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

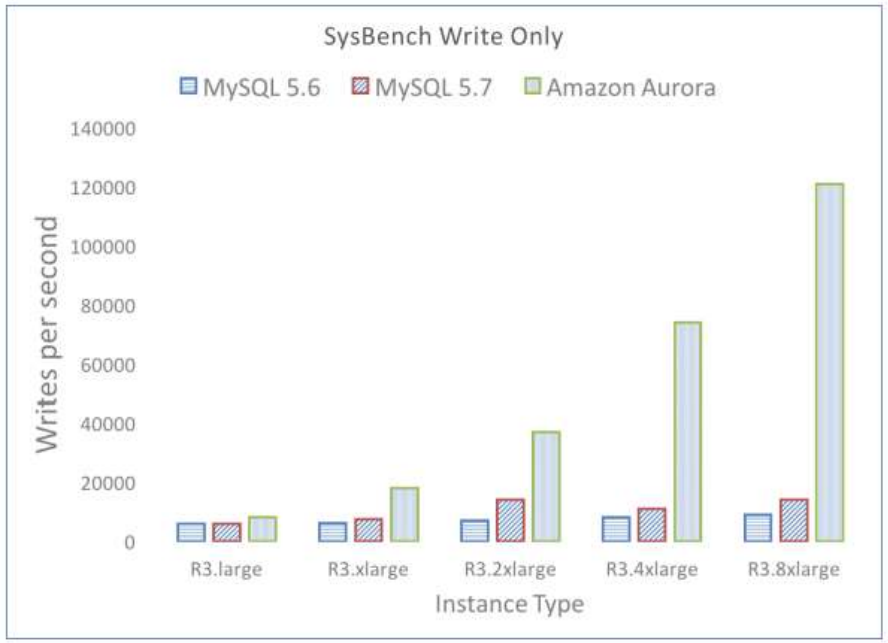


Figure 7: Aurora scales linearly for write-only workload

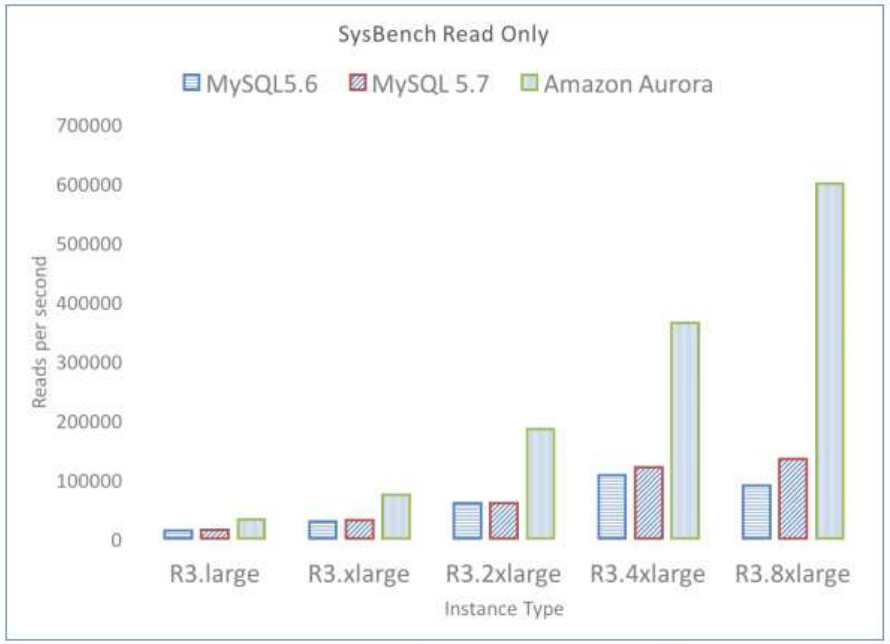


Figure 6: Aurora scales linearly for read-only workload

The I/O bottleneck faced by traditional database systems changes in this environment. Since I/Os can be spread across many nodes and many disks in a multi-tenant fleet, the individual disks and nodes are no longer hot.

Instead, the bottleneck moves to the network between the database tier requesting I/Os and the storage tier that performs these I/Os.

Although most operations in a database can overlap with each other, there are several situations that require synchronous operations. These result in stalls and context switches. One such situation is a disk read due to a miss in the database buffer cache. A reading thread cannot continue until its read completes.

A cache miss may also incur the extra penalty of evicting and flushing a dirty cache page to accommodate the new page. Background processing such as checkpointing and dirty page writing can reduce the occurrence of this penalty, but can also cause stalls, context switches and resource contention.

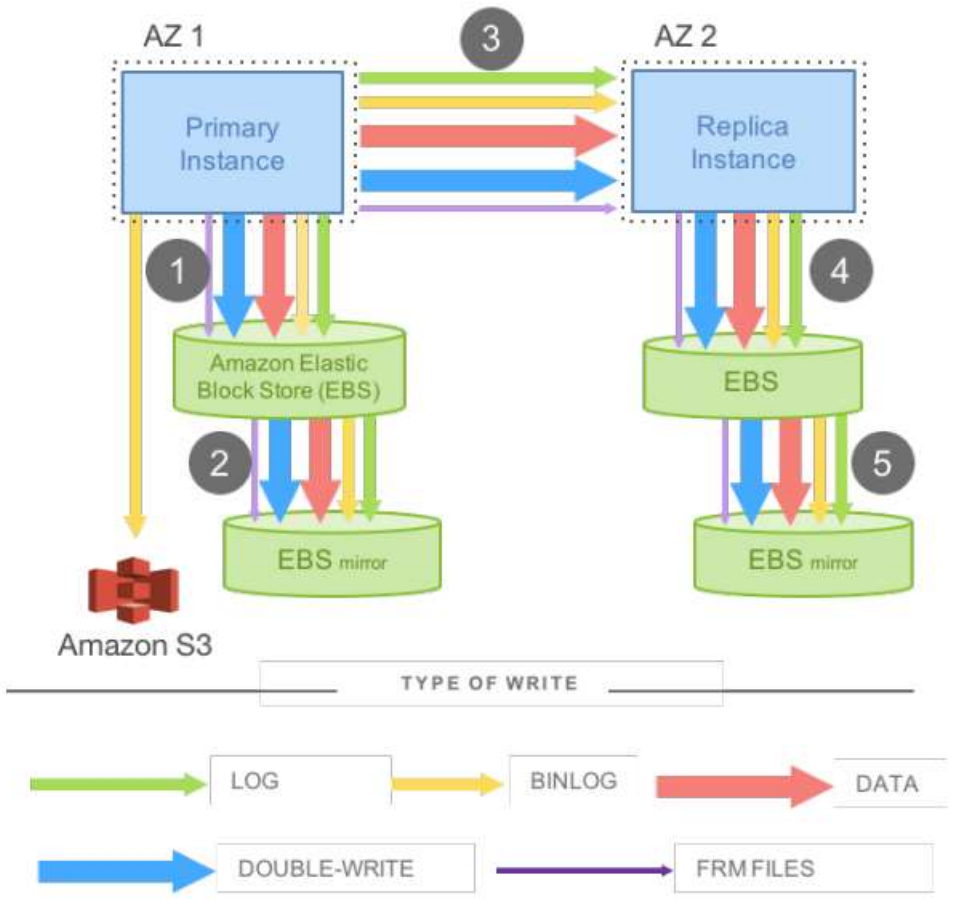
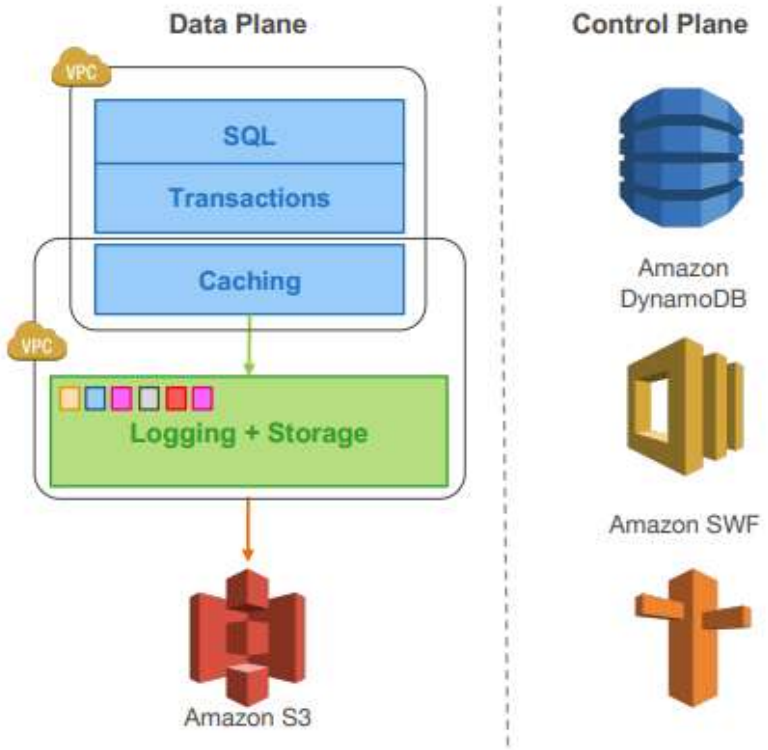


Figure 2: Network IO in mirrored MySQL

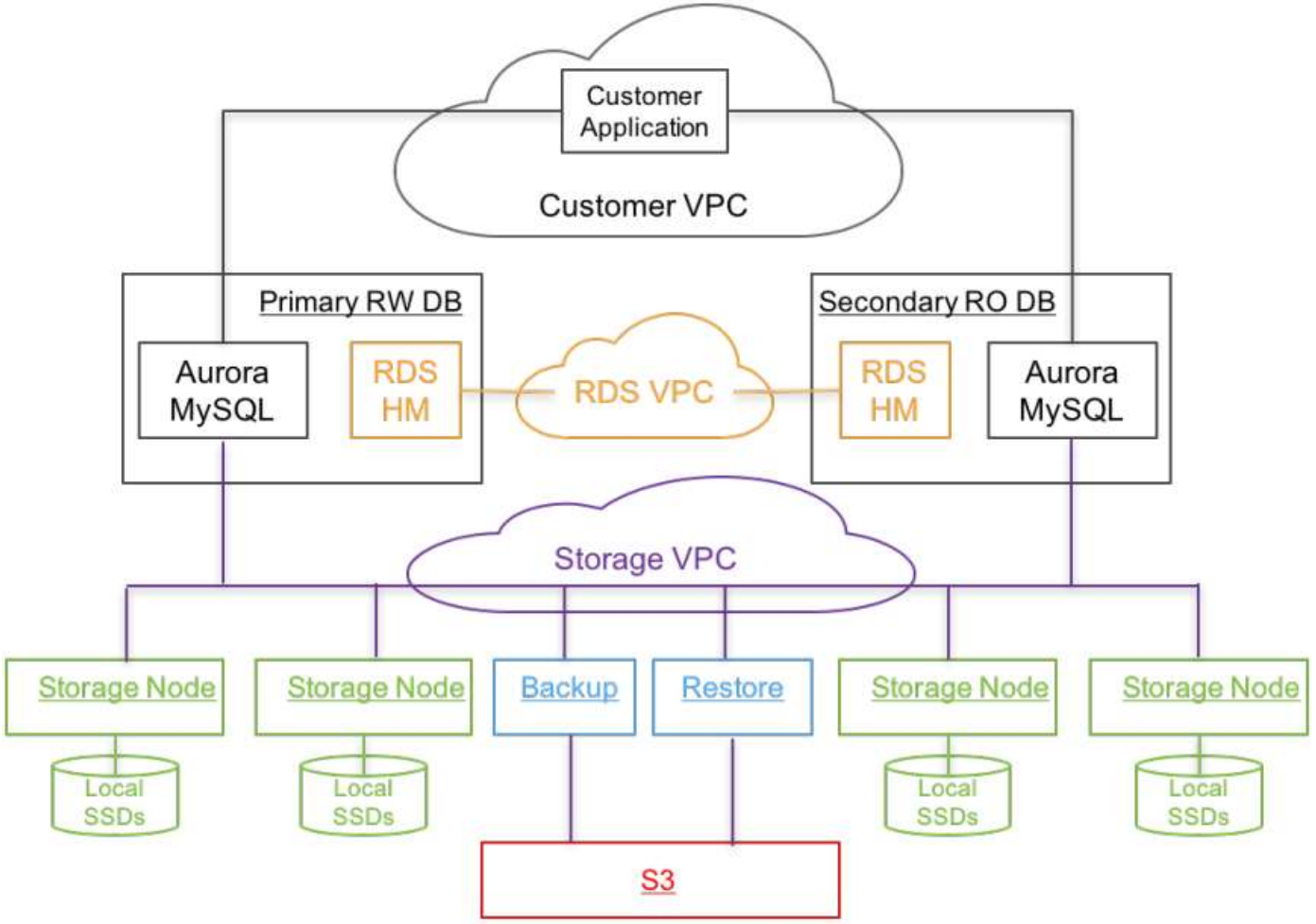
MySQL that generates many different actual I/Os for each application write. The high I/O volume is amplified by replication, imposing a heavy packets per second (PPS) burden. Also, the I/Os result in points of synchronization that stall pipelines and dilate latencies.

First, steps 1, 3, and 5 are sequential and synchronous. Latency is additive because many writes are sequential. Jitter is amplified because, even on asynchronous writes, one must wait for the slowest operation, leaving the system at the mercy of outliers.

Second, user operations that are a result of OLTP applications cause many different types of writes often representing the same information in multiple ways.



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 scaleout storage service, purpose-built for Aurora.


Customer Application
Customer VPC
Primary RW DB
Aurora MySQL
RDS HM
RDS VPC
Secondary RO DB
RDS HM
Aurora MySQL
Storage VPC
Storage Node
Storage Node
Backup
Restore
Storage Node
Storage Node
Local SSDs
Local SSDs
Local SSDs
Local SSDs
S3

In this paper, we describe three contributions:

- How to reason about durability at cloud scale and how to design quorum systems that are resilient to correlated failures. (Section 2).
- How to leverage smart storage by offloading the lower quarter of a traditional database to this tier. (Section 3).
- How to eliminate multi-phase synchronization, crash recovery and checkpointing in distributed storage (Section 4).

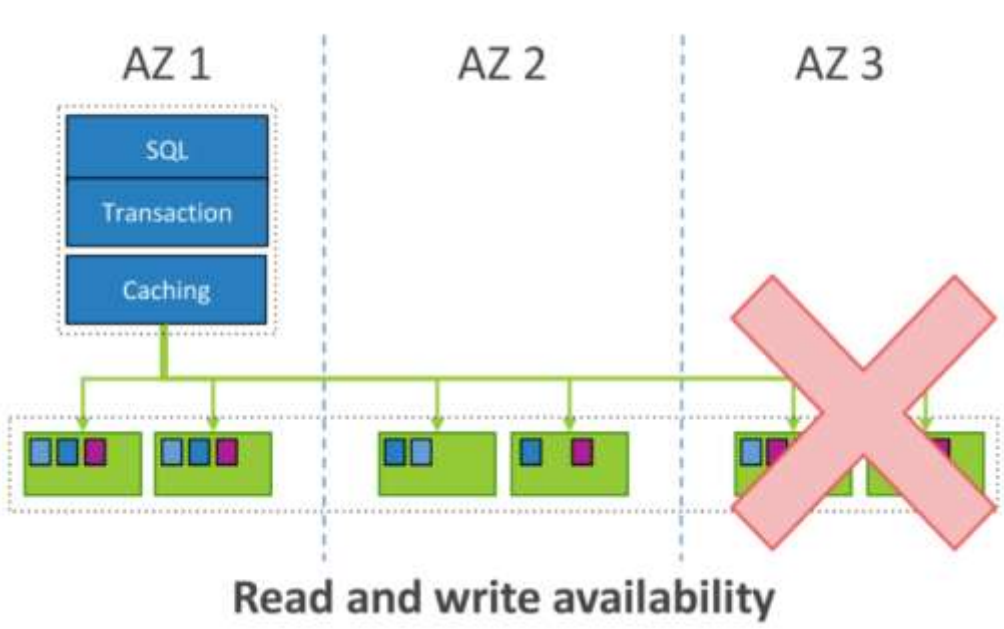
Quorums

One approach to tolerate failures in a replicated system is to use a quorum-based voting protocol as described in [6].

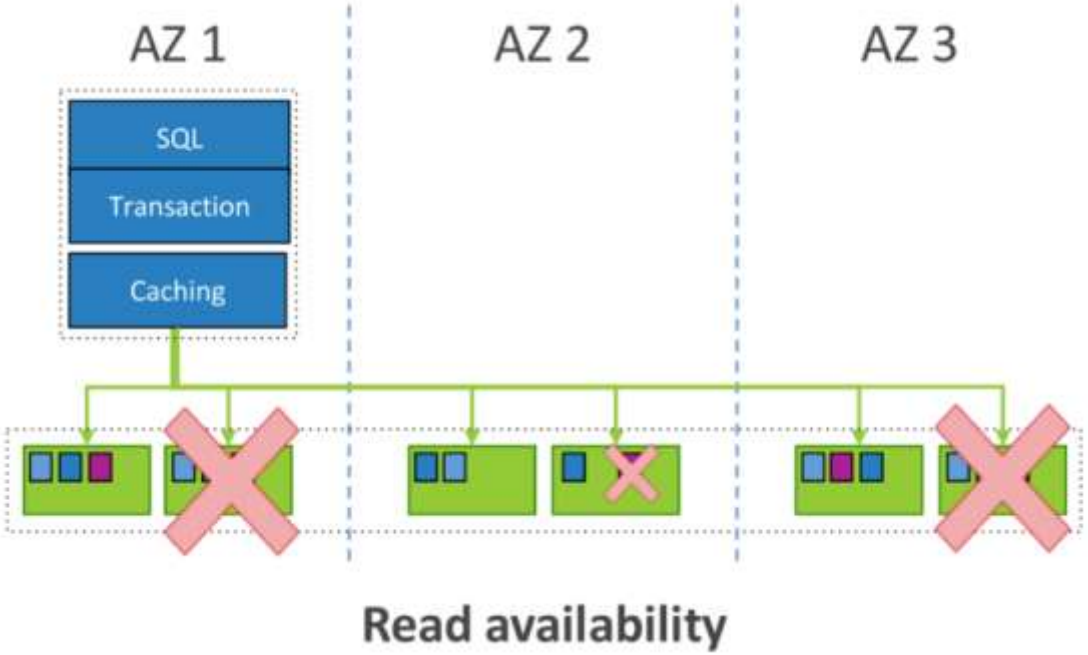
If each of the V copies of a replicated data item is assigned a vote, a read or write operation must respectively obtain a read quorum of Vr votes or a write quorum of Vw votes. To achieve consistency, the quorums must obey two rules. First, each read must be aware of the most recent write, formulated as Vr + Vw > V. This rule ensures the set of nodes used for a read intersects with the set of nodes used for a write and the read quorum contains at least one location with the newest version. Second, each write must be aware of the most recent write to avoid conflicting writes, formulated as Vw > V/2.

A common approach to tolerate the loss of a single node is to replicate data to (V = 3) nodes and rely on a write quorum of 2/3 (Vw = 2) and a read quorum of 2/3 (Vr = 2).

Quorums



Read and write availability



Read availability

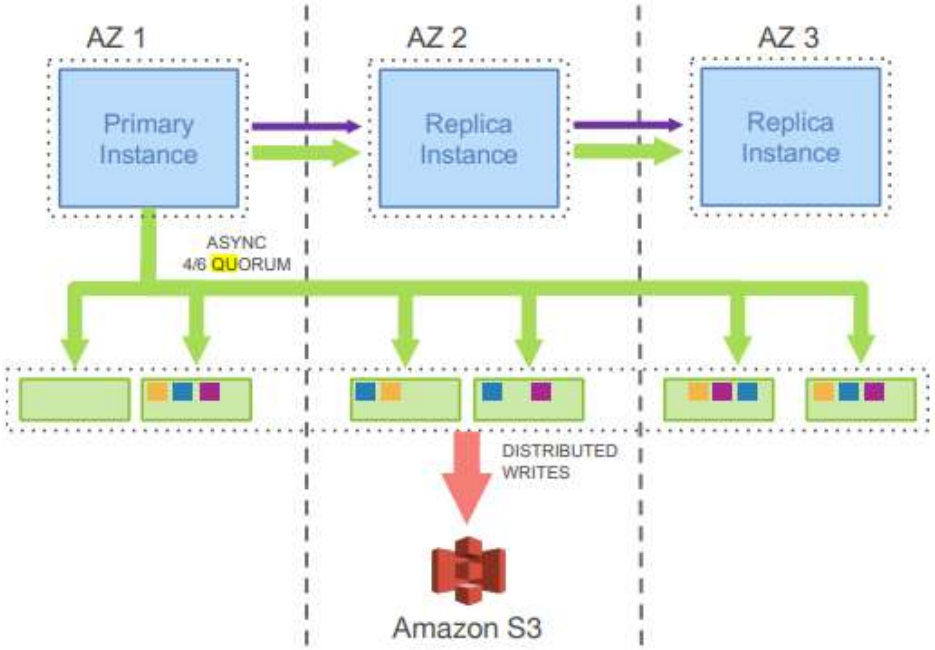
Segmented Storage

We instead focus on reducing MTTR to shrink the window of vulnerability to a double fault. We do so by partitioning the database volume into small fixed size segments, currently 10GB in size. These are each replicated 6 ways into Protection Groups (PGs) so that each PG consists of six 10GB segments, organized across three AZs, with two segments in each AZ.

Segments are now our unit of independent background noise failure and repair. We monitor and automatically repair faults as part of our service. A 10GB segment can be repaired in 10 seconds on a 10Gbps network link.

heat management
OS and security patching
software upgrades

THE LOG IS THE DATABASE



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. Instead, the log applicator is pushed to the storage tier where it can be used to generate database pages in background or on demand. We continually materialize database pages in the background to avoid regenerating them from scratch on demand every time.

The IO flow batches fully ordered log records based on a common destination (a logical segment, i.e., a PG) and delivers each batch to all 6 replicas where the batch is persisted on disk and the database engine waits for acknowledgements from 4 out of 6 replicas in order to satisfy the write quorum and consider the log records in question durable or hardened. The replicas use the redo log records to apply changes to their buffer caches.

IO Traffic

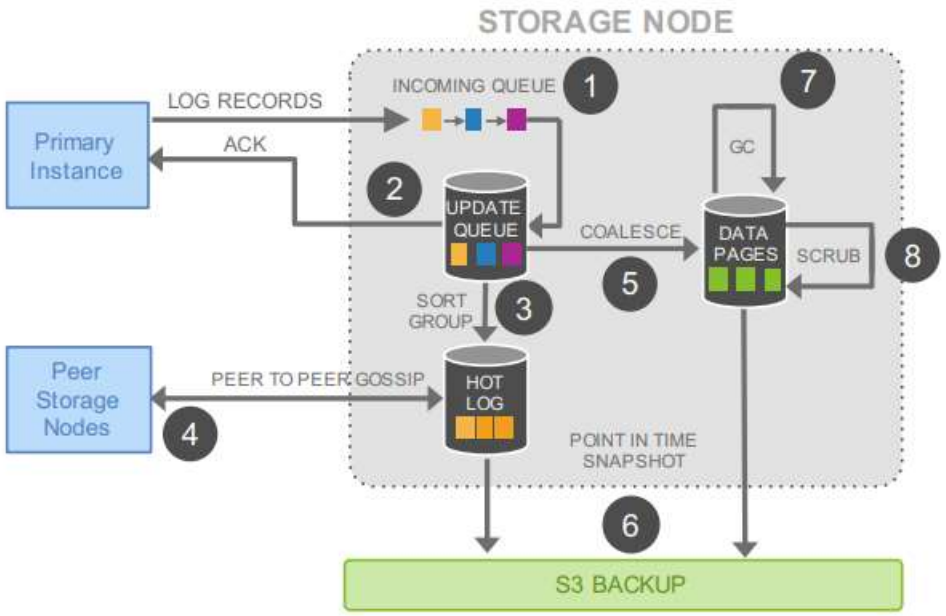


Figure 4: IO Traffic in Aurora Storage Nodes

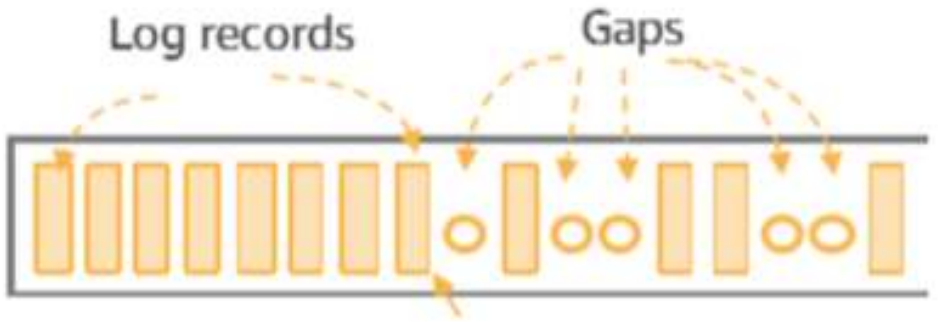
Let's examine the various activities on the storage node in more detail. As seen in Figure 4, it involves the following steps:
(1) receive log record and add to an in-memory queue,
(2) persist record on disk and acknowledge,
(3) organize records and identify gaps in the log since some batches may be lost,
(4) gossip with peers to fill in gaps,
(5) coalesce log records into new data pages,
(6) periodically stage log and new pages to S3,
(7) periodically garbage collect old versions, and finally
(8) periodically validate CRC codes on pages.

Note that not only are each of the steps above asynchronous, only steps (1) and (2) are in the foreground path potentially impacting latency.

THE LOG MARCHES FORWARD

In practice, each log record has an associated Log Sequence Number (LSN) that is a monotonically increasing value generated by the database.

At a high level, we maintain points of consistency and durability, and continually advance these points as we receive acknowledgements for outstanding storage requests. Since any individual storage node might have missed one or more log records, they gossip with the other members of their PG, looking for gaps and fill in the holes.



THE LOG MARCHES FORWARD

SCL: Note that each segment of each PG only sees a subset of log records in the volume that affect the pages residing on that segment. Each log record contains a backlink that identifies the previous log record for that PG. Segment Complete LSN (SCL) that identifies the greatest LSN below which all log records of the PG have been received.

VCL: Volume Complete LSN The storage service determines the highest LSN for which it can guarantee availability of all prior log.

CPL: The database can, however, further constrain a subset of points that are allowable for truncation by tagging log records and identifying them as Consistency Point LSNs.

VDL: We therefore define VDL or the Volume Durable LSN as the highest CPL that is smaller than or equal to VCL and truncate all log records with LSN greater than the VDL.

THE LOG MARCHES FORWARD

1. Each database-level transaction is broken up into multiple mini-transactions (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) that are ordered and must be performed atomically.

2. Each mini-transaction is composed of multiple contiguous log records (as many as needed).

3. The final log record in a mini-transaction is a CPL.

主机生成的事务日志	LSN,每个日志记录生成的唯一日志序列号			LSN单调递增			CPL,每个Mini事务产生的最后一个LSN为一个CPL即一致性点		
			CPL		CPL	CPL		CPL	CPL
	LSN1	LSN2	LSN3	LSN4	LSN5	LSN6	LSN7	LSN8	LSN9
	T1-Mini-t1			T2-Mini-t1		T1-Mini-t2	T3-Mini-t1		T2-Mini-t2
	T1			T2		T1	T3		T2
	一个事务可有多个Mini事务，每个Mini事务可有多个日志记录。同一个事务的日志未必相邻，但每个Mini事务的最后一个日志的LSN就是一个CPL								

网络 网络 网络									
存储节点S1	VDL，持久化点。事务提交，在VDL点之前的日志可以被作为垃圾被回收					这个VCL是6个存储节点中公共的最大点	VCL，所有存储节点接收到的最大连续日志LSN		
			VDL			VCL			SCL
	LSN1	LSN2	LSN3	LSN4	LSN5	LSN6	LSN7	LSN8	LSN9
S2	LSN1	LSN2	LSN3	LSN4	LSN5	LSN6	LSN7	LSN8	LSN9
S3	LSN1	LSN2	LSN3	LSN4	LSN5	LSN6	LSN7	LSN8	LSN9
S4	LSN1	LSN2	LSN3	LSN4	LSN5	LSN6	LSN7	LSN8	LSN9
S5	LSN1	LSN2	LSN3	LSN4	LSN5	LSN6	LSN7	gap	gap
S6	LSN1	LSN2	LSN3	LSN4	LSN5	LSN7	gap	LSN8	gap
	将被GC						运行期间gap将被Gossip协议添补； 恢复期间，大于VCL部分将被TRUNCATE掉		

Normal Operation

Writes: In the normal/forward path, as the database receives acknowledgements to establish the write quorum for each batch of log records, it advances the current VDL. At any given moment, there can be a large number of concurrent transactions active in the database, each generating their own redo log records. The database allocates a unique ordered LSN for each log record.

Commits: When a client commits a transaction if and only if, the latest VDL is greater than or equal to the transaction's commit LSN. As the VDL advances, the database identifies qualifying transactions that are waiting to be committed and uses a dedicated thread to send commit acknowledgements to waiting clients.

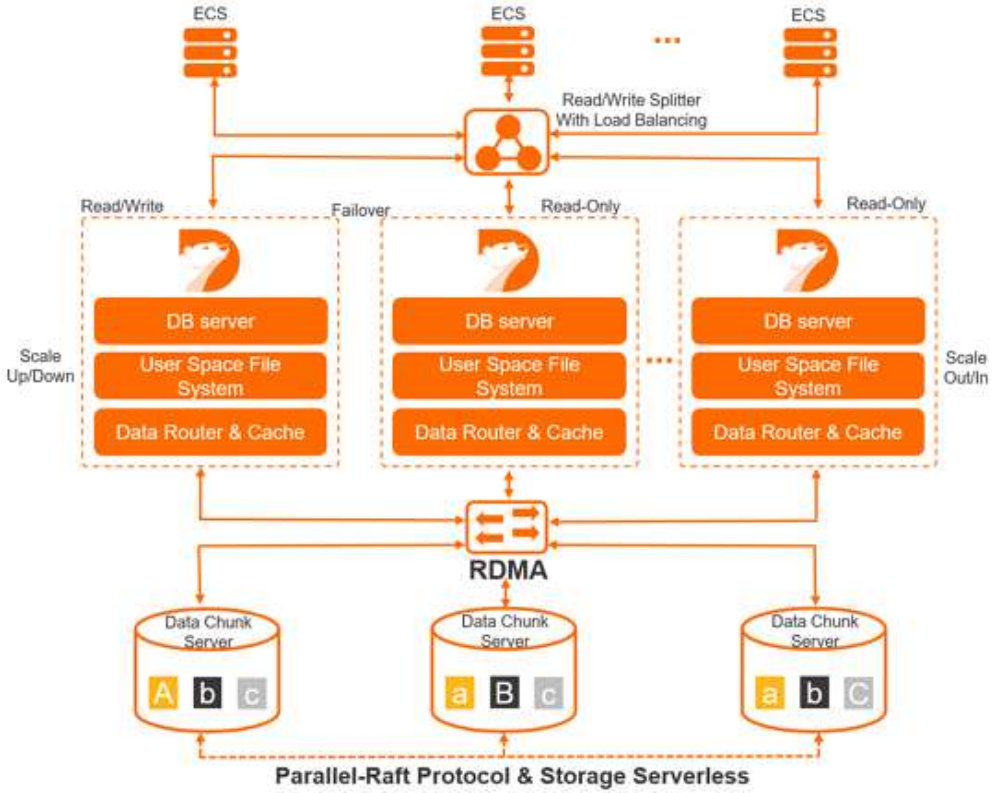
Reads: When reading a page from disk, the database establishes a read-point, representing the VDL at the time the request was issued. The database can then select a storage node that is complete with respect to the read point, knowing that it will therefore receive an up to date version.

Replicas

To minimize lag, the log stream generated by the writer and sent to the storage nodes is also sent to all read replicas. In the reader, the database consumes this log stream by considering each log record in turn. If the log record refers to a page in the reader's buffer cache, it uses the log applicator to apply the specified redo operation to the page in the cache. Otherwise it simply discards the log record. Note that the replicas consume log records asynchronously from the perspective of the writer, which acknowledges user commits independent of the replica. The replica obeys the following two important rules while applying log records: (a) the only log records that will be applied are those whose LSN is less than or equal to the VDL, and (b) the log records that are part of a single mini-transaction are applied atomically in the replica's cache to ensure that the replica sees a consistent view of all database objects. In practice, each replica typically lags behind the writer by a short interval (20 ms or less).

比较项	Spanner	Aurora
架构	Shared Nothing	Shared Disk
弹性	Yes	Yes
持续可用	Paxos-based	Quorum-based
写扩展	Yes	No
分库分表透明	Yes	No
分布式事务	Yes	No
MySQL/Postgres兼容	No	Yes

Aurora的优势在于，它改变了传统关系型数据库的读写框架，使其适用于弹性伸缩的云端，获得了可用性和持久性，同时降低了网络传输，使得整个系统响应低延时。上云之后，还有更多的优点，例如按需伸缩，多租户共享基础设施降低成本，故障能够快速恢复等。为此它继承了社区版MySQL的一些弱点，比如不能执行复杂的查询，单点写，写水平扩展需要依赖其它中间件方案。同时目前最大支持64TB的数据量。


ECS
ECS
ECS
Read/Write Splitter
With Load Balancing
Read/Write
Failover
Read-Only
Read-Only
DB server
User Space File System
Data Router & Cache
DB server
User Space File System
Data Router & Cache
DB server
User Space File System
Data Router & Cache
Scale Up/Down
Scale Out/In
RDMA
Data Chunk Server
Data Chunk Server
Data Chunk Server
Parallel-Raft Protocol & Storage Serverless


客户端
客户端
客户端
负载均衡
数据库代理中间件
数据库代理中间件
数据库代理中间件
读/写
读
主节点
故障转移
只读节点
只读节点
日志 I/O
数据页面 I/O
日志存储
页面存储
veDB MySQL实例

THANKS

And Your Slogan Here

Speaker name and title

www.uestc.edu.cn