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: quorums and correlated failure

by Anurag Gupta | on 14 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.

Amazon Aurora storage is a highly distributed system that needs to meet the stringent performance, availability, and durability requirements of a high-end relational database. This post is the first of a four-part series that covers some of the key elements of our design.

There isn't a lot of publicly available material discussing tradeoffs in real-world durability, availability, and performance at scale. Although this series is based on the considerations involved in designing a transactional database, I believe it should be relevant to anyone architecting systems involving the coordination of mutable distributed state.

In this first post, I discuss how we arrived at the decision to use quorums for Aurora storage and why we distribute six copies of data across three Availability Zones (AZs). Some of this material is also discussed in our recent SIGMOD paper.

Why distributed storage is a good idea, but hard to do well
Let's first discuss why distributed storage is a good idea. It's easy to make a database run fast by collocating both the database software and the storage on a single box. The problem is that boxes fail. It takes time to recover from a backup after a failure. Many systems can't tolerate losing recent data that hasn't been backed up yet.

Beyond accounting for failures, separating the database instance from its storage improves flexibility. Customers shut databases down. They size them up and down. They add and remove read replicas. Decoupling the storage from the database makes these operations easy, since the underlying storage

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

Of course, moving storage away from compute just creates a dependency on still more devices that can independently fail. That's why people use replication—either synchronous or asynchronous. If failures are independent, then replication improves durability.

But replication has its own problems. In synchronous replication, all copies must acknowledge before you consider a write to be durable. This approach puts you at the mercy of the slowest disk, node, or network path. Asynchronous replication improves latency, but can result in data loss if there's a failure before data is replicated and made durable. Neither option is attractive. Failures require changes to the replica membership set. This approach is also awkward. Recreating a dropped replica is expensive, so people generally are conservative doing so. This conservatism means that you might see a few minutes of unavailability before the replica is fenced off.

A quorum model

Aurora instead uses a quorum model, where you read from and write to a subset of copies of data. Formally, a quorum system that employs V copies must obey two rules. First, the read set, V_r, and the write set, V_w, must overlap on at least one copy.

This approach means that if you have three copies of data, the read set and the write set can be two, ensuring each sees the other. This rule ensures that a data item is not read and written by two transactions concurrently. It also ensures that the read quorum contains at least one site with the newest version of the data item.

Second, you need to ensure that the quorum used for a write overlaps with prior write quorums, which is easily done by ensuring that $V_w > V/2$. This rule ensures that two write operations from two transactions cannot occur concurrently on the same data item. Here are some possible quorum models.

V (#copies)	**V_w (write quorum)**	**V_r (read quorum)**
1	1	1
2	2	1
3	2	2
4	3	2
5	3	3
6	4	3
7	4	4

Quorum systems have some nice properties. They can deal with the long-term failure of a node as easily as they deal with a transient failure (for example, due to a reboot) or slowness of one of the participants.

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

of four and a read set of three. We issue writes to all six copies of data and acknowledge the write as complete once we obtain an acknowledgement from four of the six copies. If one of the nodes is running slow, it's fine—the others respond quickly and this node catches up when it can. If one of the nodes is briefly unavailable, it's also fine—there's no loss of write or read availability and the node continues accepting requests when back up. And, if the node is permanently down, we'll figure out that it hasn't responded for a while and introduce a new member to the quorum set using a membership change protocol.

So why six copies? The preceding statements are true even with 2/3 quorum set, which is popular in many production distributed systems. Such systems can transparently handle one fault. They rely on it being extremely unlikely that two independent faults could occur during the time it takes to repair one of them.

The problem with 2/3 quorums is that not all faults are independent. Let's say you had a 2/3 quorum with one copy of data in each of three AZs. In a large distributed system, such as the ones that we operate in the AWS Cloud, there's a continuous low-level background noise of node, disk, and network path failures. These are, in and of themselves, entirely fine. A 2/3 quorum tolerates these failures transparently. The background noise is low enough that it is extremely unlikely that we would see two faults at the same time.

However, the reason we separate AWS Regions into AZs is to create areas of fault isolation. Let's imagine that one of the three AZs in an AWS Region goes down. It might be down permanently, because of a roof collapse, fire, flood, tornado, or similar. Or it might be down for a short period of time because of a power outage, bad software deployment, or similar.

That failure causes one copy to be lost in each and every quorum at the same time. The small number of quorums that were already handling a fault now have a dual fault. At that point, you only have 1/3 copies readable, and can't ensure that this copy has seen all writes. This copy might have been the one that was skipped when the other two were written. In such a case, the quorum is not writable, readable, or repairable, and the database volume is lost.

A six-copy quorum model can tolerate losing an entire AZ without losing write availability and is able to lose an AZ plus one additional fault without losing data. As long as you have valid read quorum, you can rebuild additional copies of data to obtain a full repaired quorum. It's easy to see that an "AZ+1" fault model requires a minimum of three AZs and two copies in each of those three AZs. You can run a 3/4 quorum or a 3/5 quorum and still meet the "AZ+1" objective, but only in environments with four or five independent AZs in the region.

Are six copies sufficient?
Six copies are necessary, but are they sufficient? Reasoning about this question

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

availability. In a six-copy quorum model, losing read availability means losing four of the six copies of data—either with four independent failures, two independent failures and an AZ failure, or two independent AZ failures. The most likely of these is that one had a failed node, then a full AZ failure, then another node died while we were in the midst of repairing the first failed node.

That's unlikely, but for a bad enough MTTF and MTTR, it can happen. Past a point, it is hard to improve MTTF and the likelihood of independent failures. So our best bet is to reduce MTTR.

In Aurora, we do this by breaking up the database volume into 10 GB chunks, each replicated independently into protection groups using six copies. A large database volume might spread over thousands of nodes. At 10 GB, on a 10 Gbit network, it takes under a minute to repair quorum. Even allowing for detection time and hysteresis to avoid repairing a transient issue, MTTR only is a few minutes. Because the other failures are independent of this one, it's unlikely that you'll see three additional independent failures or an AZ failure and one additional failure in this time frame. The probability is so low that you can even introduce faults. Software deployments to the storage tier are straightforward. You can simply stop a node, install software, and restart—the system transparently accommodates this fault and can absorb yet more.

This approach also helps with heat management. You can simply mark a segment on a hot disk or node as dead, and it is automatically repaired onto another node in the storage fleet.

But what if we actually had a fire or flood and lost an AZ for months while we rebuilt? In that case, we'd have lost two of our six copies, and any additional double-fault or loss of an AZ loses our database volumes. Call us paranoid, but we actually worry about things like this—the MTTR to rebuild an AZ after a catastrophic failure is high.

We recently deployed software to introduce a degraded mode for such cases. In this mode, we can behind the scenes reduce to a 3/4 write quorum and 2/4 read quorum on the long-term loss of an AZ. We can then repair back to a full six-copy 3-AZ quorum once it is again available. This approach allows us to repair from a transient loss of one of the remaining AZs and to tolerate one additional failure without losing write availability.

In the next few posts, I'll discuss how Aurora deals with the drawbacks of the approach sketched preceding:

- Performance (quorum reads are slow)
- Cost (six copies are expensive)
- Availability (membership changes are expensive when you break up a volume into small chunks)