UDC 004.65

V. Nikitin, E. Krylov

A COLLISION-RESISTANT HASHING ALGORITHM FOR MAINTAINING CONSISTENCY IN DISTRIBUTED NOSQL DATABASES

Abstract: A distributed database is a combination of copies of databases of one or different types using computer networks. Management of such systems is transparent to end users, that cannot be said about emergency situations and certain changes in the number of nodes. Globally defined properties include consistency, availability, and allocation tolerance. They appear as a result of the need for horizontal extension, which entails the need for copies of the stored data. This is due not only to the issue of productivity, but also to the issue of availability. These two properties are diametrically opposite: technologies and methods which improve one of them, worsen automatically the condition of the other.

In addition, any existing information system uses a large set of algorithms. Each algorithm is necessary to solve some problem. The latter is quite diverse: sorting, structuring and searching for data, obtaining a unique digital fingerprint from a data set. The possibilities of usage are not limited to a certain direction and only encourage researchers to seek for new ones. This includes hashing algorithms, which are widely used in databases, in checking the integrity of files and network packets. Hashing has a wide range of usage and is not limited to use only for checking integrity, but can be used as an analogue for indexing instead of balanced trees by building hash tables [1].

Despite a great diversity, new problems arise that need to be solved. With the development of data transmission and storage technologies, there is a nicessity to improve consistency support in distributed NoSQL databases. Existing hashing algorithms are deterministic and based on bitwise operations, which make it impossible to predict collisions. Thus, the main goal of developing a new algorithm is the idea of creating such an algorithm that will improve collision resistance when changing the size of input data and which will allow estimating the possible number of collisions.

Keywords: distributed databases, distributed systems, hashing, hash functions, consistency, collision resistance, data consistency.

Introduction

There is a CAP theorem that states that a distributed system can support only two of the three desired properties [2]. These properties include consistency, availability, and partition tolerance (fig. 1).

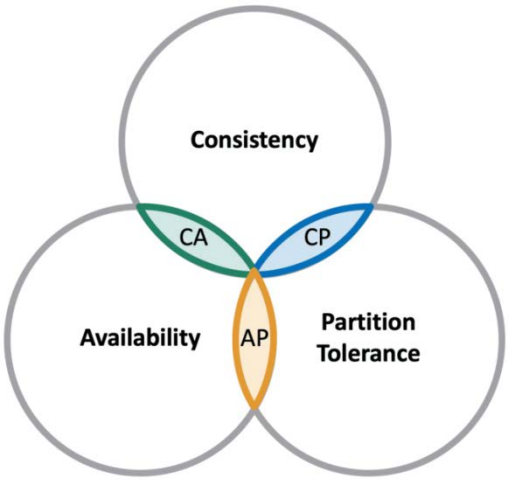

Figure 1. The CAP theorem

A high level of consistency causes clients to read the most updated data regardless of the node. A high level of availability ensures a stable and continuous response to customer requests, regardless of the relevance of data. The ability to partition ensures stable and continuous operation of the system even in the presence of communication problems of nodes or their failure.

It should be noted that the properties are interrelated and the improvement of one property leads to the deterioration of another.

The task of maintaining consistency is of great importance in distributed systems. It is difficult to imagine the normal operation of critical services due to the inconsistency of data on different nodes. Having up-to-date information as soon as possible can save lives and save money that may be lost due to outdated information.

Formulation of the problem

Several methods are used to reconcile data in distributed systems. One method involves the use of a central node that captures changes and sends updated data to other nodes. This approach is called centralized and resembles a "star" topology (fig. 2).

This approach allows you to maintain high consistency if clients send requests only to the central node for writing, and others are used as backups and work to read data. Also, it should be noted that this approach provides a high level of availability, since there are always agreed nodes that can replace in case when the central one fails, but there will be a certain delay until a new central node is selected.

As an example, you can cite MongoDB. A MongoDB cluster consists of two types of data storage:

– primary

ISSN 1560-8956

The primary node receives all write operations.

– secondary

Secondary nodes receive replicated data from the primary to maintain an identical data set.

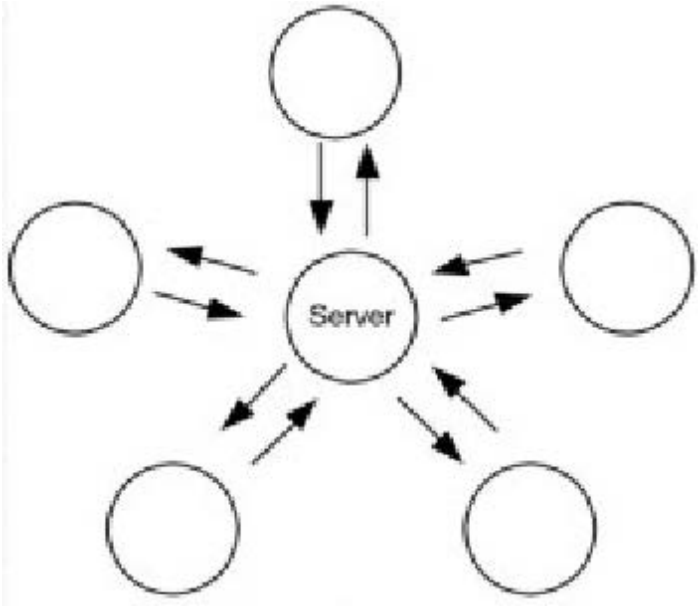

Figure 2. Centrilized approach

By default, the primary node handles all reading and writing. Additionally, a MongoDB client can direct some or all reads to secondary members. Entries must be sent to the primary. If the primary participant fails, all recordings are suspended until a new primary is selected from one of the secondary participants. According to the MongoDB documentation, this process takes up to 12 seconds to complete. To increase availability, a cluster can be distributed across geographically distinct data centers.

Another method is based on the fact that there is no primary node and the nodes coordinate data directly with each other (fig. 3).

When receiving updates, a node notifies everyone else about it. This is a fairly reliable method, but requires an additional mechanism to resolve conflicts when updating the same pieces of data, if they have been updated before. It provides a high level of availability, but can have a negative impact on consistency because communication between multiple nodes may be missing. On the one hand, reconciliation occurs almost instantaneously, and on the other hand, an update may not be available at all on a particular node.

This way of maintaining consistency is used in Cassandra. A Cassandra cluster is a collection of instances, called nodes, connected in a peer-to-peer, shared-nothing architecture. There is no master node, and each node can perform all database operations, and each can serve client requests. Data is distributed between nodes based on an agreed hash of the partition key. A partition has one or many rows, and each node can have one or more

partitions. However, a partition can reside on only one node. Data has a replication factor that determines the number of copies (replicas) to make. Replicas are automatically stored on different nodes. The node that receives the request from the client first is the coordinator. The task of the coordinator is to forward the request to the nodes containing the data for that request and send the results back to the coordinator.

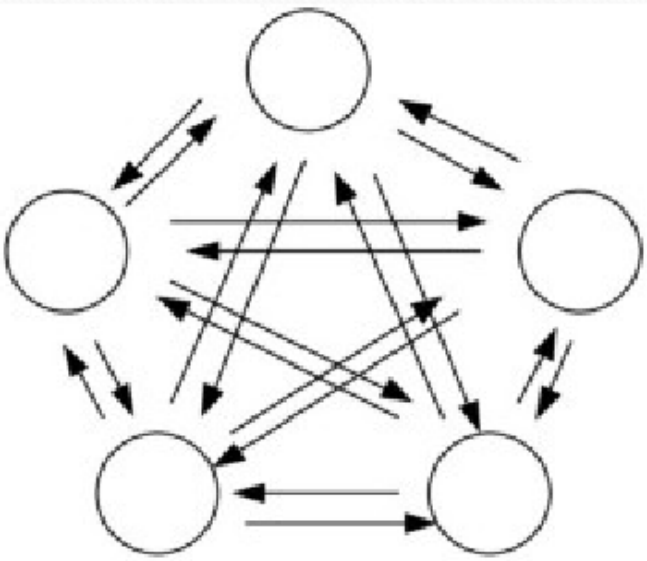

Figure 3. No centralized node approach

Any node in the cluster can act as a coordinator (fig. 4).

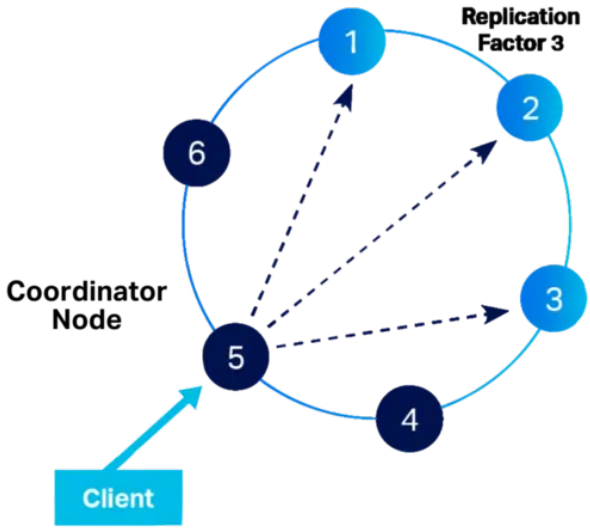

Figure 4. Cassandra’s work schema

The last method of maintaining consistency is based on the principle of gossip protocol (fig. 3). Each node sends an update to a randomly chosen other one and they propagate it to each other. In this case, the situation is completely opposite to the previous method. There is a possibility that clients will try to retrieve inconsistent data, although there

ISSN 1560-8956

is less risk that a node will not be updated due to a lack of communication between two nodes because it will be notified by a third node [3].

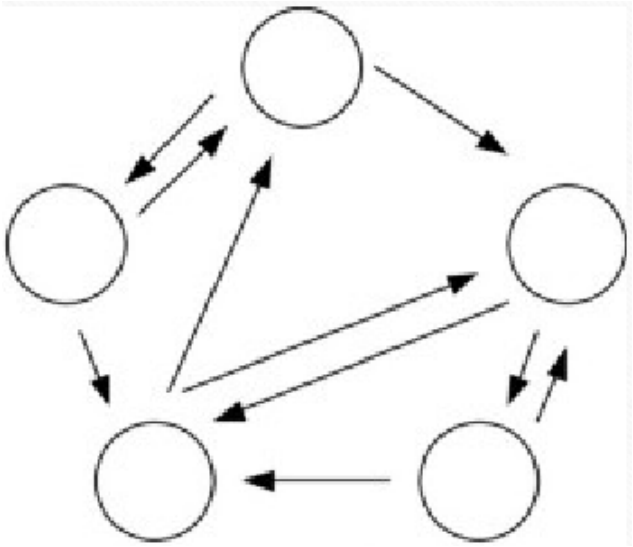

Figure 5. Gossip protocol based approach

Regardless of how a system reconciles data, it may have additional hashing mechanisms, such as a consistent hashing algorithm. An example is Riak, which uses a consistent SHA1-based hashing algorithm that is mathematically proven to provide a perfectly even distribution around a 160-bit space (ring). This 160-bit space is divided into equal partitions called "vnodes". These nodes, in turn, are evenly distributed among the physical nodes participating in the cluster. An even distribution around the 160-bit space and an even distribution of vnodes between physical nodes ensures an even distribution of keys in the cluster. Nodes participating in a Riak cluster are equals – meaning any node can serve any request – and due to the nature of consistent hashing, each node in the cluster knows where data should be placed within the cluster (fig. 6).

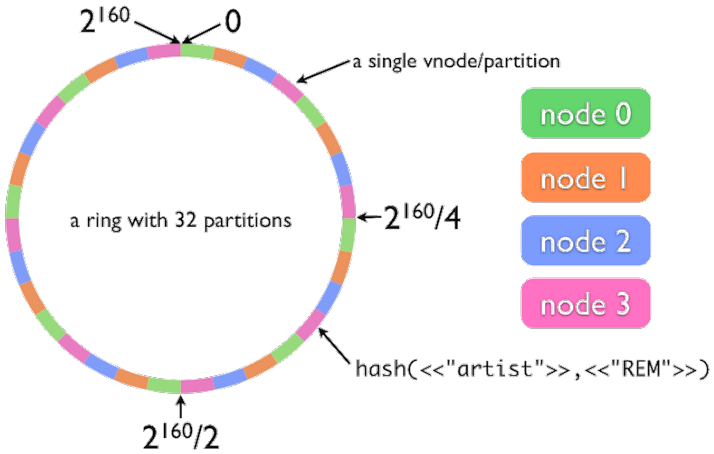

Figure 6. Riak’s consistent hashing

In addition to the agreed hashing algorithm, Active Anti Entropy technology can be activated. This mechanism constantly corrects conflicting data in the background. Where Read Repair fixes data as it is read, AAE fixes all data regardless of whether it is actively being accessed by running a background process that constantly checks for inconsistencies. AAE uses a Merkle Tree hash exchange to find these inconsistencies. When a difference is detected at the top of the tree, the database recursively scans the tree until it finds the exact values with the difference, and then sends the least amount of data necessary to restore balance. Riak has such a mechanism and SHA1 is used as a hashing algorithm.

The problem is that the most common hashing algorithms are based on bitwise operations, which in turn can lead to collisions when hashing the data itself in the Merkle tree. To solve this problem, it is possible to use an algorithm that will be aimed at hashing data, and to obtain the resulting hash - an algorithm that is already in use.

In most cases, the most common hashing algorithms are used, such as MD5, SHA1, SHA2. They refer to cryptographic algorithms, in which one of the main criteria is the impossibility of reproducing the input array of data [4].

Mathematical description of the PH-2 algorithm

To increase the level of data consistency in a distributed system, the PH-2 algorithm was developed, which is based on the PH-1 algorithm [5].

PH2 hash consists of 6 parts (fig. 7). Each part takes an appropriate value depending on the input data array.

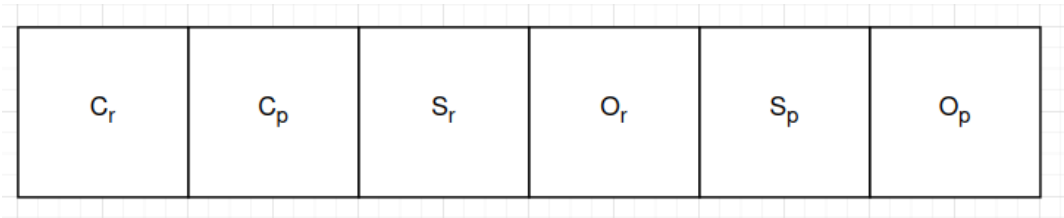

Figure 7. Structure of PH2 hash value

The number of non-prime numbers is encoded in the first part. The number of prime numbers is encoded in the second part. The third and fourth parts represent the sum of non-prime numbers and the number of overflows due to the limitation of the bit rate of the computing machine. By overflow we mean the number of nulls due to the maximum allowable value of an integer, since it is physically impossible to describe any large integer. The fifth and sixth parts are the same as the third and fourth parts, but using prime numbers. As a block size, it is recommended to take a value of 32 bits, which in turn means that the size of the resulting hash value will be equal to 192 bits.

Let's enter the parameters that are necessary for calculations:

1) Let there be a non-negative number s characterizing the block size in bits;

ISSN 1560-8956

2) Let there be a set of prime non-negative numbers P = {2, 3, 5, … , p_k}, where each $p_i <= 2^s$;

3) Let there be a set of non-negative integers N = {n_1, n_2, … , 2^s};

4) Let there be a set of non-negative integers B = {b_1, b_2, … , b_n}, which were obtained as a result of converting blocks of bits into numbers.

Let's calculate all the necessary components of the hash value:

1) Calculate the sum of all non-prime numbers

$$S_r^* = \sum_{i=1}^{n} b_i, \quad (1)$$

under the condition $b_i \notin P$;

2) Calculate the sum of all prime numbers

$$S_p^* = \sum_{i=1}^{n} b_i, \quad (2)$$

under the condition $b_i \in P$;

3) Calculate the number of non-prime numbers

Since the division operation can be represented in the form:

$$a = b \times c + r, \quad (3)$$

where a - divided, b - divisor, c - incomplete part, r - remainder 0 <= r <= |b|.

Find the number of overflows of non-prime numbers:

$$C_r = \sum_{i=1} 1, \quad (4)$$

where $\forall i \in \mathbb{N}, S_r^* - 2^s \times i \geq 0$;

4) Calculate the number of prime numbers

Use the formula 3 to calculate amount of overflow of prime numbers:

$$C_p = \sum_{i=1} 1, \quad (5)$$

where $\forall i \in \mathbb{N}, S_p^* - 2^s \times i \geq 0$;

5) Calculate the number of overflows of non-prime numbers

$$O_r = S_r \bmod 2^s, \quad (6)$$

6) Calculate the number of overflows of prime numbers

$$O_p = S_p \bmod 2^s, \quad (7)$$

7) Calculate the sum of all non-prime numbers, which can be saved taking into account the bit rate of the operating system

$$S_r = S_r^* \bmod 2^s, \quad (8)$$

8) Calculate the sum of all prime numbers, which can be saved taking into account the operating system's bit rate

$$S_p = S_p^* \bmod 2^s, \quad (9)$$

Having received all the components, it is necessary to concatenate the bytes in the order indicated in fig. 7 and get the resulting hash.

Examples of hashing using PH2

Consider several examples of hashing:

– PH2-48

This variant of the algorithm requires 8 bits per block. This means that the maximum possible block value can be $2^8 = 255$.

Let the line “Hello World!!” be the input data array.

Convert each symbol into an integer in the decimal number system (Table 1).

Table 1. Represantion of ASCII symbols as integers

H	e	l	o	‘ ‘	W	r	d	!
72	101	108	111	32	87	114	100	33

Get a sequence of numbers in the decimal number system:

B = (72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33, 33)

Divide the sequence B into two sequences P and R, which will contain prime and non-prime numbers, respectively:

P = (101)

R = (72, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33, 33)

Now we can calculate all the necessary parts to form the hash value:

1) Calculate the sum of all non-prime numbers

$S_r^* = \sum_{i=1}^{N_r} r_i = 72 + 108 + 108 + 111 + 32 + 87 + 111 + 114 + 108 + 100 + 33 + 33 = 1017$

2) Calculate the sum of all prime numbers

$$S_p^* = \sum_{i=1}^{N_p} p_i = 101$$

3) Calculate the number of non-prime numbers

$$C_r = 12$$

4) Calculate the number of prime numbers

$$C_p = 1$$

5) Calculate the number of overflows of non-prime numbers

$$O_r = S_r^* / 2^s = 1017 / 255 = 3$$

6) Calculate the number of overflows of prime numbers

$$O_p = S_p^* / 2^s = 101 / 255 = 0$$

7) Calculate the sum of all non-prime numbers, which can be saved taking into account the bit rate of the operating system

$S_r = S_r^* \bmod 2^s = 1017 \bmod 255 = 252$

8) Calculate the sum of all prime numbers, which can be saved taking into account the operating system's bit rate

ISSN 1560-8956

$$S_p = S_p^* \bmod 2^s = 101 \bmod 255 = 101$$

We have calculated all the necessary parts and this makes it possible to obtain the resulting hash using concatenation and using the symbol '.' as a separator between the parts:

$$h = C_r.C_p.S_r.O_r.S_p.O_p = 12.1.252.3.101.0 =$$

$$= 00000100.00000001.11111100.00000011.01100101.00000000$$

– PH2-96

The algorithm is the same, but with the difference that in this case there will be a smaller number of blocks.

The size of the block in this case is equal to 2 bytes. The output sequence is 13 bytes long. In order to break it into blocks, you need to add zero bytes to the original sequence of bytes.

Get a sequence of numbers in the decimal number system:

B = (72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33, 33, 0)

Table 2 shows the resulting blocks.

Table 2. Resulting bit sequences due to concatenation of adjacent bytes

72, 101	01001000 01100101
108, 108	01101100 01101100
111, 32	01101111 00100000
87, 111	01010111 01101111
114, 108	01110010 01101100
100, 33	01100100 00100001
33, 0	00100001 00000000

We convert into numbers in the decimal numbering system:

B = (18533, 27756, 28448, 22383, 29292, 25633, 8448)

Divide the sequence B into two sequences P and R, which will contain prime and non-prime numbers, respectively.

P = (25633)

R = (18533, 27756, 28448, 22383, 29292, 8448)

We will perform the same calculations as in the example for PH2-48:

1) Calculate the sum of all non-prime numbers

$$S_r^* = \sum_{i=1}^{N_r} r_i = 18533 + 27756 + 28448 + 22383 +$$

$$+ 29292 + 8448 = 134860 \text{ !!!} \bmod 65535 = 3790$$

2) Calculate the sum of all prime numbers

$$S_p^* = \sum_{i=1}^{N_p} p_i = 25633$$

3) Calculate the number of non-prime numbers

$$C_r = 6$$

4) Calculate the number of prime numbers

$$C_p = 1$$

5) Calculate the number of overflows of non-prime numbers

$$O_r = S_r^* // 2^s = 134860 // 65535 = 2$$

6) Calculate the number of overflows of prime numbers

$$O_p = S_p^* // 2^s = 25633 // 65535 = 25633$$

7) Calculate the sum of all non-prime numbers, which can be saved taking into account the bit rate of the operating system

$$S_r = S_r^* \bmod 2^s = 134860 \bmod 65535 = 3790$$

8) Calculate the sum of all prime numbers, which can be saved taking into account the operating system's bit rate

$$S_p = S_p^* \bmod 2^s = 25633 \bmod 65535 = 25633$$

We have calculated all the necessary parts and this makes it possible to obtain the resulting hash using concatenation and using the symbol '.' as a separator between the parts:

$$h = C_r . C_p . S_r . O_r . S_p . O_p = 6.1.3790.2.25633.0$$

– collision examples

Let there be a sequence of bytes:

$$B_1 = [140, 230, 250, 41, 7, 250, 250, 179]$$

The resulting hash when using the PH2-48 algorithm will be equal to:

$$h_1 = 5.3.100.4.227.0$$

It can be seen from the algorithm itself that the order of bytes has no influence on the formation of the hash value. Let there be a sequence:

$$B_2 = [230, 140, 41, 250, 250, 7, 179, 250]$$

B_2 consists of the same elements as B_1. The resulting hash for B_2 will be the same as for B_1:

$$h_2 = 5.3.100.4.227.0$$

Another variant of the collision is overflow, which occurs as a result of exceeding the permissible capacity of the block.

Let there be a sequence B_3 consisting of:

1) Block 255 in the amount of 259 blocks;

2) A single block with value 100;

3) A single block with value 41;

4) A single block with value 7;

5) A single block with value 179.

The hash value of this sequence will look like:

$h_3 = 5.3.100.4.227.0$

ISSN 1560-8956

It can be seen that the hash value of h_3 matches the hashes of h_1 and h_2 that were obtained from sequences B_1 and B_2.

Collision stability of the PH2 algorithm

The PH2 algorithm, like any other hashing algorithm, is subject to collisions. Several experiments were conducted to analyze the collision resistance. The essence of the experiments consisted in the generation of 1,000,000 random data of a certain size and their subsequent hashing using PH2-48 (fig. 8) and PH2-192 (Fig. 9). A total of 121 iterations were performed for each of the algorithms, starting with a block size of 8 bytes up to and including 128 bytes.

Fig. 8 and 9 show that as the block size increases from 8 bits to 32 bits and as the input data size increases, the number of collisions decreases. The obtained results look negative against the background of other algorithms, such as SHA1 and SHA2, because in their cases the number of collisions is 0 for any size of input data.

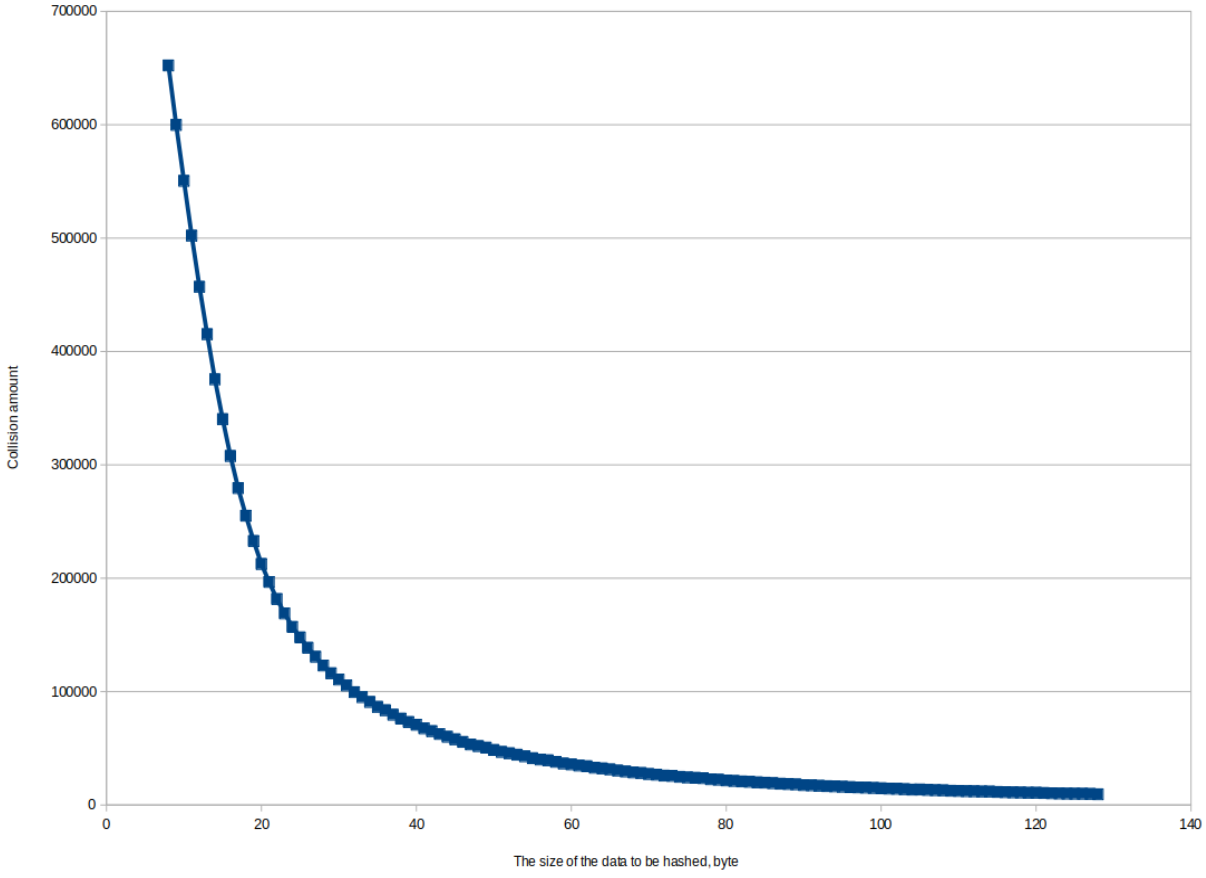

Figure 8. The number of collisions when hashing with the PH2-48 algorithm

Despite the obvious loss when hashing data of the same size, the developed PH2 algorithm guarantees the uniqueness of the resulting hash value when the size of the data being hashed is changed. The smallest change in size for even 1 byte will produce a different

hash value compared to the past input data. Experimentally, it was not possible to obtain a single collision, and the mathematical description is a confirmation of this property.

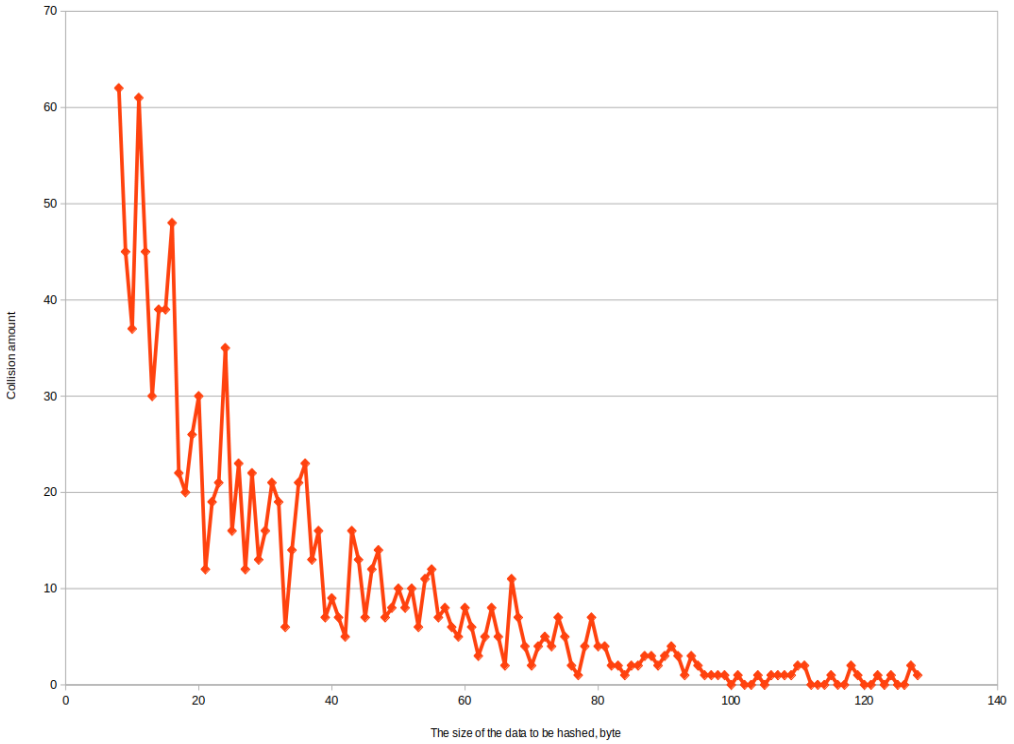

Figure 9. The number of collisions when hashing with the PH2-192 algorithm

Performance comparison of PH2 with some SHA2 algorithms

In fig. 10 it can be seen that PH2 variants take less time to hash data compared to SHA256 and SHA512.

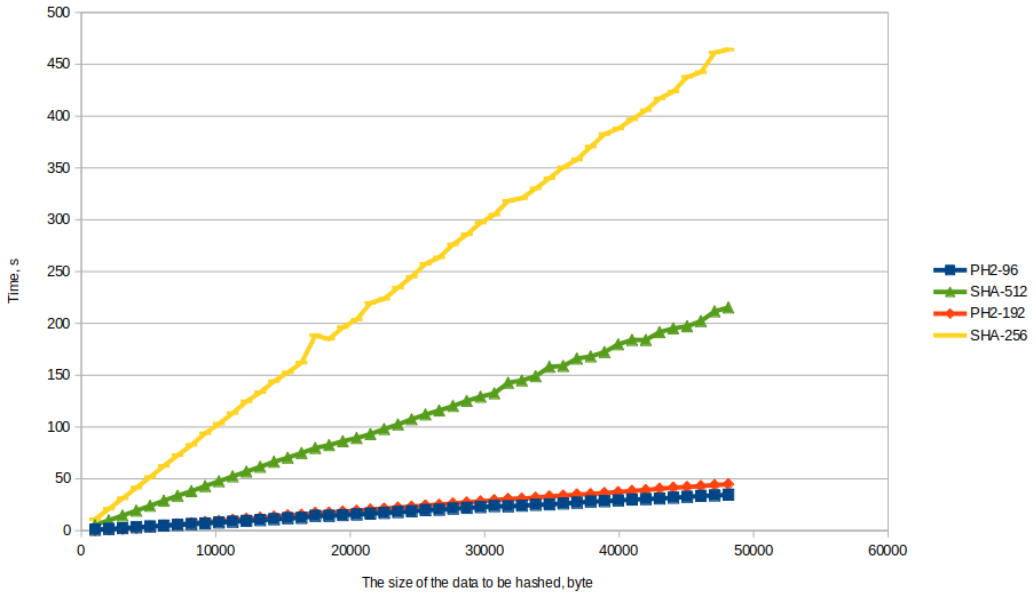

Figure 10. Hash time with PH2-96, PH2-192, SHA256 and SHA512 algorithms

ISSN 1560-8956

Conclusions

Currently, there are many hashing algorithms, but for the most part, their main purpose is to obtain a cryptographic fingerprint. This means that in some sense the question of uniqueness is neglected, which has no place in the question of ensuring consistency in distributed NoSQL databases.

It is to maintain consistency in such systems that a deterministic algorithm was developed with the generation of a hash value of a fixed length. It is sensitive to changes in the size of the input data and has a high hash rate.

The developed algorithm has a mathematical description that makes it possible to theoretically estimate the degree of collision resistance and estimate possible collision cases. It can be used in combination with a Merkle tree to obtain the root hash value used in Active Anti Entopy technology. In this case, his area of responsibility may be the hashing of the stored data on the node of the distributed system, and subsequent hashing using existing algorithms.

Another way to use it can be to create a technology based on the "gossip" protocol to maintain consistency in distributed NoSQL databases for particularly important information, which will be an additional mechanism on the same level as Active Anti Entropy.

REFERENCES

1. Combined indexing method in nosql databases / V. Nikitin та ін. // Adaptive Systems of Automatic Control Interdepartmental scientific and technical collection. 2021. № 38 (1). С.3-9 URL: https://doi.org/10.20535/1560-8956.38.2021.232948;

2. Gilbert S., A. Lynch N. Perspectives on the CAP Theorem // Computer. 2012. № 45 (2). P.1-10 URL: https://groups.csail.mit.edu/tds/papers/Gilbert/Brewer2.pdf

3. RRG: redundancy reduced gossip protocol for real-time N-to-N dynamic group communication / V. Wing-Hei Luk та ін. // Journal of Internet Services and Applications. 2013. № 4 (14). С.1-19 URL: https://rdcu.be/cZ1Fk

4. Comparison of hashing methods for supporting of consistency in distributed databases / V. Nikitin та ін. // Adaptive Systems of Automatic Control Interdepartmental scientific and technical collection. 2022. No 1 (40). P.48-53 URL: http://asac.kpi.ua/article/view/261646/258069;

5. Modification of hashing algorithm to increase rate of operations in NoSQL databases / V. Nikitin та ін. // Adaptive Systems of Automatic Control Interdepartmental scientific and technical collection. 2021. No 2 (39). P.39-43 URL: http://asac.kpi.ua/article/download/247395/244688;