您似乎是从中国境内访问我们的网站的。请导航至我们的优化版网站：amazonaws-china.com。

Amazon Aurora Under the Hood: Reducing Costs Using Quorum Sets

by Anurag Gupta | on 21 AUG 2017 | in Amazon Aurora, Aurora, Database | Permalink | Comments | Share

Anurag Gupta runs a number of AWS database services, including Amazon Aurora, which he helped design. In this under the hood series, Anurag discusses the design considerations and technology underpinning Aurora.

This post is the third in a four-part series discussing how Amazon Aurora uses quorums. I hope the discussion is useful to you as you design your own distributed systems. In this post, we discuss how to manage costs in a quorum system.

The basic problem we're tackling is that Aurora uses a quorum of six copies spread across three Availability Zones (AZs), using four out of six copies to write and three out of six copies for read/repair. In the first post of this series, I discussed why six is the minimum number of copies necessary. In my second post, I discussed how we can avoid the performance penalties of quorums for both writes and reads. But it's still a lot of copies of data, and that carries costs. The low price for storage in Amazon Aurora might make us think that there's something unusual going on. There is.

To understand what we do, you have to go back to the basic definition of a quorum. People generally talk about quorums as a set of like elements where the write set represents a majority of elements, and the read and write sets overlap. Although this is correct, it's a simplification. The basic requirement is only that the read and write sets are subsets of the entire quorum membership set, for any legal write subset, at least one member is also contained within any legal read subset, and that each write subset overlaps with prior write subsets. That seems like the same thing, but it's not.

The difference is that there is no requirement that quorum members be the same as each other. We can construct quorum sets that mix and match quorum subsets that have different latency, cost, or durability characteristics. We can then use the rules of Boolean logic to create more sophisticated read

Create a Free AWS Account

Search

Search

Posts by Product

Amazon Aurora

AWS Database Migration Service (DMS)

Amazon DynamoDB

Amazon EC2

Amazon ElastiCache

Amazon Elasticsearch Service

AWS IOT

Amazon Kinesis

AWS Lambda

Amazon RDS for MySQL

Amazon RDS for Oracle

Mixing full and tail segments of data
In Aurora, a database volume is made up of 10 GB segments of data. These segments are replicated as a protection group, with six copies spread across three AZs. But the six copies are not all the same. Half of the copies are *full segments,* which contain both data pages and log records for that 10 GB portion of the volume. The other half are *tail segments,* which contain only log records. Each AZ contains one full segment and one tail segment.

Most databases have far more data block storage than redo log storage. Using a mix of full and tail segments takes the physical storage requirements of Aurora from six times the size of the database to a little bit more than three. For a system that is designed to tolerate "AZ+1" failure cases, that's the minimum replication factor you can have.

The use of a mix of full segments and tail segments changes how we have to construct our read and write sets. You can use the rules of Boolean logic to ensure overlap across subsets and do it accurately even for arbitrarily complex distribution of members. In our case, our write quorum is four out of six of any segment OR three out of three of full segments. Our read quorum is then three out of six of any segment AND one out of three of full segments. You can see from the preceding that we have an overlap on all segments in the quorums as well as an overlap on our full segments. By doing this, we can write log records to the four out of six segments that we did previously. At least one of these is a full segment and generates a data page. We read data from full segments, using the optimization described in the last post to avoid quorum reads, and instead reading from the one we know has the data we need.

We use the read quorum as a way to rebuild failed segments and repair impaired quorums. We also use it to rebuild our local state if we have to restart the database write master node. If one of our tail segments fails, that's easy. We just repair it from any one of the three other copies we know was written, just as we would in a simple quorum model.

If one of our full segments fails, it's a bit more complicated. The one that failed could have been the copy that we wrote to as part of our write. But in that case, we know that we have another full segment, even if it hasn't seen the most recent write. We also have enough copies of the redo log record that we can rebuild a full segment to be up to date. We also gossip between the segments of a quorum to ensure that any missing writes are quickly filled in. This reduces the probability we need to rebuild a full segment without adding a performance burden to our write path.

Controlling costs with quorum sets of unlike members
Using quorum sets of unlike members is a good way to contain costs. There are many options available. You might have quorums that combine the local disk for low latency and remote disks for durability/availability. You might

Amazon RDS for SQL Server

AWS Schema Conversion Tool (SCT)

RSS Feed

Subscribe to this blog's feed

Recent Posts

Introducing Amazon S3 and Microsoft Azure SQL Database Connectors in AWS Database Migration Service

Viewing Amazon Elasticsearch Service Slow Logs

Replicating Amazon EC2 or On-Premises SQL Server to Amazon RDS for SQL Server

Querying on Multiple Attributes in Amazon DynamoDB

Automating Cross-Region and Cross-Account Snapshot Copies with the Snapshot Tool for Amazon Aurora

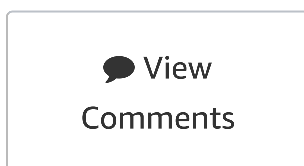
Automating SQL Caching for Amazon

combine data across AWS Regions to improve disaster recovery. There are a lot of moving parts that you need to get right, but the payoffs can be significant. For Aurora, the quorum set model described earlier lets us achieve storage prices comparable to low-cost alternatives, while providing high durability, availability, and performance.

So far in this series, we've discussed how to size quorums, how to avoid the penalty of read and write amplification, and, with this post, how to control costs. In the next post, we talk about how to manage quorum membership in a large-scale distributed system in the face of failures without expensive coordination.

If you have questions, leave a comment here or ping us at aurora-pm@amazon.com.

Read Next: Amazon Aurora Under the Hood: Quorum Membership



Migrating a SQL Server Database to a MySQL-Compatible Database Engine

Using Amazon Redshift for Fast Analytical Reports

Testing Amazon RDS for Oracle: Plotting Latency and IOPS for OLTP I/O Pattern

Get Started with Amazon Elasticsearch Service: Filter Aggregations in Kibana

Useful Documentation Links

- Cloud Databases with AWS
- Amazon RDS
- AWS Database Migration Service
- Amazon DynamoDB
- Amazon ElastiCache
- Amazon Redshift

AWS Blogs

- AWS Blog
- AWS Big Data