Enabling new database services with a disaggregated storage

Chong Chen, Alex Depoutovitch

Huawei Research Canada
chongchen@huawei.com,alex.depoutovitch@huawei.com

1 Introduction

In modern-day cloud databases, the core database engine is an important but relatively small component. A lot of effort goes into creating infrastructure services around it. Examples of such services include continuous backup and fast or even instantaneous restore of terabyte-scale databases, nearly zero-time recovery with zero data loss after region-scale disasters, and instantaneous scalability in the face of changing workloads. Services may also include "time travel" - instantaneous rewind of a database to any point in the past, support of data replication to multiple geo-distributed zones, and others. These features are often as important as core database engine. Decoupled compute and storage, which is a de facto standard in cloud-native databases, opens many new possibilities for designing such services. In this talk, we first present a novel decoupled, append-only intelligent storage architecture that enables all of the above features in a uniform way. Next, we show how this architecture enables horizontal scalability, load balancing, instantaneous restore and time-travel. We discuss the technical challenges and explain how we designed the system and technology to overcome them. The described architecture is currently implemented in Taurus (GaussDB for MySQL) cloud-native database and used by thousands of customers worldwide.

2 Horizontal scalability

Horizontal scalability is often achieved by adding more read replica nodes to serve reads and master nodes to serve write workloads. Disaggregated storage shared between nodes allows nearly instantaneous creation of read replicas and masters. However, read replicas must retrieve and replay logs from the master to keep buffer pools in sync. This log replay process results in a time delay known as read replica lag. We present a new log delivery and replay architecture aimed at reducing this lag to a single-digit millisecond level without the overheads of synchronous replication. Our disaggregated storage layer serves as a delivery mechanism for logs and provides multi-version reads for pages. In our architecture, read replicas don't put additional loads on the master, allowing nearly unlimited read query scalability.

Another way to achieve horizontal scalability is to have a multi-master architecture. However, from our previous multi-master implementation, we have found that a generic multi-master implementation brings a lot of complexity and synchronization overhead. We have also found that the majority of customers who require multi-master architecture for scalability reasons have naturally partitioned workloads and use multi-master clusters for scaling and load-balancing. Many of them are SaaS (Software As A Service) providers who deploy and manage a fleet of database instances to support a large number of tenants. Our decoupled compute and storage architecture is based on a stateless compute layer, disaggregated storage, and a proxy system. We show how it allows SaaS providers to nearly instantaneously migrate tenant logical databases for load balancing, remove and add new servers for scalability with a negligible business interruption.

3 Robust restore

Synchronous data replication protects data from most hardware failures. However, it does not protect against user and application errors. The amount of data stored in a database can be very large; for example, Taurus currently supports 128TB, and there is a demand for increasing it even further. Even with state-of-the-art hardware, it can take many hours or even days to restore the database from a backup in full. Our disaggregated storage layer allowed us to implement an on-demand restore service, which makes the database available during restore very quickly, independent of database size, by loading the working set of the database first and then loading the rest of the database in the background. We are going to present the overall architecture of this approach and the challenges we have experienced.

Finally, we are going to present how our append-only disaggregated storage layer enabled us to implement fast time-travel for a database. With this feature, recent database modifications can be quickly rewound, independent of database size, allowing for quick fixes of user errors.