AE: An Asymmetric Extremum Content Defined Chunking Algorithm for Fast and Bandwidth-Efficient Data Deduplication

Yucheng Zhang†, Hong Jiang‡, Dan Feng†*, Wen Xia†, Min Fu†, Fangting Huang†, Yukun Zhou†
†Wuhan National Laboratory for Optoelectronics
School of Computer, Huazhong University of Science and Technology, Wuhan, China
‡University of Nebraska-Lincoln, Lincoln, NE, USA
*Corresponding author: dfeng@hust.edu.cn

***Abstract*—Data deduplication, a space-efficient and bandwidth-saving technology, plays an important role in bandwidth-efficient data transmission in various data-intensive network and cloud applications. Rabin-based and MAXP-based Content-Defined Chunking (CDC) algorithms, while robust in finding suitable cut-points for chunk-level redundancy elimination, face the key challenges of (1) low chunking throughput that renders the chunking stage the deduplication performance bottleneck and (2) large chunk-size variance that decreases deduplication efficiency. To address these challenges, this paper proposes a new CDC algorithm called the Asymmetric Extremum (AE) algorithm. The main idea behind AE is based on the observation that the extreme value in an asymmetric local range is not likely to be replaced by a new extreme value in dealing with the boundaries-shift problem, which motivates AE's use of asymmetric (rather than symmetric as in MAXP) local range to identify cut-points and simultaneously achieve high chunking throughput and low chunk-size variance. As a result, AE simultaneously addresses the problems of low chunking throughput in MAXP and Rabin and high chunk-size variance in Rabin. The experimental results based on four real-world datasets show that AE improves the throughput performance of the state-of-the-art CDC algorithms by 3x while attaining comparable or higher deduplication efficiency.**

I. Introduction

According to a study of International Data Corporation (IDC), the amount of digital information generated in the whole world is about 1.8ZB in 2012, and that amount will reach 40ZB by 2020 [1]. How to efficiently store and transfer such large volumes of digital data is a challenging problem. Moreover, IDC also shows that about three quarters of digital information is duplicated. As a result, data deduplication, a space and bandwidth efficient technology that prevents redundant data from being stored in storage devices and transmitted over the networks, is one of the most important methods to tackle this challenge. Due to its significant data reduction efficiency, chunk-level deduplication is used in various fields, such as storage systems [2], [3], Redundancy Elimination (RE) in networks [4], [5], file-transfer systems (rsync [6]) and remote-file systems (LBFS [7]).

Chunk-level deduplication schemes divide the input data stream into chunks and then hash each chunk to generate its fingerprint that uniquely identifies the chunk. Duplicate chunks can be removed if their fingerprints are matched with those of previously stored or transmitted chunks. As the first and key stage in the chunk-level deduplication workflow, the chunking algorithm is responsible for dividing the input data stream into chunks of either fixed size or variable size for redundancy detection. Fix-Sized Chunking (FSC) [8] marks chunks' boundaries by their positions and thus is simple and extremely fast. The main drawback of FSC is its low deduplication efficiency that stems from the boundaries-shift problem. For example, if one byte is inserted at the beginning of an input data stream, all current chunk boundaries declared by FSC will be shifted and no duplicate chunks will be identified and eliminated. Content-Defined Chunking (CDC) divides the input data stream into variable-sized chunks. It solves the boundaries-shift problem by declaring chunk boundaries depending on local content. As a result, the CDC algorithm outperforms the FSC algorithm in terms of deduplication efficiency and has been widely used in bandwidth- and storage-efficient applications [9], [10]. To provide the necessary basis to facilitate the discussion of and comparison among different CDC algorithms, we list below some key properties that a desirable CDC algorithm should have.

1) **Content defined.** To avoid the boundaries-shift problem, the algorithm should declare the chunk boundaries based on local content, i.e., the cut-points for chunking must be content defined.
2) **Low computational overhead.** CDC algorithms need to check almost every byte in an input data stream to find the chunk boundaries. This means that the algorithm execution time is approximately proportional to the number of bytes of the input data stream, which can take up significant CPU resources. Hence, in order to achieve higher deduplication throughput, the chunking algorithm should be simple and devoid of time-consuming operations.
3) **Small chunk size variability.** The variance of chunk size has a significant impact on the deduplication efficiency. The smaller the variance of the chunk size is, the higher the deduplication efficiency will be achieved [11].
4) **Ability to identify and eliminate low-entropy strings.** The content of real data may sometimes include low-entropy strings [12]. These strings include very few distinct characters but a large amount of repetitive bytes. In order to achieve higher deduplication efficiency, it is desirable for the algorithm to be capable of detecting and eliminating these duplicate strings.
5) **Less limitations on chunk size.** Minimum and maximum

978-1-4799-8381-0/15/$31.00 ©2015 IEEE

thresholds are often imposed on chunk size to avoid chunks being too short or too long. These measures reduce chunk size variance, but also make the chunk boundaries position-dependent and thus not truly content-defined, which also reduces the deduplication efficiency [13].

The Rabin fingerprint [14] based CDC algorithm (Rabin) is widely employed for redundancy elimination in both storage systems [15], [2], [16] and networks [5], [17]. The main problems of the Rabin algorithm are its low chunking throughput, which renders the chunking process the performance bottleneck of the deduplication workflow [18], [19], and large chunk size variance that lowers the deduplication efficiency [11]. MAXP [20] is a CDC approach that addresses the chunk-size variance problem of Rabin by treating the local extreme values as cut-points. Owing to its smaller chunk size variance and lower memory overhead than Rabin, MAXP was recommended to be used in redundancy elimination in networks [10], [21]. MAXP slides a fix-sized symmetric window over the byte stream on a byte-by-byte basis, and checks whether the value of the byte at the center of the current window is the strictly extreme value in the window. The byte found to be the extreme value is declared a cut-point (chunk boundary). This strategy of finding the local maximum values dictates that the MAXP algorithm recheck some previously compared bytes in the reverse direction of the stream and thus requires more than one comparison and more than five conditional branch operations per byte scanned [22], which significantly lowers its chunking throughput.

In other words, while the MAXP algorithm improves the Rabin algorithm by reducing the chunk-size variance, the problem of low chunking throughput remains in both algorithms. To this end, we propose the Asymmetric Extremum chunking algorithm (AE), a new CDC algorithm that significantly improves the chunking throughput of the above existing algorithms while providing comparable or better deduplication efficiency by using the local extreme value in a variable-sized asymmetric window to overcome the aforementioned boundaries-shift problem. With a variable-sized asymmetric window, instead of a fix-sized symmetric window as in MAXP, the AE algorithm finds the extreme value in the window without having to backtrack and thus requiring only one comparison and two conditional branch operations per byte scanned. Therefore, AE's simplicity makes it very fast. It also has smaller chunk size variance than existing CDC algorithms and imposes no limitation on chunk size. Moreover, AE is able to eliminate more low-entropy strings than the other algorithms. Our experimental evaluations of AE, based on four real-world datasets, show that AE achieves a chunking throughput of 1GBytes/s, which is about $3\times$ higher than the state-of-the-art CDC algorithms, with comparable or higher deduplication efficiency.

The rest of paper is organized as follows. In Section II, we present the background and motivation for this research. We describe the detailed design and implementation of our AE algorithm and analyze some of its key properties in Section III. We present the experimental setup and evaluation results in Section IV and conclude the paper in Section V.

II. Background and Motivation

In this section, we first provide the necessary background for the AE research by introducing the challenges facing the existing CDC algorithms, and then motivate our research by analyzing our key observations.

A. Background

The Rabin fingerprint [14] based CDC algorithm (Rabin) was first used to eliminate redundant network traffic [17]. It runs a sliding-window hash along the byte stream, declaring a chunk boundary whenever the k-lowest-order bits of the hash are equal to a pre-determined value. The Rabin algorithm often imposes a minimum and a maximum threshold on the chunk size to avoid chunks being too short or too long. This is because very short chunks imply more fingerprints to be stored and processed and thus not cost-effective, while very long chunks reduce the deduplication efficiency. Nevertheless, Rabin suffers from two major drawbacks, namely, its time-consuming fingerprint computation that results in low chunking throughput and its large chunk size variance that reduces deduplication efficiency. Recognizing the impact of the chunk-size variance on deduplication efficiency, Eshghi et al. [11] proposed the TTTD algorithm to improve Rabin's deduplication efficiency. To reduce the chunk-size variance, the TTTD algorithm introduces an additional backup divisor that has a higher probability of finding cut-points. When it fails to find a cut-point using the main divisor within a maximum threshold, it returns the cut-point found by the backup divisor, if any. If no cut-point is found by either of the divisors, it returns the maximum threshold. However, adding an additional divisor decreases the chunking throughput, meaning that the TTTD algorithm aggravates Rabin's performance bottleneck.

MAXP [20], [10] is a state-of-the-art CDC algorithm, which is first used in remote differential compression of files. Unlike Rabin that must compute a hash first, the MAXP algorithm treats the bytes directly as digits, which helps reduce the computational overhead. MAXP attempts to find the strict local extreme values in a fixed-size symmetric window, and then uses these points as chunk boundaries to divide the input stream. The main disadvantages of this strategy is that when declaring an extreme value, the algorithm must move backwards by a fixed-size region to check if there is any value greater (if the extreme value is the maximum value) than the value of the current position being examined. This backtracking process requires many extra conditional branch operations and increases the number of comparison operations for each byte examined. Since MAXP needs to check every byte in the input data stream, any additional conditional branch operations result in a decreased chunking throughput.

EndRE [21] proposes an adaptive Samplebyte algorithm for declaring fingerprint. The Samplebyte algorithm combines the CDC algorithm's robustness to small changes in content with the efficiency of the FSC algorithm. It uses one byte to declare a fingerprint and stores $1/p$ representative fingerprints for content matching, where p is the sampling period. To avoid over-sampling, it skips $p/2$ bytes when a fingerprint has been found. By increasing p, the algorithm can skip more bytes and thus improve the chunking throughput. However, the design principle of Samplebyte dictates that the sampling period p be smaller than 256 Bytes, which means that the expected average chunk size must be smaller than 256 Bytes when used in the chunk-level deduplication. Unfortunately, a chunk granularity of 256 bytes or smaller is too fine to be cost efficient or practical, which makes the Samplebyte algorithm inappropriate for coarse-grained chunk-level deduplication.

TABLE I. PROPERTIES OF THE STATE-OF-THE-ART CDC ALGORITHMS.

Properties	Rabin	TTTD	Samplebyte	MAXP	AE
Content Defined	Yes	Yes	Yes	Yes	Yes
Computational overheads	High	High	Low	High	Low
Chunk size variance	High	Low	High	Low	Low
Ability to eliminate low-entropy strings	Low	Middle	Low	None	High
Limitations on chunk size	Yes	Yes	Yes	No	No

B. Challenges and Motivation

Chunk-level deduplication is one of the main deduplication methods due to its ability to exploit cross-file redundancy. Because of higher deduplication efficiency than FSC [23], CDC algorithms are preferred in chunk-level deduplication. However, the low chunking throughput of the existing CDC algorithms hinders their wider applications because of the deduplication performance bottleneck [19], [24]. To alleviate the performance bottleneck and increase the throughput of the deduplication system, P-Dedup [19] harnesses the idle CPU resources to pipeline and parallelize the compute-intensive processes. StoreGPU [24] and Shredder [18] exploit underutilized GPU resources to improve the chunking throughput. However, these schemes achieve their performance improvement from either additional resources or parallelization of the deduplication processes, but not from improving the chunking algorithm itself.

Table I compares the state-of-the-art CDC algorithms by summarizing their key properties. The Rabin algorithm has the problems of high computational overheads and high chunk size variance. The TTTD algorithm improves the problem of high chunk size variance but lowers the chunking throughput. The MAXP algorithm is computationally expensive and cannot eliminate low-entropy strings. The Samplebyte algorithm is confined to the fine-grained deduplication (the expected average chunk size is smaller than 256 Bytes) and has limited applicability. In addition, Samplebyte is not strictly content-defined, since its strategy of skipping $p/2$ bytes (a half of the expected average chunk size) makes many chunk boundaries position-dependent, which will result in the boundaries-shift problem.

The various problems facing the state-of-the-art CDC algorithms summarized in Table 1 motivate us to propose a new chunking algorithm to overcome these problems. In fact, our experimental observation finds that detecting local extreme values in an asymmetric window can not only deal with the boundaries-shift problem for Content-Defined Chunking, but also increase the chunking throughput and detect low-entropy strings. As a result, our proposed Asymmetric Extremum (AE) chunking algorithm, by using an asymmetric window to find the local extreme value for chunking as elaborated in the next section, is able to remove the chunking-throughput performance bottleneck of deduplication, and better satisfy the key desirable properties of CDC algorithm to achieve high deduplication efficiency and performance.

III. ASYMMETRIC EXTREMUM CHUNKING ALGORITHM

In this section we describe the design of the AE chunking algorithm and analyze its key properties.

A. The AE Algorithm Design

In AE, a byte has two attributes: position and value. Each byte in the input data stream has a position number, and the position of the n^{th} byte ($1 \leqslant n \leqslant stream\ length$) in the stream is n. Each interval of S consecutive characters/bytes in the input data stream is treated as a value. The value of every such interval in the data stream is associated with the position of the first byte of the S consecutive characters/bytes that constitute this value. Therefore, each byte in the stream, except for the very last $S-1$ bytes, has a value associated with it.

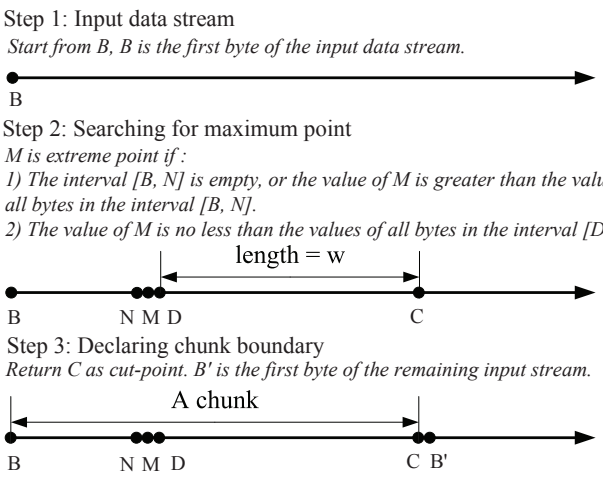


Fig. 1. The workflow of the AE chunking algorithm, where N, M, D in the figure are neighboring positions, and so are the positions of C and B′.

Definition 1: Given an input data stream, it is defined to start from the leftmost byte. If a byte A is on the left of byte B, A is said to be before B, and B appears after A.

Definition 2: Given a byte P in the input data stream, the w consecutive bytes immediately after P are defined to be the *right window* of P, and w is referred to as the *window size*.

The extreme value in the AE algorithm can be either the maximum value or the minimum value. For convenience of discussion, in what follows in this section, we assume that the extreme value is the maximum value. Starting from the very first byte of the stream or the first byte after the last cut-point, AE attempts to find the first byte of the input data stream that satisfies the following two conditions.

- It is the first byte or its value is greater than the values of all bytes before it.
- Its value is *not less than* the values of all bytes in its right window.

The first byte found to meet these conditions is referred to as a *maximum point*. These two conditions make sure that the maximum point has the local maximum value. There are two further implications. First, this first byte can be a maximum point. Second, AE allows for ties between the byte being examined and bytes in the right window. If a maximum point has been found, AE returns the rightmost byte in its right window as a cut-point. The workflow of AE is described in Figure 1. Algorithm 1 below provides a more detailed implementation of the AE chunking algorithm.

From the algorithm description above, we know that the minimum chunk size of AE is $w+1$. In what follows we discuss the average chunk size of AE.

Theorem 1: Consider a byte in position p in the current input data stream, starting from the first byte after the last cut point (i.e., excluding the bytes in the data stream that have already been chunked), the probability of this byte being a maximum point is $1/(w+p)$, where w is the window size.

Algorithm 1 Algorithm of AE Chunking

Input: input string, Str; left length of the input string, L;
Output: chunked position (cut-point), i

```
1: Predefined values: window size, w; length of interval S;
2: function AECHUNKING(Str, L)
3:     i ← 1
4:     max.value ← Str[i].value
5:     max.position ← i
6:     i ← i + 1
7:     while i < L do
8:         if Str[i].value <= max.value then
9:             if i = max.position + w then
10:                return i
11:            end if
12:        else
13:            max.value ← Str[i].value
14:            max.position ← i
15:        end if
16:        i ← i + 1
17:    end while
18:    return L
19: end function
```

Proof: We assume that the content of real data is random, an assumption that is reasonably justified by our experimental evaluation and previous work such as MAXP [20]. According to the conditions set for a maximum point, if byte p is a maximum point, it should have the maximum value in the interval $[1, p+w]$. In this interval, each byte is equally likely to be of the maximum value. Thus, the probability of byte p being the maximum point is $1/(w+p)$. Note that if position p is the maximum point, the chunk size will be $p+w$. ■

Now we determine the range of possible positions of the maximum point, namely, the position x that makes the cumulative probability equal to 1. According to Theorem 1, we can compute the position x using the following equation:

$$\frac{1}{w+1} + \frac{1}{w+2} + \frac{1}{w+3} + \cdots + \frac{1}{w+x} = 1.$$

The left side of this equation is approximately equal to $ln(w+x) - lnw$. Thus, the value of x is approximately $(e-1)*w$. For each possible position, the expected chunk size is equal to the probability of being the maximum point multiplying by the chunk size if it is the maximum point. Finally, we compute the expected average chunk size by adding the expected chunk size of all possible positions, and the result, namely, the expected average chunk size, is $(e-1)*w$.

B. Properties of the AE algorithm

In this section, we analyze the AE algorithm in regards to the desirable properties of CDC algorithms listed in Section I.

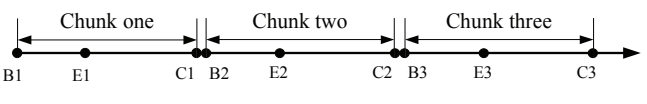


Fig. 2. An example of efficiency loss.

*1) **Content defined**:* The MAXP algorithm considers a byte with the local maximum value a chunk boundary. Therefore, any modifications within a chunk, as long as they do not replace the local maximum value, will not affect the adjacent chunks, since the chunk boundaries will simply be re-aligned. Unlike MAXP, the AE algorithm returns the w^{th} position after the maximum point as the chunk boundary. It puts the maximum points inside the chunks instead of considering them as chunk boundaries. This strategy may slightly decrease the deduplication efficiency, but AE is still content defined since the maximum points inside the chunks can also re-align the chunk boundaries.

TABLE II. COMPUTATIONAL OVERHEADS OF THE THREE ALGORITHMS. p IS THE EXPECTED AVERAGE CHUNK SIZE.

Algorithm	Computational overhead per byte scanned
Rabin	1 OR, 2 XORs, 2 SHIFTs, 2 ARRAY LOOKUPs, 1 CONDITIONAL BRANCH
MAXP	2 MOD, $2 - \frac{1}{p}$ COMPARISONs, $5 + \frac{1}{p}$ CONDITIONAL BRANCHes
AE	1 COMPARISON, 2 CONDITIONAL BRANCHes

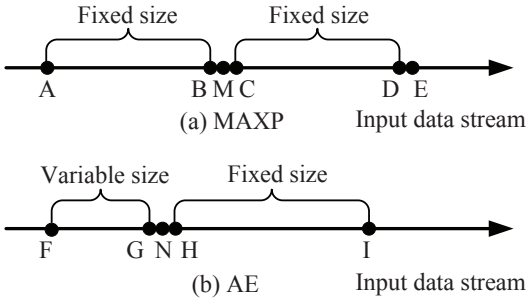


Fig. 3. Illustration of the key difference between the MAXP and AE algorithms, where B, M, C are neighboring positions, so are the positions of D and E and positions of G, N, and H.

Take Figure 2 for example, E1, E2, E3 are the three maximum points, C1, C2, C3 are the cut-points of the three corresponding chunks. If there is a modification (insertion or deletion) in the interval [B1, E1) in Chunk 1, Chunk 2 will not be affected since the chunk boundary will be re-aligned by the maximum point E1. If the modification is in the interval (E1, C1], the starting point of Chunk 2 will be changed, and E2 will re-align the boundary to keep Chunk 3 from being affected. If a sequence of consecutive chunks has been modified, the loss of efficiency is determined by the position of the modification in the last modified chunk. If the modification is *before* the maximum point, there is no efficiency loss. Otherwise, only one duplicate chunk that is immediately after this modified region will be affected. In fact, the loss of deduplication efficiency is small since the modifications are often localized to one or a very small number of regions [2]. In addition, the deduplication efficiency is also determined by many other factors, such as chunk-size variance and the ability to eliminate low-entropy strings. As we will see shortly, AE's ability to eliminate low-entropy strings and reduce chunk-size variance has more than compensated for this relatively small loss of deduplication efficiency.

*2) **Computational overheads**:* Table II shows the computational overheads of the three algorithms, AE, MAXP and Rabin. As shown in the table, the Rabin algorithm needs 1 OR, 2 XORs, 2 SHIFTs and 2 ARRAY LOOKUPs per byte examined to compute the fingerprints and one conditional branch to judge the chunk boundaries. While both the MAXP and AE algorithms use comparison operations to find the local maximum values, their strategies are quite different and it is this difference that makes AE much faster than MAXP. Figure

3 shows the difference between the two algorithms. As shown in the figure, MAXP finds the maximum values in a fixed-size window [A, D]. If the byte M that is in the center of this window has the maximum value in the window, its value must be strictly greater than that of any byte in both regions of [A, B] and [C, D]. Assuming that all bytes in the window [A, D] have been scanned and M has the maximum value and has been returned as a cut-point, most of the bytes in region [C, D] must be scanned again when MAXP processes the byte E. This means that MAXP needs an array to store the information of the bytes in the fixed-size region immediately before the current byte. Moreover, it requires two modular operations to update the array, and $2 - \frac{1}{p}$ comparison and $5 + \frac{1}{p}$ conditional branch operations to find the local maximum value.

In contrast, AE only needs to find the maximum value in an asymmetric window [F, I], which includes a fixed-size region [H, I] and a variable-size region [F, G], whose size is determined by the content of the input data stream. As a result, we only need to store a candidate maximum point and the position of the candidate maximum point, and do not need to backtrack to declare the local maximum value. Therefore, AE only needs one comparison and 2 conditional branch operations. Note that the numbers of comparisons and conditional branches of MAXP and AE are based the descriptions of the algorithms ([20], [22] for MAXP and this section for AE), whose derivations are omitted here due to space constraint. Clearly, AE requires much fewer operations, particularly the time-consuming conditional branch and table lookup operations, than the other two algorithms.

3) ***Chunk size variance****:* In this section we analyze the chunk size variance of the AE algorithm. We use the probability of a long region not having any cut-point to estimate the chunk size variance.

Theorem 2: AE has no maximum point in a given range, if and only if in each interval of w consecutive bytes in this range, there exists at least one byte that satisfies the first condition of the maximum point, namely, it is the first byte or its value is greater than the values of all bytes before it.

Proof: In this range, if there exists one byte in each interval of w consecutive bytes whose value is greater than the values of all bytes before it, then the second condition of the maximum point, namely, its value is *not less than* the values of all bytes in its right window, will never be satisfied. In other words, there is no maximum point in the range. ■

Given an interval $[cw + a + 1, cw + a + w]$, where c is a constant, the probability of no byte satisfying the first condition of maximum point is:

$$\prod_{i=1}^{w}(1 - \frac{1}{a+i}) = \frac{a}{a+w}.$$

So the complementary probability, that there exists at least one byte satisfying the first condition of the maximum point, is $w/(w + a)$. Divide the interval into subintervals with the length of w and then number them from 1 to m. Consider the p^{th} subinterval $[(p - 1)w + 1, pw]$. The probability of no maximum point in it is:

$$\frac{w}{(p-1) * w + w} = \frac{1}{p}.$$

Multiplying the probabilities of the continuous m subintervals, we have $\frac{1}{m!}$. Given that the average chunk size of AE is $(e - 1) * w$, the probability of no maximum point in m consecutive chunks of average chunk size becomes:

$$P(AE) = \frac{1}{[(e-1) * m]!},$$

here m should be more than 1. Next we compare the probabilities of very long chunks among the AE, MAXP and Rabin algorithms. Table III shows formulas to calculate the theoretical probability of no cut-points in a region of length $m \times$ *average-chunk-size* [20] and lists such probabilities when $m = 2, 3, \cdots, 8$ for the three algorithms, where Rabin_0 represents the Rabin algorithm without minimum threshold, and Rabin_0.25 represents Rabin with a minimum threshold on chunk size, and the ratio of the minimum threshold to the expected average chunk size is 0.25. As can be seen from the table, the probability of generating exceptionally long chunks by AE is much lower than the other two algorithms, which also means that AE has smaller chunk-size variance.

TABLE III. PROBABILITY OF NO CUT-POINTS IN A REGION OF LENGTH $m \times$ *average-chunk-size*.

m	AE	Rabin_0	Rabin_0.25	MAXP
	$\frac{1}{[(e-1)*m]!}$	e^{-m}	$e^{-(k+1)m}$	$\frac{2^{2m}}{(2m)!}$
2	0.0938	0.1353	0.0907	0.6667
3	0.0064	0.0498	0.0273	0.0889
4	$2.56 * 10^{-4}$	0.0183	0.0082	0.0063
5	$6.85 * 10^{-6}$	0.0067	0.0025	$2.82 * 10^{-4}$
6	$1.32 * 10^{-7}$	0.0025	$7.47 * 10^{-4}$	$8.55 * 10^{-6}$
7	$1.94 * 10^{-9}$	$9.12 * 10^{-4}$	$2.25 * 10^{-4}$	$1.88 * 10^{-7}$
8	$2.25 * 10^{-11}$	$3.35 * 10^{-4}$	$6.77 * 10^{-5}$	$3.13 * 10^{-9}$

4) ***Dealing with low-entropy strings****:* Ties between the byte being examined and the bytes in the *right window* may appear in the input data stream. If a tie happens to be between two local maximum values, we can break the tie by one of the following two strategies: (1) selecting the first maximum value or (2) going beyond the right window to search for a strictly maximum value. Strategy (1) can help identify and eliminate low-entropy strings. AE allows for ties in its right window and the maximum point can be the first byte, so that it can divide the low-entropy strings into fixed-size chunks whose size is $w + 1$. On the other hand, Strategy (2), which is used in the MAXP algorithm, will lead the algorithm to miss detecting and eliminating low-entropy strings. Note that the AE algorithm cannot detect all low-entropy strings. If the length of a low-entropy string is greater than $2w + 2$, then AE can identify a part of it. Furthermore, Strategy (2) requires more conditional branch operations in finding the maximum points. For these reasons, we chose Strategy (1) for AE.

5) ***The limitations on chunk size****:* AE's strategy of finding the maximum values implies that the length of its chunk will be greater than or equal to $w+1$, so that an artificial minimum limitation on chunk size is unnecessary. In addition, according to Table III, the probability of AE generating exceptionally long chunks is extremely small. This, combined with the fact that AE is able to find cut-points in low-entropy strings, makes it unnecessary for AE to impose the maximum threshold on chunk size, a point that is amply demonstrated in the detailed sensitivity study of the AE algorithm in the next section.

IV. PERFORMANCE EVALUATION

In this section, we present the experimental evaluation of our AE algorithm in terms of multiple performance metrics. To

characterize the benefits of AE, we also compare it with two state-of-the-art CDC algorithms, namely, Rabin and MAXP.

A. Evaluation Setup

We implemented the AE and MAXP chunking algorithm in an open-source deduplicaiton prototype system called Destor [25] on the Ubuntu 12.04.2 operating system running on an 8-core Intel i7 2.8GHz system with 16GB memory and 1TB 7200rpm hard disk. The deduplication system uses the SHA-1 hash function to generate chunk fingerprints for the detection and elimination of duplicate chunks. Note that we did not use a simpler but weaker hash function such as Jenkins [26] to generate fingerprints or a limited cache to store the fingerprints when processing network traffic since we are most concerned with the performance of the chunking algorithm in this paper.

1) Datasets: To evaluate the three CDC algorithms, we use the following four real-world datasets.

Network Traffic dataset: The network traffic in this dataset was collected using wireshark from our research laboratory consisting of 17 IP addresses, 11 for desktop computers and 6 for laptops. All laptops used WiFi connectivity to access the network. The trace collection spanned a period of 7 days and yielded about 53G of data.

Similar Movie-Files dataset: This dataset is composed of 24 movie files constituting 12 different movies. In order to collect this dataset, we first chose 6 popular and 6 unpopular movie titles according to the IMDB rating [27], and then divided them into three groups, each of which included 2 popular and 2 unpopular titles. Finally, we collected two different movie files for each title. In the first group, resolutions of the two movie files of each title are different. In the other two groups, the differences lie in the subtitle and dubbing respectively. Note that the two files of each title have the same file format since there is almost no duplicate data between two videos of different format. The total size of this dataset is 31G.

TAR dataset: This dataset includes 20 versions of GCC, 35 versions of GLIB, 15 versions of GDB, 10 versions of EMACS, and 40 versions of Linux kernels. Each version of these free software was packaged as a tar file. The total size of this dataset is 32G.

VMDK dataset: This dataset consists of 1.85T of 125 backups of an Ubuntu virtual machine. Since all backups are full backup, there exists a large amount of duplicate content in this dataset.

The first dataset represents the typical network traffic. The second and third datasets represent common shared network resources. The third and fourth datasets are very common in backup systems. And the last three datasets respectively represent the data of low, middle and high redundancy.

2) Evaluation Methodology: To better evaluate the performance of AE chunking algorithm, the Rabin and MAXP algorithms were also implemented for comparisons. Every dataset was tested several times by each chunking algorithm with different average chunk sizes. For the Rabin algorithm, we still use the typical configuration Rabin_0.25 (see Section III). We also impose a maximum threshold on chunk size whose value is 8 times the expected average chunk size. Because of the minimum threshold, the real average chunk size of Rabin will be greater than the expected average chunk size. It is approximately equal to the expected average chunk size plus the minimum threshold. For the sake of fairness, for each test, we first processed using Rabin to get the real average chunk size, and then adjusted the real average chunk size to the same value when using other algorithms. For convenience of discussion, Rabin's expected average chunk sizes are used as labels to distinguish different tests on each dataset.

Since the AE algorithm is extreme-value based, the extreme value can be either maximum value or minimum value. As such, it is necessary to find out how sensitive is the AE algorithm to the choice of the form of extreme value, maximum or minimum. Therefore, we implemented both versions of AE, AE_MAX and AE_MIN, to carry out a sensitivity study. In addition, in order to experimentally verify our theoretic analysis and conclusion that it is not necessary to impose a maximum threshold on AE, as summarized in Table III, we also evaluated AE's sensitivity to the maximum chunk-size threshold by implementing the same threshold on AE as that imposed on Rabin, which leads to two more AE versions of AE_MAX_T and AE_MIN_T.

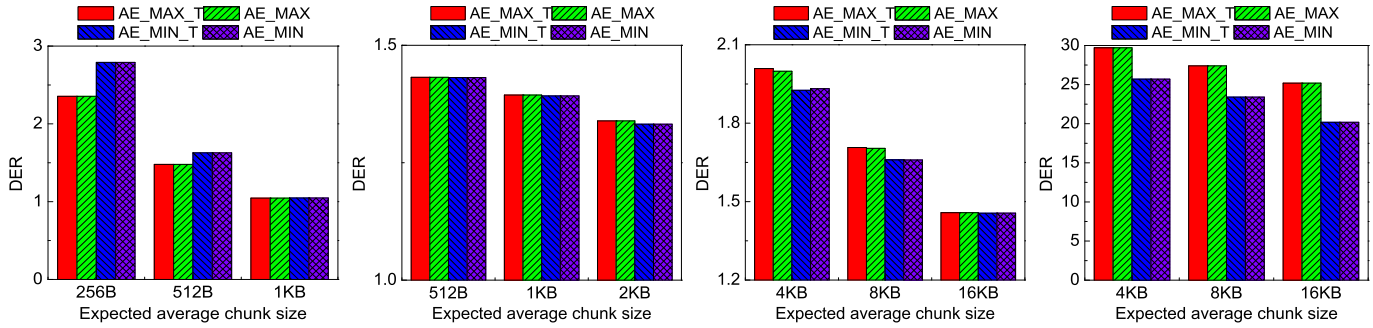
B. Deduplication efficiency

In this section, we evaluate the deduplication efficiency of our algorithm. We use deduplication elimination ratio (DER), which we define as the ratio of the size of input data to the size of data need to be actually stored/transferred, to measure the deduplication efficiency. Therefore, the greater the value of DER is, the higher the deduplication efficiency is.

1) Sensitivity of AE to Design Parameters: Figure 4 shows AE's sensitivity to the key design parameters, i.e., the form of extreme value (maximum vs. minimum) and the necessity of imposing a maximum chunk-size threshold, in terms of deduplication efficiency across the four datasets under AE_MAX, AE_MIN, AE_MAX_T and AE_MIN_T. Consistent with our theoretic analysis of Table III, the experimental results show that the AE without a maximum threshold achieves nearly identical deduplication efficiency as the AE with it. Specifically, the gains in deduplication efficiency from adding the maximum threshold are negligibly small (i.e., on average 0.004%). But for some datasets, it actually reduces the deduplication efficiency, since the cut-points declared by AE with a maximum threshold is no longer strictly content-defined but position dependent. For example, on the Tars dataset when using the 4K expected average chunk size (Figure 4(c)), the DER values of AE_MIN and AE_MIN_T are 1.9319 and 1.9259 respectively, a reduction of 0.3%.

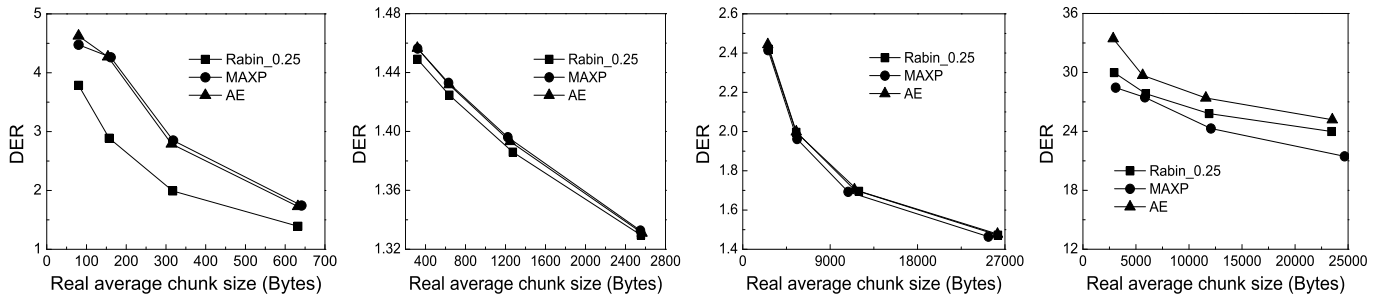
Another key observation is that the deduplication efficiency of AE_MAX and AE_MIN differs across datasets. On the network traffic dataset (Figure 4(a)), AE_MIN obtains higher deduplication efficiency than AE_MAX. On the movies dataset (Figure 4(b)), their performance is comparable. While on the remaining two datasets (Figures 4(c) and 4(d)), AE_MAX outperforms AE_MIN. Recall that, just like MAXP, the AE algorithm treats the bytes directly as digits. The deduplication efficiency of AE_MAX and AE_MIN depends on the frequency and the encoding of the characters in the input data stream. We also tested other datasets (not shown here for space considerations), and we found that AE_MAX can obtain the same or higher deduplication efficiency than AE_MIN on all datasets except the Network traffic dataset.

2) Comparison of DER among AE, Rabin, and MAXP: In this comparison, based on the AE sensitivity study above, we used AE_MIN in the network traffic dataset and AE_MAX in the others. We also tested the MAXP algorithm using either



(a) Network Traffic (b) Movies (c) Tars (d) VMDK

Fig. 4. Deduplication efficiency of AE. AE_MAX and AE_MIN respectively represent AE using maximum and minimum as extreme values. AE_MAX_T and AE_MIN_T denote the AE_MAX and AE_MIN with a maximum threshold.



(a) Network Traffic (b) Movies (c) Tars (d) VMDK

Fig. 5. Deduplication efficiency of the three chunking algorithms on the four real-world datasets. Rabin_0 represents the Rabin algorithm without imposing a minimum chunk-size threshold.

TABLE IV. LOW-ENTROPY STRINGS ELIMINATION RATIO OF THE THREE ALGORITHMS.

(a) Expected average chunk size on the four datasets are 256B, 512B, 4KB and 4KB respectively.

Algorithm	Network Traffic	Movies	Tars	VMDK
Rabin_0.25	0.003%	0	0	2.28%
MAXP	0	0	0	0
AE	0.016%	0.009%	0.006%	6.08%

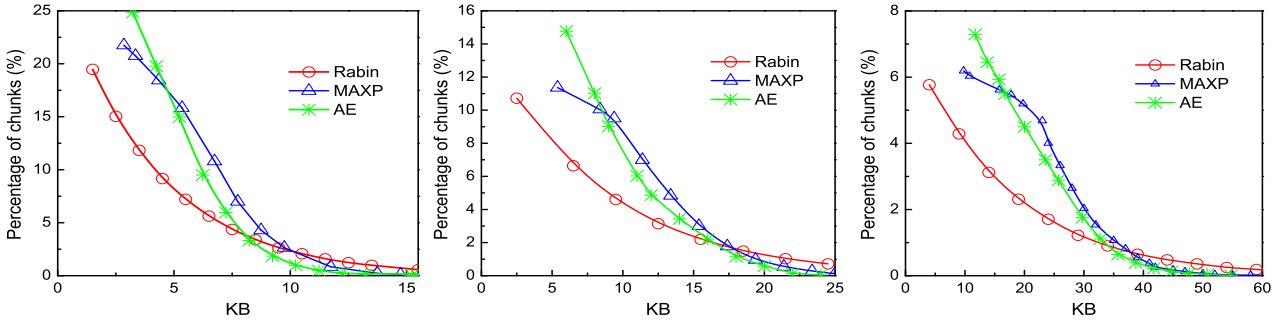
(b) Expected average chunk size on the four datasets are 512B, 1KB, 8KB and 8KB respectively.

Algorithm	Network Traffic	Movies	Tars	VMDK
Rabin_0.25	0.002%	0	0	1.27%
MAXP	0	0	0	0
AE	0.009%	0.002%	0.003%	3.31%

the maximum or the minimum value on the four datasets (not shown here for space considerations), and the results are similar to and consistent with those of AE. Therefore, we use MAXP with minimum value on the Network Traffic dataset and use the maximum value on the others. Figure 5 shows the results of this comparison. In the figure we can see that AE achieves comparable deduplication efficiency to MAXP on the first two datasets, and both AE and MAXP outperform Rabin in the DER measure. On the third dataset (Figure 5(c)), AE achieves almost the same efficiency as Rabin, and their deduplication efficiency are slightly higher than MAXP. On the last data set (Figure 5(d)), AE achieves higher DER than the other two algorithms. The main reason for AE's superior DER performance is its ability to detect and eliminate more low-entropy strings and smaller chunk size variance.

3) Benefits from detecting low-entropy strings: Here we evaluate and discuss the benefits brought by the AE algorithm with its ability to detect the low-entropy strings. Rabin can also detect some low-entropy strings with the help of the maximum threshold when the low-entropy strings are long enough. However, MAXP does not have this capability. Table IV shows the low-entropy strings elimination ratio (LER), which we define as the ratio of the size of the low-entropy strings that can be eliminated to the size of input data. As shown in the table, AE eliminates more low-entropy strings than Rabin and MAXP. For the first three datasets, the benefit of low-entropy strings detection is not obvious, since there are only very small amounts of such strings in these datasets. But on the last dataset, our algorithm detects 72G and 38.6G more low-entropy strings than Rabin when the expected average chunk sizes are 4KB and 8KB respectively. Note that all detected low-entropy strings have the same length. For Rabin, the length is equal to Rabin's maximum chunk-size threshold. For AE, it is equal to $w+1$. Therefore, all of these low-entropy strings can be eliminated directly by chunk-level deduplication.

4) Chunk size variance: Another reason for AE's superior DER performance is its smaller chunk size variance than other two algorithms. We have proved it in theory in Table III, here we test it using the real-world datasets. We selected a gcc file at the version of 4.7.0 in the TAR dataset, and processed it using the three algorithms with the expected average chunk size of 4KB, 8KB, and 16KB respectively. Figure 6 depicts the distribution. In this figure we can see that AE achieves more uniform chunk-size distribution, which also means that AE has smaller chunk-size variance than the other two algorithms.



(a) Expected average chunk size is 4KB (b) Expected average chunk size is 8KB (c) Expected average chunk size is 16KB

Fig. 6. Distribution of the chunk size for the three algorithms. Expected average chunk sizes are 4KB, 8KB and 16KB respectively.

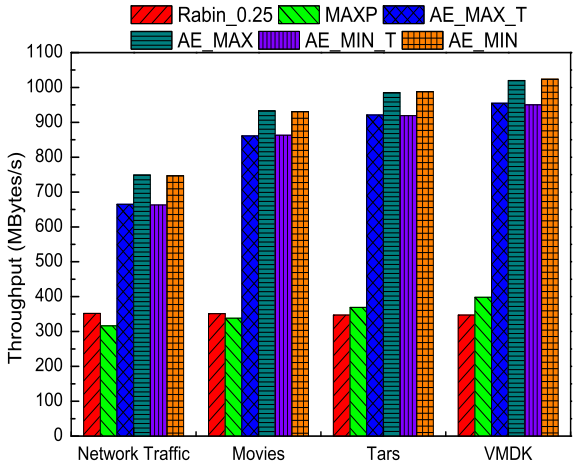


Fig. 7. Chunking throughput of the three algorithms on the four datasets.

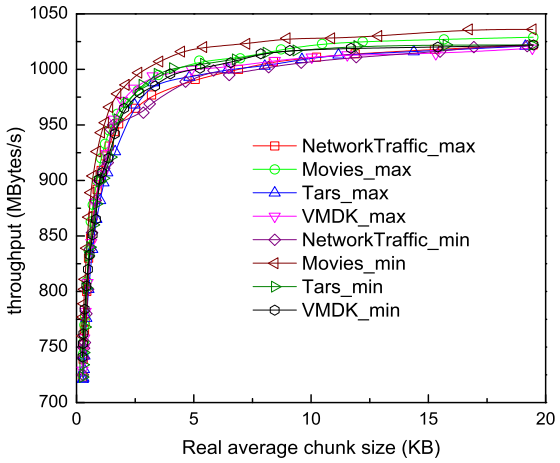


Fig. 8. Sensitivity of chunking throughput to real average chunk size.

C. Chunking throughput

Figure 7 compares the chunking throughput among the AE, Rabin, and MAXP algorithms on the four datasets. The expected average chunk sizes used on the four datasets are 256B, 1KB, 4KB, and 16KB respectively, which serves to test the sensitivity of the algorithms to chunk size. In Figure 7, we draw the following observations. First, imposing a maximum threshold reduces the chunking throughput as shown in the results of AE with/without a maximum chunk-size threshold in Figure 7. This is because imposing a maximum threshold on the chunk size requires one more conditional branch per byte scanned. Combining this observation with the earlier evaluation of deduplication efficiency and sensitivity study of AE, we can draw the conclusion that it is unnecessary to impose a maximum chunk-size threshold on AE.

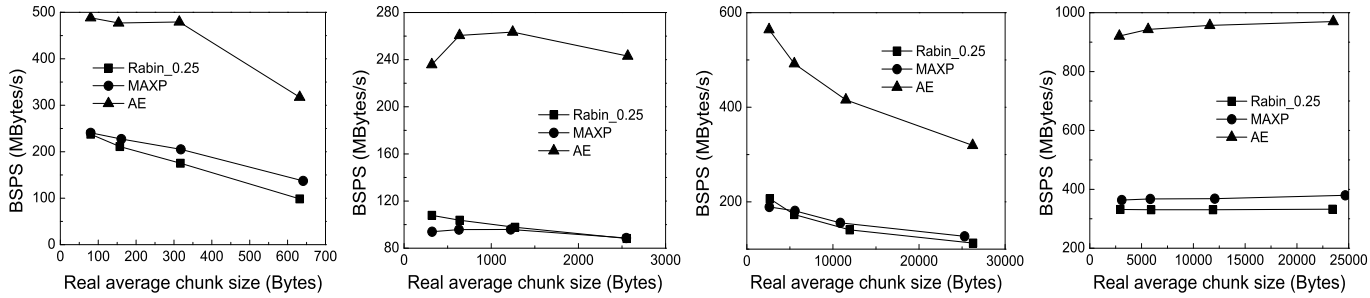
Second, the AE algorithm outperforms the other two algorithms in terms of the chunking throughput. AE achieves a throughput of 1026 MBytes/s, which is about 3 times higher than the other two algorithms. Third, the chunking throughput of the AE algorithm improves as the average chunk size increases. To further measure the sensitivity of the chunking throughput to average chunk size and the dataset, we tested the four datasets using both AE_MAX and AE_MIN. For each test, we adjusted the real average chunk size range from 250B to 20KB. Figure 8 presents the results of our tests. In this figure, we can see that the chunking throughput increases sharply when the real average chunk size is less than 2KB, and after that, the increase in chunking throughput gradually becomes slow. The figure also shows that the chunking throughput of AE is largely unaffected by the type of dataset. In addition, the chunking throughput of MAXP has a small improvement as the average chunk size increases, from 316 to 398 MBytes/s. The chunking throughput of Rabin is largely independent of the chunk size.

D. Bytes saved per second

In this section, we use the metric, called "Bytes Saved Per Second" (BSPS) [28], which considers both the deduplication efficiency and the chunking throughput performance, to measure the efficiency of different CDC algorithms. BSPS can be calculated using following expression.

$$BSPS = \frac{the\ size\ of\ deduplicated\ data}{the\ size\ of\ the\ input\ stream} \times throughput$$

Figure 9 plots the BSPS of the deduplication systems employing the AE, MAXP, and Rabin based chunking algorithms on the four datasets. As can be seen from the figure, the Rabin algorithm achieves comparable BSPS performance to the MAXP algorithm, while the AE algorithm improves the BSPS performance of the state-of-the-art CDC algorithms by a factor of 2.6 (2.6x) on average. The main reason why AE achieves the highest performance of BSPS is its fast and efficient chunking scheme that finds the extreme value in an asymmetric sliding window without having to backtrack, which has the potential to significantly speedup the process of redundancy elimination in data-intensive network and cloud applications.



(a) Network Traffic (b) Movies (c) Tars (d) VMDK

Fig. 9. Deduplicated data per second achieved by the three algorithms.

V. Conclusion

We presented AE, a new CDC algorithm that effectively employs an asymmetric sliding window to find the local extreme value for fast content-defined chunking. As a result, AE is shown to have lower computational overheads and thus higher chunking throughput, smaller chunk size variance, and the ability to eliminate more low-entropy strings than the state-of-the-art algorithms. Finally, our experimental evaluation based on four real-world datasets demonstrates the robust and superior performance of AE in terms of deduplication efficiency and chunking throughput over the state-of-the-art Rabin and MAXP chunking algorithms.

Acknowledgment

The work was partly supported by National Basic Research 973 Program of China under Grant No. 2011CB302301; NSFC No. 61025008, 61173043, 61232004, and 6140050892; 863 Project 2013AA013203; US NSF under Grants IIS-0916859, CCF-0937993, CNS-1116606, and CNS-1016609; Fundamental Research Funds for the Central Universities, HUST, under Grant No. 2014QNRC019. This work was also supported by Key Laboratory of Information Storage System, Ministry of Education, China.

References

[1] J. Gantz and D. Reinsel, "The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east," *IDC iView: IDC Analyze the Future*, 2012.

[2] E. Kruus, C. Ungureanu, and C. Dubnicki, "Bimodal content defined chunking for backup streams," in *Proc. USENIX FAST, 2010*.

[3] W. Xia, H. Jiang, D. Feng, and Y. Hua, "Silo: A similarity-locality based near-exact deduplication scheme with low ram overhead and high throughput." in *Proc. USENIX ATC, 2011*.

[4] S. Sanadhya, R. Sivakumar, K.-H. Kim, P. Congdon, S. Lakshmanan, and J. P. Singh, "Asymmetric caching: improved network deduplication for mobile devices," in *Proc. ACM MobiCom, 2012*.

[5] A. Anand, V. Sekar, and A. Akella, "Smartre: an architecture for coordinated network-wide redundancy elimination," in *Proc. ACM SIGCOMM, 2009*.

[6] A. Tridgell, *Efficient algorithms for sorting and synchronization*. Australian National University Canberra, 1999.

[7] A. Muthitacharoen, B. Chen, and D. Mazieres, "A low-bandwidth network file system," in *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5, 2001, pp. 174–187.

[8] S. Quinlan and S. Dorward, "Venti: A new approach to archival storage," in *Proc. USENIX FAST, 2002*.

[9] P. Kulkarni, F. Douglis, J. D. LaVoie, and J. M. Tracey, "Redundancy elimination within large collections of files." in *Proc. USENIX ATC, 2004*.

[10] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee, "Redundancy in network traffic: findings and implications," in *Proc. ACM SIGMETRICS, 2009*.

[11] K. Eshghi and H. K. Tang, "A framework for analyzing and improving content-based chunking algorithms," *Hewlett-Packard Labs Technical Report TR*, vol. 30, 2005.

[12] S. Schleimer, D. S. Wilkerson, and A. Aiken, "Winnowing: local algorithms for document fingerprinting," in *Proc. ACM SIGMOD, 2003*.

[13] A. El-Shimi, R. Kalach, A. Kumar, A. Ottean, J. Li, and S. Sengupta, "Primary data deduplication-large scale study and system design." in *Proc. USENIX ATC, 2012*.

[14] M. O. Rabin, *Fingerprinting by random polynomials*. Center for Research in Computing Techn., Aiken Computation Laboratory, Univ., 1981.

[15] W. Xia, H. Jiang, D. Feng, L. Tian, M. Fu, and Y. Zhou, "Ddelta: A deduplication-inspired fast delta compression approach," *Performance Evaluation*, vol. 79, pp. 258–272, 2014.

[16] W. Xia, H. Jiang, D. Feng, and L. Tian, "Combining deduplication and delta compression to achieve low-overhead data reduction on backup datasets," in *Proc. IEEE DCC, 2014*.

[17] N. T. Spring and D. Wetherall, "A protocol-independent technique for eliminating redundant network traffic," in *Proc. ACM SIGCOMM, 2000*.

[18] P. Bhatotia, R. Rodrigues, and A. Verma, "Shredder: Gpu-accelerated incremental storage and computation." in *Proc. USENIX FAST, 2012*.

[19] W. Xia, H. Jiang, D. Feng, L. Tian, M. Fu, and Z. Wang, "P-dedupe: Exploiting parallelism in data deduplication system," in *Proc. IEEE NAS, 2012*.

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

[21] B. Agarwal, A. Akella, A. Anand, A. Balachandran, P. Chitnis, C. Muthukrishnan, R. Ramjee, and G. Varghese, "Endre: An end-system redundancy elimination service for enterprises." in *Proc. USENIX NSDI, 2010*.

[22] N. S. Bjorner, Y. Gurevich, and D. Teodosiu, "Efficient chunking algorithm," 2012, uS Patent 8,117,173.

[23] D. T. Meyer and W. J. Bolosky, "A study of practical deduplication," in *Proc. USENIX FAST, 2011*.

[24] S. Al-Kiswany, A. Gharaibeh, E. Santos-Neto, G. Yuan, and M. Ripeanu, "Storegpu: exploiting graphics processing units to accelerate distributed storage systems," in *Proc. ACM HPDC, 2008*.

[25] M. Fu, "Destor: An experimental platform for chunk-level data deduplication," https://github.com/fomy/destor, 2014.

[26] B. Jenkins, "Hash functions," *Dr Dobb's Journal*, 9707, Sept. 1997.

[27] "IMDb," http://www.imdb.com/chart/top/, IMDb chart Top 250.

[28] Y. Fu, H. Jiang, N. Xiao, L. Tian, and F. Liu, "Aa-dedupe: An application-aware source deduplication approach for cloud backup services in the personal computing environment," in *Proc. IEEE CLUSTER, 2011*.