CS 764: Topics in Database Management Systems
Lecture 24: Amazon Aurora

Xiangyao Yu
11/28/2022
Announcement

Project report (DDL: Dec. 19)

- Sample reports available from the course website
- 5–7 pages sufficient. Content is more important than length
- Submit to the Hotcrp website (like the proposal)
Amazon Aurora: Design Considerations for High Throughput Cloud-Native Relational Databases

Alexandre Verbitski, Anurag Gupta, Debanjan Saha, Murali Brahmadesam, Kamal Gupta, Raman Mittal, Sailesh Krishnamurthy, Sandor Maurice, Tengiz Kharatishvili, Xiaofeng Bao

ABSTRACT
Amazon Aurora is a relational database service for OLTP workloads offered as part of Amazon Web Services (AWS). In this paper, we describe the architecture of Aurora and the design considerations leading to that architecture. We believe the central constraint in high throughput data processing has moved from compute and storage to the network. Aurora brings a new architecture to the relational database to address this constraint, most notably by pushing redo processing to a multi-tenant state out storage service, purpose-built for Aurora. We describe how doing so not only reduces network traffic, but also allows for crash recovery, failovers to replicas without loss of data, fault-tolerant, self-healing storage. We then describe how Aurora achieves consensus on durable state across numerous stores.

Amazon Aurora development team wins the 2019 ACM SIGMOD Systems Award*
By Werner Vogels on 04 July 2019 10:00 AM | Permalink | Comments (2)
Cloud Database Architecture

On-premises
- Fixed and limited hardware resources
- Shared-nothing architecture

Cloud
- Virtually infinite computation & storage, Pay-as-you-go price model
- Disaggregation architecture
Feature 1: Computation and storage layers are disaggregated
  • Autoscaling computation and storage nodes
Feature 2: Limited computation can happen in the storage layer
  • REDO processing

Disadvantage: Network bottleneck
  • Lower bandwidth and higher latency
What functions to push to the storage layer?

- Concurrency control
- Indexing
- Buffer manager
- Logging
Computation Pushdown in Cloud OLTP

What functions to push to the storage layer?

- Concurrency control
- Indexing
- Buffer manager
- Logging

Push redo processing into the storage service
Aurora – Single Master
Quorum-Based Voting Protocol

Data replicated into $V$ copies

A write must acquire votes from $V_w$ copies
A read must acquire votes from $V_r$ copies

\[ V_r + V_w > V \]
\[ V_w > V / 2 \]

Copy 1  Copy 2  Copy 3
Quorum-Based Voting Protocol

Data replicated into $V$ copies

A write must acquire votes from $V_w$ copies
A read must acquire votes from $V_r$ copies

$V_w + V_w > V \implies V_w > V/2$
$V_r + V_w > V$

For three copies:

$V = 3.$

$V_w = 3.$

For $V_r$:

$V_r \geq 2$

$V_r = 1$
Quorum-Based Voting Protocol

Data replicated into $V$ copies

A write must acquire votes from $V_w$ copies
A read must acquire votes from $V_r$ copies

\[ V_w + V_w > V \implies V_w > V / 2 \]
\[ V_r + V_w > V \]

For three copies
\[ V_w \geq 2 \]
\[ V_r \geq 2 \]

For six copies
\[ V_w \geq 4 \]
\[ V_r \geq 3 \]
3-Way Replication

AZ: Availability zone
- AZs fail independently

Data is unavailable if one AZ is unavailable and one other copy is unavailable
6-Way Replication

Can read if one AZ fails and one more node fails (AZ+1)
  - Allow to rebuild a write quorum by adding additional replica

Can write if one AZ fails
Segmented Storage

Availability is determined by
- MTTF: Mean time to failure
- MTTR: Mean time to repair

Maximize availability
=> Minimize MTTR (MTTF is hard to reduce)

Segment: 10 GB block. Basic unit of failure and repair
Protection Group (PG): Six replication copies of a segment
Network IO in MySQL

IO traffic
- REDO Log
- Binary log
- Data
  - Double-write
  - metadata (FRM)

Latency
- Steps 1, 3, and 5 are sequential and synchronous
Binary Log vs. REDO Log in MySQL

1. REDO log generated by InnoDB; Binlog generated by MySQL and supports other storage engines
2. REDO log is physical, Binlog can be either physical or logical
3. A transaction writes a single Binlog record but potentially multiple REDO records
MySQL vs. Aurora

MySQL: DB writes both log and data pages to storage
Aurora: DB writes only REDO log to storage
  • The storage layer replays the log into data pages
# MySQL vs. Aurora – Network IO

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Transactions</th>
<th>IOs/Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mirrored MySQL</td>
<td>780,000</td>
<td>7.4</td>
</tr>
<tr>
<td>Aurora with Replicas</td>
<td>27,378,000</td>
<td>0.95</td>
</tr>
</tbody>
</table>
Only Steps 1 & 2 are in the foreground path
Identify gaps in the log
Gossip with peers to fill gaps
Coalesce log records into data pages
Periodically stage log and pages to S3
Periodically garbage collect old versions and periodically validate CRC code on pages

* Cyclic redundancy check (CRC) is an error-detecting code
Forward Processing – Write and Commit

Write: flush REDO log to storage
Commit: after all the log records are properly flushed
Forward Processing – Read

Buffer hit: read from main memory of the DB server
Buffer hit: read from main memory of the DB server
Buffer miss: read page from storage
Forward Processing – Eviction

Buffer hit: read from main memory of the DB server
Buffer miss: read page from storage
Dirty eviction: discard dirty page (no write back to storage)
  • The page in storage will be updated through replaying the REDO log

Aurora: discard dirty page
MySQL: evict dirty page to storage
Three votes to read data

The DB server knows which node contains the latest value

=> A single read from the update-to-date node
If page is in replica’s local buffer, update the page
Otherwise, discard the log record
Evaluation – Aurora vs. MySQL

 SysBench Write Only

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>MySQL 5.6</th>
<th>MySQL 5.7</th>
<th>Amazon Aurora</th>
</tr>
</thead>
<tbody>
<tr>
<td>R3.large</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3.xlarge</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3.2xlarge</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3.4xlarge</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3.8xlarge</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

 SysBench Read Only

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>MySQL 5.6</th>
<th>MySQL 5.7</th>
<th>Amazon Aurora</th>
</tr>
</thead>
<tbody>
<tr>
<td>R3.large</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3.xlarge</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3.2xlarge</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3.4xlarge</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3.8xlarge</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Evaluation – Varying Data Sizes

Table 2: SysBench Write-Only (writes/sec)

<table>
<thead>
<tr>
<th>DB Size</th>
<th>Amazon Aurora</th>
<th>MySQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 GB</td>
<td>107,000</td>
<td>8,400</td>
</tr>
<tr>
<td>10 GB</td>
<td>107,000</td>
<td>2,400</td>
</tr>
<tr>
<td>100 GB</td>
<td>101,000</td>
<td>1,500</td>
</tr>
<tr>
<td>1 TB</td>
<td>41,000</td>
<td>1,200</td>
</tr>
</tbody>
</table>

Performance drops when data does not fit in main memory
Evaluation – Real Customer Workloads

Aurora 3X faster on r3.4xlarge

Before: 15ms
After: 5.5ms

Aurora Migration
Evaluation – Real Customer Workloads

Figure 9: SELECT latency (P50 vs P95)

Figure 10: INSERT per-record latency (P50 vs P95)
Aurora Multi-Master

Any DB instance can access any data
The storage nodes detect conflicts at page granularity
  • Pushing down concurrency control to the storage layer

Aurora Serverless

- [https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/aurora-serverless.how-it-works.html](https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/aurora-serverless.how-it-works.html)
How does it work...

- Get server from warm pool
- Transfer buffer pool
- Look for safe scale point
How does it work in practice?

Scales up quickly as load increases

Scales down more gradually
Any pitfalls of this design?

Alternative DBs in industry with innovations different from Aurora?

Does Aurora support geo-replication well?

Network vs. compute vs. storage, which one is the bottleneck?

Aurora depends on MySQL and Postgres; does that hinder its development?

How to handle case where storage node writes data but does not replicate to other replicas?

Is S3 used as WAL in Aurora?
Before Next Lecture

Submit review for