Expert Systems with Applications 41 (2014) 2415–2423

Contents lists available at ScienceDirect

Expert Systems with Applications

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

SBBS: A sliding blocking algorithm with backtracking sub-blocks for duplicate data detection

GuiPing Wang[a,*], ShuYu Chen[b], MingWei Lin[a], XiaoWei Liu[c]

[a] *College of Computer Science, Chongqing University, Chongqing 400044, China*
[b] *College of Software Engineering, Chongqing University, Chongqing 400044, China*
[c] *HuaWei Research Institute, Chengdu, Sichuan 610041, China*

ARTICLE INFO

Keywords:
Data deduplication
Duplicate data detection
Sliding blocking algorithm
Backtracking
SBBS
Content-defined chunking algorithm

ABSTRACT

With the explosive growth of data, storage systems are facing huge storage pressure due to a mass of redundant data caused by the duplicate copies or regions of files. Data deduplication is a storage-optimization technique that reduces the data footprint by eliminating multiple copies of redundant data and storing only unique data. The basis of data deduplication is duplicate data detection techniques, which divide files into a number of parts, compare corresponding parts between files via hash techniques and find out redundant data. This paper proposes an efficient sliding blocking algorithm with backtracking sub-blocks called SBBS for duplicate data detection. SBBS improves the duplicate data detection precision of the traditional sliding blocking (SB) algorithm via backtracking the left/right 1/4 and 1/2 sub-blocks in matching-failed segments. Experimental results show that SBBS averagely improves the duplicate detection precision by 6.5% compared with the traditional SB algorithm and by 16.5% compared with content-defined chunking (CDC) algorithm, and it does not increase much extra storage overhead when SBBS divides the files into equal chunks of size 8 kB.

1. Introduction

Storage systems often contain redundant copies of data: identical files or sub-file regions, possibly stored on a single host, on a shared storage cluster, or backed-up to secondary storage (Meyer & Bolosky, 2011). Redundant data are caused by duplicate copies or regions of files. Therefore, the terms "redundant data" and "duplicate data" are interchangeable in the remainder of this paper. With the explosive growth of data, storage systems are facing increasingly serious storage pressure due to a mass of redundant data. It is beneficial to reduce duplicate data as many as possible (Lim, 2011). Through eliminating or reducing redundant data in storage systems, data deduplication techniques significantly improve the utilization of storage systems, and reduce the data volume during network transmission in case of remote backup (Ao, Shu, & Li, 2010).

Duplicate data detection techniques are the basis of data deduplication. These techniques divide the files (usually, the old version and the new version of a same file) into a number of parts, compare corresponding parts between files via hash techniques and find out redundant data. According to detection granularity, duplicate data detection techniques can be classified into three categories: file level, chunk level, and byte level.

File level techniques (Douceur, Adya, Bolosky, Simon, & Theimer, 2002; Harnik, Pinkas, & Peleg, 2010) compute the hash values of the whole files via hash functions such as SHA-1, MD5, etc. These techniques compare the hash values and determine whether these two files are identical. The implementation of file level techniques is simple, but the duplicate data detection precision is quite low. When the new file is modified slightly, the whole file is considered to be changed and no redundant data is detected.

Chunk level techniques (Bhagwat, Pollack, & Long, 2006; Denehy & Hsu, 2003; Eshghi & Tang, 2005; Litwin, Long, & Schwarz, 2012; Vrable, Savage, & Voelker, 2009; You, Pollack & Long, 2005) have a finer granularity. These techniques divide the files into chunks, and compute the hash value of each chunk. Finally, they compare the hash values of corresponding chunks between files and find out redundant chunks. The precision and the time complexity of these techniques depend on the division of chunks and the selection of hash functions.

Byte level techniques compare the files byte by byte. These techniques have highest detection precision, but meanwhile result in high time complexity. Therefore, they are not available for large files.

Detection granularity has a significant impact on the precision and efficiency of duplicate data detection techniques. Chunk level techniques make a well trade-off between performance and efficiency.

* Corresponding author. Tel.: +86 18696561721.
E-mail address: w_guiping@163.com (G. Wang).

0957-4174/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.eswa.2013.09.040

Among chunk level techniques, the traditional sliding blocking (SB) algorithm can be implemented without much effort, and its duplicate detection precision is relatively high. Therefore, the traditional SB algorithm is widely adopted in deduplication systems. However, this algorithm fails to detect redundant data in matching-failed segments resulting from inserting or deleting data segments in chunks. The precision of this technique will be improved once this issue is effectively solved.

Accordingly, this paper proposes a novel and efficient sliding blocking algorithm with backtracking sub-blocks called SBBS for duplicate data detection. For matching-failed segments, SBBS continues to detect duplicate data in sub-blocks, thus improving the detection precision.

The contributions of this paper are mainly listed as follows: (1) For the traditional SB algorithm, it clearly analyzes operations of inserting and deleting data in chunks, and deduces the concept of *matching-failed segment* due to such operations; (2) Based on the above analyses, it summarizes the principle of the proposed SBBS, i.e., improving duplicate data detection precision via backtracking the left/right 1/4 and 1/2 sub-blocks of matching-failed segments; (3) It designs and implements SBBS, which can detect duplicate data as many as possible in matching-failed segments.

Experimental results show that the proposed SBBS improves the duplicate detection precision by 6.5% compared with the traditional SB algorithm and by 16.5% compared with content-defined chunking (CDC) algorithm, and it does not increase much extra storage overhead when SBBS divides the files into equal chunks of size 8 kB.

The remainder of this paper is organized as follows. Section 2 introduces related work. For the traditional SB algorithm, Section 3 analyzes inserting and deleting data in chunks. The proposed SBBS is detailed in Section 4. Experiments and analyses are presented in Section 5. Finally, conclusions and future work are given in Section 6.

2. Related work

2.1. Deduplication & duplicate data detection

The ever-growing volume and value of digital information has raised a demand for long-term data protection through backup and archiving systems. Yet redundancy in data aggravates storage pressure in these systems. Deduplication is critical for improving the utilization of storage systems. Therefore, it receives sustained concern in literature.

The foremost step of deduplication is duplicate data detection. File level, chunk level and byte level duplicate detection techniques find out redundant data via data matching between files (e.g., Eshghi et al., 2005; Min, Yoon, & Won, 2011).

Besides data of traditional structures, a few solutions focus on duplicate data detection in more complex hierarchical structures (e.g., XML data). Leitao, Calado, and Herschel (2013) present a novel method for XML duplicate detection, called XMLDup. The most prominent feature of XMLDup is that it uses a Bayesian network to determine the probability of two XML elements being duplicates.

Duplicate is one modality of similar documents. The other modality is near-duplicate. Two documents are near-duplicates if one document is a modification of the other document. The modification can be insertion, deletion, or replacement of parts of the text (Lin, Liao, & Lee, 2013). A few solutions address detecting near-duplicate documents (de Carvalho, Laender, Goncalves, & da Silva, 2012; Lin et al., 2013).

Although new storage techniques have been applied, duplicate data detection is still a continuously concerned issue. For example, since solid state disks (SSD) based on NAND flash chip have been deployed as storage devices in computer systems including personal computers and server storage systems, many studies in literature address how to identify and eliminate duplicate regions in consideration of flash memory characteristics (Lim, 2011; Seo & Lim, 2010; Wu & Wu 2012).

This paper focuses on chunk level duplicate data detection techniques. In particular, it implements an optimization of the traditional SB algorithm.

2.2. Chunk level techniques

Chunk level techniques compute hash value of each chunk and try to find out redundant chunks. Chunk level techniques can be classified into three typical categories, which are fixed-sized partition (FSP), variable-sized partition (VSP), and sliding blocking (SB).

2.2.1. Fixed-sized partition

The FSP algorithm illustrated in Fig. 1 works as follows. The old file is divided into equal and non-overlapping chunks according to a pre-defined size. The hash value of each chunk is computed and stored in a table. When a new file is detected, it is divided into chunks in the same way as the old file is done. For each chunk in the new file, its hash value is computed. FSP then detects whether there is a same stored hash value or not. If yes, it marks this chunk as a redundant chunk. If not, it stores the new hash value and detects next chunk. Extra storage in FSP includes storage space for the table storing hash values and the disk addresses of associated chunks.

FSP has several advantages, such as easy implementation, rapid matching, etc. But it is sensitive to the operations of inserting and deleting data, even if only a small number of bytes are involved. Moreover, it is unable to optimize according to the contents of the files.

2.2.2. Variable-sized partition

A typical technique of VSP is content-defined chunking (CDC) (Bobbarjung, Jagannathan, & Dubnicki, 2006), also called content-based chunking (Eshghi et al., 2005). CDC divides files into variable-sized chunks based on files' content. For example, the Rabin's fingerprint (Broder, 1993) is used to partition files into chunks because it is efficient to compute over a sliding window (Bhagwat, Pollack, & Long, 2006; Eshghi et al., 2005; You et al., 2005). The size of each chunk is different, usually ranged between a maximum value and a minimum one (Denehy et al., 2003; Muthitacharoen, Chen, & Maziéres, 2001). The CDC algorithm illustrated in Fig. 2 works as follows.

(1) The old file is divided into variable-sized chunks according to the files' content in the same way as the new file is done, which is described below. Then the hash value of each chunk is computed and stored in a table.

(2) When a new file is detected, a sliding window of fixed-size is moved from the head of the new file to the end byte by byte. The

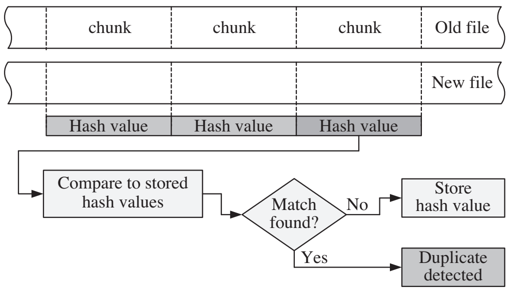

Fig. 1. Fixed-sized partition algorithm.