Methods for improving the performance data processing on an example of the task of constructing routes in DTN networks

Shichkina Yulia[1], Mikhail Kupriyanov[2], Alexander Koblov[3]
Saint Petersburg Electrotechnical University "LETI", Saint Petersburg, Russia
[1]Corresponding author
E-mail: [1]*strange.y@mail.ru*, [2]*mikhail.kupriyanov@gmail.com*, [3]*koblow.a.a@gmail.com*

Received 15 May 2017; accepted 23 May 2017
DOI https://doi.org/10.21595/vp.2017.18629

Abstract. The article considers how to speed up data processing in the construction route in DTN network, in particular by applying multi-thread processing using the PHP library "pthreads", selecting the noSQL MongoDB database and applying additional mathematical methods (fuzzy logic) to reduce the amount of data. The results of testing the PHP library "pthreads" on the obtained data set are presented. And also, it is shown the results of comparing the execution time of queries in the DBMS MySQL and MongoDB.

Keywords: SQL, noSQL, MySQL, MongoDB, Data Mining, PHP "pthreads", database, performance, query.

1. Introduction

The software market of computer systems has a large number of commercial database management systems, which are various in functional capabilities, for all mass computer models and for different operating systems.

Even before an experienced database administrator, there is often a difficult choice between databases of the same class, especially between databases that are different in nature. This article presents the results of a comparative analysis of classical relational databases with unstructured, appeared not so long ago, using the example of freely distributed database management systems MySQL and MongoDB [1].

Another problem that arises before the user of data of a larger size is the problem of applying modern methods of parallelizing data processing. Therefore, the special attention in the article is paid to the results of testing the PHP library "pthreads" in order to confirm the possibility of its application to specific data.

Very often the demand for high-loaded solutions does not arise from the very beginning. The transition to radically different technologies in large projects requires huge amount of time and material resources, which companies do not always have. Small project teams may not have the financial opportunity to hire software architects and developers to implement complex functionality. In these cases, companies have to work with accessible tools and technologies.

This article shows how it is possible to use PHP to create a highly loaded solution that implements the process of data cleaning and transformation to prepare for further analysis using Data Mining methods. The solution was tested on test sets of 25.000 and 1.000.000 records.

Testing of MySQL and MongoDB and PHP "pthreads" was done using database queries with information about the location of moving objects. Devices installed on these objects are connected a dynamic network through WI-FI.

2. Formulation of the problem

Let's consider the task of extracting useful information from the database which contains data obtained from devices that transmit its GPS coordinates. Solving this task, it was done preliminary cleaning and transformation of data for accelerated usage of the finite methods of Data Mining. It was implemented in PHP. Also, it was done deeper purification and transformation of data by

applying fuzzy cross-sections in databases was implemented in Java. As a repository of information, the NoSQL database MongoDB of version 3.2 was used.

More detailed data processing and data transformation processes, as well as results of applying fuzzy cross-sections in databases as a method of deep cleaning are described in [2, 3].

3. Preliminary cleaning and transformation of data without using the pthreads library

The accumulated knowledge of GPS transmitting devices is the logging information which together with the coordinates contains the acceleration, rotation, speed, data on the transmitting and receiving devices, as well as other service information. Such a set of data is redundant for the task. In order to apply knowledge extraction methods in the future, it is necessary to prepare the available information.

Preliminary cleaning and transformation in our task consists of the following operations:

• During the cleaning phase, it is necessary to exclude the processing of garbage information and data that were obtained by incorrect reception of a signal from GPS satellites.

• At the transformation stage, it is necessary to make a passage through the two-dimensional array of geographical coordinates and to make a connection to the data related to a certain sector of the geographical coordinate plane to the point in the new coordinate space.

This set of operations was implemented using PHP 7.0 with the mongodb-1.2 driver on a test stand with the characteristics given in Table 1 without using the PHPLIB library for MongoDB.

Despite the fact that most databases easily scale to work with multiple threads, PHP scripts will be executed in a single stream if no additional preparation won't be taken.

Table 1. Characteristics of the test stand

CPU	
Model name:	Intel(R) Core(TM) i3-4030U CPU @ 1.90GHz
Socket:	1
Core(s) per socket:	2
Thread(s) per core:	2
Memory	
RAM:	12 Gb
Storage:	120 Gb (SSD)
Operating system	
Description:	Linux Mint 18 Sarah
Linux Kernel:	4.4.0-21-generic
Database	
MySQL	5.7.17 (InnoDB)
MongoDB	3.4.2 (WiredTiger)
Programming language	
PHP	PHP 7.0.5-4+donate.sury.org~xenial+1 (cli) (ZTS)

With this approach, it is impossible to get the maximum efficiency. The loading graph of logical cores will look like this (Fig. 1).

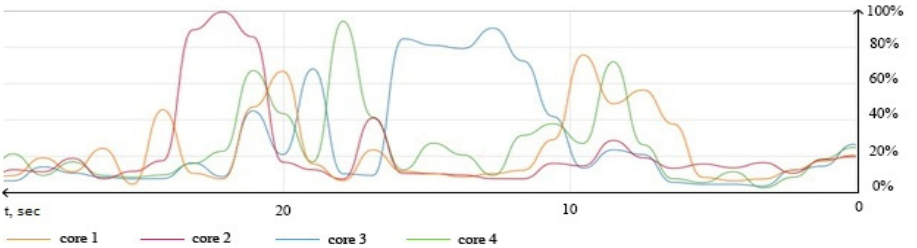

Fig. 1. Logical cores loading graph during processing set of 35.000 entries

As can be seen from the graph, all the threads are involved in the processing of data. This is due to the fact that in MongoDB since version 3.0 it is used the new WiredTiger storage engine which works with all available threads, but none of the threads is loaded for maximum performance. This indicates the irrational use of available resources.

The only correct way out of this situation is to start the processing in parallel with several threads.

4. Parallel implementation of the preliminary cleaning and data transformation using the "pthreads" library

Since the "pthreads" library implements the synchronized() method in the Threaded class, it remains to decide at which point it is possible to run a separate thread.

Within the problem of preliminary cleaning and transformation of data such places were:

• process of cleaning from anomalous values for each coordinate (longitude and latitude). In this case, it is possible to run two threads (one for each of the coordinates).

• bypassing the coordinate space by longitude and latitude when it is implemented the nested loop "for". The inner loop can be run in a separate thread.

It can be seen that the diagram obtained during performing the sequential solution (Fig. 1) is comparatively similar to the diagram obtained using "pthreads" with 1 open thread (Fig. 2).

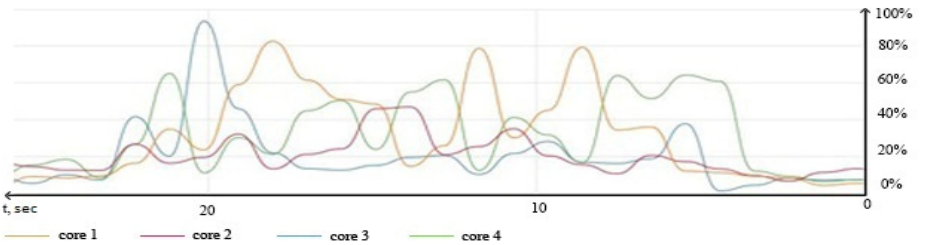

Fig. 2. Logical cores loading graph during processing set of 35.000 entries with "pthreads" library and 1 thread

After adding second thread the logical cores loading becomes more consistent. After adding the third thread the average loading rate comes to 80 % and leaves the space for users with less demanding computations, but there's a lot of unused potential left.

After making additional threads, it becomes noticeable that the loading of logical cores grows, and reaches its peak after using all available threads (Fig. 3). It shows that with help of the "pthreads" extension it is possible to make PHP work quite efficiently not only for web applications, but also for solving problems related to extracting knowledge from large data.

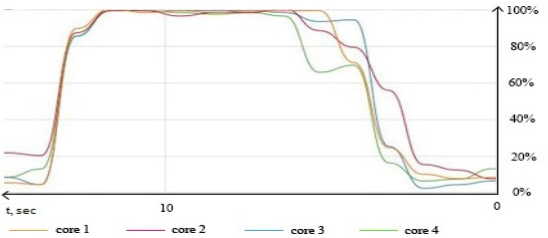

Fig. 3. Logical cores loading graph during processing set of 35.000 entries with "pthreads" library and 4 threads

For comparison, it is necessary to measure the execution time of a sequential and parallel solutions of the preliminary cleaning task and transformation of data on test sets. To carry out more accurate calculations on each set it is necessary to perform 100 measurements and to list

average values in the Table 2.

The results show that the execution time of a sequential solution on a standard PHP 7.0 without using Zend Thread Safety (NTS) is practically the same as for a sequential solution of a task running in PHP using Zend Thread Safety (ZTS) as well as the time of the solution executed with "pthreads" and creating only one thread. However, if it is open more than one thread than it is get a tremendous performance boost, due to which the execution time of the task decreases with the rate of cubic regression. Fig. 4 shows how the execution time of the solution depends on the number of open threads.

Table 2. Test results

	Execution time, sec.	
Algorithm	25.000 entries	1.000.000 entries
Sequential (NTS)	17.09	233.85
Sequential (ZTS)	16.99	228.97
Parallel (1 thread)	17.00	224.94
Parallel (2 threads)	10.01	127.32
Parallel (3 threads)	9.12	98.67
Parallel (4 threads)	8.27	79.67

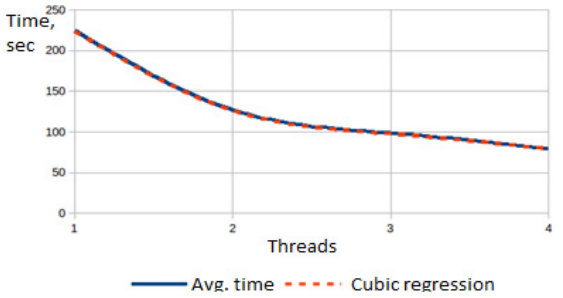

Fig. 4. Dependency of execution time and number of created threads on set of 1.000.000 entries

5. Testing of database management systems

Initially, for the solution of the problem of constructing routing database used MongoDB [4] as the most suitable tool from the point of view of the structure of stored data and the expected speed of response to queries. But, similar problem can be solved with the use of well-known database management system – MySQL [5].

Testing MySQL and MongoDB was done on the same computers, with identical DBMS performance settings, indexes and the same queries. The description of the test stand is given in Table 1 above.

For the purity of the experiment, similar data sets were prepared for both MongoDB and MySQL. The first data set with 25.000 records illustrates work on small projects, such as online stores, where quick access to records is important, but there is no need to process large data sets. The second set of data – 500000 records, illustrates the operation of the DBMS under medium load and the third set gives an understanding of the advisability of using a specific solution in the context of the choice of two DBMSs.

Table. 3 shows the results of the runtime measurements for each DBMS on three data sets with the output of four parallel data processing threads.

However, with the increase in the number of records, the speed of MySQL's operation increases significantly and MongoDB emerges as the undisputed leader. For clarity of comparison Fig. 5 shows the relationship of the application time on all data sets using 1 and 4 threads.

Fig. 6 demonstrates that running MySQL on large data sets significantly lower than MongoDB. It also becomes apparent that the performance of a relational database running in four threads on

large data sets is comparable to the performance of MongoDB, which processes the data sequentially.

Table 3. Results of measurements

DBMS	Datasets		
	25000	500000	1000000
MySQL (1 thread), sec.	7.7116	387.3475	469.0456
MySQL (2 thread), sec.	4.6347	219.7399	278.2428
MySQL (3 thread), sec.	4.2134	214.9975	230.4368
MySQL (4 thread), sec.	4.1275	188.2683	217.9863
MongoDB (1 thread), sec.	17.3954	99.6958	225.6027
MongoDB (2 thread), sec.	9.8326	56.1974	130.8886
MongoDB (3 thread), sec.	9.7207	49.5986	112.5380
MongoDB (4 thread), sec.	9.3348	47.4589	99.3104

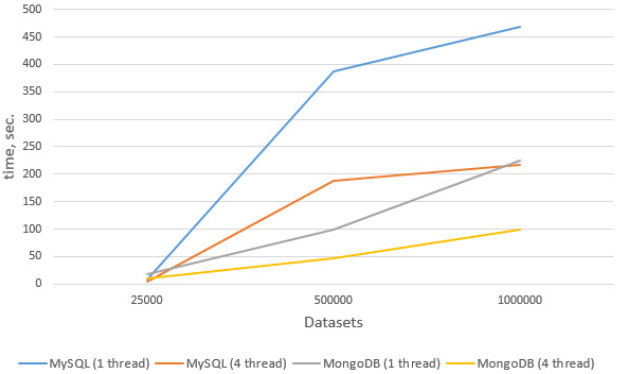

Fig. 5. The graph of the dependence of the execution time on the number of records in the data set

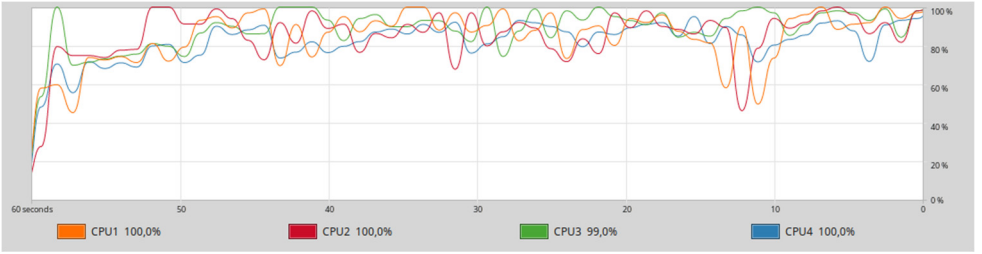

Fig. 6. The graph of the history of loading logical cores from the System monitor application when you start MySQL in 4 threads

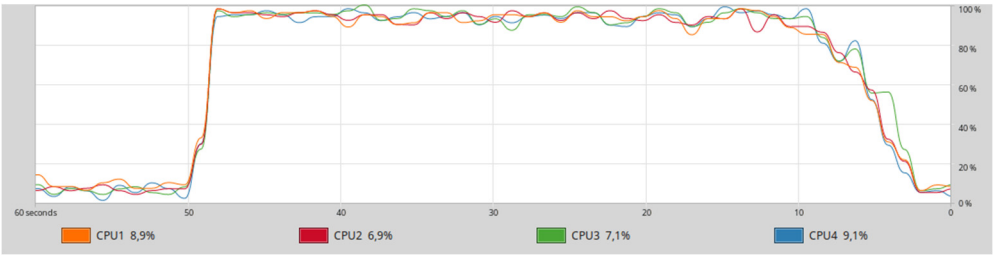

Fig. 7. A graph of the history of loading logical cores from the System monitor application when MongoDB is started in 4 threads

The graphics loading logic processor cores (Fig. 6, 7) show that loading MySQL (Fig. 6) is not constant and has dips 60 % while MongoDB (Fig. 7) has a constant loading of 80-90 % without

dips.

This aspect illustrates the frugality of the presented database management systems to system resources in general and in particular to processor time.

6. Conclusions

In the modern world, the volume of produced and processed data inevitably grows, and the question of rational use of available resources for the creation of new information systems becomes actual.

The results show that with the "pthreads" library it is possible to force PHP applications to use CPU time more effectively and get a boost in the execution time of the preliminary cleaning task and data transformation more than 2 times on a comparatively weak processor.

The results of DBMS testing showed that the use of classical relational databases such as MySQL is rational only for small projects that do not have large amounts of data, or there is no need to quickly obtain a result. However, setting the goal of qualitative data processing, we inevitably come to increase the sample to increase the representativeness. But, increasing volumes, do not forget about the relevance, which data can lose due to too long processing. Therefore, in order to satisfy the need in these criteria it would be better to use appropriate NoSQL databases as the most suited to processing of large data, it was shown in example MongoDB.

Acknowledgements

The paper has been prepared within the scope of the state project "Initiative Scientific Project" of the main part of the state plan of the Ministry of Education and Science of Russian Federation (Task No. 2.6553.2017/8.9 BCH Basic Part).

References

[1] DB-Engines Ranking. https://db-engines.com/en/ranking

[2] **Shichkina Y. A., Degtyarev A. B., Koblov A. A.** Technology of cleaning and transforming data using the knowledge discovery in databases (KDD) technology for fast application of data mining methods. Selected Papers of the 7th International Conference Distributed Computing and Grid-technologies in Science and Education, Dubna, Russia, 2016, p. 428-431.

[3] **Shichkina Y. A., Kupriyanov M. S., Plotnikova A., Ya Domaratsky** Application of fuzzy sections for constructing dynamic routing in the network DTN. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 9870, 2016, p. 63-75, https:/doi.org/10.1007/978-3-319-46301-8_6.

[4] **McCreary Dan, Kelly Ann** Making Sense of NoSQL: A Guide for Managers and the Rest of Us. Manning Publications, 2013.

[5] **Vaish Gaurav** Getting Started with NoSQL. Packt Publishing, 2013.