OceanBase Unitization: Building the Next Generation of Online Map Applications

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

Wei Sun*
AMap, Alibaba
shengweng.sw@autonavi.com

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

Jinlong Liu
AMap, Alibaba
fuchen.ljl@autonavi.com

Ziyun Wei
Cornell University
zw555@cornell.edu

Fusheng Han
OceanBase, Ant Group
yanran.hfs@oceanbase.com

Liang Wang
AMap, Alibaba
wl275647@autonavi.com

Xiaowei Zhai
AMap, Alibaba
zhaixiaowei.zxw@autonavi.com

***Abstract*—Distributed database systems are extensively utilized to provide cloud services for online map platforms, offering consistency, disaster recovery, and high performance, whereas traditional systems relying on singly-homed architecture face challenges in scaling for large-scale services. In this paper, we propose the architectural design of OceanBase (OB), a distributed database system that "unitizes" services and operations into individual machines. The unitization approach migrates from a singly-homed to a multi-homed design across multiple regions. By leveraging this feature, OceanBase ensures data replication and seamless service handover when a machine goes offline. However, communication overhead between regions can sometimes be burdensome. To address this issue, OceanBase unitizes read and write operations, and employs a hybrid centralization and unitization approach that is dynamically optimized for Online Transaction Processing (OLTP) and Online Analytical Processing (OLAP). To validate our design, we deploy OceanBase on AMap, an online map application platform supporting large-scale distributed services. Through a series of experiments, we demonstrate that OceanBase exhibits enhanced disaster tolerance capabilities and achieves improved performance for both write-intensive and read-intensive benchmarks.**

I. Introduction

With the rapid growth of data volume, the use of distributed database systems has become the prevailing trend in today's Internet industry. Leading Internet companies, such as Google and Alipay, have developed their own distributed database systems [1]–[3]. In a distributed scenario, the database system is deployed across multiple nodes and interconnected by a network, creating a logically unified database system. By distributing the workload across multiple nodes, they can handle concurrent transaction requests with minimal latency, making them highly efficient in serving a large number of users and supporting robust business operations.

In distributed systems, high availability and high performance have become two critical metrics for evaluating effectiveness. In a standalone database system, failure of a single server can result in significant downtime and block subsequent query requests. This limitation makes single-point servers and applications susceptible to bottlenecks. To address this issue, many systems employ disaster recovery techniques such as adding machines and splitting applications. For example, in numerous financial business scenarios, the horizontal expansion model across multiple Internet data centers (IDCs) has been widely adopted. However, for systems with hundreds of millions of users, relying solely on IDC-level disaster recovery may prove insufficient. It becomes imperative to consider disaster recovery on the deployment level by distributing IDCs across geographically distinct areas, safeguarding against potential threats such as earthquakes and tsunamis. Normally, typical systems such as bank systems apply the classic "two cities and three centers" requirement for IDC deployment.

Disaster recovery plays a crucial role in ensuring a system's resilience in the face of catastrophic damage. However, it is equally important for a system to be capable of handling large-scale requests promptly, minimizing latency for end-users. Even though single-point reading and writing can guarantee data consistency, they will lead to performance bottlenecks with a high volume of requests. To address this, many systems opt to distribute data and computational resources across different nodes, enabling the distribution of services. Each user's request is directed to an independent node, where read and write operations are executed on partitioned data within that node. For some sophisticated cases, a user's service may involve multiple requests, each of which can be routed to a different node. Hence, accessing a geographically proximate node becomes crucial in significantly enhancing response speed, thereby improving overall system performance.

Building upon the requirements just described, we introduce the concept of "OceanBase unitization". The term "unit" refers to a self-contained ensemble that can execute all business operations. This ensemble encompasses all necessary services for various businesses. The "unit" is employed as the fundamental deployment entity, and multiple units are distributed across all IDCs within the entire site. Each unit handles all applications required by the system, with the data being a partitioned subset of the dataset based on certain criteria. In contrast to the traditional service-oriented architecture (SOA), where services are organized in layers with varying numbers of nodes, the unitized architecture retains an integrated structure. However,

* These authors contributed equally to this work.
† Chuanhui Yang is the corresponding author.

2375-026X/25/$31.00 ©2025 IEEE
DOI 10.1109/ICDE65448.2025.00312

in this case, each node within a layer exclusively belongs to a specific unit. When an upper layer invokes a lower layer, only nodes within the same unit are selected. This approach facilitates the implementation of a multi-active solution in the unitized architecture, effectively isolating system deployment and core traffic. Consequently, it enables more flexible scalability and robust disaster tolerance solution.

From an availability perspective, data are logically and physically partitioned into sub-databases and tables that are assigned to each unit. This enables requests sent to a failed unit to be handled by another active unit, ensuring continuous service availability. Requests tend to access the geographically proximate units according to a routing policy. In terms of performance, concurrent requests are efficiently distributed across multiple units, achieving a balanced workload. In addition, depending on the specific application scenarios, read and write (R/W) operations can be either unitized or centralized. To reduce synchronization overhead for OLAP workloads, we strategically transition to centralized writing. This flexible design maximizes concurrent transaction throughput and enables high-performance query responses in Big Data environments.

AMap [4], a leading Chinese provider of mobile digital maps, navigation, and real-time traffic information services, offers users a comprehensive platform for various services such as navigation and ride-hailing. To handle its significant volume of user requests, AMap relies on distributed database systems. Previously, AMap utilized MongoDB [5], but it encountered issues such as high CPU usage and service interruptions due to inefficiencies in the distributed document storage system. To overcome these challenges, AMap migrated its services to Lindorm [6] and Elasticsearch (ES) [7], which improved retrieval tasks but lacked the capability to execute full SQL-like queries as in traditional relational databases. As a result, designing a cloud-native and industry-independent architecture has become the future direction for AMap servers. Specifically, for AMap's core services, the key areas of research and development areas include providing cross-regional disaster recovery and ensuring high throughput during high concurrent request situations. This study primarily focuses on deploying OceanBase [2] on AMap's application platform, using a unitized architecture. The contributions of this paper can be summarized as follows:

1) Disaster recovery: OceanBase's unitization, and three-site five-center deployment provide seamless unit switching to ensure that there is no data loss or interruption, achieving high data availability and business continuity.
2) High performance: OceanBase enables users to access data centers that are geographically close to them, reducing latency and minimizing network congestion. To enhance high throughput for R/W requests, it implements a combination of unitized and centralized approaches.
3) Consistency: It implements replication synchronization among different units to guarantee data consistency.
4) We deploy the unitized OceanBase systems on AMap for many applications, providing high availability and throughput for OLAP and OLTP. OceanBase's unitized architecture has shown significant performance advantages in AP and TP, outperforming its competitors, showcasing strong data processing capabilities and efficient query mechanisms.

This paper is organized as follows. §II presents OceanBase-based AMap business scenarios. §III provides an overview of the OceanBase distributed database system. §IV describes the design of OceanBase unitization. We present the experiments in §V to prove the high availability and high performance of OceanBase unitization built for AMap applications. We present lessons learned in §VI, and review the related work in §VII. Finally, we state the conclusions in §VIII. OceanBase is an open-source project under Mulan Public License 2.0 [8] and the source code referenced in this paper is available on both Gitee [9] and GitHub [10].

II. OceanBase-based AMap Business Scenarios

We introduce three typical business scenarios, where OceanBase addresses the challenges posed by each scenario using a unitized architecture.

A. Use Case 1: Strong Consistent Finance

The AMap finance and settlement service primarily offers the settlement and financial data services, which require strong data consistency, data loss prevention, and cross-regional disaster recovery capabilities. As shown in Figure 1, user data, monetary flows, and information flows are controlled by the business channels, financial channels, and financial accounting modules, respectively.

AMap offers a variety of services, with each complete service consisting of multiple sub-services that form a complex business chain. The business modules control the interaction between services, monitor and manage the various states of the business processes. In the event of a failure in any part of the business chain, AMap's system must promptly detect the faulty service, restart any unfinished processes, and ensure the normal execution of business processes from the user's perspective.

In terms of monetary and information flows, the finance and settlement service plays a crucial role in calculating payments with B-side merchants. It requires a high level of data consistency to avoid errors in payment calculations arising from inconsistency. Thus, to enhance the availability of the finance and settlement system, AMap requires a database system that supports disaster recovery across regions and provides strong data consistency.

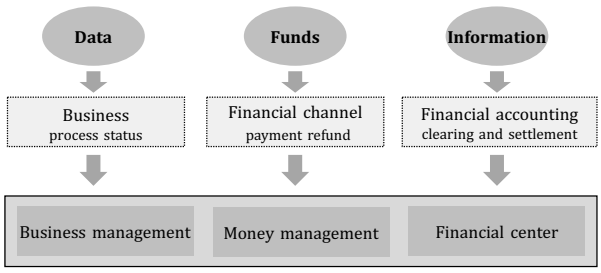

Fig. 1. Financial settlement business workflow.

Example 1. *When a user places a transaction order, the upstream e-commerce business interacts with downstream suppliers, generating transaction and order data. As an order is created, funds are calculated and disbursed to downstream suppliers. To enhance the consumer experience, the service offers prepayment to users and safeguards against consumer losses caused by problems with downstream fund services. All these services involve a circulation of funds, necessitating strong consistency in reading and writing data to ensure data integrity and prevent situations where multiple or incorrect payments occur due to data synchronization delays.*

B. Use Case 2: Cloud Synchronization (OLTP)

AMap cloud synchronization is a fundamental business service and is responsible for cloud storage of data and synchronization across multiple devices. As shown in Figure 2, different end-point systems (mobile phones, tablets, in-car systems) simultaneously access the cloud system from different nodes and perform multiple read and write operations on cloud data (such as user location information, user account information, etc). In the corresponding business scenarios, it is necessary to ensure accurate data writing across multiple devices, timely updates, and fast data retrieval after switching devices. As AMap's business rapidly expands, the cloud synchronization system must store a vast amount of data. Therefore, in AMap's cloud synchronization service, the database system must support massive data storage, enable multiple point writing, and ensure excellent performance for read and write operations.

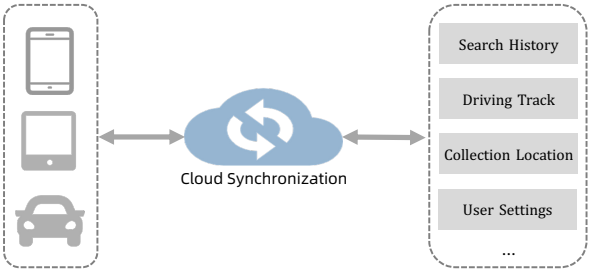

Fig. 2. Cloud synchronization for multi-device data.

Example 2. *Users mark and save places of interest on AMap. They have the option to create multiple favorite directories, which enable them to view and manage their collections at any time. With this feature, users can easily navigate and plan routes to their saved locations, facilitating the discovery and navigation to their favorite places during their daily travels. In addition, users can share their favorite places and collections with other users. AMap provides services for managing and sharing geographic information, which requires efficient write operations to ensure that the information is up-to-date using real-time synchronization.*

C. Use Case 3: Online Review Service and High-Definition Map (OLAP)

The AMap review service is an information platform based on user evaluations that aims to provide users with more accurate and practical review information and assist them in making better choices. It consists of two main aspects: user comments and merchant responses. Users can evaluate merchants on AMap platforms, providing ratings and comments on aspects such as service quality, product quality, and environment. Merchants can respond to user comments, explain and resolve issues, and communicate with users through the AMap platform interface.

Normally, the contents of comments are monitored by the Content Moderation Service, and therefore write transactions per second (TPS) may not be very high. However, the content will be displayed across various end points within AMap, requiring a short response time (RT). Unlike Use Case 2, the review service involves fewer write operations than read operations. As a result, we are motivated to centralize the write operations, aiming to minimize synchronization overhead. Generally, the overall RT must be controlled within 15ms for optimal performance. Figure 3 presents a pain point analysis of the AMap review service and OceanBase solution.

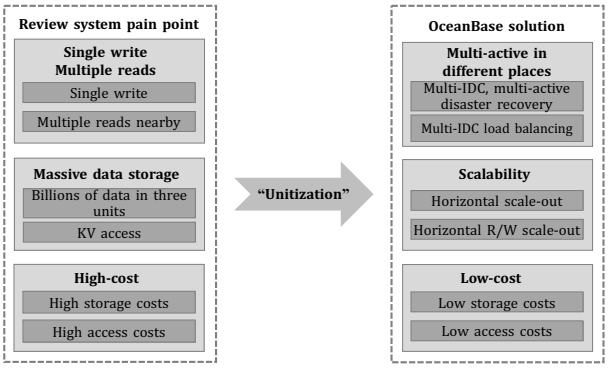

Fig. 3. Review service pain point analysis and OceanBase solution.

Example 3. *In the AMap review service, merchants can post articles describing their products, and users can browse these posts and comments on those that pique their interest. AMap's rating system ensures that user reviews are updated in real-time, enabling other users to access the most current evaluation information simultaneously. Merchants can utilize the AMap platform to respond to user reviews, resolving issues and enhancing the quality of their services. Given this interaction model, the frequency of user posts is notably lower than the volume of post browsing. As a result, it is essential to provide an efficient read operation to cater to the needs of a large user base.*

AMap High-Definition Map is another OLAP service, and it is a highly accurate geographical information service that provides centimeter-level positioning accuracy and detailed road and traffic data, including 3D modeling and real-time dynamic information. This kind of map is a key foundation for autonomous driving, advanced driver-assistance systems (ADAS), intelligent transportation systems, and urban planning and location services. Its high accuracy and detailed data are crucial for the development of intelligent transportation and autonomous driving technologies.

III. OceanBase Overview

In this section, we introduce the basic architecture of OceanBase from two aspects: the storage and transaction engines.

A. Storage Engine

Its storage engine leverages an LSM-tree structure, dividing data into static SSTables and dynamic MemTables. It uses a hybrid row-column format and stores data in fixed-size 2MB macro blocks comprising several 16KB micro blocks, which are the smallest I/O units containing multiple data rows. It adopts user-defined data compression in micro blocks, offering over 10× compression ratio compared to traditional methods and supporting various encoding techniques suitable for columnar compression. It also features B+Tree and Hash indexing, optimizes for hot data access, and ensures high query performance. It uses a shared-nothing architecture with multi-version concurrency control, relying on the Paxos protocol for consistency, eliminating single points of failure, and ensuring system availability. It scales dynamically, rebalancing data across nodes after expansion without impacting upper services.

B. Transaction Engine

OceanBase employs a concurrency control model that ensures high-performance, scalability, and low-latency transactions with built-in high availability and disaster recovery. As shown in Figure 4, tables are distributed across nodes in different partitions, with fault tolerance enabled by the Paxos protocol. To maintain external consistency, it uses a Global Timestamp Service (GTS) with peer-to-peer nodes, where each node keeps a Global Timestamp Cache. OceanBase offers snapshot read and read-committed isolation levels for consistency in distributed systems. It uses a multiversion storage approach, allowing transactions to be tagged with global read and commit version numbers. New versions are stored in memory for updates along with the transaction's global commit version number.

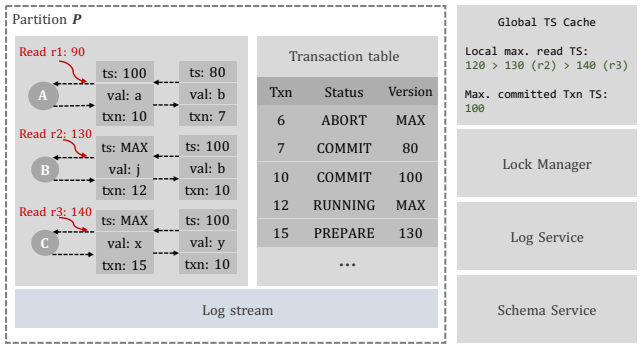

Fig. 4. Multi-version read consistency implementation.

Updates in a partition are tracked with version, value, and transaction ID, allowing for multiple versions. Transactions are tracked in a memory table with their ID, status, and version. Consistency protocols ensure log synchronization across the majority of nodes for a partition. Lock Manager orchestrates concurrent transaction access to prevent conflicts and ensure data integrity, while Log Service [11] manages operation logs for persistence and recovery, enhancing reliability and fault tolerance. Schema Service maintains metadata and schema information, facilitating database structure management in response to business changes.

IV. System Design

According to the three typical business scenarios described in Section II, OceanBase employs a unitization architecture, providing robust resilience capabilities, ensuring high data consistency and improving concurrent R/W efficiency.

A. Unitization Overview

Commonly, a distributed architecture in the industry mainly includes single-active, dual-active, multi-active, and cold backup architectures. From the perspective of disaster recovery capabilities, these architectures can achieve cross-IDC disaster recovery at application level. However, because they use asynchronous replication technology, databases cannot achieve zero recovery point objective (RPO) at the data center level.

Multi-Site High Availability (MSHA) is a robust deployment architecture for distributed systems. The unitization architecture of logical data centers is the solution for OceanBase to achieve MSHA. The term "unit" refers to a self-contained collection that can perform all business operations, including their services and allocated data. The unitization architecture takes the unit as the basic deployment unit, deploying several units in each data center. Optimal routing policies are employed to efficiently handle requests that access multiple units across various data centers, minimizing network overhead. If one unit fails, other units seamlessly take over incoming requests, ensuring high availability. By deploying units across different regions, OceanBase can easily support availability in regional level. Data are partitioned based on specific criteria, enabling multi-point read and write operations and mitigating centralized operation bottlenecks. To accommodate the growing number of user requests, the deployment can be horizontally expanded by adding more units. This facilitates system scalability, effectively handling increased user demand.

From the unitization perspective, there are three types of data in a system:

Partitionable Data: Data can be partitioned according to a selected dimension, which can truly be unitized. This type of data is usually the core of unitization architecture. Examples include order, payment transaction, and account data. The higher the proportion of this type of data in the system, the higher the degree of unitization will be. If the system consists entirely of this type of data, a perfect unitization architecture can be built.

Global Data (Not Frequently Accessed by Key Business Chains): these data cannot be partitioned and only exist globally. A typical example is configuration data, which may be accessed by key business chains, but not frequently. Therefore, even if the access speed is not fast enough, this will not significantly impact business performance.

Global Data (Frequently Accessed by Key Business Chains): this type of data is similar to the previous type, but

differs in that they are frequently accessed by key business chains. If the system is not deployed in a geo-distributed manner, the data have little impact on the distributed system. However, if we want to achieve MSHA by means of unitization, this type of global data will be accessed by different units, resulting in additional overheads.

As shown in Figure 5, the unitization architecture in OceanBase includes Region Zone (RZone), Global Zone (GZone), and City Zone (CZone). The GZone deploys data and business activities that cannot be split, whereas RZones deploy business and corresponding data that can be split. A GZone is globally deployed only once, and is relied upon by RZones. Each data shard within an RZone has five replicas, achieving a three-site, five-IDC deployment. Each shard has only one writable primary replica; the remaining replicas maintain strong data consistency using the Paxos protocol [12]. Each RZone achieves unit encapsulation, independently handling all its own operations. CZones emerge because the GZone is globally deployed only once, but RZones in different cities may require remote access to GZone services and data, resulting in significant latency. Therefore, a CZone is deployed in each city as a read-only replica of the GZone, providing services to the RZone in that city.

The unitization architecture accomplishes application and data splitting in a distributed architecture. In geographically distributed scenarios, it resolves remote access latency issues through traffic routing. In multi-user scenarios with multi-point reading and writing, the unitization architecture decides whether to centralize write operations based on different application requirements, ensuring high throughput in both OLTP and OLAP.

B. Unitization for Disaster Recovery

Financial services require strong data consistency and cross-region disaster recovery. Considering the costs of deployment and availability, we choose an architecture with five data centers across three cities. Furthermore, this scalable architecture allows for horizontal expansion, enabling the addition of more data centers across multiple cities.

OceanBase enables financial data to be simultaneously written to multiple regional data replicas, ensuring availability of complete data even if one region becomes unavailable. The Paxos consensus protocol in OceanBase guarantees that a response is successful only when multiple replicas of data have been successfully written, ensuring strong data consistency for distributed replicas.

1) Deployment of five centers across three cities: Maximizing the proportion of RZones and minimizing the GZone can improve the efficiency of a unitization architecture system. In a typical unitization architecture system with three cities and five centers, the units can be deployed as shown in Figure 5.

Firstly, we deploy RZones in the two main cities and the last RZone in the third city to run sharded applications. We ensure that at least four sets of RZones are deployed, with each set being deployed across data centers in the two cities (i.e., City 1 and City 2), and the fifth RZone is deployed

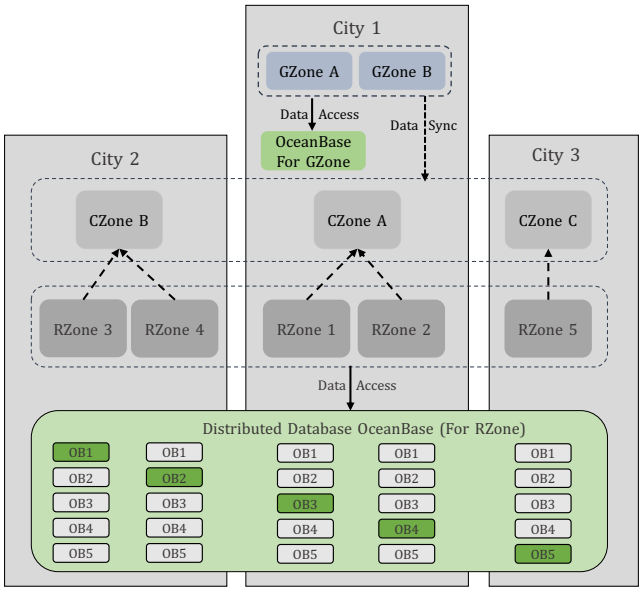

Fig. 5. Architecture of OceanBase unitization.

in the third city (i.e., City 3). The GZone can handle all public services, reducing deployment costs, development costs, and operating costs. We deploy the primary GZone in one of the main cities (City 1) to handle non-horizontal scalable businesses. Optionally, we can deploy a backup GZone in the other main city (City 2) as a remote disaster recovery point for the primary GZone. The backup GZone is designed to ensure the availability of disaster recovery capabilities and can handle a small amount of daily business traffic. To simplify the overall architecture, we introduce a CZone, solely for backup purposes. More practically, if it is observed that the cross-city access from the RZone in City 2 or the backup GZone to GZone data in City 1 appears to cause an intolerable impact on business operations (such as excessively long response time), a decision will be made to trigger the deployment of an additional CZone. This additional deployment helps to alleviate any adverse effects on network conditions and maintain optimal performance.

Example 4. *Taking the example shown in Figure 5 as an illustration, when there is a failure in RZone 1, we switch RZone 1 to RZone 2, realizing local disaster recovery. Database sharding is performed initially, and the replica of Shard 1 in RZone 2 is promoted to the primary replica. After promotion is complete, the traffic from RZone 1 is switched to RZone 2, achieving local disaster recovery with a Recovery Point Objective (RPO) of 0 and a Recovery Time Objective (RTO) of less than 8 seconds. Similarly, for remote disaster recovery, if RZone 1 experiences a failure, the replica in RZone 3 is switched to the primary replica, taking charge of future requests. This process also achieves an RPO of 0 and an RTO of less than 8 seconds.*

2) Write-Ahead Log: OceanBase ensures data correctness and integrity during system crashes through a write-ahead log

(WAL) mechanism. OceanBase possesses native capabilities to support concurrent writes to multiple data replicas without synchronous delay issues. In traditional database systems, write operations typically require synchronous writes, where the operation waits for the data to be written to disk before returning a success response. This synchronous write approach introduces write latency, reducing system write performance and throughput. However, a WAL approach named PALF (Paxos-backed Append-only Log File system) [11] is employed to handle write operations in OceanBase. When data are written to OceanBase, the write operation is first recorded in the WAL, which immediately returns a success response. This asynchronous write eliminates the need to wait for disk confirmation. OceanBase performs background log flushing to persist the data onto disk. This approach enables concurrent writes to multiple replicas without synchronous delay issues. Multiple write operations can be processed concurrently, writing to different replicas without blocking one another.

3) Multi-Paxos-based replication: OceanBase relies on the protocol of Multi-Paxos [13], an extension of the Paxos algorithm that uses multiple phases and rounds. It achieves efficient consensus by introducing one leader and multiple acceptors. In the OceanBase implementation, the Multi-Paxos protocol ensures the consistency and reliability of distributed transactions. It ensures data consistency among nodes, and provides high availability and fault tolerance through mechanisms such as leader failover. Multi-Paxos optimizes the Basic-Paxos by electing a leader within the Paxos cluster. During the leader's term, all proposals are initiated by the leader. This strengthens the assumption of the protocol, that no other server proposes during the leader's term. As a result, for subsequent proposals, the generation of log IDs and the *Prepare* phase can be simplified. The unique leader generates log IDs and directly performs the *Accept* phase, achieving consensus once a majority confirms the proposal (each redo log corresponds to a proposal).

C. Unit Writing and Reading

In the cloud synchronization scenario, OceanBase's multi-unit synchronization links and the ability of OceanBase Migration Service (OMS) to synchronize data in seconds ensure low-latency data synchronization across multiple data centers. For cloud synchronization, the nature of this application involves data synchronization across multiple endpoints. By using a partition key such as the available ID, all operations are completed within a single unit, improving R/W performance.

1) Routing: Business requests are linked by routing data. The traffic routing layer consists of two types: ingress traffic routing and cross-zone service invocation routing.

Ingress traffic routing: The challenge of ingress traffic routing lies in accurately determining the unit to which a business request belongs and routing it accordingly. To address this, it is crucial to have a solution that enables each business request to carry specific identifiers. By extracting, recognizing, and calculating these identifiers, the corresponding unit for the request can be determined.

To achieve this, a collaboration between the business application and the ingress traffic routing system is necessary. Identifiers are set in the business request cookies as part of the solution. When a user initiates a business transaction from a client (e.g., PC, mobile app, or other platforms), typically the first request made is a login request, which is challenging to associate with a specific unit. The routing policy will detect the location information and route the request to the geographically closest unit.

After the user logs in, the responsibility lies with the authentication center (login system) to write the user ID into the response cookie, adhering to a predefined format. From that point on, the client is required to include this cookie in every subsequent request within the same transaction.

Access to the routing gateway then identifies the cookie value in subsequent requests. Using the unitization service routing rules, it parses and calculates the cookie value to determine the appropriate unit for the transaction. Subsequently, it routes the request to the correct unit accordingly.

Cross-zone service invocation routing: Routing across zones is automatically handled by the RPC (remote procedure call) service based on the invocation parameters. Cross-zone service routing can occur in two situations: The main application is processed in GZone but requires the use of services in RZone (shard data). If the involved RZone is in the same city as GZone, it will result in intra-city cross-zone service invocation. If the involved RZone is in a different city from GZone, it will result in inter-city cross-zone service invocation.

Cross-zone service invocation is transparent to the application and is automatically handled by the RPC service framework of the unitization system. However, much like ingress traffic routing, the request must include data shard identification information in the service invocation parameters, which can then be parsed within the units. This enables the service framework to accurately parse and calculate the target unit for the service invocation.

2) OceanBase Migration Service: OceanBase Migration Service (OMS) [14] is a comprehensive tool designed exclusively for the OceanBase database, enabling efficient data transmission and synchronization. With OMS, users can seamlessly synchronize replicated data across various database instances, ensuring real-time migration and synchronization with minimal risk and cost.

In the three-site, five-IDC deployment of OceanBase, the implementation of the Paxos protocol and multi-replica deployment enables robust cross-region disaster recovery. In the event of a regional failure in the business center, the disaster recovery center seamlessly takes over by switching the domain name. OMS facilitates the creation of a disaster recovery center through its sub-second switch and real-time verification functionalities. Moreover, OMS ensures high availability of components by deploying multiple resource nodes. In the event of a fault in any OMS unit node, the synchronization link automatically connects to other available resource nodes, ensuring uninterrupted transmission.

3) Data loss issue: In scenarios where a user switches from one unit to another, data loss may occur if the previously written data have not yet been fully synchronized due to network delays. This means that when the user transitions to another unit, the newly written data may not be immediately visible or accessible in the new unit, resulting in data loss.

As shown in Figure 6, a potential issue arises when a user executes operations ❶ and ❷, but experiences a unit switch between the two operations. Assuming that the data delay is greater than the user operation time, this can lead to permanent data differences and data errors in OceanBase instances.

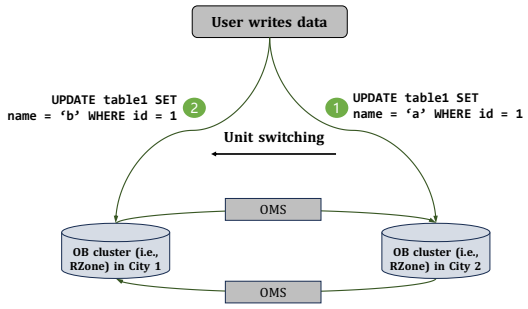

Fig. 6. Data loss problem.

To provide a clearer explanation, let us consider an example. For the OceanBase cluster (i.e., RZone) in City 1, if the data delay is significant, operation ❷ may be executed first, followed by the execution of operation ❶ through synchronization. Consequently, the final data in this scenario would be *id = 1* and *name = 'a'*. Conversely, for the OceanBase cluster (i.e., RZone) in City 2, operation ❶ could be executed initially, with operation ❷ occurring later. In this case, the final data would be *id = 1* and *name = 'b'*. These discrepancies in the final data highlight the potential data errors that can occur due to delays and unit switches.

To address this issue, we have implemented a "write ban time" on the business side. Additionally, OceanBase leverages the OMS to synchronize data across IDCs. It is crucial to prevent data overwriting during OMS data synchronization. OMS supports data re-chasing before start points and after end points, associated with timestamp. To ensure data integrity, the business side locks write operations between the start and end timestamps of the OMS process.

D. Centralized Writing and Unit Reading

The business characteristics of the review service indicate that the write TPS will not be excessively high. Although centralized writing meets the current requirements for system development, multi-point writing offers better disaster recovery capabilities despite introducing challenges like data conflicts and increased complexity. As a result, we decided to adopt the approach of centralized writing and unit reading.

The architecture of centralized writing and unit reading is shown in Figure 7. OceanBase adopts a centralized writing architecture, where all write operations are concentrated and processed at the central node. This simplifies the architecture and reduces data overlap issues. After centralized writing, data can be accessed through unit reading, making the architecture simpler without worrying about data overlap.

Centralized writing to the primary node should be synchronized with backup nodes. OceanBase achieves primary-backup data synchronization across three cities. This means that real-time data synchronization can be performed between databases in three cities, ensuring data consistency and reliability. With its native synchronization capabilities between clusters, OceanBase achieves data synchronization with sub-second latency, ensuring data immediacy and accuracy.

As shown in Figure 7, the City 2 central primary cluster is readable and writable, achieving strong consistency for reads. City 1 and City 3 are backup clusters, synchronized through OceanBase's native cluster replication capability with sub-second latency, providing high-throughput, low-latency read operations for weekly-consistent heterogeneous storage. Data consistency is ensured as follows:

Real-time synchronization: OceanBase achieves data consistency with the native synchronization capability between primary and backup clusters. **Offline analysis**: OceanBase utilizes a data analysis and monitoring platform to ensure data consistency in offline analysis scenarios.

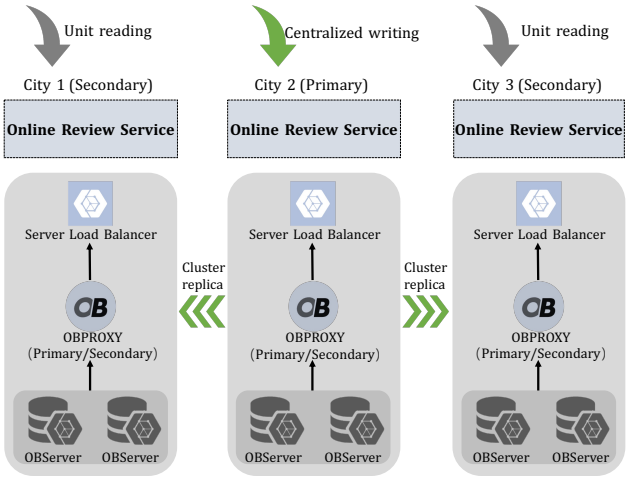

Fig. 7. Centralized Writing and Unit Reading for Online Review Service.

E. Unitization for Distributed Transactions

Distributed Transaction-eXtended (DTX) is a financial-grade middleware for distributed transactions that ensures the eventual consistency of business activities in large-scale distributed environments. It is widely used within Ant Group for core financial operations such as transactions, transfers, and red packets, serving billions of users' financial transactions. DTX can be seamlessly integrated with service frameworks (such as SOFABoot, Spring Cloud, Dubbo), data sources (such as data access proxy ODP, RDS, MySQL, OceanBase), and message queues, among other technology middleware products of Ant Group.

Two-phase commit (2PC) + GTS is a linearly consistent transaction process for distributed transactions, which is described in Algorithm 1.

Algorithm 1 Distributed transaction execution.

```
// Notify transaction decisions to participants P.
procedure LEADER(P)
    // Retrieve the maximal timestamp
    t ← max{gts, max_readable_ts, log_ts}
    // Send ready request to participants
    SENDREQUEST(P, t)
    // Wait for pre-commit from participants
    C ← WAITPRECOMMIT(P)
    // Make decision to commit or abort
    d ← COMMITORABORT(C)
    // Send decision to participants
    NOTIFYDECISIONS(P, d)
end procedure
// Execute the txn T on participant with a leader L.
procedure PARTICIPANT(L, T)
    // Receive timestamps from the leader
    t ← RECEIVETS(L)
    // Execute the transaction and write to logs
    r ← EXECUTETX(t, T)
    WRITELOGS(r)
    // Pre-commit phase
    max_readable_ts ← NOW
    // send pre-commit to the leader
    SENDPRECOMMIT(L, max_readable_ts)
    d ← RECEIVEDECISIONS(L)
    // commit the transaction?
    if d = commit then
        COMMIT(L, T)
    else
        ABORT(T)
    end if
    CLEANUP(T)
end procedure
```

The transaction leader generates a prepare version, which consists of the maximum value among the global timestamp (gts) at the tenant level (ensured by Paxos for stability), the maximum readable timestamp ($max\_readable\_ts$), and the log timestamp (log_ts). The transaction leader sends a prepare request to the participant nodes along with the prepare version. Upon receiving the prepare request, the participant nodes execute the transaction operations and save the results to their local transaction logs. After executing the transaction operations, the participant nodes advance their local maximum readable timestamp. The participant nodes send pre-commit messages to the transaction manager, indicating completion of the prepare phase. The pre-commit message includes the participant nodes' maximum readable timestamp. Upon receiving pre-commit messages from all participant nodes, the transaction manager sends instructions to all participant nodes. The participant nodes perform commit or abort operations that depend on the instructions from the transaction manager. In the case of a commit operation, the participant nodes release row locks and respond to the transaction manager with a success message after a majority of the log replications are successful. In the case of an abort operation, the participant nodes respond directly to the transaction manager with an abort message. Finally, the participant nodes release the transaction context and clean up related resources.

OceanBase adopts an early lock release (ELR) mechanism to improve transaction processing throughput and reduce transaction processing latency. The time for which a transaction holds a lock includes four aspects: 1) data writing, 2) log serialization, 3) network communication for synchronous backup, and 4) log flushing time. After optimization: 1) Once log serialization is completed and submitted to the Buffer Manager, the trigger for releasing row locks begins without waiting for log majority flush, thereby reducing the time for which a transaction holds a lock. 2) After a transaction releases the lock, subsequent transactions are allowed to operate on the same row, achieving concurrent updates by multiple transactions and improving throughput.

Under the unitization architecture, distributed transactions can be initiated in both the GZones and the RZones. Transactions initiated in the GZone are written to a single table in a DTX server. However, for transactions initiated in the RZone, the DTX server routes the transaction logs to the appropriate target databases and tables based on their shard IDs. In the case of transaction recovery, the recovery tasks are exclusively executed in the GZone. These tasks monitor both the single database table in the GZone and the sharded databases and tables in the RZones. By analyzing the transaction log for sharding information, they determine the target zone to which the transaction should be routed. This robust approach ensures efficient and reliable transaction handling, enabling seamless recovery even in complex distributed environments.

V. EXPERIMENTS

In this section, we evaluate the unitized OceanBase system based on the AMap benchmark.

A. Experimental Setup

We have deployed OceanBase 4.1.0 in three clusters, with each cluster containing an Alibaba Cloud ECS instance with 500GB disk space. We have configured the clusters in varying scales, including: 1) 32-core CPU and 128GB memory; 2) 64-core CPU and 256GB memory. We conducted a comprehensive comparison between OceanBase and two commercial distributed database systems on the cloud, namely CloudDB-A and CloudDB-B. To ensure a fair evaluation, each of these systems was deployed in the same region, with an identical scale of clusters.

In the case of CloudDB-A, we specifically selected the version that offers high availability and an x86 architecture. The specifications of the CPU, memory, and disk were kept consistent with those of OceanBase. Additionally, for the storage type, we opted for ESSD PL1 cloud disk storage. For CloudDB-B, we enabled it to create replicas for readers for multi-region deployment, achieving similar capabilities of availability and performance compared to OceanBase.

To comprehensively assess the performance of OceanBase, we have systematically selected and carefully trimmed Sysbench [15], which simulates various business scenarios from AMap. In our evaluation, we have created a total of 30 tables, with each table containing 500K records. This benchmark consists of five sub-benchmarks, each catering to different aspects of the system's functionality:

1) RO (Read Only): We simulate user interactions with published posts and comments. This sub-benchmark consists of ten "Select" queries, which are designed to retrieve contents based on a given ID, and four additional queries, which contain aggregated functions. All these read-only queries are formulated as distributed transactions.
2) RW (Read Write): To simulate realistic cases in the AMap review system, we introduce four write queries to the RO sub-benchmark. These queries formalized as distributed transactions encompass actions such as deletion, insertion, and updates for posts and comments.
3) WO (Write Only): We simulate extreme scenarios for cloud synchronization, where multiple end devices simultaneously modify the same information. WO sub-benchmark focuses solely on write queries, including operations such as delete, insert, and update on a given ID.
4) Insert: This sub-benchmark measures performance in inserting new tuples into the tables.
5) Update: The Update sub-benchmark evaluates the efficiency of update operations based on a given ID.

To ensure accurate performance measurements, we have executed each query with five warmup runs. The reported performance values represent the execution time at the 95th percentile.

B. Performance Comparison

In this experiment, we demonstrated the capabilities of OceanBase by comparing it with other two baselines. To evaluate their performance across various stress levels, we initialized a "user node" for each sub-benchmark and continually sent requests to the databases. We gradually escalated the request frequency to 100 thousand transactions per second (TPS). To assess the performance, we analyze two key metrics: the throughput of successful responses and the average response time (RT) for each request. The throughput represents the number of successful responses generated per unit of time, whereas the average response time is calculated by measuring the duration from the first request sent to the last response received, divided by the total number of requests.

To assess system scalability, we conducted experiments by gradually increasing the number of threads in the "user node". Each thread consistently sent requests at the same frequency, resulting in a higher rate of request transmission as the number of threads increased. This evaluation enabled us to examine system performance under varying levels of thread concurrency. Additionally, we evaluated the system's performance under different scales. We assessed how the system handles workloads in scenarios with varying levels of available resources and distributed processing.

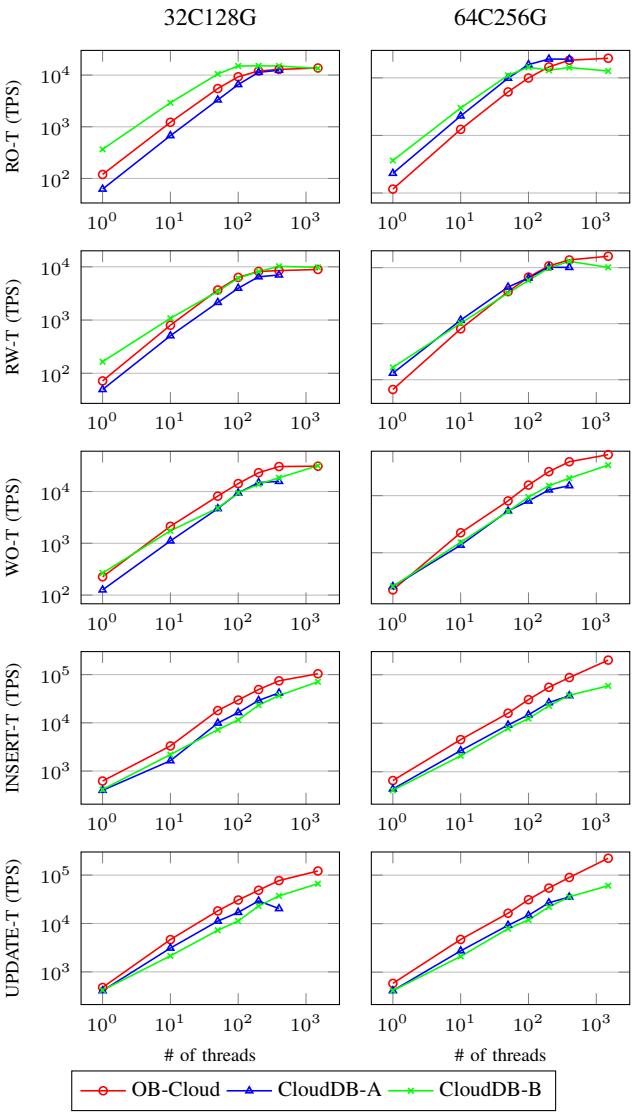

Fig. 8. Throughput of different distributed database systems with varying request frequency and ECS instance modes.

Figure 8 compares the performance (TPS) of various distributed database systems across all sub-benchmarks, using different cluster scales ranging from 32 cores with 128GB memory to 64 cores with 256GB memory. The user node spawns threads (ranging from 4 to 1500) that send benchmark requests continuously. When the previous request is answered, the thread immediately sends the next request. Missing points in the figure indicate that the high pressure of requests has overwhelmed the database instances.

In read-intensive sub-benchmarks such as RO, OB-cloud outperformed the other systems when subjected to high pressure of requests and large-scale CPU settings. When the number of cores is increased to 64, OB-cloud and Cloud-B could handle requests with 1500 threads. OB-cloud achieves a higher

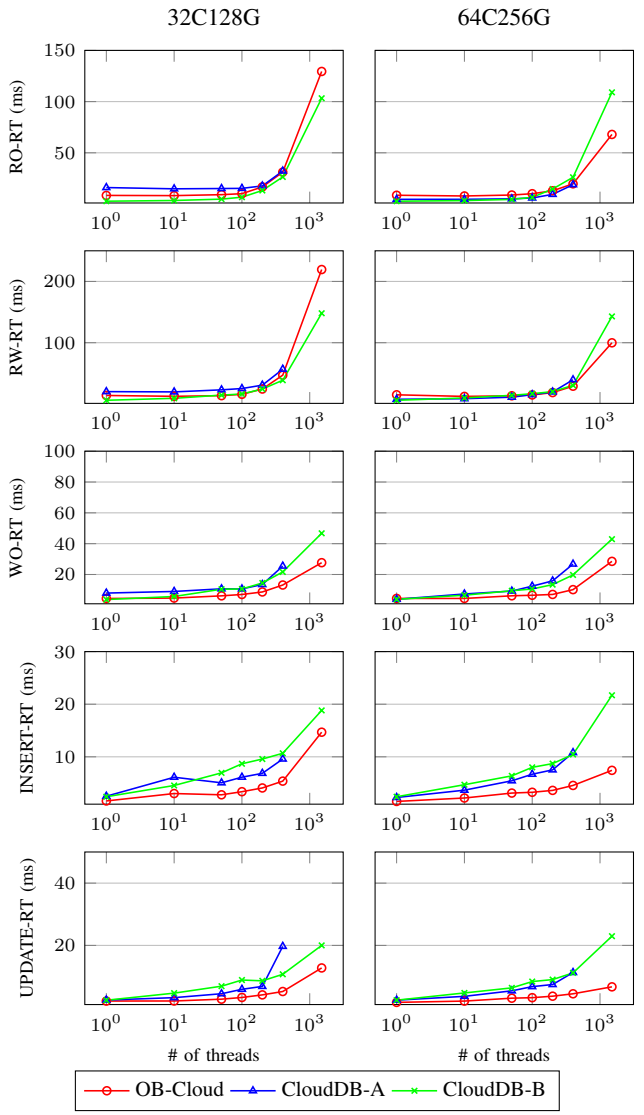

Fig. 9. Response time of different distributed database systems when varying the frequency of requests and ECS instance modes.

throughput because individual units assisted in processing requests, thereby avoiding overwhelming the primary node.

In write-intensive or high-contention sub-benchmarks such as INSERT and UPDATE, OB-cloud consistently achieved the highest throughput across all experimental settings for two reasons. Write operations are processed by individual units in OceanBase. Firstly, OceanBase uses an early lock release strategy that is highly optimized for distributed transactions. This enables participant nodes to commit before the logs are serialized on the disk. Moreover, OceanBase employs an offline synchronization mechanism called OMS to ensure data consistency among units. In contrast, the other systems require data modifications on the primary node to be synchronized with backup nodes to complete the entire procedure. We have also included the average response time of requests in Figure 9, which exhibited a similar trend.

C. Storage Cost Comparison

We evaluated the efficiency of OceanBase's compression scheme. Given that CloudDB-A and CloudDB-B prioritize performance and availability over storage optimization, we conduct our comparison on OceanBase with two commercial database systems deployed locally, namely DB-A and DB-B. These baselines apply dedicated compression schemas to enhance storage efficiency and reduce costs. In Figure 10, we gradually increase the size of the original table by inserting rows, ranging from 10 million to 100 million. With an average row size of 1K, this leads to the creation of table sizes ranging from 10GB to 100GB. Clearly, OceanBase outperforms DB-A by a 8.2× and DB-B by a 2.8× in space-saving capabilities.

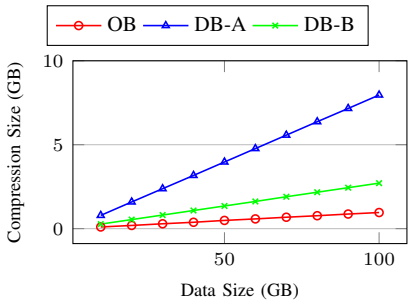

Fig. 10. Compression sizes of different distributed database systems when varying size of dataset.

D. Availability Comparison

OceanBase ensures high availability by distributing units across multiple regions. To assess system availability, it is important to simulate scenarios where clusters encounter disasters or failures. For consistency, we execute the RW sub-benchmarks using identical cluster configurations, table schemas, and sizes. Failures can occur in various components, such as the OB server process, node machine, or network. Table I presents eight common failure cases and provides instructions on how to simulate them during experiments. These simulations enable us to evaluate resilience and performance in the face of different failure scenarios.

TABLE I
FAILURE CASES FOR AVAILABILITY EXPERIMENTS.

Type	Descriptions	Commands
F1	Stop the process of observer	kill -9 observer
F2	Suspend the process of observer	kill -19 observer
F3	Terminate the process of observer	kill -15 observer
F4	Reboot any node of cluster	reboot
F5	Machine abnormal power-off cold reboot	echo b>/proc/sysrq-trigger
F6	Network partition failure	set firewall on iptables
F7	Hang the disk of commit logs	kill the OB writer
F8	Change the locality of replica	decrease the number of replica

In each failure case, when requests are directed towards a region that is no longer accessible due to OB cluster failure, an initial lack of response is observed. However, the unitization

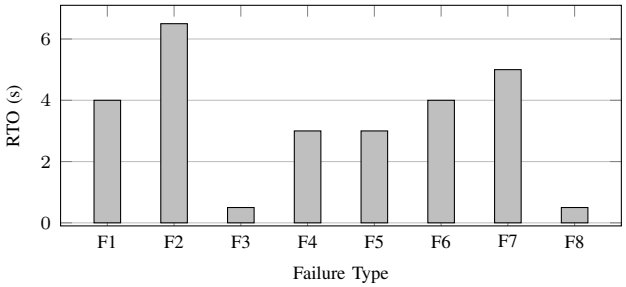

Fig. 11. RTO of OceanBase for different failure cases.

coordinator promptly detects this issue and redirects those requests to active units. To quantitatively evaluate system recovery performance, we measure the RTO. This RTO is calculated as the duration from when the request encounters a "no response" error until it is successfully answered by other units. Figure 11 provides a detailed analysis of the RTO for each failure case. Notably, the RTO for OceanBase's unitization deployment remained consistently below 6 seconds. F2 shows the largest RTO due to occasional failures in receiving the node reset message by the OBserver. However, this issue can be effectively mitigated through parameter tuning. This demonstrates the system's ability to swiftly recover from failures and resume normal operations, ensuring minimal disruption to request processing.

VI. Lessons Learned

In this section, we detail what we have learned in building OceanBase utilization for AMap.

A. Performance Optimization

In order to solve the performance problems caused by too long links in distributed transactions, optimization can be done from the following perspectives. First, we use tracking technology to achieve full-link monitoring, including comprehensive analysis of business processes, application services, and database SQL statements. In short, business operations, application service calls and database queries are linked through the tracking mechanism to obtain the complete business execution path. Secondly, the tracking information from the application layer is passed to the database layer to ensure that each SQL statement and each row of data is associated with the corresponding business link. In this way, each SQL statement in each business link can be identified, and the execution time of these SQLs can be captured in the database audit log.

By collecting this data and analyzing it from multiple dimensions, we can identify, for example, the slowest-executing SQL statements and perform performance optimization on them; the most frequently called services, for which we might consider introducing a caching mechanism to reduce database load; and SQL statements executed across cities, regions, or data centers, which can be categorized and optimized to reduce potential delays.

B. Business Continuity

Ensuring business continuity of AMap by using OceanBase unitization involves resilience planning and implementing technological solutions to address disruptions. High availability and disaster recovery are essential for AMap's database-dependent geospatial data services. OceanBase's load balancing distributes incoming traffic across nodes, eliminating single points of failure and providing consistent service. Dynamic capacity scheduling allows the system to adjust resources according to user demand, critical during peak times or traffic spikes. Strict change management, involving thorough testing of system updates or changes, helps maintain service stability and prevent outages. Disaster recovery in OceanBase may include geographically distributed clusters for failover during regional disruptions, assuring uninterrupted service. Daily operations benefit from comprehensive monitoring and automated self-healing systems to detect and correct failures, thus minimizing downtime. A well-rehearsed disaster recovery plan, detailing failover procedures and roles, is crucial for rapid recovery and continuous operations, even during extensive system failures, ensuring AMap's service reliability. OceanBase unitization and its availability features provide a robust foundation for these continuity strategies.

C. System Operation and Maintenance

Challenges faced by system operation and maintenance mainly include equipment stability and traffic fluctuation management. AMap systems that adopt a distributed structure may encounter stability problems with blade servers, which may cause business interruption. In order to alleviate these problems, the system is designed to have stand-alone self-healing capabilities so that it can quickly resume normal operation in the event of a failure. In addition, in the face of possible traffic surges, we ensured that link capacity assessment and comprehensive stress testing were carried out before the system went online.

In addition, in order to ensure the smooth operation of the system, we have also implemented daily operation and maintenance inspections, such as early warning mechanisms for message backlog and database suspension. The system's end-of-day batch processing jobs, including distributed transaction scheduling and offline batch processing, are performed according to predetermined scheduling links, ensuring overall operational efficiency. In order to ensure the accuracy and consistency of the data, we will automatically check the upstream and downstream data (bypass check) to ensure the accuracy of the transaction data. These preparations and measures together form a key part of system operation and maintenance to deal with various difficulties that may be faced after going online.

VII. Related Work

A. Disaster Tolerance

1) Disaster recovery: Disaster recovery methods [16]–[19] have been extensively utilized to ensure system availability and mitigate the impact of disasters. Cloud Standby [20] establishes and maintains a secondary IDC, serving as a means for data recovery following a disaster. Dual-active disaster recovery [21], [22] enables both primary and standby IDCs to provide services simultaneously, with data synchronization

occurring unidirectionally from the primary to the secondary IDC. Numerous cloud services [23]–[25] have adopted a Multi-site High Availability architecture [26], [27], which ensures continuous operation and service availability across multiple geographical sites or IDCs. This architecture is specifically designed to minimize downtime and provide redundancy in the face of failures or disruptions at any single site. For elastic computing capabilities, Alibaba Cloud and Ant Group have introduced the concept of unitization [28], [29], in which a fully functional operating system and resources are deployed across regions to facilitate cross-region disaster recovery.

2) Log-based recovery: Disaster tolerance depends on the recovery algorithm implemented on each node [30]. Various cloud databases [31]–[34] customize their failure recovery methods to suit different architectures. PolarDB Serverless [32] utilizes an ARIES-style recovery algorithm [35]. This algorithm distinguishes between different types of nodes, such as primary nodes and read replicas, and employs specific algorithms to ensure quick recovery. Amazon Aurora [31] uses the storage layer to effectively manage a sequential log, thereby improving performance in write operations. On the other hand, Microsoft Socrates [33] introduces an independent XLOG service that separates logs from computation and storage. This dedicated tier ensures optimal low latency and high throughput for storing transaction logs. SolarDB [36] separates storage and computation, persisting write-ahead logging into durable storage before transaction commitment.

3) Partition and replication architecture: To eliminate the need to wait for the recovery of a single node, many systems adopt explicit partitioning and replication techniques across multiple nodes. Many production systems [37]–[39] use primary-secondary replication for backup support. The backup replica also enables users to read and write concurrently, providing high throughput [40]–[45]. VoltDB [46] employs a multi-node architecture where tables are horizontally partitioned into "shards" that are replicated across different nodes. Besides, many distributed systems have developed consensus algorithms [12], [47] for data consistency. Google Spanner [48] is designed to manage replicated data across data centers by implementing a single Paxos state machine to ensure consistent updates across replicas. Similarly, Calvin [49] also supports a Paxos-based synchronous replication mechanism but replicates transaction inputs rather than their effects. The recovery process begins from a checkpoint and leverages the deterministic nature of the database to recover by replaying transaction inputs. PaxosStore [50] built a high-availability storage system to support the comprehensive business of WeChat. It developed a layered design for the Paxos-based storage protocol on Bitcask and LSM-tree.

B. Distributed Transactions

Distributed transactions are executed across multiple nodes by data or service partitioning techniques. Calvin [49] splits transactions into jobs and schedules them among different nodes. It establishes a deterministic job execution order to minimize the need for costly coordination. RedT [51] is an RDMA-enhanced distributed transaction processing protocol with availability guarantees. Compared with 2PC-Paxos, it decreases the number of inter-DC round-trips from a maximum of six to two. Lindorm [6] combines a cluster of compute nodes and shared storage to support low latency queries for time series data.

Amazon Dynamo [52] is a highly decentralized key-value (KV) storage system that performs transactions on partitioned data. Cassandra [53] is a distributed storage system designed for managing large amounts of structured data across multiple commodity servers. MongoDB [5] supports multi-document transactions, enabling transactions across operations, databases, documents, and shards. OceanBase utilizes in-memory and disk data structures to mitigate document-related overhead.

Disaggregation is widely adopted in distributed database systems. PolarDB Serverless [32] employs a decoupled architecture, separating CPU resources on compute nodes from remote memory and storage pools. DAGOR [54] adopts a different approach by dividing services into microservices. It carefully controls the granularity of each microservice, enabling real-time load monitoring and collaborative load shedding among relevant services in the presence of overload conditions. Citus [55] is an open-source distributed database engine for PostgreSQL [56]. It extends PostgreSQL to operate across clusters, providing distributed queries, and transactions to handle data-intensive applications.

VIII. Conclusions and Future Work

This paper demonstrates the unitization architecture of OceanBase, which is a reliable and high-performance distributed relational database, for AMap platforms. Adoption of OceanBase unitization architecture in AMap avoids cross-data center access, and ensures that all operations can be completed in one data center. We have showcased that our unitization architecture ensures high availability by distributing data and services across different units. Furthermore, it offers excellent performance through unitized or centralized read and write operations. The OceanBase-based AMap system achieves a cloud-native and industry-agnostic architecture that caters to various user scenarios at a massive scale.

In future, AMap will migrate the business of structured data scenarios to OceanBase, such as AMap Footprint (structured storage), with the goal of reducing costs by more than 50%. After migration, AMap can enjoy the benefits of OceanBase's linear optimization. The number of nodes that can compress data is minimized. The system can automatically expand elastically when there is a big promotion or large volume. Moreover, because OceanBase has distributed capabilities, AMap does not need to pay attention to the load. To solve the load-balancing problem, automatic and transparent expansion is fully realized. Currently, AMap has been exploring unstructured data scenarios with OceanBase, with the goal of migrating the taxi-hailing feature platform (typical KV scenarios) to OceanBase, with an expected cost reduction target of over 50%.

REFERENCES

[1] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. C. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford, "Spanner: Google's globally-distributed database," in *10th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2012, Hollywood, CA, USA, October 8-10, 2012*, C. Thekkath and A. Vahdat, Eds. USENIX Association, 2012, pp. 251–264.

[2] Z. Yang, C. Yang, F. Han, M. Zhuang, B. Yang, Z. Yang, X. Cheng, Y. Zhao, W. Shi, H. Xi, H. Yu, B. Liu, Y. Pan, B. Yin, J. Chen, and Q. Xu, "Oceanbase: A 707 million tpmc distributed relational database system," *Proc. VLDB Endow.*, vol. 15, no. 12, pp. 3385–3397, 2022.

[3] Z. Yang, Q. Xu, S. Gao, C. Yang, G. Wang, Y. Zhao, F. Kong, H. Liu, W. Wang, and J. Xiao, "Oceanbase paetica: A hybrid shared-nothing/shared-everything database for supporting single machine and distributed cluster," *Proc. VLDB Endow.*, vol. 16, no. 12, pp. 3728–3740, 2023.

[4] "Amap," https://www.amap.com/, 2024.

[5] "MongoDB," https://www.mongodb.com/, 2024.

[6] C. Shen, Q. Ouyang, F. Li, Z. Liu, L. Zhu, Y. Zou, Q. Su, T. Yu, Y. Yi, J. Hu *et al.*, "Lindorm tsdb: A cloud-native time-series database for large-scale monitoring systems," *Proceedings of the VLDB Endowment*, vol. 16, no. 12, pp. 3715–3727, 2023.

[7] "Elasticsearch," https://www.elastic.co/, 2024.

[8] "MulanPubL-2.0," https://license.coscl.org.cn/MulanPubL-2.0/index.html, 2024.

[9] "OceanBase," https://gitee.com/oceanbase, 2024.

[10] "OceanBase," https://github.com/oceanbase, 2024.

[11] F. Han, H. Liu, B. Chen, D. Jia, J. Zhou, X. Teng, C. Yang, H. Xi, W. Tian, S. Tao, S. Wang, Q. Xu, and Z. Yang, "PALF: replicated write-ahead logging for distributed databases," *Proc. VLDB Endow.*, vol. 17, no. 12, pp. 3745–3758, 2024.

[12] L. Lamport, "Paxos made simple," *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)*, pp. 51–58, 2001.

[13] T. D. Chandra, R. Griesemer, and J. Redstone, "Paxos made live: an engineering perspective," in *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing, PODC 2007, Portland, Oregon, USA, August 12-15, 2007*, I. Gupta and R. Wattenhofer, Eds. ACM, 2007, pp. 398–407.

[14] "OceanBase Migration Service (OMS)," https://en.oceanbase.com/docs/oms-en, 2024.

[15] A. Kopytov, "Sysbench: Scriptable database and system performance benchmark," 2018.

[16] M. Choy, H. V. Leong, and M. H. Wong, "Disaster recovery techniques for database systems," *Communications of the ACM*, vol. 43, no. 11es, pp. 6–es, 2000.

[17] E. Lau, "Fast crash recovery for a distributed column-store database management system," 2005.

[18] A. Gupta and J. Shute, "High-availability at massive scale: Building google's data infrastructure for ads," in *Real-Time Business Intelligence and Analytics: International Workshops, BIRTE 2015, Kohala Coast, HI, USA, August 31, 2015, BIRTE 2016, New Delhi, India, September 5, 2016, BIRTE 2017, Munich, Germany, August 28, 2017, Revised Selected Papers 9*. Springer, 2019, pp. 63–81.

[19] P. Bhardwaj, K. Lohani, R. Tomar, and R. Srivastava, "Comparative analysis of traditional and cloud-based disaster recovery methods," in *Intelligent Computing Techniques for Smart Energy Systems: Proceedings of ICTSES 2021*. Springer, 2022, pp. 105–117.

[20] A. Lenk and S. Tai, "Cloud standby: disaster recovery of distributed systems in the cloud," in *Service-Oriented and Cloud Computing: Third European Conference, ESOCC 2014, Manchester, UK, September 2-4, 2014. Proceedings 3*. Springer, 2014, pp. 32–46.

[21] N. Shuping and W. Feng, "The network architecture design of distributed dual live data center," in *2019 IEEE International Conference on Power, Intelligent Computing and Systems (ICPICS)*. IEEE, 2019, pp. 638–642.

[22] M. Yong, Y. Tang, S. Gong, and C. Xu, "Design of dual-active and multi-active framework of information system in cloud environment," in *2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, vol. 9. IEEE, 2020, pp. 766–769.

[23] "Huawei Cloud," https://cloud.huawei.com/, 2024.

[24] "Alibaba Cloud," https://www.alibabacloud.com/, 2024.

[25] "Amazon Web Services (AWS)," https://aws.amazon.com/, 2024.

[26] O. Torbjornsen, "Multi-site declustering strategies for very high database service availability," *Department of Computer and Information Science. Trondheim, NTNU*, 1995.

[27] M. Q. Hiep, H. Yang, and Y. Kim, "Dynamic policy management system for high availability in a multi-site cloud," in *2020 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 2020, pp. 359–362.

[28] "Best Practices for Cross-Zone Disaster Recovery and Multi-Site High Availability on the Cloud," https://www.alibabacloud.com/blog/best-practices-for-cross-zone-disaster-recovery-and-multi-site-high-availability-on-the-cloud_599397, 2024.

[29] "Ant Group," https://antgroup.com/, 2024.

[30] J. E. B. Moss, "Log-based recovery for nested transactions," in *VLDB'87, Proceedings of 13th International Conference on Very Large Data Bases, September 1-4, 1987, Brighton, England*, P. M. Stocker, W. Kent, and P. Hammersley, Eds. Morgan Kaufmann, 1987, pp. 427–432.

[31] A. Verbitski, A. Gupta, D. Saha, M. Brahmadesam, K. Gupta, R. Mittal, S. Krishnamurthy, S. Maurice, T. Kharatishvili, and X. Bao, "Amazon aurora: Design considerations for high throughput cloud-native relational databases," in *Proceedings of the 2017 ACM International Conference on Management of Data*, 2017, pp. 1041–1052.

[32] W. Cao, Y. Zhang, X. Yang, F. Li, S. Wang, Q. Hu, X. Cheng, Z. Chen, Z. Liu, J. Fang *et al.*, "Polardb serverless: A cloud native database for disaggregated data centers," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 2477–2489.

[33] P. Antonopoulos, A. Budovski, C. Diaconu, A. Hernandez Saenz, J. Hu, H. Kodavalla, D. Kossmann, S. Lingam, U. F. Minhas, N. Prakash *et al.*, "Socrates: The new SQL server in the cloud," in *Proceedings of the 2019 International Conference on Management of Data*, 2019, pp. 1743–1756.

[34] H. T. Vo, S. Wang, D. Agrawal, G. Chen, and B. C. Ooi, "LogBase: A Scalable Log-structured Database System in the Cloud," *Proc. VLDB Endow.*, vol. 5, no. 10, pp. 1004–1015, 2012.

[35] 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 Transactions on Database Systems (TODS)*, vol. 17, no. 1, pp. 94–162, 1992.

[36] T. Zhu, Z. Zhao, F. Li, W. Qian, A. Zhou, D. Xie, R. Stutsman, H. Li, and H. Hu, "Solardb: Toward a shared-everything database on distributed log-structured storage," *ACM Transactions on Storage (TOS)*, vol. 15, no. 2, pp. 1–26, 2019.

[37] "openGauss," https://opengauss.org/, 2024.

[38] R. Taft, I. Sharif, A. Matei, N. VanBenschoten, J. Lewis, T. Grieger, K. Niemi, A. Woods, A. Birzin, R. Poss *et al.*, "Cockroachdb: The resilient geo-distributed sql database," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 1493–1509.

[39] P. Chairunnanda, K. Daudjee, and M. T. Özsu, "Confluxdb: Multi-master replication for partitioned snapshot isolation databases," *Proceedings of the VLDB Endowment*, vol. 7, no. 11, pp. 947–958, 2014.

[40] "Apache CouchDB," https://couchdb.apache.org/, 2024.

[41] "ArangoDB," https://www.arangodb.com/, 2024.

[42] "ExtremeDB: Cluster Distributed Database System," https://www.mcobject.com/cluster/, 2024.

[43] "Galera Cluster for MySQL," https://galeracluster.com/, 2024.

[44] M. A. Georgiou, A. Paphitis, M. Sirivianos, and H. Herodotou, "Hihooi: A database replication middleware for scaling transactional databases consistently," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 2, pp. 691–707, 2020.

[45] Y. Lu, X. Yu, and S. Madden, "STAR: scaling transactions through asymmetric replication," *Proc. VLDB Endow.*, vol. 12, no. 11, pp. 1316–1329, 2019.

[46] M. Stonebraker and A. Weisberg, "The voltdb main memory dbms." *IEEE Data Eng. Bull.*, vol. 36, no. 2, pp. 21–27, 2013.

[47] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 USENIX annual technical conference (USENIX ATC 14)*, 2014, pp. 305–319.

[48] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild *et al.*, "Spanner: Google's globally distributed database," *ACM Transactions on Computer Systems (TOCS)*, vol. 31, no. 3, pp. 1–22, 2013.

[49] A. Thomson, T. Diamond, S.-C. Weng, K. Ren, P. Shao, and D. J. Abadi, "Calvin: fast distributed transactions for partitioned database systems," in *Proceedings of the 2012 ACM SIGMOD international conference on management of data*, 2012, pp. 1–12.

[50] J. Zheng, Q. Lin, J. Xu, C. Wei, C. Zeng, P. Yang, and Y. Zhang, "Paxosstore: high-availability storage made practical in wechat," *Proceedings of the VLDB Endowment*, vol. 10, no. 12, pp. 1730–1741, 2017.

[51] Q. Zhang, J. Li, H. Zhao, Q. Xu, W. Lu, J. Xiao, F. Han, C. Yang, and X. Du, "Efficient distributed transaction processing in heterogeneous networks," *Proceedings of the VLDB Endowment*, vol. 16, no. 6, pp. 1372–1385, 2023.

[52] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," *ACM SIGOPS operating systems review*, vol. 41, no. 6, pp. 205–220, 2007.

[53] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS operating systems review*, vol. 44, no. 2, pp. 35–40, 2010.

[54] H. Zhou, M. Chen, Q. Lin, Y. Wang, X. She, S. Liu, R. Gu, B. C. Ooi, and J. Yang, "Overload control for scaling wechat microservices," in *Proceedings of the ACM Symposium on Cloud Computing*, 2018, pp. 149–161.

[55] U. Cubukcu, O. Erdogan, S. Pathak, S. Sannakkayala, and M. Slot, "Citus: Distributed postgresql for data-intensive applications," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 2490–2502.

[56] "PostgreSQL," https://www.postgresql.org/, 2024.