amazon Aurora

Presented by Jiaxiao Zhou (周佳孝 Mossaka)

Microsoft

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

Amazon Web Services

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 novel architecture to the relational database to address this constraint, most notably by pushing redo processing to a multi-tenant scale-out storage service, purpose-built for Aurora. We describe how doing so not only reduces network traffic, but also allows for fast crash recovery, failovers to replicas without loss of data, and fault-tolerant, self-healing storage. We then describe how Aurora achieves consensus on durable state across numerous storage nodes using an efficient asynchronous scheme, avoiding expensive and chatty recovery protocols. Finally, having operated Aurora as a production service for over 18 months, we share lessons we have learned from our customers on what modern cloud applications expect from their database tier.

Keywords

Databases; Distributed Systems; Log Processing; Quorum Models; Replication; Recovery; Performance; OLTP

1. INTRODUCTION

IT workloads are increasingly moving to public cloud providers. Significant reasons for this industry-wide transition include the ability to provision capacity on a flexible on-demand basis and to pay for this capacity using an operational expense as opposed to capital expense model. Many IT workloads require a relational OLTP database; providing equivalent or superior capabilities to on-premise databases is critical to support this secular transition.

In modern distributed cloud services, resilience and scalability are increasingly achieved by decoupling compute from storage

The I/O bottleneck faced by traditional database systems char in this environment. Since I/Os can be spread across many nc and many disks in a multi-tenant fleet, the individual disks nodes are no longer hot. Instead, the bottleneck moves to network between the database tier requesting I/Os and the stor tier that performs these I/Os. Beyond the basic bottlenecks packets per second (PPS) and bandwidth, there is amplificatio traffic since a performant database will issue writes out to storage fleet in parallel. The performance of the outlier stor node, disk or network path can dominate response time.

Although most operations in a database can overlap with e other, there are several situations that require synchron operations. These result in stalls and context switches. One s situation is a disk read due to a miss in the database buffer ca A reading thread cannot continue until its read completes. A ca miss may also incur the extra penalty of evicting and flushir dirty cache page to accommodate the new page. Backgro processing such as checkpointing and dirty page writing reduce the occurrence of this penalty, but can also cause st context switches and resource contention.

Transaction commits are another source of interference; a stal committing one transaction can inhibit others from progress Handling commits with multi-phase synchronization proto such as 2-phase commit (2PC) [3][4][5] is challenging in a clo scale distributed system. These protocols are intolerant of fai and high-scale distributed systems have a continual "backgro noise" of hard and soft failures. They are also high latency high scale systems are distributed across multiple data centers.

Alex Verbitski
Principal SDE at Amazon

Experience

Principal SDE
Amazon
Mar 2013 – Present · 8 yrs
Greater Seattle Area

http://aws.amazon.com/rds/aurora/
Do you want to start building vNext? We have number of open positions.

Microsoft
15 yrs

SDEII->Senior SDE->Principal SDE in the Microsoft SQL Server Engine Team
2002 – Feb 2013 · 11 yrs
Redmond, WA

Have worked on SQL Server releases since SQL 2000 64bit all the way to SQL 2012+

Areas owned/worked on: ...see more

SDET -> Lead SDET
1998 – 2002 · 4 yrs
Redmond, WA

Worked on Real Time Communication capabilities present in the Windows 2000/XP/WS2003.
For some time I was also responsible for the interoperability aspects of Microsoft real time collaboration software. Have participated in sevei ...see more

I am not an expert on this topic
I am just a learner

Agenda

1. Why reading Aurora paper?
2. Background knowledge
3. History of AWS
4. Why traditional MySQL replicated storage failed?
5. Offloading redo processing
6. Quorum
7. To be continued next week!

Why Reading Aurora Paper

1. Because its from Amazon
2. Table 1: Network IOs for Aurora vs MySQL

Configuration	Transactions	IOs/Transaction
Mirrored MySQL	780,000	7.4
Aurora with Replicas	27,378,000	0.95

3. Some interesting quotes:
    a. "*We believe the central constraint in high throughput data processing has moved from compute and storage to the network*"
    b. "*as far as the engine is concerned, the log is the database*"

What is Aurora?

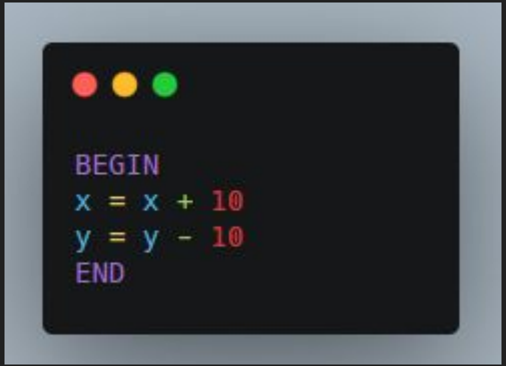
A high throughput <u>OLTP</u> (online transactional processing) database that compromises neither availability and durability in a cloud-scale environment (AWS)

What is OLTP?

Reference: What is OLTP? | IBM

Transaction💶: a business deal

Transaction processing 🏧: electronic fund transfer



Characteristics:

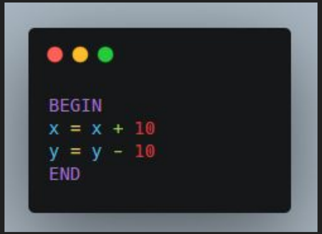
1. **Available 24/7/365**
2. **Small work units (INSERT, UPDATE, QUERY, DELETE)**
3. **High throughput (#transactions per second TPS)**
4. **High Concurrency**

OLTP Desirable Properties

ACID

- *Atomicity:* all performed, or none performed
- *Consistency:* data is in consistent state when a transaction starts and when it ends.
- *Isolation:* transaction's updates are not visible to other transactions before commit
- *Durability:* committed changes to DB are permanent.

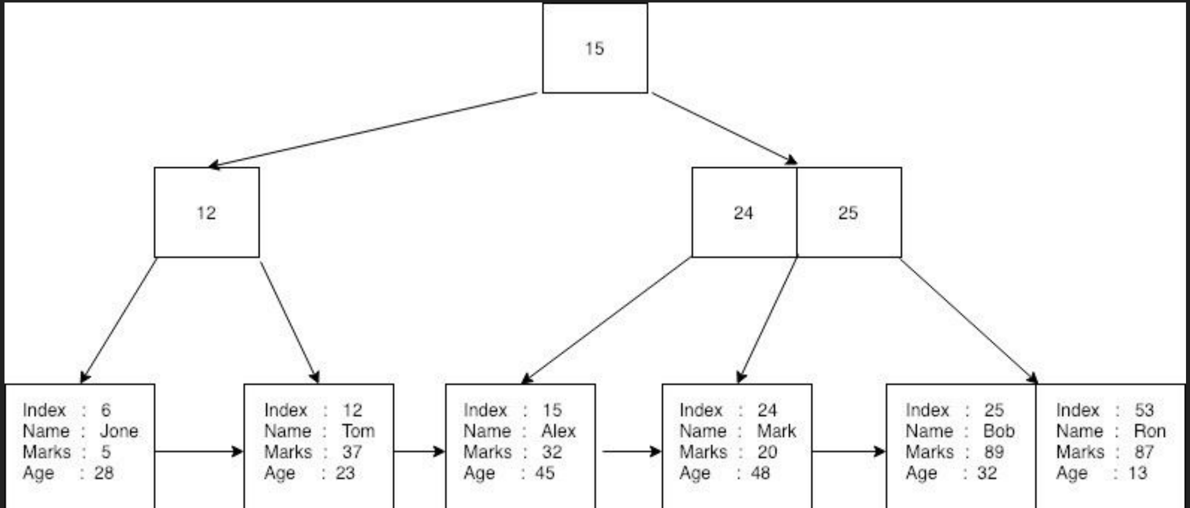
Generic Transactional DB



When a transaction comes in:

The DB locks `x` and `y`, release after commit finishes

DB stores data in database pages using B+tree for indexing. (reference)



Generic Transactional DB

DB stores data in database pages using B+tree for indexing. (reference)

DB caches database pages

DB uses Write-Ahead logging (WAL) for providing atomicity and durability. The changes are first recorded in the log, persist to a stable storage, and then commit and write to the database. (reference)

WAL enables crash-recovery

1. Replay “new” values for all committed Ts in log (redo)
2. Replay “old” values for uncommitted Ts in log (undo)

Once committed, DB release locks, and reply to client

What is AWS?

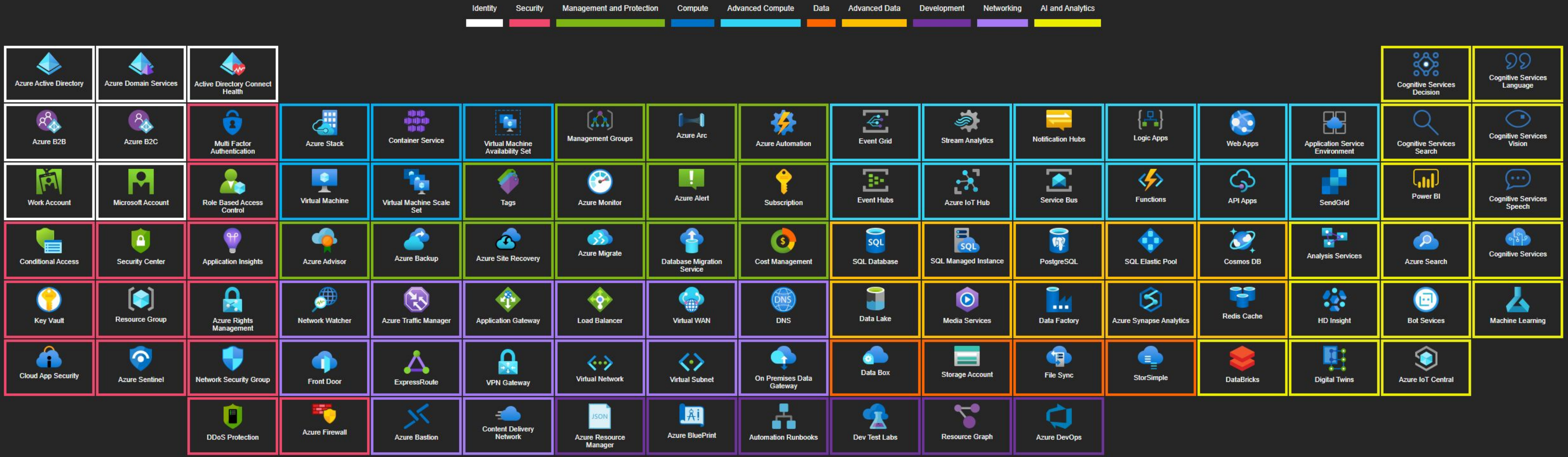
[Jerry Hargrove - AWS History (awsgeek.com)]

Service Name	Category	Announced	Released
Amazon **Simple Storage Service (S3)**	Storage	13-Mar-2006	13-Mar-2006
Amazon **Simple Queue Service (SQS)**	Application Integration	03-Nov-2004	11-Jul-2006
Amazon **SimpleDB**	Database	13-Dec-2007	
Amazon **Elastic Block Store (EBS)**	Storage	20-Aug-2008	20-Aug-2008
Amazon **EC2**	Compute	24-Aug-2006	23-Oct-2008
Amazon **EMR**	Analytics	02-Apr-2009	
Amazon **CloudWatch**	Management & Governance	17-May-2009	
Elastic Load Balancing (ELB)	Networking & Content Delivery	18-May-2009	18-May-2009
Amazon **Simple Notification Service (SNS)**	Application Integration	07-Apr-2010	
Amazon **CloudFront**	Networking & Content Delivery	18-Nov-2008	09-Nov-2010
Amazon **Route 53**	Networking & Content Delivery	06-Dec-2010	06-Dec-2010
AWS **Elastic Beanstalk**	Compute	19-Jan-2011	
Amazon **Simple Email Service (SES)**	Customer Engagement	25-Jan-2011	
AWS **CloudFormation**	Management & Governance	25-Feb-2011	25-Feb-2011
AWS **Identity & Access Management**	Security, Identity & Compliance	02-Sep-2010	03-May-2011
AWS **Direct Connect**	Networking & Content Delivery	03-Aug-2011	03-Aug-2011
Amazon **VPC**	Networking & Content Delivery	26-Aug-2009	03-Aug-2011
Amazon **ElastiCache**	Database	22-Aug-2011	
Amazon **DynamoDB**	Database	18-Jan-2012	18-Jan-2012

What is AWS?

[Jerry Hargrove - Periodic Table of Amazon Web Services (awsgeek.com)](https://awsgeek.com)

Azure Periodic Table | Data#3



History led to Aurora

Cloud VMM for renting.

Great for hosting web sites

Not ideal for MySQL

- Limited scaling options
- Limited fault-tolerance

Chain replication (pair of 2 servers) (reference)

Used as a block-storage service for EC2 and will be available if an EC2 instance crashes.

Not ideal either: not fault-tolerant enough (what if an entire datacenter went down?), network load is not efficient.

AWS Regions and AZ

AWS Region is a separate geographic area.

AWS Region provides complete isolation

AWS Region has multiple availability zones

US East (Ohio) Region
Availability Zones: 3
Launched 2016

US West (Oregon) Region
Availability Zones: 4
Launched 2011
Local Zones: 2
Launched 2019

“In modern distributed cloud services, resilience and scalability are increasingly achieved by decoupling compute from storage and by replicating storage across multiple node”

Replicated Storage for MySQL (RDS)

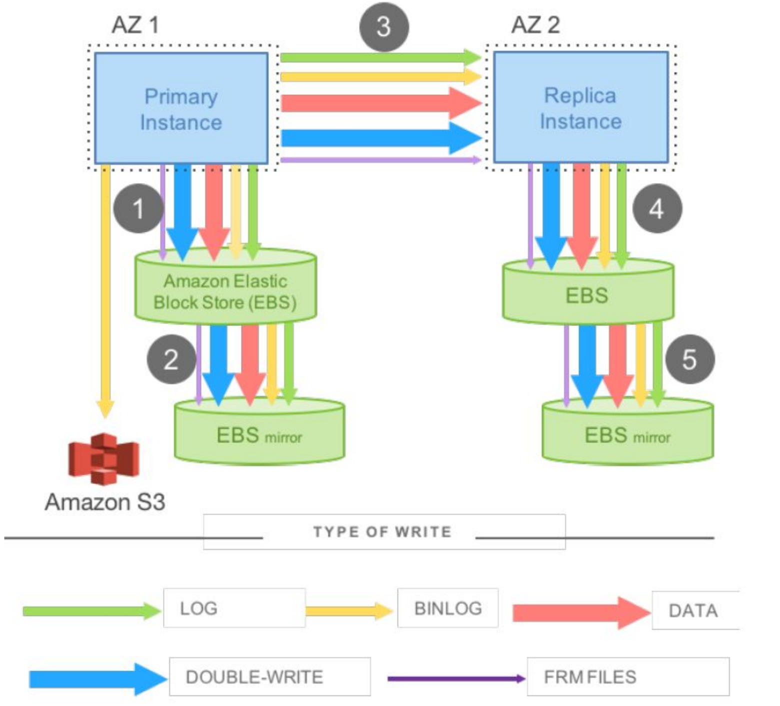


Figure 2: Network IO in mirrored MySQL

RDS

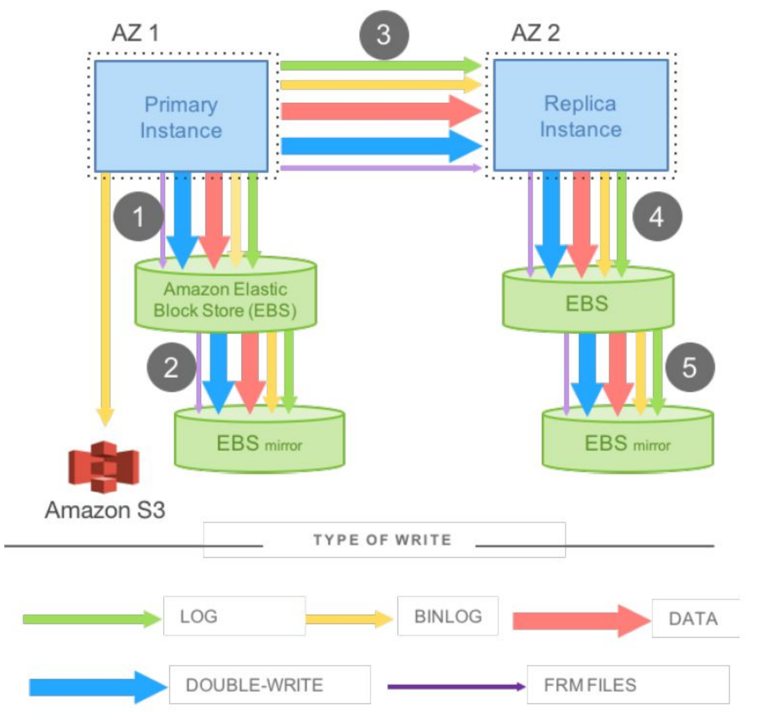


Figure 2: Network IO in mirrored MySQL

The burden of amplified writes

Problem: The mirrored MySQL sends many bytes of date over the network. (Each data page is 8KB)

MySQL thinks it is writing to a local disk.

How does Aurora deal with it🌟

Aurora’s solution

Offloading Redo Processing to Storage

The storage servers store the “data pages” in B+tree that make up the tables and indices.

Traditional: reading old data from storage, modifying them, and writing back entire pages.

Aurora’s solution: sends just the log records to storage servers. This offloads interpreting the log entries and modifications to background.

Aurora's solution

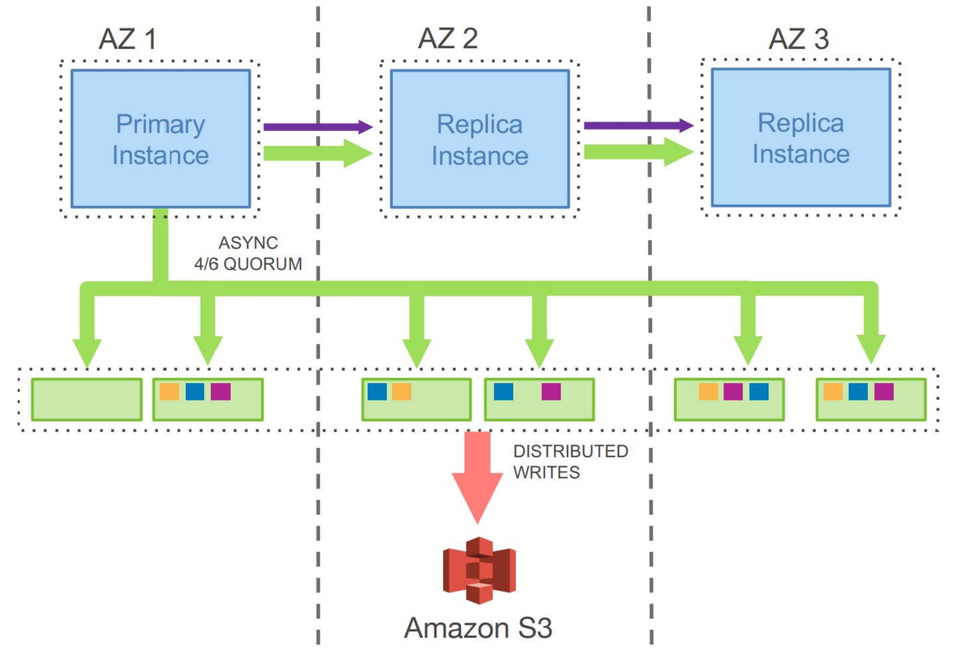


Figure 3: Network IO in Amazon Aurora

Now, storage server needs replication.

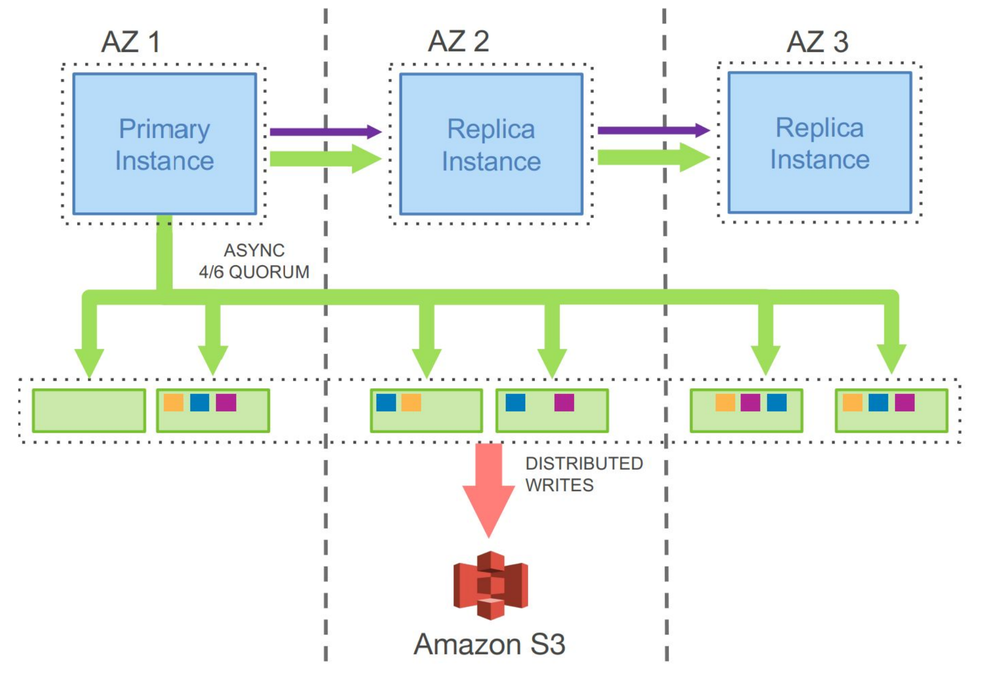


Figure 3: Network IO in Amazon Aurora

Aurora does 6 replicas!!

Quorum-based voting protocol

Goal: being able to read latest data even if some failures

Suppose we have N replicas, R read replicas and W write replicas.

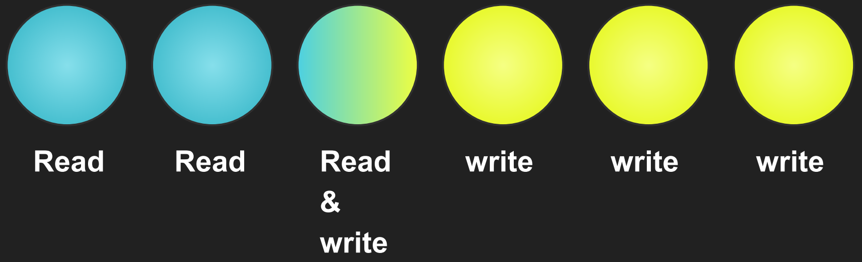
We want to make sure R + W = N + 1

Quorum in Aurora

N = 6, R = 3, W = 4

This means Aurora can tolerant 1 AZ write failure, 1AZ + 1 read failure

Compare to CRAQ: no need to wait for failures



Note

Since database server and storage are decoupled, we were only talking about storage fault-tolerance.

What if the database server is down?

Maybe a new EC2 server will automatically start and recovers from log + data on the storage servers.

We briefly touched Aurora's two BIG ideas 💥

1. Quorum writes for better fault-tolerance without too much waiting (Unlike CRAQ)
2. Offloading repo processing to Storage Servers - Sending to many replicas, but not much data!

Table 1: Network IOs for Aurora vs MySQL

Configuration	Transactions	IOs/Transaction
Mirrored MySQL	780,000	7.4
Aurora with Replicas	27,378,000	0.95

Next week 

What does Aurora quorum write look like?

What does Aurora quorum red look like?

What does Storage Server look like?

How to do quick crash recovery?