Future Generation Computer Systems 98 (2019) 406–418

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Accelerating content-defined-chunking based data deduplication by exploiting parallelism

Wen Xia [a,b], Dan Feng [c,*], Hong Jiang [d], Yucheng Zhang [b], Victor Chang [e], Xiangyu Zou [a,b]

[a] Harbin Institute of Technology,(Shenzhen), Shenzhen, China
[b] Peng Cheng Laboratory, Shenzhen, China
[c] School of Computer, WNLO, Huazhong University of Science and Technology, Wuhan, China
[d] Department of Computer Science and Engineering, University of Texas at Arlington, TX, USA
[e] International Business School Suzhou and Research Institute of Big Data Analytics, Xi'an Jiaotong-Liverpool University, Suzhou, China

ARTICLE INFO

Article history:
Received 28 August 2018
Received in revised form 4 December 2018
Accepted 5 February 2019
Available online 29 March 2019

Keywords:
Content-defined chunking
Data deduplication
Backup storage systems
Performance evaluation

ABSTRACT

Data deduplication, a data reduction technique that efficiently detects and eliminates redundant data chunks and files, has been widely applied in large-scale storage systems. Most existing deduplication-based storage systems employ content-defined chunking (CDC) and secure-hash-based fingerprinting (e.g., SHA1) to remove redundant data at the chunk level (e.g., 4 KB/8 KB chunks), which are extremely compute-intensive and thus time-consuming for storage systems. Therefore, we present P-Dedupe, a pipelined and parallelized data deduplication system that accelerates deduplication process by dividing the deduplication process into four stages (i.e., chunking, fingerprinting, indexing, and writing), pipelining these four stages with chunks & files (the processing data units for deduplication), and then parallelizing CDC and secure-hash-based fingerprinting stages to further alleviate the computation bottleneck. More important, *to efficiently parallelize CDC with the requirements of both maximal and minimal chunk sizes* and inspired by the MapReduce model, we first split the data stream into several segments (i.e., "Map"), where each segment will be running CDC in parallel with an independent thread, and then re-chunk and join the boundaries of these segments (i.e., "Reduce") to ensure the chunking effectiveness of parallelized CDC. Experimental results of P-Dedupe with eight datasets on a quad-core Intel i7 processor suggest that P-Dedupe is able to accelerate the deduplication throughput near linearly by exploiting parallelism in the CDC-based deduplication process at the cost of only 0.02% decrease in the deduplication ratio. Our work provides contributions to big data science to ensure all files go through deduplication process quickly and thoroughly, and only process and analyze the same file once, rather than multiple times.

1. Introduction

A recent report from IDC [1] suggests that the total size of worldwide digital information is about 4.4 ZB in 2013, and that size will be about 44 ZB in 2020. As an efficient data reduction technique, data deduplication has been wildly used as a system-level data reduction technology to improve storage and network efficiency [2–5]. Data deduplication removes duplicate data by storing only one physical instance and other duplicate data refers to it. Specifically, deduplication adopts a secure hash function (i.e., the cryptographically secure hash signature, such as SHA-1) to calculate a fingerprint (i.e., hash digest) [2,6], to uniquely and globally represent data regions (e.g., 8 KB data chunks) and thus detect & eliminate duplicates in storage systems. Storage workload studies from Microsoft [7,8] and EMC [9,10] suggest that deduplication technique detects about 50%–85% of the duplicate data in their production systems, respectively.

The chunk-level deduplication strategy detects more duplicates than its file-level counterpart, and thus is most widely used in the existing deduplication systems [7]. The chunk-level deduplication efficiently eliminates the redundancy in storage systems but requiring more processing time, which will be the bottleneck in the write path for deduplication-based storage systems [11–16]. The reason is that the deduplication process, that typically consists of four stages: (*S*1) content-defined chunking (CDC), (*S*2) fingerprinting, (*S*3) looking up the fingerprint index, and (*S*4) storing the data chunks and metadata, are time-consuming, especially the chunking and fingerprinting steps occupy large amount of CPU resources. More specifically, the chunking (i.e., content-defined chunking [17]) and fingerprinting (e.g., SHA1) tasks all need to compute the hash signatures, which are time-consuming

* Corresponding author.
E-mail addresses: xiawen@hit.edu.cn (W. Xia), dfeng@hust.edu.cn (D. Feng), hong.jiang@uta.edu (H. Jiang), hust.yczhang@gmail.com (Y. Zhang), victor.chang@xjtlu.edu.cn (V. Chang), xdnzxy@gmail.com (X. Zou).

https://doi.org/10.1016/j.future.2019.02.008
0167-739X/

in the write path in deduplication-based storage systems [14,15, 18,19].

With multicore/manycore processors becoming increasingly more popular, the current computer systems are able to offer a much powerful parallel computing power for any CPU-intensive tasks [13,16,20]. Meanwhile, our observation of data deduplication tasks (see Section 2.3) suggests that the data deduplication tasks to be organized and viewed by data units (i.e., data chunks and files) and functional units (e.g., content-defined chunking, fingerprinting, and indexing), are mutually independent. Therefore, it is able to take advantage of the abundant computing power in modern CPUs for deduplication by maximally parallelizing the computation functional units (i.e., CDC and fingerprinting) that are then fed by the indexing and storing stages for the data units, such as data chunks.

Finally, we present P-Dedupe, a pipelined and parallelized data deduplication system that fully mines parallelism of deduplication tasks to alleviate the hash calculation bottleneck. By dividing the deduplication process into four stages (i.g., chunking, fingerprinting, indexing, and writing), P-Dedupe first pipelines these four independent stages. To fully exploit parallelism in P-Dedupe, we then propose the parallel CDC & fingerprinting approaches to maximize the computation efficiency for deduplication. More specifically, *to efficiently parallelize CDC* and inspired by the *MapReduce* model [20], we first split data stream into several data segments (i.e., "Map"), where each segment will be running CDC in parallel with an independent thread, and then re-chunk & join the boundaries of the adjacent segments (i.e., "Reduce") to ensure that the chunking effect of parallelized CDC will be nearly identical to that of the conventional sequential CDC. In addition, we analyze and demonstrate in-depth on the effectiveness of P-Dedupe running the classic CDC approach with the requirements of both maximal and minimal chunk sizes.

Therefore, by fully exploiting parallelism of deduplication process, P-Dedupe greatly accelerate the hash calculation tasks for deduplication and alleviate the computation bottleneck in deduplication-based storage systems. Extensive experiments of P-Dedupe on a quad-core Intel i7 processor, tested on the eight datasets, suggest that fully exploiting parallelism for chunking & fingerprinting can accelerate the throughput of CDC based deduplication by a factor of 3∼4 while only decreasing the deduplication ratio by about 0.02%, on all the three studied content-defined chunking algorithms, namely, Rabin [17], Adler [21], and Gear [22].

In the rest of the paper, Section 2 introduces necessary background, related work, and motivation for P-Dedupe. Section 3 elaborates the detailed design and techniques of P-Dedupe system. Section 4 presents and discusses experimental results of P-Dedupe. Section 5 concludes P-Dedupe and the future work.

2. Background, related work, and motivation

2.1. Background and related work

Recently, chunk-level data deduplication becomes one of the most popular data reduction method in storage systems for improving storage and network efficiency. 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 [2,3,6,23]. In the past ten years, data deduplication technique has been demonstrated its space efficiency functionality in the large-scale production systems of Microsoft [7,8] and EMC [9,10]. As detailed in Table 1, several published papers from both industry and academia indicate that *data deduplication (at the 8 KB chunk level) is able to detect duplicates of about 68%, 90%, and 27% over the total logical storage space in primary storage [7], backup storage [9,10], and HPC data centers [24], respectively*.

Because of the great space saving efficiency, deduplication techniques have been widely applied for data reduction in the backup storage systems [23,25,26], primary storage systems [5,8, 14,27], visual machines [28], cloud storage [29], and other emerging applications [4,30]. Such as, CAFTL [11] and CA-SSD [12] schemes employ deduplication techniques to eliminate duplicate writes and remove duplicate data blocks and thus finally improves space efficiency and write performance.

For the chunk splitting methods used in data deduplication systems, many studies have been also investigated the efficiency of Fix-Sized Chunking (FSC) [6,31–33], Content-Defined Chunking (CDC) [2,7,34–36]. Generally, CDC-based approaches can address the "boundary-shift" problem (see Section 2.3) and thus detect more duplicates than FSC-based approaches when using similar average chunk size. A recent study [7] from Microsoft suggests that CDC-based approaches detect about 12% more redundant data than the FSC-based approaches in Microsoft production storage workloads of about 162 TB.

One challenge facing data deduplication is the scalability of the fingerprints indexing, which faces the on-disk index-lookup performance bottleneck when the size of fingerprints exceed the RAM space. DDFS [23] exploits the inherent backup stream locality to maximize the hit ratio of the fingerprints cache and thus reduce accesses to on-disk fingerprints index for inline deduplication. ChunkStash [25] uses SSDs instead of HDDs to store fingerprints for deduplication indexing and thus improves the index-lookup performance, and employs Cuckoo hash as data structure to organize the KV-store in memory while exploiting backup-stream locality.

Another challenge facing data deduplication is that the high I/O throughput performance required by the storage systems while the inline data deduplication processing is time-consuming because of the frequent hash computation of CDC and fingerprinting.

Currently, two kinds of methods are proposed to improve efficiency of the hash calculating process and thus alleviate the computation bottleneck: *hardware-based* and *software-based* approaches. The *hardware-based* approaches refer to using GPGPU or a dedicated coprocessor to reduce the time overheads for chunking and hashing calculations and thus minimize the storage I/O performance degradation induced by deduplication. Two typical work, namely, StoreGPU [13,19] and Shredder [15] take advantages of GPGPU's great computing power to speed up the computation tasks (e.g., hashing, erasure coding, etc.) in storage systems.

The *software-based* approaches make full use of the parallelism in deduplication tasks instead of introducing additional hardware devices for fast computing. Guo et al. [37] present an event-driven, multi-threaded client–server interaction model to boost the throughput of the FSC-based deduplication system. It also propose methods of "progressive sampled indexing" and "grouped mark-and-sweep" to improve performance and scalability of deduplication systems. Ma et al. [38] introduce an adaptive pipelining model for the compute-intensive sub-tasks of fingerprinting, compressing, etc. in Fix-Sized Chunking based deduplication system. Liu et al. [39] parallelize fingerprinting calculation for deduplication by first dividing each data chunk into several pieces, then fingerprinting concurrently of these pieces, and finally combining the fingerprints as the chunk's. However, the above mentioned solutions only focus on Fix-Sized Chunking (FSC) based approach. Due to the internal content-dependency as discussed in Section 2.2, it is challenging for the CDC-based deduplication to fully parallelize the computation tasks.

Table 1
Summary of recently studies on data deduplication of large-scale real-world datasets.

Institutes	Workloads	Before deduplication	After deduplication
Microsoft	857 desktop computers [7]	162 TB	52 TB
EMC	Over 10,000 production backup systems [9]	682 TB	89 TB
	6 large backup datasets [10]	33 TB	2.7 TB
Mainz Univ.	Four HPC data centers [24]	1213 TB	886 TB

2.2. Independence and dependence

Recently, the bandwidth of storage hardware devices grows steadily while the computing power of one single core stagnates [13,19], the CDC and fingerprinting stages of the deduplication process are computationally very expensive. Our experimental observations (running on Ubuntu 12.04.2 and CPU 2.8 GHz Intel i7 quad-core) show the throughputs of running Rabin-based CDC, SHA1-based fingerprinting, and SHA256-based fingerprinting on a single core are 245 MB/s, 345 MB/s, and 150 MB/s respectively. Since the hash calculations (e.g., Rabin, SHA1, and SHA256) are **dependent** on the contents of deduplication-processed data streams/files, it remains a challenge to speed up the CDC and fingerprinting calculations.

Literature review from real-world CDC-based deduplication systems described above show that data deduplication subtasks are actually running **independently**. Generally, CDC-based deduplication process can be divided into four stages of ***content-defined chunking, fingerprinting, indexing, and storing***. The storing stage includes writing the non-duplicate data and system metadata accordingly. All of the above mentioned deduplication subtasks can be viewed independent since they are serially processed on each independent data unit (i.e., chunk). Specifically, the content-defined chunking of the chunk A_2 can be running concurrently with the stages of fingerprinting, indexing, and storing of its previous chunk A_1. Hence, the four stages of deduplication (i.e., chunking, fingerprinting, indexing, and storing) could be pipelined to fully utilize the computing power of the multicore/manycore processors.

2.3. Challenge of parallelizing content-defined chunking

Content-Defined Chunking (CDC) scheme is introduced to address the known "boundary-shift" issue [2]: if the insertion or deletion operation happens in one chunk of a file (e.g., the chunk C_5 in Fig. 1), it will affect its subsequent (CDC-calculated) chunks, which can reduce the deduplication efficiency of the Fix-Sized Chunking (FSC) based approaches. As shown in Fig. 1, CDC scheme employs a technique of the sliding window to run forward on the byte content and determine the chunk boundary once the hash value of the current sliding window meets the pre-defined condition. Hence, as shown in Fig. 1, for the modified file V_2 (over file V_1, where the chunk C_2 is changed), CDC scheme is able to obtain the right cut-points of data chunks C_3 and C_4 whose content remains unmodified.

To verify the duplicate detection effects of the CDC and FSC approaches, we use eight datasets for evaluations (the detailed experiment environment and datasets characteristics can be referred to in Section 4.1), which represent the workloads of source code repository, virtual machine hosting, and database images respectively. Fig. 2 shows the numbers of duplicate detection with the chunking approaches of CDC and FSC, which demonstrates that CDC-based approach detects 7%–80% more redundant data than FSC-based approach. It is worth to note that CDC works best on the Redis dataset, this is because modifying some fields of the Redis database images head will affect its subsequent chunks according to our observation.

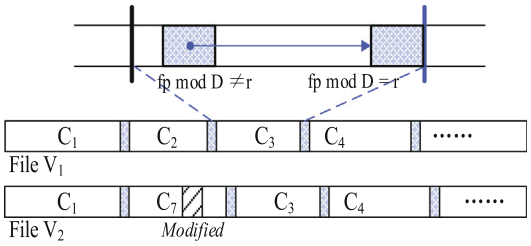

Fig. 1. An example of the sliding window scheme used in the Content-Defined Chunking (CDC). Hash value of the sliding window, f, is calculated using the Rabin hash [2]. If the lowest log_2D bits of the hash value meets the pre-defined requirement (i.e., $fmodD = r$), the current position is marked as a "cut-point" (i.e., chunk boundary) for CDC. Otherwise, the sliding window will move forward byte-by-byte to find the correct chunk boundary.

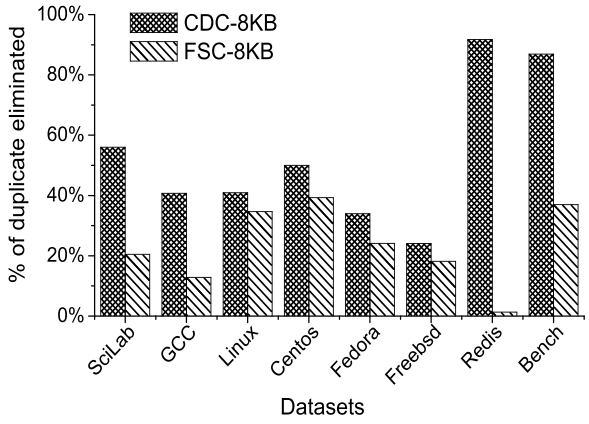

Fig. 2. The duplicate detection effects of the content-defined chunking (Rabin-based CDC) and the fix-sized chunking (FSC) running on the eight datasets with average chunk size of 8 KB.

As the CDC-based deduplication detect much more duplicates than the FSC-based approaches especially when files are frequently modified and stored multiple copies [7,34,40], many production deduplication systems use the CDC-based schemes [23, 26]. Nevertheless, the CDC calculation is time-consuming, which seriously degrades the write throughput of inline-deduplication-based storage systems. Therefore, fully exploiting parallelism of CDC and other compute-intensive tasks are one of hottest research topics for deduplication-based storage systems.

Parallelizing the CDC task, however, is challenging and cannot be easily parallelized as parallel fingerprinting. As described in Fig. 1, the Rabin-based CDC employs the sliding window technique that is heavily **dependent** on the data content, which is difficult to be parallelized. In particular, Rabin's hash value for a sliding window (e.g., bytes sequence $B_1, B_2, \ldots, B_\alpha$), is defined as below:

$$Rabin(B_1, B_2, \ldots, B_\alpha) = \{\sum_{x=1}^{\alpha} B_x p^{\alpha - x}\} modD \quad (1)$$

where α is the length of the sliding window and D is the expected average chunk size. The Rabin-based CDC is a rolling hash that can calculate the hash value in an incremental manner. In particular, for the sliding window that move forward byte-by-byte on data content, the hash value is able to be calculated incrementally according to the last one as detailed below:

$$Rabin(B_i, B_{i+1}, \ldots, B_{i+\alpha-1}) = \{[Rabin(B_{i-1}, \ldots, B_{i+\alpha-2}) - B_{i-1}P^{\alpha-1}]p + B_{i+\alpha-1}\} mod S \quad (2)$$

Therefore, the CDC process cannot be running concurrently on one data chunk, because the chunk boundaries (i.e., "cut-points") always remain uncertain until the current CDC process for this chunk is finished. As a result, such strong data dependence makes it very difficult to parallelize the CDC process. To make the things worse and complicated, the conventional CDC approaches employ the maximum and minimum chunk size requirements to eliminate the edge cases of generating extremely large or small chunks by CDC [2,7,26,36,40,41]. For example, they impose minimum and maximum chunk size of 2 KB and 64 KB respectively when chunking the data stream with 8 KB average chunk size.

But, according to the independence among the deduplicated chunks and files, as well as the tasks of chunking and fingerprinting as discussed in Section 2.2, it is possible to first pipeline the deduplication tasks and then further parallelize CDC and fingerprinting, and fully utilize the computing power of multicore/manycore processors in computer systems, as analyzed below.

Given the time overhead consumed by the content-defined chunking, fingerprinting, indexing, and storing stages, denoted by T_c, T_f, T_i, and T_w respectively. Here D denotes the deduplication factor (the ratio of the data size before/after deduplication), then T_w/D represents the time of writing the amount of non-duplicate data. The write throughput can be estimated as below (without deduplication):

$$X_{put} = 1/T_w \quad (3)$$

The write throughput with traditional serial deduplication's can be calculated as below:

$$X_{put} = 1/(T_c + T_f + T_i + T_w/D) \quad (4)$$

The write throughput with the pipelined deduplication can be derived as below:

$$X_{put} = 1/Max(T_c, T_f, T_i, T_w/D) \quad (5)$$

When we further parallelize the chunking and fingerprinting tasks with N parallel threads, the throughput can be derived as below:

$$X_{put} = 1/Max(\frac{T_c}{N}, \frac{T_f}{N}, T_i, \frac{T_w}{D}) \quad (6)$$

As the previous studies [25,26,42,43] suggest that the indexing throughput for deduplication tends to be very fast by effectively exploiting redundant data locality, about several hundreds of Megabits or Gigabytes per seconds, which is faster than the commonly-known speeds of chunking and fingerprinting. Hence, if hash-computation parallelism makes "T_c/N" and "T_f/N" $\leq$ "T_w/D", the write throughput will be about D times faster than that without deduplication. Therefore, fully exploiting parallelism of deduplication tasks is able to greatly improve the deduplication throughput and significantly alleviate the computation bottleneck in deduplication-based storage systems.

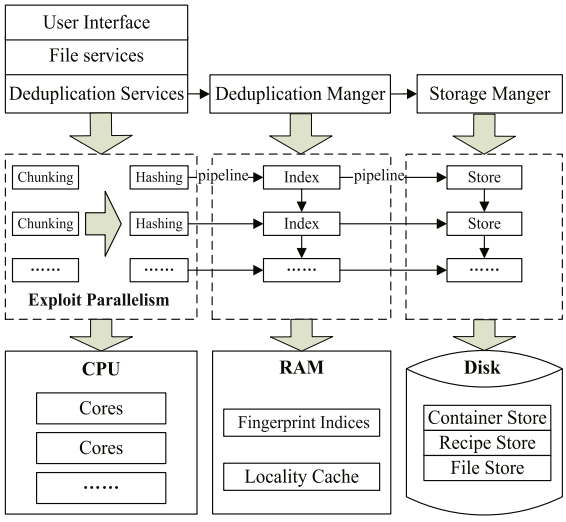

Fig. 3. P-Dedupe system architecture.

3. Design and implementation

3.1. System architecture overview

P-Dedupe aims to provide high throughput for deduplication-based storage systems. This section first provides an overview of P-Dedupe system, which finally focuses on pipelining and parallelizing hash calculation tasks, such as CDC and fingerprinting. As shown in Fig. 3, P-Dedupe is made up of several system components as below:

- *User Interface* refers to the data/file access interface (e.g., write, read, etc.) in deduplication-based storage systems.
- *File Services* manages file metadata and name space, where the file metadata needs to be further updated when chunk-level deduplication is complete.
- *Deduplication Services* offer the pre-processing functions of data deduplication, including parallelized content-defined chunking, parallelized secure-hash based fingerprinting, etc. *Deduplication Services* fully utilize the abundant computing power of modern CPUs to accelerate the CDC and fingerprinting process (detailed in Section 3.2, 3.3, and 3.4).
- *Deduplication Manager* is in charge of managing the chunk-fingerprint index for deduplication. Note that as this paper aims to maximally accelerate hash calculation for data deduplication, we put all the fingerprints (i.e., *fingerprint indices*) in the RAM for simplification. Other deduplication systems [23,26,42] that provides high-performance fingerprint indexing by exploiting locality in cache (called *locality cache*) can be employed here for managing and indexing fingerprints between RAM and disks.
- *Storage Manager* includes *File Store*, *Recipe Store*, and *Container Store*. *File Store* is responsible for storing file metadata. *Container Store* is responsible for storing non-duplicate chunks using a fixed-size structure (e.g., 4 MB) [23]. *Recipe Store* establishes the mapping relations (called file recipe [44–46]) between *File Store* and *Container Store* after chunks deduplicated.

To put things into perspective, we provide a general workflow of the P-Dedupe system process. A data stream goes through *User Interface* to the *File Services* layer and then stores the corresponding file metadata, while entering the P-Dedupe system. *Deduplication Services* use by content-defined chunking technique

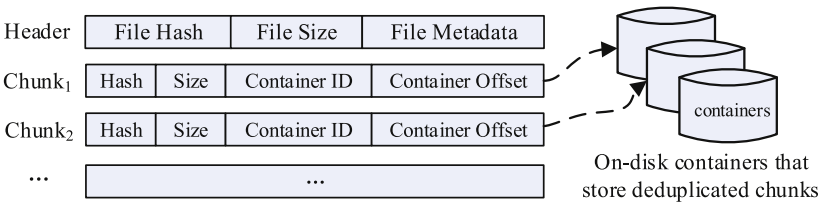

Fig. 4. Data structure of the file recipe in the P-Dedupe system.

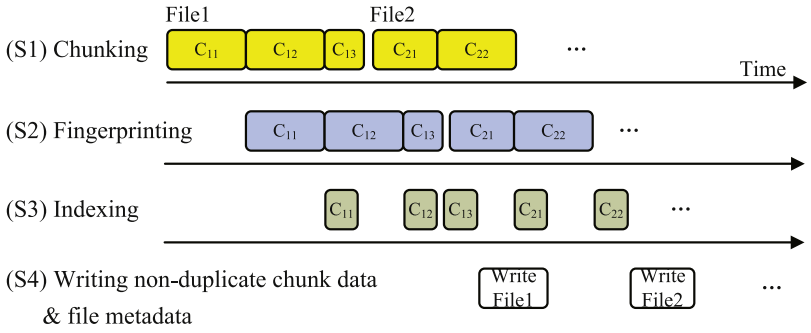

Fig. 5. Pipelining Deduplication in P-Dedupe system.

to split the input data stream into several chunks and then calculate the chunks' fingerprints. *Deduplication Manager* look up the chunks' fingerprints in the index database to determine the duplicate chunks. Finally, for the non-duplicate chunks, the chunk data and the metadata will be updated in *Container Store*, *File Store*, and *Recipe Store* accordingly. For the duplicate chunk, only the metadata (the mapping information points to the duplicate chunks) will be recorded.

For the read operation, the client initiates it through *User Interface* and *File Services* layer. *Deduplication Manager* gets the chunk-to-file mapping relationship from *Recipe Store* based on the metadata in *File Store*. The chunks will then be obtained from *Container Store* according to the mapping relationship and reconstructed into a file with its the metadata. Note that for file modification, the metadata of the modified (new) chunks in the file recipe will be updated. For the file deletion, garbage collection will be required for reclaiming the storage space of the deleted chunks in *Container Store*, which has been exploited in many studies [37,43,45,46].

Fig. 4 depicts the data structure of *Recipe Store*, which efficiently builds the mapping relations between the deduplicated chunks and files in P-Dedupe. To give an example, when the chunk C_2 (logically belongs to $File_2$ as shown in Fig. 4) is detected to be duplicate to the chunk C_1 (logically belongs to $File_1$) that is already stored in *Container Store*, P-Dedupe only needs to record C_1's mapping relationship (that points to the physical address of chunk C_2) in $Recipe_{F2}$ in *Recipe Store*. Therefore, the system is able to reconstruct $File_2$ according to $Recipe_{F2}$'s chunk-to-file mapping information.

3.2. Pipelining deduplication

As discussed in Section 2.3, by utilizing the abundance of computing power of the modern CPUs, thus it is able to improve the deduplication system throughput by exploiting parallelism pf the computation tasks accordingly.

As shown in Fig. 5, P-Dedupe's pipelining technique divides the deduplication process into four stages: ($S1$) content-defined chunking (CDC), ($S2$) secure-hash-based fingerprinting, ($S3$) looking up the fingerprints, and ($S4$) storing chunk data and metadata (such as file recipe in Fig. 4). Since deduplication process runs on chunk-by-chunk, the pipeline can be designed to run with the data unit of chunks in the whole deduplication process. As a result, as illustrated in Fig. 5, the compute-intensive stages (i.e., $S1$ and $S2$) are separated from the I/O-intensive stages (i.e., $S3$ and $S4$) in the chunk-based deduplication pipeline. This pipelined technique greatly reduces the waiting time on the chunking and fingerprinting stages and thus maximize the computation efficiency in P-Dedupe.

In addition to the aforementioned chunk-level deduplication pipeline, the file metadata updating can be also considered into the deduplication pipeline (see stage S4 in Fig. 5), especially when there are many input small files. Thus P-Dedupe can get operating file metadata right after deduplication. This greatly improves the pipeline efficiency.

Although P-Dedupe's pipeline scheme is designed for the CDC-based systems [17], it can be also applied for any data deduplication systems using Fix-Sized Chunking (FSC), where the stage $S1$ "chunking" in the above pipeline in Fig. 5 would be removed.

3.3. Mapreduce content-defined chunking

The main challenge for parallel deduplication in P-Dedupe lies in the Content-Defined Chunking (CDC), which employs a sliding window technique running byte-by-byte on the input data stream (see Fig. 1).

Inspired by the Phoenix project [20] that uses the MapReduce model to accelerate computation tasks on multi-core and multiprocessor systems, we propose a "mapreduced" CDC approach to parallelize the content-defined chunking on several divided but contiguous substreams, also called data "**segment**", of the same data stream of appropriate length (e.g., 1 MB or 2 MB) that may generate chunks following the CDC rules. Specifically, P-Dedupe first splits a data stream into many fix-sized data segments (i.e., "Map") and the size of each segment should be much larger than the pre-defined maximum chunk size for CDC. Then the CDC is running on the data segments concurrently. After the CDC of these segments are finished individually, the boundaries among those segments will be slightly corrected accordingly (i.e., "Reduce"). This is because the ends of each segments are automatically determined as the chunk "cut-points" in P-Dedupe, which does not strictly follow the CDC rules.

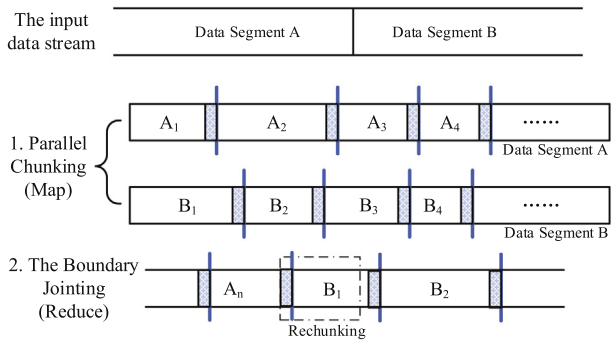

Fig. 6. The parallel CDC runs concurrently with two threads. The CDC task is "mapreduced": the data stream is mapped to two segments *A* and *B* and then the boundaries of segments *A* and *B* are re-chunked for correction.

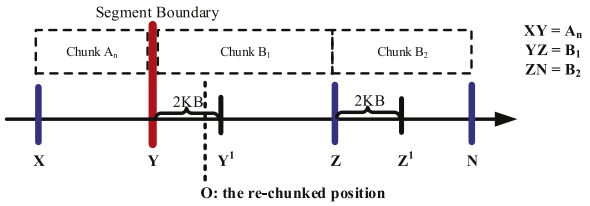

Fig. 7. Illustration of the boundary jointing (re-chunking) approach after parallel chunking of data segments *A* and *B*. The most possible re-chunked position *O* of chunks A_n and B_1 is located between the *Y* and Y^1.

Fig. 6 provides an example of parallel CDC: two segments *A* and *B* are running CDC concurrently to generate chunks $A_1 \sim A_n$ and $B_1 \sim B_n$ respectively, then the last chunk in segments *A*, A_n, may be determined as a "tentative chunk" since the last "cut-point" may not strictly meet the CDC rule as shown in Fig. 1 in Section 2.3. The correct "cut-point" for chunk A_n may be located inside B_1, and we need to re-chunk them (i.e., A_n and B_1) for correction as the 2nd step of parallelizing CDC as shown in Fig. 6.

As discussed in Section 2.3, *most of CDC-based deduplication systems use maximum and minimum chunk sizes to eliminate the pathological cases of many extremely large and small chunks* [2,7,26,36,40,41]. Thus in this paper, we mainly discuss how to re-chunk the two chunks in the segment boundaries with requirements of the maximum/minimum chunk sizes for CDC. When maximum/minimum chunk sizes are not required, P-Dedupe only needs to re-chunk the regions of two CDC sliding windows on the two adjacent chunks in the segment boundary while achieving the same chunk positions as the sequential CDC approach [47].

Fig. 7 gives an example of re-chunking the two chunks (i.e., A_n and B_1) in the segment boundary after parallel chunking of data segments *A* and *B* when the average, minimum, and maximum chunk sizes are configured to 8 KB, 2 KB, and 64 KB respectively, the same as LBFS [2]. *Here we assume the re-chunked position between chunks A_n and B_1 is located at O, and the next chunked position after the chunk XO is located at O^1.* Thus if P-Dedupe gets the same chunked positions as the sequential CDC approach at the cut-points *O* and O^1, it means that P-Dedupe does not sacrifice the deduplication ratio while efficiently parallelizing content-defined chunking on each segment. But according to our observation, when P-Dedupe re-chunks A_n and B_1, there will be five cases of locations of *O* and O^1 as follows.

- *Case 1*: $O \in [Y, Y^1]$, $O^1 = Z$. In this case, chunks A_n and B_1 (i.e., *XY* and *YZ* in Fig. 7) will be re-chunked into chunks *XO* and *OZ*, which will be the same case as the sequential CDC approach.

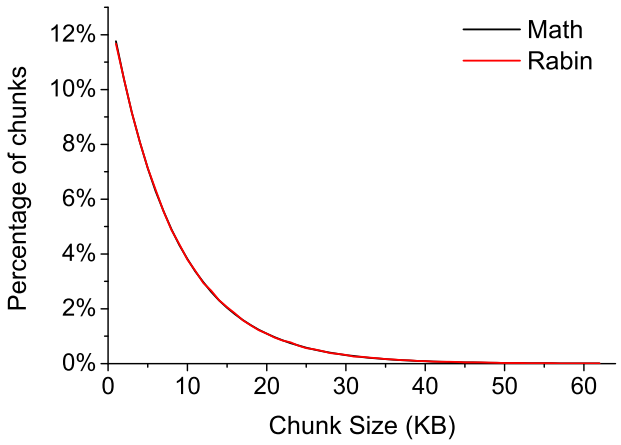

Fig. 8. Chunk-size distribution of Rabin-based CDC approach with 8 KB average chunk size and without maximum/minimum chunk size requirements. "Rabin" denotes our experimental observation of chunk-size distribution and "Math" denotes the theoretical exponential distribution of Eq. (1).

- *Case 2*: $O \in [Y, Y^1]$, $O^1 \in (Z, N]$. This case happens when P-Dedupe finds the re-chunked position *O* between $[Y, Y^1]$ but the *OZ* is smaller than the minimum chunk size of 2 KB (thus the position *Z* would not be an effective cut-point for rechunking).
- *Case 3*: $O \in (Y^1, Z)$, $O^1 = Z$. This case happens when P-Dedupe does not find the chunked position between $[Y, Y^1]$, the *XZ* is greater than the maximum chunk size of 64 KB (thus the position *Z* would not be an effective cut-point as "O" for rechunking), and *OZ* is greater than the minimum chunk size of 2 KB.
- *Case 4*: $O \in (Y^1, Z)$, $O^1 = (Z, N]$. This case happens when P-Dedupe does not find the chunked position between $[Y, Y^1]$, the *XZ* is greater than the maximum chunk size of 64 KB, but *OZ* is smaller than the minimum chunk size of 2 KB (thus the position *Z* would not be an effective cut-point for rechunking).
- *Case 5*: $O = Z$. This is the best case, and we only need to merge the chunks A_n and B_1 into a new chunk (i.e., *XZ*).

In *Cases 1, 3, and 5*, P-Dedupe achieves the same chunking effect as the sequential CDC but does not in *Cases 2 and 4* by only re-chunking the chunks A_n and B_1 in the segments boundary. Now we further analyze *Cases 2 and 4*. As shown in Fig. 8, our experimental observation and mathematic analysis suggest that cumulative distribution of chunk size *X* in Rabin-based CDC approach with average chunk size of 8 KB (no configuration of the max/min chunk sizes) follows the exponential distribution as below:

$$P(X \leq x) = F(x) = (1 - e^{-\frac{x}{8192}}), x \geq 0 \quad (7)$$

It is worth noting that this theoretical exponential distribution of Eq. (1) is based on the assumption that data content and its calculated hash values of content (for chunking) follow the uniform distribution. The chunk-size distribution shown in Fig. 8 demonstrates that the predictive results based on Eq. (7) are totally identical to our experimental observation of distributions, which provide convincing results for our further analysis of *Cases 2 and 4*.

Eq. (7) and Fig. 8 suggest that the probability of chunk size $x \geq$ 64 KB is equal to $e^{(-8)} \approx 0.000335$, which means *Case 3* happens at an extremely low probability. Meanwhile, the probability of chunk size $x \leq 2$ KB is equal to $(1 - e^{(-0.25)}) \approx 0.2212$, which means the events of *Cases 1* and 2 ($O \in [Y, Y^1]$) happen at the

probability of about 0.2212. The probability that *Cases 2 and 4* happen can be calculated as blow:

- *Case 2* is equal to the event that two sequential chunks are both smaller than 2 KB, whose probability is about 0.049.
- *Case 4* is equal to the event that the first chunk is greater than 64 KB and the next chunk is smaller than 2 KB, whose probability is about 0.000074.
- *Cases 1, 3, and 5* happens at the probability of $(1 - 0.049 - 0.000074) = 0.959926$.

Cases 2 and 4 results in a bad situation where the chunked position O^1 needs to be re-chunked. In the worst case, all the subsequent chunks positions $O^2 \sim O^n$ will need to be re-calculated. *Here we argue that we just simply consider the chunked position Y (from parallel segment-chunking) as the O^1 in Cases 2 and 4, to minimize the overhead of boundary jointing in P-Dedupe*. Even though the *Cases 2 and 4* happen, their subsequent chunked positions $O^2 \sim O^n$ will be identical to the sequential CDC approach at a very high probability (the same situation as *Cases 1, 3, and 5*).

To put things more clearly, the chunks whose boundaries are not identical to the sequential CDC approach, we called "dirty" chunks in P-Dedupe. Base on the above discussions of the five cases, the "dirty" chunk is a rare event only existing in the *Case 2* and *Case 4*. Specifically, the expected ratio of "dirty" chunks in each segment generated by *Case 2* could be calculated as below:

$$E_{case2} = \sum_{n=1}^{\infty} 0.2211^{n+1} \times 0.7789 \times \frac{10\ \text{KB} + n \times 2\ \text{KB}}{SegmentSize} \approx \frac{0.6258\ \text{KB}}{SegmentSize} \tag{8}$$

As *Case 4* happens at a probability that about 1/1000 of *Case 2*, the expected ratio of dirty chunks generated by *Cases 2 and 4* are nearly equal to the result shown in Eq. (8), which means that less than 0.1% of chunks are dirty chunks in P-Dedupe when using data segment of 1 MB for parallelizing chunking.

Therefore, P-Dedupe achieves nearly the same chunking effect of the sequential CDC approach by only re-chunking the two chunks in the segment boundaries after parallel chunking. Meanwhile, P-Dedupe linearly speedups the deduplication throughput via this parallel chunking scheme, as evaluated and demonstrated in Section 4. It is noteworthy that the chunking of the files smaller than the segments size (e.g., 1 MB) could be easily be parallelized by allocating one chunking thread for one file since the chunking processes among different files are totally independent thus could be running in parallel. And our work focus on parallelizing the content-defined chunking of the large files, which are the main sources for deduplication [7,42].

3.4. Parallelizing fingerprinting

As P-Dedupe pipelines the deduplication sub-tasks and parallelizes the time-consuming content-defined chunking, the fingerprinting will be the next bottleneck for P-Dedupe as shown in Fig. 5. As discussed in Section 2.3, the time waiting for fingerprinting can be also reduced by exploiting parallelism since the different data chunks, the subjects of fingerprinting, could be running independently. Hence, P-Dedupe can directly parallelize fingerprinting on data chunks generated from parallelized CDC (i.e., Stage *S3* in Fig. 5) by allocating one thread for fingerprinting each chunk with the technique of thread pool [48].

The challenge facing parallel fingerprinting is that the output sequence of chunks' calculated fingerprints must follow the input sequence (i.e., the sequence of chunks in a file) in order to feed into the indexing and writing stages in the pipeline correctly as shown in Fig. 5 in Section 3.2. Take an example of the parallel fingerprinting three chunks with the sequence of $X - Y - Z$ may generate an out-of-sequence completion (e.g., $X - Z - Y$ or $Y - X - Z$) because of the dynamic runtime conditions of CPUs and the various sizes of data chunks generated by CDC. Therefore, the synchronization of the parallel fingerprinting threads is considered into the P-Dedupe pipeline, so that the chunks' fingerprints are fed to the following stages *S3 & S4* in the pipeline in the correct sequence, which outputs the same results as the traditional sequential CDC-based deduplication approach. For synchronization of parallel fingerprinting, the functions of *pthread_mutex_lock()* and *pthread_cond_wait()* of thread pool [48] are used to wait for the fingerprinting of chunks running in the sequential order as the input. According to our observation, the overhead of this synchronization is negligible since the most of the chunks are of the similar size after chunking and the parallel chunking and fingerprinting are running faster in P-Dedupe.

4. Performance evaluation

4.1. Experimental setup

Experimental Platform. P-Dedupe system is evaluated on the Ubuntu 12.04.2 with a quad-core and eight-thread Intel i7 processor running at 2.8 GHz, 64 GB RAM, and two 1 TB 7200 RPM hard disks.

Configurations for Data Reduction. We configure the Rabin and SHA-1 algorithms for CDC and fingerprinting respectively in P-Dedupe. The maximum, average, and minimum chunk sizes for P-Dedupe are set 64 KB, 8 KB, and 2 KB respectively, which is also used in LBFS [2]. ***We parallelize the chunking and fingerprinting tasks of both 4 threads in P-Dedupe, and parallel threads is managed by the thread pool in our implementation***. In addition, we also test other content-defined chunking algorithms, namely, Gear-based chunking [22] and Adler-based chunking [21] to further evaluate P-Dedupe's parallel chunking approach. The fingerprint index is all put into RAM to maximally examine the deduplication throughput accelerated by P-Dedupe via exploiting parallelism for chunking and fingerprinting. The thread pool technique [48] is used for programming and managing multi-threads in P-Dedupe.

Evaluation Metrics. *Deduplication ratio (DR)* is measured in terms of the percentage of duplicate data eliminated by deduplication. *Deduplication throughput (DT)* is measured by the processing throughput at which the input files/data is processed by deduplication. We run each experiment for many times to get the stable and average results of the deduplication throughput. Since high-bandwidth fiber channel adapters and storage device (e.g., PCM) are fairly expensive, we instead use a RAMdisk-driven emulation in our testbed (to ensure the transferring and storing stages would not be performance bottlenecks) to experimentally validate the deduplication throughput of P-Dedupe, which is as the same as that in the Shredder [15] and StoreGPU [13,19] research. To better evaluate P-Dedupe, we implement the traditional serial deduplication as the "*Baseline*" approach (i.e., without pipelining and parallelizing the deduplication tasks) for comparison.

Evaluation Datasets. Table 2 introduces the workload characteristics of eight evaluated datasets, which is also detailed as below:

- **SciLab**, **GCC**, and **Linux** datasets are three open-source projects representing workloads of the large-scale code datasets, and are publicly downloadable [49–51]. Each version of these software projects is tarred together to simplify and speedup the backup process [9].

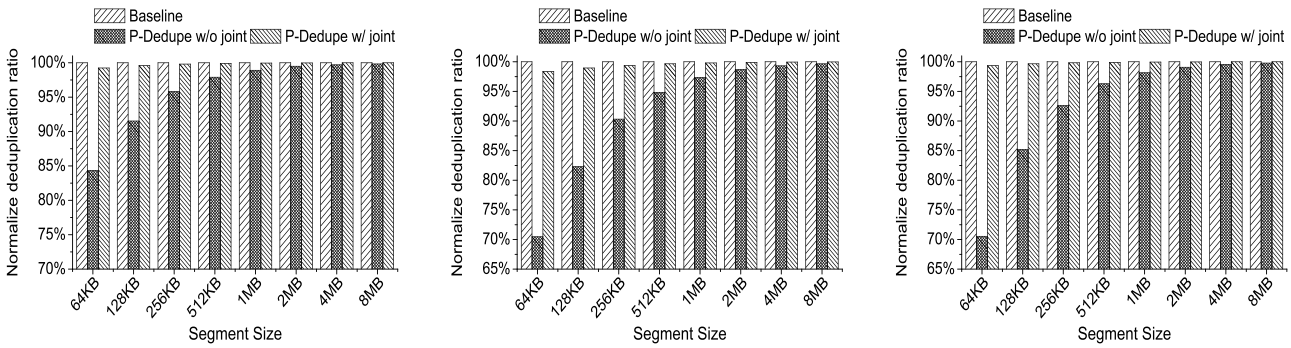

Fig. 9. Deduplication ratio as a function of different parallel data segment size among the approaches of Baseline, P-Dedupe with and without segment boundary jointing.

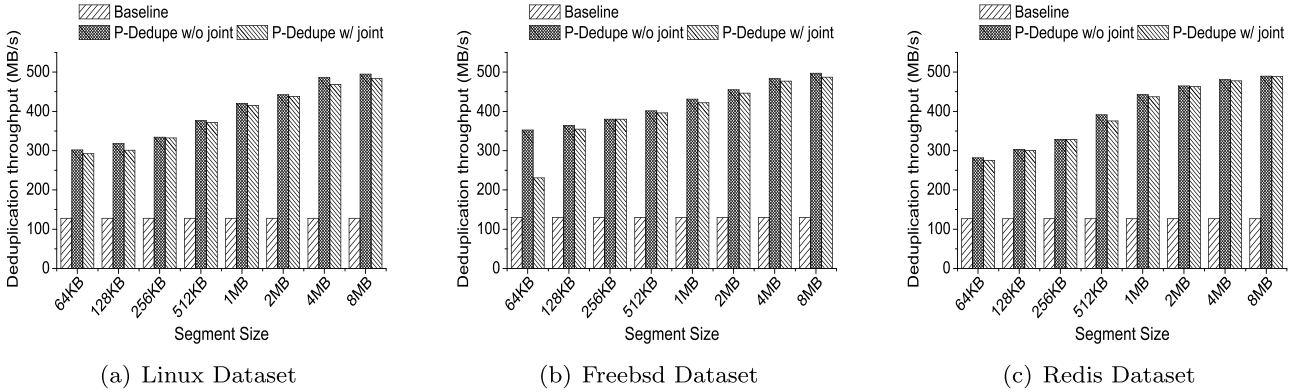

Fig. 10. Deduplication throughput as a function of different parallel data segment size among the approaches of Baseline, P-Dedupe with and without segment boundary jointing on the three typical datasets.

- **Centos**, **Fedora**, and **Freebsd** datasets are VM images of three well-known OS release versions from the same website [52]. These three datasets represent workloads of hosting virtual machine images.
- **RDB** dataset is collected from a running Redis database [53]. The dump.rdb files are backed up as the snapshots of Redis database. Finally, 20 versions are backed up to represent a typical deduplication workload of the database datasets.
- **Bench** refers to a benchmark dataset that is collected from a personal cloud storage benchmark [54]. The frequently used filesystem operations of creating, deleting, and modifying are simulated (as suggested by Tarasov et al. [55]) on the snapshots of this benchmark to get 20 backups.

Table 2
Workload characteristics of eight datasets used in our evaluation, the Dedupe ratio is calculated by the sequential CDC based deduplication at the average chunk size of 8 KB.

Datasets	Versions/images	Size	Dedupe ratio
SciLab	15	5.92 GB	56.04%
GCC	43	15.9 GB	40.75%
Linux	39	16.4 GB	40.92%
Centos	23	40.8 GB	50.02%
Fedora	18	46.9 GB	33.97%
Freebsd	24	27.1 GB	24.05%
Redis	30	162 GB	91.74%
Bench	20	107 GB	86.92%

4.2. Sensitivity study of P-Dedupe

This subsection presents evaluation of P-Dedupe with several important design factors, such as the data segment size (i.e., the parallel chunking unit) and the average chunk size. Since the segment boundary jointing is also important for the deduplication ratio in the P-Dedupe system as discussed in Section 3.3, we evaluate both the approaches of P-Dedupe with and without segment boundary jointing (P-Dedupe w/ joint and w/o joint for short). Thus P-dedupe w/o joint is simple and easy to implement,and is similar to the parallel CDC approach proposed in Shredder [15]. The results shown in Figs. 9, 10, 11 are all tested with 8 KB avg. chunk size for CDC.

Parallel Data Segment Size. Fig. 9 shows the deduplication ratio of P-Dedupe that is normalized to the Baseline approach (i.e., the traditional serial deduplication) with different data segment sizes. As shown in Fig. 9, the deduplication ratio of P-Dedupe w/o joint increases as a function of the increase of data segment size. This is because the events of *Cases 2* and *4* can be reduced by increasing the data segment size as discussed in Section 3.3. Fig. 9 also shows the segment boundary jointing algorithm significantly improves the deduplication ratio of P-Dedupe, which achieves nearly the same deduplication ratio of the Baseline approach. Note that the segment size for parallel chunking in deduplication system can be configured by users accordingly. The detailed considerations about the segment size include thread numbers, file sizes, the data container size (i.e., the storage unit) [23], the rewriting scheme (for better restore performance) [3].

Fig. 10 shows the deduplication throughput of Baseline and P-Dedupe approaches with different data segment sizes. Deduplication throughput of Baseline remains the same (about 130 MB/s

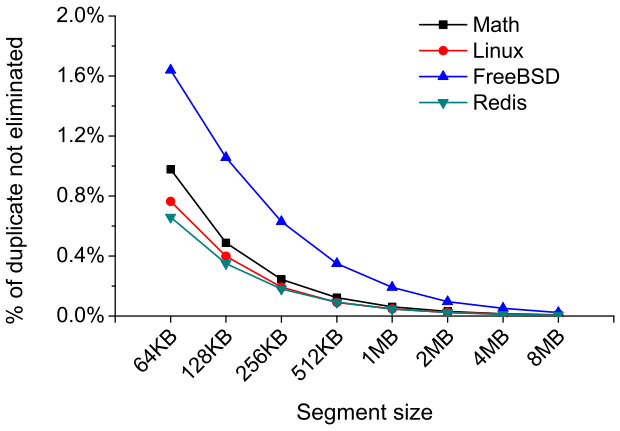

Fig. 11. Percentage of duplicate not eliminated by P-Dedupe (with segment boundary jointing)

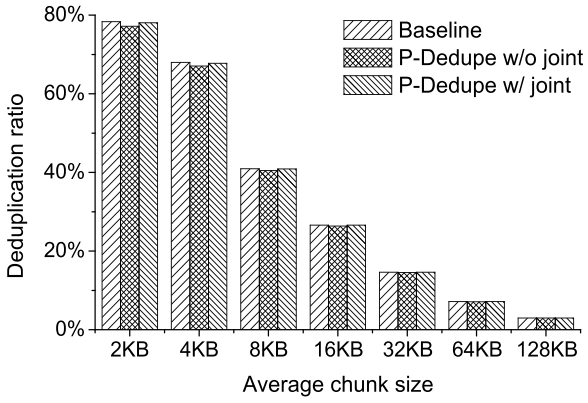

Fig. 12. Deduplication ratio of the P-Dedupe and Baseline approaches as a function of the average chunk size.

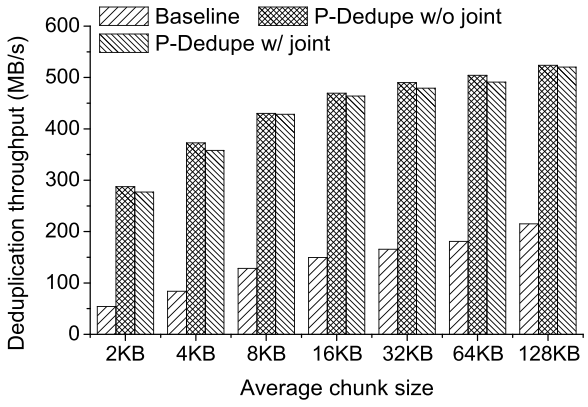

Fig. 13. Deduplication throughput of the P-Dedupe and Baseline approaches as a function of the average chunk size.

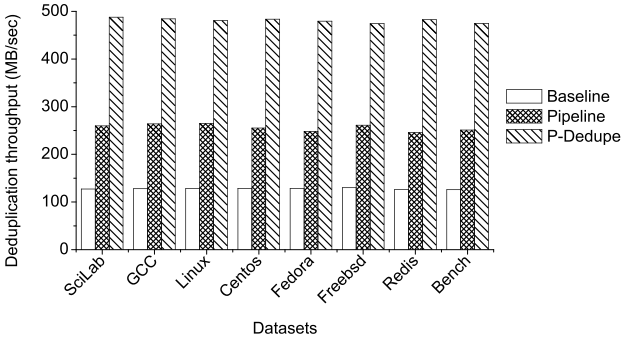

Fig. 14. Deduplication throughput of the approaches of Baseline, Pipeline, and P-Dedupe.

Table 3
Deduplication ratio of P-Dedupe under eight datasets.

Dataset	Baseline	P-Dedupe w/o joint	P-Dedupe w/ joint	P-Dedupe missed
SciLab	56.04%	55.67%	56.01%	0.0365%
GCC	40.75%	40.54%	40.74%	0.0083%
Linux	40.92%	40.80%	40.92%	0.0054%
Centos	50.02%	49.64%	49.97%	0.0481%
Fedora	33.97%	33.70%	33.95%	0.0187%
Freebsd	24.05%	23.89%	24.04%	0.0123%
Redis	91.75%	91.33%	91.74%	0.0102%
Bench	86.92%	86.53%	86.90%	0.0136%

on average) while that of P-Dedupe increases in proportion to the size of the data segment. This is because the larger the segment size is, the more parallelism can be exploited for chunking and fingerprinting. This figure also shows P-Dedupe with the segment boundary jointing achieves nearly the same deduplication throughput as that without boundary jointing.

Fig. 11 further studies the relationship between deduplication ratio and parallel segment size. Here the "Math" curve denotes the estimated duplicate data not eliminated by P-Dedupe according to Eq. (2) in Section 3.3. This figure shows that the deduplication-ratio curves of the Linux and Redis datasets are nearly identical to the curve of "Math". But the FreeBSD dataset detects less duplicates than the estimated value of "Math". This is because Eq. (2) assumes that the data content for content-defined chunking is random (it follows the uniform distribution), while virtual machine images contains a large amount of uninitialized content (such as zero blocks [31]), which results in generating many chunks of the configured maximal chunk size after CDC. However, the trend shown in Fig. 11 the three real-world datasets is consistent with our analysis of Eq. (2). P-Dedupe achieves nearly the same deduplication ratio of the Baseline approach while greatly parallelizing the tasks of CDC and fingerprinting for data deduplication.

Average Chunk Size. Figs. 12 and 13 show the deduplication performance of Baseline, P-Dedupe w/o joint, and P-Dedupe w/ joint using different average chunk sizes and parallel segment size of 1 MB. These two figures suggest that P-Dedupe achieves nearly the same deduplication ratio and about 3–4 times higher deduplication throughput than the Baseline approach regardless of the average chunk size. In addition, the deduplication ratio decreases with the average chunk size, which is generally consistent with the previous studies [7,9]. But the larger the average chunk size is, the higher the deduplication throughput of both the Baseline and P-Dedupe approaches is. This is because that the smaller chunk translates into a larger number of chunks, thus consuming more time for the operations of chunking, fingerprinting, indexing, etc.

Table 4
Illustration of three CDC approaches: Rabin [2], Adler [21], and Gear [22]. Here '*a*' and '*b*' refer to the value of the first and last byte in the sliding window respectively, '*N*' refers to the length of the sliding window for CDC, and 'U', '*T*', '*A*', '*G*' refer to the predefined arrays for each CDC approaches [2,21,22].

Name	Pseudocode
Rabin	$hash = ((hash \hat{} U(a)) \ll 8) \vert b \hat{} T[hash \gg N]$
Adler	$S_1 += A(b); S_2 += S_1; hash = (S_2 \ll 16) \vert S_1;$
Gear	$hash = (hash \ll 1) + G(b)$

4.3. Deduplication performance of P-Dedupe

This subsection evaluates the overall performance of P-Dedupe for the eight datasets. We configure P-Dedupe with average chunk size of 8 KB and data segment size of 4 MB for higher deduplication ratio.

Table 3 shows deduplication ratio achieved by the approaches of Baseline, P-Dedupe w/ joint and P-Dedupe w/o joint. Actually, the percentage of duplicates not detected (missed) by P-Dedupe with the segment boundary jointing is only 0.05% less than the Baseline approach, which means that P-Dedupe achieves nearly the same deduplication ratio as Baseline. Meanwhile, P-Dedupe runs about 4 times faster than the Baseline approach, about 2 times faster than the Pipeline approach as shown in Fig. 14, where "Pipeline" denotes the approach of pipelining the four stages of deduplication as shown in Fig. 5. Note that the Pipeline approach obtains the same chunking effect as Baseline thus deduplication ratio results of the former are omitted in Table 3. In all, P-Dedupe achieves about 99.98% deduplication ratio of the Baseline approach for all eight datasets while maximally parallelizing CDC and fingerprinting for data deduplication.

4.4. Other chunking algorithms

This subsection evaluates the deduplication performance of the P-Dedupe approach combined with the Adler- and Gear-based chunking algorithms. As two known rolling hash algorithms, Adler [21] and Gear [22] also provide the robust chunking efficiency in CDC based deduplication system, and can be used as alternatives for Rabin in CDC-based deduplication. To better illustrate their differences, Table 4 shows the pseudocode implementation of the three CDC approaches, namely, Rabin, Adler, and Gear. They all compute the rolling hash values in an incremental manner but with different hashing schemes.

In order to comprehensively evaluate the efficiency of P-Dedupe's pipelining and parallelizing scheme, Adler- and Gear-based chunking schemes are tested to replace Rabin-based chunking in the P-Dedupe system. Here, we also configure the P-Dedupe and Baseline approaches with average chunk size of 8 KB and data segment size of 4 MB. Note that since the Gear-based chunking runs much faster than SHA-1 based fingerprinting, we configure chunking and fingerprinting with 3 and 5 threads respectively in Gear-based P-Dedupe system. For Adler-based P-Dedupe system, we still parallelize chunking and fingerprinting with both 4 threads (the same as the above Rabin-based P-Dedupe).

Tables 5 and 6 show that the deduplication ratio and throughput of P-Dedupe with Adler- and Gear-based chunking respectively. These two tables suggest that the Baseline approaches with Adler- and Gear-based chunking achieve the similar deduplication ratio of that with Rabin-based chunking, and the P-Dedupe approach also achieve the same deduplication ratio of the Baseline approach. Meanwhile, P-Dedupe achieves about 3∼4 times higher deduplication throughput than the Baseline approach by its pipeline and parallel scheme. Therefore, the results shown in Tables 5 and 6 demonstrate that P-Dedupe well parallelizes content-defined chunking for deduplication without sacrificing the deduplication ratio no matter which CDC approach is actually used.

Table 5
Deduplication performance of P-Dedupe with Adler-based chunking.

Dataset	Deduplication ratio		Deduplication throughput	
	Baseline	P-Dedupe	Baseline	P-Dedupe
SciLab	56.67%	56.63%	133 MB/s	459 MB/s
GCC	42.81%	42.80%	129 MB/s	500 MB/s
Linux	41.39%	41.38%	131 MB/s	491 MB/s
Centos	50.28%	50.24%	134 MB/s	506 MB/s
Fedora	34.16%	34.13%	133 MB/s	504 MB/s
Freebsd	24.45%	24.44%	135 MB/s	503 MB/s
Redis	91.76%	91.75%	130 MB/s	497 MB/s
Bench	86.93%	86.93%	130 MB/s	496 MB/s

Table 6
Deduplication performance of P-Dedupe with Gear-based chunking.

Dataset	Deduplication ratio		Deduplication throughput	
	Baseline	P-Dedupe	Baseline	P-Dedupe
SciLab	56.32%	56.25%	172 MB/s	586 MB/s
GCC	43.21%	43.15%	162 MB/s	590 MB/s
Linux	41.67%	41.67%	168 MB/s	585 MB/s
Centos	50.40%	50.29%	172 MB/s	588 MB/s
Fedora	34.34%	34.27%	167 MB/s	603 MB/s
Freebsd	26.75%	26.72%	172 MB/s	583 MB/s
Redis	91.79%	91.76%	168 MB/s	612 MB/s
Bench	86.92%	86.89%	169 MB/s	594 MB/s

4.5. Our research contributions and comparison with other work

We sum up our research contributions as follows. First, we present the P-Dedupe, a parallel deduplication-based storage system, to accelerate the CDC and fingerprinting computation process. We demonstrate the theories, design and implementation to justify our pioneering approach. Second, we test different large datasets on P-Dedupe. Results of performance evaluation show that deduplication ratio can be maintained high and the execution time can be reduced due to P-Dedupe's acceleration. Compared to one recent work [56], the deduplication can be used combining with encryption algorithms to ensure the original authenticity of the files. In our case, deduplication process can ensure no duplicate data files exist while performing experiments for large datasets, thus increasing the system efficiency. In addition, we argue that the work of P-Dedupe provides an important step when we work on machine learning, and big data analytic science, since we only process, study, and analyze the same file once, rather than multiple times.

5. Conclusion and future work

To alleviate the hash computation bottleneck in data deduplication systems, we present P-Dedupe that fully exploits parallelism of the CDC-based deduplication tasks in deduplication systems. P-Dedupe first pipelines the four stages of the deduplication tasks with the processing units of chunks and files, and then further parallelizes content-defined chunking and secure-hash-based fingerprinting to maximally alleviate the hash computation bottleneck for deduplication. *To ensure the chunking effectiveness of parallelizing CDC*, P-Dedupe first split the data stream into several segments, each segment will be running CDC in parallel with an independent thread, and then uses a segment-boundary jointing approach to quickly re-chunk and joint the two chunks at

the boundaries of the adjacent segments. In addition, we analyze and demonstrate in-depth on the deduplication efficiency of P-Dedupe running the classic CDC with the requirements of both maximal and minimal chunk sizes. Evaluation results suggest that P-Dedupe effectively parallelizes the content-defined chunking for deduplication only at the cost of slightly decreasing the deduplication ratio while accelerating the deduplication throughput nearly linearly with the number of CPU cores.

As P-Dedupe's idea of pipelining deduplication and then further parallelizing chunking & fingerprinting is also applied to deduplication systems with the GPGPU devices. We plan to study P-Dedupe with more thread-level parallelism on GPGPU devices and solve more potential challenges for P-Dedupe due to much higher processing throughput in the GPGPU devices.

Acknowledgments

We are grateful to the anonymous reviewers for their insightful comments and feedback on this work. This research was partly supported by National Key Research and Development Program of China under Grant 2017YFB0802204, NSFC, China, No.61821003, No.61502190, No.U1705261, No.61832007, No.61772222, No.61772180 and No.61672010; The Scientific Research Fund of Hubei Provincial Department of Education, China, B2017042; US NSF under Grants CCF-1704504 and CCF-1629625. The preliminary manuscript is published in the proceedings of IEEE International Conference on Networking, Architecture, and Storage (IEEE NAS), 2012.

References

[1] The digital universe of opportunities: Rich data and the increasing value of the internet of things, 2014, http://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm, EMC Digital Universe with Research & Analysis by IDC.
[2] A. Muthitacharoen, B. Chen, D. Mazieres, A low-bandwidth network file system, in: Proceedings of the ACM Symposium on Operating Systems Principles, SOSP'01, ACM Association, Banff, Canada, 2001, pp. 1–14.
[3] W. Xia, H. Jiang, D. Feng, F. Douglis, P. Shilane, Y. Hua, M. Fu, Y. Zhang, Y. Zhou, A comprehensive study of the past, present, and future of data deduplication, Proc. IEEE 104 (9) (2016) 1681–1710.
[4] R.N.S. Widodo, H. Lim, M. Atiquzzaman, SDM: Smart deduplication for mobile cloud storage, Future Gener. Comput. Syst. 70 (2016) S0167739X16302084.
[5] Y. Zhou, Y. Deng, L.T. Yang, R. Yang, L. Si, LDFS: A low latency in-line data deduplication file system, IEEE Access PP (99) (2018) 1–1.
[6] S. Quinlan, S. Dorward, Venti: A new approach to archival storage, in: Proceedings of USENIX Conference on File and Storage Technologies, FAST'02, USENIX Association, Monterey, CA, USA, 2002, pp. 89–101.
[7] D. Meyer, W. Bolosky, A study of practical deduplication, in: Proceedings of the USENIX Conference on File and Storage Technologies, USENIX'11, USENIX Association, San Jose, CA, USA, 2011, pp. 229–241.
[8] A. El-Shimi, R. Kalach, A. Kumar, et al., Primary data deduplication-large scale study and system design, in: Proceedings of the 2012 USENIX Conference on USENIX Annual Technical Conference, USENIX Association, Boston, MA, USA, 2012, pp. 285–296.
[9] G. Wallace, F. Douglis, H. Qian, et al., Characteristics of backup workloads in production systems, in: Proceedings of the Tenth USENIX Conference on File and Storage Technologies, FAST'12, USENIX Association, San Jose, CA, 2012, pp. 1–14.
[10] P. Shilane, M. Huang, G. Wallace, 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, USENIX Association, San Jose, CA, USA, 2012, pp. 49–64.
[11] F. Chen, T. Luo, X. Zhang, CAFTL: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives, in: Proceedings of the 9th USENIX Conference on File and Storage Technologies, FAST'11, USENIX Association, San Jose, CA, USA, 2011, pp. 1–14.
[12] A. Gupta, R. Pisolkar, B. Urgaonkar, A. Sivasubramaniam, Leveraging value locality in optimizing NAND flash-based SSDs, in: Proceedings of the 9th USENIX Conference on File and Storage Technologies, FAST'11, USENIX Association, San Jose, CA, USA, 2011, pp. 91–103.
[13] A. Gharaibeh, S. Al-Kiswany, S. Gopalakrishnan, et al., A GPU accelerated storage system, in: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC'10, ACM Association, Chicago, Illinois, USA, 2010, pp. 167–178.
[14] ZFS-Deduplication, http://bit.ly/p6l7LC.
[15] P. Bhatotia, R. Rodrigues, A. Verma, Shredder: GPU-accelerated incremental storage and computation, in: Proceedings of the 10th USENIX Conference on File and Storage Technologies, FAST'12, USENIX Association, San Jose, CA, USA, 2012, pp. 1–15.
[16] W. Xia, H. Jiang, D. Feng, L. Tian, 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, USENIX Association, San Jose, CA, USA, 2012, pp. 1–2.
[17] M. Rabin, Fingerprinting by Random Polynomials, Center for Research in Computing Techn., Aiken Computation Laboratory, Univ., 1981.
[18] J. Bowling, Opendedup: Open-source deduplication put to the test, Linux J. 2013 (228) (2013) 2.
[19] S. Al-Kiswany, A. Gharaibeh, E. Santos-Neto, G. Yuan, M. Ripeanu, StoreGPU: Exploiting graphics processing units to accelerate distributed storage systems, in: Proceedings of the 17th International Symposium on High Performance Distributed Computing, HPDC'08, ACM Association, Boston, MA, USA, 2008, pp. 165–174.
[20] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, C. Kozyrakis, Evaluating mapreduce for multi-core and multiprocessor systems, in: Proceedings of IEEE 13th International Symposium on High Performance Computer Architecture, HPCA'07, IEEE, 2007, pp. 13–24.
[21] C. Dubnicki, E. Kruus, K. Lichota, C. Ungureanu, Methods and systems for data management using multiple selection criteria, Google Patents, Dec 1 2006, US Patent App. 11/566, 122.
[22] W. Xia, H. Jiang, D. Feng, L. Tian, M. Fu, Y. Zhou, Ddelta: A deduplication-inspired fast delta compression approach, Perform. Eval. 79 (2014) 258–272.
[23] B. Zhu, K. Li, R. Patterson, Avoiding the disk bottleneck in the data domain deduplication file system, in: Proceedings of the 6th USENIX Conference on File and Storage Technologies, vol. 8, FAST'08, USENIX Association, San Jose, CA, USA, 2008, pp. 1–14.
[24] D. Meister, J. Kaiser, A. Brinkmann, 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, IEEE Computer Society Press, Salt Lake City, Utah, USA, 2012, pp. 1–11.
[25] B. Debnath, S. Sengupta, J. Li, ChunkStash: Speeding up inline storage deduplication using flash memory, in: Proceedings of the 2010 USENIX Annual Technical Conference, USENIX'10, USENIX Association, Boston, MA, USA, 2010, pp. 1–14.
[26] M. Lillibridge, K. Eshghi, D. Bhagwat, 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, USENIX Association, San Jose, CA, 2009, pp. 111–123.
[27] R. Koller, R. Rangaswami, I/O deduplication: Utilizing content similarity to improve I/O performance, ACM Trans. Storage 6 (3) (2010) 1–26.
[28] A. Clements, I. Ahmad, M. Vilayannur, J. Li, Decentralized deduplication in SAN cluster file systems, in: Proceedings of the 2009 Conference on USENIX Annual Technical Conference, USENIX Association, 2009, p. 8.
[29] V. Michael, S. Stefan, M. Geoffrey, Cumulus: Filesystem backup to the cloud, in: Proceedings of 7th USENIX Conference on File and Storage Technologies, FAST'09, 2009.
[30] Y. Deng, X. Huang, L. Song, Y. Zhou, F. Wang, Memory deduplication: An effective approach to improve the memory system, J. Inf. Sci. Eng. 33 (5) (2017) 1103–1120.
[31] K. Jin, E.L. Miller, The effectiveness of deduplication on virtual machine disk images, in: Proceedings of SYSTOR'09: The Israeli Experimental Systems Conference, ACM Association, Haifa, Israel, 2009, pp. 1–14.
[32] V. Tarasov, D. Jain, G. Kuenning, S. Mandal, K. Palanisami, P. Shilane, S. Trehan, E. Zadok, Dmdedup: Device Mapper Target for Data Deduplication.
[33] C.-H. Ng, M. Ma, T.-Y. Wong, P.P. Lee, J. Lui, Live deduplication storage of virtual machine images in an open-source cloud, in: Proceedings of the 12th International Middleware Conference, International Federation for Information Processing, 2011, pp. 80–99.
[34] C. Policroniades, I. Pratt, Alternatives for detecting redundancy in storage systems data, in: Proceedings of USENIX Annual Technical Conference, General Track, USENIX'04, USENIX Association, Boston, MA, USA, 2004, pp. 73–86.
[35] D. Meister, A. Brinkmann, dedupv1: Improving deduplication throughput using solid state drives (SSD), in: Proceedings of IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST, IEEE Computer Society Press, Incline Village, Nevada, USA, 2010, pp. 1–6.
[36] E. Kruus, C. Ungureanu, C. Dubnicki, Bimodal content defined chunking for backup streams, in: Proceedings of the 7th USENIX Conference on File and Storage Technologies, USENIX Association, San Jose, CA, USA, 2010, pp. 1–14.

[37] F. Guo, P. Efstathopoulos, Building a high-performance deduplication system, in: Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference, USENIX'11, USENIX Association, Portland, OR, USA, 2011, pp. 271–284.
[38] J. Ma, B. Zhao, G. Wang, J. Liu, Adaptive pipeline for deduplication, in: Proceedings of the 28th IEEE Symposium on Mass Storage Systems and Technologies, MSST'12, IEEE Computer Society Press, Pacific Grove, CA, USA, 2012, pp. 1–6.
[39] C. Liu, Y. Xue, D. Ju, D. Wang, A novel optimization method to improve de-duplication storage system performance, in: Proceedings of the 15th International Conference on Parallel and Distributed Systems, ICPADS'09, IEEE, 2009, pp. 228–235.
[40] J. Min, D. Yoon, Y. Won, Efficient deduplication techniques for modern backup operation, IEEE Trans. Comput. 60 (6) (2011) 824–840.
[41] B. Romański, Ł. Heldt, W. Kilian, K. Lichota, C. Dubnicki, Anchor-driven subchunk deduplication, in: Proceedings of the 4th Annual International Systems and Storage Conference, SYSTOR'11, ACM Association, Haifa, Israel, 2011, pp. 1–13.
[42] W. Xia, H. Jiang, D. Feng, Y. Hua, 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, USENIX Association, Portland, OR, USA, 2011, pp. 285–298.
[43] J. Wei, H. Jiang, K. Zhou, et al., MAD2: A scalable high-throughput exact deduplication approach for network backup services, in: Proceedings of 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST'10, IEEE Computer Society Press, Incline Village, Nevada, USA, 2010, pp. 1–14.
[44] D. Meister, A. Brinkmann, T. Süß, File recipe compression in data deduplication systems, in: Proceedings of the 11th USENIX Conference on File and Storage Technologies, FAST'13, USENIX Association, San Jose, CA, USA, 2013, pp. 183–197.
[45] Y.-K. Li, M. Xu, C.-H. Ng, P.P. Lee, Efficient hybrid inline and out-of-line deduplication for backup storage, ACM Trans. Storage 11 (1) (2014) 2.
[46] M. Fu, D. Feng, Y. Hua, X. He, Z. Chen, W. Xia, F. Huang, Q. Liu, Accelerating restore and garbage collection in deduplication-based backup systems via exploiting historical information, in: Proceedings of the 2014 USENIX Annual Technical Conference, USENIX ATC 14, USENIX Association, Philadelphia, PA, USA, 2014, pp. 181–192.
[47] W. Xia, H. Jiang, D. Feng, L. Tian, M. Fu, Z. Wang, P-dedupe: Exploiting parallelism in data deduplication system, in: Proceedings of the 7th International Conference OnNetworking, Architecture and Storage, NAS'12, IEEE Computer Society Press, Xiamen, China, 2012, pp. 338–347.
[48] Y. Ling, T. Mullen, X. Lin, Analysis of optimal thread pool size, ACM SIGOPS Oper. Syst. Rev. 34 (2) (2000) 42–55.
[49] SciLab archive, ftp://ftp.kernel.org.
[50] GNU software archive, http://ftp.gnu.org/gnu/.
[51] Linux archive, ftp://ftp.kernel.org.
[52] VMs Archives, http://www.thoughtpolice.co.uk/vmware//.
[53] Redis key-value database, http://redis.io/.
[54] I. Drago, E. Bocchi, M. Mellia, H. Slatman, A. Pras, Benchmarking personal cloud storage, in: Proceedings of the 2013 Conference on Internet Measurement Conference, IMC'13, ACM Association, Barcelona, Spain, 2013, pp. 205–212.
[55] V. Tarasov, A. Mudrankit, W. Buik, P. Shilane, G. Kuenning, E. Zadok, Generating realistic datasets for deduplication analysis, in: Proceedings of the 2012 USENIX Conference on Annual Technical Conference, USENIX'12, USENIX Association, Boston, MA, USA, 2012, pp. 261–272.
[56] V. Chang, M. Ramachandran, Towards achieving data security with the cloud computing adoption framework, IEEE Trans. Serv. Comput. 9 (1) (2016) 138–151.

Wen Xia received the Ph.D. degree in computer science from Huazhong University of Science and Technology (HUST), Wuhan, China, in 2014. He is currently an associate professor in the school of computer science and technology at Harbin Institute of Technology, Shenzhen. His research interests include data reduction, storage systems, cloud storage, etc. He has published more than 30 papers in major journals and international conferences including FGCS, IEEE-TC, IEEE-TPDS, USENIX ATC, USENIX FAST, INFOCOM, IFIP Performance, IEEE DCC, MSST, IPDPS, HotStorage, etc. He is a member of ACM, CCF, IEEE, and USENIX.

Dan Feng received the B.E., M.E., and Ph.D. degrees in Computer Science and Technology in 1991, 1994, and 1997, respectively, from Huazhong University of Science and Technology (HUST), China. She is 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 international conferences, including IEEE-TC, IEEE-TPDS, ACM-TOS, JCST, FAST, USENIX ATC, ICDCS, HPDC, SC, ICS, IPDPS, and ICPP. She has served as the program committees of multiple international conferences, including SC 2011, 2013 and MSST 2012, 2015. She is a member of IEEE and a member of ACM.

Hong Jiang received the B.Sc. degree in Computer Engineering in 1982 from Huazhong University of Science and Technology, Wuhan, China; the M.A.Sc. degree in Computer Engineering in 1987 from the University of Toronto, Toronto, Canada; and the Ph.D. degree in Computer Science in 1991 from the Texas A&M University, College Station, Texas, USA. He is currently Chair and Wendell H. Nedderman Endowed Professor of Computer Science and Engineering Department at the University of Texas at Arlington. Prior to joining UTA, he served as a Program Director at National Science Foundation (2013.1–2015.8) and he was at University of Nebraska-Lincoln since 1991, where he was Willa Cather Professor of Computer Science and Engineering. He has graduated 13 Ph.D. students who upon their graduations either landed academic tenure-track positions in Ph.D.-granting US institutions or were employed by major US IT corporations. 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 an Associate Editor of the IEEE Transactions on Parallel and Distributed Systems. He has over 200 publications in major journals and international Conferences in these areas, including IEEE-TPDS, IEEE-TC, Proceedings of the IEEE, ACM-TACO, JPDC, ISCA, MICRO, USENIX ATC, FAST, EUROSYS, LISA, SIGMETRICS, ICDCS, IPDPS, MIDDLEWARE, OOPLAS, ECOOP, SC, ICS, HPDC, INFOCOM, ICPP, etc., and his research has been supported by NSF, DOD, the State of Texas and the State of Nebraska. Dr. Jiang is a Fellow of IEEE, and Member of ACM.

Yucheng Zhang received the Ph.D. degree in computer science from Huazhong University of Science and Technology (HUST), Wuhan, China, in 2018. He is currently an assistant professor in the school of computer science and technology at 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-TC, INFOCOM, FAST, USENIX ATC, etc.

Victor Chang is currently a Senior Associate Professor (Reader), Director of Ph.D. (June 2016 C May 2018), Director of MRes and Interim Director of IMIS at International Business School Suzhou (IBSS), Xian Jiaotong-Liverpool University (XJTLU), Suzhou, China, since June 2016. He is also a very active and contributing key member at Research Institute of Big Data Analytics (RIBDA), XJTLU. He is an Honorary Associate Professor at the University of Liverpool and Visiting Researcher at the University of Southampton, UK. Previously he worked as a Senior Lecturer at Leeds Beckett University, UK, for 3.5 years. Within 4 years, he completed Ph.D. (CS, Southampton) and PGCert (Higher Education, Fellow, Greenwich) while working for several projects at the same time. Before becoming an academic, he has achieved 97% on average in 27 IT certifications. He won a European Award on Cloud Migration in 2011, IEEE Outstanding Service Award in 2015, best papers in 2012 and 2015, the 2016 European award: Best Project in Research, 2016 SEID Excellent Scholar, Suzhou, China, Outstanding Young Scientist award in 2017, 2017 special award on Data Science, 2017 and 2018 INSTICC Service Awards and numerous awards since 2012. He is a visiting scholar/Ph.D. examiner at several universities, an

Editor-in-Chief of IJOCI & OJBD journals, Editor of FGCS, Associate Editor of TII, founding chair of two international workshops and founding Conference Chair of IoTBDS http://www.iotbd.org and COMPLEXIS http://www.complexis.org since Year 2016. He was involved in different projects worth more than €12.5 million in Europe and Asia. He has published 3 books as sole authors and the editor of 2 books on Cloud Computing and related technologies. He gave 17 keynotes at international conferences. He is widely regarded as one of the most active and influential young scientist and expert in IoT/Data Science/Cloud/security/AI/IS, as he has experience to develop 10 different services for multiple disciplines. He is founding conference chair for IoTBDS, COMPLEXIS and FEMIB to build up and foster active research communities globally.

Xiangyu Zou is currently a visiting master student majoring in computer science at Harbin Institute of Technology, Shenzhen, China. His research interests include data reduction, distributed systems, etc.