OceanBase Paetica: A Hybrid Shared-nothing/Shared-everything Database for Supporting Single Machine and Distributed Cluster

Zhifeng Yang*
OceanBase, Ant Group

Quanqing Xu*
OceanBase, Ant Group

Shanyan Gao
OceanBase, Ant Group

Chuanhui Yang†
OceanBase, Ant Group

Guoping Wang
OceanBase, Ant Group

Yuzhong Zhao
OceanBase, Ant Group

Fanyu Kong
OceanBase, Ant Group

Hao Liu
OceanBase, Ant Group

Wanhong Wang
OceanBase, Ant Group

Jinliang Xiao
OceanBase, Ant Group

OceanBaseLabs@service.oceanbase.com

ABSTRACT

In the ongoing evolution of the OceanBase database system, it is essential to enhance its adaptability to small-scale enterprises. The OceanBase database system has demonstrated its stability and effectiveness within the Ant Group and other commercial organizations, besides through the TPC-C and TPC-H tests. In this paper, we have designed a stand-alone and distributed integrated architecture named Paetica to address the overhead caused by the distributed components in the stand-alone mode, with respect to the OceanBase system. Paetica enables adaptive configuration of the database that allows OceanBase to support both serial and parallel executions in stand-alone and distributed scenarios, thus providing efficiency and economy. This design has been implemented in version 4.0 of the OceanBase system, and the experiments show that Paetica exhibits notable scalability and outperforms alternative stand-alone or distributed databases. Furthermore, it enables the transition of OceanBase from primarily serving large enterprises to truly catering to small and medium enterprises, by employing a single OceanBase database for the successive stages of enterprise or business development, without the requirement for migration. Our experiments confirm that Paetica has achieved linear scalability with the increasing CPU core number within the stand-alone mode. It also outperforms MySQL and Greenplum in the Sysbench and TPC-H evaluations.

PVLDB Reference Format:
Zhifeng Yang, Quanqing Xu, Shanyan Gao, Chuanhui Yang, Guoping Wang, Yuzhong Zhao, Fanyu Kong, Hao Liu, Wanhong Wang, and Jinliang Xiao. OceanBase Paetica: A Hybrid Shared-nothing/Shared-everything Database for Supporting Single Machine and Distributed Cluster. PVLDB, 16(12): 3728 - 3740, 2023.
doi:10.14778/3611540.3611560

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

Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:10.14778/3611540.3611560

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

1 INTRODUCTION

Initially, we have designed and developed the OceanBase system in version 0.5, which utilizes a divided storage and computing layer, thus resulting in significant scalability. For further improvement of the performance, we have then implemented version 3.0 of the system, which features enhancements that enable higher throughput and lower write latency, thus being capable of supporting the various business operations within the Ant Group and other commercial organizations. Furthermore, we made the system available as open-source, and shared the design and technology of the system within the open-source community. We have commercialized OceanBase, for applying its technology and capabilities to the businesses of numerous large, medium, and small enterprises since 2017. It is worth noting that OceanBase was the only distributed database that passed the TPC-C benchmark in 2020. However, the version 3.0 of OceanBase was not that well-suited to medium and small enterprises, probably owing to the overhead incurred by the log streams and partition bounds in small-scale machines, besides the additional overhead resulting from the interaction among the distributed components during the deployment.

The emergence of distributed databases has resolved the issue of horizontal scalability, but the stand-alone performance and SQL functionality are significantly inferior when compared to centralized databases, e.g., Oracle [23], MySQL [34], and PostgreSQL [37]. Many distributed databases have emerged during this process, and some of them are distributed storage systems that only support simple NoSQL functionality or limited SQL functionality, such as Amazon Dynamo [19]. Furthermore, certain distributed databases support both horizontal scalability and complete SQL functionality, often referred to as NewSQL, such as CockroachDB [42] and Google Spanner [9]. Citus [17] is an open source distributed PostgreSQL for data-intensive applications through the PostgreSQL extension APIs. However, their single-node performance is less than one-third of that of MySQL.

Therefore, the choice between the stand-alone and distributed system has become strenuous. Thus the typical decision-making is

based on the data volume; i.e., if the data volume is relatively small, a centralized database with complete functionality is chosen. Further, if the data volume is huge, a distributed database or distributed storage system is selected, thus sacrificing the functionality and stand-alone performance in order to address the issue by modifying the business or adding machines.

We further enhanced OceanBase [47] to version 4.0, expecting that it would better support small-scale enterprises. The system integrates several storage shards with a shared log stream and provides a high-availability service. Owing to the advancement in technology, contemporary machines have come to feature multiple cores, large amounts of DRAM, and high-speed storage devices. This highlights the importance of considering both horizontal and vertical scalability in the design of a distributed database system. Accordingly, we have developed Paetica[1] as a hybrid shared-nothing/shared-everything cloud database system capable of supporting both stand-alone and distributed integrated architecture. We will describe the concept of Paetica in detail with the following contributions.

- We propose Paetica, a stand-alone and distributed integrated architecture that is implemented in version 4.0 of the OceanBase system. Paetica features independent SQL, transaction, and storage engines in both the stand-alone and distributed systems, which enables the dynamic configuration switching by the user. The integrated architecture design allows OceanBase to operate efficiently without incurring the distributed interaction overhead in the stand-alone mode. Furthermore, while operating in the distributed mode, the system achieves high performance besides providing disaster tolerance.
- We have developed a stand-alone and distributed integrated SQL engine that is capable of processing SQL in diverse situations. The engine has been designed to execute SQL both in the serial and parallel manner to fully utilize the available CPU cores. Furthermore, in distributed execution scenarios, the engine is capable of parallelism across multiple machines that allows efficient processing of SQL commands.
- We have constructed a stand-alone and distributed integrated LSM-Tree storage engine that includes various compaction optimizations for both the stand-alone and distributed modes. These optimizations include the techniques such as incremental major compaction and staggered round-robin compaction, which intends to achieve a balance between the write performance and storage space utilization.
- For the stand-alone and distributed integrated transaction processing engine, we have proposed an optimized version of the 2-Phase Commit (2PC) protocol. This optimization intends to reduce the message processing and log volume, and subsequently decrease transaction latency. In the stand-alone mode, Paetica does not require the use of 2PC and instead utilizes a single log stream to process transactions without accessing the global time service (GTS). Consequently, the efficiency of the transaction engine is comparable to that of a stand-alone database.

[1]Paetica is OceanBase version 4.0.

We have conducted scalability experiments to demonstrate the linear scalability of Paetica. Our OLTP (Online Transaction Processing) experiments also demonstrate that Paetica exhibits high concurrency and low latency in both stand-alone and distributed modes. We have also compared OceanBase 4.0 with MySQL 8.0 in a separate experiment and found that OceanBase 4.0 performs better than MySQL 8.0 in small-scale and stand-alone situations. Furthermore, we have compared Paetica with OceanBase version 3.1 and Greenplum [31] 6.22.1 on the TPC-H [7] 100GB experiments, and it is observed that Paetica outperforms OceanBase 3.1 5x on average. Compared with Greenplum 6.22.1, Paetica demonstrates a superior performance across all queries.

The paper is organized as follows. §2 presents the OceanBase evolution. §3 provides an overview of the stand-alone and distributed architecture. §4 and §5 describe the integrated engine of SQL and transaction processing. We present the experiments in §6 to prove the efficacy and economy of Paetica. We present discussion including polymorphisim, dynamism and native multi-tenancy in §7, and we review the related work in §8. Finally, we give the conclusions in §9. OceanBase is an open-source project under Mulan Public License 2.0 [2] and the source code referenced in this paper is available on both gitee [3] and GitHub [4].

2 OCEANBASE EVOLUTION

In this section, we illustrate the evolution of OceanBase from version 0.5 to version 4.0.

2.1 OceanBase 0.5

OceanBase [4] has been developed since 2010. Figure 1 is the overall architecture diagram of OceanBase version 0.5. Concomitantly, OceanBase has been divided into two layers, viz., storage and computing. The upper layer is a service layer that provides SQL services statelessly, and the lower layer is a storage cluster composed of two kinds of servers: *ChunkServer* and *UpdateServer*. The *ChunkServer* is characterized by the capability for automatic partitioning and horizontal scalability of storage. The *UpdateServer* utilizes the Paxos protocol [28] to attain strong consistency and availability. However, the *UpdateServer* does not possess the capability for distributed transactions. Such an architecture can enable OceanBase to better support businesses similar to Taobao favorites [1]. Further, it has certain scalability, particularly a relatively strong scalability of reading, and the SQL layer is stateless and can be scaled freely.

Despite the advantages of this architecture, a major issue is that the *UpdateServer* node is a single-point write, multi-point read architecture, which is similar to PolarDB [13][29] and makes it difficult to expand when higher levels of concurrency become necessary. Furthermore, according to Figure 1, splitting the storage and SQL layers results in a high query delay. It is difficult to control the network jitter, and controlling the jitter of latency can be challenging under conditions of extremely high latency requirements.

2.2 OceanBase 1.0 ~ 3.0

To address the aforementioned issues, OceanBase has abandoned its previous architecture and developed the version 1.0 ~ 3.0, which is characterized by a fully peer-to-peer (P2P) structure as shown Figure 2. Each *OBServer* contains SQL, storage, and transaction

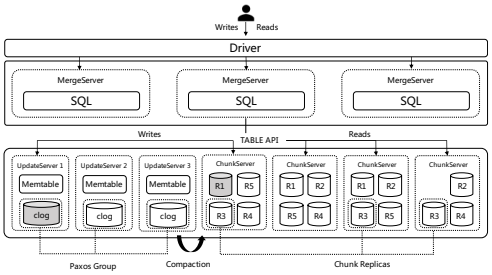

Figure 1: OceanBase 0.5

engines. All the servers are able to process SQL and handle transactions besides simultaneously storing data. As depicted in this diagram, the vertical direction represents the distributed and scalable layer, whereas the horizontal direction represents the replication layer. The horizontal direction provides high availability capabilities, whereas the vertical direction is achieved through the continuous addition of machines to enhance the overall scalability of the service. It employs several optimizations to achieve low latency. In the stand-alone mode, local execution plan, local transaction API and the elimination of network overhead in the read and write operations are the key features that contribute to the low latency. In the distributed mode, the use of duplicated tables, parallel execution engines, and multiple partition indexes are the key factors that enable the low latency performance.

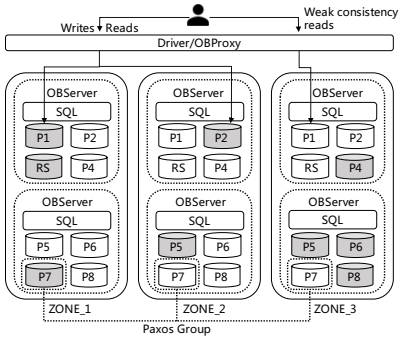

Figure 2: OceanBase 1.0~3.0

Prior the evolution to OceanBase 4.0, the original architecture had excellent scalability. Under this scalability, we performed the TPC-C benchmark [6][16] with OceanBase 3.0. OceanBase is the only distributed database that passed the TPC-C benchmark at that time. This also reflects that the OceanBase 3.0 architecture has exceptional adaptability in terms of horizontal scalability. According to Figure 2, from three nodes to 1,557 nodes OceanBase's ranking with a tpmC [6] index of 707 million, as the number of nodes increases, the entire tpmC indicator has appreciable linear scalability. While performing the TPC-C evaluation, OceanBase utilizes a large cluster composed of 1,557 machines, and within eight hours of pressure testing, it has the ability to process twenty million transactions per second. This result shows that the previous architecture can support excellent scalability, and almost this scalability and concurrent processing capabilities can satisfy the requirements of most current online service systems, globally. Furthermore, by employing a single zone deployment over a distributed storage system, OceanBase passed TPC-H benchmark test and gained over 15 million QphH@30,000GB [5] in May 2021.

2.3 OceanBase 4.0

However, with the iteration of business requirements, we developed the OceanBase 4.0 architecture, as shown in Figure 3. OceanBase 4.0 has the following features.

- *More partitions*: The architecture of OceanBase 4.0 reduces the cost of partition maintenance. Furthermore, the optimization of memory should not be underestimated. In the previous versions, we have maintained one metadata for every 2MB macroblock, and the ratio of metadata to data is approximately 1:1000. Therefore, for models with larger disks, the overhead of metadata would also increase significantly. In this iteration, we have made the storage memory overhead as on-demand loading, thus maintaining only the root node (very small) in memory. When the user needs to access the metadata, the leaf node and data node are then loaded. This method reduces the overhead of the resident memory and brings the feature of small-size memory optimization.
- *More DDL support*: In OceanBase 4.0, Data Definition Language (DDL) allows the users to easily modify partitions and alter primary keys, thereby facilitating existing database usage practices. The implementation of DDL is relatively straightforward. Initially, a hidden table is created, and the pre-DDL transaction is initiated with attained snapshot point. The original table is then locked for reading and writing. Subsequently, a Data Manipulation Language (DML) statement is employed to supplement the data in the hidden table. Finally, the original table is renamed. The process involves three key technologies, viz., 1) table locking, which prevents write operations while the modifications are being made, 2) Partitioned DML (PDML), which is used to accelerate the queries and streamline the code, and 3) direct insertion, which allows for writing directly to static data, thus avoiding memory overload and providing faster speeds.
- *Less resource consumption*: In OceanBase 4.0, the production specifications have been decreased from 16C/64G to 4C/16G (CPU/memory), thus improving the user efficiency. We have primarily optimized the following aspects, viz., 1) thread stack optimization to reduce thread-local variables and use of *SmartVar* to decrease the stack variables, 2) improvement of metadata overhead from per-partition to per-logstream storage, thus allowing metadata to be loaded on demand, and 3) improved stability through the default activation of input restriction, thus enabling greater stability under 4C/16G.
- *Tenant isolation*: We have optimized the tenant coupling logic, primarily in the following three aspects, viz., 1) tenant-level merging, whereby the default merging behavior is triggered by tenants instead of cluster-wide, 2) tenant-level metadata, in which the metadata is adjusted from the system tenant to the user tenant and the *TableId* and *TenantId* are decoupled, and 3) tenant I/O isolation, with *Clog* (Commit log) files being split into tenants and *SSTable* supporting the tenant-level I/O restriction.

For medium and small-scale enterprises, the core change in the OceanBase 4.0 architecture is the introduction of dynamic log streams. Originally, we have equated the granularity of transaction expansion and storage expansion together. However, if the storage is divided into several shards, the transaction processing and high availability capabilities are also based on such shards. We have decoupled these two concepts in OceanBase 4.0, hence several storage shards will share a transaction log stream and the high-availability service corresponding to this log stream.

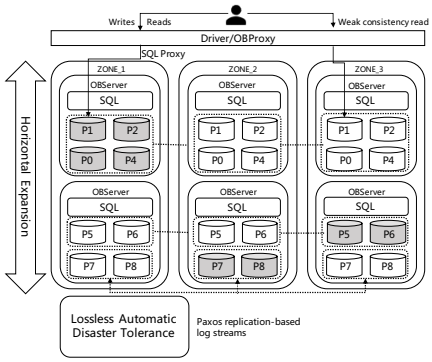

Figure 3: OceanBase 4.0

The core idea behind this change is that we hope to support certain applications on a smaller scale. For example, in large-scale applications such as Ant Group, the OceanBase 3.0 architecture will not encounter bottleneck problems, but as OceanBase becomes common, especially for various small and medium-sized enterprises, OceanBase 3.0 is difficult to be applied to such users. However, if the number of log streams and the number of partitions are bound together, in multiple scenarios, it cannot be adapted to support small and medium-sized enterprises. In other words, if there are too many log streams, the overhead will appear to be greater at a smaller scale.

3 STAND-ALONE AND DISTRIBUTED INTEGRATED DATABASE ARCHITECTURE

The architecture of OceanBase 4.0 can use the OceanBase database in a distributed manner besides the stand-alone mode (similar to MySQL [34]), for using OceanBase. First, if OceanBase is deployed on a stand-alone basis, or as a single-container tenant in an OceanBase cluster, it can provide the same efficiency and performance while using a stand-alone database. Second, if the distributed mode is employed, there is no need to pay additional costs for its introduction at the tenant level, transaction level, and single SQL execution level. For this case, we need to employ a stand-alone and distributed integrated architecture in the design of the SQL layer, storage layer, and transaction layer to satisfy the requirements of diverse situations and give consideration of all levels.

3.1 Shared-everything or shared-nothing?

The debate surrounding the database development in regard to shared-nothing [27][41] and shared-everything [12][21] architectures has been long-standing. Figure 4 elucidates that a physical cluster is neither entirely shared-nothing nor entirely shared-everything. A single node on each machine is a multi-processor structure with a high number of CPUs and strong storage and computing capabilities. These capabilities should not be disregarded, and should be an area of consideration while performing scale-ups for the stand-alone and centralized databases. Distributed databases should consider both the horizontal and vertical expansions.

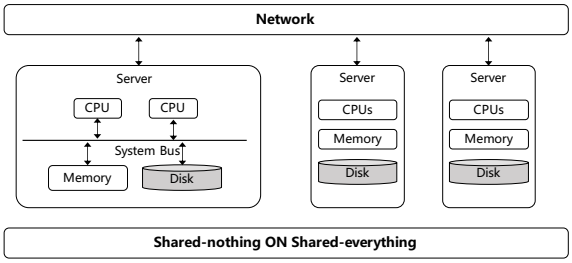

Figure 4: Shared-nothing on shared-everything

If we want to take advantage of the hardware vertical expansion capabilities, we should consider the role of database. We can imagine such a distributed database. The left part of Figure 5 is the actual architecture diagram of several distributed databases, such as TiDB [25]. It is similar to the structure of OceanBase 0.5, with dedicated computing nodes and storage nodes. The computing node forms one layer, and the storage node is another layer. These two layers are abstracted separately. Among them, the module that handles the global affairs through GTM/TSO (global transaction manager, timestamp oracle) needs to interact with this module while performing the multi-machine interaction.

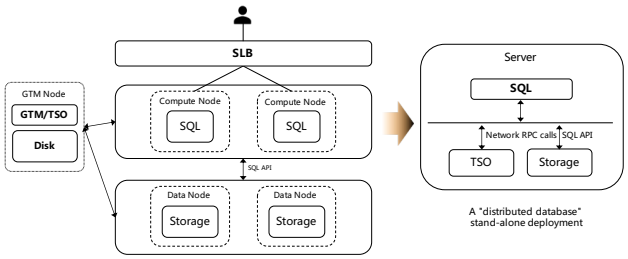

Figure 5: From a Distributed Database to a Stand-alone One

As shown in right part of Figure 5, if this distributed database is plugged into a node in a very simple way and it runs as a stand-alone machine, we identify the biggest problem. The interaction between these components is very expensive when working in a single machine. This overhead is essentially an additional unnecessary one for a single-machine database like MySQL, hence obviously such a simple method is difficult for comparison with a single-machine database.

3.2 Stand-alone and distributed integration: taking into account both stand-alone and distributed scenarios

The system architecture of stand-alone and distributed integrated databases is illustrated in Figure 6. With a stand-alone machine, there must be a three-tier system that consists of an independent SQL engine, transaction engine, and storage system respectively. For a comparison, a distributed database combines the distributed

SQL engine, distributed transaction engine, and distributed storage engine, thus creating three separate layers. The combination of the three engines of the stand-alone and distributed systems is desirable, while utilizing the same code base and allowing for dynamic interaction and alteration. When converting between the stand-alone and distributed modes, data migration is required and performed online. In order to reduce the impact on the business, the migration is divided into two parts. For the leader replica, the first step is to switch the leader so that the role of the data replica to be migrated becomes a follower, so as to ensure that there is no read and write on it. The second step is to perform data migration on the follower replica.

A stand-alone and distributed integrated architecture requires both the scalability of a distributed system, and the functionality and stand-alone performance of a centralized database. The basic requirements for a database include the ACID (Atomicity, Consistency, Isolation, Durability) properties of transactions. The challenge against the distributed databases is the mechanism to ensure the ACID properties of transactions in exceptional scenarios. The core issue is the implementation of data recovery based on redo logs and to ensure the atomicity of distributed transactions in exceptional scenarios.

The stand-alone database needs to provide various capabilities to maximize its scalability. At the SQL layer, this requires the parallel execution capabilities, whereas at the transaction layer, there must be support for scalable core technologies such as multi-version concurrency control (MVCC) [11][14]. Furthermore, the utilization of technologies such as group commit [24] is essential for enabling concurrent execution of multiple transactions on a single machine. Finally, at the storage layer, parallel I/O on a single machine must be supported to make full use of multiple disks and storage bandwidth.

Increasing the distributed scalability is not the only factor that must be taken into consideration when deploying a stand-alone database in a distributed cluster. To achieve a satisfactory performance, the stand-alone efficiency must be optimized and the database must be capable of efficiently executing serial commands. Furthermore, the distributed transactions must be capable of adaptive optimization in order to enable successful stand-alone transactions.

OceanBase has implemented a pioneering technology, the stand-alone and distributed integrated LSM-Tree [36] storage engine, that can be employed both in the stand-alone and distributed scenarios. In the former case, merge operations can be conducted without interference between front and back, whereas a distributed strategy such as staggered round-robin compaction can be employed in distributed deployments. Both the stand-alone and distributed scenarios are taken into account for the SQL, transaction, and storage layers in its design, with no extra overhead occurring in either of the cases, thus fulfilling OceanBase's design goal.

Thus, we require two basic properties: First, while working on a single machine, OceanBase has no additional components, i.e., a single process can work without the complex interactions between the redundant processes. Under the single-process and multi-thread model, each component can actually complete the interaction using simple function calls, which is a very critical point. Second, there are vertical interactions between all components in Figure 6, which are SQL, transaction, and storage layers. If it is in a stand-alone machine, we employ the method of function call to do the interaction; if it is between nodes, and necessary, then there must be an interaction between nodes, and then the need to use RPC to interact.

In accordance with the aforementioned shared-nothing and shared-everything combination architecture, function calls are employed within each *OBServer* node to directly enable communication among the SQL, transaction and storage engines, similar to a stand-alone database. To enable the communication among multiple nodes, the layering structure is relaxed to enable an optimal, suitable, and efficient solution. For instance, when the SQL from one node sends a message to the SQL layer on another node, the latter is granted access to its storage nodes. The best choice is contingent upon the desired outcome and can vary based on the load. OceanBase offers the capability to access remote storage directly. The concept is also applied at the transaction processing layer to avoid unnecessary interaction.

4 STAND-ALONE AND DISTRIBUTED INTEGRATED SQL ENGINE

The OceanBase execution engine has to deal with multiple situations according to the expectation that it can be adaptive and optimal for each of them. On a higher level, each SQL execution has two modes, viz., serial or parallel executions. Figure 7 illustrates the stand-alone and distributed adaptive execution engine. Serial execution comprises of local, remote, and distributed execution. For the parallel execution, there are parallel queries and DML operations that improve the performance. The detailed descriptions are given in the following.

4.1 Serial execution, parallel within a single machine, and distributed parallel

During the serial execution, if the accessed data is on the local machine, then there is no difference between the remote processing and the processing of native or stand-alone SQL. For the data located on another node, there are two ways to access it, viz., remote data acquisition or remote execution [39]. In the former one, the data is pulled to the local machine, and in the latter, the transaction processing and storage access processing is forwarded to the other node and the entire transaction is returned. If a single SQL accesses data from multiple nodes, then the calculation can be pushed to each node with the intention of achieving the effect of serial execution with a minimal overhead in the case of a single machine. This also allows for the distributed execution capabilities. Parallel query is supported and can be either local or distributed, with parallel DML write operations also supported.

Serial execution plans are an efficient method for the small-scale businesses to process the data without context switching or remote data access. However, introducing parallel capabilities into a stand-alone OceanBase allows for increased SQL processing capacity and faster response times, if larger amounts of data need to be accessed. Although several open-source stand-alone databases may not support this capability, it is possible to increase the parallelism using multi-core servers. Distributed execution plans can be employed to further increase the scale of data processing, thus allowing for parallelism on multiple machines and the potential to surpass the limits of single-machine CPUs, up to thousands of cores.

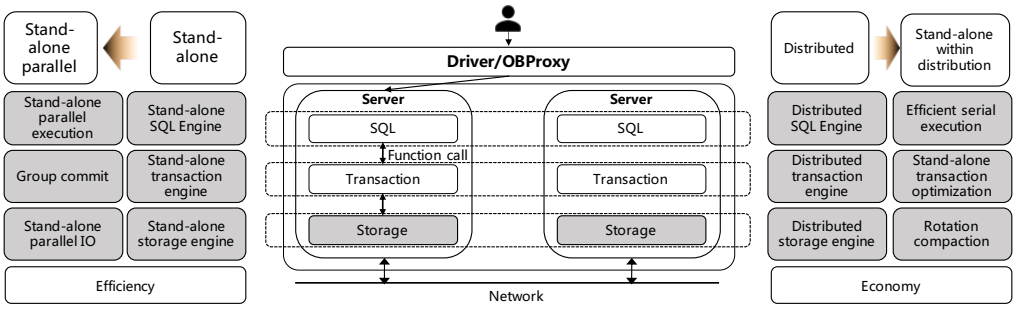

Figure 6: Stand-alone and Distributed Integrated Database

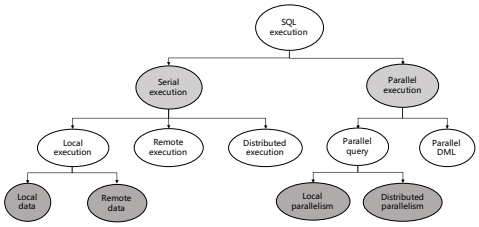

Figure 7: Stand-alone and Distributed Adaptive SQL Engine

4.2 Serial execution: DAS execution and distributed execution

There are two execution modes for the serial execution: direct access store (DAS) execution [44] and distributed execution. One of the methods of DAS [35] execution is to pull data. If the data is located remotely, and it is a simple point query or index access back to the table query, application of this method consumes the least of resources. We will pull this data to the local, along with its execution. There is no difference in form between the plan and the local execution plan, and this action will be performed automatically in the executor. However, it is preferable to push down the calculation than to pull the data, hence we also support the distributed execution. This distributed execution does not increase the consumption of additional resources. We will ensure that its parallelism and the previous DAS execution are identical.

For certain specific queries or large-scale scans, we will dynamically and adaptively select the two, and evaluate the one with the better performance based on the cost. OceanBase utilizes an execution engine to facilitate the Hybrid Transaction/Analytical Processing (HTAP) workloads. With respect to traditional Online Transaction Processing (OLTP), OceanBase employs a pull-based data access methodology and a bottom-up computational execution approach for the traditional Online Analytics Processing (OLAP).

The parallel execution framework can adaptively process the parallelism in a single machine and distributed parallelism, since they are the same framework. All the parallel processing workers can form multiple threads on the machine or threads on many nodes. We have a layer of adaptive data transmission layer in the distributed execution framework. For the parallelism in a single machine, the transmission layer will automatically convert the data interaction between the threads into memory copy. Accordingly, the two different scenarios are completely abstracted by the data transmission layer. The parallel execution engine has no difference with respect to implementation of the scheduling layer for parallelism in a single machine and distributed parallelism.

5 STAND-ALONE AND DISTRIBUTED INTEGRATED TRANSACTION PROCESSING ENGINE

Since it is more difficult to achieve scalability in the transaction processing, we search the reason for considering both the stand-alone and distributed during the transaction processing. In addition, we proposed and implemented a fully distributed deadlock detection and resolution algorithm named LCL (Lock Chain Length) [46], which is dynamically scalable and applied to OceanBase Paetica.

5.1 Traditional 2PC vs. OceanBase 2PC

Figure 8 is the traditional distributed transaction processing process [8][10][22][38], which is categorized into the opening phase of the transaction and the commit phase of the transaction, and Figure 9 corresponds to the transaction processing process of OceanBase. Previous works [30, 32, 48] attempt to reduce the overhead of 2PC and synchronous replication. OceanBase transaction commit protocol proposes a new approach, termed the two-phase commit protocol for the participants and coordinators.

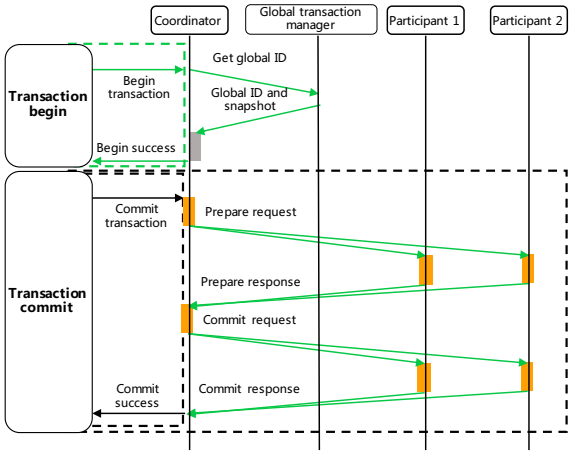

Figure 8: Traditional 2PC

Compared with traditional 2PC in Figure 8, OceanBase 2PC in Figure 9 has significantly less message processing and log volume than traditional 2PC from the instant the transaction is submitted to the successful commit, which gives it a great advantage in latency. This advantage will also support the transaction-handling capability of OceanBase in the distributed scenarios, unlike several

other distributed databases that incur a large overhead. There are two scenarios to access GTS: 1) Statement snapshot acquisition; 2) Acquisition of transaction submission version number. For stand-alone transactions, 1) there is no need to visit GTS, which has been optimized in OceanBase 4.0. 2) The transaction submission version number will obtain GTS to meet the external consistency, though this acquisition is only an interface call, and will not actually send RPC, hence the efficiency is relatively high.

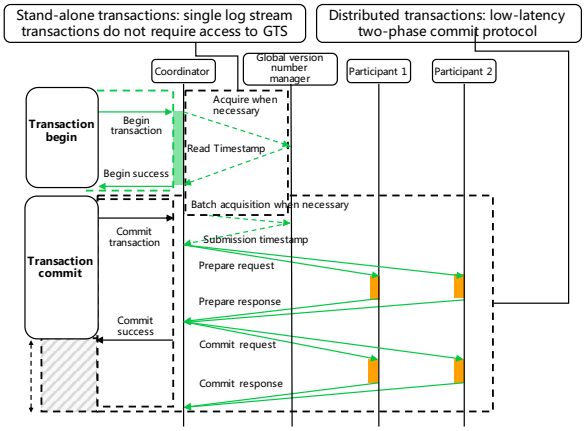

Figure 9: OceanBase 2PC

5.2 Log stream and version number manager

If a single transaction involves only a single log stream, generally speaking, and if the amount of business data is within the tolerable granularity of load balancing, the log stream does not need to be particularly large. There is only one log stream in a high probability, when deploying in a stand-alone mode, and any transaction in a log stream does not need to go through the two-phase commit.

OceanBase employes the transaction version number to identify the order of committed transactions, and determines the visibility of multi-version data under the snapshot isolation level. OceanBase's transaction version number service takes into account both stand-alone and distributed performance. In a distributed scenario, it obtains the transaction version number through the global timestamp service. In a stand-alone deployment scenario, the global timestamp service and transaction context must be on one node. It can be configured to use the local timestamp service to obtain the transaction version number, and obtain the version number through a function call, thereby avoiding the overhead of context switching caused by RPC.

The transactions in a distributed database system are not independent of one another, and they can involve the synchronization of single or multiple log streams. In OceanBase, we has developed an optimization for the single-log stream transactions, which enables the transactions to fetch a local version number without compromising the global consistency. If all the transactions and workloads do not involve distributed transactions, the entire transaction processing is analogous to the same procedures in a stand-alone database.

6 PERFORMANCE EVALUATION

In the evaluation, we perform experiments with different database systems and configurations.

6.1 Experimental Configuration

We use the following databases: OceanBase 3.1, OceanBase 4.0, MySQL 8.0, Greenplum 6.22.1, and RDS 8.0 in the performance evaluation. Firstly, the single-node experiments in §6.2, §6.3, and §6.4 are performed on a two-way Intel Xeon Platinum 8163 CPU @ 2.50 GHz server. Secondly, the single-node experiments in §6.5 and §6.6 are done on 32 cores Intel(R) Xeon(R) Platinum 8396B CPU, 128GB DRAM, and three 500GB ESSD PL1. Thirdly, the single-node experiments in §6.7 are performed on ecs.r6.xlarge, ecs.r6.2xlarge, ecs.r6.4xlarge, and ecs.r6.8xlarge instances, respectively in Alibaba Cloud (AliCloud).

6.2 Stand-alone performance scalability

The following experiment, as shown in Figure 10, have been run on a two-way Intel Xeon Platinum 8163 CPU @ 2.50 GHz server. The Sysbench [26] dataset is 1,000,000, with a stress test of 1,500 concurrency and 30 tables. OceanBase expands the experiment with the enhancement of hardware performance, from 4 cores to 64 cores, and can achieve basic linear expansion from 9×10^4, 1.8×10^5, 3.7×10^5, 6.9×10^5, 1.2×10^6 within the scale of 64 cores.

In the *point select* and *read-only* scenarios, an increase in the number of CPU cores by one-fold resulted in a corresponding one-time increase in the performance when the number of the server's CPU cores was equal to or less than 32 vCPU. However, when the number of the server's CPU cores exceeds 32 vCPU, a lack of physical cores implies that the performance does not scale up linearly. Furthermore, an upgrade from 32 vCPU to 64 vCPU resulted in a 50% increase in the performance of the two scenarios. In the three pure write scenarios of *insert*, *update*, and *write only*, when the number of the server's CPU cores is below 32 vCPU, a one-fold increase in the number of CPU cores resulted in an approximate increase in performance by 1.2 times, thus exhibiting full linearity. Conversely, when the number of the server's CPU cores exceeds 32 vCPU, the performance change resembled that of the read scenario and no longer displayed a linear increase. Thus, we have observed that OceanBase can provide linear expansion.

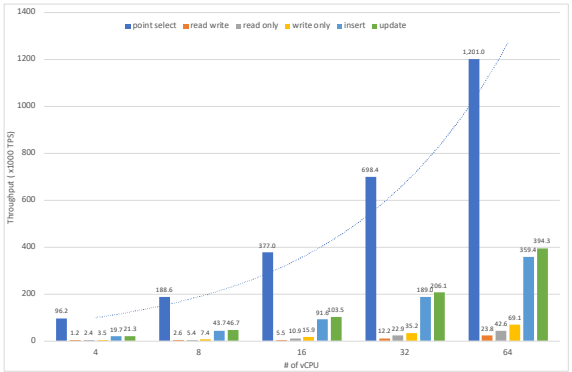

Figure 10: OceanBase Stand-alone Scalability

6.3 High concurrency and low latency OLTP

Originally, a stand-alone database or a centralized database can support a business with one machine. Following the introduction of distributed databases, the scalability is excellent. If we intend to

support the same amount of concurrency in a three-replica-based distributed database system, we may have to employ three nodes to provide services together to achieve the same performance and efficiency as a single machine. Figure 11 illustrates high concurrency and low latency OLTP scenarios.

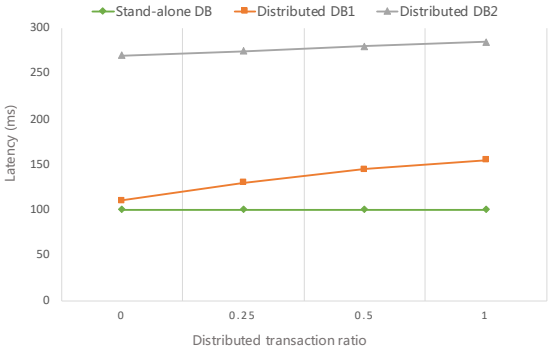

Figure 11: Low latency OLTP

This is obviously not the distributed database that users want. The key issue is with respect to the performance, we should not only look at the increase in throughput along with the speed of a single SQL or a single transaction that can be processed from the perspective of the application. Although we know that the delay is increasing with the increasing proportion of distributed transactions, from the perspective of a single part of transaction processing, there are two key points that should be satisfied, viz., 1) the delay needs to be low enough even if it is composed of completely distributed transactions, 2) the system should not produce extra latency when there are not so many distributed transactions.

The optimization of the SQL, storage, and transaction layers for the stand-alone granularity transactions and SQL is a priority, even in a distributed environment. The users are denied a substantial degree of control after these components are layered independently. Therefore, providing methods to reduce the amount of RPC requests by placing the data on a node would be beneficial. It is important that the users and DBAs have a control capability with extreme performance requirements. OceanBase has taken advantage of the control offered by C++, which allows it to precisely manage memory and reduce latency. Furthermore, providing the operators with the ability to partition the data is also essential, as a table group allows the users to avoid distributed transaction overheads. A table group is not a physical object instead of a logical concept that represents a group or a set of tables. Tables that belong to such a group must meet some constraints. All tables must have the same locality (including replica type, number, and location), primary zone (leader location and priority), and the same partitioning method.

6.4 OceanBase 4.0 vs. MySQL 8.0

According to Figure 12 and Figure 13, if there is no distributed transaction scenario and they are deployed in a distributed situation, the effect of a single machine is to be determined. OceanBase 4.0 and MySQL 8.0 Enterprise Edition have been compared under the same hardware condition. According to the data, OceanBase 4.0 performs better than MySQL under the same hardware environment, and even for small-scale scenarios, we can use the OceanBase distributed database as a stand-alone database with confidence. Figure 12 compares the TPS experimental results of OceanBase 4.0 and MySQL 8.0, which shows the performance of the two databases in a high-concurrency environment. In the *insert* and *update* scenarios, the performance of OceanBase 4.0 is 1.7× and 2.0× that of MySQL 8.0, respectively. In addition, in the *read-only* scenario, OceanBase 4.0 also has twice in the performance improvement.

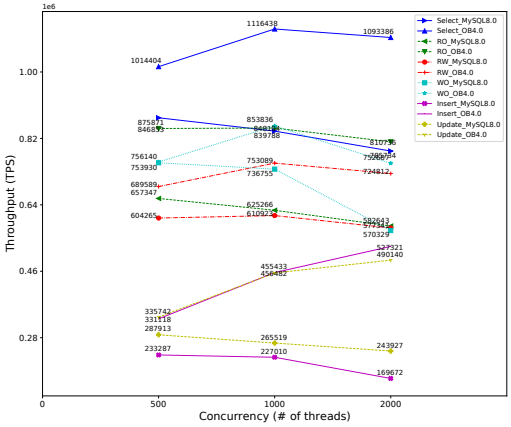

Figure 12: OceanBase 4.0 vs. MySQL 8.0 in throughput

The latency of transaction processing is the most concerned indicator for users, as shown in Figure 13. Especially in the *insert* and *update* scenarios, OceanBase 4.0 shortens the latency by almost 50% compared to MySQL 8.0. This implies that OceanBase 4.0 can perform better in scenarios where data is frequently updated, significantly improving the user experience. The two results illustrated in Figure 12 and Figure 13 show that even in the single-machine environment, the comprehensive performance of OceanBase 4.0 in handling non-distributed transactions is higher than that of MySQL 8.0. Concomitantly, these results fully verify the high concurrency and low latency features of Paetica in OLTP transactions.

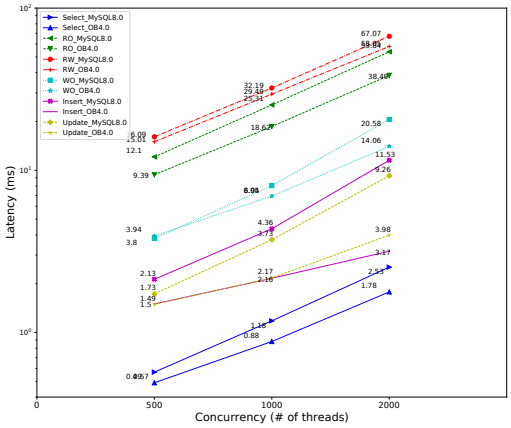

Figure 13: OceanBase 4.0 vs. MySQL 8.0 in latency

There is an obvious problem with the traditional way of using MySQL master and backup databases, i.e., the data loss during disaster recovery. This problem cannot be solved, but OceanBase must first be able to make three machines highly available. From 4.0 onwards, we will officially recommend a deployment method,

such that there is an OceanBase node in each zone, which is called a single-machine three-copy, without much additional overhead. The *OBProxy* layer provides the capability of continuous business connections. If we do not need such a capability, we can achieve better high availability capabilities than MySQL without deploying any proxy like MySQL.

For example, at the beginning, it was completely deployed on a single machine, and it was enough for the business to do some prototype experiments. With the expansion of business scale, the requirements for computing and storage are getting higher and higher. In the process, this node can be expanded vertically. It used to be 4 cores, though currently it has 16 cores. We can buy it on ECS (Elastic Compute Service), if on cloud, by directly applying for the specification upgrades.

In this manner, OceanBase is able to leverage the improved CPUs to provide better performance. Moreover, if high availability is not necessary and a three-node cost is not desired, then a primary-standby database would be sufficient, and OceanBase can also provide this capacity. This kind of high availability is commensurate with the traditional databases and the primary-standby database also remains an option.

6.5 Comparison between OceanBase 3.1 and 4.0

The results presented in Table 1 indicate the execution time of SQL queries on the TPC-H 100GB experiment. As can be observed, the OceanBase community edition 4.0 demonstrates a reduced execution time as compared to the previous version, OceanBase community edition 3.1, with an average improvement of 5x. The greatest performance increase was observed in query *Q6*, with a 3086.96% improvement. This query assesses the potential revenue gain from the elimination of specific company-wide discounts within a specified percentage range for a given year. The enhanced DDL support and performance optimization implemented in OceanBase 4.0 allows for more efficient processing of such complex queries.

The data presented in Table 1 indicates that the proposed model, Paetica, is capable of effectively processing the queries in a comprehensive environment. This suggests that Paetica is a promising approach for addressing this particular problem.

6.6 OceanBase 4.0 vs. Greenplum 6.22.1

Figure 14 presented in this study illustrates the performance comparison of OceanBase 4.0 against Greenplum 6.22.1 [31] on the TPC-H 100GB benchmark, with the system configuration of 32 cores Intel(R) Xeon(R) Platinum 8396B CPU, 128GB DRAM, and three 500GB ESSD PL1 on each machine. The x-axis depicts the set of queries employed in the TPC-H benchmark, whereas the y-axis represents the time required for executing each query. As depicted in the figure, OceanBase consistently demonstrates a superior performance in comparison to Greenplum across all queries, with the most significant improvement observed in query *Q17*. Particularly, OceanBase achieved an execution time of 0.61s, whereas Greenplum required 28.29s. Query *Q17* evaluates the potential loss of average yearly revenue resulting from the discontinuation of orders for small quantities of specific parts and has the potential to reduce the overhead expenses by focusing on larger shipments. The newly designed SQL and storage engines in OceanBase 4.0

Table 1: Performance Comparison between OceanBase (OB) Community Edition (CE) 4.0 and 3.1

Query	OB CE 4.0 (s)	OB CE 3.1 (s)	Improvement
Q1	2.34	14.04	500.00%
Q2	0.14	1.12	700.00%
Q3	0.72	13.57	1784.72%
Q4	0.56	2.51	348.21%
Q5	2.25	12.31	447.11%
Q6	0.23	7.33	3086.96%
Q7	1.52	10.38	582.89%
Q8	0.70	11.42	1531.43%
Q9	5.22	30.99	493.68%
Q10	1.24	6.84	451.61%
Q11	0.23	1.22	430.43%
Q12	1.62	8.64	433.33%
Q13	2.41	7.59	214.94%
Q14	0.36	1.51	319.44%
Q15	0.79	3.01	281.01%
Q16	0.66	2.66	303.03%
Q17	0.63	8.60	1265.08%
Q18	0.93	7.88	747.31%
Q19	0.78	9.36	1100.00%
Q20	1.17	10.95	835.90%
Q21	2.42	12.27	407.02%
Q22	1.24	4.05	226.61%
Total	28.16	188.25	568.50%

enable the efficient processing of this complex query. Furthermore, it is worth mentioning that in query *Q11*, they have the minimum execution time among all queries. This feature owes to *Q11* in only finding the most important subset of suppliers' stock in a given notion, which is not a complex query. This observation indicates that OceanBase 4.0 is capable of efficiently processing an extensive range of queries, including both the complex and simple ones.

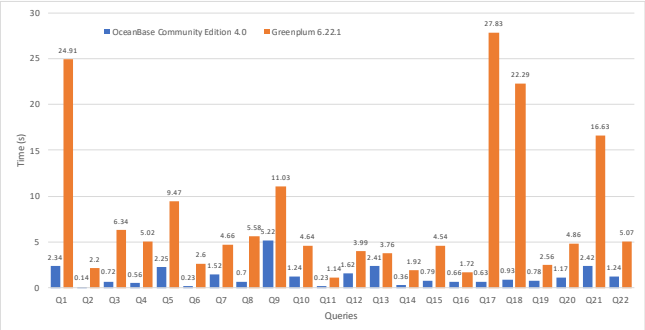

Figure 14: TPC-H 100GB Performance Results

The data presented in Figure 14 demonstrates that OceanBase is capable of effectively processing queries and achieving high performance on the TPC-H benchmark. This suggests that OceanBase 4.0 is a potential solution for addressing the problem of data management and query processing.

6.7 OceanBase 4.0 vs. RDS 8.0

Alibaba Cloud Database RDS (Relational Database Service) offers high availability, reliability, security, and scalability of the hosted database services, with performance equivalent to the commercial databases and a price below that of ECS self-built databases and self-purchased server-built databases, thereby saving a considerable amount of deployment and maintenance costs. Figure 15 and Figure 16 demonstrate the scalability of the small-scale high-availability deployment from the perspectives of throughput and latency, respectively. In Sysbench, there are six operations, which are simplified as follows: *point select* (*ps*), *read only* (*ro*), *write only* (*wo*), *read write* (*rw*), *insert* (*is*), *update* (*up*).

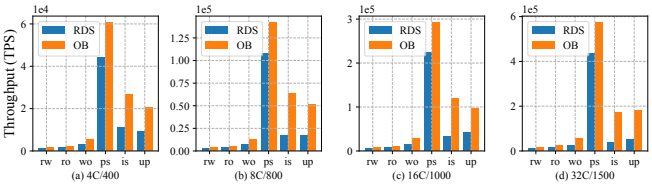

Figure 15: OceanBase 4.0 vs. RDS 8.0 in throughput

As shown in Figure 15, 1) for the *insert* and *update* scenarios, OceanBase's throughput is over 2x that of RDS 8.0, and 2) for the other scenarios, OceanBase 4.0 is 20%-50% higher than RDS 8.0 in throughput, thus highlighting the efficiency of Paetica. As illustrated in Figure 15(b), experiments conducted on an ecs.r6.2xlarge instance with 8 vCPUs, 64GB of DRAM and a cloud disk PL1 shows that Paetica has outperformed RDS 8.0 across all workloads, even with the insertion throughput per second of 1675.48% exceeding that of RDS 8.0. Both Figure 15 and Figure 16 demonstrate that in each scenario the throughput and latency of OceanBase 4.0 exhibit the expected linear expansion with the increasing number of CPU cores, whereas the linear expansion of RDS is relatively poor in the writing scenario.

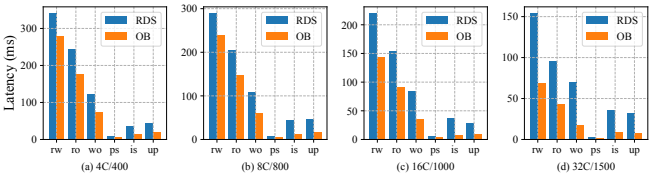

Figure 16: OceanBase 4.0 vs. RDS 8.0 in latency

7 DISCUSSION

7.1 Polymorphism and dynamism

OceanBase has the capacity to structurally arrange the data in a variety of polymorphic forms. It can be deployed in a stand-alone form for the small-scale enterprises, with the option to establish a primary-backup database or multiple replicas. If further availability is desired, a three-replica configuration can be applied. It is possible to progress to a fully-distributed OceanBase database in tandem with the business growth or company expansion.

The conversion or expansion between the varied polymorphic forms is achievable with the reversals of each arrow as depicted in Figure 17, with the dynamic efficacy of each arrow transformation being retained without the need to substitute the OceanBase binary. The system remains completely functional in all three forms, thus providing dynamic transformation. This characteristic enhances the flexibility and reduces the cost for the user.

7.2 Native multi-tenancy

We have considered the issue of discomfort for the users who are transitioning from MySQL to OceanBase, during the design process of the OceanBase stand-alone and distributed integrated architecture. Particularly, the implementation of multiple tenants requiring individual tenant creation upon entry have been considered. Following the thorough discussions with the users and customers, it has been determined that the multi-tenant feature should be maintained, particularly for the instances of private deployment and public cloud deployment, which coincides with the points in certain cloud database services [20, 43]. For the private deployment scenarios, as shown in Table 2, small businesses may not wish to undergo the complex process of setting up servers. However, owing to the current availability of hardware featuring numerous cores and minimal specifications, OceanBase's native multi-tenant capability allows for the division of the stand-alone into multiple databases to be employed without the necessity of any further container management systems, provided no additional infrastructure is required.

OceanBase can be easily vertically scaled for the large-scale deployments. In the distributed scenarios, it has the added capability of being able to scale in both the vertical and horizontal directions on a per-tenant basis. One tenant consists of 4 cores and 3 containers, whereas another machine may choose to operate in the stand-alone mode and can be allocated 16 cores directly. Alternatively, it may opt not to be distributed. In such cases, the multi-tenancy offers a level of granularity that is easily managed.

Deployment on the public cloud or within a large OceanBase cluster within an enterprise renders it even more valuable. Even for the small-scale deployments, such as an OceanBase server utilizing 16 cores, though with the need for a smaller tenant, such as a 0.5 core tenant, OceanBase is capable of supporting 0.5 core tenants internally within the database. This allows the users to precisely control costs and enables the vertical scaling within the tenants on the cloud or load balancing among the tenants. Both of these dimensions can be made elastic and freely scalable. Consequently, we have maintained all of OceanBase's functions and the ability to have multiple tenants in different forms.

8 RELATED WORK

8.1 Stand-alone and distributed integrated system

A plethora of stand-alone and distributed integrated database systems are available, which facilitates the users with the ability to deploy the databases in an adaptive manner. In this study, we will examine the current state-of-the-art systems within the industry and classify their characteristics, as outlined in Table 3.

Apache Cassandra [15] stores the data on multiple nodes in a cluster and replicates the data across the cluster to ensure high availability. It has been designed to be highly fault-tolerant, with the ability to handle failures of individual nodes without affecting the

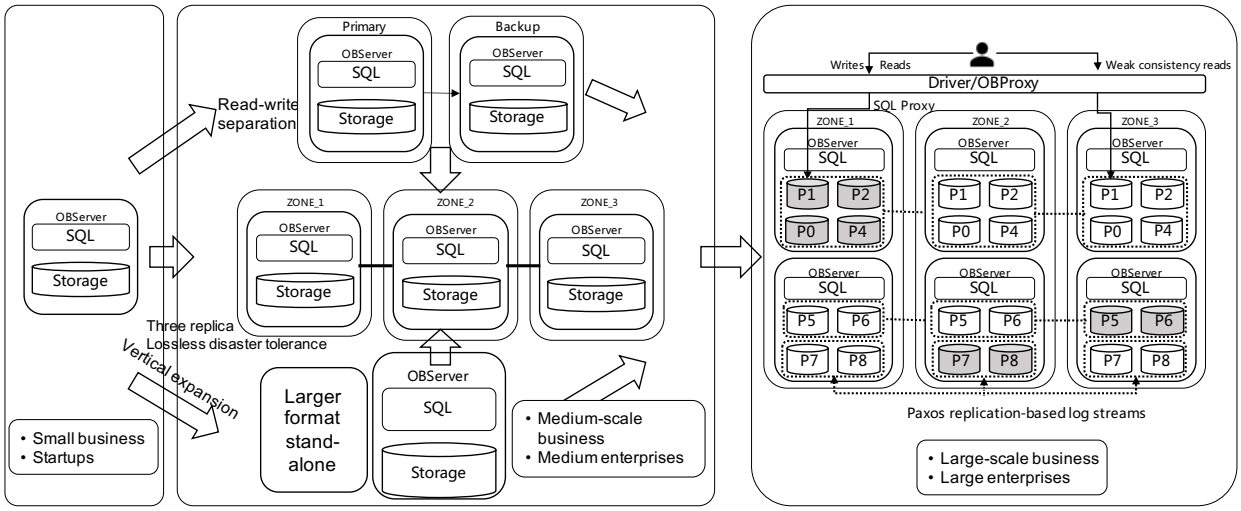

Figure 17: OceanBase Multiform On-demand Scalability

Table 2: Characteristics of different deployment

	Stand-alone small size	Stand-alone large size	Distributed
Privatization Deployment	Great value No need for container management	Great value Easy to scale-up	Great value Resiliency and load balancing
Public Cloud Deployment	Selling tenants Better value for money	In-tenant scale-up Inter-tenant load balancing	In-tenant scale-out

Table 3: Features of different systems

Feature	Cassandra	MongoDB	HBase	Redis	MySQL	PostgreSQL	Paetica
Data model	Column-oriented	Document-oriented	Column-oriented	Key-value store	Row-oriented	Row-oriented	Hybrid row-column-oriented
ACID transactions	No	Yes (single document)	No	No	Yes	Yes	Yes
Scalability	Horizontally scalable	Horizontally scalable	Horizontally scalable	Horizontally scalable	Horizontally scalable	Horizontally scalable	Horizontally scalable& Vertically scalable
Replication	Masterless, peer-to-peer replication	Master-slave, automatic or manual sharding	Master-slave	Master-slave or masterless	Master-slave replication	Master-slave, publisher-subscribers	Master-slave & paxos-based replication
Data consistency	Tunable consistency	Eventual consistency	Strict consistency	NA (not a database)	Tunable consistency	Tunable consistency	Tunable consistency
Indexing	Secondary, composite, and custom indexes	Primary, secondary, and geospatial indexes	Row key	Primary, secondary, and geospatial indexes	Primary, unique, full-text indexes	Primary, expression, partial	Local, global, and unique indexes

availability of the system. It has also been designed to handle large volumes of data that are distributed across multiple data centers. However, it does not support ACID transactions, neither in the stand-alone nor in the distributed mode.

MongoDB [33] stores the data in JSON-like documents with optional schemas, and it supports horizontal scaling, which allows it to handle large amounts of data and high levels of read and write throughput. It is able to deploy a stand-alone instance of MongoDB for management purposes. However, it is important to note that stand-alone instances are typically only suitable for testing and development owing to their lack of replication and high availability. Whereas, a replica set is a group of MongoDB deployments that maintain the same data set and provide redundancy and high availability. Replica sets are typically used as the basis for all production

deployments. MongoDB only supports eventual consistency and is not extensively applicable for the industry that mandates strong data consistency.

Apache HBase [45] has been designed to provide real-time read and write access to large amounts of data, with low latency and high throughput. It can be deployed in either the stand-alone or distributed mode. In the stand-alone mode, HBase operates independently and utilizes the local filesystem rather than the Hadoop Distributed File System (HDFS) [40]. In the distributed mode, HBase daemons are distributed across all nodes in the cluster, thus allowing it to handle large amounts of data in a distributed environment. It is limited on inadequate indexing methods.

8.2 High-performance stand-alone database without distributed functionality

Redis [18] has been known for its high performance and low latency, making it well-suited for real-time applications that require fast data access. It supports a wide range of data structures, including strings, hashes, lists, sets, and sorted sets, thus allowing the users to store and manipulate data in a multitude of ways. It is scalable and can be employed in a distributed architecture, thus allowing it to handle large amounts of data across multiple servers. However, it does not provide data consistency and transaction, thus not satisfying the requirement of a robust database.

MySQL [34] is a widely-utilized relational database management system (RDBMS) known for its high-performance capabilities in managing and storing the data within various applications. The system employs an MVCC mechanism to effectively manage the concurrent access to the database. To optimize the query performance, MySQL supports a range of indexing options, including B-tree and hash indexes. However, it is important to note that MySQL is not inherently designed as a distributed database like OceanBase, though it can be configured to operate within a distributed environment through the implementation of techniques such as replication and sharding.

Replication, a feature within MySQL, enables the data to be duplicated from a primary server to one or more secondary servers, thereby increasing the availability and scalability. Sharding, on the other hand, involves the partitioning of a large database into smaller, more manageable units called "shards", which can be stored on separate servers. This approach allows for a greater horizontal scalability as more servers can be included in the system to accommodate the increasing load.

PostgreSQL [37] is a powerful relational database management system that is extensively used in academic research and other fields. One of its key features is its support for MVCC, which allows multiple users to access the same data simultaneously without mutual interference. Furthermore, PostgreSQL includes the built-in support for spatial data types and functions, enabling researchers to efficiently store and query the data with a geographic component. The system also has the capability for horizontal scalability that allows for the handling of large volumes of data and a high number of concurrent users without compromising the performance. Furthermore, PostgreSQL supports advanced indexing options such as B-Tree, R-Tree, Hash, and GiST, which can be utilized to optimize performance for specific types of queries. PostgreSQL can be employed as a distributed database, but it does not natively provide distributed functionality. However, it does offer mechanisms such as streaming replication and logical replication, besides the ability to utilize a foreign data wrapper, which provides access to the data stored in external databases as if it were stored locally.

8.3 High-performance distributed database without stand-alone functionality

Google Spanner [9] is a NewSQL database that shards data across many sets of Paxos state servers in data centers, globally. It has designed and implemented important database features atop their distributed-systems infrastructure via managing cross-datacenter replicated data. Spanner supports distributed transactions and provides an SQL-based query language. It combines and extends ideas from 1) the database community: an easy-to-use and semi-relational interface, transaction processing, and SQL engines, and 2) the system community: scalability, fault tolerance, consistent replication, and wide-area distribution. However, it cannot support good stand-alone performance owing to its distributed structure.

CockroachDB [42] that is inspired by Google Spanner is a scalable SQL-based distributed database to support the global OLTP workloads while maintaining strong consistency and high availability. Meanwhile, it is resilient to disasters through replication and automatic recovery mechanisms. Similar to OceanBase, it also utilizes a standard distributed shared-nothing architecture, where all nodes are used for both data storage and computation. Similarly, its stand-alone performance is not good due to a similar reason to Google Spanner.

TiDB [25] is a Raft-based HTAP database, which supports multi-Raft storage and HTAP engines. It has three core components, viz., 1) a distributed storage layer consisting of a row store (TiKV) and a columnar store (TiFlash), 2) a Placement Driver for managing Regions, that provides strictly increasing and globally unique timestamps, and 3) a computation engine layer that is stateless and scalable. However, due to its distributed structure, its stand-alone performance needs to be improved.

9 CONCLUSION

For the users, the stand-alone and distributed integrated capability of Paetica has two dimensions corresponding to value and meaning. First, Paetica can be deployed in a stand-alone mode, thus providing the ability of a stand-alone database. The administrator can dynamically transform the form, at any time, from a stand-alone configuration to a distributed configuration or vice versa. Second, Paetica has a significant performance advantage even in distributed deployment scenarios, as compared to other layered distributed databases. With a careful control, the efficiency and performance of Paetica can be comparable to that of a stand-alone database, even when the proportion of single-machine transactions increases. The experiments indicate that our design improves the scalability and performance of OceanBase 4.0. In the stand-alone mode, OceanBase 4.0 performs better than MySQL under the identical hardware environment. In the distributed mode, the query performance of OceanBase 4.0 is 6x that of Greenplum on TPC-H 100GB experiments.

REFERENCES

[1] 2021. Favorite of Taobao.com. https://shoucang.taobao.com.
[2] 2021. MulanPubL-2.0. https://license.coscl.org.cn/MulanPubL-2.0/index.html.
[3] 2021. OceanBase. https://gitee.com/oceanbase.
[4] 2021. OceanBase. https://github.com/oceanbase.
[5] 2021. OceanBase: 15 million QphH@30,000GB. http://tpc.org/3375.
[6] 2021. OceanBase: 707 million tmpC. http://tpc.org/1803.
[7] 2023. TPC-H. https://www.tpc.org/tpch/.
[8] Y Al-Houmaily and P Chrysanthis. 1995. Two-phase commit in gigabit-networked distributed databases. In *Int. Conf. on Parallel and Distributed Computing Systems (PDCS)*. Citeseer.
[9] 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.
[10] Claude Barthels, Ingo Müller, Konstantin Taranov, Gustavo Alonso, and Torsten Hoefler. 2019. Strong consistency is not hard to get: Two-Phase Locking and Two-Phase Commit on Thousands of Cores. *Proceedings of the VLDB Endowment* 12, 13 (2019), 2325–2338.
[11] Philip A Bernstein and Nathan Goodman. 1983. Multiversion concurrency control—theory and algorithms. *ACM Transactions on Database Systems (TODS)* 8, 4 (1983), 465–483.
[12] William Bridge, Ashok Joshi, M Keihl, Tirthankar Lahiri, Juan Loaiza, and N MacNaughton. 1997. The oracle universal server buffer manager. In *PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES*. Citeseer, 590–594.
[13] Wei Cao, Yang Liu, Zhushi Cheng, Ning Zheng, Wei Li, Wenjie Wu, Linqiang Ouyang, Peng Wang, Yijing Wang, Ray Kuan, Zhenjun Liu, Feng Zhu, and Tong Zhang. 2020. POLARDB Meets Computational Storage: Efficiently Support Analytical Workloads in Cloud-Native Relational Database. In *18th USENIX Conference on File and Storage Technologies, FAST 2020, Santa Clara, CA, USA, February 24-27, 2020*, Sam H. Noh and Brent Welch (Eds.). USENIX Association, 29–41.
[14] Michael J Carey and Waleed A Muhanna. 1986. The performance of multiversion concurrency control algorithms. *ACM Transactions on Computer Systems (TOCS)* 4, 4 (1986), 338–378.
[15] Apache Cassandra. 2014. Apache cassandra. *Website. Available online at http://planetcassandra. org/what-isapache- cassandra* 13 (2014).
[16] Transaction Processing Performance Council. 2010. TPC BENCHMARK™ C Standard Specification Revision 5.11 Standard Specification.
[17] Umur Cubukcu, Ozgun Erdogan, Sumedh Pathak, Sudhakar Sannakkayala, and Marco Slot. 2021. Citus: Distributed PostgreSQL for data-intensive applications. In *Proceedings of the 2021 International Conference on Management of Data*. 2490–2502.
[18] Maxwell Dayvson Da Silva and Hugo Lopes Tavares. 2015. *Redis Essentials*. Packt Publishing Ltd.
[19] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: Amazon's highly available key-value store. *ACM SIGOPS operating systems review* 41, 6 (2007), 205–220.
[20] Mostafa Elhemali, Niall Gallagher, Bin Tang, Nick Gordon, Hao Huang, Haibo Chen, Joseph Idziorek, Mengtian Wang, Richard Krog, Zongpeng Zhu, et al. 2022. Amazon {DynamoDB}: A Scalable, Predictably Performant, and Fully Managed {NoSQL} Database Service. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. 1037–1048.
[21] Vibby Gottemukkala, Edward Omiecinski, and Umakishore Ramachandran. 1994. A scalable sharing architecture for a parallel database system. In *Proceedings of 1994 6th IEEE Symposium on Parallel and Distributed Processing*. IEEE, 110–117.
[22] Jim Gray and Leslie Lamport. 2006. Consensus on transaction commit. *ACM Transactions on Database Systems (TODS)* 31, 1 (2006), 133–160.
[23] Rick Greenwald, Robert Stackowiak, and Jonathan Stern. 2013. *Oracle essentials: Oracle database 12c*. " O'Reilly Media, Inc.".
[24] Pat Helland, Harald Sammer, Jim Lyon, Richard Carr, Phil Garrett, and Andreas Reuter. 1987. Group Commit Timers and High Volume Transaction Systems. In *High Performance Transaction Systems, 2nd International Workshop, Asilomar Conference Center, Pacific Grove, California, USA, September 28-30, 1987, Proceedings (Lecture Notes in Computer Science)*, Dieter Gawlick, Mark N. Haynie, and Andreas Reuter (Eds.), Vol. 359. Springer, 301–329.
[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] Alexey Kopytov. 2018. Sysbench: Scriptable database and system performance benchmark.
[27] KR Krish, Aleksandr Khasymski, Guanying Wang, Ali R Butt, and Gaurav Makkar. 2013. On the use of shared storage in shared-nothing environments. In *2013 IEEE International Conference on Big Data*. IEEE, 313–318.
[28] Leslie Lamport. 2001. Paxos made simple. *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)* (2001), 51–58.
[29] Feifei Li. 2019. Cloud-native database systems at Alibaba: Opportunities and challenges. *Proceedings of the VLDB Endowment* 12, 12 (2019), 2263–2272.
[30] Yi Lu, Xiangyao Yu, Lei Cao, and Samuel Madden. 2021. Epoch-based commit and replication in distributed OLTP databases. (2021).
[31] 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.
[32] Hatem A. Mahmoud, Faisal Nawab, Alexander Pucher, Divyakant Agrawal, and Amr El Abbadi. 2013. Low-Latency Multi-Datacenter Databases using Replicated Commit. *Proc. VLDB Endow.* 6, 9 (2013), 661–672. https://doi.org/10.14778/2536360.2536366
[33] DB Mongo. 2015. Mongodb.
[34] AB MySQL. 2001. MySQL.
[35] Takatsugu Ono, Yotaro Konishi, Teruo Tanimoto, Noboru Iwamatsu, Takashi Miyoshi, and Jun Tanaka. 2014. FlexDAS: A flexible direct attached storage for I/O intensive applications. In *2014 IEEE international conference on big data (big data)*. IEEE, 147–152.
[36] 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.
[37] Behandelt PostgreSQL. 1996. PostgreSQL. *Web resource: http://www. PostgreSQL. org/about* (1996).
[38] George Samaras, Kathryn Britton, Andrew Citron, and C Mohan. 1995. Two-phase commit optimizations in a commercial distributed environment. *Distributed and Parallel Databases* 3, 4 (1995), 325–360.
[39] Amir Shaikhha, Mohammad Dashti, and Christoph Koch. 2018. Push versus pull-based loop fusion in query engines. *Journal of Functional Programming* 28 (2018), e10.
[40] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. 2010. The Hadoop Distributed File System. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. 1–10. https://doi.org/10.1109/MSST.2010.5496972
[41] Huaiming Song, Xian-He Sun, and Yong Chen. 2011. A hybrid shared-nothing/shared-data storage scheme for large-scale data processing. In *2011 IEEE Ninth International Symposium on Parallel and Distributed Processing with Applications*. IEEE, 161–166.
[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] Alexandre Verbitski, Anurag Gupta, Debanjan Saha, Murali Brahmadesam, Kamal Gupta, Raman Mittal, Sailesh Krishnamurthy, Sandor Maurice, Tengiz Kharatishvili, and Xiaofeng Bao. 2017. Amazon aurora: Design considerations for high throughput cloud-native relational databases. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 1041–1052.
[44] Tobias Vinçon, Christian Knödler, Leonardo Solis-Vasquez, Arthur Bernhardt, Sajjad Tamimi, Lukas Weber, Florian Stock, Andreas Koch, and Ilia Petrov. 2022. Near-data processing in database systems on native computational storage under htap workloads. *Proceedings of the VLDB Endowment* 15, 10 (2022), 1991–2004.
[45] Mehul Nalin Vora. 2011. Hadoop-HBase for large-scale data. In *Proceedings of 2011 International Conference on Computer Science and Network Technology*, Vol. 1. IEEE, 601–605.
[46] Zhenkun Yang, Chen Qian, Xuwang Teng, Fanyu Kong, Fusheng Han, and Quanqing Xu. 2023. LCL: A Lock Chain Length-based Distributed Algorithm for Deadlock Detection and Resolution. In *2023 IEEE 39th International Conference on Data Engineering*. IEEE, xxx–xxx.
[47] 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, and Quanqing Xu. 2022. OceanBase: A 707 Million tpmC Distributed Relational Database System. *Proceedings of the VLDB Endowment* 15, 12 (2022), 3385–3397.
[48] Qian Zhang, Jingyao Li, Hongyao Zhao, Quanqing Xu, Wei Lu, Jinliang Xiao, Fusheng Han, Chuanhui Yang, and Xiaoyong Du. 2023. Efficient Distributed Transaction Processing in Heterogeneous Networks. *Proc. VLDB Endow.* 16, 6 (2023), 1372–1385.