Two-Threshold Chunking (TTC): Efficient Chunking Algorithm For Data Deduplication For Backup Storage

Anand Bhalerao, Ambika Pawar

Abstract: Large amount of data gets generated every day and storing that data efficiently becomes a heuristic task. Backup storages are more prominently used media for storing every day, the generated data. The significant amount of data that is stored in the backup storage is redundant and leads to the wastage of storage space. Storage space can be saved and processing speed of backup media can be improved using deduplication and variable size chunking. Various chunking algorithms have been presented in the past to improve deduplication process. This paper presents the design of an efficient chunking algorithm to achieve high throughput and to reduce processing time.

Index Terms: Backup Storages, Chunk Level Deduplication, Chunking Algorithm, Cloud Storages, Deduplication, Redundant data, Variable-Size Chunking

1. INTRODUCTION

In cloud data storages, where the chances of having more redundant data is high, the data deduplication technique is used because of its scalability and efficiency as compared to conventional compression techniques such as Huffman Compression [1] and LZ Compression [2]. There are two major benefits of data deduplication, first one is it detects and removes the chunk level redundancy unlike conventional compression techniques which works at the string/byte level. Other benefit is that it detects redundant chunk with the help of fingerprint whereas the traditional compression techniques [3] which does byte level comparison which increases the processing time.

2 DATA DEDUPLICATION

2.1 Chunk-Level Deduplication

In this section we will discuss different chunk-level Deduplication techniques [10]: Figure 2 shows the classification of chunking algorithms and their characteristics.

<table>
<thead>
<tr>
<th>Type of chunking</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable-size</td>
<td>Fewer boundary shift</td>
<td>Generates more indexes</td>
</tr>
<tr>
<td>Fixed-size</td>
<td>Generates indexes</td>
<td>Few More boundary shift</td>
</tr>
</tbody>
</table>

2.2 Various Variable-Size Chunking Algorithms

There are various chunking algorithms available like Rabin, TD, TTTD, MAXP, Bimodal [11] and MCDC [12] etc. and few of them are explained below:
chunk boundary. Using these chunk boundaries large chunks are divided into small chunks using secondary and smaller divisor. So there is more probability of getting same fingerprint. TD tries to overcome the boundary shifting problem of Breaking Large Chunks into Fixed Size Sub-Chunks algorithm [7]. TD has two divisors D and D’ where D is secondary backup divisor and D is main divisor and D’ is smaller than D. TD uses these divisors to find the breakpoints. TD algorithm scans the data stream, it starts from the last chunk boundary. It matches D and D’ fingerprint at each position by memorizing the position when the last D’ fingerprint match. The position is declared as a breakpoint if D-match is found before reaching the threshold Tmax. In case the D match is not found before reaching the Tmax threshold then earlier D’ match position is used as the breakpoint else, the threshold should be used as a breakpoint.

Two Thresholds, Two Divisors (TTTD): TTTD algorithm [7] utilizes concept of TD algorithm and the SCM (Small Chunk Merge) [7] algorithm. TTTD modifies the TD algorithm, it considers minimum threshold for chunk size. This algorithm searches and compares fingerprints in backup and main divisor, until it passes the threshold. This algorithm takes four input values: D (main divisor), D’ (backup divisor), Tmin (minimum chunk size threshold) and Tmax (maximum chunk size threshold).

**Table 2: Advantages and Disadvantages of Chunking Algorithms**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>TD</td>
<td>Reduces chunk size variance</td>
<td>Misses out redundant chunks</td>
</tr>
<tr>
<td></td>
<td>High throughput</td>
<td>Low computational overhead</td>
</tr>
<tr>
<td>AE</td>
<td>High throughput</td>
<td>Less computational overhead</td>
</tr>
<tr>
<td>MAXP</td>
<td>Reduces computational overhead</td>
<td>Low chunking throughput</td>
</tr>
<tr>
<td>AE opt1</td>
<td>Low throughput for low entropy strings</td>
<td>Low chunking throughput</td>
</tr>
<tr>
<td>AE opt2</td>
<td>Low throughput for higher entropy strings</td>
<td>Low chunking throughput</td>
</tr>
<tr>
<td>TTTD</td>
<td>Low throughput for fixed-size chunks</td>
<td>Low chunking throughput</td>
</tr>
<tr>
<td></td>
<td>Low throughput for variable-size chunks</td>
<td>Low chunking throughput</td>
</tr>
</tbody>
</table>

**Fig. 2 Classification of Chunking Algorithms**

MAXP: [8] Overcomes the problem of chunk-size variance in Rabin-based chunking algorithm by working on local extreme values. MAXP is frequently applied in network storage systems. It divides windows at byte level. MAXP has an approach in which it finds the local extreme value using which MAXP algorithm revisits some already compared bytes. Thus MAXP reduces the throughput. AE Algorithm: [9] Addresses the boundary shifting problem occurring in various variable-size chunking algorithms. In AE algorithm asymmetric window of variable size is used for addressing the boundary shifting problem. AE is very fast as it does not backtrack and does single comparison. Also it has smaller chunk size variance as compared to other variable-size chunking algorithms.

**3 PERFORMANCE METRICS**

In this section various performance metrics for finding the efficiency of different chunking algorithms have been presented. DER (Deduplication Elimination Ratio): DER is a measure of the deduplication efficiency. DER is calculated using input data size and total size of all non-redundant chunks generated as given in the following equation:

\[ \text{DER} = \frac{(\text{Input Data Size})}{(\text{Total Size of non-redundant chunks})} \]

CPU Overhead: CPU/Computation overhead is one of the important metrics of the Deduplication technique. In chunking phase every byte has been compared in order to find the chunk boundary. This comparison incurs high computational cost. Processing Time: Total time required for completing deduplication process, including all the four phases is called as processing time. This metric can be the bottleneck because of the chunking algorithm. So it is important to design efficient chunking algorithm. Number of Chunks Generated: Total number of chunks generated affects the performance of the duplication technique. The algorithm should not generate too less or too high number of chunks. Too high number of chunks will require more comparisons to find duplicate chunks, which eventually lead to high processing time. Too less number of chunks affects the chunking throughput as redundancy will get reduced.

**4 PROPOSED TWO-THRESHOLD CHUNKING (TTC) ALGORITHM**

In this section the proposed Two-Threshold Chunking (TTC) Algorithm [13][14][15] for chunking in deduplication is discussed. TTC has two threshold values (Tmin and Tmax) which are used to restrict the chunks formation, so that it will not have more chunk size variance. To get the breakpoint for the chunk, the algorithm finds the maximum point in the incoming data stream and the point which satisfies both the Tmin and Tmax value will be considered as the breakpoint.

**4.1 Flow of Two-Threshold Chunking Algorithm**

Fig. 3 Shows how Two Threshold Chunking algorithm works. ‘A’ is the starting point for the incoming data stream. Tmin and Tmax are the threshold values. First the algorithm will find the maximum value in the data stream, let say ‘B’ is the maximum point. Then declare ‘B’ as the breakpoint for the chunk and generate chunk from ‘A’ to ‘B’. For the next chunk consider next point after ‘B’ as the starting point and try to find maximum point from it and repeat this till the end of the data stream.
4.2 Pseudo Code for Two-Threshold Chunking Algorithm

Input: input data, D; Length of the input data, L;

Predefined value: Tmin, Tmax;

Function Two_Threshold (D, L, Tmin, Tmax)
1. breakpoint = 1;
2. Max.value = D[breakpoint];
3. breakpoint = breakpoint + 1;
4. For (; breakpoint < L; breakpoint++)
   If (Max.value < Tmin or Max.value > Tmax)
      return breakpoint; //breakpoint
   Else
      Max.value = D[breakpoint];
   End if
End function

Algorithm takes input as the data stream, Tmin and Tmax value. Max.value will store the maximum value in the data stream and Tmin and Tmax are the threshold values.

5 RESULT AND DISCUSSION

To evaluate the performance of the algorithm, experimental setup consists of Intel Core i5-3320M CPU @ 2.60 GHz system with 4GB RAMS. A file is given as the input to the algorithm. Comparison is mainly based on the Average chunks size with DER factor, Processing Time and number of chunks generated. The figures given below show the comparison between AE_Opt1 and Two-Threshold Algorithm.

In Figure 4, the DER factor for Two-Threshold Chunking Algorithm is higher in all the cases. Figure 5 shows the deduplication process time with the given chunking algorithm. For smaller chunk size takes more time to process the data In Figure 6 Number of chunks for TTC and AE_Opt1 In Figure 7, the numbers of chunks generated while chunking are shown. The process should not generate so many chunks that it will affect the performance of the deduplication. In Figure 7, chunk size variance between TTC and AE_Opt1 is given. Due to the use of threshold values in TTC the chunk size variance is low compared to AE_Opt1. Threshold values limit the chunk size limit which helps to lower the chunk size variance.


### Table 3 A

**COMPARISON BETWEEN AE_Opt1 AND TTC ALGORITHM FOR 1 KB AVG. CHUNK SIZE**

<table>
<thead>
<tr>
<th>Properties/Algorithms</th>
<th>AE_Opt1</th>
<th>TTC</th>
</tr>
</thead>
<tbody>
<tr>
<td>DER</td>
<td>17</td>
<td>23</td>
</tr>
<tr>
<td>Time (Sec.)</td>
<td>11.74</td>
<td>7.72</td>
</tr>
<tr>
<td>No. Of Chunks</td>
<td>690</td>
<td>442</td>
</tr>
</tbody>
</table>

### Table 3 B

**COMPARISON BETWEEN AE_Opt1 AND TTC ALGORITHM**

<table>
<thead>
<tr>
<th>Properties/Algorithm</th>
<th>AE_Opt1</th>
<th>TTC</th>
</tr>
</thead>
<tbody>
<tr>
<td>DER</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Time</td>
<td>More</td>
<td>Less</td>
</tr>
<tr>
<td>No. Of chunks</td>
<td>More</td>
<td>Moderate</td>
</tr>
</tbody>
</table>

Table 3 (A) and (B) summarizes the overall comparison between Two-threshold and AE_Opt1 algorithm.

6 CONCLUSION

In proposed algorithm the threshold values help to maintain the chunk generation in given chunk size limit. The maximum point helps to find the breakpoint for the algorithm. The proposed Two-Threshold Chunking algorithm works better than AE_Opt1 algorithm with respect to time, throughput, number of chunks and chunk size variance.

REFERENCES


[2]. Huffman coding, Available at: [https://www.cs.auckland.ac.nz/software/AlgAnim/huffman.htm](https://www.cs.auckland.ac.nz/software/AlgAnim/huffman.htm)


