NewSQL Through the Looking Glass

Geomar A. Schreiner
PPGCC
Federal University of Santa Catarina
Florianópolis, Brazil
University of Oeste de Santa Catarina
Chapecó, Brazil
schreiner.geomar@posgrad.ufsc.br

Ronan Knob
Federal University of Santa Catarina
Florianópolis, Brazil
ronan.knob@gmail.com

Denio Duarte
Federal University of Fronteria Sul
Chapecó, Brazil
duarte@uffs.edu.br

Patricia Vilain
PPGCC
Federal University of Santa Catarina
Florianópolis, Brazil
patricia.vilain@ufsc.br

Ronaldo dos Santos Mello
PPGCC
Federal University of Santa Catarina
Florianópolis, Brazil
r.mello@ufsc.br

ABSTRACT

Several applications require to handle large and heterogeneous data volumes as well as thousands of OLTP transactions per second. Traditional relational databases are not suitable for these requirements. On the other hand, NoSQL databases are able to deal with *Big Data*, but lacks the support to the traditional ACID properties. NewSQL is a new class of databases that combines the support to OLTP transactions of relational databases with the high availability and scalability of NoSQL databases. However, few works in the literature explore the differences among different NewSQL solutions. In this paper, we discuss the main features of the most prominent NewSQL products, besides we present benchmarking results for analyzing their performance . We believe that both analysis can be useful as a guide to a future choice of NewSQL technologies.

CCS CONCEPTS

• **Information systems → Relational database model**; *Data layout*; Distributed database transactions.

KEYWORDS

NewSQL, benchmark, comparative analysis, YCSB, Voter

ACM Reference Format:
Geomar A. Schreiner, Ronan Knob, Denio Duarte, Patricia Vilain, and Ronaldo dos Santos Mello. 2019. NewSQL Through the Looking Glass. In *The 21st International Conference on Information Integration and Web-based Applications & Services (iiWAS2019), December 2–4, 2019, Munich, Germany*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3366030.3366080

iiWAS2019, December 2–4, 2019, Munich, Germany

ACM ISBN 978-1-4503-7179-7/19/12…$15.00
https://doi.org/10.1145/3366030.3366080

1 INTRODUCTION

The proliferation of Web technologies and the growth of mobile devices connected to the Internet have brought the need for handling large volumes of heterogeneous data in a short amount of time. This kind of data is usually classified as *Big Data*. A new generation of applications, such as financial systems and online games, are characterized by having a large number of users as well as user interactions with the system, generating a massive amount of data and multiple OLTP transactions per second. In general, OLTP transactions are short-lived transactions that do not process so much data.

For decades, traditional relational databases (RDBs) have been used to store and handle application data. However, they are not able to deal with a massive data volume and, at the same time, provide high availability and ACID guarantee for OLTP transactions. Motivated by these challenges, new DB solutions have emerged like the NoSQL DBs [10, 18]. These solutions offer features such as high availability and scalability, combined with a distributed architecture. Different from traditional RDBs, NoSQL DBs are based on horizontal scaling, *i.e.*, new machines can be added to a cluster to increase the system performance. Although NoSQL DBs can handle large volumes of data with high availability, they usually do not support the traditional ACID properties, which characterize the OLTP transactions.

More recently, the NewSQL paradigm has emerged to combine the benefits of the relational paradigm with the NoSQL paradigm for *Big Data* management. NewSQL systems are a new generation of database that tries to provide the same performance of NoSQL DBs for OLTP workloads with the full support of ACID properties [17]. A NewSQL DB have to consider two important features: *(i)* A *lock-free* concurrency control mechanism; and, *(ii)* A *shared-nothing* distributed architecture [18]. Although they share general characteristics, as mentioned above, each NewSQL system has a particular strategy to perform data manipulation operations. Therefore, it is pertinent to analyze NewSQL solutions through *benchmarks* (standardized testing protocols) capable of measuring performance to verify their effectiveness in real situations.

This paper aims to contribute to the literature about the NewSQL paradigm by comparing four related solutions: *VoltDB, NuoDB,*

MemSQL and *Cockroach*. The comparison is accomplished by running and analysing the results of two benchmarks (*YCSB* and *Voter*). The choice of the solutions consider the following criteria: *(i)* their ranking on the *DB-Engines*[1] site; *(ii)* the number of website mentions and search interests (via Google Trends[2]); *(iii)* a review of the annual survey of the *Stackoverflow* website in 2017[3]; and *(iv)* a free license for testing. Moreover, the selection of benchmarks tools takes into account those which the main focus is on OLTP transactions.

As side effect, we also present an X-ray study of the four selected NewSQL solutions.

The remainder of this article is organized as follows. Section 2 presents the related works, while Section 3 describes the selected NewSQL DBs. The experimental environment and the *benchmarks* chosen are described in Section 5. In Section 6 the results are discussed and, finally, the final considerations are in Section 7.

2 RELATED WORK

In the literature, we find little initiatives that compare exclusively NewSQL DBs. It is noteworthy that there are some works comparing NewSQL and NoSQL DBs [4–6]. However, they perform a theoretical analysis between solutions of two different paradigms, not comparing only NewSQL solutions.

The work of Pavlo and Aslett [17] makes a historical analysis of NewSQL database management systems (DBMSs). However, there is no experimental evaluation. On the other hand, the works of Kaur and Sachdeva [9] and Oliveira and Bernardino [16] report performance evaluations for NewSQL products. The former work makes a comparison between the *NuoDB*, *Cockroach*, *VoltDB* and *MemSQL* DBMS by using simple evaluation metrics that can help in the process of choosing a solution. However, they are not appropriated metrics as they do not consider a distributed environment on which NewSQL solutions are typically used. The latter evaluates the performance of *VoltDB* and *MemSQL* through the *TPC-H* benchmark software. TPC-H, however, focuses on OLAP transactions, while NewSQL solutions prioritize OLTP transactions in their development. Moreover, this work also does not take into account distributed environments to conduct the experiments.

Our proposal differs from the above one since we compare NewSQL solutions using two benchmark tools focusing on OLTP transactions in a distributed environment.

3 NEWSQL

NewSQL is a new class of modern RDBs that supplies the demand for large OLTP workloads without renouncing the ACID properties [10, 17, 18]. As stated before, this new DB generation transcends the traditional RDBs by incorporating essential features to *Big Data* management, such as high availability and scalability [4]. In addition to offering SQL as the primary access language, as well as ACID guarantees, a DB needs to address some other characteristics to be considered a NewSQL DB [17].

Unlike traditional RDBs that are disk-oriented, *i.e.*, use full disk persistence, and the main memory is considered only to optimize some tasks, NewSQL DBs are usually *in-memory* DBs. The main memory (RAM) is the principal storage space, avoiding the high costs with data accesses on the hard drive. The benefit of this approach is to enable specific optimizations since the system does not need mechanisms to deal with the constant exchange of data between disk and main memory.

The use of the main memory as the main storage space is not new [2, 19]. The difference is that the NewSQL systems reallocate a subset of the DB to persistent memory to reduce its memory consumption [13]. This technique allows the DBMS to support a larger chunk of data than the total available main memory without having to use virtual memory as drive-oriented architectures. The most common approach to manage the main memory is to create a monitoring mechanism that identifies not frequently accessed tuples and reallocates them from memory to the disk.

NewSQL DBs are, primarily, distributed databases. The idea is not new: the first products with distributed versions were the System R and INGRES in the eighties [12]. In the NewSQL DB architecture, since they focus on OLTP, tables are divided horizontally into multiple fragments, called partitions or shards [1]. A module of the DBMS assigns each tuple to a fragment based on the values of a set of attributes (possibly unitary) using a range or hash function. Partitioning is a well-known technique that can be used to mitigate the performance degradation caused by distributed transactions. Each partition stores co-related data, and so transactions can run locally.

Another critical feature of NewSQL DBs is the concurrency control [17]. Concurrency control allows multiple users to access the DBs at the same time and execute their transactions independently. They essentially provide the guarantees of atomicity and isolation (from ACID properties) in the system. To implement the concurrency, a few NewSQL systems uses a variant of the timestamp ordering (TO) strategy. In TO, the DBMS assumes that transactions do not perform operations that violate the serializability criteria. The most widely used NewSQL system protocol is the *decentralized multi-version concurrency control (MVCC)*. It creates a version of a tuple in the DB when a transaction updates it. On maintaining multiple versions of a tuple, it allows transactions to be completed even if another transaction updates the same tuple. It also allows long-term (and read-only) transactions not to block writers.

As stated previously, NewSQL DBs are distributed DBs whose data is organized into partitions. So, in order to ensure high data availability and durability, they must support strongly consistent replication [17]. The replication methods implemented by NewSQL DBs consider a wide-area network (WAN) replication. This is a modern operating environment, where it is now trivial to deploy systems in multiple data centers geographically dispersed. Any NewSQL DBMS can be configured to provide synchronous WAN data updates, but this can cause significant slowness in normal operations. Thus, they provide asynchronous replication methods.

Another essential feature of a NewSQL DBMS is to provide recovery mechanisms in critical cases. Fault tolerance is the ability of the system to handle a structural failure. For traditional RDBs, the primary concern for fault tolerance is to ensure that no updates were missed [15]. Unlike the traditional RDBs, the NewSQL DBs must minimize downtime since modern applications are online all the time.

[1] https://db-engines.com/en/ranking
[2] https://trends.google.com/trends/
[3] https://insights.stackoverflow.com/survey/2017

On considering a distributed system with replicas, when the master goes offline for some reason, another node takes its place. At this time, transactions continue to be received and processed. If the old master returns, it first needs to get updates from the current master (and potentially other responses) that it has missed while inactive, and only then it might resume the master role if it is the case.

Regarding the previously presented features, there are some points to consider for adopting the new class of NewSQL DBs. NewSQL DBs are built for distributed execution and optimized for multi-node environments. It includes parts such as the query optimizer and the communication protocol between nodes. Also, NewSQL DBs are responsible for distributing the DB across its capabilities with a custom engine, rather than relying on a *ready-to-use* distributed file system (e.g., HDFS) or a storage structure (for example, Apache Ignite).

On the other hand, the disadvantage of NewSQL DBs is the concern of companies with the adoption of these technologies. Unfortunately, as a new and poorly analyzed technology, its usage by large organizations is not significant. In other words, it means that the number of people who have been in contact with the NewSQL systems is still tiny compared to the big names in traditional DBMSs. Also, an organization is unlike to give up access to administration and control tools designed for the most popular systems [17]. Some of the NewSQL solutions solve this problem by maintaining compatibility with legacy systems such as *MySQL* (*Clustrix* and *MemSQL*).

This paper contributes to the literature as a guide of the NewSQL DBs main characteristics. We present in detail four prominent NewSQL DBs and compare them in terms of features and throughput (using benchmarks). We believe that the results of our study might help users to decide whether or not a NewSQL DB is suitable for their needs.

4 NEWSQL SYSTEMS

This section describes the main features of the NewSQL DB selected by this work. As stated in Section 1, we select the NewSQL systems based on the following criteria: *(i)* their ranking position at the *DB-Engines* site; *(ii)* the number of website mentions and search interest; *(iii)* the review of the annual survey of the Stackoverflow website in 2017 and; *(iv)* a free license to allow testing. From these criteria analysis, we selected *VoltDB*, *NuoDB*, *Cockroach* and *MemSQL* NewSQL DBMS. They are detailed in the following.

4.1 VoltDB

VoltDB is a DBMS maintained by a company called Volt Inc. It was developed in 2014 by Michael Stonebraker and it is available in enterprise and community versions, the latter one licensed under the *GNU Affero General Public License* [20].

VoltDB is a distributed DB that considers main memory storage to maximize data throughput, eliminating the costly disk access. The architecture of VoltDB uses replication across multiple servers to ensure scalability, fault tolerance, and high availability. Also, it is compatible with ACID features and ANSI standard SQL language, which ensures a fast learning curve for advanced database users and allows transactions to be made straight from the application.

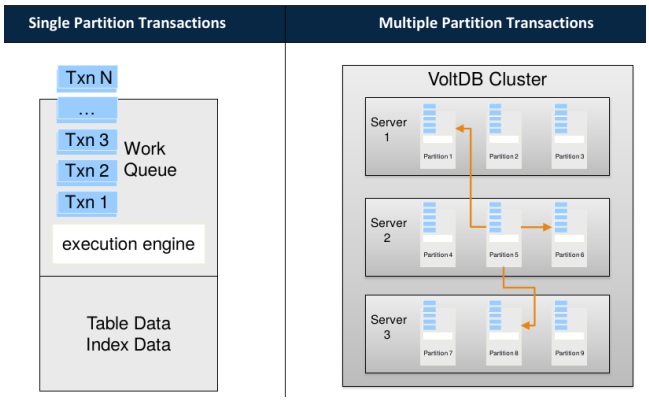

Figure 1: Overview of the VoltDB architecture [20]

Figure 1 shows an overview of the VoltDB architecture. In the right side (*Multiple Partition Transaction*), we see the cluster architecture. It holds *servers*, delimited by rectangles that, in turn, hold processors (*sites*) that store VoltDB partitions. A closer look of a site is presented in the left part of the figure. Each site has a *Work Queue*, the entire *Execution Engine*, and the *tables* and *indexes* data.

A site is the smallest processing part of VoltDB. It is basically a running process that executes in a dedicated processor core and stores a slice of data (a *partition*). Each site executes a series of tasks (a *Work Queue*), and the queue is maintained in a serializable way to avoid locks and latches.

All transactions are sent to the sites, and the *Execution Engine* is responsible for verifying if the site can execute the operations. A transaction can be classified as single-partitioned (*single-site transaction*) if all of its operations can be executed within a single partition. In this case, it allows that the cluster executes other requests in parallel. In such a scenario, the cluster can operate without locks and can be scalable. For a transaction that requires data from multiple partitions (*multi-site transaction*), one site acts as the coordinator. It controls the work that must be done by the other sites, collects the result, and completes (or not) the task. This coordination makes distributed transactions slower than the single-site ones. However, data integrity is guaranteed, and the architecture can support multiple parallel operations.

In VoltDB, tables are partitioned based on a column that the developer or designer specifies, usually based on the application workload. VoltDB optimizes the access to the partitioning columns at runtime through indexes and views. VoltDB also allows users to replicate tables into all partitions in the cluster, which may be required for small but heavily accessed tables. Additionally, the VoltDB query optimizer can dynamically check a query and create copies of the most commonly used tables to optimize performance if transactions heavily manipulate them.

In terms of scalability, VoltDB scales independently of the number of sites without sacrificing network traffic. By attaching one more server to the cluster, or removing it, VoltDB quickly distributes demand among sites, starting with the most accessed content, in a scheme called "no wait architecture".

It is important to notice that VoltDB is not optimized for all data management issues. VoltDB is designed for those applications

that need to process large amounts of data quickly (OLTP fashion), such as financial applications, social networks, among others. For these applications, the requirements are scalability, reliability, high availability, and optimum throughput. VoltDB is not suitable for cases such as collecting and grouping massive historical datasets, which is a typical OLAP demand [20].

4.2 NuoDB

NuoDB was first released in 2010 and its available in three versions: *(i) Community* (a free version); *(ii) Professional*; and *(iii) Enterprise*. The Community version (analyzed by this work) has some scalability restrictions. In terms of the NuoDB architecture, the Community version allows one administration instance, one *Storage Management (SM)* and three *Transaction Processings (TPs)*. The other versions, it is possible to hold unlimited SMs and TEs. There are also other minor user experience advantages, such as access to the graphical management interface and differentiated product support [7].

NuoDB claims to supports OLTP and OLAP loads at the same time (it is called *HTAP*) [7]. It utilizes a technology called *Durable Distributed Scale-out*, modern architecture with separated services for transaction processing and storage management [7]. This second layer enables distributed processing that can be deployed under multiple data centers and optimized for memory speeds, continuous availability, and elastic growth.

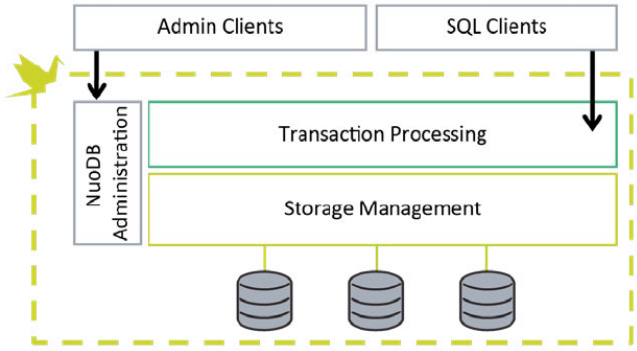

Figure 2: Overview of the NuoDB architecture [7]

NuoDB is a two-tier distributed architecture. It can be deployed in a data center, many data centers, or over a hybrid cloud while maintaining ACID guarantees and speed of access to memory. If a data center experiences a disaster, other partners can assume the workload without any downtime. The application automatically manages the complexity of load balancing, conflicts, and other issues that may come from this transition. Figure 2 shows a vision of the NuoDB architecture. The architecture is composed of two modules: ((*i*)) *TP* layer and *(ii) SM* layer.

The TP layer receives SQL requests. It contains memory nodes called Transaction Engines (TEs) that maintain high memory performance. When an application makes requests to NuoDB, a TE creates an in-memory cache for the application workload. Requests for non-cached files are fed with memory caches from other TEs or the SM layer.

The SM layer, in turn, consists of storage managers processes nodes called Storage Managers (SMPs), which have both memory and disk components. SMPs also offer guarantees of data durability. Multiple SMPs can be used to increase data redundancy. The two layers described above make NuoDB scalable as it merely needs to add or remove TEs and SMPs to scale the DB as needed.

NuoDB is optimized to work in cloud environments, in the form of DBaaS services. The product behaves logically to the end-user as a single environment and provides ACID features in ANSI-standard SQL transactions and queries [7]. NuoDB internal structure is designed to scale elastically, and the product has orchestration tools to manage and administer the DB and the application environment.

4.3 CockroachDB

Cockroach DB is a database system created by *Cockroach Labs* in 2015. The Cockroach DB project is designed to be an open-source database, distributed at the level that an instance can be raised on a regular personal computer and assists in processing requests [11]. The system is available ins two versions: (*i*) *Core* and (*ii*) *Enterprise*. The difference between versions is that the Enterprise version allows geo-partitioning, special user permissions and better cluster monitoring services.

Usually, NewSQL DBs uses the main memory as final storage, but Cockroach DB also uses the disk. It takes advantage of an atomic block structure for block writing, which makes it easy to support transactions with ACID characteristics.

The architecture of Cockroach DB receives SQL commands via an API and converts to Key-Value (KV) data, which is extremely fast to manipulate. This architecture is made up of 5 layers that interact directly to each other. Figure 3 shows the architecture of Cochroack DB.

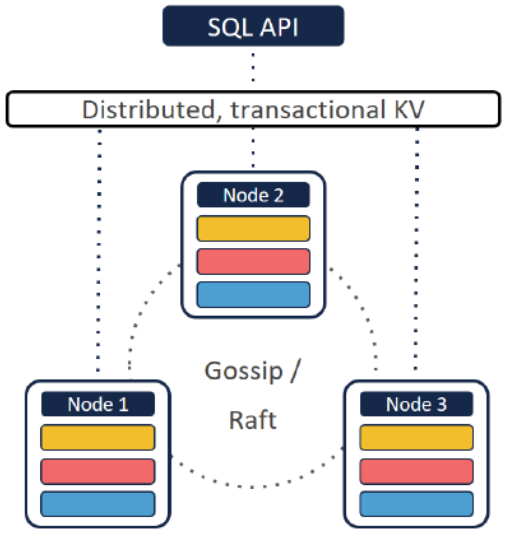

Figure 3: Overview of the Cockroach DB architecture [11]

The SQL Layer (*SQL API*) receives SQL commands and defines the execution plan. The plan is passed to the subsequent layers: Distributed and Transactional KV. In terms of consistency, ACID characteristics are guaranteed at the transaction layer. All statements are treated as transactions, including simple ones. Transactions that access data in different nodes (distributed ones) use a two-phase commit process (Distributed layer does the job). In the first phase, when writing operation occurs, the system creates an in-memory transaction record. All the write operations are submitted to the nodes as a *write intention* to be analyzed. The second phase is the commit that occurs after the writing intentions are accepted.

The *Distributed, transactional KV* layers (Figure 3) are also responsible for routing the operation of the query plan and managing the cluster. The distributed layer stores a monolithic sorted map with Key-Value pairs that describe all cluster data and its location. The map is divided into *ranges* so that the key lookup is not centered on one node. The map maintains the location of data in the nodes. Also, the layers are responsible for copying data between nodes (replication) and applying a consensus algorithm to ensure consistency. The consensus algorithm (Gossip or Raft protocol) works with a quorum of active nodes to accept a transaction. The minimum quorum is three active nodes. If the minimum of nodes considering the replication factor is not reached, the system loses reliability and becomes unstable until the fault nodes are restored.

Each CockroachDB node is composed of least one dedicated space where the DB process reads and writes data to disk. Data is saved via the *RocksDB* API, which stores key-value data on disk. There are three RocksDB instances in each Cockroach instance (node)(the color rectangles in the Figure 3): (*i*) the first one stores the log, (*ii*) the second one stores SQL temporary data, and (*iii*) the last one stores all data of the node. There is also a shared cache block between stores on the same node. For security reasons, a cache block maintains a collection of copies of data portions stored in the node.

4.4 MemSQL

MemSQL is a distributed and in-memory database management system developed by *MemSQL Inc.* The system is available in two versions: the *Developer* version, which is free, but is not recommended for real application since the features are limited, and the *Enterprise* version. MemSQL scales horizontally and maintains compatibility with data processing ecosystem technologies (orchestration platforms, IDEs, and BI tools). In Figure 4, an overview of the MemSQL architecture is presented.

The architecture of MemSQL is basically a two-tiered architecture consisting of (*i*) Aggregator nodes and (*ii*) Leaf nodes. The first one, shown at the top of Figure 4, acts as a query router and a gateway in the distributed system. It stores metadata, distributes queries on leaf nodes, and combines the results to send back to the client. The number of Aggregators defines the level of parallelism, that is, the number of requests that can be processed simultaneously [14].

The system uses a leaf tier function for storage proposes. Data is distributed to leaf nodes transversely into partitions to implement parallelized query execution. Increasing the number of leaf nodes in the system increases the speed of task execution, especially those requiring large queries and aggregations. It also increases the DB

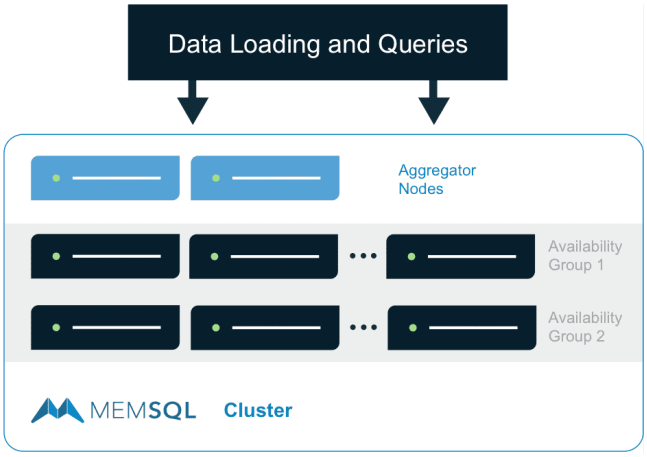

Figure 4: Overview of the MemSQL architecture [14]

ability to execute parallel tasks. Figure 4 does not show the Leaf tier.

The Availability Groups, shown in the middle of the Figure 4, are responsible for dividing the application workloads. For example, one group of aggregators can serve application A, while another group application B.

Communication between nodes is accomplished via SQL commands under a MySQL protocol. Both layers of the architecture implement automatic failure recovery plans to provide fault tolerance. The proportion of aggregator nodes and leaf nodes in the application determines the capacity and performance of the cluster, which may vary by application domain on which they are assigned [14].

4.5 Comparison

This section presents a comparison of the previously presented NewSQL systems. We group the some features and organize them in the Table 1 as follows: (*i*) *Year* denotes the year that the first version was available; (*ii*) *In-Memory* highlights if the system uses the main memory to store the data; (*iii*) *Concurrency Control* shows what algorithm the system applies to execute concurrency control; (*iv*) *Replication* shows the replication model; (*v*) *Foreign Key* informs if the system supports foreign keys; and (*vi*) *Streaming* highlights if the system deals with streaming queries.

As shown in Table 1, all approaches are recent, the VoltDB, for example, was first launched in 2010. It is the first commercial NewSQL system and is based on HStore [8]. The newest system is the Cockroach DB, released in 2014. Almost all approaches use the main memory as the principal resource to store their data. Cockroach DB is the only one of the evaluated approaches that stores data also in the disk. MemSQL and NuoDB consider the disk as an extra resource if the data does not fit in the main memory. Additionally, VoltDB is the only one that maintains all the data in the main memory. So, for running an application, the cluster needs enough RAM to store all the data.

The majority of the analyzed systems use MVCC as a protocol for concurrency control [17]. In this protocol, for every transaction,

Produts	Year	In-Memory	Concurrency Control	Replication	Foreign Key	Streaming
VoltdDB	2010	Yes	TO	Master-Master Master-Slave	No	Yes
NuoDB	2013	Yes	MVCC	NuoDB*	Yes	No
Cockroach DB	2014	No	MVCC	Master-Master	Yes	No
MemSQL	2012	Yes	MVCC	Master-Slave	Yes	Yes

Table 1: Comparison of NewSQL systems

a new version of the target tuple is created, allowing that all transactions commit without using lock protocols. The MVCC applied by NuoDB is different from the others. It employs a Gossip or Raft protocol to broadcast data version information between nodes [17]. VoltDB is the only one that uses TO protocol for concurrency. Basically, it serializes the execution of all transactions eliminating the concurrency for a transaction executed in single-sited mode. For multi-sited transactions, VoltDB uses a coordinator to ensure the ACID properties.

Each approach applies different techniques to replicate data in a cluster. VoltDB uses two different techniques for replication. The *Master-Master* technique is applied in a regular cluster. VoltDB executes the same transaction in the same order in the data as well as their replicas to ensure consistency. Additionally, it uses *Master-Slave* replication in a geographically distributed cluster. Cockroach DB applies a Master-Master replication to facilitate the load balance in the cluster. In turn, MemSQL uses a Master-Slave technique. Write operations are executed first in the affected partition and then copied and sent to the replicas. Different from the other NewSQL systems, NuoDB uses a proprietary replication technique. The Storage Management tier is responsible for the replicas; however, we do not find further information about how it works.

Also, according to Table 1, all approaches support foreign keys but VoltDB.

Streaming query support is also a desirable NewSQL feature. It allows DBMS dealing with data, usually generated in real-time and a form of a data sequence [12]. NuoDB and Cockroach DB do not have support for this feature. VoltDB architecture executes all instructions in a serialized way and creates special tables that are used in the streaming process. MemSQL enables users to construct real-time data streaming pipelines to execute their queries.

5 EXPERIMENTAL EVALUATION

This section details the settings and the considered benchmarks for our experimental evaluation. In order to execute the experiments, we use a standardized infrastructure as well as a specific cluster configuration for each NewSQL DBMS. All DBMSs have been installed and configured by default (without any optimization) in a cluster with three physical hosts/machines. All the experiments were performed on hosts of $Intel^R\ Core^{TM}$ i5-7200 processor (4 2.50 GHz physical cores) with 8 GB RAM (DDR3 1333Mhz), 320GB hard drive (5400 RPM) and Xubuntu 16.04 Server LTS operating system 64 bits. Due to configuration issues, *VoltDB* was installed on the *cluster* using *Docker*. The other products were installed directly in the machines. It is noteworthy that although all products were installed on the same three machines, only one of them was active during the execution of the experiment. Also, the connection between the hosts was set via *Ethernet* (100 Mbps) without access to the external network.

We use the *OLTP-Bench* tool [3] benchmarking system to generate the workloads, which was executed on a node external to the *cluster*. OLTP-Bench is a benchmark framework that supports several commercial DBMSs, as well as a good load of distinct benchmarks. Basically, the tool generates a transaction queue that executes according to the chosen benchmark specification and a user-supplied configuration file. This queue is executed in parallel by *workers* (a worker emulates a user) configured as a numeric parameter in the input file. During the tests, OLTP-Bench collects execution statistics, returning a file with the results at the end of the execution.

On considering the characteristics of NewSQL DBMSs, we select *benchmarks* that could simulate a scenario with simple transactions but applied to a large amount of data. In this context, the chosen benchmarks were *Yahoo! Cloud Serving Benchmark (YCSB)* and *Voter*. Both of them are detailed in the following.

Among the metrics provided in the OLTP-Bench output file, we selected two of them: *(i)* the rate of transactions executed over time (*Throughput*) and *(ii)* the latency of the transactions, *i.e.*, the analysis of the general average of the latencies as well as the analysis of the 90th and 99th percentiles by transaction type. The standard deviation of the observed samples is also calculated.

5.1 YCSB

YCSB is a workload-generating application and package with standard payloads that cover interesting parts of performance appraisal, such as read and write load; and table scan. Each load represents a mix of reading and writing operations with different data volumes and requests.

The test framework consists of a single table, named *usertable*, with N fields. Each record is identified by a primary key (*e.g.*, "user234123") and each field is named as *field0*, *field1*, . . ., *fieldN*. Field values are random strings of random size ASCII characters. The parameters *number of fields* (N) and *scale factor* (F) are given *a priori*.

During the test execution, the benchmark makes several random choices, such as the operations to be executed (*e.g.*, Insert, Update, Read or Scan), what is the record to read or write, and how many records the query needs to examine. Random distributions govern these decisions. In the execution parameters, some factors related to the test base volume scale, and the workload data are configured. In

this work, we used the scale factor 1000, 64 virtual users emulated for manipulation (64 simultaneous connections), and a test limit of 300 seconds. The total amount of data generated was 18.2 GB.

5.2 Voter

Voter is a benchmark based on the software used by a television talent show aired in Japan and Canada. Users call to vote for their favorite candidate. Upon receiving a call, the application invokes the transaction that updates the total number of votes of each participant. Votes cast by each user are stored in a DB and have a configurable upper limit. A separate transaction is periodically invoked to compute the total votes during the program.

This benchmark was developed to saturate the DB with small transactions, all of them updating a small number of records. The *Voter* DB is composed of three tables that hold data about the candidates and the calling user. In addition, there are two *views* that are queried to update the status of the television program. In this work, the scale factor (1000) and the number of emulated virtual users (64) were used as parameters. The data volume generated was 2.6 GB.

6 RESULTS ANALYSIS

This section presents the results obtained by the execution of the benchmarks presented in the previous section.

6.1 *Benchmark* YCSB

YCSB benchmark results were collected from 5 test runs for each NewSQL solution. The bar graph in Figure 5 shows the results for the first metric: the number of transactions executed per second.

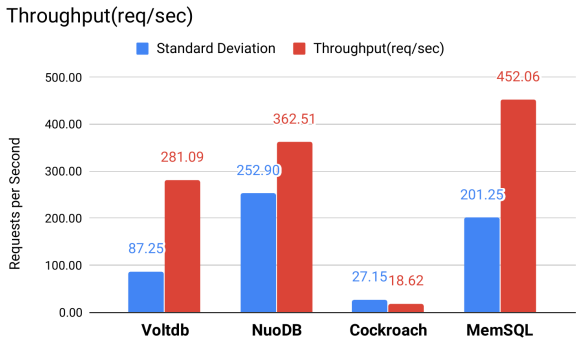

Figure 5: Average of transactions per second for *YCSB benchmark*.

From the graph of Figure 5, we notice that *MemSQL* got the best performance, averaging 456.06 transaction per second, followed by NuoDB with 362.51 transaction. *VoltDB* and *Cockroach* has got the worst performance, 281.09 and 18.62, respectively. The standard deviation shows that the degree of dispersion between the results was high. We can claim that *VoltDB* is the most stable DBMS for YCSB benchmark. Its standard deviation is one-third of the average. On the other hand, *Cockroach*'s standard deviation is very close to its average. We believe that the reason for this unbalanced execution is related to the way YCSB works. All transactions are performed on one table, and this can provoke locks during the execution. *VoltDB* uses TO strategy (see Table 1) to guarantee the concurrency control. TO seems to be lighter than MVCC when the scenario is composed of transactions on one table. Notice that *Cockroach*, again, has handled only 18.62 transactions per second. Its throughput is three times worse than the third DBMS in the ranking.

The other considered metric is the average latency of transactions. The results for this metric are shown in the bar graph of Figure 6. As can be seen, *CockroachDB* also had higher average transaction latency when not considering the transaction type. The standard deviation shows that the degree of dispersion between the results was high for *CockroachDB* and *NuoDB*, indicating that this DBMS had a large value discrepancy, unlike other products that presented a more consistent latency. Regarding the mean, the best average latency is *MemSQL* (0.171 per second), followed by *NuoDB*, *VoltDB*, and *CockroachDB*.

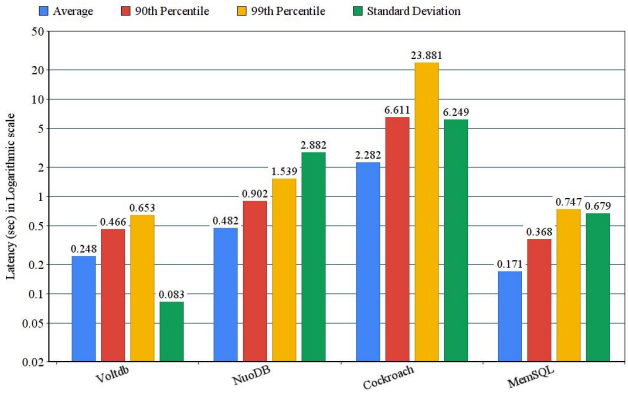

Figure 6: Average of latency per second for *YCSB benchmark*.

An analysis of the 90 and 99 percentiles show that a significant amount of the most latent transactions are found in a small number of transactions at these points. The most obvious case is CockroachDB (Figure 6): there is a big difference between the average 2.28 seconds on average latency and the 23.88 seconds on average latency in the 99 percentile.

The CockroachDB results for both metrics are discrepant with other products. This is because CockroachDB primary storage is not the main memory, like the other products, but in *KV* structures on disk (using RocksDB). It uses a percentage of main memory for the cache but uses much of its disk-based operations, which generates the observed latency and flow rate.

Discrepancies are also noted for *NuoDB*. In both average transactions per second and average latency analysis, the product shows a somewhat high degree of dispersion. Such results tend to occur given the limitations of the free version, where it is possible to use storage in only one node of the cluster even though it is possible to use the three-node transaction execution component. This single storage can increase execution latency as nodes perform part of the test on one node, transferring the entire result to the node holding the data.

6.2 Voter *Benchmark*

Like YCSB, we collect the results for 5 test runs of the Voter benchmark for each NewSQL DBMS. Figure 7 shows the resulting graphic of the first metric: the number of transactions executed per second.

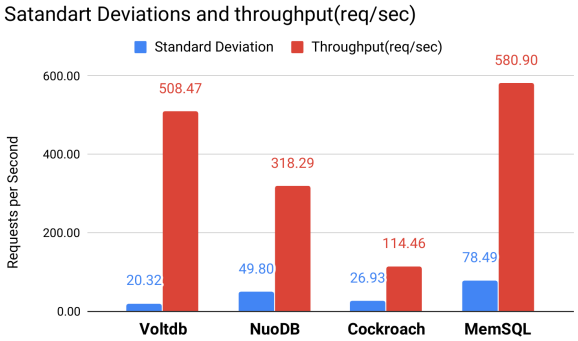

Figure 7: Average of transactions per second for *Voter benchmark*.

From the graph of Figure 7, we notice that MemSQL holds, again, the best results, averaging 580.9 transactions every second of the test, followed by VoltDB, which executed 508.47 transactions per second. *NuoDB* presented an intermediate execution time, leaving again *CockroachDB* in the last position. The standard deviation reveals that the degree of dispersion between the results was lower than in the last experiment, highlighting a good uniformity in the execution.

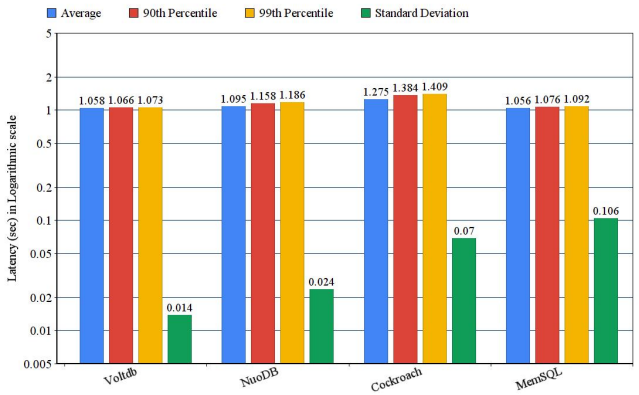

Figure 8: Average of transactions per second for *Voter benchmark*.

The graphic of Figure 8 presents the average transaction latency obtained for *Voter*. Again, we observe a higher latency for *CockroachDB*. The standard deviation shows that the degree of dispersion between the results was very low, showing a good uniformity in execution. The *Voter* test shows a very different situation from that observed with the same test executed with *YCSB* (Figure 6). In *Voter*, the variation obtained between one product and another is small, both in the average and in the 90 and 99 percentiles. It demonstrates that, in smaller transactions, such as this benchmark, the DBMSs operate similarly.

Voter seems to better explore the expected features of a *NewSQL* DB. As observed earlier, the transactions consist of a telephone number vote on an American Idol program participant. Thus, this benchmark brings greater stress on the tested products, simulating multiple users by voting for their favorite participant. *MemSQL* and *VoltDB* DBs presented very similar results, demonstrating that they are capable of handling multiple simultaneous low latency requests. *NuoDB* features an architecture with more complex levels of storage, which slightly impairs its performance with a larger volume of simpler transactions. Finally, *Cockroach* presented the worst results. Its architecture, despite bringing new algorithms, is still limited to some operations that use the disk, depreciating its performance when compared to the other DBMSs.

7 CONCLUSION

This paper analyzes four products that are based on the *NewSQL* data management paradigm. A benchmark technique was employed, and the chosen benchmarks were managed through a framework called *OLTP-Bench*. The considered benchmarks have specific characteristics for the evaluation of different scenarios of OLTP transactional environments. The benchmark *YCSB* is widely used for distributed database evaluation, holding a more complex structure (a more substantial number of tables) that involves a combination of transactions ranging from writes and heavy reads to table scans. *Voter*, on the other hand, presents a more straightforward structure, having a focus on DB saturation with several quick requests (inserts and updates) in a small set of tables.

The results revealed that the *MemSQL* DBMS had maintained in the first position for the observed characteristics, obtaining a high throughput rate and low latency. *VoltDB* and *NuoDB* products behaved similarly in most of the analyzed results, even with the restrictions on the free version of *NuoDB*. SGBD *CockroachDB* had obtained the worst results, with considerable discrepancies in the observed metrics, especially in the average transaction rates per second.

As future works, we intend to perform the same analysis by considering geographical partitioning of nodes for further verification of the analyzed characteristics. Another interesting point is to test the scalability of the NewSQL products, adding more nodes and cores. We also intend to include traditional RDBs in the evaluation to prove the advantages that the *NewSQL* paradigm rises.

ACKNOWLEDGMENTS

This work was partially supported by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

REFERENCES

[1] Mohammed Al-Kateb, Paul Sinclair, Grace Au, and Carrie Ballinger. 2016. Hybrid Row-column Partitioning in Teradata&Reg;. *Proc. VLDB Endow.* 9, 13 (Sept. 2016), 1353–1364.

[2] David J DeWitt, Randy H Katz, Frank Olken, Leonard D Shapiro, Michael R Stonebraker, and David A. Wood. 1984. Implementation Techniques for Main Memory Database Systems. *SIGMOD Rec.* 14, 2 (June 1984), 1–8. https://doi.org/10.1145/971697.602261

[3] Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudre-Mauroux. 2013. OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases. *Proc. VLDB Endow.* 7, 4 (Dec. 2013). https://doi.org/10.14778/2732240.2732246

[4] Katarina Grolinger, Wilson A. Higashino, Abhinav Tiwari, and Miriam AM Capretz. 2013. Data management in cloud environments: NoSQL and NewSQL data stores. *JoCCASA* (Dec 2013). https://doi.org/10.1186/2192-113X-2-22

[5] Yuri Gurevich. 2015. *Comparative Survey of NoSQL/NewSQL DB Systems.* Ph.D. Dissertation. The Open University.

[6] Omar Hajoui, Rachid Dehbi, Mohammed Talea, and Zouhair Ibn Batouta. 2015. AN ADVANCED COMPARATIVE STUDY OF THE MOST PROMISING NOSQL AND NEWSQL DATABASES WITH A MULTI-CRITERIA ANALYSIS METHOD. *Journal of Theoretical & Applied Information Technology* 81, 3 (2015).

[7] NuoDB INC. 2018. NuoDB: Architecture (White Paper). http://go.nuodb.com/white-paper.html, Last access: 05/07/2019.

[8] Robert Kallman, Hideaki Kimura, Jonathan Natkins, Andrew Pavlo, Alexander Rasin, Stanley Zdonik, Evan P. C. Jones, Samuel Madden, Michael Stonebraker, Yang Zhang, John Hugg, and Daniel J. Abadi. 2008. H-store: A High-performance, Distributed Main Memory Transaction Processing System. *Proc. VLDB Endow.* 1, 2 (Aug. 2008), 1496–1499. https://doi.org/10.14778/1454159.1454211

[9] K. Kaur and M. Sachdeva. 2017. Performance evaluation of NewSQL databases. In *2017 International Conference on Inventive Systems and Control (ICISC).* 1–5. https://doi.org/10.1109/ICISC.2017.8068585

[10] Rakesh Kumar, Neha Gupta, Shilpi Charu, and Sunil Kumar Jangir. 2014. Manage Big Data through NewSQL. In *National Conference on Innovation in Wireless Communication and Networking Technology–2014, Association with THE INSTITUTION OF ENGINEERS (INDIA).*

[11] Cochroach Labs. 2018. Architecture Overview. https://www.cockroachlabs.com/docs/stable/architecture/overview.html Último acesso em: 21/06/2018.

[12] Patrick Valduriez (auth.) M. Tamer Özsu. 2011. *Principles of Distributed Database Systems, Third Edition* (3 ed.). Springer-Verlag New York.

[13] Lin Ma, Joy Arulraj, Sam Zhao, Andrew Pavlo, Subramanya R. Dulloor, Michael J. Giardino, Jeff Parkhurst, Jason L. Gardner, Kshitij Doshi, and Stanley Zdonik. 2016. Larger-than-memory Data Management on Modern Storage Hardware for In-memory OLTP Database Systems. In *Proceedings of the 12th International Workshop on Data Management on New Hardware (DaMoN '16).* ACM, New York, NY, USA, Article 9, 7 pages. https://doi.org/10.1145/2933349.2933358

[14] MemSQL. 2018. MemSQL Architecture: Technology Innovations Power Convergence of Transactions and Analytics. https://www.memsql.com/content/architecture/ Último acesso em: 06/10/2018.

[15] C. Mohan, Don Haderle, Bruce Lindsay, Hamid Pirahesh, and Peter Schwarz. 1992. ARIES: A Transaction Recovery Method Supporting Fine-granularity Locking and Partial Rollbacks Using Write-ahead Logging. *ACM Trans. Database Syst.* 17, 1 (March 1992), 94–162. https://doi.org/10.1145/128765.128770

[16] João Oliveira and Jorge Bernardino. 2017. NewSQL Databases-MemSQL and VoltDB Experimental Evaluation.. In *KEOD*. 276–281.

[17] Andrew Pavlo and Matthew Aslett. 2016. What's Really New with NewSQL? *SIGMOD Rec.* 45, 2 (Sept. 2016), 45–55. https://doi.org/10.1145/3003665.3003674

[18] Michael Stonebraker. 2012. Newsql: An alternative to nosql and old sql for new oltp apps. *Communications of the ACM. Retrieved* (2012), 07–06.

[19] J. Valdes, H. Garcia-Molina, and R. Lipton. 1984. A Massive Memory Machine. *IEEE Trans. Comput.* 33, 05 (may 1984), 391–399. https://doi.org/10.1109/TC.1984.1676454

[20] VoltDB. 2015. VoltDB Technical Overview. http://www.odbms.org/wp-content/uploads/2013/11/VoltDBTechnicalOverview.pdf, Last access: 05/07/2019.