OceanBase: A 707 Million tpmC Distributed Relational Database System

Zhenkun Yang, Chuanhui Yang, Fusheng Han, Mingqiang Zhuang, Bing Yang, Zhifeng Yang, Xiaojun Cheng, Yuzhong Zhao, Wenhui Shi, Huafeng Xi, Huang Yu, Bin Liu, Yi Pan, Boxue Yin, Junquan Chen, Quanqing Xu
OceanBase
OceanBaseLabs@list.alibaba-inc.com

ABSTRACT

We have designed and developed OceanBase, a distributed relational database system from the very basics for a decade. Being a scale-out multi-tenant system, OceanBase is cross-region fault tolerant, which is based on the shared-nothing architecture. Besides sharing many similar goals with alternative distributed DBMS, such as horizontal scalability, fault-tolerance, etc., our design has been driven by the demands of typical RDBMS compatibility as well as both on-premise and off-premise deployments. OceanBase has fulfilled its design goal. It implements the salient features of certain mainstream classical RDBMS, and most applications on them can run on OceanBase, with or without a few minor modifications. Tens of thousands of OceanBase servers have been deployed in Alipay.com as well as many other commercial organizations. It has also successfully passed the TPC-C benchmark test and seized the first place with more than 707 million tpmC. This paper presents the goals, design criteria, infrastructure, and key components of OceanBase including its engines for storage and transaction processing. Further, it details how OceanBase achieves the above leading TPC-C benchmark in a distributed cluster with more than 1,500 servers from 3 zones. It also describes lessons what we have learnt in building OceanBase for more than a decade.

PVLDB Reference Format:
Zhenkun Yang, Chuanhui Yang, Fusheng Han, Mingqiang Zhuang, Bing Yang, Zhifeng Yang, Xiaojun Cheng, Yuzhong Zhao, Wenhui Shi, Huafeng Xi, Huang Yu, Bin Liu, Yi Pan, Boxue Yin, Junquan Chen, Quanqing Xu. OceanBase: A 707 Million tpmC Distributed Relational Database System. PVLDB, 15(12): 3385 - 3397, 2022.
doi:10.14778/3554821.3554830

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/oceanbase/obdeploy.

1 INTRODUCTION

Strong transaction guarantee, relational model, and excellently expressible Structured Query Language (SQL) make Relational Database Management System (RDBMS) the crucial information infrastructure of the majority of business systems. For the last three decades, the development of Internet platforms has facilitated the flourishing global businesses, e.g., the likes of Alipay.com, Amazon.com, and Taobao.com, serve the general populace instead of a single organization. Classical centralized RDBMS are not capable of meeting the requirements of the scalability, cross-region fault tolerance, and cost-effectiveness of these businesses.

We launched the design and development of OceanBase [6, 7], a commodity hardware-based distributed relational database system from the very basics, in May 2010. OceanBase has been first used as the Favorite of Taobao.com [3] in 2011, a service similar to the Wish List of Amazon.com [11]. Thereafter, it was used by Alipay.com, in 2014, and by Zhejiang E-Commerce Bank in 2015, and many other commercial banks, insurance companies, and other organizations for communication and energy applications.

This paper first presents the detailed design goals and criteria, system architecture, SQL engine and multi-tenancy of OceanBase in §2. Second, it presents an LSM-tree-based [35] storage engine, and discusses the asymmetric read and write design, daily incremental major compaction, and replica type in §3. Third, in §4, it proposes the transaction processing engine including the timestamp service, transaction processing, isolation level, and replicated table in OceanBase. Fourth, in §5, we performed the TPC-C benchmark test of OceanBase in 2020. §6 presents lessons learnt in building OceanBase. §7 provides a brief review of the related work. Finally, we conclude our work in §8. We briefly list our contributions in the following items.

- We have built OceanBase, a distributed relational database from the very basics, since 2010. As a scale-out multi-tenant system, OceanBase is cross-region fault tolerant, and it supports the shared-nothing architecture. In case of the failure of a minority of the nodes, its *RPO* (Recovery Point Objective) turns zero, and its *RTO* (Recovery Time Objective) is less than 30 seconds.
- We present an LSM-tree-based storage engine, which achieves the performance close to that of the in-memory database after multiple optimizations. An asymmetric read and write data block storage system as well as a daily incremental major compaction have been designed and implemented.
- We propose a Paxos-based 2PC named *OceanBase 2PC* to improve the distributed transaction processing capability and reduce the transaction latency, which introduces the Paxos protocol to 2PC, thus making the distributed transactions have an automatic fault tolerance. Compared with the traditional 2PC, the state of the coordinator does not persist in OceanBase 2PC, thereby reducing the number of Paxos

Proceedings of the VLDB Endowment, Vol. 15, No. 12 ISSN 2150-8097.
doi:10.14778/3554821.3554830

synchronizations from three to two and further truncating the transaction latency to only one Paxos synchronization.

- We have performed the TPC-C benchmark test of OceanBase to reach 707 million tpmC in 2020, which is the best global record, hitherto.

OceanBase is an open source project under Mulan Public License 2.0 [5] and the source code is available on both gitee [6] and GitHub [7].

2 DESIGN OVERVIEW

We work on the design of OceanBase that supports the fast scale-out (scale-in) on the commodity hardware, to achieve high performance and low total cost of ownership (TCO), cross-region deployment, and fault tolerance. It is compatible with certain mainstream classical RDBMS. In this section, we introduce our design goals and criteria, and discuss the system infrastructure and deployment of OceanBase.

2.1 Goals

Goals of OceanBase include the following.

1) **Fast scale-out (scale-in) on commodity hardware to achieve high performance and low TCO.**
Similar to the high-end server and SAN storage, the classical centralized RDBMS are highly expensive and difficult to expand. Sharding and resharding can be extremely taxing on human resource and time [19]. OceanBase should be much more cost-effective than classical RDBMS and, e.g., be able to scale-out and scale-in quickly, before and after a business promotion event, respectively.
2) **Cross-region deployment and fault tolerance.**
High availability of the classical RDBMS system is solely based on the high availability of the hardware, e.g., the high-end server and SAN storage. Database master and backup mirroring cannot guarantee both the service availability and data integrity following the failure of the master database. OceanBase should be cross-region fault tolerant, and hence it guarantees the data integrity even in case of the failure of one region.
3) **Compatible with some mainstream classical RDBMS.**
Hundreds of thousands of legacy applications are running on various classical RDBMS. The cost, time, and risk of the migration of these legacy applications from the classical RDBMS to OceanBase should be minimized. This invokes the compatibility of OceanBase with these classical RDBMS. It is discussed in details in §2.2.

2.2 Criteria of Design

In the past several decades, classical RDBMS have been adopted by multiple independent software development vendors, solution providers, and used by numerous organizations. Various complex SQL statements and stored procedures, consisting of a few to tens of thousands of SQL statements, have been running on these RDBMS to support various types of businesses. Each new relational database has to encounter the following challenges:

- The cost, time, and risk of migrating the businesses from the old database to the new database.
- The cost and time of learning of the new database and migrating their solution from the old database to the new database by independent software vendors and solution providers, etc.
- The cost and time of learning the new database by the third-party database service providers or by users themselves.

Being a general-purpose relational database system, the design and implementation of OceanBase comply with the following criterion:

Criterion 2.1. *Native compatibility with certain mainstream classical RDBMS, and taking into account, the needs of the large, medium, and small organizations.*

- Native compatibility with certain mainstream classical RDBMS that implements the salient features of these classical RDBMS while being compatible with all of them, including the data types, secondary index, view, trigger, cursor, constrains, functions, and stored procedure. Applications on these RDBMS should be able to run on OceanBase with or without only a few minor modifications.
- Suitability for large, medium and small organizations such that a large online shopping and payment organization may need tens of thousands of high-profile database servers, whereas a small organization may need only a few low-profile database servers. Hence, one OceanBase cluster may consist of tens of thousands of high-profile servers to meet the requirements of a large organization, or it may also consist of a few low-profile servers to meet the cost and performance requirements of a small organization.

2.3 Infrastructure

OceanBase supports the shared-nothing architecture, and its overall architecture is shown in Figure 1. Multiple servers in a distributed cluster of OceanBase concurrently provide database services with high availability. In Figure 1, the application layer sends a request to the proxy layer (i.e., OBProxy), and after the routing of the proxy service, it is sent to a database node (OBServer) of the actual service data, and the execution result follows the reverse path to the application layer. Different components in the whole process achieve high availability in different ways.

Each OceanBase cluster consists of several zones, viz., 1, 3, or 5 zones. These zones can be restricted to one region or spread over multiple regions. In each zone, OceanBase can be deployed as shared-nothing. Transactions are replicated among the zones using Paxos [27]. OceanBase supports the cross-region disaster tolerance for multiple regions and zero data loss (*RPO*=0, *RTO*<=30 seconds) [10].

Database tables, especially the large ones, are partitioned explicitly by the user and these partitions are the basic units for the data distribution and load balance. For the convenience of discussion, non-partitioned table is considered as a partitioned table with one partition. For every partition, there is a replica in each zone, and these replicas form a Paxos group.

In each node, OceanBase is similar to a classical RDBMS. Subsequent to an OceanBase node receiving a SQL statement or a group of SQL statements (e.g., a stored procedure), it compiles the SQL statement(s) to produce a SQL execution plan. If it is a local plan, it

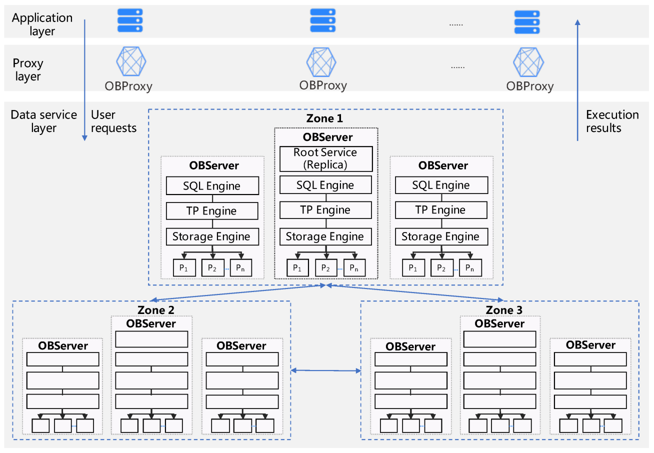


Figure 1: System Architecture of OceanBase.

is executed by this OceanBase node as in a classical RDBMS. Otherwise, this plan is executed using the so-called two-phase commit protocol and the OceanBase node acts as the coordinator. A transaction is committed only after its redo logs persist on the majority of all corresponding Paxos groups.

Among all Paxos groups of an OceanBase cluster, one Paxos group is in charge of the management of the cluster, e.g., load balance, adding (removing) nodes, failure detection of nodes, and fault tolerance of failure nodes.

2.4 SQL Engine

The SQL engine of OceanBase is the data computing hub of the entire database. When it receives a SQL request after a series of processes such as syntax analysis, semantic analysis, query rewriting, and query optimization, the executor is responsible for the execution. If the SQL statement involves a large amount of data, the query execution engine of OceanBase implements a series of techniques such as distributed query execution, data reshuffle, vertical (horizontal) parallel execution, dynamic join filter, dynamic partition pruning, and global queueing.

The generation of the execution plan (a.k.a hard parse in some database vendor's system) is a relatively complicated process over a long-time interval, specifically in the OLTP scenario. The optimizer needs to perform query transformation, physical optimization and finally the engine needs to do code generation, which usually dominates the whole execution time from end-to-end point of view. Since many systems also have plan cache alike components implemented, OceanBase uses a fast parser that is a super lightweight framework to do only lexical analysis, and then attempts to match an existing plan in the plan cache. The consideration behind is that it does not require a grammar/syntax checking as long as it can match a statement already in the plan cache. This approach is 10 times faster as compared to a normal parser.

Figure 2 describes the execution process of a SQL statement and lists the relationships between various modules. Subsequent to receiving the SQL request string sent by the user, the *Parser* will divide the string into individual words, and parse the entire request according to the pre-set grammatical rules. The *Resolver* will translate the tokens in the SQL request into corresponding

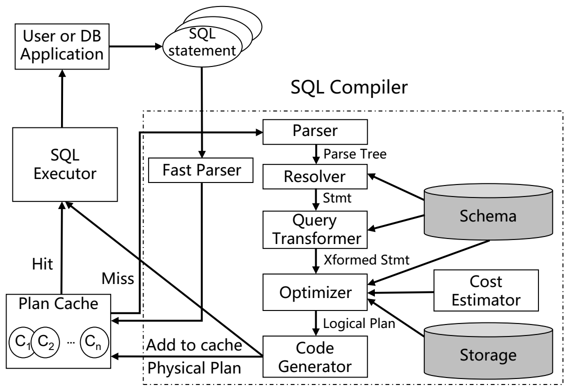


Figure 2: SQL Engine.

objects such as libraries, tables, columns, and indexes, according to the database schema, and generate a statement tree. After the *Resolver*, the *Transformer* analyzes the semantics of the user SQL, and rewrites the user SQL into other equivalent forms according to the internal rules or cost models, and provides it to the subsequent *Optimizer* for further optimization. The *Optimizer* is responsible for generating the best execution plan, though the output result cannot be executed immediately. Further, it needs to be converted into executable code by the *Code Generator*. When the SQL execution plan is generated, the *Executor* will start the execution process of the SQL. For different types of execution plans, the logic of *Executor* is very different. (1) For local execution plans, the *Executor* will merely call from the operator at the top of the execution plan, and the entire execution process will be completed by the operator's own logic, and the execution results will be returned. (2) For distributed plans, the *Executor* needs to divide the execution tree into multiple schedulable threads according to the preselected divisions, and send them to relevant nodes for execution through RPC.

2.5 Multi-tenancy

For OceanBase, multi-tenancy is an important feature, which forms the basis of database object management and resource management. It also has a significant impact on the system operation and maintenance. OceanBase includes two categories of tenants, viz., system and ordinary tenants.

2.5.1 System Tenant. The system tenant is built-in to the system and has three main functions, viz., (1) being the container of the system table, where all system tables are stored in the space of the system tenant, and (2) being the container for users with cluster management functions: cluster level management functions, such as addition or deletion of the tenants, and modifying system configuration items, are only allowed to be performed by users under system tenants. (3) It provides resources required to perform the system maintenance and management actions.

2.5.2 Ordinary Tenant. An ordinary tenant can be regarded as an instance of OceanBase, which is created by the system tenant according to the needs, e.g., the needs of a business. While creating a tenant, besides specifying the tenant's name, the primary issue is to specify the resources it occupies. An ordinary tenant is equivalent to a MySQL instance; hence it naturally has all the features that a database instance should have. Specifically, (1) it can create its own users, (2) all objects such as the database and table can be created, and (3) it has its own independent information, e.g., schema. Further, (4) it has its own independent system variables, and (5) other features of a database instance.

2.5.3 Resource Isolation. To ensure that there is the least competition for resources among tenants and ensure stable business operation, OceanBase isolates resources among tenants as much as possible. In OceanBase, a *Resource Unit* is regarded as a basic unit for allocating resources to tenants. A *Resource Unit* can be analogous to a Docker container, and it includes CPU, Memory, IOPS, Disk Size, and Session Number. Multiple *Resource Units* can be created on an OBServer. Each *Resource Unit* created on an OBServer will occupy a portion of the OBServer's CPU, memory and other resources. The resource allocation of OBServer will be recorded in an internal table for DBA to view. A tenant can place multiple *Resource Units* on multiple OBServers, but a specific tenant can only have one *Resource Unit* on an OBServer. Multiple *Resource Units* of a tenant are independent of each other.

The resource isolation in OceanBase is the behavior of the OBServer to control the resource allocation among multiple local *Resource Units*, which is the local behavior of the OBServer. Similar technologies are Docker and virtual machines, but OceanBase does not rely on Docker or virtual machine technology, and it implements resource isolation within the database. Compared with Docker and virtual machines, the tenant isolation of OceanBase database is more lightweight. The effects of OceanBase resource isolation are as follows:

1) Memory is completely isolated, e.g., the memory used by various operators in the SQL execution process is separated, and the memory exhaustion of one tenant will not affect another tenant.
2) CPUs are isolated through user-mode scheduling.
3) Data structures are separated, e.g., SQL's Plan Cache is tenant-separated, and the retirement of Plan Cache of one tenant will not affect another tenant.
4) Transaction-related data structures are separated, e.g., a transaction of a tenant is suspended and does not affect other tenants.

2.6 Features

OceanBase has six main features as listed below.

- High performance: the storage adopts the read-write separation architecture, the thorough performance optimization of the computing engine, and the performance of the quasi memory database.
- Low cost: using PC server, the high storage compression ratio reduces the storage cost, and efficiently reduces the computing cost, besides the multi-tenant deployment making full use of the system resources.
- High availability: data is stored with multiple copies, and therefore, the failure of a few copies does not affect the data availability. Through the deployment of 5-zone among 3-region, the automatic lossless disaster recovery of the urban level faults has been realized.

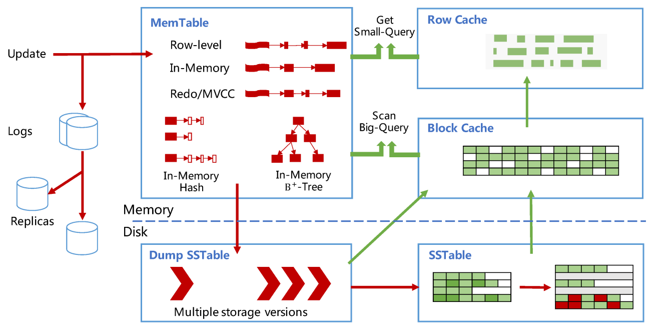


Figure 3: Storage Engine of OceanBase.

- Strong consistency: strong consistency is guaranteed by Paxos. By default, read and write operations have been performed in the primary replica to ensure strong consistency.
- Scalability: the cluster nodes are all peer-to-peer, and each node has computing and storage capabilities without a single point of bottleneck. It supports linear, online expansion, and contraction.
- Compatibility: it is compatible with the common MySQL functions and MySQL front, besides the background protocols. Businesses can be migrated from MySQL to OceanBase with zero modification or a few minor modifications.

3 STORAGE ENGINE

In this section, we introduce our LSM-tree-based storage engine, which supports the asymmetric read and write data block, daily incremental major compaction, and different replica types.

3.1 LSM Tree-Based Architecture

OceanBase has a Log-Structured Merge-tree (LSM-tree) storage system, similar to Bigtable [17]. Based on the LSM-tree architecture, the storage engine of OceanBase is depicted in Figure 3. The data has been grouped into two parts, viz., the static baseline data (placed in SSTable) and dynamic incremental data (placed in MemTable). SSTable is read-only and will not be modified after it is generated. MemTable supports reading and writing, and is stored in the memory. Database data manipulation language (DML) operations such as insert, update, and delete are first written to MemTable. When MemTable reaches a certain size, it is dumped to disk and becomes SSTable. While querying, we need to query SSTable and MemTable separately, merge the query results, and return the merged query results to the SQL layer. Concurrently, *Block Cache* and *Row Cache* are implemented in the memory to reduce the random reading of the baseline data.

When the incremental data in the memory reaches a certain scale, OceanBase will perform minor compaction, i.e., to convert MemTable to SSTable. Furthermore, the system will perform the daily incremental major compaction to merge the mutations and produce a new version of the baseline, and a more detailed discussion is given in §3.3. Since OceanBase adopts a baseline plus increment design, a part of the data is in the baseline and the remaining in the increment. In principle, each query needs to read both the baseline and increment. Therefore, OceanBase has made a lot of optimizations, especially with respect to the single-row.

Besides caching the data blocks inside OceanBase, rows are also cached. Row caching will significantly accelerate the query performance of a single row. For "empty checks" where rows do not exist, Bloom filters could be constructed and cached. Bulk of the OLTP business operations are small queries. Through small query optimization, OceanBase achieves a performance close to that of the in-memory database.

3.2 Asymmetric Read and Write

There are often heavy read and write operations in a database. Similar to a classical RDBMS, the basic read unit of OceanBase, called microblock, has a relatively small size, e.g., 4KB and 8KB, and it is configurable through a database administrator. On the contrary, the write unit of OceanBase, called macroblock, is 2MB. Macroblock is also the basic unit of allocation and garbage collection of the storage system. Many microblocks are packed into a macroblock and this makes the disk utilization more efficient at the cost of a larger write amplification.

3.3 Daily Incremental Major Compaction

OceanBase is essentially based on a baseline plus incremental storage engine, which is different from the classical relational databases. OceanBase divides the data into many macroblocks of 2MB size. During a major compaction, if there is certain data modification (insert, update, delete) within a macroblock, the macroblock will be rewritten, otherwise, the macroblock will be reused in the new baseline data without any IO cost. This makes the major compaction cost of OceanBase significantly lower than that of LevelDB [4] and RocksDB [9]. Furthermore, OceanBase staggers the normal service and the merge time through a round-robin compaction mechanism, thus isolating the normal user requests from the interference of the compaction operation.

Since the major compaction is generally scheduled during the off-peak business time, which implies further vacant CPU clock cycles and memory, OceanBase can employ a more aggressive compression algorithm which yields a higher data compression ratio with no performance damage. Generally, only the modified macroblocks need to be rewritten during the daily incremental major compaction. When the schema of a table is modified, e.g., adding or removing a column, or changing the attribute of a column, it is just a metadata operation, as is the case in classic RDBMS. For example, when adding a column, the new column will gradually be filled in the background through progressive merge, so that the query performance will be better. To minimize the ensuing impact on the business, this full rewriting can be configured for a gradual execution, e.g., 10% per day.

Unlike a major compaction, a minor compaction in OceanBase will compact the in-memory mutations, viz. the MemTable, to the disk and free the memory occupied by the MemTable. Several minor compactions might be merged in another, usually larger, compaction.

3.4 Replica Type

In OceanBase, a complete replica of a partition or table consists of the baseline, mutation increment, and redo log and such a complete replica is called a full replica. Besides full replica, there are certain other replica types as listed below.

- Data replica: a data replica consists of the baseline and redo log. A data replica copies the minor compactions (minor compacted mutations) from a full replica on demand. A data replica can be upgraded to a full replica when it completes the replaying redo log after the last minor compaction has received from a full replica. Data replica can reduce both the CPU and memory cost by eliminating the redo log replay and MemTable.
- Log replica: a log replica consists of redo log only. The Log replica is a member of the corresponding Paxos group, though there exists neither MemTable nor SSTable. By the deployment of two full replicas and one log replica instead of three full replicas, a system still owns quite a high availability while the storage and memory cost is significantly reduced.

Table 1 gives a comparison of different replica types.

Table 1: Replica comparison.

Type	Log	MemTable	SSTable
Full replica	Yes, vote	Yes	Yes
Data replica	Yes, vote	No	Yes
Log replica	Yes, vote	No	No

4 TRANSACTION PROCESSING ENGINE

In this section, we proceed to present our transaction processing engine in details.

4.1 Partition and Paxos Group

A table partition is the basic unit for the data distribution, load balance, and Paxos synchronization. Typically, there is a Paxos group for each partition. For example, there is a Paxos group for a partition *P*7, in which *P*7 in *Zone 3* is a primary replica as depicted in Figure 4.

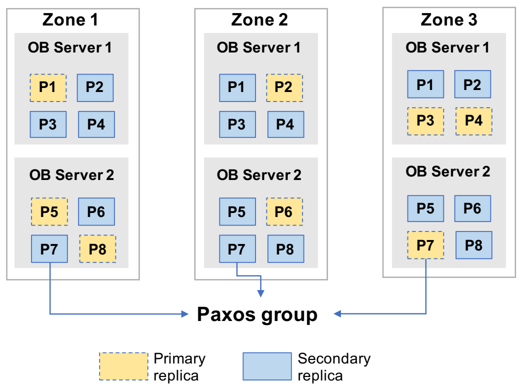


Figure 4: Paxos group.

4.2 Timestamp Service

To enable a high available timestamp service, a timestamp Paxos group has been used. Paxos leader of the timestamp Paxos group is often in the same region as Paxos leaders of the table partitions.

Each OceanBase node retrieves the timestamp from the timestamp Paxos leader periodically.

4.3 Transaction Processing

The traditional two-phase commit (2PC) protocol is often used to implement the distributed transactions. Taking the distributed transfer as an example, assuming that the account *UA* on the server node *A* transfers money to the account *UB* on the server node *B*, the modification of the account value is completed during the execution of the SQL statement, including the verification of the value, etc. Node *A* and node *B* respectively check whether the status of the accounts *UA* and *UB* are normal, the balance of account *UA* is sufficient to be transferred out (no limit exceeded), and the account *UB* can be transferred in (not frozen). The account *UA* and *UB* will be locked if the check is passed. The steps of the two-phase commit are given below.

1) **prepare phase**: The *prepare* phase is to generate redo logs and persist all the determined values.
2) **commit phase**: If the *prepare* operations of the accounts *UA* and *UB* are successful in the first phase, the node *A* is notified to deduct the balance of the account *UA*. Further, the node *B* is notified to add the balance of the account *UB*, and the transfer is successful. Alternatively, node *A* and node *B* are notified to roll back the operation of the corresponding account, and the transfer is cancelled.

However, in a shared-nothing scenario, if a node, such as *A*, fails during the execution of the two-phase transaction, the status of *A*'s operation on account *UA* is inaccessible. The *prepare* of account *UA* may not be completed, or completed and succeeded, or completed but failed. Further, Node *A* may recover quickly, or it may not recover for a long time, or is permanently damaged. It is not even possible to assess the failure or recovery of the node *A*, and hence the execution result of the distributed transaction is uncertain. For example, the monitoring node cannot always determine the status of the monitored node, since the monitoring node itself or the communication between it and the monitored node may be abnormal.

OceanBase introduces the Paxos distributed consistency protocol to 2PC [24], enabling the distributed transactions with automatic fault tolerance. As shown in Figure 5, each participant in the two-phase commit contains multiple copies, and the copies are readily available through the Paxos protocol. When a participant node fails, the Paxos protocol can quickly elect another replica to replace the original participant to continue providing services, and restore the state of the original participant. Thereby, it determines the execution of the distributed transaction and continues advancing the completion of the two-phase commit agreement.

To enhance the performance of the distributed transaction processing and reduce the latency, OceanBase further improves the traditional two-phase commit protocol by adopting the optimization of operations of both the participant and the coordinator. According to Figure 6, the coordinator maintains the state of the distributed transaction, and responds to the client after performing the *prepare* and *commit* operations. Each operation needs to be logged before the coordinator processes. This reduces the two-phase commit to

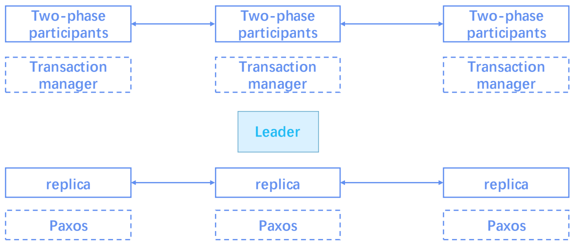
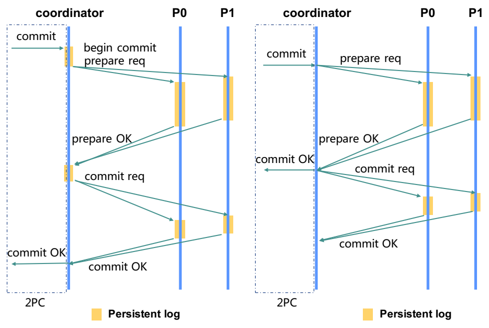


Figure 5: Paxos-based 2PC.



(a) Traditional 2PC (b) OceanBase 2PC

Figure 6: Traditional 2PC vs. OceanBase 2PC.

three Paxos synchronizations instead of four synchronizations because traditional 2PC with presumed abort [32] does not require logging in the coordinator at the beginning of 2PC, as shown in Figure 6(a). In OceanBase, the first participant of each distributed transaction is the coordinator of the two-phase commit. Furthermore, the coordinator terminates in the state of the two-phase commit, but dynamically constructs it through the local state of all the participants during a disaster recovery. This results in two, instead of three Paxos synchronizations, in a two-phase commit, and the delay of a two-phase commit is further reduced to one Paxos synchronization, as shown in Figure 6(b). Note that OceanBase can add a switch to control whether a read optimization is turned on. When this optimization is turned on, it causes an extra network round trip in OceanBase 2PC. When all *prepare* logs of a transaction are persisted on a majority of the corresponding Paxos group, all states of the transaction are persisted. In a fault scenario, OceanBase 2PC protocol will continue to advance the transaction to the final completion according to whether all participants of the transaction have completed the *prepare* phase.

4.4 Isolation Level

Usually, higher transaction isolation level implies a higher cost and lower performance. For SQL statements with relatively complex predicates, to the best of our knowledge, the performance cost of serializable isolation is significantly high. Therefore, the mainstream classical RDBMS do not take serialization isolation as their default isolation level. For compatibility and performance considerations, OceanBase supports *read committed* and *snapshot isolation*, and makes the former as the default isolation level. A normal read will only happen on the leader, and it refers to a query of general *select*.

In contrast, OceanBase supports a weakly consistent query, which needs to be explicitly specified by a user through hint in SQL.

4.5 Replicated Table

Certain tables are frequently accessed by other ones. For example, a dimension table may be joined to fact tables through a foreign key, frequently. This could significantly degrade the performance in a distributed database system owing to the frequent network accessing.

In OceanBase, a replicated table is replicated on each OceanBase node. There are two kinds of replicated tables, viz., synchronously replicated and asynchronously replicated tables. For a synchronously replicated table, a mutation is committed only after the mutation is performed by every replica of the table. This ensures that any transaction in any OceanBase node will see the same content of a synchronously replicated table. But synchronous mutation of all replicas deteriorates the write performance of the corresponding replicated table.

For an asynchronously replicated table, however, a mutation is committed after the redo log of the mutation continues on a majority of the Paxos group of the table. Thereafter, the mutation will spread to all replicas of the table in a cascading mode. Asynchronous mutation of all replicas guarantees the write performance of the corresponding replicated table such that replicas except the Paxos leader may have the slightly old version of data. If a transaction encounters a very old version of an asynchronous replicated table, it will attempt a remote replica.

5 TPC-C BENCHMARK TEST

In this section, we proceed to present the TPC-C benchmark test that we performed in 2020, including the benchmark configuration, challenges, and results.

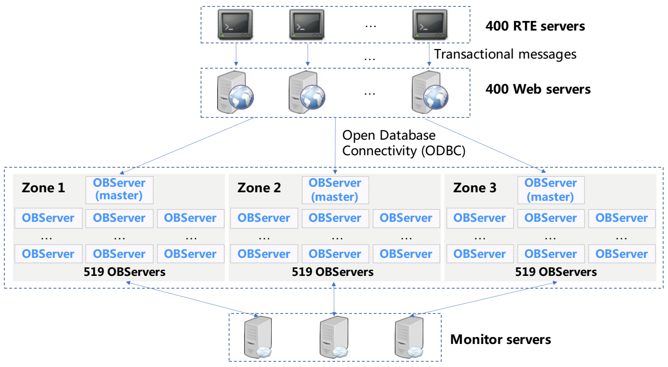


Figure 7: TPC-C benchmark test topology.

5.1 Benchmark Configuration

In May 2020, OceanBase performed the TPC-C benchmark test for the second time. TPC-C benchmark test topology is depicted in Figure 7. A total of 2,360 Alibaba Cloud ECS servers have been employed in this benchmark test.

- There are 400 remote terminal emulator (RTE) servers to emulate the total 559,440,000 users, and RTE parameters is shown in Table 2. A remote terminal emulator emulates a user entering input data on the input (output) screen by generating and sending the transactional messages to a web server.

Table 2: RTE parameters.

Parameters	Setting
Ramp-up Duration	3,300 seconds
Ramp-down Duration	150 seconds
Measurement Interval	28,800 seconds
Database Scale	55,944,000 warehouses
Total terminals	559,440,000
Terminals/Driver	55,944
Number of RTEs nodes/instances	10,000

- There are 400 web servers. A web server receives requests from several RTEs and implements TPC-C transactions as SQL stored procedures [1] via the SQL engine of OceanBase and invokes the data servers through the Open Database Connectivity (ODBC) interface.
- The OceanBase cluster in this benchmark test consists of 1,557 servers in a shared-nothing architecture. Each server has 84 vCPUs (2.5GHz Intel Xeon Platinum 8163 Skylake hyper-threading processor), 712GB RAM, and 3.5TB*4 SSD. These servers are evenly divided into 3 zones with 519 servers in each zone, with 1,554 data servers and 3 management servers in charge of the management of the cluster. These 1,554 data servers store all the data of the 9 TPC-C tables and run all 5 TPC-C transactions. Note that in order to be compatible with legacy databases, e.g., MySQL, as much as possible, tables need to be explicitly partitioned. All tables except the ITEM table are partitioned and spread to all data servers. The ITEM table is configured as a synchronous replicated table and is replicated on each data server, which is a space-time tradeoff.
- There are 3 monitor servers in charge of the monitoring of the OceanBase cluster.

5.2 Challenges

Many challenges that have been encountered and solved, while running the TPC-C benchmark test and meeting the requirements specified by the benchmark on a cluster consisting of 1500+ servers, are given below.

The first challenge is the durability requirement by the benchmark specification. Single region with 3 zones deployment has been chosen for this shared-nothing database system running on commodity hardware. Each Paxos group consists of 3 replicas, viz., a full replica, a data replica, and a log replica. Compared with the 3 full replicas, this configuration cuts two-third of RAM used by MemTable and one-third of storage used by the baseline data. It also cuts certain CPU cycles by eliminating the redo log replaying on the data replica and log replica. However, this also accrues certain network bandwidth cost and disk IO cost as every data replica needs

[1]Stored procedures are considered part of the application program in TPC-C benchmark [21].

to copy the minor compacted MemTable (minor compactions) from the corresponding full replica, periodically.

The second challenge is with respect to the time-consuming process of the generation of the initial data of the benchmark test. Owing to the number of rows per warehouse (499,000+) and the huge number of warehouses (total 55,944,000 and 36,000) for each OceanBase node, and the required data population, the generation of the initial data of the benchmark test is heavily taxing. To cut this initial data generation time, the OceanBase cluster is first configured as 1 replica instead of 3 replicas and no-logging. Besides, thousands of rows are batch-inserted into the database through a single transaction during the initial data generation period. Following the insertion of all initial data, the database is reconfigured as 3 replicas, i.e., one full replica, one data replica, and one log replica, and the partitions are re-replicated and rebalanced automatically. No-logging is also turned off, and the system is ready for testing after the above-said re-replicating and rebalancing, along with a major compaction, are completed.

The third challenge is the ITEM table. Since ITEM table is frequently accessed by the transactions in the TPC-C benchmark, it should be replicated on every OceanBase node to avoid remote access and guarantee the performance. Besides, ACID properties must be met for any transaction on the ITEM table. Therefore, the ITEM table is configured as a synchronous replicated table.

The fourth challenge is that the variations of the cumulative performance throughput during the benchmark test measurement interval should not exceed 2% according to the TPC-C benchmark specification. There are several background operations in OceanBase, e.g., minor compaction, merging compaction (merging several minor compactions into one), and copying minor compactions from a full replica to a data replica. The CPU threads pool for all the background operations are reserved to minimize the performance throughput variations. The upper limit of the MemTable size should be chosen to be small enough for minor compaction to be completed before the RAM is exhausted by new transactions during minor compaction. Further, it must be large enough to exploit the RAM of each node and produce minimal minor compactions. Finally, the cumulative variations of the performance throughput during the eight hours benchmark test measurement interval is less than 1%.

5.3 Results

5.3.1 tmpC. Using the above configuration, TPC-C benchmark tests have been run on OceanBase clusters with 3, 9, 27, 81, 243, 510, and 1554 data servers for an eight hour measurement interval. The performance (*tpmC*) of these tests are displayed in Figure 8. We can see that *tpmC* rises linearly as the number of data nodes increases. Figure 8 shows that OceanBase is highly scalable. Furthermore, OceanBase, with an online transaction processing performance of 707 million tpmC in 2020 [8], has broken the TPC-C world record of 60 million tpmC that it created in 2019 [1]. The cumulative tpmC variations during these tests are quite small as depicted in Figure 9, where the smallest and largest top jitters are 0.03% and 0.37%, respectively, and the largest and smallest bottom jitters are -0.03% and -0.81%, respectively.

5.3.2 New-Order. New-Order response time is shown in Figure 10. The maximum, ninetieth percentile, average, and minimum response times are reported for the *New-Order* transaction type, and

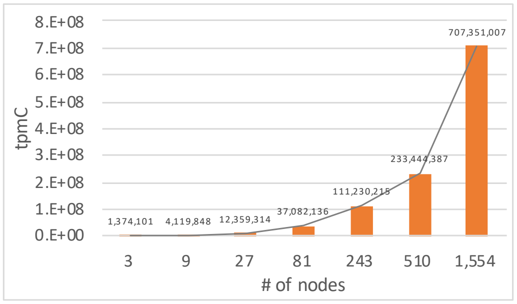


Figure 8: tpmC.

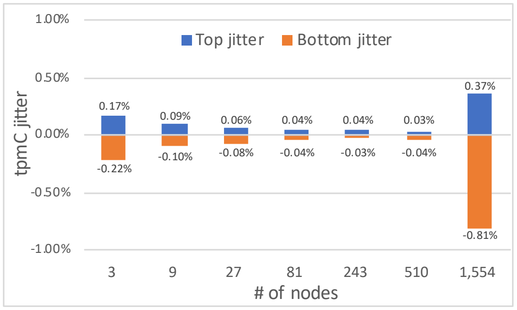


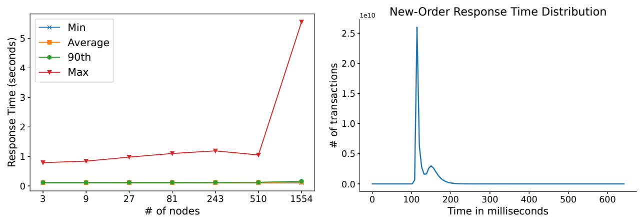
Figure 9: tpmC jitter.

the last three ones are overlapped as shown in Figure 10(a). All the minimum ones are 103ms, whereas the average and ninetieth percentile ones are proximate, with the smallest and largest differences being 5ms and 33ms, respectively. The 90th percentile response time for *New-Order* is greater than the average response time of that transaction [2]. There are marked gaps only for all the maximum ones. From the perspective of *New-Order*, we can see that OceanBase is highly scalable. This is because: 1) Through the transparent forwarding of stateless OBProxy, distributed transactions are greatly reduced, and local transactions are also optimized. 2) Distributed transactions are accelerated via OceanBase 2PC. 3) Stored procedures speed up transaction execution. 4) The optimized LSM-tree reduces transaction write operations. 5) Fast SQL parameterization is very efficient, which is useful for OLTP short query scenarios.

Meanwhile, the *New-Order* response time distribution is shown in Figure 10(b), where the vast majority of all *New-Order* transactions are completed within 20ms. The vast majority of all the other transactions, except *Delivery*, are also completed within 20ms. Note that the emulated display delay is 100ms according to the requirements of the TPC-C benchmark test [21], and real response time is its experimental value minus 100ms.

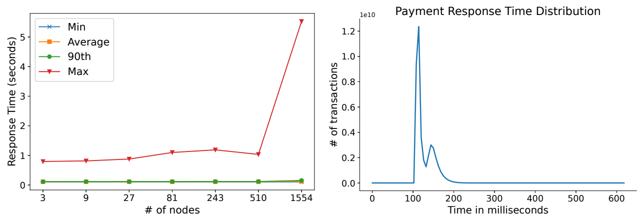
5.3.3 Payment. Figure 11 shows the *Payment* response time. Figure 11(a) demonstrates the minimum, average, ninetieth percentile, and maximum response times for the *Payment* transaction type, where the first three ones are overlapped. The smallest average one is 110ms (3 nodes), and the largest average one is 123ms (1,554 nodes), whereas the smallest ninetieth percentile one is 114ms (3 nodes), and the largest ninetieth percentile one is 154ms (1,554 nodes). Their differences are smaller than 100ms, viz. 4ms and 31ms.

[2]Based on one of rules of TPC-C: if the 90th and the average response times are different by less that 100ms, then they are considered equal [21].



(a) Scalability (b) RT Distribution with 1,554 nodes

Figure 10: New-Order Response Time (RT).

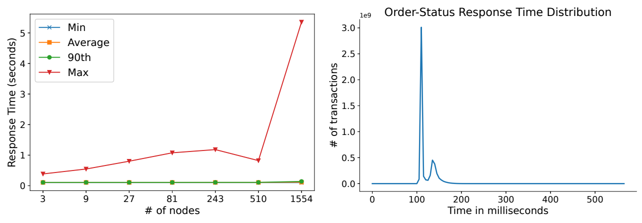


(a) Scalability (b) RT Distribution with 1,554 nodes

Figure 11: Payment Response Time (RT).

The 90th percentile response time is greater than the average response time in *Payment*. For the first three response times, they are almost stable lines with negligible gaps. Similarly, Figure 11(a) presents that OceanBase is highly scalable because of the five same reasons with the *New-Order* transaction type. Concurrently, the *Payment* response time distribution with 1,554 nodes is shown in Figure 11(b).

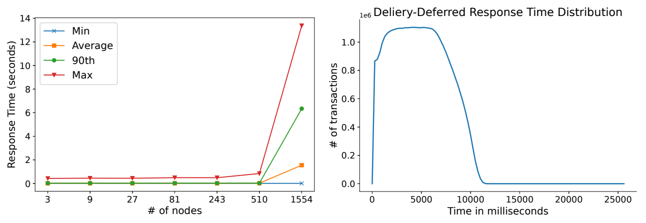
5.3.4 Order-Status. Order-Status response time is illustrated in Figure 12. The maximum, ninetieth percentile, average, and minimum response times are reported for the *Order-Status* transaction type, where the last three ones are overlapped as shown in Figure 12(a). The smallest and largest averages are 107ms and 117ms, respectively, whereas the smallest and the largest ninetieth percentile is 109ms and 141ms, respectively. Their differences are 2ms and 24ms, which are much smaller than 100ms. Simultaneously, Figure 12(b) demonstrates the *Order-Status* response time distribution with 1,554 data nodes. Due to the five same reasons, the *Order-Status* transaction type has similar trends to the first two transaction types.



(a) Scalability (b) RT Distribution with 1,554 nodes

Figure 12: Order-Status Response Time (RT).

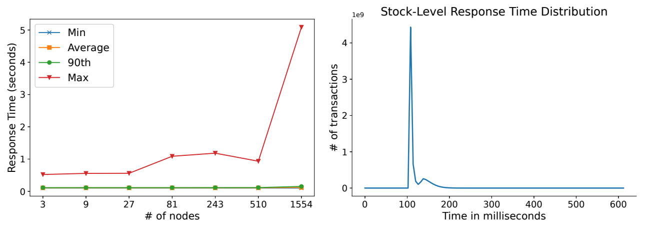
5.3.5 Delivery. Figure 13 illustrates the *Delivery* response time. Figure 13(a) shows the ninetieth percentile, maximum, minimum, and average response times for the *Delivery* transaction type. In *Delivery*, the 90th percentile response time is greater than the average response time. For 3, 9, 27, 81, 243, and 510 nodes, the smallest and the largest average values are 17ms and 19ms, respectively, while the smallest and largest ninetieth percentile values are 22ms and 27ms, respectively. Their differences are much smaller than 100ms, viz., 5ms and 8ms. Meanwhile, the *Delivery* response time distribution over 1,554 nodes is shown in Figure 13(b). The average response time is 1.55 seconds and the ninetieth percentile response time is 6.34 seconds, which are longer than those of other transaction types, because the *Delivery* transactions are heavy tasks.



(a) Scalability (b) Delivery RT with 1,554 nodes

Figure 13: Delivery Response Time (RT).

5.3.6 Stock-Level. Stock-Level response time is shown in Figure 14. The maximum, ninetieth percentile, average, and minimum response times are reported for the *Stock-Level* transaction type, and the last three ones are overlapped as shown in Figure 14(a). With respect to *Stock-Level*, the 90th percentile response time is greater than the average response time. The smallest and largest average values are 109ms (3 nodes) and 120ms (1,554 nodes), respectively, whereas the smallest and the largest ninetieth percentile values are 114ms (3 nodes) and 152ms (1,554 nodes), respectively. Their differences are much smaller than 100ms, viz., 11ms and 32ms. Simultaneously, Figure 14(b) illustrates the *Stock-Level* response time distribution over 1,554 data nodes. The *Stock-Level* transaction type has similar trends to other ones except the *Delivery* transaction type owing to the same previous five reasons.



(a) Scalability (b) RT Distribution with 1,554 nodes

Figure 14: Stock-Level Response Time (RT).

Owing to the space constraint, we do not present more TPC-C benchmark results, e.g., keying and think times. More TPC-C benchmark results and information are shown in [2].

6 LESSONS IN BUILDING OCEANBASE

In this section, we detail what we have learnt in building OceanBase for over ten years.

6.1 From NoSQL to NewSQL

We initially built OceanBase as a distributed NoSQL storage system for the Favorite of Taobao.com [3] in 2010. As shown in Figure 15, there are three stages for OceanBase evolution, viz., (1) Stage 1: Towards a distributed architecture from 2010 to 2012, primarily focusing on distributed NoSQL storage, (2) Stage 2: Native distributed database from 2013 to 2016, starting with SQL engine and enhancing Paxos-based high availability, (3) Stage 3: Hybrid transactional and analytical processing (HTAP) system from 2017 to current, primarily aiming at distributed transactions, inter-region fault tolerance and HTAP.

The lessons learnt from practice, while building OceanBase from a distributed NoSQL storage system to a distributed NewSQL relational database system, are given as follows. 1) The application layer should not use a database system as a key-value storage system, and should rely on the advanced features of database; 2) The stored procedure still has great value for OLTP applications; 3) For the applications using distributed databases, every transaction and every SQL should have a timeout set because of the higher failure rate of distributed system networks and nodes. If the application sets a timeout, the database may be retried in many failure situations to improve robustness. The infinite timeout period may lead to *logical* deadlock in complex situations, which is not conducive to the overall disaster recovery.

6.2 Both cost and performance

The LSM-tree architecture is naturally more suitable for data encoding and compression. Part of the extra CPU consumption is exchanged for a great reduction in storage cost. Concurrently, it has no impact on the performance of OLTP services. Instead, the encoding features are used in certain scenarios for improved performance, e.g., higher cache hit rate, faster query, and lower IO cost.

OceanBase encodes and compresses data in the microblock according to the mode specified by the user table including *flat* and *encoding*. When *encoding* is turned on in the user table [3], the data in each microblock will be encoded in the column according to the column dimension. The encoding rules include Dictionary Encoding, Run-Length Encoding, and Delta Encoding. Coding can help users to compress the data, besides accelerating the subsequent query speed by the extracted feature information in the column. After encoding and compression, OceanBase supports further lossless compression of microblock data using a user-specified general compression algorithm for improving the data compression rate. For example, a business system of Alipay was migrated from Oracle to OceanBase, and the data was compressed from 100TB to 33TB.

6.3 Data validation

As a financial-level relational database, OceanBase has persistently prioritised data quality and security first. Every data part involving persistence in the whole data link will enhance data validation and protection. Concomitantly, by exploiting the inherent advantages of multi-copy storage, data validation between replicas is added to further validate that the overall data is consistent.

[3]It is supported in commercial OceanBase instead of open-source OceanBase.

In a common deployment mode, each user table of OceanBase will have multiple copies in the cluster. When the cluster is merged daily, all copies will generate consistent baseline data based on the globally unified snapshot version. Using this feature, all copies will be compared with the checksum of the data when the merge is completed to ensure complete consistency. Further, based on the index of the user table, the checksum of the index column will continue to be compared to ensure that the index is consistent with the column corresponding to the original table.

6.4 Partitioning vs. sharding

In OceanBase, partitioning refers to breaking a table into smaller, easier-to-manage parts according to certain rules specified by users, such as secondary partitioning and virtual column-based partitioning. However, sharding is based on hash, and there is no secondary sharding. Each partition is an independent object with its own name and optional storage characteristics. For applications accessing OceanBase database, only one table or one index is logically accessed, though this table may be composed of dozens of physical partitions. Partitioning is completely transparent to the application and does not affect the business logic of the application.

From the perspective of the application, there is only one schema object. Access to partitioned tables does not require modification of the SQL statement. Partitions are useful for many different types of database applications, especially those that manage large amounts of data. A partitioned table can consist of one or more partitions that are managed individually and can operate independently of other partitions. OceanBase stores the data of each table partition in its own SSTable, and each SSTable contains a part of the table data. OceanBase can support from tens of thousands to millions of partitions.

Compared with sharding, partitioning has the following benefits:

1) **Improved usability.** An unavailable partition does not imply that the object is unavailable. The query optimizer automatically removes unreferenced partitions from the query plan. Therefore, queries are not affected when a partition is unavailable. However, the queries are affected when a shard is unavailable.
2) **Manage objects more easily.** Partition objects have fragments that can be managed collectively or individually. Different from sharding, DDL (Data Definition Language) statements can operate on partitions rather than entire tables or indexes. Therefore, resource-intensive tasks such as rebuilding indexes or tables can be decomposed. If something goes wrong, we just redo the partition move instead of the table move. Also, we can *TRUNCATE* the partitions to avoid to *DELETE* large amounts of data.
3) **Reduce contention for shared resources in OLTP systems.** In OLTP scenarios, partitioning can reduce the contention for shared resources, but sharding cannot, e.g., DML is distributed over many partitions instead of one table.
4) **Enhance query performance in the data warehouse.** In OLAP scenarios, partitioning can speed up the processing of ad-hoc queries. However, sharding only can limitedly accelerate it due to no secondary sharding. Partition keys have natural filtering capabilities. For example, to query the

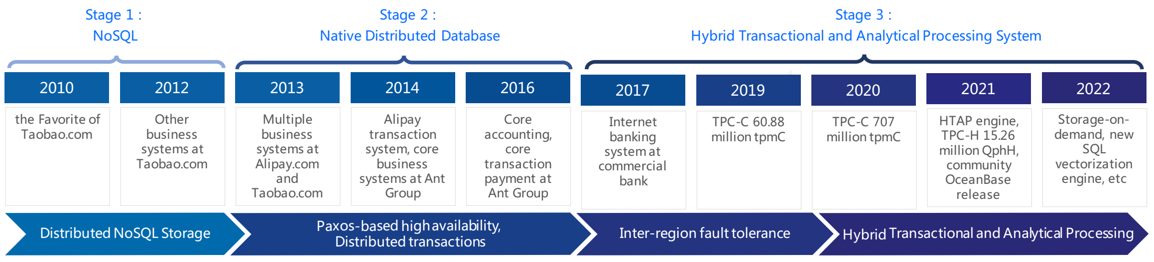

Stage 1 : NoSQL
Stage 2 : Native Distributed Database
Stage 3 : Hybrid Transactional and Analytical Processing System
2010
the Favorite of Taobao.com
2012
Other business systems at Taobao.com
2013
Multiple business systems at Alipay.com and Taobao.com
2014
Alipay transaction system, core business systems at Ant Group
2016
Core accounting, core transaction payment at Ant Group
2017
Internet banking system at commercial bank
2019
TPC-C 60.88 million tpmC
2020
TPC-C 707 million tpmC
2021
HTAP engine, TPC-H 15.26 million QphH, community OceanBase release
2022
Storage-on-demand, new SQL vectorization engine, etc
Distributed NoSQL Storage
Paxos-based high availability, Distributed transactions
Inter-region fault tolerance
Hybrid Transactional and Analytical Processing

Figure 15: OceanBase Evolution.

sales data of a quarter, when the sales data is partitioned by sales time, only one partition or several partitions need to be queried, not the entire table.

5) **Provide better dynamic load balancing performance.** The storage unit and load balancing unit of OceanBase are partitions. Different partitions can be stored on different nodes. Therefore, a partitioned table can distribute different partitions on different nodes, so that the data of a table can be distributed evenly in the entire cluster. However, sharding can hardly achieve dynamic load balancing.

6.5 Internet vs. Non-Internet companies

With the development of the Internet, the processing of massive data has increasingly become a problem for large Internet companies. The services provided by traditional IT companies are no longer applicable in terms of system scalability and cost-effectiveness. Large Internet companies uses large memory and multiple CPUs to maximize high performance, while traditional IT companies pursue mainstream configuration and high cost performance. For databases, compared with traditional enterprises, one of the biggest differences of Internet enterprises is high concurrency.

In traditional commercial enterprises and banks, users need to conduct business and access databases through special equipments such as bank terminals, ATM teller machines, and POS machines. Hundreds and thousands of concurrent requests to databases are relatively common, whereas requests exceeding tens of thousands of concurrent requests are scarce. On the Internet, every user can initiate a shopping transaction and access the database. Hundreds of thousands of concurrent database accesses are a common occurrence, and millions or even tens of millions of concurrent accesses can also be seen (such as Taobao.com, Tmall.com and Alipay.com under “Double 11”). Owing to such large concurrent access, the cost of commercial database software and its highly reliable database server and shared storage has become unaffordable. Compared with traditional databases, one of key features of OceanBase is the grayscale upgrade of software versions for large-scale distributed clusters with commodity machines. Through the grayscale upgrade, OceanBase avoids the “one-shot deal” upgrade of traditional databases, which greatly reduces the risk of database maintenance and upgrades.

7 RELATED WORK

The literature available in Google, viz. for 2012 and 2013, has established the position of Spanner [19, 20] and F1 [40, 41] as the founder of NewSQL database supporting both OLTP and OLAP. Spanner has evolved into a complete database with its own SQL Engine [15], taking into account both OLAP and OLTP scenarios since 2012.

7.1 In-Memory Database Systems

Peloton [36] is a single-node, in-memory database designed for self-driving database management with SQL support. Here, data origination [14] has been utilized for the HTAP workloads at run time. It employs lock-free, multi-version concurrency control (MVCC) to support real-time analytics.

SAP HANA [22] is an in-memory database engine designed for HTAP workloads. To provide real-time analytics (OLAP), it depends on MVCC with a main and delta data-structures, in which the delta is periodically merged with the main. A scale-out extension of HANA named SAP HANA SOE [23] has been proposed to support large scale analytics over real-time data. BatchDB [30] depends on the primary-secondary replication with dedicated replicas, and each being optimized for OLTP or OLAP, and it minimizes load interaction between OLTP and OLAP engines, thus enabling the real-time analysis over fresh data for both HTAP workloads.

HyPer [26] is a hybrid OLTP and OLAP main memory database system based on virtual memory snapshots, and it relies on single-threaded in-core partitioned database. It employs an MVCC implementation that offers serializability, fast transaction processing, and fast scans [33].

MemSQL [39] supports both HTAP workloads offering distributed query processing, dynamic code generation, and high performance in-memory data-structures, e.g., lock-free skip-lists. With respect to the generation of highly efficient distributed query execution plans with fast optimization times, the MemSQL Query Optimizer [18] has proposed as a modern optimizer for MemSQL designed to optimize complex queries efficiently and effectively.

7.2 Distributed Database Systems

Google is a pioneer in distributed HTAP systems. Google Percolator [37] is one of precursors of the distributed transaction processing systems used in production despite its relatively long transaction latency (e.g., a few seconds) and its lack of a query language, e.g., SQL. Google Spanner [19][15] implements quite perfect distributed transaction processing. There are several similarities between the Spanner and OceanBase: (1) Both use commodity hardware instead of a high-end server and high reliable storage used by classical RDBMS. (2) Both employ LSM-tree. (3) Both utilize Paxos replication instead of primary and backup mirroring. (4) Both offer SQL interface.

F1 Lightning [43] enables high-performance analytic queries over hybrid query workloads on top of the existing OLTP systems. It has been deployed for the business-critical transactional databases, e.g., AdWords [31]. It saves up to orders of magnitude in computational resources, and it decreases the query latency without compromising on the query semantics. However, there are significant differences between Spanner and OceanBase:

- OceanBase is identical to classical RDBMS despite its distributed property. OceanBase implements various kinds of data types, viz., SQL syntax, various constraints, various triggers, cursor and stored procedure, and JDBC and ODBC of classical RDBMS. OceanBase is also compatible with certain mainstream classical RDBMS that enables the applications based on these RDBMS to migrate to OceanBase with or without only a few minor modifications.
- OceanBase can be deployed using hundreds or even thousands of high-profile nodes which are suitable for high performance and huge volume of data of large organizations. It can also be deployed for employing a few low-profile nodes which are suitable for small organizations.
- Many organizations do not own a high-precision atomic clock, and therefore, OceanBase is designed to be independent of an atomic clock. OceanBase can work normally if the clock skew between the servers is under certain threshold, e.g., 100ms and clock skew has no impact on the performance and transaction response time.
- OceanBase provides multi-tenancy that is supported by certain classical RDBMS. It further cuts the TCO of customers.

CockroachDB [42] is inspired by Google Spanner, but instead of TrueTime API, it uses HLC (hybrid logical clock), i.e., NTP (Network Time Protocol) and logical clock to replace TrueTime timestamp. It relies on HLC to do transactions, and the accuracy of the timestamp cannot achieve a delay within 10 ms. Therefore, the commit wait needs to be specified by the user, which is unfriendly to the user. Furthermore, it employs Raft [34] as the data replication protocol, and its underlying storage is based on RocksDB. CockroachDB owns a standard shared-nothing architecture, and it employs range-partitioning on the keys to split the table data into contiguous ordered chunks of size ~64 MiB. OceanBase, by itself, does not split the partition or table of users.

The design of YugabyteDB [12] is very similar to that of Google Spanner, and it is an open source NewSQL database developed by Yugabyte. Similar to CockroachDB, YugabyteDB uses hybrid logical clocks instead of specialized hardware clocks for the TrueTime protocol. It supports both snapshot isolation and serializability, and it allows the users to select the level of transaction isolation they need. Furthermore, it is compatible with the PostgreSQL protocol.

TiDB [25] is an open-source distributed database system supporting SQL interface that is compatible with the MySQL wire protocol. It is designed to support both OLTP and OLAP workloads. Although TiDB is independently developed, it adopts open-source products on several key modules, e.g., RocksDB for storage layer and Spark SQL [13] for the distributed parallel computing.

As a large-scale data-warehouse system, Greenplum [29] presents another pathway to evolve into an HTAP database. It adds OLTP capability to a traditional OLAP system, and supports a fine-grained resource isolation. By running the OTLP and OLAP workloads simultaneously in a single system, Greenplum has effectively utilized the CPU and memory using the resource group.

There are numerous works on HTAP from the academic community. For example, ES^2 [16] is an elastic cloud data storage system, which is designed to support both OLTP and OLAP within the same storage. Umzi [28] is the multi-version and multi-zone LSM-like indexing method to support the evolving data across multiple zones with a consistent and unified indexing view in an HTAP database system. Raza et al. [38] have modeled an HTAP database system as a set of three individual engines, including OLTP, OLAP, and Resource and Data Exchange (RDE) engines. To meet the workload data freshness requirements, a scheduling algorithm has been devised to traverse the HTAP design spectrum through the elastic resource management.

8 CONCLUSION

We have built OceanBase, a cross-region fault tolerant distributed database system from the very basics, since 2010. OceanBase can be deployed in a shared-nothing architecture and can scale from a few nodes up to thousands of nodes. It is also compatible with certain mainstream classical RDBMS, e.g., MySQL. This greatly lowers the threshold of migration of business from these centralized RDBMS to OceanBase and significantly cuts the time, cost, and risk of the migration, too. OceanBase has been employed to serve the Favorite of Taobao.com since June 2011. Hitherto, tens of thousands of OceanBase nodes are running in the production units of many organizations. Hence, OceanBase has the following advantages:

❶ **Native distributed architecture.** It has a good horizontal expansion and automatic load balancing capability. No additional database middleware products are required, and there is no invasion over the upper applications, making the distributed architecture completely transparent to the applications.

❷ **Excellent data reliability and service availability.** It strictly follows the Paxos protocol to achieve magnificent data reliability ($RPO = 0$) and service availability ($RTO < 30$ seconds).

❸ **Strict verifications of OLTP.** Its product maturity and stability are perfectly guaranteed by tens of thousands of OceanBase servers running in production for OLTP for years by many commercial organizations.

❹ **Exceptional performance.** It has been verified by hundreds of applications within Alibaba for many years, including the extreme test of "Double 11" for consecutive years.

❺ **Perfect MySQL compatibility.** It meets the needs of most users in the industry and reduces the difficulty of business migration.

Whereas a distributed database compatible with mainstream classical centralized RDBMS is very valuable, its implementation is quite tedious and very challenging. For example, a true distributed deadlock detection and resolution algorithm was not implemented in OceanBase until recently (and it will be presented in another paper). Besides, part of a few constraint features and a few DDL operations are still under development or will be implemented in the near future. OceanBase SQL optimizer, especially for complex SQL statements, is under continuous fine tuning.

REFERENCES

[1] 2019. OceanBase: 60 million tpmC. http://tpc.org/1799.

[2] 2020. Ant Financial TPC Benchmark^{TM} C Full Disclosure Report. http://tpc.org/results/fdr/tpcc/ant_financial~tpcc~alibaba_cloud_elastic_compute_service_cluster~fdr~2020-05-17~v01.pdf.

[3] 2021. Favorite of Taobao.com. https://shoucang.taobao.com.

[4] 2021. LevelDB. https://github.com/google/leveldb.

[5] 2021. MulanPubL-2.0. https://license.coscl.org.cn/MulanPubL-2.0/index.html.

[6] 2021. OceanBase. https://gitee.com/oceanbase.

[7] 2021. OceanBase. https://github.com/oceanbase.

[8] 2021. OceanBase: 707 million tmpC. http://tpc.org/1803.

[9] 2021. RocksDB. https://rocksdb.org/.

[10] 2021. RPO and RTO in OceanBase. https://github.com/kioco/oceanbase-1.

[11] 2021. Wish List of Amazon.com. https://www.amazon.com/hz/wishlist/intro.

[12] 2021. YugabyteDB. https://www.yugabyte.com.

[13] Michael Armbrust, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi, and Matei Zaharia. 2015. Spark SQL: Relational Data Processing in Spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, Timos K. Sellis, Susan B. Davidson, and Zachary G. Ives (Eds.). ACM, 1383–1394.

[14] Joy Arulraj, Andrew Pavlo, and Prashanth Menon. 2016. Bridging the Archipelago between Row-Stores and Column-Stores for Hybrid Workloads. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). ACM, 583–598.

[15] David F Bacon, Nathan Bales, Nico Bruno, Brian F Cooper, Adam Dickinson, Andrew Fikes, Campbell Fraser, Andrey Gubarev, Milind Joshi, Eugene Kogan, et al. 2017. Spanner: Becoming a SQL system. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 331–343.

[16] Yu Cao, Chun Chen, Fei Guo, Dawei Jiang, Yuting Lin, Beng Chin Ooi, Hoang Tam Vo, Sai Wu, and Quanqing Xu. 2011. ES^2: A cloud data storage system for supporting both OLTP and OLAP. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, Serge Abiteboul, Klemens Böhm, Christoph Koch, and Kian-Lee Tan (Eds.). IEEE Computer Society, 291–302.

[17] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. 2008. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)* 26, 2 (2008), 1–26.

[18] Jack Chen, Samir Jindel, Robert Walzer, Rajkumar Sen, Nika Jimsheleishvilli, and Michael Andrews. 2016. The MemSQL Query Optimizer: A modern optimizer for real-time analytics in a distributed database. *Proc. VLDB Endow.* 9, 13 (2016), 1401–1412.

[19] James C Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, Jeffrey John Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, et al. 2013. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems (TOCS)* 31, 3 (2013), 1–22.

[20] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson C. Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. 2012. 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*, Chandu Thekkath and Amin Vahdat (Eds.). USENIX Association, 251–264.

[21] Transaction Processing Performance Council. 2010. TPC BENCHMARK™ C Standard Specification Revision 5.11 Standard Specification.

[22] Franz Färber, Norman May, Wolfgang Lehner, Philipp Große, Ingo Müller, Hannes Rauhe, and Jonathan Dees. 2012. The SAP HANA Database – An Architecture Overview. *IEEE Data Eng. Bull.* 35, 1 (2012), 28–33.

[23] Anil K. Goel, Jeffrey Pound, Nathan Auch, Peter Bumbulis, Scott MacLean, Franz Färber, Francis Gropengießer, Christian Mathis, Thomas Bodner, and Wolfgang Lehner. 2015. Towards Scalable Real-time Analytics: An Architecture for Scale-out of OLxP Workloads. *Proc. VLDB Endow.* 8, 12 (2015), 1716–1727.

[24] Jim Gray and Leslie Lamport. 2006. Consensus on transaction commit. *ACM Trans. Database Syst.* 31, 1 (2006), 133–160.

[25] Dongxu Huang, Qi Liu, Qiu Cui, Zhuhe Fang, Xiaoyu Ma, Fei Xu, Li Shen, Liu Tang, Yuxing Zhou, Menglong Huang, et al. 2020. TiDB: a Raft-based HTAP database. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3072–3084.

[26] Alfons Kemper and Thomas Neumann. 2011. HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, Serge Abiteboul, Klemens Böhm, Christoph Koch, and Kian-Lee Tan (Eds.). IEEE Computer Society, 195–206.

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

[28] Chen Luo, Pinar Tözün, Yuanyuan Tian, Ronald Barber, Vijayshankar Raman, and Richard Sidle. 2019. Umzi: Unified Multi-Zone Indexing for Large-Scale HTAP. In *Advances in Database Technology - 22nd International Conference on Extending Database Technology, EDBT 2019, Lisbon, Portugal, March 26-29, 2019*, Melanie Herschel, Helena Galhardas, Berthold Reinwald, Irini Fundulaki, Carsten Binnig, and Zoi Kaoudi (Eds.). 1–12.

[29] Zhenghua Lyu, Huan Hubert Zhang, Gang Xiong, Gang Guo, Haozhou Wang, Jinbao Chen, Asim Praveen, Yu Yang, Xiaoming Gao, Alexandra Wang, et al. 2021. Greenplum: A Hybrid Database for Transactional and Analytical Workloads. In *Proceedings of the 2021 International Conference on Management of Data*. 2530–2542.

[30] Darko Makreshanski, Jana Giceva, Claude Barthels, and Gustavo Alonso. 2017. BatchDB: Efficient Isolated Execution of Hybrid OLTP+OLAP Workloads for Interactive Applications. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu (Eds.). ACM, 37–50.

[31] Aranyak Mehta, Amin Saberi, Umesh V. Vazirani, and Vijay V. Vazirani. 2007. AdWords and generalized online matching. *J. ACM* 54, 5 (2007), 22.

[32] C. Mohan, Bruce G. Lindsay, and Ron Obermarck. 1986. Transaction Management in the R* Distributed Database Management System. *ACM Trans. Database Syst.* 11, 4 (1986), 378–396.

[33] Thomas Neumann, Tobias Mühlbauer, and Alfons Kemper. 2015. Fast Serializable Multi-Version Concurrency Control for Main-Memory Database Systems. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, Timos K. Sellis, Susan B. Davidson, and Zachary G. Ives (Eds.). ACM, 677–689.

[34] Diego Ongaro and John K. Ousterhout. 2014. In Search of an Understandable Consensus Algorithm. In *2014 USENIX Annual Technical Conference, USENIX ATC '14, Philadelphia, PA, USA, June 19-20, 2014*, Garth Gibson and Nickolai Zeldovich (Eds.). USENIX Association, 305–319.

[35] Patrick O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O'Neil. 1996. The log-structured merge-tree (LSM-tree). *Acta Informatica* 33, 4 (1996), 351–385.

[36] Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd C. Mowry, Matthew Perron, Ian Quah, Siddharth Santurkar, Anthony Tomasic, Skye Toor, Dana Van Aken, Ziqi Wang, Yingjun Wu, Ran Xian, and Tieying Zhang. 2017. Self-Driving Database Management Systems. In *8th Biennial Conference on Innovative Data Systems Research, CIDR 2017, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*.

[37] Daniel Peng and Frank Dabek. 2010. Large-scale Incremental Processing Using Distributed Transactions and Notifications. In *9th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2010, October 4-6, 2010, Vancouver, BC, Canada, Proceedings*, Remzi H. Arpaci-Dusseau and Brad Chen (Eds.). USENIX Association, 251–264.

[38] Aunn Raza, Periklis Chrysogelos, Angelos Christos Anadiotis, and Anastasia Ailamaki. 2020. Adaptive HTAP through elastic resource scheduling. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2043–2054.

[39] Nikita Shamgunov. 2014. The MemSQL In-Memory Database System. In *Proceedings of the 2nd International Workshop on In Memory Data Management and Analytics, IMDM 2014, Hangzhou, China, September 1, 2014*, Justin J. Levandoski and Andrew Pavlo (Eds.).

[40] Jeff Shute, Mircea Oancea, Stephan Ellner, Ben Handy, Eric Rollins, Bart Samwel, Radek Vingralek, Chad Whipkey, Xin Chen, Beat Jegerlehner, Kyle Littlefield, and Phoenix Tong. 2012. F1: the fault-tolerant distributed RDBMS supporting google's ad business. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, K. Selçuk Candan, Yi Chen, Richard T. Snodgrass, Luis Gravano, and Ariel Fuxman (Eds.). ACM, 777–778.

[41] Jeff Shute, Radek Vingralek, Bart Samwel, Ben Handy, Chad Whipkey, Eric Rollins, Mircea Oancea, Kyle Littlefield, David Menestrina, Stephan Ellner, John Cieslewicz, Ian Rae, Traian Stancescu, and Himani Apte. 2013. F1: A Distributed SQL Database That Scales. *Proc. VLDB Endow.* 6, 11 (2013), 1068–1079.

[42] Rebecca Taft, Irfan Sharif, Andrei Matei, Nathan VanBenschoten, Jordan Lewis, Tobias Grieger, Kai Niemi, Andy Woods, Anne Birzin, Raphael Poss, et al. 2020. Cockroachdb: The resilient geo-distributed sql database. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1493–1509.

[43] Jiacheng Yang, Ian Rae, Jun Xu, Jeff Shute, Zhan Yuan, Kelvin Lau, Qiang Zeng, Xi Zhao, Jun Ma, Ziyang Chen, et al. 2020. F1 Lightning: HTAP as a Service. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3313–3325.