Сучасний стан наукових досліджень та технологій в промисловості. 2021. № 2 (16) *ISSN 2522-9818 (print)* *ISSN 2524-2296 (online)*

UDC 004.657; 004.051; 004.413.5 DOI: https://doi.org/10.30837/ITSSI.2021.16.019

O. MAZUROVA, A. NABOKA, M. SHIROKOPETLEVA

RESEARCH OF ACID TRANSACTION IMPLEMENTATION METHODS FOR DISTRIBUTED DATABASES USING REPLICATION TECHNOLOGY

Today, databases are an integral part of most modern applications designed to store large amounts of data and to request from many users. To solve business problems in such conditions, databases are scaled, often horizontally on several physical servers using replication technology. At the same time, many business operations require the implementation of transactional compliance with ACID properties. For relational databases that traditionally support ACID transactions, horizontal scaling is not always effective due to the limitations of the relational model itself. Therefore, there is an applied problem of efficient implementation of ACID transactions for horizontally distributed databases. The **subject** matter of the study is the methods of implementing ACID transactions in distributed databases, created by replication technology. The **goal** of the work is to increase the efficiency of ACID transaction implementation for horizontally distributed databases. The work is devoted to solving the following **tasks**: analysis and selection of the most relevant methods of implementation of distributed ACID transactions; planning and experimental research of methods for implementing ACID transactions by using of NoSQL DBMS MongoDB and NewSQL DBMS VoltDB as an example; measurements of metrics of productivity of use of these methods and formation of the recommendation concerning their effective use. The following **methods** are used: system analysis; relational databases design; methods for evaluating database performance. The following **results** were obtained: experimental measurements of the execution time of typical distributed transactions for the subject area of e-commerce, as well as measurements of the number of resources required for their execution; revealed trends in the performance of such transactions, formed recommendations for the methods studied. The obtained results allowed to make functions of dependence of the considered metrics on loading parameters. **Conclusions**: the strengths and weaknesses of the implementation of distributed ACID transactions using MongoDB and VoltDB were identified. Practical recommendations for the effective use of these systems for different types of applications, taking into account the resources consumed and the types of requests.

Keywords: distributed database; transaction; performance; ACID; NOSQL; NewSQL; MongoDB; VoltDB.

Introduction

Databases (DB) are the basis of the vast majority of modern software applications in various spheres of human life. These are medical solutions, financial and banking systems, social networks, etc. Today, databases and their shells, DBMSs, store and process data, the number of which is significantly increasing every year.

Because the resources of any server on which the DBMS is deployed are limited, to store new amounts of data requires its scaling, vertical or horizontal [1]. Vertical scaling, which consists in increasing the hardware resources (increase in RAM, permanent media) on the database server machine, obviously has limitations and does not provide the necessary result for the implementation of large systems. For this reason, horizontal scaling has become more popular, the essence of which is the introduction of additional node machines, which are combined into a single cluster. Thus, the database is stored on several physical machines, and there are no restrictions on this type of scaling.

Thus, a horizontally distributed database consists of several physical nodes that are hosted on different physical servers and store certain data throughout the database [2]. For this type of scaling, there are two distribution technologies: sharding and replication. The essence of sharding is to place part of the data of the entire database on a particular server, while in the case of replication; copies of all database data are stored on all nodes of the cluster simultaneously.

On the other hand, it is critical for modern databases to execute ACID transactions [3], which have become an integral part of business operations (money transfer, ticket booking) for most areas. But for horizontally scalable databases, there is a problem with performing such ACID transactions, which is that different nodes of the database cluster can interact with each other using a network TCP protocol, which in itself does not support transactionality. This problem applies to both horizontal scaling technologies, but is especially relevant for replicated databases, where the same data is on different physical servers, which must always be in a consistent state [4].

Thus, the developers of distributed systems are faced with the problem of choosing and implementing methods to support distributed transactions when creating applications based on replicated databases.

Analysis of recent research and publications

Classic relational DBMSs were the first to face the problems of horizontal scaling and implementation of distributed transactions: PostgreSQL, MySQL, SQL Server, Oracle and others. Due to the features of the relational data model, such as reference integrity and unique constraints, the corresponding DBMS cannot be easily distributed to several physical machines. In addition, relational ACID principles have been developed to operate within a single address space, and have some difficulty in implementing in a distributed environment. Therefore, most relational databases do not provide out-of-the-box capabilities to support distributed ACID transactions.

Later, to solve the problems of horizontal scaling, databases of a new class NoSQL were developed, in which the developers tried to overcome the limitations inherent in relational data models and the SQL language [5-8]. Since NoSQL DBMSs are not based on the concept of DB schema and relationships between tables, they provide a fairly easy possibility of horizontal scaling from the beginning [9].

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) *Innovative technologies and scientific solutions for industries. 2021. No. 2 (16)*

However, most NoSQL DBMSs support transactionality at the single-record level only, which prevents them from being used in a number of important transactional business operations. A number of developers of NoSQL systems to implement this shortcoming implement additional functionality. An example of such a DBMS is MongoDB [10-11], which fully supports distributed ACID transactions. This one of the most popular systems shows quite high performance results and has integration with a large number of programming languages.

As an attempt to combine the advantages of classic relational and NoSQL DB, a new class of DBMS has recently appeared - NewSQL [12-13]. NewSQL DBMSs have both built-in support for horizontal scaling and, inherent in relational systems, support for transaction ACID and most SQL functionality. And although this class of systems has a fairly high potential, but it has not yet gained much popularity among developers. However, the DBMS VoltDB, which is quite popular among systems of this class, already has integration with many programming languages.

It should not be forgotten that one of the first methods of implementing distributed transactions, which is still used by developers, is their implementation at the application level. That is, the application itself takes responsibility for fixing transactions on all nodes of the cluster, for example, using the template Two-phase commit [14].

Given the analysis, more modern and promising for further research and comparison are the methods of implementing distributed ACID transactions in NoSQL DBMS (on the example of MongoDB) and NewSQL DBMS (on the example of VoltDB).

The purpose of this article is to improve the implementation of ACID transactions for horizontally distributed databases by comparing the effectiveness of such methods on the example of using NoSQL DBMS MongoDB and NewSQL DBMS VoltDB, and develop appropriate recommendations for their application.

This study requires the design of a pilot DB and related transactions, as well as the planning and conduct of a series of experiments on different DB volumes, loads, and DB access modes. Evaluation of the effectiveness of the use of methods should be carried out taking into account such criteria [15-16] as:

- the average execution time of read requests, as well as transactional write requests;
- resources required to perform these queries: database size, RAM required, and percentage of CPU time.

In addition to the absolute values of measurements, it is advisable to determine and analyze the functions of their growth.

Materials and methods

A DB was designed for the experiment, which will then be implemented as a BSON model for MongoDB, as well as a relational model for VoltDB.

The design area is e-commerce, for which distributed systems are created and which is associated with heavy workload and big data. In addition, the real field of activity allowed for a clear demonstration of the need for distributed transactions for applied tasks.

The simplified structure of the experimental DB contains the following basic essences:

- product categories: have only a name;
- goods: have a name, description, quantity in stock, price, and also belong to a certain category;
- properties of goods: have the name of the property, its meaning, and also belong to a particular product;
- customers: have a name, surname, e-mail address, telephone number, and physical address;
- orders: have a specific delivery date, discount, and also belong to a specific customer;
- order elements: indicate a specific product, have the ordered quantity, and also belong to a specific order.

The data structures developed for the experiment are shown in fig. 1 (relational scheme DB) and in fig. 2 (in BSON format).

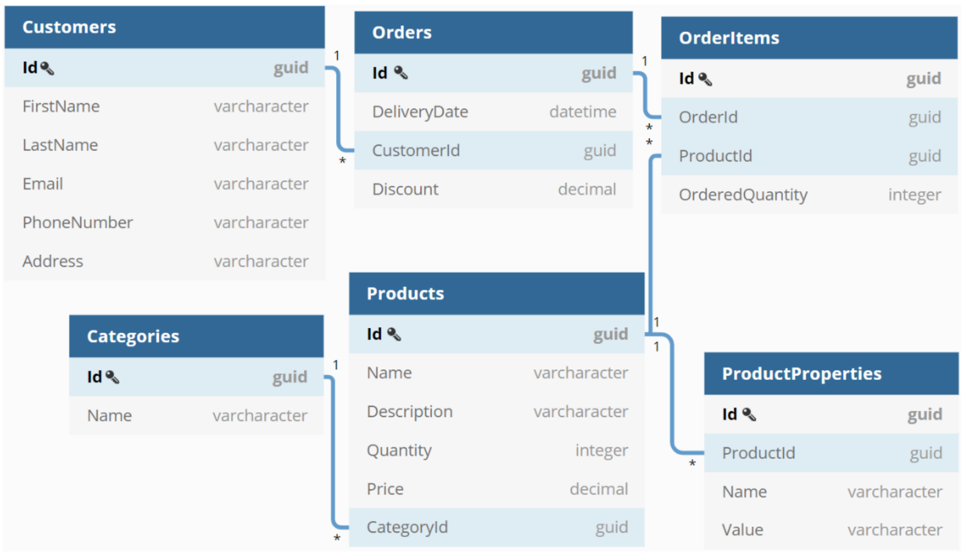
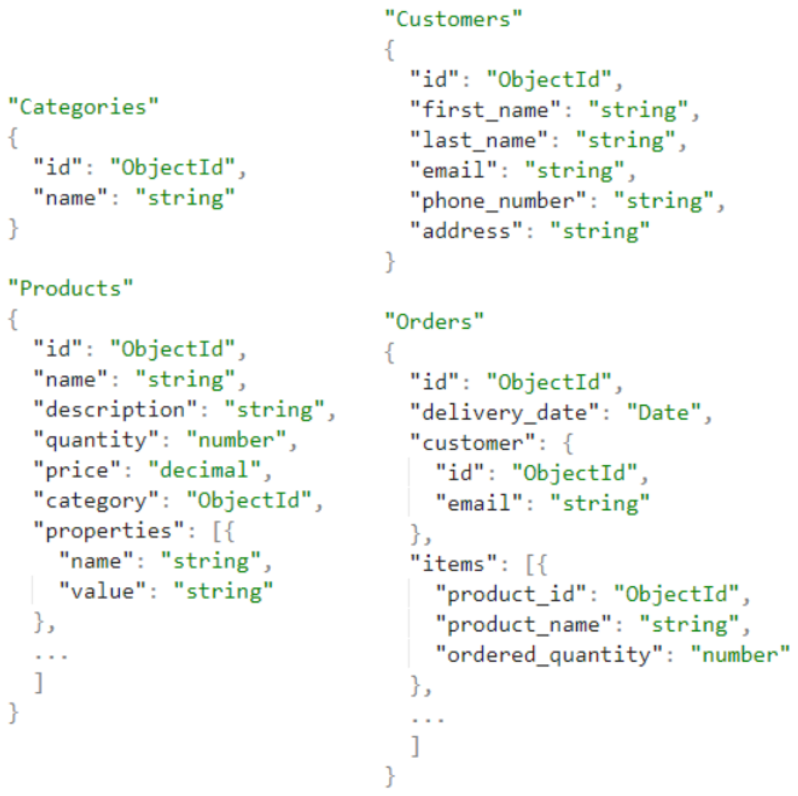


Fig. 1. Relational DB scheme

The following e-commerce business operations were selected to design queries for experiments:

- withdrawal of the most popular categories of goods;
- view and create orders;
- change in the quantity of goods in the warehouse and the price of the goods;
- adding or changing a discount for a specific order;
- adding new categories of goods and the goods themselves for them.



```
"Categories"
{
  "id": "ObjectId",
  "name": "string"
}

"Products"
{
  "id": "ObjectId",
  "name": "string",
  "description": "string",
  "quantity": "number",
  "price": "decimal",
  "category": "ObjectId",
  "properties": [{
    "name": "string",
    "value": "string"
  },
  ...
  ]
}
```

```
"Customers"
{
  "id": "ObjectId",
  "first_name": "string",
  "last_name": "string",
  "email": "string",
  "phone_number": "string",
  "address": "string"
}

"Orders"
{
  "id": "ObjectId",
  "delivery_date": "Date",
  "customer": {
    "id": "ObjectId",
    "email": "string"
  },
  "items": [{
    "product_id": "ObjectId",
    "product_name": "string",
    "ordered_quantity": "number"
  },
  ...
  ]
}
```

Fig. 2. BSON DB structure

For the experiment, transactions based on SQL and BSON queries for VoltDB and MongoDB, respectively, were developed for selected business processes. All transactions are implemented with a standard DBC isolation level for many DBMSs.

For the experiment, the following deployment of DB server nodes was adopted:

- measurements were made for a cluster of DB virtual server machines;
- each server runs in the Azure cloud on a B2s virtual machine with 2 CPUs, 4 GB of RAM and 8 GB of hard drive.
- Windows Server 2016 operating system.

The experimental load on the DB cluster was formed taking into account the following factors:

- the amount of data stored in the database;
- number of simultaneous connections to the database;
- the number of replica nodes in the DB cluster on which transaction results must be recorded simultaneously and consistently.

For a series of experiments, modes associated with specific discrete values for the above factors were identified (table 1).

Table 1. *DB characteristics for study modes*

Regime	Total entities in the database	Number of simultaneous connections	The number of replica nodes in th cluster
Basic	122 064	10	2
Basic+	568 064	30	3
Basic++	1 266 064	50	-
Medium	2 420 064	100	4
Intensive	8 050 064	300	6

The experiment is based on the idea of measuring the execution time of requests and resource consumption by gradually changing one mode to another for one factor, while the values of the other two factors will remain static. This will establish the dependence of these metrics on the specific load factor DB. An experiment will also be performed with a gradual simultaneous change of modes of all factors to compare the most important criteria that affect the execution time of queries.

It was decided to compare all measurements not only through absolute values, but also by identifying the functions of dependence and comparing their growth rate graphically.

For research on the basis of certain business processes, a number of requests were made, namely:

- withdrawal of the three most popular categories of goods;

- withdrawal of the order and its elements by the order ID;

- transaction with the operation of inserting an order with a variable number of order elements, which will be equal to the number of elements for this mode;

- transaction with the operation of updating goods, the number of which is less than 10, namely, reducing their price by 10%;

- transaction to delete all orders, the total amount of which is on a certain segment [x; x + 500], where x ∈ [8,000; 13,000];

- complex transaction with operations of insertion of many orders with their elements, updating with increase in quantity of the goods in a warehouse which current quantity is less than 15, and also updating with transfer of date of the order in a certain range for some days forward;

- a comprehensive transaction with the insertion of new categories and products for them, the insertion of new customers and orders for them, as well as the renewal of the discount for those orders whose delivery date belongs to a certain segment.

To conduct the experiment, special software was developed, the essence of which is to automatically deploy a DB cluster with the number of nodes according to a given mode, fill it with data according to this mode and perform parallel queries to the cluster according to a given mode. The software measures the execution time of each request and constantly monitors the resources consumed.

Research results and their discussion

Let's consider the main most interesting trends in the performance of experiments to support ACID transactions for NoSQL DBMS MongoDB and NewSQL DBMS VoltDB.

Let's start with the analysis of such a resource as the DB volume. In fig. 3 shows a histogram showing for each mode the DB dimensions for both DBMSs with the same number of entities as they contain.

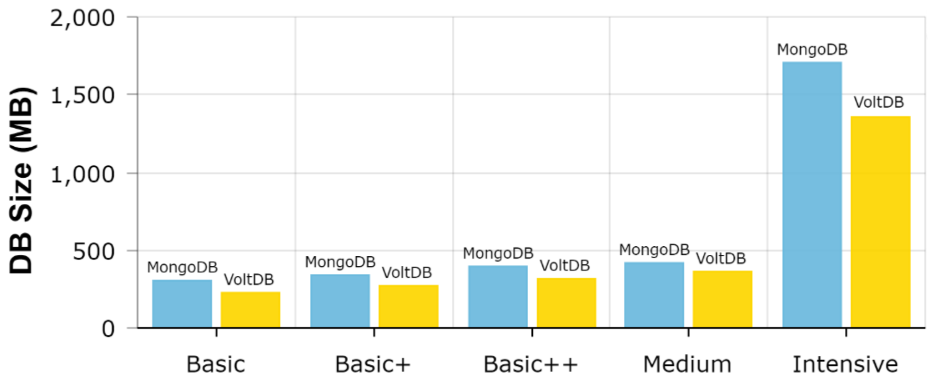


Fig. 3. Comparative diagram of the size of the database

As you can see, VoltDB requires less memory to store the same amount of information in a DB than MongoDB. As the amount of data increases, the DB gap only widens, suggesting that the DBMS VoltDB generally requires less memory to store a single entity than MongoDB does.

Consider the results of measurements of CPU time used as a percentage for both methods. During the experiments it was found that the CPU time in both cases does not depend on the number of replicas in the cluster. But interestingly, MongoDB is characterized by a more rapid growth of CPU consumption when increasing only the factor of the amount of data stored in the database, and when increasing only the number of simultaneous CPU connections, time also increases, but rather slowly and this growth is dampening. On the other hand, VoltDB, on the other hand, shows an increase of only 1% from the lowest mode to the highest when the data grows, but shows a progressive increase with the number of connections. Comparative tables with absolute values for both methods are given in table 2 and table 3.

Thus, in terms of CPU usage, it is better to use VoltDB when it is known in advance that the application will have a large growing amount of data and a more static number of users. Conversely, it is better to use MongoDB when the system aims to attract a large number of users with a more or less fixed data set.

Table 2. *The results of experiments with changing only the amount of data*

Regimes	MongoDB CPU (%)	VoltDB CPU (%)
Data Basic	1.1	1
Data Basic+	3.5	1.1
Data Basic++	3.7	1.3
Data Medium	7.1	1.4
Data Intensive	7.2	2

Table 3. *The results of experiments with changing only the number of simultaneous connections*

Regimes	MongoDB CPU (%)	VoltDB CPU (%
Connection Basic	1.1	1
Connection Basic+	2.5	1.4
Connection Basic++	3.6	2
Connection Medium	3.5	4.2
Connection Intensive	3.7	10

If the proportions of the amount of data and users are unknown in advance or a characteristic increase of both values is assumed, it is better to pay attention to the measurements of the CPU metric, when all loaded factors simultaneously and gradually increase (table 4).

As you can see, in this case VoltDB consumes less CPU and this consumption grows more slowly than in the case of MongoDB. That is why in the case when the application must be ready for a rapid growth of both data and users, it is still better to choose VoltDB in terms of CPU consumption.

Table 4. *The results of experiments, when all load factors increase simultaneously*

Regimes	MongoDB CPU (%)	VoltDB CPU (%)
Basic	1.1	1
Basic+	1.9	1.4
Basic++	6.3	2.2
Medium	10.2	4.7
Intensive	16.5	13

Let's consider the results of measurements of the used RAM consumed by the DB server during transactions in different load modes. Note that from the start, MongoDB consumes several times less RAM than VoltDB. For example, when changing the amount of data, MongoDB consumes from 91 MB of RAM for the lowest level and up to 99 MB for the highest level, and when changing the number of connections from 83 MB to 97 MB. Even in the presence of a certain statistical error, it is clear that growth, if any, is very slow. At the same time, VoltDB consumes when changing data from 218 MB to 560 MB, and when changing connections from 218 MB to 1360 MB. The corresponding histograms of RAM consumption at change of data quantity (fig. 4) and quantity of connections (fig. 5) are resulted. The histograms show that when the amount of data changes, the difference in RAM is smaller than when changing the number of simultaneous connections.

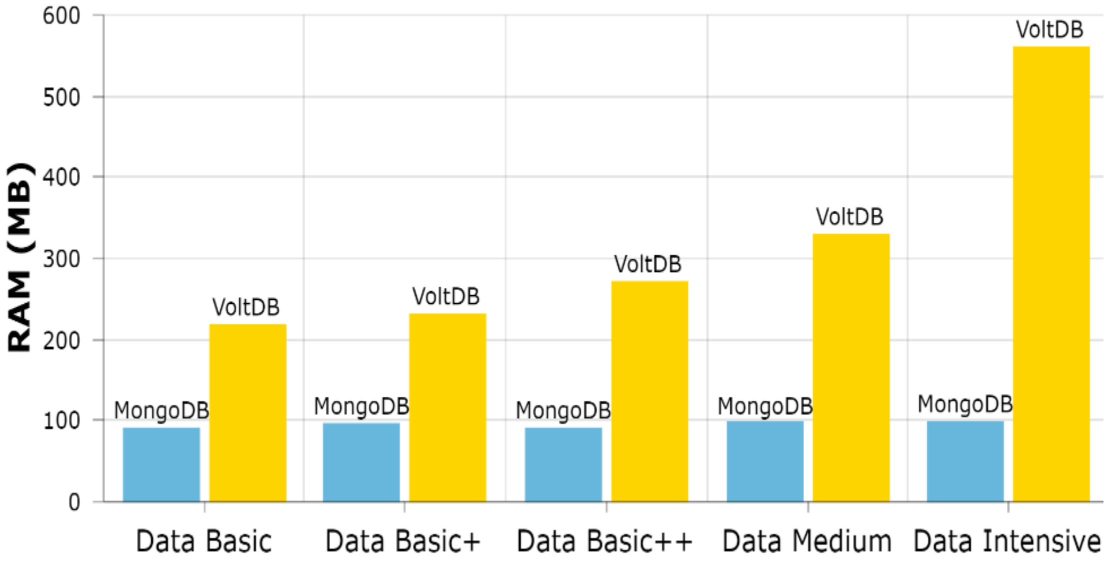


Fig. 4. Comparative histogram of RAM consumption when changing the amount of data

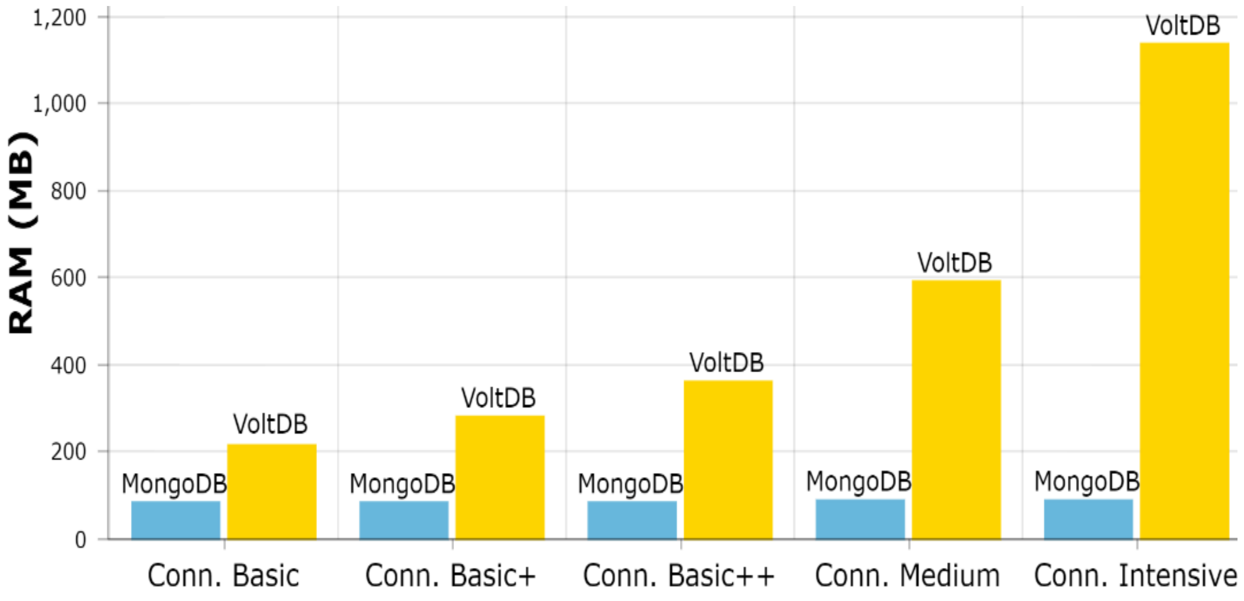


Fig. 5. Comparative histogram of RAM consumption when changing the number of connections

Also note that the amount of RAM for both methods does not actually depend on the number of replicas.

More interesting is the situation with increasing RAM consumption while increasing all load factors. And although in absolute terms MongoDB still loses VoltDB, but shows significant growth at high loaded levels (table 5).

Table 5. *The results of measurements of RAM consumption while changing all factors*

Regimes	MongoDB RAM (MB)	VoltDB RAM (MB
Basic	91	218
Basic+	94	322
Basic++	97	439
Medium	182	692
Intensive	370	1369

We see that for both methods, although RAM consumption doubles between the last two modes, VoltDB still requires an order of magnitude more RAM at high load. Therefore, in terms of using the DB server RAM, it is more efficient to use MongoDB.

Further on we consider the results of measurements of the average execution time of transactions.

Let's consider the results for the operation of inserting a single order with its elements. The number of elements to be inserted will increase when the load mode is changed:

- Basic: 30 elements;
- Basic +: 40 elements;
- Basic ++: 50 elements;
- Medium: 60 items;
- Intensive: 100 items.

The experiment showed that MongoDB executes this transaction several times faster than VoltDB when changing any factor. At the same time, it was found that the data quantity factor in DB does not affect the insert transaction speed for MongoDB, but significantly affects the rate for VoltDB (fig. 6). This revealed a completely opposite situation with the factor of the number of replicas, on which there is dependence for MongoDB (fig. 7), but not for VoltDB. And although the growth is noticeable, but in fact the time changes by only 100 ms.

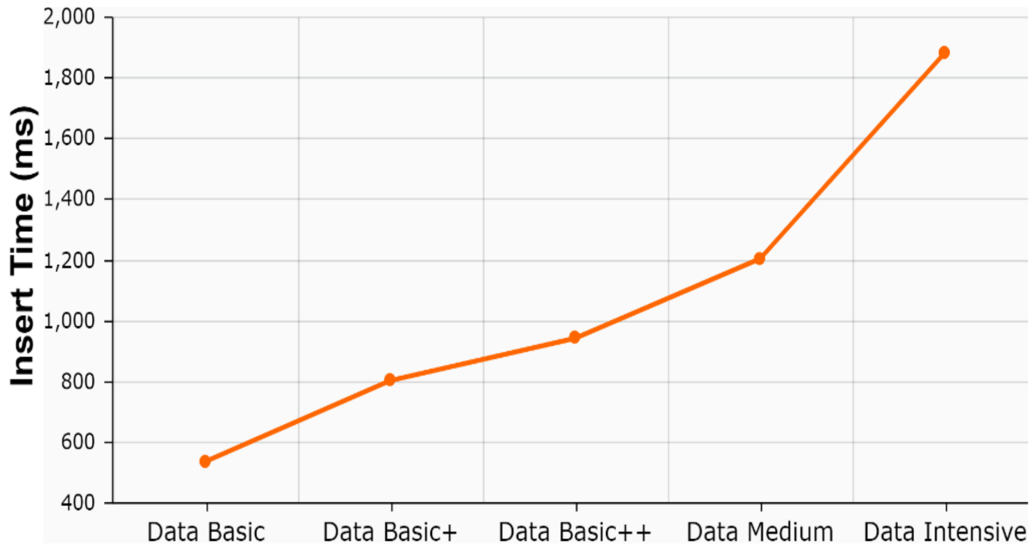


Fig 6. Graph of growth of time of execution of transaction on an insert from quantity of data in basis for VoltDB

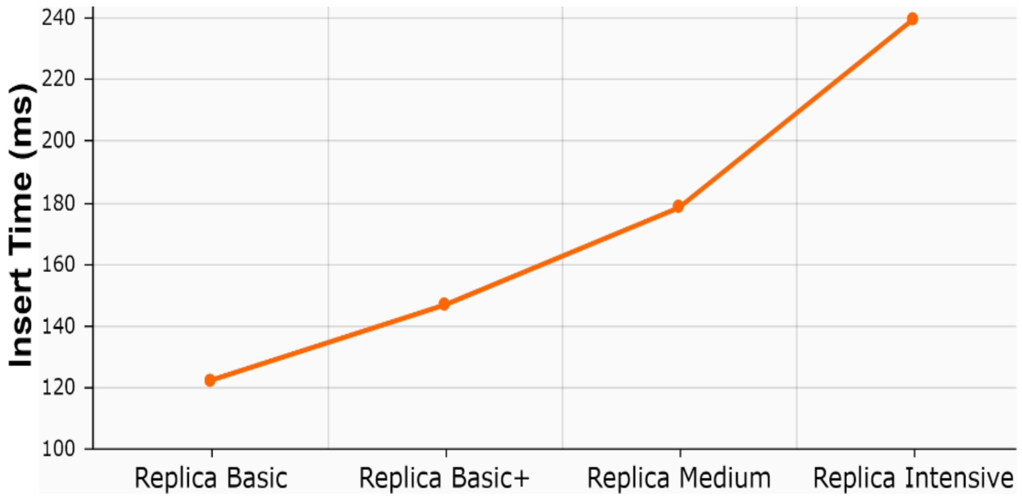


Fig. 7. Graph of growth of time of execution of transaction on an insert from quantity of data in basis for VoltDB

It was found that the insertion time for both methods depends on the number of simultaneous connections. However, MongoDB spends several times less time and this time increases several times slower than for VoltDB (fig. 8).

It was also found that with a simultaneous increase in all load factors, the execution time of the transaction on the insert increases significantly for both methods. In this way, you can find out the maximum bandwidth of the insertion operation for both methods at the highest levels of each factor, as well as at the same time the highest level of all factors simultaneously. A series of experiments showed that MongoDB in all layouts has a higher insertion bandwidth than VoltDB.

Consider the results of experiments to perform an update transaction, the essence of which is to update the price of all goods, the number of which is less than a certain threshold. This in turn means that the DBMS will have to scan the entire table / collection to find such products. Fig.9 shows a histogram comparing both methods for this transaction in terms of maximum execution time when changing each load factor separately and together.

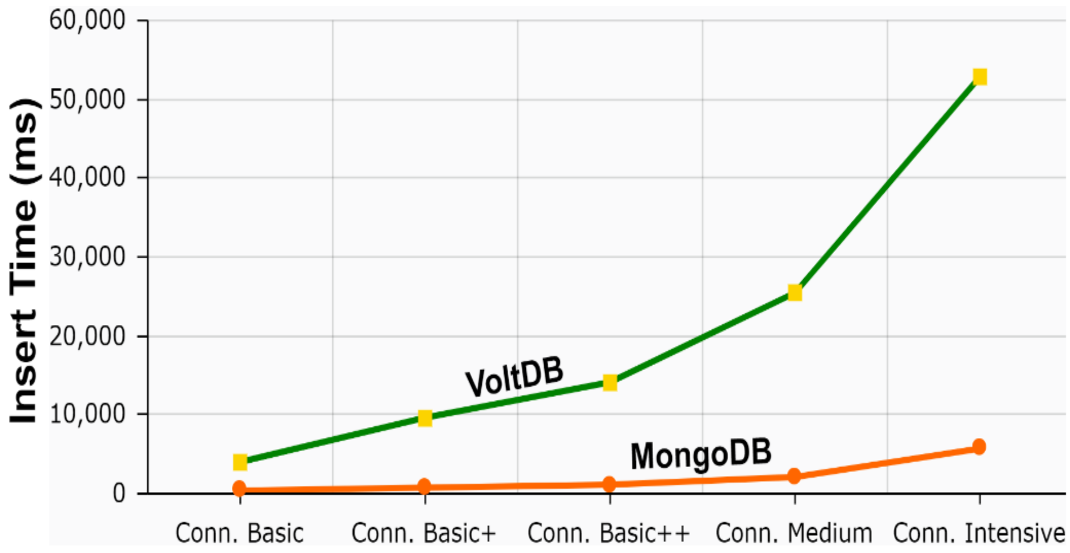


Fig 8. Graph of comparison of time of execution of insertion for both methods at a variable number of connections

It was found that for the transaction, the VoltDB update shows an order of magnitude better results both in terms of absolute values (the time difference reached 37 times) and in terms of growth. Thus, the growth rate with increasing various load factors for VoltDB increases approximately 1.5-2 times, while in the case of MongoDB, for each higher load mode, the time can increase by an order of magnitude. The execution time for MongoDB depends on all load factors, and for VoltDB it depends only on the amount of data and the number of simultaneous connections.

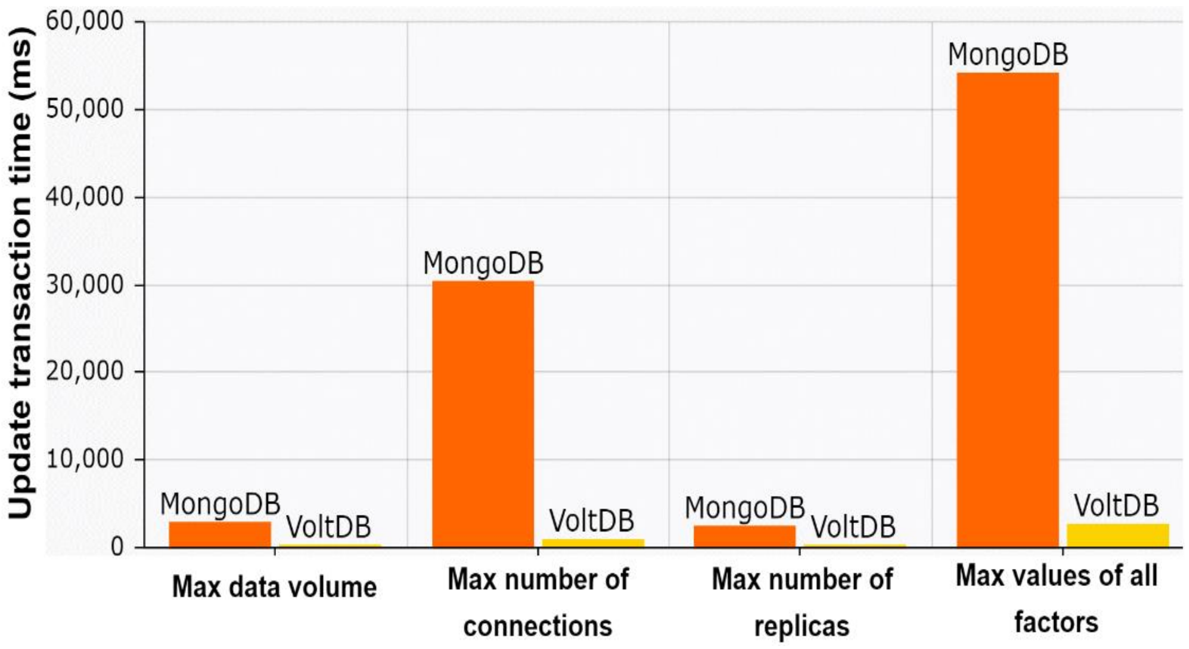


Fig. 9. Comparative histogram of the maximum execution time of the update transaction

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2021. No. 2 (16)

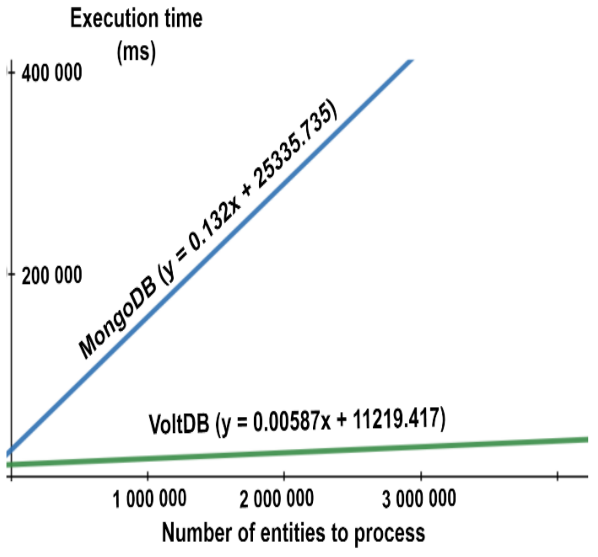
Therefore, you can immediately see a huge difference in the execution time of this transaction for the compared DBMS. With a variable amount of data and replicas, VoltDB updates in less than 120 ms. Given that during this transaction about 10% of the total number of goods are updated, you can determine the maximum bandwidth of each method (table 6). The table shows that VoltDB updates data much more efficiently, especially the gain compared to MongoDB is seen when increasing only the amount of data. The smallest gap between the methods is observed in the case when only the number of replicas increases.

Table 6. *The detected maximum number of entities that can be updated in one second*

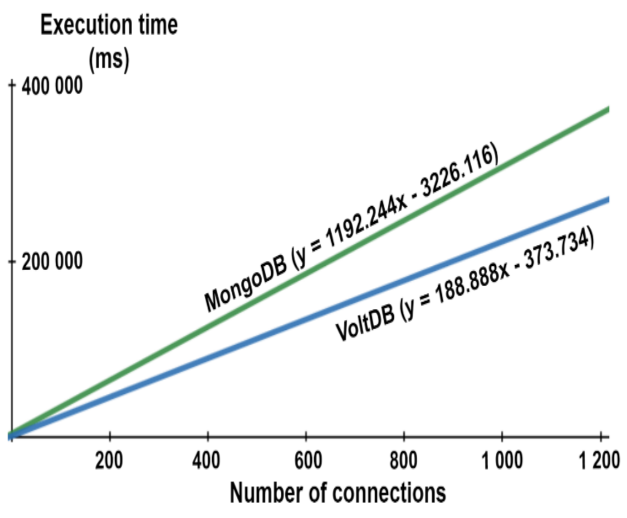
Types of load changes	MongoDB (number of entities)	VoltDB (number of entities)
When the amount of data changes	1824	93283
When changing the number of connections	7	248
When changing the number of replicas	85	1585
When all factors change	92	1927

Next, we consider the execution of a transaction with a request to delete all orders, the amount of which is in a certain range. This transaction also requires scanning the entire table / collection and performing a grouping operation to find the order amount through the price of its items. Grouping will take place for order items by the ID of the order to which they belong. After the grouping operation, the total price of the order will be calculated as the sum of the prices of all its elements.

Experiments have shown that the deletion transaction, as for the previous request, is performed an order of magnitude faster with VoltDB when changing any factor. Runtime for both DBMSs increases most slowly when the number of replica nodes in the cluster changes. The rate of increase of execution time when changing the amount of data and the number of connections will be better investigated by graphically displaying the corresponding linear regression functions. The following are graphs of the time dependency of the removal transaction from the amount of data (fig. 11a) and the number of simultaneous connections (fig. 10b) for both DBMSs.



a)



b)

Fig. 10. Functions of growth of execution time of transaction of removal at change (a - volume of data, b - number of connections)

As you can see from both graphs, for this query, VoltDB removal works much faster at any load in absolute values, and the time increase is not as rapid as in the case of MongoDB.

A number of complex transactions were also investigated in the paper. The first of these transactions contains two insert operations and two update operations. For this transaction, MongoDB worked an order of magnitude faster than VoltDB. The following is a histogram comparing the maximum execution time of this transaction when changing each load factor separately (fig. 11).

We see that the largest gap in time occurs when increasing the number of simultaneous connections, and the smallest when entering new nodes-replicas in the cluster. An even greater gap in execution time (almost 40 times) is observed with a simultaneous increase in the values of all load factors. Thus, for VoltDB at maximum load, the execution time is 675549.34 ms, and for MongoDB 16980.45 ms.

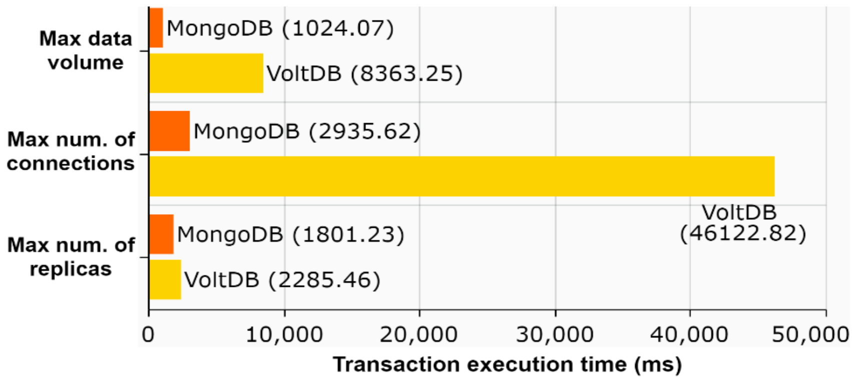


Fig. 11. Comparative histogram of the maximum execution time of transactions when changing various factors

We see that the largest gap in time occurs when increasing the number of simultaneous connections, and the smallest when entering new nodes-replicas in the cluster. An even greater gap in execution time (almost 40 times) is observed with a simultaneous increase in the values of all load factors. Thus, for VoltDB at maximum load, the execution time is 675549.34 ms, and for MongoDB 16980.45 ms.

The other complex transaction contains 6 insert operations and 1 update operation. MongoDB also executes this transaction an order of magnitude faster than VoltDB. Moreover, the execution time for VoltDB is affected by all load factors, while for MongoDB there is no characteristic dependence on the number of simultaneous connections. Comparison of the maximum execution time of this transaction when changing each load factor separately and all at the same time are given in table 7.

Table 7. *Experiments results*

Load factors	MongoDB, execution time (ms)	VoltDB, execution time (ms)	Difference
Max data volume	2213.16	40693.78	18.4 times
Max number of connections	1126.16	42720.91	37.9 times
Max number of replicas	1222.05	4492.47	3.7 times
Max value of all factors	25183.41	613669.34	24.4 times

It can be seen that MongoDB executes this transaction several tens of times faster than VoltDB. Moreover, the largest time gap for DBMS is manifested at the peak values of the number of connections, and the smallest at the maximum number of replicas. In general, MongoDB more efficiently captures the ACID of a transaction with a large number of write requests in a distributed DB.

Finally, consider the performance of read requests. Although they are not part of ACID transactions, this type of operation is most common in distributed DBs and its performance can significantly influence the choice of a method.

The first read request will consider a request to receive an order with its elements by the ID of this order. For both DBMSs at the DB level, index readings will be performed to check the effectiveness of the indexed search. As expected, for both methods, the amount of data in the database does not have a strong effect on the execution time of the search, because the logarithmic function grows very slowly. But it immediately becomes clear that MongoDB reads an order of magnitude faster than VoltDB - an average of 17 ms versus 125 ms. The number of replicas in the cluster also does not affect the search speed for both methods, because in this case, the interaction still takes place with only one node. On the other hand, the query execution time for both methods depends on the number of simultaneous connections, while MongoDB reads an order of magnitude faster and with increasing connections the time increases much more slowly than in the case of VoltDB (Table 8).

At simultaneous increase of indicators of all factors the approximately same tendency, as well as in the table above remains. Thus, it is clear that MongoDB is much more efficient at reading the index. This is primarily due to the fact that Mongo searches for an order in only one collection, and VoltDB searches the index in two tables: the order and its elements. As you can see, the difference in time differs by an order of magnitude, and for the maximum load in terms of connections, in general, by several orders of magnitude. This once again proves the stability of MongoDB with a large number of users, as well as its advantages when performing an index read operation.

Table 8. *The results of measuring the execution time of the read request when changing the factor of the number of connections*

Regimes	MongoDB (ms)	VoltDB (ms)
Basic	15.06	123.68
Basic+	17.87	146.38
Basic++	17.48	179.38
Medium	21.18	187.07
Intensive	25.53	10993.58

The latter we will analyze the implementation of a more complex request to obtain the three most popular categories of goods, which will include grouping, sorting and data limiting operations. When performing this query, both methods have a characteristic increase in runtime from the amount of data and the number of simultaneous connections. As in the previous case, the measurement results immediately show that MongoDB executes this request an order of magnitude faster than VoltDB. Below are graphs of the dependence of the query execution time for both methods on the increase in data volume (fig. 12) and the number of simultaneous connections (fig. 13).

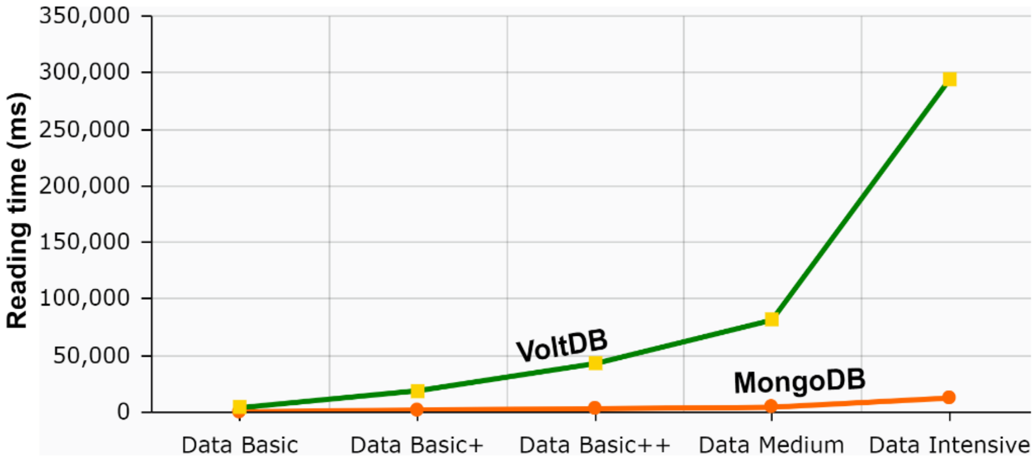


Fig. 12. Comparing of the category read time growth graphs based on data volume

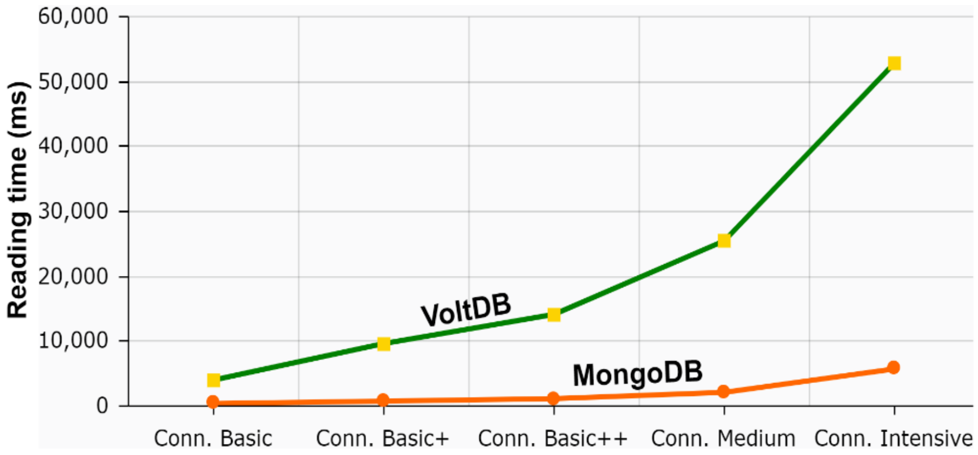


Fig. 13. Comparing of the category read time growth graphs based on number of connection

Both series of experiments give a similar result, the graphs clearly show that VoltDB executes this request much slower than MongoDB, and execution time increases many times faster. At a maximum load of all factors, VoltDB executes this request in 358923.22 ms, while MongoDB - in 80733.55 ms. That is, the difference at peak load is almost 4.5 times.

Therefore, based on the measurements made, it can be concluded that MongoDB performs a read request with grouping, sorting and limiting operations is several times more efficient than VoltDB.

As a result of the experimental study, the main trends in the performance of the compared DBMS were identified in a specific situation, which allows us to make recommendations for their application. Consider the recommendations made on the basis of the resources consumed for both methods. The amount of resources consumed directly affects the hardware of the virtual machine on which the DB server is located, and therefore the price of this machine.

Experiments show that the VoltDB consumes less storage space and requires slightly less CPU time, while in any load mode it consumes several times more RAM. Therefore, less disk space usage by VoltDB allows you to choose this DBMS in cases where you need to save on the size of the permanent media of the virtual machine, namely:

- when creating applications in which the DB must store tens of gigabytes of data and, at the same time, such data is not processed very often or intensively, so the speed of their processing is not a priority (for example, in storage systems DB backups or outdated data low probability);
- when creating applications where the same machine stores data from other databases or just other files; also the processing speed should not be a priority for the application.

Also, VoltDB's lower CPU resource consumption allows you to choose this approach when one or more applications that require CPU and multithreaded computing are deployed on the same virtual machine with the DB server. Examples of such applications are computer vision systems, which require a lot of CPU time to solve the classification problem. With this approach, you can save on CPU resources, which are the most expensive.

On the other hand, due to less RAM consumption, it is more appropriate to use MongoDB in the following cases:

- creating applications that require a lot of RAM to run continuously, but host one or more applications on the same virtual machine (for example, in systems that use different levels of cache stored in RAM for faster response , such as web servers that cache the client's response to expedite the response);

- creating applications where other repositories that require a large amount of RAM (for example, in In-Memory DB, such as Redis or Memcached) must be located on the same virtual machine.

All this will save on RAM, which is also quite expensive to rent.

Further on let's consider the recommendations developed based on the average execution time of various queries and transactions.

Based on the results of experiments, the implementation of distributed ACID transactions using MongoDB is more appropriate in the cases:

- creation of applications where read operations take precedence over other write operations (for example, in various analytical systems, data processing systems, online stores, search engines, which use such complex queries as indexed search); because MongoDB executes both simple index read requests and complex grouping, sorting, and limiting queries an order of magnitude faster, it is therefore more performance-efficient for this type of application;

- creation of applications where insert operations prevail over update and deletion operations, which is typical of data collection systems, which are then analyzed; usually, in such situations, the initial data is processed, which no longer changes (for example, in centralized data logging systems, control systems for various environmental metrics using the Internet of Things, ticket booking systems);

- creation of applications where within the ACID transaction many entries for insertion, updating or removal should be performed (for example, in systems with complex transactions, such as, bank applications with the function of transfer of funds, applications for construction of various graphic objects and charts numerical data, as well as applications with the function of testing and automated assessment of user knowledge); because MongoDB executes and fixes transactions with several separate requests faster on all nodes, so the more of these requests, the greater the difference in the time of its processing of transactions compared to VoltDB);

- creation of applications where the speed of any data change operations under high load from users is critical (for example, in e-commerce systems where there is a peak load of users during the holidays or in online competition systems, where to participate at a certain time hundreds of users must connect at once.); because MongoDB handles many concurrent connections very efficiently for most data change operations, MongoDB executes most of these requests faster than VoltDB; and even for such transactions, which are still faster in VoltDB, the gap with MongoDB is still smaller than for other load factors.

On the other hand, using VoltDB to implement distributed ACID transactions is more efficient in cases:

- creation of applications where update operations outperform other performance-critical and execution-critical operations (for example, in shared document editing systems, where multiple users simultaneously and continuously edit the same document, as well as systems for financial exchanges , where exchange rates and stock prices of companies are constantly updated); VoltDB, as measurements have shown, executes this type of transaction an order of magnitude faster than MongoDB;

- creation of applications where the deletion operation is performed with the same frequency as the insertion operation, or where the speed of data deletion is very important (for example, in systems of transmission of secret values through one-time pages, where they must be deleted immediately after the first reading, and in systems with the display of various data and metrics in real time, where obsolete data should be deleted as soon as possible so as not to take up disk space);

- creation of applications where a large number of replicas are expected from clusters (for example, in hotel reservation systems around the world, where DB servers are located in different geographical areas); The recommendation is due to the fact that VoltDB showed that the factor of the number of replica nodes in the cluster does not affect the speed of most queries, and if some queries are still characterized by an increase in time with increasing number of replicas, this growth is minimal and fluctuates around a few percent.

However, in situations where the method of implementing distributed ACID transactions is chosen before the start of application development and it is not known which load factor will be a priority and which types of requests will predominate in the application, a more reliable option is MongoDB. After all, for this DBMS with a simultaneous increase in all load factors, the execution time of most requests and transactions is less than for VoltDB.

Conclusions and prospects for further development

In terms of performance, a study of methods for implementing distributed ACID transactions, namely the built-in capabilities of DBMS MongoDB and VoltDB. A series of experiments was performed to measure the resources consumed and the execution time of various queries and transactions.

To perform the experiment, DB structures were designed for the relational and BSON models, as well as transactions to these databases. The experiments were performed with increasing load, which allowed to compare not only the absolute values of the metrics, but also their trends. The experiments used metrics on the execution time of queries and the resources required to execute those queries. Based on the analysis of these metrics, the comparative advantages and disadvantages of each approach were identified; analyzed what load factors affect specific types of requests and resources used.

Thus, the study provided a complete picture of the performance of both methods during different types of workload, based on which the main recommendations for the effective use of a method for a particular type of application were formulated.

References

1. Tamer Özsu, M. (2020), *Principles of Distributed Database Systems*, Springer International Publishing, 674 p.
2. Maran, M. M., Paniavin, N. A., Poliushkin, I. A. (2020), "Alternative Approaches to Data Storing and Processing", *V International Conference on Information Technologies in Engineering Education (Inforino)*. DOI: https://doi.org/10.1109/inforino48376.2020.9111708
3. Blokdyk, G. (2018), *ACID Transactions Second Edition*, 5STARCooks, 282 p.
4. Kemme, B., Peris, R. J., Patiño-Martínez, M. (2010), *Database Replication (Synthesis Lectures on Data Management)*, Morgan and Claypool Publishers, 154 p. DOI: https://doi.org/10.2200/S00296ED1V01Y201008DTM007
5. Moniruzzaman, A. B. M., Hossain, S. A. (2012), "NoSQL Database: New Era of Databases for Big data Analytics – Classification, Characteristics and Comparison", *International Journal of Database Theory and Application*, No. 4, P. 1.
6. Kuzochkina, A., Shirokopetleva, M., Dudar, Z. (2018), "Analyzing and Comparison of NoSQL DBMS", *International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T)*, P. 560–564. DOI: https://doi.org/10.1109/INFOCOMMST.2018.8632133
7. Sahatqija, K., Ajdari, J., Zenuni, X., Raufi, B., Ismaili, F., (2018), "Comparison between relational and NOSQL databases", *41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, P. 216–221. DOI: https://doi.org/10.23919/mipro.2018.8400041
8. Győrödi, C. A., Dumşe-Burescu, D. V., Zmaranda, D. R., Győrödi, R. Ş., Gabor, G. A., Pecherle, G. D. (2020), "Performance Analysis of NoSQL and Relational Databases with CouchDB and MySQL for Application's Data Storage", *Applied Sciences*, No. 10 (23), P. 8524. DOI: https://doi.org/10.3390/app10238524
9. Sadalage, P., Fowler, M. (2012), *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence, 1st Edition*, Addison-Wesley Professional, 192 p.
10. Chodorow, K. (2016), *MongoDB: The Definitive Guide: Powerful and Scalable Data Storage, 3rd Edition*, O'Reilly Media, 514 p.
11. Palanisamy, S., Suvitha Vani P. (2020), "A survey on RDBMS and NoSQL Databases MySQL vs MongoDB", *International Conference on Computer Communication and Informatics (ICCCI)*. DOI: https://doi.org/10.1109/iccci48352.2020.9104047
12. Pavlo, A., Aslett, M. (2016), "What's Really New with NewSQL?", *SIGMOD Record*, Vol. 45(2), P. 45–55. DOI: https://doi.org/10.1145/3003665.3003674
13. Astrova, I., Koschel, A., Wellermann, N., Klostermeyer, P. (2021), "Performance Benchmarking of NewSQL Databases with Yahoo Cloud Serving Benchmark", *Proceedings of the Future Technologies Conference (FTC) 2020, Vol. 2. FTC 2020. Advances in Intelligent Systems and Computing, Vol. 1289*, Springer, Cham. DOI: https://doi.org/10.1007/978-3-030-63089-8_17
14. Bhiri, S., Gaaloul, K., Perrin, O., Godart, C., (2005), "Overview of Transactional Patterns: Combining Workflow Flexibility and Transactional Reliability for Composite Web Services", *In: van der Aalst W.M.P., Benatallah B., Casati F., Curbera F. (eds) Business Process Management. BPM 2005, Lecture Notes in Computer Science*, Vol. 3649, Springer, Berlin, Heidelberg. DOI: https://doi.org/10.1007/11538394_37
15. Priya, M., Kalpana, R. (2017), "Distributed and Parallel Processing of Location based spatial query with Approximate Transformation", *Ninth International Conference on Advanced Computing (ICoAC)*, P. 334–338. DOI: https://doi.org/10.1109/ICoAC.2017.8441297
16. Gomes, C., Borba, E., Tavares, E., Meuse Nogueira de O. Junior (2019), "Performability Model for Assessing NoSQL DBMS Consistency", *IEEE International Systems Conference (SysCon)*. DOI: https://doi.org/10.1109/syscon.2019.8836757

Received 27.05.2021

Відомості про авторів / Сведения об авторах / About the Authors

Мазурова Оксана Олексіївна – кандидат технічних наук, доцент, Харківський національний університет радіоелектроніки, доцент кафедри програмної інженерії, Харків, Україна; email: oksana.mazurova@nure.ua; ORCID: https://orcid.org/0000-0003-3715-3476.

Мазурова Оксана Алексеевна – кандидат технических наук, доцент, Харьковский национальный университет радиоэлектроники, доцент кафедры программной инженерии, Харьков, Украина.

Mazurova Oksana – PhD (Engineering Sciences), Associate Professor, Kharkiv National University of Radio Electronics, Associate Professor of the Department of Software Engineering, Kharkiv, Ukraine.

Набока Артем Олександрович – Харківський національний університет радіоелектроніки, магістр спеціальності 121 - Інженерія програмного забезпечення, Харків, Україна; email: artem.naboka@nure.ua; ORCID: https://orcid.org/0000-0003-2178-7984.

Набока Артем Александрович – Харьковский национальный университет радиоэлектроники, магистр специальности 121 – Инженерия программного обеспечения, Харьков, Украина.

Naboka Artem – Kharkiv National University of Radio Electronics, Master of Specialty 121 - Software Engineering, Kharkiv, Ukraine.

Широкопетлєва Марія Сергіївна – Харківський національний університет радіоелектроніки, старший викладач кафедри програмної інженерії, Харків, Україна; email: marija.shirokopetleva@nure.ua, ORCID: https://orcid.org/0000-0002-7472-6045.

Широкопетлева Мария Сергеевна – Харьковский национальный университет радиоэлектроники, старший преподаватель кафедры программной инженерии, Харьков, Украина.

Shirokopetleva Mariya – Kharkiv National University of Radio Electronics, Senior Lecturer of the Department of Software Engineering, Kharkiv, Ukraine.

ДОСЛІДЖЕННЯ МЕТОДІВ РЕАЛІЗАЦІЇ РОЗПОДІЛЕНИХ ACID ТРАНЗАКЦІЙ ЗА ТЕХНОЛОГІЄЮ РЕПЛІКАЦІЇ

Сьогодні бази даних є невід’ємною частиною більшості сучасних застосувань, призначених для зберігання великих обсягів даних та звертань від багатьох користувачів. Для рішення бізнес-задач в таких умовах бази даних масштабуються, найчастіше горизонтально на декількох фізичних серверах з використанням технології реплікування. При цьому багато бізнес-операцій потребують реалізації транзакційності з дотриманням ACID-принципів:. Для реляційних СУБД, які традиційно підтримують ACID транзакції, горизонтальне масштабування не завжди ефективно через обмеження самої реляційної моделі. Отже існує прикладна проблема ефективної реалізації ACID транзакцій для горизонтально розподілених баз даних. **Предметом** дослідження є методи реалізації ACID транзакцій в розподілених базах даних, що створено за технологією реплікування. **Мета** роботи – підвищення ефективності реалізації ACID транзакцій для горизонтально розподілених баз даних. В роботі вирішуються наступні **завдання**: аналіз та вибір найбільш актуальних методів реалізації розподілених ACID транзакцій; планування та експериментальне дослідження методів реалізації ACID транзакцій на прикладі використання NoSQL СУБД MongoDB та NewSQL СУБД VoltDB; заміри метрик продуктивності використання цих методів та формування рекомендації щодо їх ефективного використання. Використовуються такі **методи**: системний аналіз; методи проектування реляційних баз даних та їх об’єктів; методи оцінки продуктивності баз даних. Отримано наступні **результати**: проведено експериментальні виміри часу виконання типових розподілених транзакцій для предметної області електронної комерції, а також заміри кількості ресурсів, що необхідні для їх виконання; виявлено тренди продуктивності виконання таких транзакцій, сформовані рекомендації щодо методів, що досліджувалися. Отримані результати дозволили скласти функції залежності розглянутих метрик від параметрів навантаження. **Висновки**: були виявлені сильні та слабкі сторони реалізації розподілених ACID транзакцій за допомогою СУБД MongoDB і VoltDB. Запропоновано практичні рекомендації щодо ефективного використання даних систем для різних типів додатків з урахуванням споживаних ресурсів та типів запитів.

Ключові слова: розподілена база даних; транзакція; продуктивність; ACID; NOSQL; NEWSQL; MongoDB; VoltDB.

ИССЛЕДОВАНИЕ МЕТОДОВ РЕАЛИЗАЦИИ РАСПРЕДЕЛЕННЫХ ACID ТРАНЗАКЦИЙ ПО ТЕХНОЛОГИИ РЕПЛИЦИРОВАНИЯ

Сегодня базы данных являются неотъемлемой частью большинства современных приложений, предназначенных для хранения больших объемов данных и обращений от большого количество пользователей. Для решения бизнес-задач в таких условиях базы данных масштабируются, чаще всего горизонтально на нескольких физических серверах с использованием технологии репликации. При этом многие бизнес-операции требуют реализации транзакционности с соблюдением ACID-принципов. Для реляционных СУБД, которые традиционно поддерживают ACID транзакции, горизонтальное масштабирование не всегда эффективно из-за ограничений самой реляционной модели. Поэтому существует прикладная проблема эффективной реализации ACID транзакций для горизонтально распределенных баз данных. **Предметом** исследования являются методы реализации ACID транзакций в распределенных базах данных, созданных на основании технологии репликации. **Цель** работы – повышение эффективности реализации ACID транзакций для горизонтально распределенных баз данных. В работе решаются следующие **задачи**: анализ и выбор наиболее актуальных методов реализации распределенных ACID транзакций; планирование и экспериментальное исследование методов реализации ACID транзакций на примере использования NoSQL СУБД MongoDB и NewSQL СУБД VoltDB; замеры метрик производительности использования этих методов и формирование рекомендации по их эффективному использованию. Используются следующие **методы**: системный анализ; методы проектирования реляционных баз данных и их объектов; методы оценки производительности баз данных. Получены следующие **результаты**: проведены экспериментальные измерения времени выполнения типовых распределенных транзакций для предметной области электронной коммерции, а также замеры количества ресурсов, необходимых для их выполнения; определены тренды производительности выполнения таких транзакций; сформированы рекомендации по исследуемым методам. Полученные результаты позволили найти функции зависимости рассмотренных метрик от параметров нагрузки. **Выводы**: были выявлены сильные и слабые стороны реализации распределенных ACID транзакций с помощью СУБД MongoDB и VoltDB. Предложены практические рекомендации относительно целесообразности использования данных систем для различных типов приложений с учетом потребляемых ресурсов и типов запросов.

Ключевые слова: распределенная база данных; транзакция; производительность; ACID; NOSQL; NEWSQL; MongoDB; VoltDB.

Бібліографічні описи / Bibliographic descriptions

Мазурова О. О., Набока А. О., Широкопетлєва М. С. Дослідження методів реалізації розподілених ACID транзакцій за технологією реплікації. *Сучасний стан наукових досліджень та технологій в промисловості*. 2021. № 2 (16). С. 19–31. DOI: https://doi.org/10.30837/ITSSI.2021.16.019

Mazurova, O., Naboka, A., Shirokopetleva, M. (2021), "Research of ACID transaction implementation methods for distributed databases using replication technology", *Innovative Technologies and Scientific Solutions for Industries*, No. 2 (16), P. 19–31. DOI: https://doi.org/10.30837/ITSSI.2021.16.019