A Survey on Logging and Resilience System in Cloud-native Database

Huiyu Bao, Wendi Li, Zhikang Hao

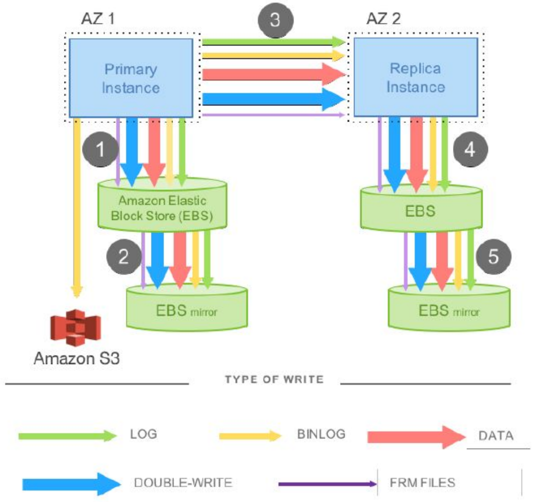
Motivation

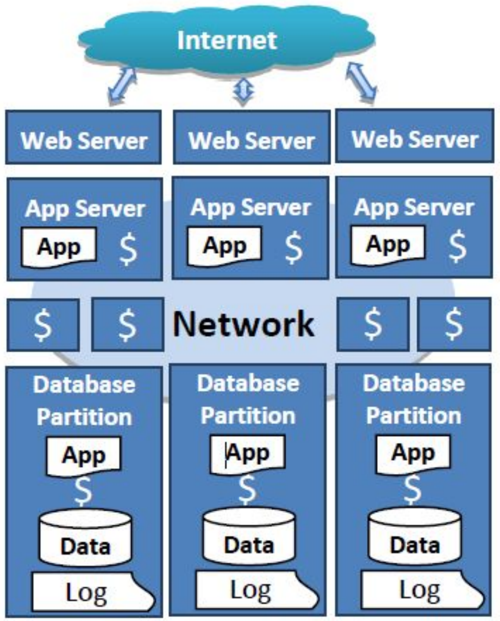
- crucial part of database system
- help database system fast recover to normal status
- different from tranditional database

Background

tranditional database

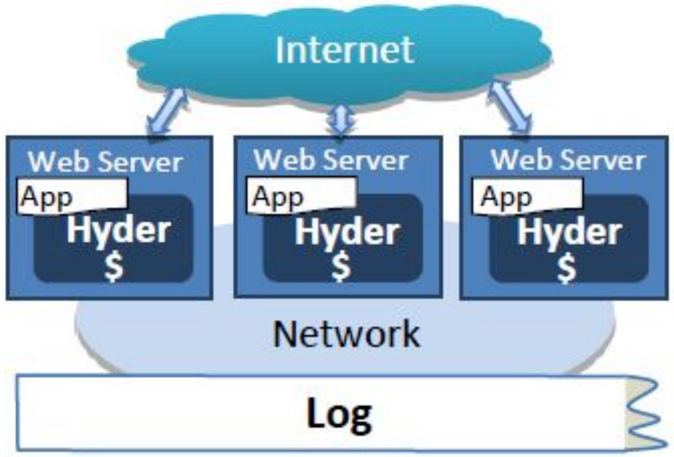
- system R: shadow paging strategy, makes a copy when updating
- ARIES: no-force, steal strategy
- bring in lots of writing and network traffic
- also need more time to recover





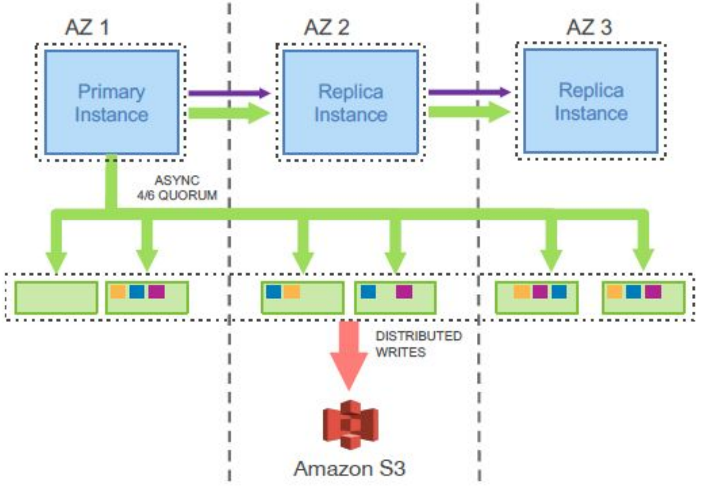
Microsoft Hyder

- avoiding partition, distributed programming, layers of caches, load balancing
- data-sharing architecture, all servers read and write from entire db
- each update transaction executes on one machine and writes to shared log
- each log record is a multi-page stripe, log operations includes AppendStripe and Get Stripe



Amazon Aurora

- only writes that cross the network are redo log records
- log applicator is pushed to the storage tier to generate db pages
- log system help storage nodes minimize latency of foreground write
- normal db operation, redo log stream



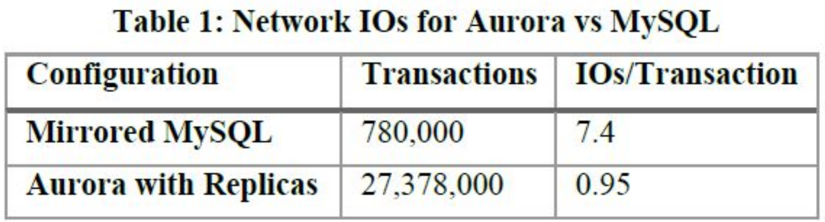
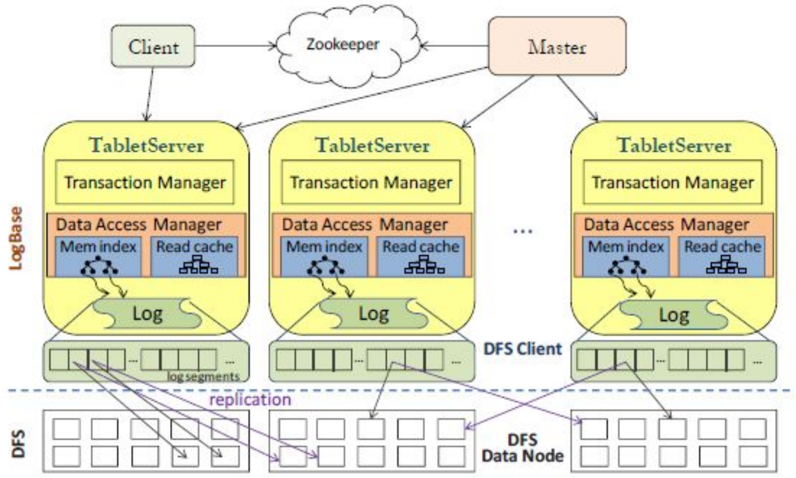


Table 1: Network IOs for Aurora vs MySQL

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

LogBase

- a log-structured database, use append-only files to implement its log segments
- provides a db abstraction on top of segmented log, fine-grained access
- bottom layer of structure: log repository
- tablet servers employ a shared distributred file system to maintain log data,
- also provide fault-tolerance when failures happen

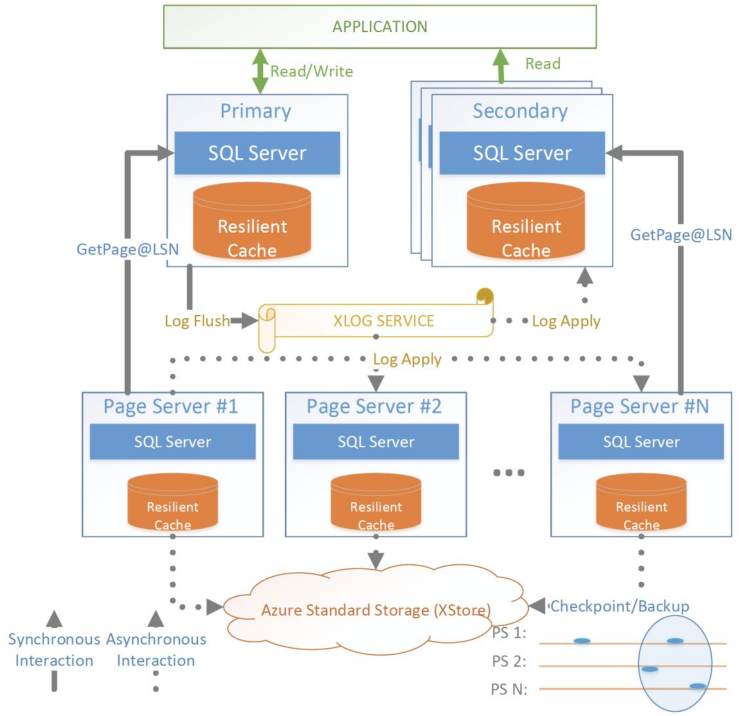


CockroachDB

- uses range-partitioning on the keys to divide the data into contiguous ordered chunks of size ~64 MiB, that are stored across the cluster (called Range).
- maintains a consistent, ordered log of updates across a Range's replicas.
- each replica individually applies commands to the storage engine as Raft declares them to be committed to the Range's log.
- uses Range-level leases, only one replica is allowed to serve authoritative up-to-date reads or propose writes.

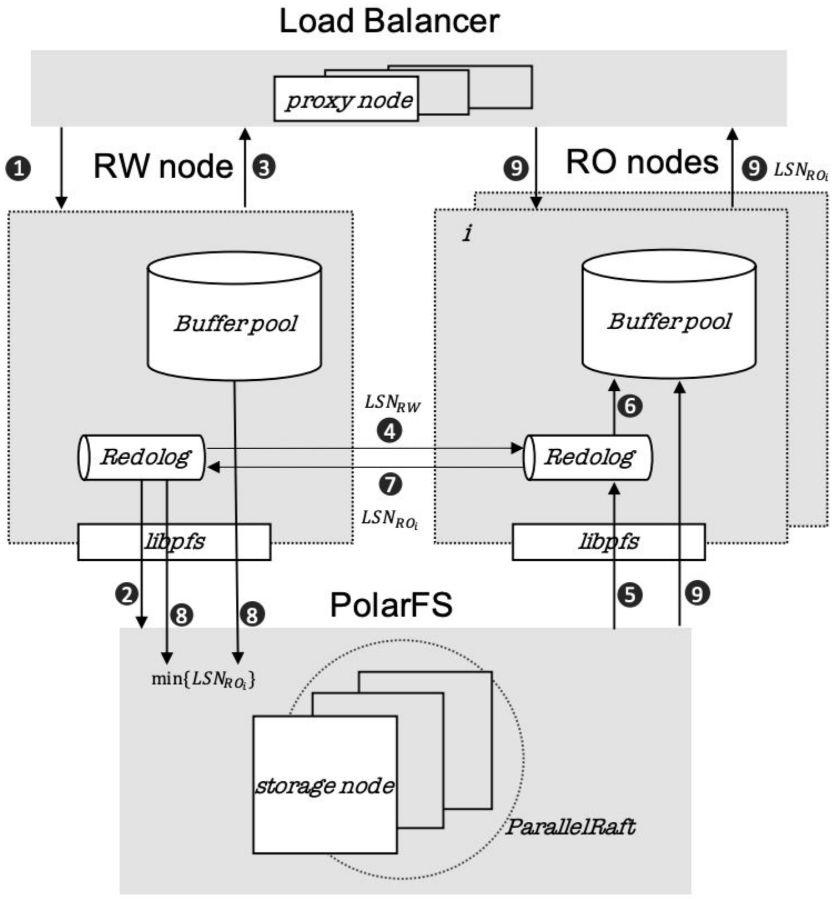
Socrates

- separates database log from storage and treats the log as a first-class citizen.
- with a separate logging service, we can tune and tailor the log specifically to its specific access pattern.
  - Socrates makes the log durable and fault-tolerant by replicating the log
  - asymmetry of log access: Recently created log records are in high demand whereas old log records are only needed in exceptional cases. Socrates keeps recent log records in main memory and distributes them in a scalable way (potentially to hundreds of machines) whereas old log records are destaged and made available only upon demand.



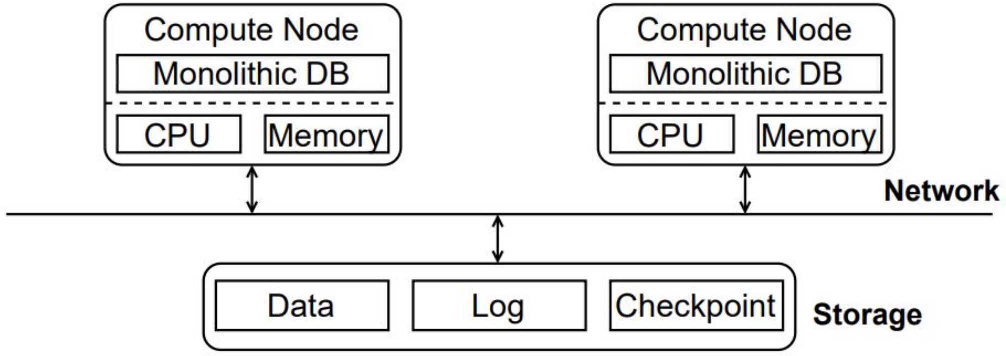
PolarDB

- PolarDB adopts a similar approach, which is closer to Socrates.
- logs and pages are separately stored in two types of chunks (i.e., log chunk and page chunk).
- Redo logs are first persisted to log chunks and then asynchronously sent to page chunks, where logs are applied to update pages.



LegoBase

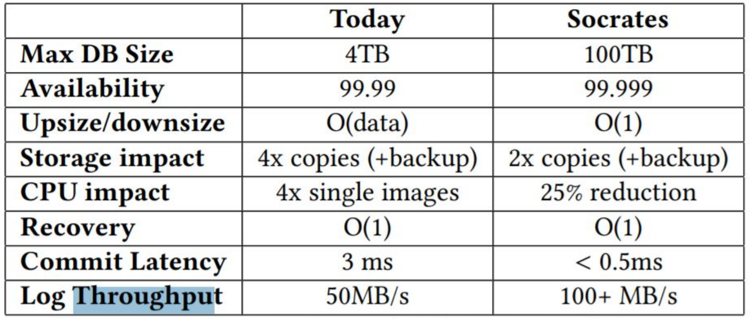
- Logs are replicated to a collection of replicas, and a consensus protocol named ParallelRaft is used to guarantee the data consistency among replicas.
- An I/O request is not recognized to be committed until it is persistently recorded to the logs on a majority of replicas.



Socrates & PolarDB

Socrates transforms the on-premise SQL Server into a DBaaS and further separates logging and storage from database kernel to dedicated services.

PolarDB Serverless allows nodes of each type to failover independently. It includes one primary and multiple read replicas. PolarDB Serverless adopts an ARIES-style recovery algorithm, which makes failed read replicas can be easily replaced with a new one using pages in the shared memory.



	Today	Socrates
Max DB Size	4TB	100TB
Availability	99.99	99.999
Upsize/downsize	O(data)	O(1)
Storage impact	4x copies (+backup)	2x copies (+backup)
CPU impact	4x single images	25% reduction
Recovery	O(1)	O(1)
Commit Latency	3 ms	< 0.5ms
Log Throughput	50MB/s	100+ MB/s

Table 1: Socrates Goals: Scalability, Availability, Cost

Comparison

CockRoachDB:

Socrates:

Aurora: Consistent Log, single truth over all nodes

LogBase: Decreases latency of interesting by removing log phase but adding compacting phase which increase reading throughput

Evaluations & Analysis

We will use the data from the papers to evaluate logging and their trend.

In the future we will implement the database on cloud (e.g. CoLab) and evaluate the logging by benchmarking the database with regular queries and recovery using TPC/YCSB. We are going to record the latency and throughput for each available implementation and identify the effect of logging for each approach.

Evaluation

The evaluation involves:

Logging throughput

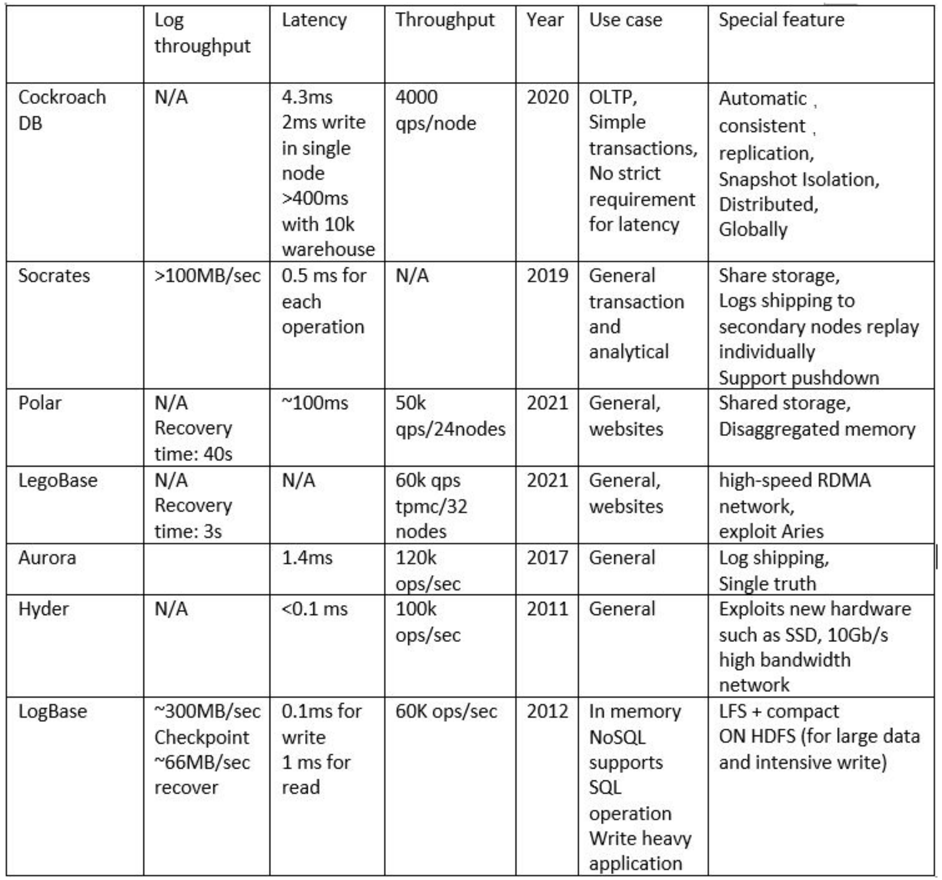
Latency of execution (single operation)

Throughput

Use case

Year

features



	Log throughput	Latency	Throughput	Year	Use case	Special feature
Cockroach DB	N/A	4.3ms 2ms write in single node >400ms with 10k warehouse	4000 qps/node	2020	OLTP, Simple transactions, No strict requirement for latency	Automatic , consistent , replication, Snapshot Isolation, Distributed, Globally
Socrates	>100MB/sec	0.5 ms for each operation	N/A	2019	General transaction and analytical	Share storage, Logs shipping to secondary nodes replay individually Support pushdown
Polar	N/A Recovery time: 40s	~100ms	50k qps/24nodes	2021	General, websites	Shared storage, Disaggregated memory
LegoBase	N/A Recovery time: 3s	N/A	60k qps tpmc/32 nodes	2021	General, websites	high-speed RDMA network, exploit Aries
Aurora		1.4ms	120k ops/sec	2017	General	Log shipping, Single truth
Hyder	N/A	<0.1 ms	100k ops/sec	2011	General	Exploits new hardware such as SSD, 10Gb/s high bandwidth network
LogBase	~300MB/sec Checkpoint ~66MB/sec recover	0.1ms for write 1 ms for read	60K ops/sec	2012	In memory NoSQL supports SQL operation Write heavy application	LFS + compact ON HDFS (for large data and intensive write)

Discussion

The evolution of logging system design is closely related to architecture, hardware, file system and workload.

It is necessary to visit logging system design problem regularly.

Q&A