An Empirical Study to Detect the Collision Rate in Similarity Hashing Algorithm Using MD5

Tushaar Gangavarapu*
Worldwide Deals, Automated Advertising
Amazon.com, Inc.
Bangalore, India
tushaargvsg45@gmail.com

Jaidhar C.D.
Department of Information Technology
National Institute of Technology Karnataka
Mangalore, India
jaidharcd@nitk.edu.in

***Abstract*—Similarity Hashing (SimHash) is a widely used locality-sensitive hashing algorithm employed in the detection of similarity, in large-scale data processing, including plagiarism detection and near-duplicate web document detection. Collision resistance is a crucial property of cryptographic hash algorithms that are used to verify the message integrity in internet security applications. A hash function is said to be collision-resistant if it is hard to find two different inputs that hash to the same output. In this paper, we present an empirical study to facilitate the detection of collision rate when SimHash is employed to check the integrity of the message. The analysis was performed using bit sequences with length varying from 2 to 32 and Message Digest 5 (MD5) as the internal hash function. Furthermore, to enable faster collision detection with more significant speedup and efficient space utilization, we parallelized the process using a distributed data-parallel approach with synchronous computation and optimum load balancing. Collision detection is desirable, owing to its applicability in digital signature systems, proof-of-work systems, and distributed content systems. Our empirical study revealed a collision rate of 0% to 0.048% in SimHash (with MD5) with the variation in the length of the bit sequence.**

***Index Terms*—Collision Rate, Collision Search, Integrity, MD5, SimHash**

I. Introduction

In today's world of open communication and computing, providing a way to check the integrity of the stored messages or transmitted messages through an unreliable medium is of vital importance [1]. The integrity of the message guarantees that the message is not tampered with, in the transit and is usually achieved by utilizing hash functions. It is quite evident from the pigeonhole principle that every hash function with fewer outputs than inputs would result in some of the inputs hashing to the same output, i.e., the collision of hashes is plausible with most hashing schemes [2], [3]. Collision resistance is a vital property of a cryptographic hash function, which ensures the *difficulty* of finding two distinct inputs that hash to the same output value. While collision resistance is desirable, it does not imply the non-existence of collisions.

Cryptographic hash functions are customarily designed to ensure collision resistance. The *birthday paradox* gives a definitive upper bound on the collision resistance, i.e., if an attacker computes $\sqrt{2^N}$ hash operations (for a hash digest of $N-$bit size) on random input, then it is likely that matching outputs exist [4]. Most hash functions including Message Digest 5 (MD5) [5] and Secure Hash Algorithm 1 (SHA-1) [6] that were estimated to be collision-resistant, were later broken [7], [8], [9]. The impact of collisions is essentially application-dependent, and determining the collision rate can help estimate the collision resistance of a hash function. The use of hash functions in the security of digital signature schemes, proof-of-work systems, distributed content systems, data integrity schemes, e-cash, group signature, and a multitude of other cryptographic protocols makes it almost mandatory to determine the collision rate of the underlying hash functions.

In 1994, MD4 [10] was broken by attacking the last two rounds [4], [11], [12]. MD5 was broken in 1998, using the modular differential attack, and the collisions can be generated in about 15 minutes to an hour [13], [8], which is estimated by exploiting the weakness in the internal structure of MD5. Other hash functions including MD4, RIPEMD [14], and HAVAL-128, [15] can also be broken using a modular differential attack [8], [7]. SHA-1 is not broken yet, but a collision was found with the complexity of less than 2^{69} hash operations [9], [4]. It can be noted that the existing literature does not provide any collision information or collision search strategies for collision detection in SimHash [16], which is a locality-sensitive hashing scheme.

For currently unbroken cryptographic hashing schemes, there do not exist any known internal structural weaknesses, thus implying that the collision rate detection is the only guaranteed way of proving their collision resistance. SimHash, developed by Moses Charikar, is widely used in detecting similarity in large-scale data processing applications. When SimHash is used to check the message integrity, the need for the detection of its collision rate becomes vital. In this paper, we present an efficient empirical analysis of the collision rates in SimHash algorithm through a distributed data-parallel dictionary-updation approach, with optimal load balancing and synchronous computation. This study employs MD5 as the internal hashing scheme, and the analysis is presented for bit sequences ranging from 2 to 32 bits in length. Furthermore, the execution time taken to measure the collision rate is also detailed, to give an overall estimate of the time taken to identify collisions in SimHash.

* work done at the National Institute of Technology Karnataka.

978-1-7281-2087-4/19/$31.00

The rest of the paper is structured as follows: Section II presents a brief overview of the SimHash algorithm. Section III reviews the existing literature and work previously carried out in this domain. The proposed methodology to compute the collision rates for SimHash is presented in great detail, in Section IV. Section V presents the obtained experimental results, followed by conclusions and discussion on future research possibilities in Section VI.

II. Background: Review of SimHash

While most hash algorithms including MD5, SHA-256, and HAVAL-128 hash different inputs (even with the slightest of the variations) to entirely different hash digests, SimHash hashes similar inputs (in terms of the Hamming distance) to similar (closer) hash digests. Consider the following example:

```
phrase1 = "magic is all within you"
phrase2 = "magic is all in you"
phrase1.MD2 = 923ce24b045b25ad82341c2a8ac65f65
phrase2.MD2 = c0d972488d0c98763ab1f596a63e35f3
hammingDistance(phrase1.MD2, phrase2.MD2) = 65

phrase1.SimHash = 2da266b7f30b82d9
phrase2.SimHash = 2da366b773a382fd
hammingDistance(phrase1.SimHash, phrase2.SimHash) = 7
```

The SimHash algorithm uses an internal hashing algorithm to hash shingles or $n-$grams obtained from a given phrase. Each hash digest corresponding to each $n-$gram is then utilized to arrive at the final similarity hash digest. Algorithm 1 details the entire procedure employed to obtain the SimHash digest for a given input phrase, the specified value of n in $n-$grams (shingle size), and the defined internal hashing scheme.

Algorithm 1: SimHash Algorithm

Input: input phrase, shingle size, hash algorithm
Output: SimHash digest

```
n−grams ← inputPhrase.shingles(shingleSize)
hashDigests ← []
for shingle ∈ n−grams do
    hashDigest ← binary(shingle.hashAlgorithm)
    hashDigests.append(hashDigest)
SimHashBits ← [0] * len(hashDigests[0])
for hashDigest ∈ hashDigests do
    for idx ← 0 to len(hashDigest) do
        if hashDigest[idx] = 1 then
            SimHashBits[idx] ← SimHashBits[idx] + 1
        else
            SimHashBits[idx] ← SimHashBits[idx] − 1
SimHashDigest ← string.empty
for idx ← 0 to len(SimHashBits) do
    if SimHashBits[idx] > 0 then
        SimHashDigest.append('1')
    else
        SimHashDigest.append('0')
return SimHashDigest
```

The hash digests obtained through SimHash for similar input phrases often have low Hamming distance and higher Jaccard similarity. This property of SimHash is extremely practical in near-duplicate detection [17], [18]. By using SimHash for near-duplicate detection, we can reduce the time complexity from O(N^2) for pair-wise comparison to O(N).

III. Related Work

In the past, many cryptographic algorithms including MD4, MD5, SHA-0, and SHA-1, were broken by exploiting the structural weaknesses of the underlying hashing schemes [8], [9], [7]. Most of the studies concerning SimHash in the existing literature aim at evaluating the applicability of this locality-sensitive algorithm to near-duplicate detection in data processing applications, including plagiarism checking [19] and email spam detection [20].

Sood and Loguinov [21] proposed a significantly faster and a greater space-efficient approach to detect similar document pairs in large-scale data collections. Their bit-flipping algorithm resulted in certain performance overhead. Fu *et al*. [22] presented a document-based query searchable encryption scheme over encrypted cloud document, based on similarity hashing and trie based indexing. Jiang and Sun [23] proposed a semi-supervised SimHash algorithm to search high-dimensional data. Their algorithm learned the optimal feature weights from prior knowledge, to relocate the data, ensuring that similar data inputs have similar hash digests. Ho *et al*. [20] employed the SimHash algorithm with a parallel processing framework and meet-in-the-middle attack, to detect spam emails.

The existing research only presents the applications of the SimHash algorithm without any reference to its collision rate (and thus, the collision resistance). Hence, we conclude that there exist no state-of-the-art studies concerning the determination of the collision rates (resistance) for the SimHash algorithm.

IV. Methodology

Collision detection aims at finding two distinct inputs (here bit sequences) hashing to the same digest. Firstly, 2^n distinct bit sequences of length (n) varying from 2 to 32 are generated. Then, the SimHashes for the generated bit sequences are computed using the procedure in Algorithm 1, with an internal hashing scheme as MD5 and the shingle size of two. All the hash digests are stored in a hash map (dictionary) to ensure a constant lookup complexity (O(1)). In the hash map, we store the obtained hash digest as the key and the count of its occurrence as the corresponding value.

The computation for lower-order sequences (up to 16 bits in length) is manageable and does not require any parallel considerations. However, for higher-order bit sequences, the computational complexity of collision detection is very high, especially in terms of the time taken. Thus, the need to parallelize the entire process of collision detection becomes more relevant when dealing with higher-order bit sequences. In this study, we employ a distributed data-parallel approach using OpenMP [24], [25], MPI [26], and multiprocessing (in Python), to reduce the time complexity of the SimHash collision detection process efficiently.

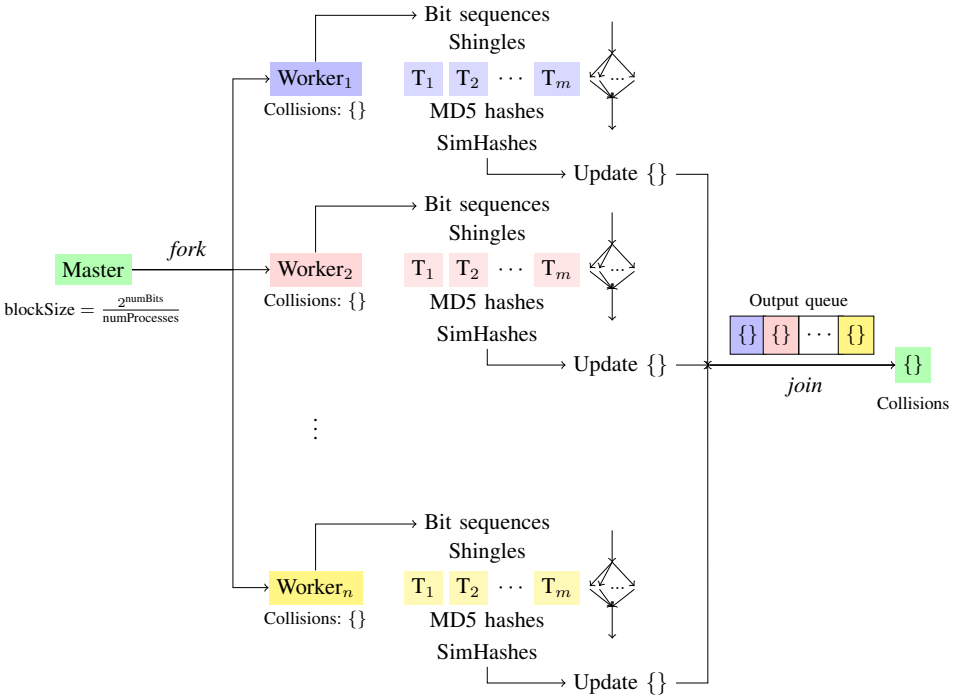

Fig. 1. Distributed data-parallel approach with optimal load balancing and synchronization to detect the collisions in SimHash (with MD5).

TABLE I
COLLISION RATE IN SIMHASH USING MD5 AS THE INTERNAL HASHING ALGORITHM.

#Bits	#Collisions	Collision rate (%)	#Processes	Time (s)
2	0	0	1	0.00007000
4	0	0	1	0.00032000
8	0	0	1	0.00613700
16	0	0	1	29.3268820
20	120	0.01144409180	4	464.700325
24	8,128	0.04844665527	4	6235.20520
28	41,523	0.01546852291	16	104092.872
32	1,438,275	0.03348744940	64	225431.219

The entire distributed data-parallel workflow employed in the detection of SimHash (with internal MD5 hashing scheme) collisions is depicted in Fig. 1. In our distributed data-parallel approach, the master process computes the block size as $\frac{2^{\text{numBits}}}{\text{numProcesses}}$. The master process (denoted as *Master*, in Fig. 1) then divides the task into several worker processes (denoted by *Worker*$_i$, $i \in [1, n]$, in Fig. 1), which then compute the workload corresponding to the predetermined block size. Different worker processes are then run on multiple machines with identical computing power. Each worker process maintains a collision dictionary into which it updates the SimHashes and their counts. The workload per worker process involves the generation bit sequences, $n-$grams (shingles), and MD5 hashes, along with SimHash computations for all the shingles. Each process spawns several threads (denoted by T_i, $i \in [1, m]$, in Fig. 1) to distribute the computation of MD5 hashes and SimHashes, thus ensuring further parallelization. Synchronization among the threads during the updation of the collision dictionary is ensured through locks. Once a worker process completes its workload, it enqueues its corresponding collision dictionary into a process output queue maintained by the master process. All the collision dictionaries from the process output queue are then merged by adding the values (counts) for same keys (SimHash digests) across various dictionaries.

Furthermore, we recorded the execution times, to measure the overall time taken in the identification of the collision rate (and thus, the collision resistance) for a given bit sequence length. Execution time for every bit sequence length (2 to 32) is collected eight times to overrule the bias caused due to any other system processes that are not under the control of the experimenter. Moreover, with every run of the experiment, the order of experimentation for a specific bit length was shuffled to ensure an unbiased measurement of the time taken. The individual measurements were then averaged to obtain the overall time taken to identify collisions accurately.

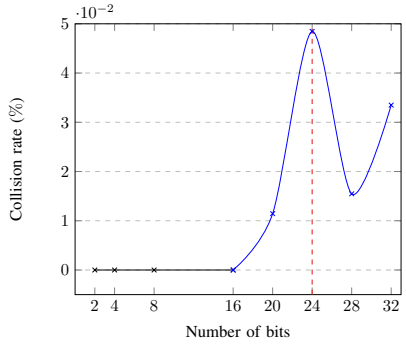

Fig. 2. A graph depicting the variation in the collision rate with the increasing number of bits.

V. Experimental Results

All the results presented in this study are obtained using multiple nearly identical machines with an i5 7200U at $4\times$ 3.1 GHz processor, an 8 GB DDR3 at 1333 MHz memory, and $10/100/1000$ Gigabit LAN network.

The collision rates are computed as $\frac{\text{numcollisions}}{2^{\text{numBits}}}$. The variation in the collision rate (%) plotted against the variation in the number of bits is presented in Fig. 2. It can be observed that the collision rates for lower-order bit sequences (2 to 16 bits) is 0%. However, a maximum collision rate of 0.048% can be observed for $24-$bit sequences (marked with a red dotted line in Fig. 2). Table I tabulates the experimental results obtained for bit sequences with length varying from 2 to 32 bits.

It is evident from Fig. 2 that the collision rate increases for higher-order collisions (with a maximum value at 24 bits). It can also be observed from Table I that the time taken in the determination of the collision rate increases exponentially with the increase in the number of bits. Approximately a duration of a day and four hours for $28-$bit sequences, and two days and 14 hours for $32-$bit sequences was required to determine their respective collision rates. Distributed data parallelization with synchronization and optimal load balancing resulted in a greater speedup and more efficient storage utilization than the sequential counterparts.

VI. Conclusions

Evaluating the collision resistance of a cryptographic hashing algorithm plays a pivotal role in applications requiring integrity, such as digital signature schemes, e-cash, and proof-of-work systems. SimHash is a widely used locality-sensitive algorithm used in many large-scale data processing applications. In this paper, we presented a distributed data-parallel framework with synchronization and optimal load balancing, to detect the collision rates of the SimHash algorithm with a more significant speedup and efficient storage utilization. We presented our analysis using bit sequences with length varying from 2 to 32 bits. It was observed that the time taken to detect the collisions increases exponentially with the increase in the number of bits. As a part of the future work, we aim at analyzing the bit patterns of the SimHash digests in great detail, to try and exploit any internal structural weaknesses.

References

[1] H. Krawczyk, M. Bellare, and R. Canetti, "Hmac: Keyed-hashing for message authentication," Tech. Rep., 1997.

[2] S. Goldwasser and M. Bellare, "Lecture notes on cryptography," *Summer course "Cryptography and computer security" at MIT*, vol. 1999, p. 1999, 1996.

[3] J. Floyd, "What do Hash Collisions Really Mean?" Jul 2008, [Online; accessed 21. Dec. 2018] URL: https://permabit.wordpress.com/2008/07/18/what-do-hash-collisions-really-mean.

[4] R. Pass, "Lecture 21: Collision-Resistant Hash Functions and General Digital Signature Scheme," *Course on Cryptography at Cornell University*, Nov 2009, uRL: https://www.cs.cornell.edu/courses/cs6830/2009fa/scribes/lecture21.pdf.

[5] R. Rivest, "The md5 message-digest algorithm," Tech. Rep., 1992.

[6] J. H. Burrows, "Secure hash standard," DEPARTMENT OF COMMERCE WASHINGTON DC, Tech. Rep., 1995.

[7] X. Wang, D. Feng, X. Lai, and H. Yu, "Collisions for hash functions md4, md5, haval-128 and ripemd." *IACR Cryptology ePrint Archive*, vol. 2004, p. 199, 2004.

[8] X. Wang and H. Yu, "How to break md5 and other hash functions," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2005, pp. 19–35.

[9] X. Wang, Y. L. Yin, and H. Yu, "Finding collisions in the full sha-1," in *Annual international cryptology conference*. Springer, 2005, pp. 17–36.

[10] R. Rivest, "The md4 message-digest algorithm," Tech. Rep., 1992.

[11] B. Den Boer and A. Bosselaers, "An attack on the last two rounds of md4," in *Annual International Cryptology Conference*. Springer, 1991, pp. 194–203.

[12] H. Dobbertin, "Cryptanalysis of md4," in *International Workshop on Fast Software Encryption*. Springer, 1996, pp. 53–69.

[13] J. Katz, A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. CRC press, 1996.

[14] H. Dobbertin, "Ripemd with two-round compress function is not collision-free," *Journal of Cryptology*, vol. 10, no. 1, pp. 51–69, 1997.

[15] J. Seberry, "Haval a one-way hashing algorithm with variable length of output 1 yuliang zheng josef pieprzyk," 1993.

[16] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. ACM, 2002, pp. 380–388.

[17] G. S. Manku, A. Jain, and A. Das Sarma, "Detecting near-duplicates for web crawling," in *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 141–150.

[18] S. Buyrukbilen and S. Bakiras, "Secure similar document detection with simhash," in *Workshop on Secure Data Management*. Springer, 2013, pp. 61–75.

[19] C. Sadowski and G. Levin, "Simhash: Hash-based similarity detection," 2007.

[20] P.-T. Ho, H.-S. Kim, and S.-R. Kim, "Application of sim-hash algorithm and big data analysis in spam email detection system," in *Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems*. ACM, 2014, pp. 242–246.

[21] S. Sood and D. Loguinov, "Probabilistic near-duplicate detection using simhash," in *Proceedings of the 20th ACM international conference on Information and knowledge management*. ACM, 2011, pp. 1117–1126.

[22] Z.-J. Fu, J.-G. Shu, J. Wang, Y.-L. Liu, and S.-Y. Lee, "Privacy-preserving smart similarity search based on simhash over encrypted data in cloud computing," *ŁŁ*, vol. 16, no. 3, pp. 453–460, 2015.

[23] Q. Jiang and M. Sun, "Semi-supervised simhash for efficient document similarity search," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Association for Computational Linguistics, 2011, pp. 93–101.

[24] B. Chanduka, T. Gangavarapu, and C. D. Jaidhar, "A single program multiple data algorithm for feature selection," in *Intelligent Systems Design and Applications*. Cham: Springer International Publishing, 2020, pp. 662–672.

[25] T. Gangavarapu, H. Pal, P. Prakash, S. Hegde, and V. Geetha, "Parallel openmp and cuda implementations of the n-body problem," in *Computational Science and Its Applications – ICCSA 2019*. Cham: Springer International Publishing, 2019, pp. 193–208.

[26] M. Snir, S. Otto, S. Huss-Lederman, J. Dongarra, and D. Walker, *MPI–the Complete Reference: The MPI core*. MIT press, 1998, vol. 1.