*Intelligent Automation & Soft Computing*
DOI:10.32604/iasc.2021.019201
***Article***

Tech Science Press

# A Resource-constrained Edge IoT Device Data-deduplication Method with Dynamic Asymmetric Maximum

**Ye Yang[1], Xiaofang Li[2], Dongjie Zhu[3,*], Hao Hu[3], Haiwen Du[4], Yundong Sun[4], Weiguo Tian[3], Yansong Wang[3], Ning Cao[1] and Gregory M.P. O'Hare[5]**

[1]School of Artificial Intelligence, Wuxi Vocational College of Science and Technology, Wuxi, 214028, China
[2]Department of Mathematics, Harbin Institute of Technology, Weihai, 264209, China
[3]School of Computer Science and Technology, Harbin Institute of Technology, Weihai, 264200, China
[4]School of Astronautics, Harbin Institute of Technology, Harbin, 150001, China
[5]School of Computer Science, University College Dublin, Dublin, Dublin4, Ireland
[*]Corresponding Author: Dongjie Zhu. Email: zhudongjie@hit.edu.cn
Received: 06 April 2021; Accepted: 13 May 2021

**Abstract:** Smart vehicles use sophisticated sensors to capture real-time data. Due to the weak communication capabilities of wireless sensors, these data need to upload to the cloud for processing. Sensor clouds can resolve these drawbacks. However, there is a large amount of redundant data in the sensor cloud, occupying a large amount of storage space and network bandwidth. Deduplication can yield cost savings by storing one data copy. Chunking is essential because it can determine the performance of deduplication. Content-Defined Chunking (CDC) can effectively solve the problem of chunk boundaries shifted, but it occupies a lot of computing resources and has become a bottleneck in deduplication technology. This paper proposes a Dynamic Asymmetric Maximum algorithm (DAM), which uses the maximum value as the chunk boundaries and reducing the impact of the low-entropy string. It also uses the perfect hash algorithm to optimize the chunk search. Experiments show that our solution can effectively detect low-entropy strings in redundant data, save storage resources, and improve sensor clouds system throughput.

**Keywords:** Internet of Things; Sensor clouds; Intelligent transportation; Data deduplication

## 1 Introduction

With the rapid development of intelligent transportation, wireless sensors are widely used in fields such as automatic driving [1], noise monitoring [2], and traffic flow detection [3,4]. IDC (International Data Corporation) shows the global data storage capacity will be reaching 175ZB by 2025. Among them, more than 79ZB of data will be generated by Internet of Things (IoT) devices. We are living amid an era of innovation by the Internet of Everything. The IoT has millions of edge devices that are connected, and these devices will generate data continuously every day. The intelligent traffic management system [5] deploys many wireless sensor devices in the areas [6,7] that need to be monitored and obtains real-time

data according to actual scenarios. The wireless sensors in the vehicle map the environment during driving and need to collect a large amount of wireless sensor data (for example, a video stream with a data rate of about 1GB/s), communicate with other IoT devices, share data, and provide users with practical information [8–10]. The amount of data obtained from wireless sensors will increase geometrically as the number of devices increases, and a large amount of data cannot meet the application requirements of the device. The shortcomings of the wireless sensor's weak computing power, limited computing, and communication capabilities cannot support data storage and processing. Due to limited bandwidth resources, advanced technologies such as edge computing and fog computing cannot be fully utilized [11–14]. Compared with resource-constrained IoT devices, resource-rich cloud platforms are more suitable for data storage [15]. In the upcoming 5G era, cloud data transmission is more convenient [16–18]. People will become accustomed to store their data to the cloud storage systems for data management. In the existing scenario, the data collected by massive wireless sensors have a concurrency bottleneck, which causes the remote data upload to fail [19]. To bring a good experience to users, researchers optimized the data access from the perspective of data caching [20] and prefetch grouping [21,22] and improved the user experience of the sensor clouds platform. However, there is a large amount of redundant data between different sensor clouds users, which occupies many system resources, resulting in a reduction in system throughput, which has become a severe challenge. The deduplication technology can detect and eliminates redundant data, which can greatly save storage resources for storage systems in IoT scenarios. Research from Microsoft shows that eliminating redundant file chunks in a data storage system can save three-quarters of the data storage space [23].

The traditional deduplication methods are file-level deduplication. It checks whether the backup file is the same as the index backup [24]. If the fingerprints of chunks are distinct, it will be storing the file and update the index, otherwise only point out the location directly where the file is storing. In form, it is like the hard link of the file system; an index node can correspond to multiple directory files. The chunk-level deduplication technology divides the file into data chunks and checks whether the data chunk is like the previous data chunk. If it is distinct, the data chunk storing in the disk; otherwise, it is not stored. A small number of changes in the file may cause the file-level deduplication to restore the file. Chunk-level deduplication has a finer granularity than file-level approaches. More redundant data can be found, so it is widely used in cloud storage. Chunk-level deduplication technology divides a file into multiple data chunks. It calculates a hash for each chunk (e.g., MD5 [25], SHA-1 [26], SHA-256 [27]) to form a fingerprint, which is used as a unique chunk identifier. If the two data chunks have the same fingerprint, they will not be stored; otherwise, the unique data chunks will be stored.

Although the chunk-level deduplication system can detect more redundant data, it requires more computational time, which has become the bottleneck of the chunk-level deduplication technology. The reason is that the chunk-level deduplication technology is divided into the following four stages: (1) Content-Defined Chunking. (2) Generating fingerprints. (3) Looking up the index. (4) Storing data chunks. Among them, the chunk and fingerprint index consume many system resources in chunk-level deduplication, which is the root cause of the decline in the system throughput. The current mainstream chunk algorithm is divided into Fixed-Size Chunking (FSC) and Content-Defined Chunking (CDC). FSC approaches divide the file or data stream into data chunks of uniform length. The advantage is that the computational cost is small, and the system throughput is high, while one of the disadvantages is content shifted. For example, if you add or delete a byte anywhere in the file, all the chunk boundaries may change, which will affect the definition of the chunk boundary. It cannot find duplicate data chunks in a small number of modified duplicate files.

CDC approaches can effectively solve the problem of chunk shifted. The algorithm uses bytes that meet predetermined conditions as chunk boundaries. Research shows that in most data sets, the CDC approaches can detect more redundant data than FSC approaches [28]. Most systems use the content-defined chunking

based on Rabin, but this algorithm has an enormous computational cost. It cannot eliminate the impact of low-entropy strings on redundant data. There may be many low-entropy strings in the real data. These low-entropy character strings which contain few characters, but there are many redundant bytes. It will affect the selection of chunk boundaries, resulting in a reduction in the detection rate. The chunk and fingerprint index search also limit the throughput of deduplication. The capacity of the fingerprint index of a large-scale deduplication system can easily exceed the primary storage capacity, so it is necessary to store the fingerprint index on the disk. Still, random I/O reads will limit the system performance.

## 2 Background and Motivations

In this section, we will introduce the challenges brought by the existing chunk-level deduplication algorithm, provide the necessary background introduction for our proposed RAM optimization algorithm, and stimulate our research results through analysis and observation.

### *2.1 Background*

Intelligent transportation systems are currently being widely used in people's daily lives. Their goal is to use public resources to provide people with high-quality public services [29]. The intelligent transportation system deploys many wireless sensors in every corner of the city, forming a heterogeneous communication system. The context provider collects wireless sensor data to provide users with useful information. However, the edge-constrained IoT devices have the characteristics of limited storage capacity and computing power, and can no longer meet the demand for increasingly large amounts of data. The sensor clouds can solve the shortcomings of wireless sensors. Wireless sensors transmit data to the cloud, and the cloud platform performs data cleaning, processing, and storage of these data, reducing the burden of resource-constrained IoT devices. However, there is a large amount of redundant data in the massive data, which will cause the consumption of storage space and communication bandwidth.

Data deduplication has been used in many cloud storage products, such as Microsoft, EMC, Dropbox [30]. Deduplication technology saves storage space by eliminating redundant data and keeping the only copy. Chunking is a crucial part of deduplication technology. It helps save time and space, thus improve the user's experience by choosing the right partition method. The chunking approaches can be divided into two categories: Fixed-Size Chunking (FSC) and Content-Defined Chunking (CDC). We will detail the difference between the two and the current advanced research methods in 2.1.1 and 2.1.2 below.

#### *2.1.1 FSC*

FSC is common, but it has the problem of boundary shifted. Fig. 1 illustrates how the chunk boundaries shift happens.

The data stream is composed of hexadecimal bytes. The fixed chunk size is 8 bytes. Blue bytes represent chunk boundaries, while the red represents the insertion of new byte, and pink represents the location of chunk boundaries in data stream A. Data stream B has an 0xA8 inserted. It is clearly shown that the chunk boundaries of Chunk 0-4 have shifted, and even Chunk 5 appeared. In the data stream C, there is also a byte inserted in the C header, but byte 8A in the header of Chunk 1 was deleted by coincidence, and the chunk boundary of Chunk 2-4 does not shift. DataStream D and data stream A have a deduplication rate of only 25%, which will undoubtedly reduce the deduplication rate.

#### *2.1.2 CDC*

The most widely used CDC approach is the Rabin fingerprint-based CDC algorithm [31,32]. The Rabin algorithm uses polynomials to cooperate with sliding windows to implement fingerprint calculation, as shown in Fig. 2.

$$Rabin(b_0, b_1, \ldots, b_\beta) = \left( \sum_{x=0}^{M} b_0 p^{\beta - x} \right) \bmod r \tag{1}$$

where $\beta$ is the sliding window length, $p$ is a predefined constant, and $r$ is the predefined average chunk size. The sliding window continuously moves on the data stream. When the fingerprint of the next window needs to be calculated, only the calculation formula (2) needs to be incremented.

$$Rabin(b_i, b_{i+1}, \ldots, b_{i+\beta-1}) = [Rabin(b_{i-1}, b_i, \ldots, b_{i+\beta-2}) * p - b_{i-1} p^M + b_{i+\beta-1}] \bmod r \tag{2}$$

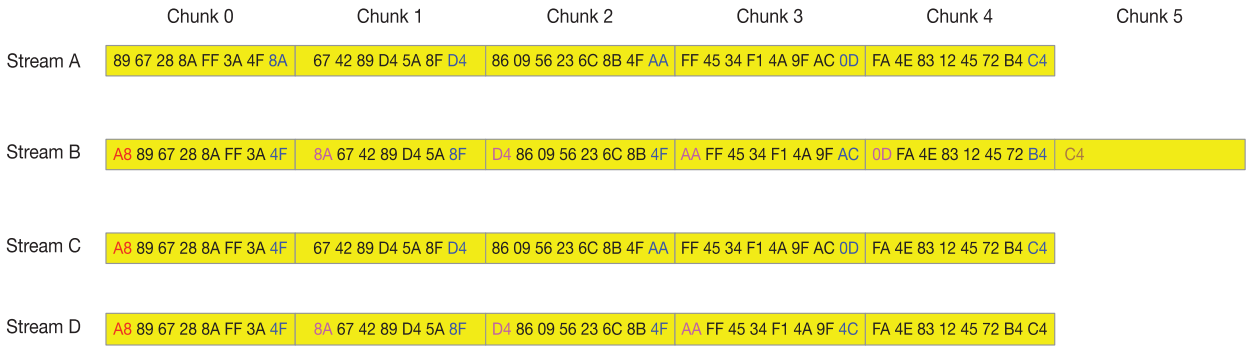


**Figure 1:** FSC approach boundary shifted when inserting or deleting happens

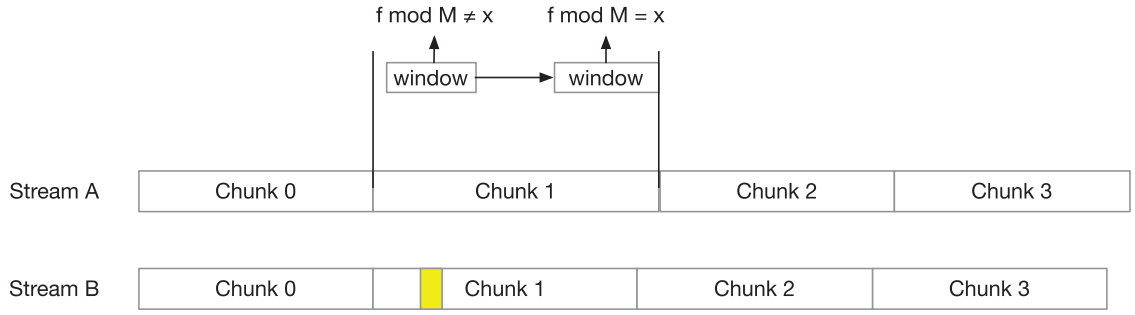


**Figure 2:** An example based on the Rabin algorithm. If the hash value f of the sliding window satisfies the predetermined condition, the mark position is used as the chunk boundary, otherwise, the chunk boundary is found with the sliding window byte by byte

Although the Rabin fingerprint algorithm can effectively solve the problem of chunking shift, it has the disadvantages of large calculation amount and low throughput. The chunking stage becomes the bottleneck of the deduplication system. Moreover, in extreme cases, such as frequent encounters of chunk boundaries or encountering low-entropy strings, the selection of chunk boundaries will be affected, resulting in uneven distribution of chunk sizes, thereby reducing the efficiency of deduplication.

LMC [33] slides two symmetric fixed windows on the data stream to find whether the central byte value is the extreme point and uses the local extreme point as a chunk boundary. It solves the shortcomings of chunk size variance with extremely low computational overhead. However, when the extreme point is found, the data stream needs to be traced back to determine whether the point is an extreme point, which reduces the throughput of the chunk algorithm. AE [34] uses an asymmetric extreme value method, using a sliding window and a fixed window to determine the split point. The split point is in the middle of the sliding window and the bald window, reducing the number of byte comparisons, thereby reducing the

computational overhead. RAM [35] places extreme value on the boundary of the chunk and uses a fixed window and a sliding window. The sliding window looks for a byte larger than any value in the fixed window as the chunk boundary. The details are shown in Fig. 3.

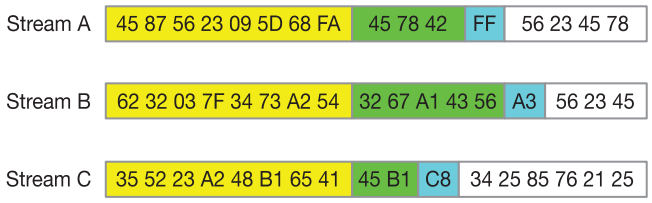


**Figure 3:** The example of how RAM algorithms work

### *2.2 Motivation*

To verify the deduplication rate between fixed and content-based chunking deduplication. We used two versions of Linux system images for deduplication detection. As shown in Fig. 4, we define DER as the ratio of the input data size to the actual data size to be stored. The larger the value, the better. The experimental results show that the duplicate data detected using the FSC approaches is much smaller than the CDC approaches. This is due to the change at the 432nd byte, which generates a chunk shift and reduces the rate of duplicate data detection. Compared with the FSC approaches, the CDC approaches can detect more redundant data, so it has been used in many deduplication systems. Still, in large-scale data storage, the CDC approaches require a lot of consumption computing resources seriously to reduce the throughput of the storage system.

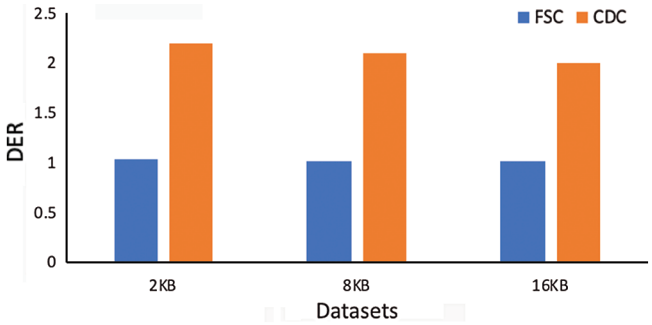


**Figure 4:** Comparison of deduplication rate between FSC and CDC

Tab. 1 shows the current advanced CDC approaches and compares their performance from multiple perspectives. The FSC algorithm has a high throughput rate, but it cannot effectively eliminate redundant data. The Rabin and LMC algorithms can effectively identify redundant data, but the throughput rate is low. And it cannot effectively eliminate low-entropy strings. The AE algorithm is superior to the Rabin algorithm in throughput, and the AE algorithm can effectively eliminate low-entropy character strings. RAM has a high throughput rate, but it cannot effectively eliminate low-entropy strings. Therefore, how to accelerate the speed of CDC and fingerprint indexing and increase the deduplication rate is still a challenge.

The low-entropy string mentioned in Section 1 has two patterns [14]. The first pattern consists of the same character, such as aaaaaaa, and the second pattern consists of a series of repeated substrings, such as abcdabcdabcdabcd. Due to the existence of low-entropy strings, the selection of chunk boundaries can be affected, which directly leads to a reduction in the deduplication rate. Therefore, how to detect more

low-entropy character strings in the stored data to improve the deduplication rate is currently a research trend. After the completion of partitioning, how to effectively process the block index is also a significant research field. Because massive amounts of data generate huge data chunks, it is difficult to find a unique chunk identifier to determine whether the data is stored within a limited time.

**Table 1:** A comparison between existing CDC approaches

| Algorithms | Content Defined | Throughout | Efficient for low-entropy string |
|---|---|---|---|
| FSC | No | High | No |
| Rabin | Yes | Low | No |
| LMC | Yes | Low | No |
| AE | Yes | Mid | Yes |
| RAM | Yes | High | No |

Currently, bloom filters [36] and hash tables [37] are mainly used for fingerprint indexing. Bloom filter is a space-efficient probabilistic data structure, so it is often used as an index structure. The bloom filter uses multiple hash mapping functions and a bit array. When an element inserts in a bloom filter, it will be hashed multiple times to map the element to the corresponding bit and set to 1. When checking whether an element exists in the collection, only when all the mapping bits are 1, it means that the element may exist in the collection. Traditional bloom filters can optimize cache hit rate and disk access. Unfortunately, as the number of inserted elements increases. Thus, the error rate goes up. When multiple elements are mapping the same bit, it may be an error to check whether the element exists. Today's network applications and services demand more and more memory space, and bloom filters are widely in many applications. But it may require a huge memory space. For example, data chunks need the three hash calculations, and three bits need mapping to the physical addresses of files or chunks. 16TB of data requires nearly 1GB of memory space, which is extremely expensive for memory devices. Therefore, an algorithm for reducing the false alarm rate and improving the search efficiency is urgently needed.

Therefore, we propose a Dynamic Asymmetric Maximum algorithm to reduce the impact of low-entropy strings on chunk boundaries and optimize the chunk management scheme to improve the overall throughput of the system.

## 3 Design

In this section, we describe the deduplication system and our proposed algorithm.

### *3.1 Overall Design*

According to the chunk-level data deduplication technology introduced in Section 1, we designed the overall process of the system, as shown in Fig. 5.

Our design includes the following three stages: (1) The system split the data stream into chunks. (2) Calculate the fingerprint for each chunk as an identifier. (3) Use the index to establish the mapping relationship between the data chunk and the fingerprint and store the unique identification chunk. We use the SHA-1 hash function to calculate the fingerprint. The fingerprint is used to identify whether the chunk is unique. In the chunking stage of the system, as shown in (1) of Fig. 5. We propose a new CDC algorithm, called Dynamic Asymmetric Maximum Algorithm (DAM) based on asymmetric chunking

algorithm in Section 3.2. As for the indexing stage, as shown in (3) of Fig. 5. We use a perfect hashing algorithm. This optimization increases index-lookup performance on disk.

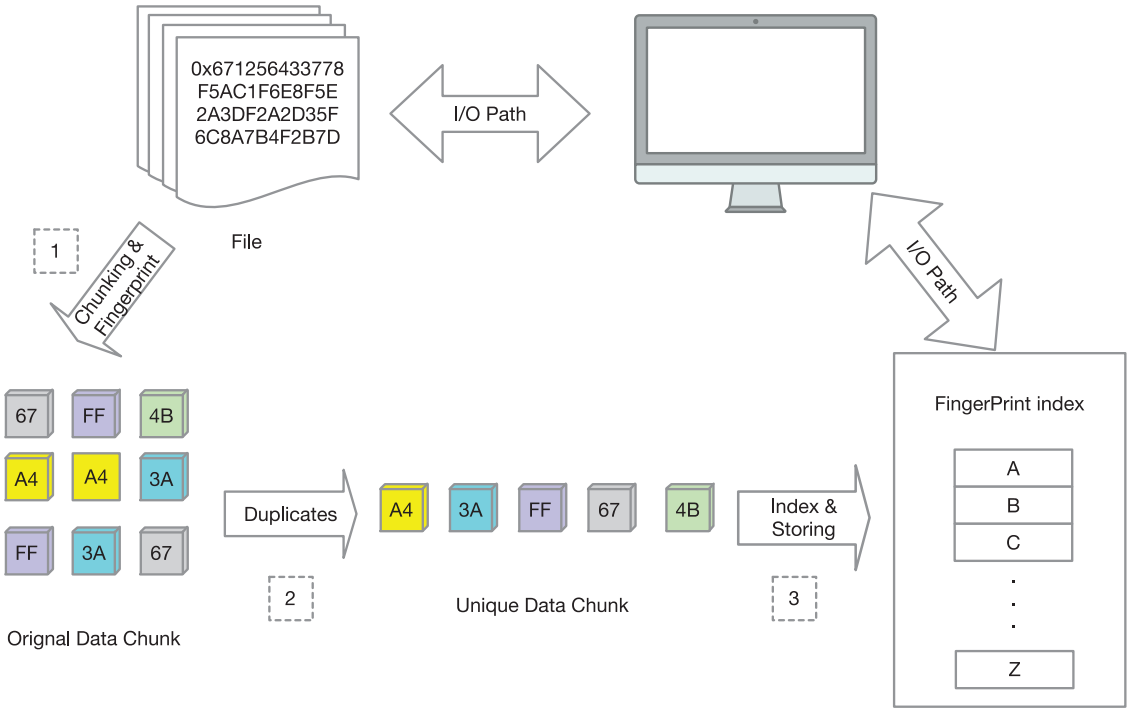


**Figure 5:** The architecture of our system

### *3.2 Dynamic Asymmetric Maximum Algorithm*

In this section, we analyze the low-entropy string pattern proposed in Section 2.2. For the first pattern of low-entropy string, we think that the maximum and minimum values in a region are the same. So, we can define a threshold for the expected chunk size. When the algorithm processes the byte of the local byte stream, it will check whether the maximum value and the minimum value are equal. If they are equal, the string is considered to be a low-entropy string. As a cutting point, otherwise continue to search for the cutting point backward. The pseudocode shown in Algorithm 1.

For the second pattern of low-entropy strings, which is composed of the same substring. Divide the original string into multiple substrings, and then determine whether these strings are all equal. The time complexity of this comparison is $O(len * subsum)$, where *len* is the length of the string, and *subsum* is the number of factors of *len*. If using this method to find low-entropy strings, it will affect the efficiency of deduplication. Therefore, we propose an optimization algorithm for dynamic jump windows, only detect the local strings when they reach a predefined condition. The pseudocode is shown in Algorithm 2.

### *3.3 Perfect Hashing Algorithm*

To optimize the overall throughput of the deduplication system and improve the efficiency of repeated lookups, we propose a perfect hash function algorithm. The perfect hash function does not need to keep the index mark in the main memory, it can accurately locate the object on the disk and find it quickly [38]. The perfect hash algorithm structure diagram is shown in Fig. 6.

**Algorithm 1:** Algorithm of DAM Chunking for first low-entropy string pattern

**Input:** input string stream, *Str*; the length of the string, *L*

**Output:** the chunk boundary, *i*

```
Predefined value: window size, w; threshold, Cp
function DAMChunking(Str, L)
    i ← 1
    max .value ← 0
    min .value ← 1e7
    while i < L do
        if max .value ≥ Str[i].value and min .value ≤ Str[i].value then
            if max .value = min .value and i = Cp then
                return i
            end if
        else if Str[i].value ≥ max .value then
            if i > w then
                return i
            else
                max .value = Str[i].value
            end if
        else if Str[i].value ≤ min .value then
            min .value = Str[i].value
        end if
        i ← i + 1
    end while
    return L
end function
```

The algorithm is divided into two stages: (1) The key set of length $n$ is divided into $N$ buckets using a hash function. (2) Use the perfect hash function to calculate the offset of each bucket to calculate the storage range, and offset $offset[i]$ stores the number of keys in $bucket[i]$. For the constructor of the key-value $x$ as formula (3). The pseudocode is shown in Algorithm 3.

$$PHF(x) = PHF_i(x) + offset[i] \tag{3}$$

**Algorithm 2:** Algorithm of DAM-1 Chunking for second low-entropy string pattern

**Input:** highly redundant string, *Str*; the length of the string, *L*

**Output:** Whether is low-entropy string

```
function DAM-1Chunking(Str, L)
    i ← 1
    j ← −1
    while i < L do
        while j ≥ 0 and Str[i] !=Str[j + 1] do
            j ← next[j]
        end while
        if Str[i] =Str[j + 1] then
            j ← j + 1
                return i
        end if
        next[i] ← j
    end while
    if L − 1 − next[L − 1]! = 1 and L%(L − 1 − next[L − 1]) = 0 then
        return True
    end if
    end while
    return False
end function
```

**Algorithm 3:** Perfect Hashing Algorithm

1: Split Key Set $s$ into buckets $B_i = \{x \in s | h(x) = i\}, 0 \le i < \omega$
2: Using the radix algorithm that takes $q(x), x \in s_i$ as a sorting key from Buckets
3: Compute the function $f_i$ for each bucket
4: Obtain $M_i$(the maximum value of $f_i$ on buckets $B_i$ ) and $offet[i]$
5: Compute $offet[i+1] = offet[i] + M_i$
6: Write $offet[i], f[i]$ to disk.

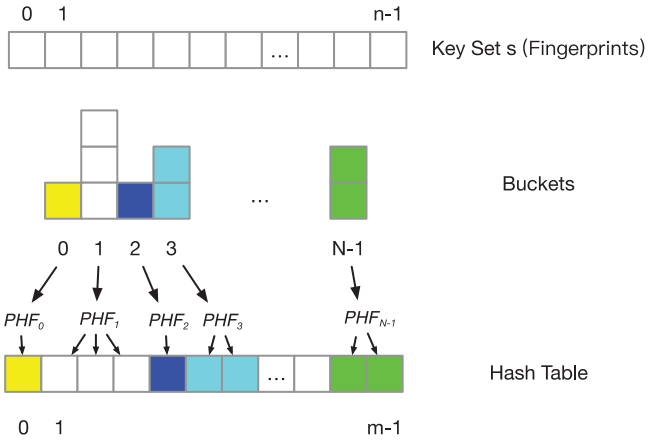


**Figure 6:** The architecture of perfect hash algorithm

## 4 Experiment

In this section, we are implementing the entire system process. To show the advantages of DAM, we have implemented the AE, Rabin and LMC algorithms in the system to compare their performance.

The server is configured with a 4-core 8-thread Intel i7 4700 2.7 GHZ, 8 GB DDR4, 2133 MHz and a 500GB HDD hard disk. We use the SHA-1 hash function to generate a fingerprint for each data chunk, which is used to detect unique data chunks and use the perfect hash algorithm for index lookups. AE used different version numbers of different Linux distributions as data sets for deduplication verify. Therefore, we selected seven different versions of the Linux operating system image. The total size of this dataset is 8.7 GB. The operating system images have similar content in different locations and have low-entropy strings.

The average block size is directly related to deduplication efficiency. To simulate the network transmission environment of wireless sensors, we will set the average block size to a length like the TCP packet size. If the average block size is too small, it will directly affect the efficiency of duplicate data detection, and if it is too large, it will cause network transmission packet loss. For the sake of fairness, we first use the Rabin algorithm to get the expected block size and then adjust the average block size of other algorithms according to the actual situation. We define the deduplication rate (DER) as the ratio of the data input size to the size of data needs to be stored. Therefore, the higher DER, the smaller the actual storage data size.

Fig. 7 shows the results of comparing the four algorithms on the datasets, namely Rabin, LMC, AE, DAM. The DAM algorithm is close to the AE algorithm at the beginning. But as the chunk size increment, the DAM performance is better than AE because it can capture more low-entropy strings and achieve a better deduplication rate. This can be seen in the 13% improvement in duplicates found for DAM over LMC.

In Fig. 8, it represents the chunk distribution of the four algorithms. We can see that the chunk distribution of the DAM algorithm is like the AE algorithm, but it is more concentrated because the DAM algorithm can effectively detect low-entropy strings and reduce the impact on the chunk boundary. The distribution of Rabin and LMC algorithm is more dispersed than others.

Now, we evaluate the DAM's detection efficiency for low-entropy strings. We mainly focus on the first pattern of low-entropy strings. The low-entropy string in the second pattern is challenging to detect, so the threshold is set according to the block size only in a specific scene. The Rabin and LMC algorithm cannot detect low-entropy strings. However, when the data stream is long enough, the Rabin algorithm can set the maximum threshold to detect some low-entropy strings. Rabin_0.25 sets a quarter of the expected block

length as a threshold, which detects low-entropy strings. Tab. 2 shows the low-entropy string detection rate of the expected block size on the experimental dataset. AE and DAM can detect more strings than Rabin and LMC in the datasets. DAM detects low-entropy strings up to 10.6 x more than Rabin_0.25 and 1.98 x more than AE.

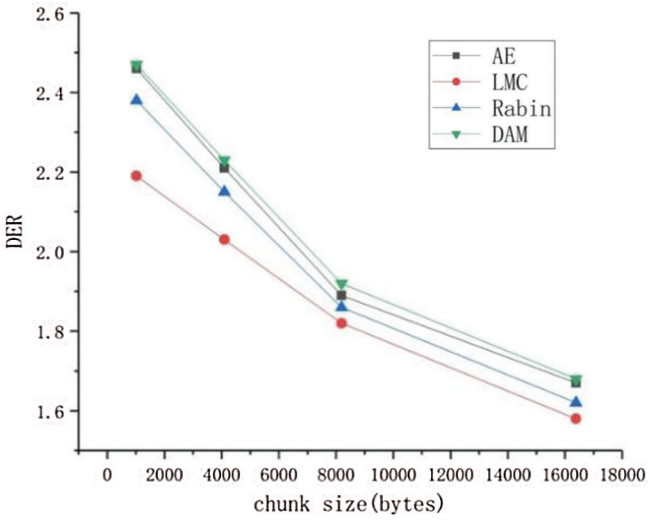


**Figure 7:** Comparison of the deduplication efficient between different algorithms. Different colors represent different approaches

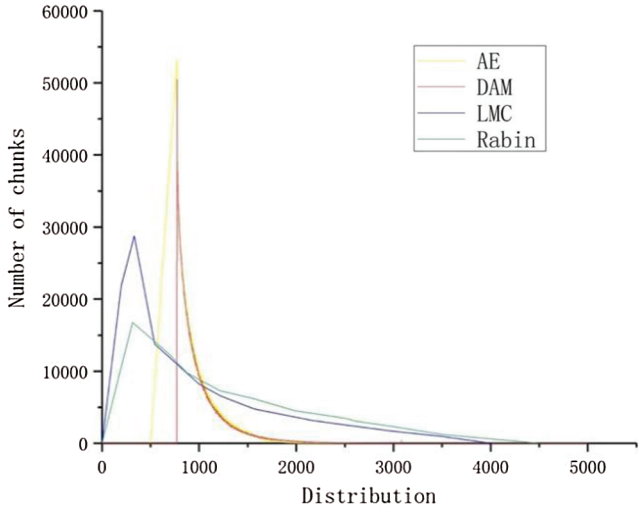


**Figure 8:** The result of the experiment to explore the effects of chunk distribution. Different colors represent different approaches

**Table 2:** The low-entropy string detection rate of the expected block size

| Algorithm | 2KB | 4KB | 8KB | 16KB |
|---|---|---|---|---|
| Rabin | 0 | 0 | 0 | 0 |
| Rabin_0.25 | 0.98% | 0.47% | 0.14% | 0.08% |
| LMC | 0 | 0 | 0 | 0 |
| AE | 4.82% | 1.74% | 1.38% | 1.02% |
| DAM | 7.52% | 5.17% | 3.44% | 1.62% |

## 5 Conclusion

In intelligent transportation systems, resource-constrained IoT devices have limited computing and storage capabilities. Therefore, sensor clouds are used for data collection, processing, and visualization. With the explosive growth of the number of wireless sensors, there is a lot of duplicate data in massive data. These duplicate data occupy storage space and a large amount of communication bandwidth, so it is necessary to use duplicate data technology for data deletion. The content-defined chunking approaches in the deduplication system require many system resources. To alleviate the high computational overhead bottleneck of the CDC approaches in data deduplication systems. In this paper, we compared the difference between FSC and CDC approaches and have discussed the importance of CDC approaches in deduplication system. We propose a Dynamic Asymmetric Maximum algorithm, which can effectively identify low-entropy strings by predetermined thresholds, eliminate redundant data, and optimize chunk indexes. We conducted an experimental comparison in the Linux open-source version image. The experimental results show that, compared with other duplicate data schemes, DAM can eliminate the impact of low-entropy strings on deduplication, improve the throughput rate in the fingerprint search phase, significantly improve the performance of deduplication, and save space for cloud storage.

**Acknowledgement:** The authors are grateful to the anonymous referees for having carefully read earlier versions of the manuscript. Their valuable suggestions substantially improved the quality of exposition, shape, and content of the article.

**Funding Statement:** This work is supported by State Grid Shandong Electric Power Company Science and Technology Project Funding under Grant no. 520613200001,520613180002, 62061318C002, Weihai Scientific Research and Innovation Fund (2020).

**Conflicts of Interest:** All authors declare that they have no conflicts of interest to the present study.

## References

[1] Z. Zhang, H. Liu, L. Rai and S. Zhang, "Vehicle trajectory prediction method based on license plate information obtained from video-imaging detectors in urban road environment," *Sensors*, vol. 20, no. 5, pp. 1258, 2020.

[2] Y. Wang, P. Wang, Q. Wang, Z. Chen and Q. He, "Using vehicle interior noise classification for monitoring urban rail transit infrastructure," *Sensors*, vol. 20, no. 4, pp. 1112, 2020.

[3] M. A. Aljamal, H. M. Abdelghaffar and H. A. Rakha, "Developing a neural-kalman filtering approach for estimating traffic stream density using probe vehicle data," *Sensors*, vol. 19, no. 19, pp. 4325, 2019.

[4] D. Kim and S. Kim, "Network-aided intelligent traffic steering in 5g mobile networks," *Computers, Materials & Continua*, vol. 65, no. 1, pp. 243–261, 2020.

[5] X. Zhao, H. Zhu, X. Qian and C. Ge, "Design of intelligent drunk driving detection system based on Internet of Things," *Journal of Internet of Things*, vol. 1, no. 2, pp. 55–62, 2019.

[6] H. Teng, Y. Liu and A. Liu, "A novel code data dissemination scheme for Internet of Things through mobile vehicle of smart cities," *Future Generation Computer Systems*, vol. 94, pp. 351–367, 2019.

[7] Y. Wang, K. Jia and P. Liu, "Nonlinear correction of pressure sensor based on depth neural network," *Journal of Internet of Things*, vol. 2, no. 3, pp. 109–120, 2020.

[8] M. Z. A. Bhuiyan, J. Wu, G. Wang and T. Wang, "e-Sampling: Event-sensitive autonomous adaptive sensing and low-cost monitoring in networked sensing systems," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 12, no. 1, pp. 1–29, 2017.

[9] B. J. Chang and J. M. Chiou, "Cloud Computing-based analyses to predict vehicle driving shockwave for active safe driving in intelligent transportation system," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 2, pp. 852–866, 2019.

[10] B. Hu and Z. Sun, "Research on time synchronization method under arbitrary network delay in wireless sensor networks," *Computers, Materials & Continua*, vol. 61, no. 3, pp. 1323–1344, 2019.

[11] X. Chen, S. Leng and Z. Tang, "A millimeter wave-based sensor data broadcasting scheme for vehicular communications," *IEEE Access*, vol. 7, pp. 149387–149397, 2019.

[12] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher and V. Young, "Mobile edge computing—A key technology towards 5G," *ETSI White Paper*, vol. 11, no. 11, pp. 1–16, 2015.

[13] M. Peng, S. Yan, K. Zhang and C. Wang, "Fog-computing-based radio access networks: Issues and challenges," *Ieee Network*, vol. 30, no. 4, pp. 46–53, 2016.

[14] G. Qiao, S. Leng, K. Zhang and Y. He, "Collaborative task offloading in vehicular edge multi-access networks," *IEEE Communications Magazine*, vol. 56, no. 8, pp. 48–54, 2018.

[15] I. Ali, A. Gani and I. Ahmedy, "Data collection in smart communities using sensor cloud: recent advances, taxonomy, and future research directions," *IEEE Communications Magazine*, vol. 56, no. 7, pp. 192–197, 2018.

[16] L. Ferdouse, A. Anpalagan and S. Erkucuk, "Joint communication and computing resource allocation in 5G cloud radio access networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 9, pp. 9122–9135, 2019.

[17] A. Prasad, P. Lundén and M. Moisio, "Efficient mobility and traffic management for delay tolerant cloud data in 5G networks," in *IEEE 26th Annual Int. Sym. on Personal, Indoor, and Mobile Radio Communications, PIMRC 2015*, China: Hong Kong, pp. 1740–1745, 2015.

[18] C. Zhao, T. Wang and A. Yang, "A heterogeneous virtual machines resource allocation scheme in slices architecture of 5g edge datacenter," *Computers, Materials & Continua*, vol. 61, no. 1, pp. 423–437, 2019.

[19] J. Peng, K. Cai and X. Jin, "High concurrency massive data collection algorithm for IoMT applications," *Computer Communications*, vol. 157, pp. 402–409, 2020.

[20] D. Zhu and H. Du, "Massive files prefetching model based on LSMT neural network with cache transaction strategy," *CMC-Computers, Materials & Continua*, vol. 63, no. 2, pp. 979–993, 2020.

[21] D. Zhu, H. Du and Y. Wang, "An IoT-oriented real-time storage mechanism for massive small files based on Swift," *International Journal of Embedded Systems*, vol. 12, no. 1, pp. 72–80, 2020.

[22] D. Zhu, H. Du and N. Cao, "SP-TSRM: A data grouping strategy in distributed storage system," in *Int. Conf. on Algorithms and Architectures for Parallel Processing*, Cham: Springer, pp. 524–531, 2018.

[23] D. T. Meyer and W. J. Bolosky, "A study of practical deduplication," *ACM Transactions on Storage (ToS)*, vol. 7, no. 4, pp. 1–20, 2012.

[24] Q. He, Z. Li and X. Zhang, "Data deduplication techniques," in *2010 Int. Conf. on Future Information Technology and Management Engineering, FITME 2010*, Changzhou, China, vol. 1, pp. 430–433, 2010.

[25] R. Rivest and S. Dusse, "The MD5 message-digest algorithm," 1992.

[26] X. Wang, Y. L. Yin and H. Yu, "Finding collisions in the full SHA-1," in *Annual int. cryptology conf.*, Berlin, Heidelberg: Springer, pp. 17–36, 2005.

[27] H. Gilbert and H. Handschuh, "Security analysis of SHA-256 and sisters," in *Int. workshop on selected areas in cryptography*. Berlin, Heidelberg, pp. 175–193, 2003.

[28] D. T. Meyer and W. J. Bolosky, "A study of practical deduplication," *ACM Transactions on Storage (ToS)*, vol. 7, no. 4, pp. 1–20, 2012.

[29] L. Zhu, F. R. Yu and Y. Wang, "Big data analytics in intelligent transportation systems: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 1, pp. 383–398, 2018.

[30] D. Meister and A. Brinkmann, "Multi-level comparison of data deduplication in a backup scenario," in *The Israeli Experimental Systems Conf., SYSTOR 2009*, Haifa, Israel, pp. 1–12, 2009.

[31] Q. He, Z. Li and X. Zhang, "Data deduplication techniques," in *2010 Int. Conf. on Future Information Technology and Management Engineering*, Changzhou, China, pp. 430–433, 2010.

[32] M. Bellare and P. Rogaway, "The exact security of digital signatures-How to sign with RSA and Rabin," in *Int. conf. on the theory and applications of cryptographic techniques*, Berlin, Heidelberg, pp. 399–416, 1996.

[33] N. Bjørner, A. Blass and Y. Gurevich, "Content-dependent chunking for differential compression, the local maximum approach," *Computer and System Sciences*, vol. 76, no. 3–4, pp. 154–203, 2010.

[34] Y. Zhang, D. Feng and H. Jiang, "A fast asymmetric extremum content defined chunking algorithm for data deduplication in backup storage systems," *IEEE Transactions on Computers*, vol. 66, no. 2, pp. 199–211, 2016.

[35] R. N. S. Widodo, H. Lim and M. Atiquzzaman, "A new content-defined chunking algorithm for data deduplication in cloud storage," *Future Generation Computer Systems*, vol. 71, pp. 145–156, 2017.

[36] D. S. Wilkerson Schleimer and A. Aiken, "Winnowing: Local algorithms for document fingerprinting," in *2003 ACM SIGMOD int. conf. on Management of data*, New York, US, pp. 76–85, 2003.

[37] S. Ratnasamy, B. Karp and L. Yin, "GHT: A geographic hash table for data-centric storage," in *1st ACM international workshop on Wireless sensor networks and applications*. Atlanta Georgia, USA, pp. 78–87, 2002.

[38] F. C. Botelho, P. Shilane and N. Garg, "Memory efficient sanitization of a deduplicated storage system," in *11th USENIX Conf. on Fil and Storage Technologies*, Berkeley, CA, United States, pp. 81–94, 2013.