The Design of Fast Content-Defined Chunking for Data Deduplication Based Storage Systems

Wen Xia, *Member, IEEE*, Xiangyu Zou, *Student Member, IEEE*, Hong Jiang, *Fellow, IEEE*, Yukun Zhou, Chuanyi Liu, Dan Feng, *Member, IEEE*, Yu Hua, *Senior Member, IEEE*, Yuchong Hu, and Yucheng Zhang

Abstract—Content-Defined Chunking (CDC) has been playing a key role in data deduplication systems recently due to its high redundancy detection ability. However, existing CDC-based approaches introduce heavy CPU overhead because they declare the chunk cut-points by computing and judging the rolling hashes of the data stream byte by byte. In this article, we propose FastCDC, a Fast and efficient Content-Defined Chunking approach, for data deduplication-based storage systems. The key idea behind FastCDC is the combined use of five key techniques, namely, gear based fast rolling hash, simplifying and enhancing the Gear hash judgment, skipping sub-minimum chunk cut-points, normalizing the chunk-size distribution in a small specified region to address the problem of the decreased deduplication ratio stemming from the cut-point skipping, and last but not least, rolling two bytes each time to further speed up CDC. Our evaluation results show that, by using a combination of the five techniques, FastCDC is 3-12X faster than the state-of-the-art CDC approaches, while achieving nearly the same and even higher deduplication ratio as the classic Rabin-based CDC. In addition, our study on the deduplication throughput of FastCDC-based Destor (an open source deduplication project) indicates that FastCDC helps achieve 1.2-3.0X higher throughput than Destor based on state-of-the-art chunkers.

Index Terms—Data deduplication, content-defined chunking, storage system, performance evaluation

1 Introduction

Data deduplication, an efficient approach to data reduction, has gained increasing attention and popularity in large-scale storage systems due to the explosive growth of digital data. It eliminates redundant data at the file- or chunk-level and identifies duplicate contents by their cryptographically secure hash signatures (e.g., SHA1 fingerprint). According to deduplication studies conducted by Microsoft [1], [2] and EMC [3], [4], about 50 and 85 percent of the data in their production primary and secondary storage systems, respectively, are redundant and could be removed by the deduplication technology.

In general, chunk-level deduplication is more popular than file-level deduplication because it identifies and removes redundancy at a finer granularity. For chunk-level deduplication, the simplest chunking approach is to cut the file or data stream into equal, fixed-size chunks, referred to as Fixed-Size Chunking (FSC) [5]. Content-Defined Chunking (CDC) based approaches are proposed to address the *boundary-shift* problem faced by the FSC approach [6]. Specifically, CDC declares chunk boundaries based on the byte contents of the data stream instead of on the byte offset, as in FSC, and thus helps detect more redundancy for deduplication. According to some recent studies [1], [2], [7], [8], CDC-based deduplication approaches are able to detect about 10-20 percent more redundancy than the FSC approach.

Currently, the most popular CDC approaches determine chunk boundaries based on the Rabin fingerprints of the content, which we refer to as Rabin-based CDC [6], [9], [10]. Rabin-based CDC is highly effective in duplicate detection but time-consuming, because it computes and judges (against a condition value) Rabin fingerprints of the data stream byte by byte [11]. In order to speed up the CDC process, other hash algorithms have been proposed to replace the Rabin algorithm for CDC, such as SampeByte [12], Gear [13], etc. Meanwhile, the abundance of computation resources afforded by multi and manycore processors [14], [15] or GPU processors [16], [17], [18] has been leveraged for CDC acceleration.

Generally, CDC consists of two distinctive and sequential stages: *(1) hashing* in which fingerprints of the data contents are generated and *(2) hash judgment* in which fingerprints are compared against a given value to identify and declare chunk cut-points. Our previous study of delta compression, Ddelta [13], suggests that the Gear hash (i.e., $fp = (fp << 1) + G(b)$, see Section 3.2) is more efficient as a rolling hash for

- *W. Xia is with the Harbin Institute of Technology, Shenzhen 518055, China, Cyberspace Security Research Center, Peng Cheng Laboratory, Shenzhen 518055, China, and also with the Wuhan National Laboratory for Optoelectronics, Wuhan 430074, China. E-mail: xiawen@hit.edu.cn.*
- *X. Zou and C. Liu are with the Harbin Institute of Technology, Shenzhen 518055, China, and also with the Cyberspace Security Research Center, Peng Cheng Laboratory, Shenzhen 518055, China. E-mail: xdnzxy@gmail.com, liuchuanyi@hit.edu.cn.*
- *H. Jiang is with the Department of Computer Science and Engineering, University of Texas at Arlington, TX 76019. E-mail: hong.jiang@uta.edu.*
- *Y. Zhou, D. Feng, Y. Hua, Y. Hu, and Y. Zhang are with the Wuhan National Laboratory for Optoelectronics, School of Computer Sci.&Tech., Huazhong University of Science and Technology, Wuhan 430074, China. E-mail: {ykzhou, dfeng, csyhua, yuchonghu, cszyc}@hust.edu.cn.*

Manuscript received 29 July 2019; revised 21 Feb. 2020; accepted 25 Mar. 2020. Date of publication 2 Apr. 2020; date of current version 17 Apr. 2020. (Corresponding author: Chuanyi Liu.) Recommended for acceptance by T. Kosar. Digital Object Identifier no. 10.1109/TPDS.2020.2984632

1045-9219 © 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on May 29,2020 at 23:55:24 UTC from IEEE Xplore. Restrictions apply.

CDC. To the best of our knowledge, Gear appears to be one of the fastest rolling hash algorithms for CDC at present since it use much less calculation operations than others. However, our empirical and analytical studies on Gear-based CDC obtain three important observations:

- Observation ①: the Gear-based CDC has the potential problem of low *deduplication ratio* (i.e., the percentage of redundant data reduced), about 10-50 percent lower on some datasets (detailed in Section 4.2). This is because, in the *hash judgment* stage of Gear-based CDC, the sliding window size is very small, only 13 bytes in its current implementation [13].
- Observation ②: the *hash judgment* stage becomes the new performance bottleneck in Gear-based CDC. This is because the accelerated *hashing* stage by Gear, has shifted the bottleneck to the *hash judgment* stage.
- Observation ③: Enlarging the predefined minimum chunk size (used in CDC to avoid the very small-sized chunks [6]) can further speed up the chunking process (called *cut-point skipping* in this paper) but at the cost of *decreasing the deduplication ratio* in Gear-based CDC. This is because many chunks with skipped cut-points are not divided truly according to the data contents (i.e., content-defined). Our large scale study (detailed in Section 4.3) suggests that skipping this predefined min chunk size usually increases the chunking speed by the ratio of $\frac{\underline{Predefined\ min\ chunk\ size}}{Expected\ avg.\ chunk\ size}$ but decreases the deduplication ratio (about 15 percent decline in the worst case).

Therefore, motivated by the above three observations, we proposed FastCDC, a Fast and efficient CDC approach that addresses the problems of low deduplication efficiency and expensive hash judgement faced by Gear-based CDC. To address the problems observed in the 1st and 2nd observations, we use an approach of *enhancing and simplifying the hash judgment* to further reduce the CPU operations during CDC for data deduplication. Specifically, FastCDC pads several zero bits into the mask value in its hash-judging statement to enlarge the sliding window size to the size of 48 Bytes used by Rabin-based CDC, which makes it able to achieve nearly the same deduplication ratio as the Rabin-based CDC; Meanwhile, by further simplifying and optimizing the hash-judging statement, FastCDC decreases the CPU overhead for the hash judgment stage in CDC.

For the 3rd observation and to further speed up chunking, FastCDC employs a novel normalized Content-Defined Chunking scheme, called *normalized chunking*, that normalizes the chunk-size distribution to a specified region that *is guaranteed to be larger than the minimum chunk size* to effectively address the problem facing the cut-point skipping approach. Specifically, FastCDC selectively changes the number of mask bits '1' in the hash-judging statement of CDC, and thus it normalizes the chunk-size distribution to a small specified region (e.g., 8KB$\sim$16KB), i.e., the vast majority of the generated chunks fall into this size range, and thus minimizes the number of chunks of either too small or large in size. The benefits are twofold. ①, it increases the deduplication ratio by reducing the number of large-sized chunks. ②, it reduces the number of small-sized chunks, which makes it possible to combine with the cut-point skipping

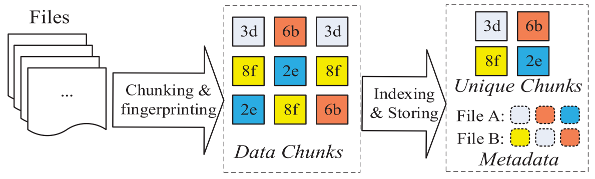


Fig. 1. General workflow of chunk-level data deduplication.

technique above to maximize the CDC speed without sacrificing the deduplication ratio.

In addition, we propose a technique called "rolling two bytes each time" for FastCDC, which further reduces the calculation operations in the *hashing* stage by rolling two bytes each time to calculate the chunking fingerprints in the *hashing* stage, and then judging the even and odd bytes respectively in the *hash judgement* stage. This further accelerates the chunking process while achieving exactly the same chunking results.

Our evaluation results based on seven large-scale datasets, suggest that FastCDC is about 3-12$\times$ faster than the state of art, while ensuring a comparably high deduplication ratio. In addition, we have incorporated FastCDC in Destor [19], an open source data deduplication system, and evaluation shows that Destor using FastCDC helps achieve about 1.2-3.0X higher system throughout than using other CDC approaches. Meanwhile, due to its simplicity and effectiveness, FastCDC has been adopted as the default chunker by several known open source Github projects to speed up the detection of duplicate contents, such as Rdedup [20], Content Blockchain [21], etc. The released Rdedup version 2.0.0 states: "rdedupe store performance has been greatly improved by implementing many new techniques" and "our default CDC algorithm is now FastCDC".

2 Background

Recently, chunk-level data deduplication becomes one of the most popular data reduction method in storage systems for improving storage and network efficiency. As shown in Fig. 1, it splits a file into several contiguous chunks and removes duplicates by computing and indexing hash digests (or called fingerprints, such as SHA-1) of chunks [5], [6], [22], [23]. The fingerprints matching means that their corresponding chunks are duplicate, which thus simplifies the global duplicates detection in storage systems. In the past ten years, data deduplication technique has been demonstrated its space efficiency functionality in the large-scale production systems of Microsoft [1], [2] and EMC [3], [4].

Chunking is the first critical step in the operational path of data deduplication, in which a file or data stream is divided into small chunks so that each can be duplicate-identified. Fixed-Size Chunking (FSC) [5] is simple and fast but may face the problem of low deduplication ratio that stems from the *boundary-shift* problem [6], [24]. For example, if one or several bytes are inserted at the beginning of a file, all current chunk cut-points (i.e., boundaries) declared by FSC will be shifted and no duplicate chunks will be detected.

Content-Defined Chunking (CDC) is proposed to solve the *boundary-shift* problem. CDC uses a sliding-window technique on the content of files and computes a hash value (e.g., Rabin fingerprint [6], [9]) of the window. A chunk cut-point

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on May 29,2020 at 23:55:24 UTC from IEEE Xplore. Restrictions apply.

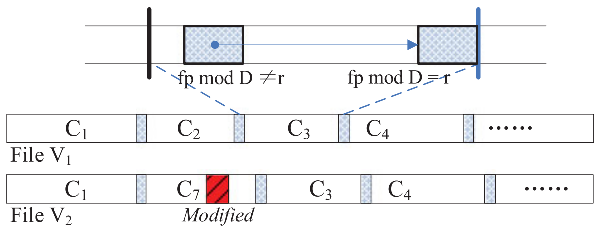


Fig. 2. The sliding window technique for the CDC algorithm. The hash value of the sliding window, *fp*, is computed via the Rabin algorithm (this is the *hashing stage* of CDC). If the lowest $log_2 D$ bits of the hash value matches a threshold value r, i.e., *fp* mod $D = r$, this offset (i.e., the current position) is marked as a chunk cut-point (this is the *hash-judging stage* of CDC).

is declared if the hash value satisfies some pre-defined condition. As shown in Fig. 2, to chunk a file V_2 that is modified from the file V_1, the CDC algorithm can still identify the correct boundary of chunks C_1, C_3, and C_4, whose contents have not been modified. As a result, CDC outperforms FSC in terms of deduplication ratio and has been widely used in backup [3], [22] and primary [1], [2] storage systems.

Although the widely used Rabin-based CDC helps obtain a high deduplication ratio, it incurs heavy CPU overhead [14], [16], [18], [25]. Specifically, in Rabin-based CDC, Rabin hash for a sliding window containing the byte sequence B_1, $B_2, \ldots, B_\alpha$ is defined as a polynomial

$$Rabin(B_1, B_2, \ldots, B_\alpha) = A(p) = \left\{ \sum_{x=1}^{\alpha} B_x p^{\alpha - x} \right\} mod\ D, \quad (1)$$

where D is the average chunk size, α is the number of bytes in the sliding window, and p is a number representing an irreducible polynomial [9]. Rabin hash is a *rolling hash* algorithm since it is able to compute the hash in an iterative fashion, i.e., the current hash can be incrementally computed from the previous value as

$$\begin{aligned} &Rabin(B_2, B_3, \ldots, B_{\alpha+1}) \\ &= \{[Rabin(B_1, B_2, \ldots, B_\alpha) - B_1 p^{\alpha-1}]p + B_{\alpha+1}\} modS. \end{aligned} \quad (2)$$

However, Rabin-based CDC is time-consuming because it computes and judges the hashes of the data stream byte by byte, which renders the chunking process a performance bottleneck in deduplication systems. There are many approaches that accelerate the CDC process for deduplication systems and they can be broadly classified as either algorithmic oriented or hardware oriented. We summarize below some of these approaches that represent the state of the art.

Algorithmic-Oriented CDC Optimizations. Since the frequent computations of Rabin fingerprints for CDC are time-consuming, many alternatives to Rabin have been proposed to accelerate the CDC process [12], [13], [24]. SampleByte [12] is designed for providing fast chunking for fine-grained network redundancy elimination, usually eliminating duplicate chunks as small as 32-64 bytes. It uses one byte to declare a fingerprint for chunking, in contrast to Rabin that uses a sliding window, and skips $\frac{1}{2}$ of the expected chunk size before chunking to avoid generating extremely small-sized strings or chunks (they called "avoid oversampling"). Gear [13] uses fewer operations to generate rolling hashes by means of a small random integer table to map the values of the byte contents, so as to achieve higher chunking throughput. AE [24] is a non-rolling-hash-based chunking algorithm that employs an asymmetric sliding window to identify extremums of data stream as cut-points, which reduces the computational overhead for CDC. Yu *et al.* [26] adjust the function for selecting chunk boundaries such that if weak conditions are not met, the sliding window can jump forward, avoiding unnecessary calculation steps. RapidCDC [27] leverages the data locality to record the chunking positions to reduce the CDC computations for the duplicate chunks to appear next time.

Hardware-Oriented CDC Optimizations. StoreGPU [16], [17] and Shredder [18] make full use of GPU's computational power to accelerate popular compute-intensive primitives (i.e., chunking and fingerprinting) in data deduplication. P-Dedupe [14] pipelines deduplication tasks and then further parallelizes the sub-tasks of chunking and fingerprinting with multiple threads and thus achieves higher throughput. SS-CDC [28] proposes a two-stage prallel content-defined chunking approach without compromising deduplication ratio.

It is noteworthy that there are other chunking approaches trying to achieve a higher deduplication ratio but introduce more computation overhead on top of the conventional CDC approach. TTTD [29] and Regression chunking [2] introduces one or more additional thresholds for chunking judgment, which leads to a higher probability of finding chunk boundaries and decreases the chunk size variance. MAXP [30], [31], [32] treats the extreme values in a fixed-size region as cut-points, which also results in smaller chunk size variance. In addition, Bimodal chunking [33], Subchunk [34], and FBC [35] re-chunk the non-duplicate chunks into smaller ones to detect more redundancy.

For completeness and self-containment we briefly discuss *other relevant deduplication issues* here. A typical data deduplication system follows the workflow of chunking, fingerprinting, indexing, and storage management [19], [22], [36], [37]. The fingerprinting process computes the cryptographically secure hash signatures (e.g., SHA1) of data chunks, which is also a compute-intensive task but can be accelerated by certain pipelining or parallelizing techniques [14], [38], [39], [40]. Indexing refers the process of identifying the identical fingerprints for checking duplicate chunks in large-scale storage systems, which has been well explored in many previous studies [19], [22], [41], [42]. Storage management refers to the storage and possible post-deduplication processing of the non-duplicate chunks and their metadata, including such processes as related to further compression [13], defragmentation [43], reliability [44], security [45], etc. In this paper, we focus on designing a very fast and efficient chunking approach for data deduplication since the CPU-intensive CDC task has been widely recognized as a major performance bottleneck of the CDC-based deduplication system [17], [18], [27], [28].

3 FastCDC Design and Implementation

3.1 FastCDC Overview

FastCDC aims to provide high performance CDC. And there are three metrics for evaluating CDC performance, namely, deduplication ratio, chunking speed, and the average generated chunk size. Note that the average generated

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on May 29,2020 at 23:55:24 UTC from IEEE Xplore. Restrictions apply.

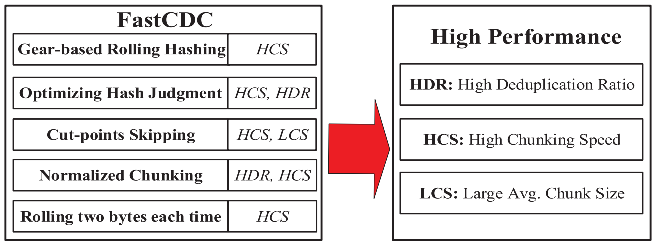

FastCDC
Gear-based Rolling Hashing | HCS
Optimizing Hash Judgment | HCS, HDR
Cut-points Skipping | HCS, LCS
Normalized Chunking | HDR, HCS
Rolling two bytes each time | HCS
High Performance
HDR: High Deduplication Ratio
HCS: High Chunking Speed
LCS: Large Avg. Chunk Size

Fig. 3. The five key techniques used in FastCDC and their corresponding benefits for high performance CDC.

TABLE 1
The Hashing Stage of the Rabin- and Gear-Based CDC

Name	Pseudocode	Speed
Rabin	$fp = ((fp \hat{} U(a)) << 8) \| b \hat{} T[fp >> N]$	Slow
Gear	$fp = (fp << 1) + G(b)$	Fast

Here 'a' and 'b' denote contents of the first and last byte of the sliding window respectively, 'N' is the length of the content-defined sliding window, and 'U', 'T', 'G' denote the predefined arrays [6], [11], [13]. 'fp' represents the fingerprint of the sliding window.

chunk size is also an important CDC performance metric since it reflects the metadata overhead for deduplication indexing, i.e., the larger the generated chunk size is, the fewer the number of chunks and thus the less metadata will be processed by data deduplication.

Generally, it is difficult, if not impossible, to improve these three performance metrics simultaneously because they can be conflicting goals. For example, a smaller average generated chunk size leads to a higher deduplication ratio, but at the cost of lower chunking speed and high metadata overheads. Thus, *FastCDC is designed to strike a sensible trade-off among these three metrics so as to strive for high performance CDC, by using a combination of the five techniques with their complementary features* as shown in Fig. 3.

- Gear-based rolling hashing: due to its hashing simplicity and rolling effectiveness, Gear hash is shown to be one of the fastest rolling hash algorithms for CDC, and we introduce it and discuss its chunking efficiency in detail in Section 3.2.
- Optimizing hash judgment: using a zero-padding scheme and a simplified hash-judging statement to speed up CDC without compromising the deduplication ratio, as detailed in Section 3.3.
- Sub-minimum chunk cut-point skipping: enlarging the predefined minimum chunk size and skipping cut-points for chunks smaller than that to provide a higher chunking speed and a larger average generated chunk size, as detailed in Section 3.4.
- Normalized chunking: selectively changing the number of mask '1' bits for the hash judgment to approximately normalize the chunk-size distribution to a small specified region that is just larger than the predefined minimum chunk size, ensuring both a higher deduplication ratio and higher chunking speed, as detailed in Section 3.5.
- Rolling two bytes each time: furhter speeding up chunking without affecting the chunking results by reducing the calculation operations in the hashing stage by rolling two bytes each time to calculate the chunking fingerprints in the *hashing* stage, and then judging the even and odd bytes respectively in the *hash judgement* stage, as detailed in Section 3.7.

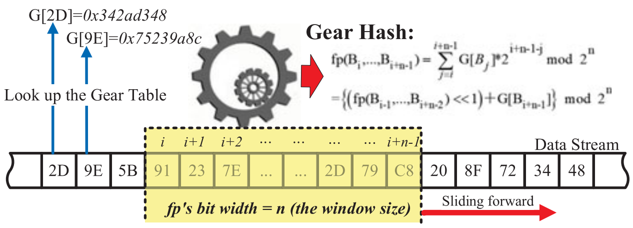

G[2D]=0x342ad348
G[9E]=0x75239a8c
Look up the Gear Table
Gear Hash:
$fp(B_i,\ldots,B_{i+n-1}) = \sum_{j=i}^{i+n-1} G[B_j]*2^{i+n-1-j} \bmod 2^n$
$= \{(fp(B_{i-1},\ldots,B_{i+n-2}) << 1) + G[B_{i+n-1}]\} \bmod 2^n$
i i+1 i+2 … … … … i+n-1
Data Stream
2D 9E 5B 91 23 7E … … 2D 79 C8 20 8F 72 34 48
fp's bit width = n (the window size)
Sliding forward

Fig. 4. A schematic diagram of the Gear hash.

In general, the key idea behind FastCDC is the combined use of the above five key techniques for CDC acceleration, especially employing normalized chunking to address the problem of decreased deduplication ratio facing the cut-point skipping, and thus achieve high performance CDC on the three key metrics.

3.2 Gear-Based Rolling Hashing

In this subsection, we elaborate on and analyze the Gear-based rolling hash, and then introduce the new challenges and opportunities after we introduce Gear-based CDC. Gear-based rolling hash is first employed by Ddelta [13] for delta compression, which helps provide a higher delta encoding speed and is suggested to be a good rolling hash candidate for CDC.

A good hash function must have a uniform distribution of hash values regardless of the hashed content. As shown in Fig. 4, Gear-based CDC achieves this in two key ways: (1) It employs an array of 256 random 64-bit integers to map the values of the byte contents in the sliding window (i.e., the calculated bytes, whose size is the bit-width of the *fp*); and (2) The addition ("+") operation adds the new byte in the sliding window into Gear hashes while the left-shift ("$<<$") operation helps strip away the last byte of the last sliding window (e.g., B_{i-1} in Fig. 4). This is because, after the "$<<$" and modulo operations, the last byte B_{i-1} will be calculated into the *fp* as the $(G[B_{i-1}] << n)\ mod\ 2^n$, which will be equal to zero. As a result, Gear generates uniformly distributed hash values by using only three operations (i.e., "+", "$<<$", and an array lookup), enabling it to move quickly through the data content for the purpose of CDC. Table 1 shows a comparison among the two rolling hash algorithms: Rabin and Gear, which suggests Gear uses far fewer calculation operations than Rabin, thus being a good rolling hash candidate for CDC.

To better illustrate Gear-based CDC, Algorithm 3.2 provides the detailed chunking pseudo code that uses the *Gear* table for calculating the rolling fingerprints and the hash judging statement similar to the classical Rabin-based CDC. In Fig. 5, we compare the chunk-size distributions each generated by Rabin- and Gear-based CDC on the random-number workload, and against the mathematical analysis based on Equation (3) (see Section 3.4), which indicates that the three are almost identical (for more chunk-size distribution results, see Fig. 12 in Section 4.2). And the previous study Ddelta [13] also suggests Gear is considered to be a good rolling hash candidate for CDC both on the hashing efficiency and on the

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on May 29,2020 at 23:55:24 UTC from IEEE Xplore. Restrictions apply.

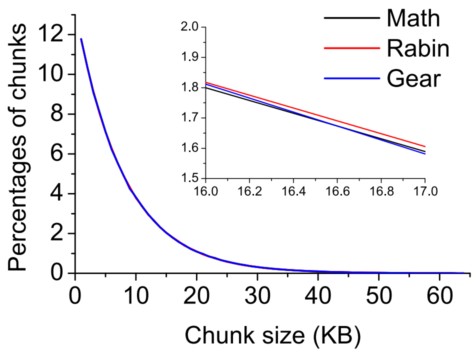


Fig. 5. Chunk-size distributions of Rabin- and Gear-based CDC approaches with average chunk size of 8KB (without max/min chunk size requirement). "Rabin" and "Gear" denote our experimental results after CDC and "Math" denotes theoretical analysis, where they are shown to be nearly identical.

chunking efficiency. However, according to our experimental analysis, there are still challenges and opportunities for Gear-based CDC, such as low deduplication ratio, expensive hash judgment, further acceleration by skipping, etc. We elaborate on these issues as follows.

Algorithm 1. GearCDC8KB

```
Input: data buffer, $src$; buffer length, $n$
Output: chunking breakpoint $i$
$MinSize \leftarrow 2KB$;    $MaxSize \leftarrow 64KB$;
$i \leftarrow MinSize$;    $fp \leftarrow 0$;
if $n \leq MinSize$ then
  return $n$
while $i < n$ do
  $fp = (fp << 1) + Gear[\ src[i]\ ]$;
  if $(fp \ \& \ 0x1fff \ == \ 0x78) \ || \ i \ >= \ MaxSize$ then
    return $i$;
return $i$;
```

Low Deduplication Ratio Due to the Limited Sliding Window Size. The Gear-based CDC has the potential problem of low *deduplication ratio*, about 10-50 percent lower on some datasets (see the evaluation results in Section 4.2). This is becase, the traditional hash judgment for the Rabin-based CDC, as shown in Fig. 2 (i.e., "*fp mod D==r*"), is also used by the Gear-based CDC [13] as shown in Algorithm 3.2. But this results in a smaller sized sliding window used by Gear-based CDC since it uses Gear hash for chunking. For example, as shown in Fig. 6, the sliding window size of the Gear-based CDC will be equal to the number of the '1' bits used by the mask value. Therefore, when using a mask value of 0x1fff (i.e., $2^{13} - 1$, there are thirteen '1' bits) for the expected chunk size of 8 KB, the sliding window for the Gear-based CDC would be 13 bytes while that of the Rabin-based CDC would be 48 bytes [6]. The smaller sliding window size of the Gear-based CDC can lead to more chunking position collisions (i.e., randomly marking the different positions as the chunk cut-points), resulting in the decrease in deduplication ratio.

The Expensive Hash Judgment. In Gear-based CDC, the accelerated *hashing* stage by the fast Gear, has shifted the bottleneck to the *hash judgment* stage that requires more operations as shown in Algorithm 3.2. Our implementation and in-depth analysis of Gear-based CDC on several datasets

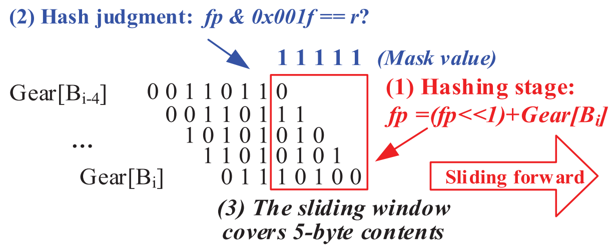


Fig. 6. An example of the sliding window technique used in the Gear-based CDC. Here CDC consists of two stages: hashing and hash judgment. The size of the sliding window used for hash judgment is only 5 bytes because of the computation principles of the Gear hash.

(detailed in Section 4) suggest that *its hash-judging stage accounts for more than 60 percent of its CPU overhead during CDC after the fast Gear hash is introduced*. Thus, there is a lot of room for the optimization of the hash judging stage to further accelerate the CDC process as discussed later in Section 3.3.

Further Speed up Chunking by Skipping. Another observation is that the minimum chunk size used for avoiding extremely small-sized chunks, can be also employed to speed up CDC by the cut-point skipping, i.e., eliminating the chunking computation in the skipped region. But this minimum chunk size for cut-point skipping approach decreases the deduplication ratio (as demonstrated in the evaluation results in Fig. 12c in Section 4.3) since many chunks are not divided truly according to the data contents, i.e., not really content-defined.

The last observation from the minimum chunk size skipping motivates us to consider a new CDC approach that (1) keeps all the chunk cut-points that generate chunks larger than a predefined minimum chunk size and (2) enables the chunk-size distribution to be normalized to a relatively small specified region, an approach we refer to as *normalized chunking* in this paper, as described in Section 3.5.

3.3 Optimizing Hash Judgment

In this subsection, we optimize the hash judgment stage on top of the Gear-based CDC, which helps further accelerate the chunking process and increase the deduplication ratio to reach that of the Rabin-based CDC. More specifically, FastCDC incorporates two main optimizations as elaborated below.

Enlarging the Sliding Window Size by Zero Padding. As discussed in Section 3.2, the Gear-based CDC employs the same conventional hash judgment used in the Rabin-based CDC, where a certain number of the lowest bits of the fingerprint are used to declare the chunk cut-point, leading to a shortened sliding window for the Gear-based CDC (see Fig. 6) because of the unique feature of the Gear hash. To address this problem, FastCDC enlarges the sliding window size by padding a number of zero bits into the mask value. As illustrated by the example of Fig. 7, FastCDC pads five zero bits into the mask value and changes the hash judgment statement to "*fp & mask == r*". If the masked bits of *fp* match a threshold value r, the current position will be declared as a chunk cut-point. Since Gear hash uses one left-shift and one addition operation to compute the rolling hash, this zero-padding scheme enables 10 bytes (i.e., $B_i, \ldots, B_{i+9}$), instead of the original five bytes, to be involved in the final hash judgment by the five masked one bits (as the red box shown in Fig. 7) and thus makes the

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on May 29,2020 at 23:55:24 UTC from IEEE Xplore. Restrictions apply.

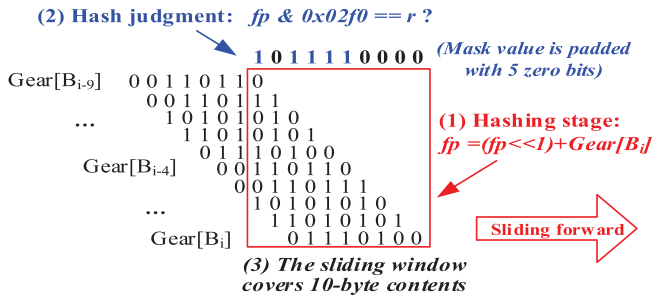

(2) Hash judgment: fp & 0x02f0 == r ?
1 0 1 1 1 1 0 0 0 0
(Mask value is padded with 5 zero bits)
Gear[Bi-9]
...
Gear[Bi-4]
...
Gear[Bi]
(1) Hashing stage: fp =(fp<<1)+Gear[Bi]
Sliding forward
(3) The sliding window covers 10-byte contents

Fig. 7. An example of the sliding window technique proposed for FastCDC. By padding y zero bits into the mask value for hash judgment, the size of the sliding window used in FastCDC is enlarged to about *5+*y bytes, where y*=5* in this example.

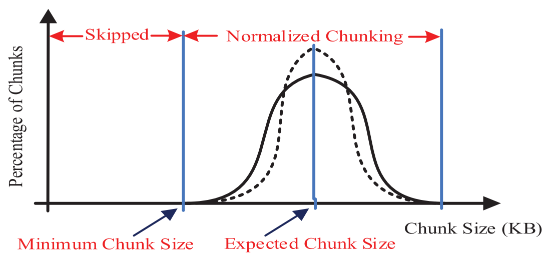


Fig. 8. A conceptual diagram of the normalized chunking combined with the subminimum chunk cut-point skipping. The dotted line shows a higher level of normalized chunking.

sliding window size equal or similar to that of the Rabin-based CDC [6], minimizing the probability of the chunking position collision. As a result, FastCDC is able to achieve a deduplication ratio as high as that by the Rabin-based CDC.

Simplifying the Hash Judgment to Accelerate CDC. The conventional hash judgment process, as used in the Rabin-based CDC, is expressed in the programming statement of "*fp mod D==r*" [6], [13]. For example, the Rabin-based CDC usually defines D and r as *0x02000* and *0x78*, according to the known open source project LBFS [6], to obtain the expected average chunk size of 8 KB. In FastCDC, when combined with the zero-padding scheme introduced above and shown in Fig. 7, the hash judgment statement can be optimized to "*fp & Mask==0*", which is equivalent to "*!fp & Mask*". Therefore, FastCDC's hash judgment statement reduces the register space for storing the threshold value r and avoids the unnecessary comparison operation that compares "*fp & Mask*" and r, thus further speeds up the CDC process as verified in Section 4.2.

3.4 Cut-Point Skipping

Most of CDC-based deduplication systems impose a limit of the maximum and minimum chunk sizes, to avoid the pathological cases of generating many extremely large- or small-sized chunks by CDC [1], [6], [33], [34], [37], [46]. A common configuration of the average, minimum, and maximum parameters follows that used by LBFS [6], i.e., 8, 2, and 64 KB. Our previous study [13] and experimental observations (see Fig. 11 in Section 4.2, using curve fitting) suggest that the cumulative distribution of chunk size X in Rabin-based CDC approaches with an expected chunk size of *8 KB* (without the maximum and minimum chunk size requirements) generally follows an exponential distribution as follows:

$$P(X \le x) = F(x) = (1 - e^{-\frac{x}{8192}}),\ x \ge 0. \tag{3}$$

Note that this theoretical exponential distribution in Equation (3) is based on the assumption that the data content and Rabin hashes of contents (recall Equation (1) and Fig. 2 for CDC) follow a uniform distribution. Equation (3) suggests that the value of the expected chunk size will be 8 KB according to exponential distribution.

According to Equation (3), the chunks smaller than 2 KB and larger than 64 KB would account for about 22.12 and 0.03 percent of the total number of chunks respectively. This means that imposing the maximum chunk size requirement only slightly hurts the deduplication ratio but skipping cut-points before chunking to avoid generating chunks smaller than the prescribed minimum chunk size, or called *sub-minimum chunk cut-point skipping* , will impact the deduplication ratio significantly as evidenced in Fig. 12c. This is because a significant portion of the chunks are not divided truly according to the data contents, but forced by this cut-point skipping.

Given FastCDC's goal of maximizing the chunking speed, enlarging the minimum chunk size and skipping sub-minimum chunk cut-point will help FastCDC achieve a higher CDC speed by avoiding the operations for the hash calculation and judgment in the skipped region. This gain in speed, however, comes at the cost of reduced deduplication ratio. To address this problem, we will develop a normalized chunking approach, to be introduced in the next subsection.

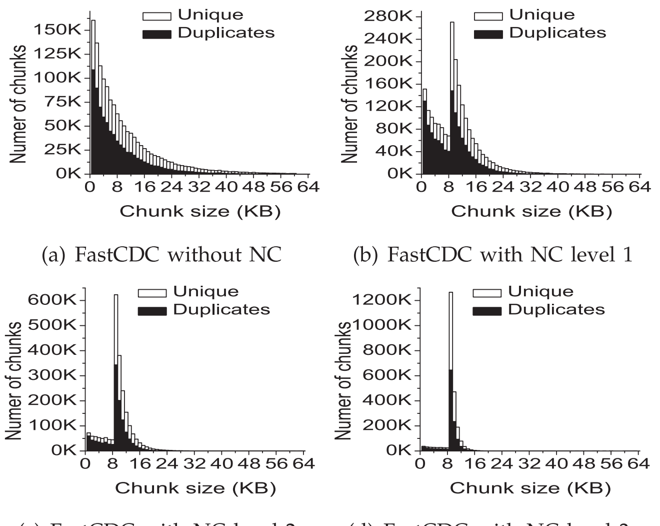
It is worth noting that this cut-point skipping approach, by avoiding generating chunks smaller than the minimum chunk size, also helps increase the average generated chunk size. In fact, the average generated chunk size exceeds the expected chunk size by an amount equal to the minimum chunk size. This is because the F(x) in Equation (3) is changed to $(1 - e^{-\frac{x-MinSize}{8192}})$ after cut-point skipping, thus the value of the expected chunk size becomes 8 KB + minimum chunk size, which will be verified in Section 4.3. The speedup achieved by skipping the sub-minimum chunk cut-point can be estimated by $1+\frac{\text{the minimum chunk size}}{\text{the expected chunk size}}$. The increased chunking speed comes from the eliminated computation on the skipped region, which will also be evaluated and verified in Section 4.3.

3.5 Normalized Chunking

In this subsection, we propose a novel chunking approach, called normalized chunking, to solve the problem of decreased deduplication ratio facing the cut-point skipping approach. As shown in Fig. 8, normalized chunking generates chunks whose sizes are normalized to a specified region centered at the expected chunk size. After normalized chunking, there are almost no chunks of size smaller than the minimum chunk size, which means that normalized chunking enables skipping cut-points for subminimum chunks to reduce the unnecessary chunking computation and thus speed up CDC.

In our implementation of normalized chunking, we selectively change the number of effective mask bits (i.e., the number of '1' bits) for the hash-judging statement. For the traditional CDC approach with expected chunk size of 8 KB (i.e., 2^{13}), 13 effective mask bits are used for hash judgment

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on May 29,2020 at 23:55:24 UTC from IEEE Xplore. Restrictions apply.



(a) FastCDC without NC (b) FastCDC with NC level 1

(c) FastCDC with NC level 2 (d) FastCDC with NC level 3

Fig. 9. Chunk-size distribution of FastCDC with normalized chunking (NC) at different normalization levels.

TABLE 2
Workload Characteristics of the Seven Datasets Used in the Performance Evaluation

Name	Size	Workload descriptions
TAR	56 GB	215 tarred files from several open source projects such as GCC, GDB, Emacs, etc. [47]
LNX	178 GB	390 versions of Linux source code files (untarred). There are totally 16, 381, 277 files [48].
WEB	237 GB	102 days' snapshots of the website: *news.sina.com*, which are collected by crawling software *wget* with a maximum retrieval depth of 3.
VMA	138 GB	90 virtual machine images of different OS release versions, including CentOS, Fedora, etc. [49]
VMB	1.9 TB	125 backups of an Ubuntu 12.04 virtual machine image in use by a research group.
RDB	1.1 TB	198 backups of the Redis key-value store database snapshots, i.e., dump.rdb files.
SYN	2.1 TB	300 synthetic backups. The backup is simulated by the file create/delete/modify operations [50].

(e.g., $fp \,\&\, 0x1fff{=}{=}r$). For normalized chunking, more than 13 effective mask bits are used for hash judgment (e.g., $fp \,\&\, 0x7fff{=}{=}r$) when the current chunking position is smaller than 8 KB, which makes it harder to generate chunks of size smaller than 8 KB. On the other hand, fewer than 13 effective mask bits are used for hash judgment (e.g., $fp \,\&\, 0x0fff{=}{=}r$) when the current chunking position is larger than 8 KB, which makes it easier to generate chunks of size larger than 8 KB. Therefore, by changing the number of '1' bits in FastCDC, the chunk-size distribution will be approximately normalized to a specified region always larger than the minimum chunk size, instead of following the exponential distribution (see Fig. 5).

Generally, there are three benefits or features of normalized chunking (NC):

- NC reduces the number of small-sized chunks, which makes it possible to combine it with the cut-point skipping approach to *achieve high chunking speed without sacrificing the deduplication ratio* as suggested in Fig. 8.
- NC further improves the deduplication ratio by reducing the number of large-sized chunks, which compensates for the reduced deduplication ratio caused by reducing the number of small-sized chunks in FastCDC.
- The implementation of FastCDC does not add additional computing and comparing operations. It simply separates the hash judgment into two parts, before and after the expected chunk size.

Fig. 9 shows the chunk-size distribution after normalized chunking in comparison with FastCDC without NC on the TAR dataset (whose workload characteristics are detailed in Table 2 in Section 4.1). The normalization levels 1, 2, 3 indicate that the normalized chunking uses the mask bits of (14, 12), (15, 11), (16, 10), respectively, where the first and the second integers in the parentheses indicate the numbers of effective mask bits used in the hash judgment before and after the expected chunk size (or *normalized chunk size*) of 8 KB. Fig. 9 also suggests that the chunk-size distribution is a reasonably close approximation of the normal distribution centered on 8 KB at the normalization level of 2 or 3.

As shown in Fig. 9, there are only a very small number of chunks smaller than 2 or 4 KB after normalized chunking while FastCDC without NC has a large number of chunks smaller than 2 or 4 KB (consistent with the discussion in Section 3.4). Thus, when combining NC with the cut-point skipping to speed up the CDC process, only a very small portion of chunk cut-points will be skipped in FastCDC, leading to nearly the same deduplication ratio as the conventional CDC approaches without the minimum chunk size requirement. In addition, normalized chunking allows us to enlarge the minimum chunk size to maximize the chunking speed without sacrificing the deduplication ratio.

It is worth noting that the chunk-size distribution shown in Fig. 9 is not truly normal distribution but an approximation of it. Actually, it follows an improved exponential distribution calculating from Equation (3) as follows (taking the NC 2 as an example and using the average chunk size of 8 KB)

$$P(X \le x) = F(x) = \begin{cases} 1 - e^{-\frac{x}{8192*4}}, & 0 \ \le x \le 8192 \\ 1 - e^{-\frac{x}{8192/4}}, & \quad x \ > \ 8192 \end{cases}. \tag{4}$$

Therefore, Figs. 9c and 9d shows a closer approximation of normal distribution of chunk size achieved by using the normalization levels 2 and 3. Interestingly, the highest normalization level of NC would be equivalent to Fixed-Size Chunking (FSC), i.e., all the chunk sizes are normalized to be equal to the expected chunk size. Since FSC has a very low deduplication ratio but extremely high chunking speed, it means that there will be a "sweet spot" among the normalization level, deduplication ratio, and chunking speed, which will be studied and evaluated in Section 4.

3.6 Putting It All Together

To put things together and in perspective. Algorithm 3.6 describes FastCDC combining the three key techniques: optimizing hash judgment, cut-point skipping, and normalized chunking (with the expected chunk size of 8 KB). The

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on May 29,2020 at 23:55:24 UTC from IEEE Xplore. Restrictions apply.

data structure "Gear" is a predefined array of 256 random 64-bit integers with one-to-one mapping to the values of byte contents for chunking [13].

Algorithm 2. FastCDC8KB (with NC)

```
Input: data buffer, $src$; buffer length, $n$
Output: chunking breakpoint $i$
$MaskS \leftarrow 0x0000d9f003530000LL$;        // 15 '1' bits;
$MaskA \leftarrow 0x0000d93003530000LL$;        // 13 '1' bits;
$MaskL \leftarrow 0x0000d90003530000LL$;        // 11 '1' bits;
$MinSize \leftarrow 2\ KB$;   $MaxSize \leftarrow 64\ KB$;
$fp \leftarrow 0$;  $i \leftarrow MinSize$;  $NormalSize \leftarrow 8\ KB$;
if $n \leq MinSize$ then
  return $n$;
if $n \geq MaxSize$ then
  $n \leftarrow MaxSize$;
else if $n \leq NormalSize$ then
  $NormalSize \leftarrow n$;
for ; $i < NormalSize$; $i{+}{+}$; do
  $fp = (fp << 1) + Gear[\ src[i]\ ]$;
  if ! ( $fp\ \&\ MaskS$ ) then
    return $i$;                 //if the masked bits are all '0';
for ; $i < n$; $i{+}{+}$; do
  $fp = (fp << 1) + Gear[\ src[i]\ ]$;
  if ! ( $fp\ \&\ MaskL$ ) then
    return $i$;                 //if the masked bits are all '0';
return $i$;
```

Algorithm 3. RabinCDC8KB(With NC)

```
Input: data buffer, $src$; buffer length, $n$
Output: chunking breakpoint $i$
$MinSize \leftarrow 2\ KB$;   $MaxSize \leftarrow 64\ KB$;
$fp \leftarrow 0$;  $i \leftarrow MinSize$;  $NormalSize \leftarrow 8\ KB$;
if $n \leq MinSize$ then
  return $n$;
if $n \geq MaxSize$ then
  $n \leftarrow MaxSize$;
else if $n \leq NormalSize$ then
  $NormalSize \leftarrow n$;
for ; $i < NormalSize$; $i{+}{+}$; do
  $fp = ((fp \hat{\ } U(a)) << 8) | b \hat{\ } T[fp >> N]$;
  if $fp\ \&\ (8192 * 2 - 1)\ == 0x78$ then
    return $i$;
for ; $i < n$; $i{+}{+}$; do
  $fp = ((fp \hat{\ } U(a)) << 8) | b \hat{\ } T[fp >> N]$;
  if $fp\ \&\ (8192/2 - 1)\ == 0x78$ then
    return $i$;
return $i$;
```

As shown in Algorithm 3.6, FastCDC uses normalized chunking to divide the chunking judgment into two loops with the optimized hash judgment. Note that FastCDC without normalized chunking is not shown here but can be easily implemented by using the new hash-judging statement "! $fp\ \&\ MaskA$" where the $MaskA$ is padded with 35 zero bits to enlarge the sliding window size to 48 bytes as that used in the Rabin-based CDC [6]. Note that $MaskA$, $MaskS$, and $MaskL$ are three empirically derived values where the padded zero bits are almost evenly distributed for slightly higher deduplication ratio according to our large scale tests.

FastCDC implements normalized chunking by using mask value $MaskS$ and $MaskL$ to make the chunking judgment harder or easier (to generate chunks smaller or larger than the expected chunk size) when the current position is smaller or larger than the expected chunk size, respectively. And the number of '1' bits in $MaskS$ and $MaskL$ can be changed for different normalization levels. The minimum chunk size used in Algorithm 3.6 is $2\ KB$, which can be enlarged to $4\ KB$ or $8\ KB$ to further speed up the CDC process while combining with normalized chunking. Tuning the parameters of minimum chunk size and normalization level will be studied and evaluated in the next Section.

In addition, we implement the normalized chunking scheme in Rabin-based CDC as shown in Algorithm 3.6. This improved Rabin-based CDC is also evaluated in the next Section. Note that the hash judgment optimization for Gear is not applied for Rabin. This is because there are many zero hash values generated by Rabin [9], which results too many positions satisfying the chunking judgment "$(fp\&MaskValue) == false$" and thus the generated average chunk size after Rabin-based CDC will be far blow the expected average chunk size.

3.7 Rolling Two Bytes Each Time

Besides the techniques mentioned in the above subsection, we also propose another independent technique called "rolling two bytes each time" on top of FastCDC. As shown in Algorithm 3.7, the core idea of this technique is that the fingerprint fp is rolling two bytes each time (i.e., "$fp << 2$") in contrast to the traditional way of rolling one byte each time (see Algorithm 3.2), and then we judge the even and odd bytes respectively to determine the chunks' boundaries. Specifically, ① for the even bytes, we use the *Gear_ls* table (*Gear_ls* contains elements from the *Gear* table, which are all left shift one bit) and the mask value *MaskA_ls* (i.e., *MaskA* $<< 1$) for the hash judgment in FastCDC, this is because when we judge the *fp* for the even bytes, *fp* has been already left shift two bits (as described in Algorithm 3.7); ② for the odd bytes, we process the fingerprints using $Gear$ table and the mask value *MaskA* in the traditional way.

Algorithm 4. Rolling Two Bytes Each Time on FastCDC8KB (Without NC for Simplicity)

```
Input: data buffer, $src$; buffer length, $n$
Output: chunking breakpoint $i$
$MaskA \leftarrow 0x0000d93003530000LL$;     // 13 '1' bits ;
$MaskA\_ls \leftarrow (MaskA << 1)$;   $fp \leftarrow 0$;  $i \leftarrow MinSize$;
$MinSize \leftarrow 2KB$;     $MaxSize \leftarrow 64KB$;
if $n \leq MinSize$ then
  return $n$;
if $n \geq MaxSize$ then
  $n \leftarrow MaxSize$;
while $i < (n/2)$ do
  $fp = (fp << 2) + Gear\_ls[\ src[2 * i]\ ]$;
  if ! ( $fp\ \&\ MaskA\_ls$ ) then
    return $2 * i$;
  $fp += Gear[\ src[2 * i + 1]\ ]$;
  if ! ( $fp\ \&\ MaskA$ ) then
    return $2 * i + 1$;
return $n$;
```

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on May 29,2020 at 23:55:24 UTC from IEEE Xplore. Restrictions apply.

TABLE 3
A Comparison Among the Rabin-Based CDC (RC), Gear-Based CDC (GC), and FastCDC (FC) Approaches in Terms of the Deduplication Ratio and the Average Size of Generated Chunks, as a Function of the Expected Chunk Size

Dataset	CDC	Expected Chunk Size of 4K (B)		Expected Chunk Size of 8K (B)		Expected Chunk Size of 16K (B)	
		Dedup Ratio	Avg. Chunk Size	Dedup Ratio	Avg. Chunk Size	Dedup Ratio	Avg. Chunk Size
TAR	RC	55.02%	5770	46.66%	12449	38.62%	25168
	GC	51.20% (–6.94%)	6786 (+17.6%)	43.55% (–6.67%)	14120 (+13.4%)	34.94% (–9.52%)	30919 (+22.9%)
	FC	54.39% (–1.14%)	5759 (–0.19%)	46.65% (-0.02%)	12334 (–0.92%)	38.68% (+1.55%)	25388 (+0.87%)
LNX	RC	96.65%	3847	96.30%	6021	95.94%	8261
	GC	96.72% (+0.07%)	3501 (–8.99%)	96.37% (+0.07%)	5684 (–5.59%)	96.01% (+0.07%)	8007 (–3.07%)
	FC	96.65% (–0.00%)	3860 (+0.33%)	96.31% (+0.01%)	6012 (–0.15%)	95.95% (+0.01%)	8246 (–0.18%)
WEB	RC	87.38%	5029	75.98%	11301	63.77%	23221
	GC	74.00% (–15.3%)	7264 (+44.4%)	57.37% (–24.5%)	19460 (+72.2%)	31.86% (–50.0%)	38888 (+67.5%)
	FC	90.02% (+3.02%)	5426 (+7.89%)	83.20% (+9.50%)	11552 (+2.22%)	72.92% (+14.3%)	23402 (+0.77%)
VMA	RC	41.63%	6535	36.70%	13071	31.38%	26191
	GC	41.11% (–1.24%)	5894 (–9.80%)	35.88% (–2.23%)	12419 (–4.98%)	30.52% (–2.74%)	24960 (–4.70%)
	FC	41.61% (–0.04%)	6468 (–1.02%)	36.40% (–0.81%)	13150 (+0.60%)	31.19% (–0.60%)	26334 (+0.54%)
VMB	RC	96.40%	5958	96.12%	11937	95.75%	24100
	GC	96.41% (+0.01%)	5622 (–5.63%)	96.05% (–0.07%)	11477 (–3.85%)	95.66% (–0.09%)	23260 (–3.48%)
	FC	96.39% (–0.01%)	6021 (+1.05%)	96.08% (+0.04%)	12138 (+1.68%)	95.70% (–0.05%)	24384 (+1.17%)
RDB	RC	95.35%	5473	92.57%	10964	87.38%	21946
	GC	95.15% (–0.20%)	5830 (+6.52%)	92.26% (–0.33%)	11666 (+6.40%)	86.82% (–0.64%)	23307 (+6.20%)
	FC	95.32% (–0.03%)	5479 (+0.10%)	92.58% (+0.01%)	10970 (+0.05%)	87.39% (+0.01%)	21909 (–0.16%)
SYN	RC	98.27%	5828	97.36%	11663	96.03%	23349
	GC	98.45% (+0.18%)	4922 (–15.5%)	97.65% (+0.29%)	9818 (–15.8%)	96.44% (+0.41%)	19624 (–16.0%)
	FC	98.28% (+0.01%)	5799 (–0.49%)	97.37% (+0.01%)	11598 (–0.55%)	96.04% (+0.01%)	23247 (–0.43%)

This technique is simple but very useful for FastCDC since it reduces one shift operation when chunking each two bytes compared with the traditional approach (rolling one byte each time as shown in Algorithm 3.2) while ensuring exactly the same chunking results. Note that it requires to lookup one more table and increases additional computation operations of '$2*i$' and '$2*i+1$', but these overheads are minor and FastCDC using this technique is about 30-40 percent faster than rolling one byte each time according to our evaluation results discussed in the next section.

4 Performance Evaluation

4.1 Experimental Setup

Experimental Platform. To evaluate FastCDC, we implement a prototype of the data deduplication system on the Ubuntu 18.04.1 operating system running on an Intel Xeon(R) Gold 6,130 processor at 2.1 GHz, with a 128 GB RAM. To better evaluate the chunking speed, another Intel i7-8700 processor at 3.2 GHz is also used for comparison.

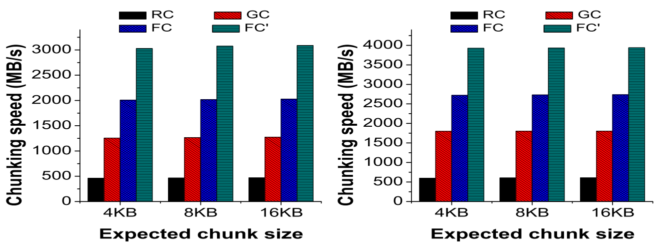
Configurations for CDC and Deduplication. Three CDC approaches, Rabin-, Gear-, and AE-based CDC, are used as the baselines for evaluating FastCDC. Rabin-based CDC is implemented based on the open-source project LBFS [6] (also used in many published studies [7], [19] or project [51]), where the sliding window size is configured to be 48 bytes. The Gear- and AE-based CDC schemes are implemented according to the algorithms described in their papers [13], [24], and we obtain performance results similar to and consistent with those reported in these papers. Here all the CDC approaches are configured with the maximum and minimum chunk sizes of $8\times$ and $\frac{1}{4}\times$ of the expected chunk size, the same as configured in LBFS [6]. The deduplication prototype consists of approximately 3,000 lines of C code, which is compiled by GCC 7.4.0 with the "-O3" compiler option to to maximize the speed of the resulting executable.

Performance Metrics of Interest. Chunking speed is measured by the in-memory processing speed of the evaluated CDC approaches and obtained by the average speed of five runs. *Deduplication ratio* is measured in terms of the percentage of duplicates detected after CDC, i.e., $\frac{The\ size\ of\ duplicate\ data\ detected}{Total\ data\ size\ before\ deduplication}$. *Average chunk size* after CDC is $\frac{Total\ data\ size}{Number\ of\ chunks}$, which reflects the metadata overhead for deduplication indexing.

Evaluated Datasets. Seven datasets with a total size of about 6 TB are used for evaluation as shown in Table 2. These datasets consist of the various typical workloads of deduplication, including the source code files, virtual machine images, database snapshots, etc., whose deduplication ratios vary from 40 to 98 percent, which will be detailed in Table 3 in the next subsection.

4.2 A Study of Optimizing Hash Judgment

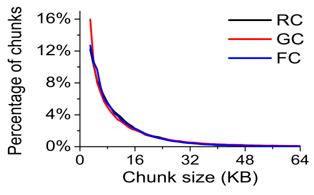
This subsection discusses an empirical study of FastCDC using techniques of the optimized hash judgment and 'rolling two bytes each time'. Fig. 10 shows the chunking speed of the four CDC approaches running on the RDB dataset, as a function of the expected chunk size and all using the minimum chunk size of $\frac{1}{4}\times$ of that for cut-point skipping. In general, the Rabin-based CDC has the lowest speed, and Gear-based CDC are about $3\times$ faster than Rabin. FastCDC using optimized hash judgement (i.e., FC′ in Fig. 10) is



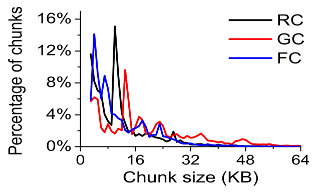
(a) Intel Gold 6130 (b) Intel i7-8700

Fig. 10. Chunking speed, as a function of the expected chunk size, of Rabin-based CDC (RC), Gear-based CDC (GC), FastCDC using optimized hash judgement (FC) and rolling two bytes each time (FC′) on two CPU processors.

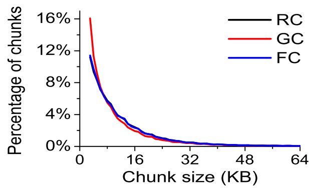
Authorized licensed use limited to: University of Texas at Arlington. Downloaded on May 29,2020 at 23:55:24 UTC from IEEE Xplore. Restrictions apply.



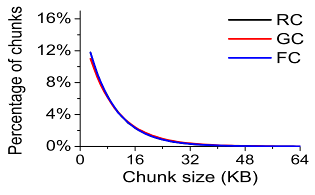
(a) Dataset TAR



(b) Dataset WEB

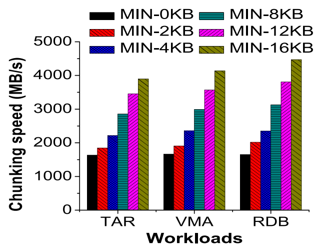


(c) Dataset VMA

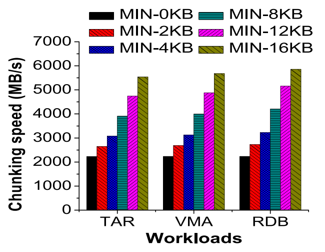


(d) Dataset RDB

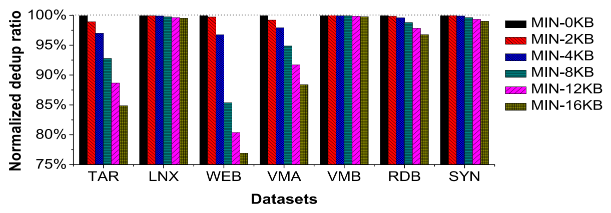
Fig. 11. Chunk-size distribution of the RC, GC, and FC approaches on the four typical datasets.



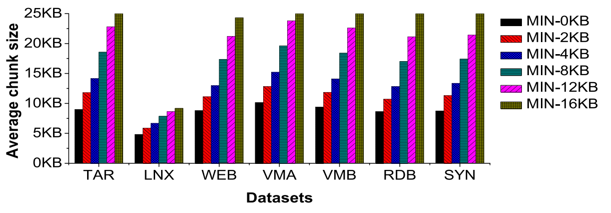
(a) Speed on intel Gold 6130



(b) Speed on intel i7-8700



(c) Deduplication ratio



(d) Average chunk size

Fig. 12. Chunking performance of FastCDC with the expected chunk size of 8KB but different minimum chunk sizes on two different CPU processors.

about 5× faster than Rabin and 1.5× faster than Gear regardless of the speed of the CPU processor and the expected chunk size. The high chunking speed of FastCDC stems from its simplification of the hash judgment after the fast Gear hash is used for chunking as described in Section 3.3. Meanwhile, FastCDC using 'rolling two bytes each time' (i.e., FC′ in Fig. 10) further increases the chunking speed by 40-50 percent since it further reduces calculation operation during CDC. Note that FC′ achieves exactly the same chunking results as FC, thus we do not discuss metrics of deduplication ratio and generated chunk size for FC′ in the remainder of this paper.

Table 3 shows the deduplication ratio and the average size of generated chunks (post-chunking) achieved by the three CDC approaches. We compare the Gear-based CDC (GC), and FastCDC (FC) approaches against the classic Rabin-based CDC (i.e., the baseline: RC) and record the percentage differences (in parentheses).

In general, FastCDC achieves nearly the same deduplication ratio as Rabin regardless of the expected chunk size and workload, and the difference between them is tiny as shown in the 3rd, 5th, 7th columns in Table 3 except on the WEB dataset. On the other hand, the Gear-based CDC has a much lower deduplication ratio on the datasets TAR and WEB due to its limited sliding window size as discussed in Section 3.2.

For the metric of the average size of generated chunks, the difference between the Rabin-based CDC and FastCDC is smaller than ±1.0 percent on most of the datasets. For the datasets WEB, FastCDC has 7.89 percent larger average chunk size than Rabin-based CDC, which is acceptable since the larger average chunk size means fewer chunks and fingerprints for indexing in a deduplication system (without sacrificing deduplication ratio) [3]. But for the Gear-based CDC, the average chunk size differs significantly in some datasets while its deduplication ratio is still a bit lower than other CDC approaches due to its smaller sliding window size.

We also compare the chunk-size distributions of the three tested chunking approaches in Fig. 11: FastCDC has nearly the same chunk-size distribution as Rabin on datasets TAR, VMA, and RDB, which generally follows the exponential distribution as discussed in Section 3.4. Note that the results in Fig. 11b are very different from others. This is because there are many zero bytes in this dataset according to our observation, which makes the chunking fingerprints not so random (thus not follow the the exponential distribution). However, comparing with Rabin and Gear, FastCDC's chunk-size distribution on WEB is the most similar to other datasets, which explains why FastCDC achieves the highest deduplication ratio on WEB among the three tested chunking approaches (see Table 3).

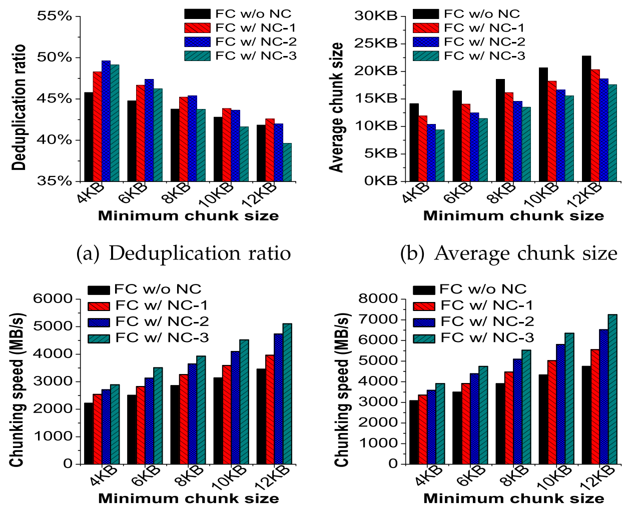
In summary, FastCDC with the optimized hash judgment achieves a chunking speed that is 5× higher than Rabin-based CDC while satisfactorily solving the problems of low deduplication ratio and smaller sliding window size faced by Gear-based CDC.

4.3 Evaluation of Cut-Point Skipping

This subsection discusses the evaluation results of cut-point skipping technique. Figs. 12a and 12b show the impact of applying different minimum chunk sizes on the chunking speed of FastCDC. Since the chunking speed is not so sensitive to the workloads, we only show the three typical workloads in Fig. 12. In general, cut-point skipping greatly accelerates the CDC process since the skipped region will not be hash-processed by CDC. The speedup of the FastCDC applying the minimum chunk sizes of 4 and 2 KB over the FastCDC without the constraint of the minimum chunk size (i.e., Min-0 KB) is about 1.25× and 1.50× respectively, which is almost consistent with the equation $1+\frac{\textit{the minimum chunk size}}{\textit{the expected chunk size}}$ as discussed in Section 3.4.

Figs. 12c and 12d show the impact of applying different minimum chunk sizes on the deduplication ratio and average

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on May 29,2020 at 23:55:24 UTC from IEEE Xplore. Restrictions apply.



(a) Deduplication ratio (b) Average chunk size

(c) Speed on intel Gold 6130 (d) Speed on intel i7-8700

Fig. 13. Evaluation of comprehensive performance of normalized chunking with different normalization levels.

generated chunk size of FastCDC. In general, deduplication ratio declines with the increase of the minimum chunk size applied in FastCDC, but not proportionally. For the metric of the average generated chunk size in FastCDC, it is approximately equal to the summation of the expected chunk size and the applied minimum chunk size. This means that the MIN-4 KB solution has the average chunk size of *8+4=12 KB*, leading to fewer chunks for fingerprints indexing in deduplication systems. Note that the increased portion of the average generated chunk size is not always equal to the size of the applied minimum chunk size, because the Rabin hashes of contents may not strictly follow the uniform distribution (as described in Equation (3) in Section 3.4) on some datasets.In addition, the average chunk sizes of dataset LNX are smaller than the minimum chunk size, which results from the many very small files whose sizes are much smaller than the minimum chunk size in LNX.

In summary, the results shown in Fig. 12 suggest that cut-point skipping helps obtain higher chunking speed and increase the average chunk size but at the cost of decreased deduplication ratio. The decreased deduplication ratio will be addressed by normalized chunking as evaluated in the next two subsections.

4.4 Evaluation of Normalized Chunking

In this subsection, we conduct a sensitivity study of normalized chunking (NC) on the TAR dataset, as shown in Fig. 13. Here the expected chunk size of FastCDC without NC is 8 KB and the normalized chunk size of FastCDC with NC is configured as the 4 KB + minimum chunk size. The normalization levels 1, 2, 3 refer to the three pairs of numbers of effective mask bits (14, 12), (15, 11), (16, 10) respectively that normalized chunking applies when the chunking position is smaller or larger than the normalized (or expected) chunk size, as discussed in Section 3.5.

Figs. 13a and 13b suggest that normalized chunking (NC) detects more duplicates when the minimum chunk size is about 4, 6, and 8 KB but slightly reduces the average generated chunk size, in comparison with FastCDC without NC. This is because NC reduces the number of large-sized chunks as shown in Fig. 9 and discussed in Section 3.5. The results also suggest that NC touches the "sweet spot" of deduplication ratio at the normalization level of 2 when the minimum chunk size is 4, 6, or 8 KB. This is because the very high normalization levels tend to have a similar chunk-size distribution to the Fixed-Size Chunking as shown in Fig. 9 in Section 3.5, which fails to address the *boundary-shift* problem and thus detects fewer duplicates. Figs. 13c and 13d suggest that NC, when combined with the approach of enlarging the minimum chunk size for cut-point skipping, greatly increases the chunking speed on the two tested processors.

In general, considering the three metrics of chunking speed, average generated chunk size, and deduplication ratio as a whole, as shown in Fig. 13, NC-2 with MinSize of 8 KB maximizes the chunking speed without sacrificing the deduplication ratio. Note that NC-2 with MinSize of 6 KB achieves the highest deduplication ratio among those NC approaches whose average chunk size $\geq$ that of Rabin and FastCDC tested in Table 3 (with the minimum chunk size of 2 KB).

4.5 Comprehensive Evaluation of FastCDC

In this subsection, we comprehensively evaluate the performance of FastCDC with the combined capability of the five key techniques: Gear-based rolling hash, optimizing hash judgment, cut-point skipping, rolling two bytes each time, and normalized chunking using "NC-2" and minimum chunk size of 6 KB/8 KB as suggested by the last subsection. Finally, twelve CDC approaches are tested for evaluation:

- RC-v1 (or RC-MIN-2 KB) is Rabin-based CDC used in LBFS [6]; RC-v2 and RC-v3 refer to Rabin-based CDC using normalized chunking with a minimum chunk size of 4 and 6 KB respectively.
- FC-v1 is FastCDC uses the techniques of optimizing hash judgment and cut-point skipping with a minimum chunk size of 2 KB; FC-v2 and FC-v3 refer to FastCDC using all the four techniques with a minimum chunk size of 6 and 8 KB, respectively.
- FC′-v1, FC′-v2, and FC′-v3 are FastCDC using the technique of rolling two bytes each time on top of FC-v1, FC-v2, and FC-v3 respectively.
- AE-v1 and AE-v2 refer to AE-based CDC [24] and its optimized version [52];
- Fixed-Size Chunking (FIXC) is also tested for comparison using the average chunk size of 10 KB (to better understand content-defined chunking).

Evaluation results in Table 4 suggest that FC-v1, FC-v2, AE-v2, and RC-v2 achieves nearly the same deduplication ratio as RC-v1 in most cases, which suggests that the normalized chunking scheme works well on both Rabin and FastCDC. Note that FIXC works well on the datasets LNX and VMB, because LNX has many files smaller than the fixed-size chunk of 10 KB (and thus the average generated chunk size is also smaller than 10 KB) and VMB has many structured backup data (and thus VMB is suitable for FIXC).

Table 5 shows that RC-v1, RC-v2, AE-v1, AE-v2, FC-v1, and FC-v2 generate similar average chunk size. But the approaches of RC-v3 and FC-v3 has a much larger average chunk size, which means that it generates fewer chunks and thus less metadata for deduplication processing. Meanwhile,

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on May 29,2020 at 23:55:24 UTC from IEEE Xplore. Restrictions apply.

TABLE 4
Comparison of Deduplication Ratio Achieved by the Nine Chunking Approaches

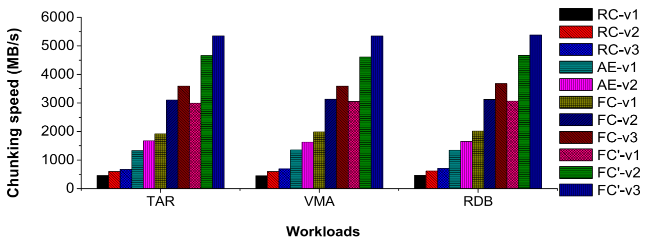
Dataset	FIXC	RC-v1	RC-v2	RC-v3	AE-v1	AE-v2	FC-v1	FC-v2	FC-v3
TAR	15.77%	46.66%	47.42%	45.37%	43.62%	46.41%	46.65%	47.39%	45.40%
LNX	95.68%	96.30%	96.28%	96.19%	96.25%	96.13%	96.31%	96.28%	96.19%
WEB	59.96%	75.98%	83.16%	80.39%	83.08%	83.18%	83.20%	83.29%	80.92%
VMA	17.63%	36.70%	37.79%	36.52%	38.10%	38.17%	36.40%	37.66%	36.39%
VMB	95.68%	96.12%	96.17%	96.11%	95.82%	96.15%	96.08%	96.17%	96.11%
RDB	16.39%	92.57%	92.96%	92.24%	88.82%	92.83%	92.58%	92.97%	92.23%
SYN	79.46%	97.36%	97.91%	97.67%	97.54%	97.86%	97.37%	97.90%	97.67%

TABLE 5
Average Chunk Size Generated by the Nine Chunking Approaches on the Seven Datasets

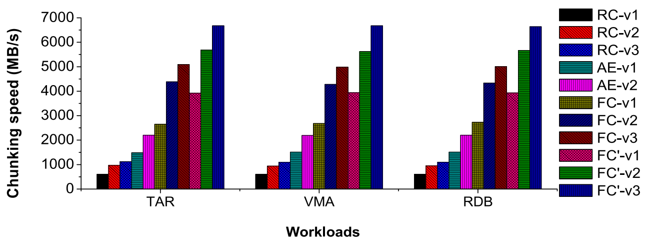
Dataset	FIXC	RC-v1	RC-v2	RC-v3	AE-v1	AE-v2	FC-v1	FC-v2	FC-v3
TAR	10239	12449	12664	14772	12187	12200	12334	12801	14918
LNX	6508	6021	7041	7636	6274	6162	6012	7042	7636
WEB	10240	11301	12174	14148	11977	11439	11552	11880	13951
VMA	10239	13071	13505	15628	13098	13559	13150	13595	15746
VMB	10239	11937	12970	15094	12303	12254	12138	13034	15166
RDB	10239	10964	12587	14728	11943	12102	10970	12583	14725
SYN	10240	11663	12221	14271	11956	11997	11598	12239	14289

RC-v3 and FC-v3 still achieves a comparable deduplication ratio, slightly lower than RC-v1 as shown in Table 4, while providing a much higher chunking speed as discussed later.

Fig. 14 suggests that FC′-v3 has the highest chunking speed, about 12× faster than the Rabin-based approach, about 2.5× faster than FC-v1. This is because FC′-v3 is the final FastCDC using all the five techniques to speed up the CDC process. In addition, FC′-v2 is also a good CDC candidate since it has a comparable deduplication ratio while also working well on the other two metrics of chunking speed and average generated chunk size. Meanwhile, normalized chunking also helps accelerate Rabin-based CDC (i.e., RC-v2 and RC-v3) while achieving comparable deduplication ratio and average chunk size. But this acceleration is limited since the main bottleneck for Rabin-based CDC is still the rolling hashing computation.



(a) Speed on intel Gold 6130



(b) Speed on intel i7-8700

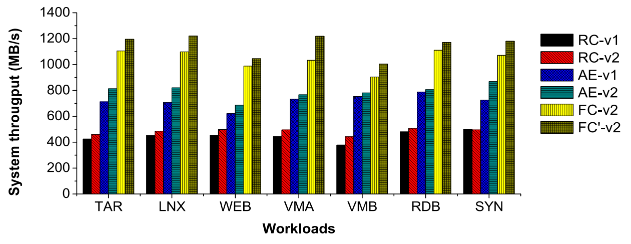
Fig. 14. Chunking speed of the 11 CDC approaches.

Table 6 further studies the CPU overhead among the eight CDC approaches. The CPU overhead is averaged on 1,000 test runs by the Linux tool "Perf". The results suggest that FC′-v3 has the fewest instructions for CDC computation, the higher instructions per cycle (IPC), and thus the least CPU time overhead, i.e., CPU cycles. Generally, FastCDC greatly reduces the number of instructions for CDC computation by using the techniques of Gear-based hashing, optimizing hash judgment, and rolling two bytes each time (i.e., FC′-v1), and then minimizes the number of computation instructions by enlarging the minimum chunk size for cut-point skipping and combining normalized chunking (i.e., FC′-v2 and FC′-v3). In addition, FastCDC increases the IPC for the CDC computation by well pipelining the instructions of hashing and hash-judging tasks in up-to-date processors. Therefore, these results clearly reveal the reason why FastCDC is much faster than Rabin- and AE-based CDC is that the former not only reduces the number of instructions and branches, but also increases the IPC for the CDC process.

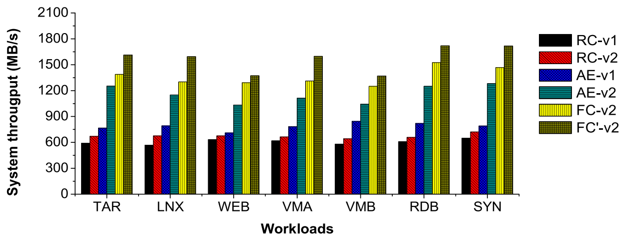
TABLE 6
Number of Instructions, Instructions Per Cycle (IPC), and CPU Cycles Required to Chunk Data *Per Byte* by the 11 CDC Approaches on the Intel i7-8770 Processor

Approaches	Instructions	IPC	CPU cycles	branches
RC-v1	19.54	2.49	7.85	2.44
RC-v2	11.22	2.30	4.88	1.02
RC-v3	9.72	2.27	4.28	0.88
AE-v1	11.75	3.77	3.12	3.84
AE-v2	7.00	3.08	2.27	2.00
FC-v1	7.32	3.89	1.88	1.63
FC-v2	4.89	3.83	1.28	1.02
FC-v3	4.23	3.72	1.14	0.88
FC′-v1	5.28	3.87	1.36	1.13
FC′-v2	3.57	3.59	0.99	0.76
FC′-v3	3.09	3.47	0.89	0.66

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on May 29,2020 at 23:55:24 UTC from IEEE Xplore. Restrictions apply.



(a) Throughputs on intel Gold 6130



(b) Throughputs on intel i7-8700

Fig. 15. System throughputs of Destor running with the six CDC approaches on the two CPUs.

In summary, as shown in Tables 4, 5, 6 and Fig. 14, FastCDC (i.e., FC′-v2 recommended) significantly speeds up the *Content-Defined Chunking* process and achieves a *comparable and even higher deduplication ratio* with the similar average chunk size by using a combination of the five key techniques proposed in Section 3.

4.6 Impact of CDC on Overall System Throughput

To understand the impact of the different CDC algorithms on the overall throughput of data deduplication system, we implemented them in the open-source Destor deduplication system [19]. In this evaluation, we use a Ramdisk-driven emulation to avoid the performance bottleneck caused by disk I/O. And for each dataset, we only use 5 GB, a small part of its total size in the evaluation. In addition, to examine the maximum impact of different CDC algorithms on the system throughput, we configure Destor with: (1) using the fast intel ISA-L library for SHA1 computation [53] (SHA1 speed would be about 3-4 GB/s on our tested CPUs); (2) indexing all fingerprints in RAM; (3) pipelining the deduplication subtasks (i.e., chunking, fingerprinting, indexing, etc.).

Fig. 15 shows that FastCDC (i.e., FC′-v2) helps achieve about 1.2-3.0X higher overall system throughout than RC-v1, RC-v2, AE-v1, and AE-v2, while achieving a comparable or even higher deduplication ratio as shown in Table 4. This is because when Destor pipelines the deduplication subtasks and the CDC becomes the bottleneck of the system, acceleration of the CDC can directly benefit the overall system throughput before the system meets another performance bottleneck.

5 Conclusion

In this paper, we propose FastCDC, a much faster CDC approach for data deduplication than the state-of-the-art CDC approaches while achieving a comparable deduplication ratio. The main idea behind FastCDC is the combined use of five key techniques, namely, Gear-based fast rolling hashing, optimizing the hash judgment for chunking, subminimum chunk cut-point skipping, normalized chunking, and rolling two bytes each time. Our experimental evaluation demonstrates that FastCDC obtains a chunking speed that is about 3-12× higher than that of the state-of-the-art CDC approaches while achieving nearly the same deduplication ratio as the classic Rabin-based CDC. In addition, our study of overall system throughput shows that Destor [19] using FastCDC helps achieve about 1.2-3.0X higher overall system throughout than using other CDC approaches.

FastCDC has been adopted as the default chunker in several Github projects (for quickly detecting duplicate contents), such as Rdedup [20], Content Blockchain [21], etc. We have also released the FastCDC source code at https://github.com/Borelset/destor/tree/master/src/chunkingto be shared with the deduplication and storage systems research community.

Acknowledgments

The authors are grateful to the anonymous reviewers for their insightful comments and constructive suggestions. This research was partly supported by NSFC No. 61972441, No. 61872110, No. 61872414, No. 61772212, No. 61821003, No. 61772222, and No. 61832007, the major project of National Science and Technology of China No. 2017ZX01032-101, the Open Project Program of Wuhan National Laboratory for Optoelectronics No. 2018WNLOKF008, the Shenzhen Science and Technology Program under Grant No. JCYJ20190806143405318, Key R&D Program for Guangdong Province under Grant No.2019B010136001, and the US National Science Foundation under Grant CCF-1704504 and CCF-1629625. The preliminary manuscript appeared in the proceedings of USENIX ATC 2016. In this journal version, we included more techniques on FastCDC and additional measurement results from our analysis and testbed experiments.

References

[1] D. Meyer and W. Bolosky, “A study of practical deduplication,” in *Proc. 9th USENIX Conf. File Storage Technol.*, 2011, Art. no. 1.
[2] A. El-Shimi *et al.*, “Primary data deduplication–large scale study and system design,” in *Proc. USENIX Conf. Annu. Tech. Conf.*, 2012, Art. no. 26.
[3] G. Wallace *et al.*., “Characteristics of backup workloads in production systems,” in *Proc. 10th USENIX Conf. File Storage Technol.*, 2012, Art. no. 4.
[4] P. Shilane *et al.*, “WAN optimized replication of backup datasets using stream-informed delta compression,” in *Proc. FAST.*
[5] S. Quinlan and S. Dorward, “Venti: A new approach to archival storage,” in *Proc. 1st USENIX Conf. File Storage Technol.*, 2002, pp. 7–es.
[6] A. Muthitacharoen, B. Chen, and D. Mazieres, “A low-bandwidth network file system,” in *Proc. 18th ACM Symp. Operating Syst. Princ.*, 2001.
[7] D. Meister, J. Kaiser, A. Brinkmann, T. Cortes, M. Kuhn, and J. Kunkel, “A study on data deduplication in HPC storage systems,” in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2012, pp. 1–11.
[8] C. Policroniades and I. Pratt, “Alternatives for detecting redundancy in storage systems data,” in *Proc. Annu. Conf. USENIX Annu. Tech. Conf.*, 2004, Art. no. 6.
[9] M. O. Rabin, *Fingerprinting by random polynomials*. Center for Research in Computing Techn., Aiken Computation Lab. Univ., 1981.
[10] A. Broder, “Some applications of Rabin’s fingerprinting method,” in *Sequences II: Methods in Communications, Security, and Computer Science*. Berlin, Germany: Springer, 1993, pp. 1–10.
[11] C. Dubnicki, E. Kruus, K. Lichota, and C. Ungureanu, “Methods and systems for data management using multiple selection criteria,” U.S. Patent App. 11/566,122, Dec. 1, 2006.

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on May 29,2020 at 23:55:24 UTC from IEEE Xplore. Restrictions apply.

[12] B. Aggarwal *et al.*, "EndRE: An end-system redundancy elimination service for enterprises," in *Proc. 7th USENIX Conf. Netw. Syst. Des. Implementation*, 2010, Art. no. 28.

[13] W. Xia *et al.*,"Ddelta: A deduplication-inspired fast delta compression approach," *Perform. Eval.*, vol. 79, pp. 258–272, 2014.

[14] W. Xia *et al.*, "P-Dedupe: Exploiting parallelism in data deduplication system," in *Proc. IEEE 7th Int. Conf. Netw. Archit. Storage*, 2012, pp. 338–347.

[15] M. D. Lillibridge, "Parallel processing of input data to locate landmarks for chunks," U.S. Patent 8 001 273, Aug. 16, 2011.

[16] S. Al-Kiswany *et al.*, "StoreGPU: Exploiting graphics processing units to accelerate distributed storage systems," in *Proc. 17th Int. Symp. High Perform. Distrib. Comput.*, 2008, pp. 165–174.

[17] A. Gharaibeh *et al.*, "A GPU accelerated storage system," in *Proc. 19th ACM Int. Symp. High Perform. Distrib. Comput.*, 2010, pp. 167–178.

[18] P. Bhatotia, R. Rodrigues, and A. Verma, "Shredder: GPU-accelerated incremental storage and computation," in *Proc. 10th USENIX Conf. File Storage Technol.*, 2012, Art. no. 14.

[19] M. Fu *et al.*, "Design tradeoffs for data deduplication performance in backup workloads," in *Proc. 13th USENIX Conf. File Storage Technol.*, 2015, pp. 331–344.

[20] Rdedup Project. [Online]. Available: https://github.com/dpc/rdedup

[21] Content Blockchain Project. [Online]. Available: https://github.com/coblo

[22] B. Zhu, K. Li, and R. H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system," in *Proc. 6th USENIX Conf. File Storage Technol.*, 2008, Art. no. 18.

[23] W. Xia *et al.*, "A comprehensive study of the past, present, and future of data deduplication,"

[24] Y. Zhang *et al.*, "AE: An asymmetric extremum content defined chunking algorithm for fast and bandwidth-efficient data deduplication," in *Proc. IEEE Conf. Comput. Commun.*, 2015, pp. 1337–1345.

[25] Y. Cui *et al.*, "QuickSync: Improving synchronization efficiency for mobile cloud storage services," in *Proc. 21st Annu. Int. Conf. Mobile Comput. Netw.*, 2015, pp. 592–603.

[26] C. Yu, C. Zhang, Y. Mao, and F. Li, "Leap-based content defined chunking—Theory and implementation," in *Proc. 31st Symp. Mass Storage Syst. Technol.*, 2015, pp. 1–12.

[27] F. Ni and S. Jiang, "RapidCDC: Leveraging duplicate locality to accelerate chunking in CDC-based deduplication systems," in *Proc. ACM Symp. Cloud Comput.*, 2019, pp. 220–232.

[28] F. Ni, X. Lin, and S. Jiang, "SS-CDC: A two-stage parallel content-defined chunking for deduplicating backup storage," in *Proc. 12th ACM Int. Conf. Syst. Storage*, 2019, pp. 86–96.

[29] K. Eshghi and H. K. Tang, "A framework for analyzing and improving content-based chunking algorithms," Hewlett Packard Laboratories, Palo Alto, CA, USA, *Tech. Rep. HPL-2005–30(R.1)*, 2005.

[30] D. Teodosiu, N. Bjorner, Y. Gurevich, M. Manasse, and J. Porkka, "Optimizing file replication over limited bandwidth networks using remote differential compression," Microsoft Research TR-2006–157, 2006.

[31] A. Anand *et al.*, "Redundancy in network traffic: Findings and implications," in *Proc. 11th Int. Joint Conf. Meas. Model. Comput. Syst.*, 2009, pp. 37–48.

[32] N. Bjørner, A. Blass, and Y. Gurevich, "Content-dependent chunking for differential compression, the local maximum approach," *J. Comput. Syst. Sci.*, vol. 76, pp. 154–203, 2010.

[33] E. Kruus, C. Ungureanu, and C. Dubnicki, "Bimodal content defined chunking for backup streams," in *Proc. 8th USENIX Conf. File Storage Technol.*, 2010, Art. no. 18.

[34] B. Romański *et al.*, "Anchor-driven subchunk deduplication," in *Proc. 4th Annu. Int. Conf. Syst. Storage*, 2011, Art. no. 16.

[35] G. Lu, Y. Jin, and D. H. Du, "Frequency based chunking for data de-duplication," in *Proc. IEEE Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst.*, 2010, pp. 287–296.

[36] W. Xia *et al.*, "SiLo: A similarity-locality based near-exact deduplication scheme with low RAM overhead and high throughput," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, 2011, pp. 26–28.

[37] M. Lillibridge *et al.*, "Sparse indexing: Large scale, inline deduplication using sampling and locality," in *Proc. 7th Conf. File Storage Technol.*, 2009, pp. 111–123.

[38] W. Xia *et al.*, "Accelerating data deduplication by exploiting pipelining and parallelism with multicore or manycore processors," in *Proc. 10th USENIX Conf. File Storage Technol.*, 2012, pp. 1–2.

[39] F. Guo and P. Efstathopoulos, "Building a high-performance deduplication system," in *Proc. USENIX Annu. Tech. Conf.*, 2011, Art. no. 25.

[40] W. Xia *et al.*,"Accelerating content-defined-chunking based data deduplication by exploiting parallelism," *Future Gener. Comput. Syst.*, vol. 98, pp. 406–418, 2019.

[41] W. Xia, H. Jiang, D. Feng, and Y. Hua, "Similarity and locality based indexing for high performance data deduplication," *IEEE Trans. Comput.*, vol. 64, no. 4, pp. 1162–1176, Apr. 2015.

[42] B. Debnath, S. Sengupta, and J. Li, "ChunkStash: Speeding up inline storage deduplication using flash memory," in *Proc. USENIX Annu. Tech. Conf.*, 2010, Art. no. 16.

[43] M. Lillibridge, K. Eshghi, and D. Bhagwat, "Improving restore speed for backup systems that use inline chunk-based deduplication," in *Proc. 11th USENIX Conf. File Storage Technol.*, 2013, pp. 183–198.

[44] D. Bhagwat, K. Pollack, D. D. Long, T. Schwarz, E. L. Miller, and J. Paris, "Providing high reliability in a minimum redundancy archival storage system," in *Proc. 14th IEEE Int. Symp. Model. Anal. Simul.*, 2006, pp. 413–421.

[45] Y. Zhou *et al.*, "SecDep: A user-aware efficient fine-grained secure deduplication scheme with multi-level key management," in *Proc. 31st Symp. Mass Storage Syst. Technol.*, 2015, pp. 1–14.

[46] J. Min, D. Yoon, and Y. Won, "Efficient deduplication techniques for modern backup operation," *IEEE Trans. Comput.*, vol. 60, no. 6, pp. 824–840, Jun. 2011.

[47] GNU archives. [Online]. Available: http://ftp.gnu.org/gnu/

[48] Linux archives. [Online]. Available: ftp://ftp.kernel.org/

[49] VMs archives. [Online]. Available: http://www.thoughtpolice.co.uk

[50] V. Tarasov *et al.*, "Generating realistic datasets for deduplication analysis," in *Proc. USENIX Annu. Tech. Conf.*, 2012, Art. no. 24.

[51] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proc. Int. Conf. Parallel Archit. Compilation Techn.*, 2008, pp. 72–81.

[52] Y. Zhang *et al.*,"A fast asymmetric extremum content defined chunking algorithm for data deduplication in backup storage systems," *IEEE Trans. Comput.*, vol. 66, no. 2, pp. 199–211, Feb. 2017.

[53] Intel ISA-L: Intelligent Storage Acceleration Library. [Online]. Available: https://github.com/intel/isa-l

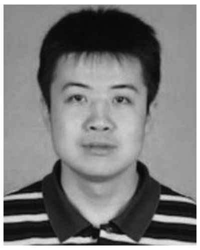
Wen Xia (Member, IEEE) received the PhD degree in computer science from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 2014. He is currently an associate professor with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen. His research interests include data reduction, storage systems, cloud storage, etc. He has published more than 40 papers in major journals and conferences including the *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *Proceedings of the IEEE*, USENIX ATC, FAST, HotStorage, MSST, DCC, IPDPS, etc.

Xiangyu Zou (Student Member, IEEE) is currently working toward the PhD degree majoring in computer science at the Harbin Institute of Technology, Shenzhen, China. His research interests include data deduplication, storage systems, etc. He has published several papers in major journals and international conferences including the *Future Generation Computing Systems*, MSST, and HPCC.

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on May 29,2020 at 23:55:24 UTC from IEEE Xplore. Restrictions apply.

Hong Jiang (Fellow, IEEE) received the BSc degree in computer engineering from the Huazhong University of Science and Technology, Wuhan, China, in 1982, the MASc degree in computer engineering from the University of Toronto, Toronto, Canada, in 1987, and the PhD degree in computer science from the Texas A&M University, College Station, Texas, in 1991. He is currently a chair and Wendell H. Nedderman endowed professor of Computer Science and Engineering Department, University of Texas at Arlington. Prior to joining UTA, he served as a program director with National Science Foundation (2013.1–2015.8) and he was with the University of Nebraska- Lincoln since 1991, where he was Willa Cather professor of computer science and engineering. His present research interests include computer architecture, computer storage systems and parallel I/O, high performance computing, big data computing, cloud computing, performance evaluation. He recently served as the associate editor of the *IEEE Transactions on Parallel and Distributed Systems* and the *IEEE Transactions on Computers*. He has more than 200 publications in major journals and international Conferences in these areas, including the *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, ISCA, MICRO, USENIX ATC, FAST, EUROSYS, SC, etc.

Yukun Zhou is currently working toward the PhD degree majoring in computer science at the Harbin Institute of Technology, Shenzhen, China. His research interests include data reduction, distributed systems, etc. He has published several papers in major journals and international conferences including the *Proceedings of the IEEE*, *Future Generation Computing Systems*, *Performance Evaluation*, USENIX ATC, MSST, IPDPS, INFOCOM, etc.



Chuanyi Liu received the PhD degrees in computer science and technology from Tsinghua University, Beijing, China, in 2010. He is currently an associate professor with the Harbin Institute of Technology, Shenzhen. His research interests include the mass storage systems, cloud computing and cloud security, and data security. He has published more than 30 papers in major journals and international conferences including the *IEEE Access*, CCS, ICDCS, ICS, etc.

Dan Feng (Member, IEEE) received the BE, ME, and PhD degrees in computer science and technology from the Huazhong University of Science and Technology (HUST), China, in 1991, 1994, and 1997, respectively. She is currently a professor and the dean of the School of Computer Science and Technology, HUST. Her research interests include computer architecture, massive storage systems, and parallel file systems. She has more than 100 publications in major journals and conferences, including the *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *ACM Transactions on Storage*, FAST, USENIX ATC, SC, etc.

Yu Hua (Senior Member, IEEE) received the BE and PhD degrees in computer science from the Wuhan University, Wuhan, China, in 2001 and 2005, respectively. He is currently a professor with the Huazhong University of Science and Technology, China. His research interests include file systems, cloud storage systems, non-volatile memory, etc. He has more than 100 papers to his credit in major journals and international conferences including *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Computers*, OSDI, MICRO, FAST, USENIX ATC, SC, etc. He serves for multiple international conferences, including ASPLOS, SOSP, FAST, USENIX ATC, ICS, RTSS, SoCC, ICDCS, INFOCOM, IPDPS, DAC, MSST, and DATE. He is a distinguished member of CCF and a senior member of ACM.

Yuchong Hu received the BE and PhD degrees in computer science from the University of Science and Technology of China, Hefei, China, in 2005 and 2010, respectively. He is currently an associate professor with the Huazhong University of Science and Technology. His research interests include network coding/erasure coding, cloud computing, and network storage. He has more than 30 publications in major journals and conferences, including the *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *IEEE Transactions on Information Theory*, *ACM Transactions on Storage* , FAST, INFOCOM, DSN, etc.

Yucheng Zhang received the PhD degree in computer science from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 2018. He is currently an assistant professor with the Hubei University of Technology, Wuhan, China. His research interests include data deduplication, storage systems, etc. He has several papers in refereed journals and conferences including *IEEE Transactions on Computers*, *Future Generation Computing Systems*, *Proceedings of the IEEE*, USENIX ATC, FAST, IPDPS, INFOCOM, MSST, etc.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/csdl.

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on May 29,2020 at 23:55:24 UTC from IEEE Xplore. Restrictions apply.