Design of Amazon Aurora[1]: A Cloud-Native Relational Database

1st Ahmet Uysal
Computer Engineering Department
Koç University
İstanbul, Turkey
auysal16@ku.edu.tr

***Abstract*—Aurora is a purposefully built relational database service for cloud that is offered by Amazon Web Services. In this paper, design considerations and general structure of Aurora will be examined. Aurora brings a novel architecture to the relational database to address network constraint of cloud environment, most notably by pushing redo processing to storage layer. In Aurora, storage is implemented as an independent fault-tolerant service. To minimize amount of traffic between the database engine and storage, Aurora implements some key components such as redo logging, durable storage, crash recovery, and backup mechanisms in storage layer.**

***Index Terms*—databases, distributed systems, log processing, quorum models, replication, recovery, performance, oltp**

I. Introduction

Aurora is a relational database service offered by Amazon Web Services. In a cloud environment, challenges of database design are different from the ones of traditional database systems since available resources and technical limitations are different. On traditional database systems, the speed of I/O operations is the limiting factor, i.e., the bottleneck, that determines the performance of the system. However, since I/Os can be spread to multiple devices in cloud environment, communication between database and storage constitutes the bottleneck. Amazon Aurora is a cloud-native relational database that is specifically designed in light of strengths and weaknesses of cloud environments, and expectations from cloud ecosystem such as scalability.

Aurora makes a clear distinction between database and storage layers. Storage is implemented as an independent fault-tolerant service. The database engine is a fork of MySQL, however, some of its responsibilities are completely delegated to storage layer to optimize network traffic between the layers. Redo logging, durable storage, crash recovery, and backup mechanisms are some examples that are normally implemented in MySQL engine but implemented at the storage layer in Aurora. Unnecessary writes and communication are also removed to optimize performance. Also, Aurora further decreases the stalls by utilizing asynchronous execution where possible.

Another significant design choice of Aurora is how it treats the redo logs. In traditional systems, a redo log is generated when a data page is modified. These logs are used to construct after-image from before-image of the page. In Aurora, “redo log is the database”. Redo logs are the online write operations that cross the network between database and storage layers. Redo logs are processed in storage layers to construct data pages, however, these are similar to caches from Aurora’s design perspective. Their only purpose is to answer queries faster when possible. They are not counted as a part of the database.

II. Durability

Durability is an essential characteristic of a database system. Once an update is made to the system, it should be readable. Aurora uses a quorum based model to satisfy durability. Specifications of the quorum based system are based on the overall architecture of Amazon Web Services.

A. AWS Structure

AWS is divided into regions according to the geographical regions. These regions are further divided into *availability zones (AZs)*. Availability zones in the same region are connected to each other via low latency connections, however, they are isolated from each other. AZs are an important precaution against rare large-scale failures such as floods, long lasting power outages and fires. Dividing regions into availability zones significantly reduces the chance of a customer having all of her nodes fail at the same time.

B. Quorum Sizes

Aurora aims to prevent data loss even in the case of failure of an availability zone and a random node failure at another node. This is called *AZ+1 fault-tolerance*. Aurora also aims to retain its write ability in the case of an AZ failure.

These requirements are accomplished by replicating each items 6 times as 2 copies at 3 different AZs. The selected write quorum size is 4 ($V_w = 4$). This value ensures that each write operation contains at least one member from the previous write operation ($V_w > V/2$). The selected read quorum size is 3 ($V_r = 3$). This value ensures that each read quorum contains at least one member from the previous write operation ($V_r + V_w > V$). In the case of AZ+1 failure, a read quorum can be selected as they are still 3 functional copies. In the case of an AZ failure, a write quorum can be formed as there are 4 copies functioning normally.

C. Segmented Storage

In cloud systems, there is a constant noise of node and network failures. In order to ensure an *AZ+1 fault-tolerant* system satisfies durability requirement well enough, we need to decrease the likelihood of two independent failures in the time frame it takes to repair a malfunctioning copy. Design of Aurora focuses on the ladder component and tries to minimize the time it takes to repair a failure (Mean Time to Repair - MTTR). For this purpose, database volume is partitioned into fixed sized (10 GB) segments. These segments are replicated 6 ways as described in previous section.

III. Redo Logs

In traditional databases, a **redo log** is generated when a data page is modified. These logs are used to construct after-image from before-image of the page. In Aurora, "redo log is the database". Redo logs are the only write operations that cross the network between database and storage layers. Redo logs are processed asynchronously in the storage layer to create data pages. This operation is only done for optimizing response time to read requests. It carries a similar purpose with caches in a traditional storage system. It is not counted as an essential part of data. This view allows Aurora to significantly shrink the amount of data it passes between database and network layers.

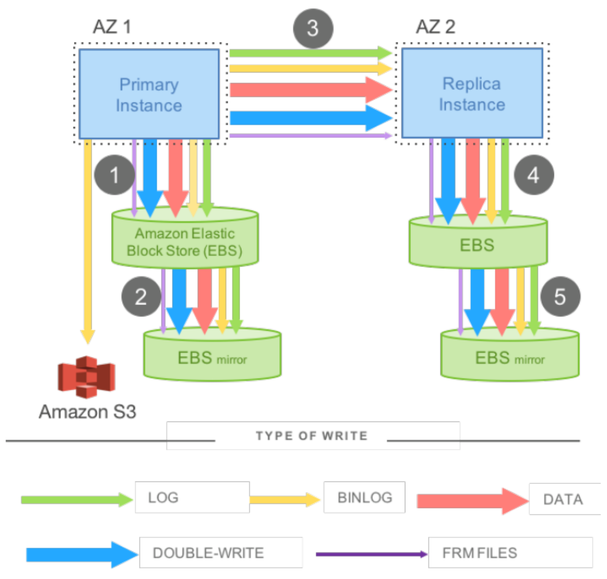

Fig. 1. Network IO in mirrored MySQL

Figure 1 illustrates different data types that the database engine needs to write: the redo log, the binary log, modified data-pages, second temporary copy of the page, and page metadata. Note that steps 1, 3, and 5 are sequential and synchronous. Latency in this scenario is additive because many writes are sequential.

Figure 2 illustrates a write operation in an Aurora cluster with one primary instance and two replicas in different AZs. In this scenario, the primary instance only writes redo logs to storage and streams this logs to other instances. Storage services process redo data to construct data pages and may backup these pages in Amazon S3 storage for faster failure recovery. However, these steps are entirely optional as described in previous sections.

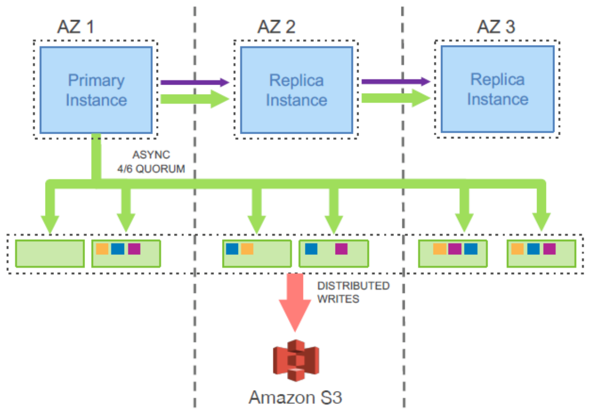

Fig. 2. Network IO in Amazon Aurora

IV. Design of the Storage Layer

Store is implemented as an independent fault-tolerant service in Aurora architecture. The main goal of Aurora's storage service is to minimize foreground write request latency. Majority of storage processing is run asynchronously in background and does not affect delays in the system. Unlike traditional database systems, background processing has negative correlation with foreground processing in Aurora. This allows the system to balance between its activities based on the current load on the system.

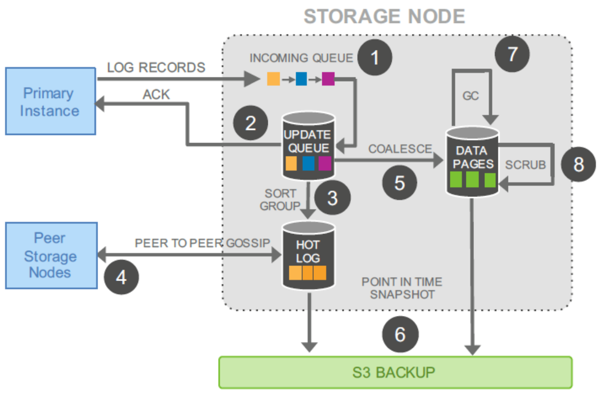

Fig. 3. IO Traffic in Aurora Storage Nodes

Figure 3 shows various activities on storage nodes. (1) Storage node get redo log records from database engine and adds them to an in-memory queue. Then, (2) it persists a record in disk and acknowledges the database engine. These are the only steps that run in foreground (visible to database engine). All other activities are handled in background and does not affect delay. (3) The node then organizes the logs and looks for missing logs using sequence numbers of the logs. Nodes have a peer-to-peer gossip mechanism to recover missing logs (4). Logs are then processed to construct data pages (5). Both the logs and the pages can be backed up (6). Nodes also periodically run garbage collection (7) and page validation (8) mechanisms.

V. Results

Performance and scalability of Aurora is measured against MySQL using standard database benchmarks. Aurora performed significantly better compared to MySQL as shown in the graphs below.

A. Scaling With Instance Size (CPU and Memory)

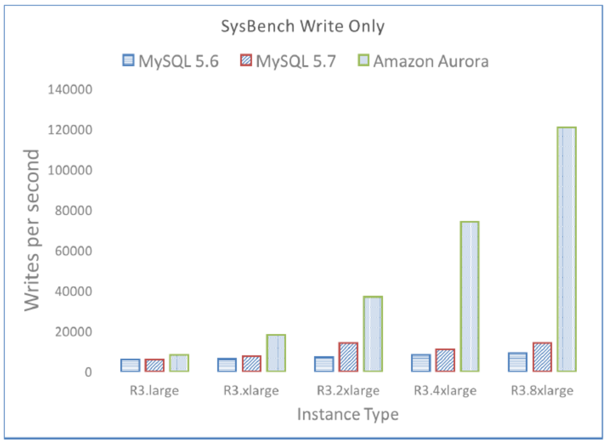

Fig. 4. Comparison of write performance between Aurora and MySQL

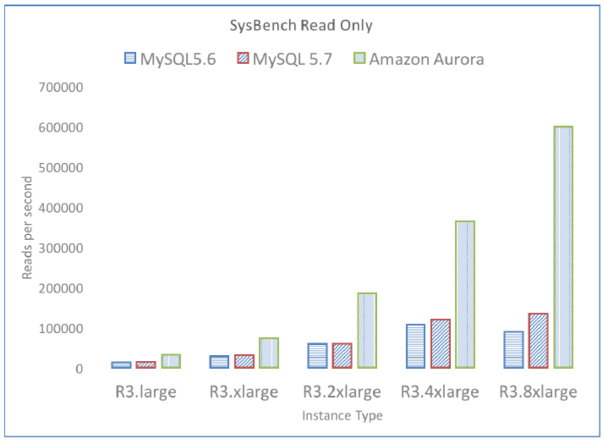

Fig. 5. Comparison of read performance between Aurora and MySQL

Connections	**Amazon Aurora**	**MySQL**
50	40000	10000
500	71000	21000
5000	110000	13000

B. Scaling with User Connections

References

[1] A. Verbitski, X. Bao, A. Gupta, D. Saha, M. Brahmadesam, K. Gupta, R. Mittal, S. Krishnamurthy, S. Maurice, and T. Kharatishvili, "Amazon aurora: Design considerations for high throughput cloud-native relational databases," 05 2017, pp. 1041–1052.