A Review of NewSQL Database Systems

Siying Cen

Introduction

SQL database, or RDBMS (Relational Database Management Systems), is the most well-known and the most traditional database. It is based on the relational model and data is stored in the form of a table or relationship. With the exponential growth of data, structured approach is not enough to meet the needs of big data, expecting quick responsive and scalability.

To solve this problem, another type of database system called NoSQL was developed to provide scalability for big data applications. NoSQL databases consist of key-value pairs, documents, graphical databases, or wide columns without a standard architecture. It can also be scaled horizontally rather than vertically extend like RDBMS.

NoSQL database provides solution for big data application system; however, it is not enough due to some shortcomings, such as ACID (Atomicity, Consistency, Isolation and Durability) properties not guaranteed. Also, earlier versions of databases fail to be compatible with NoSQL.

As a result, the idea of NewSQL appears aiming at making relational SQL more scalable. NewSQL is the combination of SQL and NoSQL which provides scalable performance as NoSQL. NewSQL implements distributed system, so it can be applied on cloud computing platforms easily.

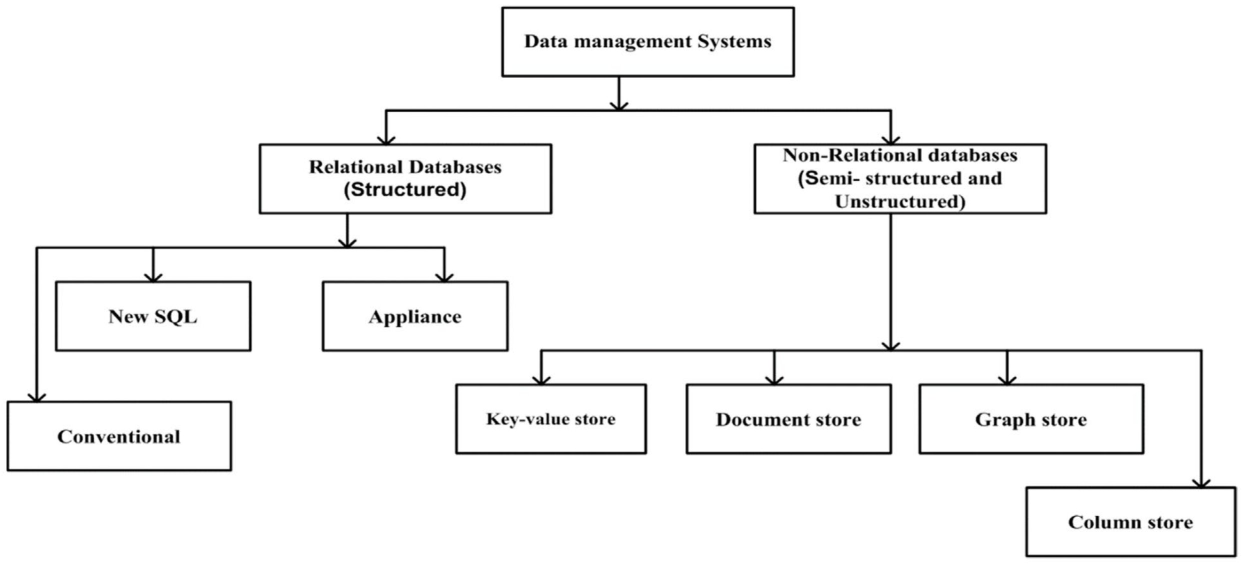


Figure 1-1. Tree structure of Data Management System [1].

Table 1-1: Comparison of SQL, NoSQL and NewSQL [2]

Features	SQL	NoSQL	NewSQL
Relational	Yes	No	Yes
ACID	Yes	No (CAP instead)	Yes
SQL	Yes	No	Yes
OLTP	Not fully	Supported	Fully
Scaling	No	Yes	Yes
Query Complexity	Low	High	Very High
Distributed	No	Yes	Yes

Design and properties of NewSQL systems

2.1 Spanner and F1

Spanner is a scalable, globally distributed database developed by Google in 2012 [3]. It has strict transactional guarantees for even inter-row transactions. A *universe* is a Spanner deployment, and zones are units of administrative deployment, which are analogues to deployment of BigTable servers.

There is one zonemaster and hundreds to thousands of spanservers in a zone; the proxies are used by clients to locate the spanservers. The organization of Spanner is shown in Figure 2-1.

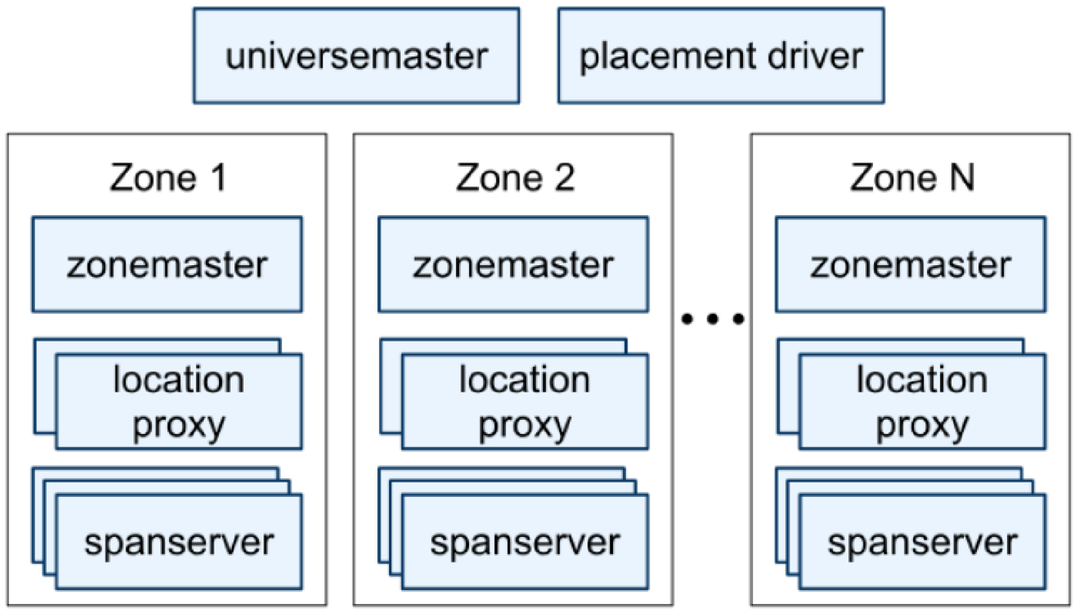


Figure 2-1. Spanner Server Organization [3].

There are two important features in Spanner, which are difficult to implement on a distributed database: first, Spanner provides external consistency for read and write operations; second, it maintains globally consistent read operations through the database under a timestamp. They enable Spanner to support consistent backups and MapReduce execution, and atomic schema changes, all implemented on a global scale, even if there are transactions in progress. These features all benefit from Spanner's global time synchronization mechanism, which gives a timestamp when the data is submitted. Because time is serialized, there is external consistency. The timestamp order reflects the commit order.

The global time synchronization mechanism is provided using a TrueTime API with GPS and atomic clocks. This TrueTime API can reduce the time deviation of different data centers to 10ms. This API provides a precise time and a range of errors.

$$\text{(key:string, timestamp:int64)} \rightarrow \text{string}$$

Figure 2-2. Timestamp assignment to data in Spanner [3].

Spanner uses the Paxos protocol to synchronize redo logs across multiple copies to ensure that data is consistent across multiple copies. There are three roles in the Paxos protocol: Proposer, Acceptor and Proposer, respectively the reader and the Proposer, and Acceptor is the storage node. The entire protocol describes how a resolution process (Paxos Instance) is initiated each time a Proposer writes a variable when there are multiple proposers and acceptors in the system. The resolution process ensures that even if multiple proposers write at the same time, the results will not differ on an Acceptor node. Specifically, once a value submitted by a Proposer is accepted by a majority of acceptors, that value is selected and the variable is no longer modified to any other value during the course of a resolution. If another Proposer wants to write another value, it must start the next round of resolution processes, which are Serializable between them.

The basic Paxos protocol guarantees that values will not change once they are selected, but there is no guarantee that they will be selected. In other words, the voting algorithm doesn't necessarily converge. There are two ways to accelerate the process of convergence: one is to submit an opportunity to another Proposer with a random delay in the event of a conflict, and the other is to try to have only one Proposer in the system to submit. Both methods are used in the Spanner system, where a leader node is selected in a number of Proposer via a round of Paxos protocol with random delay. All the following write operations go through this leader node, and the leader node generally has a "long life". In wan environment, the average "tenure" can reach more than one day.

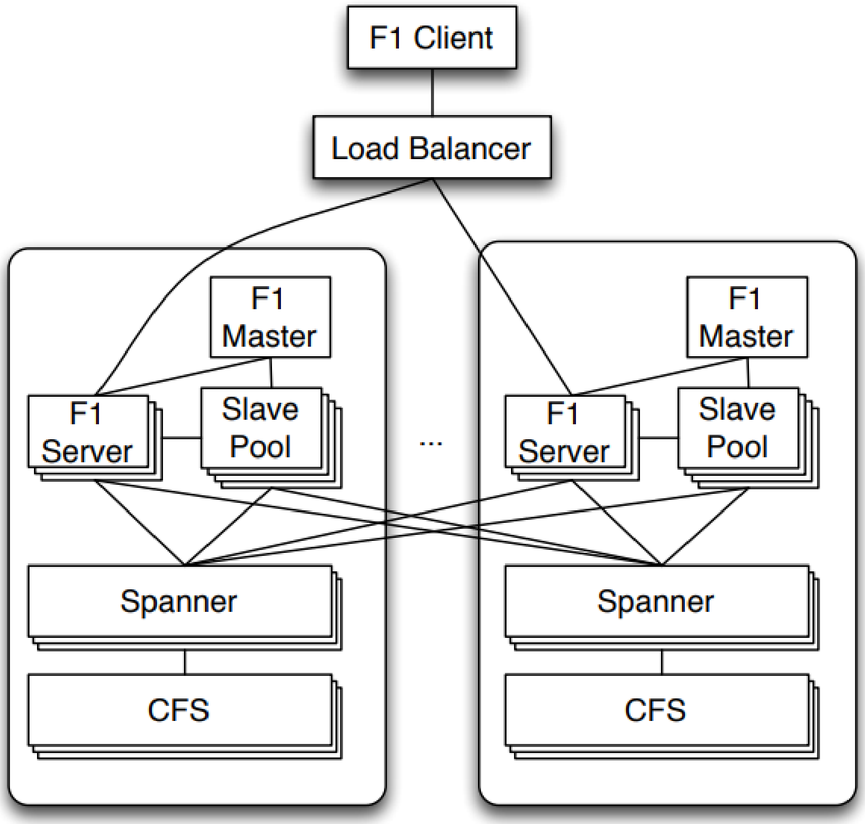


Figure 2-3. The basic architecture of F1 system [4].

F1 system is built on Spanner, providing distributed SQL, transaction consistent secondary indexes and other functions based on the rich features of Spanner. Since 2012, Google used F1 to successfully replace the old manual MySQL Shard scheme in AdWords advertising business. The early Spanner data model was much like Bigtable, but Spanner later adopted the F1 data model.

F1 has a relational schema much like an RDBMS, with the addition of inter-table hierarchy and support for a data type called Protocol Buffer. What about the hierarchy between tables? Actually is the primary key part of the child table must refer to the parent table, such as: table Customer have a primary key (CustomerId), its child table Campaign of the primary key must also contain (CustomerId), becomes this form (CustomerId, CampaignId); To push if Campaign also has a child table AdGroup, AdGroup primary key must be this kind of form - (CustomerId, CampaignId, AdGroupId). The primary key of the outermost parent table Customer, CustomerId, is called a root row. All the subtable-related rows that reference this root row constitute the directory in the Spanner. The following figure compares the hierarchy of RDBMS with F1 and Spanner:

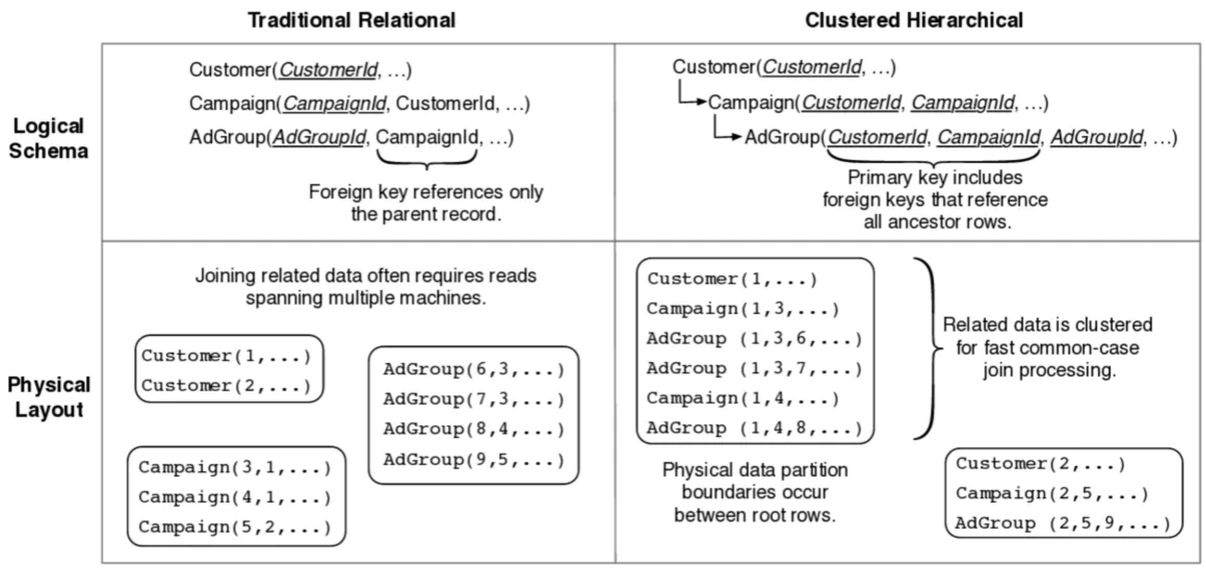

Traditional Relational
Clustered Hierarchical
Logical Schema
Customer(CustomerId, …)
Campaign(CampaignId, CustomerId, …)
AdGroup(AdGroupId, CampaignId, …)
Foreign key references only the parent record.
Customer(CustomerId, …)
Campaign(CustomerId, CampaignId, …)
AdGroup(CustomerId, CampaignId, AdGroupId, …)
Primary key includes foreign keys that reference all ancestor rows.
Physical Layout
Joining related data often requires reads spanning multiple machines.
Customer(1,...)
Customer(2,...)
AdGroup(6,3,...)
AdGroup(7,3,...)
AdGroup(8,4,...)
AdGroup(9,5,...)
Campaign(3,1,...)
Campaign(4,1,...)
Campaign(5,2,...)
Customer(1,...)
Campaign(1,3,...)
AdGroup (1,3,6,...)
AdGroup (1,3,7,...)
Campaign(1,4,...)
AdGroup (1,4,8,...)
Related data is clustered for fast common-case join processing.
Physical data partition boundaries occur between root rows.
Customer(2,...)
Campaign(2,5,...)
AdGroup (2,5,9,...)

Figure 2-4. The comparison of data storage in a traditional normalized relational schema and a clustered hierarchical schema used in an F1 database [4].

2.2 OceanBase [5]

OceanBase is a high-performance distributed database system developed independently by Alibaba. Unlike other database systems, OceanBase faced a different challenge when it was born: Internet companies had a lot of concurrent traffic. Concurrent access to hundreds and thousands of databases is quite common, while concurrent access to tens of thousands of databases is quite rare. In internet business, every user can directly lead to database access, various sales promotion activity has hundreds of thousands, millions, even thousands of users in a relatively short period of time to buy goods and pay, resulting in a database of hundreds of thousands, millions or even tens of millions of concurrent access.

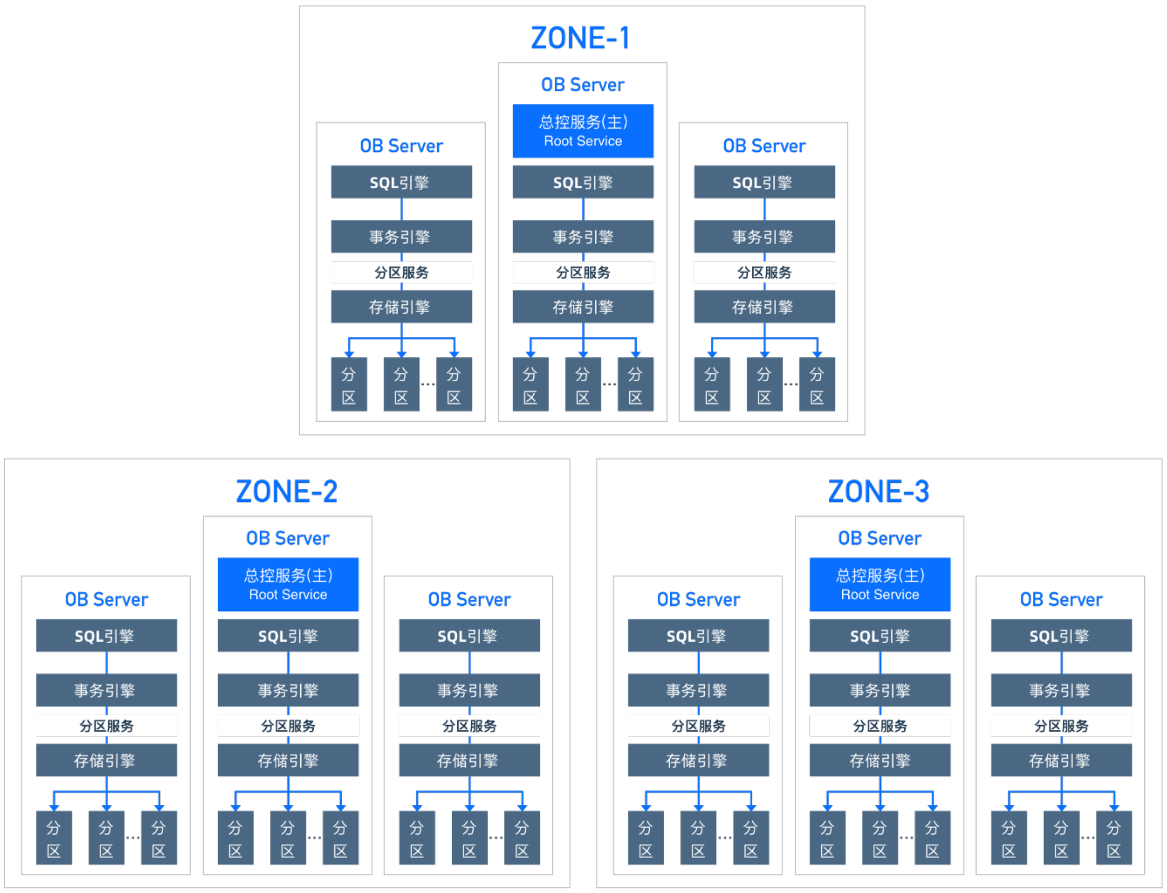


Figure 2-5. Architecture of OceanBase [6].

- High scalability

The traditional relational database has perfect functions, but the database itself is not extensible. Database operation and maintenance staff spend a lot of time to do database expansion, including read and write separation, vertical separation, horizontal separation and so on.

OceanBase uses distributed technology and a shared-nothing architecture to ensure that access from the business is automatically distributed across multiple database hosts. It is also capable of hosting its database using a cheap PC server with the technology. With these two changes, operations personnel can easily increase the capacity and performance of the system by increasing the number of servers (PC servers are also much cheaper than minicomputers and mainframes).

- High reliability

The stability and reliability of database system depend on database software, database server hardware and database storage hardware. The use of PC servers can bring high scalability and cost reduction, while the reliability of the hardware is correspondingly somewhat reduced.

How to ensure the reliability of the system? One of OceanBase's basic assumptions is that the hardware is unreliable, so OceanBase must ensure that a small number of hardware (server, storage, network, etc.) exceptions that occur at any given time do not affect the business.

To solve this, OceanBase introduced the Paxos protocol. Each transaction must be synchronized to more than half of the libraries (including the main library itself) after the execution of the main library, for example, 2 of the 3 libraries, or 3 of the 5 libraries, before the transaction is successful. In this way, business is not affected when a few libraries, such as one in three or two in five, exception. The distributed transaction Consistency protocol (PAXOS) is mainly used to ensure the reliability of data in distributed systems.

- Data accuracy

Many Internet services can allow data error to a certain extent, but the electronic commerce and Internet companies are different, it is very high demand for the consistency of the data, not lost any of the payment data (alibaba use OceanBase alipay system).

OceanBase's design differs from that of a classic relational database in that read transactions are basically distributed and concurrently executed, write transactions are currently centrally serialized, namely Serializable, and since any write transaction is not visible to other read and write transactions until commit, OceanBase is strongly consistent. In this way, the design scheme can be guaranteed not to lose data.

- High performance

The OceanBase architecture has the advantage of supporting both cross-row and cross-table transactions as well as linear scaling of the storage server. However, the single point of UpdateServer, which limits OceanBase cluster's overall read-write performance, is an obvious disadvantage of OceanBase. To achieve high performance based on high scalability, low cost, high reliability, and data consistency, OceanBase differs from traditional databases in terms of data access. As in many industries, although the total amount of data is very large, there is a limit to how much data can be added, deleted or changed over a period of time.

According to this feature, OceanBase records the additions, deletions, modifications and other modifications over a period of time in the form of increments, which also keeps the main data relatively stable (called the baseline data) over a period of time. Because the incremental data is relatively small, OceanBase typically keeps it in the memory of a separate server, UpdateServer. The performance of the system write transaction is greatly improved by keeping the add, delete and change records in memory. Not only that, but since the frozen memory table is no longer modified, it can also be converted to sstable format and saved to an SSD solid state disk or disk. The memory taken up after a dump to an SSD solid disk can be released and still provide higher performance read services, which also alleviates the memory requirements of an UpdateServer in extreme cases.

2.3 CockroachDB

CockroachDB is a distributed database system that supports SQL, ACID for distributed transactions, and ANSI SQL's highest isolation level Serializability. It is a cloud-based system capable of supporting global OLTP workloads with strong consistency and high availability.

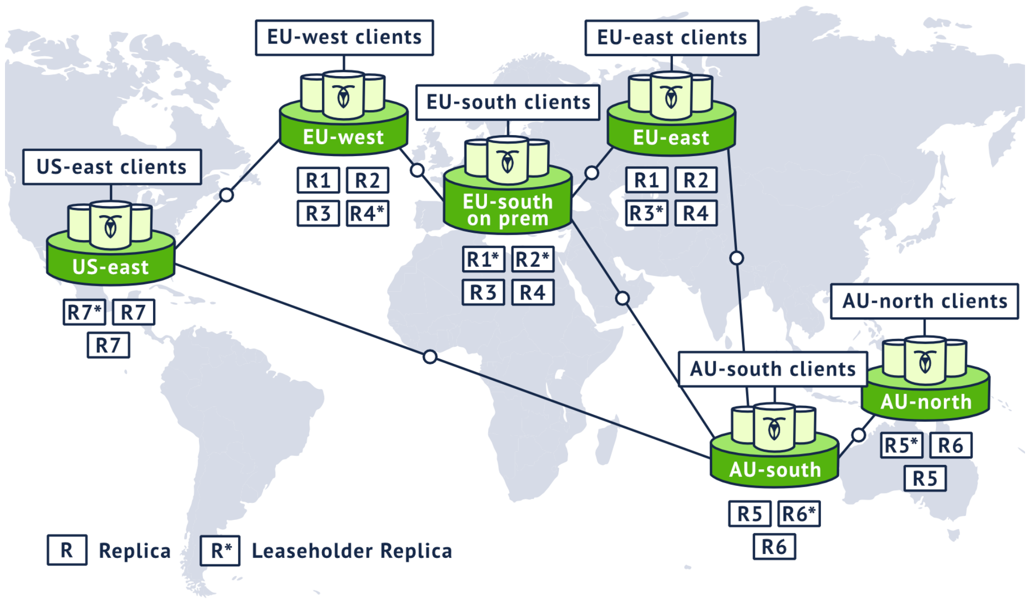


Figure 2-7. A global CockroachDB cluster [7].

CockroachDB's transaction is lock-free and does not require any read-write locks, so it is natural to maintain multiple versions of the data, identified by timestamp.

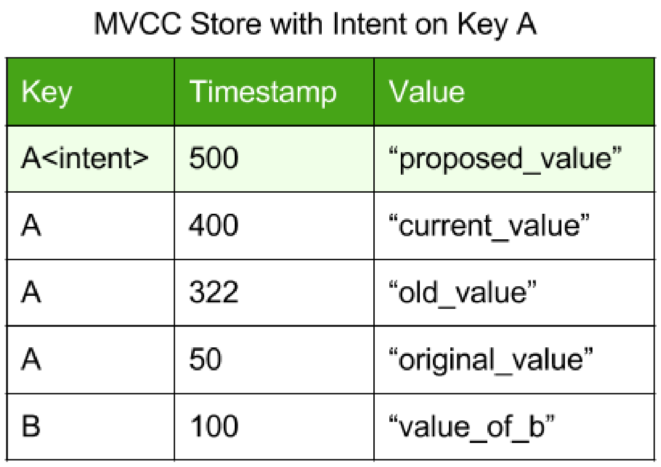
A and *I* in *ACID* are closely related and both are guaranteed by concurrency control protocol. The following describes how *A* is guaranteed, and then how *I* is guaranteed in the case of concurrency. Concurrency control protocol guarantees *A* and *I*.

How can atomicity be guaranteed when a distributed transaction may read and write data on multiple nodes? We know 2PC is used in distributed transactions, which means do prepare first, then read the data needed in calculation, finally write the calculated data into each node without be in effect; thus, the other affairs cannot read the data temporarily in the system. CockroachDB calls this type of data that has been written to each node but is not in effect called a Write Intent, which is stored with the actual data but is just not read externally.

At the beginning of a transaction, a Record is written into the underlying storage system. This Record is called a Transaction Record, which records the Transaction ID, Transaction status, Pending, Committed, or Aborted. To commit a Transaction, just change the Transaction state in the Transaction Record to Committed and roll back the Transaction to Aborted. Once the transaction state has been successfully modified, it can be returned to the client and the legacy Intent is handled asynchronously: at commit, override the original intent value, delete the Intent, and rollback immediately delete the Intent.

Then when the client comes over and reads this, if it encounters a Write Intent, it will go to the Transaction Record along the Write Intent and look at the state of the Transaction. If the state is committed, it returns the value in the Write Intent and Abort will return the original value. If it is Pending, it means the transaction is still running and has encountered a writing conflict.

Table 2-1 [8].



MVCC Store with Intent on Key A

Key	Timestamp	Value
A<intent>	500	“proposed_value”
A	400	“current_value”
A	322	“old_value”
A	50	“original_value”
B	100	“value_of_b”

In a distributed system, it is difficult to support Linearizability because the clocks between different machines have errors and require a global clock. CockroachDB does not use an atomic clock or single timestamp oracle. Instead, it tries to synchronize clock offset between machines based on physical time (NTP). NTP errors of 250ms or more are not strictly guaranteed, which makes it difficult for CockroachDB to hold the Linearizability to the same, and the performance is poor. Ultimately though CockroachDB supports Linearizability, it is not officially recommended. By default, CockroachDB supports Serializable isolation levels, but does not guarantee Linearizability.

To implement Serializability, we need to ensure that the scheduling of transactions is acyclic. CockroachDB avoids the three conflicts(RW, WR, WW) by going in the opposite direction of timestamp, thus ensuring that there are no edges in the diagram that go in the same direction as timestamp, and thus acyclic.

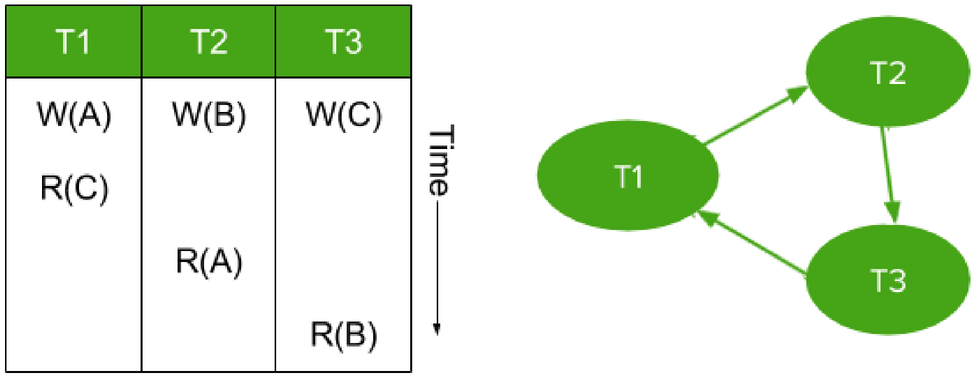


Figure 2-8. Example of a transaction history with a cyclic serializability graph. This history is not serializable [8].

In general, CockroachDB's concurrency control protocol is a lock-free, unlocked, and optimistic protocol. For applications with strong data contention, frequent Restart transactions are not desirable. Moreover, NTP is not always guaranteed to keep the clock error between machines within a range beyond which Serializability is violated.

2.4 TiDB

TiDB is a cloud-native, open-source HTAP (Hybrid Transactional and Analytical Processing) Database developed by PingCAP[9]. It is compatible with MySQL syntax, supporting unlimited horizontal extension, having strong consistency and high availability. TiDB is now increasingly used by many commercial companies in the business system.

TiDB consists of three core components: TiDB Server, PD Server and TiKV Server. TiDB Server is the computing layer, responsible for SQL statements implementation; TiKV Server is the storage layer which provides distributed storage for TiDB; PD Server is responsible for cluster management, including cluster-related metadata storage, global transactions management and load balancing of TiKV clusters.

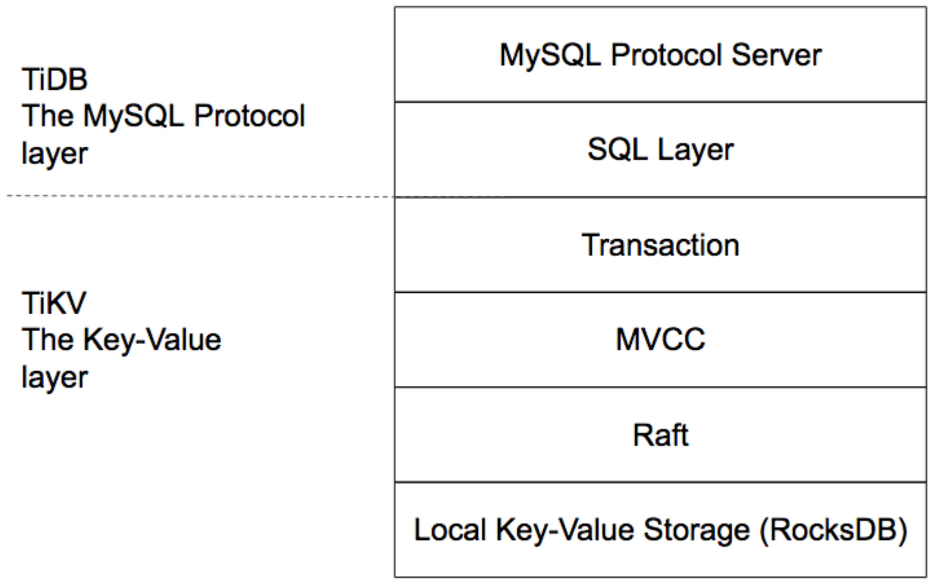


Figure 2-9. Architecture of TiDB system[10].

Tikv supports unlimited horizontal expansion with strong consistency and high availability. When a client sends a request to Tikv, Tikv first parses the protocol and then distributes the request to the Tikv thread. The Tikv thread executes some transaction logic and sends the request to the Raft thread. After Tikv copies the Raft Log and applies it to RocksDB, the entire write request is completed.

Tikv implements remote replication using Raft algorithm. Raft is a consensus algorithm equivalent to Paxos algorithms. Tikv was designed to store huge amounts of data. One Raft is not enough. Thus, it slices data by range, and then treat the data from each range as a separate Raft Group. This method is called multi-raft.

For the same Region, the consistency of the key operation in it can be guaranteed through the Raft consistency protocol, but if we want to operate on multiple data at the same time, and the data falls on different regions, we need a distributed transaction to ensure the consistency of the operation. The most common approach for distributed transactions is to use 2PC, but traditional 2PC requires a coordinator, and we also need mechanisms to ensure high availability of the coordinator. Here, TiKV references Google's Percolator and optimizes 2PC to provide distributed transaction support.

The Tikv underlayer uses Facebook open source RocksDB, high-performance persistent key-value storage, to achieve the full potential of data storage that is now rapidly stored under server load (especially flash storage). For TiKV, any data will eventually be converted into one or more key-values and stored in RocksDB.

TiDB supports most MySQL syntax, including cross-row transactions, joins, and subqueries, and users can directly use existing MySQL client connections. If the existing business has been developed based on MySQL, in most cases, the stand-alone MySQL can be directly replaced without modifying the code, including most existing MySQL operation tools (such as PHPMyAdmin, Navicat, MySQL Workbench, etc.), as well as backup and recovery tools can be used directly.

Discussion

Serializability and Linearizability are two golden standards for evaluating distributed database systems.

Serializability is the strongest guarantee for transaction isolation, and Linearizability is the strongest guarantee for 'Consistency' in the CAP domain. Spanner holds the Linearizability based on atomic clocks, which holds a precision error of about 7ms. CockroachDB does not hold a precision atomic clock, instead useing an NTP synchronous clock. The clock deviation between machines can be controlled to within 250ms, but it is not absolute, which is affected by factors such as network delay and system load. If using the same approach as Spanner to implement Linearizability, performance is obviously poor. Thus by default, CockroachDB only provides Serializability (assuming the clock deviation is still within a range and can be configured), instead of Linearizability, which in turn does not provide Strict Serializability. TiDB, like Percolator, provides a unique clock source through Timestamp Oracle, and similarly Linearizability can be achieved.

Spanner/F1 has been applied to Google AdWords business since 2012 and OceanBase has been used in Alibaba applications since 2013. Spanner/F1 and OceanBase are used by their companies only, while CockroachDB and TiDB are open source.

When selecting NewSQL database, we should consider both the compatibility with traditional technologies and the foresight of new technologies, including the following aspects:

- ACID support: Because the NewSQL database is intended to handle OLTP, good ACID property support is required.
- SQL compatibility: To reduce operational and development costs, we need NewSQL database compatibility with traditional SQL databases such as MySQL and PostgreSQL. The more compatible the application, the easier it will be to migrate from a traditional SQL database to a NewSQL database. If 100% compatibility is achieved, seamless migration of the application system can be achieved.
- Distribution and scalability: Distributed is the foundation of the new generation architecture, extended performance against massive data volumes. Therefore, distribution and scalability are the basic requirements of the new generation of NewSQL database
- OLTP and OLAP support: NewSQL database should be able to handle OLTP and OLAP at the same time, to meet the needs of more data applications
- Multi-model and multi-tenant: NewSQL database should be a multi-model multi-mode database engine, which can handle a variety of data application scenarios, in line with the architectural concept of micro service and cloud database.

Outlook

NewSQL databases represent the future direction of new databases design, which aims at combining features of traditional SQL databases with scalable distributed NoSQL databases. According to CAP theorem, SQL provides 'CA' properties while NoSQL ensures 'AP' features. We can view NewSQL as a transient system between 'CA' and 'AP' systems.

NewSQL distributed database can be most compatible with relational database, such as support for SQL, multi-level index, support for ACID transaction with strong consistency and transparency, so that the development and migration cost of the application can be minimized. Compared with traditional relational databases, NewSQL has the advantage of being able to scale resiliently and seamlessly. The availability and performance of the whole system are not comparable to those of a stand-alone database. The value of this revolutionary change is undoubtedly huge. Imagine that storing and querying vast amounts of structured data would no longer be a headache, freeing up business and increasing productivity. However, NewSQL usually only support basic SQL queries (insert, delete, update, select), so there is still a long way to go for NewSQL supporting SQL uncompromisingly.

We shall notice that in the field of software design, it is difficult for closed-source software to succeed in the future, and the enterprise IT software stack is gradually shifting from closed-source proprietary software to open source solutions. Take TiDB as an example, it stores data in Facebook open source RocksDB, generates strictly increasing and globally unique timestamps via timestamp oracle, connects with Hadoop ecosystem by extending to Spark, and finally publish TiDB to open source community for collaboration. Many users provide valuable feedback to TiDB developer and contribute codes actively.

References

[1] Ramzan, Shabana, Imran Sarwar Bajwa, and Rafaqut Kazmi. "Challenges in NoSQL-Based Distributed Data Storage: A Systematic Literature Review." Electronics 8.5 (2019): 488.

[2] Binani, Sneha, Ajinkya Gutti, and Shivam Upadhyay. "SQL vs. NoSQL vs. NewSQL-A comparative study." database 6.1 (2016): 1-4.

[3] Corbett, James C., et al. "Spanner: Google's globally distributed database." ACM Transactions on Computer Systems (TOCS) 31.3 (2013): 1-22.

[4] Shute, Jeff, et al. "F1: A distributed SQL database that scales." (2013).

[5] 阳振坤. "OceanBase 关系数据库架构." *华东师范大学学报: 自然科学版* 5 (2014): 141-148.

[6] https://www.oceanbase.com/product/oceanbase

[7] Taft, Rebecca, et al. "CockroachDB: The Resilient Geo-Distributed SQL Database." Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. 2020.

[8] Grim, Max Wouter. "Consistency analysis of CockroachDB under fault injection." Science (2016).

[9] Huang, Dongxu, et al. "TiDB: a Raft-based HTAP database." Proceedings of the VLDB Endowment 13.12 (2020): 3072-3084.

[10] https://zhuanlan.zhihu.com/p/25142743