Leap-based Content Defined Chunking --- Theory and Implementation

Chuanshuai Yu, Chengwei Zhang, Yiping Mao, Fulu Li
Huawei Technologies
Agenda

- Introduction & motivation
- Proposed algorithm
- Algorithm validation
- Conclusion
A “good” chunking algorithm should satisfy the following criteria:
1. outputting right data chunks so that the dedup can approach to the best inherent de-duplication ratio.
2. fast chunking speed with a low CPU overhead.

We will focus on both (1) and (2) in this work.
1. Calculate the fingerprint of each window of pre-defined size and with the end-point lying between the minimum and the maximum boundary points.

2. If the fingerprint satisfies a given condition, then the chunk end-point is set at the end-point of the window.

3. If not, then the window is slid forward by one byte and repeat the step 1 and 2, until reach to the maximum boundary point and in this case, set the chunk end-point at the maximum boundary point.

**Issue:** Heavy computation due to byte sliding mechanism.
Similar setting as in sliding-window-based CDC, except that:

1. Instead of checking the fingerprint of one window, the proposed algorithm check k windows (say, k=24) to set chunk end-point.

2. Only when all of k windows' fingerprints meet the given conditions the chunk is set.

3. It will leap forward k bytes as soon as one window fails the check condition (The detailed leap procedure is addressed in next slide).
Leap Procedure

1. Starting from the right-most one among k windows, backward check the given condition for each of k windows.
2. If all k windows satisfy given condition, then a chunk end-point is set at the most-right window’s end-point.
3. Once encounter the first window that fails the check condition, leap 24 bytes forward from the failed point.
4. Repeat step 1 – step 3 until reach to or jump over the maximum boundary point and in this case, set the chunk end-point at the maximum boundary point.

Since the probability of the failed condition for each window is high(= ¼ in our typical design), leap forwarding will take place before check out all k windows in most time (averagely check 3 windows) and hence speed up the chunking.
The Secondary Conditions

• To reduce the probability of forcing set of a chunk, the well-known TTTD algorithm introduces the secondary conditions.

• Similarly, we can also introduce the secondary conditions in proposed leap-based chunking algorithm.
## Theoretical Analysis

<table>
<thead>
<tr>
<th>Probability of one failed window check</th>
<th>Action after failed check</th>
<th>Average amount checked</th>
<th>Average cost of each check</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sliding window based</td>
<td>Sliding 1 byte</td>
<td>1X</td>
<td>1X</td>
</tr>
<tr>
<td>Leap based</td>
<td>Leap 24 bytes</td>
<td>1/5 X</td>
<td>2.5X</td>
</tr>
</tbody>
</table>

Theoretically, the computation cost of one chunking by the leap-based algorithm is about half of the one by the sliding-window-based algorithm.
The distributions of chunk sizes outputted from two algorithms are similar.
The distributions of chunk sizes outputted from two algorithms with TTTD conditions are similar.
## Datasets Used in Experiments

<table>
<thead>
<tr>
<th>Type</th>
<th>Size(KB)</th>
<th>How generated and description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmware</td>
<td>81300750</td>
<td>This dataset is gotten by backuping 10 VMware files of Windows7 system by NetBackup software.</td>
</tr>
<tr>
<td>oracle_tbs_rman</td>
<td>14427720</td>
<td>This dataset is gotten by backuping a real database by RMAN interface.</td>
</tr>
<tr>
<td>oracle_tbs_dmp</td>
<td>10602880</td>
<td>This is the dmp file of a real database.</td>
</tr>
<tr>
<td>oracle_tbs_dbf</td>
<td>15990792</td>
<td>This is the dbf file of a real database.</td>
</tr>
<tr>
<td>sys</td>
<td>153871100</td>
<td>This dataset collects data of 20 C disks. The data are packed together without compression.</td>
</tr>
<tr>
<td>ISO</td>
<td>45486080</td>
<td>This dataset collects 20 ISO install files different versions of Windows operating system.</td>
</tr>
<tr>
<td>office</td>
<td>18114600</td>
<td>This dataset collects all kinds of office files, including doc, xls, ppt and so on. These files are packed together without compression.</td>
</tr>
<tr>
<td>music</td>
<td>4556260</td>
<td>This dataset collects all kinds of music files. These files are packed together without compression.</td>
</tr>
<tr>
<td>video</td>
<td>11327510</td>
<td>This dataset collects all kinds of video files. These files are packed together without compression.</td>
</tr>
<tr>
<td>pdf</td>
<td>4714870</td>
<td>This dataset collects all kinds of pdf files. These files are packed together without compression.</td>
</tr>
</tbody>
</table>

All datasets were collected from real production environments.
The distributions of chunk sizes outputted from two algorithms with a secondary condition agree with the theoretical analysis.
The average chunk sizes outputted from the two algorithms with a secondary condition agree with the theoretical analysis.
The deduplication ratio delivered by two algorithms are almost the same.
Leap-based CDC algorithm with the secondary condition speeds up the chunking by 50%~100%.
Conclusion

- The leap-based CDC algorithm can speed up chunking in 50% ~ 100% range.

- The leap-based algorithms outputs the similar distribution of chunk sizes with sliding-window-based algorithm and hence delivers the similar de-duplication ratio.
Thank You