Does the Content Defined Chunking Really Solve the Local Boundary Shift Problem?

Wenlong Tian [1,4], Ruixuan Li [1], Zhiyong Xu [2,3], and Weijun Xiao [4]

[1]School of Computer Science and Technology, Huazhong University of Science and Technology, China
[2]Math and Computer Science Department, Suffolk University, USA
[3]Shenzhen Institute of Advanced Technology, Chinese Academy of Science, China
[4]Electrical and Computer Engineering, Virginia Commonwealth University, USA
Email: wenlongtian@hust.edu.cn; rxli@hust.edu.cn; zxu@suffolk.edu; wxiao@vcu.edu

***Abstract*—Data chunking is one of the most important issues in a deduplication system, which not only determines the effectiveness of deduplication such as deduplication ratio, but also impacts the modification overhead. It breaks the file into chunks to find out the redundancy by fingerprint comparisons. The content-defined chunking algorithms such as TTTD, BSW CDC, and RC, can resist the boundary shift problem caused by small modifications. However, we observe that there exist a lot of consecutive maximum chunk sequences in various benchmarks. These consecutive maximum chunk sequences will lead to local boundary shift problem when facing small modifications. Based on this observation, we propose a new chunking algorithm, Elastic Chunking. By leveraging dynamic adjustment policy, elastic chunk can quickly find the boundary to remove the consecutive maximum chunk sequences. To evaluate the performance, we implement a prototype and conduct extensive experiments based on synthetic and realistic datasets. Compared with TTTD, BSW CDC and RC algorithms, proposed chunking algorithm can achieve the higher deduplication ratio and throughput.**

Keywords: Deduplication; Chunking Algorithm; Boundary Shift; Content-Defined

I. Introduction

Data Deduplication is an important technique to improve storage utilization and save the network bandwidth by significantly removing the redundant data in cloud storage scenario [1, 2]. There has been many research done in the literature, such as [3–5], for improving the performance and efficiency of deduplication. In a deduplication system, removing the duplicated data is based on the comparison of data's fingerprints. If matched, it can be removed as duplicated data, vice versa. As the core component of deduplication, chunking algorithm has been extensively studied in recent years.

The basic idea of chunking algorithm is to break large files into small chunks for detecting data redundancy. As an initial step, fixed-size chunking algorithm is proposed to quickly split the files into chunks. In fixed-size chunking algorithm, the size of each chunk is identical. It can quickly complete the fixed-size chunking process. However, it is vulnerable by modifications that easily lead to the boundary shift problem. A small modification causes a lot of unique chunks. Therefore, fixed-size chunking algorithm is not suitable for high deduplication ratio datasets.

To solve the boundary shift problem, content-defined chunking algorithms are proposed to make the chunk boundary related with the content. When small modifications occur, it does not impact the decision of chunk boundary. Hence, small modifications do not lead to the global boundary shift as fixed-size chunking algorithm. As a classical content-defined chunking algorithm, Muthitacharoen et al. propose a sliding window(BSW) based content-defined chunking algorithm [6]. The decision of chunk boundary is determined by comparing with Rabin-based fingerprints [7] in a sliding window with the threshold. If the comparison result is satisfied with predefined condition, the location of sliding window will be chunked as a boundary. Otherwise, the window on files slides to the next location. Once no chunk boundary is found, the parameter of maximum chunk size is applied to generate a forced chunk boundary.

Even though the BSW chunking algorithm prevents the boundary shift problem compared with fixed-size chunking algorithm to some extent, it performs poorly on real datasets [8] because of the impaction from data's modification. In other words, it is still possible to decrease the modification impact by observing the substantial variance of chunk sizes. Based on quantitative analysis for previous content-defined chunking algorithms, TTTD suggests that avoiding too small or too large chunk size can reduce the modification overhead by theoretical analysis [8]. Based on this scheme, RC further argues that maximum length chunks will impact deduplication ratio since it is not content defined [9]. The core idea of RC is to decrease the maximum length chunks by multi-stage divisor values.

Although RC undoubtedly decreases the number of maximum chunks, the cost of chunking process is increased by multi-loop iterations to eliminate maximum chunks. What's worse, it is inaccurate to eliminate whole max-length chunks since there still have redundancy among maximum length chunks. Taking extra process to eliminate these kind of maximum length chunks obviously leads to unnecessary overhead. However, based on our observation, the main problem for max-length chunks is the local boundary shift problem that will be discussed in the next section. Simply decreasing the probability of maximum length chunks can not effectively resist the local boundary shift problem. More detailed analysis in Section II.

Yu et al. propose a leap-based CDC algorithm to improve the deduplication throughput by skipping the unnecessary boundary decision computation [10]. It replaces the Rabin-based hash method by the Pseudo-random Transformation to support its leaping property which does not belong to Rabin-based category. Moreover, it also does not outperform TTTD deduplication ratios. Note that we only focus on the Rabin-based CDC algorithms in this paper which finishes the chunking process without interacting with server. In other words, chunking algorithm breaks the files into chunks without any help from the server. The content-defined chunking algorithms based on Rabin fingerprints include Basic Sliding Window(BSW) CDC [6], TTTD[8], and RC [9].

As we describe in more detail in Section II, the Rabin-based CDC algorithms do not consider the local boundary shift problem, since the probability to generate maximum length chunks during the chunking process can not eliminated. Even though the idea of content-defined chunking algorithm is to resist boundary shift problem, the appearance of maximum length chunks still have a chance to form small consecutive maximum chunk sequence. The

978-1-5090-6468-7/17/$31.00

probability is related with the parameter setting for each algorithm. Thus, the more scatter modifications it has on the sequence, the larger impaction for deduplication ratio it induces in chunking algorithm. More specifically, once one bit is changed in the first maximum length chunk in a consecutive max-length chunk pattern, these small sequences degrade to a local fixed-sized chunking situation. Unfortunately, the consecutive maximum length chunks result in local boundary shift problem that is not handled by existing Rabin-based CDC chunking algorithms.

To investigate local boundary shift problem, we analyze it from both theoretical and experimental aspects. From the theoretical aspect, we demonstrate that the probability of maximum length chunks is changed as the variance of the parameter setting for each algorithm. Since there exist a large number of chunks, it is impossible to ignore maximum length chunks. To further illustrate the local boundary shift problem, we conduct an experimental analysis over a deduplication dataset. It still has a lot of small consecutive maximum length chunk sequences after the chunking process. The situation is drastically aggravated as the increasing number of chunks.

To overcome the local boundary shift problem, we propose a novel chunking algorithm, Elastic Chunking, to dynamically decrease the consecutive maximum chunk sequence. Based on this scheme, comprehensive experiments are performed. Compared with other algorithms, Elastic Chunking algorithm can effectively handle the local boundary shift problem. It can also acquire higher throughput compared with other Rabin-based CDC algorithms. To the best of our knowledge, we are the first to give out detailed theoretical analysis about local boundary shift problem and consider the drawbacks about consecutive max-length chunk pattern in modification situation. Overall, our contributions are as follows:

- First, we analyze the problem of local boundary shift problem caused by consecutive max-length chunk pattern from theoretical and experimental aspects. Local boundary shift problem severely degrade the chunking algorithm and enlarge the impaction by modification in online deduplication scenario.
- Second, we summarize two principles to solve the local boundary shift problem and design a novel chunking algorithm, Elastic chunking, to dynamically eliminate the consecutive maximum chunk sequences. Our design greatly resists the impacts caused by local boundary shift problem.
- Finally, we implement chunking algorithms and conduct comprehensive experiments to evaluate proposed chunking algorithm. Experiment results show that our new chunking algorithm can effectively handle the local boundary shift problem.

The rest of the paper is organized as follows. In Section II, we analyze the local boundary shift problem from theoretical and experimental aspects. In Section III, we present two principles to overcome the local boundary shift problem for designing chunking algorithm and propose our chunking algorithm. Then, in Section IV, we conduct several experiments to evaluate our chunking algorithm. Section V will discuss related work and finally, we conclude the paper and give the future work in Section VI.

II. Local Boundary Shift

To understand the local boundary shift problem in this section, we firstly explain the content-based chunking process and abstract the local boundary shift problem. Then, we further illustrate the problem from both experimental results and theoretical analysis.

A. Description of Local Boundary Shift Problem

During the chunking process, a file, which length is larger than average chunk size, is split into several small chunks. The length of each chunk is not larger than average chunk size. In Fixed-size chunking algorithm, a small modification leads to the whole boundary of each chunk shift to a new place. Therefore, the duplicate parts is treated as a new chunk after modification because of the boundary shift. This issue is called boundary shift problem. Even the existing rabin-based CDC algorithms can solve this problem, we observe that exist a lot of small consecutive maximum length chunk sequences in the chunking process. Moreover, the deduplication ratio after modification has a severe degradation. For example, during chunking process, the chunk results are ck_1,ck_2,$ck_3 \cdots ck_n$, where ck_2 to ck_n are max-length chunks. Once a small modification affect the ck_2's boundary decision, the boundary from ck_2 to ck_n must be changed. Then, the situation is similar to the fixed-size chunking which can be easily impact by small modifications.

TABLE I: consecutive max-length pattern statistics

Consecutive Pattern	Count
≥ 1000	1
$< 1000 \&\& \geq 500$	3
$< 1000 \&\& \geq 500$	93
$< 100 \&\& \geq 10$	3184
$< 10 \&\& \geq 5$	5142
$< 5 \&\& \geq 2$	26606
$= 1$	90807

The reason for local boundary shift problem is the maximum chunk size parameter in CDC chunking algorithm. The original goal of maximum chunk size parameter is to decrease the modification impact for deduplication ratio and in case of large chunk size. However, the existing Rabin-based CDC algorithms can not eliminate the probability of max-length chunk during chunking process, even though different parameters setting in chunking algorithm will impact the probability of max-length chunks, which will be justified shortly. When a maximum length chunk is adjacent to the other new maximum length chunk, it consists a two consecutive max-length chunk sequence. Furthermore, the situation becomes more severe when there exist too many sporadic consecutive max-length chunk sequences.

B. Experimental Observation

To show the existence of consecutive max-length chunk sequences, we take a vmdk file as an original file which size is 4.58 GB. Then, it is split by TTTD chunking algorithm. In Table I, the consecutive pattern denotes the number of maximum length chunk in a sequences. Count means the number of sequence which belongs to the same consecutive pattern. Then, we counted the number of the consecutive max-length chunk sequence in various consecutive pattern conditions. The reason why we choose the vmdk is that this type of file is very common in both our daily life and deduplication system. Table I shows results based on different patterns after the process of vmdk.

As shown in Table I, we can learn that the consecutive max-length pattern phenomenon is very common in realistic datasets. Although the occurrences of consecutive max-length pattern are few when the consecutive pattern larger than 500, there are much smaller consecutive max-length pattern occurs when the pattern less than 100. As we describe more detail in Section IV, we also use the synthetic workloads to simulate the extreme impacts by local boundary shift problem. When considering the local boundary shift problem in BSW Chunking algorithm, the situation is even worse.

C. Theoretical Analysis

Based on above observations, we further explore the local boundary shift problem from theoretical analysis. It is noted that the deduplication ratio for each chunking algorithms in our discussions is no more than using TTTD paradigm which includes RC. Therefore, we choose BSW and TTTD for the theoretical analysis. For BSW chunking algorithm, there are three parameters such as the maximum chunk size L_2, the minimum chunk size L_1, and divisor value D. x is the difference of L_2 and L_1. There is a sliding window which slides from each file's beginning to the end. Then, the CDC chunking algorithm calculates the hash value based on the content of the sliding

TABLE II: Probability Distribution of BSW Chunking Algorithm

	0	1	2	3	$\cdots$	x-2	x-1	x
μ	L_1	L_1+1	L_1+2	L_1+3	$\cdots$	L_2-2	L_2-1	L_2
μ^2	L_1^2	$(L_1+1)^2$	$(L_1+2)^2$	$(L_1+3)^2$	$\cdots$	$(L_2-2)^2$	$(L_2-1)^2$	$(L_2)^2$
p	$\frac{1}{D}$	$(1-\frac{1}{D})\frac{1}{D}$	$(1-\frac{1}{D})^2\frac{1}{D}$	$(1-\frac{1}{D})^3\frac{1}{D}$	$\cdots$	$(1-\frac{1}{D})^{(L_2-L_1-2)}\frac{1}{D}$	$(1-\frac{1}{D})^{(L_2-L_1-1)}\frac{1}{D}$	$(1-\frac{1}{D})^{(L_2-L_1)}$

TABLE III: Probability Distribution of TTTD Chunking Algorithm

	0	1	2	$\cdots$	x-2	x-1	x
μ	L_1	L_1+1	L_1+2	$\cdots$	L_2-2	L_2-1	L_2
μ^2	L_1^2	$(L_1+1)^2$	$(L_1+2)^2$	$\cdots$	$(L_2-2)^2$	$(L_2-1)^2$	$(L_2)^2$
P	p_0	p_1	p_2	$\cdots$	p_{x-2}	p_{x-1}	p_x
P_{D1}	$\frac{1}{D1}$	$(1-\frac{1}{D1})\frac{1}{D1}$	$(1-\frac{1}{D1})^2\frac{1}{D1}$	$\cdots$	$(1-\frac{1}{D1})^{(x-2)}\frac{1}{D1}$	$(1-\frac{1}{D1})^{(x-1)}\frac{1}{D1}$	$(1-\frac{1}{D1})^{(x)}$
P_{D2}	$(1-\frac{1}{D2})^{(x)}$	$(1-\frac{1}{D2})^{(x-1)}\frac{1}{D2}$	$(1-\frac{1}{D2})^{(x-2)}\frac{1}{D2}$	$\cdots$	$(1-\frac{1}{D2})^2\frac{1}{D2}$	$(1-\frac{1}{D2})\frac{1}{D2}$	$\frac{1}{D2}$

window. if the remainder of the hash value and divisor value D is equal to a predefined threshold, the position of sliding window is set as a chunk boundary. Otherwise, the sliding window moves forward one bit. Furthermore, the chunk process repeats the above calculations until finding a suitable boundary. When there is no chunk boundary found until reaching the maximum chunk size position, the position of maximum chunk size is set as a forced boundary. Based on BSW chunking algorithm, we defined the probability of choosing L_1 as the chunk boundary is $\frac{1}{D}$ in the initial step. the probability of L_1+1 as chunk boundary is $(1-\frac{1}{D})*\frac{1}{D}$. As the sliding window moves to the end of file, the probability that L_1+i is set as a chunk boundary is $(1-\frac{1}{D})^i*\frac{1}{D}$. Table II shows the discrete probability distribution for BSW chunking algorithm where x equals the difference of L_2 and L_1.

Similar to BSW CDC chunking algorithm, we also build the probabilistic distribution for the TTTD algorithm. Table III shows the discrete probabilistic distribution of TTTD chunking algorithm that has four parameters. L_1 denotes the minimum chunk size and L_2 is the maximum chunk size, x equals to the difference value of L_1 and L_2. D_1 means the main divisor value of TTTD algorithm and D_2 is the second divisor value. We denote P_i as the probability where L_1+i bit is chosen as a chunk boundary. P_i equals to $P_{D1}+(1-\frac{1}{D_1})^x P_{D2}$ when the chunk is a normal chunk. When the chunk size is maximum length, the probability of P_i is $(1-\frac{1}{D_1})^x\frac{1}{D_2}$.

From Table II and III, the probability of max-length chunk is $(1-\frac{1}{D})^{(L_2-L_1)}$ in BSW CDC chunking algorithm and $(1-\frac{1}{D_1})^x\frac{1}{D_2}$ in TTTD chunking algorithm, respectively. Obviously, the probability of maximum length chunk is related with the chunking algorithm's parameters. For example, the probability of max-length chunk, p_{max}, is about 35.9% in BSW when D is 2000 and x equals to 2048. As for TTTD, when x equals 2048, D_1 is 2000 and D_2 is 1000, p_{max} equals to 3.59%. Hence, p_{max}^n is the probability of consecutive max-length pattern where n means the number of max-length chunks in a consecutive max-length. Since P_{max}^n is less than $P_{max}^{(n-1)}$, the probability of short consecutive maximum chunk sequences must be larger than the long consecutive maximum chunk sequences. However, the probability of short consecutive maximum chunk sequences can not be ignored. More specifically, the count of consecutive max-length pattern sequences is drastically increasing with the augment of total chunk number, which is consistent with our experimental results.

III. Elastic Chunking

In this section, we summarize two principles in designing the chunking algorithm to resist the local boundary shift problem based the theoretical analysis in Section II. Then, we further propose an novel chunking algorithm, elastic chunking algorithm, to dynamically restrain the formalization of consecutive maximum chunk sequence.

A. Design Principles for Resisting Local Boundary Shift

Based on above problem, the purpose of elastic chunking is to solve the local boundary shift problem. Hence, it must restrain the formalization of consecutive maximum length chunk sequence. Moreover, elastic chunking algorithm should also keep the better or the same deduplication ratio and higher or the same throughput compared with other chunking algorithms. It is noted that there is no scheme to prevent the local boundary shift problem in existing Rabin-based CDC algorithm since the parameter of maximum chunk size results in the consecutive maximum chunk sequences. However, the setting of maximum chunk size can not be removed to overcome the local boundary shift problem since it reduces the impaction for deduplication ratio under modification scenario and prevents the large chunk size. Therefore, It is very important that how we can achieve the goal of elastic chunking algorithm without removing the parameter of maximum chunk size. Then, we summarize two principles to solve the local boundary shift problem.

Principle 1: Once too many consecutive max-length chunks occur during the chunking process, the chunking algorithm need to dynamically increasing the probability of finding the boundary, otherwise it will form a consecutive max-length chunk sequence.

The basic idea in principle 1 is to restrain the formalization of consecutive max-length chunk sequence. When chunking process generates a max-length chunk, it probably becomes a beginning of consecutive max-length chunk sequence. Thus, we need to suitably enlarge the probability of boundary decision when the former chunk size is maximum chunk size. It is noted that appropriate enlarging the probability of boundary decision does not violate the content defined property in CDC because of the data locality. Content-defined chunking algorithms can still tolerate the modifications located in normal chunks. Consequently, we propose the principle 1 to avoid the local boundary problem.

However, For each enlarging probability of finding a chunk boundary, it corresponds to a new chunking rule. Different chunking rules for distinct data section in chunking process, such as original data and modified data, easily lead to different chunking results. Therefore, a lot of unique chunks are produced in unmodified data section. Thus, the principle 1 can only ensure to restrain the formalization of consecutive maximum chunks sequence. But, it can not keep the higher or the same deduplication ratio with traditional rabin-based CDC. The design of elastic chunking algorithm should dynamically

adjust the parameters in chunking process. To handle this problem, we further propose a principle 2 as follow.

Principle 2: Once the dilemma of the consecutive max-length chunks is eliminated, the chunking algorithm needs to resume the probability of finding the boundary to the initial state, or it will impact the deduplication ratio.

The basic idea of principle 2 is to keep the generation of each non-maximum length chunk under the same chunking rule. As we mentioned above, principle 1 can not keep the higher or the same deduplication ratio compared with traditional Rabin-based CDC algorithms. What if we resume the chunking rule to initial state when dealing with non-maximum chunk, it can keep the same data distribution section being split under the same chunking rule. In other words, principle 2 can keep the same or higher deduplication ratio with traditional rabin-based CDC based on principle 1. deduplication ratio. Therefore, dynamically resuming the probability of boundary decision can handle above problem.

What's more, it can also speed up the chunking throughput and deduplication ratio by combining principle 1 with principle 2. There are two reasons. Firstly, When elastic chunking algorithm restricts the formalization of consecutive maximum length sequence, the chunk number gradually increase under this condition. And the chunking process does skip the scanning of minimum chunk size bits for each chunk. Therefore, the chunking algorithm can skip more bits. Secondly, as the suppression of consecutive maximum length chunk sequence, the number of content-defined chunk number also increase which can definitely improve the deduplication ratio. In Subsection III-B, we design the elastic chunking algorithm to solve the local boundary problems and decrease the impact of small scatter modification in deduplication ratio by combining the Principle 1 and 2.

B. Elastic Chunking Algorithm

In this part, we introduce the details of elastic chunking algorithm. It has seven parameters: T_{min} means the minimum chunk size threshold, T_{max} denotes the maximum chunk size, and sub_t_{Max} means the sub-maximum chunk size. D_1 and D_2 are the main divisor value and second divisor value, respectively, while D_2 is half of D_1. r_1 is a value in the range of $[0, D_1)$. r_2 is a value in the range of $[0, D_2)$. sub_t_{Max} is one percent of t_{Max}.

In practice, the elastic chunking treats each file as a bit stream and scan it based on sliding window from the beginning to the end. During the scanning process, elastic chunking algorithm records the whole chunk boundaries. The rabin-based fingerprint value corresponding for the sliding window content is divided by D_1 and D_2. Once the remainder of D_1 is equivalent to the predefined value r_1, the current location of sliding window is treated as a chunk boundary. If only the remainder of D_2 equals to the predefined value r_2 before sub_t_{Max}, the location is considered as a backup chunk boundary. When the position is between sub_t_{Max} and t_{Max}, the location is also set as a backup chunk boundary, once the corresponding remainder divided by D_2 is equal to r_2 or a value in a remainder set, $rlist$. The latest backup chunk boundary is considered as the chunk boundary if no chunk boundary is found until the maximum chunk length location.

Once neither a chunk boundary nor a backup chunk boundary is found until to the maximum chunk length location, the current location is forced as the chunk boundary since the maximum chunk size is T_{max}. If the current chunk is a maximum length chunk, a new random value is added into $rlist$ which is no greater than D_2. It is noted that $rlist$ is dynamically adding or deleting some remainder values to restrict the formalization consecutive maximum chunk sequence or restore to initial chunking state. More specifically, all the values in collection $rlist$ must be in the range of $[0, D_2)$. Elastic chunking algorithm adds a new random value into $rlist$ for each maximum length chunk, which is in the range of $[0, D_2)$. Once a content-defined chunk boundary is detected, $rlist$ will be emptied.

Algorithm 1: Elastic Chunking

Input: *A file and the chunking algorithm's parameters:* D_1, D_2, t_{Max}, t_{Min}, sub_t_{Max}, r_1, r_2
Output: *the chunk boundary list* $breakpointlist$

```
while scan the whole file from left to right do
    if scan_postion < t_Min then
        continue;
    value = rabin_hash(read(position, file));
    if scan_position < sub_t_Max then
        if value%D_1 = r_1 then
            addbreakpoints(position);
            rlist.clear();
            continue;
        else
            if value%D_2 == r_2 then
                backup_breakpoint = position;
                rlist.clear();
                continue;
    if scan_position == sub_t_Max then
        if backup_breakpoint ≢ Null then
            addbreakpoints(backup_breakpoint);
            rlist.clear();
            continue;
    if scan_position < t_Max then
        rmd_D_1 = value%D_1;
        rmd_D_2 = value%D_2;
        if rmd_D_1 == r_1 then
            addbreakpoints(position);
            rlist.clear();
            continue;
        if rmd_D_2 ∈ rlist||rmd_D_2 == r_2 then
            backup_breakpoint = position;
            rlist.clear();
    else
        if backup_breakpoint ≢ Null then
            addbreakpoints(backup_breakpoint);
            rlist.clear();
            continue;
        else
            max_length_count++;
            add_new_r_to_rlist();
            addbreakpoints(t_Max);
return breakpointlist;
```

Elastic chunking algorithm increases the probability of boundary decision by an adjustment policy to adding more compared remainder values when facing the consecutive maximum length chunk sequence or back to initial chunking state after a content-defined chunk boundary. In other words, once the current chunk is non-maximum length chunk, the set $rlist$ is emptied. Based on principle 2, the size of $rlist$ is dynamically increasing to resist the formalization of consecutive maximum chunk sequences.

The detailed procedure is shown as Algorithm 1. The function of $addbreakpoints$ in Algorithm 1 means recording a chunk boundary position. The $rabin\_hash$ calculates the rabin-based fingerprint hash value in the content of sliding window [7]. The $backup\_breakpoint$ is a temporary variable to store the backup boundary position. Although there may have several backup breakpoints during a boundary decision process, only the latest backup breakpoint is stored in backup_breakpoint. Moreover, the function of $add\_new\_r\_to\_rlist$ adds the new random remainder value into $rlist$. During the chunking process, $rlist$ is dynamically adjusting. The more consecutive maximum length chunks occur during the whole chunking process, the larger $rlist$ is. Therefore, it can effectively resist the consecutive max-length chunk pattern to avoid local boundary shift problem while also keep the high deduplication ratio. It is noted that scanning process can skip T_{min} bits after each chunk boundary location, since the parameter T_{min} is the minimum chunk size.

IV. Performance Evaluation

In this section, we present the details of experimental design, results, and analysis. Even though there are some research works on chunking algorithms, most of them are dedicated to the throughput of chunking algorithm. As for the deduplication ratio, TTTD type chunking algorithm always achieve the best deduplication ratio among these Rabin-based chunking algorithms without any interaction with server. Note that, we consider that RC belongs to TTTD category chunking algorithm since it uses the multi-stage to vary the expected chunk size similar to TTTD chunking algorithm. Leap-based TTTD [10] replaces the rabin-based hash by pseudo-random transformation which does not use rabin fingerprint hash. In this paper, we did not compare the leap-based TTTD [10] with our algorithm since the optimization parameters of pseudo-random transformation were not publicly available. We implement the BSW CDC algorithm[6] , TTTD algorithm[8] and RC[9] as comparison algorithms.

A. Experimental Setup

All the chunking experiments are conducted under the same equipment machine with a Intel(R) Core(TM) i7-4702MQ@2.20GHz 8core CPU, 8GB RAM, and a 1TB 5400rpm hard disk. The OS installed is Ubuntu 16.04 LTS 64-bit System. All chunking algorithms are implemented in Java.

In our experiments, we use three datasets. First one is artificial dataset which comes from vmdk file trace[11]. The second one is Fslhome trace[12]. The third one is Linux Archive files[13]. To highlight the local boundary shift problem, we choose artificial datasets as ideal datasets. Then, we also show the benefits under different real datasets, including Linux Archive files and Fslhome trace. Since the Linux Kernel archive files have variant versions, we take it to simulate the realistic scenarios. Moreover, Fslhome trace contains snapshots of student's home directory from a shared network file system. The snapshots were collected in the File System and Storage Lab(FSL) at Stony Brook University.

B. Impacts of Deduplication Ratio by Local Boundary Shift Problem

In this part, we take two type methods to preprocess the artificial datasets before conducting experiments for highlighting the local boundary shift problem. In the first method, each file of artificial datasets is modified by adding one bit before the location of each consecutive maximum length chunk sequence. Note that the location of consecutive maximum length chunk sequence is recorded on unmodified artificial datasets. Since different average chunk size will have distinct probability of maximum length chunk, we conduct the chunking process on modified artificial datasets. In the second method, we randomly modify the file in unmodified artificial datasets based on Poisson's distribution from 0.2% to 16%.

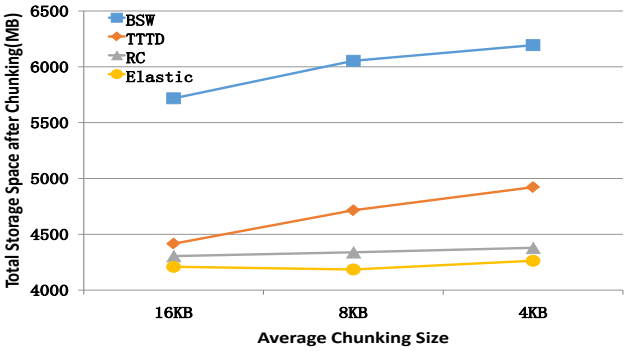

Fig. 1: Total Size of Artificial Dataset in the First Method

As shown in Figure 1, the final size of modified artificial datasets by elastic chunking algorithm is compared with other algorithms. The size after chunking in sliding CDC chunking algorithm increase 476MB as the average chunk size ranged from 16KB to 4KB. TTTD chunking algorithm presents a similar tendency as sliding CDC chunking algorithm. It is because that one bit modification in method1 results in the following k maximum chunks as new chunks where k is the chunk number in a consecutive maximum length chunk sequence. Since BSW and TTTD have lots of consecutive maximum chunk sequences, the final size of them are increasing even though the average of chunking size decrease. RC has the similar tendency compared with elastic chunking, since it decreases the whole maximum length chunks by multi-stage divisor decision. However, the chunking efficiency of RC is not better than proposed elastic chunking which will be analyzed in following subsections.

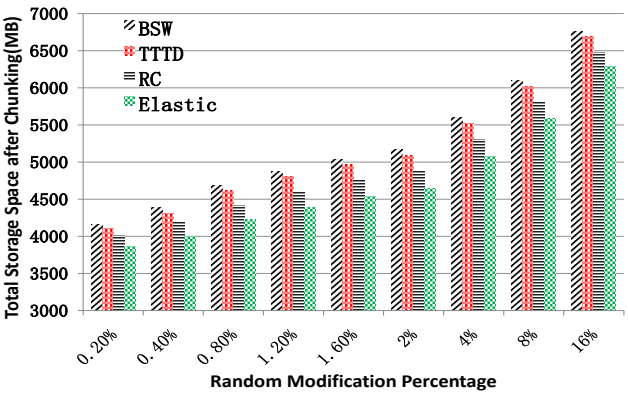

Fig. 2: Total Size of Artificial Dataset in the Second Method

As shown in Figure 2, we evaluate the changing final size after chunking the modified artificial datasets based on method2. We can easily figure out that randomly modification will also produce huge new blocks leading to the final size keeps increasing. When increasing size under 16% percentage of random modification is 2604MB more

than the situation under 0.20% percentage of random modification in sliding-window CDC chunking algorithm. TTTD and RC performs better than BSW CDC chunking algorithm. But the elastic chunking can acquire lesser size compared with other algorithms. It is because that elastic chunking can achieve less impact of deduplication ratio under the randomly modification scenario. Since elastic chunking can tolerant the boundary shift problem, it can decrease the impact of modification located at non-maximum chunk's data sections. Moreover, elastic chunking can further solve the local boundary shift problem based on the dynamical adjustment scheme. Consequently, elastic chunking have a smaller final size compared with others in distinct random modification proportion.

C. Benefits on Real Datasets by Solving Local Boundary Shift Problem

To further observe the benefits on the real datasets, we conduct the experiment on two real datasets. The first one is the Fslhome [12]. The second one is Linux kernel [13]. Each dataset includes several former modified versions. For example, the Fslhome trace collected the students' home directories for every day which consist of the office documents, source code, virtual images and other miscellaneous files. To test the elastic chunking algorithm on real datasets, we respectively test the three chunking algorithms on 16KB, 8KB, and 4KB average chunk sizes by measuring the sizes after removing duplicate chunks.

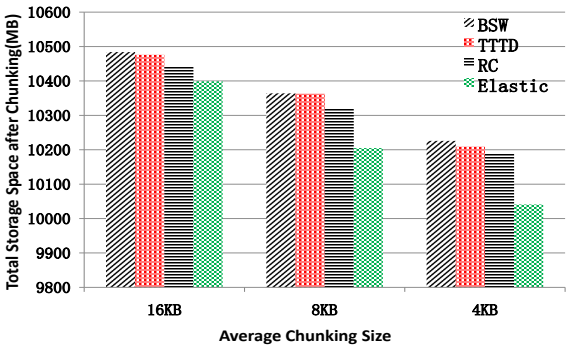

Fig. 3: Total Size of Fslhome Datasets after Chunking

As shown in Figures 3 and 4, Sliding CDC ,TTTD and RC have the decline tendency as the average chunk size changed from 16KB to 4KB. It is because the less average chunk size it has, the more duplicate data the chunking algorithm can remove. However, Our method can find more duplicate data compared with others. For example, when the average chunk size is 8KB, RC is better by removing 43MB and 109MB more data than TTTD in Fslhome and linux workloads, respectively. Moreover, elastic chunking can further save 148MB and 174MB compared with RC when average chunk size is 8KB. As the average chunk size is decreasing, the more duplicate data can be discovered compared with others. It is because that elastic chunking algorithm also restricts the consecutive max-length chunk sequences to solve the local boundary shift problem. In Fslhome and Linux Archive workloads, there exist a lot of modification version files which easily lead to local boundary shift problem while other chunking algorithms ignore the penalties caused by maximum length chunks. In elastic chunking algorithm, the local boundary shift problem is solved by dynamically adjustment scheme to limit the formalization of consecutive maximum length sequence. From Figures 3 and 4, one can clearly see that the elastic chunking algorithm performs well on real datasets.

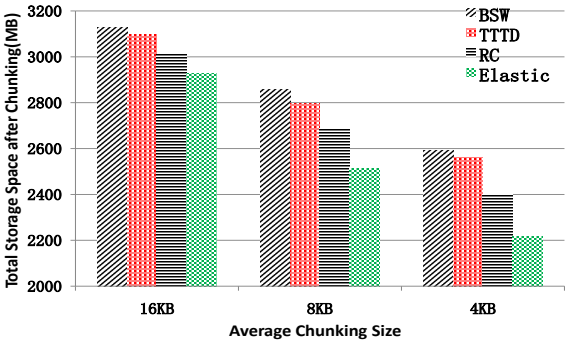

Fig. 4: Total Size of Linux Dataset after Chunking

D. Throughput Comparison

To further evaluate the throughput of elastic chunking algorithm compared with others, we measure the throughput on Fslhome and Linux Archive workloads. The results of chunking throughputs of BSW CDC chunking algorithm , TTTD, RC and elastic chunking algorithm on two real datasets from 4KB to 16KB are shown in Table IV. Based on the results, a few observations are as follows: First, the throughput in rabin-based chunking algorithms is largely independent with the average chunk size. From Table IV, each chunking algorithm's throughput is almost the same. However, different chunking algorithm has the variant throughputs under the same condition. For example, When the average chunk size is 16KB, the throughput in elastic chunking is 26.4% higher than RC. Compared with TTTD and BSW, elastic chunking algorithm outperforms by 16.6% and 17.7%, respectively. Even though the TTTD and RC can discover more duplicate data compared with BSW, Both TTTD and RC take too much time in boundary decision process because of multi-stage loop process. Therefore, BSW CDC owns a higher throughput compared with TTTD and RC. However, elastic chunking can acquire a faster throughput than others since it do not need always taking multi-stage method to decrease the maximum chunks. Elastic chunking algorithm is to dynamically restrict the consecutive maximum chunk sequence, not all the maximum chunks. Therefore, Our chunking algorithm can have the highest throughput compared with other chunking algorithms.

TABLE IV: Throughput Comparison in Real Datasets(MB/s)

Workloads	BSW	$TTTD$	RC	$Elastic$	ACS(KB)
Fslhome	305.13	302.38	281.42	355.76	16
Fslhome	303.37	300.19	277.58	325.51	8
Fslhome	300.79	298.83	273	315.29	4
Linux	303.59	287.36	269.44	350.2	16
Linux	295.93	281.17	274.48	327.38	8
Linux	298.07	272.38	256.32	312.84	4

V. Related Work

Chunking algorithm is an important component in deduplication system [14–17]. The basic idea is to split the large file into small chunks for removing more duplicate data. Therefore, how to decide the boundary of each chunk is the key process in chunking algorithm. In other words, the efficiency of chunking algorithm directly determines the deduplication ratio and throughput in a deduplication system. The simplest and fastest approach for high throughput is to break the large file into fixed-size chunks [18, 19]. However, boundary shift problem will occur when a modification edit near the beginning of a file in Fixed-size chunking algorithm[20].

To overcome this problem, two main boundary judgment methods are proposed in content-defined chunking, rabin-based fingerprint value [7] and the maximum or minimum extreme value[21]. As for the Rabin-based fingerprint [6], basic sliding window content defined chunking algorithm(BSW CDC) is proposed by producing variable size chunks in deduplication to solve the boundary shift problem. Since the chunk boundary is related to content, BSW CDC can effectively decrease the impact of modification compared with Fixed-size chunking algorithm. It runs a sliding-window hash along the byte stream. When the hash value in sliding window equals to a predetermined value, a chunk boundary was declared. Imposing a minimum and maximum parameters in BSW algorithm is to avoid too small or too large chunks. The BSW approach is appealing because it helps make the deduplication ratio more stable and improves deduplication performance with skipping of the minimum chunk size when searching for breakpoints. Our algorithm followed this limitation.

However, even though the BSW content-defined chunking algorithm can effectively inhibit the boundary shift problem compared with fixed-size chunking algorithm, it performs poorly on real data [8]. There is still some improvement space on BSW algorithm, such as the modification overhead to deduplication ratio. A small modification may enlarge several new chunks to be upload. What's more, [22] shows that small semantic changes on documents cause lots of small modification in the binary representation of files, which will greatly impact the deduplication ratio. After a quantitative analyzing for former content-defined chunking algorithms, TTTD [8] suggests that reducing chunk size variability can decrease the modification overhead. To solve above problem, it argue that chunking algorithm need avoid too small or too large chunk size to decrease the modification overhead by theoretical analysis. Based on TTTD algorithm, TTTD-s algorithm [23] is proposed to make some improvement on the trade-off and solving the redundancy computing boundary problems caused by second divisor in TTTD. However, it potentially decreases the average chunk size by diminishing the divisor value during chunking process. What's more, it is very hard to grasp the average chunk size.

In addition, RC [9] adopted secondary, third, fourth and fifth conditions to reduce the proportion of forced boundaries. However, it was the enforce multi-stage version of TTTD in essence. So the real problem of maximum length chunk is not whether the chunk is content defined,but the risk of consecutive maximum length chunks. What's more, RC[9] takes a fixed divisor value list as the second divisor in TTTD to minimize the probability of Maximum length chunks for each boundary decision like a multi-stage TTTD. More specifically, each forward processing in RC need (k-1) additional loop match, where k means the regression level parameter in RC. When K=2, RC chunking will have the same similarities with TTTD chunking algorithm. the parameter k in RC chunking algorithm are predefined before chunking process, not dynamic, which will cost too much overhead. What's more, it can only improve a little deduplication ratio compared with TTTD as discussed in Section IV.

Bimodal CDC [24] also propose a dynamically method to vary the expected chunk size to perform content-defined chunking with several interactions with server-side. This algorithm first chunks the data stream into large chunks and then splits part of them into small chunks. Similarly, Lu also mixed chunks of different average size together, but determined whether to chunk the data stream into large chunks or small chunks according to the reference count[25]. However, they have to check the fingerprint index to determine whether to split large chunks or merge small chunks with the interactions with server-side. It is noted that we do not focus on the chunking algorithm with interaction with server. To further improve the chunking throughput, Leap-based TTTD present a leap-based CDC algorithm with a leap procedure to improve the deduplication performance[10]. But it was not based on the fingerprint value and the optimal parameters is not published yet. When putting the Leap scheme into TTTD, the deduplication ratio is no more than using TTTD.

There are also some other chunking algorithm based on the extreme value such as MAXP[21] [26] and AE[27]. But they are based on the extreme value in boundary decision, not based on Rabin. In our focus, TTTD type can achieve the highest deduplication ratio in our focus algorithm field. However, most of them are simply think the maximum length chunk is one of the reasons that impact the deduplication ratio. However, we discover that the real problem is the local boundary shift problem caused by maximum length chunks, not the whole maximum length chunks. Once one bit is changed in the first maximum length chunk in a consecutive max-length chunk pattern, these small sequences will degrade to a local fixed-sized chunking situation. Unfortunately, existing content based chunking algorithms namely BSW, TTTD and RC, ignore the degradation caused by consecutive max-length chunk pattern.

To overcome the local boundary shift problem, elastic chunking algorithm is proposed to dynamically restrain the formalization of consecutive maximum length sequence. By reducing the consecutive maximum length sequence, it decreases the impact of small scatter random modification in deduplication scenario. the existing Rabin-based chunking algorithm such as BSW, TTTD, and RC chunking algorithms may easily be affected by small scatter modifications, especially in those frequently revised storage scenarios.

VI. Conclusions and Future Work

Deduplication is one of the most popular methods since it can efficiently improve the utilization of server storage and save the network bandwidth by removing the redundancy. As an important component of deduplication, chunking algorithm directly determines the deduplication ratio and throughput of deduplication system. However, the consecutive maximum length chunk sequence easily results in local boundary shift problem. It can sacrifice the deduplication ratio and throughput when a small modification before the consecutive maximum length chunk sequence. Moreover, existing rabin-based chunking algorithms, which without the interaction with server during chunking process, do not consider this problem in the design of chunking algorithm.

In this paper, we present the local boundary shift problem. Then, we further analyze the problem from both theoretical and experimental aspects. To overcome the local boundary shift problem, we propose a novel chunking algorithm, elastic chunking, by dynamically limiting the formalization of consecutive maximum length chunk sequences. Our experimental results clearly show that elastic chunking algorithm can effectively solve the local boundary problem and achieve an higher deduplication ratio and throughput compared with other chunking algorithms.

In the future work, we plan to broadly test elastic chunking algorithm in more diverse workloads and integrate it into secure deduplication scenario. In addition, we also plan to speed up the chunking process and reduce the energy consumption under the fog computing scenario.

VII. Acknowledgement

This work is supported by the National Key Research and Development Program of China (No. 2016YFB0800402), National Natural Science Foundation of China under grants 61572221, U1401258, 61433006, 61300222, 61370230 and 61173170, Innovation Fund of Huazhong University of Science and Technology under grants 2015TS069 and 2015TS071, Science and Technology Support Program of Hubei Province under grant 2015AAA013 and 2014BCH270, and Science and Technology Program of Guangdong Province under grant 2014B010111007. This work is also partly supported by the National Science Foundation under grand CNS 1526190.

REFERENCES

[1] Dutch T. Meyer and William J. Bolosky, "A study of practical deduplication," in *9th USENIX Conference on File and Storage Technologies, San Jose, CA, USA*, 2011, pp. 1–13.

[2] Nohhyun Park and David J. Lilja, "Characterizing datasets for data deduplication in backup applications," in *Proceedings of the 2010 IEEE International Symposium on Workload Characterization, Atlanta, GA, USA*, 2010, pp. 1–10.

[3] Fanglu Guo and Petros Efstathopoulos, "Building a high-performance deduplication system," in *2011 USENIX Annual Technical Conference, Portland, OR, USA*, 2011.

[4] Fred Douglis, Abhinav Duggal, Philip Shilane, Tony Wong, Shiqin Yan, and Fabiano C. Botelho, "The logic of physical garbage collection in deduplicating storage," in *15th USENIX Conference on File and Storage Technologies, Santa Clara, CA, USA*, 2017, pp. 29–44.

[5] João Paulo and José Pereira, "Efficient deduplication in a distributed primary storage infrastructure," *ACM Transactions on Storage (TOS)*, vol. 12, no. 4, pp. 20, 2016.

[6] Athicha Muthitacharoen, Benjie Chen, and David Mazieres, "A low-bandwidth network file system," in *ACM SIGOPS Operating Systems Review*. ACM, 2001, vol. 35, pp. 174–187.

[7] Michael O Rabin et al., *Fingerprinting by random polynomials*, Center for Research in Computing Techn., Aiken Computation Laboratory, Univ., 1981.

[8] Kave Eshghi and Hsiu Khuern Tang, "A framework for analyzing and improving content-based chunking algorithms," *Hewlett-Packard Labs Technical Report TR*, vol. 30, no. 2005, 2005.

[9] Ahmed El-Shimi, Ran Kalach, Ankit Kumar, Adi Ottean, Jin Li, and Sudipta Sengupta, "Primary data deduplication-large scale study and system design.," in *USENIX Annual Technical Conference*, 2012, vol. 2012, pp. 285–296.

[10] Chuanshuai Yu, Chengwei Zhang, Yiping Mao, and Fulu Li, "Leap-based content defined chunking - theory and implementation," in *IEEE 31st Symposium on Mass Storage Systems and Technologies, Santa Clara, CA, USA*, 2015, pp. 1–12.

[11] "Vmdk trace," https://sourceforge.net/projects/thoughtpolicevm/files/.

[12] "Fslhome trace," http://tracer.filesystems.org/.

[13] "Linux kernal," https://www.kernel.org/.

[14] Ashok Anand, Vyas Sekar, and Aditya Akella, "Smartre: an architecture for coordinated network-wide redundancy elimination," in *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Barcelona, Spain*, 2009, pp. 87–98.

[15] Wen Xia, Hong Jiang, Dan Feng, Lei Tian, Min Fu, and Yukun Zhou, "Ddelta: A deduplication-inspired fast delta compression approach," *Performance Evaluation*, vol. 79, pp. 258–272, 2014.

[16] Wen Xia, Hong Jiang, Dan Feng, Fred Douglis, Philip Shilane, Yu Hua, Min Fu, Yucheng Zhang, and Yukun Zhou, "A comprehensive study of the past, present, and future of data deduplication," *Proceedings of the IEEE*, vol. 104, no. 9, pp. 1681–1710, 2016.

[17] Dutch T Meyer and William J Bolosky, "A study of practical deduplication," *ACM Transactions on Storage (TOS)*, vol. 7, no. 4, pp. 14, 2012.

[18] Andrew Tridgell, *Efficient algorithms for sorting and synchronization*, Ph.D. thesis, Australian National University Canberra, 1999.

[19] Andrew Tridgell and Paul Mackerras, "The rsync algorithm," *Australian National University*, 1996.

[20] Chuck Yoo Young Chan Moon, Ho Min Jung and Young Woong Ko, "Data deduplication using dynamic chunking algorithm," pp. 59–68, 2012.

[21] Nikolaj Bjørner, Andreas Blass, and Yuri Gurevich, "Content-dependent chunking for differential compression, the local maximum approach," *Journal of Computer and System Sciences*, vol. 76, no. 3-4, pp. 154–203, 2010.

[22] Dirk Meister and André Brinkmann, "Multi-level comparison of data deduplication in a backup scenario," in *Proceedings of of SYSTOR 2009: The Israeli Experimental Systems Conference 2009, Haifa, Israel*, 2009, p. 8.

[23] Teng Sheng Moh and Bing Chun Chang, "A running time improvement for the two thresholds two divisors algorithm," in *Southeast Regional Conference, 2010, Oxford, Ms, Usa, April*, 2010, pp. 1–6.

[24] Erik Kruus, Cristian Ungureanu, and Cezary Dubnicki, "Bimodal content defined chunking for backup streams," in *8th USENIX Conference on File and Storage Technologies, San Jose, CA, USA*, 2010, pp. 239–252.

[25] Guanlin Lu, *An efficient data deduplication design with flash-memory based solid state drive*, University of Minnesota, 2012.

[26] Ashok Anand, Chitra Muthukrishnan, Aditya Akella, and Ramachandran Ramjee, "Redundancy in network traffic: findings and implications," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 1, pp. 37–48, 2009.

[27] Yucheng Zhang, Hong Jiang, Dan Feng, Wen Xia, Min Fu, Fangting Huang, and Yukun Zhou, "AE: an asymmetric extremum content defined chunking algorithm for fast and bandwidth-efficient data deduplication," in *2015 IEEE Conference on Computer Communications, Hong Kong*, 2015, pp. 1337–1345.