Examensarbete

HÖGSKOLAN
I SKÖVDE

EN PRESTANDAJÄMFÖRELSE AV DATABASHANTERINGSSYSTEM ÖVER OLIKA WORKLOADS

A PERFORMANCE COMPARISON OF DATABASE MANAGEMENT SYSTEMS ACROSS DIFFERENT WORKLOADS

Bachelor Degree Project in Information Technology
Basic level 30 ECTS
Spring 2022

Alfred Jakobsson
Mário Le Duy

Supervisor: András Márki
Examinor: Yacine Atif

Abstract

This study conducted an experiment on NoSQL and NewSQL database management systems where the average throughput of Cassandra, CockroachDB, MongoDB, and VoltDB was compared using five workloads composed of different proportions of read and update queries. How much these different workload compositions affect throughput for each individual database management system was also investigated.

The results showed that VoltDB had the highest throughput overall, and its throughput was affected the least by the workloads' composition. MongoDB had similar high throughput consistency across workloads but at a much lower throughput level, and its throughput was affected much more by the workload compositions than VoltDB. Cassandra had extremely high throughput for 100 percent update workloads, even beating VoltDB in certain cases, but showed underwhelming results for all other workloads. CockroachDB's throughput was by far the worst at workloads that had any update queries, but was comparable and sometimes even better than Cassandra and MongoDB with 100 percent read workloads. CockroachDB's throughput proved to be the most affected by the query composition of workloads.

Keywords: Database, Benchmarking, Performance, NoSQL, NewSQL, Throughput, Cassandra, CockroachDB, MongoDB, VoltDB

Table of contents

1 Introduction

Most of the applications developed today by different companies or developers require storing and managing some form of data, thus choosing the right one for the task is essential. There are different types of database management systems available on the market to choose from right now, which makes the choice even more difficult.

NewSQL databases are very similar to relational-database systems but are designed to scale much better than the traditional standardized systems and work well with online transaction policies. NoSQL databases on the other hand are known for their scalability and usage in big data and IoT systems. A lack of studies shows how different NewSQL databases and non-relational databases perform when working with different types of workloads.

In this paper, an experiment is conducted comparing the performance of NoSQL and NewSQL database management systems that focuses on average data throughput across different workloads and how changes in query compositions are affecting these individual systems in terms of throughput. The database management systems chosen for this experiment are MongoDB, Cassandra, VoltDB, and CockroachDB. This study provides general information on the average data throughput of queries for anyone of interest, whether it is companies developers, or students.

2 Background

In this chapter, a brief explanation of the concepts used in this paper is described to give a better understanding of the performed experiment and the research in general.

2.1 Basic concepts

This section describes the basic concepts used in this study. These concepts are a prerequisite in subsequent explanations, and will also be used on their own elsewhere in the paper.

2.1.1 Database

Databases store information that can be accessed later on. Some of their strength compared to conventional file systems is that they can manage redundancy, inconsistency, and concurrent usage by multiple users (Khasawneh, AL-Sahlee & Safia, 2020).

2.1.2 Database management system

Database management systems (DBMS) are software systems where users can manipulate data in a database and are mainly used to store, retrieve and query different types of data. There are different operations that the user can take advantage of to manipulate the data such as read, write, update, and delete queries.

DBMSs can be split into three different categories, and they are as follows: relational database management systems (RDBMS), non-relational databases (NoSQL), and NewSQL databases which are a combination of RDBMS and NoSQL. Even though some of the DBMSs belong in the same category, they may not necessarily use the same query language as others.

2.1.3 Queries

Queries are requests that retrieve or alter the information stored in databases. They are structured by the query language in which they are written, which is usually defined by the DBMS. Some query languages are used in many different DBMS, like the popular Structured query language (SQL).

Multiple queries can also form what is called a Transaction. These Transactions are executed completely or not at all, which ensures that the database is in a proper state in case of mid-transaction failure.

2.1.4 Workloads

Workloads are a set number of tasks that need to be done or completed in a certain time frame, for this particular study a workload can be characterized by different parameters such as query types, number of queries/operations, record count, number of client threads, etc.

2.1.5 Relational database management system

Relational database management systems (RDBMS) store data in tables, using predefined schemas that represent relations between data. This type of storage enables RDBMS to handle complex querying with relative ease (Khasawneh, AL-Sahlee & Safia, 2020). RDBMS supports strong transactional reliability by adhering to four properties (Atomicity,

Consistency, Isolation, Durability) called the ACID properties (Frank L., Pedersen, Frank H.C. & Larsson, 2014):

- Atomicity, either all or none of the transaction's updates are executed.
- Consistency, the database is consistent both before and after the execution of a transaction.
- Isolation, locks prevent transactions from interfering with each other.
- Durability, transactions are stored stably and can be recovered in case of a server restart.

Queries in most RDBMS are written in SQL, which is why they are also called SQL databases (Khasawneh, AL-Sahlee & Safia, 2020).

2.2 NoSQL

NoSQL databases are often referred to as non-relational DBMSs that are schema-free meaning that they can support all kinds of data whether it is structured, semi-structured, or unstructured data. Some of the NoSQL features provide the users with high scalability and availability that standard relational database management systems such as MySQL do not.

All NoSQL databases are built based on the CAP theorem (Consistency, Availability, and Partition). These three criteria need to be taken into consideration when building a new solution, but it has been proven that no more than two criteria can be fulfilled for a distributed system architecture (Khasawneh et al., 2020).

Consistency: Data must be distributed between all nodes equally when there is an update in the system so that all clients can view and modify the received data to achieve consistency.

Availability: Despite the potential failure of nodes, the system must still be up and running to execute any requests from the client.

Partition Tolerance: In case of network or connection failure between two different nodes in the cluster, the system must still be up and running.

There are four different data models surrounding the NoSQL databases that can be categorized into Key-Value store, Document-oriented, Graph-oriented, and Column-oriented databases (Anusha et al., 2021). MongoDB, Cassandra, Redis, and CouchDB are some of the more popular NoSQL databases that are offered on the market.

2.3 Column-Oriented databases

Column-oriented databases store their data in columns rather than in rows to optimize the performance of real-time analytics in different applications. Tables consisting of columns are stored as rows for the optimization to work (Khasawneh et al., 2020). A collection of columns is called a column family, and multiple columns can be added to the family of columns without having to pre-announce them. Column-oriented databases perform better when handling heavy write operations like log aggregation for various applications (Jogi &

Sinha, 2016). Examples of well-known column-oriented databases are HBase, and Cassandra (Anusha et al., 2021).

2.3.1 Cassandra

Cassandra is an open-source non-relational distributed database system that provides services with no single point of failure and high availability. Cassandra offers a peer-to-peer architecture that removes the usual constraints following the master-slave architecture regarding scalability and high availability. The benefit of using Cassandra is that it can store different kinds of data, whether it is structured, semi-structured, or unstructured data. While it is a column-oriented data model, it is not bound to the usual way of storing data in tables with rows and columns. It is possible to have rows that do not have similar columns in the database (Anusha et al., 2021).

Cassandra sacrifices consistency and focuses more on availability and partitioning regarding the CAP theorem, which makes it an AP system (Araujo et al., 2021).

Any kind of data is usually stored in a *node* which is the basic component of Cassandra, while a collection of nodes is called a *data center* and a collection of data centers is called a cluster (Magdum & Barhate, 2018). Cassandra uses its own language called Cassandra Query Language (CQL) to manipulate data, which works in a similar fashion to SQL (Gupta & Mittal 2020).

2.4 Document-oriented databases

A document-oriented database belongs to one of the four basic groups of NoSQL databases. The most common way to store and retrieve documents in this type of database is in a JSON, BSON, XML format. Every document that is created in the database is connected to a key, and the documents contain different fields and values. (Hashem & Ranc, 2016). Popular DBMS of this type are MongoDB, CouchDB, and RavenDB (Filip and Čegan, 2020).

2.4.1 MongoDB

MongoDB is an open-source NoSQL document-oriented DBMS. The collected data in MongoDB is stored as BSON (Binary JSON) documents. Collections in MongoDB organize the collected data, and each collection can have multiple documents inside of them. These documents are building the database and are composed of fields with different data types and values. MongoDB is schema-free which means it can handle unstructured data very well and provide high scalability and availability (Eyada et al., 2020).

MongoDB sacrifices availability and focuses more on consistency and partitioning regarding the CAP theorem, which makes it a CP system (Gupta et al., 2017).

2.5 NewSQL

NewSQL is a modern class of RDBMS that aims to provide similar scalability to NoSQL while maintaining ACID properties. As such, NewSQL can execute a large quantity of concurrent transactions using SQL instead of proprietary APIs, which assists development by diverting

attention away from API-specific logic and eventual changes due to updates (Pavlo & Aslett, 2016).

Most NewSQL DBMSs scale by splitting the data into separate subsets called partitions or shards. These partitions are created from closely related data fragments from multiple tables, where relationships are usually identified by the (primary/foreign) key property. All data concerning a specific entity is stored on a single partition. For example, a customer's information (from the customer table) is stored together with their order records and account details (Pavlo & Aslett, 2016). This means most transactions will only need to access data from one partition, thus reducing communication overhead.

Each partition is managed by a single node, where queries requiring data from the partition are executed. Furthermore, a query's execution can be divided to multiple partitions to later be converged into a single result (Pavlo & Aslett, 2016).

2.5.1 VoltDB

VoltDB (also known as Volt Active Data) is a NewSQL DBMS which stores data in memory to maximize data throughput, as it targets (and is optimized for) applications that require large amounts of data fast, like financial applications or social networks (Schreiner et al., 2019).

VoltDB was released in 2010 and is based on H-Store, one of the earliest NewSQL DBMS developed. It scales by partitioning data over multiple servers forming what is called a cluster. Each server within a cluster contains a number of processors (called sites) that store partitions. Sites also contain the entire execution engine and a queue for incoming transactions, which allows the cluster to execute requests in parallel, as long as requests only require data from one partition (Schreiner et al., 2019).

VoltDB can improve performance further by allowing entire tables to be replicated (copied) into partitions, which reduces cross-partition communication if the table data is accessed frequently. VoltDB also optimizes performance based on received queries, by creating copies of tables that are frequently used (Schreiner et al., 2019).

2.5.2 CockroachDB

CockroachDB is a NewSQL DBMS developed in 2015 which primarily focuses on fault tolerance, availability, and the effective distribution of partitions and replicas over a large geographical area. It also focuses on transaction performance, with cross-partition transactions as the main point of improvement (Taft et al., 2020).

CockroachDB stores data on disk using a key-value format, which is fast for both write and scan queries. The key-value store cannot interpret whole SQL statements, so a separate layer in each node converts them into low-level read and write requests which it can manage (Taft et al., 2020).

Queries are not only converted, but also optimized. With CockroachDB's geographically distributed and partitioned nature, most optimizations unique to CockroachDB target just that. For example, assigning a cost to replicas depending on its distance to the gateway node of a query (Taft et al., 2020).

CockroachDB has two different query execution engines; Row-at-a-time execution engine, which processes one row at a time and is compatible with all SQL features supported by CockroachDB. Vectorized execution engine operates on column-oriented batches of data, which has lower interpreter overhead (due to row-at-a-time's iterative nature) but can only execute a subset of SQL queries (Taft et al., 2020).

3 Problem

The problem chapter presents the aim of this study and the motivation behind it, followed by the research question that needs to be answered at the end of the study. Finally, the objectives and hypotheses are discussed.

3.1 Aim

This study aims to investigate how well different DBMS perform in terms of data throughput across different workloads. In this study, we specifically aim to compare the following DBMS: MongoDB, Cassandra, CockroachDB, and VoltDB.

3.2 Motivation

Throughput (the number of operations per second) is chosen for comparison as it is a common measurement of DBMS performance. Throughput gives an indication of how fast the DBMS is, which could affect the number of users it can service simultaneously or the amount each user can interact with the database.

Previous studies have compared the throughput of DBMS before, like Schreiner, Knob, Duarte, Vilain, and Mello (2019) study of four different NewSQL DBMS, which also included a latency comparison. However, no studies were found that compared the throughput of DBMS using different workload compositions.

Knowing how well databases can handle workloads that are read-heavy, update-heavy, or some intermediate step in between is valuable information for all developers or database managers attempting to use databases. This is especially helpful when picking the most suitable DBMS for the application's predicted workload.

Different dataset sizes were used in this study to show how well DBMS throughput scales with larger datasets. Knowing how well throughput scales with dataset size can make it easier for developers to select the most suitable DBMS for their application's expected usage. Note, due to mistakenly also increasing the number of operations together with dataset size, results from each dataset size cannot be reliably compared to each other. Ultimately, this meant that throughput scaling comparisons between different dataset sizes had to be dropped from this study.

NoSQL and NewSQL DBMS were chosen for comparison as both are suitable for handling the increased size and variety of structured, semi-structured, and unstructured data known as Big Data (Li & Manoharan, 2013; Schreiner et al., 2019). Including two different database types allowed for a broader search for newer DBMS to compare.

Big Data is also often accompanied by huge amounts of users and user interactions with the system (Schreiner et al., 2019). As users desire and expect a responsive system experience, the importance of good data throughput rises, making it an interesting topic for study.

3.3 Research Questions

These are the research questions covered in this study:

RQ1: Is there a significant difference between Cassandra, CockroachDB, MongoDB, and VoltDB in terms of average data throughput?

RQ2: Does the read/update query composition of workloads have a significant effect on average throughput for Cassandra, CockroachDB, MongoDB and VoltDB?

3.4 Hypothesis

Each research question has a corresponding null hypothesis, listed below.

H1: There is no significant difference in terms of average data throughput between MongoDB, Cassandra, CockroachDB, and VoltDB.

H2: The throughput of MongoDB, Cassandra, CockroachDB, and VoltDB is not significantly affected by the read/update query composition of workloads.

3.5 Objectives

1. Identify which DBMS and benchmarking tool to include in the study [Both]
2. Setup environment and compatible workloads to compare the different DBMS [Individually]
3. Compare and contrast the selected DBMS concerning average data throughput and different workloads [Individually]
4. Analyze and evaluate results to answer research question [Both]

3.6 Responsibilities

Division of work between researchers can be seen in Table 1 below. It shows who was responsible for researching and benchmark testing each DBMS.

Table 1 Division of responsibilities

DBMS/Name	Mário	Alfred
MongoDB	X	
Cassandra	X	
CockroachDB		X
VoltDB		X

4 Method

This chapter goes over the chosen research method used for this study and different alternative methods that could have been used for the study instead.

4.1 Experiment

A controlled experiment analyzing the four different DBMSs was conducted. Doing a controlled experiment generated empirical data that was easier to read and make a comparison out of, thus reducing the time spent on analyzing the results. Having control over the different variables and avoiding external factors affecting the results is very important. By conducting a controlled experiment, we can see the cause and effect of our inputs on the output (Wohlin et al., 2012).

4.2 Alternative methods

Experiment was not the only method considered for this study. The methods not implemented were simply not suitable for a comparative study such as this.

Case studies are one of those methods. It would be technically possible to measure data throughput at companies or organizations if permitted. However, a case study would limit DBMS choice as some (newer) DBMS are not widely used commercially. Also, it is impossible to control the amount of incoming requests and the workload composition in live environments, as queries would originate from real users. This could generate unreliable (and less generalizable) results as throughput would be dependent on the number of active users, and their imposing workload could favor one type of query, e.g., read queries. Data throughput of DBMS has been studied for decades, which also makes it a poor study topic for a case study as they are most suitable to explain phenomena in fields not yet understood (Berndtsson, Hansson, Olsson & Lundell, 2007).

Interviewing experts or developers of selected DBMS is also a possibility. However, it would be difficult to form questions that generate useful answers regarding data throughput using different workload compositions. The aim of the study would need to be changed, which is the main reason why interviews were discarded. Another reason is that experts and developers are generally quite busy and might not have time for interviews.

Questionnaires could be used instead of interviews, reaching a broader audience which could make the results more generalizable. However, it would be impossible to control who is answering the questionnaire, and due to its subjective nature, answers regarding performance in any kind would be unreliable and ultimately useless in a comparison. Again, the method does not fit the aim of the study, which is why it was discarded.

4.3 Approach

First, research articles comparing DBMS performance were studied to give insight into which independent and dependent variables are of interest to DBMS consumers and researchers. The selected variables can be seen in the list below.

Dependent variables: Throughput (number of operations per second).

Independent variables: Read/update query composition of the workload, workload size in terms of number of operations, and dataset size in terms of number of rows.

After interesting variables were identified, available benchmarking tools that could manage them were researched and compared. From these, the benchmarking tool Yahoo! Cloud Serving Benchmark (YCSB) was chosen for its widespread use in the research community. The datasets used in benchmarking tests were also generated by YCSB.

After the benchmarking tool was chosen, the list of compatible DBMS was examined and we selected four DBMS based on type (NoSQL and NewSQL) and age (when the initial release date was). Selecting two NoSQL and two NewSQL DBMS meant that the work was more easily divided among the two researchers and allowed for a broader selection of newer DBMS. Selecting (and comparing) older and newer DBMS from each type was meant to put the results in a familiar context. Initially, ScyllaDB was selected as the newer NoSQL DBMS, but incompatibility with the benchmarking setup meant that it was later replaced by MongoDB. MongoDB is much older than ScyllaDB, but it was chosen as it was almost guaranteed to work with the benchmarking setup (which we really could not afford to change at that point).

To be able to start benchmark testing, the servers hosting the databases needed to be set up. The databases were hosted on computers in the network and administrators lab at the University of Skövde, and each individual set up their assigned DBMSs and made sure they worked properly. Each DBMS was installed on a separate disk to reduce potential interference, as the databases were locally hosted. The benchmarking tool YCSB was also installed on each disk, this allowed for concurrent benchmarking as all computers in the computer lab have the same hardware.

Benchmark tests with YCSB started with loading (inserting) randomly generated text data, in this study, benchmark tests were performed on datasets of 10 000, 100 000, and 1 000 000 rows. The workloads were also run with a different number of operations, matching the size of the dataset, for example, workloads with a size of 10 000 operations were exclusively run on datasets of 10 000 rows, workloads with 100 000 operations were exclusively run on datasets of 100 000 rows, and so on.

Changing both dataset size and the number of operations simultaneously was done due to an incorrect assumption on how YCSB worked, namely that they both had to match. It was not taken into consideration that the number of operations could affect throughput. Changing two independent variables meant their exact effect on throughput could not be determined (more on that in threats to validity and future work sections).

After the dataset is loaded, the YCSB benchmark can be run with a specified workload, composed of different proportions of read, update, scan, insert, delete, and read-and-modify queries. This study only used workloads with combinations of two selected query types (read and update), as the cause and effect would be more difficult to distinguish when using workloads of three or more query types. Read and update were chosen as they seemed to be the most commonly used query types of the ones available with YCSB. One important thing to note is that the YCSB update query does not read before updating, it assumes the row that will be updated exists within the database.

Workloads included in this study and their query composition can be viewed in table 1. Workload B for example is composed of 75% read and 25% update queries.

Table 2 Query composition of workloads A-E

Workload	**Read percentage**	**Update percentage**
Workload A	100%	0%
Workload B	75%	25%
Workload C	50%	50%
Workload D	25%	75%
Workload E	0%	100%

Every benchmark test workload configuration was run ten times to get an average throughput value, which is a more reliable measurement and less susceptible to variance. The 1 000 000 dataset size tests were only run five times each, as they took very long to complete for some DBMS.

When all benchmarking tests had been completed, the average throughput results were visualized using boxplots and line charts.

5 Related work

Studies evaluating or directly comparing database performance have been done previously, for both NoSQL and NewSQL databases. None, however, have included the four databases in this study while measuring data throughput across different workloads. Still, interesting research with similarities to this study has been found.

Kaur and Sachdeva (2017) performed a qualitative comparison of four NewSQL databases, VoltDB, MemSQL, CockroachDB, and NouDB, two of which are included in this study. They also benchmarked these DBMS, but focused on query latency and execution time instead of throughput.

The latency was measured on read, write and update queries, which showed that nouDB had the lowest latency for all but update, in which memSQL was faster. CockroachDB had the highest latency for all queries while having relatively low execution time (although not compared to nouDB which was the fastest). Of the three query types, VoltDB seemed to excel at write queries, but had the longest execution time of all four DBMS.

Schreiner et al. (2019) compared the throughput and latency of VoltDB, MemSQL, CockroachDB, and NouDB. This article includes two DBMS from this study but more importantly, also benchmarks throughput. They also used the same benchmarking tool YCSB for measuring throughput, and also showed a throughput comparison between two benchmarking tools by including the tool Voter.

For YCSB, memSQL had the best results while VoltDB showed to be the most stable (in terms of throughput variation). CockroachDB had the worst results, believed to be caused by storing data on disk instead of (much faster) memory. In Voter, memSQL has the best throughput and latency (about equal with VoltDB), although the latency difference is much smaller for all DBMS, due to Voter using smaller transactions. CockroachDB still has the lowest throughput in Voter but showed improved stability.

Cui and Chen (2021) conducted an experiment comparing the two mainstream column-oriented DBMSs (HBase and Cassandra), focusing on the performance of read and write operations and also using the benchmarking tool YCSB for their experiment. The results were based on certain predefined workloads created by the benchmarking tool that was run against generated dataset sizes of 3GB and 10GB. Similarly to our experiment, we compared Cassandra against other DBMS using different workloads and data sizes. Cassandra proved to be performing much better than HBase when it came to write and single read/write operations. HBase, on the other hand, performed better at mixed write and read operations.

Magdum and Barhate (2018) did conduct an experiment investigating the performance of two different NoSQL databases (Cassandra and Elasticsearch), but the aim of this study was to see which of them was better at handling streams of data using a Twitter Streaming API and Python. Our study, on the other hand, was more focused on the throughput aspect when comparing Cassandra and the other DBMS. The authors analyzed insert, search, update, and delete operations. Based on the results, Cassandra performed better for operations that required higher scalability, but Elasticsearch did outperform Cassandra in execution time when the data needed to be modified or aggregated.

Jansson, Vukosavljevic, and Catovic (2021) have performed a similar thesis project at the University of Skövde, where they benchmarked and compared the query latency of multi-model, key-value, and document NoSQL DBMS. They also used the same benchmarking tool YCSB for their tests. Their paper was used as the primary source of inspiration regarding the general approach, benchmarking setup, benchmarking tool, and report structure.

6 Experiment

This section explains how the experiment was set up and run. It also includes the results generated from the experiment and the conclusions formed around them.

6.1 Benchmarking setup

All DBMS were hosted in the University of Skövde network and administrators lab using machines with the following hardware:

- Intel(R) Core(TM) i5-6600 CPU @ 3.30GHz, 4 cores
- 32 GB DDR4 RAM @ 2133MHz
- 160 GB HDD

The operating system Ubuntu 20.05 LTS (server) was installed on all machines and the DBMS and benchmarking tool was set up via command-line instructions. CockroachDB and MongoDB had to be built from the source, meaning version-specific packages needed to be installed manually. Meanwhile, VoltDB and Cassandra were installed from extractable archive files.

All DBMSs consisted of a single node cluster since we were not interested in investigating how the DBMSs reacted to potential node failure or certain aspects of data replication but the overall throughput performance itself.

All DBMSs were also configured with their default cache settings, as database optimization was not a part of this study and would require a lot more tests. Database optimization is not our expertise, so further changes to the databases could also jeopardize the validity of the comparisons.

6.1.1 Benchmarking tool (YCSB)

The Yahoo! Cloud Serving Benchmark (YCSB) is a popular tool for measuring (and comparing) performance across DBMSs with different storage methods and query languages. YCSB's approach for fair cross-DBMS comparisons is to only provide tests for common elements prevalent in all DBMS. This includes limiting dataset generation to synthetic strings for both keys and values, as well as only running simple create, read, update and delete operations on primary keys (Reniers et al., 2017).

YCSB manages differences in query language by implementing a database interface layer for each database, which executes queries in their native query language (Reniers et al., 2017). It also handles various storage methods by assuming that they are all built upon key-value stores, where the value can be a document, a collection of columns, a node with edges, or some other new form of storage (Reniers et al., 2017).

YCSBs approach for cross-DBMS fairness gives confidence in the results later in this section.

YCSB workload distribution of queries on the other hand is not exactly even as shown in *Table 1*. Take, for example, Workload B, it will not execute exactly 75% of read and 25% of update queries based on the selected number of operations although very close to that ratio.

But since we are running each workload ten times, it will give us an average that is very close to those numbers.

To make the benchmarking tests more comparable, certain parameters such as client thread count and operation count had the same value for all of the performed experiments for each DBMS. All of the benchmarking tests ran with 32 client threads, thus ensuring a constant flow of requests to the database. This negates the risk of the client causing a throughput bottleneck where not enough requests are sent to the database at any given time. Avoiding this bottleneck also indirectly speeds up the testing process by increasing throughput. The client threads operate sequentially in YCSB benchmark tests (YCSB, 2021), which means there is no risk of concurrency-related issues, and all operations during the tests were completed successfully.

The operation count parameter was selected based on the corresponding dataset size, which in this case was either 10 000, 100 000, or 1 000 000. This means that depending on the dataset size, we ran the same amount of operations.

YCSB benchmarks results include runtime, query latency and other metadata in addition to throughput, so a script was created to extract the throughput results from the hundreds of results files using text matching (regular expressions).

6.2 Results

6.2.1 Results for 10 000 rows/operations

This subsection contains results from workload A-E running 10 000 operations on datasets of 10 000 rows.

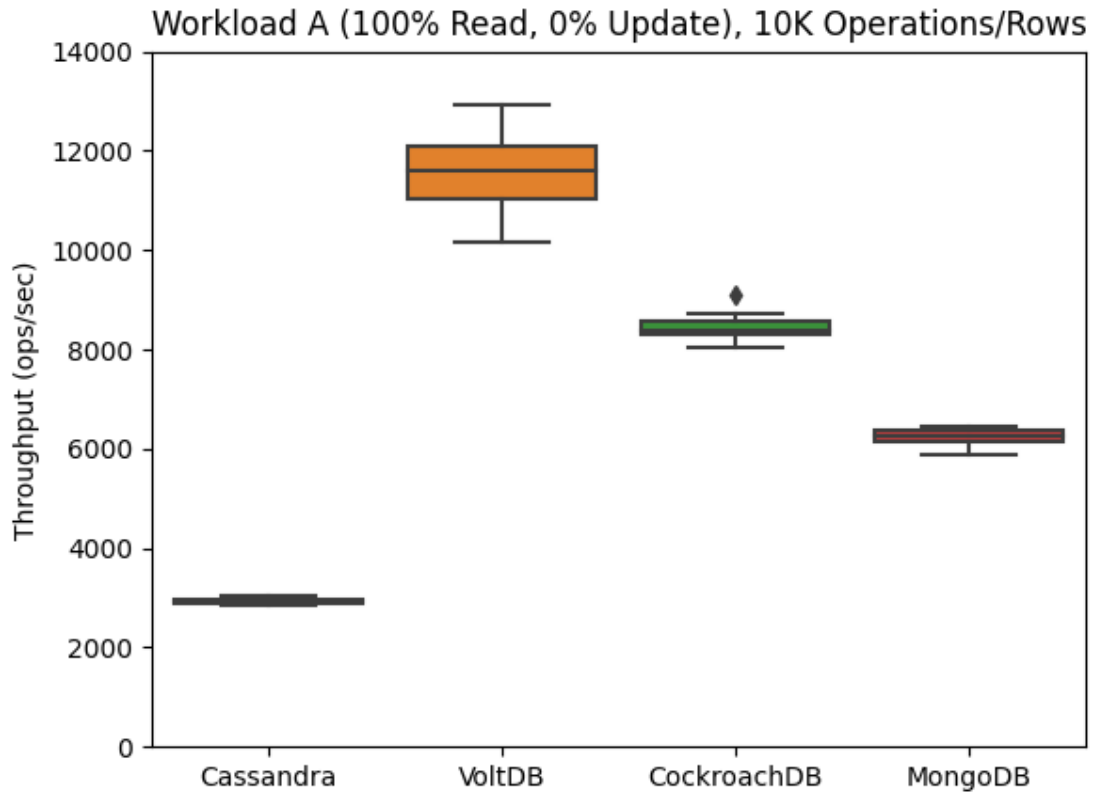

Figure 1 Throughput comparison of CockroachDB, VoltDB, Cassandra, and MongoDB running workload A with 10 000 operations/rows

Figure 1 shows the first result for Workload A with 10 000 rows, where VoltDB has the highest throughput and throughput variation compared to others. CockroachDB comes in at second place with an average throughput of 8447.2 operations/second. MongoDB comes second to last place, and Cassandra ends in the last place with an average throughput of 2918 operations/second.

Tukey tests were performed to measure how significant the throughput difference is and the generated p-values were less than 0.05 for all database combinations. This means with a 95 percent confidence level, there is a significant difference in throughput for CockroachDB, VoltDB, Cassandra, and MongoDB when running workload A on 10 000 rows/operations.

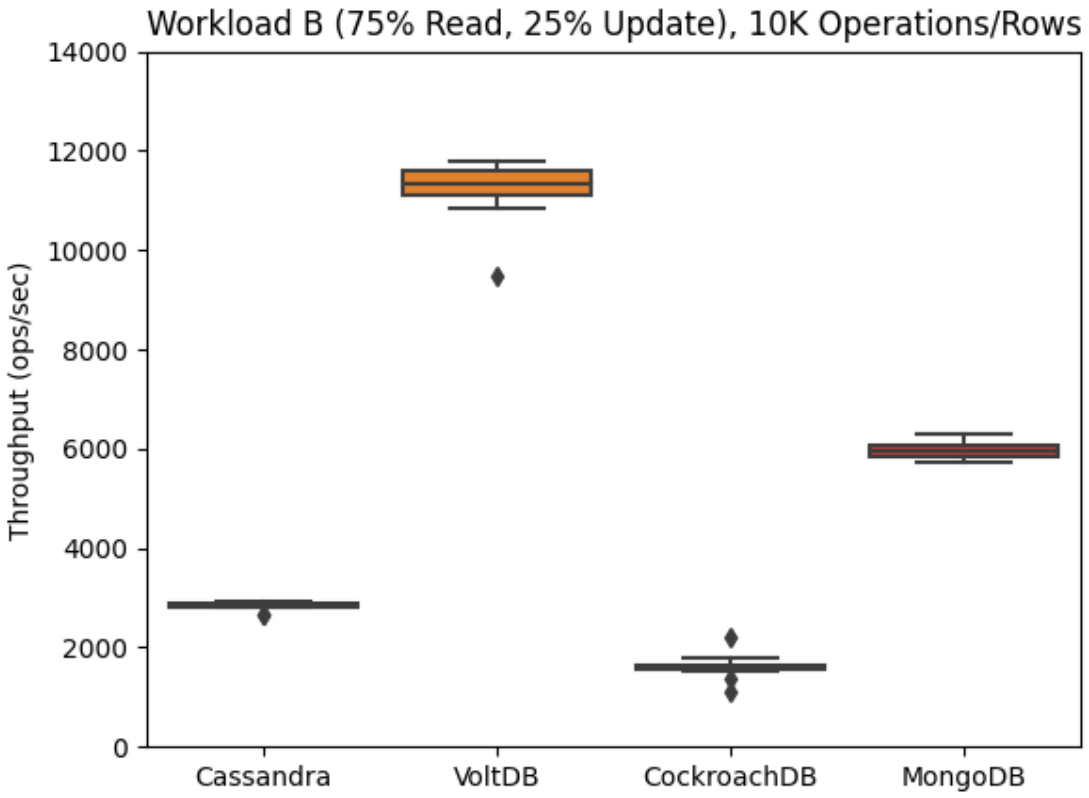

Figure 2 Throughput comparison of CockroachDB, VoltDB, Cassandra, and MongoDB running workload B with 10 000 operations/rows

Compared to workload A, figure 2 shows that the average throughput of CockroachDB takes a downward plunge (8447 to 1608 operations/second) for workload B on 10 000 rows/operations. This results in CockroachDB dropping from having the second-highest throughput to having the lowest. Also, an increased number of throughput outliers is present for CockroachDB in workload B. The average throughput of VoltDB, Cassandra, and MongoDB in figure 2 is very similar to workload A in figure 1. However, VoltDB has a visible improvement in throughput spread, with more values close together and a much smaller minimum-maximum range.

Again, Tukey tests were performed to measure how significant the throughput difference is and the generated p-values were less than 0.05 for all database combinations. This means with a 95 percent confidence level, there is a significant difference in throughput for CockroachDB, VoltDB, Cassandra, and MongoDB when running workload B with 10 000 rows/operations.

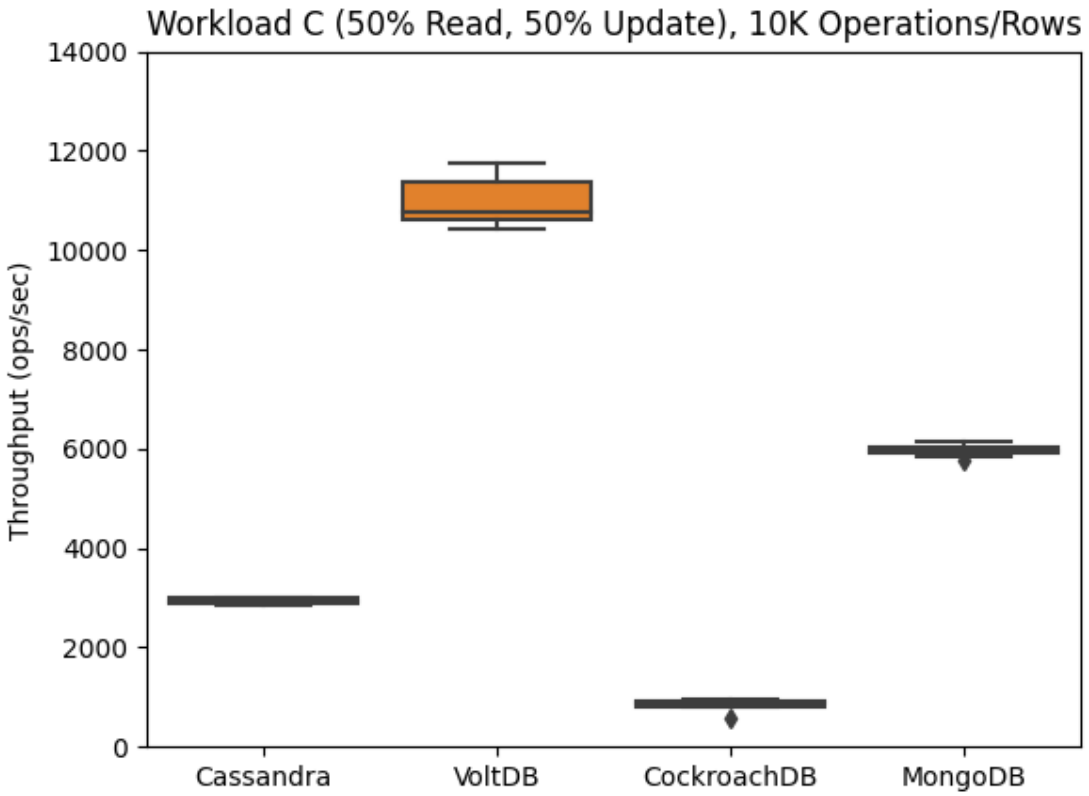

Figure 3 Throughput comparison of CockroachDB, VoltDB, Cassandra, and MongoDB running workload C with 10 000 operations/rows

Figure 3 shows Workload C with 10 000 operations/rows consisting of a 50/50 ratio of read and update queries while VoltDB is at the top with the highest throughput. VoltDB has also a much bigger throughput spread compared to the other DBMSs. CockroachDB is decreasing in throughput with the continuously increased amount of update queries. Both MongoDB and Cassandra throughput has not changed much when it comes to workloads with 10 000 operations/rows.

The Tukey tests generated p-values less than 0.05 for all database combinations. This means with a 95 percent confidence level, there is a significant difference in throughput for CockroachDB, VoltDB, Cassandra, and MongoDB when running workload C with 10 000 rows/operations.

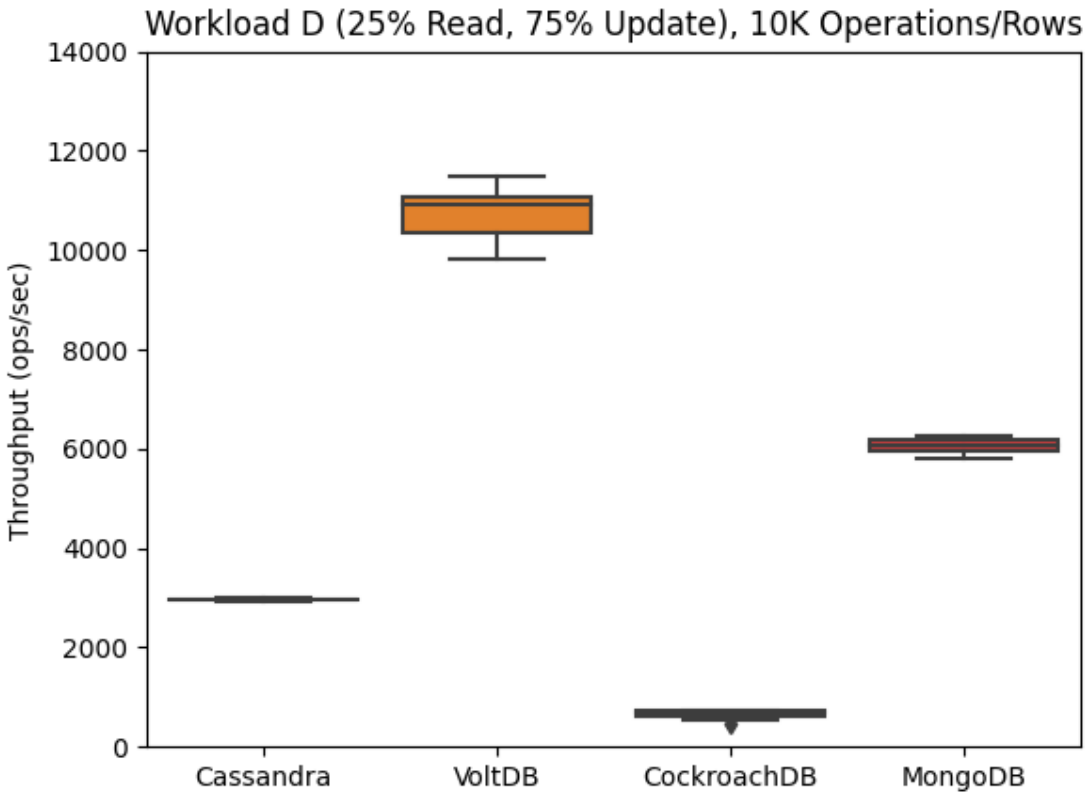

Figure 4 Throughput comparison of CockroachDB, VoltDB, Cassandra, and MongoDB running workload D with 10 000 operations/rows

The average throughput of CockroachDB, VoltDB, Cassandra, and MongoDB remains largely unchanged when going from workload C to D for 10 000 rows/operations (see figure 3 & 4). The most notable change is that the gap between the first quartile and the median is much larger for VoltDB, which indicates the spread of lower throughput values has increased.

The Tukey tests generated p-values less than 0.05 for all database combinations. This means with a 95 percent confidence level, there is a significant difference in throughput between CockroachDB, VoltDB, Cassandra, and MongoDB when running workload D with 10 000 rows/operations.

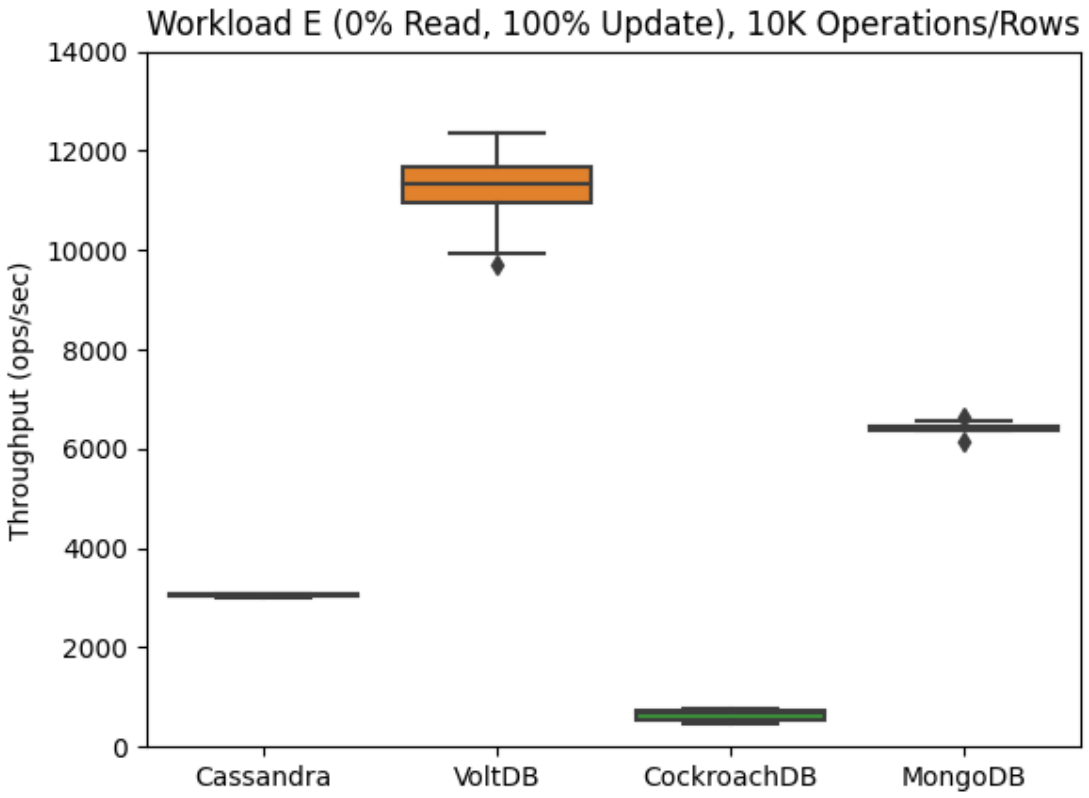

Figure 5 Throughput comparison of CockroachDB, VoltDB, Cassandra, and MongoDB running workload E with 10 000 operations/rows

VoltDB still has the highest throughput for workload E with 10 000 operations/rows (see Figure 5). Not much has changed for the other DBMSs when comparing them to workloads B, C, and D with 10 000 operations/rows.

The Tukey tests generated p-values less than 0.05 for all database combinations. This means with a 95 percent confidence level, there is a significant difference in throughput between CockroachDB, VoltDB, Cassandra, and MongoDB when running workload E with 10 000 rows/operations.

6.2.2 Results for 100 000 rows and operations

This subsection contains results from workload A-E running 100 000 operations on datasets of 100 000 rows.

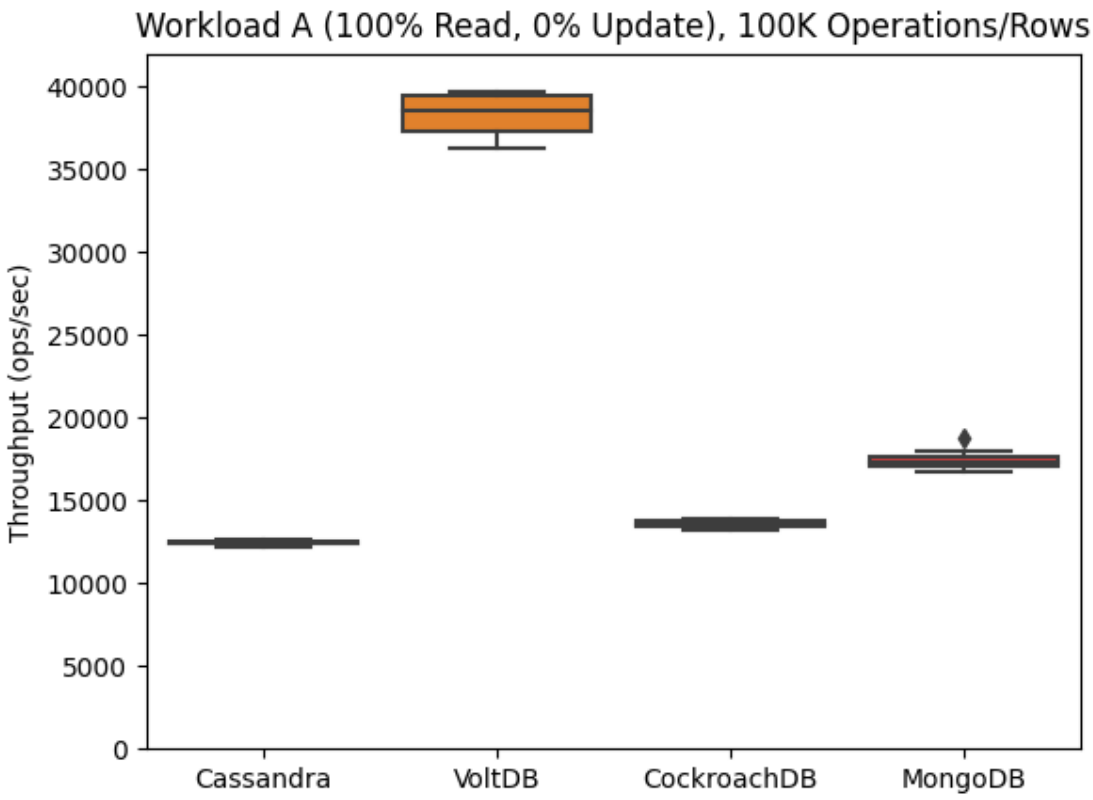

Figure 6 Throughput comparison of CockroachDB, VoltDB, Cassandra, and MongoDB running workload A with 100 000 operations/rows

In Figure 6 VoltDB has still the highest throughput for Workload A with 100000 operations/rows. MongoDB has the second-highest throughput with an average of 17406 operations/second. In third place is CockroachDB with an average throughput of 13568 operations/second. Cassandra falling behind a bit short with an average throughput of 12461 operations/second.

Tukey tests were performed to measure how significant the throughput difference is and the generated p-values were less than 0.05 for all database combinations. This means with a 95 percent confidence level, there is a significant difference in throughput between CockroachDB, VoltDB, Cassandra, and MongoDB when running workload A with 100 000 operations/rows.

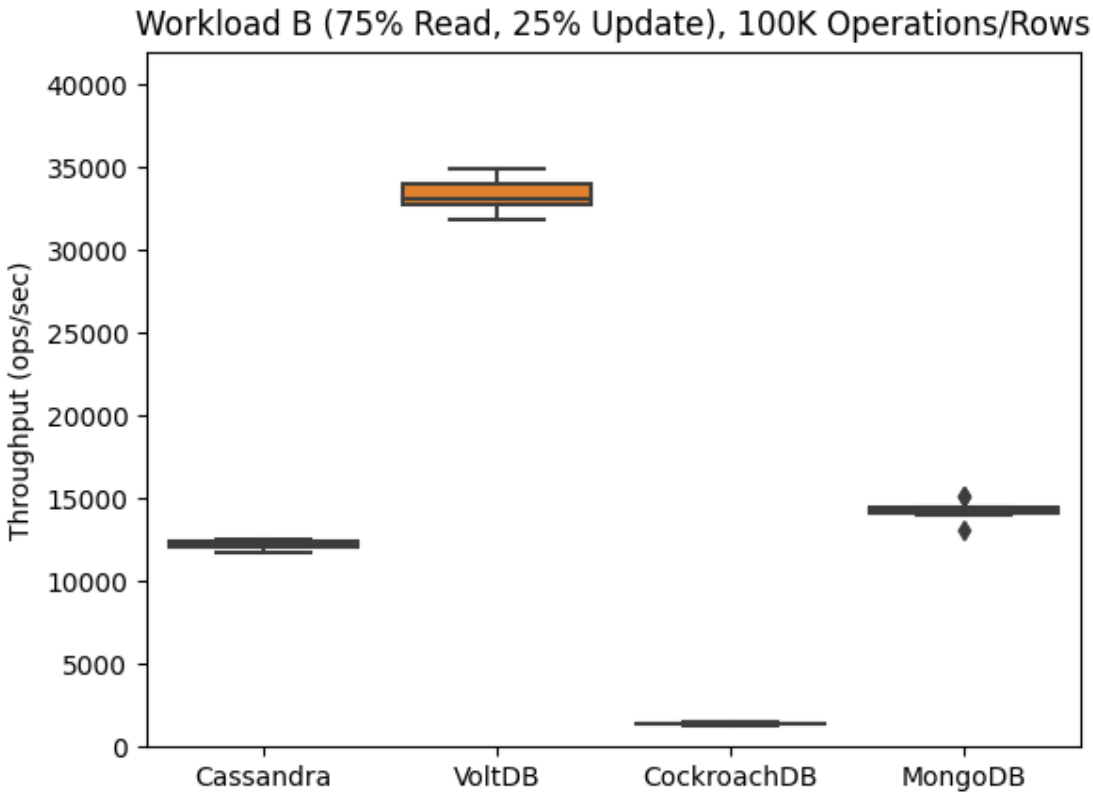

Figure 7 Throughput comparison of CockroachDB, VoltDB, Cassandra, and MongoDB running workload B with 100 000 operations/rows

As seen previously with 10 000 rows/operations, replacing 25 percent of a workload's read queries with update queries results in a massive drop (13569 to 1399 operations/second) in average throughput for CockroachDB for 100 000 rows/operations as well (see figure 7). This results in CockroachDB having the lowest throughput for workload B at 100 000 rows/operations by a significant margin. Both VoltDB and MongoDB show a moderate decrease in average throughput compared to workload A, while Cassandra's throughput remained roughly the same (still lower than MongoDB). VoltDB's throughput spread has shrunk a bit in workload B, but the minimum-maximum throughput range remains largely unchanged.

The Tukey tests generated p-values less than 0.05 for all database combinations. This means with a 95 percent confidence level, there is a significant difference in throughput between CockroachDB, VoltDB, Cassandra, and MongoDB when running workload B with 100 000 rows/operations.

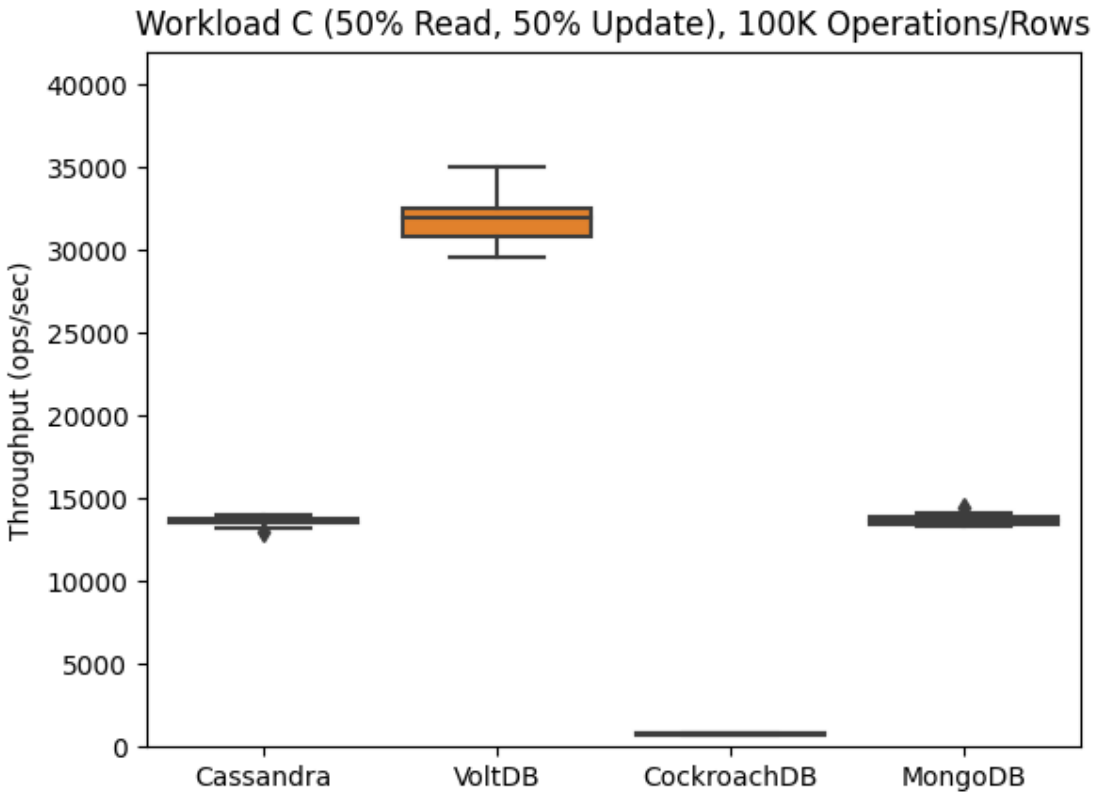

Figure 8 Throughput comparison of CockroachDB, VoltDB, Cassandra, and MongoDB running workload C with 100 000 operations/rows

Figure 8 shows that VoltDB has the highest throughput for workload C with 100 000 rows. MongoDB managed to get a tiny bit higher average throughput compared to Cassandra (13553 vs. 13716 operations/second), getting the second-highest throughput. CockroachDB is still decreasing slightly and has the lowest throughput.

The Tukey tests generated p-values less than 0.05 for all database combinations except for Cassandra and MongoDB, where the p-value is 0.9688. This means there is a 95 percent confidence level that there is no significant difference between Cassandra and MongoDB running workload C on 100 000 rows, but there is a significant difference for all other database combinations.

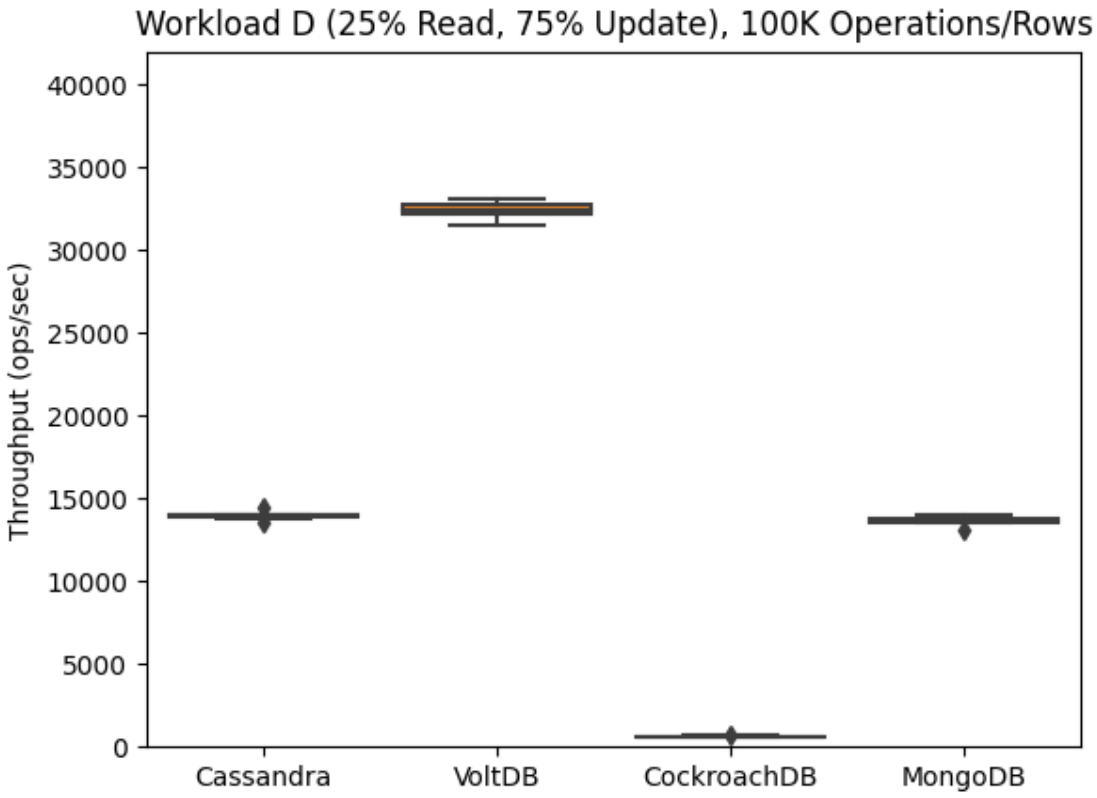

Figure 9 Throughput comparison of CockroachDB, VoltDB, Cassandra, and MongoDB running workload D with 100 000 operations/rows

Workload D at 100 000 rows/operations (see figure 9) shows no significant change in average throughput for all databases, compared to workload C (see figure 8). Most notably, the throughput variation is smaller for Cassandra, MongoDB, and especially for VoltDB, where the minimum and maximum throughput range is much shorter.

The Tukey tests generated p-values less than 0.05 for all database combinations except for Cassandra and MongoDB, where the p-value is 0.175. This means there is a 95 percent confidence level that there is no significant difference between Cassandra and MongoDB running workload D on 100 000 rows, but there is a significant difference for all other database combinations.

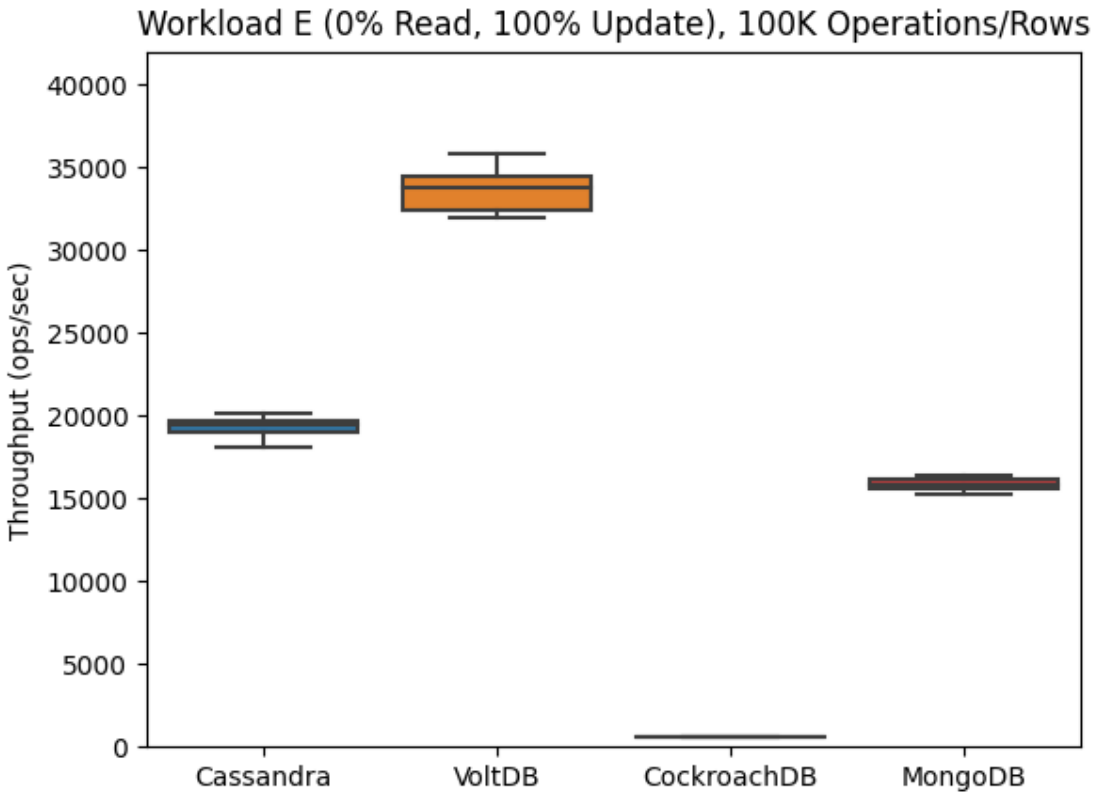

Figure 10 Throughput comparison of CockroachDB, VoltDB, Cassandra, and MongoDB running workload E with 100 000 operations/rows

Figure 10 shows VoltDB with the leading highest throughput. Cassandra has increased drastically in throughput compared to workload D with 100 000 operations/rows, thus taking second place for the highest throughput. CockroachDB has an all-time low average throughput of 577 operations/second and has the lowest throughput.

The Tukey tests generated p-values less than 0.05 for all database combinations. This means with a 95 percent confidence level, there is a significant difference in throughput between CockroachDB, VoltDB, Cassandra, and MongoDB when running workload E with 100 000 rows/operations.

6.2.3 Results for 1 000 000 rows and operations

This subsection contains results from workload A-E running 1 000 000 operations on datasets of 1 000 000 rows.

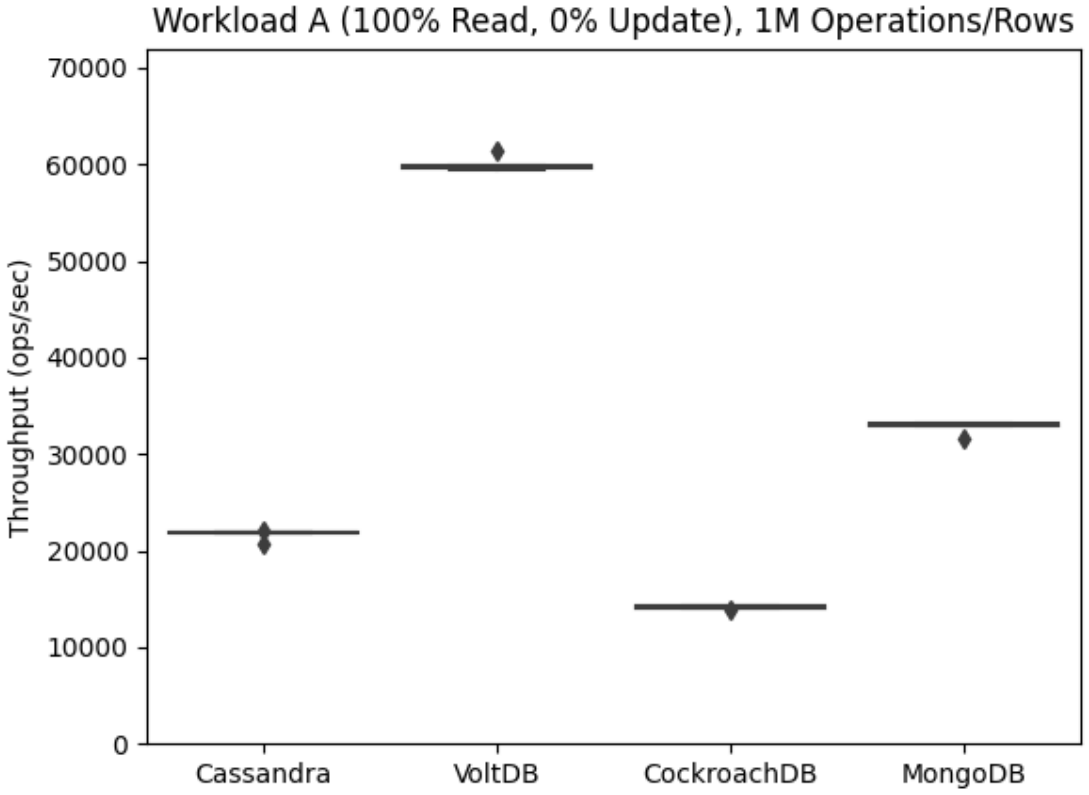

Figure 11 Throughput comparison of CockroachDB, VoltDB, Cassandra, and MongoDB running workload A with 1 000 000 operations/rows

In Figure 11 VoltDB has still the highest throughput for Workload A with 1 000 000 operations/rows. MongoDB has the second-highest throughput with an average of 32 754 operations/second. In third place is Cassandra with an average throughput of 21 638 operations/second. CockroachDB is in last place with an average throughput of 14 146 operations/second.

Tukey tests were performed to measure how significant the throughput difference is and the generated p-values were less than 0.05 for all database combinations. This means with a 95 percent confidence level, there is a significant difference in throughput between CockroachDB, VoltDB, Cassandra, and MongoDB when running workload A with 1 000 000 rows/operations.

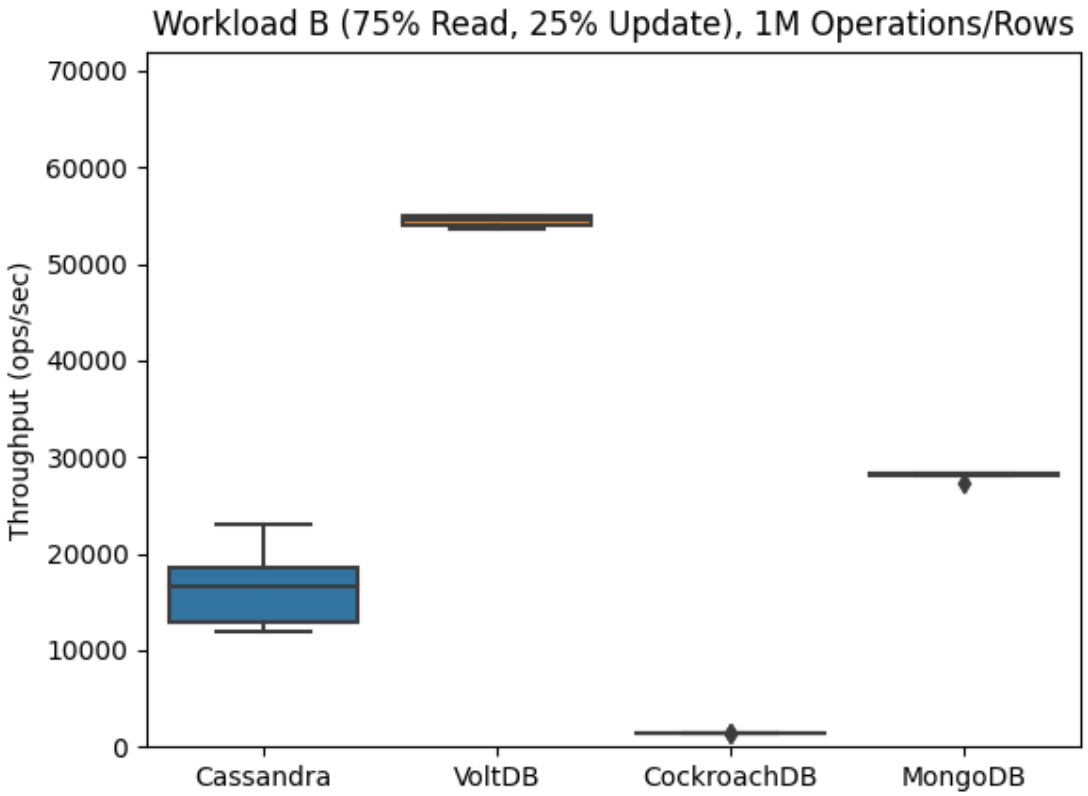

Figure 12 Throughput comparison of CockroachDB, VoltDB, Cassandra, and MongoDB running workload B with 1 000 000 operations/rows

Similarly to what happened with 10 000 and 100 000 rows/operations, there is a clear drop in throughput for CockroachDB (14147 to 1352 operations/second) when going from workload A to B for 1 000 000 rows as well (see figure 12). VoltDB (which still has the highest throughput), MongoDB, and Cassandra show a small to moderate decrease in average throughput compared to workload A, while Cassandra's throughput spread has grown considerably. This indicates that Cassandra is the worst choice of the four DBMS regarding throughput stability when running workload B at 1 000 000 rows/operations.

The Tukey tests generated p-values less than 0.05 for all database combinations. This means with a 95 percent confidence level, there is a significant difference in throughput between CockroachDB, VoltDB, Cassandra, and MongoDB when running workload B with 1 000 000 rows/operations.

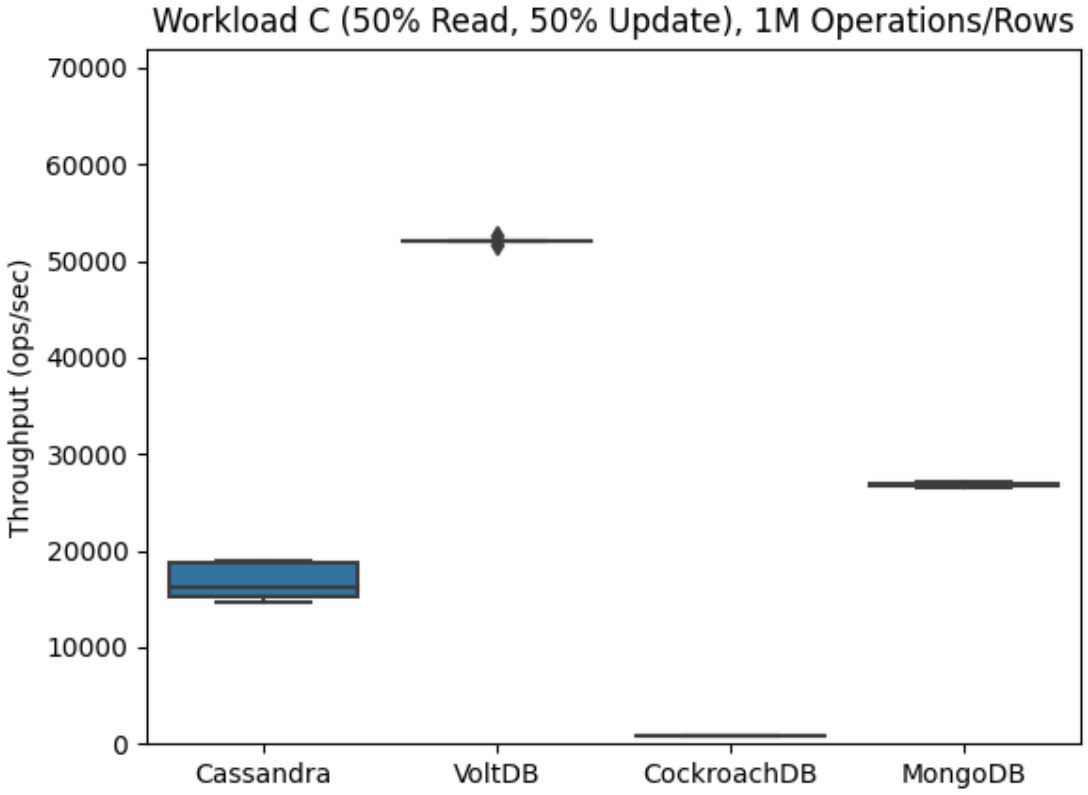

Figure 13 Throughput comparison of CockroachDB, VoltDB, Cassandra, and MongoDB running workload C with 1 000 000 operations/rows

Figure 13 shows the final result for Workload C with 1 000 000 operations/rows, VoltDB still has the highest throughput. There are no big changes compared to Workload B (see Figure 12) except for Cassandra having less variance in overall throughput.

The Tukey tests generated p-values less than 0.05 for all database combinations. This means with a 95 percent confidence level, there is a significant difference in throughput between CockroachDB, VoltDB, Cassandra, and MongoDB when running workload C with 1 000 000 rows/operations.

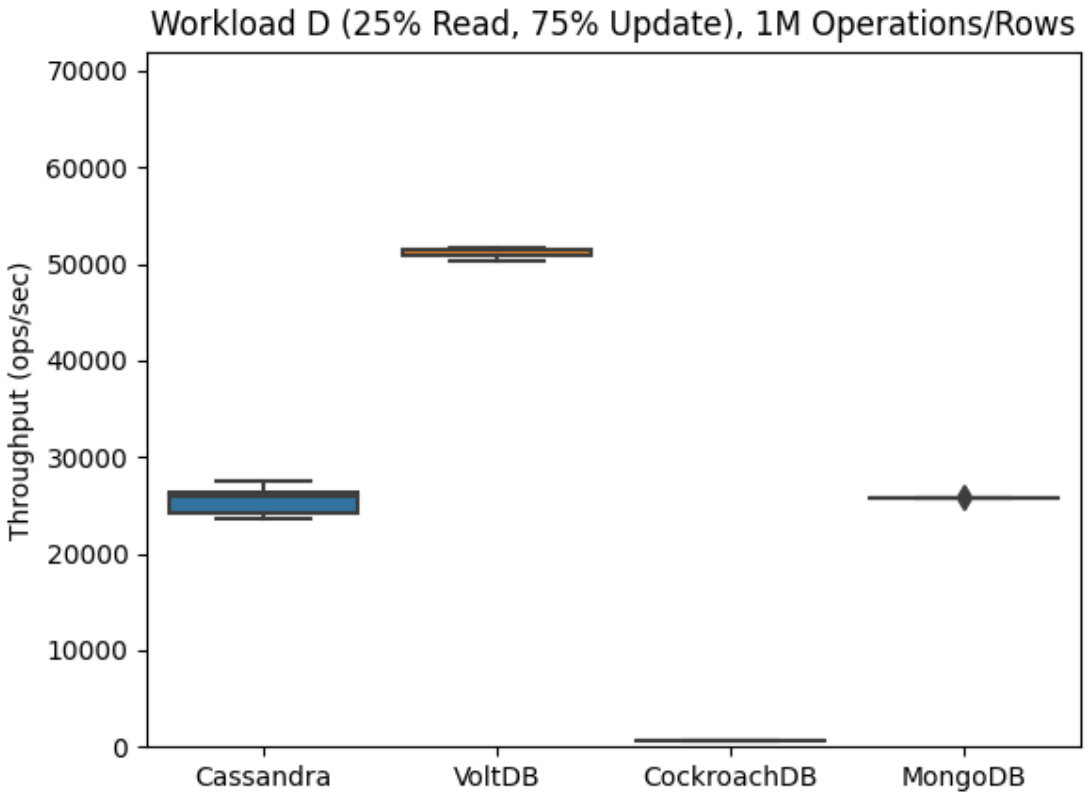

Figure 14 Throughput comparison of CockroachDB, VoltDB, Cassandra, and MongoDB running workload D with 1 000 000 operations/rows

With workload D at 1 000 000 rows/operations (see figure 14), Cassandra's average throughput has grown considerably compared to workload C (see figure 13), even catching up to and matching MongoDB's throughput. VoltDB still has the highest average throughput in workload D, but with a slight increase in spread. CockroachDB's throughput remains basically unchanged and is still way lower than the other three DBMS.

The Tukey tests generated p-values less than 0.05 for all database combinations except for Cassandra and MongoDB, where the p-value is 0.952. This means there is a 95 percent confidence level that there is no significant difference between Cassandra and MongoDB running workload D on 1 000 000 rows, but there is a significant difference for all other database combinations.

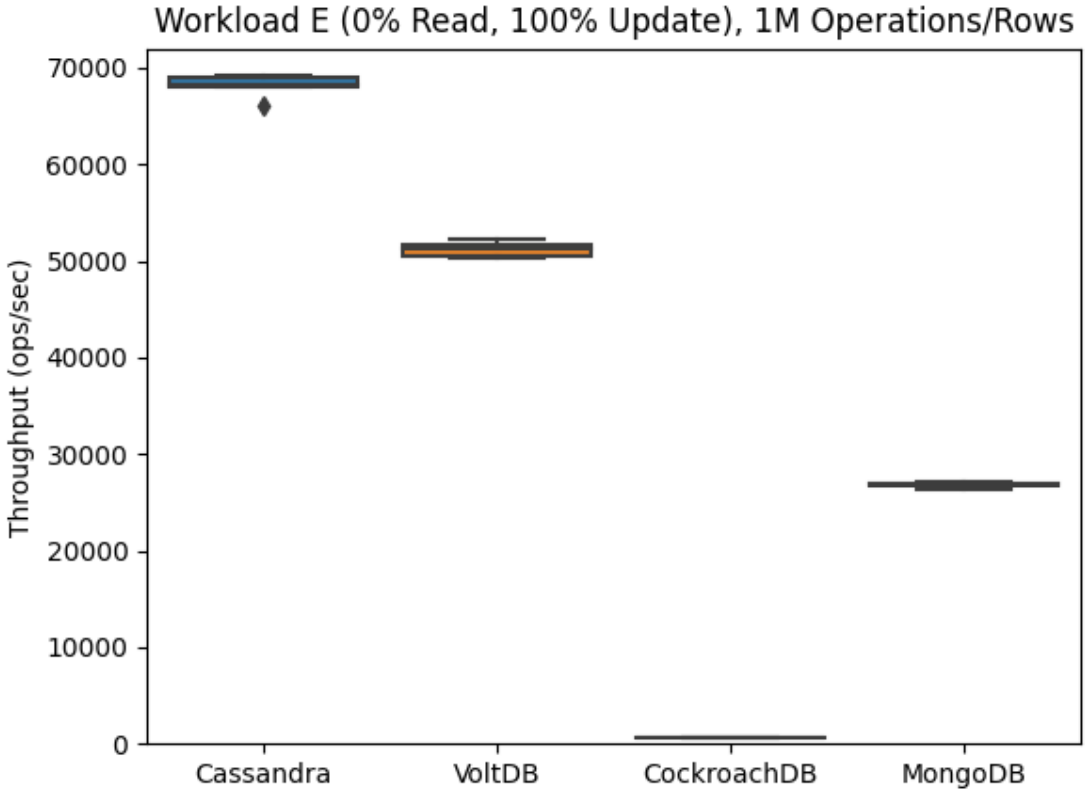

Figure 15 Throughput comparison of CockroachDB, VoltDB, Cassandra, and MongoDB running workload E with 1 000 000 operations/rows

Figure 15 shows the final results for workload E containing 1 000 000 operations/rows, for the first time VoltDB does not have the highest throughput. Cassandra has the highest throughput instead with a mean of 68103 operations/second. Cassandra scaled significantly higher with 100% update queries compared to the other DBMSs. The difference here between the highest (Cassandra) and lowest throughput (CockroachDB) DBMS in terms of mean is 68103 vs. 617 operations/second.

The Tukey tests generated p-values less than 0.05 for all database combinations. This means with a 95 percent confidence level, there is a significant difference in throughput between CockroachDB, VoltDB, Cassandra, and MongoDB when running workload E with 1 000 000 rows/operations.

6.2.4 Throughput change by workload

The line charts in this subsection will help to show the difference in throughput when the workload transitions from being read-heavy to update-heavy.

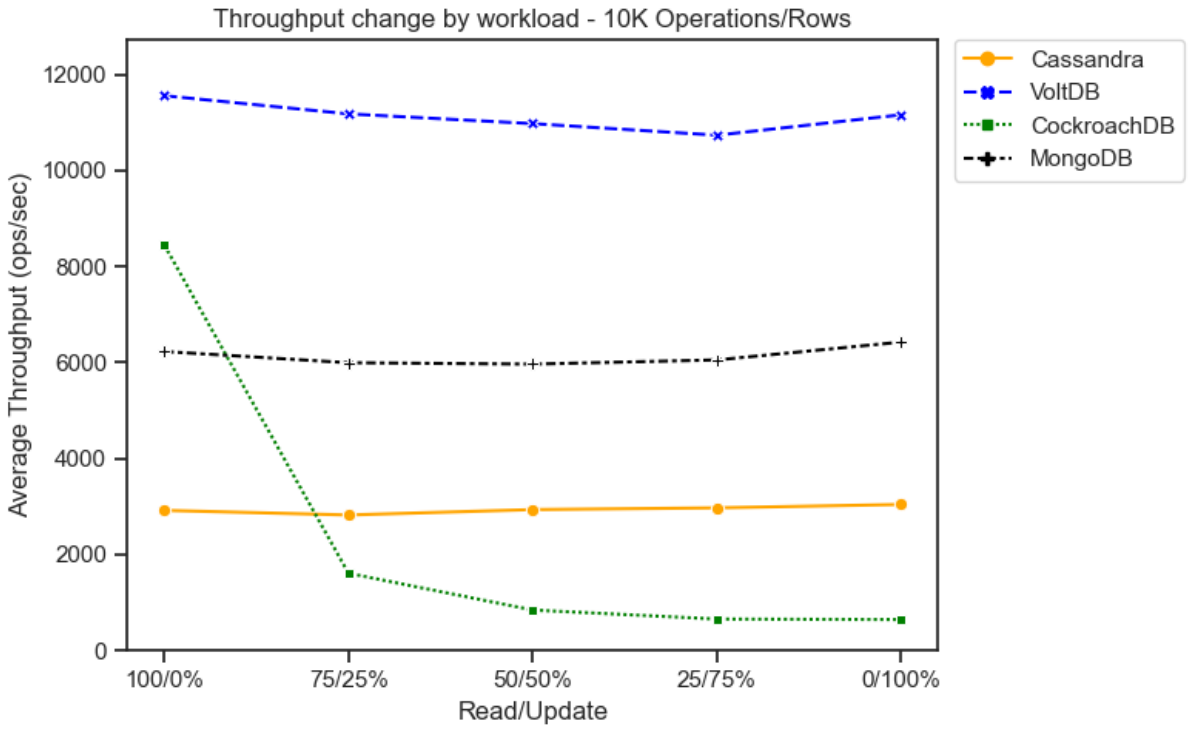

Figure 16 Average throughput across all workloads with 10 000 rows/operations

Figure 16 shows the average throughput for workloads with 10 000 dataset size, where except for CockroachDB, the throughput does not visually change very much when becoming more update-heavy, the throughput remains constant across workloads. CockroachDB has high throughput for 100 percent read workloads, but plummets with any inclusion of update queries and the throughput steadily gets even worse.

ANOVA tests were performed for each database to measure if there was a significant change in throughput when the workload became more update-heavy. The generated p-values were less than 0.05 for Cassandra, CockroachDB, and MongoDB, but 0.119 for VoltDB. This means with a 95 percent confidence level, the read/update query composition of workloads does have a significant impact on throughput for Cassandra, CockroachDB, and MongoDB on 10 000 rows and operations.

How much the workload composition affected throughput was also investigated, and this effect size was calculated using the Eta Squared algorithm. Simplified, this algorithm generates a value between 0 and 1, which is the proportion of how much the variance can be attributed to an effect, like changing the workload composition.

VoltDB had an Eta squared value ($\eta 2$) of 0.1473, which indicates a large effect size (>0.14) even though the workload composition's effect on throughput was statistically insignificant. However, this effect size is minuscule compared to the other databases, where the $\eta 2$-values

were: 0.5712 for MongodB, 0.6437 for Cassandra and 0.9960 for CockroachDB. This means CockroachDB's throughput is by far the most affected by a change in workload composition.

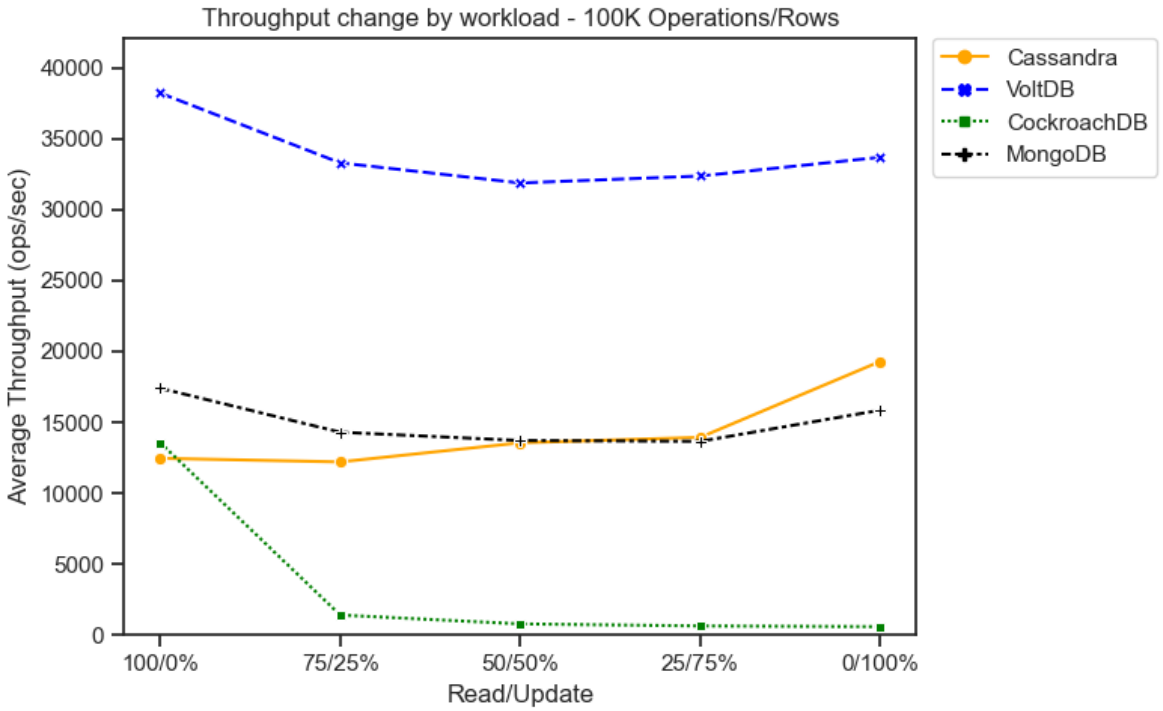

Figure 17 Average throughput across all workloads with 100 000 rows/operations

For 100 000 rows/operations (see figure 17), the average throughput is visually less constant across workloads than 10 000 rows/operations. Both VoltDB and MongoDB show a throughput deficit centered around 50 percent read and 50 percent update. Cassandra's throughput is increased with more update-heavy workloads and shows a moderate spike at 100 percent update queries. As with 10 000 rows/operations, CockroachDB's throughput plummets at 75 percent read and 25 percent update and never recovers.

Again, ANOVA tests were performed to measure if the workload's composition had a significant impact on throughput. The generated p-values were less than 0.05 for all four databases. This means with a 95 percent confidence level, the read/update query composition of workloads does have a significant impact on throughput for Cassandra, CockroachDB and MongoDB on 100 000 rows and operations.

The effect size was also measured, where the η2-values were: 0.7960 for VoltDB, 0.9180 for MongoDB, 0.9830 for Cassandra and 0.9996 for CockroachDB. This is a sizable increase for all databases compared to 10 000 rows/operations, meaning the throughput is generally more affected by the workload composition.

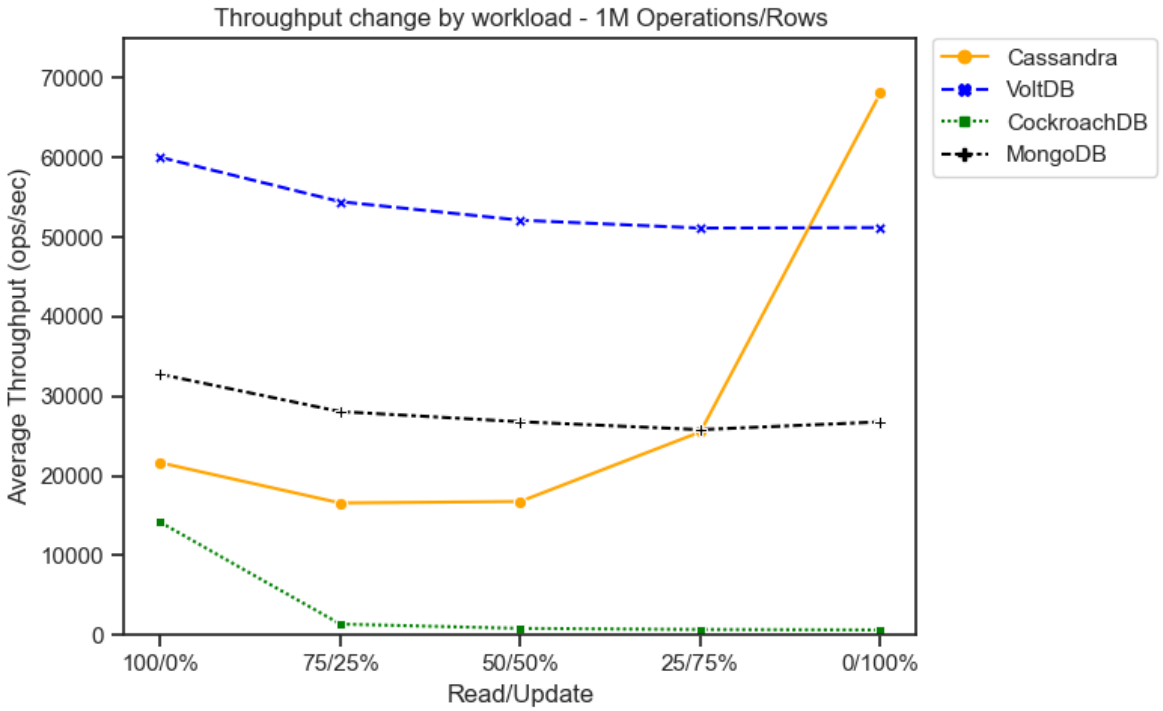

Figure 18 Average throughput across all workloads with 1 000 000 rows/operations

Figure 18 shows the average throughput for all workloads with 1 000 000 rows and operations. VoltDB is the overall winner and has the highest average throughput for all workloads except for Workload E (100% Update) where Cassandra spikes, taking the highest average throughput for that workload. Cassandra is performing much better the more update-heavy it gets. MongoDB has had a constant throughput throughout all the different sets of rows/operations. CockroachDB still plummets down in average throughput with the increase in update operations.

ANOVA tests were performed for the last time to measure if the workload's composition had a significant impact on throughput. The generated p-values were less than 0.05 for all four databases. This means with a 95 percent confidence level, the read/update query composition of workloads does have a significant impact on throughput for Cassandra, CockroachDB and MongoDB on 1 000 000 rows and operations.

The effect size was also measured, where the η2-values were: 0.9707 for VoltDB, 0.9811 for MongoDB, 0.9880 for Cassandra and 0.9998 for CockroachDB. All of the DBMS had slight increase in effect size and have the same standings as 100 000 rows/operations, meaning that the throughput is still affected by a change in a workload composition.

6.3 Conclusions & Analysis

In this section the conclusion and analysis of the results are discussed.

6.3.1 Hypothesis testing

Each of the two research questions are answered by either accepting or rejecting their corresponding null hypothesis.

The first hypothesis (RQ1):

"There is no significant difference in terms of average data throughput between MongoDB, Cassandra, CockroachDB, and VoltDB."

The results show a visible difference in throughput between Cassandra, CockroachDB, MongoDB, and VoltDB for most workloads across different dataset sizes and the number of operations. Running Tukey tests with 95 percent confidence level confirmed there is a significant difference in 51 out of 54 database combinations across all read/update workloads and sizes.

Only Cassandra and MongoDB had similar enough throughput to not be considered significantly different, at specifically workload C and D with 100 000 rows/operations and workload D with 1 000 000 rows/operations.

As the throughput is significantly different in a vast majority of cases, the first null hypothesis is rejected, and there does exist a significant difference in throughput between Cassandra, CockroachDB, MongoDB and VoltDB.

The second hypothesis (RQ2):

"The throughput of MongoDB, Cassandra, CockroachDB, and VoltDB are not significantly affected by the read/update query composition of workloads."

Most databases showed a visual throughput change when increasing the proportion of update queries in the workload. This change can be especially seen in CockroachDB's substantial drop in throughput, and Cassandra's throughput spike with update-heavy workloads.

The ANOVA tests for each dataset size and number of operations showed that the workload composition had a significant effect on throughput for all databases except for VoltDB at 10 000 rows/operations. This means the workload composition had an effect on throughput in 11 out of 12 cases.

As the throughput is significantly affected in a vast majority of cases, the second null hypothesis is also rejected and the workload composition does have a significant effect on throughput.

6.3.2 Answering the research questions

RQ1:

"Is there a significant difference between Cassandra, CockroachDB, MongoDB, and VoltDB in terms of average data throughput? "

As the first null hypothesis was rejected, the results show that Cassandra, CockroachDB, MongoDB, and VoltDB do have a significant difference in average throughput.

The results also show that VoltDB has the highest throughput for nearly all workloads and dataset/operation sizes. The only DBMS with a recorded higher throughput was Cassandra, which had extremely high throughput with workloads consisting of 100 percent update queries (for larger dataset/number of operation sizes). MongoDB's throughput is consistently decent, outperforming Cassandra at most non-update-heavy workloads, but is still far behind VoltDB. CockroachDB had comparatively terrible throughput throughout most of the tests, but is comparable and sometimes even better than Cassandra and MongoDB with 100 percent read workloads.

RQ2:

"Does the read/update query composition of workloads have a significant effect on average throughput for Cassandra, CockroachDB, MongoDB and VoltDB?"

As the second null hypothesis was also rejected, the results show that the read/update composition of workloads does have a significant effect on throughput for Cassandra, CockroachDB, MongoDB and VoltDB.

The results also show that VoltDB and MongoDB are about equally the best for throughput consistency across different read/update workloads, but VoltDB, with its high average throughput, is less affected by changes in workload composition.

Cassandra, which has fairly consistent throughput at smaller datasets and lower number of operations, shows a massive increase in throughput when running workloads with higher proportions of update queries. This increase is especially prominent with larger datasets and large number of operations.

CockroachDB's throughput is incredibly low on any workload with an update component, but it is pretty high with 100 percent read workloads, especially with smaller datasets and a lower number of operations.

7 Discussion

This section summarizes how each objective was achieved, discussions regarding the results, how this study can be of use, related ethics, and potential threats to validity. Some of these threats can be addressed by performing a different study, these (and other potentially interesting) studies are discussed in the final subsection *Future work*.

7.1 General discussion

To reach our aim of this study, certain objectives had to be completed in a specific order for us to do that and they are as follows:

1. Identify which DBMS and benchmarking tool to include in the study [Both]
2. Setup environment and compatible workloads to compare the different DBMS [Individually]
3. Compare and contrast the selected DBMS concerning average data throughput and different workloads [Individually]
4. Analyze and evaluate results to answer our research questions [Both]

We solved the first objective by doing prior research about different NoSQL and NewSQL DBMS. We then ended up selecting two NoSQL DBMS (Cassandra and MongoDB) and two NewSQL DBMS (VoltDB and CockroachDB). Scylla was selected at first instead of MongoDB but was replaced due to some issues with the benchmarking tool running the Scylla driver. Cassandra and MongoDB have different data model types, the former being column-oriented while the latter being document-oriented. As for the benchmarking tool that was used for the experiment, we chose Yahoo! Cloud Serving Benchmark to do the performance testing on all DBMS.

For the second objective, we set up four environments with the same hardware on the NSA lab computers at University of Skövde and selected appropriate workloads accordingly that worked on all of the DBMS.

For the third objective, we gathered all the necessary data from text files that contained the overall throughput and workload compositions for the experiment. The text files were generated with help from the benchmarking tool whenever we ran a single test. We then calculated the average throughput for every DBMS and workload composition for the comparison.

For the fourth and final objective, we analyzed and evaluated the results that we decided to show through boxplot diagrams and line graphs. Once that was concluded we could then come to a conclusion for our study and answer our research questions and hypotheses. Tukey tests were performed to see if there was a significant difference in throughput between all DBMS, as it was one of the research questions for this study. ANOVA tests were also conducted to see if there was a significant effect on throughput when changing the workload composition. Then finally Eta Squared Algorithm was used to determine how big of an effect it had on throughput on a scale from 0 to 1.

The results from this study show that VoltDB is the best DBMS for throughput, as it has the highest throughput across a vast majority of read/update workloads. VoltDB is not just better due to consistency. Its throughput is also much higher than the other DBMS, nearly doubling the average throughput of the next best DBMS in most workloads (see figures 16, 17 & 18). The VoltDB results also coincide with a study by Schreiner et al. (2019), which also showed that VoltDB has a much higher throughput than CockroachDB with an unspecified workload.

Schreiner et al. (2019) study also had CockroachDB show inferior throughput results compared to all DBMS included in their study. They suggest it is due to CockroachDB running on slower disks instead of memory, which we also believe is the cause of our low throughput results for CockroachDB. The disk's effect on throughput can also be seen with CockroachDB having decent read, but subpar update results, which coincides with disks typically having better read speeds compared to write speeds. MongoDB, which also is a disk-based DBMS, does not have nearly as bad throughput results as CockroachDB. We suspect this is due to MongoDB extensively caching recent data in memory, which CockroachDB does not.

CockroachDB's poor results can also be seen in Kaur and Sachdeva's (2017) study, where the write and update latency is considerably worse than for VoltDB. Though, the difference in read performance is much smaller compared to our results, with CockroachDB's latency being less than ten percent higher compared to VoltDB. Interestingly, our results contrast when it comes to execution time. CockroachDB had a much lower execution time compared to VoltDB, which is the opposite of our throughput results, where VoltDB far outperforms CockroachDB. We suspect this might also be due to a difference in disk speeds, but it is hard to determine when they do not disclose what type of disk was used in their experiments.

We speculate that using a faster SSD instead of an HDD should show higher throughput results for CockroachDB and possibly other disk-based DBMS, but identifying exactly how much storage speed affects disk-based DBMS throughput requires further research.

Cassandra's increase in throughput for larger datasets can be seen in Cui and Chen (2021). The overall throughput is slightly increased for both 100% read, and 50/50 read and update workloads. They tested five different workloads with 3 000 000 rows/operations and 10 000 000 rows/operations, focusing on throughput and some other factors. Their results showed an overall throughput of 25 343 operations/second for Workload A (100% Read) and 26 863 operations/second for Workload C (50/50% Read and Update). There is approximately 17% increase in throughput for Workload A and a huge 60% increase for Workload C when comparing their 3 000 000 rows/operations to our 1 000 000 rows/operations results.

The results also show an unusual spike in average throughput for Cassandra on workload E with 1 000 000 rows/operations. Going from an average throughput of 19 303 operations/second with 100 000 rows/operations to 68 103 operations/second with 1 000 000 rows/operations.

Abramova and Bernardino (2013) study comparing the execution time (which is inversely related to throughput) of Casssandra and MongoDB also had incredible results for Cassandra at 100 percent update workloads. However, the execution time didn't have a massive drop at 100 percent update, instead the execution time steadily shrunk with each increase in update

composition. This is contrary to our results, where the throughput decreased before spiking at 100 percent update queries. We believe the reason for our initial loss in throughput is related to the number of client threads. Abramova and Bernardino (2013) only used one client thread while we used 32. Araujo et. al (2021) results indicate that lower client thread counts show an increase in throughput, whilst higher thread counts show a throughput decrease when going from a 100 percent read workload to 50 percent read, 50 percent update workload.

The results showed that MongoDB and VoltDB have the best throughput consistency across workloads, but also that they behave very similarly to each other (except that VoltDB has a much higher average throughput). As to why they share near-identical throughput patterns across read/update workloads could not be determined in this study, as it would require a more in-depth literature study of how specifically MongoDB and VoltDB work or interviews with their respective developers, which is outside this study's scope.

The results also show that the workload composition had a much smaller impact on throughput for 10 000 rows/operations compared to the larger dataset sizes and higher number of operations. But as both size variables were changed simultaneously, it is impossible to know whether or not it is the dataset size or the number of operations that causes the workload composition to have a larger impact. We believe the lower number of operations is what causes the throughput to be more constant across workloads, as the strain on the DBMS might not be enough to cause any bottlenecks.

To uncover exactly what causes the workload composition to have a lesser effect on throughput at smaller datasets and lower number of operations, further research is required where the size variables are changed separately.

Our study differs from Jansson, Vukosavljevic, and Catovic (2021) study as we measured the throughput of predefined workloads instead of query latency. We also included NewSQL databases in our comparison while they exclusively compared NoSQL databases, so the only database in common between our studies is MongoDB. Our study also investigates if different workload compositions significantly affect throughput.

7.2 Contribution

Our study is aimed toward particular consumers such as developers and even big IT-companies handling large amounts of data that either want to use these types of DBMS or are currently using them at this moment.

Developers and database managers can use this study's results to choose a DBMS most suitable for their application's workload, which could make more efficient use of the system's hardware and lower the total amount of hardware components needed. This could potentially lower the database system's complexity and initial financial cost. However, this study does not take CPU usage, memory usage, and power draw into account when comparing DBMS throughput, so its relevance to hardware efficiency is mostly speculative. Still, since all DBMS had access to the same hardware setup, basing efficiency on throughput results is not entirely unfounded.

If we could make more efficient use of database hardware, the need for energy costly cooling might decrease, making the system cheaper to run and more environmentally friendly. Increased hardware efficiency might also lower the total amount hardware needed to run the system, which would decrease the eventual amount of non-recyclable E-waste that pollutes the atmosphere and groundwater with toxic materials.

7.3 Ethics

As this study is centered around an experiment that benchmarks the throughput of different types of DBMS software, it does not directly involve any human participants. Also, all DBMS software involved in this study are open source, meaning they can be downloaded and used for free. They are available in GitHub repositories or as download links on their respective websites. The benchmarking tool YCSB is also open source and can be found in a GitHub repository as well.

A PowerShell script was created to retrieve throughput values, database names, workload ids, and dataset/operation sizes from all benchmark result files and convert them into a single file containing rows and columns. The source code for this script is not included in this paper as it mainly iterates over files and extracts values based on regular expressions, which has plenty of documentation already.

Similarly, the python scripts created to visualize results using boxplots and line graphs are also not included in the paper, for similar reasons to the PowerShell script.

No personal information was used for the datasets when benchmarking the different DBMS. The datasets were randomly generated by the benchmarking tool, and the data itself consisted of different letters, numbers, and symbols.

Everything that is described within this study is properly referenced or is based on our results.

7.4 Threats to validity

The most prominent threat to validity in this study is the internal threat caused by changing two independent variables simultaneously, namely the dataset size and the number of operations. This makes it impossible to know precisely the effect each individual variable has on the throughput. To mitigate the risk of faulty comparisons and reaching incorrect conclusions, all attempts to analyze and compare throughput scaling has been removed.

One internal validity threat is the benchmarking tool YCSB itself, as its limitations to ensure fairness excludes some performance-enhancing features unique to certain NoSQL databases (Reniers et al., 2017). As this study includes two NoSQL DBMS, this could lead to an unfair comparison where DBMS are not performing with their full capabilities. Unable to identify if Cassandra and MongoDB possess similar features (and whether or not they would be utilized during testing), YCSB remained the number one choice for its baseline fairness across NoSQL and NewSQL databases.

Randomly generated datasets that were briefly mentioned in the Ethics section are another potential threat to validity. Since the data is randomly generated every time a user performs a load (insert) operation, other researchers would not be able to reproduce the same experiment. To mitigate this issue, we ran the benchmarking tests several times to minimize the variance of throughput.

There is one external validity threat concerning this study and it is the selection of the benchmarking tool. The results may vary based on the selected benchmarking tool, to counter this we have selected a well-known benchmarking tool YCSB that has been used in previous studies for similar experiments.

Choosing which DBMS to include in this study based on compatibility with YCSB could be considered an external validity threat, as only a subset of all DBMS is compatible with it. Selecting DBMS this way should not be an issue as YCSB's list of compatible DBMS is extensive and includes many different types of DBMS. This list can be viewed at YCSB's GitHub repository. Note that it does not include CockroachDB and many other RDBMS and NewSQL DBMS, which are compatible via the Java Database Connectivity API (JDBC).

Another external validity threat is the generalizability of using HDDs instead of SSDs in benchmarking tests. It is common for cloud storage solutions to use an SSD-HDD hybrid approach, where the faster SSDs handle most of the querying. Changes are later flushed to HDDs, which mainly serve as large and relatively cheap storage (Wang et al., 2022). This threat is especially impactful considering only some of the databases in this study use on-disk storage, where performance is suspected to be directly connected to the speed of the storage device.

Unfortunately, the University of Skövde could not provide SSDs for thesis projects, only HDDs. With limited funds and both researchers living in separate countries, constructing our own benchmarking setup would only cause more problems than it would solve. Therefore, HDDs were used as it was the only feasible and available solution, which could be representative outside of cloud storage solutions.

An additional external validity threat is the host machine's hardware. Some hardware parts can favor one of the databases in terms of performance more than the other, thus giving us unfair results. The most common hardware parts affecting the database's performance are the CPU and RAM. The fact that the benchmark tests were run on desktop CPUs and RAM is also an external validity threat, as most data centers' DBMS run on hardware (CPU, RAM) specifically made for a server environment.

As with storage, the amount of available CPUs and RAM we could use in the experiment was very limited. The networks and administrators lab at the University of Skövde was chosen for its easy access to multiple machines with the same model of CPU and RAM, which promotes baseline fairness for the benchmark tests.

7.5 Future work

One easy way to proceed with further research would be to include more DBMS, as this study only involves a small subset of all available DBMS. Prioritizing newer DBMS could be of

interest, as they generally have less previous research. Likewise, DBMS with recent major updates would also need more research, as newer versions can have changes in performance.

Including more workloads could be difficult in comparison, as YCSB only allows read, update, scan and insert queries. Comparing more complex workloads would then require avoiding using YCSB as a benchmarking tool. Instead of (or in addition to) manually benchmarking complex workload performance, a comparison of each DBMS underlying workload management system could be performed. The workload management system optimizes the use of system resources by monitoring and controlling the execution of requests (Zhang, Martin, Powley & Chen, 2017) and could be studied to give a possible insight into why specific (complex) workloads perform in a certain way.

As disk speed is suspected to impact CockroachDB's throughput, to which extent disk speed affects throughput also needs to be investigated. We propose a study that benchmarks CockroachDB's throughput using three different storage devices with different speeds, such as HDDs, SATA SSDs and NVMe SSDs. Including other disk-based DBMS is also encouraged, as it could reveal if (or how much) they are affected by disk speed. This proposed study would hopefully pinpoint the effect of disk speed on the throughput of disk-based DBMS, and propose an optimum disk speed for each DBMS included.

This study does not limit/measure CPU usage or memory usage and does not measure power draw during benchmark tests, which makes it difficult to take hardware efficiency into account when comparing throughput. Allowing DBMS unfettered access to hardware made throughput results more generalizable (as DBMS functioned to their fullest potential), but made the results a little less useful in the process. To remedy this, a secondary benchmark study that limits the amount of system resources DBMS can access is needed.

This proposed study's throughput results would hopefully be able to show which DBMS has the highest throughput to CPU/memory/power usage ratio. If this study also used different workloads when comparing throughput, patterns might be found where certain workloads use more of a specific system resource, which could be of interest when selecting which DBMS to use. Hardware efficiency is an important aspect to consider as one of the major financial costs of running a database system is power consumption.

To help understand the reason why throughput is fairly constant across workloads at 10 000 rows/operations and why the workload composition has a larger impact on throughput for larger sizes, similar studies can be conducted where the dataset size and the number of operations are changed separately. Only changing one variable at a time would also enable throughput scaling comparisons between the databases and in the process, clarify the effect each variable has on DBMS throughput.

Another suggestion would be adding extra nodes for each DBMS would make the results of the experiment more generalizable and more trustworthy since it would replicate how real-world data-centers work. Investigating this further would lead to making the decision for consumers easier for their desired product.

More studies need to be conducted regarding the Cassandra spike for Workload E with 1 000 000 rows/operations despite it being discussed briefly in the discussion section. We strongly

believe that investigating this behavior could be beneficial for projects that require very heavy update operations that come with big datasets, as it can potentially save extra unnecessary costs. This could be investigated by interviewing the developers of Cassandra or with a more in-depth literature study of Cassandra's inner workings. Additional benchmark studies comparing Cassandra's throughput across read/update workloads could also be performed, but with a varying number of client threads. This could hopefully show if Cassandra's throughput is more affected (and how it is affected) by the number of client threads compared to other DBMS.

References

Abramova, V. and Bernardino, J. (2013). NoSQL databases: MongoDB vs cassandra. In Proceedings of the international C* conference on computer science and software engineering (pp. 14-22).

Anusha, K., Rajesh, N., Kavitha, M. and Ravinder, N. (2021). Comparative Study of MongoDB vs Cassandra in big data analytics. *2021 5th International Conference on Computing Methodologies and Communication (ICCMC)*. IEEE, pp. 1831–1835.

Araujo, J.M.A., de Moura, A.C.E., da Silva, S.L.B., Holanda, M., de Oliveira Ribeiro, E. and da Silva, G.L. (2021). Comparative performance analysis of NoSQL Cassandra and MongoDB databases. *2021 16th Iberian Conference on Information Systems and Technologies (CISTI)*. IEEE, pp. 1–6.

Berndtsson, M., Hansson, J., Olsson, B. and Lundell, B. (2007). *Thesis projects: a guide for students in computer science and information systems*. Springer Science & Business Media.

Cui, X. and Chen, W. (2021). Performance comparison test of HBase and Cassandra based on YCSB. *2021 IEEE/ACIS 19th International Conference on Computer and Information Science (ICIS)*. IEEE, pp. 70–77.

Eyada, M.M., Saber, W., El Genidy, M.M. and Amer, F. (2020). Performance evaluation of IoT data management using MongoDB versus MySQL databases in different cloud environments. *IEEE access: practical innovations, open solutions*, 8, pp. 110656–110668. doi: 10.1109/access.2020.3002164.

Filip, P. and Čegan, L. (2020). Comparison of MySQL and MongoDB with focus on performance. *2020 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS)*. IEEE, pp. 184–187.

Frank, L., Pedersen, R.U., Frank, H.C. and Larsson, NJ. (2014). The Cap Theorem Versus Databases with Relaxed Acid Properties. *Proceedings of the 8th International Conference on Ubiquitous Information Management and Communication* (pp. 1-7).

Gupta, A., Tyagi, S., Panwar, N., Sachdeva, S. and Saxena, U. (2017). NoSQL databases: Critical analysis and comparison. *2017 International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN)*. IEEE, pp. 293–299.

Gupta, Y. K. and Mittal, T. (2020). Comparative study of Apache pig & Apache Cassandra in Hadoop distributed environment. *2020 4th International Conference on Electronics, Communication and Aerospace Technology (ICECA)*. IEEE, pp. 1562–1567.

Hashem, H. and Ranc, D. (2016). Evaluating NoSQL document oriented data model. *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*. IEEE.

Jansson, J., Vukosavljevic, A., Catovic, I. (2021). Performance comparison between multi-model, key-value and documental NoSQL database management systems. Skövde: School of informatics, University of Skövde.

Jogi, V. D. and Sinha, A. (2016). Performance evaluation of MySQL, Cassandra and HBase for heavy write operation. *2016 3rd International Conference on Recent Advances in Information Technology (RAIT)*. IEEE, pp. 586–590.

Kaur, K. and Sachdeva, M. (2017). Performance evaluation of NewSQL databases. In *2017 International Conference on Inventive Systems and Control (ICISC)*. IEEE, pp. 1-5.

Khasawneh, T. N., AL-Sahlee, M. H. and Safia, A. A. (2020). SQL, NewSQL, and NOSQL Databases: A Comparative Survey. *2020 11th International Conference on Information and Communication Systems (ICICS)*. IEEE, pp. 013–021.

Li, Y. and Manoharan, S. (2013). A performance comparison of SQL and NoSQL databases. In *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*. IEEE, pp. 15-19.

Magdum, J. and Barhate, R. (2018). Performance analysis of DML operations on NoSQL databases for streaming data. *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*. IEEE, pp. 1–6.

Pavlo, A. and Aslett, M. (2016). What's really new with NewSQL?. *ACM Sigmod Record*, *45*(2), pp.45-55.

Reniers, V., Van Landuyt, D., Rafique, A. and Joosen, W. (2017). On the state of nosql benchmarks. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion* (pp. 107-112).

Schreiner, G.A., Knob, R., Duarte, D., Vilain, P. and Mello, R.D.S. (2019). Newsql through the looking glass. *Proceedings of the 21st International Conference on Information Integration and Web-based Applications & Services* (pp. 361-369).

Taft, R., Sharif, I., Matei, A., VanBenschoten, N., Lewis, J., Grieger, T., Niemi, K., Woods, A., Birzin, A., Poss, R. and Bardea, P. (2020). Cockroachdb: The resilient geo-distributed sql database. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (pp. 1493-1509).

YCSB (2021). *Tips and FAQ*. https://github.com/brianfrankcooper/YCSB/blob/master/doc/tipsfaq.html [2022-05-14].

Wang, S., Lu, Z., Cao, Q., Jiang, H., Yao, J., Dong, Y., Yang, P. and Xie, C. (2022). Exploration and Exploitation for Buffer-Controlled HDD-Writes for SSD-HDD Hybrid Storage Server. *ACM Transactions on Storage (TOS)*, *18*(1), pp.1-29.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., & Wesslén, A. (2012). Experimentation in Software Engineering. Publisher: Springer-Verlag Berlin Heidelberg. ISBN:978-3-6

Zhang, M., Martin, P., Powley, W. and Chen, J. (2017). Workload management in database management systems: A taxonomy. *IEEE Transactions on Knowledge and Data Engineering*, *30*(7), pp.1386-1402.