UDC 004.65

V. Nikitin, E. Krylov

COMPARISON OF HASHING METHODS FOR SUPPORTING OF CONSISTENCY IN DISTRIBUTED DATABASES

Abstract: Distributed systems provide ability to use any services independently from users’ geolocation and to save productivity on the high level. Highly loaded systems need particularly meticulous optimization at all levels. Such systems include government, military and financial institutions, social networks, IoT and more. Data technologies are improving and giving developers more opportunities to enhance the usage of existing protocols. The growing intensity of information exchange makes it difficult to maintain consistency in distributed systems. This, in turn, may lead to the quality deterioration of end-user services or impose a number of restrictions with increasing latency. As both scenarios are undesirable, there is a need to improve parallel service processes to maintain consistency.

One of the undesirable factors is the possibility of collisions during the synchronization of distributed database nodes. Hashing is applied to quickly check the changes. This method is simple in terms of implementation, but collision is unstable, especially in conditions of different variations of input data sets. Conflict situations may not be isolated due to the rapid updating of data and changes in their volume, which in turn produces a large number of combinations. In any case, the storage of outdated data can create conflict situations at the user level. The criticality of this depends on the industry where the system operates and how important data consistency is. For example, domain DNS updates are performed once a day and in most cases this is sufficient, but in the case of military systems this is unacceptable, as it can lead to human loss and financial loss due to the use of obsolete information.

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

Introduction

Consistency creates the illusion of business continuity in the system. To support it, it is necessary to have a complex mechanism that requires strict optimization not only on a single node, but also when transporting service metrics between nodes.

Gossip protocol is applied to maintain consistency in distributed systems. It describes the interaction of nodes with each other for communication. In distributed databases, it is used to exchange information about the current state of nodes and start synchronization when discrepancies are found [1].

The Merkel tree is used to identify discrepancies in distributed databases. This structure contains summary information about the large amounts of data (such as tables or documents). The hash function for receiving the target digest has a key place.

It is mathematically known that each hash function has a collision probability, but the main issue is the computation time to find such input. This leads to the creation of new and more stable algorithms. This problem is usually solved by size enlarging the resulting digest and ensuring even distribution.

Formulation of the problem

To detect changes, it is necessary to have "fingerprints" of data from different nodes and compare them. If the "prints" are different, then the consistency is broken and you need to start the synchronization process. The Merkel tree is a data structure that contains the resulting information for certain data blocks [2]. You can see an example of this data structure in fig. 1.

The principle of operation of this structure is quite simple:

1) a hash is calculated for each data block;

2) pairwise concatenation of calculated hashes;

3) if a hash is left without a pair, it remains without concatenation;

4) each value obtained becomes an input block for finding the hash again;

5) repeat steps 1 to 4 until the root hash remains.

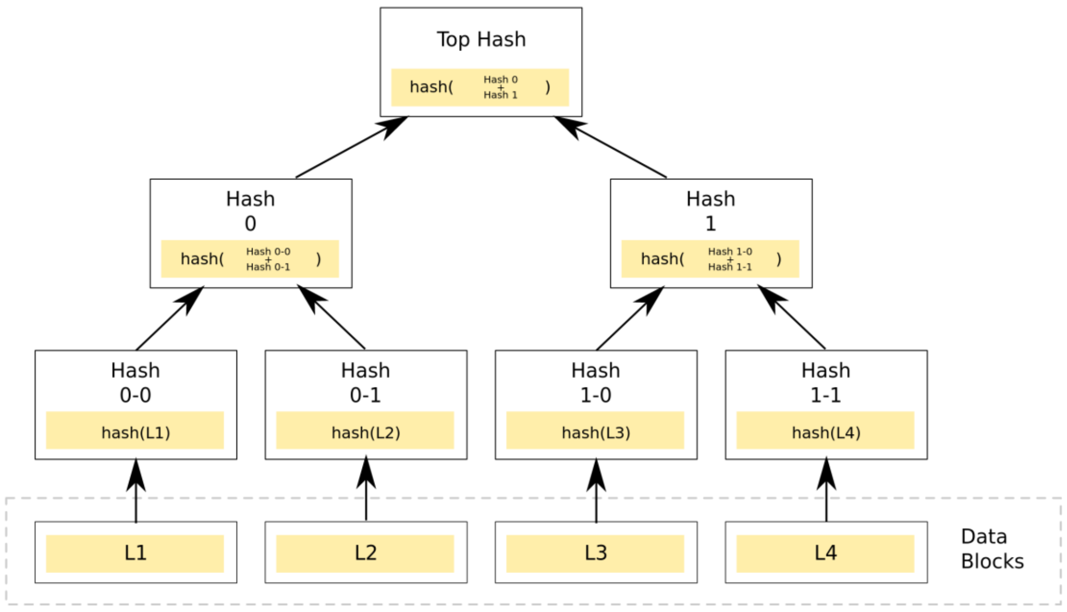

Figure 1. The structure of the Merkel tree

The resulting root hash is used to check between different nodes of the distributed system. Multi-stage hashing cannot guarantee the absence of collisions that can occur when transferring different blocks of data to a hash function. Mathematically, it is possible to calculate the total number of possible unique hash values, but it is impossible to even predict the possible number of collisions. This is a consequence of the use of algorithms that have been obtained empirically and are based entirely on the randomness of the impact of bitwise operations on the resulting set of bits.

There is a great variety of hashing algorithms and the question arises as to the feasibility of using the developed algorithm PH-1 [3]. Comparison of algorithms can be done by many criteria, but this is impractical, because the direction of research is the consistency of distributed databases, and the collision vulnerabilities of existing algorithms are of a particular interest. Also, the aim is to determine the limitations for the developed algorithm PH-1.

Collision stability of the existing hashing algorithms

Hashing is a convenient method of comparing data to determine their identity. Any destination uses this explicitly or implicitly: verification of sent files and network packets, secure storage of passwords, fast data retrieval. This leads to a wide variety of hash functions: hash functions based on division, multiplicative hashing schemes, hashing of variable length strings, cryptographic hash functions [3]. Algorithms of the latter type should be as resistant to collisions as possible, as they are used to create certificates.

Comparison with hash functions based on division

This is the simplest and a very conflict-resistant algorithm. The hash is the value that remains due to the remainder of the division by the maximum number of hashes:

$$h(K) = \text{K mod M}, \tag{1}$$

where K – input data; M – maximum number of hashes.

For the experiment we can take the function as in formula 2.

$$h(K) = \sum K[i] \ \text{mod } 70, \tag{2}$$

This function is a simplified case of functions of this kind. An arbitrary set of bytes is fed to the input, from which the sum is obtained and the operation of obtaining the remainder when dividing by 70 is performed. In the case when K is the text, the division into blocks is based on the code system.

For example, in the case of ASCII, each term is a number from 0 to 255, because 1 byte is used to encode one character. In the case of UTF-8, the character can be encoded from 2 to 4 bytes, which significantly increases the value of the term.

It is obvious that we will get the same hash value infinitely with increasing the amount by 70. The results of the calculations are given in Table 1.

You can see that all four examples lead to the same hash value using division-based algorithm. The results of PH-1 algorithm indicate that PH-1 is collision-resistant to collision vulnerabilities due to the rules of modular arithmetic, because for all the same values were obtained unique hashes.

Comparison with cryptographic hash functions

Additional requirements are imposed on cryptographic hash functions: irreversibility, resilience to collisions of the first kind and resilience to collisions of the second kind.

The most common cryptographic hashing algorithms are MD5 and the SHA family, the algorithms of which were discussed in a previous article [3].

To create a dataset, a special algorithm was applied, which uses the construction of a differential path and collision with the selected MD5 prefix. This enabled us to obtain a dataset of 1000 pairs of files; each file is 128 bytes in size. Each pair is an example of a collision, because by hashing each file from a pair and comparing it with the hash of another, the same hash values are obtained. Fig. 2 shows the results of hashing 50 pairs of such files.

Table 1.

The results of hashing using a division-based and PH-1 algorithm

Input data	Division-based hash	PH-1 hash
1000110	0	207
10001100	0	204
11010010	0	250
100011000	0	158

In fig. 2 you can see that different hash values were obtained because they have different resulting bit sequences when using PH-1. Among the obtained hash values there were the same in length, but differed radically in bit combinations. Hash values for the other 950 file pairs are also different, indicating that the PH-1 is resistant to MD5 vulnerabilities.

After research, the SHA-1 algorithm is also not resistant to collisions and there are ready algorithms for attacks on it, but the algorithm and implementation have not been made public due to the avoidance of situations using the algorithm by hackers. Currently, there are two messages in the public domain that result in the same hashes. The PH-1 algorithm was used for these two files and we can see that different hash values were obtained because they have different resulting bit sequences. The dataset from one pair is too small to draw conclusions about collision resistance, but a positive result for the only available pair gives a credit of confidence for future research in this area.

It is worth noting the shortcomings of the PH-1 algorithm. From the above mentioned results (Fig. 1, Table 1) it can be seen that the hash length is not fixed compared to the same MD5 and SHA. The problem is also not only the length but also the size of the resulting hash. It depends entirely on the size of the input data and will grow proportionally. Based on the above, it is possible to make assumptions about the limitations of PH-1:

1) hashing of small data

From the Table 1 you can see that the hashing of 32-bit data resulted in an 8-bit hash, i.e. 1 byte. This is far less than in case of using hash functions with a fixed hash length. In this case, the MD5 hash would be 128 bits in size; SHA-1 would have 160 bits; SHA-2 would have 224 bits and more [4].

2) hashing of particularly important data

Analyzing the conducted experiments, it is obvious that the algorithm has collision resistance to vulnerabilities of existing hashing algorithms and this gives credibility and feasibility to use it despite the proportional increase in the size of the hash value.

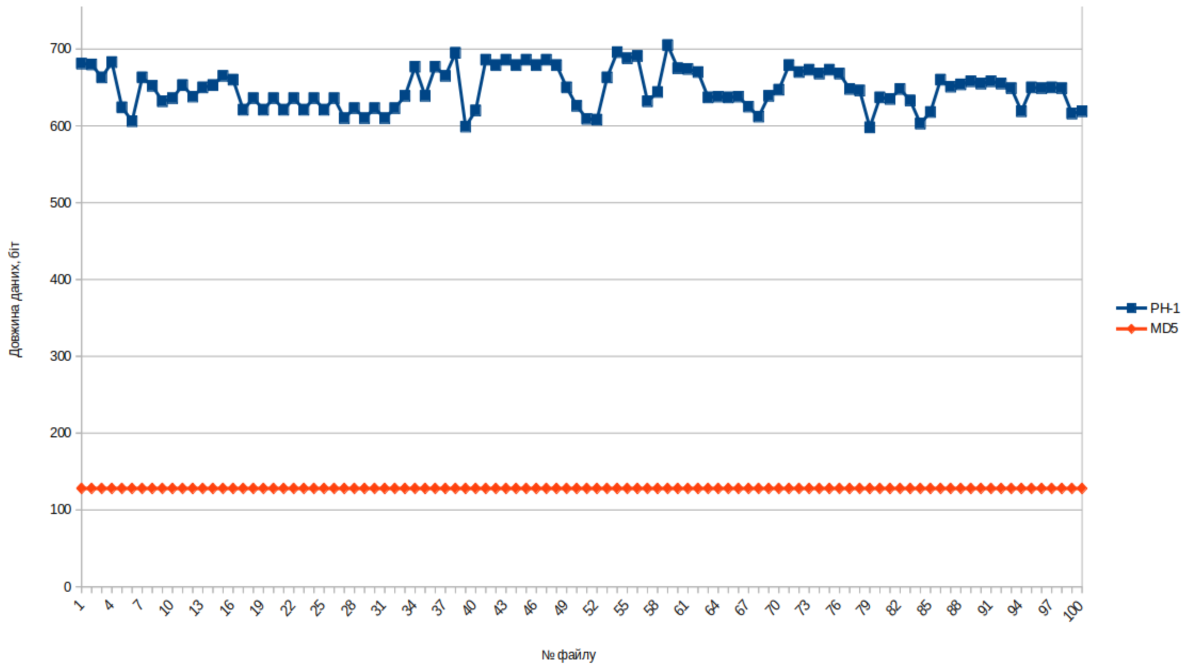

Figure 2. Comparison of hash lengths of PH-1 and MD5

CONCLUSIONS

Existing hashing algorithms are widely used. This affects collision resistance, as compressing data of any size will inevitably result in collision hashes. As a result, the consistency of distributed databases is compromised because hash functions are used to check for changes.

In practice, it was tested that the developed PH-1 algorithm has collision resistance to hash algorithm vulnerabilities, such as division-based hashing and cryptographic algorithms MD5 and SHA-1. In turn, this algorithm has a disadvantage in the form of the resulting hash value proportional to the size of the input data. This makes it impossible to use the algorithm to hash large data.

A possible use may be the hashing of small and important data. One of the possible examples of application may be maintaining consistency in distributed databases of personal information of users, bank details, analytics of financial transactions.

REFERENCES

1. Aguilera M.K., Terry D.B. The Many Faces of Consistency // IEEE Database Engineering Bulletin. 2016. № 39. C.3-13 URL: http://sites.computer.org/debull/A16mar/p3.pdf

2. Haider F. Compact Sparse Merkle Trees // OSF Preprints. 2018. № 955. C.1-8 URL: https://doi.org/10.31219/osf.io/8mcnh

3. 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. № 2 (39). C.39-43 URL: http://asac.kpi.ua/article/download/247395/244688

4. Comparison of hash function algorithms against attacks: A review / A. Maetouq та ін. // International Journal of Advanced Computer Science and Applications. 2018. № 9 (8). C.98-103 URL: https://thesai.org/Downloads/ Volume9No8 /Paper_13-Comparison_of_Hash_Function_Algorithms.pdf