Evaluating CockroachDB vs YugabyteDB

PostgreSQL features, architecture, benchmarks

Karthik Ranganathan, Co-founder/CTO, Yugabyte

Distributed SQL databases

SQL capabilities + resilient to failures + scalable + geo-distributed

yugabyteDB vs CockroachDB

yugabyteDB is a **distributed SQL database** built for:

- high performance (low Latency)
- cloud native (run on Kubernetes, VMs, bare metal)
- open source (Apache 2.0)

Evaluation Criteria

- RDBMS feature support
- Performance - using YCSB
- At-scale performance
- Architectural takeaways
- Licensing model

In parallel, we'll also look at architectural differences.

RDBMS Feature Support

Both DBs support PostgreSQL wire-protocol

However, there are architectural differences

yugabyteDB

- Reuses PostgreSQL codebase

Cockroach DB

- Rewritten SQL layer

Reusing PostgreSQL vs Rewriting

Feature	CockroachDB v19.2	YugabyteDB v2.1
Expression-based indexes	✗	✓
Partial indexes	✗	✓
Table functions	✗	✓
Stored procedures (SQL, pl-pgsql)	✗	✓
Advanced operators and built-ins (multidimensional arrays, `jsonb_agg()`, `jsonb_to_record()`, etc.)	✗	✓
Triggers	✗	✓
User-defined types	✗	✓
Temporary tables	✗	✓
Row level security	✗	✓
Column level privileges	✗	✓
Support for PostgreSQL extensions	✗	✓ *

**YugabyteDB only reuses the query layer of PostgreSQL, so it currently only supports extensions that use the query layer.*

Cockroach Labs blog post:

Yugabyte uses PostgreSQL for SQL optimization, and a portion of execution.

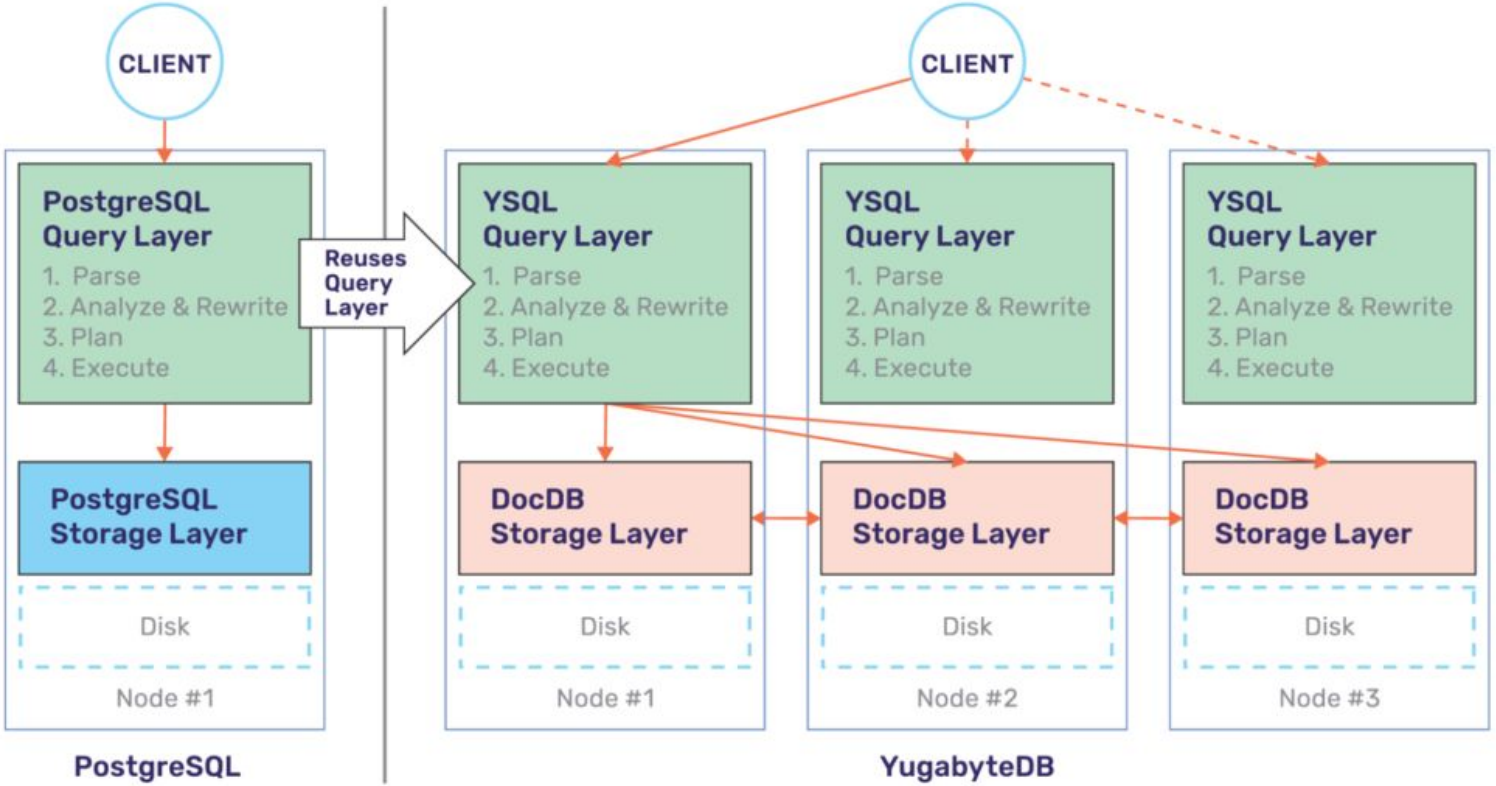
Reusing PostgreSQL results in monolithic SQL architecture

How YugabyteDB is architected:

Enhancing PostgreSQL to a distributed architecture is being accomplished in three phases:

- SQL layer on distributed DB
- Perform more SQL pushdowns
- Enhance optimizer

Phase #1 - SQL layer on distributed DB



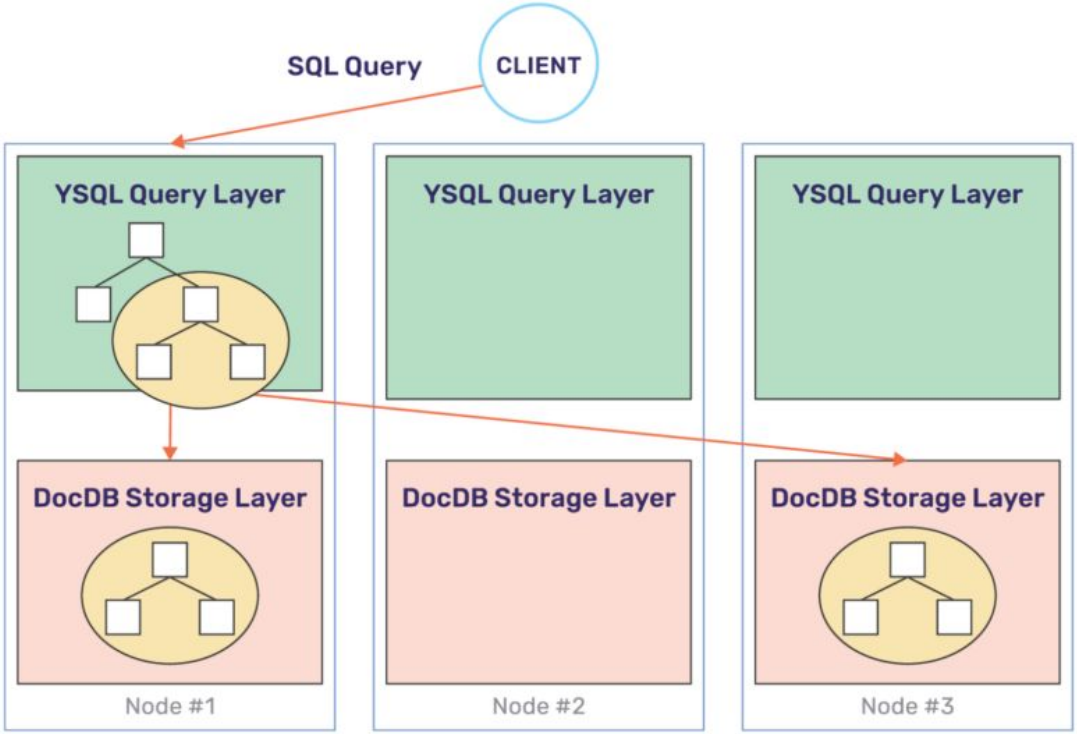
Phase #2: Perform SQL Pushdowns

1. Build PostgreSQL execution tree

2. Identify subtree to pushdown

3. Identify nodes to send pushdown to

4. Execute subtree using PostgreSQL libraries on DocDB storage layer



Generic pushdown mechanism in YugabyteDB

Phase #3: Enhance PostgreSQL Optimizer

- **Table statistics based hints**
  - Piggyback on current PostgreSQL optimizer that uses table statistics
- **Geographic location based hints**
  - Based on “network” cost
  - Factors in network latency between nodes and tablet placement
- **Rewriting query plan for distributed SQL**
  - Extend PostgreSQL “plan nodes” for distributed execution

Advantages of reusing PostgreSQL

- Support advanced RDBMS features
- Robust design, code, documentation by PostgreSQL
- Keep quality high

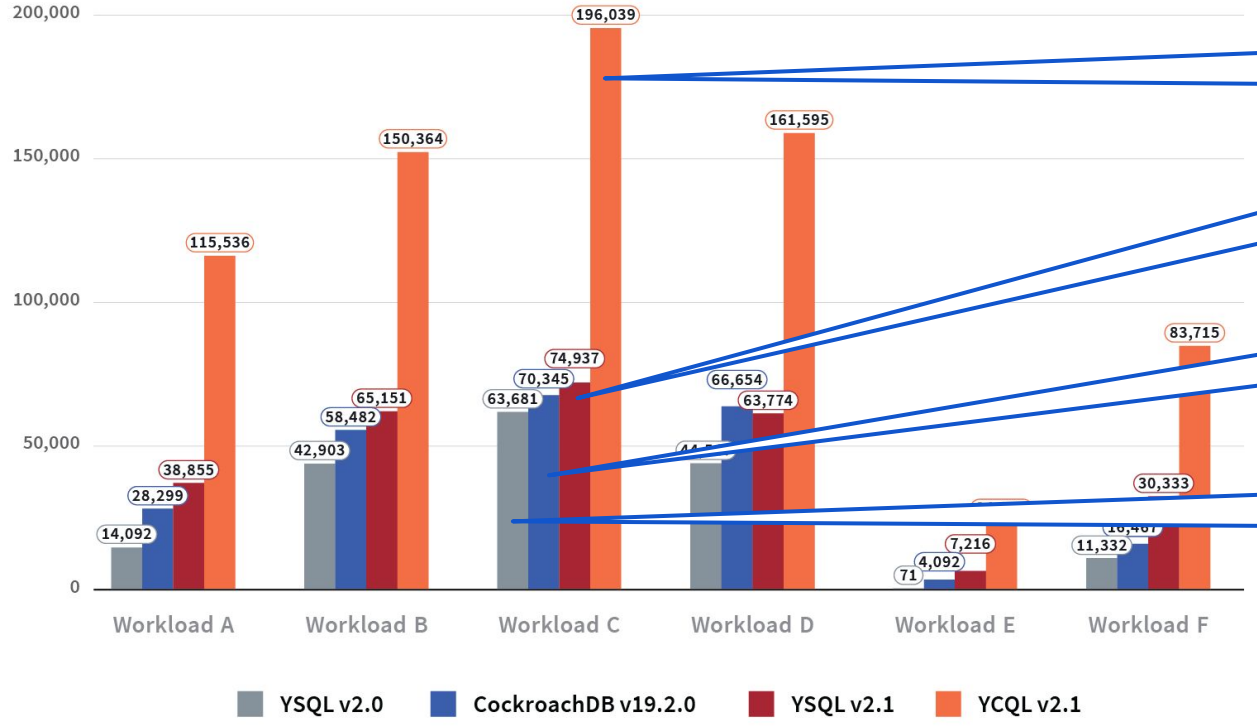
Regression Tests	PostgreSQL	YugabyteDB (current coverage)
Files	192	114 (59%)
Lines	68,803	33,459 (49%)
SQL Statements	29,292	14,943 (51%)

PostgreSQL regression tests currently in YugabyteDB.

Aim: get to 100% coverage

Performance
Using YCSB

YCSB Benchmark Comparison



YCQL 2.1 results
(reported by Yugabyte with standard driver)

YSQL 2.1 results
(reported by Yugabyte with custom driver)

CRDB v19.2.0 results
(reported by Cockroach Labs with custom driver)

YSQL 2.0 results
(reported by Cockroach Labs)

YugabyteDB takeaways

- YSQL perf dramatically increased from v2.0 to v2.1
- YCQL performance is much higher
- Over time, YSQL perf will match that of YCQL

Overall takeaways

- Issue #1: vendor-specific benchmarks are hard to run
- Issue #2: These benchmarks have too little data
- Repeat at scale:
  - Use standard YCSB driver with JDBC binding
  - Use large datasets to understand perf at scale

Performance at scale

YCSB with 450M rows

Benchmark details

450M rows (~1.3TB) loaded using the YCSB benchmark

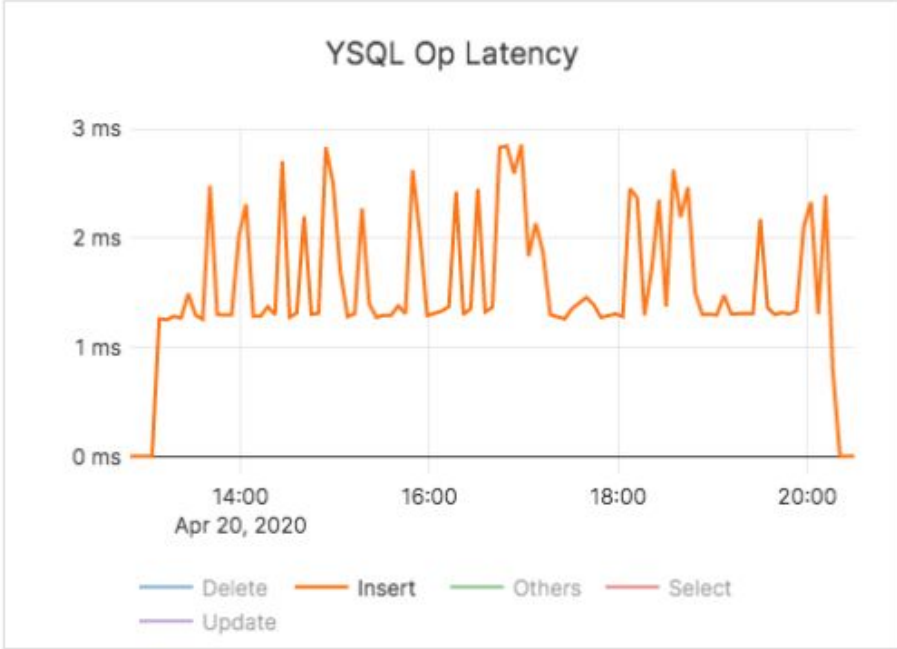
- 3 node cluster, replication factor = 3 (in AWS, us-west-2)
- Each node was c5.4xlarge (16 vCPU, 2 x 5TB gp2 EBS SSD)
- CockroachDB v19.2.6 (range sharding, hash not GA at the time)
- YugabyteDB v2.1 (using YSQL, range and hash sharding)
- Default isolation levels for both DBs

Loading Data - YugabyteDB was 3x faster

	CockroachDB (range)	YSQL (range)	YSQL (hash)
Time to load 450M rows	25h 45m 14s	9h 12m 31s	7h 9m 25s

CockroachDB throughput drops over time

YugabyteDB throughput



yugabyteDB

- Reuses PostgreSQL codebase
- RocksDB enhanced
- C/C++ for higher perf
- Sustains high write throughput

Cockroach DB

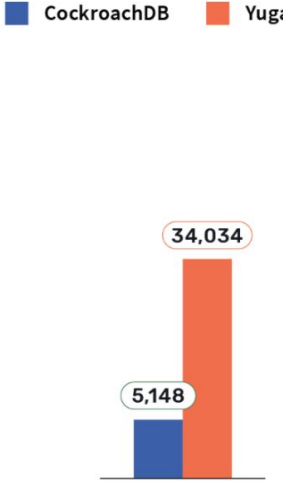
- Newly written SQL layer
- RocksDB blackbox
- Go/C++ (inter-language switch)
- Write throughput drops over time

YugabyteDB delivers 3x higher throughput on average

— Update Heavy Workload —

Example: Session store recording recent actions

Higher is better

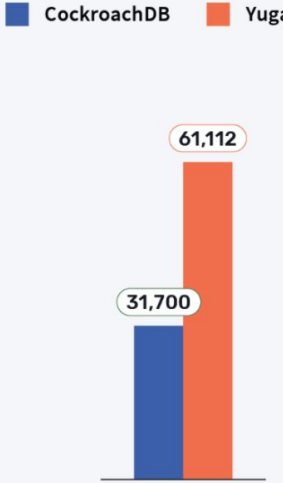


Workload A: 50% SELECT / 50% UPDATE
YCSB benchmark, 450M rows

— Read Only Workload —

Example: User profile cache

Higher is better

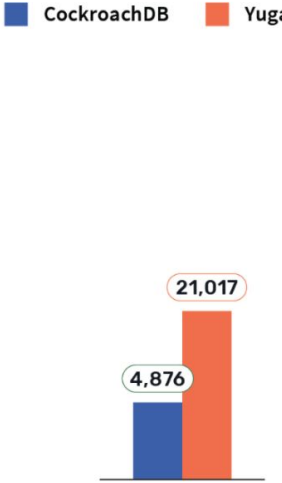


Workload C: 100% SELECT
YCSB benchmark, 450M rows

— Read-Modify-Write-Workload —

Example: User database

Higher is better



Workload F: Read-modify-write
YCSB benchmark, 450M rows

YugabyteDB delivers 4.5x lower latency on average

— Update Heavy Workload —

Example: Session store recording recent actions

Lower is better

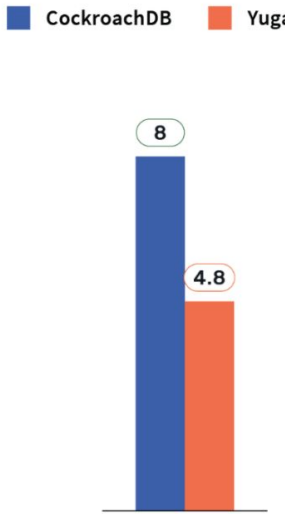
Update Latency
(milliseconds)

Workload A: 50% SELECT / 50% UPDATE
YCSB benchmark, 450M rows

— Read Only Workload —

Example: User profile cache

Lower is better



Read Latency
(milliseconds)

Workload C: 100% SELECT
YCSB benchmark, 450M rows

— Read-Modify-Write-Workload —

Example: User database

Lower is better

Read-Modify-Write Latency
(milliseconds)

Workload F: Read-modify-write
YCSB benchmark, 450M rows

yugabyteDB

- Reuses PostgreSQL codebase
- Sustains high write throughput
- Higher throughput, lower latency
  - RocksDB enhanced
  - C/C++ for higher perf

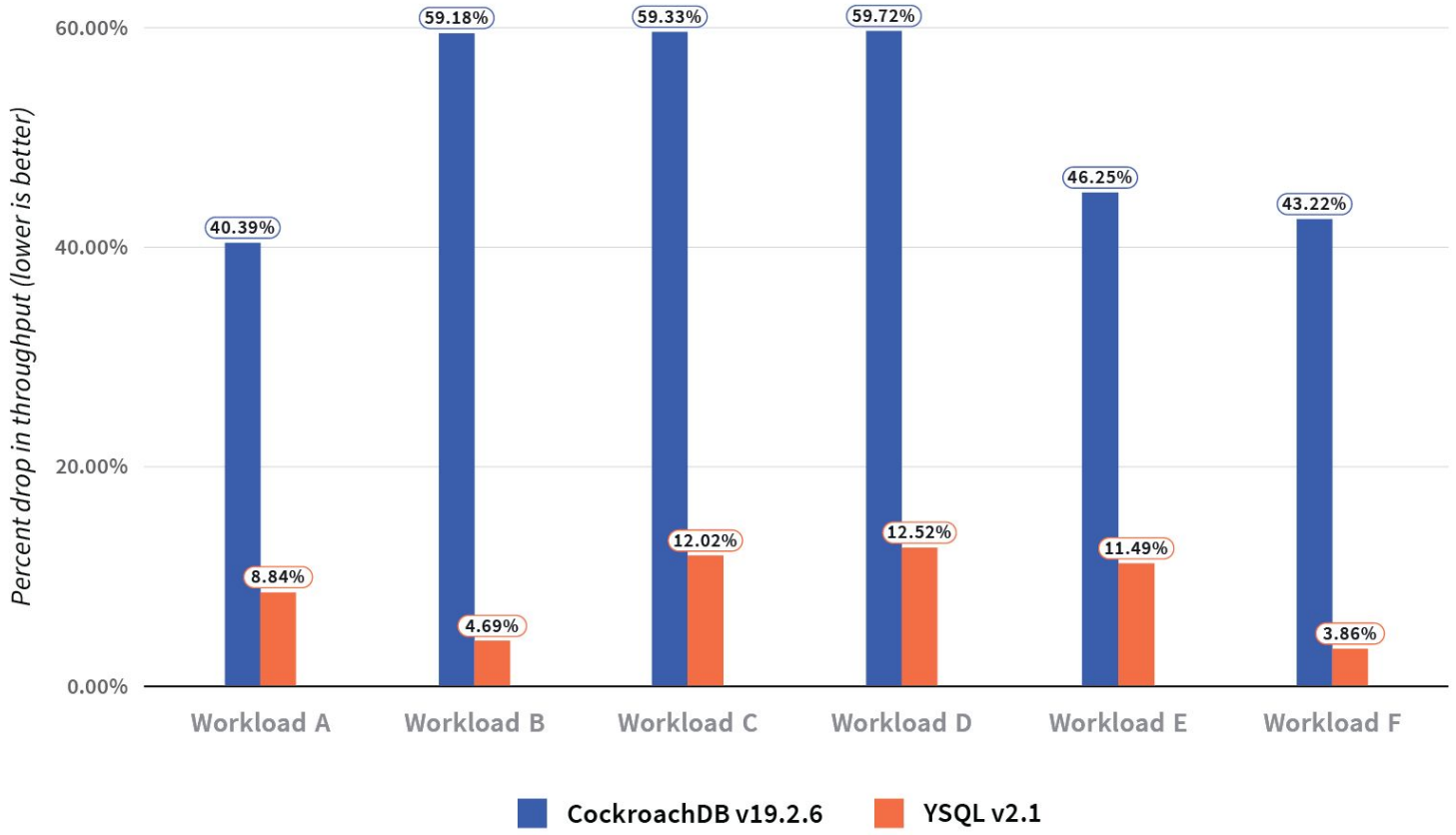
Cockroach DB

- Newly written SQL layer
- Write throughput drops over time
- Lower performance
  - RocksDB blackbox
  - Go/C++ (inter-language switch)

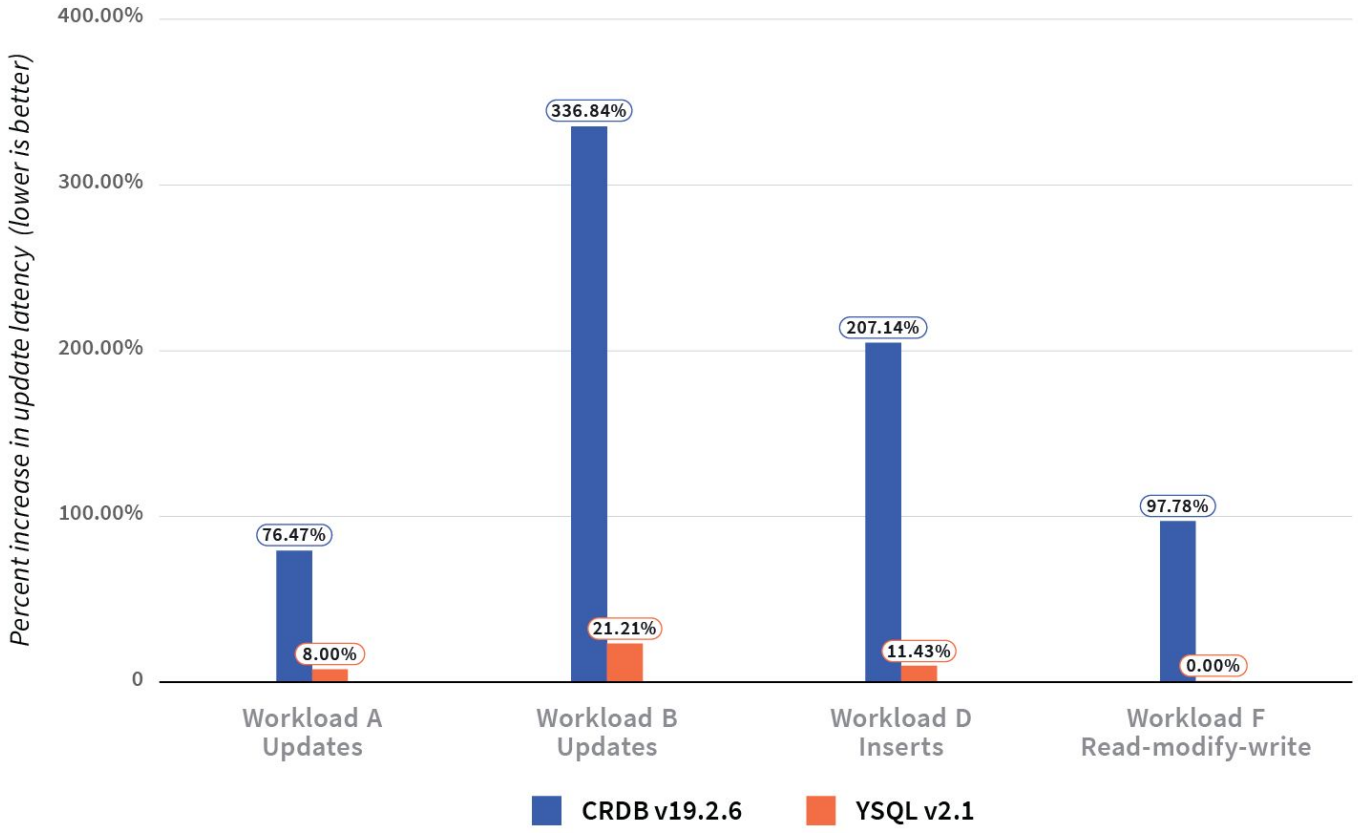
Performance with larger datasets

How does perf of each DB get affected when going from a small dataset to large dataset (450M rows)?

YCSB benchmark — Drop in throughput going from 1M to 450M rows



YCSB benchmark — Increase in update latency going from 1M to 450M rows



yugabyteDB

- Reuses PostgreSQL codebase
- Sustains high write throughput
- Higher throughput, lower latency
  - RocksDB enhanced
  - C/C++ for higher perf
- Large datasets: 9% lower throughput, 11% more latency

Cockroach DB

- Newly written SQL layer
- Write throughput drops over time
- Lower performance
  - RocksDB blackbox
  - Go/C++ (inter-language switch)
- Large datasets: 51% lower throughput, 180% more latency

Architectural takeaways
Observations loading 1B rows

Loading 1B rows through YCSB

3TB total dataset (1TB per node)

- YugabyteDB completed loading successfully in 26 hrs
- CockroachDB failed to load in reasonable time
- Throughput kept dropping over time

Next step - investigate why CockroachDB was not able to load 1B rows

Issue #1: CRDB unevenly uses multiple disks

```
Node1:
/dev/nvme1n1    5.0T  454M  5.0T   1% /mnt/d0   <-- barely used
disk
/dev/nvme2n1    5.0T  455G  4.6T   9% /mnt/d1

Node 2:
/dev/nvme1n1    5.0T  455G  4.6T   9% /mnt/d0
/dev/nvme2n1    5.0T  234M  5.0T   1% /mnt/d1   <-- barely used
disk

Node 3
/dev/nvme1n1    5.0T  455G  4.6T   9% /mnt/d0
/dev/nvme2n1    5.0T  174M  5.0T   1% /mnt/d1   <-- barely used
disk
```

Two disks supplied, only one disk was utilized.
Because CRDB shards (ranges) reuse same RocksDB

YugabyteDB can leverage multiple disks

```
Node1:
/dev/nvme1n1    4.9T  273G  4.7T   6% /mnt/d0
/dev/nvme2n1    4.9T  223G  4.7T   5% /mnt/d1

Node 2:
/dev/nvme1n1    4.9T  228G  4.7T   5% /mnt/d0
/dev/nvme2n1    4.9T  268G  4.7T   6% /mnt/d1

Node 3
/dev/nvme1n1    4.9T  239G  4.7T   5% /mnt/d0
/dev/nvme2n1    4.9T  258G  4.7T   6% /mnt/d1
```

Two disks supplied, both are utilized.
Each YugabyteDB shard (tablet) uses separate RocksDB

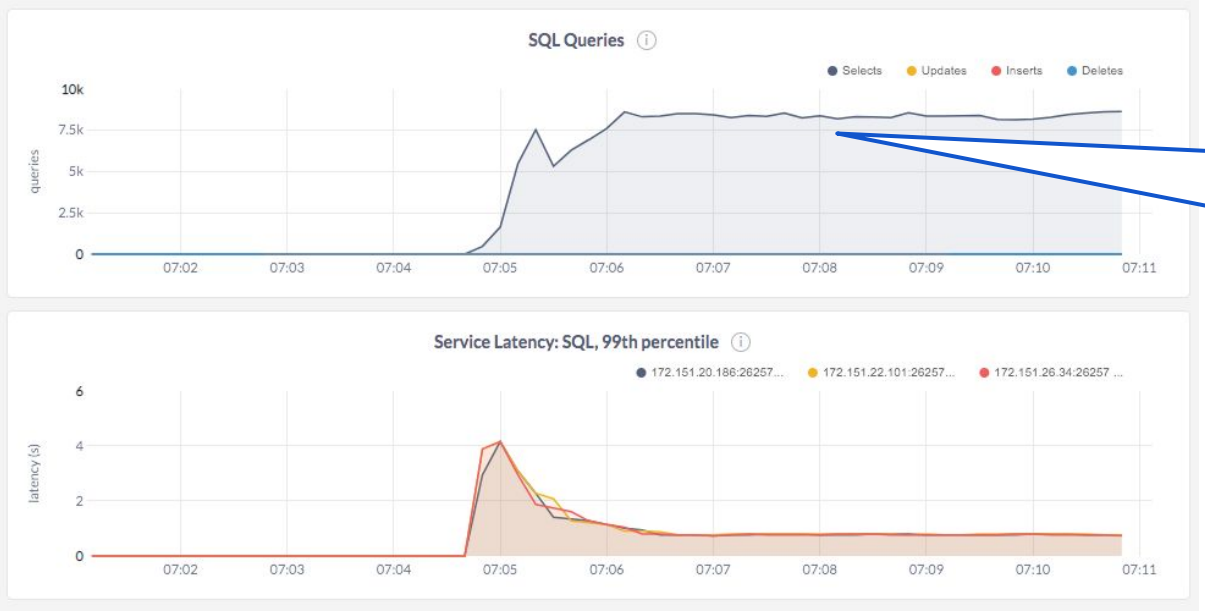
yugabyteDB

- Reuses PostgreSQL codebase
- Sustains high write throughput
- Higher throughput, lower latency
  - RocksDB enhanced
  - C/C++ for higher perf
- Large datasets: 9% lower throughput, 11% more latency
- Leverage multiple disks

Cockroach DB

- Newly written SQL layer
- Write throughput drops over time
- Lower performance
  - RocksDB blackbox
  - Go/C++ (inter-language switch)
- Large datasets: 51% lower throughput, 180% more latency
- Leverage 1 disk (maybe per table?)

Issue #2: Compactions affect CRDB perf



Observed: 8.5K reads/sec
6 hours later: 40K reads/sec

Right after loading data, query performance was poor.

Read amplification increases with SSTables

More read amplification = DB needs to consult more files.

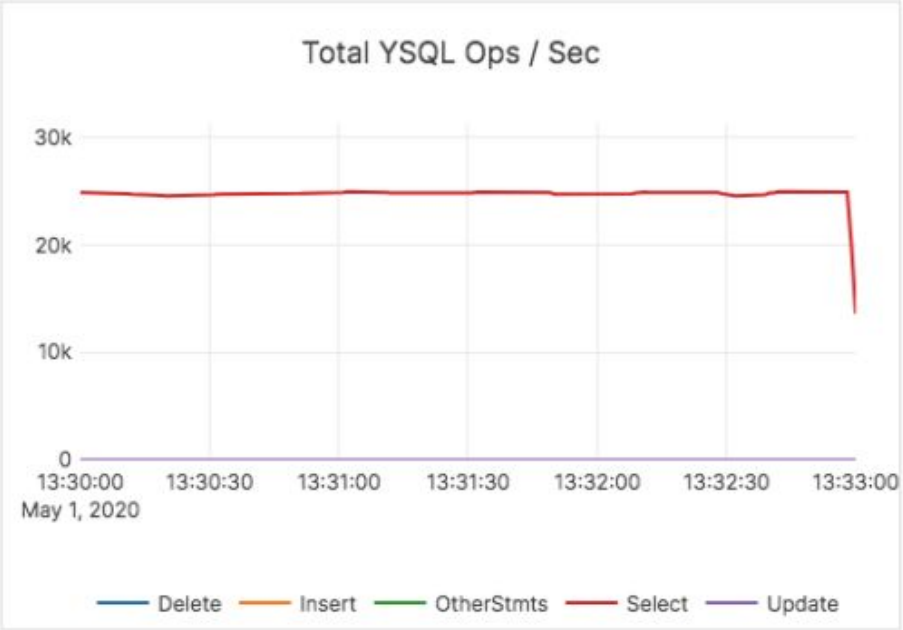
Solution: wait for many hours

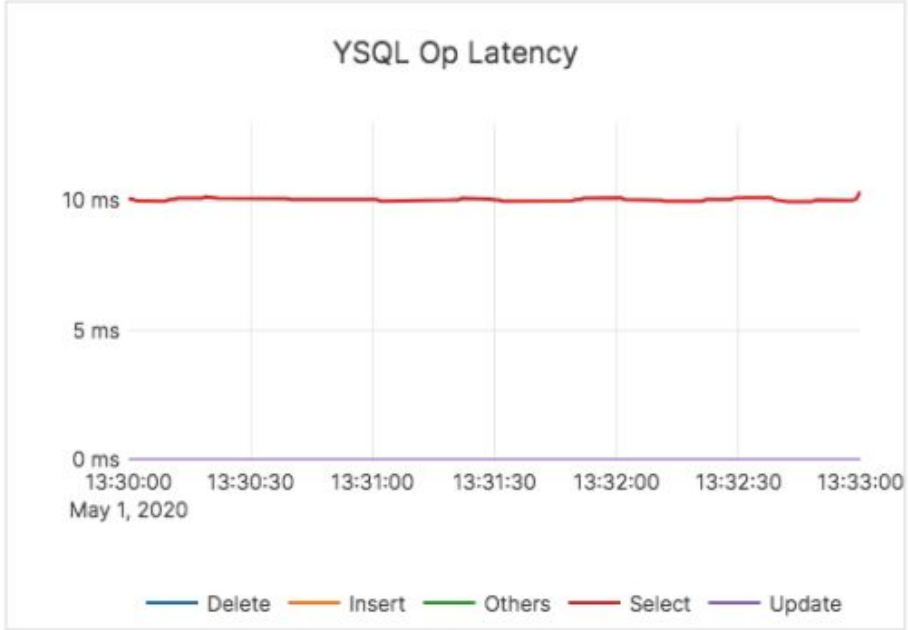
1. Compactions ran from 06:00 to 12:00

2. Decrease in number of SSTable files

3. Read amplification drops, improvement in performance

YugabyteDB perf high right after compaction





Performance right after loading 1B rows

yugabyteDB

- Reuses PostgreSQL codebase
- Sustains high write throughput
- Higher throughput, lower latency
  - RocksDB enhanced
  - C/C++ for higher perf
- Large datasets: 9% lower throughput, 11% more latency
- Leverage multiple disks
- Compactions tuned for perf

Cockroach DB

- Newly written SQL layer
- Write throughput drops over time
- Lower performance
  - RocksDB blackbox
  - Go/C++ (inter-language switch)
- Large datasets: 51% lower throughput, 180% more latency
- Leverage 1 disk (maybe per table?)
- Compactions policy impacts perf

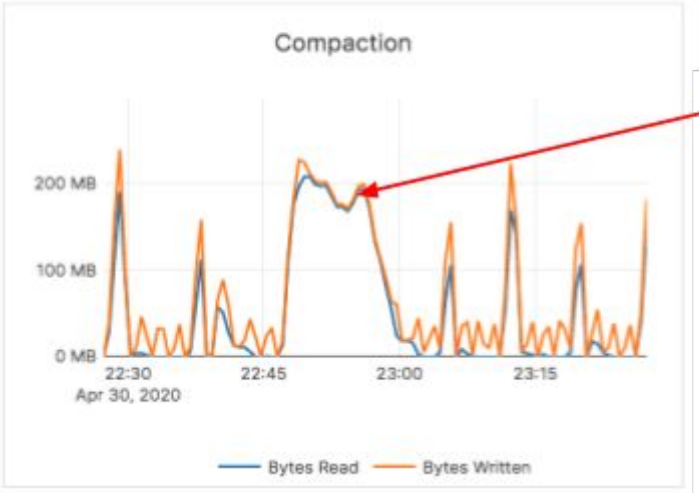
Issue #3: Backpressure writes

Loading data in a node with 2 x 5TB EBS gp2 SSDs

- Disk could become bottleneck
- DB can no longer keep up with writes
- Backpressure client to rate limit
- Happens often in the real world

CockroachDB goes into a mode where reads are severely penalized

To ensure good read performance, YugabyteDB applies backpressure on writes



This allows compactions to stay caught up, so read performance is always good

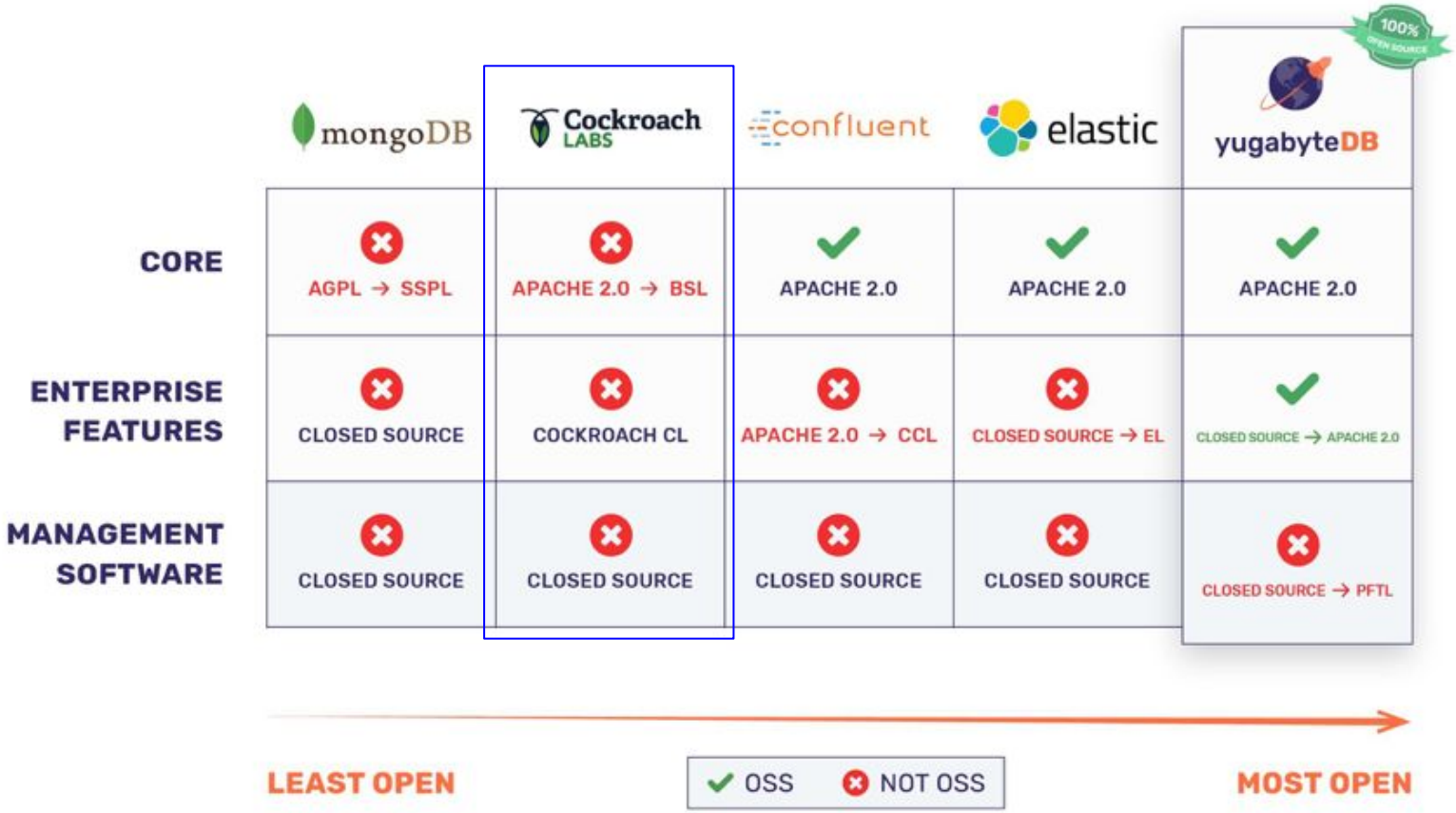
yugabyteDB

- Reuses PostgreSQL codebase
- Sustains high write throughput
- Higher performance
  - RocksDB enhanced
  - C/C++ for higher perf
- Large datasets: 9% lower throughput, 11% more latency
- Leverage multiple disks
- Compactions tuned for perf
- Backpressure when overloaded

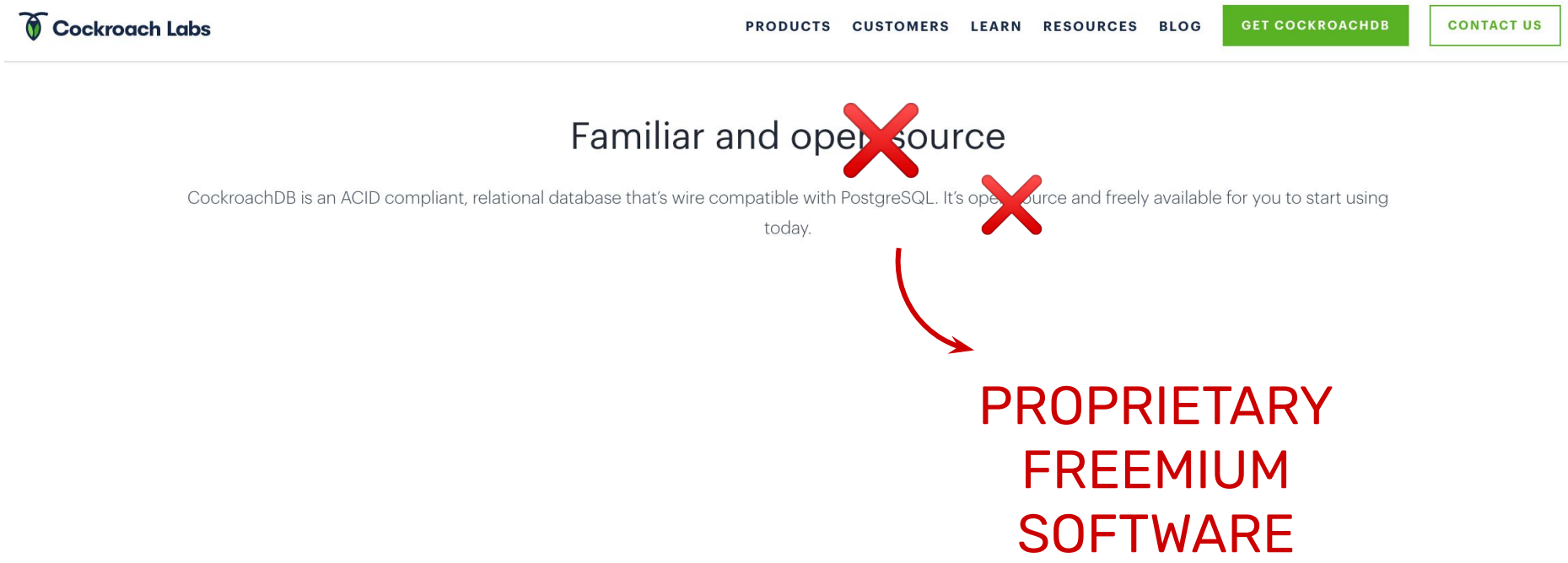
Cockroach DB

- Newly written SQL layer
- Write throughput drops over time
- Lower performance
  - RocksDB blackbox
  - Go/C++ (inter-language switch)
- Large datasets: 51% lower throughput, 180% more latency
- Leverage 1 disk (maybe per table?)
- Compactions could affect perf
- No backpressure

Licensing Model



Don’t fall for fake open source marketing



Thank You!

We 💛 stars! Give us one

github.com/yugabyte/yugabyte-db

Join our community

yugabyte.com/slack

Thanks!