UDC 004.65

V. Nikitin, E. Krylov, Y. Kornaga, V. Anikin

MODIFICATION OF HASHING ALGORITHM TO INCREASE RATE OF OPERATIONS IN NOSQL DATABASES

Abstract: Optimizing the database is quite a difficult task and involves solving a range of interrelated problems. This is to ensure acceptable performance and functionality of the database, user convenience, optimization of resources consumed, for example, by the criterion of minimizing memory costs and maximizing network usage. The most important aspect of optimization is to increase the performance of the database. To increase the performance of the database, you can use general methods to increase the speed of programs, such as increasing the power of hardware, operating system configuration, optimizing the structure of external media and placing the database on them, and others. In addition, special tools are used to optimize the database, already built into. In particular, most modern relational databases have a special component - query optimizer, which allows you quickly and efficiently process selection requests and data manipulation requests.

The most common way to optimize database performance is to compress the database. It optimizes the placement of database objects on external media and the return of free disk space for future use. The most common compression technology is based on differences, when a value is replaced by information about its differences from the previous value. Another type of compression technology is based on hierarchical data compression. The essence is in the encoding of individual characters with bit strings of different lengths. Indexing and hashing are used to speed up access to database data at the request of users. Indexing speeds up search operations in the database, as well as other operations that require search: delete, edit, sort. The purpose of using indexing is to speed up data retrieval by reducing the number of disk I / O operations. Another common way to organize records and access data is hashing, a technology for quick direct access to a database record based on a given value of some record field, usually a key one.

Keywords: database; SQL database; NoSQL database; indexing; binary tree; hashing; prime numbers.

Introduction

Hashing is the acquisition of a bit sequence from an array of input data of arbitrary length, which is performed according to a certain algorithm. The practice is widely used in many processes of various information systems. For example, when transmitting information over the network, so-called checksums are used, which are calculated from the sent data and allow to avoid forgery by a third party. Another example is getting a file over a network. In this case, the hash check will be performed on the side of the recipient and will be checked with the hash value specified by the sender. In turn, the recipient generates a hash from the

received file and compares it with the sender's hash [1]. Databases are no exception and can use hashing for quick direct access to a stored record based on a normal numeric field value.

Formulation of the problem

Since the main purpose of databases is to store information, from the user's point of view, the most important criterion is the speed of processing requests. To do this, there are special database management systems (DBMS). Given this, it is impossible to say that the speed of request processing depends on the hardware of the server and the optimization of requests [2]. Algorithms that use a database also affect the processing speed and may not use the hardware efficiently enough.

NoSQL databases actively use the B-Tree algorithm for indexing, which has the disadvantage of having to verify compliance with the rules of the algorithm and balancing by adding or removing elements. Additional operations due to the algorithm are undesirable and require additional hardware resources.

As an alternative, an extendible hashing algorithm could be used instead of the B-Tree, but its use is impractical due to the lack of a collision-free hashing algorithm. This means that there is no hash function that would guarantee a single hash for different inputs [3].

Existing hashing algorithms

MD5 (Message Digest 5) is an algorithm that is one of the most widely used cryptographic hash functions, which accepts input of arbitrary length and calculates a fixed 128-bit hash value.

The principle of operation of this algorithm:

1) Add fill bits

A single byte is added to the input data array. After that, the zero bits are added until the remainder of the division of the length of the input data by 512 will not give 448;

2) Add length

The length is added to the data array from the previous paragraph in the form of a 64-bit sequence;

3) Initialization of the buffer

You should understand the buffer as constants that will be used to calculate the hash. Four "words" are used, each of which has a 32-bit length;

4) Message processing

For processing special logical functions which give out 32-bit "words" from input 32-bit "words" turn out. Also, an array of 64 elements is formed, which are obtained by using the function of obtaining the absolute value and the sine function. After that, four rounds of transformations are performed. Each round has 16 elementary transformations;

5) Getting a hash

The hash is obtained by concatenating the values contained in the buffers.

This algorithm allows you to get a hash and use it in checks instead of the original information. It is widely used in Unix systems to store user passwords in 128-bit encrypted form and check files for integrity. The disadvantage of this algorithm is the instability to hash collisions, ie it is possible to create hashes for at least two different input data sets. As a result, it is recommended to use SHA series algorithms.

SHA-3 - the algorithm is a function for creating hashes of the selected length from the input data array of any size. This algorithm can generate hashes of length 224, 256, 384 or 512. It was adopted as a new FIPS standard in 2015. The principle of operation is based on the function of mixing with compression to the selected size of the "cryptographic sponge". The basis of the compression function of the algorithm is the function f, which performs mixing of the internal state of the algorithm. State A takes the form of a 5x5 array, the elements of which are 64-bit words initiated by zero bits. Thus, the state size is 1600 bits. The time arrays B, C, D are initialized. The function f performs 24 rounds, in each of which operations with indices modulo 5 and operation XOR of the round constant on the word A are performed [0, 0].

Before performing the compression function, the operation of XOR fragments of the original message with fragments of the original state is performed. The result is processed by the function f. This overlay, combined with the compression function performed for each block of input data, is the "absorbent" phase of the cryptographic sponge. The resulting hash value is calculated during the execution phase of the cryptographic sponge "squeezing". The basis of the last phase is also the function f [4].

The proposed solution

In order to use the extended hashing algorithm in non-relational databases, it is necessary to have a stable hashing algorithm with respect to collisions and at the same time compressing the input data array. That is, the algorithm must satisfy the conditions:

1) Suppose there is a set of input data $X = \{x_1, x_2, \dots, x_n\}$ and hash function $h(x)$. Then there should be no such that x_i, when $h(x_i) = h(x_j)$, where x_i та $x_j \in X$ and $i \neq j$;

2) Suppose there is a set of input data $X = \{x_1, x_2, \dots, x_n\}$ and and a set of input data sizes $S_1 = \{s(x_1), s(x_2), \dots, s(x_n)\}$. Then, there are sets $H = \{h(x_1), h(x_2), \dots, h(x_n)\}$ and $S_1 = \{s(h(x_1)), s(h(x_2)), \dots, s(h(x_n))\}$. Then each $s(x_i) = s(h(x_i))$, where i – values from 1 to n.

When the first condition is met, there is no need for a collision avoidance mechanism. Otherwise, the data may degenerate into a linear list, which will reduce performance and increase memory usage. If the second condition is not met, then it makes no sense to use the algorithm.

The basis for the algorithm is the use of prime numbers, as well as the use of binary number system in computer systems. The algorithm itself involves block hashing and binding to the bit rate of the system on which the algorithm will run. It satisfies the two conditions

described earlier, but this is only confirmed empirically with a limited number of tests. Description of the hashing algorithm:

1) Determining the size of the block

This step is necessary to determine the fixed length of the block in bits. For convenient operation, the block size is chosen equal to L and corresponds to the maximum number of bits needed to describe an insignificant integer;

2) Convert input data into a single binary sequence

Data such as strings can have different encodings, such as ASCII, UTF. In the first case, each character is encoded with 8 bits. In such cases it is necessary to cut the senior bits which are equal to zero;

3) Filling

If the length of a single binary sequence is not a multiple of L, then you need to supplement the sequence with bits equal to zero;

4) Breaking into blocks

The resulting bit sequence is divided into blocks of fixed length L. Thus, we obtain N blocks. Each block can be represented as an unsigned integer. Blocks representing zero in decimal form are discarded;

5) Representation of each block as a product of prime numbers

Since each block can be represented by an unsigned integer, we can use the basic theorem of arithmetic. By this theorem, every positive integer greater than one can be factorized as:

$$n = p_1 \cdot p_2 \cdot \ldots \cdot p_n \tag{1}$$

where n – a natural number, $p_1, p_2, \ldots, p_n$ – a prime numbers;

6) Definition of "fillers"

This is necessary in order to add uniqueness to each block hash. Fillers are natural numbers that start with 4 and are not prime numbers. Each block corresponds to only one unique placeholder and its value is greater than the previous one;

7) Block hash calculation

The hash of a block is the sum of prime numbers that were obtained as a result of factorization and n numbers of "placeholder", where n is the number of placeholder numbers that must be added to the hash, and can be calculated by the formula:

$$n = L - n_{пр} \tag{2}$$

where $n_{пр}$ – the number of prime numbers on which the decimal value of the block was factorized;

8) Obtaining the resulting hash

Since each hash of the block is an unsigned integer, the first hashes are supplemented to the maximum value from the next hash. The resulting hash is obtained by the concatenation operation of all received hashes of blocks, ie:

 ISSN 1560-8956

$$H = H_1 \cdot H_2 \cdot \ldots \cdot H_N \quad (3)$$

Thus filling takes place from right to left.

The algorithm has been tested for a limited number of tests, but has shown good resistance to collisions with respect to the string data type. To implement the algorithm, the C ++ programming language was used, which allows working at the level of bit operations. The test results are shown in table. 1.

Table 1

Results of testing

Test description	Amount of collisions	The resulting hash
Line “KQAT” and generation of 1 000 000 random lines with length 4 symbols	0	Got hash for “KQAT” line: 1011111011110110 (16 bits)
Generation and hashing of random lines with length from 4 to 20 symbols	0	The largest hash: 11110011010000101000101111111 (28 bits)

Conclusions

The algorithm has a disadvantage in the size of the resulting hash. Compared with MD5 and SHA-3, the proposed algorithm does not have a statically defined size. Also, cryptographic stability can be considered a disadvantage, because in the case of a unique hash it is possible to calculate the original data set. This can be corrected by the so-called "salt", ie an additional array of input data, which will further change the resulting hash. A static block of dimension L can be used as a "salt", but since the goal is to use an algorithm for indexing in non-relational databases, this is not a critical place and object of further research.

In further research it is planned to use a mathematical apparatus to prove its properties, as well as statistical research on a large sample for practical proof of the effectiveness of the proposed method.

REFERENCES

1. Anton Yudhana, Abdul Fadlil, Eko Prianto. Performance Analysis of Hashing Methods on the Employment of App URL: https://www.researchgate.net/publication/328020140_Performance_Analysis_of_Hashing_Methods_on_the_Employment_of_App

2. Michael L. Rupley, Jr.. Introduction to Query Processing and Optimization. URL: https://clas.iusb.edu/computer-science-informatics/research/reports/TR-20080105-1.pdf

3. V. Nikitin. Combined indexing method in nosql databases / V. Nikitin, E. Krylov, Y. Kornaga, V. Anikin. - Adaptive Systems of Automatic Control Interdepartmental scientific and technical collection. – 2021. – №1(38) – DOI: https://doi.org/10.20535/1560-8956.38.2021.232948;

4. Sandeep Kumar, Er Piyush Gupta. A Comparative Analysis of SHA and MD5 Algorithm URL: https://www.researchgate.net/publication/263656830_A_Comparative_ Analysis_ of_SHA_and_MD5_Algorithm