DBMS Performance Issues on a Single-Board Computer Raspberry Pi 3 Model B

Igor Lazarev[0000-0001-8033-4673] and Ilya Gosudarev [0000-0003-4236-5991]

ITMO University, St. Petersburg, Russia.
il5498@yandex.ru
goss@itmo.ru

Abstract. In this article, using the Raspberry Pi 3 Model B platform as an example, we consider the problem of performance of different DBMS in the process of using them to provide a single board computer as a web server. The existing studies do not reveal the database management system (DBMS) performance problems on single board computer platforms. A comparative analysis of the performance of different operations using MariaDB relational DBMS and the same operations using MongoDB NoSQL DBMS was conducted. The data for the analysis were obtained in the process of experiment, during which the performance of such operations on databases as sampling and updating was tested. The Node.js. platform was used as a testing application. Testing was carried out on a test stand that included a single board computer Raspberry Pi 3 Model B and Windows PC. Besides the DBMS, the web server performance analysis was conducted in conjunction with the file system. As a result of the testing, the real performance degradation was found to be compliant with the expected one and some interesting performance features of different DBMS were identified. As a result of the testing the conclusions were made about the possibility of using Raspberry Pi 3 Model B platform as an authentication web-server.

Keywords: Single-board computers, Raspberry Pi, Database Management Systems, MySQL, JSON files, MongoDB, Node.js

1 Introduction

At the moment, among the most important requirements for hardware and technical solutions are mobility, scalability, low operating costs and the possibility of customization. These qualities have led to the spread of single-board computers in a number of areas, including the routing of network requests, management of components of the Internet of Things, the deployment of software-controlled hardware and software solutions in the field.

The study showed that up to 85% of solutions based on single board computers tend to implement monolingual paradigm, mainly based on the programming language C. The monolingual platform is understood as the management of all or almost all aspects of the platform's operation using the same programming language. For example, a one programming language can be used to control sensors and LEDs on a computer as well as to process the data from these sensors and deliver them to the IoT client, e.g. via a web server. However, the evolution of JavaScript and the development of its ecosystem create reasons for the migration to this language as the basis of the monolingual platform. Refactoring of the existing code and practical experience allowed us to identify a number of related problems.

In the previous studies, one of the authors needed to ensure the performance of the web server when working with the database to avoid problems with data transmission to the client side and their reflection. In the course of this work relational DBMS MySQL was selected. This choice was due to the convenience of using the relational data model and the wide popularity of the database. At application deployment developed during work on final qualifying work on platform Raspberry Pi the problem of absence of the official distribution kit of server MySQL has been revealed. This fact revealed the need to replace the MySQL database with another one that supports the work on a single-board computer Raspberry Pi. DBMS such as MariaDB and MongoDB were considered as possible options. In addition to comparing the results of using the DBMS, a performance comparison was made when using JSON file stored in the file system as a database. This choice was conditioned by the necessity to check the ability of the file system or hardware data storage to avoid reducing the working speed.

MariaDB relational database surpasses MySQL performance by 3-5% due to improved query optimizer and many other performance related improvements, as well as complete drop-in-replacement MySQL 5.5, which is a strong argument for analyzing the performance of this particular database on the Raspberry Pi platform [1].

At the same time, the article A Comparison of Database Performance of MySQL and MariaDB with OLTP Workload v3.0. states that MySQL performance is higher than MariaDB performance when using approximately the same amount of system resources [5]. On the other hand, the article NVM Aware MariaDB Database System states that MariaDB has implemented a solution that increases the efficiency of using this DBMS on systems with non-volatile memory [6]. Also, the article by Yusuf Abubakar shows that MariaDB relational DBMS is the most productive for data reading operation among all considered relational DBMS such as PostgreSQL, MySQL and SQLite [7]. Thus, there are a lot of articles considering MariaDB relational DBMS performance that do not come to a final solution of the issue of positioning this DBMS in the relational DBMS performance rating.

There is also a question of the rationality of MongoDB use considering the disadvantages of this DBMS in the form of a large load on the system's RAM and the need to independently interrupt slow queries [2]. At the same time, the ability to easily scale MongoDB [4] can provide the use of the cluster of Raspberry Pi 3 Model B instances as the hardware part of the distributed database created by MongoDB No-SQL DBMS.

Since the study considers the DBMS performance on the Raspberry Pi 3 Model B platform, the use of the classical criterion of the number of operations for a certain period of time, such as in the article on the creation of the LinkBench benchmark [8], seems irrational due to the fact that single-board computers are worse than specialized data storage systems all performance parameters, the time spent on one operation of selecting or updating data is used as a comparison criterion for a DBMS.

Despite the presence of the conducted researches on the topic of comparing the performance of relational DBMS MySQL and NoSQL DBMS MongoDB [3] or other relational DBMS PostgreSQL and MongoDB [4], it can be noted that these researches did not consider the issue of DBMS data performance on single-board computer platforms.

2 Data mining

2.1 Test bench specification

The following DBMS testing platforms were selected: Windows PC and Raspberry Pi 3 Model B single board computer. These platforms are widely used in various areas of application development. While the PC has the best hardware features, Raspberry Pi 3 Model B is compact, low power consumption and low cost. The need for tests on a desktop PC was due to the possibility that a single-board computer Raspberry Pi 3 Model B might not have enough performance, so that the use of single-board computers to deploy web servers within the concept of monolingual programming using the JavaScript language would be irrational. Thus, it became necessary to determine an acceptable threshold for performance degradation. For this purpose, the hardware characteristics of desktop PC and Raspberry Pi 3 Model B were compared. SSD on Windows PC and MicroSD card on Raspberry Pi 3 Model B were used as a data storage. MicroSD card used as read-only memory in Raspberry Pi 3 Model B by default. A comparison of all components of the platforms used is shown in Table 1. As a result of the comparison, it was decided to establish an acceptable threshold performance reduction of 200%.

Table 1. Technical characteristics of two platforms

	PC	Raspberry Pi
Processor model	AMD A8-7410	Broadcom BCM2837
Clock frequency	2.2 GHz	1.2 GHz
Number of core	4	4
RAM capacity	8 GB (up to 16 GB)	1 GB
ROM capacity	930 GB	32 GB

Node.js and Express technologies were chosen as the stack of technologies used for testing various DBMS. The choice of these technologies is due to the fact that this study is part of a more global study of the possibility of using JavaScript programming language in IoT.

The performance of the database management system for various basic operations, such as sampling, insertion and modification, was selected as the benchmark for comparison. This criterion was chosen due to the need for fast data exchange between the DBMS and the server.

The process of experimenting to determine a suitable DBMS was organized as follows. Tests were developed to determine the timing of operations. In addition, data similar to the real data had to be generated. The data could be used to fill various database management systems, in two versions. In the first case, 20,000 records were used. This number is due number is due to the one of the author's previous researches, in which a database of about 20,000 unique people was used. Thus, the choice of this number allows to estimate the DBMS indicators that can be used to assess their performance in real life tasks. The second option contained 100,000 entries. The choice of such a number of records is based on the possibility of increasing the number of records in the existing database, according to experts' estimates, up to 100,000 in 3-5 years.

2.2 Relational DBMS MariaDB testing

To test the data access performance in relational DBMS MariaDB, the samples were made by the year of birth and by the surname. Such samples allowed to test the selection as a large set of values and a less one from the volume data array. The code fragment responsible for sampling small amount of data from relational DBMS MariaDB is shown in Figure 1.

```
const maria = require('mariadb/promise');
const format = require('mariadb').format;
const connection_options = require('../../config').connection_options;

async function select1FromMariaDB100k(){
    let connection = await maria.createConnection(connection_options);
    let [rows, fields] = await connection.execute('SELECT * FROM data100k WHERE name="Сергей Бобров"');
    connection.end();
}

select1FromMariaDB100k();
```

Fig. 1. Code fragments for sampling small amount of data from rDBMS MariaDB

Thus, in the case of small data set sampling, their filtration is performed by name, while in the case of large data set sampling the filtration is performed by the birth year. A sample code sample that selects a large amount of data is shown in Figure 2.

```
const maria = require('mariadb/promise');
const format = require('mariadb').format;
const connection_options = require('../../config').connection_options;

async function selectAnyFromMariaDB100k(){
    let connection = await maria.createConnection(connection_options);
    let [rows, fields] = await connection.execute('SELECT * FROM data100k WHERE yearOfBirth="1991"');
    connection.end();
}

selectAnyFromMariaDB100k();
```

Fig. 2. Code fragments for sampling a lot amount of data from rDBMS MariaDB

Also, the DBMS was tested for the data alteration speed. To perform this operation, we decided to use the method of updating the data of one column, if the data of another column correspond to the value strictly specified earlier. The code fragments responsible for this part of the testing are shown in Figure 3.

```
const maria = require('mariadb/promise');
const format = require('mariadb').format;
const connection_options = require('../../config').connection_options;

async function updateMariaDB100k(){
    let connection = await maria.createConnection(connection_options);
    let [rows, fields] = await connection.execute(`
UPDATE data100k SET address_city="Санкт-Петербург" WHERE yearOfBirth="1991"
    `);
    connection.end();
}

updateMariaDB100k();
```

Fig. 3. Code fragments for updating a lot amount of data in rDBMS MariaDB

2.3 NoSQL DMS MongoDB testing

The NoSQL DBMS MongoDB performance testing process is in general similar to the MariyaDB relational DBMS performance testing process, except for a few points, such as no need to format the SQL query to the DBMS and different DBMS connection implementation. Thus, unlike the MariaDB client implementation, the MongoDB client implementation, due to the fact that this DBMS supports queries to databases in JavaScript language, uses built-in functions together with the built-in MongoDB functions. The code that selects a small amount of data from the database is shown in Figure 4.

```
const MongoClient = require("mongodb").MongoClient;

async function select1FromMongoDB100k(){
    const client = new MongoClient("mongodb://localhost:27017/", { useNewUrlParser: true });
    await client.connect();
    const db = client.db('test');
    const cursor = db.collection('data100ks').find({ name: 'Сергей Бобров' });
    client.close();
}

select1FromMongoDB100k();
```

Fig. 4. Code fragments for sampling small amount of data from NoSQL DBMS MongoDB

It follows from the code that the selection of large and small amounts of data differ slightly both in queries to MariaDB DBMS and in queries to MongoDB DBMS. The key difference between the queries to these DBMS is the use of SQL language in the case of queries to MariaDB DBMS and the use of built-in functions in the case of queries to MongoDB DBMS. The code responsible for selecting a significant amount of data from the MongoDB is shown in Figure 5.

```
const MongoClient = require("mongodb").MongoClient;

async function selectAnyFromMongoDB100k(){
    const client = new MongoClient("mongodb://localhost:27017/", { useNewUrlParser: true });
    await client.connect();
    const db = client.db('test');
    const cursor = db.collection('data100ks').find({ yearOfBirth: '1991' });
    client.close();
}

selectAnyFromMongoDB100k();
```

Fig. 5. Code fragment for sampling a lot of data from NoSQL DBMS MongoDB

The procedure of updating data in the MongoDB DBMS differs significantly from the procedures considered earlier. Firstly, there are several methods of data updating in MongoDB first. They are given below in Table 2.

Table 2. Methods of updating data in MongoDB DBMS NoSQL [9]

Method	Description
updateOne()	Update a single document in a collection
updateMany()	Update multiple documents in a collection
replaceOne()	Replace a document in a collection with another document

Also, one of the operators in Update operation in MongoDB - $set - is used to perform the data update request. The full list of operators and description of their behavior is given below in Table 3.

Table 3. Operators using for updating data in MongoDB DBMS NoSQL [9]

Operator	Behavior
$currentDate	Sets the value of a field to the current date, either as a Date or a timestamp. The default type is Date.
$inc	Increments a field by a specified value.
$min	Updates the value of the field to a specified value if the specified value is less than the current value of the field. This operator can compare values of different types, using the BSON comparison order.

$max	Updates the value of the field to a specified value if the specified value is greater than the current value of the field. This operator can compare values of different types, using the BSON comparison order.
$mul	Multiply the value of a field by a number.
$rename	Updates the name of a field.
$set	Replaces the value of a field with the specified value.
$setOnInsert	If an update operation with upsert: true results in an insert of a document, then assigns the specified values to the fields in the document. If the update operation does not result in an insert, this operator does nothing.
$unset	Deletes a particular field.

Based on the above mentioned, the code was developed to update the data in the MongoDB DBMS, which is shown below in Figure 6.

```
async function updateMongoDB100k(){
    const client = new MongoClient("mongodb://localhost:27017/", { useNewUrlParser: true });
    await client.connect();
    const db = client.db('test');
    let r = await db.collection('data100ks').updateMany(
      {yearOfBirth: "1991"},
      {$set:
       {address_city: "Санкт-Петербург"}
      }
    );
    client.close();
}
```

Fig. 6. Code fragment for updating a lot of data in NoSQL DBMS MongoDB

As a result of the code considered earlier it was possible to accumulate a large amount of data. In total, more than 25 thousand data sampling and updating operations were performed for all operations. The results of time measurements of these operations are analyzed in the next section.

3 Data analysis

From the data accumulated during the experiment, a simple non-repeatable sample was built consisting of 3600 records of the time spent on data sampling and updating operations in different DBMS on different platforms and operating systems. With the help of statistical transformations, various data were obtained. Thus, for example, the data on average values of the time spent on executing different operations in different DBMS are given in Table 4.

Table 4. Mean values of testing results

		Raspberry Pi 3 Model B		Windows PC	
		20 000 records (ms)	100 000 records (ms)	20 000 records (ms)	100 000 records (ms)
MariaDB	Select one	148.437	165.327	132.555	286.954
	Select some	271.432	649.541	158.275	300.375
	Update	236.934	670.093	144.052	361.388
MongoDB	Select one	201.167	205.050	1098.593	1105.547
	Select some	205.257	204.003	1098.147	1104.457
	Update	311.510	657.830	1154.953	1117.373
JSON files	Select one	76.680	142.217	31.540	52.660
	Select some	60.317	160.650	24.583	42.000
	Update	24.480	105.993	12.370	31.933

From the table above it follows that the average decrease in performance on Raspberry Pi 3 Model B platform running Ubuntu 18.04 (Bionic) is approximately 193% compared to the performance of the PC platform running Windows.

Also, it is important to note that despite the 2.78 times lower performance when working with flat JSON files on the Raspberry Pi 3 Model B platform running on Ubuntu 18.04 (Bionic) compared to the PC platform running on Windows 10, the performance of MongoDB DBMS on the Raspberry Pi 3 Model B platform is significantly higher than its performance on the PC platform. Thus, the average time spent on the operation of selecting a unique value from the database deployed on the Raspberry Pi 3 Model B platform and containing 100,000 records is 539% faster than the time spent on an identical operation on the Windows PC platform.

This is caused by an interesting statistical anomaly seen in Figures 7 and 8.

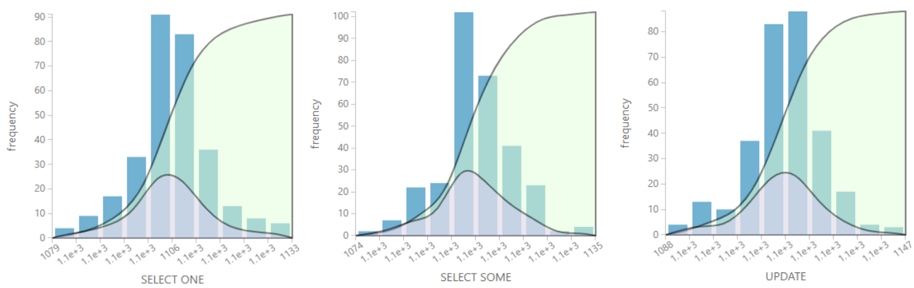

Fig. 7. Histogram of distribution of sample values from the data obtained during MongoDB DBMS testing on Windows PC.

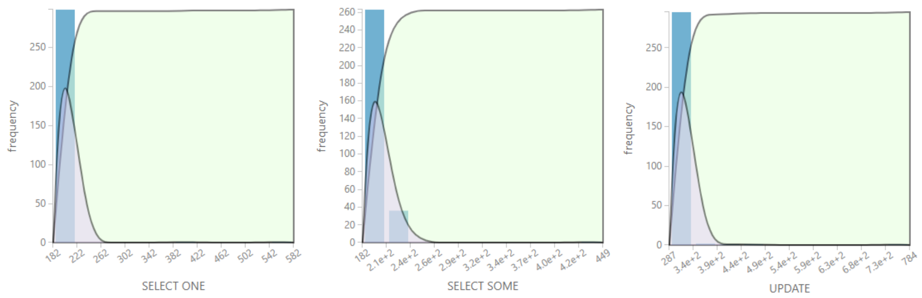

Fig. 8. Histogram of distribution of sample values from the data obtained during MongoDB DBMS testing on Windows PC.

From the diagrams shown in Fig. 7 we can see that the time spent on operations with the MongoDB DBMS deployed on a Windows PC is subject to the Laplace-Gauss distribution. And in Fig. 8 it can be seen that the time spent on the same operations with Raspberry Pi 3 Model B with the Ubuntu 18.04 operating system (Bionic) is subject to the Poisson distribution.

To test the hypothesis that the performance degradation on the Raspberry Pi 3 Model B platform running Ubuntu 18.04 (Bionic) depends not so much on optimizing various DBMS for this platform as on the platform performance itself, besides the table with the average values of the time spent on executing the data sampling and updating operations, the tables with its minimum and maximum values are given below. In Table 5 the minimum testing results is shown. The maximal testing results is shown in Table 6.

Table 5. Minimum values of testing results

		Raspberry Pi 3 Model B		Windows PC	
		20 000 records (ms)	100 000 records (ms)	20 000 records (ms)	100 000 records (ms)
MariaDB	Select one	140	143	128	282
	Select some	269	643	147	291
	Update	235	665	140	355
MongoDB	Select one	157	182	1078	1079
	Select some	182	182	1073	1074
	Update	287	630	1129	1088
JSON files	Select one	75	140	30	52
	Select some	57	155	23	41
	Update	20	101	6	31

Table 6. Maximum values of testing results

		Raspberry Pi 3 Model B		Windows PC	
		20 000 records (ms)	100 000 records (ms)	20 000 records (ms)	100 000 records (ms)
MariaDB	Select one	140	143	128	282
	Select some	269	643	147	291
	Update	235	665	140	355
MongoDB	Select one	157	182	1078	1079
	Select some	182	182	1073	1074
	Update	287	630	1129	1088
JSON files	Select one	75	140	30	52
	Select some	57	155	23	41
	Update	20	101	6	31

It follows from the tables above that the average DBMS performance decrease on the Raspberry Pi 3 Model B platform under Ubuntu 18.04 (Bionic) calculated according to the minimum time spent on data sampling and updating operations is 214% in comparison with the performance of the investigated DBMS on the Windows PC platform. At the same time, the average DBMS performance decrease on the same platforms calculated by the maximum time spent on executing the data sampling and updating operations is 190%. Proceeding from the fact that the previously mentioned performance degradation calculated on the basis of the average time spent on data sampling and updating operations which is 193% it follows that the real performance degradation corresponds to the theoretical one calculated in the paragraph 2.1.

4 Results

In the course of the study it was revealed that the problem of monolingual programming on the Raspberry Pi 3 Model B platform using the JavaScript language, associated with the lack of an official MySQL server distribution, is solved by using an alternative database management system MariaDB. At the same time, the study showed that the use of DBMS MariaDB leads to a decrease in performance by 111-216 percent. Thus, the study found that the Raspberry Pi 3 Model B platform is not an effective choice for working with databases, due to a significant decrease in database performance. At the same time, the platform can be used effectively to deploy applications that do not require high performance data storage and processing operations. For example, Raspberry Pi 3 Model B can be used as an authorization server for web-based applications or as a distributed DBMS cluster element.

The current study has revealed a number of issues that will be the subject of the upcoming research. In the future it is possible to investigate the issue of performance of different DBMS on a cluster of some Raspberry Pi platforms, this study can determine the importance of the Raspberry Pi platform for use in the development of high-load web applications.

References

1. Ivanov, Ivan. (2019). Reasons to migrate to MariaDB. [On-line] - https://www.researchgate.net/publication/331074421_Reasons_to_migrate_to_MariaDB. Last accessed 26.01.2020
2. Stepovik A. N. Analiz preimushhestv i nedostatkov nereljacionnyh SUBD na primere MongoDB //Nauchnoe Obespechenie Agropromyshlennogo Kompleksa. – 2018. – S. 595-597.
3. Shichkina Ju. A., Kuprijanov M. S., Koblov A. A. Sravnenie proizvoditel'nosti re-ljacionnyh i nereljacionnyh baz dannyh na primere MySQL i MongoDB //Informacionnye sistemy i tehnologii v modelirovanii i upravlenii. – 2017. – S. 213-219.
4. B.A.Novikov & M.Y.Levin, "Sravnitel'nyi analiz proizvoditel'nosti SQL i NoSQL SUBD" [Comparative Analysis of the Performance of SQL and NOSQL DBMS],Computer tools in education, no.4, pp.48–63, 2017 (in Russian).
5. Tongkaw, Sasalak & Tongkaw, Aumnat. (2016). A comparison of database performance of MariaDB and MySQL with OLTP workload. 117-119. 10.1109/ICOS.2016.7881999.
6. Lindström, Jan & Das, Dhananjoy & Mathiasen, Torben & Arteaga, Dulcardo & Talagala, Nisha. (2015). NVM aware MariaDB database system. 10.1109/NVMSA.2015.7304362.
7. Abubakar, Yusuf. (2014). BENCHMARKING POPULAR OPEN SOURCE RDBMS: A PERFORMANCE EVALUATION FOR IT PROFESSIONALS. International Journal of Advanced Computer Technology (IJACT). 3. 39.
8. Armstrong, Timothy & Ponnekanti, Vamsi & Borthakur, Dhruba & Callaghan, Mark. (2013). LinkBench: a database benchmark based on the Facebook social graph. Proceedings of the ACM SIGMOD International Conference on Management of Data. 1185-1196. 10.1145/2463676.2465296.
9. MongoDB Documentation. [On-line] - https://docs.mongodb.com/. Last accessed 26 January 2020.