Cloud-Native Databases: A Survey

Haowen Dong ⓘ, Chao Zhang ⓘ, Guoliang Li ⓘ, *Fellow, IEEE*, and Huanchen Zhang ⓘ

(Survey Paper)

Abstract—**Cloud databases have been widely accepted and deployed due to their unique advantages, such as high elasticity, high availability, and low cost. Many new techniques, such as compute-storage disaggregation and the log is the database, have been proposed recently to seek for higher elasticity and lower cost. To better harness the power of cloud databases, it is crucial to study and compare the pros and cons of their key techniques. In this paper, we offer a comprehensive survey of cloud-native databases. Particularly, we investigate and summarize the state-of-the-art cloud-native OLTP and OLAP databases, respectively. In the first part, we discuss three types of architectures of cloud-native OLTP database. Then we introduce their key techniques including data placement strategy, storage layer consistency, compute layer consistency, multi-layer recovery, and HTAP optimization. In the second part, we present two kinds of architectures of cloud-native OLAP databases. Then we take a deep dive into their key techniques regarding storage management, query processing, serverless computing, data protection, and machine learning in databases. Finally, we discuss the research challenges and opportunities.**

Index Terms—**Cloud-native databases, database architecture, disaggregation, log is data, serverless.**

I. INTRODUCTION

TRADITIONAL database vendors provide service-level objective (SLO), e.g., 99.99% high availability, and signs Service Level Agreement (SLA) with database customers. Nowadays, cloud database vendors [12], [22], [25], [27], [94] are increasingly proliferating because of their better SLOs, such as high elasticity, high availability, and cost-efficient services [1], [50], [75]. As a result, many on-premise databases are moving toward cloud data service.

Both customers and cloud vendors can benefit from cloud databases. From the perspective of customers, cloud databases own four main advantages as follows.

1) *Elasticity:* The workloads of the cloud customers usually change periodically (e.g., peaks and valleys), and the cloud customers do not need to worry about the computing resources and the cloud databases can dynamically schedule the resources by benefiting from the underlying cloud services.

2) *Availability:* The cloud customers have high-availability requirements to tolerate computing-server failures and data-center failures. Cloud databases maintain multiple replicas to guarantee high availability. Besides, the cross-region deployment of the data center ensures quick recovery from extreme disasters such as earthquakes and power outages.

3) *Flexibility:* The cloud customers do not want to maintain the hardware and software, and the out-of-box feature of the cloud databases eases the burden of the complicated deployment process. Moreover, the automated management service reduces customers' operation and maintenance costs.

4) *Low Price:* The customers only want to pay for the on-demand resources and service costs rather than the provisioned cost in a fixed period. Cloud databases adopt the pay-as-you-go pricing model to enable this.

In terms of cloud vendors, cloud databases also bring three merits as follows.

1) *Expanded Market Scale:* Due to high maintenance cost of on-premise databases, many small businesses and individuals who lack professional maintenance skills or teams cannot use databases. Due to the out-of-the-box flexibility of cloud databases, small companies can use cloud databases, and thus expand the market scale.

2) *Reduced Unit Cost:* Thanks to the large-scale data centers, it realizes the scale effect and reduces the unit cost by sharing the resources among the users. The operation and maintenance cost is reduced by benefiting from the scale effect.

3) *Improved Resource Utilization:* When using on-premise databases, the hardware resources are bounded to their customers, leaving the resources to be idle when it comes to a fluctuating workload. With cloud databases, the systems will dynamically allocate resources to different users according to their workload status, which improves resource utilization.

The development of cloud databases can be divided into two stages: 1) the stage of cloud-hosting databases and 2) the stage of the cloud-native databases.

At the stage of cloud-hosting databases, customers can choose the offered data service by the cloud vendors (i.e., databases as a service (DBaaS)), then pay for the on-demand resource fee based on the service level agreement (SLA) [65], [67]. However, those providers regard the deployed databases as a general kind of software without any underlying optimizations, and customers
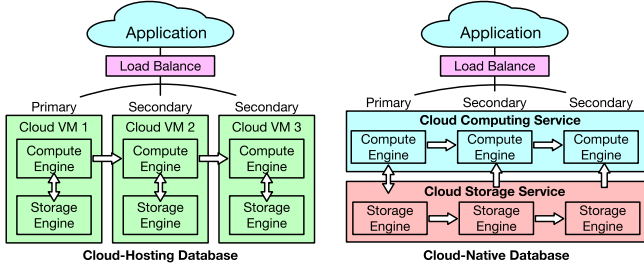
Fig. 1. A comparison of cloud-hosting architecture and cloud-native architecture.

Fig. 2. An overview of cloud-native databases.

must provision the resources and tune the database performance on their own. Moreover, the elastic scheduling capability of cloud services cannot be fully utilized as the resources are scheduled at an instance level.

Cloud-native databases are proposed to improve the elasticity and reduce the cost of cloud-hosting databases. The foremost innovation is the *disaggregation of compute and storage* architecture [94], [96], which decouples the storage from the compute nodes, then connects the compute nodes to shared cloud storage through a high-speed network. On the one hand, the disaggregation architecture enables customers to scale the compute and storage resources independently, thereby bringing more elasticity for the customers. On the other hand, providers can alleviate the write amplification problem by only writing the log (without writing dirty pages) to the storage layer and the dirty pages are replayed based on the log in the storage layer (i.e., the log is the database).

As shown in Fig. 1, compared with cloud-hosting architectures, the computing and storage resources coupled in the virtual machine can be split to achieve independent expansion, which improves the elasticity, availability, and efficiency of the system.

Since cloud applications have different types of workloads, e.g., write-heavy or read-heavy, there has emerged two types of cloud-native databases: 1) cloud-native online transaction processing (OLTP) databases, and 2) cloud-native online analytical processing (OLAP) databases. Both types of databases adopt the disaggregation architecture, but they own disparate techniques and face different challenges. In summary, there are five main challenges that need to address, including log-based transaction processing, multi-layer data consistency, failure recovery, cache-based query processing, and serverless computing.

Challenge 1. Log-based Transaction Processing: Since the storage is disaggregated, it is challenging to support efficient transaction processing based on the cloud storage. As the log becomes the first-citizen, it is rather hard to handle the cache miss when the log has yet to be replayed.

Challenge 2. Multi-Layer Data Consistency: Cloud-native OLTP databases focus on processing transactions in the cloud. However, the main challenge is to ensure the data consistency in the multiple layers, e.g., the compute layer, the storage layer, or even the memory layer.

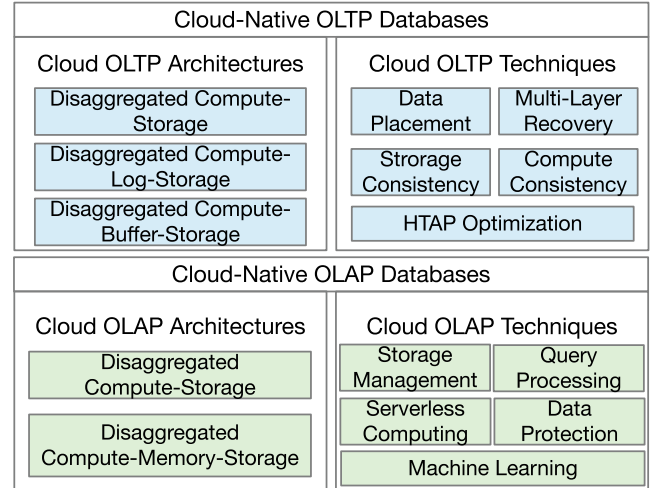*Challenge 3. Failure Recovery:* For the cloud-native databases, it is more complex to provide high availability as each layer may occur exceptions. Thus, a major concern is how to quickly recover the databases when facing compute/storage node failures.

Challenge 4. Cache-based Query Processing: Cloud-native OLAP databases target at scalable query processing with a remote cloud storage. To reduce the network traffic, they need to design effective caching strategies and computational pushdown on the storage side. However, finding an optimal yet cost-efficient query plan is challenging due to the trade-off between performance and cost.

Challenge 5. Serverless Computing: Many cloud databases have supported *serverless computing* that can dynamically schedule resources for users' workloads with the pause-and-resume policy, but it is still challenging to adaptively schedule the resources for the workloads in a query granularity [83] as the resources are provisioned in the instance level.

Fig. 2 presents an overview of key techniques of cloud-native databases. In this survey, we introduce the state-of-the-art techniques of cloud-native OLTP and OLAP databases, respectively. We introduce each type of cloud-native database from two aspects. First, we introduce a taxonomy of their disaggregated architectures. Then we present the representatives for each category. Second, we take a deep dive into their key techniques regarding OLTP and OLAP workloads. We summarize how existing approaches address the above-mentioned challenges.

A. Cloud-Native OLTP Databases

1) Cloud-Native OLTP Architectures: Cloud-native OLTP databases emphasize concurrency and low latency in transaction processing. The architecture design needs to consider the consistency of the primary and secondary nodes, the durability and availability of the storage layer, and the efficiency of query processing. We classify the architectures of cloud-native OLTP databases into three categories as follows:

1) Disaggregated Compute-Storage OLTP Architecture: The first category has a two-layer architecture, where the compute layer processes the transactions on volatile devices, and the

storage layer maintains the data's durability and availability based on the cloud storage service.

2) Disaggregated Compute-Log-Storage OLTP Architecture: The second category separates the data durability and availability management by physically splitting the log storage and page storage.

3) Disaggregated Compute-Buffer-Storage OLTP Architecture: The third category adds a shared buffer layer, which aims to improve the efficiency of data synchronization among computing nodes and reduce the average latency of reading data from the storage layer.

2) Cloud-Native OLTP Techniques: According to the functional modules of the OLTP techniques, we categorize them into five types:

1) Data Placement Strategy: Data placement strategy considers organization of logs and data in the disaggregated architecture. We introduce two types of data placement strategies that organize the logs and pages in the cloud. The first type is i) coupled log-page strategy [94]. The second type is ii) disaggregated log-page strategy [12].

2) Storage Layer Consistency: The storage layer needs to maintain multiple data replicas to ensure high availability, which requires the consistency of these replicas. We introduce two types of storage layer consistency. The first type is i) quorum-based consistency protocol [94]. The second type is ii) Paxos-based consistency protocol [22].

3) Compute Layer Consistency: Computing layer consistency refers to the method of updates synchronization from the primary nodes to secondary nodes. We introduce three ways to maintain consistency among all compute nodes. The first type is i) sync based on persistent storage [94]. The second type is ii) sync based on local cache status [27]. The third type is iii) sync based on the shared remote buffer [22].

4) Multi-layer Recovery: According to the hierarchical division in the architecture, fault recovery techniques can be divided into three levels. The first level is i) No-Redo Recovery in the Compute Layer [94]. The second type is ii) Two-Tier ARIES based on Buffer Layer [115]. The third type is iii) Optimizations in the Storage Layer [95].

5) HTAP Optimization: We discuss HTAP optimizations in cloud-native databases, which include three types. The first type is i) dynamic storage format transformation [35]. The second type is ii) heterogeneous data replicas [38]. The third type is iii) unified table storage design [78].

B. Cloud-Native OLAP Databases

1) Cloud-Native OLAP Architectures: Cloud-native OLAP databases emphasize efficiency and throughput in analytical query processing. The architecture design needs to consider the elasticity of computation to support fluctuating workloads, as well as the local cache and shared memory for efficient query processing. The architectures of cloud-native OLAP databases are classified into two categories as follows:

1) Disaggregated Compute-Storage OLAP Architecture: The first category has a two-layer architecture, where the compute layer executes the queries with the local SSDs, and the storage layer persists the entire data with the computational pushdown.

2) Disaggregated Compute-Memory-Storage OLAP Architecture: The second category owns a three-layer architecture, where a shuffle memory pool is disaggregated to process the distributed joins more efficiently.

2) Cloud-Native OLAP Techniques: We present five types of cloud-native OLAP techniques.

1) Storage Management: The disaggregation of functional modules in the cloud-native environment results in differences in data management methods. We introduce three types of storage management techniques. The first type is i) Metadata storage management [25], the second type is ii) Data partitioning [15], [37], and the third type is iii) Semi-structured data management [25], [59], [111].

2) Query Processing: Compute nodes read data from remote storage services, which drives the query processing optimizations to reduce network transmission. We introduce three types of query processing techniques. The first type is i) Columnar scan with pushdown [70], [96], [103], which aims to push the computation into the storage side. The second type is ii) Columnar scan with caching and pushdown [102]. The third type is iii) Columnar scan with the shuffle memory pool [59].

3) Serverless Computing: Serverless computing intends to make customers use the data analytical services without considering the server deployment and configuration. We introduce two types of serverless computing methods in cloud databases. The first type is i) Serverless with functions as a service [73], where queries are adaptively executed based on the cloud function services. The second type is ii) Serverless with the elastic query engine [16], which enables to perform the queries by dynamically provisioning the query engine.

4) Data Protection: Protecting user data privacy and security is the basis for customers to use cloud services. We present two types of techniques: i) Software-based data protection [25] and ii) Hardware-based data protection, e.g., the enclave in Intel SGX [11].

5) Machine Learning: We will look at emerging cloud database techniques for machine learning, such as Sagemaker [55]. Moreover, we will introduce how cloud databases can benefit from machine learning techniques [52], [53], [93].

C. Contributions

Differences with existing surveys: In this paper, we focus on the fundamental techniques of cloud-native databases [50]. We also summarize the pros and cons of various architectures and techniques. Before the emergence of cloud-native databases, Sakr [81] reviewed cloud-hosting databases. Mansouri et al. [58] surveyed the key techniques of cloud storage management. Narasayya et al. [65], [66] discussed various cloud data services. Unfortunately, existing works neglected many fundamental techniques of cloud-native databases, such as data consistency, data synchronization, and failure recovery. Last but not least, we review newly-emerged techniques, such as the cloud-native HTAP techniques, pushdown-based query processing, and machine learning-based optimization.
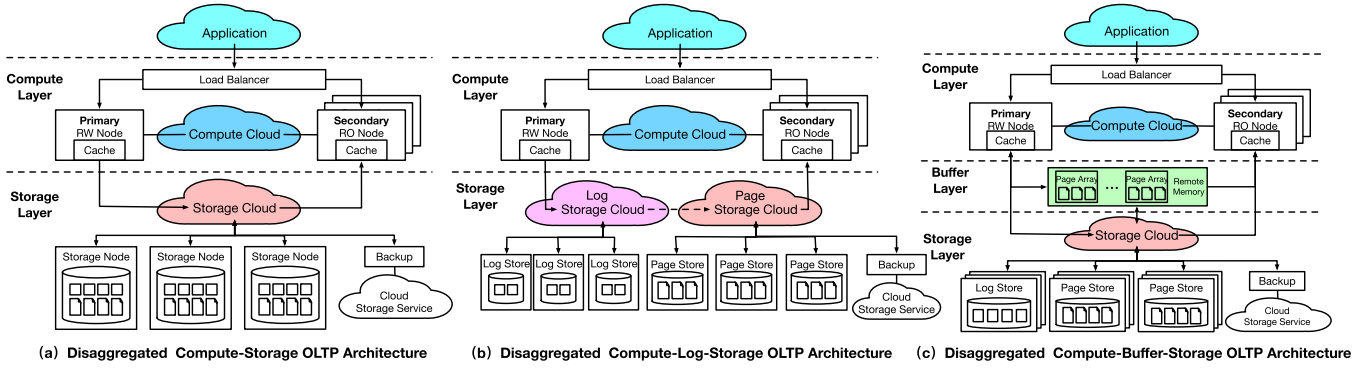
Fig. 3. Architectures of cloud-native OLTP databases.

To summarize, we make the following contributions:

1) We survey cloud-native databases from the perspective of system architectures. We introduce a taxonomy of cloud-native OLTP and OLAP databases, respectively. We also discuss their pros and cons.

2) We summarize the key techniques of cloud-native databases concerning the OLTP and OLAP workload. We take a deep dive into the key techniques concerning transaction processing, data replication, database recovery, storage management, query processing, serverless computing, data protection, and machine learning.

3) We provide new research challenges and discuss future directions, including multi-writer architecture, fine-grained serverless, SLA-aware cloud-native HTAP techniques, and multi-cloud data service.

II. CLOUD-NATIVE OLTP ARCHITECTURES

OLTP database systems are designed for transaction processing scenarios, which means they should guarantee ACID properties during query processing [36]. However, the coupled compute-storage architecture in cloud-hosting databases suffers from write amplification due to coupled resource scheduling [48], [94]. The disaggregated architecture designs in cloud-native databases are introduced to solve the above problems. According to the degree of separation management of storage services and the use of remote memory services, the architectures of cloud-native OLTP databases can be classified into three categories (shown in Fig. 3): 1) Disaggregated Compute-Storage OLTP Architecture, 2) Disaggregated Compute-Log-Storage OLTP Architecture and 3) Disaggregated Compute-Buffer-Storage OLTP Architecture.

A. Disaggregated Compute-Storage OLTP

1) Design Motivation: This category of databases adopts a disaggregation architecture that separates the compute and storage modules in the cloud. The design motivation of this architecture can be concluded as the following three aspects. i) Elasticity. it aims to schedule the computing and storage resources independently, which could avoid the waste of resources caused by resource coupling in cloud-hosting databases. ii) Efficiency.

Dirty page flushing is eliminated under this architecture, which significantly reduces the write amplification. iii) Availability. Because of the multiple disaggregated modules, it must provides a multi-level failure tolerance to reduce the average recovery time compared to instance-level recovery.

2) Data Access Path: The data access path is different from the cloud-hosting databases. The primary node will only transfer redo logs and metadata to the storage layer during the data writing process. The storage nodes will asynchronously replay the logs in the background to update records, avoid dirty page transmission, and relieve the network bottleneck in the cloud environment. Nevertheless, reading data from pages without the dirty page flush-back may suffer from the update delay caused by the asynchronous log replaying. Therefore, the databases organize the redo logs into the linked list structure in the order of log serial number (LSN), which allows the storage nodes to read the records by directly analyzing the redo logs.

3) Pros and Cons: Compared with cloud-hosting databases, cloud-native databases have the following advantages. i) Low Write Latency. The write operation can commit once the redo logs are persistent without waiting for the updates of record pages. ii) Reduced Write Amplification. Since the data update is pushed down to the storage layer, which avoids the dirty page transmission and relieves the network pressure. iii) Improved Elasticity. Computing and storage are supported by different cloud services. The independent scheduling process improves the system's elasticity. The limitation of this architecture is the read latency. The compute nodes send read requests to the storage layer when the cache misses, which may suffer extra log chain analyzing latency.

4) Representatives: The representative databases with the disaggregated compute-storage architecture include Aurora [94] and AlloyDB [35]. These two systems use a similar system architecture design but with different technique implementations. For the common part, they implement the same log processing techniques, like "the log is the database" in Aurora and "Log Processing Service" in AlloyDB. For the difference, Aurora optimizes storage management based on Quorum mechanisms by extending data replicas; it also implements non-blocking failure recovery. While AlloyDB optimizes the HTAP workload via dynamic data format transformation in the compute nodes.

B. Disaggregated Compute-Log-Storage OLTP

1) Design Motivation: This category of databases extra separates the storage service for logs and pages based on the first category of databases. Logs guarantee the persistence of updates, while pages provide high-efficiency query processing. The design motivations can be concluded as two aspects. i) Efficiency. First, a fast cloud storage service for logs can significantly reduce the write commit latency. Second, standard cloud storage service for pages can avoid high costs. ii) Elasticity. It can improve the systems' elasticity if these two storage services are scheduled independently.

2) Data Access Path: The disaggregation of log and page storage influences the data access path. This architecture separates the data read and write path. Compute nodes only write to log storage and read from page storage. The storage layer handles the synchronizations of log and page storage internally. However, due to the asynchronous updates and the network latency across different storage services, the page updates could lag in storage nodes.

3) Pros and Cons: Compared with the first category of databases, the disaggregated compute-log-storage architecture has the following advantages. i) Low Write Latency. The write commits latency further declines with the help of the fast cloud storage service for logs. ii) Improved Elasticity. The databases' elasticity is improved with the disaggregation of different storage services. Standard storage for pages has a relatively-low cost, and fast storage for logs improves the transaction processing performance. The limitation of this architecture is the synchronize latency. The compute nodes could be blocked and continue to wait for the synchronization in storage nodes when data lags.

4) Representatives: The representative databases with the disaggregated compute-log-storage architecture include Azure HyperScale [12] and Huawei Taurus Database [27]. The main differences between these two systems are the storage management method. Taurus adds Storage Abstract Layer (SAL) [27] in each compute node to handle the data access on the storage layer. While HyperScale implements XLOG [12] service to take responsibility for similar functions. The difference is that the XLOG service is separated from compute layer as an independent layer, which achieves further independence on manageability and fault tolerance.

C. Disaggregated Compute-Buffer-Storage OLTP

1) Design Motivation: This category of databases expands the shared buffer for databases. The buffer is supported by remote shared memory service [98], which provides much lower latency data access than the persistent storage service. The design motivation can be concluded in three aspects. i) Efficiency. The read latency can be significantly reduced with the remote memory. ii) Throughput. If all the compute nodes share the remote buffer, it could reduce the duplicate read requests from different compute nodes. iii) Elasticity. Since the memory resource allocation is independent of persistent storage service, it could further improve the elasticity of databases.

2) Date Access Path: The shared buffer provides an additional layer of buffer on top of the local cache in each compute node.

Unlike the local cache, the buffer is shared by all compute nodes, which allows the primary node to transfer the updates to secondary nodes. Besides, since the buffer is shared by multiple nodes, it could become the bottleneck of the network. Hence, the shared buffer will not flush back dirty pages, and the redo logs still guarantee the update's durability.

3) Pros and Cons: Compared with the first two categories of databases, the disaggregated compute-buffer-storage architecture has the following advantages. i) Low Read Latency. The read latency is significantly reduced when data is cached in the remote buffer. ii) Improved Read Throughput. The number of duplicate read from different compute nodes is reduced since all nodes share the buffer. iii) Improved Elasticity. Memory disaggregation enables the elastic scheduling of memory resources, hence the higher elasticity. The limitation of this architecture is the high network cost. Fully utilizing the performance of remote memory requires an expensive RDMA network for low network latency. Besides, it has a high requirement of network bandwidth since all the compute nodes need to share the same buffer.

4) Representatives: The representative databases with the disaggregated compute-buffer architecture include Alibaba PolarDB Serverless [22], which builds a shared buffer for all compute nodes based on the remote memory service. The data updates from the primary node can be written to the shared buffer layer and can be synchronized to secondary nodes, which improves data synchronization performance. The main challenge is to keep the data consistent between the primary node and shared buffer, which will be discussed in the next section.

D. Summary of the Cloud-Native OLTP Architectures

Table I presents a comparison of the cloud-native OLTP architectures concerning read and write performance, availability, elasticity, and cost.

1) Disaggregated compute-storage: These databases don't require fast storage and remote memory service, which have the lowest cost. Particularly, the primary node only writes the log to the storage layer, which is more efficient than the cloud-hosting architecture. Reading records requires additional log replay, which affects read efficiency.

2) Disaggregated compute-log-storage: These databases require fast storage service to reduce log write latency, increasing costs but improving write performance. Databases' elasticity and availability are higher than the first category because of the further separation of storage services.

3) Disaggregated compute-buffer-storage: These databases require remote memory service with low network latency, which demands an expensive RDMA network. Hence, they have a high cost. Nevertheless, they provide a better read performance with the shared remote buffer. Since the remote memory service is independent of computing and storage, it enhances the system's elasticity. Besides, the remote buffer can accelerate the recovery of the compute layer, which improves the availability as well.

III. CLOUD-NATIVE OLTP TECHNIQUES

This section will introduce the fundamental techniques in cloud-native OLTP databases. We classify them into five groups:

TABLE I
A CLASSIFICATION OF CLOUD-NATIVE OLTP DATABASES BASED ON THE ARCHITECTURE

OLTP Architecture	Representatives	Write	Read	Availability	Elasticity	Cost
Disaggregated Compute-Storage	Aurora	Medium	Medium	High	High	Low
Disaggregated Compute-Log-Storage	HyperScale	High	Medium	Excellent	Excellent	Medium
Disaggregated Compute-Buffer-Storage	PolarDB Serverless	High	High	Excellent	Excellent	High

TABLE II
AN OVERVIEW OF KEY TECHNIQUES OF CLOUD-NATIVE OLTP DATABASES

| Technique Type	Main Approaches	Cloud Databases	Pros.	Cons.
Data Placement Strategy	Coupled Page-Log	Aurora [94]	Low Sync Latency	Extra Log Analysis
	Disaggregated Page-Log	HyperScale [12]	More Elasticity	Long Sync Latency
Storage Layer Consistency	Quorum-based Protocol	Aurora [95]	Strong Concurrency	Extra Sync Phase
	Paxos-based Protocol	PolarDB [22]	Strong Consistency	Complex Procedure
Compute Layer Consistency	Persistent Storage based	Aurora [94]	High Availability	Long Sync Delay
	Local Cache based	Taurus [27]	Low Sync Latency	Cache misses
	Remote Shared Buffer based	PolarDB [22]	Low Read Latency	Cache Inconsistent
Mutli-layer Recovery	No-Redo in Compute Layer	Aurora [94]	Fast Recovery	Redo in Storage Node
	Two-Tier ARIES in Buffer Layer	LegoBase [115]	Reduced Recovery Time	High Cost
	Optimizations in Storage Layer	Aurora [95]	Improved Storage Availability	More Data Replicas
HTAP Optimization	Storage Format Transformation	AlloyDB [35]	Reduced Storage Space	Large Search Space
	Heterogeneous Data Replicas	TiDB [38]	Strong Isolation	Reduced Freshness
	Unified Table Storage	SinglestoreDB [78]	Low Read Latency	High Memory Cost
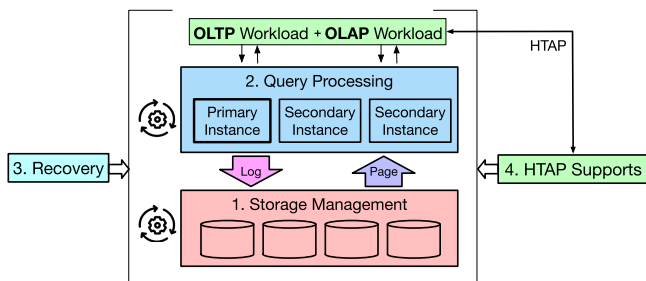
Fig. 4. An overview of cloud-native OLTP techniques.

data placement strategy, storage layer consistency, compute layer consistency, multi-layer recovery, and HTAP optimization. The relationship between these five parts is depicted in Fig. 4. Data placement strategy refers to the data organization and placement methods in the cloud. As there are multiple instances in both the storage layer and compute layer to ensure high availability, storage layer consistency and compute layer consistency care about the consistent protocols in the cloud. Multi-layer recovery mechanisms are designed to provide a fine-grained method to recover the failure in multiple layers. Finally, HTAP optimizations add the OLAP support based on the original OLTP mechanism. Table II summarizes the main approaches in each group, as well as their advantages and limitations.

A. Data Placement Strategy

In cloud-native databases, the data placement strategy refers to organizing different data types in databases, mainly focusing on the logs and pages. The data placement strategy determines the transaction processing workflow. They are influenced by the architecture design, which can be categorized as 1) Coupled

Page-Log Placement Strategy and 2) Disaggregated Page-Log Placement Strategy. For the former type, a unified cloud storage service supports log and page storage, which can provide physical correlation to reduce network pressure. The coupled placement strategy is used in disaggregated compute-storage architecture. For the latter one, the disaggregated placement strategy is used in disaggregated compute-log-storage architecture. Isolated cloud storage services support log and page storage, which separates the read and write process of transactions to achieve both low write latency and high read throughput.

1) Coupled Page-Log Placement Strategy: In the cloud-native OLTP databases, redo logs keeps the updating history, which means any record at any database version can be analyzed from the redo logs. Hence, databases can directly load records from redo logs. Unlike traditional databases that read records through data pages, the coupled page-log placement strategy uses the same cloud storage service to store the log and page data. The fundamental difference lies in the data processing process within the storage layer, summarized as "the log is the database."

As shown in Fig. 5, the same storage node saves pages and redo logs simultaneously. The data update from the compute layer only requires the storage node to complete the persistence of the redo log. Thus, the dirty pages will not flush back to the storage layer, which significantly reduces the write amplification in cloud-hosting architectures due to the updates of multiple replicas. A read operation from compute layer requires the storage node to load the record with a specific version from the redo logs. However, the overhead of loading records will increase with the growth of historical data, most of which has already expired. Therefore, the page materialization controls the storage capacity and the read time by discarding expired redo logs. This process is done asynchronously in the background of the storage node, which avoids the update delay of direct
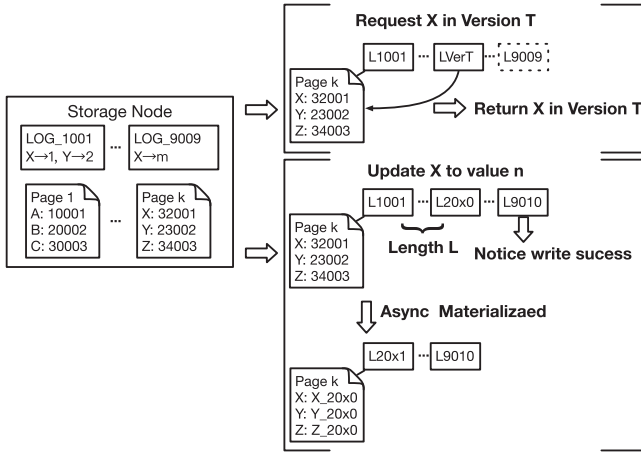
Fig. 5. The coupled data placement strategy.

Fig. 6. The disaggregated data placement strategy.

page updating. As shown in Fig. 5, update requests on value X will not be directly written into the page (Page k) to which it belongs. Instead, the database will generate the redo log (L9010) and append it to the storage node. During the read requests, the storage node will ignore the redo logs later than the transaction (version T). The page materialization will consume the redo logs and update the page, which reduces the length of log chains and accelerate the read operations.

In summary, this strategy has the following advantages: 1) Reduced write amplification. Dirty pages do not flush to the storage layer, significantly reducing network pressure. 2) Reduced update delay. Storage nodes can directly load records from log data, which avoids the update delay of page data. The main limitation of this strategy is the extra process of redo log analysis during the reading process.

2) Disaggregated Page-Log Placement Strategy: Pages and logs stored in persistent storage have different responsibilities in the database system. Pages can directly read a specific version of the record, which is mainly used in the reading process. Besides, it guarantees the availability of the database. In contrast, logs can be written to disk sequentially, which is mainly used in the reading process and guarantees the durability of the transactions. The disaggregated page-log storage placement strategy places the pages and logs based on their different features. The fundamental difference between this strategy and coupled one can be summarized as "the disaggregation of availability and durability."

As shown in Fig. 6, logs and pages are persisted in separated storage using the cloud storage service. Since the data update from compute layer only requires the persistence of the logs, the compute layer only needs to write the redo logs to the log storage, which is supported by fast storage services and gets lower write latency. At the same time, pages will be stored in standard cloud storage services to reduce the cost. The update logs will be batched asynchronously to page storage. Considering the possible unavailability of page storage nodes, the log transferring does not require all page storage nodes to complete the sync. The nodes inside the page store supplement these missing logs from other nodes through the Gossip protocol [24].
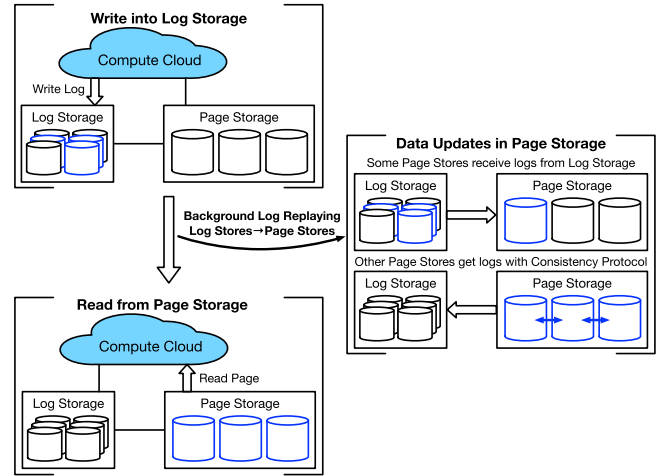
Compared with the coupled strategy, this method has the following advantages: 1) Reduced data write latency. The log persistence is backed by fast cloud storage, which improves the write performance. 2) Better elasticity. The scheduling of storage services is independent, enhancing the system's elasticity. The main limitation of this strategy is the larger read latency caused by synchronization across storage services when the cache misses.

B. Storage Layer Consistency

In cloud-native databases, storage layer consistency techniques are used to maintain the consistency among multiple data replicas in the storage layer. These techniques are based on original distributed systems protocols with specific optimization for cloud environments, which can be categorized as 1) Quorum-based Protocol and 2) Paxos-based Protocol. The quorum-based protocol is derived from the quorum algorithm [89] with some mechanisms to enhance consistency. In comparison, the Paxos-based protocol is derived from the Paxos-like algorithm (including Paxos [47] & Raft [68]) with customized mechanisms to improve the concurrency.

1) Quorum-Based Protocol: Quorum-based voting [89] is a classic method to guarantee the consistency in the distributed systems. The quorum algorithm sets the minimum voting number that a distributed transaction has to obtain, which is then used to solve the read-write and write-write conflicts among storage nodes. Migrating the quorum algorithm to the cloud environment mainly faces two challenges: 1) high availability requirement and 2) low recovery latency.

For the first challenge, most distributed systems implement the quorum algorithm with three data replicas, which provide single-node fault tolerance. However, data centers for cloud service are deployed geographically isolated and require extreme availability [34]. Therefore, Aurora increases the number of replicas for improving the system's reliability [94]. Cloud services can be divided into multiple fault-tolerant independent regions through the isolated physical deployment. Hence, the

probability of simultaneous failure in different regions is extremely small. Based on the above facts, the cloud databases can maintain two replicas in three regions to achieve "region + 1"-level fault tolerance. Even if a single region fails, at least four replicas still run normally to ensure high availability.

For the second challenge, a possible solution of reducing failure recovery time is to prepare a new replica before the system breakdowns. In the case of multiple replicas, the database can migrate data in advance and can generate a backup instance after a single replica is abnormal. Since the data migration is performed asynchronously, the backup instance will not replace the abnormal one immediately due to the high migration cost. Instead, they will run simultaneously and be controlled by the quorum set mechanism [95]. Backup and abnormal instances and the rest of the normal replicas form two quorum sets. Multiple sets are managed in a logical "or" manner. Query processing only requires at least one set to complete. Instances with long-term exceptions will be removed, and the database will discard the quorum sets containing such instances.

The advantage of quorum-based protocol is the high concurrency supported by the simple algorithm workflow. The limitation is that quorum-based protocols do not guarantee linearizability. Replicas implement extra gossip protocols to fill up the missing updates caused by temporary exceptions in certain replicas.

2) Paxos-Based Protocol: Paxos [46], [47] is a family of protocols to reach the consensus in a network of unreliable or fallible participants. Since the Raft protocol [68] can be regarded as the simplified Paxos with stronger assumptions, we categorize all methods derived from Paxos and Raft as Paxos-like protocols. Classical Paxos-like algorithms strictly follow the linearization process, which limits the concurrency of transaction processing. Therefore, how to improve the concurrency is the most important problem in applying the Paxos-like algorithms to the cloud-native databases.

Traditional databases require logs to be committed in a strict order, which means the previous logs must be committed successfully. Such a mechanism limits the concurrency due to the strict committing order. ParallelRaft [21] makes two optimizations to improve the performance. Out-of-order acknowledging and committing are allowed in ParallelRaft when the writing ranges of log entries are not overlapping, which is considered not conflicted. Besides, ParallelRaft optimizes the catch-up processes for lagging followers to re-synchronize with the leader.

The advantage of paxos-based protocol is the linearizable features supported by the Paxos-like algorithms. The limitation is that Paxos-based protocols limit the system's concurrent processing efficiency, which requires customized optimizations such as out-of-order committing.

C. Compute Layer Consistency

In cloud-native databases, compute layer adopts the "single-writer, multi-reader" architecture. That is, the primary node handles update queries and syncs the data to secondary nodes. All the secondary nodes are read-only and just update their status to the primary node. The synchronization process requires it
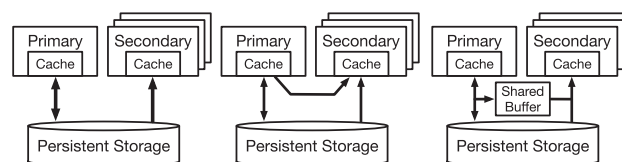
(a) Sync with Persistent Storage   (b) Sync with Local Cache   (c) Sync with Shared Buffer

Fig. 7.   Compute layer synchronization.

to be low-latency and high-reliable, which can be categorized into three types: 1) Persistent storage based, 2) Local cache based, and 3) Remote shared buffer based. Notice that metadata synchronization always adopts direct transmission, and the data size is much smaller than log and page data. Therefore, this part mainly focuses on the synchronization of log and page data.

1) Persistent Storage Based Synchronization: The first synchronization method is based on the persistent storage. As shown in Fig. 7(a), the primary node transfers the redo logs to the storage layer. Combining the data placement strategy, the dirty pages in the primary node never flush back to the storage layer. The storage layer internally replays the redo logs to update the data pages. Since all the compute nodes share the storage service, the secondary nodes receive the updates once the corresponding logs have been replayed in the storage layer. The single-writer architecture only allows one primary node to update data at any time, thereby eliminating the possibility of write-write conflicts and guaranteeing strong data consistency. However, the network transmission that crosses different services suffers from long network latency. Besides, as the logs are replayed asynchronously, it significantly increases the update delay of secondary nodes.

2) Local Cache Based Synchronization: The second synchronization method is based on the local cache status in secondary nodes. This method aims to directly update the cache data of the secondary node and clear its dirty pages. As shown in Fig. 7(b), the primary node directly transfers redo logs to secondary nodes. The secondary nodes will update the dirty pages in the local cache based on these logs, achieving cache consistency with the primary node. The main challenge of this method lies in network transmission, which mainly includes two aspects: 1) Bandwidth. The network bandwidth of the primary node is limited, and simultaneous transmission to multiple secondary nodes may become the bottleneck. 2) Latency. All the replicas need to obtain the logs, which may lead to stragglers that affect the overall performance, namely, "the bucket effect".

For bandwidth issues, the compute layer can push down the transmission task to the fast storage service (e.g., the log storage) to reduce the pressure on the network bandwidth [27]. In this way, the computing layer distributes the transmission tasks to multiple nodes of the fast storage service, which significantly reduces the transmission pressure of a single node. For latency issues, the step of receiving logs in secondary nodes is controlled by a loose protocol [12]. The primary node does not require the secondary node to confirm the receiving process. The secondary nodes allow transmission failure. Moreover, they only need to read the missing part through the storage layer without affecting the correctness of the system.

3) Remote Shared Buffer Based Synchronization: The third synchronization method is based on the remote shared buffer. The primary and secondary nodes share the same remote buffer, which makes it possible to transfer the data updates. As shown in Fig. 7(c), the update requests in the primary node must update data in the local cache and remote buffer simultaneously. Secondary nodes can directly load the record from the remote buffer. This method has two following challenges: 1) Consistency. Updates in the primary node's local cache and remote buffer do not satisfy strict atomicity. 2) Network. The shared buffer is accessed by multiple nodes simultaneously and has a high requirement on the network access.

For the consistency issue, PolarDB serverless [22] proposes a cache invalidation mechanism to ensure the consistency between the primary node and shared buffer. A specific table in the shared cache records the consistency relationship. Then the secondary nodes will ignore the invalid pages. The update of the table and data satisfies atomicity, whose delay is much lower than directly udate the corresponding pages in the remote shared buffer. For the network issue, the RDMA network can support both high-bandwidth and low-latency network requirements. Therefore, the system requires a high-speed RDMA network deployed in the hardware layer.

D. Mutli-Layer Recovery

In cloud-native databases, different cloud services support various functional modules, resulting in the fault-tolerant independence between the modules. On the one hand, independent fault tolerance produces high availability. On the other hand, the physical isolation of different cloud services significantly increases network latency for failure recovery, which demands specific treatment in the failure recovery phase. According to the architecture design of the cloud-native databases, the recovery optimization can be performed at different layers, which can be classified into the following three categories: 1) No-Redo Recovery in Compute Layer, 2) Two-Tier ARIES in Buffer Layer, and 3) Optimization in Storage Layer.

1) No-Redo Recovery in Compute Layer: Failure recovery of the computing layer requires restarting the computing nodes. As shown in Fig. 8, traditional database systems use a monolithic architecture. The write-back policy postpones the flush process of dirty pages and causes some page updates to be lost under abnormal circumstances. Therefore, databases need to write redo logs to the persistent storage before committing the transactions to avoid the loss of page updates. ARIES algorithm [63] is a classic recovery algorithm in database systems, which contains three main stages: analysis, redo, and undo. During the redo stage, the database will scan the required logs sequentially based on the analysis results to restore the dirty page status. However, as cloud-native databases use a disaggregated architecture and follow the philosophy of "the log is the database", the compute layer does not need to sync the dirty page status to the storage layer for the durability of transactions, and the storage layer can directly load data from the redo logs without sending it to the compute nodes. Therefore, the redo process is pushed down
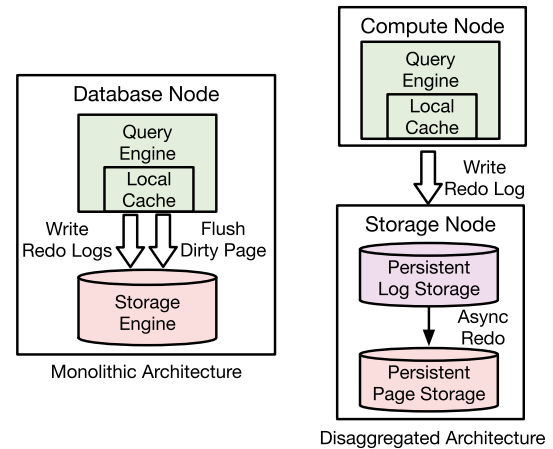

Fig. 8. The comparison of monolithic architecture and disaggregation architecture on failure recovery.

to the storage layer, which reduces data transmission between layers and the recovery latency of the compute nodes.

2) Two-Tier ARIES in Buffer Layer: The exceptions in the buffer layer will not affect the durability and availability of the system. However, the disaggregation of cloud computation and memory services produces independent fault tolerance, which means the compute nodes and remote buffer are unlikely to fail simultaneously. Based on the above assumption, LegoBase [115] proposes the two-tier ARIES protocol to handle the failure of the compute node and the remote buffer. Such a protocol extends the traditional ARIES algorithm by creating checkpoints into two layers: 1) the remote buffer layer and 2) the persistent storage layer. The compute nodes and the remote buffer forms the first-tier ARIES. The network transmission cost of this part is small, and checkpoints can be recorded more frequently to reduce the failure recovery time. The first-tier protocol can deal with failure recovery in most cases, except for the case that the computing nodes and remote memory are abnormal simultaneously. In this case, the persistent storage in the second-tier ARIES will guarantee the worst-case failure recovery.

In summary, this algorithm is similar to the traditional ARIES one in the worst case. Nevertheless, it significantly reduces the recovery time in most cases with the help of remote shared buffer.

3) Recovery Optimization in Storage Layer: The storage layer is the foundation of the system's durability and availability in cloud-native databases, which maintains multiple replicas simultaneously to ensure the extremely high-reliability requirements. Particularly, the storage layer has two types of optimization in failure recovery: 1) More replicas; and 2) Pre-failure recovery preparation.

The most basic way of improving fault tolerance is to increase the number of redundant replicas, e.g., doubling replicas in each available zone [95]. Moreover, it could expand the number of nodes in the log storage [27]. The main limitation of this method is that it introduces additional storage overhead. The second approach requires pre-preparing new standby nodes when partial replica anomalies are detected, e.g., the quorum set mechanism
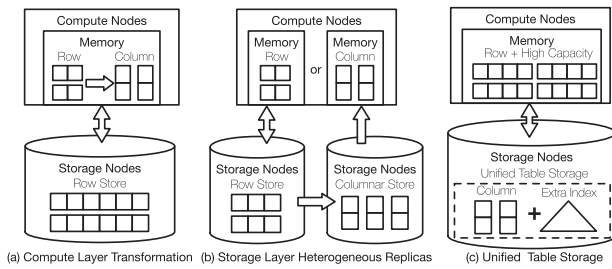
Fig. 9. HTAP optimization.

in Aurora [95]. Such a method has a smaller storage overhead but will occupy network bandwidth while generating backup nodes.

E. HTAP

Traditional OLTP database systems are generally used for transactional workloads, so they have implemented many techniques to optimize the efficiency of transactional processing, e.g., row-format page organization and index structures. However, with the further development of data-intensive applications in recent years, it calls for real-time analysis requirements for the transactional databases, e.g., real-time fraud detection [79]. These demands drive the OLTP databases to add support for real-time analytical workloads [51], [74], [107], [108]. Regarding the cloud-native OLTP databases, there exists three types of HTAP optimization (shown in Fig. 9): 1) Dynamic storage format transformation in the compute layer; 2) Heterogeneous data replicas in the storage layer; and 3) Unified Table Storage. These techniques add particular optimizations for analytical workloads based on the original OLTP databases. Therefore, the ACID properties of the databases will not be affected.

1) Storage Format Transformation: The first type is the dynamic storage format transformation in the compute layer. Conventionally, the pages of the storage layer in the OLTP databases organize records in row format. Such an organization method has obvious performance advantages in dealing with transactions and point queries. However, it is not suitable for analytical queries, which are mainly composed of large-scale aggregation and scan queries. AlloyDB [87] proposes a method to transform the row-format data to the columnar data in the compute nodes dynamically. Particularly, the optimizer analyzes the characteristics of the workload and predicts data that is likely to be frequently accessed in analytical workloads. Then, in the process of reading records from the storage layer, this part of the data is directly converted into the columnar format and kept in the cache. As a result, the analytical queries can directly use the data in the columnar format in the cache to speed up the query.

The above approach has two main challenges: 1) Storage format transformation will yield additional computing and storage overhead, so selecting which data to be converted is critical to the system performance. AlloyDB adopts the method of machine learning to assist in the work of data selection. However, few details have been revealed. 2) Storage format conversion should minimize the impact on the efficiency of transaction processing. Therefore, AlloyDB can keep data in both storage formats in the

cache at the same time [87]. The optimizer chooses which type of data to scan based on workload type. However, it is challenging to select an optimal execution plan due to the exponential growth of the planning space.

2) Heterogeneous Data Replicas: The second type of method [20], [38] maintains heterogeneous data replicas in the storage layer. The main difference from the first method is that it persists the row-wise and columnar replicas in the storage layer rather than the compute layer. Particularly, when handling the transaction requests, the master node asynchronously replicates the logs to the secondary nodes for data synchronization.

From the implementation perspective, the overall architecture do not need to be modified. The columnar format replicas are stored in read-only nodes, which are the learners of the row format replicas in the consensus protocol. Hence, it ensures the consistency of the heterogeneous replicas without influencing the origin OLTP system. Analytical workloads will be allocated with extra computing resources on demand according to the workload's intensity, benefiting from cloud services' elastic scheduling capability. Therefore, handling analytical workloads will not influence the computing resources for transaction processing.

From performance perspective, this method's advantage is that it isolates the performance of transactional and analytical processing, meaning that both transactions and queries can be efficiently processed at the same time. Moreover, the excellent isolation facilitates the flexible scheduling of the heterogeneous workloads. However, since the columnar format replicas are the learners of row format ones, it must face the problem of data freshness due to the data transmission and transformation. That is, recent updates on the primary node must take certain time to be transferred and transformed to the columnar replica, causing analytical workloads to have a version lag compared to transactional workloads. Furthermore, this method adds additional computation and storage resources for the OLAP workload.

3) Unified Table Storage: The third type of method is a unified table storage design for both OLTP and OLAP workloads, which is employed in the SingleStoreDB (S2DB) [78]. The main difference is that S2DB does not persist data into different layouts, which is often adopted in other HTAP systems. The unified table storage contains two parts: 1) In-memory row store. The in-memory storage is developed from its predecessor MemSQL [86], which implements a lock-free skiplist to index the rows and use the pessimistic concurrency control to avoid conflicts. This part is mainly used to improve the OLTP performance. 2) On-disk column store. WAL logs on the disk supports durability, which are written to the storage sequentially. Other data pages are organized in columnar format to optimize the aggregation and scan operations in analytical queries. Besides, it constructs the secondary and unique indexes on the column store, which provides the optimization on point-queries on the columnar store. In addition to the extra indexes on the disk, the key to maintain the high performance of S2DB is that the in-memory row store needs to cover most of the search requirements. Otherwise, on-disk column storage will degrade the performance in transaction processing compared with on-disk row store.

Overall, this method does not need to copy data into different layouts, which saves the computation and I/O overhead caused by the data conversion. The limitation of this method is that maintaining the high cache hit rate is necessary for the high seeking performance which requires more memory resources.

IV. CLOUD-NATIVE OLAP ARCHITECTURES

Cloud-native OLAP databases target at large-scale data analytics with elastic and scalable cloud services. Compared to share-nothing MPP data warehouses, cloud-native OLAP databases increase the elasticity with the disaggregation architecture and achieve high availability with the cloud storage and cross-region availability zones. We classify the cloud-native OLAP architectures into two categories: 1) disaggregated compute-storage OLAP architecture and 2) disaggregated compute-memory-storage OLAP architecture.

A. Disaggregated Compute-Storage OLAP

This category of databases [3], [15], [96] adopts a disaggregated compute-storage architecture, and the compute layer and the storage layer are connected to a high-speed network. As shown in Fig. 10(a), the compute layer consists of a service manager and compute clusters, the service manager provides a collection of services that manage the metadata, resources, queries, and security. The compute clusters perform the queries with elastic compute resources, and each worker node has the local SSD for caching.

1) Motivation and Key Features: There are three motivations for such an architectural design. First, the traditional coupled architectures can only manage the resources at the instance level, and storage and compute resources need to be scaled independently for high elasticity. Second, the cloud service should tolerate cluster and node failures for high availability. Thus a disaggregated architecture can have smaller downtime as it can handle the failures of compute and storage nodes separately. Lastly, since the workloads are heterogeneous (either high I/O bandwidth or heavy computation), different hardware configurations could be used to compute the storage nodes. In summary, this architecture features i) disaggregation of compute and storage, ii) multi-tenancy and serverless, iii) elastic data warehouses, iv) local SSD caching, and v) cloud storage service, such as AWS S3 [80].

2) OLAP Workflow: Processing the queries in the cloud mainly involves three steps. First, the queries are parsed, rewritten, and optimized with the catalog statistics in the metadata storage. Second, the query plans are compiled and sent to the computer clusters for execution. The computer nodes perform the tasks with the local attached SSDs that can be treated as the local cache. Third, if the local cache is not hit, the data will be loaded from the cloud storage with the optional computation pushdown.

3) Pros and Cons: Compared to on-premise share-nothing OLAP architectures, the disaggregated compute-storage architecture has higher availability, where cluster and node failures can be recovered quickly because of the data replication across many availability zones and the scalable cloud service. It is more cost-efficient in two-fold. First, resources are virtualized

(a) Disaggregated Compute-Storage OLAP Architecture

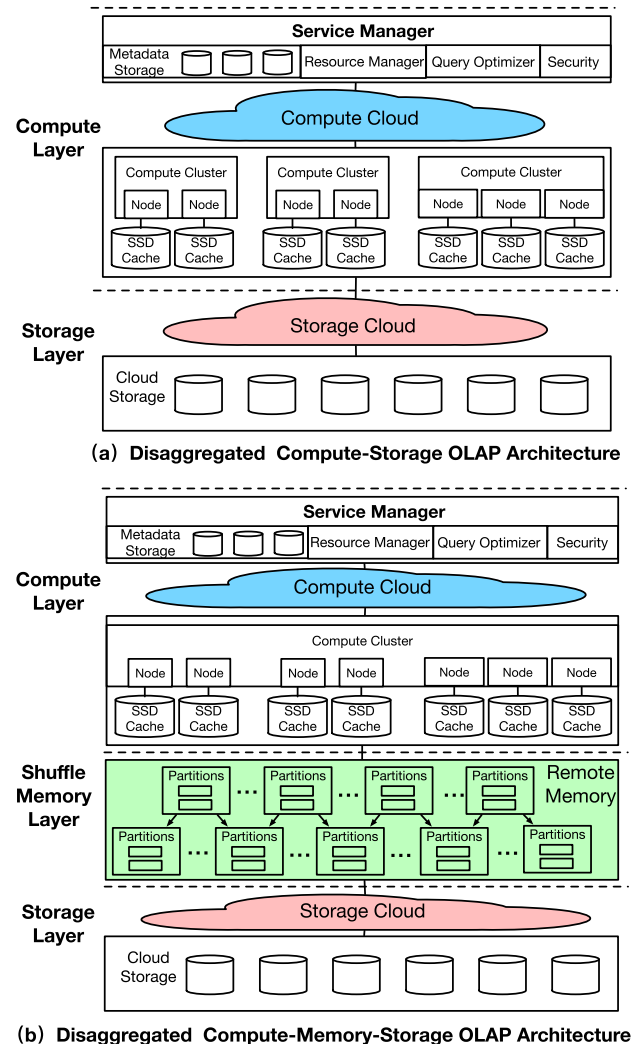**(b) Disaggregated Compute-Memory-Storage OLAP Architecture**

Fig. 10. Architectures of cloud-native OLAP databases.

and shared by multiple tenants. Second, serverless computing provides the pay-as-you-go model in a query-level granularity. Finally, since the compute and storage resources can be scheduled on demand individually, it provides better elasticity. However, the major limitation of the first architecture is that network traffic becomes the bottleneck when the local cache misses. Therefore, it needs to design efficient and effective caching and computation pushdown strategies.

4) Representatives: Two representatives are Snowflake [25], [96] and Redshift [70]. Snowflake relies on cloud services to manage multiple virtual warehouses, workloads, security, and metadata. In the compute layer, it provides multiple VWs(Virtual Warehouse), where each VW is a cluster and consists of multiple EC2 instances. Normally, one query is executed in one VW for one tenant, and each VW can be started or shut down at any point. For data storage, it combines local ephemeral storage and cloud storage(e.g., AWS S3) to store data. Another representative is Redshift [15], [70], which was initially an MPP data warehouse and then transformed into a cloud-native database. It also contains multiple compute clusters, each with a leader node as the coordinator, with multiple compute nodes. Particularly, it has an

acceleration layer with various components. First, the spectrum nodes are customized for querying semi-structured data using partiQL [5]. The advanced query accelerator (AQUA) [70] service leverages FPGAs [72] to accelerate query processing. The compilation as a service (CaaS) [15] service is for caching the code generation. The data of each cluster is managed in the Redshift managed storage (RMS) backed by the Amazon S3.

B. Disaggregated Compute-Memory-Storage OLAP

As shown in Fig. 10(b), the second architecture consists of three layers, a compute layer, a shuffle memory layer, and a storage layer. Similar to the first architecture, the compute layer has a service manager and a compute cluster. The main difference is that the compute cluster schedules the jobs for the workers in a centralized fashion. Moreover, it contains a shared memory pool to accelerate the shuffle process of complex operations such as aggregations and joins.

1) Motivation and Key features: There are three motivations. First, as memory is an expensive resource, it needs to be disaggregated and scaled independently for high elasticity. Second, it is preferable to achieve centralized scheduling, enabling better resource utilization for query processing. Third, when it comes to complex and costly workloads, it is challenging to cope with large intermediate results as the high I/O overhead is the bottleneck. In summary, this architecture features i) disaggregation of compute, memory, and storage, ii) shuffle memory layer for speeding up complex operations such as joins and aggregations, iii) multi-tenancy and serverless computing, iv) local SSD caching, and v) cloud storage service.

2) OLAP Workflow: For query processing, this architecture processes the data in parallel with multiple stages. Specifically, the worker nodes load the columnar data (e.g., ORC and Parquet files) from the shared storage, apply the filters locally, and send the data to the next stage. Then the system performs multiple shuffle operations to aggregate and sort the partial data by keys.

3) Pros and Cons: Compared to the on-premise share-nothing OLAP architectures, the disaggregated compute-memory-storage architecture has higher throughput, where the shuffle memory tier can significantly reduce I/O overhead by avoiding writing intermediate results to the disks. It has higher resource utilization as compute resources are virtualized and scheduled in a centralized way. Finally, since the compute, memory, and storage resources can be scheduled individually, it provides better elasticity. However, the major limitation of the second architecture is that shuffle memory tier could incur a high cost, so it needs to design efficient and effective pushdown and scheduling algorithms to reduce the data loaded to memory.

4) Representatives: A representative that adopts the three-tier architecture is BigQuery [59] which is built on the Dremel query engine [60]. It introduces a shared memory tier to accelerate the shuffle processing of the distributed joins, which significantly reduces the latency by avoiding writing and reading the intermediate results from disks. Moreover, it supports semi-structured data querying based on the Dremel query engine. Regarding storage management, it relies on the colossus file system [31]

with the capacitor format [59] that is similar to Parquet and ORC. For query processing, it adopts the producer-consumer model, where the producers in each worker generate partitions and send them to the in-memory nodes for shuffling, then the consumers combine the received partitions and do the operations locally. Another representative is Databricks Lakehouse [104], which support data analytic over the data lakes with Spark SQL [14] directly. It has also developed an ACID table storage layer over the cloud object store, called Delta lake [13], and a vectorized query engine, called Photon [18], which can integrate with the Spark SQL runtime.

C. Summary of the Cloud-Native OLAP Architectures

Table III presents a comparison of the cloud-native OLAP architectures concerning computation, storage, throughput, elasticity, isolation, and cost. The first category has the disaggregated compute-storage architecture. For the computation, it employs multiple clusters with various worker nodes. For the storage, it relies on local SSD caching and cloud storage. It has a high throughput based on scalable cloud computing. Its elasticity is also high because of the disaggregated architecture. Since the clusters are isolated and a query is typically only executed in one cluster, it has excellent performance isolation. By embracing the multi-tenancy with the elastic cloud service, it saves a large amount of cost for the cloud provider. The second category adopts the disaggregated compute-memory-storage architecture. For the computation, it employs multiple worker nodes with a shuffle memory layer. For the storage, it leverages the shared memory pool and the cloud storage. As the memory layer is disaggregated for shuffling, it has excellent throughput and elasticity. However, it leads to high costs due to the high price of in-memory computing. In addition, compared to the first category, it has lower performance isolation due to the shared memory pool.

V. CLOUD-NATIVE OLAP TECHNIQUES

This section introduces the key techniques of cloud-native OLAP databases in detail. Table IV summarizes five types of key techniques, including storage management, query processing, serverless computing, data protection, and machine learning. It also summarizes their pros and cons.

As shown in Fig. 12, storage management is the cornerstone of the cloud data service, which focuses on organizing and partitioning the data for optimizing the queries in the cloud. Query processing aims to handle queries with the local cache and the elastic cloud storage. By taking as input the SQL requests, serverless computing responds to each query by provisioning and scaling the resources on demand. Data protection relies on software-based or hardware-enabled techniques to protect data from stealing and tampering throughout the cloud service. Machine learning techniques include two parts: employing AI techniques to optimize the service quality of cloud-native DBMS (AI4DB) and harnessing the power of cloud-native DBMS to support AI.

TABLE III
A COMPARISON OF TWO CLOUD-NATIVE OLAP ARCHITECTURES

OLAP Architecture	Computation	Storage	Throughput	Elasticity	Isolation	Cost
Disaggregated Compute-Storage	Multiple Clusters with Worker Nodes	Local SSD Caching+ Cloud Storage	High	High	Excellent	Medium
Disaggregated Compute-Memory-Storage	Multiple Worker Nodes with Shuffle Layer	Shared Memory Pool+ Cloud Storage	Excellent	Excellent	High	High

TABLE IV
AN OVERVIEW OF KEY TECHNIQUES OF CLOUD-NATIVE OLAP DATABASES

| Technique Type	Main Approach	Cloud Database	Pros	Cons
Storage Management	Metadata Store Based Optimization	Snowflake [25]	High Throughput	Extra Cost
	Join Key-based Data Partitioning	Redshift [70]	High Efficiency	Cost Oblivious
	Columnar Format for Semi-Structured Data	BigQuery [59]	High Throughput	Storage Overhead
Query Processing	Columnar Scan with Pushdown	PushdownDB [103]	Low Cost	No Cache
	Scan with Caching and Pushdown	FlexPushdownDB [102]	High Throughput	Low Scalability
	Scan with Shuffle Memory Tier	BigQuery [59]	High Throughput	High Cost
Serverless Computing	Functions as a Service	Starling [73]	High Elasticity	Stateless Functions
	Serverless Databases	Athena [16]	High Throughput	High Cost
Data Protection	Key-based Data Protection	Snowflake [96]	High Scalability	Decrypted Access
	Enclave-based Data Protection	Azure [11]	High Security	Low Efficiency
Machine Learning	ML-enabled Cloud Data Service	Redshift [70]	High Quality	Low Adaptation
	SQL-based ML Pipeline	SageMaker [55]	High Elasticity	Training Overhead
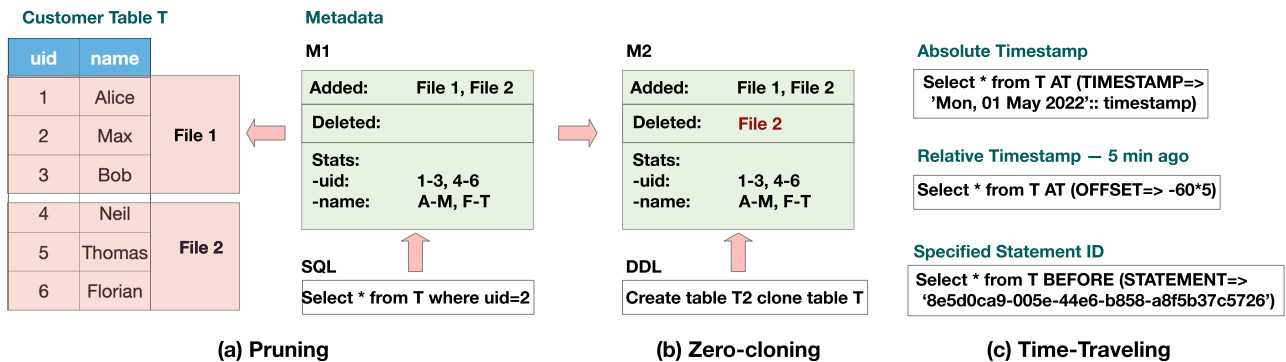
Fig. 11. Three key techniques based on metadata store.
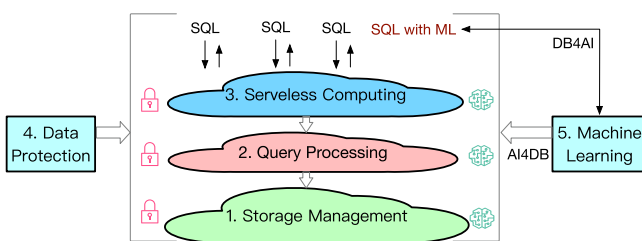

Fig. 12. An overview of cloud-native OLAP techniques.

A. Storage Management

We introduce three techniques of storage management: 1) metadata store based optimization; 2) join key based partitioning; and 3) column store for semi-structured data.

1) Optimization With Metadata Store: For cloud-native OLAP databases, metadata is managed in the layer of cloud service separately, which contains information for schema, data version, location, statistics, logs, etc. With metadata, the cloud databases can enable three optimizations: pruning, zero-copy cloning, and time traveling. Particularly, pruning means that the scanning data can be pruned without touching the underlying cloud storage; zero-copy cloning refers to cloning data without creating new copies; time traveling enables querying the historical data based on MVCC, which is similar to the flashback query in RDBMS. Fig. 11 depicts an example of each technique. Consider a customer table T, which is partitioned into two files and saved in the storage. The metadata file stores the range of uid and name for each file. Suppose a SQL query that requests the customer data with uid = 2. The metadata can be used to prune the data of file 2 because only file 1 covers the range of uid of 2. For the DDL operation that creates table T2 cloning from table T, the cloud database simply creates a new metadata file M2 from M1 without making physical copies of table files, namely, the zero-copy cloning technique (Note that at the time of cloning, file 2 has been deleted). Time-traveling technique utilizes timestamp information in the metadata. As shown in Fig. 11(c), the first two SQL queries find the data with an absolute and relative
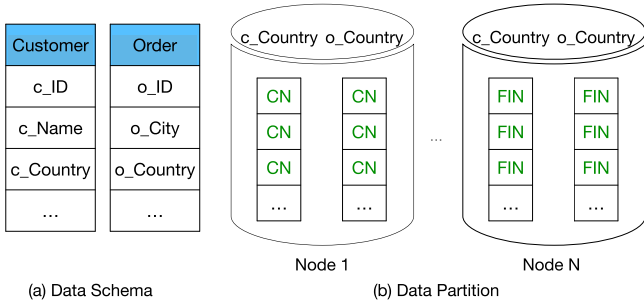
Fig. 13.   An example of data partition.

Fig. 14.   Three key techniques for query processing.

timestamp, respectively; the third query scans a versioned table with a specified statement ID.

For the pros, metadata-based optimization can largely improve query performance. However, the main challenge of metadata management are 1) how to serve the metadata request with super low latency; 2) how to provide the scalability of the metadata service.

2) Data Partitioning With Key Selection: Although cloud databases can always read persisted data from the cloud storage, the network traffic could become the bottleneck. Hence, how to organize the ephemeral data in the local cluster is also essential. To improve the query performance, one of the most important issues is to select the partition keys for large tables to distribute the data shards across the compute nodes. Take a schema in Fig. 13(a) as an example, it has a customer table and an order table, and these two tables can be joined on the country field. By partitioning both tables on the country field and placing the data partitions with the same hash value to the same node, it enables the join operation locally and can minimize network communication. However, selecting an optimal partition key set for the cloud databases is a non-trivial problem. First, existing partitioning solutions in distributed databases rely on tailored cost models [37] to which the customers have no access. Second, the cost models are inaccurate due to the uniform and independent assumption. There exist two solutions for cloud-native databases. The first one is the join graph approach [71] proposed by Redshift. Its basic idea is to build a multi-join graph based on a query workload. Then it performs random walks over the graph to select partition keys. In a join graph, each node represents a table; each edge denotes a join between two tables; the join weight on the edges denotes the join number from the queries. By randomly walking the join graph, it greedily selects the partition keys with the largest weight to collocate the joins in the same nodes. For the pros, it has high efficiency as it can efficiently build the join graph and can search for a solution in the graph. For the cons, it neglects the cost of different types of joins, leading to a suboptimal solution. The second method is to leverage deep reinforcement learning (DRL) [37] for selecting the partition keys. DRL can explore column combinations as partition keys and learns from the partitioning feedback, e.g., the reward. Such a method extracts partition features as a vector of tables, query frequencies, and foreign keys. Then it uses DQN models to partition the tables for a workload. To migrate the learned models to new workloads, it trains a cluster of Deep Q-Network models
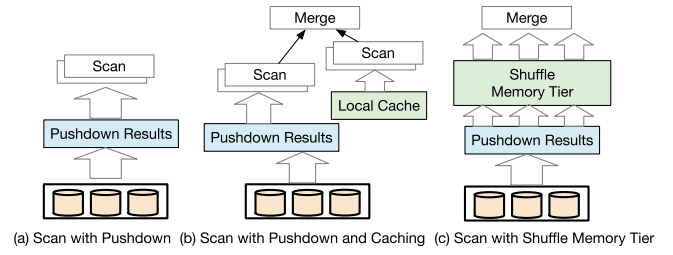
on typical workloads. Then it picks one with the most similar features for a new workload.

The major problem of the DRL-based method is the high training overhead. Since it needs to train the agent in an online fashion, it still consumes a large amount of time to make the learning process converge.

3) Columnar Format for Semi-Structured Data: Representing semi-structured data in a columnar format can speed up the query processing over the nested data [109], [110]. As semi-structured data such as HTML and JSON files are growing rapidly, it is crucial to manage a large amount of nested data in the cloud. There exist two major methods for encoding semi-structured data. The first method encodes the documents with lengths and presences of the fields, where the length implies the number of occurrences of each repeated field and the presence uses a boolean value to indicate whether or not an optional field is null. Two columnar formats, ORC and Apache Arrow [59], adopt such a representation. The second method encodes the documents with repetition levels and definition levels. Particularly, the repetition level tells which repeated field is changed compared to the previous record and the definition level indicates the length of the repeated or optional fields. Two columnar formats, Parquet and Capacitor [59], adopt such a representation.

There is a trade-off between the file size and query performance. To read a nested field, the first method requires access to its ancestor information, as only the ancestor field tracks the nested information. Nevertheless, it has a smaller file size as the information is denormalized in the separated tables. The second method can directly access the child fields without reading other tables as it repeats the ancestor information for each field. However, it has a larger file size due to the redundant information about the common ancestors. Besides the schema-based encoding method, there is a schema-less method [25] that can infer the data type and cluster the frequently-accessed paths automatically.

B. Query Processing

We introduce three types of query processing, including 1) columnar scan with pushdown [70], [96], [103], 2) columnar scan with caching and pushdown [102], and 3) columnar scan with shuffle memory pool [59]. As shown in Fig. 14, the first type loads the pushdown results from the cloud storage. The second one merges the results from both the pushdown results and the local cache. The third one loads the pushdown results from

the cloud storage, then performs the queries using the shuffle memory tier.

1) Columnar Scan With Pushdown: This type of query processing [20], [103], [106] aims to reduce network traffic by pushing down the computation to the storage side. A representative is Amazon Simple Storage Service (S3), which has exposed the Select API, by which users can specify the bucket and key of the S3 objects, then the unwanted data can be filtered with simple computations, such as selection and projection. When it comes to highly-selective operators, S3 Select can reduce a large amount of data in the storage side, thereby saving the computation cost on the compute layer. However, S3 Select does not mean that it is always cheaper than computing on normal EC2 nodes due to the more expensive pricing model for scanning ($0.002/GB) and returning data ($0.0007/GB).

PushdownDB [103] has studied the relation between the pushdown performance and its price. It particularly extended the S3 Select API to support more operations, including index scan, hash-join, group by, and top-k. For instance, it designed an offset index table based on S3 Select, which has the form of |indexed value | first_byte_offset | last_byte_offset|. Finding the objects involves two phases. First, the S3 objects are filtered using the index table and the offset of the target data is returned. Second, the data is fetched using the cheaper HTTP API instead of the S3 Select API. To push down the join, it builds a bloom filter for the join key of the small table, then adopts a substring-based matching strategy to perform the join using S3 Select.

Overall, these pushdown operations can have a lower cost and higher throughput regarding highly selective operators. Otherwise, it could have no payoffs due to the pushdown cost. Another drawback of the pushdown-only methods is that they make no use of the cache data.

2) Columnar Scan With Caching Pushdown: The second type of query processing is to scan the data with both caching and pushdown. The main idea is that since the local cache is more efficient than pushdown, it can be combined to further speed up the queries. FlexPushdown [102] is a representative of such a technique. Specifically, it consists of two parts: hybrid execution and cache replacement. For hybrid execution, it organizes the columnar data with segments and transforms the original query plan to a separable query plan with the consideration of the local cache and computation pushdown. For instance, suppose a scan query retrieves two attributes A and B, if all the segments of A are cached, these data can be scanned using local cache while the filters on segments of B are pushed down to the cloud storage, and finally the segments are merged at the compute nodes. Regarding cache replacement, it employs a weighted LFU strategy to manage the cache data. Intuitively, the larger the pushdown computation cost is, the larger weight the related data has for caching. As a result, it relies on a benefit-based caching framework by calculating a segment's weight $w(s) = (t_{net}(s) + t_{scan}(s) + t_{compute}(s))/size(s)$, where $t_{net}(s)$ is the time of network transfer, $t_{scan}(s)$ is the time of data scanning, and $t_{compute}(s))$ is the time of computation from the query. For the pros, it has high throughput as it can utilize local cache. However, it has low scalability due to the limited capacity of local cache.
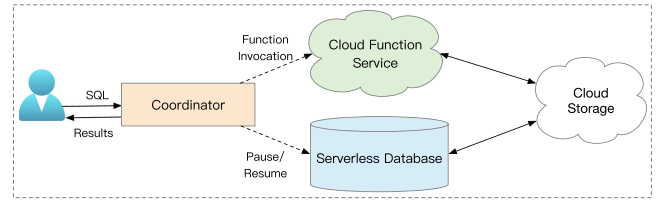

Fig. 15. An overview of serverless computing.

3) Columnar Scan With Shuffle Memory Tier: The third category uses a shuffle memory to perform queries. This technique is associated with the second disaggregated OLAP architecture. BigQuery [59] is a representative, which follows the map-reduce-style processing paradigm that partitions and processes the data with multiple phases. It adopts the producer-consumer execution model, where producers in each worker generate partitions and send them to the in-memory nodes for shuffling. Consumers in the next stage asynchronously combine the partitions and do the operations locally. For the shuffle phase of (n-1), workers use the consumers to receive partitions and use producers to generate new partitions. Then the distributed in-memory nodes conduct the shuffling. Regarding the shuffle phase of (n+1), the workers do the same operations with new consumers and producers. Finally, a single worker merges the results and returns to the coordinator. For the pros, it has high throughput as the shuffle phase is conducted using the shared memory. For the cons, it incurs high costs due to high pricing of in-memory computing.

C. Serverless Computing in Cloud Databases

Serverless computing is expected to be the next generation of cloud computing [84], which allows the programmers to write functions and code in the cloud without caring about server management, including resource provision and scaling, fault tolerance, and system monitoring. By combining cloud databases with serverless computing, users can enjoy the auto-scaling feature and pay for the used resources in a query granularity. Generally speaking, there are two implementations of serverless computing in cloud databases (see Fig. 15). The first type is i) serverless with functions as a service (FaaS) [64], [73], [91], where queries are adaptively executed by invoking the cloud function services. The second type is ii) serverless databases [16], [77], which automate the process of provisioning and scaling for the queries at the level of the database instance.

1) Functions as a Service: The first type of serverless computing technique [64], [73], [91] relies on serverless functions to process the queries. Particularly, the function as a service (FaaS) such as AWS Lambda [9], Azure Functions [62], Google Cloud Functions [33], allows to invoke multiple functions in a few milliseconds, and users are charged only for used resources. With FaaS, users could invoke many parallel jobs to scan, join, and aggregate tables in the cloud storage. As shown in Fig. 15, the workflow is as follows: users submit the SQL queries to a coordinator, which compiles the query and uploads the code to a cloud function service. Then, the coordinator schedules the tasks by provisioning resources and invoking them through the

function service. Afterward, the function service executes the tasks in the cloud.

This line of work is mainly driven by the research community. Two representatives are Starling [73] and Lambada [64], both of which build a query engine on top of the cloud function and storage service. Starling [73] implements the coordinator, which generates the C++ code for the specified query plan and invokes the AWS Lambda functions. The intermediate results are exchanged with the AWS cloud storage, i.e., S3. It makes two optimizations. First, it uses tuned models to detect stragglers, which increase the overall latency of parallel query processing. Then it invokes functions with duplicate computation. Second, it employs function-based combiners to reduce the overhead of large shuffling. Lambada [64] implements a part of TPC-H queries [29] using a Python front-end, and the code is generated based on its own compilation and execution framework. It uses three types of cloud storage service to exchange states: i) Amazon S3 for a large amount of data, ii) DynamoDB [88] for a small portion of data, and iii) Amazon Simple Queuing System (SQS) [5] for passing messages such as query results. To address the limitations of slow invocations of multiple tasks, it uses the two-level invocations that enable the first-level workers to invoke the second-level workers internally. Apart from the query processing, there exist works that focus on FaaS-based data analytics with specific programming languages (e.g., Python), such as Cloudburst [91] and general serverless computing runtime like NightCore [41].

There are two main challenges for FaaS-based query processing, First, since functions are stateless and cannot communicate with each other, their states are hard to keep and exchange. Simply using cloud storage often incurs large latency. Thus it calls for new methods for stateful serverless computing. There exists a number of works focusing on developing a unified storage service, including Pocket [45], Boki [40], Anna [99], and Jiffy [44]. However, they target general programming languages, and it is unclear how they can be applied and optimized for query processing. Second, it is challenging for users to decide how many resources (e.g., the number and size of the functions) should be obtained before performing the task [64], [73]. Therefore, how to balance the trade-off between cost and performance remains critical [43].

2) Serverless Databases: The second type of approach [76], [77], [85] supports serverless computing with database instances by dynamically scheduling the resources. This line of work is mainly led by commercial cloud data services, such as Aurora Serverless [6], Athena Serverless [16], and Azure Serverless [61]. These services have a tailored resource unit for scheduling. For instance, Aurora Serverless V2 [7] defines Aurora Capacity Unit (ACU), where the minimum unit is 0.5*ACU, and each ACU has 2 GiB memory (the CPU and network is the same as an instance's). Depending on the input size and the predicated resources, BigQuery [59] and AutoExecutor [85] can vary the number of executors for performing the tasks from multiple tenants. Four key operations in serverless computing are *provisioning, pausing, resuming, and scaling*, where provisioning aims to allocate the resources based on the issued queries; pausing stops the service tentatively and charges no fee
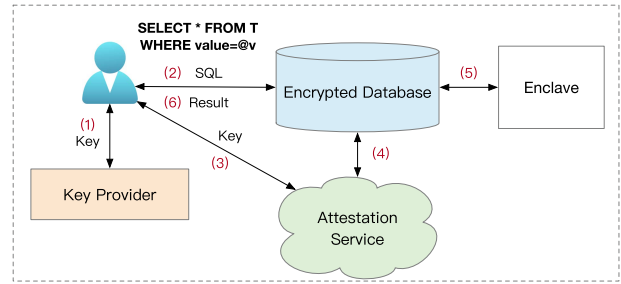

Fig. 16. Enclave-based query processing.

for users; resuming starts the service again with the provisioned resources; scaling allows for smoothly scaling up/down when the access pattern of workloads change. For provisioning and scaling, the main problem is to predict the required resources for a query workload. However, it is a challenging problem as even an expert can hardly estimate the resources needed for a given query [85]. For pausing and resuming, the main problem is to predict the arrival pattern of the workload. The main challenge is starting a database is expensive after a pause period, and resources could be wasted for a proactive resume period. Therefore, an adaptive model that can predict the pause/resume patterns is needed [76], [77].

D. Data Protection

Security is one of the most important issues in cloud databases. There are two main types of data protection techniques: software-based data protection [25] and hardware-based data protection [11].

1) Key-Based Data Protection: The first type of security method relies on key management services such as AWS CloudHSM [8] to manage the encryption keys for users. A representative is Snowflake [25], which utilizes an encryption key hierarchy that has four levels: root keys, account keys, table keys, and file keys. The keys are managed with life cycles and would be rotated and re-keyed periodically to ensure security. For instance, each key is rotated once per month, and data is re-keyed once per year in Snowflake. There are two main challenges for software-based data protection. First, data is decrypted for query processing. Second, the cloud vendors may be untrusted, so the keys may be stolen.

2) Enclave-Based Data Protection: The hardware-based data protection utilizes customized hardware, e.g., Enclave [10], for data protection. An enclave is a kind of Trusted Execution Environment(TEE), which has a virtual address space of a process that cannot be accessed by other processes, including operating system. Moreover, it assumes both database systems and cloud providers are untrusted, so it adopts a *bring-your-own-key* model, where only the data owners have the keys to access the encrypted data. Fig. 16 shows the design of enclave-based query processing: 1) the user requests a key from the key provider for the protected data (e.g., at a column granularity); 2) then the user issues a query "select * from T where value = @v" with the obtained key; 3) the attestation service verifies the key; 4) notifies the result to the encrypted database; 5) the

DBMS fetches the data and invokes the enclave for evaluation, and enclave will decrypt the data to plaintext and evaluates the filter; and 6) finally the query results are sent back to the user. There are two main challenges for hardware-based data protection. The first challenge is how to perform the computation over ciphertext directly, particularly for the range queries [2]. The second challenge is how to improve the efficiency and the scalability of the enclave due to its limited computing resources and space.

E. Machine Learning

Intersecting cloud-native databases with machine learning (ML) is another major trend for modern data-intensive applications. On the one hand, machine learning can benefit cloud-native databases by optimizing various database tasks [4], [15], [28], [37], [54], [100], [113], [114]. On the other hand, cloud-native databases can facilitate machine learning techniques with SQL-enabled ML pipelines [26], [30], [55].

1) ML-Enabled Cloud Data Service: Advanced ML techniques have been widely studied in the setting of cloud database tasks such as workload management [15], partition-key selection [37], knob tuning [4], [19], [54], [112], buffer size tuning [49], and index tuning [100]. For instance, AutoWLM [15] tunes the workload concurrency by predicting the memory consumption and execution time for the workload. It featurizes the query plans and trains an XGBoost [23] model for each cluster to predict the query performance. Ottertune [4] is an automatic knob tuning service that leverages Gaussian Process (GP) to tune the database configurations interactively. Further, CDBTune [112] and Qtune [54] employ the deep reinforcement learning (DRL) [56] to search for the optimal knobs in the large exponential space. Further, Hunter [19] combines the traditional ML techniques such as Genetic algorithm (GA) with DRL to address the cold-start problem, then it improves the performance based on multiple cloned database instances. Wu et al. [100] employs Monte Carlo Search Tree (MCTS) to build the indexes with the given budget of what-if calls. There are two main challenges for the ML-based cloud data service. First, most of the services are optimized independently. Thus it is hard to optimize the overall performance due to the interaction of the tuned components. Second, the machine learning models will become inaccurate due to the data drift or workload drift, and it is challenging to migrate a trained model to a new workload and dataset effectively and efficiently.

2) SQL-Based ML Pipeline: Using cloud-native databases for machine learning brings many benefits. First, it supports an SQL-enabled machine-learning pipeline backed with high elasticity and availability. Second, it brings the model to the data without additional data transferring overhead. Third, it supports AutoML [42], [82], [101] for the users, such as automatic model selection, training, and hyper-parameter tuning. For instance, Sagemaker [26], [55] supports the syntax of "Create Model" to train a model automatically, then it can make predictions with the SQL function. In order to perform ML inference locally, it invokes the Neo service to compile the model, and Neo transforms the machine learning models into inference code

and brings the models to the databases. BigQueryML [30] also enables a similar functionality, where users can leverage SQL tools to import, build and invoke advanced ML models based on TensorFlow [57].

VI. RELATED WORK

There is a general lack of a comprehensive survey on the cloud-native database as it is a relatively new field for both industry and academia. Particularly, Sakr [81] reviewed cloud-hosting databases. The survey discussed several topics, such as NoSQL databases, Database-as-a-Service (DaaS), and virtualized database servers. It also presented several future directions, including true elasticity, data consistency, live migration, SLA management, transaction support, and benchmarking. Mansouri [58] surveyed the storage management techniques in the cloud, namely, Storage as a Service (StaaS). The survey introduced cloud storage based on the intra-cloud and inter-cloud storage architectures. It also covered the topics of the data model, data replication, data consistency, transaction, and data management cost. Gartner [75] compared different cloud database systems from the business perspective. By weighing the business value with a set of evaluation criteria such as service quality and market record, the report classified the cloud DBMSs or cloud vendors into four roles in a Magic Quadrant, including niche players, visionaries, challengers, and leaders. It also discussed the strengths and weaknesses of each cloud DBMS. Narasayya et al. [65], [66] reviewed the cloud data services. The survey discussed various topics, including workloads and architectures, multi-tenancy and virtualization technologies, SLAs and pricing models, resource management, efficiency, and cost, as well as serverless databases.

Our work is different from existing surveys in three aspects. First, we classify the cloud-native databases into two types, OLTP-oriented and OLAP-oriented. We give a taxonomy for each type based on their disaggregation architecture and summarize their pros and cons. Therefore, our taxonomy is based on the architectures rather than the specific product. Second, our work covers a wide spectrum of advanced techniques developed by state-of-the-art cloud-native databases, including HTAP techniques, serverless computing, and machine learning. These update-to-date techniques are rarely reviewed and summarized in the previous work. Third, we give new future directions that existing works have not been discussed.

VII. OPEN PROBLEMS AND OPPORTUNITIES

Multi-Writer Architecture: Existing cloud databases only support a single writer and multiple readers, which may cause a large Recovery Time Objective (RTO) if the primary node has any failure. Besides, such an architecture has a limited capacity for highly-concurrent write transactions due to the single read-write (RW) node. Thus, it calls for cloud-native multi-writer techniques that can scale out write capabilities. Two promising architectures are 1) the share-storage architecture [97], [105] and 2) coherent cache architecture [69], where the former supports multiple RW nodes accessing the same storage with an RDMA network, and the latter enables the multi-writer with a

coherent cache layer. The challenge is to handle skewed write as the storage layer will accept write requests from multiple RW nodes [116].

Fine-Grained Serverless: Existing elastic databases mainly support provisioning the resources for a query with coarse-grained serverless (VMs or specific units, e.g., Aurora Capacity Unit). However, they are not cost-efficient and may suffer from the high latency of elastic scaling. One promising direction is to combine the advantage of FaaS-based Serverless and databases, where the former has a lower starting cost that can be used to address the cold-start problem, and the latter has a better performance. The challenge is to balance the trade-off between cost and performance.

SLA-Aware Cloud-Native HTAP: Existing cloud-native HTAP solutions only care about how to improve the HTAP performance, which may not be cost-efficient. For instance, transforming the row data to column data may accelerate query processing, but it also brings the higher dollar cost of memory computing. Two main challenges are 1) how to organize the data storage to achieve the best performance with the satisfied SLA [66], [81], and 2) how to judiciously schedule the resources for OLTP and OLAP workloads with SLA-aware optimization.

Multi-Cloud Data Service: As multi-cloud has become available, more and more data-intensive applications can benefit from using multi-cloud data services. However, it also poses new challenges to cloud-native databases with higher complexity. First, it is challenging to provide high availability as the data is stored across the cloud vendor. Thus, data migration in real time can largely affect availability [39]. Second, it is hard to maintain data consistency between the cloud vendors when the data is updated frequently. Third, it is challenging to have a cost-efficient execution plan for query processing as different cloud vendors have different pricing models. Even for the different regions in the same cloud vendor, the offering resources are disparate. A promising direction is sky computing [92], which aims to build an abstraction on top of inter-cloud services. For example, Skyplane [39] has been developed to facilitate data migration across clouds, and the SkyPilot [90] framework has supported the ML workload using multiple cloud providers such as AWS [5], Google Cloud [32], and Azure Cloud [17] simultaneously.

VIII. CONCLUSION

This paper offers a comprehensive survey of cloud-native databases. We summarize the state-of-the-art cloud-native architectures and techniques. We introduce three types of cloud-native OLTP architectures including 1) disaggregated compute-storage OLTP architecture, 2) disaggregated compute-log-storage OLTP Architecture, and 3) disaggregated compute-buffer-storage OLTP architecture. We also introduced their key techniques including data placement, storage layer consistency, compute layer consistency, multi-layer recovery, and HTAP optimization. Furthermore, we present two types of cloud-native OLAP architectures, including two-layered compute-storage OLAP architecture and three-layered compute-memory-storage OLAP architecture. We also summarize their key techniques regarding

storage management, query processing, serverless computing, data protection, and machine learning. Finally, we discuss the research challenges and opportunities for cloud-native databases, including multiple write architecture, fine-grained serverless, SLA-aware cloud-native HTAP techniques, and multi-cloud data service.

REFERENCES

[1] D. Abadi et al., "The seattle report on database research," *Commun. ACM*, vol. 65, no. 8, pp. 72–79, 2022.

[2] D. Agrawal, A. E. Abbadi, F. Emekçi, and A. Metwally, "Database management as a service: Challenges and opportunities," in *Proc. IEEE 25th Int. Conf. Data Eng.*, 2009, pp. 1709–1716.

[3] J. Aguilar-Saborit et al., "POLARIS: The distributed SQL engine in azure synapse," *Proc. VLDB Endowment*, vol. 13, no. 12, pp. 3204–3216, 2020.

[4] D. V. Aken, A. Pavlo, G. J. Gordon, and B. Zhang, "Automatic database management system tuning through large-scale machine learning," in *Proc. ACM Int. Conf. Manage. Data*, 2017, pp. 1009–1024.

[5] Amazon Web Service, "Amazon Web Services," 2023. [Online]. Available: https://aws.amazon.com/

[6] Amazon Web Service, "Aurora Serverless," 2023. [Online]. Available: https://aws.amazon.com/rds/aurora/serverless/

[7] Amazon Web Service, "Aurora Serverless V2," 2023. [Online]. Available: https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/aurora-serverless-v2.html

[8] Amazon Web Service, "AWS CloudHSM," 2023. [Online]. Available: https://aws.amazon.com/cloudhsm/

[9] Amazon Web Service, "AWS Lambda," 2023. [Online]. Available: https://aws.amazon.com/lambda/

[10] P. Antonopoulos et al., "Azure SQL database always encrypted," in *Proc. ACM Int. Conf. Manage. Data*, 2020, pp. 1511–1525.

[11] P. Antonopoulos et al., "Azure SQL database always encrypted," in *Proc. ACM Int. Conf. Manage. Data*, 2020, pp. 1511–1525.

[12] P. Antonopoulos et al., "Socrates: The new SQL server in the cloud," in *Proc. ACM Int. Conf. Manage. Data*, 2019, pp. 1743–1756.

[13] M. Armbrust et al., "Delta lake: High-performance ACID table storage over cloud object stores," *Proc. VLDB Endowment*, vol. 13, no. 12, pp. 3411–3424, 2020.

[14] M. Armbrust et al., "Spark SQL: Relational data processing in spark," in *Proc. ACM Int. Conf. Manage. Data*, 2015, pp. 1383–1394.

[15] N. Armenatzoglou et al., "Amazon redshift re-invented," in *Proc. ACM Int. Conf. Manage. Data*, 2022, pp. 2205–2217.

[16] AWS, "Severless interactive query service," 2023. [Online]. Available: https://aws.amazon.com/athena/

[17] Azure, "Azure cloud," 2023. [Online]. Available: https://azure.microsoft.com/en-us

[18] A. Behm et al., "Photon: A fast query engine for lakehouse systems," in *Proc. ACM Int. Conf. Manage. Data*, 2022, pp. 2326–2339.

[19] B. Cai et al., "HUNTER: An online cloud database hybrid tuning system for personalized requirements," in *Proc. ACM Int. Conf. Manage. Data*, 2022, pp. 646–659.

[20] W. Cao et al., "POLARDB meets computational storage: Efficiently support analytical workloads in cloud-native relational database," in *Proc. 18th USENIX Conf. File Storage Technol.*, 2020, pp. 29–41.

[21] W. Cao et al., "PolarFS: An ultra-low latency and failure resilient distributed file system for shared storage cloud database," *Proc. VLDB Endowment*, vol. 11, no. 12, pp. 1849–1862, 2018.

[22] W. Cao et al., "PolarDB serverless: A cloud native database for disaggregated data centers," in *Proc. ACM Int. Conf. Manage. Data*, 2021, pp. 2477–2489.

[23] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, B. Krishnapuram, M. Shah, A. J. Smola, C. C. Aggarwal, D. Shen, and R. Rastogi, Eds., 2016, pp. 785–794.

[24] T. Clarkson, D. Gorse, J. Taylor, and C. Ng, "Epidemic algorithms for replicated database management," *IEEE Trans. Comput.*, vol. 1, pp. 1552–1561, 1992.

[25] B. Dageville et al., "The snowflake elastic data warehouse," in *Proc. ACM Int. Conf. Manage. Data*, 2016, pp. 215–226.

[26] P. Das et al., "Amazon SageMaker autopilot: A white box AutoML solution at scale," in *Proc. 4th Int. Workshop Data Manage. End-to-End Mach. Learn.*, 2020, pp. 2:1–2:7.

[27] A. Depoutovitch et al., "Taurus database: How to be fast, available, and frugal in the cloud," in *Proc. ACM Int. Conf. Manage. Data*, 2020, pp. 1463–1478.

[28] B. Ding, S. Das, R. Marcus, W. Wu, S. Chaudhuri, and V. R. Narasayya, "AI meets AI: Leveraging query executions to improve index recommendations," in *Proc. ACM Int. Conf. Manage. Data*, 2019, pp. 1241–1258.

[29] M. Dreseler, M. Boissier, T. Rabl, and M. Uflacker, "Quantifying TPC-H choke points and their optimizations," *Proc. VLDB Endowment*, vol. 13, no. 8, pp. 1206–1220, 2020.

[30] Google, "What is BigQuery ML?" 2020. [Online]. Available: https://cloud.google.com/bigquery-ml/docs/introduction

[31] Google, "A peek behind colossus," 2021. [Online]. Available: https://cloud.google.com/blog/products/storage-data-transfer/a-peek-behind-colossus-googles-file-system

[32] Google, "Google cloud," 2023. [Online]. Available: https://cloud.google.com/

[33] Google, "Google function," 2023. [Online]. Available: https://cloud.google.com/functions/

[34] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: Research problems in data center networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, pp. 68–73, 2008.

[35] A. Gutmans, "Introducing AlloyDB for PostgreSQL: Free yourself from expensive, legacy databases," 2022. [Online]. Available: https://cloud.google.com/blog/products/databases/introducing-alloydb-for-postgresql

[36] T. Haerder and A. Reuter, "Principles of transaction-oriented database recovery," *ACM Comput. Surv.*, vol. 15, no. 4, pp. 287–317, 1983.

[37] B. Hilprecht, C. Binnig, and U. Röhm, "Learning a partitioning advisor for cloud databases," in *Proc. ACM Int. Conf. Manage. Data*, D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, and H. Q. Ngo, Eds., 2020, pp. 143–157.

[38] D. Huang et al., "TiDB: A raft-based HTAP database," *Proc. VLDB Endowment*, vol. 13, no. 12, pp. 3072–3084, 2020.

[39] P. Jain, S. Kumar, S. Wooders, S. G. Patil, J. E. Gonzalez, and I. Stoica, "Skyplane: Optimizing transfer cost and throughput using cloud-aware overlays," 2022, *arXiv:2210.07259*.

[40] Z. Jia and E. Witchel, "Boki: Stateful serverless computing with shared logs," in *Proc. ACM SIGOPS 28th Symp. Operating Syst. Princ.*, R. van Renesse and N. Zeldovich, Eds., 2021, pp. 691–707.

[41] Z. Jia and E. Witchel, "Nightcore: Efficient and scalable serverless computing for latency-sensitive, interactive microservices," in *Proc. 26th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2021, pp. 152–166.

[42] A. V. Joshi, "Amazon's machine learning toolkit: Sagemaker," in *Machine Learning and Artificial Intelligence*, Berlin, Germany: Springer, 2020, pp. 233–243.

[43] S. Kassing, I. Müller, and G. Alonso, "Resource allocation in serverless query processing," 2022, *arXiv:2208.09519*.

[44] A. Khandelwal, Y. Tang, R. Agarwal, A. Akella, and I. Stoica, "Jiffy: Elastic far-memory for stateful serverless analytics," in *Proc. 17th Eur. Conf. Comput. Syst.*, 2022, pp. 697–713.

[45] A. Klimovic, Y. Wang, P. Stuedi, A. Trivedi, J. Pfefferle, and C. Kozyrakis, "Pocket: Elastic ephemeral storage for serverless analytics," in *Proc. 13th USENIX Conf. Operating Syst. Des. Implementation*, 2018, pp. 427–444.

[46] L. Lamport, "The part-time parliament," *ACM Trans. Comput. Syst.*, vol. 16, no. 2, pp. 133–169, 1998.

[47] L. Lamport et al., "Paxos made simple," *ACM SIGACT News*, vol. 32, no. 4, pp. 18–25, 2001.

[48] J. Levandoski, D. Lomet, S. Sengupta, R. Stutsman, and R. Wang, "High performance transactions in deuteronomy," in *Proc. Conf. Innov. Data Syst. Res.*, 2015.

[49] F. Li, "Cloud native database systems at alibaba: Opportunities and challenges," *Proc. VLDB Endowment*, vol. 12, no. 12, pp. 2263–2272, 2019.

[50] G. Li, H. Dong, and C. Zhang, "Cloud databases: New techniques, challenges, and opportunities," *Proc. VLDB Endowment*, vol. 15, no. 12, pp. 3758–3761, 2022.

[51] G. Li and C. Zhang, "HTAP databases: What is new and what is next," in *Proc. ACM Int. Conf. Manage. Data*, 2022, pp. 2483–2488.

[52] G. Li, X. Zhou, and L. Cao, "AI meets database: AI4DB and DB4AI," in *Proc. ACM Int. Conf. Manage. Data*, 2021, pp. 2859–2866.

[53] G. Li, X. Zhou, and L. Cao, "Machine learning for databases," *Proc. VLDB Endowment*, vol. 14, no. 12, pp. 3190–3193, 2021.

[54] G. Li, X. Zhou, S. Li, and B. Gao, "QTune: A query-aware database tuning system with deep reinforcement learning," *Proc. VLDB Endowment*, vol. 12, no. 12, pp. 2118–2130, 2019.

[55] E. Liberty et al., "Elastic machine learning algorithms in Amazon Sage-Maker," in *Proc. ACM Int. Conf. Manage. Data*, 2020, pp. 731–737.

[56] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," in *Proc. Int. Conf. Learn. Representations*, 2016.

[57] N. Makrynioti, R. Ley-Wild, and V. Vassalos, "Machine learning in SQL by translation to tensorflow," in *Proc. ACM Int. Conf. Manage. Data*, 2021, pp. 2:1–2:11.

[58] Y. Mansouri, A. N. Toosi, and R. Buyya, "Data storage management in cloud environments: Taxonomy, survey, and future directions," *ACM Comput. Surveys*, vol. 50, no. 6, pp. 91:1–91:51, 2018.

[59] S. Melnik et al., "Dremel: A decade of interactive SQL analysis at web scale," *Proc. VLDB Endowment*, vol. 13, no. 12, pp. 3461–3472, 2020.

[60] S. Melnik et al., "Dremel: Interactive analysis of web-scale datasets," *Proc. VLDB Endowment*, vol. 3, no. 1, pp. 330–339, 2010.

[61] Microsoft, "Azure Cosmos DB Serverless," 2023. [Online]. Available: https://azure.microsoft.com/en-us/blog/build-apps-of-any-size-or-scale-with-azure-cosmos-db/

[62] Microsoft, "Azure function," 2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/azure-functions/functions-overview

[63] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz, "ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging," *ACM Trans. Database Syst.*, vol. 17, no. 1, pp. 94–162, 1992.

[64] I. Müller, R. Marroquín, and G. Alonso, "Lambada: Interactive data analytics on cold data using serverless cloud infrastructure," in *Proc. ACM Int. Conf. Manage. Data*, 2020, pp. 115–130.

[65] V. R. Narasayya and S. Chaudhuri, "Cloud data services: Workloads, architectures and multi-tenancy," *Found. Trends Databases*, vol. 10, no. 1, pp. 1–107, 2021.

[66] V. R. Narasayya and S. Chaudhuri, "Multi-tenant cloud data services: State-of-the-art, challenges and opportunities," in *Proc. ACM Int. Conf. Manage. Data*, 2022, pp. 2465–2473.

[67] V. R. Narasayya et al., "Sharing buffer pool memory in multi-tenant relational database-as-a-service," *Proc. VLDB Endowment*, vol. 8, no. 7, pp. 726–737, 2015.

[68] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, 2014, pp. 305–319.

[69] Oracle, "Oracle RAC," 2021. [Online]. Available: https://www.oracle.com/de/database/real-application-clusters/

[70] I. Pandis, "The evolution of Amazon redshift," *Proc. VLDB Endowment*, vol. 14, no. 12, pp. 3162–3163, 2021.

[71] P. Parchas, Y. Naamad, P. V. Bouwel, C. Faloutsos, and M. Petropoulos, "Fast and effective distribution-key recommendation for Amazon redshift," *Proc. VLDB Endowment*, vol. 13, no. 11, pp. 2411–2423, 2020.

[72] J. Paul, B. He, and C. T. Lau, "Query processing on OpenCL-based FPGAs: Challenges and opportunities," in *Proc. IEEE 24th Int. Conf. Parallel Distrib. Syst.*, 2018, pp. 937–945.

[73] M. Perron, R. C. Fernandez, D. J. DeWitt, and S. Madden, "Starling: A scalable query engine on cloud functions," in *Proc. ACM Int. Conf. Manage. Data*, 2020, pp. 131–141.

[74] M. Pezzini, D. Feinberg, N. Rayner, and R. Edjlali, "Hybrid transaction/analytical processing will foster opportunities for dramatic business innovation," *Gartner*, pp. 4–20, 2014.

[75] M. Pezzini, D. Feinberg, N. Rayner, and R. Edjlali, "Magic quadrant for cloud database management systems," *Gartner*, pp. 1–37, 2021.

[76] O. Poppe et al., "Seagull: An infrastructure for load prediction and optimized resource allocation," *Proc. VLDB Endowment*, vol. 14, no. 2, pp. 154–162, 2020.

[77] O. Poppe et al., "Moneyball: Proactive auto-scaling in Microsoft azure SQL database serverless," *Proc. VLDB Endowment*, vol. 15, no. 6, pp. 1279–1287, 2022.

[78] A. Prout et al., "Cloud-native transactions and analytics in singlestore," in *Proc. ACM Int. Conf. Manage. Data*, 2022, pp. 2340–2352.

[79] X. Qiu et al., "Real-time constrained cycle detection in large dynamic graphs," *Proc. VLDB Endowment*, vol. 11, no. 12, pp. 1876–1888, 2018.

[80] Randall Hunt, "S3 select and glacier select retrieving subsets of objects," 2018. [Online]. Available: https://aws.amazon.com/blogs/aws/s3-glacier-select/

[81] S. Sakr, "Cloud-hosted databases: Technologies, challenges and opportunities," *Clust. Comput.*, vol. 17, no. 2, pp. 487–502, 2014.

[82] S. K. K. Santu, M. M. Hassan, M. J. Smith, L. Xu, C. Zhai, and K. Veera-machaneni, "AutoML to date and beyond: Challenges and opportunities," *ACM Comput. Surv.*, vol. 54, no. 8, pp. 175:1–175:36, 2022.

[83] J. Schleier-Smith, "Serverless foundations for elastic database systems," in *Proc. Conf. Innov. Data Syst. Res.*, 2019.

[84] J. Schleier-Smith et al., "What serverless computing is and should become: The next phase of cloud computing," *Commun. ACM*, vol. 64, no. 5, pp. 76–84, 2021.

[85] R. Sen, A. Roy, and A. Jindal, "Predictive price-performance optimization for serverless query processing," in *Proc. Int. Conf. Extending Database Technol.*, 2023, pp. 118–130.

[86] N. Shamgunov, "The MemSQL in-memory database system," in *Proc. IMDM, VLDB*, 2014, pp. 106.

[87] R. M. Sheshadri Ranganath, "AlloyDB for PostgreSQL under the hood: Columnar engine," 2022. [Online]. Available: https://cloud.google.com/blog/products/databases/alloydb-for-postgresql-columnar-engine

[88] S. Sivasubramanian, "Amazon dynamoDB: A seamlessly scalable non-relational database service," in *Proc. ACM Int. Conf. Manage. Data*, 2012, pp. 729–730.

[89] D. Skeen, "A quorum-based commit protocol," Tech. Rep., Cornell Univ., Ithaca, NY, 1982.

[90] SkyPilot Team, "SkyPilot: Run jobs on any cloud, easily and cost effectively," 2023. [Online]. Available: https://skypilot.readthedocs.io/en/latest/

[91] V. Sreekanti et al., "Cloudburst: Stateful functions-as-a-service," *Proc. VLDB Endowment*, vol. 13, no. 11, pp. 2438–2452, 2020.

[92] I. Stoica and S. Shenker, "From cloud computing to sky computing," in *Proc. Workshop Hot Topics Operating Syst.*, 2021, pp. 26–32.

[93] J. Sun, J. Zhang, Z. Sun, G. Li, and N. Tang, "Learned cardinality estimation: A design space exploration and A comparative evaluation," *Proc. VLDB Endowment*, vol. 15, no. 1, pp. 85–97, 2021.

[94] A. Verbitski et al., "Amazon aurora: Design considerations for high throughput cloud-native relational databases," in *Proc. ACM Int. Conf. Manage. Data*, 2017, pp. 1041–1052.

[95] A. Verbitski et al., "Amazon aurora: On avoiding distributed consensus for I/OS, commits, and membership changes," in *Proc. ACM Int. Conf. Manage. Data*, 2018, pp. 789–796.

[96] M. Vuppalapati et al., "Building an elastic query engine on disaggregated storage," in *Proc. 17th Usenix Conf. Netw. Syst. Des. Implementation*, 2020, pp. 449–462.

[97] Q. Wang, Y. Lu, and J. Shu, "Sherman: A write-optimized distributed b tree index on disaggregated memory," in *Proc. ACM Int. Conf. Manage. Data*, 2022, pp. 1033–1048.

[98] R. Wang, J. Wang, S. Idreos, M. T. Özsu, and W. G. Aref, "The case for distributed shared-memory databases with RDMA-enabled memory disaggregation," 2022, *arXiv:2207.03027*.

[99] C. Wu, V. Sreekanti, and J. M. Hellerstein, "Autoscaling tiered cloud storage in anna," *Proc. VLDB Endowment*, vol. 12, no. 6, pp. 624–638, 2019.

[100] W. Wu et al., "Budget-aware index tuning with reinforcement learning," in *Proc. ACM Int. Conf. Manage. Data*, Z. Ives, A. Bonifati, and A. E. Abbadi, Eds., 2022, pp. 1528–1541.

[101] A. Yakovlev et al., "Oracle AutoML: A fast and predictive automl pipeline," *Proc. VLDB Endowment*, vol. 13, no. 12, pp. 3166–3180, 2020.

[102] Y. Yang et al., "FlexPushdownDB: Hybrid pushdown and caching in a cloud DBMS," *Proc. VLDB Endowment*, vol. 14, no. 11, pp. 2101–2113, 2021.

[103] X. Yu et al., "PushdownDB: Accelerating a DBMS using S3 computation," in *Proc. IEEE Int. Conf. Data Eng.*, 2020, pp. 1802–1805.

[104] M. Zaharia, A. Ghodsi, R. Xin, and M. Armbrust, "Lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics," in *Proc. Conf. Innov. Data Syst. Res.*, 2021.

[105] E. Zamanian, C. Binnig, T. Kraska, and T. Harris, "The end of a myth: Distributed transaction can scale," *Proc. VLDB Endowment*, vol. 10, no. 6, pp. 685–696, 2017.

[106] C. Zhan et al., "AnalyticDB: Real-time OLAP database system at alibaba cloud," *Proc. VLDB Endowment*, vol. 12, no. 12, pp. 2059–2070, 2019.

[107] C. Zhang, G. Li, and T. Lv, "HyBench: A new benchmark for HTAP databases," *Proc. VLDB Endowment*, vol. 17, no. 5, pp. 939–951, 2024.

[108] C. Zhang, G. Li, J. Zhang, X. Zhang, and J. Feng, "HTAP databases: A survey," *IEEE Trans. Knowl. Data Eng.*, early access, Apr. 19, 2024, doi: 10.1109/TKDE.2024.3389693.

[109] C. Zhang and J. Lu, "Selectivity estimation for relation-tree joins," in *Proc. 32nd Int. Conf. Sci. Stat. Database Manage.*, 2020, pp. 1–12.

[110] C. Zhang and J. Lu, "Holistic evaluation in multi-model databases benchmarking," *Distrib. Parallel Databases*, vol. 39, no. 1, pp. 1–33, 2021.

[111] C. Zhang, J. Lu, P. Xu, and Y. Chen, "UniBench: A benchmark for multi-model database management systems," in *Proc. Technol. Conf. Perform. Eval. Benchmarking*, Springer, 2018, pp. 7–23.

[112] J. Zhang et al., "An end-to-end automatic cloud database tuning system using deep reinforcement learning," in *Proc. ACM Int. Conf. Manage. Data*, 2019, pp. 415–432.

[113] J. Zhang, C. Zhang, G. Li, and C. Chai, "AutoCE: An accurate and efficient model advisor for learned cardinality estimation," in *Proc. IEEE 39th Int. Conf. Data Eng.*, 2023, pp. 2621–2633.

[114] J. Zhang, C. Zhang, G. Li, and C. Chai, "PACE: Poisoning attacks on learned cardinality estimation," *Proc. ACM Manage. Data*, vol. 2, no. 1, pp. 1–27, 2024.

[115] Y. Zhang et al., "Towards cost-effective and elastic cloud database deployment via memory disaggregation," *Proc. VLDB Endowment*, vol. 14, no. 10, pp. 1900–1912, 2021.

[116] T. Ziegler, P. A. Bernstein, V. Leis, and C. Binnig, "Is scalable OLTP in the cloud a solved problem?," in *Proc. Conf. Innov. Data Syst. Res.*, 2023.

Haowen Dong received the bachelor's degree in computer science from Tsinghua University. He is currently working toward the PhD degree with Tsinghua University. His research interests focus on cloud-native databases.

Chao Zhang received the PhD degree in computer science from the University of Helsinki, Finland. He is a postdoctoral researcher with Tsinghua University. He has given a tutorial on HTAP databases in SIGMOD 2022 and gave a tutorial on cloud databases in VLDB 2022. He serves as a PC member of SIGMOD 2024-2025, VLDB 2023-2024 Tutorial, and ICDE 2023. His research interests focus on heterogeneous database management systems.

Guoliang Li (Fellow, IEEE) is a full professor with the Department of Computer Science, Tsinghua University. His research interests include database systems, large-scale data cleaning and integration. He received VLDB 2017 early research contribution award, TCDE 2014 early career award, Best of SIGMOD 2023, SIGMOD Research Highlight Award, VLDB 2023 Industry Best Paper Runner-up, DASFAA 2023 Best Paper Award, CIKM 2017 best paper award, and ICDE 2018 best papers. He served as a general chair of SIGMOD 2021, a demo chair of VLDB 2021, and an industry chair of ICDE 2022.

Huanchen Zhang received the PhD degree from Computer Science Department, Carnegie Mellon University. He is an assistant professor in the IIIS (Yao Class) with Tsinghua University. His research interest is in database management systems with particular interests in indexing data structures, data compression, and cloud databases. He is the recipient of the 2021 SIGMOD Jim Gray Dissertation Award.