Native Distributed Databases: Problems, Challenges and Opportunities

Quanqing Xu
OceanBase, Ant Group
xuquanqing.xqq@oceanbase.com

Chuanhui Yang*
OceanBase, Ant Group
rizhao.ych@oceanbase.com

Aoying Zhou
East China Normal University
ayzhou@dase.ecnu.edu.cn

ABSTRACT

Native distributed databases, crucial for scalable applications, offer transactional and analytical prowess but face data intricacies and network challenges. Under the CAP theorem's constraints, latency and replication issues necessitate creative approaches to maintenance, security, and upgrades. Progress in consistency algorithms, network technology, automation, and machine learning for optimization presents significant potential. Embracing hybrid transactional/analytical processing (HTAP), these databases represent an evolutionary leap in data management, aiming to reconcile performance with the complexities inherent in distributed environments. OceanBase is introduced as a case study, and its strong TPC-C and TPC-H benchmark performances underscore OceanBase as a top-tier distributed database. We also discuss possible opportunities for native distributed databases.

PVLDB Reference Format:
Quanqing Xu, Chuanhui Yang, and Aoying Zhou. Native Distributed Databases: Problems, Challenges and Opportunities. PVLDB, 17(12): 4217 - 4220, 2024.
doi:10.14778/3685800.3685839

1 INTRODUCTION

Native distributed databases are built to scale across interconnected nodes, ensuring high availability and resilience against failures [16]. They use advanced replication algorithms (e.g., Raft [20] and Paxos [13]) for data synchronization while preserving ACID properties, essential for eliminating single points of failure. Such databases, e.g., Google Spanner [1] and OceanBase [30], provide robust solutions for consistency, data partitioning, and management. Their capability to handle heavy data demands makes them vital for businesses operating over distributed networks, offering scalable, fault-tolerant database systems.

They excel in distributed computing, scaling elastically and ensuring data durability with sophisticated replication. Designed around the CAP theorem, they balance consistency, availability, and partition tolerance, making them ideal for extensive, reliable data management across vast computing environments. As they evolve, these databases are incorporating cloud technologies and machine learning to enhance scalability and reliability, adeptly meeting both transactional and analytical needs [16]. They mark a new phase in data management systems, crucial for modern, scalable infrastructure demands, embodying resilience and efficiency.

The proliferation of data and an increased reliance on robust data management systems emphasize the significance of native distributed databases in modern technological ecosystems. They are instrumental in revolutionizing data management by optimizing scalability and availability, benefiting from the agility of cloud technologies and the predictive power of machine learning. By supporting both transactional and analytical processes, these databases signify a shift towards more streamlined, cost-effective, and capable data management solutions, poised to meet the escalating requirements for scalable and fault-tolerant data infrastructures.

We present a 1.5-hour tutorial, which is divided into seven sections as follows: **1) Overview of Native Distributed Database (~5min).** It offers scalable, resilient, and efficient large-scale data management. **2) Data Replication and Synchronization (~15min).** Distributed databases maintain data integrity through advanced replication despite failures. **3) Consistency Models (~15min).** It describes a variety of consistency models, ranging from strict consistency to eventual consistency. **4) Distributed Transactions (~15min).** It discusses distributed transactions by ensuring atomicity, consistency, isolation, and durability (ACID) across multiple nodes in a distributed environment. **5) Query Processing (~15min).** It focuses on query processing by distributing and executing queries efficiently across various nodes, optimizing for reduced network latency and strategic data placement. **6) Case Study: OceanBase (~10min).** It describes that OceanBase offers a high-performance, scalable, shared-nothing architecture, excelling in OLTP/OLAP integration. **7) Opportunities (~15min).** Embracing Serverless architecture [3], AI4DB and DB4AI [17, 35], multi-model [16], and vector database capabilities [9], it presents opportunities for unprecedented scalability, autonomous operation, and versatile data handling in modern computing environments.

Target Audience. The intended audience includes database researchers, developers, and students who aspire to study database kernel techniques, as well as database administrators (DBAs) who desire to better tune their database systems. The tutorial is self-contained and does not require any prerequisite knowledge.

2 TUTORIAL OUTLINE

2.1 Overview of Native Distributed Database

Native distributed databases offer unified systems with high availability, fault tolerance, scalability, and performance across multiple nodes and locations. Key problems of native distributed databases include: **1) Data Replication and Synchronization**: These databases handle synchronization among nodes to keep replicas up-to-date, which is crucial for data accuracy and disaster recovery. **2) Consistency Models**: They offer various consistency models ranging from

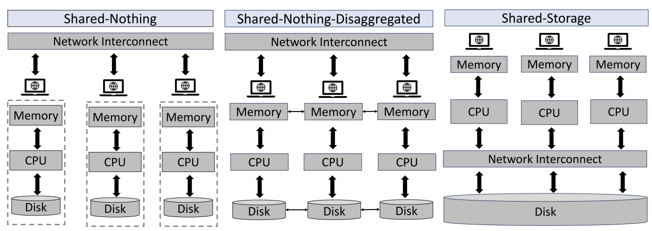
---

*Chuanhui Yang is the corresponding author.

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.

Proceedings of the VLDB Endowment, Vol. 17, No. 12 ISSN 2150-8097.
doi:10.14778/3685800.3685839

strong consistency to eventual consistency, allowing the system to balance between consistency, availability, and partition tolerance as dictated by the CAP theorem. **3) Distributed Transactions**: Support for transactions across multiple nodes is often provided, although it may come with trade-offs in terms of performance and scalability. **4) Query Processing**: They can execute queries across nodes efficiently, often optimizing query execution plans to minimize network traffic and data movement. Table 1 outlines different mechanisms of popular distributed databases, and Figure 1 illustrates three distinct architectures of native distributed databases.



(a) Shared-Nothing (b) Shared-Nothing Disaggr. (c) Shared-Storage

Figure 1: Architectures of Native Distributed Databases

2.2 Data Replication and Synchronization

2.2.1 Data Replication. Replication in distributed databases strikes a balance between synchronous methods, which ensure immediate consistency but result in slower writes, and asynchronous methods, which allow for faster access but may introduce data discrepancies. The level of replication—be it at the row, block, or file level—is chosen based on specific needs. Asynchronous replication resolves conflicts using timestamps or custom logic to ensure data accuracy. Techniques like asymmetric-partition replication can reduce system load [15], whereas solutions like BatchDB [18] improve OLTP/OLAP workloads by pairing logical replication with a lazy strategy for enhanced performance.

2.2.2 Data Synchronization. Distributed databases balance consistency and performance using models like eventual consistency [12], which tolerates short-term discrepancies for assured long-term accuracy. Data sync frequency and network latency [26] are crucial to this consistency, influencing system design for performance optimization. Moreover, to handle simultaneous transactions and data conflicts, strategies such as version control and timestamp [29] help maintain orderly data sync and ensure steadfast consistency.

2.2.3 Challenges. In distributed databases, especially those spanning wide areas, data synchronization is essential but challenged by bandwidth limits, network latency and fluctuation, affecting efficiency and performance [24]. Optimizing bandwidth and maintaining swift recovery post-failure are crucial for data integrity and loss prevention. Ensuring transactional consistency amid partitions and securing data against unauthorized changes during replication are key hurdles. However, as demands for performance rise, evolving technologies are progressively tackling these issues, enhancing the robustness and reliability of distributed database systems.

2.3 Consistency Models

The data consistency models in distributed databases crucially impact performance, reliability, and availability, as depicted in Figure 2.

2.3.1 Strong and Eventual Consistency. Strong consistency in distributed systems ensures that operations are immediately visible and executed in sequence, providing a seamless experience but potentially limiting performance due to the need for node synchronization. Systems such as Calvin [26], PolarDB [28], and GeoGauss [34] have improved transactional efficiency and replication to deliver this consistent state without significantly affecting speed or scalability. In contrast, eventual consistency allows for short-term data anomalies in exchange for better responsiveness, with systems like Dynamo [5] managing high-performance demands through application-level conflict resolution. BlockchainDB [8] innovatively combines the flexibility of databases with the strength of blockchain technology to offer a range of consistency levels, thus optimizing data management for a variety of operational contexts.

2.3.2 Other Consistency Models. Causal consistency improves upon eventual consistency by ensuring causally related operations follow the same order across nodes, while independent operations are not strictly ordered. Efficiency gains come from minimizing dependency checks and defining external causal relationships [2]. GentleRain [7] increases throughput with time-based protocols and uses physical timestamps to save on storage and communication. Orbe [6] leverages dependency matrices and transitive causality for effective causal consistency in key-value systems. Achieving strong consistency in partitioned, replicated systems is challenging. Google Spanner [1] clusters servers and uses Paxos for log replication within groups, maintaining a consistent prefix order across data replicas to uphold its consistency standard.

2.3.3 Challenges. Choosing the right consistency for distributed databases is crucial for balancing performance, availability, and precision. Financial systems often require strong consistency, while CDNs may opt for eventual consistency, accepting brief data discrepancies. The CAP theorem advises a trade-off between consistency, availability, and partition tolerance, influenced by application needs. Databases like OceanBase [32] adapt consistency options for diverse scenarios. Data replication, key for fault tolerance, faces latency issues with synchronous methods and potential inconsistency with asynchronous ones. Developers must navigate these complexities to ensure optimal system performance and data reliability.

2.4 Distributed Transactions

2.4.1 Distributed Transaction Commit Protocols. Native distributed databases employ distributed transaction commit protocols to uphold the ACID properties across distributed nodes, ensuring data integrity and consistency. The 2PC protocol is a classic example, operating in two distinct stages. ROCOCO [19] optimizes this process by treating transactions as collections of atomic blocks, tracking dependencies before execution to allow for serializable ordering upon commit. Primo [11] avoids concurrency conflicts by ensuring transactions are conflict-free after the commit phase. We proposed OceanBase 2PC [30], a Paxos-enhanced 2PC protocol, to strengthen fault tolerance and reduce transaction latency in distributed environments, streamlining synchronizations for efficiency.

2.4.2 Distributed Version Control. One of the core principles of SAP HANA is full support for distributed query capabilities and horizontal expansion [14]. It employs MVCC to provide distributed

Table 1: Different mechanisms of different distributed databases

Architecture	Database	Storage		Transaction		Query	Schedule			Tolerance
		Replica consistency	Global snapshot	Distributed ratio	Concurrency control	Strong Consistency read on replicas	Elasticity computing	Adaptive splitting	Online storage movement	(seconds)
Shared-Nothing	VoltDB [23]	K-safety	✓	Sharding relative	Partition-based	-	✓	✗	✓	RPO=0,RTO<300
	Citus [4]	Master-slave	✗	Sharding relative	MVCC+2PL	✗	✓	✗	✓	-
	OceanBase [30]	Paxos	✓	Sharding relative	MVCC+2PL	✗	✓	✗	✓	RPO=0,RTO<8
	Dynamo [5]	Quorum	✗	No	MVCC+Vector Clocks	✗	✓	✓	✓	RPO<1,RTO~0
	Cassandra [12]	Quorum	✗	No	Lock-free	✗	✓	✓	✓	-
	Calvin [25]	Asynchronous/Paxos	✓	Partition relative	Deterministic locking	✓	-	-	-	-
Shared-Nothing-Disaggregated	Spanner [1]	Paxos	✓	High	MVCC+2PL	✓	✓	✓	✓	RPO=0,RTO~0
	TiDB [10]	Raft	✓	High	Percolator	✓	✓	✓	✓	RPO=0,RTO<60
	CockroachDB [24]	Raft	✓	High	Percolator	✓	✓	✓	✓	RPO=0,RTO<300
	FoundationDB [33]	K-Safety	✓	No	MVCC+OCC	✓	✓	✓	✓	-
Shared-Storage	Aurora [27]	Quorum	✓	No	MVCC+2PL	✗	Read-only	✓	✓	RPO<1,RTO<60
	PolarDB [3]	Raft	✓	No	MVCC+2PL	✓	Read-only	✓	✓	RPO=0,RTO<30
	Google F1 [22]	Synchronous replication	✗	Sharding relative	-	✓	✓	✓	✓	-

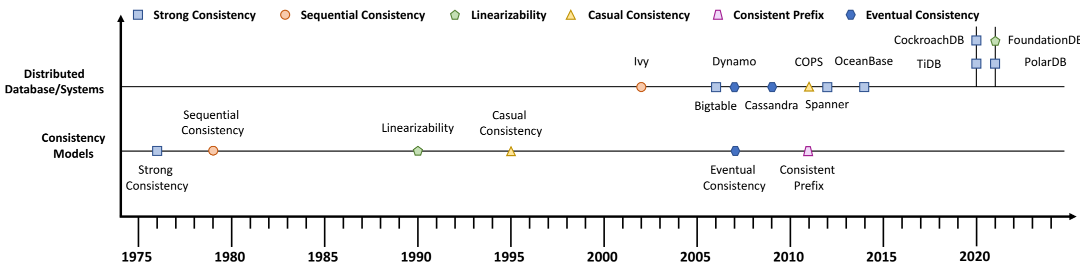


Figure 2: Consistency Models

quick search and distributed locking to synchronize multiple writers. A decentralized scalar timestamp is proposed in [29], without requiring a centralized global timestamp service. It combines MVCC to provide multiple levels of consistency, supports efficient read-only transactions, and has little impact on read-write transactions.

2.4.3 Challenges. Distributed transaction processing contends with network delays, resource locking, deadlocks, and balancing strict ACID compliance with BASE model efficiency. Challenges include fault recovery and improving transaction visibility across networks. Native distributed databases address these issues using distributed locking [31], conflict detection, multi-version concurrency control, and robust recovery protocols. Consensus algorithms like Paxos or Raft are instrumental for node coordination, striking a balance between reliability, high availability, and fault tolerance, ensuring transactions are both dependable and scalable.

2.5 Query Processing

2.5.1 SQL Executor. In native distributed databases, the SQL executor is pivotal, managing SQL statement parsing, planning, optimization, and execution. It adeptly navigates complex queries to preserve data consistency and integrity across distributed nodes. The process starts with parsing SQL into an abstract syntax tree (AST), followed by syntactic and semantic checks. From the AST, a logical plan is derived, outlining the query structure without specifics. The executor then optimizes this plan for efficiency, creating a physical plan with detailed execution methods. This plan is further tailored to the actual database environment for optimal performance. Finally, the refined plan is executed, orchestrating data operations across the network and delivering results [10].

2.5.2 SQL Optimizer. The SQL optimizer in native distributed databases is vital for query efficiency, analyzing queries to devise the best execution path [30]. It constructs logical and physical plans, estimates costs, and refines these plans considering data distribution, replica locations, and network latency. Aimed at minimizing response times and resource use, while ensuring the accuracy of results, the optimizer is continuously evolving to meet the demands of new data models and queries, enhancing the performance and scalability of distributed database systems.

2.5.3 Challenges. The SQL executor in distributed databases plays a crucial role in processes queries, ensuring node-level execution and network-wide consistency. As distributed systems increase in complexity, executors must improve performance, resource management, and error recovery. Meanwhile, the SQL optimizer takes into account data placement, replica locations, and latency, aiming to cut response times and conserve resources while preserving query accuracy. Technological advancements persistently upgrade the optimizer to handle diverse data models and queries, thereby boosting the database's performance and scalability [28].

2.6 Case Study: OceanBase

OceanBase sets a high standard for distributed databases with its LSM-tree storage architecture and Paxos-based two-phase commit transactions, complemented by a robust SQL processing engine. It leverages multitenancy and data compression strategies to scale effectively and operate efficiently. Demonstrating excellence through impressive TPC-C and TPC-H benchmark scores, OceanBase solidifies its position as a top-tier choice in the realm of distributed database solutions, prioritizing performance and scalability.

2.7 Opportunities

2.7.1 Serverless. Integrating cloud elasticity with Serverless architecture [3] simplifies application maintenance, allowing developers to focus on coding. Serverless databases enhance adaptability, scaling automatically to match workloads, which boosts efficiency and reduces hardware maintenance. However, they face challenges like fluctuating performance, maintaining data consistency, and complex debugging due to lack of fixed infrastructure. Concerns over vendor lock-in, security, privacy, and limited features also pose significant hurdles when implementing Serverless databases in sophisticated application environments.

2.7.2 AI4DB and DB4AI. The integration of AI with distributed databases, known as AI4DB and DB4AI [17, 35], enhances both database functionalities and AI efficiency. AI4DB applies machine learning to fine-tune query optimization, predictive storage management, and reliability through anomaly detection, reducing human intervention. Inversely, DB4AI equips AI with strong data support, streamlining data processing for intricate AI operations and easing machine learning workflows. This collaboration advances AI data management and accelerates training and inference, propelling advancements and driving smarter, more effective solutions in diverse fields, thus enhancing innovation and decision-making.

2.7.3 Multi-Model Database. Native distributed databases offer multi-model support, combining key-value, document, and graph data types within one platform [16], optimizing the selection of data models for specific tasks. This consolidation streamlines infrastructure, improves performance, and boosts query efficiency. Key challenges include upholding cross-model consistency, distributed ACID properties, and executing complex multi-type queries while balancing storage performance and ensuring robust security. As they mature, these flexible multi-model databases become essential in enterprises, harnessing the strengths of varied data structures.

2.7.4 Vector Database. Vector databases, specializing in similarity searches, are pivotal in various fields, improving data storage and retrieval in distributed systems to meet contemporary application requirements [21]. These databases, known for fault tolerance and high availability, enhance vector data services' reliability. Yet, merging vector and distributed databases introduces challenges such as optimizing query speed, managing high-dimensional vector storage, and allocating resources efficiently for computation-intensive tasks. These aspects are vital for sustaining a performant and sturdy infrastructure, addressing the nuanced demands of integrating vector database capabilities with the robustness of distributed systems.

3 PRESENTERS

Quanqing Xu is a technical director of OceanBase Lab, Ant Group. His research interests primarily include distributed database systems and storage systems. He is a Fellow of the IET, a distinguished member of CCF, a senior member of ACM and IEEE.

Chuanhui Yang is the CTO of OceanBase, Ant Group. His research focuses on database and distributed systems. As one of the founding members, he led the previous architecture design, and technology research and development of OceanBase, realizing the full implementation of OceanBase in Ant Group from scratch.

Aoying Zhou is a full professor at School of Data Science and Engineering, East China Normal University. His main research interests span databases, data management, digital transformation, data-driven applications. He is a fellow of CCF.

REFERENCES

[1] David F. Bacon, Nathan Bales, Nicolas Bruno, et al. 2017. Spanner: Becoming a SQL System. In *SIGMOD*. ACM, 331–343.

[2] Peter Bailis, Alan D. Fekete, Ali Ghodsi, et al. 2012. The potential dangers of causal consistency and an explicit solution. In *SOCC*. ACM, 1–7.

[3] Wei Cao et al. 2021. PolarDB Serverless: A Cloud Native Database for Disaggregated Data Centers. In *SIGMOD*. ACM, 2477–2489.

[4] Umur Cubukcu et al. 2021. Citus: Distributed PostgreSQL for Data-Intensive Applications. In *SIGMOD*. ACM, 2490–2502.

[5] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, et al. 2007. Dynamo: amazon's highly available key-value store. In *SOSP*. 205–220.

[6] Jiaqing Du, Sameh Elnikety, et al. 2013. Orbe: scalable causal consistency using dependency matrices and physical clocks. In *SOCC*. ACM, 1–14.

[7] Jiaqing Du, Calin Iorgulescu, et al. 2014. GentleRain: Cheap and Scalable Causal Consistency with Physical Clocks. In *SOCC*. ACM, 4:1–4:13.

[8] Muhammad El-Hindi, Carsten Binnig, Arvind Arasu, et al. 2019. BlockchainDB - A Shared Database on Blockchains. *Proc. VLDB Endow.* 12, 11 (2019), 1597–1609.

[9] Rentong Guo, Xiaofan Luan, et al. 2022. Manu: A Cloud Native Vector Database Management System. *Proc. VLDB Endow.* 15, 12 (2022), 3548–3561.

[10] Dongxu Huang, Qi Liu, Qiu Cui, et al. 2020. TiDB: A Raft-based HTAP Database. *Proc. VLDB Endow.* 13, 12 (2020), 3072–3084.

[11] Ziliang Lai, Hua Fan, Wenchao Zhou, et al. 2023. Knock Out 2PC with Practicality Intact: a High-performance and General Distributed Transaction Protocol. In *ICDE*. IEEE, 2317–2331.

[12] Avinash Lakshman and Prashant Malik. 2010. Cassandra: a decentralized structured storage system. *ACM SIGOPS Oper. Syst. Rev.* 44, 2 (2010), 35–40.

[13] Leslie Lamport. 1998. The Part-Time Parliament. *ACM Trans. Comput. Syst.* 16, 2 (1998), 133–169.

[14] Juchang Lee et al. 2013. SAP HANA distributed in-memory database system: Transaction, session, and metadata management. In *ICDE*. IEEE, 1165–1173.

[15] Juchang Lee et al. 2020. Asymmetric-Partition Replication for Highly Scalable Distributed Transaction Processing in Practice. *Proc. VLDB Endow.* 13, 12 (2020), 3112–3124.

[16] Feifei Li. 2019. Cloud native database systems at Alibaba: Opportunities and Challenges. *Proc. VLDB Endow.* 12, 12 (2019), 2263–2272.

[17] Guoliang Li et al. 2021. AI Meets Database: AI4DB and DB4AI. In *SIGMOD '21*. ACM, 2859–2866.

[18] Darko Makreshanski et al. 2017. BatchDB: Efficient Isolated Execution of Hybrid OLTP+OLAP Workloads for Interactive Applications. In *SIGMOD*. ACM, 37–50.

[19] Shuai Mu, Yang Cui, Yang Zhang, et al. 2014. Extracting More Concurrency from Distributed Transactions. In *USENIX OSDI '14*. USENIX Association, 479–494.

[20] Diego Ongaro and John K. Ousterhout. 2014. In Search of an Understandable Consensus Algorithm. In *USENIX Annual Technical Conference*. 305–319.

[21] James Jie Pan, Jianguo Wang, and Guoliang Li. 2023. Survey of Vector Database Management Systems. *CoRR* abs/2310.14021 (2023).

[22] Jeff Shute et al. 2013. F1: A Distributed SQL Database That Scales. *Proc. VLDB Endow.* 6, 11 (2013), 1068–1079.

[23] Michael Stonebraker and Ariel Weisberg. 2013. The VoltDB Main Memory DBMS. *IEEE Data Eng. Bull.* 36, 2 (2013), 21–27.

[24] Rebecca Taft et al. 2020. CockroachDB: The Resilient Geo-Distributed SQL Database. In *SIGMOD*. ACM, 1493–1509.

[25] Alexander Thomson et al. 2012. Calvin: fast distributed transactions for partitioned database systems. In *SIGMOD*. ACM, 1–12.

[26] Alexander Thomson et al. 2014. Fast Distributed Transactions and Strongly Consistent Replication for OLTP Database Systems. *ACM Trans. Database Syst.* 39, 2 (2014), 11:1–11:39.

[27] Alexandre Verbitski et al. 2017. Amazon Aurora: Design Considerations for High Throughput Cloud-Native Relational Databases. In *SIGMOD*. ACM, 1041–1052.

[28] Jianying Wang et al. 2023. PolarDB-IMCI: A Cloud-Native HTAP Database System at Alibaba. *Proc. ACM Manag. Data* 1, 2 (2023), 199:1–199:25.

[29] Xingda Wei et al. 2021. Unifying Timestamp with Transaction Ordering for MVCC with Decentralized Scalar Timestamp. In *NSDI*. USENIX, 357–372.

[30] Zhenkun Yang et al. 2022. OceanBase: A 707 Million tpmC Distributed Relational Database System. *Proceedings of the VLDB Endowment* 15, 12 (2022), 3385–3397.

[31] Zhenkun Yang et al. 2023. LCL: A Lock Chain Length-based Distributed Algorithm for Deadlock Detection and Resolution. In *ICDE*. IEEE, 151–163.

[32] Zhifeng Yang, Quanqing Xu, Shanyan Gao, et al. 2023. OceanBase Paetica: A Hybrid Shared-nothing/Shared-everything Database for Supporting Single Machine and Distributed Cluster. *Proc. VLDB Endow.* 16, 12 (2023), 3728–3740.

[33] Jingyu Zhou et al. 2021. FoundationDB: A Distributed Unbundled Transactional Key Value Store. In *SIGMOD*. ACM, 2653–2666.

[34] Weixing Zhou et al. 2023. GeoGauss: Strongly Consistent and Light-Coordinated OLTP for Geo-Replicated SQL Database. *Proc. ACM Manag. Data* 1, 1 (2023), 62:1–62:27.

[35] Rong Zhu et al. 2024. PilotScope: Steering Databases with Machine Learning Drivers. *Proceedings of the VLDB Endowment* 17 (2024), 980–993.