



NoSQL and NewSQL: A Comparison of Distributed Database Systems



CONTENTS

NOSQL AND NEWSQL: TWO TAKES ON DISTRIBUTED DATABASE ARCHITECTURE	3
COCKROACHDB AND SCYLLA: NOSQL AND NEWSQL	4
INTRODUCTION TO SCYLLA	4
PERFORMANCE	5
CONSISTENCY AND AVAILABILITY	5
CASSANDRA QUERY LANGUAGE AND ACID GUARANTEES	6
DATA DISTRIBUTION	6
DATA MODELS	6
NOSQL VERSUS NEWSQL: MEASURING PERFORMANCE	7
THE INITIAL DATA LOAD: 1 BILLION KEYS VS 100 MILLION KEYS	7
YCSB: RESULTS FROM WORKLOAD A	8
RESULTS FROM YCSB WORKLOADS A THROUGH F	9
SCYLLA	9
COCKROACHDB	10
CONCLUSION	11

NOSQL AND NEWSQL: TWO TAKES ON DISTRIBUTED DATABASE ARCHITECTURE

Many IT organizations have become familiar with database tradeoffs, often wrestling with the fundamental decision between relational and non-relational databases. Organizations that needed a highly available distributed database chose non-relational NoSQL, while those that needed strong consistency and transactions chose relational SQL.

Non-relational databases introduced a spate of novel query languages, the most prominent of which is the Cassandra Query Language (CQL). Common CQL-compliant databases include Apache Cassandra, Scylla, DataStax Enterprise, and even Microsoft's cloud-native Azure Cosmos DB and Amazon Keyspaces.

As cloud computing became the norm, non-relational databases proliferated. But developers began to wonder if it might be possible to re-introduce features of relational databases that had been sacrificed by the generation of non-relational, NoSQL databases. Could a distributed, highly available database also provide strong consistency? Perhaps it was possible to have one's cake and to eat it too.

A team at Google took up the challenge and established a new generation of distributed databases that came to be known as "NewSQL." The first NewSQL database, Spanner, is a "scalable, multi-version, globally distributed, and synchronously replicated database." Spanner was designed specifically to address limitations in Google's Bigtable database. According to [the white paper](#), Spanner was conceived in part for applications that require "strong consistency in the presence of wide-area replication." How does Spanner accomplish this? The paper states that "the linchpin of Spanner's feature set is [TrueTime](#)." Spanner is able to consistently order transactions across distributed servers because servers in Google datacenters around the world all share precisely the same high-precision clocks (HPC). Spanner leverages TrueTime to execute transactions more efficiently.

Concepts from the Spanner paper were also incorporated in another distributed database, CockroachDB. CockroachDB is an open-source alternative to Google's implementation of Spanner. In contrast to Google Spanner, CockroachDB runs on commodity hardware, which lacks the high-precision clocks that help Spanner to scale consistent transactions. Instead, CockroachDB uses different, though similar, [algorithms and techniques](#) to provide distributed transactions like those of Spanner.

The model traditionally used for comparing tradeoffs in distributed systems is known as the "CAP theorem." While not developed specifically for describing databases, the CAP theorem does provide a useful model for articulating the tradeoffs between NoSQL and NewSQL databases. According to the CAP theorem a distributed system can provide only two of the following three attributes: Consistency, Availability, and Partition tolerance.

NoSQL databases are generally considered to be AP systems, providing Availability and Partition tolerance at the expense of Consistency. In contrast, NewSQL databases provide Consistency, Availability and Partition tolerance. (According to Eric Brewer's [painstaking analysis](#), Google Spanner is technically a CP system that can claim to be an "effectively CA" system. Such nuances, while important, are beyond the scope of this paper.)

The fusion of strong consistency with distributed architecture is undeniably attractive. The question is whether NewSQL can deliver on this promise without compromising in other critical areas - primarily performance. Notably, the traditional CAP theorem makes no provision for performance or latency. For example, according to the CAP theorem, a database can be considered Available if a query returns a response after 30 days. Obviously, such latency would be unacceptable for any real-world application.

A newer way to model databases, called the [PACELC theorem](#), extends beyond the CAP theorem model, showing that systems can either tend towards latency sensitivity or strong

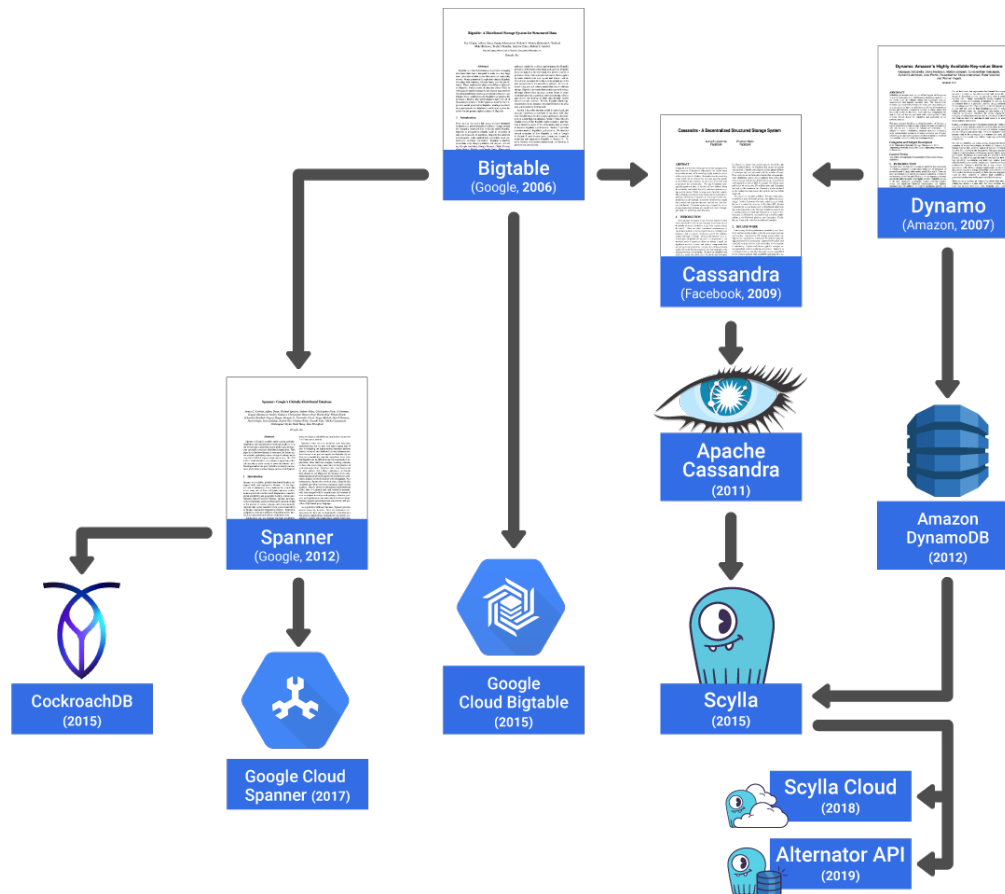


Figure 1: this “family tree” of modern distributed databases show how both NewSQL systems like CockroachDB (to the left) and NoSQL systems like Scylla (to the right) stemmed from scalability challenges addressed in the original Google whitepaper for Bigtable (center).

consistency. In this way, a NewSQL system such as CockroachDB is defined as PC/EC — focusing on being strongly consistent — whereas a NoSQL system like Scylla will be defined as PA/EL — highly available and latency sensitive.

The technical team at ScyllaDB decided to evaluate the performance characteristics of NoSQL versus NewSQL to highlight these differences. In this paper, we compare the performance characteristics of Scylla, a highly available, distributed, best-of-breed NoSQL database, against the best-in-class NewSQL database, CockroachDB. Such a comparison is admittedly an “apples to oranges” relationship, as the two databases utilize radically different architectures.

COCKROACHDB AND SCYLLA: NOSQL AND NEWSQL

INTRODUCTION TO SCYLLA

Like CockroachDB, Scylla has been under development since 2014. Scylla is a wide-column NoSQL database that uses a tunable eventual consistency model to provide fast and efficient reads and writes along with the multi-datacenter high availability.

In Scylla all nodes are equal; there is no single point of coordination and each node can serve any request. All nodes work together to provide service, even if one or more of the nodes become unavailable due to some failure. This enables Scylla to scale linearly to many nodes without performance degradation.

Built on the same underlying principles as Cassandra, Scylla is fully compatible with CQL and provides all of the functionality of Cassandra. Scylla also provides a DynamoDB-compatible API known as [Project Alternator](#).

PERFORMANCE

As a high-performance, next-generation NoSQL database, Scylla focuses on extracting every drop of available CPU and networking power to maximize performance. The computing model that underpins Scylla is completely asynchronous, running a native task-switching mechanism developed to run a million lambda functions (“continuations”) per second, per core.

In a high-performance database, control is more important than efficiency. In Scylla, every computation and I/O operation is a member of a priority class. CPU and I/O schedulers control the execution of continuations. Latency-sensitive classes, such as read and write operations, are dynamically afforded a higher priority than background maintenance tasks, such as compaction, repair, and streaming. To provide such tight control, Scylla optimizes I/O operations to be executed in the most efficient way possible on available hardware, pinning threads to CPU cores, threads to shards, and limiting filesystem/disk load to I/O that matches the hardware’s capacity.

CockroachDB’s primary focus is to provide consistency and availability, along with flexibility and support for SQL. It provides high availability the same way Scylla does — with redundancy — but it maintains consistency across replicas while serving concurrent operations. Maintaining consistency inevitably imposes additional overhead. To mitigate the overhead of consistency, CockroachDB uses a number of innovative approaches designed to provide high-performance quality of service. For example, CockroachDB leverages modern consensus algorithms, such as [Raft](#). Other innovations include a hybrid transactions scheduler that makes reads linearization cost-free. CockroachDB is written in Go and is, like applications written in Java, susceptible to garbage collection spikes.

CONSISTENCY AND AVAILABILITY

As noted, a defining feature of NewSQL databases is support for consistent transactions within distributed topologies. The problem is that CockroachDB currently supports only one transaction execution mode: *serializable isolation*. This is a powerful feature when an application requires strong isolation guarantees that are free of [anomalies](#). It also precludes a weaker isolation model where higher performance is preferred over strong transactions.

All reads and writes in CockroachDB execute in the context of transactions. A transaction is an interactive session in which a client sends requests and then finalizes them with a commit or abort keyword. To serialize transactions, CockroachDB offers a parallel two-phase commit variant along with a novel hybrid serialization scheduler. In the best case, CockroachDB is capable of committing a transaction in one round-trip time (RTT). In general, though, CockroachDB requires a serialization check on every read and write, as well as waiting until all writes have been replicated. CockroachDB uses indirection in the read path to atomically switch data visibility.

Overall, according to [Jepsen’s analysis](#), CockroachDB provides [near strong-1SR](#) consistency.

Like most non-relational databases, Scylla implements a model known as “eventual consistency.” Eventual consistency supports the rapidly growing number of modern workloads that depend heavily on availability and are less dependent on guarantees of strong consistency. For example, during partitioning caused by an outage, it is often preferable for an isolated data center to continue to accept reads and writes. While Scylla does lean strongly towards availability over consistency, it also offers an API for stronger consistency that leverages lightweight transactions (LWT).

Unlike CockroachDB, Scylla enables consistency to be tuned per transaction. For example, consistency level one ensures that queries succeed even if one node acknowledges a read or write. A stronger consistency level ensures

that a majority of replica nodes acknowledge a read or write. Where every replica node must acknowledge that the transaction successfully completed, Scylla supports a consistency level that encompasses all nodes. In other words, Scylla supports consistency, albeit only within single partition.

As we've noted, CockroachDB favors consistency over availability. Yet, it leverages a variety of innovative approaches to enhance availability. Nevertheless, CockroachDB's usage of the Raft consensus protocol dictates that a quorum of nodes within a ReplicaSet must be alive and accessible. While it enables consistency to span partitions, this requirement also places a limit on availability and performance per partition within a CockroachDB cluster.

CASSANDRA QUERY LANGUAGE AND ACID GUARANTEES

The Cassandra Query Language (CQL), employed by Scylla, can be deceptively similar to SQL. Consider the following CQL query:

```
SELECT * FROM Table;  
UPDATE Table (a, b, c) VALUES (1, 2, 3)  
WHERE Id = 0;
```

When using SQL, developers typically expect ACID guarantees. However, Scylla does not provide full ACID semantics for its operations. By definition, ACID guarantees provided in the context of a single transaction are:

- **Atomicity:** Transactions with multiple statements are treated as a single unit.
- **Consistency:** Transactions can move database state only from one valid state to another valid state.
- **Isolation:** Concurrent transactions are executed as if they had been submitted sequentially, and all individual transactions have a “before and after” relationship.
- **Durability:** The results of transaction execution are preserved, even in the event of system failures.

What Scylla does not provide, from an ACID perspective, is isolation. Isolation is typically required only by applications that perform multi-statement, cross-partition transactions. For a detailed analysis of this topic, you can read the [Jepsen analysis](#) of Scylla (See section 3.4, *Normal Writes Are Not Isolated*) and our accompanying [blog post](#).

DATA DISTRIBUTION

An even distribution of data across nodes in a cluster ensures that load can be evenly distributed and processed by all nodes. In Scylla, data is distributed as uniformly as possible using a hash function (partitioner). Token ring and cluster topology configuration is shared with client applications, enabling an efficient choice of the closest nodes, and even reaching the specific CPU core that handles the partition within the node. This capability minimizes network hops and maximizes load balancing.

CockroachDB implements a two-level indexing structure, analogous with Spanner, and also used by Bigtable and HBase, that they refer to as “order-preserving data distribution.” While this design creates complexity on the CockroachDB internals, it enables a full implementation of SQL. Transactions are used to insert and delete data into range. Each range in CockroachDB is a Raft group. The unit of replication is a range.

DATA MODELS

Scylla offers a natural RDBMS-like model where data is organized in tables that consist of rows and columns on top of the wide-column storage. A row key consists of a partition key and an optional clustering key. A clustering key defines rows ordering inside of the partition. A partition key determines partition placement.

Users define tables schemas, insert data into rows, and then read the data. There are usual concepts such as Secondary Indexes, Materialized Views, Complex Data Types, Lightweight Transactions, and other features built on top.

The wide-column data model differs from the classical RDBMS-style model in that rows are not first-class citizens but the cells are. (Rows consist of cells.)

CockroachDB offers a classic relational data model with tables and rows built on top of LSM-based key-value storage. CockroachDB is wire-compatible with PostgreSQL.

NOSQL VERSUS NEWSQL: MEASURING PERFORMANCE

To summarize, Scylla and CockroachDB have been designed and engineered to address different problems and use cases. Following different design principles, the two databases naturally take different approaches to serving a variety of workloads. With these divergent purposes in mind, it is still worthwhile to evaluate their relative performance characteristics. Again, we are comparing “apples to oranges.” Our goal is simply to highlight the performance impact on the broader tradeoff between availability and consistency.

To measure database performance under various workloads, we executed a series of benchmarks defined in the [Yahoo! Cloud Serving Benchmark](#) (YCSB) test suite. The YCSB is a widely used, industry-standard [benchmarking framework](#) that measures scalability and performance (defined as latency). The YCSB tests encompass a variety of workloads ([A through F](#)), each of which represents a different access pattern. The six benchmarks represent the following workloads:

- **Workload A:** Update Heavy, 50/50 read/write ratio
- **Workload B:** Read Mostly, 95/5 read/write ratio
- **Workload C:** Read Only, 100/0 read/write ratio
- **Workload D:** Read Latest, 95/0/5 read/update/insert ratio
- **Workload E:** Short Range, 95/5 scan/insert ratio
- **Workload F:** Read-Modify-Write, 50/50 read/read-modify-write ratio

To execute these benchmarks, we provisioned Scylla and CockroachDB on AWS public cloud infrastructure. For both databases, we created similar clusters, each consisting of 3 nodes running on storage optimized [i3.4xlarge AWS EC2](#) instances within a single geographic region (eu-north-1), evenly spread across three availability zones with the standard replication factor (RF=3) and default configuration.

To measure CockroachDB 20.1.6 performance, we used the [brianfrankcooper/YCSB 0.17.0](#) benchmark with [PostgreNoSQL](#) binding and [CockroachDB v20.1.6 YCSB port](#) to the Go programming language.

For Scylla 4.2.0, we used the [brianfrankcooper/YCSB 0.18.0 SNAPSHOT](#) with a Scylla-native binding and a Token Aware load-balancing policy.

THE INITIAL DATA LOAD: 1 BILLION KEYS VS 100 MILLION KEYS

Initially, we aimed to measure load time and throughput under the stress, populating both databases with a dataset of 1 billion keys.

The official CockroachDB [documentation](#) states that the storage capacity limit is set at 150GB per vCPU and up to 2.5TB per node total. Even so, we were unable to successfully load 1 billion keys into the CockroachDB cluster. The cluster became unresponsive after 3-5 hours of loading, generating critical errors in the logs. Similar behavior was observed during a subsequent 30-minute long sustained workload test.

Our observation shows the load throughput for the CockroachDB cluster degraded from 12K transactions per second (OPS) down to 2.5K OPS within 3-5 hours. Loading 1 billion keys at a rate of 2.5K keys per second was projected to take about 111 hours or 4.5 days. Notably, similar issues were observed by YugaByte: [[1 billion trial](#)], [[slowdown](#)], and [[results](#)].

These issues led us to reduce the dataset size for CockroachDB to 100 million keys. With the smaller dataset, loading took 7 hours and resulted in 1.1TB of data, which was later compacted to 450GB. The latency graph over this 7 hours period can be seen in Figure 2.

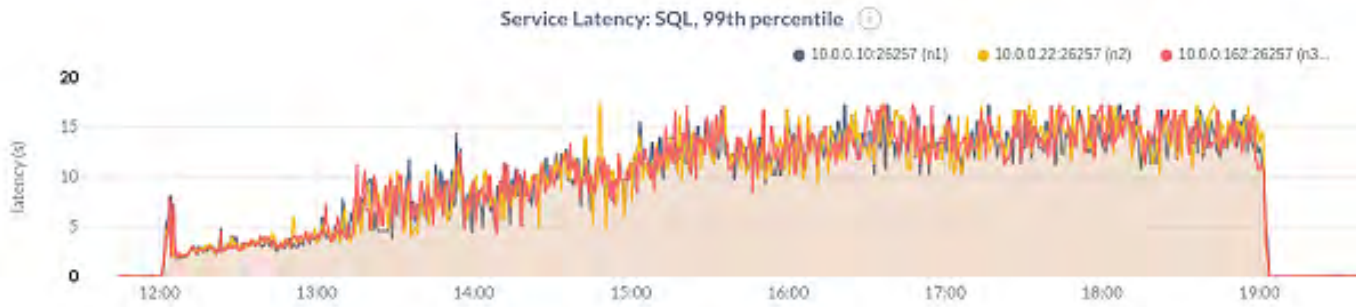


Figure 2: P99 latency for loading 100 million keys into CockroachDB.

In contrast, Scylla easily loaded 1 billion keys, which represented about 4.8TB of data, in about three hours, exhibiting the same performance characteristics as with the smaller dataset. It took just 20 minutes for Scylla to load 100 million keys, which translated into just 300GB of data.

OF NOTE: It took Scylla 20 minutes to load 100 million keys, compared to 7 hours for CockroachDB. That's **20x** more efficient at loading a comparable dataset. In fact, loading 1 billion keys took Scylla less than half the time it took CockroachDB to load 100 million keys.

YCSB: RESULTS FROM WORKLOAD A

With the initial data load complete, we were able to evaluate results produced by YCSB Workload A, which represents a 50/50 mix of read and write workloads. This benchmark tests database performance when clients actively write and read at the same time.

Under this workload, Scylla achieved 120K OPS with P99 latency under 4.6ms, while CPU utilization remained under 60%, and 150K OPS with P99 under 12ms for the 1 billion key dataset size.

The dashboard in Figure 3, taken from Scylla Monitoring Stack, displays the performance of the Scylla cluster for 1 billion keys serving 120,000 OPS at 600 µsec P99 latency for writes and below 4ms of P99 latency for reads.

The dashboard in Figure 4 displays CockroachDB's results for Workload A, running against a dataset of 100 million keys. CockroachDB produced at best about 16K OPS with P99 latency at 52ms, generating an intermediate spike of 200ms at CPU utilization that varied between 50% and 75%.

OF NOTE: Compared with CockroachDB, Scylla handled **10x** the amount of data while providing **9.3x** the throughput at **1/4** the latency.



Figure 3: Performance of Scylla cluster for 1 billion keys under Workload A.

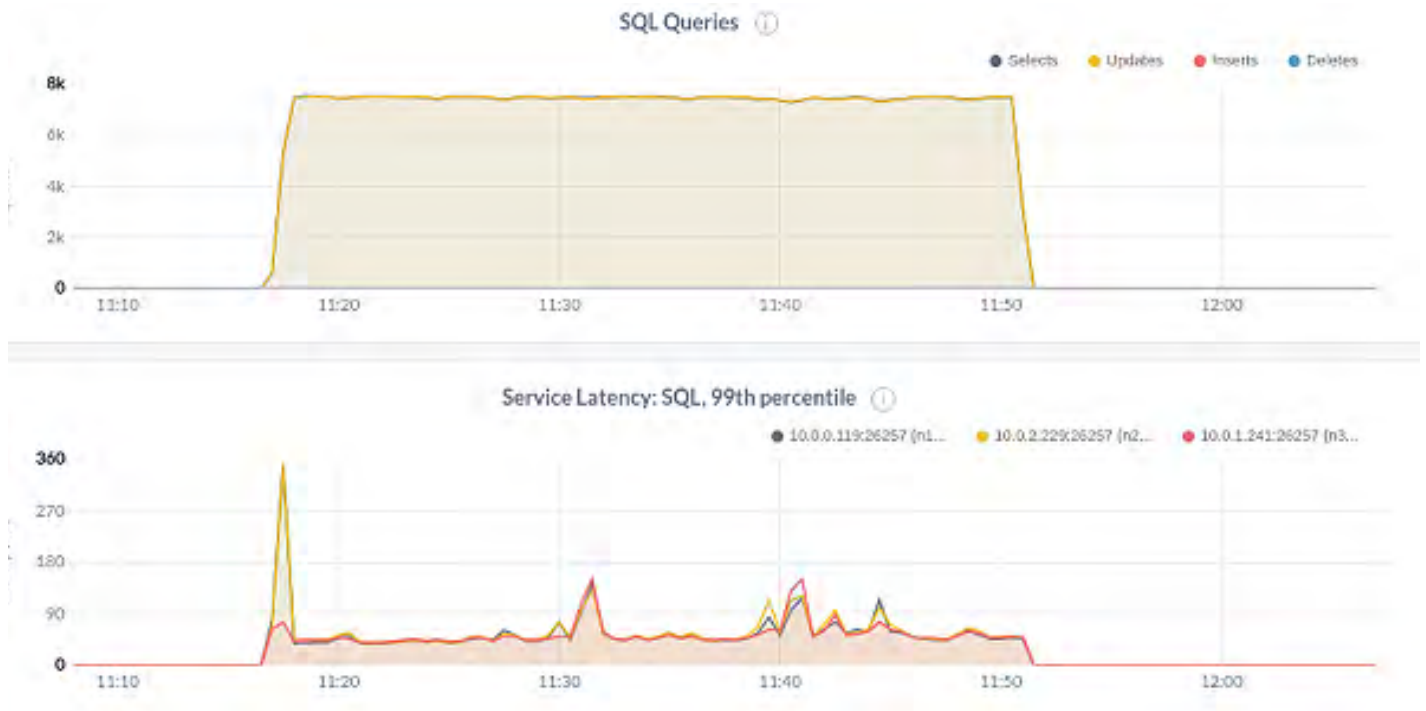


Figure 4: CockroachDB results for Workload A.

The graph in Figure 5 displays throughput and latency for a 100 million key dataset on CockroachDB versus 1 billion key dataset on Scylla. By emphasizing performance over consistency, Scylla can provide **10x** better throughput and **4x** better latency for a dataset that is **10x** larger in size.

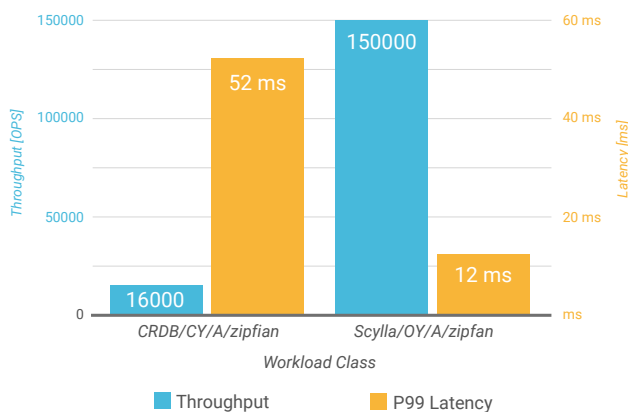


Figure 5: Workload A for CockroachDB (100M keys) vs Scylla (1B keys)

RESULTS FROM YCSB WORKLOADS A THROUGH F

Scylla

For the 1 billion key dataset, Scylla successfully managed to serve 150K-200K OPS on most of the workloads at 75-80% utilization with reasonable latency. On average, Scylla achieved 180K OPS with p99 latencies under 5.5ms at 75% load.

The charts in Figure 6 display the throughput and latency achieved by Scylla for each YCSB workload. For Workloads A-D, Scylla delivered predictable throughput between 150K and 180K OPS at 5.5ms to 12ms P99 latency.

Similarly, the Figure 7 chart demonstrates that Scylla delivers predictable performance at large scale, while approaching maximum system utilization.

For example for workload D with a dataset of 1 billion keys, Scylla demonstrated 180K OPS with p99 latency below 5.5ms and CPU utilization of only 75%. *It is worth noting that such performance and scale are rare for any open-source distributed database system.*

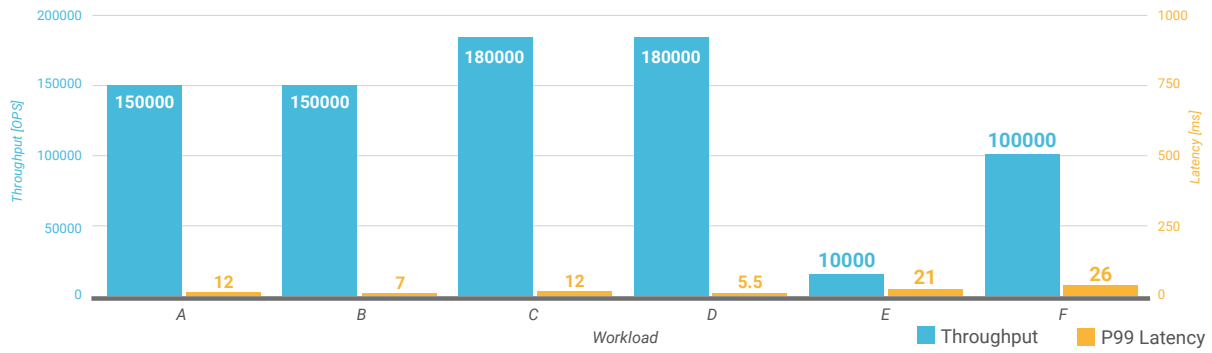


Figure 6: Scylla throughput and latency per YCSB workload.



Figure 7: ScyllaDB metrics

Workload E defines operations that are short-range scans, rather than queries against individual records. According to the YCSB benchmark, the workload involves “threaded conversations, where each scan is for the posts in a given thread assumed to be clustered by thread ID.” As expected, this workload produced the worst performance – only 10k operations per second.

YCSB Workload E is implemented in a way that does not play to the strengths of either Scylla or CockroachDB. In Scylla, range partition scans are token-based, and tokens are randomly distributed throughout the cluster. As such, many random reads across multiple nodes are required to satisfy a single scan request. Unsurprisingly, this is not [an efficient way to do range scans](#). Yet while range scans are not the

main advantage of Scylla, Scylla still did well enough and outperformed CockroachDB by **5x**.

CockroachDB

Even with the smaller dataset of 100 million keys, CockroachDB ran up against performance scalability limits much earlier than Scylla, as shown in Figure 8. For Workload A, CockroachDB recorded 16K OPS with p99 under 52ms. With Workload D, CockroachDB achieved its best result, demonstrating 40K OPS with p99 below 176ms, while running at 80% CPU utilization. Further increasing the load did not increase throughput. Instead, the higher load increased latency.

The Figure 9 graphs represents Workload E (range scans) results with 2k OPS and P99 latency of 537ms.

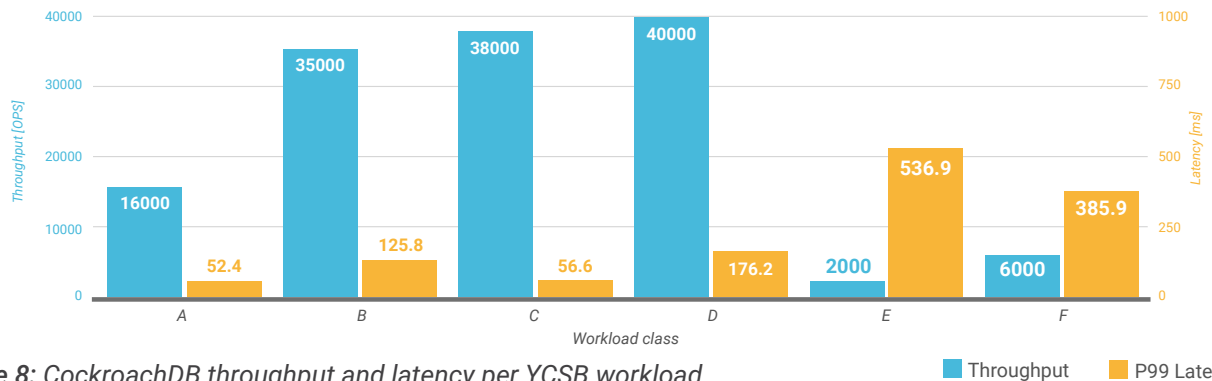


Figure 8: CockroachDB throughput and latency per YCSB workload.

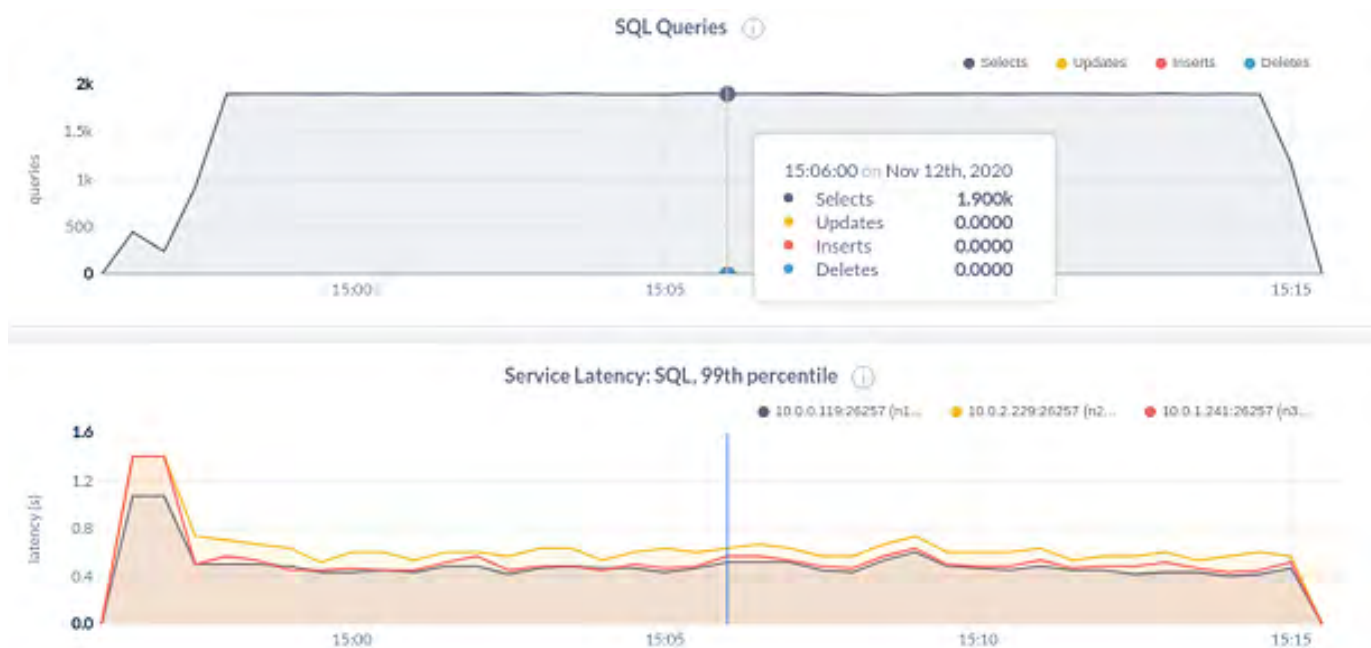


Figure 9: CockroachDB results for Workload E.

CONCLUSION

NoSQL and NewSQL models are clearly converging, each providing more functionality along with better performance and availability than traditional relational database offerings. It is not a surprise that a highly available NoSQL database such as Scylla outperforms a distributed NewSQL database such as CockroachDB by a wide margin. The results are not meant to indicate whether one should select Scylla, or even NoSQL, for every workload.

To summarize, with a dataset of 1 billion keys, Scylla demonstrated:

- Throughput **up to 10x** better than CockroachDB
- Predictable low latencies, even while handling **10x** the data

CockroachDB demonstrated throughput degradation during data loading. During the YCSB workloads, we measured throughput that closely matched the CockroachDB whitepaper, albeit with larger and more variable latencies.

Modern, cloud-native applications often require high availability and predictable low latency. Such workloads are ideal for Scylla. Requirements for strong consistency are less common. Workloads characterized by modest dataset size, and which require strong consistency guarantees and transactions along with a relational database model, involving JOINS, are ideal for a NewSQL database such as CockroachDB. While using SQL transactions can be a convenient solution to various business cases, they may end up incurring significant performance and maintenance costs as the system scales – although sometimes they are a necessary evil.

ABOUT SCYLLADB

Scylla is the real-time big data database. API-compatible with Apache Cassandra and Amazon DynamoDB, Scylla embraces a shared-nothing approach that increases throughput and storage capacity as much as 10X. Comcast, Discord, Disney+ Hotstar, Grab, Medium, Starbucks, Ola Cabs, Samsung, IBM, Investing.com and [many more leading companies](#) have adopted Scylla to realize order-of-magnitude performance improvements and reduce hardware costs. Scylla's database is available as an open source project, an enterprise edition and a fully managed database as a service. ScyllaDB was founded by the team responsible for the KVM hypervisor. For more information: [ScyllaDB.com](https://scylladb.com)

SCYLLADB.COM



SCYLLA

United States Headquarters
2445 Faber Place, Suite 200
Palo Alto, CA 94303 U.S.A.
Email: info@scylladb.com

Israel Headquarters
11 Galgalei Haplada
Herzeliya, Israel



Copyright © 2020 ScyllaDB Inc. All rights reserved. All trademarks or registered trademarks used herein are property of their respective owners.