paper: Amazon Aurora: Design Considerations for High Throughput Cloud-Native Relational Databases

by Diego Pacheco

Amazon Aurora Design Considerations for Hi... 2 MB

https://www.allthingsdistributed.com/files/p1041-verbitski.pdf

- **1 - Introduction**
- Amazon Aurora = Relation DB for OLTP workloads
- High constraint in high throughput db processing: Moved from Compute and Storage to Network.
- Aurora pushed redo processing to a multi-tenant scale-out storage service.
- With this approach Aurora: reduce network traffic, fast crash recovery, failover to replicas without data loss, fault tolerant self-healing storage.
- Aurora consensus on durable state across numeros storage engines with efficient schema engine avoiding expensive and chatty recovery protocols
- In modern distributed cloud services: Resiliency and Scalability are achieved by decoupling compute from storage by replicating storage across multiple nodes.
- This approach allow handle operations like: replace misbehaving, unreachable hosts, adding replicas, failing over from a writer to a replica, scale db size up and down.
- I/O is a traditional DB bottleneck, with Aurora I/O is spread across multiple nodes in a multi-tenant fleet, so individual disks and nodes are not HOT anymore.
- Network traffic gets an amplification in this model beyond the basic PPS(Packet per Second)
- Most operations of the Database can overlap with each other - there are several situations that requires synchronous operations.
- Which results in STALLS and CONTEXT SWITCHES
- i.e: Disk read due a miss in db cache buffer, a read thread cannot continue untils its reads completes. A cache miss introduce extra penalty(evicting, flushing) a dirty cache page to accommodate new page.
- Background processing such as checkpointing and dirty page writing can reduce the occurrence of this penalty but also can create STALLS, Context Switch and resource contention.
- Transaction commits are another source of interference: A STALL into a commiting one TX can inhibit other to processing.
- 2PC is hard in cloud systems because these protocols are fault intolerant also high scale distributed systems have "background noise" of hard and soft failures.
- 2PC protocols are HIGH LATENCY.
- Architecture: Moving Logging + Storage out of the Database engine

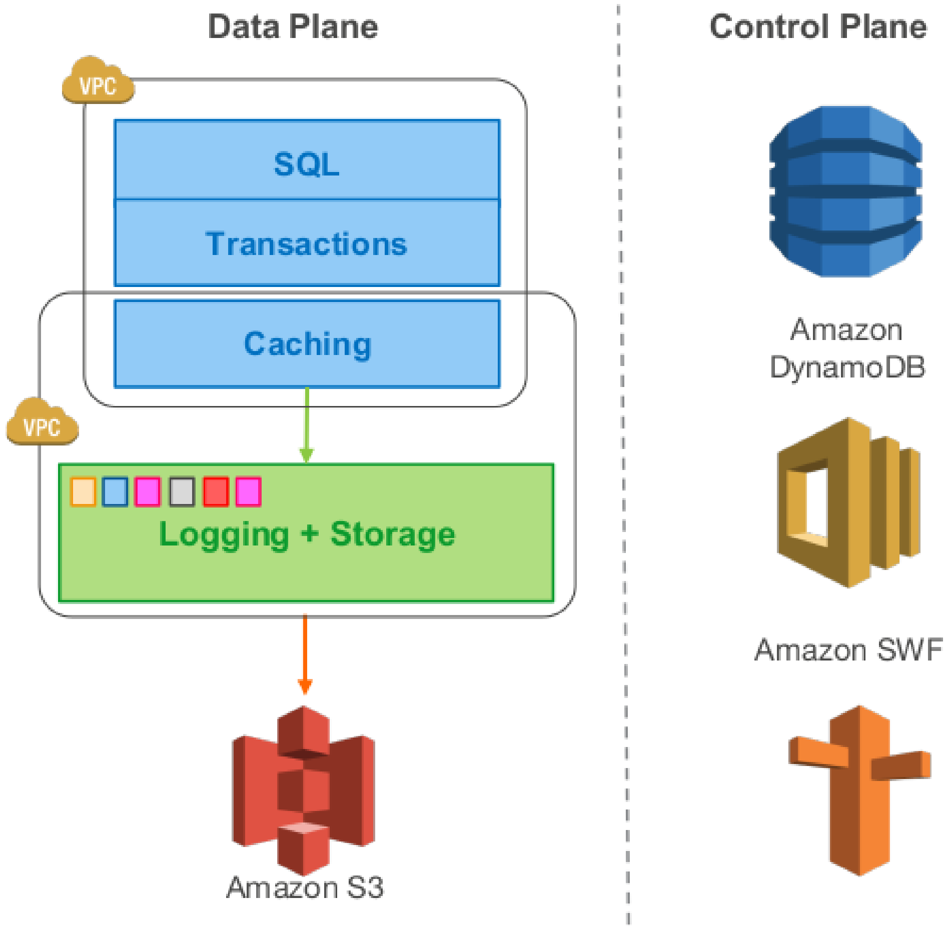

Figure 1: Move logging and storage off the database engine

- Aurora uses a SOA Architecture
- Aurora has a Multi-Tenant scale-out Storage Service that abstracts virtualized segmented REDO log and is loosely coupled to a fleet of DB instances.
- Each instances has components of traditional kernel(query processing, transactions, locking, buffer cache, access methods, undo mgmt)
- Several functions like: (redo logging, durable storage,crash recovery, backup/restore) are off-loaded to the Storage Service.
- Aurora Architecture has 3 big advantages over traditional systems:
  - 1 - Building Storage as independent, fault-tolerant, self-healing service across multiple data centers. We can PROTECT the DB from Performance VARIANCE and Transient / Permanent Failures at network/storage tiers.
  - 2 - By only writing redo logs into the Storage Service reduce the network IOPs by a order of magnitude. (significant other optimizations and improves made more throughput than mysql).
  - 3 - We move most critical and complex operations like Backup and redo recovery from one-time expensive operation to continuous async operations amortized across large distributed fleet. This yields near-instant crash recovery without checkpointing and inexpensive backups with no interference on foreground process.
- this paper discuss 3 main contributions:
  - How to reason about durability at cloud scale, how to design quorum systems that are resilient to correlated failures.
  -

How to leverage smart storage by offloading the lower quarter of traditional db.
- How to eliminate multi-phase sync, crash recovery, and checkpointing in distributed storage.

- **2.1 - Replication and correlated failures**
- Instance lifetime does not correlate well with storage lifetime. Instances fail. Customers shut them down, they resize up/down for this reasons make sense decouple storage with compute tier.
- Storage fails as well and each failure has different duration and different blast radius.
- Storage failures like i.e: transient lack of network availability to a node, temporary downtime on a reboot, or permanent failure of: disk, node, rack, leaft, spine network switch or even a DC.
- One approach to tolerate failures is replicated systems have a quorum based voting protocol.
- In order to achieve consistency the quorum must obey 2 rules:
  - 1 - Each read must be aware of most recent writes (Vr + Vw > V). So we have at least 1 location with the newest version.
  - 2 - Each write must be aware of the most recent write. Avoiding conflicting writes (Vw > V/2).
- Common approach to tolerate loss of a single node is to replicate data to 3 nodes (V3) and rely on write quorum 2/3(Vw == 2) and read quorum 2/3 (Vr == 2).
- Aurora team belive 2/3 quorum is inadequate.
- AWS AZ are isolated in some failures like: power, networking, software deployments, and flooding.
- However with large storage fleet, the background noise of failures imply that in any given point in time some subset of disks and nodes failed and are being replaced.
- This failures(^) might be spread across all AZs which could break 2/3 read quorum would be lost 2 copies and not able to tell if the 3rd is up to date.
- Aurora choosed a design point of tolerance:
  - A) Losing an entire AZ and one additional node (AZ+1) without losing data.
  - B) Losing an entire AZ without impacting the ability without the ability to write data.
- That is all done via Replicating each DATA item to 6 ways across 3 AZs with 2 copies on each item on each AZ.
- Aurora quorum model has 6 Votes(V == 6) a write quorum 4/6(Vw == 4) and a read quorum 3/6(Vr == 3)
- **2.2 Segmented Storage**
- Small fixed size segments. 10GB each segments. Which are replicated 6 ways.
- Each segment is replicated to a PG(Protection Group) in 3 azs total 6 copies.
- Segments are physically implemented with large fleet storage nodes that are provisioned as virtual hosts with attached SSD using AWS EC2.
- Volumes can grow up to 64TB.
- Segments are now a unit of background noise failure and repair. AWS monitor and repair faults as part of the service.
- A 10GB segment can be replaced in 10s on 10GPs network link.
- **2.3 Operational Advantages of Resilience**
- Since AWS Aurora can tolerate failures: Heat management is straightforward. OS And Security patches and briefly unavailability events. Software updates are managed this way.
- All this is executed one AZ at the time.
- **3. The log is the Database**
- Traditional Dbs: segmented replicated storages have a burden in relation of Network IOPs and sync STALLs
- Aurora Approach to offload LOG processing to storage service reduce network IOPs and techniques to minimize sync STALLS and unnecessary writes.
- **3.1 The burden of Amplified Writes**
- Aurora model creates untenable model for traditional DB like MySQL that generate many Actual I/Os for each application write
- The high IO volume is amplified by replication imposing heavy packets per second(PPS) burden - The IO results in points of sync that stall pipelines and dilatete latency.

- While chain of replication can reduce the network cost they still suffer from sync STALLs and additive latencies.
- MySQL Traditional DB - How Writes work
  - MySQL write data pages to objects that it expose(heap fles, b-trees) as well redo log records to write-ahead-log(WAL)
  - Each redo logs consists in: the diff between after-image and before-image of the page that was modified.
- Consider a mirrored HA mysql Replication Arch

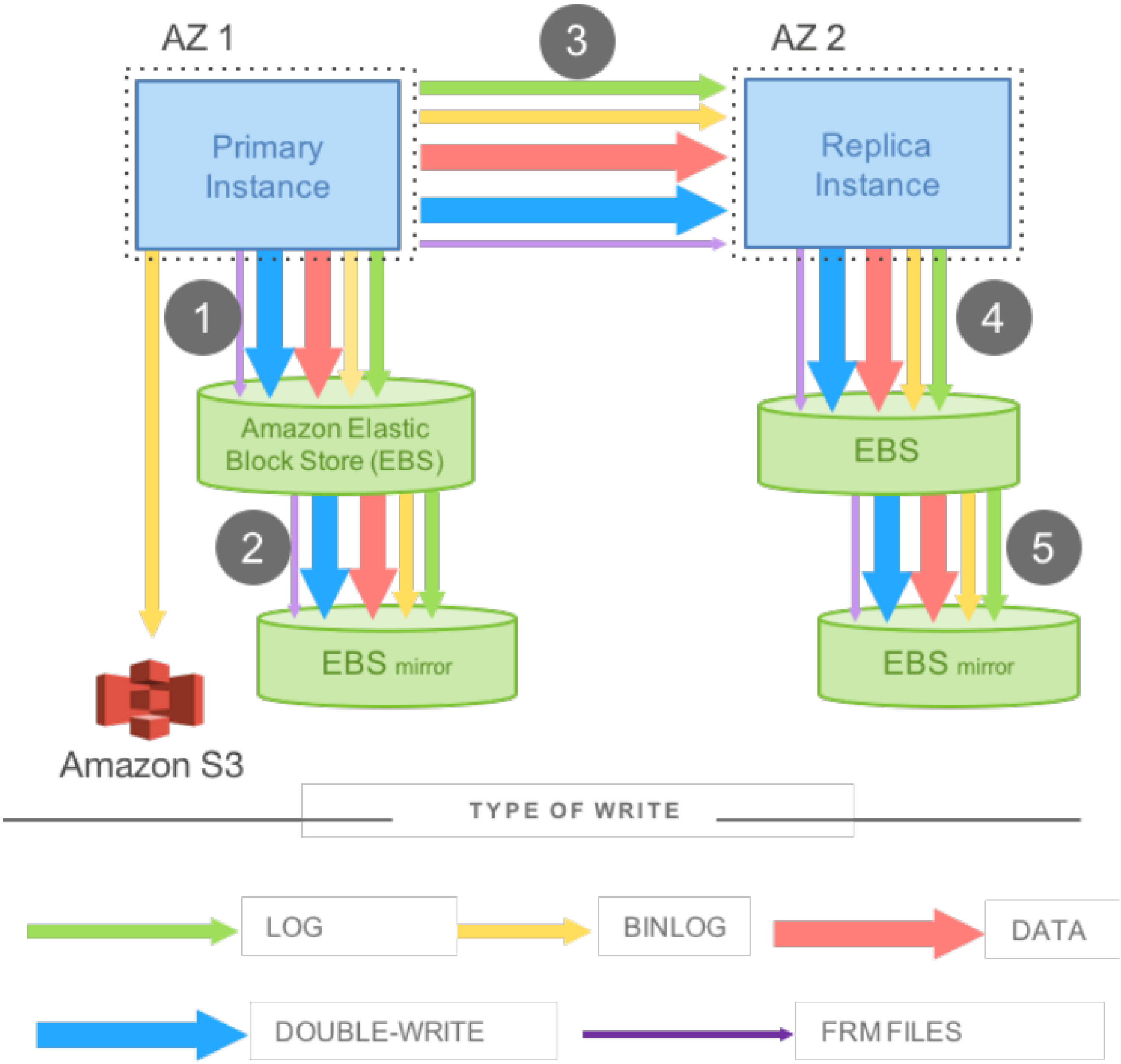

Figure 2: Network IO in mirrored MySQL

- MySQL arch is not IDEAL:
  - Data written in 1,3,5 is SEQUENTIAL and SYNC
  - Latency is additive because many writes and sequential.
  - Jitter is amplified even with async writes
  - One must wait for the slowest operations leaving the system a mercy of outliers.
- From distributed system perspective mysql can be viewed as 4/4 write quorum and is vulnerable to failures and outliers performance
- **3.2 Offloading REDO processing log**
-

When traditional DB modify data page it generates a REDO log and invokes a log applicator that apply the redo log record to the in-memory before-image of to the page that produce its after-image. Transaction Commits Requires the log to be written but the data page write might be deferred.

- Network IO in AWS Aurora

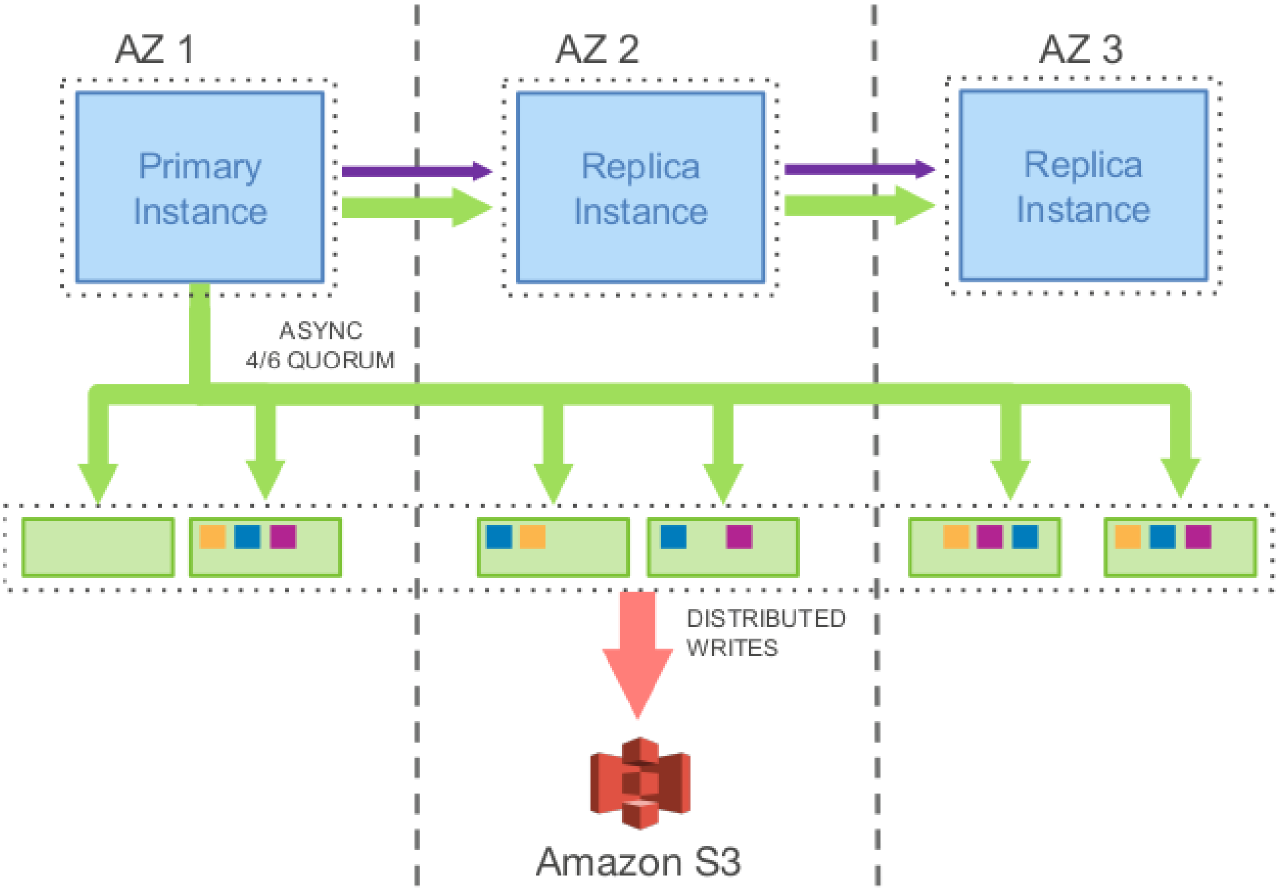

Figure 3: Network IO in Amazon Aurora

- In aurora the only writes that cross the network are REDO log writes.
- No pages are written from the database tier: not for background writes, not for checkpointing, not for caching eviction
- Instead the log applicator is pushed to the storage tier where it can be used to generate database pages in background or on demand.
- The LOG IS THE DATABASE.
- **3.3 Storage Service Design Points**
- In aurora background progressing has negative correlation with foreground processing.
- Traffic in Aurora Storage Nodes

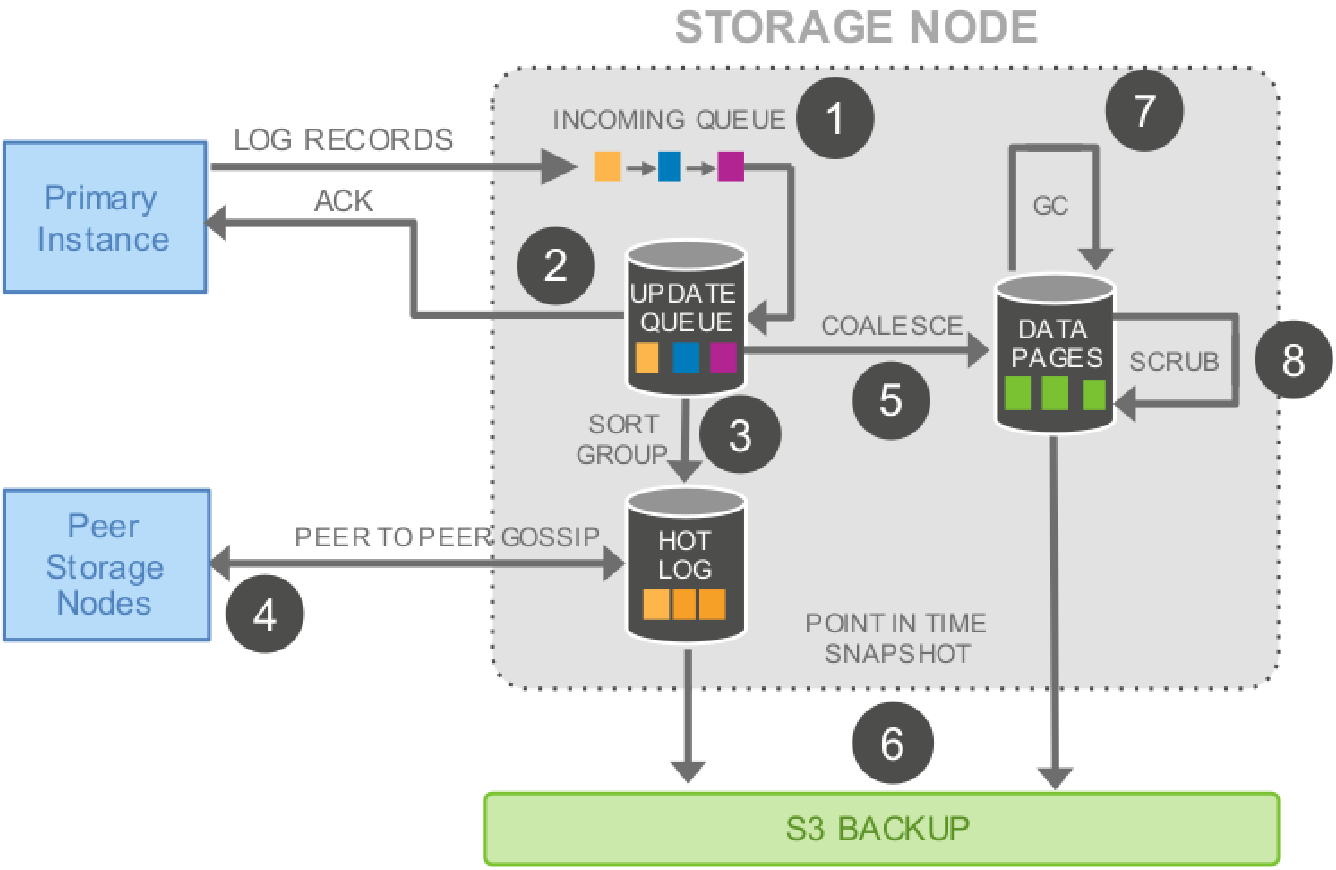

Figure 4: IO Traffic in Aurora Storage Nodes

- Storage Nodes Activity in more details:
- 1) Recive the log record and store in a in-memory queue
- 2) Persist the record in disk and acknowledge
- 3) Organize records and identify gaps since some batches may be lost
- 4) Gossip with peers to feel in gap
- 5) Coalesse log records into new data pages
- 6) Periodically stage logs and new pages to S3
- 7) Periodically garbage collect old versions and finally
- 8)Periodically validate CRC codes on pages
- All Async only step 1 and 2 are foreground process potencialy impacting latency
- **4. The Log marches Forward**
- Describe how the log is generated from the database engine - So: Durable State, Runtime State and replicate state are always consistent.
- How we implement consistency without expensive 2PC protocol.
- **4.1 Solution Sketch: Async Processing**
- Log advances as Ordered Sequence of changes
- Each log record has a associated LSN(Log Sequence Number) which is monotonically increasing value generated by the database.
- This(^) Let us simplify consensus protocol for maintaining state by approaching the problem as async fashion
- Instead of using Chatty and fault intolerant 2PC protocol.
- The logic for keep tracking partially done transactions and undoing the is kept on the database.
- However upon a restart before the database is allowed to access the storage volume, the storage does it own recovery process which is focused not on user-level TX but make sure database see a unified view of the storage despite it distributed nature.

- Writes
- LSN Limit is set to 10M so DB dont get much ahead of Storage.
- Also this(^) Introduce back-pressure and can throttle the incoming writes if the storage network cannot keep up.
- Commits
- In Aurora TX commits are completed Async
- Worker Threads do not pause per commit, they simple pull the pending requests and continue processing.
- Reads
- In aurora and like in most of databases, pages are served from the buffer cache.
- Only result in storage IO is the requested page is not present in the cache.
- IF the buffer cache is full, the system finds victim page to evict from cache.
- In traditional systems the viction is a "Dirty Page" then is flushed to the disk before replacement.
- This happens to ensure subsequent fetch of the page always result in latest data.
- The database does not need establish consensus using a read quorum under normal circumstances.
- Replicas
- In Aurora a single writer and up to 15 read replicas can all mount a single shared storage volume.
- Read replicas does not add additional cost in terms of consumed storage
- Recovery
- Most traditional databases use a recovery protocol such as ARIES that's depends of the presence of a WAL.
- Crash recovery can be a expensive operation. Reducing Checkpointing interval can help none trade-offs are required in Aurora.
- Aurora can recover pretty quickly(under 10s) and still process 100K RPS
- Aurora Architecture - Bird Eye view

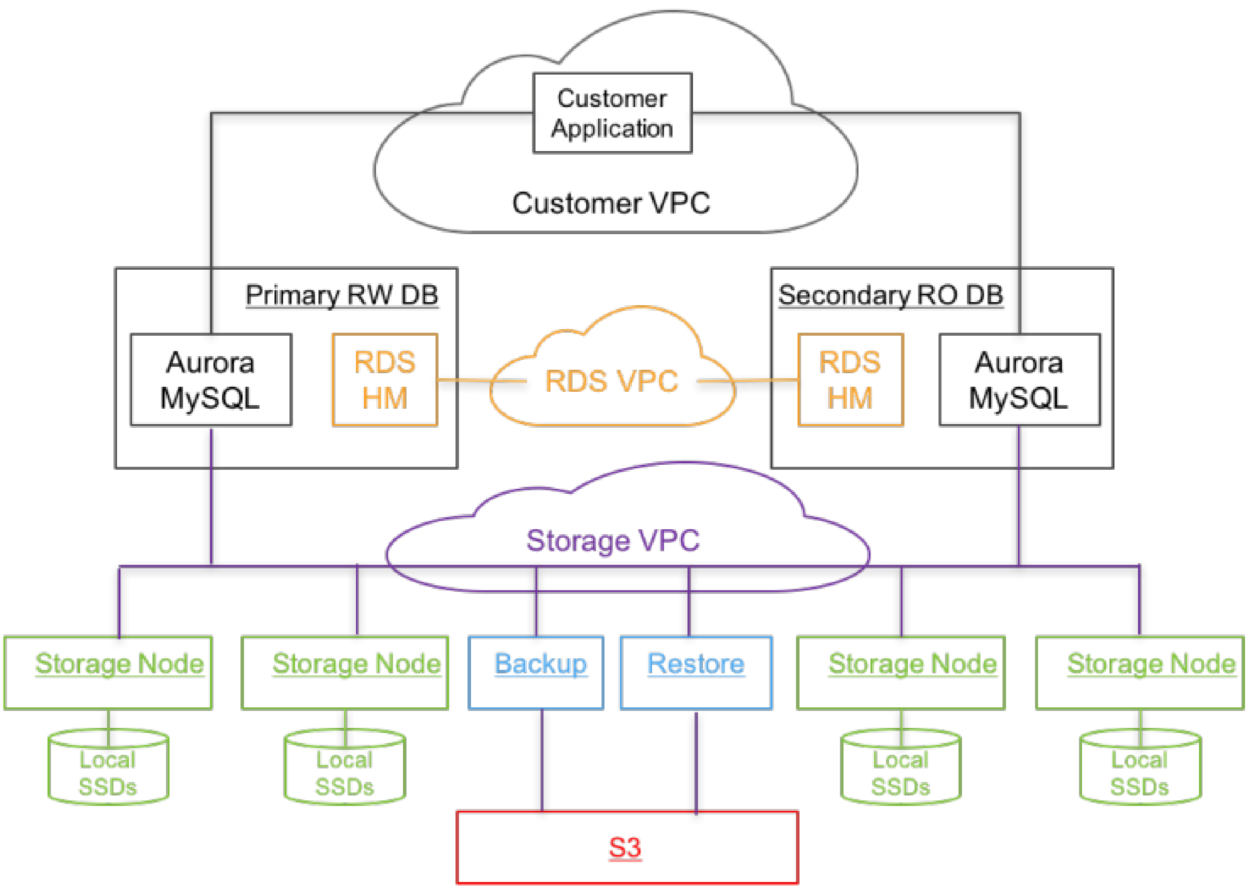

Figure 5: Aurora Architecture: A Bird's Eye View

- Aurora is a fork on MySQL InnoDB with difference on read and writes to the disk.
- AWS Aurora leverages AWS RDS as Control Plane.
- The Storage Control plane uses Amazon DynamoDB database service for persistent storage of cluster and storage volume configuration, volume metadata, detail description of data backup in s3.
- Benchmarks

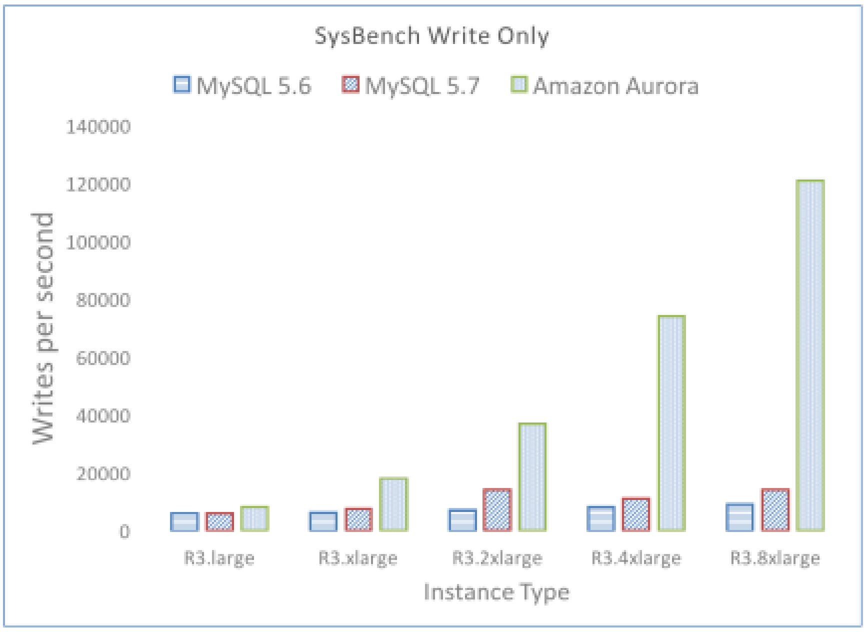

Figure 7: Aurora scales linearly for write-only workload

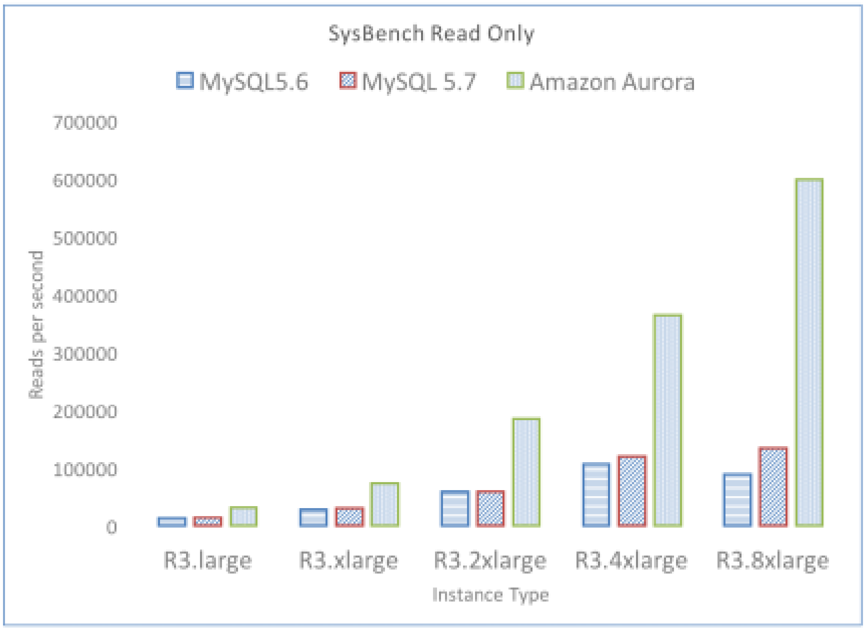

Figure 6: Aurora scales linearly for read-only workload

- Aurora can be 67X faster than MySQL with 100GB database.