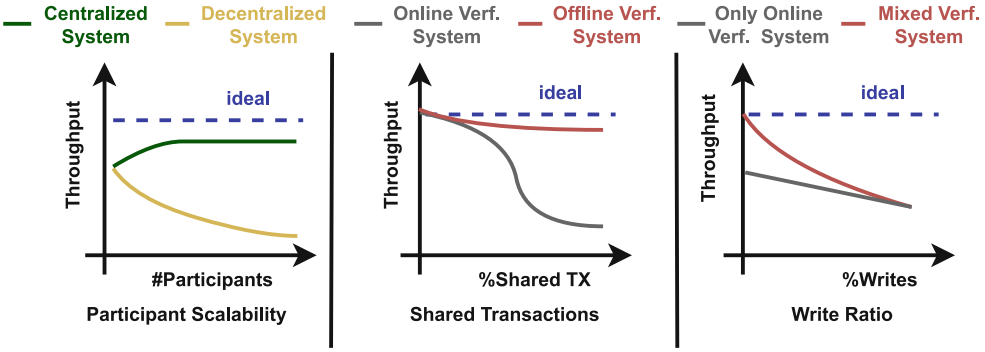

Fig. 5 Application workload to measure the end-to-end system performance. We propose to use the TPC-C benchmark as starting point and re-use its performance metrics (e.g., throughput). However, we introduce new benchmark dimensions (Participant Scalability, Shared Transactions, Write Ratio) to help uncover differences in the performance of different shared DB systems

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 [27] 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 [27].

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 (Fig. 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.

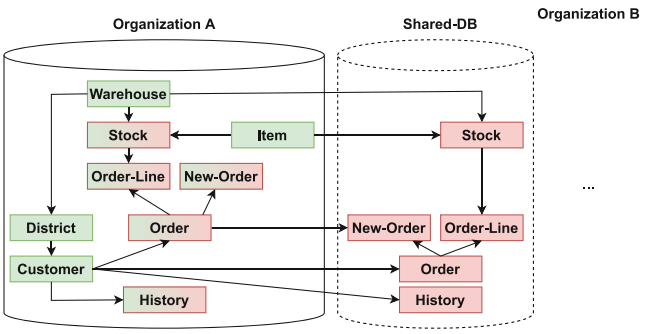

Fig. 6 Adapted TPC-C data model. In our data model, it is assumed that every warehouse belongs to a different organization. Therefore, we modify the TPC-C data model to distinguish between private data (*green*) and shared data (*red*). To support private-only transactions, i.e. orders that do not include remote order-lines, every organization has a private partition of the initially shared tables (*green/red*)

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 Fig. 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 [12], for instance, requires adding a new node to the network which has been shown to reduce the throughput of the system significantly [12] (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 Fig. 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 Fig. 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 Fig. 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.

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 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 Fig. 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 Fig. 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

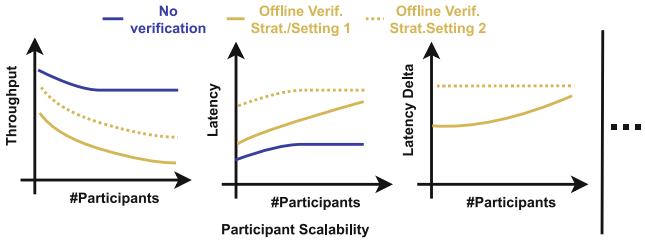

Fig. 7 Verification workloads measure verification effects that are not visible in application workloads. For example, they help to detect differences in the used verification strategies. While the workloads in this category re-use traditional performance metrics (throughput/latency), we make use of our previously described benchmark dimensions (e.g., participant scalability) to uncover performance differences of different systems and verification strategies

in this workload category, as exemplified by the *participant scalability* dimension in Fig. 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.

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., [26] or [28], 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., [25] 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. Fig. 8).

Audit Frequency. As shown in the first plot of Fig. 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,

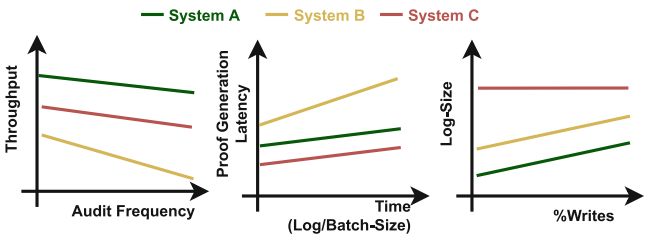

Fig. 8 Instead of measuring the audit performance of the auditor, we propose to analyze the effect of auditing on the system performance. Audit workloads consider different metrics from the perspective of the shared DB, e.g., how transaction throughput is affected by an increasing audit frequency (*left*) or how proof generation latency (*middle*) or log-size (*right*) are influenced

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 Fig. 8). For example, compared to classical verification operations, proof generation can take up to several hours [15]. 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 Fig. 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. Fig. 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.

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.

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 [28], BlockchainDB [13], Veritas [12] use a custom YCSB-like key-value benchmark to evaluate their system end-to-end without considering verification and audit workloads in particular. FalconDB [15] uses the YCSB while ChainfyDB [14] and Basil [29] 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 [30] 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 [21], for instance, is a specialized benchmark for Ledger Databases. It uses Smallbank [18] 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 [31] 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 [32] 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 system, while our focus is on the performance of shared DBs.

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 [33]) is one important area of future work.

Funding Open Access funding enabled and organized by Projekt DEAL.

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.

Conflict of interest 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.

References

1. Publications Office of the European Union European Commission launched the Support Centre for Data Sharing! – data.europa.eu. https://data.europa.eu/en/news/european-commission-launched-support-centre-data-sharing. Accessed 19 Aug 2022
2. European Comission Proposal for a REGULATION OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL on European data governance (Data Governance Act). https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A52020PC0767. Accessed 19 Aug 2022
3. Microsoft News Center Adobe, Microsoft and SAP announce the Open Data Initiative to empower a new generation of customer experiences. https://news.microsoft.com/2018/09/24/adobe-microsoft-and-sap-announce-the-open-data-initiative-to-empower-a-new-generation-of-customer-experiences/. Accessed 19 Aug 2022
4. Ottaviano P (ed) National freight data portal one step closer to reality. businesswire.com. https://www.businesswire.com/news/home/20220310005691/en/National-Freight-Data-Portal-One-Step-Closer-to-Reality. Accessed 19 Aug 2022
5. Port of Long Beach West coast ports support 'supply chain information highway. https://polb.com/port-info/news-and-press/west-coast-ports-support-supply-chain-information-highway-03-03-2022/. Accessed 19 Aug 2022
6. International Air Transport Association ONE Record Homepage. https://www.iata.org/one-record/. Accessed 19 Aug 2022
7. Catena-X Automotive Network e.V. Catena-X Homepage. https://catena-x.net/en/. Accessed 19 Aug 2022
8. Hulsen T (2020) Sharing is caring: data sharing initiatives in healthcare. Int J Environ Res Public Health 17(9):3046. https://doi.org/10.3390/ijerph17093046
9. Consumer Financial Protection Bureau CFPB outlines principles for consumer-authorized financial data sharing and aggregation. https://www.consumerfinance.gov/about-us/newsroom/cfpb-outlines-principles-consumer-authorized-financial-data-sharing-and-aggregation/. Accessed 19 Aug 2022
10. 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. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX. Accessed 19 Aug 2022

11. Eiss R (2020) Confusion over Europe's data-protection law is stalling scientific progress. Nature 584(7822):498–498. https://doi.org/10.1038/d41586-020-02454-7
12. Allen L, Antonopoulos P, Arasu A, Gehrke J, Hammer J, Hunter J et al (2019) Veritas: shared verifiable databases and tables in the cloud. CIDR 2019:1–9. https://www.cidrdb.org/cidr2019/papers/p111-gehrke-cidr19.pdf. Accessed 19 Aug 2022
13. El-Hindi M, Binnig C, Arasu A, Kossmann D, Ramamurthy R (2019) BlockchainDB: a shared database on blockchains. Proc VLDB Endow 12(11):1597–1609. https://doi.org/10.14778/3342263.3342636
14. Schuhknecht FM, Sharma A, Dittrich J, Agrawal D (2021) chainifyDB: How to get rid of your Blockchain and use your DBMS instead. CIDR 2021:1–10. http://cidrdb.org/cidr2021/papers/cidr2021_paper04.pdf. Accessed 19 Aug 2022
15. Peng Y, Du M, Li F, Cheng R, Song D (2020) FalconDB: blockchain-based collaborative database. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data SIGMOD '20. Association for Computing Machinery, Portland, pp 637–652 https://doi.org/10.1145/3318464.3380594
16. El-Hindi M, Karrer S, Doci G, Binnig C (2020) TrustDBle: towards trustable shared databases. In: Third international symposium on foundations and applications of blockchain. virtual conference, pp 1–4 (https://scfab.github.io/2020/FAB2020_p7.pdf)
17. Transaction Processing Performance Council (TPC) TPC BENCHMARK C – standard specification – revision 5.11. Transaction processing performance council. https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-c_v5.11.0.pdf. Accessed 19 Aug 2022
18. Alomari M, Cahill M, Fekete A, Rohm U (2008) The cost of serializability on platforms that use snapshot isolation. In: 2008 IEEE 24th international conference on data engineering, pp 576–585
19. Cooper BF, Silberstein A, Tam E, Ramakrishnan R, Sears R (2010) Benchmarking cloud serving systems with YCSB. In: Proceedings of the 1st ACM symposium on Cloud computing SoCC '10. Association for Computing Machinery, New York, pp 143–154 https://doi.org/10.1145/1807128.1807152
20. Xia Y, Yu X, Butrovich M, Pavlo A, Litmus DS (2022) Towards a practical database management system with verifiable ACID properties and transaction correctness. In: Proceedings of the 2022 international conference on management of data SIGMOD '22. Association for Computing Machinery, New York, pp 1478–1492 https://doi.org/10.1145/3514221.3517851
21. Zhang M, Yue C, Zhu C, Zhong Z (2022) LEDGERBENCH: a framework for benchmarking ledger databases. Bull Tech Comm Data Eng 45(2):11
22. El-Hindi M, Zhao Z, Binnig CACID-V (2021) Towards a new class of DBMss for data sharing. In: Rezig EK, Gadepally V, Mattson T, Stonebraker M, Kraska T, Wang F, al (eds) Heterogeneous data management, polystores, and analytics for healthcare. Lecture notes in computer science. Springer, Cham, pp 60–64
23. Ongaro D, Ousterhout J (2014) In search of an understandable consensus algorithm. In: 2014 USENIX annual technical conference (USENIX ATC 14). USENIX Association, Philadelphia, pp 305–319 (https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro)
24. Castro M, Liskov B (1999) Practical Byzantine Fault Tolerance. In: 3rd Symposium on Operating Systems Design and Implementation (OSDI 99). USENIX Association, New Orleans (https://www.usenix.org/conference/osdi-99/practical-byzantine-fault-tolerance)
25. Antonopoulos P, Kaushik R, Kodavalla H, Rosales Aceves S, Wong R, Anderson J et al (2021) SQL ledger: cryptographically verifiable data in Azure SQL database. In: Proceedings of the 2021 international conference on management of data SIGMOD '21. Virtual Event China: ACM, pp 2437–2449 https://doi.org/10.1145/3448016.3457558
26. Yang X, Zhang Y, Wang S, Yu B, Li F, Li Y et al (2020) LedgerDB: a centralized ledger database for universal audit and verification. Proc VLDB Endow 13(12):3138–3151. https://doi.org/10.14778/3415478.3415540
27. Neto AA, Vieira M (2011) TO BEnchmark or NOT TO BEnchmark security: That is the question. In: 2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W), pp 182–187
28. Zhang M, Xie Z, Yue C, Spitz ZZ (2020) A verifiable database system. Proc VLDB Endow 13(12):3449–3460. https://doi.org/10.14778/3415478.3415567
29. Suri-Payer F, Burke M, Wang Z, Zhang Y, Alvisi L, Crooks NB (2021) Breaking up BFT with ACID (transactions). In: Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles SOSP '21. Association for Computing Machinery, New York, pp 1–17 https://doi.org/10.1145/3477132.3483552
30. Yue C, Dinh TTA, Xie Z, Zhang M, Chen G, Ooi BC, al al Y (eds) GlassDB: practical verifiable ledger database through transparency. arxiv. Number: arxiv:2207.00944. http://arxiv.org/abs/2207.00944. Accessed 19 Aug 2022
31. Dinh TTA, Wang J, Chen G, Liu R, Ooi BC, Tan KL (2017) Blockbench. A framework for analyzing private blockchains. In: Proceedings of the 2017 ACM International Conference on Management of Data SIGMOD '17. Association for Computing Machinery, New York, pp 1085–1100. https://doi.org/10.1145/3035918.3064033
32. Shastri S, Banakar V, Wasserman M, Kumar A, Chidambaram V (2020) Understanding and benchmarking the impact of GDPR on database systems. Proc VLDB Endow 13(7):1064–1077. https://doi.org/10.14778/3384345.3384354
33. Vieira M, Madeira H (2005) Towards a security benchmark for database management systems. In: 2005 International Conference on Dependable Systems and Networks (DSN'05), pp 592–601