Research and Implementation of Chaotic Encryption for Hadoop Platform: Postprint

Authors: Xie Guobo, Yao Zhuochen

Date: 2018-09-12T00:00:00+00:00

Abstract

To address the deficiencies inherent in traditional single-machine serial encryption methods, this paper designs an operational scheme for chaotic encryption algorithms based on the Hadoop platform. Leveraging the MapReduce parallel framework along with the principles of chaotic encryption pseudo-randomness and initial-value sensitivity, we propose a parallel chaotic encryption scheme optimized specifically for the MapReduce framework and chaotic encryption. This scheme employs plaintext length as the initial value to perform preliminary iterations on three hyper-chaotic systems—Chen, Lorenz, and Rossler—individually. Concurrently, we introduce a design philosophy of partitioning plaintext data into 1 Mb blocks, where each block generates three 1 Mb-length key sequences from the Chen, Lorenz, and Rossler systems based on offset values, thereby achieving objectives such as enhanced data density security and reduced runtime memory footprint. Within this design framework, the Chen sequence is utilized for plaintext scrambling operations, the Lorenz sequence for XOR-based diffusion operations, and the Rossler sequence for modulo-assisted auxiliary diffusion operations. Experimental validation demonstrates that the algorithm, optimized according to the characteristics of both the MapReduce parallel framework and chaotic systems, not only effectively reduces memory consumption but also improves encryption speed, while the plaintext-associated encryption operations successfully achieve the goal of defending against chosen-plaintext attacks.

Full Text

Preamble

Chaotic Encryption Research and Implementation for Hadoop Platforms

Xie Guobo, Yao Zhuochen (School of Computer Science, Guangdong University of Technology, Guangzhou 510006, China)

Abstract: To address the limitations of traditional serial encryption methods in single-machine mode—such as low data security density, difficulty in meeting the efficiency requirements for encrypting exponentially growing information data, and excessive memory consumption during execution—this paper designs an operational scheme for chaotic encryption algorithms based on the Hadoop platform. Leveraging the MapReduce parallel framework along with the principles of chaotic encryption pseudo-randomness and initial value sensitivity, we propose a parallel chaotic encryption scheme optimized for the MapReduce framework and chaotic encryption. This scheme uses plaintext length as the initial value for preliminary iteration of three hyperchaotic systems (Chen, Lorenz, and Rossler). We also introduce a design concept that partitions plaintext data into 1 MB blocks, generating 1 MB-long key sequences for each of the three chaotic systems per block based on offset values. This approach enhances data security density while reducing runtime memory footprint. In our design framework, the Chen sequence performs plaintext scrambling operations, the Lorenz sequence executes XOR diffusion operations, and the Rossler sequence conducts modular auxiliary diffusion operations. Experimental results demonstrate that the optimization algorithm, tailored to both MapReduce parallel framework characteristics and chaotic system properties, effectively reduces memory usage and improves encryption speed while achieving plaintext-associated encryption operations that successfully defend against chosen-plaintext attacks.

Keywords: MapReduce; Chen; Lorenz; Rossler; hyperchaotic system; parallel encryption

0 Introduction

In recent years, with the rapid development of computer networks and application technologies, data volume in the information domain has expanded exponentially. As enterprises and individuals face privacy breaches and security threats, the demand for data security has grown increasingly urgent. Consequently, big data network security has become a hot topic in academia and a critical area for development in information security. Traditional single-machine serial encryption methods, constrained by limited memory and processor capacity, can no longer efficiently handle large-scale data encryption in terms of either performance or memory usage. For even larger data environments, computational load may far exceed the limits of a single computer. To overcome these drawbacks and enhance data protection, cloud computing has emerged, offering new approaches for big data encryption. The current standard method involves building cloud computing clusters that encapsulate individual computers as processing nodes capable of storing and processing massive datasets. This cost-effective approach has become a practical choice for many enterprises and research institutions. However, while such platforms provide efficient processing of massive data, they lack built-in high-performance encryption capabilities,

leaving sensitive research data, business information, and customer materials vulnerable to leakage. The key challenge this paper addresses is designing a parallel encryption algorithm optimized for the MapReduce framework on big data platforms.

1 Background Knowledge

1.1 Chaotic Systems

The chaotic systems employed in this paper for the Hadoop platform exhibit random behavior within deterministic systems, characterized primarily by high sensitivity to initial conditions and excellent pseudo-random properties. These features share many similarities with traditional cryptography concepts such as diffusion, keys, and iteration. Research on applying chaotic encryption theory to Hadoop big data platforms remains limited. In 2015, reference [5] proposed a parallel multi-chaotic encryption scheme based on MapReduce, utilizing low-dimensional chaotic systems (Logistic, Henon, Lorenz, Chen) to generate initial values and perturbation values for encrypting plaintext ASCII codes. In 2017, reference [6] introduced an encryption algorithm for Hadoop based on non-degenerate high-dimensional discrete hyperchaotic systems, employing stream cipher symmetric encryption and leveraging systems with all positive Lyapunov exponents for better statistical properties. Also in 2017, reference [7] presented a Hadoop architecture using the ARIA algorithm, allowing users to choose between AES or ARIA for encryption. Based on these existing algorithms, we identify several areas for improvement: low-dimensional chaotic systems have insufficiently large positive Lyapunov exponents and poor statistical properties, and previous work has not optimized for the MapReduce parallel computing framework, resulting in redundant computations for each plaintext block that consume system memory and slow down operations. To address these shortcomings, this paper designs and implements an encryption scheme based on multiple hyperchaotic systems for Hadoop. The algorithm first partitions plaintext data into 1 MB blocks and extracts four initial values based on plaintext size for 300 preliminary iterations to eliminate transient effects from the system. Due to the pseudo-random nature of chaotic systems, identical initial values and iteration counts always produce the same key sequence, enabling preliminary iterations during the MapReduce initialization phase to avoid redundant computations during parallel encryption. During each Mapper execution, Chen, Lorenz, and Rossler hyperchaotic system keys are generated sequentially based on offset values, where the Lorenz system's initial values are the 262,144th iteration output of the Chen system, and similarly for the Rossler system. For each plaintext block read by a Mapper, Chen key scrambling, Lorenz key XOR operations, and Rossler key modular operations are performed based on offset values. Finally, the Reducer concatenates the ciphertext blocks. Our encryption algorithm not only optimizes for the MapReduce framework but also leverages chaotic system properties to effectively improve encryption speed and reduce memory usage.

1.1.1 Chen Hyperchaotic Mapping The Chen hyperchaotic system [10] can be described as:

$$\begin{cases} \dot{x} = a(y - x) + w \\ \dot{y} = dx - xz + cy \\ \dot{z} = xy - bz \\ \dot{w} = yz + cw \end{cases}$$

where a, b, c, d are system parameters. When $a = 35$, $b = 3$, $c = 12$, $d = 7$, the chaotic system solution is shown in [Figure 1: see original paper].

1.1.2 Lorenz Hyperchaotic Mapping The Lorenz hyperchaotic system [1] can be described as:

$$\begin{cases} \dot{x} = a(y - x) + w \\ \dot{y} = cx - xz - y \\ \dot{z} = xy - bz \\ \dot{w} = xz + rw \end{cases}$$

where a, b, c, r are system parameters. When $a = 10$, $b = 8/3$, $c = 28$, $r = -1$, the system enters a chaotic state, as shown in [Figure 2: see original paper].

1.1.3 Rossler Hyperchaotic Mapping The Rossler hyperchaotic system [10] can be described as:

$$\begin{cases} \dot{x} = -y - z \\ \dot{y} = x + ay + w \\ \dot{z} = b + xz \\ \dot{w} = cz + dw \end{cases}$$

where a, b, c, d are system parameters. When $a = 0.25$, $b = 3$, $c = 0.5$, $d = 0.05$, the system enters a chaotic state.

2 MapReduce Parallel Multi-Chaotic Encryption Scheme

The parallel multi-chaotic hyperchaotic encryption scheme based on MapReduce proposed in this paper features optimization for the MapReduce framework and multiple positive Lyapunov exponents from multi-chaotic systems. Compared with conventional encryption algorithms, it offers larger key space, higher security, smaller memory footprint, and faster encryption speed.

2.1 Specific Encryption Scheme

Our multi-chaotic encryption scheme employs three hyperchaotic systems (Chen, Lorenz, Rossler) to encrypt plaintext through scrambling, forward/reverse XOR operations, and forward/reverse modular operations. Multiple hyperchaotic systems not only provide more positive Lyapunov exponents but also require fewer iterations for large-volume plaintext encryption compared to single chaotic sequences. Single chaotic sequences suffer from increasing memory usage and decreasing encryption efficiency as plaintext volume grows, making them unsuitable for massive data sources on big data platforms. Partitioning plaintext into blocks and generating key sequences based on offset values effectively solves this problem.

The specific encryption scheme proceeds as follows:

a) Partition the plaintext into N blocks of 1 MB each. The first $N-1$ blocks are exactly 1 MB, with the Nth block containing the remaining data. Simultaneously, extract the thousands, hundreds, tens, and units digits from the plaintext size as input to the chaotic systems for 300 preliminary iterations to skip the transient state.

b) For each block, generate key sequences using Chen, Lorenz, and Rossler chaotic mappings. All three systems are four-dimensional continuous hyperchaotic systems solved using the Euler method. Each iteration generates four random sequences, totaling twelve chaotic sequences:

$$X_C^1, X_C^2, X_C^3, X_C^4,$$
$$X_L^1, X_L^2, X_L^3, X_L^4,$$
$$X_R^1, X_R^2, X_R^3, X_R^4$$

Each sequence undergoes 262,144 iterations (1/4 MB). The Lorenz system input is the 262,144th iteration output of the Chen system, and the Rossler system input is the 262,144th iteration output of the Lorenz system.

c) The generated twelve chaotic sequences still exhibit deficiencies, including poor uniform distribution characteristics, local monotonicity, and high inter-sequence correlation. Therefore, preprocessing is required before encryption to remove integer portions and unify the value range, then shift the decimal point right by 9 digits to enhance irregularity and overall distribution uniformity [5]. The preprocessing operation is:

$$\begin{cases} X_i = (x_i \times 10^9) \bmod 127 \\ Y_i = (y_i \times 10^9) \bmod 127 \\ Z_i = (z_i \times 10^9) \bmod 127 \\ W_i = (w_i \times 10^9 + 500) \bmod 127 + 500 \end{cases} \qquad i = 1, 2, \ldots, 262143$$

After preprocessing, according to MapReduce data reading logic, we take the ASCII code M of each plaintext line and its offset value to select corresponding sequences for scrambling, followed by forward/reverse XOR diffusion, and finally forward/reverse modular diffusion for secondary interference encryption. The algorithm flowchart is shown in [Figure 3: see original paper], with specific operations as follows:

a) Scrambling operation:

$$t = A(i); \quad A(i) = A(S(i)); \quad A(S(i)) = t; \quad i = 1, 2, \ldots, 262144$$

where

$$S(i) = \begin{cases} X_C^1(i) & \theta \in [0, 262144) \\ X_C^2(i) & \theta \in [262144, 524288) \\ X_C^3(i) & \theta \in [524288, 786432) \\ X_C^4(i) & \theta \in [786432, 1048576) \end{cases}$$

b) Forward/reverse XOR operations:

(a) Forward XOR:

$$C_i^1 = C_{i-1} \oplus S(i) \oplus M_i, \quad i = 1, 2, \ldots, 262144$$

where

$$S(i) = \begin{cases} X_L^1(i) & \theta \in [0, 262144) \\ X_L^2(i) & \theta \in [262144, 524288) \\ X_L^3(i) & \theta \in [524288, 786432) \\ X_L^4(i) & \theta \in [786432, 1048576) \end{cases}$$

(b) Reverse XOR:

$$C_i^1 = C_{i-1} \oplus S(i) \oplus M_i, \quad i = 1, 2, \ldots, 262144$$

c) Forward/reverse modular operations:

(a) Forward modular:

$$C_i^1 = (C_{i-1} + S(i) + M_i) \bmod 127, \quad i = 1, 2, \ldots, 262144$$

where

$$S(i) = \begin{cases} X_R^1(i) & \theta \in [0, 262144) \\ X_R^2(i) & \theta \in [262144, 524288) \\ X_R^3(i) & \theta \in [524288, 786432) \\ X_R^4(i) & \theta \in [786432, 1048576) \end{cases}$$

(b) Reverse modular:

$$C_i^1 = (C_{i-1} + S(i) + M_i) \bmod 127, \quad i = 1, 2, \ldots, 262144$$

After these encryption operations, the final ciphertext is obtained.

2.2 MapReduce Implementation of the Encryption Scheme

In the MapReduce parallel framework, the primary components are Mapper and Reducer. Mapper serves as the base class, and by inheriting Mapper and overriding the map method, we achieve line-by-line data reading and processing. The map method divides data into multiple subtasks assigned to multiple processors for simultaneous computation. The number of Mappers is determined by the number of blocks, with each Mapper completing encryption operations for its subtask. Finally, the Reducer integrates all Mapper outputs. In our proposed parallel encryption scheme, each block's encryption is completed independently without interference between blocks. Therefore, the MapReduce implementation consists of three parts: Split, Map, and Reduce.

2.2.1 Split We partition input data into 1 MB blocks. An N MB file is divided into N blocks.

Algorithm 1: Split Algorithm

```
Input: File to be encrypted
Output: 1 MB file blocks

InputFormat()
isSplitable()
getSplits()

if((length != 0) && isSplitable()) {
    blocksize = 1048576
    return splits
}
```

For the first $N-1$ blocks, each is exactly 1 MB and transmitted as key-value pairs to N Mappers. Each key (K) represents the byte offset of the plaintext block, and value (V) contains the actual plaintext data. This allows each Mapper to perform encryption processing in parallel without interference.

2.2.2 Mapper After splitting, each Mapper independently computes a portion of the plaintext. No communication occurs between Mappers. Each Mapper performs the operations described in Section 2.1(b) through 2.1(d). When the offset equals 0 or $1048576 - (\text{mod } 1048576) < n$, operation (b) is performed once. Then, based on the offset value of each input data line, the corresponding key sequence is located for scrambling, forward/reverse XOR, and forward/reverse modular encryption processing. The output is a K-V pair where K is the offset and V is the encrypted partial ciphertext.

Algorithm 2: Mapper

```
Input: <LongWritable, Text>
Output: <LongWritable, Text>
```

```
if(offset == 0 && 1048576 - (mod 1048576) < n) {
    ChenGen()
    LorGen()
    RossGen()
    C = MessUp(M, X_C);
    C = Xor(C, X_L);
    C = Mod(C, X_R);
}
```

2.2.3 Reducer After all Mappers complete their local ciphertext encryption, the Reducer sorts the input based on the keys from the Mappers. The sorted ciphertext constitutes the complete encrypted text, which is output as the final result.

2.3 Algorithm Security Analysis

2.3.1 Key Space The parallel multi-hyperchaotic encryption scheme based on MapReduce leverages the inherent security of hyperchaotic systems, complemented by the independent computation principle of each Mapper in MapReduce. This approach not only meets the encryption requirements for massive data but also maintains the security properties of hyperchaotic systems without negative impact. Statistical analysis shows the encryption scheme's key includes: initial iteration counts (N_1, N_2, N_3) for the three hyperchaotic systems, plaintext-associated initial values and parameters $(A, B, C, R, A', B', C', R', A'', B'', C'', R'')$, and the plaintext length N.

Assuming their values are:

$$(N_1, N_2, N_3) = (300, 300, 300), \quad N = (35, 3, 12, 7, 10, 8/3, 28, -1, 0.25, 3, 0.5, 0.05)$$

For the first $N-1$ blocks, each block is 1 MB. The required storage space is 10^{700} bytes, or 85,600 bits, yielding 2^{85600} possible combinations. This key space is sufficiently large to resist brute-force attacks, and grows exponentially with increasing initial values.

The statistical distribution of ciphertext encrypted with this multi-hyperchaotic algorithm is shown in [Figure 4: see original paper]. The histogram demonstrates that the statistical properties change significantly after encryption.

2.3.3 Initial Value Sensitivity The algorithm exhibits excellent initial value sensitivity. Since text data is less intuitive than image data for observing this property, we compare ciphertext correlation using initial values of 1 and 1.00000001 to encrypt the same file. The resulting ciphertexts are shown in [Figure 5: see original paper], with a correlation of only 0.05130069 between them. The correlation comparison is illustrated in [Figure 6: see original paper].

2.3.4 Correlation Analysis MapReduce-based chaotic encryption differs fundamentally from traditional single-machine algorithms in structure. Therefore, examining inter-block correlation is crucial for security analysis. When plaintext data across all blocks is identical ([Figure 7: see original paper]), the post-encryption inter-block correlation is shown in [Figure 8: see original paper]. The results demonstrate minimal correlation between ciphertext blocks.

2.3.5 Known-Plaintext and Chosen-Plaintext Attacks In encryption processes, key parameters $\chi_i = (A, B, C, R, A', B', C', R', A'', B'', C'', R'')$ remain constant across encryptions. For plaintext-unassociated algorithms, any plaintext encryption uses the same key sequence:

$$K^e = \{K^e(0), K^e(1), \ldots, K^e(N)\}$$

An attacker can obtain the equivalent key K_k^e in two steps. If the first encrypted plaintext is M_1 with ciphertext C_1, then:

$$C_{1i} = ((C_{1i-1} \oplus S(i) \oplus M_{1i}) \mod 128), \quad i = 1, 2, \ldots, 262144$$

The attacker can derive K_k^e through inverse operations. Without plaintext-associated key generation, the equivalent key remains unchanged for the n-th encryption plaintext M_n, yielding ciphertext C_n that can be decrypted using the same key.

Our encryption scheme uses the ASCII code of the plaintext's first letter as key input to chaotic sequence computation, making the key variable for different plaintexts rather than constant. This plaintext-associated approach provides resistance against known-plaintext and chosen-plaintext attacks.

3 Experiments

3.1 Experimental Environment

The Hadoop big data platform employs a Master-Slaves architecture consisting of 4 nodes: 1 master and 3 slaves. Hardware configuration includes Intel i3-8100 CPU (3.60 GHz/6MB Cache), 24 GB RAM, 256 GB SSD, and Gigabit Ethernet. The software environment comprises Linux Ubuntu 16.04, Hadoop 2.6.5, Java JDK 1.7.0_{79}, and Eclipse Luna Service Release 2 (4.4.2).

3.2 Execution Efficiency Analysis

When running encryption algorithms on a single computer, large ciphertext computations are limited by machine performance, with significant impacts on memory usage and speed, often resulting in memory overflow failures. The MapReduce-based chaotic encryption algorithm effectively resolves this issue. Using our design to encrypt files of 64 MB, 128 MB, and 256 MB, we compared three modes: single-machine, standard MapReduce, and optimized MapReduce.

The encryption time and memory usage are shown in [Figure 9: see original paper] and [Figure 10: see original paper], with statistical data in and .

The results demonstrate that the MapReduce-optimized chaotic encryption algorithm achieves smaller memory footprint and faster encryption speed compared to both single-machine mode and MapReduce's default line-by-line reading approach. Single-machine encryption of 128 MB plaintext failed due to memory overflow, and even 64 MB file loading was problematic. The optimized MapReduce parallel algorithm generates 1 MB key sequences per block, reducing redundant iterations and enabling memory clearance after encryption, thus significantly decreasing runtime memory usage. In the Reduce stage, using a single core for ciphertext concatenation results in similar time requirements across different file sizes.

4 Conclusion

This paper proposes a parallel encryption algorithm optimized for MapReduce and chaotic systems, implementing it within the MapReduce framework. Experimental results demonstrate that the algorithm can efficiently encrypt massive datasets, reducing encryption time, improving efficiency, and solving runtime memory consumption issues. The innovation lies in using multiple hyperchaotic systems with multiple positive Lyapunov exponents for better statistical properties, and modifying MapReduce's default line-by-line reading approach to enable block-based encryption aligned with the framework's characteristics. By scientifically allocating Mappers to process plaintext blocks and generating appropriately-sized key sequences for each block, the algorithm reduces redundant iterations while enabling effective memory redistribution. Experiments show that compared with similar encryption schemes, our algorithm enhances ciphertext security, achieves superior encryption speed, and reduces runtime memory usage, providing an ideal memory footprint for big data applications.

References

[1] Zhang Yong. Chaotic digital image cryptosystem [M]. Beijing: Tsinghua University Press, 2016.

[2] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters [J]. Communications of the ACM, 2008, 51(1): 107-113.

[3] Sun Xie Hua. Image encryption algorithms and practices with implementations in C# [M]. Beijing: Science Press, 2013.

[4] Xiao Feng, Zhang Lili, Feng Fei. Parallel multi-channel color image encryption based on hybrid chaotic system [J]. Microelectronics and Computer, 2016, 33(8): 76-81.

[5] Wang Xingyu, Yang Geng, Min Zhaoe. Parallel mixed chaotic encryption scheme on MapReduce [J]. Application Research of Computers, 2015, 32(6):

1757-1760.

[6] Wen Heping, Yu Simin, Lyu Jinhu. Encryption algorithm based on Hadoop and non-degenerate high-dimensional discrete hyperchaotic system [J]. Physics Letters A, 2017, 66(23): 76-89.

[7] Song Youngho. Design and implementation of HDFS data encryption scheme using ARIA algorithm on Hadoop [C]// Proc of IEEE International Conference on Big Data & Smart Computing. 2017: 13-16.

[8] Wang Xizhong, Chen Deyun. A parallel encryption algorithm based on piecewise linear chaotic map [J]. Mathematical Problems in Engineering, 2013, 2013(5): 497-504.

[9] Cai GuoLiang, Huang JuanJuan. Synchronization for hyperchaotic Chen system and hyperchaotic Rössler system with different structure [J]. Physics Letters A, 2006, 55(8): 3997-4004.

[10] Wang Xingyuan, Zhang Yingqian, Bao Xuemei. A novel chaotic image encryption scheme using DNA sequence operations [J]. Optics and Lasers in Engineering, 2015, 2015(73): 53-61.

[11] Wang Xingyuan, Liu Lintao, Zhang Yingqian. A novel chaotic block image encryption algorithm based on dynamic random growth technique [J]. Optics and Lasers in Engineering, 2015, 2015(66): 10-18.

[12] Zhang Yingqian, Wang Xingyuan. A new image encryption algorithm based on non-adjacent coupled map lattices [J]. Applied Soft Computing, 2015, 2015(26): 10-20.

[13] Wang Xingyuan, Feng Chen, Tian Wang. A new compound mode of confusion and diffusion for block encryption of image based on chaos [J]. Communications in Nonlinear Science and Numerical Simulation, 2010, 15(9): 2479-2485.

[14] Liu Hongjun, Wang Xingyuan. Color image encryption based on one-time keys and robust chaotic maps [J]. Computers & Mathematics with Applications, 2010, 59(10): 3320-3327.

Note: Figure translations are in progress. See original paper for figures.