

Benchmarking Cloud Native Database Running on Kubernetes

Iqbal Farabi and Tara Baskara
KubeCon + CloudNativeCon Europe 2019



Hola!



Iqbal Farabi
System Engineer
Go-Jek Indonesia
@iqbal_farabi



Tara Baskara
System Engineer
Go-Jek Indonesia
@iqbal_farabi





We're from Jakarta, Indonesia



KubeCon



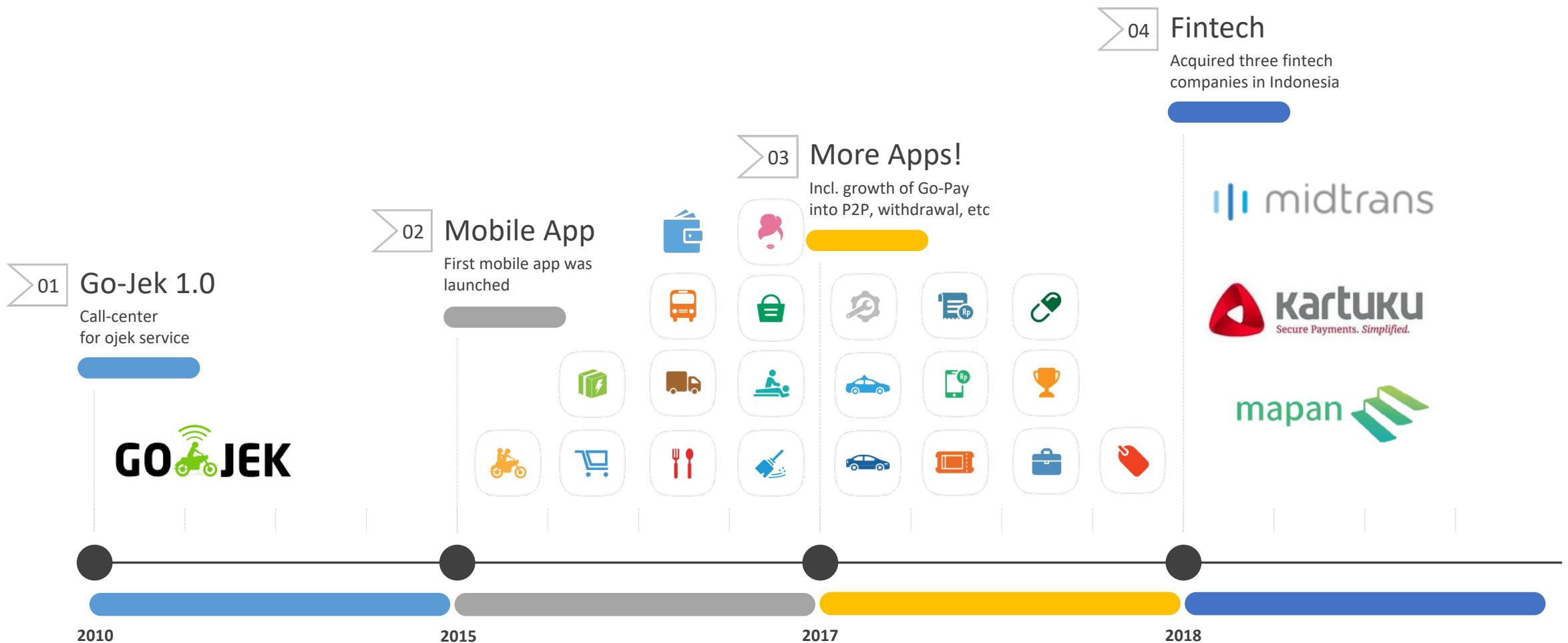
Cloud Native Con

China 2018





Brief History





JAKARTA



BANGALORE



SINGAPORE



THAILAND



VIETNAM

The Journey Continues...

GOJEK
INTERNATIONAL
EXPANSION

We officially announce our international expansion to four different markets: Vietnam, Thailand, Singapore and the Philippines!

#JUSTGOJEKIT



Outline



Today we will discuss about...

Cloud Native Database

- What is cloud native database? What databases do we pick to run the experiments on? What are their characteristics?

YCSB

- What is YCSB? What does it measure? What are the different kind of workloads defined?

Experiments and Results

- Setup, experiments, and brief explanation of the results.



Cloud Native Database











































Definition

CNCF [defines](#) cloud native technologies as, "technologies that empower organizations to build and run scalable applications in modern and dynamic environments. Cloud native technologies enable loosely coupled systems that are resilient, manageable, and observable".



Landscape

 <p>CarbonData APACHE ★ 898 Apache CarbonData Apache Software Foundation</p>	 <p>Ignite apache ★ 2,469 Apache Ignite Apache Software Foundation</p>	 <p>ArangoDB ★ 7,905 ArangoDB Funding: \$16.9M</p>	 <p>BIGCHAINDB ★ 3,163 BigchainDB Funding: \$5.37M</p>	 <p>cassandra ★ 5,110 Cassandra Apache Software Foundation</p>	 <p>Cockroach LABS ★ 16,036 CockroachDB Cockroach Labs Funding: \$53.5M</p>	 <p>Couchbase ★ 159 Couchbase Funding: \$150M</p>	 <p>CRATE.IO ★ 2,416 Crate.io Funding: \$17.9M</p>	 <p>Dgraph ★ 9,433 Dgraph Labs Funding: \$2.95M</p>
 <p>druid ★ 7,979 Druid Apache Software Foundation</p>	 <p>FoundationDB ★ 9,242 FoundationDB Apple MCap: \$965B</p>	 <p>hadoop ★ 8,927 Hadoop Apache Software Foundation</p>	 <p>hazelcast IMDG ★ 3,073 Hazelcast IMDG Funding: \$13.6M</p>	 <p>IBM DB2 ★ 5,110 IBM DB2 IBM MCap: \$124B</p>	 <p>iguazio ★ 16,036 iguazio Funding: \$48M</p>	 <p>Infinispan ★ 732 Infinispan Red Hat MCap: \$32.4B</p>	 <p>InterSystems IRIS Data Platform ★ 2,633 InterSystems IRIS Data Platform InterSystems Funding: \$98.2M</p>	 <p>MariaDB ★ 2,633 MariaDB MariaDB Corporation Funding: \$98.2M</p>
 <p>MEMSQL ★ 1,648 MemSQL Funding: \$108M</p>	 <p>Microsoft SQL Server ★ 15,921 Microsoft SQL Server Microsoft MCap: \$994B</p>	 <p>mongoDB ★ 15,921 MongoDB MongoDB MCap: \$7.56B</p>	 <p>MySQL ★ 3,724 MySQL Oracle MCap: \$188B</p>	 <p>neo4j ★ 6,372 Neo4j Neo4j Funding: \$160M</p>	 <p>noms ★ 6,849 NomsDB Salesforce MCap: \$128B</p>	 <p>ORACLE DATABASE ★ 544 Oracle Database Oracle MCap: \$188B</p>	 <p>OrientDB ★ 3,828 OrientDB SAP MCap: \$155B</p>	 <p>PERCONA ★ 554 Percona Server for MySQL Percona</p>
 <p>piloa ★ 1,648 Piloa Funding: \$3.67M</p>	 <p>PostgreSQL ★ 5,165 PostgreSQL PostgreSQL</p>	 <p>presto ★ 362 Presto Presto Software Foundation</p>	 <p>Qubole ★ 36,054 Qubole Funding: \$75M</p>	 <p>redis ★ 36,054 Redis Redis Labs Funding: \$147M</p>	 <p>RethinkDB ★ 22,072 RethinkDB Linux Foundation</p>	 <p>SCYLLA ★ 5,043 Scylla ScyllaDB Funding: \$35M</p>	 <p>snowflake ★ 5,043 Snowflake Snowflake Inc Funding: \$929M</p>	 <p>software AG ★ 1,063 Software AG MCap: \$2.7B</p>
 <p>STOLON ★ 2,068 Stolon Sorint.Lab</p>	 <p>TiDB ★ 18,411 TiDB PingCAP Funding: \$71.6M</p>	 <p>TiKV ★ 5,019 TiKV Cloud Native Computing Foundation (CNCF)</p>	 <p>VERTICA ★ 7,893 Vertica Vertica Systems Funding: \$30.5M</p>	 <p>Vitess ★ 7,893 Vitess Cloud Native Computing Foundation (CNCF)</p>	 <p>YugaByte ★ 1,063 YugaByte DB YugaByte Funding: \$24M</p>			



Filtering

- Open source
- Operational database
- ACID compliance
- Provides SQL-like API



The List

CockroachDB

- <https://www.cockroachlabs.com/>

TiDB

- <https://pingcap.com/en/>

YugaByte DB

- <https://www.yugabyte.com/>



CockroachDB

CockroachDB is a distributed SQL database built on a transactional and strongly-consistent key-value store.

It aims to:

- Scales horizontally
- Provides fault resiliency
- Supports strongly-consistent ACID transactions
- Provides a familiar SQL API with PostgreSQL-like syntax



CockroachDB Architecture

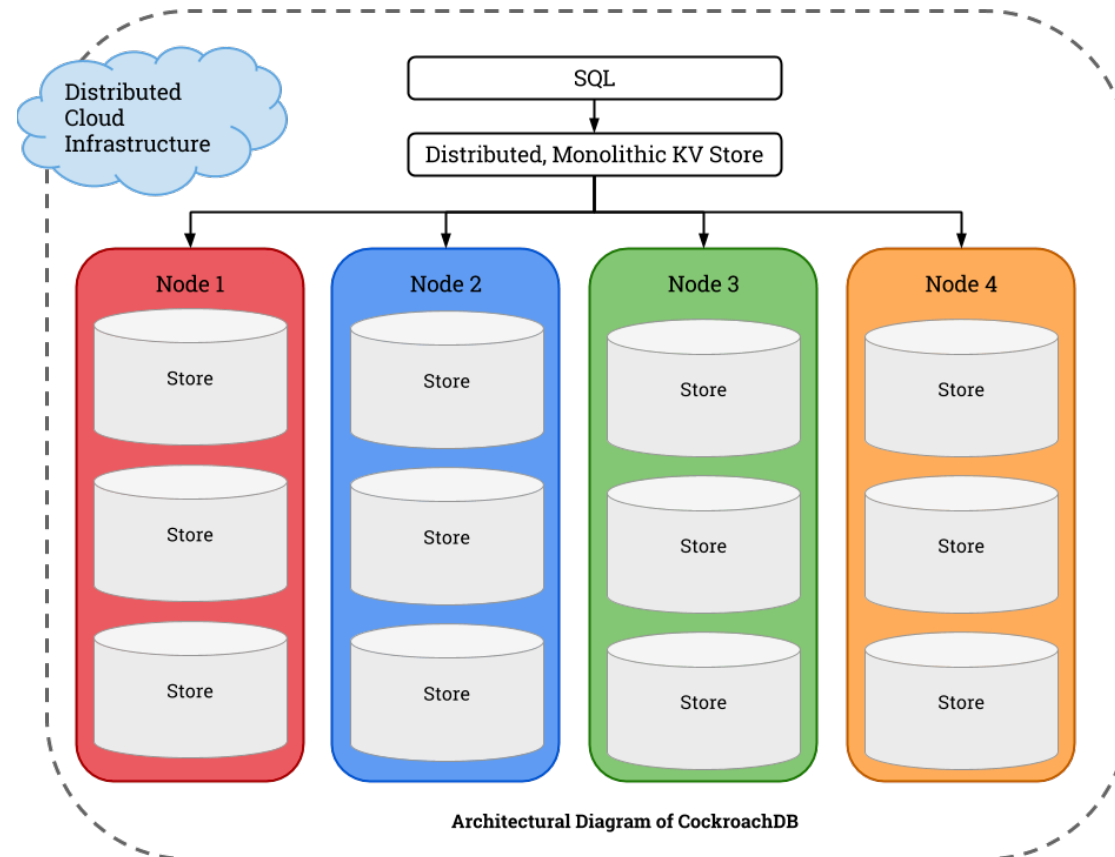


image source: <https://thenewstack.io/cockroachdb-unkillable-distributed-sql-database/>

TiDB

TiDB is an open-source NewSQL database that supports Hybrid Transactional and Analytical Processing (HTAP) workloads.

It aims to:

- Scales horizontally
- Provides fault resiliency
- Supports distributed transactions with strong consistency
- Provides a familiar SQL API with MySQL-like syntax



TiDB Architecture

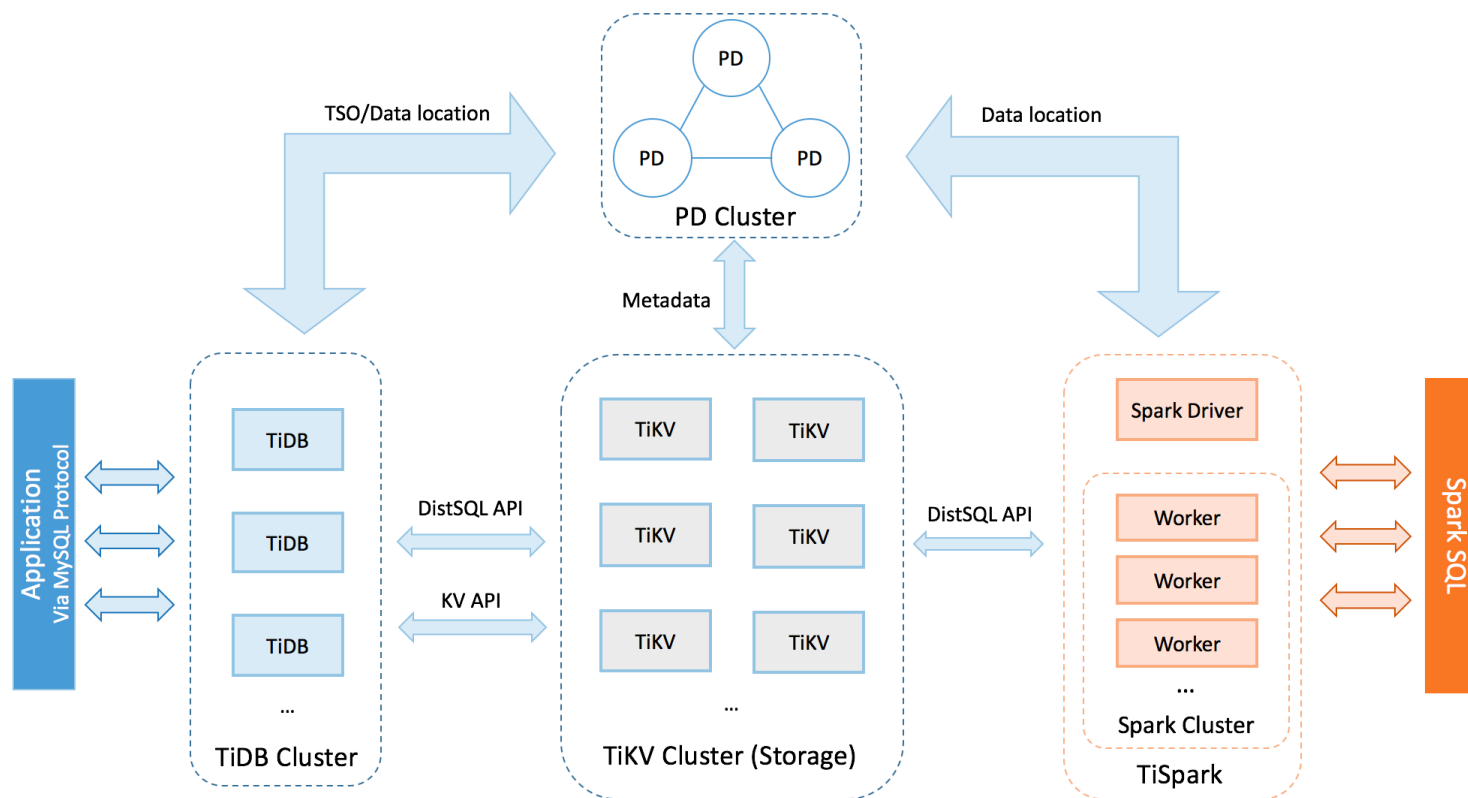


image source: <https://pingcap.com/docs/architecture/>

YugaByte DB

YugaByte DB is a distributed SQL database built on a transactional and strongly-consistent key-value store.

It aims to:

- Scales with autosharding
- Provides fault resiliency
- Supports multi-shard ACID transactions
- Provides YugaByte Structured Query Language (YSQL) and YugaByte Cloud Query Language (YCQL) APIs.



YugaByte DB Architecture

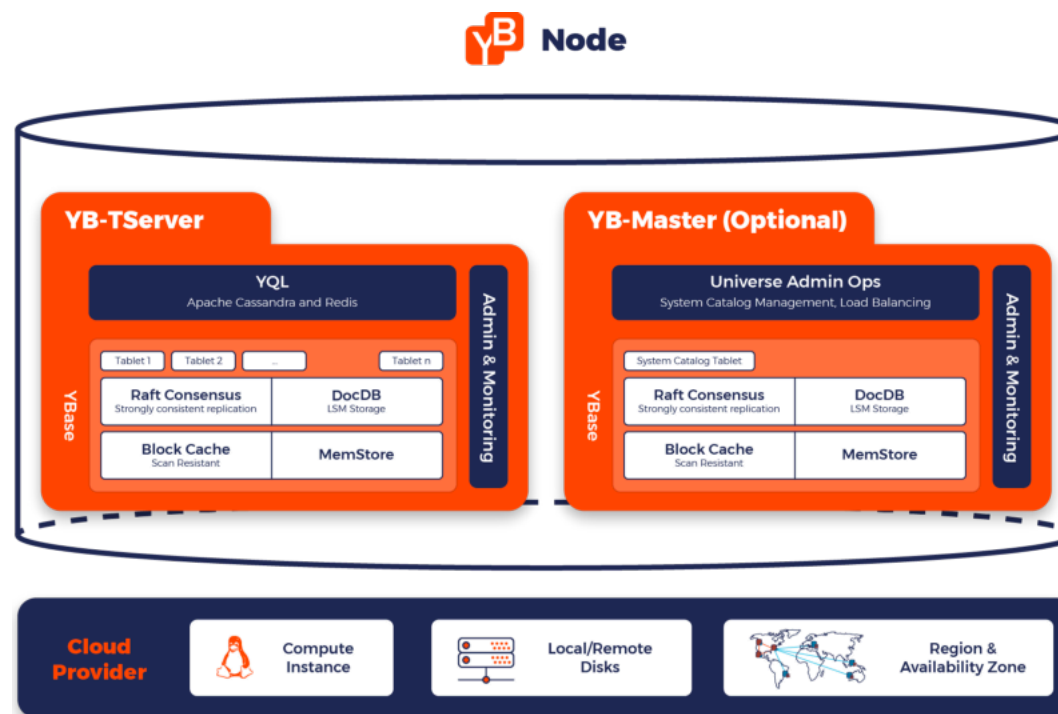


image source: <https://blog.yugabyte.com/yugabyte-db-architecture-diverse-workloads-with-operational-simplicity/>

Classifications (1)

According to Brian F. Cooper et al., in their [YCSB paper](#), there are four main tradeoffs faced by cloud serving systems:

- read vs write performance
- latency vs durability
- synchronous vs asynchronous replication
- data partitioning



Classifications (2)

Latency vs Durability

Write operations synced to disk before returning success will increase durability in the case of system failures. However it might lower throughput and increase latency.

Write operations stored in memory and synced later will increase throughput and decrease latency, but might increase the risk of data loss in case of system failures.



Classifications (3)

Synchronous vs Asynchronous Replication

Synchronous replication ensures consistency among all nodes but might increase latency.

Asynchronous replication decreases latency but might cause data loss if failure happens to nodes with data that not replicated yet.



Classifications (4)

Data Partitioning

Strictly row-based partitioning allows efficient access to an entire record of data.

Column-based partitioning allows efficient access for a subset columns when retrieving multiple records.



Classifications (5)

Database	Latency/durability	Sync/async replication	Row/column partitioning
CockroachDB	Durability	Synchronous	Row
TiDB	Durability	Synchronous	Row
YugaByte DB	Latency	Asynchronous	Row



YCSB



Yahoo! Cloud Serving Benchmark

Created by Brian F. Cooper et al. to create a **standard benchmark** and benchmarking framework to assist in the evaluation of different **cloud systems**.

Focus on **serving systems**, which serve read and write workloads, over batch or analytical systems.



Workloads Data

YCSB workloads data look like the following:

- 1 table named “usertable”
- 10 string fields, 1 primary key with content like “user123456”, 9 fields with content a random string of ASCII characters with 100 bytes length
- 1,000,000 records
- 1,000,000 operations with 100 to 1,000 threads for Workload A, B, C, and D
- 1,000,000 operations with 10 to 100 threads for Workload E



Operations

Operations performed by YCSB are:

- Insert: insert new record.
- Update: update a record by replacing the value of one field.
- Read: read a record, either one randomly chosen field or all fields.
- Scan: scan records in order, starting at randomly chosen record key with randomly chosen number of records.



Distributions

To choose which operations (insert, update, read, or scan) to perform on which records and how many records, YCSB has several built-in distributions:

- Uniform: choose an item uniformly at random.
- Zipfian: some item will be extremely popular, most records will be unpopular.
- Latest: like Zipfian with preference of latest inserted records as popular distribution.
- Multinomial: probabilities of each item can be specified.



Workloads

Workload	Operations	Distribution	Application Example
A – Update Heavy	Read: 50% Update: 50%	Zipfian	User session: session store recording recent actions
B – Read Heavy	Read: 95% Update: 5%	Zipfian	Photo tag: add tag is update, but most ops are reading tags
C – Read Only	Read: 100%	Zipfian	User profile cache
D – Read Latest	Read: 95% Insert: 5%	Latest	User status updates; people want to read the latest
E – Short Ranges	Scan: 95% Insert: 5%	Zipfian/Uniform	Threaded conversations



Experiments and Results



Cluster Setup

3 nodes GCE cluster with following specifications:

- n1-standard-16 machine type
- 1000 GB Local SSD
- 60 GB RAM



Statefulset Setup

Resource:

- 14 vCPU request, 16 vCPU limit
- 30 GB RAM request, 60 GB RAM limit
- 500 GB SSD local persistent volume



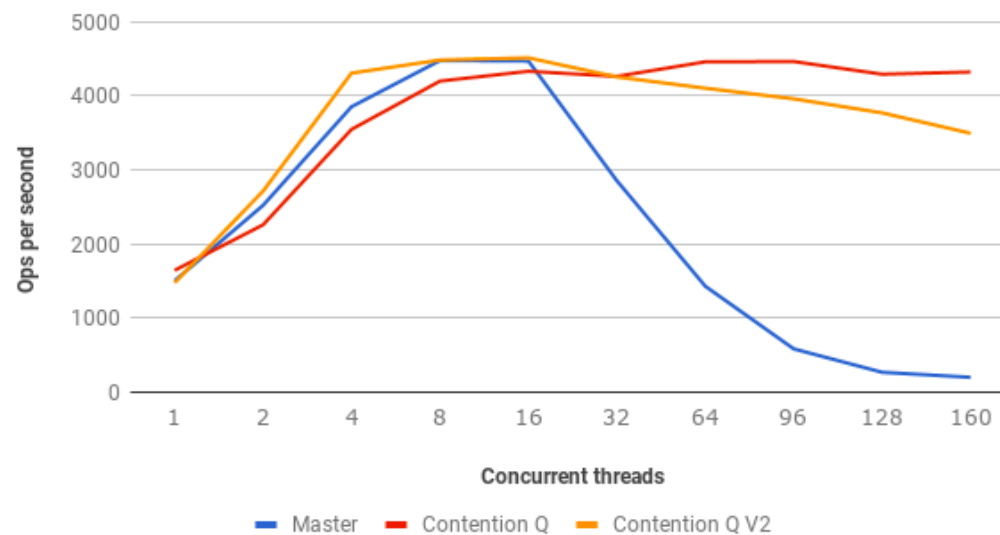
go-ycsb

All experiments in this presentation all done using a Go port of YCSB called [go-ycsb](#) created by engineers at PingCap, the company that creates TiDB.



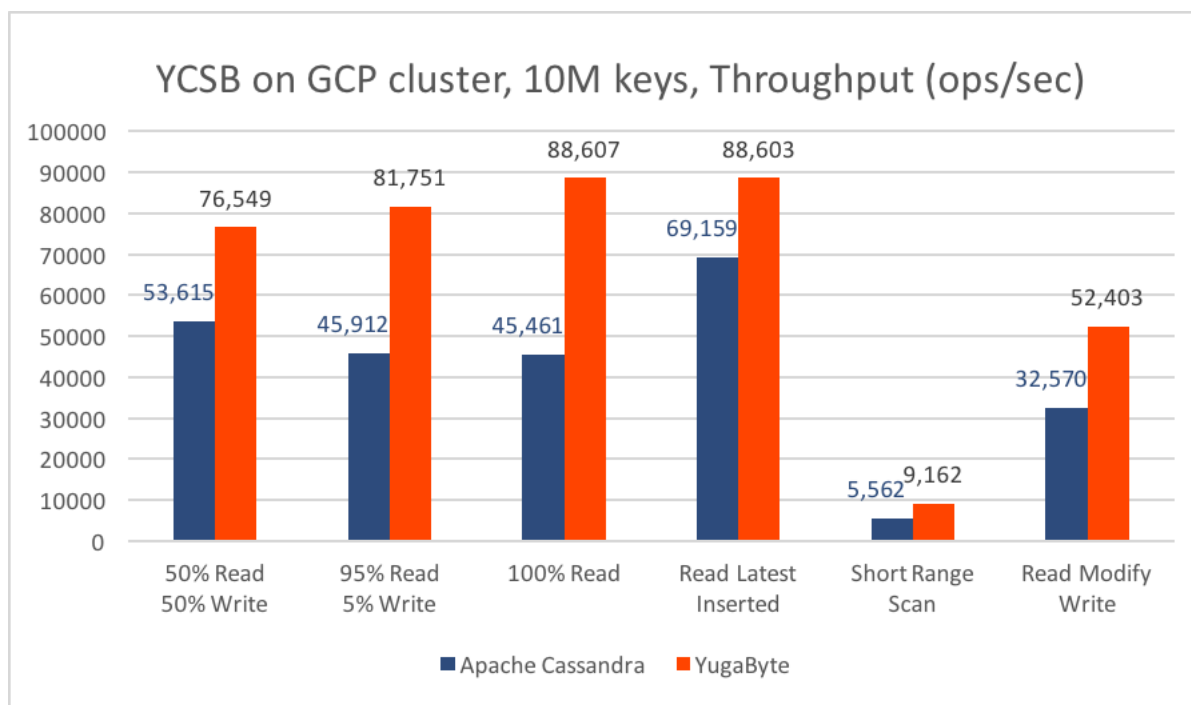
Baseline – CockroachDB

YCSB Workload A



source: <https://github.com/cockroachdb/cockroach/pull/25014>

Baseline – YugaByte DB

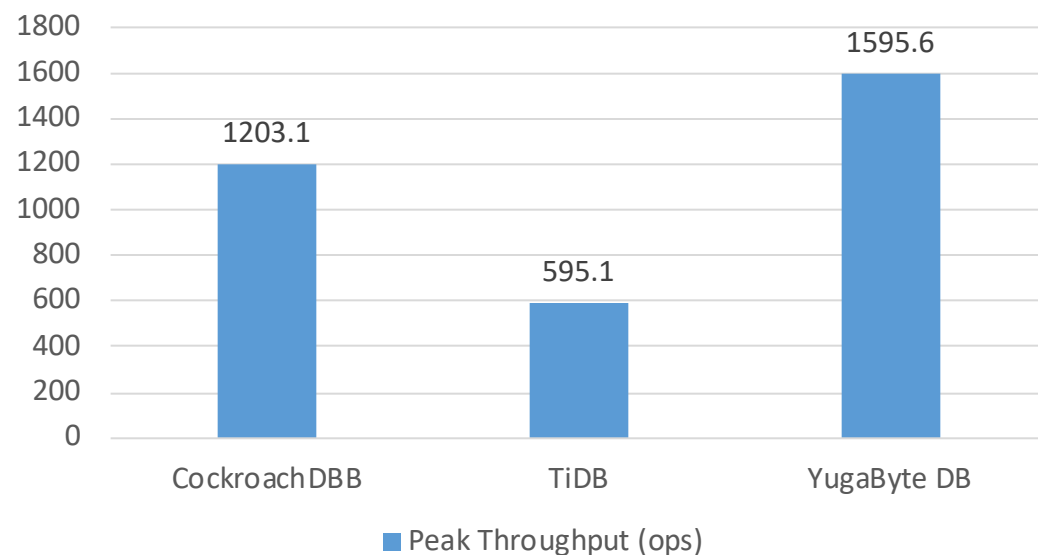


source: <https://forum.yugabyte.com/t/ycsb-benchmark-results-for-yugabyte-and-apache-cassandra/59>

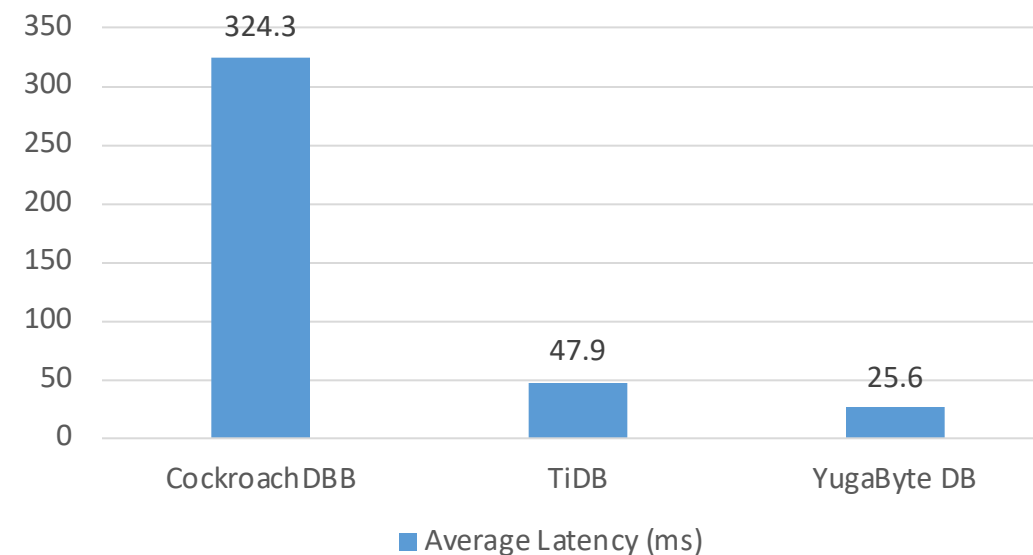


Workload A - Read

Throughput

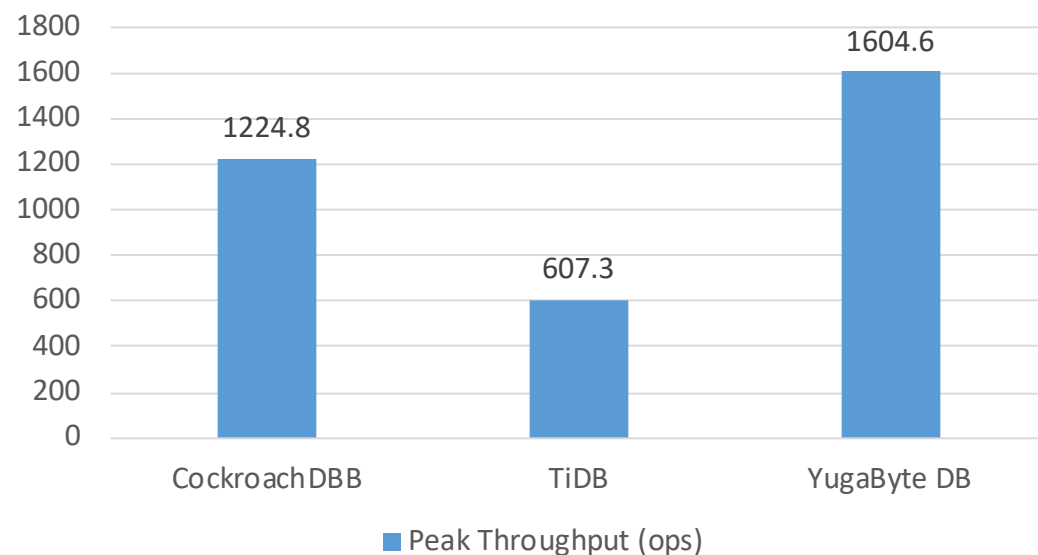


Latency

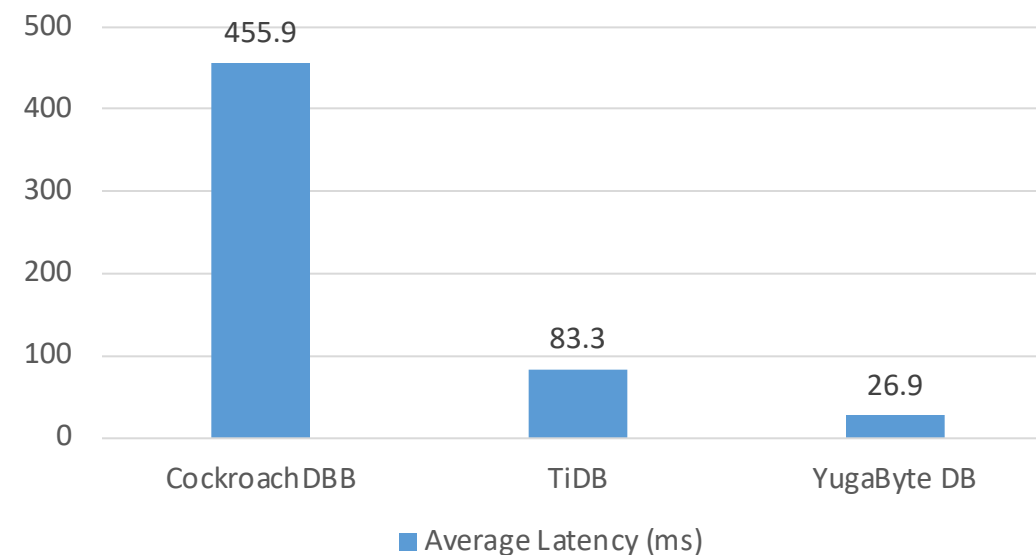


Workload A - Update

Throughput

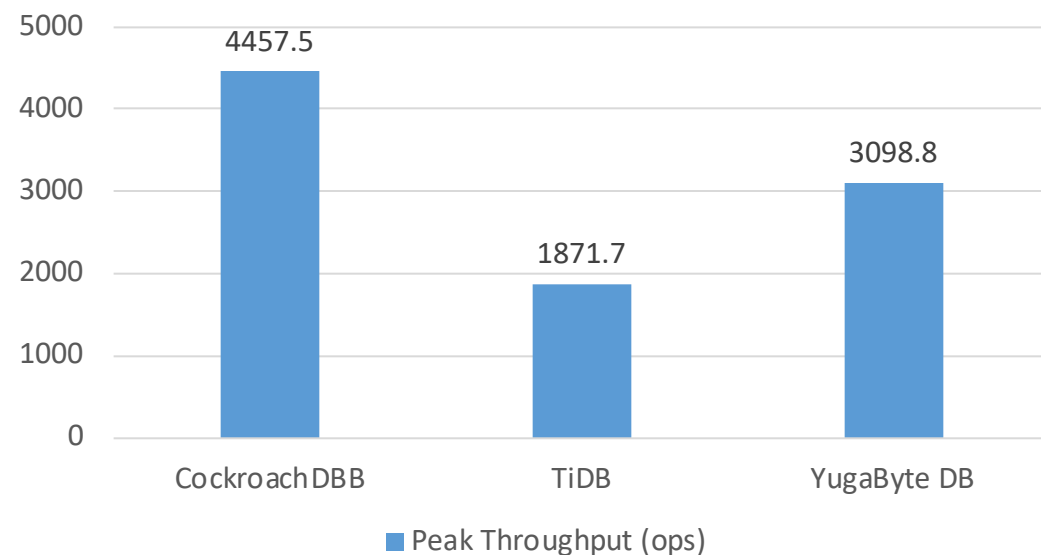


Latency

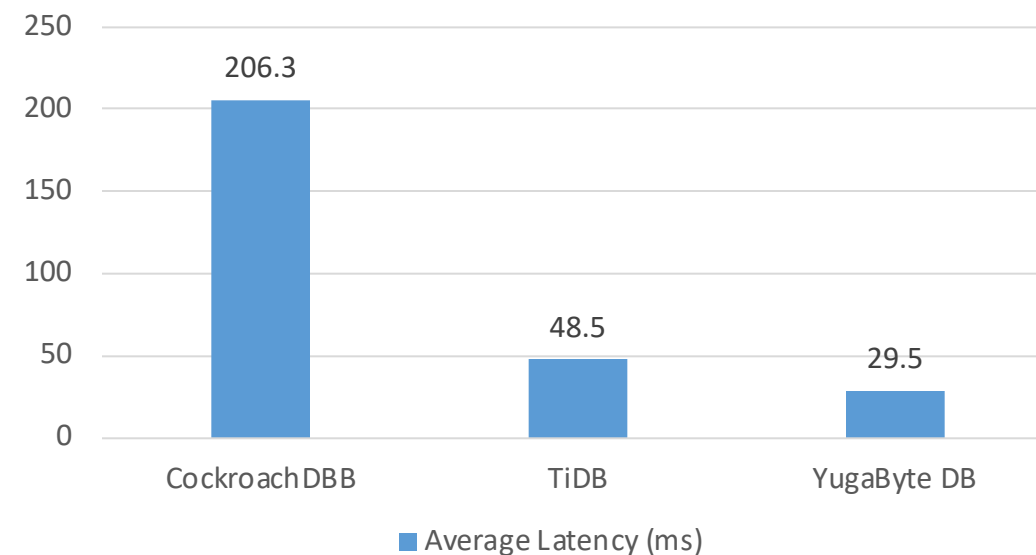


Workload B - Read

Throughput

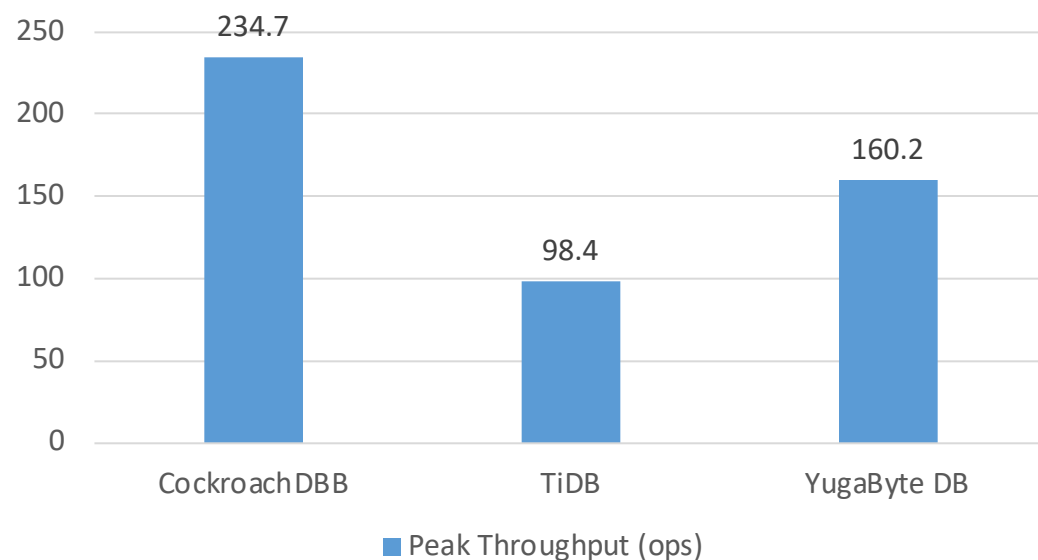


Latency

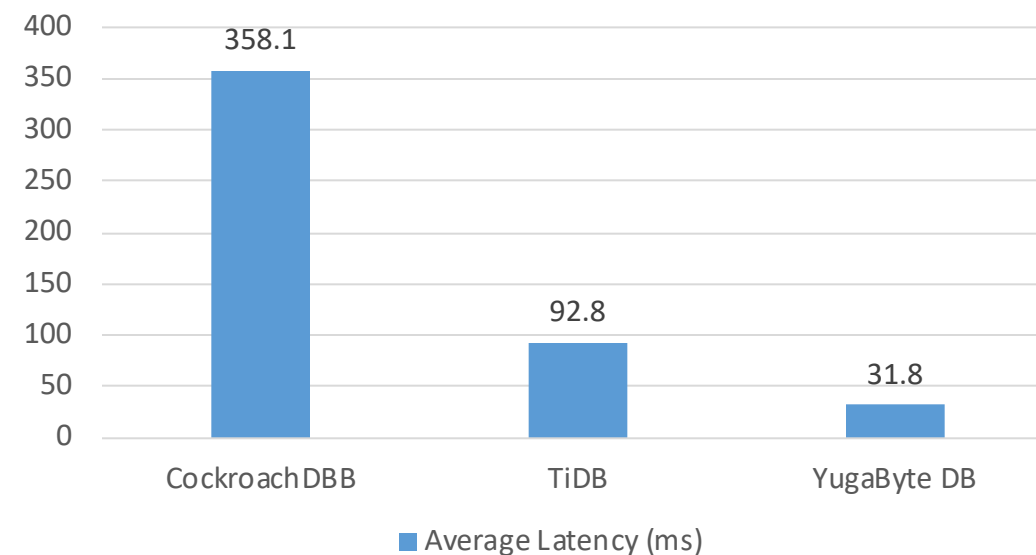


Workload B - Update

Throughput

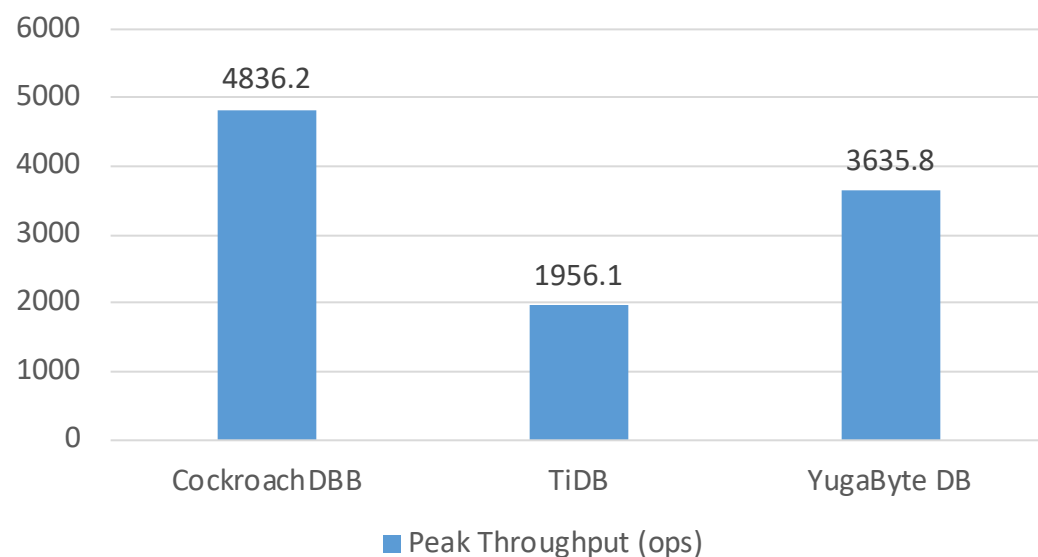


Latency

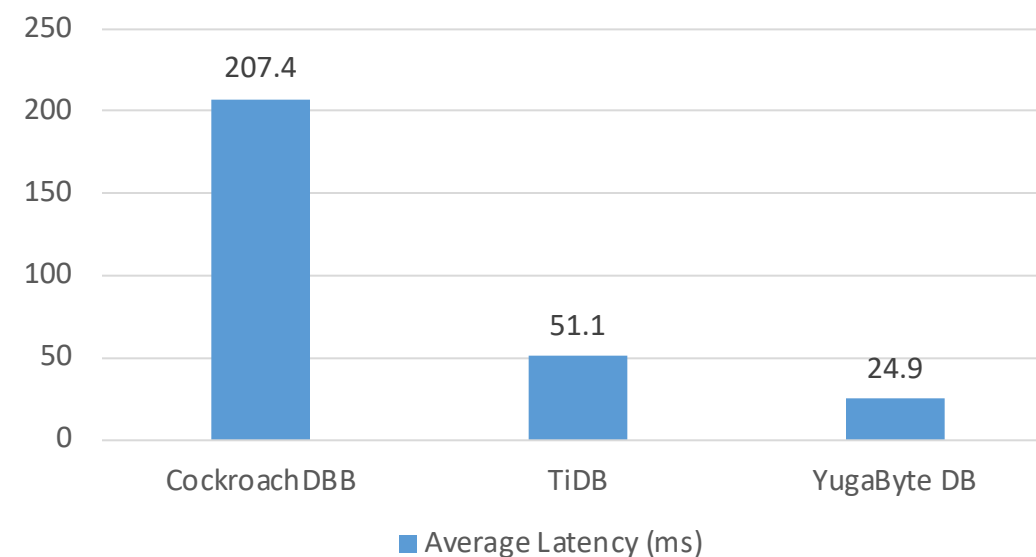


Workload C - Read

Throughput

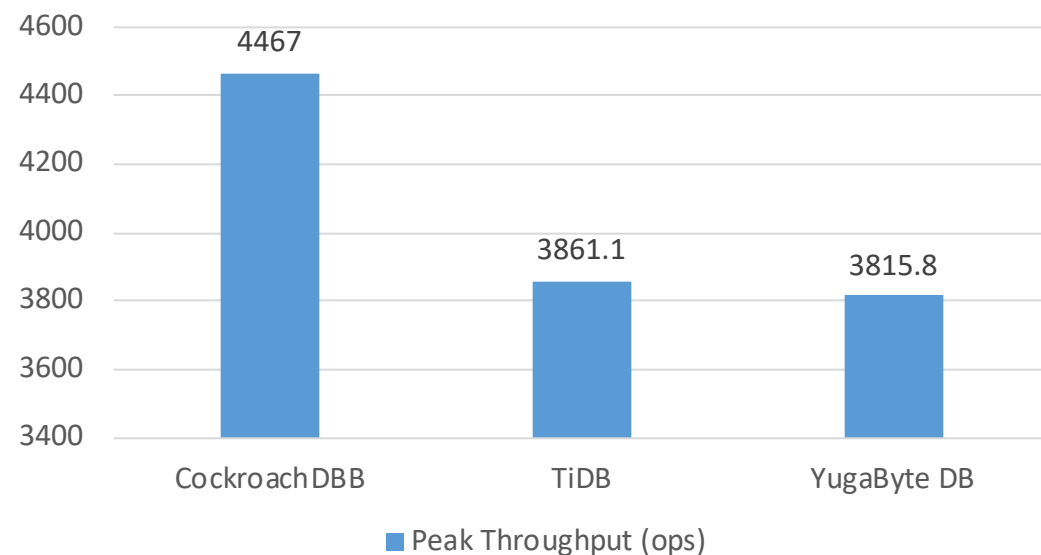


Latency

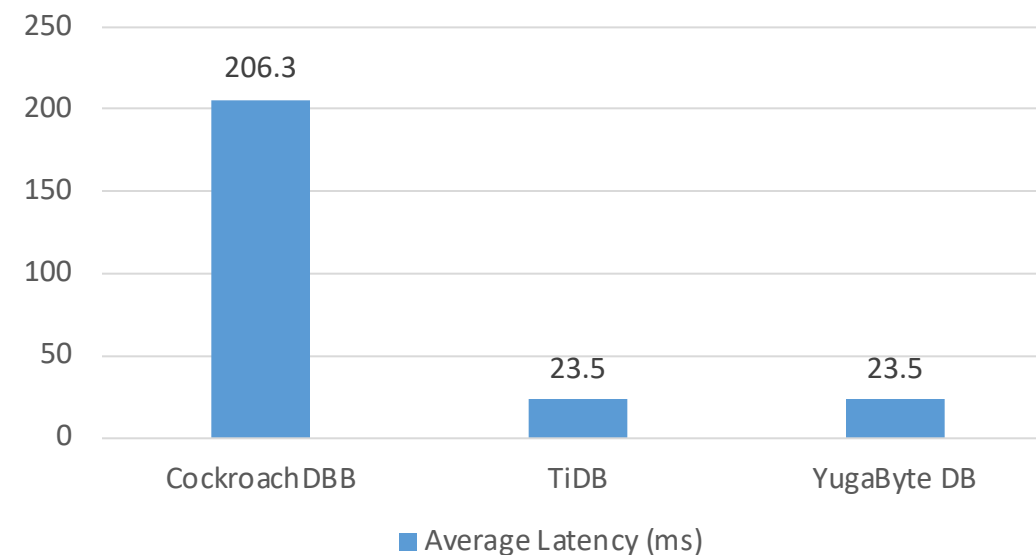


Workload D - Read

Throughput

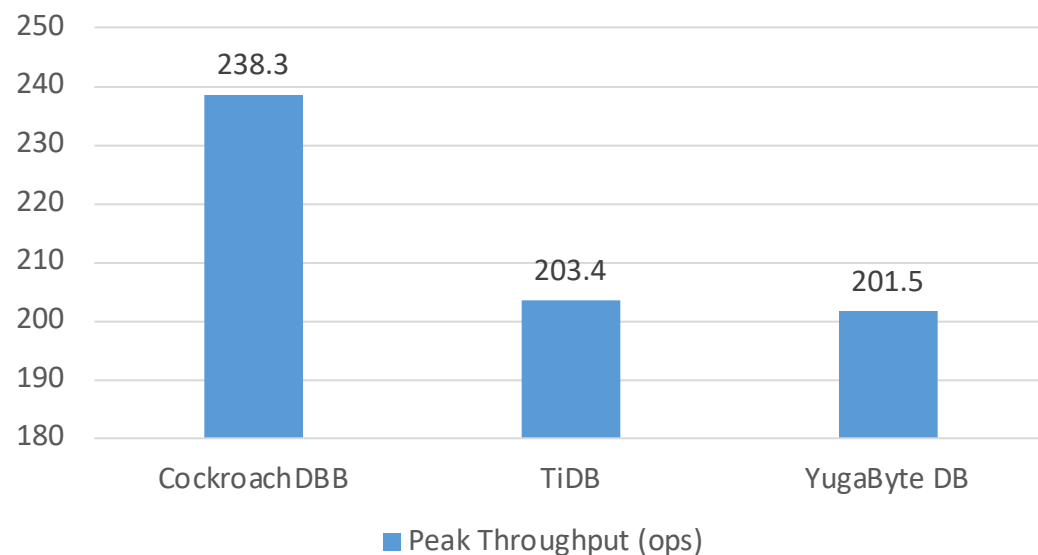


Latency

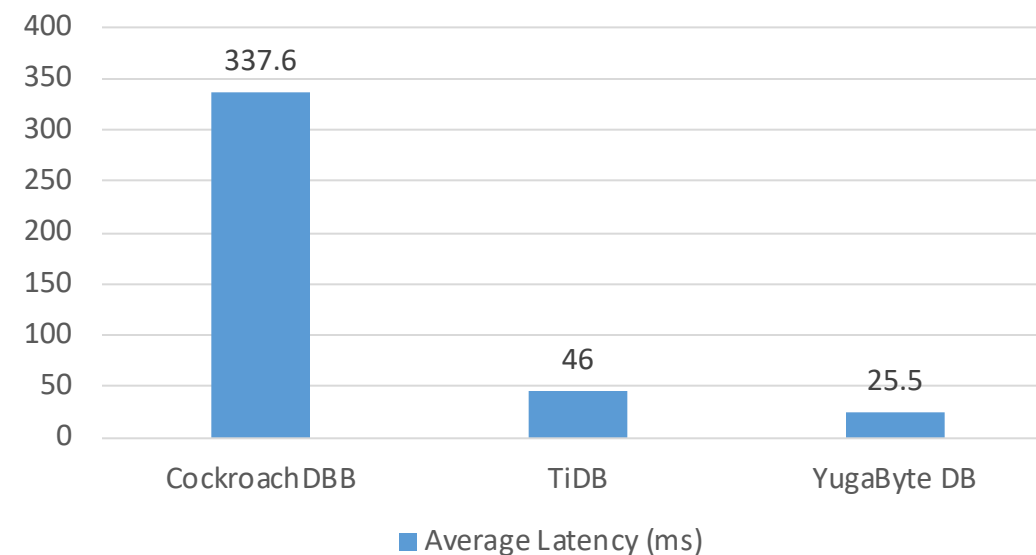


Workload D - Insert

Throughput

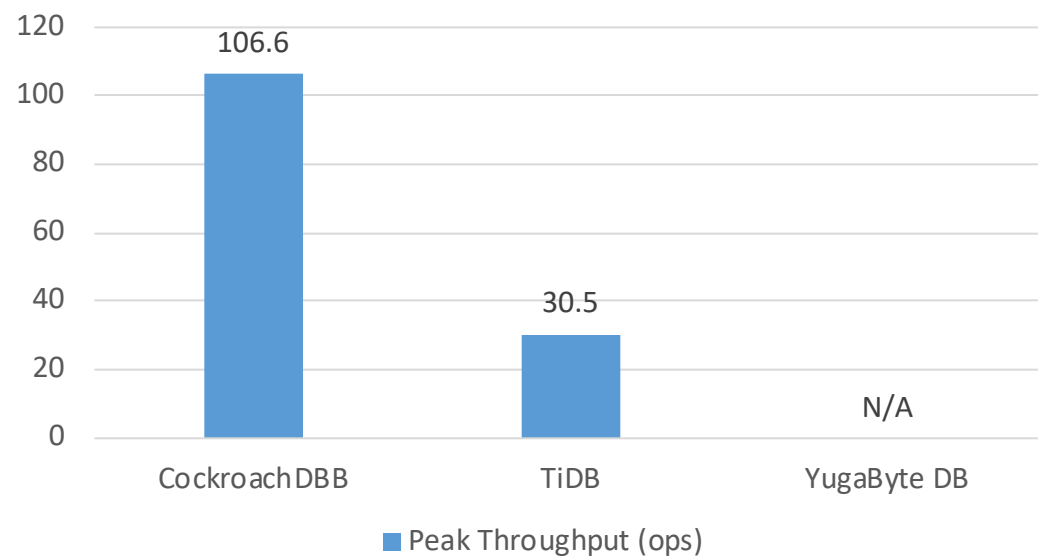


Latency

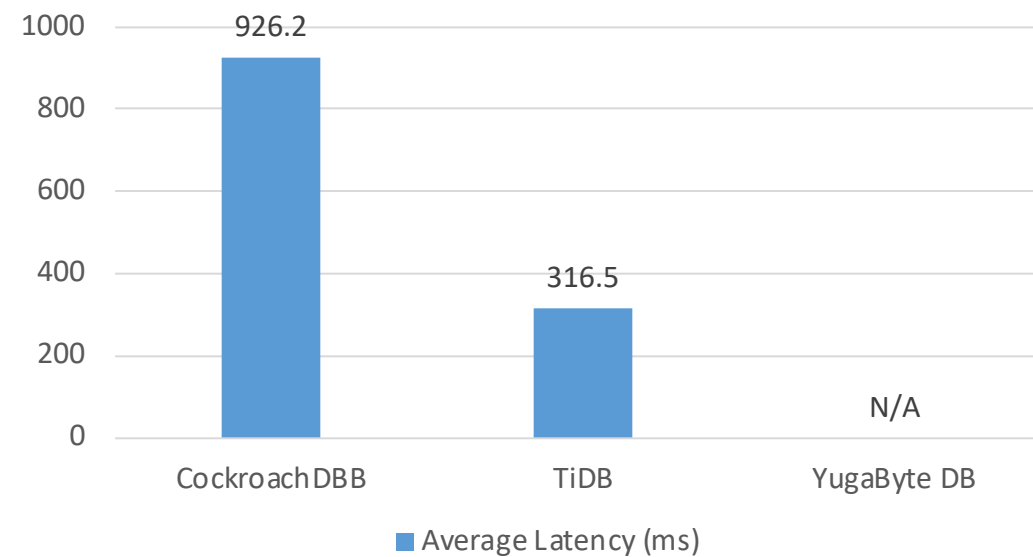


Workload E - Scan

Throughput

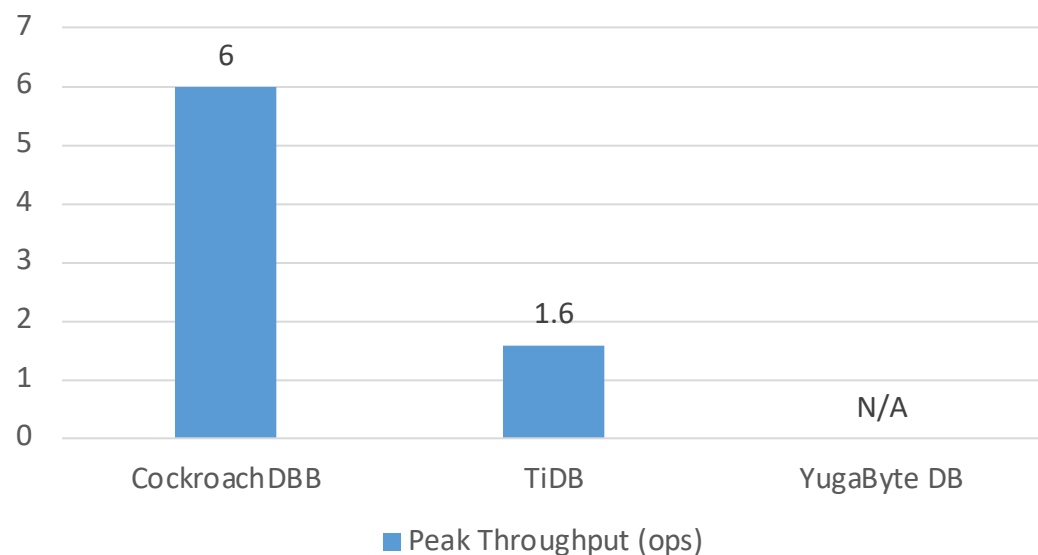


Latency

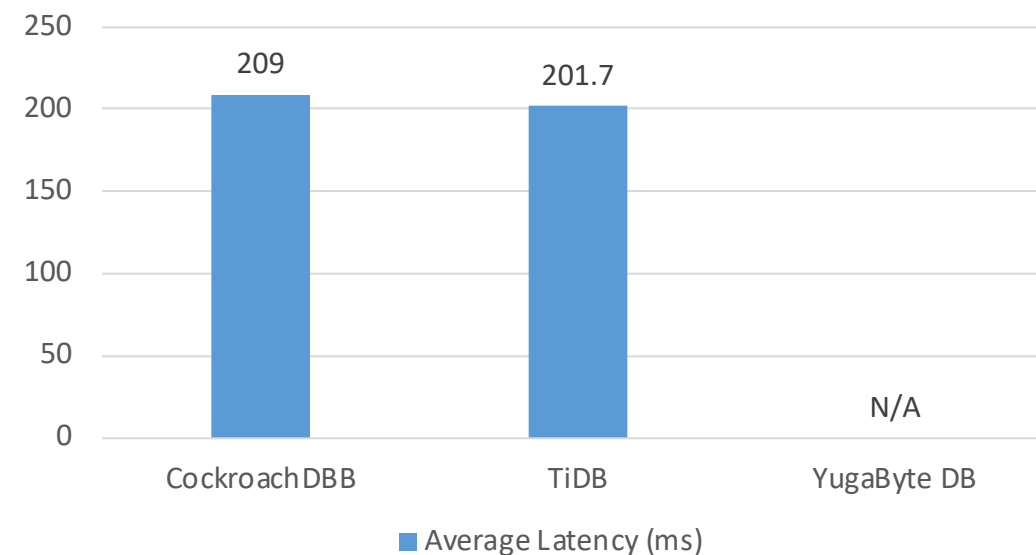


Workload E - Insert

Throughput



Latency



Discussions (1)

- All databases do not perform as well as they would if they run on a dedicated VM cluster with the same spec.
- All databases perform very well on read operations, especially in read-heavy workload (Workload B and Workload C).
- All databases perform fairly well on update operations in update-heavy workload (Workload A) but not so much in read-heavy workload (Workload B).



Discussions (2)

- All databases perform fairly well on insert operations in read-heavy workload (Workload D), but perform poorly in scan-heavy workload (Workload E).
- All databases does not perform well in scan operations in scan-heavy workload (Workload E).



Further Study

Considering we have yet to produce a satisfactory benchmarking results, we are planning to further conduct more experiments on this, especially regarding:

- Figuring out the bottleneck that prevent databases to perform as well as they do in dedicated VM cluster instead of on top of Kubernetes cluster
- Communicating closely with engineers from respective database to gain more insights on how to fine tune each database
- Working on more databases



Q & A





Gràcies!



KubeCon



CloudNativeCon

Europe 2019