A Creativity Survey of Distributed Database System

Authors: JIAHAO FU, WEI LI

Date: 2022-03-10T18:16:36+00:00

Abstract

Distributed database systems are widely used due to the rapid development of the Internet. With ever-increasing demands, boosting performance and minimizing resource and data contention have become critical considerations. An effective distributed physical design, which determines data placement, replication, and partitioning strategies, can significantly improve system performance. This paper classifies the evolution of physical design based on Michael' s work and its references, according to research problems, research methodologies, and measurement approaches. Finally, we propose several directions for future research.

Full Text

Preamble

A Survey of Distributed Database Systems

JIAHAO FU
Institute of Computing Technology, Chinese Academy of Sciences
fujiahao211@mails.ucas.ac.cn

WEI LI
Institute of Computing Technology, Chinese Academy of Sciences
liwei@ict.ac.cn

Abstract: Distributed database systems have become widely used due to the rapid development of the Internet. With ever-increasing demands, improving performance and minimizing resource and data contention have become critical considerations. An effective distributed physical design, which determines data placement, replication, and partitioning strategies, can significantly enhance system performance. This paper classifies the evolution of physical design based on

Michael' s work and related references, examining research problems, methodologies, and measurement approaches. Finally, we propose several directions for future research.

Keywords: Distributed Database System

1 Introduction

As traditional database technology matures alongside rapid advancements in computer networking and expanding application scopes, database applications have become widely deployed across networked environments. However, centralized database systems exhibit several shortcomings: data is naturally distributed across networks based on practical needs, yet centralized processing incurs substantial communication overhead; application concentration on a single computer creates a single point of failure that affects the entire system, yielding low reliability; and system scale and configuration lack flexibility, resulting in poor scalability. Consequently, database systems with distributed characteristics have attracted significant research attention. Distributed databases represent the convergence of database and network technologies, forming an important branch within the database field. These systems store and manage massive datasets, replicate and partition data, and distribute transactions across multiple nodes. The selection of data replication and partitioning solutions through distributed physical design can substantially boost performance while minimizing resource and data contention. We conducted a comprehensive survey of distributed database systems to summarize existing work and identify promising future research directions.

The remainder of this paper is organized as follows. Section 2 presents our classification of database system research objects. Section 3 introduces the classification of research methods. Section 4 compares experimental analyses across related literature. Section 5 discusses future research opportunities, and Section 6 concludes the paper.

2 Classification of Research Objects

The physical design of distributed database systems is closely related to data control. Additionally, certain data management tasks remain relevant in centralized database systems. Therefore, this section employs two independent criteria to classify research objects: (1) **Database System** type (Distributed or Centralized), representing the maximum dimension of classification, and (2) **Data Management** approach (Partition, Replication, Master, or Index). Some methods from centralized systems can be adapted for distributed systems, creating intersections between these categories.

In distributed systems, these four data management approaches are intuitive: partitioning distributes data fragments across different databases; replication stores identical data simultaneously in multiple databases; mastering provides

authoritative data management to resolve or prevent conflicts; and indexing accelerates data retrieval. These techniques also apply to backup and recovery in centralized systems and other use cases.

Based on these criteria, we present the classification in Table 1.

Table 1: Classification of Research Objects

Database System	Partition	Replication	Master	Index
Distributed	Type I: [1][4][7][9][11][13][14][15][16][17]	Type II: [3][4][8][10][12][18][19]	Type III: [2][3]	Type IV: [11]
Centralized	Type V: [11]	Type VI: [5][6]	Type VII: –	Type VIII: [11]

Type Definitions: - **Type I:** Data partitioning in distributed database systems - **Type II:** Data replication in distributed database systems - **Type III:** Data mastering in distributed database systems - **Type IV:** Data indexing in distributed database systems - **Type V:** Data partitioning in centralized database systems - **Type VI:** Data replication in centralized database systems - **Type VII:** Data mastering in centralized database systems - **Type VIII:** Data indexing in centralized database systems

2.3 Explanation of Different Types

Type I (Distributed Partitioning): References [4][7][9][13][14][15][16][17] address distributed partitioning. Reference [4] uses workload to determine replication strategy. Reference [7] focuses on fine-grained configuration for partitioned main memory databases. Reference [9] considers distributed joins. Reference [13] proposes query-centric partitioning for partially replicated systems. Reference [14] introduces partitioning for general database schemas. Reference [15] targets ad-hoc query workloads. Reference [16] addresses distributed transaction processing systems. Reference [17] examines shared-everything OLTP systems.

Type II (Distributed Replication): References [3][4][8][10][12][18][19] cover distributed replication. Reference [3] focuses on partitioned snapshot isolation databases. Reference [4] addresses both replication and partitioning based on workload. Reference [8] identifies dangers of replication and proposes solutions. Reference [10] addresses transaction scaling. Reference [12] works with partially replicated databases. Reference [18] provides a lower bound on dynamic replication algorithm performance. Reference [19] presents an update-everywhere replication approach for PostgreSQL based on snapshot isolation.

Type III (Distributed Mastering): References [2][3] address distributed mastering. Reference [2] proposes DynaMast, which dynamically transfers data

mastership among sites using a lightweight metadata-based protocol. Reference [3] proposes multi-master replication.

Types I, II, and III (Combined): Reference [1] uses historical workload to train a cost model that makes compound decisions on partitioning, replication, and mastering.

Types I, IV, V, and VIII (Partitioning and Indexing): Reference [11] focuses on database partitioning and indexing, proposing an optimization method that improves query time based on user query history patterns. It uses virtual partitioning to access raw data and employs linear programming and greedy algorithms to optimize the cost model.

Types II and VI (Replication): References [5][6] propose lazy database replication algorithms. Reference [5] guarantees ordering, while Reference [6] guarantees snapshot isolation.

Type VII (Centralized Mastering): No references address this category. In centralized systems, identical data across databases is considered backup, so data conflicts do not occur. The primary concern is enabling backup databases during fatal errors, which has received minimal research attention.

3 Classification of Research Methods

Database systems require configuration before deployment. Machine learning can sometimes generate superior configurations automatically. This section classifies research methods using two criteria: (1) **Use of Machine Learning Model** (Yes or No), and (2) **Configuration Method** (Dynamic or Static). While static configuration is straightforward, dynamic configuration can better adapt to non-static systems.

Table 2: Classification of Research Methods

Method of Configuration	Using ML Model	Not Using ML Model
Dynamic	Type I: [1]	Type II: [2][4][7][9][11][13][14][15][16][17][18]
Static	Type III: –	Type IV: [3][5][6][8][10][12][19]

Type Definitions: - **Type I:** Uses machine learning models with dynamic configuration - **Type II:** Does not use machine learning models but employs dynamic configuration - **Type III:** Uses machine learning models with static configuration - **Type IV:** Does not use machine learning models and uses static configuration

Type I: Reference [1] uses linear regression models that consume input vectors and output scalar predictions of operation latency.

Type II: References [2][4][7][9][11][13][14][15][16][17][18] employ dynamic configuration without ML. Reference [2] supports adaptive dynamic mastering with multi-mastering capabilities. Reference [4] proposes a graph-based strategy organized around databases and workload. Reference [7] enables fine-grained live reconfiguration without server shutdown. Reference [9] proposes adaptive partitioning to reduce distributed join costs. Reference [11] optimizes query time based on user query history patterns using virtual partitioning, linear programming, and greedy algorithms. Reference [13] proposes a workload-aware, scalable analytical model using linear programming. Reference [14] uses an elastic algorithm based on Heat Graph. Reference [15] creates and maintains a partition tree based on user queries for adaptive, robust partitioning. Reference [16] proposes the E-Store elastic partitioning framework with E-Monitor for hotspot tracking and E-Planner for heuristic data movement. Reference [17] uses logical partitioning and MBR-Trees based on workload. Reference [18] provides a performance lower bound for any dynamic replication algorithm.

Type III: No references belong to this category.

Type IV: References [3][5][6][8][10][12][19] downplay or do not focus on database configuration.

4 Review of Experimental Analysis

This section classifies evaluation metrics and system parameters, as shown in Table 3. The table categorizes experimental analyses by metrics and parameters, revealing that most references compare throughput, delay, and response time.

Table 3: Experimental Metrics and Parameters

Category	Subcategory	References
Parameters	Hardware	[1][3][5][6][10][12][13][17]
	Software	[1][2][3][4][5][6][7][10][12][13][14][16][17][19]
Metrics	Throughput	[1][3][5][6][10][12][13][17]
	Delay	[12]
	Response Time	[1][2][3][5][6][7][10][12][13][14][16]
	Other	[4][8][9][10][11][13][14][15][17][18][19]

Metric Definitions: - **Throughput:** The number of database transactions per unit time. The formula is:
Throughput = Total Transaction Number - **Delay:** Request response time within the database system - **Response Time:** Time from when a client request begins until it receives a response - **Other Metrics:** Include network delay, distributed transaction ratio, cost, etc.

System Parameters: - **Hardware:** Experimental hardware environment, such as network topology, number of clients or servers - **Software:** Experimen-

tal software configuration, such as configuration variables, algorithm parameters, or experimental datasets

Experimental Comparisons: Most studies evaluate throughput and response time across varying numbers of clients. Reference [1] additionally assesses model accuracy and cost. Reference [2] compares throughput and delay across different client counts. Reference [3] measures throughput, additional delay, and read-only transaction response time. Reference [4] examines distributed transaction ratios across partition numbers and graph sizes. References [5][6] evaluate throughput and response times for read-only and read-write transactions under different client loads. Reference [7] compares throughput and delay during and after reconfiguration. Reference [8] contains no experiments. Reference [9] compares running time and buffer sizes across different data volumes. Reference [10] measures throughput and algorithm state transition costs. Reference [11] compares memory costs and query times for different queries. Reference [12] evaluates response time, transaction commit ratio, and local transaction ratio. Reference [13] compares throughput and average response time across varying numbers of servers, requests, and backends. Reference [14] compares throughput, delay, and distributed transaction ratio. Reference [15] measures upfront overhead and first query runtime. Reference [16] compares throughput and delay across various algorithms. Reference [17] compares throughput, transaction execution time, and algorithm time/space overhead. Reference [18] compares average communication cost savings across different read/write modes. Reference [19] compares response time, throughput, replica count, and replication cost.

5 Discussion and Suggestion

This survey identifies several underexplored areas in distributed database systems that merit future research attention:

1. **Static Configuration via Machine Learning:** While dynamic configuration has been widely studied, using machine learning models to generate optimal static configurations as initial system states remains largely unexplored. Given the rapid advancement of AI, ML models could help make complex configuration decisions more efficiently.
2. **Dynamic Index Management:** Database indexes directly impact query execution speed, but their effectiveness after numerous update transactions is uncertain. Investigating whether dynamic index modification can accelerate queries in such scenarios represents a valuable research direction.

6 Conclusions

Distributed database systems offer substantial opportunities for future improvement. This survey builds upon M. Abebe' s work and related references to

classify research objects, methods, and metrics into distinct categories. While some areas have been extensively studied, others remain underexplored or undervalued. Our classification reveals promising research directions with strong potential for impact.

References

[1] M. Abebe, B. Glasbergen, and K. Daudjee, MorphoSys: automatic physical design metamorphosis for distributed database systems, VLDB, 2021.
[2] M. Abebe, B. Glasbergen, and K. Daudjee. DynaMast: Adaptive dynamic mastering for replicated systems. In IEEE 36th International Conference on Data Engineering (ICDE), pages 1381–1392. IEEE, 2020.
[3] P. Chairunnanda, K. Daudjee, and M. T. Ozsu. Confluxdb: Multi-master replication for partitioned snapshot isolation databases. PVLDB, 7(11):948–958, 2014.
[4] C. Curino, E. Jones, Y. Zhang, and S. Madden. Schism: a workload-driven approach to database replication and partitioning. PVLDB, 3(1-2):48–57, 2010.
[5] K. Daudjee and K. Salem. Lazy database replication with ordering guarantees. In IEEE 20th International Conference on Data Engineering (ICDE), pages 424–435. IEEE, 2004.
[6] K. Daudjee and K. Salem. Lazy database replication with snapshot isolation. In Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB), pages 715–726, 2006.
[7] A. J. Elmore, V. Arora, R. Taft, A. Pavlo, D. Agrawal, and A. El Abbadi. Squall: Fine-grained live reconfiguration for partitioned main memory databases. In Proceedings of the 2015 ACM International Conference on Management of Data (SIGMOD), pages 299–313. ACM, 2015.
[8] J. Gray, P. Helland, P. O' Neil, and D. Shasha. The dangers of replication and a solution. ACM SIGMOD Record, 25(2):173–182, 1996.
[9] Y. Lu, A. Shanbhag, A. Jindal, and S. Madden. Adaptdb: adaptive partitioning for distributed joins. PVLDB, 10(5):589–600, 2017.
[10] Y. Lu, X. Yu, and S. Madden. Star: Scaling transactions through asymmetric replication. PVLDB, 12(11):1316–1329, 2019.
[11] M. Olma, M. Karpathiotakis, I. Alagiannis, M. Athanassoulis, and A. Ailamaki. Slalom: Coasting through raw data via adaptive partitioning and indexing. PVLDB, 10(10):1106–1117, 2017.
[12] V. Padhye, G. Rajappan, and A. Tripathi. Transaction management using causal snapshot isolation in partially replicated databases. In IEEE 33rd International Symposium on Reliable Distributed Systems (SRDS), pages 105–114. IEEE, 2014.
[13] T. Rabl and H.-A. Jacobsen. Query centric partitioning and allocation for partially replicated database systems. In Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD), pages 315–330. ACM, 2017.
[14] M. Serafini, R. Taft, A. J. Elmore, A. Pavlo, A. Aboulnaga, and M. Stonebraker. Clay: fine-grained adaptive partitioning for general database schemas.

PVLDB, 10(4):445–456, 2016.
[15] A. Shanbhag, A. Jindal, S. Madden, J. Quiane, and A. J. Elmore. A robust partitioning scheme for ad-hoc query workloads. In Proceedings of the 2017 Symposium on Cloud Computing (SoCC), pages 229–241. ACM, 2017.
[16] R. Taft, E. Mansour, M. Serafini, J. Duggan, A. J. Elmore, A. Aboulnaga, A. Pavlo, and M. Stonebraker. E-store: Fine-grained elastic partitioning for distributed transaction processing systems. PVLDB, 8(3):245–256, 2014.
[17] P. Tözün, I. Pandis, R. Johnson, and A. Ailamaki. Scalable and dynamically balanced shared-everything oltp with physiological partitioning. The VLDB Journal—The International Journal on Very Large Data Bases (VLDBJ), 22(2):151–175, 2013.
[18] O. Wolfson, S. Jajodia, and Y. Huang. An adaptive data replication algorithm. ACM Transactions on Database Systems (TODS), 22(2):255–314, 1997.
[19] S. Wu and B. Kemme. Postgres-r (si): Combining replica control with concurrency control based on snapshot isolation. In IEEE 21st International Conference on Data Engineering (ICDE), pages 422–433. IEEE, 2005.

Note: Figure translations are in progress. See original paper for figures.