Journal of Digital Science

ISSN 2686-8296

Volume 4 Issue 2 **December 2022**

CONTENTS

Searching Algorithm in a nonrelational database

Roman Ceresnak [1],
Michal Kvet [1[0000-0003-3937-7473]],
Karol Matiasko [1[0000-0001-7173-2661]]

[1] University of Zilina, Zilina, Slovakia

https://doi.org/10.33847/2686-8296.4.2_2

Received 28.10.2022/Revised 28.11.2022/Accepted 23.12.2022/Published 28.12.2022

Abstract. The problem of the data growth and it is storing to the nonrelational data-bases is related to their decreasing efficiency of searching. Nowadays, a very popular database in memory will help us with decreasing the efficiency of the operation searching in this paper. This paper examines the data search-ing in applications hosted in cloud service Amazon with using of nonrela-tional database DynamoDB. It develops new procedures to provide faster response to user and to obtain the data using of nonrelational database Dy-namoDB, that will provide the demanded data and subsequently, it will transfer them to the memory. The given procedure is based on two methods. The first method is a recognition of values, to which the user refers and the provision of this data to the database in memory. The second method is re-lated to the automatic storing of the data transferred to the database in memory. We perform various experiments in the paper, which are showing us a border of efficiency respectively inefficiency from a time perspective.

Keywords: selecting data, SQL database, NoSQL database, cloud.

1. Introduction

The population growth caused, that many proved procedures, where to belong also traditional databases, started losing their efficiency gradually. In contrast with relational databases [2], databases NoSQL process and manage the big data, characterized as 3V (volume, variety, velocity) [3]. Databases NoSQL are needed to support various applications, which need various levels of performance consistency, availability, and scalability [4]. Social media such as Twitter and Facebook [5] generate for example exabytes of the data daily, whose exceed options of processing of relational databases. These applications demand high performance, but they do not have to demand strong consistency.

It is impossible to achieve the same efficiency of searching with a huge amount of the data in the nonrelational databases than with searching in the relational databases. Regarding the searching in the nonrelational databases, the data, which do not have necessary to meet the strict structural demands of systems (RDMBS), are stored, because the data for the searching can be texted, semi-structured or unstructured. A search engine database is created to help the users to quickly find the information they need in a highly qualified and cost-effective method. They are optimized for keywords and usually, they offer specialized methods such for example a full-text searching, complicated expressions of the searching.

Databases of the searching engine contain two main parts. The first content is added to an index of the database of the searching engine. When the user performs inquiry, relevant results are quickly returned with the help of the database index of the searching engine. Responses to fast searching are possible because instead of direct text searching, they search for inquiries "searching" according to the index. It

is an equivalent of pages loading in a book, related to the keyword, searching of the index in the back part of the book, in contrast to the searching of individual words on every page in the book. This type of index is called an inverted index because it transfers the data structure-oriented on a page to the data structure oriented at the keywords.

The searching efficiency is possible to make it more effective by using the database in memory. The database in memory (IMDB) is a computer system storing and searching the data records, which are situated in the main computer memory, for example in memory RAM. IDMB has an advantage with the data in RAM unlike traditional databases based on disks, causing an access delay because stored media such as units of hard disk and SSD have a markedly slower time of the access as RAM. It means IDMB is useful when fast reading and data recording are decisive.

IMDB works the way they preserve all the data in memory RAM. It is a medium, where the data are stored in RAM opposite the disks in the databases based on disk access RAM. The disk part remains untouched in some IDMB, but RAM is primary a storing medium. Some IDMB also stores the data on disk as a preventive measure to minimalize the risk of the data loss, because RAM is volatile (for example the data are losing when a computer loses energy).

The majority of IDMB also protects from the data loss in one data center (ability known as "high availability") preserving the copies ("replicas") of all the data records on several computers in a cluster. This data redundancy secures, that with an error of whichever computer any data record will not be lost. Among the most popular databases in memory, which help us to get the data with help of inquiry language, belong databases such as Redis, Memcached, and so on. Artificial intelligence will help us with the purpose of transfer of the data situated in the nonrelational database DynamoDB.

The problem related to the data transfer from the database on disk to the database in memory is the velocity of the data searching and the transfer efficiency. Artificial intelligence was used for these purposes, which secures this transfer and so it also makes the data searching more effective. The main benefits of this paper are as follow:

- It creates a new procedure of how to process the data in the memory,
- It reduces the time needed for the record searching in the nonrelational database,
- It defines the methods of how to automatically adjust the data growth to the database's size in memory.

The rest of the paper is structured as follows. "State of the art" section examines the related work. "Our contribution" part represents the designed searching model and the characteristics of this model. "The data transfer" and "The index creation" parts describe the performance of the operation in DynamoDB. "Experiments" part describes performed tests and subsequently the results of the valuation.

2. State of the art

A comparison between relational data models and nonrelational data models NoSQL was already stated in various papers. For example, between these two database types, they are showed and recorded the times needed to perform basic operations such as data selection, data insert, data update, and data deleted. Several statistics point out the fact the most common operation, which is demanded in the relational and nonrelational database, is data selection operation. Many authors showed in their papers, that the time needed to get the data in the nonrelational database is significantly worse as the time needed to get the data in the relational

database. Because of an acceleration respectively improvement of the time needed to get the data from the nonrelational database, it is possible to store the data to buffer memory and by this to reduce repeated searching in the nonrelational database. Not only the time is reduced by this method, but several accesses to the database, too. The authors performed various comparisons between databases in memory such as Redis, Memcached, and nonrelational databases Mongo, Casandra, and H2. One of the main findings of these works is the verification of data update and delete with an increasing number of the data.

We noticed a work during our research, that focuses on problem-solving with the increasing amount of the data. In [3] the authors created a module by using the library Lontar, which will send the data to the relational database by Hibernate as a framework and a relational mapper, in the case of user's demand. Subsequently, Hibernate accesses to MySQL and maps the relational data to object-oriented, and then it sends the data to the nonrelational database. The searching then works with the help of the mapper, so Lontar could be able to read the data in a relational relationship. According to the authors, some data files got better results in nonrelational database MongoDB with the operation searching as in relational database MySQL. However, in certain situations, as the authors describe further, the relational database got better results than the nonrelational database.

The authors introduced a framework capable to manipulate with the data to overcome the problems related to the decreasing efficiency of the searching in the nonrelational databases opposite the searching in the relational databases, Yet before performing of the basic operations of data selection, data insert, data update, and data delete and edit by the mapper happens. The main role of the mapper is to change the data on the base of rules, to such form, that complies with principles of nonrelational database MongoDB more, regarding the searching. By this module, the situation, when the data are faster searched with a help of nonrelational database MongoDB then relational database MySQL, happens in the majority of cases. Another concept used in this framework is the Cataloging module, which uses JSP (JAVA) as a web programming technology and MySQL as DMBS in the principle [11]. There exist two frameworks supporting it, Struts and Hibernate [11]. Structs are used to set a user's interface and the Hibernate regime is used to map the relational data to the object-oriented data, which will be used by JSP. We design a framework in our work, which also uses two database types. The first database is nonrelational database DynamoDB serving as a primary data storage. In the case, when user demands demanded data, the values will not be directly given to them from the nonrelational database, but the values will be transferred to the database in memory. The main challenge to get this aim is to transfer the data from the nonrelational model to the model in memory.

3. Our Contribution

In this part, we will introduce two modules helping us to make the searching in the nonrelational databases more effective. Data Cached Module and Data Elastic Module. DCM serves as a storage data storehouse, which role is the data transfer to the buffer memory. DEM serves on automatic adjusting to the data size.

3.1. A Subsection Sample

The created module serves on the data processing in the memory. We connected the data we store in the nonrelational database DynamoDB to highly available buffer memory Amazon DynamoDB Accelerator, very well known in short as

DAX, with the help of API interface. This method helps us with 3 accesses: Side-cache, Read-through cache, and Write-through cache.

a) Side-cache

This principle helps us with high overload during the reading of information from the memory. This principle works as follow:

1. An application first tries to load the data from the buffer memory for a given couple of key-value. If the buffer memory was filled up with the data (access to the buffer memory), the value returns. If not, step 2 follows.
2. The application loads the data from the storage of the basic data because the demanded couple key-value was not found.
3. A couple of key-value from step 3 will be written to the buffer memory to make sure the data are present when the application needs to load the data again.

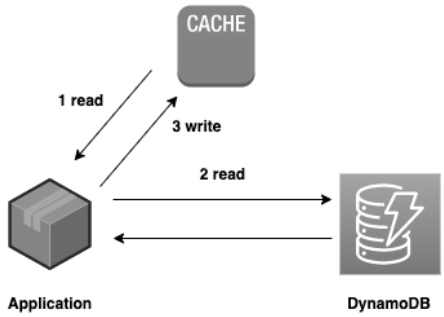

Fig. 1. Side-cache algorithm

b) Read-through cache

DAX is a buffer memory for the reading, because it is compatible with API for the reading of API DynamoDB and stores the results GetItem, BatchGetlem, Scan, and Query to the buffer memory, if they are not currently in DAX. The buffer memory for the reading is effective in difficult working loads. This principle works as follow:

1. Regarding a couple of key-value from the application, it first tries to load the data from DAX. If the buffer memory was filled up with the data (the access to the buffer memory) the value return. If not, step 2 follows.
2. Transparently for the application, if a semi-memory happens, DAX will load a couple of key-value from DynamoDB.
3. To make the data available for every reading that follows, then a couple of key-value will full up in semi-memory DAX.
4. A couple of key-value then will return the value to the application.

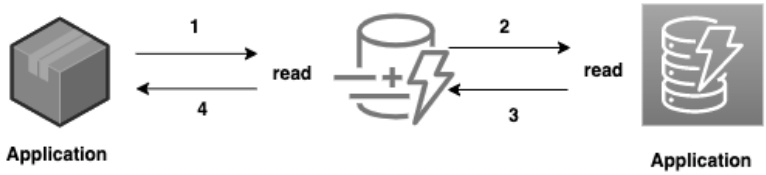

Fig. 2. Read-through cache algorithm

c) Write-through cache

Similarly to the semi-memory for the reading, a semi-memory for the data writing also operates in a line with the database and updates the semi-memory, when

the data are written to the storage of basic data. DAX has also buffered memory for writing, because it stores to the buffer memory (or updates) the items with PutItem, UpdateItem, DeleteItem and BatchWriteItem, API, because the data are written or updated in DynamoDB. At first, DAX is updated (everything is transparent for the application). The following steps indicate a procedure for buffer memory type write-through

1. The application will write itself to endpoint DAX for a given couple of key-value.
2. DAX will catch the writing and then will write a couple of key-value to DynamoDB.
3. After the successful writing, DAX hydrates buffer memory DAX with a new value so whichever following the reading of the same couple key-value results in a finding of the buffer memory. If the writing is unsuccessful an exception will return to the application.
4. Confirmation of successful writing will then return to the application.

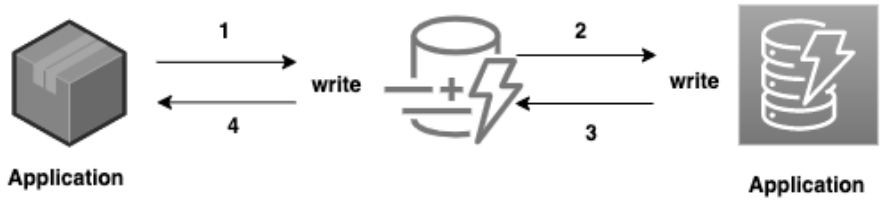

Fig. 3. Write-through cache

3.2. Data Elastic Module

The created module serves with the data increasing in the memory. Nonrelational database DynamoDB fulfills the task of a wide data storage and in the case of the data transfer to the database in memory the data, size can move from several megabytes until gigabytes. This mentioned problem is currently solved by the created module.

We configured monitoring of metrics in cloud service Amazon with the help of service Amazon CloudWatch. The mentioned service makes it possible to edit respectively to add and to remove new calculation units in the case of enabling of horizontal or vertical scaling. An advantageous characteristic of this method is the horizontal scaling, that in the case of a huge number of the data uploads invokes warning of big overload and a script for the reading of information from other replicas in service CloudWatch. The horizontal replica is the part of the script performed automatically during the configuration of the database in memory DAX by the following script.

```
aws dax decrease-replication-factor \
   --cluster-name MyNewCluster  \
   --new-replication-factor 3
```

The monitoring of the metrics in the same way with our method also makes the vertical scaling possible, what is the scaling by addition or removal of the calculation units.

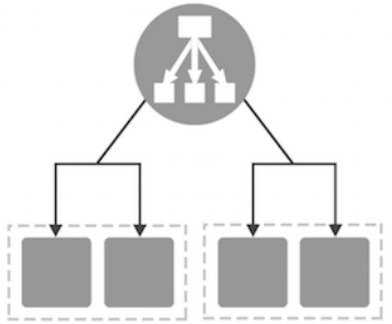

Fig. 4. Application Load Balancer for in-memory database

4. Experiments

In the very first step, we created a simple database model seen in fig 5. This database model is created from two tables, user and comment. These two tables are interconnected by identification relationship type 1:n, which means a 1 user can create various comments and various comments in the table belong right to the 1 user. Subsequently, we compared various orders, whose aim is to get information about the increasing trend of the searching with the increasing number of the data in the relational database Oracle in section A, and then also in nonrelational database DynamoDB in section B. We compared also the times of various operations during data selection in section C because an important aspect of this paper is using of the database in memory.

4.1. Experiments for relational database Oracle

In the very first step, we created a simple database model seen in fig 5. This database model is created from two tables, user and comment. These two tables are interconnected by identification relationship type 1:n, which means ta 1 user can create various comments and various comments in the table belong right to the 1 user.

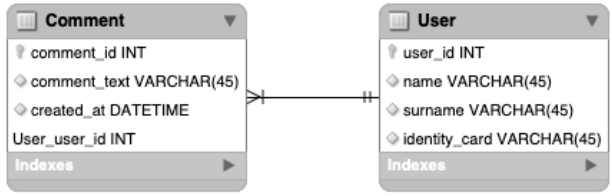

Fig. 5. Data model

We inserted 1000 records into table users and 1000 records into table comments too, based on this defined structure. In the case of this paper, we are interested only in information about the time needed for record searching. We created 3 data selection orders for these purposes looking as follow:

(1) SELECT name, surname FROM user JOIN comment USING (user_id);

*(2) SELECT * FROM user*

```
JOIN comment USING (user_id)
WHERE comment_text LIKE "%today%";

(3)      SELECT  name, surname,
to_char(created_at, 'YYYY-MM-DD HH24:MI:SS') ca FROM user
JOIN comment USING (user_id)
WHERE ca >= TRUNC(current_date)
and ca < TRUNC (current_date) + 1;
```

The first 1000 records served as very first records to us, from which our research can bounce off. The main purpose of the nonrelational databases is to effectively store a huge amount of the data, and that is, why the records for other purposes will be created with the size of 100000 records for table users and 100000 records for table comment. Subsequently, all records are deleted and the records with size 10 000 000 will be inserted to table user and comment. As the last size of the records, we chose a value of 100 000 000 records.

A generator was used for record creation with the size 1 000, 100 000, 10 000 000 and 100 000 000, which is possible to find on this address https://www.generatedata.com/ The generator provides an option to define names and types of attributes and to generate an arbitrary number of the values. After fulfilling the tables by the generated values, we recorder the times needed to perform operations (1), (2), and (3), and they are portrayed in Table 1.

Table 1. Measure time for operation (1) (2) (3) in Oracle

Count of records/operation	1 000	100 000	10 000 000	100 000 000
(1)	0,0020	0,004	0,028	0,44
(2)	0,0021	0,0042	0,031	0,45
(3)	0,0021	0,0044	0,030	0,45

The values needed to get the data from the relational database Oracle are recorded in Table 1. All achieved values for orders (1), (2), and (3) are measured in seconds.

4.2. Experiments for nonrelational database DynamoDB

We used orders (1), (2), and (3) to find out the velocity of demand in the nonrelational database DynamoDB. The values inserted into the database were left the same as in experiments in the relational database. The structure is fully the same as it is recorded in Fig 3.

Table 2. Measure time for operation (1) (2) (3) in DynamoDB

Count of records/operation	1 000	100 000	10 000 000	100 000 000
(1)	0,0035	0,0064	0,047	0,82
(2)	0,0035	0,0067	0,048	0,83
(3)	0,0037	0,0068	0,046	0,82

The values, needed to get the data from the nonrelational database DynamoDB are recorded in table 2. All achieved values for orders (1), (2), and (3) are measured in seconds. During a comparison of the value of the same operations, the values between the results of the relational and nonrelational databases are significantly

different. The nonrelational database in data selection operation is less time effective than relational database Oracle.

4.3. Experiments for the database in memory Redis

The storing in the database in memory is diametrically different than in the relational or nonrelational databases. Except for the mentioned fact, the big problem is also a limitation of the data amount by computer memory. The computer memory about 8 GB was used for testing purposes.

Table 3. Structure of data

ID	Name	Surname	Identity_card	ID
1	John	Harper	12341324	1
2	Joe	Bush	12341234	2
3	George	Obama	23524675	3
.....	.....	.....	.....	.....
.....	.....	....	.....	.....
1000	Alan	Felps	45674866	1000

As is seen in table 3, we created 100, 300, 500, and 10000 records with structure ID, Name, Surname, and Identity_card. The page on this address https://www.generatedata.com/ was used for this purpose against. 3 orders were created for purposes of testing the database in memory's effectiveness, where we were watching the velocity of getting the data. They are the following cases:

(4) MGET Name

(5) MGET Name, Surname

(6) MGET Name, Surname, Identity_card

(7) MGET Name, Surname, Identity_card, Age

We applied the same principle during filling the database as in previous steps. In the actual case, we inserted the generated values to the database, tested operations (4), (5), (6), and (7), and recorded the values. Subsequently, we deleted all the records and inserted the data about 300 records to the database and we continued like this until the size of 1000 records in the database. The recorded values for the operations are recorded in table 4.

Table 4. Measure time for Redis database

Count of records/operation	100	300	500	1 000
(4)	0,00020	0,00022	0,00021	0,00028
(5)	0,00021	0,00023	0,00025	0,00029
(6)	0,00021	0,00022	0,00025	0,00028
(7)	0,00023	0,00024	0,00028	0,00032

All achieved results we got were recorded in table 4 and are in seconds. The seventh operation is influenced by the fact, that value "age" does not exist. As it is seen, the measured values are not diametrically different from the increasing of the values. It is necessary to point out, that with defined growth of the records, it is the logic fact mirroring the efficiency of the searching in the memory.

4.4. Comparison of the final results

The values we got in experimental activity in sections A, B, and C serve right on the comparison with our designed method. The values needed to get the data with operations (1) were measured and recorded in Table 5.

Table 5. Comparison of query performance

Count of records/operation	1 000	1 000 000
Oracle	0,0020	0,44
DynamoDB	0,0035	0,82
Our Approach	0,0033	0,42

As is seen in Table 5, the values measured and got, while using operation (1), do not show any big improvement of the searching in the nonrelational table to us, with a low number of records in the table. This phenomenon is influenced by the data transfer to the memory. A factor of the transfer indicates a necessity to transfer the data from nonrelational database DynamoDB to buffer memory DynamoDAX, which takes a certain time and already when the records are got by the user. It means in operation data selection, the data are physically not got from the nonrelational database DynamoDB, but from the database in memory.

Based on the data transfer, it was possible to compare also the values between the experiments with the database in memory and the data transferred to DynamoDAX with the size of 100 and 500 records during the operation (5).

Table 6. In memory query performance

Count of records/operation	100	500
Redis	0,00020	0,00021
DynamoDAX	0,00015	0,00017

The values recorded in table 6 show us a clear way and efficiency of the data transfer. It is seen, that the values in buffer memory Dynamo DAX are more effective from the time perspective opposite database in-memory Redis.

Whole achieved results related to operation data selection in nonrelational database DynamoDB were not, before the application of our method, timely the same effect than after the application of our method. With using of machine learning and transferring the data to the database in memory, the efficiency of operation data selection in the nonrelational database became more effective after achieving 1000 000 records than with the searching of the data in relational database Oracle. A huge advantage, that results in using of cloud storage Amazon, is related also to the possibility of automatic scaling respectively adding of performance and increasing of the storage not only in nonrelational database DynamoDB, but mostly in the database in memory alternatively, if we do not need as many calculation units, so the reduction of the size of the data storage happens, and so the decreasing of the cost related to running of our designed method happens.

5. Conclusion

NoSQL databases play a significant role in storing and processing huge data and they are used in various wider social applications such as for example Twitter, Facebook, Google, and Yahoo, but they help also with the support of deciding or with creating of advanced analyses. They became the master of high effectiveness and availability of the huge data, but with the loss of the effective searching opposite the traditional databases. This document was devoted to the question of the searching in database NoSQL, concretely Dynamo DB in the cloud background of Amazon to minimalize the impact of this problem.

In this paper, we developed the searching algorithm, that can make the velocity of the searching in a nonrelational database DynamoDB more effective. The designed algorithm is composed of two parts, the first part is based on the principle of caching of the data from the nonrelational database DynamoDB to buffer memory DynamoDAX. The second part is based on the effective data management, that is able, in the case of the huge amount, to automatically create and increase the calculation units of the buffer memory, and by this to adjust the size of the database to the increasing needs of the size of incoming data. This fact relieved us from the limitation of the database size towards the data.

The experiments gave some useful information about the performance and the effectiveness of the created method. It is noted, that the system for processing artificial intelligence demanded higher overhead costs together with automatic creation of the database in memory, but this system was able to make the process of searching in the nonrelational database more effective. On the basis of the experiments, it is clearly seen, the created method is more and more effective with the increasing data amount, which is done by the data transfer to the memory.

Our further work will focus on a generalization of this model and provision of API interface for full use of the created procedure not only for cloud Amazon but also for other databases in memory such as Redis and Memcached. We also plan to evaluate the suggested systems empirically, from a perspective of consistency and performance in other backgrounds, who need a fast response for the data demand.

Acknowledgement

This work was supported by Grant System of University of Zilina No. 1/2020. (8056).

References

1. G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.
2. J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.
3. S. Jacobs and C. P. Bean, "Fine ppapers, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.
4. R. Čerešňák and M. Kvet, "Comparison of query performance in relational a non-relation databases," in *Transportation Research Procedia*, 2019.
5. R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.
6. Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].
7. M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

Aims and Objectives

Published online by ICS two times a year, Journal of Digital Science (JDS) is an international peer-reviewed journal which aims at the latest ideas, innovations, trends, experiences and concerns in the field of digital science covering all areas of the scholarly literature of the sciences, social sciences and arts & humanities. The main topics currently covered include: Artificial Intelligence Research; Digital Economics, Education, Engineering, Finance, Health Care.

The main goal of the journal is the effective dissemination of original incites/results generated by the human brain and presented/reflected in articles using modern information/digital technology.

Editorial Board

Contact information

Website: https://ics.events

Email: conf@ics.events