CLOUD COMPUTING APPLICATIONS

Cloud Databases (DBaaS)

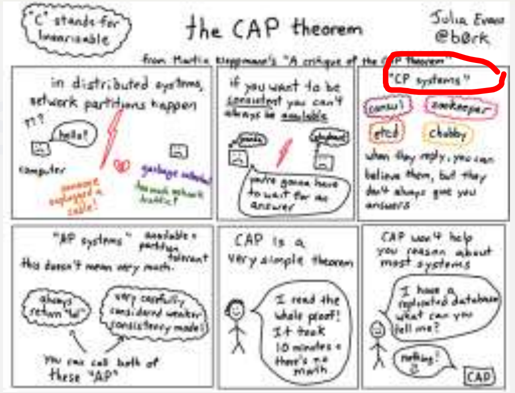
Prof. Reza Farivar

Overview

- OLTP, SQL
  - Traditional SQL RDBMS
  - Cloud Optimized
- NewSQL
  - Spanner
  - CosmosDB
- NoSQL
  - Key/Value
  - Wide-Column
  - Document
  - Graph
  - In-Memory

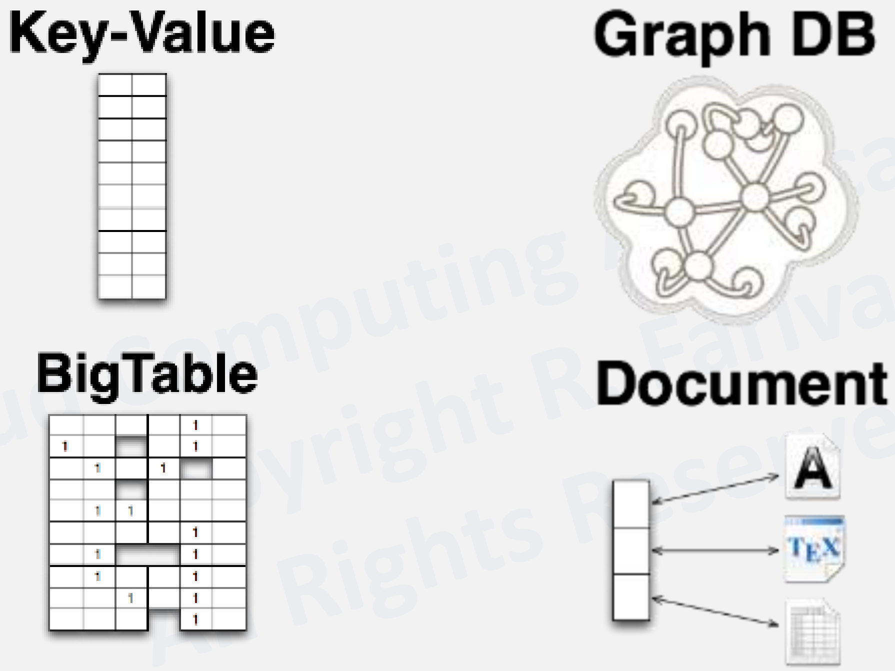
NewSQL

- NewSQL is a class of relational database management systems
  - Scalability of NoSQL systems for online transaction processing (OLTP) workloads
  - ACID guarantees of a traditional database system
- SQL as their primary interface
- Components
  - Distributed concurrency control
  - Flow control
  - Distributed query processing
- Automatically split databases across multiple nodes using Raft or Paxos consensus algorithm
- Consistency over availability (CP from CAP)
  - Strong consistency
  - Sacrificing some availability



"C" stands for linearizable
the CAP theorem
Julia Evans @b0rk
from Martin Kleppmann's "A critique of the CAP theorem"
in distributed systems, network partitions happen
hello?
computer
garbage collection
someone unplugged a cable!
too much network traffic?
if you want to be consistent you can't always be available
you're gonna have to wait for an answer
"CP systems"
consul
zookeeper
etcd
chubby
when they reply, you can believe them, but they don't always give you answers
"AP systems" available + partition tolerant
this doesn't mean very much.
always return "lol"
very carefully considered weaker consistency model!
you can call both of these "AP"
CAP is a very simple theorem
I read the whole proof! It took 10 minutes & there's no math
CAP won't help you reason about most systems
I have a replicated database what can you tell me?
nothing!
CAP

Four NoSQL Categories



CLOUD COMPUTING APPLICATIONS

Cloud Databases – Managed RDBMS

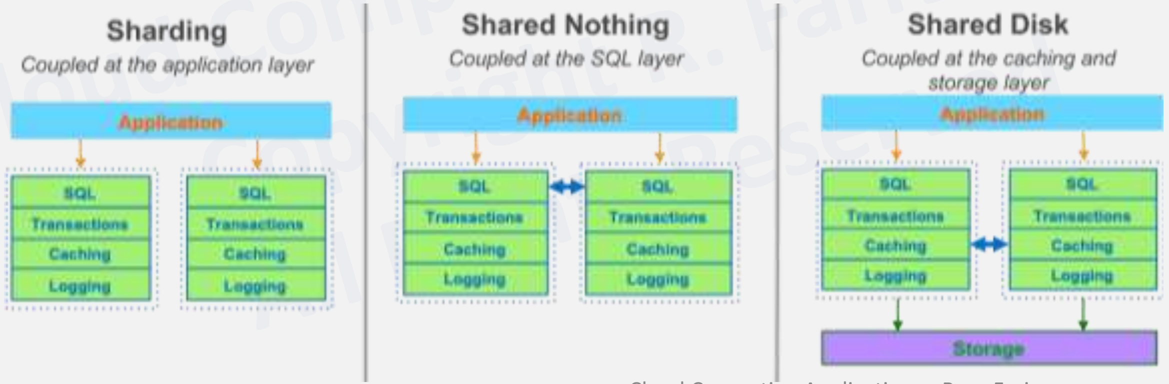
Prof. Reza Farivar

Relational Cloud Databases

- For decades, managing a relational database has been a high-skill, labor-intensive task
- Relational databases store data with predefined schemas and relationships between them
- These databases are designed to support ACID transactions, maintain referential integrity and strong data consistency.
  - Atomicity
  - Consistency
  - Isolation
  - Durability
- OLTP workloads
  - On- Line Transactional Processing (OLTP)

Relational Databases

- In large systems, failures are a norm, not an exception
- Users want to start with a small footprint and then grow massively without infrastructure limiting their velocity
- Replication
  - Storage (SAN, NAS, Aurora)
  - Database
  - Application



Sharding

- Sharding: Split data set by certain criteria and store such “shards” on separate “clusters”
  - Sharding can be considered an embodiment of the "share-nothing" architecture and essentially involves breaking a large database into several smaller databases
- One common way to split a database is splitting tables that are not joined in the same query onto different hosts
- Another method is duplicating a table across multiple hosts and then using a hashing algorithm to determine which host receives a given update

Managed Relational Databases

- Traditional single server databases running on a virtual machine
  - AWS RDS: MySQL, PostgreSQL, MariaDB, Oracle, MS SQL server
  - Azure SQL Database, Database for MySQL, PostgreSQL, MariaDB
  - Google Cloud SQL
  - IBM Cloud Databases for PostgreSQL, DB2 on cloud
- Instances are fully managed, relational MySQL, PostgreSQL, and SQL Server databases
- Cloud provider handles replication, patch management, and database management to ensure availability and performance
- Availability through failover
- Horizontal Scalability through read replicas
  - Vertical scalability by using larger machines (64 processors, 400GB RAM)

Managed Relational Databases

- Typically the database instance is accessible by most compute resources in the cloud provider's network
  - Virtual Machines (AWS EC2, Azure VMs, Google Compute Engine)
  - PaaS (Aws Elastic beanstalk, Google App Engine
  - Serverless (AWS lambda, Google Cloud Functions, Azure functions, etc.)
- Over the internet
  - SQL Proxy
  - Google Cloud SQL Proxy for public interfacing
    - ./cloud_sql_proxy -instances=***INSTANCE_CONNECTION_NAME***=tcp:3306 &
    - import pymysql
      connection = pymysql.connect(host='127.0.0.1',
      user='***DATABASE_USER***',
      password='***PASSWORD***',
      db='***DATABASE_NAME***')
- Encryption at rest and in transport

CLOUD COMPUTING APPLICATIONS

Cloud Databases – Multinode RDBMS

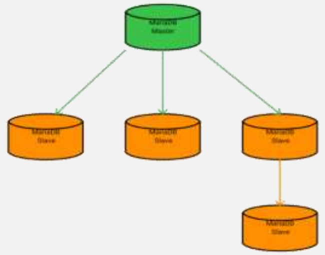
Prof. Reza Farivar

Beyond a Single Node

- Replication Options in MySQL
  - Classical MySQL Replication
    - One master, multiple slaves
    - Support for Many masters to many slaves
    - asynchronous
    - No Conflict Resolution or “Protection”
  - MySQL Group Replication
    - distributed state machine replication with strong coordination between servers
    - Built on Paxos
      - Majority vote for transaction commit
      - Network partition can stop the system
  - Multi Master - Galera
    - Multi-master
  - MySQL (NDB) Cluster
    - Synchronous

Replication in Databases

- Replication is a feature allowing the contents of one or more servers (called masters) to be mirrored on one or more servers (called slaves)
- **Scalability:** By having one or more slave servers, reads can be spread over multiple servers, reducing the load on the master.
  - The most common scenario for a high-read, low-write environment is to have one master, where all the writes occur, replicating to multiple slaves, which handle most of the reads.
- **Backup assistance:** Backups can more easily be run if a server is not actively changing the data.
  - A common scenario is to replicate the data to slave, which is then disconnected from the master with the data in a stable state. Backup is then performed from this server.



Replication and Binary Log

- The main mechanism used in replication is the binary log.
  - All updates to the database (data manipulation and data definition) are written into the binary log as binlog events.
  - The binary log contains a record of all changes to the databases, both data and structure, as well as how long each statement took to execute
    - It consists of a set of binary log files and an index
  - This means that statements such as CREATE, ALTER, INSERT, UPDATE and DELETE will be logged, but statements that have no effect on the data, such as SELECT and SHOW, will not be logged
- Slaves read the binary log from each master in order to access the data to replicate.

Database Replication

- A relay log is created on the slave server, using the same format as the binary log, and this is used to perform the replication
  - Old relay log files are removed when no longer needed
- A slave server keeps track of the position in the master's binlog of the last event applied on the slave
- This allows the slave server to re-connect and resume from where it left off after replication has been temporarily stopped
- It also allows a slave to disconnect, be cloned and then have the new slave resume replication from the same master
- There will be a measurable delay between the master and the replica. The data on the replica eventually becomes consistent with the data on the master
  - Use this feature for workloads that can accommodate this delay

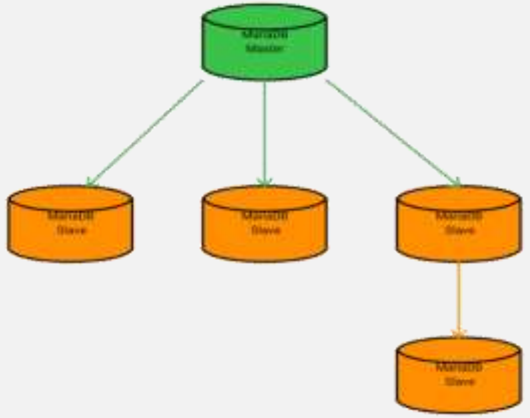
Replication Steps

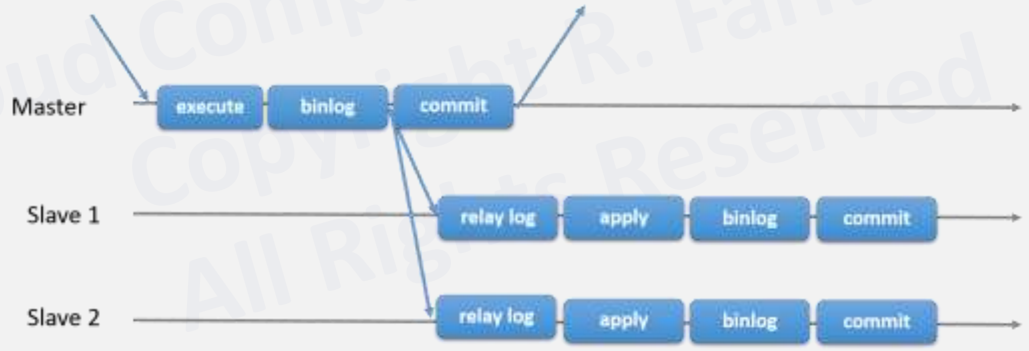
1. Replication events are read from the master by the IO thread and queued in the relay log
2. Replication events are fetched one at a time by the SQL thread from the relay log
3. Each event is applied on the slave to replicate all changes done on the master

- Replication is essentially asynchronous
  - The third step can optionally be performed by a pool of separate replication worker threads
    - **In-order** executes transactions in parallel, but orders the commit step of the transactions to happen in the exact same order as on the master
      - Transactions are only executed in parallel to the extent that this can be automatically verified
    - **Out-of-order** can execute and commit transactions in parallel
      - The application must be tolerant to seeing updates occur in different
      - Only when explicitly enabled by the application

Asynchronous Replication

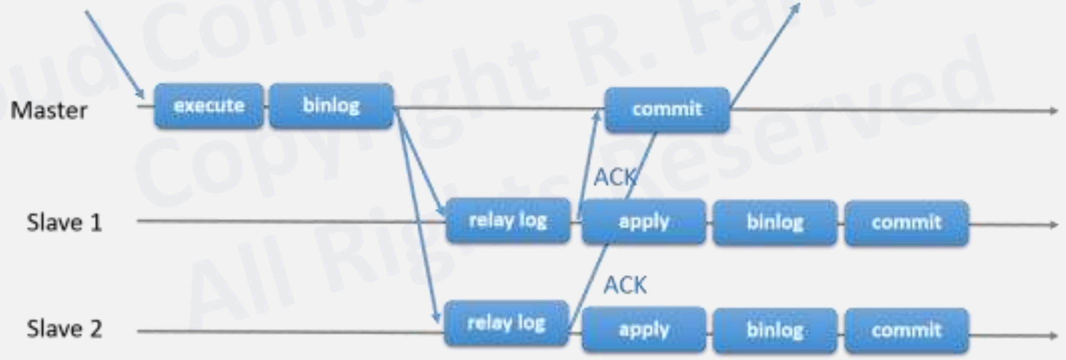
- A.k.a Standard replication
- Provides infinite read scale out
    - Most websites fit into this category, where users are browsing the website, reading articles, posts, or viewing products.
    - Updates only occur during session management, or when making a purchase or adding a comment/message to a forum.
- Provides high-availability by upgrading slave to master
- slaves read-only to ensure that no one accidentally updates them
- Eventual Consistency

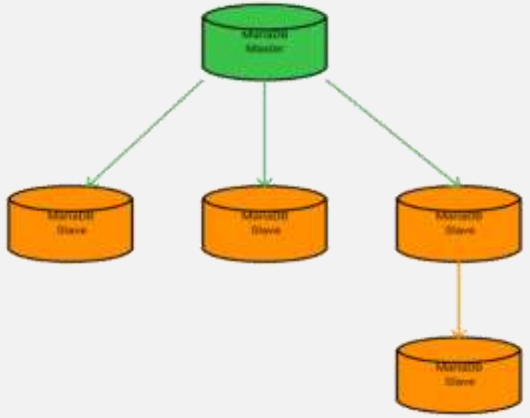




Semi-synchronous Replication

- Semi Synchronous Replication
- Better than eventual consistency
- The master waits for at least one ACK
  - Slower commits
- If no ACK received and timeout, master reverts to asynchronous
  - When at least one ACK is finally received, master goes back to semi-synchronous



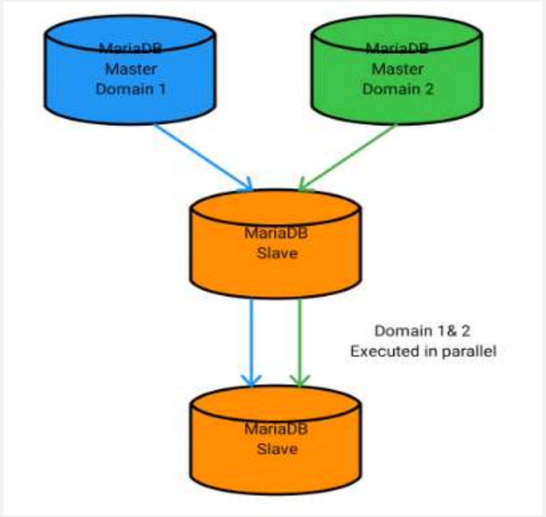


GTIDs-based Replication

- MySQL newer method based on global transaction identifiers (GTIDs)
- Transactional and therefore does not require working with log files or positions within these files
    - May simplify common replication tasks
- Replication using GTIDs guarantees consistency between master and slave as long as all transactions committed on the master have also been applied on the slave

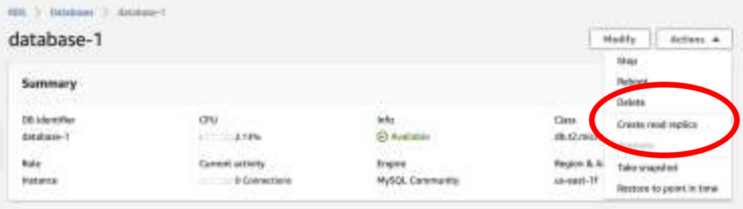
Multi-Source Replication

- Multi-source replication means that one server has many masters from which it replicates
- Allows you to combine data from different sources
- Different domains executed independently in parallel on all slaves



AWS RDS Horizontal Scaling

- Scaling beyond the compute or I/O capacity of a single DB instance for read-heavy database workloads
- Amazon RDS uses the MariaDB, MySQL, Oracle, PostgreSQL, and Microsoft SQL Server DB engines' built-in replication functionality to create a special type of DB instance called a read replica from a source DB instance
  - Up to five read replicas from one DB instance for MariaDB, MySQL
    - Similar limit of 5 replicas in Azure
    - Aurora allows 15 read replicas
  - specify an existing DB instance as the source
  - Amazon RDS takes a snapshot of the source instance and creates a read-only instance from the snapshot
  - Amazon RDS then uses the asynchronous replication method for the DB engine to update the read replica whenever there is a change to the source DB instance
- Updates to the source DB instance are **asynchronously** copied to the read replica
- If the read replica resides in a different AWS Region than its source DB instance, Amazon RDS sets up a secure communications channel between the source and the read replica
  - Amazon RDS establishes any AWS security configurations needed to enable the secure channel, such as adding security group entries
- Replicas can be "promoted" to full databases
  - They will reboot first



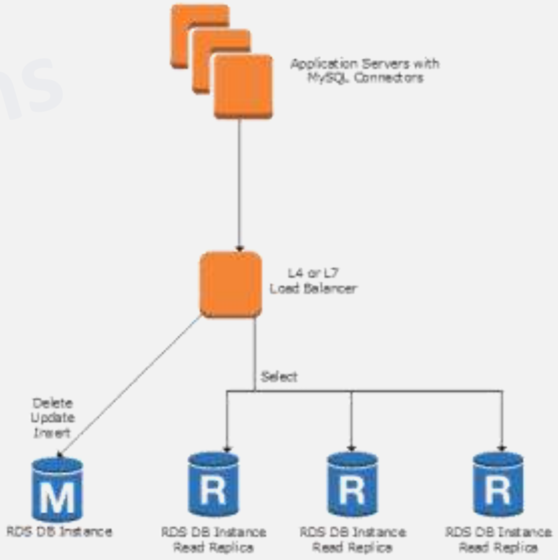
database-1

Summary

DB identifier	CPU	Info	Class
database-1	2.14%	Available	db.t2.mic
Role	Current activity	Engine	Region & A
Instance	0 Connections	MySQL Community	us-east-1f

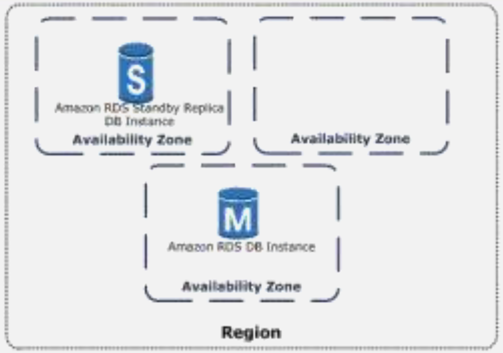
Load Balancing between RDS replicas

- Application-level load balancing
  - Different DNS record-sets using Route 53
- MySQL Connectors
  - If using the native MySQL driver, there are MySQL Connectors that allow read/write splitting and read-only endpoint load balancing without a major change in the application
- ELB does not support multiple RDS instances
- Level 4 proxy solutions
  - HAProxy: configure HAProxy to listen on one port for read queries and another port for write queries
- Level 7 proxy solutions
  - more sophisticated capability of understanding how to properly perform the read/write splits on multi-statements than a MySQL Connector does
  - This solution handles the scaling issues in a distributed database environment, so you don't have to handle scaling on the application layer, resulting in little or no change to the application itself
  - Several open-source solutions (such as MaxScale, ProxySQL, and MySQL Proxy) and also commercial solutions, some of which can be found in the AWS Marketplace


Application Servers with MySQL Connectors
L4 or L7 Load Balancer
Select
Delete Update Insert
M
RDS DB Instance
R
RDS DB Instance Read Replica
R
RDS DB Instance Read Replica
R
RDS DB Instance Read Replica

High Availability (Multi-AZ) for Amazon RDS

- Amazon RDS uses several different technologies to provide failover support
  - Multi-AZ deployments for MariaDB, MySQL, Oracle, and PostgreSQL DB instances use Amazon's failover technology
  - SQL Server DB instances use SQL Server Database Mirroring (DBM) or Always On Availability Groups (AGs)
- The high-availability feature is not a scaling solution for read-only scenarios

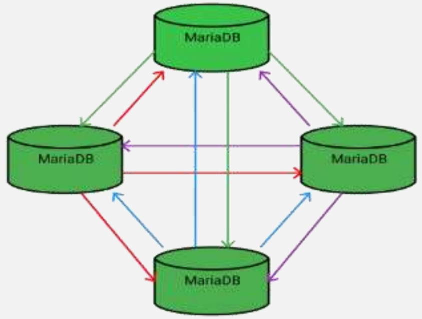


Read replicas, Multi-AZ deployments, and multi-region deployments (Amazon AWS)

Multi-AZ deploymentss	Multi-Region deployments	Read replicas
Main purpose is high availability	Main purpose is disaster recovery and local performance	Main purpose is scalability
Non-Aurora: synchronous replication; Aurora: asynchronous replication	Asynchronous replication	Asynchronous replication
Non-Aurora: only the primary instance is active; Aurora: all instances are active	All regions are accessible and can be used for reads	All read replicas are accessible and can be used for readscaling
Non-Aurora: automated backups are taken from standby; Aurora: automated backups are taken from shared storage layer	Automated backups can be taken in each region	No backups configured by default
Always span at least two Availability Zones within a single region	Each region can have a Multi-AZ deployment	Can be within an Availability Zone, Cross-AZ, or Cross-Region
Non-Aurora: database engine version upgrades happen on primary; Aurora: all instances are updated together	Non-Aurora: database engine version upgrade is independent in each region; Aurora: all instances are updated together	Non-Aurora: database engine version upgrade is independent from source instance; Aurora: all instances are updated together
Automatic failover to standby (non-Aurora) or read replica (Aurora) when a problem is detected	Aurora allows promotion of a secondary region to be the master	Can be manually promoted to a standalone database instance (non-Aurora) or to be the primary instance (Aurora)

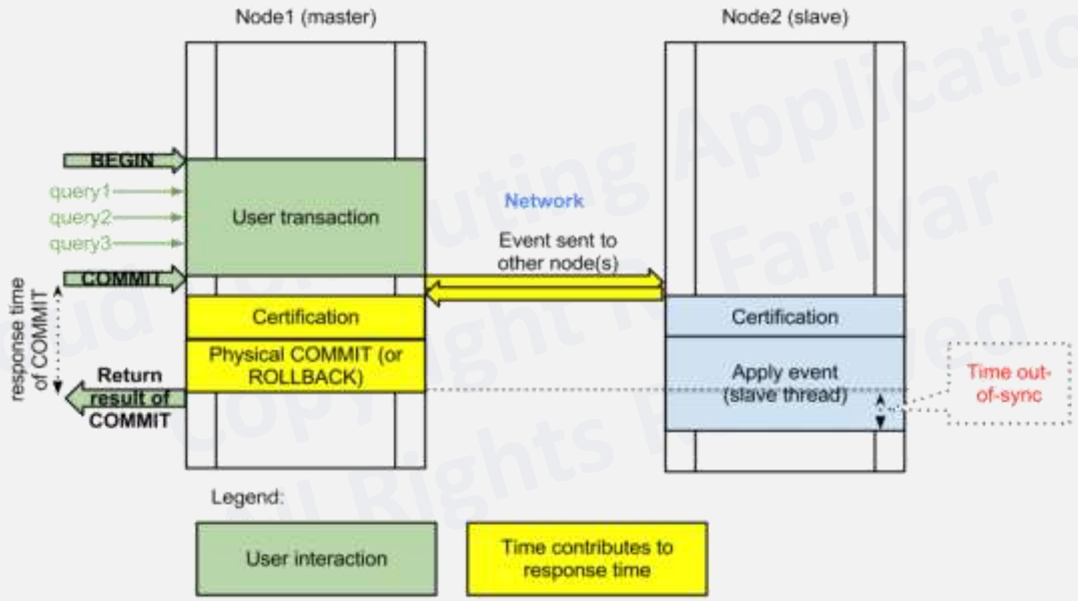
Multi-Master Cluster

- Galera (MySQL, MariaDB)
- Synchronous replication
- Active-active multi-master topology
- Read and write to any cluster node
- Automatic membership control, failed nodes drop from the cluster
- Automatic node joining
- True parallel replication, on row level
- Direct client connections, native MariaDB look & feel



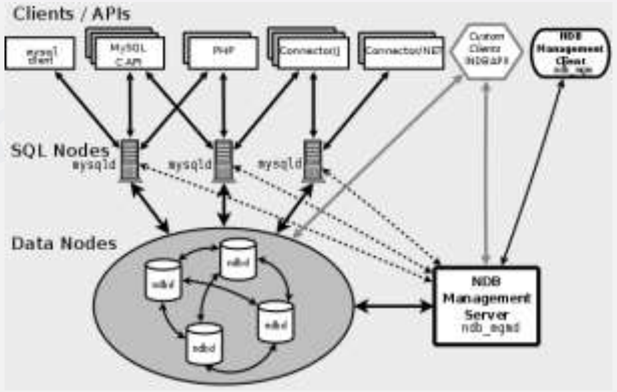
Galera Transaction Commit Flow

- Certification Based Replication
- Virtually Synchronous



MySQL NDB Cluster

- Network DataBase Engine
  - Replaces InnoDB
  - Separation of Compute and Data
    - SQL Nodes
    - Data Nodes
  - Shared Nothing Archiecture



CLOUD COMPUTING APPLICATIONS

Cloud Databases – Amazon Aurora

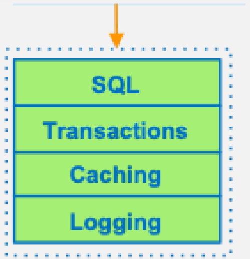
Prof. Reza Farivar

Relational Database on the Cloud

- Traditional RDBMS rely on B+Trees, replication, etc. to optimize usage on one or a few servers
- Cloud brings many new things to the table
  - Backend storage
  - Network
  - Worldwide Scalability
- How can we optimize RDBMS for the cloud?
- First step: separate storage layer from the transactional logic
  - Decoupling storage from compute
- Deuteronomy
  - Transaction Component (TC) provides concurrency control and recovery
  - Data Component (DC) provides access methods on top of LLAMA, a latch-free log-structured cache and storage manager.
- Aurora, CosmosDB both inspired by Deuteronomy

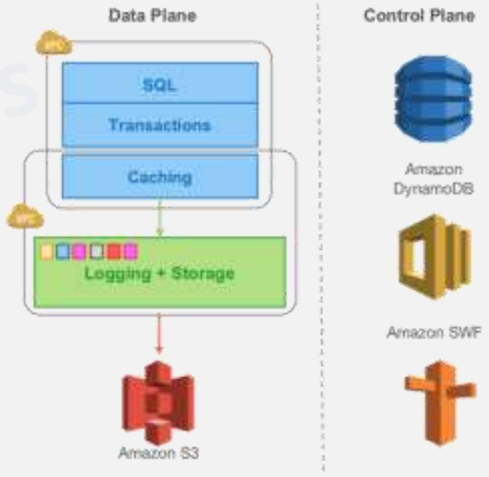
Amazon AWS Aurora

- Optimized DB engine, built from MySQL (and later PostgreSQL), with a distributed storage layer
- API compatible with MySQL or Postgres
  - i.e. you can use an existing application
- Separate storage and compute
  - Query processing, transactions, concurrency, buffer cache, and access
  - Logging, storage, and recovery that are implemented as a scale out service.
- Move caching and logging layers into a purpose-built, scale-out, self-healing, multitenant, database-optimized storage service



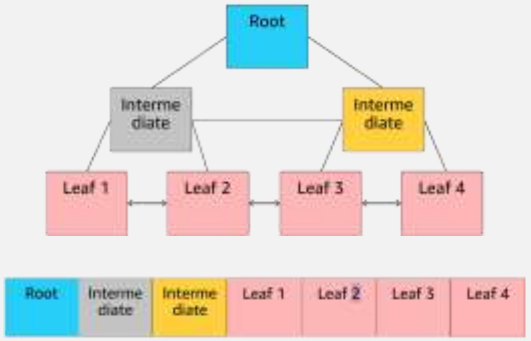
Amazon AWS Aurora

- Each instance still includes most of the components of a traditional kernel (query processor, transactions, locking, buffer cache, access methods and undo management)
- Several functions (redo logging, durable storage, crash recovery, and backup/restore) are off-loaded to the storage service



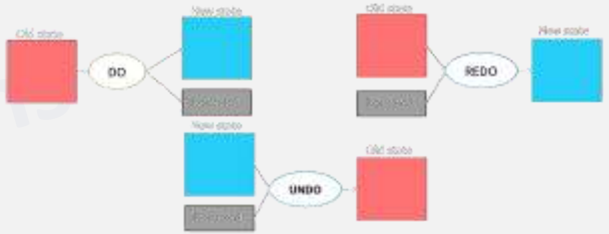
Redo Logging

- Traditional relational databases organize data in *pages* (e.g. 16KB), and as pages are modified, they must be periodically flushed to disk
  - B+ Tree
- For resilience against failures and maintenance of ACID semantics, page modifications are also recorded in *do-redo-undo log records*, which are written to disk in a continuous stream.
- rife with inefficiencies.
  - E.g. a single logical database write turns into multiple (up to five) physical disk writes, resulting in performance problems.
  - Write Amplification
  - combat the write amplification problem by reducing the frequency of page flushes
  - This in turn worsens the problem of crash recovery duration



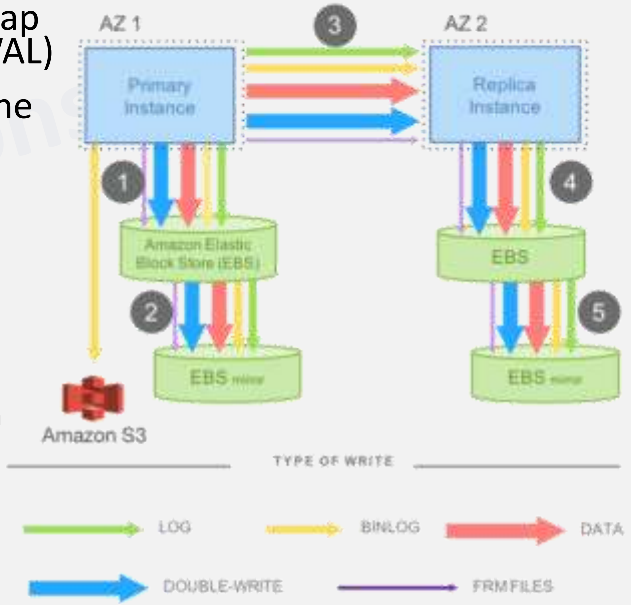
Redo Logging

- Traditional relational databases organize data in *pages* (e.g. 16KB), and as pages are modified, they must be periodically flushed to disk
  - B+ Tree
- For resilience against failures and maintenance of ACID semantics, page modifications are also recorded in *do-redo-undo log records*, which are written to disk in a continuous stream
  - redo log record : difference between the after and the before-image of a page
- rife with inefficiencies.
  - E.g. a single logical database write turns into multiple (up to five) physical disk writes, resulting in performance problems.
    - Write Amplification
  - Combat the write amplification problem by reducing the frequency of page flushes
    - This in turn worsens the problem of crash recovery duration



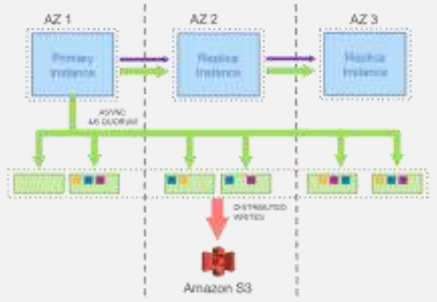
Burden of Amplified Writes

- A system like MySQL writes data pages to objects it exposes (e.g., heap files, b-trees etc.) as well as redo log records to a write-ahead log (WAL)
- The writes made to the primary EBS volume are synchronized with the standby EBS volume using software mirroring
- Data needed to be written in step 1:
  - redo log (typically a few bytes, transaction commit requires the log to be written)
  - binary (statement) log that is archived to Amazon S3 to support point-in-time restores
  - modified data pages (the data page write may be deferred, e.g. 16KB)
  - a second temporary write of the data page (double-write) to prevent torn pages (e.g. 16KB)
  - metadata (FRM) files
- Steps 1, 3, and 5 are sequential and synchronous
  - Latency is additive because many writes are sequential
  - 4/4 write quorum requirement and is vulnerable to failures and outlier performance
- Different writes representing the same information in multiple ways



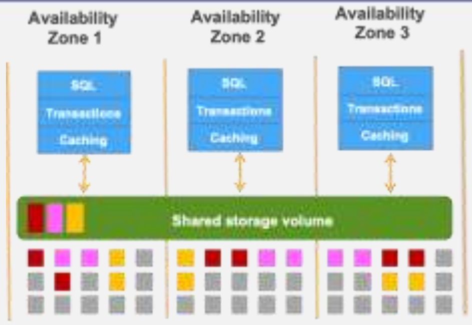
Log is the database

- In Amazon Aurora, the log is the database
- Database instances write redo log records to the distributed storage layer, and the storage takes care of constructing page images from log records on demands from the database
  - Write performance is improved due to the elimination of write amplification and the use of a scale-out storage fleet
  - 5x write IOPS on the SysBench benchmark compared to Amazon RDS for MySQL running on similar hardware
  - Database crash recovery time is cut down, since a database instance no longer has to perform a redo log stream replay
    - From 27 seconds down to 7 seconds according to one benchmark



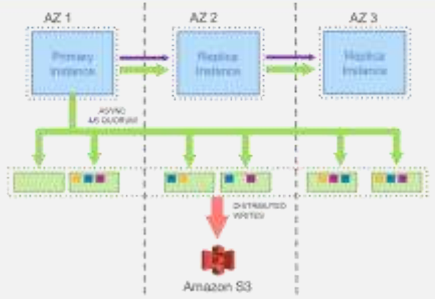
Aurora Replication and Quorum

- Everything fails all the time
  - The traditional approaches of blocking I/O processing until a failover can be carried out—and operating in "degraded mode" until recovery —are problematic at scale
    - in a large system, the probability of operating in degraded mode approaches 1
- Aurora uses quorums to combat the problems of component failures and performance degradation
  - Write to as many replicas as appropriate to ensure that a quorum read always finds the latest data
- Goal is Availability Zone+1: tolerate a loss of a zone plus one more failure without any data durability loss, and with a minimal impact on data availability
  - 4/6 quorum
  - For each logical log write, issue six physical replica writes
    - Write operation successful when four of those writes complete
    - Instances only write redo log records to storage
    - Typically 10s to 100s of bytes, makes a 4/6 write quorum possible without overloading the network
- If a zone goes down and an additional failure occurs, can still achieve read quorum (3/6), and then quickly regain the ability to write by doing a *fast repair*



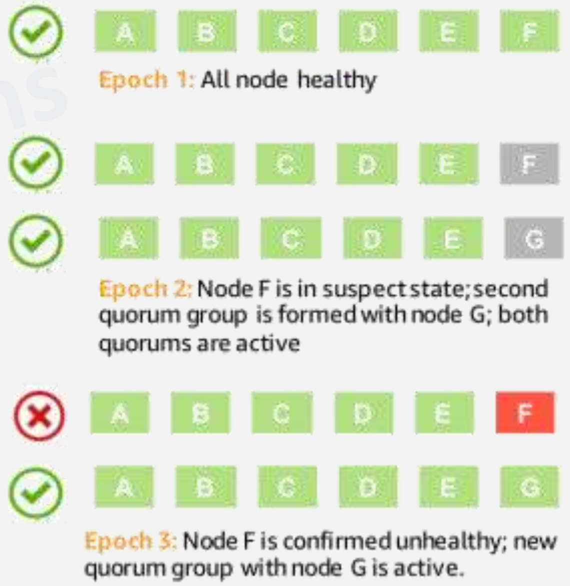
Offloading Redo Processing to Storage

- In Aurora, the only writes that cross the network are redo log records
  - No pages are ever written from the database tier, not for background writes, not for checkpointing, and not for cache eviction
- log applicator is pushed to the storage tier to generate database pages in background or on demand
  - Generating each page from the complete chain of its modifications from the beginning of time is prohibitively expensive
  - Continually materialize database pages in the background to avoid regenerating them from scratch on demand every time
- The storage service can scale out I/Os in an embarrassingly parallel fashion without impacting write throughput of the database engine
- primary only writes log records to the storage service and streams those log records as well as metadata updates to the replica instances
- database engine waits for acknowledgements from 4 out of 6 replicas in order to satisfy the write quorum



Aurora Fast Repair

- Amazon Aurora approach to replication: based on sharding and scale-out architecture
- An Aurora database volume is logically divided into 10-GiB logical units (*protection groups*), and each protection group is replicated six ways into physical units (*segments*)
- When a failure takes out a segment, the repair of a single protection group only requires moving ~10 GiB of data, which is done in seconds.
- When multiple protection groups must be repaired, the entire storage fleet participates in the repair process.
    - Massive bandwidth to complete the entire batch of repairs
- A zone loss followed by another component failure → Aurora may lose write quorum for a few seconds for a given protection group
    - Recovery is quick


A
B
C
D
E
F
Epoch 1: All node healthy
A
B
C
D
E
F
A
B
C
D
E
G
Epoch 2: Node F is in suspect state; second quorum group is formed with node G; both quorums are active
A
B
C
D
E
F
A
B
C
D
E
G
Epoch 3: Node F is confirmed unhealthy; new quorum group with node G is active.

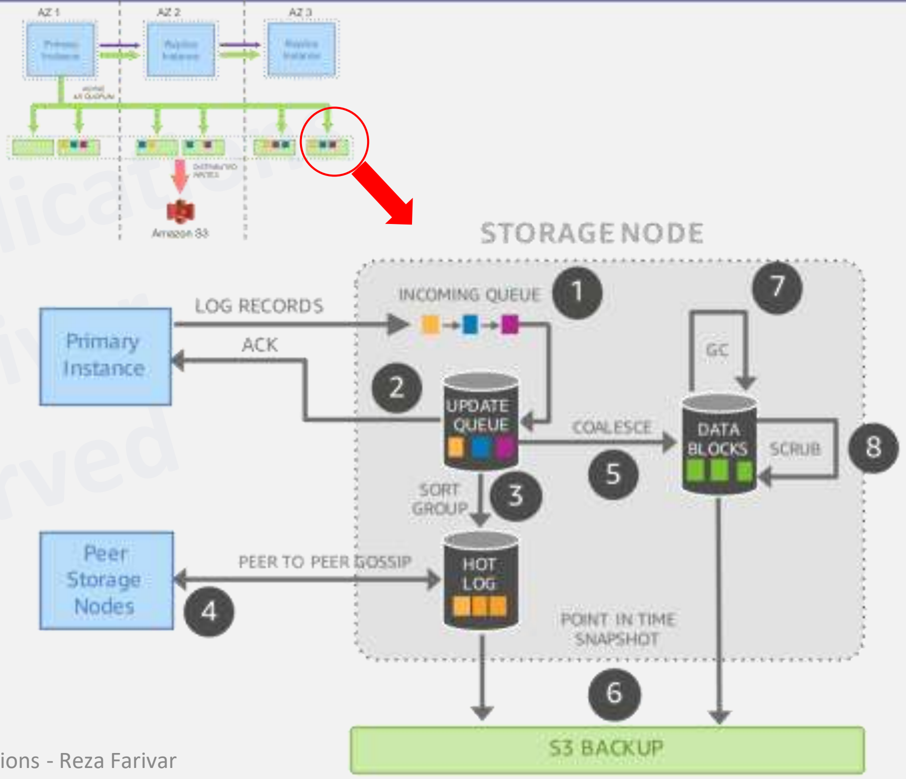
Quorum Reads

- A quorum read is expensive, and is best avoided
- We do not need to perform a quorum read on routine page reads
  - It always knows where to obtain an up-to-date copy of a page
  - The client-side Aurora storage driver tracks which writes were successful for which segments
  - The driver tracks read latencies, and always tries to read from the storage node that has demonstrated the lowest latency in the past
- The only scenario when a quorum read is needed is during recovery on a database instance restart

Amazon Aurora Storage Nodes

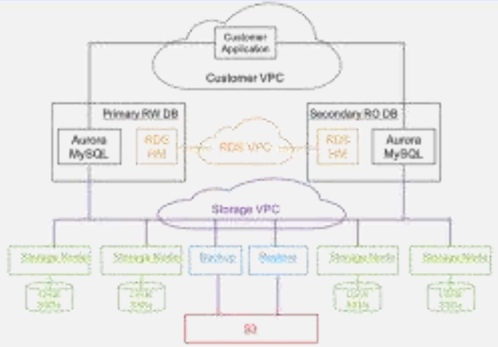
1. Receive log records and add to in-memory queue
2. persist record on disk and acknowledge, ACK to the database
3. Organize records and identify gaps in log since some batches may be lost
4. Gossip with peers to fill in holes
5. Coalesce log records into new page versions
6. Periodically stage log and new page versions to S3
7. Periodically garbage-collect old versions
8. Periodically validate CRC codes on blocks

- Each of the steps above are asynchronous
- Only steps (1) and (2) are in the foreground path potentially impacting latency



Database Engine Implementation

- The database engine is a fork of “community” MySQL/InnoDB and diverges primarily in how InnoDB reads and writes data to disk
  - In community InnoDB, a write operation results in data being modified in buffer pages, and the associated redo log records written to buffers of the WAL in LSN order
  - On transaction commit, the WAL protocol requires only that the redo log records of the transaction are durably written to disk
  - The actual modified buffer pages are also written to disk eventually through a double-write technique to avoid partial page writes
  - These page writes take place in the background, or during eviction from the cache, or while taking a checkpoint
- In addition to the IO Subsystem, InnoDB also includes the transaction subsystem, the lock manager, a B+-Tree implementation and the associated notion of a “mini transaction” (MTR).
  - An MTR is a construct only used inside InnoDB and models groups of operations that must be executed atomically (e.g., split/merge of B+-Tree pages).
- Concurrency control is implemented entirely in the database engine without impacting the storage service

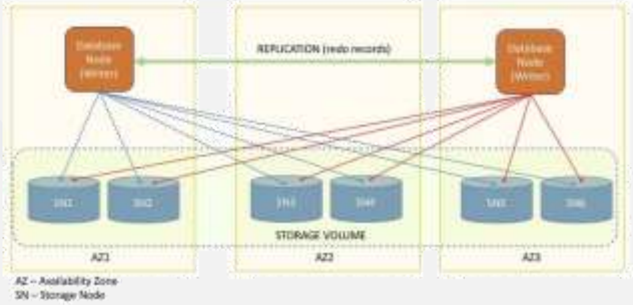


Aurora and Consensus

- Aurora leverages only quorum I/Os, locally observable state, and monotonically increasing log ordering to provide high performance, non-blocking, fault-tolerant I/O, commits, and membership changes
- Aurora is able to avoid much of the work of consensus by recognizing that, during normal forward processing of a system, there are local oases of consistency
- Using backward chaining of redo records, a storage node can tell if it is missing data and gossip with its peers to fill in gaps
- Using the advancement of segment chains, a database instance can determine whether it can advance durable points and reply to clients requesting commits
- The use of monotonically increasing consistency points – SCLs, PGCLs, PGMRPLs, VCLs, and VDLs – ensures the representation of consistency points is compact and comparable
  - These may seem like complex concepts but are just the extension of familiar database notions of LSNs and SCNs.
- The key invariant is that the log only ever marches forward.

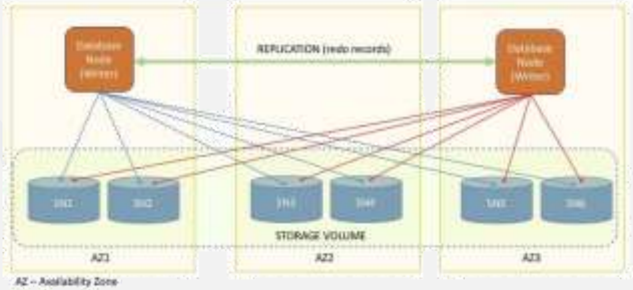
Aurora Multi-Master

- For high availability and ACID transactions across a cluster of database nodes with configurable read after write consistency
- With single-master Aurora, a failure of the single writer node requires the promotion of a read replica to be the new writer
- In the case of Aurora Multi-Master, the failure of a writer node merely requires the application using the writer to open connections to another writer
- When designing for high availability, make sure that you are not overloading writers
- Conflicts arise when concurrent transactions or writes executing on different writer nodes attempt to modify the same set of pages



Aurora Multi-Master Replication and Quorom

1. The application layer starts a write transaction
2. For cross-cluster consistency: The writer node proposes the change to all six storage nodes
3. Each storage node checks if the proposed change conflicts with a change in flight or a previously committed change and either confirms the change or rejects it
   - Each storage node compares the LSN (think of this as a page version) of the page submitted by the writer node with the LSN of the page on the node
   - It approves the change if they are the same and rejects the change with a conflict if the storage node contains a more recent version of the page
4. If the writer node that proposed the change receives a positive confirmation from a quorum of storage nodes:
   1. First, it commits the change in the storage layer, causing each storage node to commit the change
   2. It then replicates the change records to every other writer node in the cluster using a low latency, peer-to-peer replication protocol
5. The peer writer nodes, upon receiving the change, apply the change to their in-memory cache (buffer pool)
6. If the writer node that proposed the change does not receive a positive confirmation from a quorum of storage nodes, it cancels the entire transaction and raises an error to the application layer (The application can then retry the transaction)
7. Upon successfully committing changes to the storage layer, writer nodes replicate the redo change records to peer writer nodes for buffer pool refresh in the peer node



AZ – Availability Zone
SN – Storage Node

Aurora vs. RDS

- RDS offers a greater range of database engines and versions than Aurora RDS
- Aurora RDS offers superior performance to RDS due to the unique storage subsystem
- Aurora RDS offers superior scalability to RDS due to the unique storage subsystem
- The pricing models differ slightly between RDS and Aurora RDS, but Aurora RDS is generally a bit more expensive to implement for the same database workload
- Aurora RDS offers superior high availability to RDS due to the unique storage subsystem
- According to AWS, Aurora offers five times the throughput of standard MySQL, performance on-par with commercial databases, but at one-tenth the cost
  - It should be noted that these numbers are AWS marketing claims.
  - House of Brick has found the cost claims to be roughly correct when compared with Oracle Enterprise Edition deployments, but the performance advantage of Aurora in real-world scenarios is closer to 30%

http://houseofbrick.com/aws-rds-mysql-vs-aurora-mysql/

CLOUD COMPUTING APPLICATIONS

Cloud Databases – Google Cloud Spanner

Prof. Reza Farivar

Google Cloud Spanner

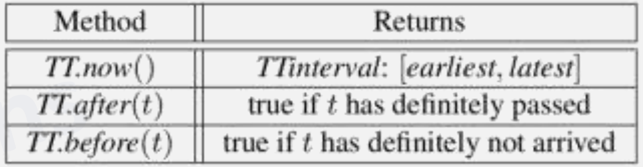
- Spanner is a distributed data layer that uses optimized sharded Paxos to guarantee consistency even in a system that spans multiple geographic regions
  - Query API is SQL (i.e. SELECT)
  - Insert and updates are done through a specialized GRPC interface
- Two-phase commit to achieve serializability
- TrueTime for external consistency, consistent reads without locking, and consistent snapshots
  - *External > Strong > Weak*

Spanner and CAP

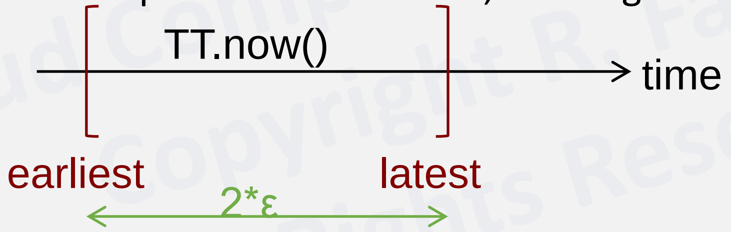
- Is Spanner C+A+P?
  - No, It is CP
  - during (some) partitions, Spanner chooses C and forfeits A
- Availability is in the 5 nines range
  - Is this acceptable to your application?
  - Effectively CA
- Spanner uses the Paxos algorithm as part of its operation to shard (partition) data across hundreds of servers
- 2PC known as the anti-availability protocol
  - because all members must be up for it to work
  - In Spanner, each member is a Paxos group
  - ensures each 2PC “member” is highly available even if some of its Paxos participants are down
- Cloud Spanner provides stale reads, which offer similar performance benefits as eventual consistency but with much stronger consistency guarantees
  - A stale read returns data from an "old" timestamp, which cannot block writes because old versions of data are immutable

TrueTime

- Heavy use of hardware-assisted clock synchronization using GPS clocks and atomic clocks to ensure global consistency
  - avoid communication in a distributed system
  - GPS and Atomic clock have different failure modes
- “Global wall-clock time” with bounded uncertainty
  - ε is worst-case clock divergence
  - Timestamps become intervals, not single values



Method	Returns
TT.now()	*TTinterval: [earliest, latest]*
TT.after(t)	true if *t* has definitely passed
TT.before(t)	true if *t* has definitely not arrived



- Consider event e_{now} which invoked tt = TT.now():
  - Guarantee: tt.earliest <= $t_{abs}(e_{now})$ <= tt.latest

Spanner

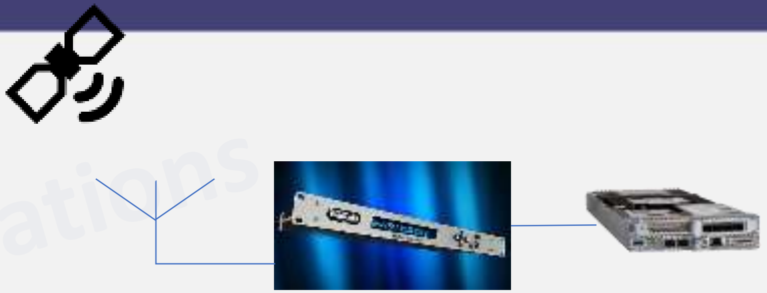
- TrueTime exposes clock uncertainty
  - Commit wait ensures transactions end after their commit time
  - Read at TT.now.latest()
- Reads dominant, make them lock-free
  - Read-Only Transaction
    - A replica can satisfy a read at a timestamp t if t <= t_{safe}.
  - Snapshot Read, client-provided timestamp
    - read at a particular time in the past
  - Snapshot Read, client-provided bound
- Read-Write Transaction less common
  - Pessimistic, use 2 phase locking
- Globally-distributed database
  - 2PL w/ 2PC over Paxos!

$$t_{safe} = min(t_{safe}^{Paxos}, t_{safe}^{TM})$$

Paxos State Machine Safe Time

Transaction Manager Safe Time

HW-based Time Synchronization



- NTP
  - Software time synchronization with one server
  - Stratum 2
  - Network delays
  - 1/2 to 100 ms accuracy
- Google Spanner
- Microsoft Azure now offers GPS clock synchronization
  - VMICTimeSync provider
  - Precision Time Protocol
  - Stratum 1 devices
    - I.e. direct connection, not through shared network, to a reference time server (Stratum 0)
    - 10 microseconds accuracy
      - For reference, GPS accuracy < 1 us, 95% of the time ≤40 nanoseconds
- Amazon Time Sync Service
  - Chrony vs. NTP

CLOUD COMPUTING APPLICATIONS

Cloud Databases – Microsoft Azure CosmosDB

Prof. Reza Farivar

Azure CosmosDB

- Globally distributed, multi-model database
  - Wire compatible with Cassandra
  - Table API
    - 5 types of consistency levels
  - MongoDB API
  - Etcd API
    - etcd is a consistent distributed key value storage
    - Compare with Zookeeper
    - Backend of Kubernetes
  - Gremlin API
- Replicated State Machine (RSM) for concurrency
  - Specific algorithm not published
  - *RAFT is also a state machine consensus algorithm*

Azure CosmosDB

- Write-optimized, resource-governed and schema-agnostic database engine
  - It automatically indexes everything it ingests
    - Internally based on a document store model
    - All JSON documents are a tree
    - Make an index for each path of the tree
  - **Synchronously** makes the index durable and highly available before acknowledging the client's updates while maintaining low latency guarantees
- BW-Tree indexing
  - Log Structure Record
  - Latch free updates
  - Atom-record-sequence (ARS) system
- Uses Lamport's TLA+ specification language to describe SLAs at each consistency level

CosmosDB Consistency Models

- **Strong:** With strong consistency, you are guaranteed to always read the latest version of an item similar to read committed isolation in SQL Server. You can only ever see data which is durably committed. Strong consistency is scoped to a single region.
- **Bounded-staleness:** In bounded-staleness consistency read will lag behind writes and guarantees global order and not scoped to a single region. When configuring bounded-staleness consistency you need to specify the maximum lag by:
  - Operations: For a single region the maximum operations lag must be between 10 and 1,000,000, and for the multi region, it will be between 100,000 and 1,000,000.
  - Time: The maximum lag must be between 5 seconds and 1 day for either single or multi-regions.
- **Session:** This is the most popular consistency level, since it provides consistency guarantees but also has better throughput.
- **Consistent Prefix:** A global order is preserved, and prefix order is guaranteed. A user will never see writes in a different order than that is which it was written.
- **Eventual:** Basically, this is like asynchronous synchronization. It guarantees that all changes will be replicated eventually, and as such, it also has the lowest latency because it does not need to wait on any commits.

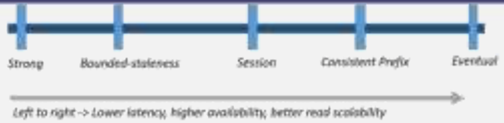

Strong
Bounded-staleness
Session
Consistent Prefix
Eventual
Left to right -> Lower latency, higher availability, better read scalability

Figure 4. Multiple well-defined consistency choices along the spectrum.

Consistency Level	Guarantees
Strong	Linearizability
Bounded Staleness	Consistent Prefix. Reads lag behind writes by *k* prefixes or *t* interval
Session	Consistent Prefix. Monotonic reads, monotonic writes, read-your-writes, write-follows-reads
Consistent Prefix	Updates returned are some prefix of all the updates, with no gaps
Eventual	Out of order reads

Azure CosmosDB Implementation

- Cosmos DB service is deployed on several replicated shared-nothing nodes across geographical regions for high-availability, low-latency, and high throughput.
- Some or all of these distributed nodes form a replica set for serving requests on a data shard that contains documents.
- Among the replicas, one of them is elected as a master to perform totally-ordered writes on the data shard.
- Writes are done on the write-quorum (W), a subset of the replica nodes, to ensure that the data is durable.
- Reads are performed on read-quorum (R), a subset of replica nodes, to get the desired consistency levels (Strong, Bounded-staleness, Session, Consistent Prefix, Eventual) as configured by users.
- Data is partitioned at logic level and is replicated at storage layer in terms of physical partitions to achieve desired availability and throughput.
- storage
    - transactional storage engine
    - analytical storage engine
    - storage engines are log-structured and write-optimized

CLOUD
COMPUTING
APPLICATIONS

Cloud Databases - NoSQL

Prof. Reza Farivar

Key/value Databases

- Key-value databases are optimized for common access patterns, typically to store and retrieve large volumes of data
- These databases deliver quick response times, even in extreme volumes of concurrent requests

- High-traffic web apps, ecommerce systems, and gaming apps
- AWS DynamoDb
- Azure CosmosDB

Wide Column Databases

- Google BigTable
  - Cloud Bigtable is a fully managed, wide-column NoSQL database that offers low latency and replication for high availability
  - This is what HBase was modeled after
- Managed Casandra
  - *Cassandra was modeled after Dynamo (paper)*
  - *DynamoDB was modeled after Casandra*
  - AWS managed Casandra
- Cassandra AMI for any cloud provider

In Memory (Cache) Databases

- In-memory databases are used for applications that require real-time access to data
- By storing data directly in memory, these databases deliver microsecond latency to applications for whom millisecond latency is not enough
- Caching, gaming leaderboards, and real-time analytics
- Common usage pattern: Cache RDS or document databases

- AWS ElastiCashe (Redis / MemCached)
- Azure Cache for Redis
- Google Memorystore (Redis / MemCached)
- IBM Redis

Document Databases

- A document database is designed to store semistructured data as JSON-like documents
- Makes it easy to store, ***query***, and index JSON data
- Content management, catalogs, and user profiles
- Non-relational database service

- AWS DocumentDB + (MongoDB compatibility)
- Azure CosmosDB
- Google Firestore
  - Targeted for mobile App support
- IBM Cloudant / IBM MongoDB

AWS DocumentDB

- Managed instance
- Implements MongoDB 3.6 API
- Storage and compute are decoupled, allowing each to scale independently
- Automatically grows the size of storage volume as the database storage needs grow
  - Grows in increments of 10 GB, up to a maximum of 64 TB
- Up to 15 low latency read replicas to increase read capacity
- Replicates six copies of data across three AWS Availability Zones (AZs)
- Access to Amazon DocumentDB clusters must be done through the mongo shell or with MongoDB drivers

Other Types of Cloud Databases

- Graph Databases
  - Covered in a different module
- Time Series Databases
  - AWS Timestream
- Blockchain / Ledgers
  - Immutable and cryptographically verifiable transactions
  - AWS QLDB
- Data Warehouses
  - Covered in a different module
  - Columnar storage
    - AWS Redshift
    - Google BigQuery
    - Azure Synapse (formerly Azure SQL Data Warehouse)