Theoretical and Applied Informatics
ISSN 1896-5334
Vol. 25 (2013), no. 3-4
pp. 223 – 238
DOI: 10.2478/thai-2013-0016

Relative and non-relative databases performance with an Android platform application

PAWEŁ BUCHWALD, MACIEJ ROSTAŃSKI, ARKADIUSZ JURASZ

Wyższa Szkoła Biznesu w Dąbrowie Górniczej
Zygmunta Cieplaka 1c
Dąbrowa Górnicza, Poland

Received 3 April 2013, Revised 27 October 2013, Accepted 2 December 2013

Abstract: This article gives an introduction to NoSQL movement and how it can help in creating modern application trends of Web 2.0 by its scalability and performance. To achieve the objectives of this work, three types of databases have been tested to see which of them will be the best to work with mobile devices working on Android system. All received results were presented and commented.

Keywords: Relative and non-relative databases, android

1. Introduction

Cooperation mechanisms of database systems are an important part of operating systems for mobile devices. The relationship between mobility solutions and databases can be seen in the operating system Android. Android and iOS are currently the most popular operating system for mobile devices such as tablets and smartphones. The Android operating system uses SQL Lite database as an efficient way to store user's data such as text messages, contact list or the list of recent calls. Due to the extensive use of a relational database, APIs available for Android provide a high level functions that allow direct manipulation of the data. This approach enables mobile application developers to implement the business logic layer included in the application and data access layer in the high-level language such as Java. In simple cases, this allows for the resignation of the SQL language.

Direct access to the database is characteristic of the client-server model and is used when the database is placed on the same mobile device as the application. In other cases, a more useful model for data access is three-layered architecture. This architecture allows the use of business logic in the server-side application. The concept of using three-layer architecture to provide access to the remote database is shown in Fig. 1.

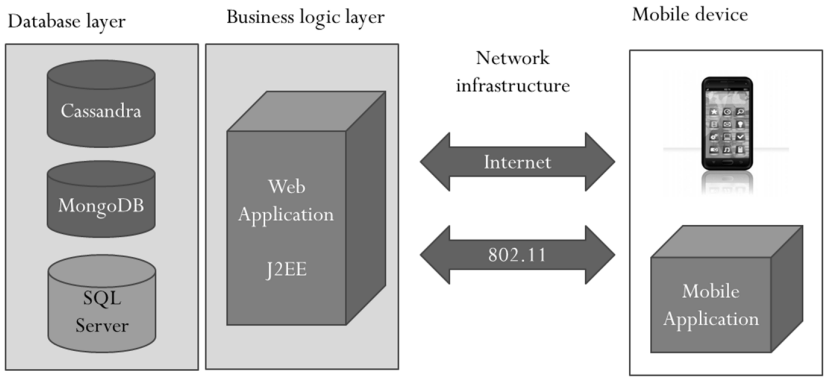

Fig 1. Three-layer architecture and access to databases.

The goal of this research was to check which type of database would be the best type for use with mobile devices. In this context, the best means the fastest, because the waiting time to complete the task by the database will affect the amount of used resources as for the battery power this plays a significant role. To achieve this goal two applications were created. The first was the web application and the other was mobile application working under Android system, which in the third quarter of 2012 had a 75% market share, that means that application will run on the most popular mobile system.

Communication with the database is implemented using an intermediate web application installed on a web server. This application via HTTP protocol allows users to communicate with the database from a mobile application. Designers creating solutions in Java language often use JSON notation. One can also use SOAP and Web Services supported by Microsoft. Android has a large set of libraries that allows for building solutions for network communication. With the availability of a network API libraries, designers can use the REST model as a way to exchange data between the mobile application and the intermediate layer. The use of middleware to communicate with the database allows to change the data storage layer without modifying the mobile application.

This paper is organized as follows: in order to present relevant research results, the discussion on relative and non-relative databases is presented; NoSQL models are described, as well as corresponding examples. Next, the testing application and research topology and environment is presented, with specific data on devices used for testing. The research results are then shown, described and analyzed, followed by final conclusions.

2. Relative and non-relative databases

The research was carried out on three types of databases. The first database is well known for many years - a relational database, and as its management system has been selected MS SQL Server 2005, and two quite new types are based on NoSQL model.

For the purpose of the research a simple database was developed in the relational model, called "big" which consisted of only one table, which in turn contained two columns (Fig. 2).

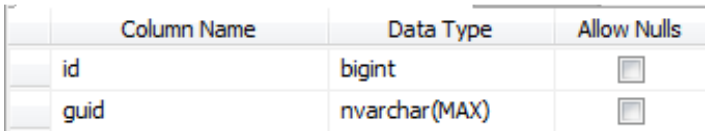

Fig. 2 Table_1 scheme (Source: SQL Server Management Studio).

Id column was used to identify each row in the table, while the *guid* column was used to store a randomly generated string.

2.1. NoSQL models

NoSQL (Not Only SQL) is a term that defines the database management system. This system however is not based on the traditional relational model. The data in NoSQL model does not require specific patterns and is lacking any joints. NoSQL movement is a response to changing needs in terms of speed and scalability of databases. Those needs are an outcome of beginning of a Web 2.0 trend and the emergence of a large number of social networking sites (one of them is Facebook with over 700mln active users [2]). In this kind of web applications even expanded server farm becomes not sufficient enough to be available to a wider range of users. Once it was realized that in the near future, the relational model cannot meet the demands posed by the large amount of data, works started to create independent systems, which are collectively known as NoSQL. Therefore, contrary to the relational database, there is no single data model. The following are the most typical models, however the amount of used models in the current NoSQL is much greater:

- Key-value database: In simple terms this is one table with two columns of text, key and value. In its simplicity, this model has an incredible advantage when it comes to reading and writing operations – it is extremely fast.
- Column database: Instead of the traditional storing data in rows data are saved in columns. Due to the fact that the columns are stored in the same data type they can be compressed by using more efficient algorithms. An example of database that applies this column model is Cassandra.
- Document-oriented database: In this model row concept was replaced by a document that contains the key-value pairs. This approach allows for accurate reflection of the actual data. Such a model can be found in the MongoDB system.
- XML Databases: During developing of multiple database systems there was a need to exchange data between them. For this task perfectly fits XML, which is one of the standards when it comes to web services. For this reason were formed database solutions that store data in XML. Examples of such solutions can be Xindice or Apache XML DB. These solutions are enhanced with a new mechanism for data querying (XQuery or XPath).

- Object-oriented database: Is the answer to the problem which is the object-relational mapping. The problem is that it is not always easy to move the relational model to an object model. Through the introduction of skeletal mapping (object-relational) such as Hibernate, it became possible to hide the symptoms of the problems associated with object-relational mapping, but it didn't not solve the problem in further investigation. Therefore new solution was created, where instead of the rows, objects are stored in the database. The biggest downside of this solution is its poor performance. There are other constraints, such as lack of query optimization and lack of support for data exchange between different languages. [1]

2.2. Cassandra and MongoDB as NoSQL examples

In this article, focus was made on two databases working on a NoSQL model which are Cassandra and MongoDB. Cassandra is built on the column database model, the data are not stored in rows but in columns. This was explained in relation to the database ("big") presented earlier in this article (section 2). Before this, one should be aware of what elements made up this data model.

At the same beginning there is Column (Fig. 3), which consists of key-value pairs and timestamp fields, which saves time event (for clarity of diagrams, this field was omitted):

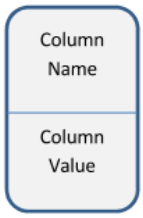

Fig. 3. Column.

Another element of the aggregator of column is a Super Column (Fig. 4), which is an array with n columns:

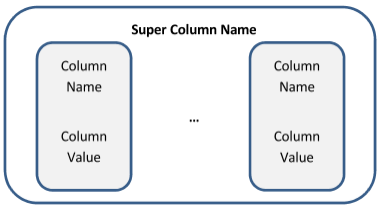

Fig. 4. Super Column.

The next element is the Column Family (Fig. 5), it is a container of columns sorted by their name, the family and the column is sorted by row key (row_key):

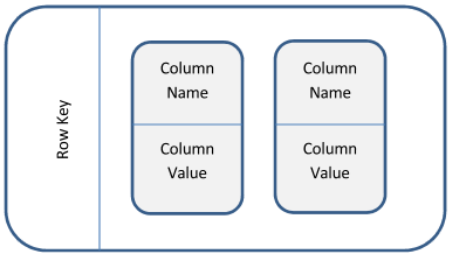

Fig. 5. Colum Family

For storing Super Columns there is a Super Column Family (Fig. 6).

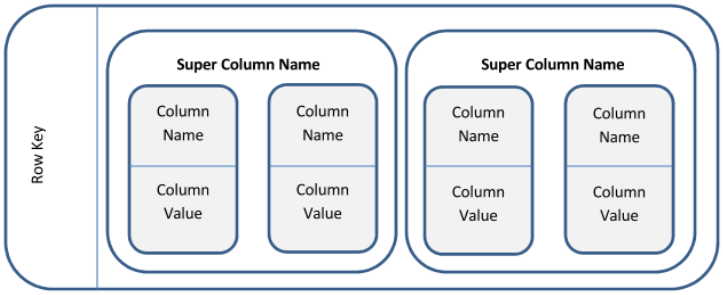

Fig. 6. Super Column Family (Source: [6]).

The highest element in this scheme is the keyspace, which is a container for Column Families. According to the relational database model presented earlier ("big") in Column database model looks as follow (Fig. 7)

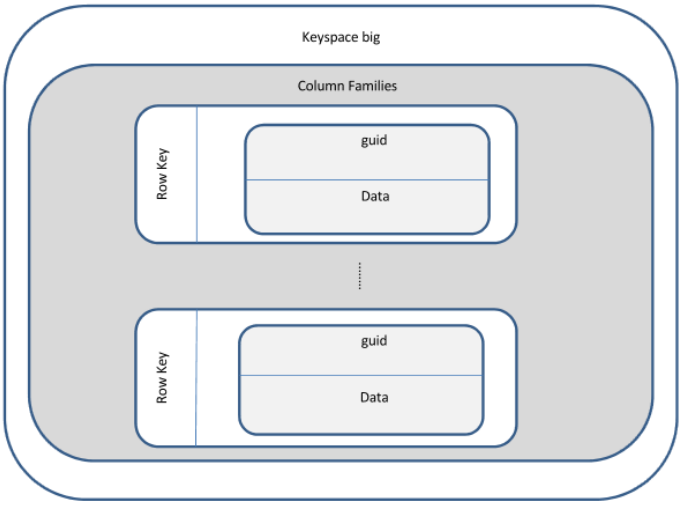

Fig. 7. Database "big" in Column model

Last database researched in this paper is MongoDB database, which is based on document-oriented model, characterized by a high scalability and performance. This database does not have a well-defined structure of the supported data because the data are stored as BSON documents in the form - Binary JSON (Java Script Binary Object Notation it is a subset of JavaScript [2]). BSON is a binary decoded JSON, the write operation use smaller memory resources than standard JSON. Example row from "big" database in JSON notation would look as follows:

```
{
        "_id": ObjectId("501bcb98076f93723f4a9b3a"),
        "guid": "ce877db0-8b56-4d16-9705-a8581cc64d60"
}
```

3. Testing environment

The equipment, on which the research was carried out, was built of three cores (AMD Athlon II X3 455) with clock frequency of 3.2 GHz. As for RAM, there were two cards, each 2GB size. The hard drive was connected with 3 Gbps SATA II interface which gives 370 Mbytes/s effective throughput.

Equipment also included smartphone with Android (version of the system is 2.3.3 - Gingerbread). For this, the HTC Desire was chosen, equipped with a Snapdragon processor clocked at 1GHz, and 576MB of RAM. Network infrastructure consisted of three main components, which were Server S, router R and smartphone A. Fig.8 shows simple topology used in test LAN.

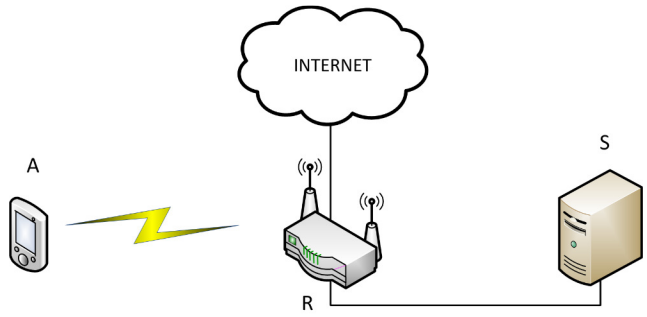

Fig. 8. Topology of the testing network (own work).

The application designed for testing was created in Java language. The solution has been tested with the use of a smartphone running under the Android operating system. The goal of the application was to initiate database operations, and to perform measuring and collecting response times. This approach enabled authors to find the total time data processing, together with the time of acquisition results to the mobile device.

Communication with the intermediate layer was achieved through the use of HttpClient class, which allowed the transfer request to the component running in the business logic layer. Business logic layer has been implemented as a Web Service. The

use of an intermediate component allowed researchers to avoid modifying the client application when it was necessary to use other data storage system.

4. Research results

The main objective of research was to investigate the execution time of basic operations on databases (retrieve, add, modify, and disposal) with a different number of records, and all of this was done from an Android application that was used as a "thin client". The device with the application creates a client that can communicate with server; this kind of pattern is known as a client-server architecture. The advantage of this solution is that the server is loaded with all the work that makes the client hardware requirements very minimal. The disadvantages are all kinds of delays due to network latency, which was included in these results. Research subjects are access times and execution times of operations on relational and non-relational databases, working on the same server and controlled from mobile device. As mentioned in the introduction, to carry out the research it was required to create two applications. One playing a role of the server that is taking upon itself the entire cost of the test operations, the second one provides the way to control these operations. In results there will be two types of the observed times. The first will determine how long the operation was performed on the server (server time) and second will determine a total operation time of the request (End-To-End time). Each measurement consisted of three attempts after which the results were recorded as the average and base one these results analyze was made.

Research took place in three categories with different number of elements, and in an additional category, designed to illustrate the performance of the actual databases through a very large number of elements. The number of items for each category is shown in Table 1.

Name	Number of elements
Category 1	1 000
Category 2	10 000
Category 3	20 000
Category 4	100 000

Tab.1. Category list in which the tests were performed

For the fourth category, the focus was only on database performance times, there were no attempts to try send such large number element to mobile device.

4.1. Results for relational model (MS SQL Server 2005)

First, the research has been carried out for Microsoft SQL database server. The following are the results in the form of graphs for each of the four categories.

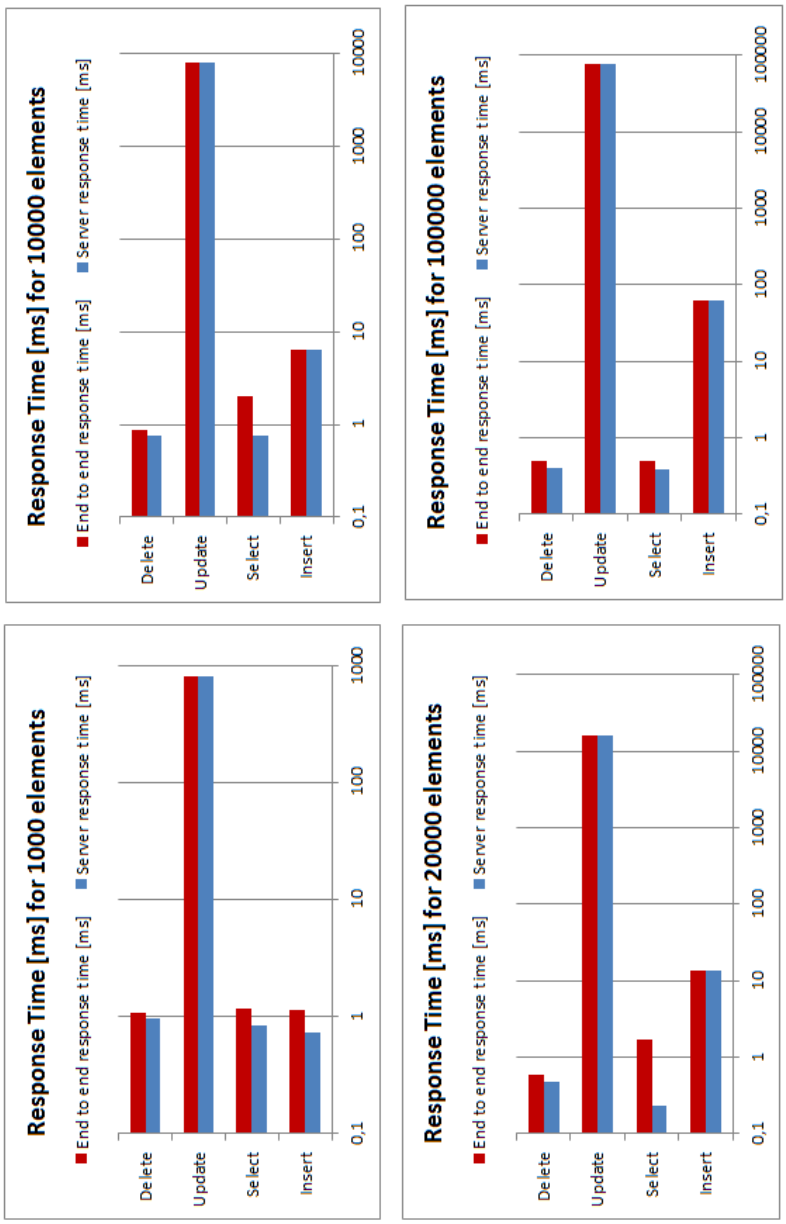

Fig. 9. Request time for SQL Server database in case of others rowset size.

From these results it can be concluded that the operation "Update" is the most time-consuming and it increases linearly. With the number of elements equal to 1 thousand, update took about 13 minutes, to 10 000 was already over 2 hours and for 100 000 -

more than 21 hours. The time of operation "Update" increased linearly in relation to the number of elements on which it is made. On the chart trend lines can also be seen, that show the forecast for the times between 20 and 100 thousand elements. One should also pay attention to the time of the "Delete" operation, which decrease together with an increasing number of elements.

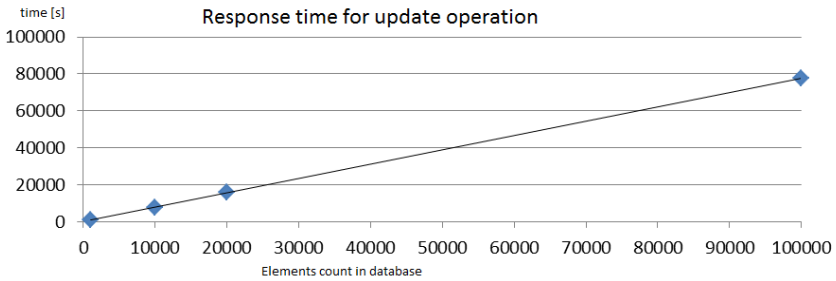

Fig. 10. Relation of response time for update operation to elements count in database

4.2. Results for column model (Cassandra)

Next test was made on Cassandra database. Below are the results for the four categories. In the case of Cassandra database operation "Update" is the most time-consuming. However the operation "Select" is surprisingly fast, where in the first three categories we can see the difference between the same operation performed on the database and the additional time needed for its message to be send and displayed by mobile application. For the fourth category, you do not see this effect because the server only sends information about the time of operation without results. Each of the other operations ("Insert" and "Delete") shows an increase of times together with increasing number of elements. Results of response times for Cassandra databases are shown on Fig 11.

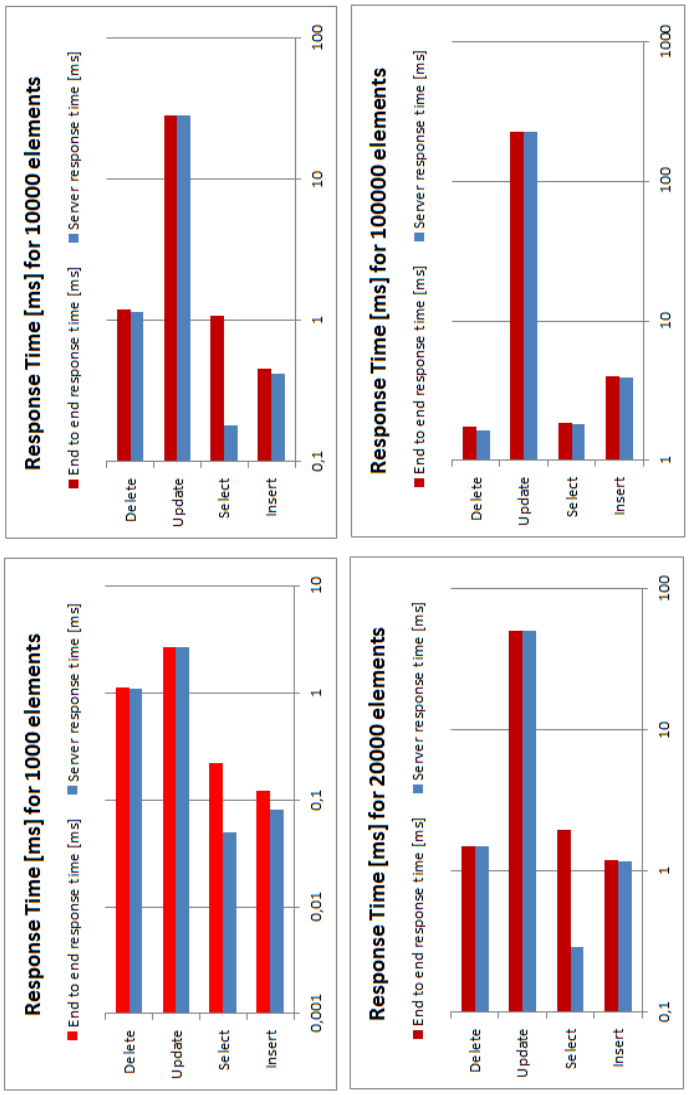

Fig.11. Response Times for Cassandra Database.

4.3. Results for document-oriented model (MongoDB)

The next section is devoted to the results measured for the last present database - MongoDB. For the presented results it is worth to notice that the operations "Select" and "Update" are performed with similar speed.

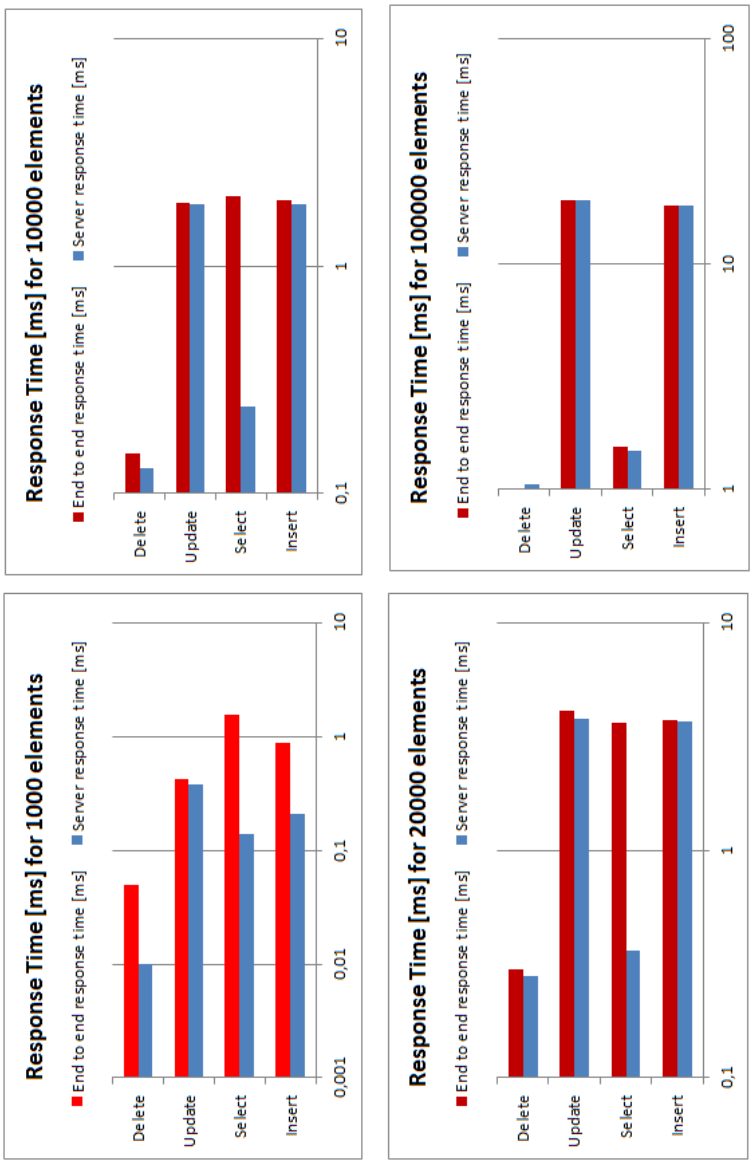

Fig. 12. Response time for MongoDB.

4.4. Results analysis

For the first category the most effective were family of NoSQL databases (Table 2). Cassandra was the best with the addition of new elements and their reading, but MongoDB has proven to be unsurpassed when it comes to updates and removing items. From the above data can also be seen that the difference of End-to-End and Server time

for the operation "Insert / Update / Delete" is similar to each other because they are performing similar task. But the situation is quite different for the operation "Select", where the difference felt by the user is about 1 sec. The reason could be the increased network latency, and also some extra time that was needed to receive all selected items.

The second class consisted of ten thousand elements and the results (Table 4.3) of this section are included in the following table:

		Server's response time [s]	End-to-End response time [s]
Ms SQL Server	Insert	0,74	1,14
	Select	0,84	1,16
	Update	815,61	816,62
	Delete	0,96	1,09
Cassandra	Insert	**0,08**	**0,12**
	Select	**0,05**	**0,22**
	Update	2,65	2,7
	Delete	1,08	1,11
MongoDB	Insert	0,21	0,88
	Select	0,14	1,55
	Update	**0,38**	**0,42**
	Delete	**0,01**	**0,05**

Tab. 2. Set of results for 1 000 elements

		Server's response time [s]	End-to-End response time [s]
Ms SQL Server	Insert	6,31	6,35
	Select	0,76	1,99
	Update	7907,02	7918,77
	Delete	0,77	0,86
Cassandra	Insert	**0,42**	**0,46**
	Select	**0,18**	**1,09**
	Update	28,64	28,69
	Delete	1,16	1,2
MongoDB	Insert	1,87	1,96
	Select	0,24	2,03
	Update	**1,87**	**1,9**
	Delete	**0,13**	**0,15**

Tab. 3. Set of results for 10 000 elements

In the second category scheme remains unchanged, the best times for data read and written are done by Cassandra, and removed and updated by MongoDB. Ms SQL Server can boast a better result in the removal of items than Cassandra, and that the operation "Select" on a larger number of components was faster. However, the biggest drawback is the update operation. Update of 10 thousand elements took over two hours while in MongoDB the same operation performed in 1.87 sec., which is a huge difference.

The third category which consists of twenty thousand elements, shows some changes. Looking at the results in the table below you can see that the operation "Select" for MS Server is done faster than any other, but "Update" operation still increases almost linearly. With twenty thousand items it took about four and a half hours to update them all. The fastest update and delete items is done using MongoDB.

The fourth category was created to examine the time required to perform an operation ordered by a much larger number of elements, which is one hundred thousand. As mentioned earlier, for this category transmission of elements was abandoned.

Additional test allowed to show that the greater number of elements has significant acceleration in MS Server database for "Select" and "Delete" operations. Unfortunately, the update operation does not continue to change its ratio in comparison to the number of elements.

		Server's response time [s]	End-to-End response time [s]
Ms SQL Server	Insert	13,51	13,65
	Select	**0,23**	**1,71**
	Update	16095,94	16097,69
	Delete	0,48	0,58
Cassandra	Insert	**1,18**	**1,21**
	Select	0,29	1,95
	Update	50,12	50,18
	Delete	1,49	1,52
MongoDB	Insert	3,71	3,74
	Select	0,36	3,65
	Update	**3,79**	**4,09**
	Delete	**0,28**	**0,3**

Tab. 4. Set of results for 20 000 elements

The fourth category was created to examine the time required to perform an operation ordered by a much larger number of elements, which is one hundred thousand. As mentioned earlier, for this category transmission of elements was abandoned.

		Server's response time [s]	End-to-End response time [s]
Ms SQL Server	Insert	63,37	63,8
	Select	**0,39**	**0,49**
	Update	77561,66	77562,99
	Delete	**0,4**	**0,49**
Cassandra	Insert	**3,96**	**4**
	Select	1,84	1,87
	Update	223,68	224,01
	Delete	1,66	1,74
MongoDB	Insert	18,16	18,19
	Select	1,48	1,54
	Update	**19,33**	**19,38**
	Delete	1,05	1,01

Tab. 5. Set of results for 100 000 elements

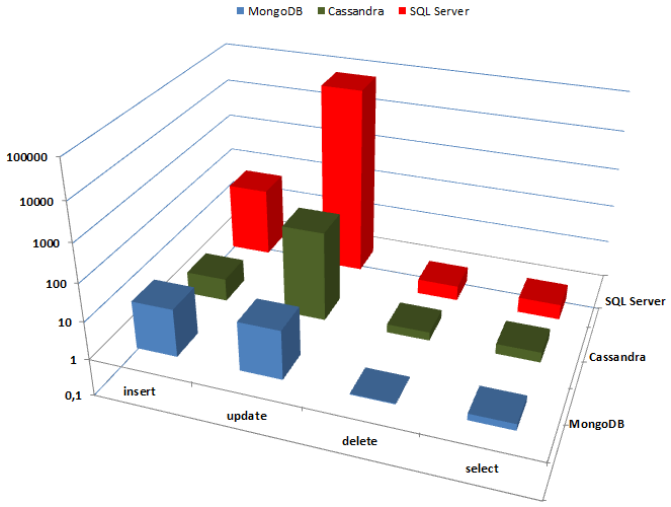

Fig. 13. Maximum Times of operations for tested databases

The Fig. 13 shows summary information about the maximum time of the select, insert, update and delete operations. Maximum response time was recorded for the SQL Server database during the update operation. SQL Server database is the only one among the tested solutions, providing transactional operations.

5. Conclusions

The research helped to show a significant difference in the performance of basic database operations for databases running on different models. The biggest surprise was the MongoDB database, which in most cases is characterized by the fastest times at renovation and disposal of records. The remaining operations were performed very close to the border with the best times in each category. The disadvantage of a NoSQL database is very weak support for the transaction. However, this is the result of one of the assumptions of non-relational databases, which says that work on the basis of the ACID (atomicity, consistency, isolation, durability) [2] is too restrictive. This problem does not occur in relational databases, which are based on this assumption.

References

[1] F. Chang et al, *BigTable: A Distributed Storage System for Structured Data*, Seventh Symposium on Operating System Design and Implementation, November 2006.

[2] M. Stonebraker and R. Cattell, *Ten Rules for Scalable Performance in Simple Operation datastores*, Communications of the ACM, June 2011.

[3] Hewitt, Eben. Cassandra: *The Definitive Guide* (1st ed.). O'Reilly Media. 2010.

[4] Capriolo, E., *Cassandra High Performance Cookbook* (1st ed.). Packt Publishing. 2011

[5] Skalski D., *NoSQL – nierelacyjne systemy baz danych*, Software Deweloper's Journal, 2011, nr 8

[6] http://www.javageneration.com