FastCDC: a Fast and Efficient Content-Defined Chunking Approach for Data Deduplication

Wen Xia, *Huazhong University of Science and Technology and Sangfor Technologies Co., Ltd.;* Yukun Zhou, *Huazhong University of Science and Technology;* Hong Jiang, *University of Texas at Arlington;* Dan Feng, Yu Hua, Yuchong Hu, Yucheng Zhang, and Qing Liu, *Huazhong University of Science and Technology*

https://www.usenix.org/conference/atc16/technical-sessions/presentation/xia

This paper is included in the Proceedings of the
2016 USENIX Annual Technical Conference (USENIC ATC '16).

June 22–24, 2016 • Denver, CO, USA

978-1-931971-30-0

Open access to the Proceedings of the
2016 USENIX Annual Technical Conference
(USENIX ATC '16) is sponsored by USENIX.

FastCDC: a Fast and Efficient Content-Defined Chunking Approach for Data Deduplication

Wen Xia[†, ‡], Yukun Zhou[†], Hong Jiang[§], Dan Feng[¶, †, *], Yu Hua[¶, †], Yuchong Hu[†], Yucheng Zhang[†], Qing Liu[†]

[†] School of Computer, Huazhong University of Science and Technology [‡] Sangfor Technologies Co., Ltd.

[§] University of Texas at Arlington [¶] WNLO, Huazhong University of Science and Technology

Abstract

Content-Defined Chunking (CDC) has been playing a key role in data deduplication systems in the past 15 years or so 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 paper, we propose FastCDC, a Fast and efficient CDC approach, that builds and improves on the latest Gear-based CDC approach, one of the fastest CDC methods to our knowledge. The key idea behind FastCDC is the combined use of three key techniques, namely, simplifying and enhancing the hash judgment to address our observed challenges facing Gear-based CDC, skipping sub-minimum chunk cut-point to further speed up CDC, and 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. Our evaluation results show that, by using a combination of the three techniques, FastCDC is about 10× faster than the best of open-source Rabin-based CDC, and about 3× faster than the state-of-the-art Gear- and AE-based CDC, while achieving nearly the same deduplication ratio as the classic Rabin-based approach.

1 Introduction

去重系统是什么

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 [12, 23] and EMC [30, 33], about 50% and 85% of the data in their production primary and secondary storage systems, respectively, are redundant and could be removed by the deduplication technology.

通过加密安全哈希签名标识重复内容

细粒度

chunk-level 更受欢迎，方法，CDC的基本优势

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) [27]. Content-Defined Chunking (CDC) based approaches are proposed to address the *boundary-shift* problem facing the FSC approach [25]. 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 [12, 22, 23, 26], CDC-based deduplication approaches are able to detect about 10-20% more redundancy than the FSC approach.

主流方法及各种优化

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 [8, 11, 25, 28]. Rabin-based CDC is highly effective in duplicate detection but time-consuming, because it computes and judges (against a condition value) the Rabin fingerprints of the data stream byte by byte [11]. A recent study, called QuickSync [9], suggests that CDC is computationally expensive for deduplication based synchronization in mobile cloud storage. In order to speed up the CDC process, other hash algorithms have been proposed to replace the Rabin algorithm for CDC, such as SampeByte [1], Gear [38], and AE [40]. Meanwhile, the abundance of computation resources afforded by multicore and manycore processors [20, 37] or GPU processors [2, 5, 15] has been leveraged for acceleration.

1. 生成fp
2. 比较fp

对chunking的原理介绍来提出问题

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 [38], suggests that the Gear hash (i.e., $fp = (fp \ll 1) + G(b)$, see Section 3) is very efficient as a rolling hash for CDC. To the best of our knowledge, Gear appears to be one of the fastest rolling hash algorithms for CDC at present. However, according to our first observation from empirical and analytical studies, the Gear-based CDC has the potential problem of low *deduplication ratio* (i.e., the percentage of re-

[*] Corresponding author: dfeng@hust.edu.cn.

由于滑动窗口的size太小，所以gear hash的去重率有可能很低

dundant data reduced) stemming from its *hash judgment* stage where the sliding window size is very small. Meanwhile, our second observation indicates that the *hash judgment* stage becomes the new performance bottleneck during CDC after the fast Gear [38] is used in the *hashing* stage, because the accelerated hashing stage by Gear, has shifted the bottleneck to the hash judgment stage. Motivated by these two observations and the need to further accelerate the CDC process, we use an approach of enhancing and simplifying the hash judgment to further reduce the CPU operations during CDC.

Our third observation suggests that the predefined minimum chunk size used to avoid generating the very small-sized chunks (e.g., LBFS [25] employs the minimum chunk size of 2KB for Rabin-based CDC) can be employed for *cut-point skipping* during CDC, i.e., judiciously skipping some identified cut-points to eliminate the CDC operations in this region. Enlarging this minimum chunk size can further speed up the chunking process but at the cost of decreasing the deduplication ratio. *This is because many chunks with skipped cut-points are not divided truly according to the data contents (i.e., content-defined).* Thus, we propose 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.

Therefore, motivated by the above observations, we proposed FastCDC, a Fast and efficient CDC approach that combines the following three key techniques.

- **Simplified but enhanced hash judgment:** By padding several zero bits into the mask value for the hash-judging statement of the CDC algorithm to enlarge the sliding window size while using the fast Gear hash, FastCDC is able to achieve nearly the same deduplication ratio as the Rabin-based CDC; By further simplifying and optimizing the hash-judging statement, FastCDC minimizes the CPU overhead for the hash judgment stage in CDC.
- **Sub-minimum chunk cut-point skipping:** Our large scale study suggests that skipping the predefined minimum chunk size (used for avoiding small-sized chunks) increases the chunking speed but decreases the deduplication ratio (about 15% decline in the worst case). This motivates us to further enlarge the minimum chunk size to maximize chunking speed while developing a counter measure for the decreased deduplication ratio in the following normalized chunking approach.
- **Normalized chunking:** By selectively changing the number of mask bits '1' in the hash-judging statement of CDC, FastCDC normalizes the chunk-size distribution to a small specified region (e.g., 8KB∼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. First, it increases the deduplication ratio by reducing the number of large-sized chunks. Second, it reduces the number of small-sized chunks, which makes it possible to combine with the cut-point skipping technique above to maximize the CDC speed while without sacrificing the deduplication ratio.

Our evaluation results from a large-scale empirical study of CDC, based on seven datasets, demonstrate that FastCDC is about 10× faster than the Rabin-based CDC, and 3× faster than the state-of-the-art Gear- and AE-based CDC, while ensuring a high deduplication ratio.

The rest of the paper is organized as follows. Section 2 presents the necessary background for this research. Section 3 discusses our key observations that motivate the design of FastCDC. Section 4 describes the three key approaches used in FastCDC. Section 5 presents and discusses our experimental evaluation of FastCDC. Section 6 draws conclusions and outlines our future work.

2 Background

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) [27] is simple and fast but may face the problem of low deduplication ratio that stems from the *boundary-shift* problem [25, 40]. 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.

chunking是什么、旧方法是什么

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 [25, 28]) of the window. A chunk cut-point is declared if the hash value satisfies some pre-defined condition. As shown in Figure 1, 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 [33, 42] and primary [12, 23] storage systems.

CDC是什么、有哪些主流CDC

Although the widely used Rabin-based CDC helps obtain a high deduplication ratio, it incurs heavy CPU overhead [2, 5, 9, 37]. Specifically, in Rabin-based CDC, the Rabin hash for a sliding window containing the byte sequence B_1,B_2,...,B_α is defined as a polynomial A(p):

主流CDC的原理简单介绍

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

D is the average chunk size and α is the num ber of bytes in the sliding window.

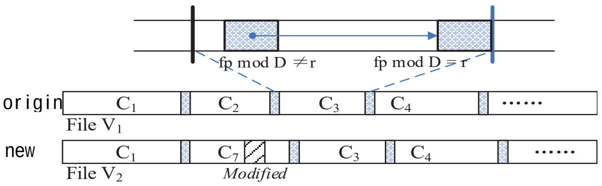


Figure 1: The sliding window technique for the CDC algorithm. The hash value of the sliding window, *fp*, is computed via the Rabin algorithm (this is the *hashing stage* of CDC). If the lowest $log_2 D$ bits of the hash value matches a threshold value r, i.e., *fp* mod $D = r$, this offset (i.e., the current position) is marked as a chunk cut-point (this is the *hash-judging stage* of CDC).

rolling hash就是能以迭代方式进行计算的hash算法。这也跟滑动窗口的思想很吻合

where D is the average chunk size and α is the number of bytes in the sliding window. 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 follows:

$$Rabin(B_2, B_3, ..., B_{\alpha+1}) = \{[Rabin(B_1, ..., B_\alpha) - B_1 P^{\alpha-1}]p + B_{\alpha+1}\} modS \tag{2}$$

主流的缺陷

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 to accelerating 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.

加速方法：
1. 基于算法加速
2. 基于硬件加速

优化方式

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 [1, 38, 40]. SampleByte [1] 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 [38] 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 [40] 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. [39] 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.

？？？？

？？？？？

Hardware-oriented CDC Optimizations. StoreGPU [2, 15] and Shredder [5] make full use of GPGPU's computational power to accelerate popular compute-intensive primitives (i.e., chunking and fingerprinting) in data deduplication. P-Dedupe [37] pipelines deduplication tasks and then further parallelizes the sub-tasks of chunking and fingerprinting with multiple threads and thus achieves higher throughput.

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 [13] and Regression chunking [12] 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 [3, 7, 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 [17], Subchunk [29], and FBC [21] re-chunk the non-duplicate chunks into smaller ones to detect more redundancy.

它这里提到一点很值得注意，通过在hashing judgment阶段引入一个或更多的额外的threshold，就更有可能获得cut-point，并且减少chunks的大小差异？？？感觉这个有可能跟我们之后的normalized方法有关联。

补充说明

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 [14, 19, 34, 42]. 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 [16, 36, 37]. 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 [10, 14, 35, 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 [38], defragmentation [18], reliability [4], security [41], etc. In this paper, we focus on designing a very fast and efficient chunking approach for data deduplication.

这个就是那个经典rewrite

3 Observation and Motivation

In this section, we elaborate on and analyze the most relevant state-of-the-art CDC approaches to gain useful insights and observations. Table 1 shows a comparison among the three rolling hash algorithms for CDC, namely, Rabin, Adler, and Gear, which suggests Gear uses far fewer calculation operations than Rabin and Adler, thus being a good rolling hash candidate for CDC.

下面要开始详细分析CDC现状了，很好

A good hash function must have a uniform distribution of hash values regardless of the hashed content. As shown in Figure 2, 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

好的哈希方法生成的哈希值最好均匀分布

256是因为一个字节为2^8
也就是说它是把一个字节映射为了一个64位整数

Name	Pseudocode	Speed
Rabin	$fp = ((fp\hat{\ }U(a)) << 8) \| b\hat{\ }T[fp >> N]$	Slow
Adler	$S_1 += A(b); S_2 += S_1; fp = (S_2 << 16) \| S_1$	Slow
Gear	$fp = (fp << 1) + G(b)$	Fast

Table 1: The hashing stage of the Rabin-, Adler-, and Gear-based CDC. 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', 'A', 'G' denote the predefined arrays [11, 25, 38]. '*fp*' represents the fingerprint of the sliding window.

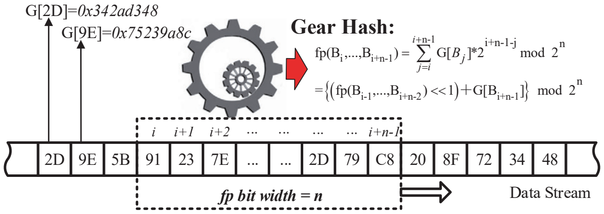

G[2D]=0x342ad348
G[9E]=0x75239a8c
Gear Hash:
$fp(B_i,...,B_{i+n-1}) = \sum_{j=i}^{i+n-1} G[B_j]*2^{i+n-1-j} \bmod 2^n$
$= \{(fp(B_{i-1},...,B_{i+n-2}) << 1) + G[B_{i+n-1}]\} \bmod 2^n$
i i+1 i+2 ... ... i+n-1
2D 9E 5B 91 23 7E 2D 79 C8 20 8F 72 34 48
fp bit width = n
Data Stream

Figure 2: A schematic diagram of the Gear hash.

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 Figure 2). 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. Note that the modulo operation is used in the hashing-judging stage as discussed later.

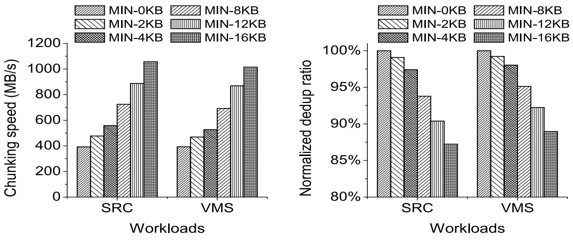
并且每次＋操作加入一个新字节，左移剥离一个旧字节【以形象化角度来看，相当于每次剥离那些用不到的信息

留下那些新旧仍然用得到的信息】

Gear-based CDC is first employed by Ddelta [38] for delta compression, which helps provide a higher delta encoding speed. However, according to our experimental analysis, there are still challenges facing the Gear-based CDC. We elaborate on these issues as follows.

Limited sliding window size. The traditional hash judgment for the Rabin-based CDC, as shown in Figure 1 (i.e., "*fp mod D==r*"), is also used by the Gear-based CDC [38]. 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 Figure 5, the sliding window size of the Gear-based CDC will be equal to the number of the bits used by the mask value. Therefore, when using a mask value of 2^{13} for the expected chunk size of 8KB, the sliding window for the Gear-based CDC would be 13 bytes while that of the Rabin-based CDC would be 48 bytes [25]. 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 (see Section 5.2).

The time-consuming hash judgment. Our implementation and in-depth analysis of the Gear-based CDC suggest that *its hash-judging stage accounts for more than 60% of its CPU overhead during CDC after the fast Gear hash is used for fingerprinting*. Thus, there is a lot of room for the optimization of the hash judging stage to further accelerate the CDC process.



(a) Chunking speed (b) Deduplication ratio

Figure 3: Rabin-based CDC performance as a function of the minimum chunk size used for cut-points skipping before chunking. Here we use the average chunk size of 8KB, Intel i7-4770 processor, and the best open-source Rabin algorithm we have access to for the speed test.

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. Figure 3 shows our experimental observation of Rabin-based CDC with two typical workloads of deduplication whose workload characteristics are detailed in Table 2 in Section 5.1. Figure 3 (a) indicates that setting the minimum chunk size for cut-point skipping at $\frac{1}{4}\times\sim2\times$ of the expected chunk size can effectively accelerate the CDC process. But this approach decreases the deduplication ratio by about 2∼15% (see Figure 3 (b)) since many chunks are not divided truly according to the data contents, i.e., not really content-defined.

The observation suggested in Figure 3 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 4.4.

In summary, the analysis and observation of the Gear-based CDC motivate us to propose FastCDC, a faster CDC approach with a higher deduplication ratio than the Gear-based CDC. The implementation of FastCDC will be detailed in the next section and its effectiveness and efficiency will be demonstrated in Section 5.

4 FastCDC Design and Implementation

4.1 FastCDC Overview

FastCDC is implemented on top of the Gear-based CDC, and aims to provide high performance CDC. Generally,

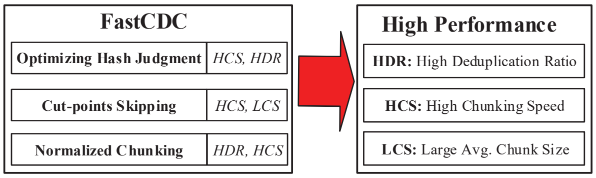
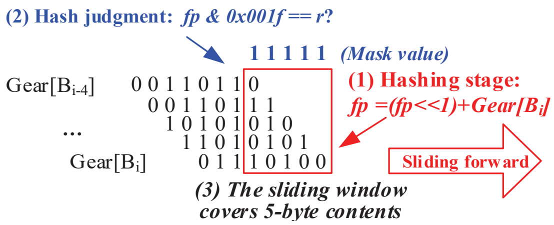


Figure 4: The three key techniques used in FastCDC and their corresponding benefits for high performance CDC.

there are three metrics for evaluating CDC performance, namely, deduplication ratio, chunking speed, and the average generated chunk size. Note that the average generated chunk size may be nearly equal to or larger than the predefined expected chunk size (e.g., 8KB) due to factors such as the detailed CDC methods and datasets. This is also an important CDC performance metric because 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. However, 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 tradeoff among these three metrics so as to strive for high performance CDC, by using a combination of the three techniques with their complementary features as shown in Figure 4.

- 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 4.2.
- 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 4.3.
- 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 4.4.

In general, the key idea behind FastCDC is the combined use of the above three key techniques on top of Gear-based CDC, 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.



说实话没太看懂。。。。

Figure 5: 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.

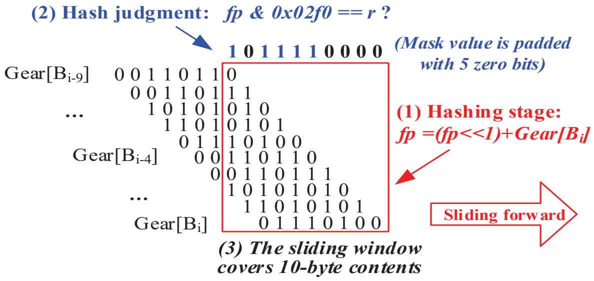


Figure 6: An example of the sliding window technique proposed for FastCDC. By padding y zero bits into the mask value for hash judgment, the size of the sliding window used in FastCDC is enlarged to about $5+y$ bytes, where y=5 in this example.

4.2 Optimizing Hash Judgment

In this subsection, we propose an enhanced but simplified hash-judging statement to accelerate the hash judgment stage of FastCDC to further accelerate the chunking process on top of the Gear-based CDC 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, 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 Figure 5) 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 Figure 6, 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, ... , B_{i+9}), instead of the original five bytes, to be involved in the final

由于改成了& 所以好像并不会影响正确性

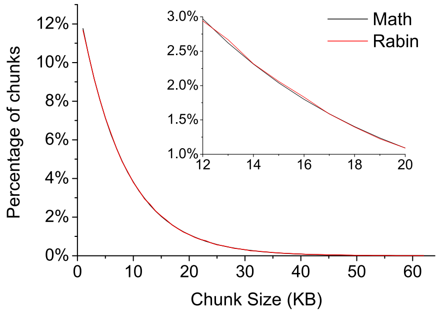


Figure 7: Chunk-size distribution of the Rabin-based CDC approach with average chunk size of 8KB and without the maximum and minimum chunk size requirements. "Rabin" and "Math" denote respectively our experimental observation and theoretical analysis (i.e., Equation (3)) of post-chunking chunk-size distribution, where they are shown to be nearly identical.

hash judgment by the five masked one bits (as the red box shown in Figure 6) and thus makes the sliding window size equal or similar to that of the Rabin-based CDC [25], 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*" [25, 38]. For example, the Rabin-based CDC usually defines D and r as *0x02000* and *0x78*, according to the known open source project LBFS [25], to obtain the expected average chunk size of 8KB. In FastCDC, when combined with the zero-padding scheme introduced above and shown in Figure 6, 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 5.2.

4.3 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 [17, 19, 23–25, 29]. A common configuration of the average, minimum, and maximum parameters follows that used by LBFS [25], i.e., 8KB, 2KB, and 64KB. Our experimental observation and mathematical analysis 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) follows an exponential distribution as follows: 这个曲线一点也不正太

$$P(X \leq x) = F(x) = (1 - e^{-\frac{x}{8192}}),\ x \geq 0. \quad (3)$$

rabin-hash 的chunk size分布

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 Figure 1 for CDC) follow a uniform distribution. Equation 3 suggests that the value of the expected chunk size will be 8KB according to the exponential distribution.

前提是 rabin-hash 均匀分布

Figure 7 shows a comparison between the actual chunk-size distribution of the real-world datasets after the Rabin-based CDC and the chunk-size distribution obtained by the mathematical analysis based on Equation 3, which indicates that the two are almost identical. According to Equation 3, the chunks smaller than 2KB and larger than 64KB would account for about 22.12% and 0.03% of the total number of chunks respectively. 这个还是太高了 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 Figure 3. 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. 确实 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 8KB + minimum chunk size, which will be verified in Section 5.3. The speedup achieved by skipping the sub-minimum chunk cut-point can be estimated by $1+\frac{\textit{the minimum chunk size}}{\textit{the expected chunk size}}$. The increased chunking speed comes from the eliminated computation on the skipped region, which will also be verified in Section 5.3.

4.4 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

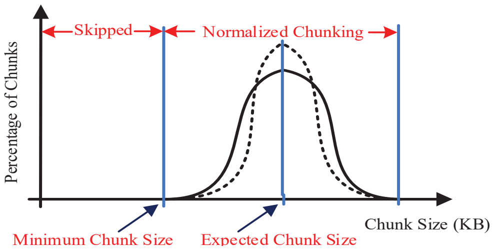


Figure 8: A conceptual diagram of the normalized chunking combined with the subminimum chunk cut-point skipping. The dotted line shows a higher level of normalized chunking.

skipping approach. As shown in Figure 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.

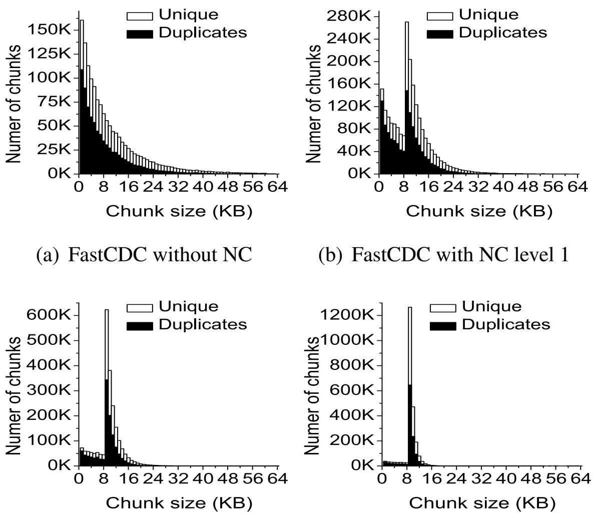
In our implementation of normalized chunking, we selectively change the number of effective mask bits (i.e., the number of ‘1’ bits) for the hash-judging statement. For the traditional CDC approach with expected chunk size of 8KB (i.e., 2^{13}), 13 effective mask bits are used for hash judgment (e.g., *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 8KB, which makes it harder to generate chunks of size smaller than 8KB. 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 8KB, which makes it easier to generate chunks of size larger than 8KB. 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 Figure 7).

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 Figure 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.



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

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

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

Figure 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 5.1). 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 8KB . Figure 9 suggests that the chunk-size distribution is a reasonably close approximation of the normal distribution centered on 8KB at the normalization level of 2 or 3.

As shown in Figure 9, there are only a very small number of chunks smaller than 2KB or 4KB after normalized chunking while FastCDC without NC has a large number of chunks smaller than 2KB or 4KB (consistent with the discussion in Section 4.3). 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 Figure 9 is not truly normal distribution but an approximation of it. Figures 9 (c) and (d) shows a closer approximation of normal distribution of chunk size achieved by using the normalization levels 2 and

Algorithm 1: FastCDC8KB

这个也就体现了它的那个标准化的三个level

```
Input: data buffer, $src$; buffer length, $n$
Output: chunking breakpoint $i$
$MaskS \leftarrow 0x0003590703530000LL$;        // 15 ‘1’ bits
$MaskA \leftarrow 0x0000d90303530000LL$;        // 13 ‘1’ bits
$MaskL \leftarrow 0x0000d90003530000LL$;        // 11 ‘1’ bits
$MinSize \leftarrow 2KB$;  $MaxSize \leftarrow 64KB$;
$fp \leftarrow 0$;  $i \leftarrow MinSize$;  $NormalSize \leftarrow 8KB$;
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 $i$;
```

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 5.

4.5 Putting It All Together

To put things together and in perspective. Algorithm 1 describes FastCDC combining the three key techniques: optimizing hash judgment, cut-point skipping, and normalized chunking (with the expected chunk size of 8KB). The 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 [38].

As shown in Algorithm 1, 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 [25]. 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 1 is 2*KB*, which can be enlarged to *4KB* 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.

Name	Size	Workload descriptions
TAR	19 GB	85 tarred files from the open source projects such as GCC, GDB, Emacs, etc.
LNX	105 GB	260 versions of Linux source code files.
WEB	36 GB	15 days’ snapshots of the website: *news.sina.com*, which are collected by crawling software *wget* with a maximum retrieval depth of 3.
VMA	117 GB	75 virtual machine images of different OS release versions, including CentOS, Fedora, Debian, etc.
VMB	1.9 TB	125 backups of an Ubuntu 12.04 virtual machine image in use by a research group.
RDB	1.1 TB	200 backups of the redis key-value store database.
SYN	1.4 TB	200 synthetic backups. The backup is simulated by the file create/delete/modify operations [31].

Table 2: Workload characteristics of the seven datasets used in the performance evaluation.

5 Performance Evaluation

5.1 Experimental Setup

Experimental Platform. To evaluate FastCDC, we implement a prototype of the data deduplication system on the Ubuntu 12.04.2 operating system running on a quad-core Intel i7-4770 processor at 3.4GHz, with a 16GB RAM. To better evaluate the chunking speed, another quad-core Intel i7-930 processor at 2.8GHz 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 [25] (also used in many published studies [14, 22] or project [6]), 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 [38, 40], 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× and $\frac{1}{4}\times$ of the expected chunk size, the same as configured in LBFS [25]. The deduplication prototype consists of approximately 3000 lines of C code, which is compiled by GCC 4.7.3 with the “-O3” compiler option.

Performance Metrics of Interest. ***Chunking speed***

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	54.81%	5561	47.58%	11873	41.23%	24067
	GC	51.68% (-5.71%)	6094 (+9.58%)	44.90% (–5.64%)	12651 (+6.55%)	38.05% (–7.71%)	28743 (+19.4%)
	FC	54.14% (-1.22%)	5722 (+2.90%)	47.64% (+0.13%)	12192 (+2.69%)	41.26% (+0.08%)	24462 (+1.64%)
LNX	RC	97.69%	3828	97.25%	5978	96.80%	8188
	GC	97.78% (+0.09%)	3473 (–9.27%)	97.33% (+0.09%)	5644 (-5.59%)	96.88% (+0.08%)	7932 (-3.13%)
	FC	97.69% (+0.00%)	3845 (+0.44%)	97.26% (+0.01%)	5969 (-0.15%)	96.82% (+0.01%)	8176 (-0.15%)
WEB	RC	96.50%	5011	95.09%	9985	93.59%	19154
	GC	95.68% (-0.85%)	7091 (+41.5%)	94.09% (-1.13%)	17069 (+70.9%)	92.92% (-0.72%)	24960 (+30.3%)
	FC	96.14% (-0.38%)	5330 (+6.37%)	94.02% (-1.43%)	10725 (+7.41%)	93.21% (-0.40%)	19740 (+3.06%)
VMA	RC	42.99%	6367	38.23%	12743	32.97%	25485
	GC	42.60% (-0.91%)	5798 (-8.94%)	37.57% (-1.73%)	12069 (–5.29%)	32.23% (-2.13%)	24177 (–5.37%)
	FC	42.97% (-0.04%)	6293 (-1.16%)	37.96% (-0.72%)	12787 (+0.35%)	32.79% (-0.55%)	25620 (+0.53%)
VMB	RC	96.41%	5958	96.13%	11937	95.76%	24100
	GC	96.41% (+0.00%)	5662 (–4.96%)	96.06% (-0.07%)	11477 (–3.86%)	95.66% (-0.10%)	23260 (–3.49%)
	FC	96.39% (–0.02%)	6021 (+1.05%)	96.09% (-0.04%)	12138 (+1.68%)	95.70% (-0.06%)	24384 (+0.01%)
RDB	RC	97.36%	5116	95.53%	10232	92.05%	20479
	GC	97.20% (-0.16%)	5463 (+6.78%)	95.21% (-0.33%)	10923 (+6.75%)	91.49% (-0.60%)	21820 (+6.55%)
	FC	97.35% (-0.02%)	5118 (+0.04%)	95.50% (-0.03%)	10238 (+0.06%)	92.01% (-0.02%)	20479 (+0.00%)
SYN	RC	95.64%	5479	93.64%	10954	90.72%	21927
	GC	96.03% (+0.41%)	5338 (-2.57%)	94.30% (+0.70%)	10675 (-2.55%)	91.43% (+0.68%)	21325 (-2.75%)
	FC	95.65% (+0.01%)	5473 (-0.11%)	93.67% (+0.03%)	10945 (-0.08%)	90.73% (+0.02%)	21924 (-0.01%)

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.

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., $\frac{\mathit{The\ size\ of\ duplicate\ data\ detected}}{\mathit{Total\ data\ size\ before\ deduplication}}$. ***Average chunk size*** is $\frac{\mathit{Total\ data\ size}}{\mathit{Number\ of\ chunks}}$ after CDC, which reflects the metadata overhead for deduplication indexing.

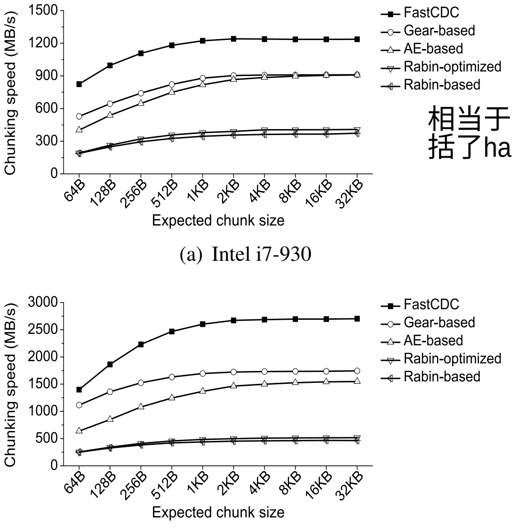
Evaluated Datasets. Seven datasets with a total size of about 5 TB are used for evaluation as shown in Table 2. These datasets consist of the various typical workloads of deduplication, including the source code files, virtual machine images, database snapshots, etc., whose deduplication ratios vary from 40% to 97%.

对于第一个优化点hash judge，我们比较的是chunking的速度和去重比率

5.2 A Study of Optimizing Hash Judgment

This subsection discusses an empirical study of FastCDC using the optimized hash judgment. Figure 10 shows the chunking speed of the five 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. The Rabin-optimized approach employs the technique of simplifying the hash judgment proposed in Section 4.2 but only achieves a little acceleration, this is because the hashing stage is the main bottleneck for Rabin-based CDC. In general, the Rabin-based CDC has the lowest speed, and the AE- and Gear-based CDC are about 3× faster than Rabin. For the AE-based CDC, its chunking speed is similar to that of the Gear-based CDC when the expected chunk size ranges from 2∼16 KB but is much lower than that of Gear when the expected chunk size is smaller than 1 KB. FastCDC is about 5× faster than Rabin and 1.5× faster than Gear and AE 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 4.2.

由于rabin算法的瓶颈是hash阶段而非hash-judge阶段，所以优化作用不明显



(a) Intel i7-930

(b) Intel i7-4770

Figure 10: Chunking speed, as a function of the expected chunk size, of Rabin-, Rabin (optimized)- Gear-, and AE-based CDC, and FastCDC on two CPU processors.

相当于是说这个统计是包括了hash和hash-judge

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) and record the percentage differences (in parentheses). AE-based CDC has nearly the same deduplication ratio as Rabin, thus is not shown in this table due to space limit.

In general, FastCDC achieves nearly the same deduplication ratio as the Rabin-based CDC regardless of the expected chunk size and workload, and the difference between them is only about ±0.1∼1.4% as shown in the 3rd, 5th, 7th columns in Table 3. On the other hand, the Gear-based CDC has a much lower deduplication ratio on the datasets TAR, WEB, and VMA due to its limited sliding window size as discussed in Section 3.

For the metric of the average size of generated chunks, the difference between the Rabin-based CDC and FastCDC is smaller than ±0.1% on most of the datasets. For the datasets TAR and WEB, FastCDC has 1∼7% 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) [33]. 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.

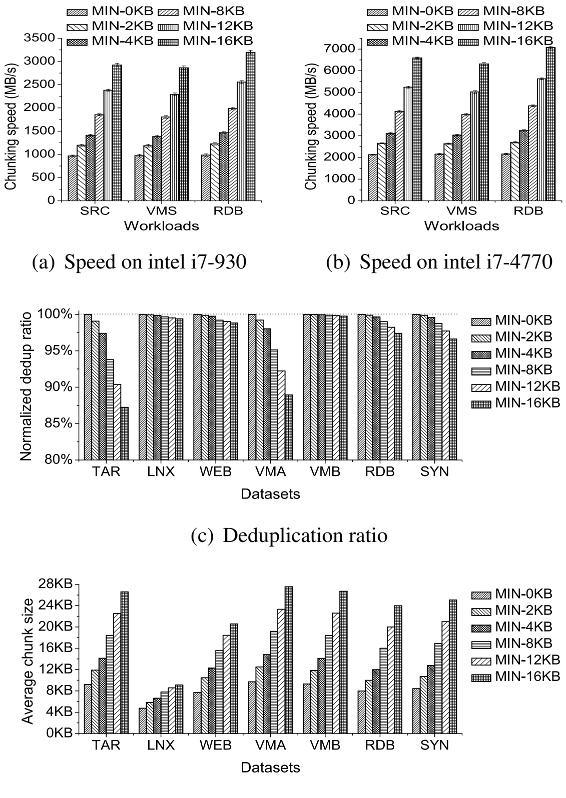
In summary, FastCDC achieves a chunking speed that is 5× higher than the Rabin-based CDC while satisfactorily solving the problem of low deduplication ratio facing the Gear-based CDC, as shown in Figure 10 and Table 3.

5.3 Evaluation of Cut-point Skipping

This subsection discusses the evaluation results of cut-point skipping technique. Figures 11 (a) and (b) 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 Figure 11. 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 4KB and 2KB over the FastCDC without the constraint of the minimum chunk size (i.e., Min-0KB) is about 1.25× and 1.50× respectively, which is consistent with the equation $1+\frac{\textit{the minimum chunk size}}{\textit{the expected chunk size}}$ as discussed in Section 4.3.

Figures 11 (c) and (d) show the impact of applying different minimum chunk sizes on the deduplication ratio and average 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-4KB 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 4.3) on some datasets.

注意到也不跟其它几个对比了耶



(a) Speed on intel i7-930

(b) Speed on intel i7-4770

(c) Deduplication ratio

(d) Average chunk size

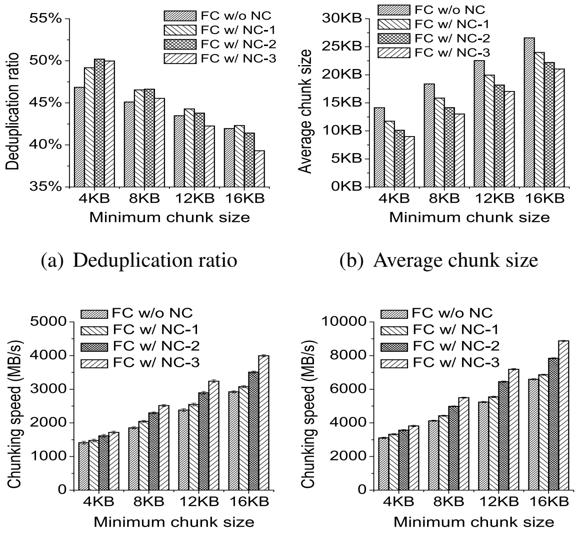
Figure 11: Chunking performance of FastCDC with the expected chunk size of 8KB but different minimum chunk sizes on two different CPU processors.

In summary, the results shown in Figure 11 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.

5.4 Evaluation of Normalized Chunking

In this subsection, we conduct a sensitivity study of normalized chunking (NC) on the TAR dataset, as shown in Figure 12. Here the expected chunk size of FastCDC without NC is 8KB and the normalized chunk size of FastCDC with NC is configured as the 4KB + 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 4.4.

Figures 12 (a) and (b) suggest that normalized chunk-



(a) Deduplication ratio

(b) Average chunk size

(c) Speed on intel i7-930

(d) Speed on intel i7-4770

Figure 12: Evaluation of comprehensive performance of normalized chunking with different normalization levels.

ing (NC) detects more duplicates when the minimum chunk size is about 4KB and 8KB 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 Figure 9 and discussed in Section 4.4. 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 4KB or 8KB. This is because the very high normalization levels tend to have a similar chunk-size distribution to the Fixed-Size Chunking as shown in Figure 9 in Section 4.4, which fails to address the *boundary-shift* problem and thus detects fewer duplicates. Figures 12 (c) and (d) 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 addition, the average chunk sizes of datasets WEB and 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 the two datasets.

In general, considering the three metrics of chunking speed, average generated chunk size, and deduplication ratio as a whole, as shown in Figure 12, NC-2 with MinSize of 8KB maximizes the chunking speed without sacrificing the deduplication ratio. NC-2 with MinSize of 4KB achieves the highest deduplication ratio but with only a small acceleration of the chunking speed .

5.5 Comprehensive Evaluation of FastCDC

In this subsection, we comprehensively evaluate the performance of FastCDC with the combined capability of the three key techniques: optimizing hash judgment, cut-point skipping, and normalized chunking (using NC-2 as suggested by the last subsection). Four approaches are tested for evaluation: RC with Min2KB (or RC-MIN-2KB) is Rabin-based CDC used in LBFS [25]; FC with Min2KB (or FC-MIN-2KB) uses the techniques of optimizing hash judgment and cut-point skipping with a minimum chunk size of 2KB; FC-NC with Min4KB and FC-NC with Min8KB refer to FastCDC using all the three techniques with a minimum chunk size of 4KB and 8KB, respectively. To better evaluate the deduplication ratio, Fixed-Size Chunking (XC) is also tested using the average chunk size of 10KB.

Dataset	RC w/ Min2KB	FC w/ Min2KB	FC-NC w/ Min4KB	FC-NC w/ Min8KB	XC w/ 10KB
TAR	47.58%	47.64%	50.19%	47.18 %	12.21%
LNX	97.25%	97.26%	97.35%	97.10%	96.51%
WEB	95.09%	94.02%	95.47%	94.44%	93.19%
VMA	38.23%	37.96%	40.31%	38.15%	18.26%
VMB	96.13%	96.09%	96.24%	96.11%	95.68%
RDB	95.53%	95.50%	96.71%	95.70%	9.80%
SYN	93.64%	93.67%	94.09%	92.62%	75.06%

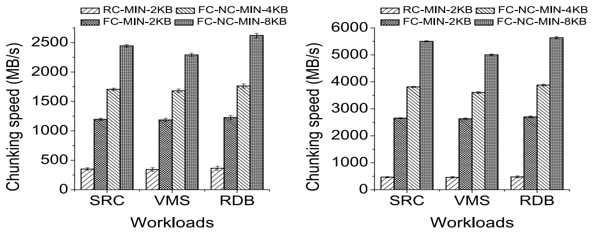
Table 4: Comparison of deduplication ratio achieved by the five chunking approaches. Note that "FC"and "FC-NC" refer to the full FastCDC without and with normalized chunking respectively, in this subsection.

Dataset	RC w/ Min2KB	FC w/ Min2KB	FC-NC w/ Min4KB	FC-NC w/ Min8KB	XC w/ 10KB
TAR	11873	12192	10347	14076	10240
LNX	5978	5969	6288	7585	6477
WEB	9985	10725	9327	12862	9513
VMA	12743	12787	11161	15031	10239
VMB	11937	12138	10850	15148	10239
RDB	10232	10238	9751	13846	10240
SYN	10954	10945	10318	14123	10240

Table 5: Average chunk size generated by the five chunking approaches on the seven datasets.

Evaluation results in Table 4 suggest that FC with Min2KB achieves nearly the same deduplication ratio as Rabin-based approach. FC-NC with Min4KB achieves the highest deduplication ratio among the five approaches while Fixed-Size Chunking (XC) has the lowest deduplication ratio. Note that XC works well on the LNX, WEB, VMB datasets, because LNX and WEB datasets have many files smaller than the fixed-size chunk of 10KB (and thus the average generated chunk size also smaller than 10KB) and VMB has many structured backup data (and thus VMB is suitable for XC).

Table 5 shows that RC and FC with Min2KB and XC generate similar average chunk size while FC-NC with Min4KB has a slightly small average chunk size. But the approach of FC-NC with Min8KB has a much smaller average chunk size, which means that it generates fewer chunks and thus less metadata for deduplication processing. Meanwhile, FC-NC with Min8KB still achieves a comparable deduplication ratio, slightly lower than RC as shown in Table 4, while providing a much higher



(a) Speed on intel i7-930 (b) Speed on intel i7-4770

Figure 13: Chunking speed of the four CDC approaches.

Approaches	Instructions	IPC	CPU cycles
RC-MIN-2KB	38, 829, 037	2.35	16, 537, 973
FC-MIN-2KB	15, 074, 950	4.37	3, 452, 146
FC-NC-MIN-4KB	11, 008, 372	4.82	2, 284, 453
FC-NC-MIN-8KB	7, 750, 124	4.82	1, 608, 033

Table 6: Number of instructions, instructions per cycle (IPC), and CPU cycles required to chunk 1MB data by the four CDC approaches on the Intel i7-4770 processor.

chunking speed as discussed later.

Figure 13 suggests that FC-NC with Min8KB has the highest chunking speed, about 10× faster than the Rabin-based approach, about 2× faster than FC with Min2KB. This is because FC-NC with Min8KB is the final FastCDC using all the three techniques to speed up the CDC process. In addition, FC-NC with Min4KB is also a good CDC candidate since it has the highest deduplication ratio while also working well on the other two metrics of chunking speed and the average generated chunk size. Note that XC is not shown here because it has almost no computation overhead for chunking.

Table 6 further studies the CPU overhead among the four CDC approaches. The CPU overhead is averaged on 1000 test runs by the Linux tool "Perf". The results suggest that FC-NC-MIN-8KB has the fewest instructions for CDC computation, the highest IPC (instructions per cycle), 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 and optimizing hash judgment (i.e., "FC-MIN-2KB"), 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-NC-MIN-8KB"). 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 explain why FastCDC is about 10× faster than Rabin-based CDC is that the former not only reduces the number of instructions, but also increases the IPC for the CDC process.

In summary, as shown in Tables 4, 5, 6 and Figure 13, FastCDC (i.e., FC-NC-MIN-8KB) significantly speeds up the chunking process and achieves a comparable deduplication ratio while reducing the number of generated chunks by using a combination of the three key techniques proposed in Section 4.

6 Conclusion and Future Work

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 three key techniques, namely, optimizing the hash judgment for chunking, subminimum chunk cut-point skipping, and normalized chunking. Our experimental evaluation demonstrates that FastCDC obtains a chunking speed that is about 10× higher than that of the Rabin-based CDC and about 3× that of the Gear- and AE-based CDC while achieving nearly the same deduplication ratio as the Rabin-based CDC.

In our future work, we plan to incorporate FastCDC in some other deduplication systems that are sensitive to the CPU overhead of content-defined chunking, such as QuickSync [9], to further explore the potentials and benefits of FastCDC. We also plan to release the FastCDC source code to be shared with the deduplication and storage systems research community.

Acknowledgments

We are grateful to our shepherd Scott Rixner and the anonymous reviewers for their insightful comments and feedback. The work was partly supported by NSFC No. 61502190, 61502191, 61232004, and 61402061; 863 Project 2013AA013203; State Key Laboratory of Computer Architecture, No. CARCH201505; Fundamental Research Funds for the Central Universities, HUST, under Grant No. 2015MS073; US NSF under Grants CNS-1116606 and CNS-1016609, Key Laboratory of Information Storage System, Ministry of Education, China, and Sangfor Technologies Co., Ltd.

References

[1] AGGARWAL, B., AKELLA, A., ANAND, A., ET AL. EndRE: an end-system redundancy elimination service for enterprises. In *Proceedings of the 7th USENIX conference on Networked Systems Design and Implementation (NSDI'10)* (San Jose, CA, USA, April 2010), USENIX Association, pp. 14–28.

[2] AL-KISWANY, S., GHARAIBEH, A., SANTOS-NETO, E., ET AL. StoreGPU: exploiting graphics processing units to accelerate distributed storage systems. In *Proceedings of the 17th international symposium on High Performance Distributed Computing (HPDC'08)* (Boston, MA, USA, June 2008), ACM Association, pp. 165–174.

[3] ANAND, A., MUTHUKRISHNAN, C., AKELLA, A., AND RAMJEE, R. Redundancy in network traffic: findings and

implications. In *Proceedings of the 11th international Joint Conference on Measurement and Modeling of Computer Systems (SIGMETRICS-Performance 2009)* (Seattle, WA, USA, June 2009), ACM Association, pp. 37–48.

[4] BHAGWAT, D., POLLACK, K., LONG, D. D., SCHWARZ, T., MILLER, E. L., AND PÂRIS, J.-F. Providing high reliability in a minimum redundancy archival storage system. In *Proceedings of The 14th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'06)* (Monterey, CA, USA, September 2006), IEEE Computer Society Press, pp. 413–421.

[5] BHATOTIA, P., RODRIGUES, R., AND VERMA, A. Shredder: GPU-accelerated incremental storage and computation. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST'12)* (San Jose, CA, USA, February 2012), USENIX Association, pp. 1–15.

[6] BIENIA, C., KUMAR, S., SINGH, J. P., AND LI, K. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques (PACT'08)* (Toronto, Canada, October 2008), ACM, pp. 72–81.

[7] BJØRNER, N., BLASS, A., AND GUREVICH, Y. Content-dependent chunking for differential compression, the local maximum approach. *Journal of Computer and System Sciences 76*, 3 (2010), 154–203.

[8] BRODER, A. Some applications of Rabin's fingerprinting method. *Sequences II: Methods in Communications, Security, and Computer Science* (1993), 1–10.

[9] CUI, Y., LAI, Z., WANG, X., DAI, N., AND MIAO, C. QuickSync: Improving Synchronization Efficiency for Mobile Cloud Storage Services. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking* (Paris, France, Sept. 2015), ACM, pp. 592–603.

[10] DEBNATH, B., SENGUPTA, S., AND LI, J. ChunkStash: speeding up inline storage deduplication using flash memory. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference (USENIX'10)* (Boston, MA, USA, June 2010), USENIX Association, pp. 1–14.

[11] DUBNICKI, C., KRUUS, E., LICHOTA, K., AND UNGUREANU, C. Methods and systems for data management using multiple selection criteria, Dec. 1 2006. US Patent App. 11/566,122.

[12] EL-SHIMI, A., KALACH, R., KUMAR, A., ET AL. Primary data deduplication–large scale study and system design. In *Proceedings of the 2012 conference on USENIX Annual Technical Conference (USENIX'12)* (Boston, MA, USA, June 2012), USENIX Association, pp. 1–12.

[13] ESHGHI, K., AND TANG, H. K. A framework for analyzing and improving content-based chunking algorithms. *Tech. Rep. HPL-2005-30(R.1),Hewlett Packard Laboratories, Palo Alto* (2005).

[14] FU, M., FENG, D., HUA, Y., HE, X., CHEN, Z., XIA, W., ZHANG, Y., AND TAN, Y. Design tradeoffs for data deduplication performance in backup workloads. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST'15)* (Santa Clara, CA, USA, February 2015), USENIX Association, pp. 331–344.

[15] GHARAIBEH, A., AL-KISWANY, S., GOPALAKRISHNAN, S., ET AL. A GPU accelerated storage system. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC'10)* (Chicago, Illinois, USA, June 2010), ACM Association, pp. 167–178.

[16] GUO, F., AND EFSTATHOPOULOS, P. Building a high-performance deduplication system. In *Proceedings of the 2011 USENIX conference on USENIX Annual Technical Conference (USENIX'11)* (Portland, OR, USA, June 2011), USENIX Association, pp. 1–14.

[17] KRUUS, E., UNGUREANU, C., AND DUBNICKI, C. Bimodal content defined chunking for backup streams. In *Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST'10)* (San Jose, CA, USA, February 2010), USENIX Association, pp. 1–14.

[18] LILLIBRIDGE, M., ESHGHI, K., AND BHAGWAT, D. Improving restore speed for backup systems that use inline chunk-based deduplication. In *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST'13)* (San Jose, CA, USA, February 2013), USENIX Association, pp. 183–197.

[19] LILLIBRIDGE, M., ESHGHI, K., BHAGWAT, D., ET AL. Sparse indexing: Large scale, inline deduplication using sampling and locality. In *Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST'09)* (San Jose, CA, February 2009), vol. 9, USENIX Association, pp. 111–123.

[20] LILLIBRIDGE, M. D. Parallel processing of input data to locate landmarks for chunks, Aug. 16 2011. US Patent 8,001,273.

[21] LU, G., JIN, Y., AND DU, D. H. Frequency based chunking for data de-duplication. In *Proceedings of 2010 IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS'10)* (Miami Beach, FL, USA, August 2010), IEEE Computer Society Press, pp. 287–296.

[22] MEISTER, D., KAISER, J., BRINKMANN, A., ET AL. A study on data deduplication in HPC storage systems. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'02)* (Salt Lake City, Utah, USA, June 2012), IEEE Computer Society Press, pp. 1–11.

[23] MEYER, D., AND BOLOSKY, W. A study of practical deduplication. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST'11)* (San Jose, CA, USA, February 2011), USENIX Association, pp. 229–241.

[24] MIN, J., YOON, D., AND WON, Y. Efficient deduplication techniques for modern backup operation. *IEEE Transactions on Computers 60*, 6 (2011), 824–840.

[25] MUTHITACHAROEN, A., CHEN, B., AND MAZIERES, D. A low-bandwidth network file system. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP'01)* (Banff, Canada, October 2001), ACM Association, pp. 1–14.

[26] POLICRONIADES, C., AND PRATT, I. Alternatives for detecting redundancy in storage systems data. In *Proceedings of USENIX Annual Technical Conference, General Track* (Boston, MA, USA, June 2004), USENIX Association, pp. 73–86.

[27] QUINLAN, S., AND DORWARD, S. Venti: a new approach to archival storage. In *Proceedings of USENIX Conference on File and Storage Technologies (FAST'02)* (Monterey, CA, USA, January 2002), USENIX Association, pp. 1–13.

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

[29] ROMAŃSKI, B., HELDT, Ł., KILIAN, W., LICHOTA, K., AND DUBNICKI, C. Anchor-driven subchunk deduplication. In *Proceedings of The 4th Annual International Systems and Storage Conference (SYSTOR'11)* (Haifa, Israel, May 2011), ACM Association, pp. 1–13.

[30] SHILANE, P., HUANG, M., WALLACE, G., ET AL. WAN optimized replication of backup datasets using stream-informed delta compression. In *Proceedings of the Tenth USENIX Conference on File and Storage Technologies (FAST'12)* (San Jose, CA, USA, February 2012), USENIX Association, pp. 1–14.

[31] TARASOV, V., MUDRANKIT, A., BUIK, W., SHILANE, P., KUENNING, G., AND ZADOK, E. Generating realistic datasets for deduplication analysis. In *Proceedings of the 2012 USENIX conference on Annual Technical Conference (USENIX'12)* (Boston, MA, USA, June 2012), USENIX Association, pp. 24–34.

[32] TEODOSIU, D., BJORNER, N., GUREVICH, Y., MANASSE, M., AND PORKKA, J. Optimizing file replication over limited bandwidth networks using remote differential compression. *Microsoft Research TR-2006-157* (2006).

[33] WALLACE, G., DOUGLIS, F., QIAN, H., ET AL. Characteristics of backup workloads in production systems. In *Proceedings of the Tenth USENIX Conference on File and Storage Technologies (FAST'12)* (San Jose, CA, February 2012), USENIX Association, pp. 1–14.

[34] XIA, W., JIANG, H., FENG, D., AND HUA, Y. Silo: a similarity-locality based near-exact deduplication scheme with low ram overhead and high throughput. In *Proceedings of the 2011 USENIX conference on USENIX annual technical conference (USENIX'11)* (Portland, OR, USA, June 2011), USENIX Association, pp. 285–298.

[35] XIA, W., JIANG, H., FENG, D., AND HUA, Y. Similarity and locality based indexing for high performance data deduplication. *IEEE Transactions on Computers 64*, 4 (2015), 1162–1176.

[36] XIA, W., JIANG, H., FENG, D., AND TIAN, L. Accelerating data deduplication by exploiting pipelining and parallelism with multicore or manycore processors. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST'12 Poster)* (San Jose, CA, USA, February 2012), USENIX Association, pp. 1–2.

[37] XIA, W., JIANG, H., FENG, D., TIAN, L., FU, M., AND WANG, Z. P-dedupe: Exploiting parallelism in data deduplication system. In *Proceedings of the 7th International Conference onNetworking, Architecture and Storage (NAS'12)* (Xiamen, China, June 2012), IEEE Computer Society Press, pp. 338–347.

[38] XIA, W., JIANG, H., FENG, D., TIAN, L., FU, M., AND ZHOU, Y. Ddelta: A deduplication-inspired fast delta compression approach. *Performance Evaluation 79* (2014), 258–272.

[39] YU, C., ZHANG, C., MAO, Y., AND LI, F. Leap-based content defined chunking—theory and implementation. In *Proceedings of the 31th Symposium on Mass Storage Systems and Technologies (MSST'15)* (Santa Clara, CA, USA, June 2015), IEEE, pp. 1–12.

[40] ZHANG, Y., JIANG, H., FENG, D., XIA, W., FU, M., HUANG, F., AND ZHOU, Y. AE: An asymmetric extremum content defined chunking algorithm for fast and bandwidth-efficient data deduplication. In *Proceedings of IEEE INFOCOM 2015* (Hongkong, April 2015), IEEE Computer Society Press, pp. 1337–1345.

[41] ZHOU, Y., FENG, D., XIA, W., FU, M., HUANG, F., ZHANG, Y., AND LI, C. SecDep: A User-Aware Efficient Fine-Grained Secure Deduplication Scheme with Multi-Level Key Management. In *Proceedings of IEEE 31th Symposium on Mass Storage Systems and Technologies (MSST'15)* (Santa Clara, CA, USA, June 2015), IEEE, pp. 1–12.

[42] ZHU, B., LI, K., AND PATTERSON, R. H. Avoiding the disk bottleneck in the Data Domain Deduplication File System. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST'08)* (San Jose, CA, USA, February 2008), vol. 8, USENIX Association, pp. 1–14.