第 42 卷　第 2 期　　计　算　机　工　程　　2016 年 2 月
Vol. 42　No. 2　　Computer Engineering　　February 2016

·软件技术与数据库·　　文章编号：1000-3428(2016)02-0038-07　　文献标识码：A　　中图分类号：TP391.1

一种基于滑动分块的重复数据检测算法

郑亚光，潘久辉

（暨南大学信息科学技术学院，广州 510632）

摘　要：当被插入或删除的字节接近于匹配失败数据段两侧时，会导致 SBBS 算法回溯功能局部甚至完全失效。为此，提出一种改进的重复数据检测算法。采用滑动与滚动相结合的窗口移动模式减少窗口计算量，利用 Rsync 滚动校验和算法与 MD5 算法优化窗口计算模式，加快匹配速度。通过回溯匹配失败数据段，检测其中的重复数据段，以提升重复数据的检测精度。实验结果表明，与 SBBS 算法相比，该算法在重复数据段均匀分布与非均匀分布时的查全率分别提高约 4.32% 和 5.28% 。

关键词：重复数据检测；匹配失败数据段；SBBS 算法；窗口计算；校验和算法；回溯

中文引用格式：郑亚光，潘久辉. 一种基于滑动分块的重复数据检测算法[J]. 计算机工程，2016，42(2)：38-44.

英文引用格式：Zheng Yaguang，Pan Jiuhui. A Duplicate Data Detection Algorithm Based on Sliding Blocking[J]. Computer Engineering，2016，42(2)：38-44.

A Duplicate Data Detection Algorithm Based on Sliding Blocking

ZHENG Yaguang，PAN Jiuhui

（College of Information Science and Technology，Jinan University，Guangzhou 510632，China）

【Abstract】The backtracking function of Sliding Blocking Algorithm with Backtracking Sub-block（SBBS）becomes partial or even complete failure when the bytes are inserted or deleted，which are close to the two sides of the matching failure data segments. Aiming at the problem，this paper proposes an improved algorithm for duplicate data detection. This algorithm takes advantage of a new movement pattern which combines with gliding and rolling to reduce the computation overhead of window，and it improves the matching speed by adopting Rsync rolling checksum algorithm and MD5 algorithm. By backtracking matching failure data segments，it detects contained duplicate data segments to improve the detection accuracy of duplicate data. Experimental results show that，compared with SBBS，the recall ratio of this algorithm in uniform distribution and non uniform distribution of duplicate data segments are increased by 4.32% and 5.28% respectively.

【Key words】duplicate data detection；matching failure data segment；Sliding Blocking Algorithm with Backtracking Sub-block（SBBS）；window calculation；checksum algorithm；backtracking

DOI：10.3969/j.issn.1000-3428.2016.02.007

1　概述

数据质量问题在现实生活中是普遍存在的，其主要来源于企业数据录入以及数据集成时的差异，而数据清洗一直是提高数据质量的有效手段[1]。数据清洗的处理对象是脏数据，按其不同的表现形式可概括为残缺数据、错误数据、重复数据。其中，由多数据源合并而导致的数据重复是数据清洗的关键问题，因此重复数据检测便成为了研究的热点[2]。提高重复数据的检测精度，一方面能够进一步优化存储空间，消除分布在存储系统中的相同文件或者数据块；另一方面能进一步减少在网络中传输的数据量，从而降低能量消耗和网络成本，并为数据复制节省网络带宽。

针对由插入或删除少量字节而导致匹配失败数据段的问题，本文提出 SBBW 算法（Sliding Blocking Algorithm with Backtracking Window），通过回溯匹配失败数据段，从而检测重复数据段。

基金项目：公安部技术研究计划基金资助项目（2014JSYJB048）；武汉大学软件工程国家重点实验室开放基金资助项目（SKLSE2012-09-37）。

作者简介：郑亚光（1990－），男，硕士研究生，主研方向为数据清洗、信息集成；潘久辉，教授、博士生导师。

收稿日期：2015-01-23　　**修回日期：**2015-03-19　　**E-mail：**2902049327@qq.com

2　相关工作

现有的研究根据检测粒度可划分为 3 个等级:文件级,块级,字节级。文件级的检测技术[3]是通过计算整个文件的哈希值从而判定 2 个文件是否重复,此方法适用于大文件,但检测精度太低,即使对文件作轻微的修改,也会导致哈希值的变化。块级的检测技术[4-5]是通过将文件分块,计算和比较 2 个文件的数据块的哈希值,从而寻找重复数据块。字节级的检测技术[6-7]是通过比较 2 个文件的字节从而计算文件的相似度,此方法检测精度高,但通常具有较高的时间复杂度,不利于大文件检测。由此可知,块级的检测技术在算法效率和检测精度之间达到了平衡。

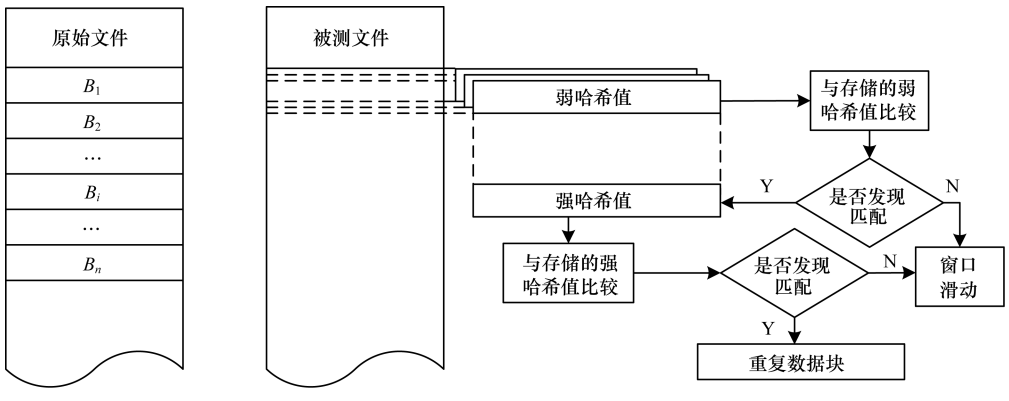
尽管目前存在许多块级别的重复数据检测方案,但解决插入问题(在原来的数据流中的某处插入少量新字节,其他部分不变)和删除问题(在原来的数据流中的某处删除少量字节)的研究依然匮乏。文献[8-9]都是基于 Rabin 指纹算法对数据进行分块,插入或删除小部分字节都只会影响一二个数据块,但由于是基于内容的划分,数据块的大小不一导致了数据块的划分长度难以掌握。文献[10]提出的滑动分块算法是通过强、弱哈希算法结合的方式计算滑动窗口内每个重叠块的哈希值,此算法可以很好地解决插入和删除问题,但是无法检测出匹配失败数据段中的重复数据段。文献[11]在滑动分块算法的基础上添加回溯子块的功能,提出 SBBS 算法可以检测出由插入、删除问题导致的匹配失败数据段中的部分重复数据段,但当插入(删除)的字节接近匹配失败数据段的两侧时会导致回溯功能的局部甚至完全失效。

本文提出的 SBBW 算法是通过改变窗口的移动模式,采用滑动与滚动相结合的窗口移动模式,降低滑动分块算法和 SBBS(Sliding Algorithm with Backtracking Sub-block)算法的窗口计算量。利用回溯窗口,以强、弱哈希算法相结合的计算模式检测匹配失败数据段中的重复数据段。

3　SBBS 算法分析

3.1　由插入、删除问题导致的匹配失败数据段

在滑动分块算法中,原始文件被划分成长度相同且互不重叠的数据块,并将计算出的每个数据块的强哈希值和弱哈希值存储在二维表中。被测文件用一个与原始文件分块长度相同的滑动窗口来计算每个重叠块的弱哈希值,并与先前存储的值进行比较,若匹配,则进一步计算窗口内数据块的强哈希值;若依然匹配,则为重复数据块;否则窗口继续滑动 1 Byte,直至窗口到达文件末尾,其处理流程如图 1所示。



(a)原始状态　　(b)滑动分块算法

图 1　滑动分块算法的处理流程

若在原始文件中插入、删除某些数据段而导致原始划分界线发生变化,那么包含这些插入、删除数据段的数据块称为由插入、删除问题导致的匹配失败数据段。滑动分块算法无法检测此类匹配失败数据段中的重复数据段,具体说明如下:

假设 B_{i-1},B_i,B_{i+1}是原始文件划分出的 3 个数据块,如图 2(a)所示。现将 B_i 中插入一个长度为 d 的数据段,则 B_i 与随后的所有数据块的下边界整体下移 d 个距离。在滑动分块算法中,当滑动窗口的下界从 B_{i-1}的下界开始移动时将无法检测出 B_i 为重复数据块,直到滑动窗口的下界到达 B_{i+1}的原始下界时,才可以与原始文件的 B_{i+1}匹配,B_i 就成为了匹配失败数据段,如图 2(b)所示。

在 SBBS 算法中,首先计算原始文件的每个数据块以及各数据块的 1/2 上子块、1/2 下子块、1/4 上子块、1/4 下子块的哈希值,并将它们存储起来,划分情况如图 3(a)所示。然后按照滑动分块算法计算被测文件中每个重叠块的哈希值,若窗口检测出 B_{i+1}为重复数据块,则通过回溯子块的方式检测匹配失败数据段 B_i:从 B_{i-1}的下界开始先计算 B_i 的

1/4上子块的哈希值,若 B_i 中 1/4 上子块与原始文件中 B_i 的 1/4 上子块匹配成功,再进一步匹配 B_i 中 1/2 上子块的哈希值,否则终止对 B_i 上子块的检测;同理,B_i 下子块的检测从 B_{i+1} 的上界开始。从图 3(b)可以看出,SBBS 算法可检测出 B_i 中 1/4 上子块和 1/2 下子块为重复数据段。

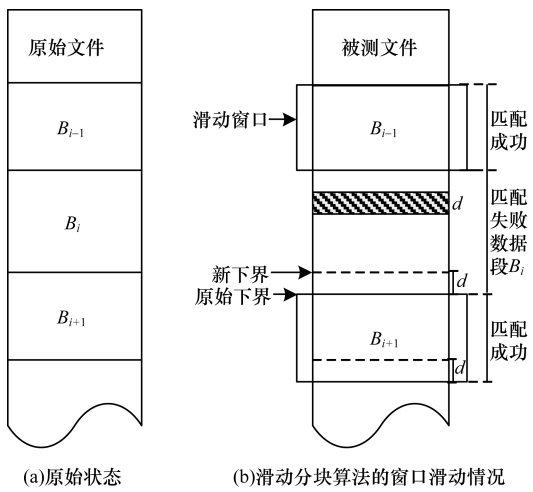


图 2 原始文件插入数据段后的边界变化

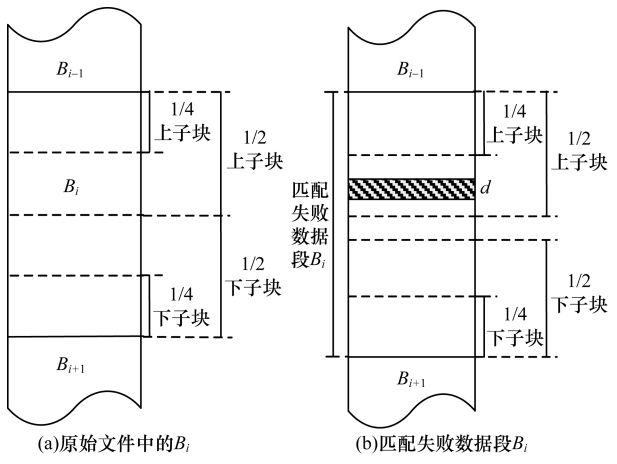


图 3 SBBS 算法对 B_i 的划分

因为删除数据段的情况与插入数据段的情况类似,所以不再赘述。SBBS 算法不仅保持了滑动分块算法的检测精度,而且可以检测出由插入、删除问题导致的匹配失败数据段中的部分重复数据段,但该算法依然存在局限性。

3.2 SBBS 算法的局限性

假设原始文件中数据块的划分长度为 *size*,在数据块 B_i 的上下 1/4 子块中分别插入长度分别为 m 和 n 的数据段。SBBS 算法会对 B_i 中上下子块进行检测,因为算法分别是从 B_{i-1} 的下界和 B_{i+1} 的上界开始计算 1/4 子块的哈希值,当 1/4 子块匹配成功时才会检测 1/2 子块。从图 4(a)可以看出,SBBS 算法对 B_i 的检测完全失效。换句话说,只要插入的数据段位于 B_i 的 1/4 上下子块中,SBBS 算法就会检测失败。对于插入的数据段而言,若 B_i 中上下子块中插入位置的概率相同,SBBS 算法通过回溯子块的方式检测出 B_i 中上下子块的重复数据段的概率只有 50% 。

因为插入或删除的数据段只影响所在子块的内容,其他子块仅是原始边界发生了变化,其内容均未改变,所以在 SBBW 算法中,首先计算原始文件的每个数据块以及数据块的各 1/4 子块的哈希值。对于匹配失败数据段 B_i,另设置一个数据块 1/4 长度的回溯窗口,滑距为 1 个字符,让其从 B_{i-1} 的下界移动到 B_{i+1} 的上界,回溯窗口在移动过程中计算窗口内数据段的哈希值,并与先前存储的 B_i 中各 1/4 子块的哈希值比较,若匹配成功,则为重复数据段。从图 4(b)可以看出,该方法可以检测出 B_i 中 1/2 重复数据段。

当在原始文件的 B_i 的上 1/4 子块中删除长度为 $d(d < size/4)$ 的数据段时,从图 5(a)可以看出,SBBS 算法只能检测出 B_i 中 1/2 下子块为重复数据段,对 1/2 上子块的检测是失效的。而从图 5(b)可以看出,采用 SBBW 算法可以检测出 B_i 中 3/4 的重复数据段。

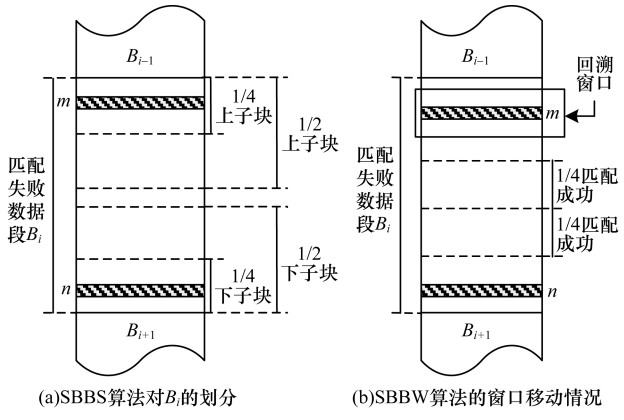


图 4 插入数据段位于 1/4 子块内的情况

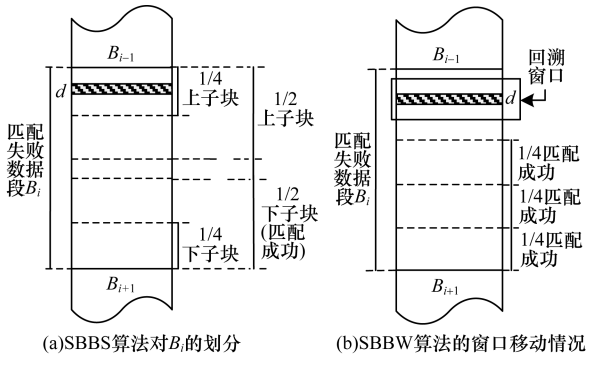


图 5 删除数据段位于 1/4 子块内的情况

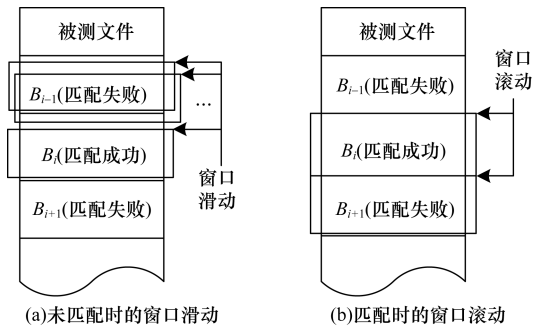
由以上分析可知:在匹配失败数据段 B_i 中,所插入或删除数据段的位置直接影响到 SBBS 算法的检测精度。当插入或删除的数据段位于 1/4 上下子块内时,就会导致 SBBS 算法的回溯功能局部甚至完全失效,而 SBBW 算法克服了 SBBS 算法的局限性,虽然 B_i 未能够被完全匹配,但进一步提高了 SBBS 算法的检测精度。

4　SBBW 算法

4.1　窗口移动模式

SBBW 算法中包含了 2 个窗口,一个是用于检测重叠块的窗口,长度与原始文件中数据块的长度相同;另一个是用于检测匹配失败数据段中的重复数据段的回溯窗口,长度是原始文件中数据块长度的 1/4。

为了减少滑动分块算法和 SBBS 算法在窗口滑动时的计算量,SBBW 算法中的窗口都采用滑动与滚动结合的移动模式。假设 B_{i-1} 和 B_{i+1} 是非重复数据块,B_i 是重复数据块,当 B_{i-1} 的哈希值匹配失败时,窗口将向下滑动 1 Byte,直至滑动到 B_i,如图 6(a)所示。当 B_i 的哈希值匹配成功时,窗口将直接滚动到下一个数据块 B_{i+1} 上计算其哈希值,如图 6(b)所示。如此循环,直至窗口到达被测文件的末尾。



(a)未匹配时的窗口滑动　　(b)匹配时的窗口滚动

图 6　窗口移动模式

由此可知,若文件被划分成 m 个数据块,数据块的大小为 2 KB,其中,重复数据块有 n 个($n \leqslant m$)。当 SBBS 算法和 SBBW 算法的滑距为 1 Byte 时,在不考虑匹配失败数据段的前提下,SBBS 算法需要计算约 2 048($m-1$)次哈希值,而 SBBW 算法仅需要计算约 2 048($m-n-1$) + n 次哈希值。

4.2　哈希校验算法

哈希校验通过把数据块压缩成数字指纹,大幅降低了字符匹配的时间,为了进一步提高数据块之间以及匹配失败数据段中数据段的匹配速度,SBBW 算法采用了强、弱哈希算法结合的计算模式。虽然弱哈希算法发生散列冲突的概率较高,但是可以快速地识别非重复数据,而强哈希算法却可以准确地识别重复数据。对窗口下的数据块(段)进行弱哈希校验,若与原始文件中数据块(段)的弱哈希值相同,则对该数据块(段)继续进行强哈希校验,若依然相等,则匹配成功;否则匹配失败。

SBBW 算法采用 Rsync 滚动校验和算法[12]作为检测重叠块和匹配失败数据段的弱哈希函数。计算公式具体如下:

$$a(k,l) = \left(\sum_{i=k}^{l} X_i\right) \bmod M \tag{1}$$

$$b(k,l) = \left(\sum_{i=k}^{l} (l-i+1)X_i\right) \bmod M \tag{2}$$

$$s(k,l) = a(k,l) + 2^{16} b(k,l) \tag{3}$$

其中,X_i 表示第 i 个数据块;$s(k,l)$ 是表示从第 k 个~第 l 个数据块的 Rsync 滚动校验和。当求第 $k+1$ 个~第 $l+1$ 个数据块的 Rsync 滚动校验和时,公式具体如下:

$$a(k+1,l+1) = (a(k,l) - X_k + X_{l+1}) \bmod M \tag{4}$$

$$b(k+1,l+1) = (b(k,l) - (l-k+1)X_k + a(k+1,l+1)) \bmod M \tag{5}$$

$$s(k+1,l+1) = a(k+1,l+1) + 2^{16} b(k+1,l+1) \tag{6}$$

由此可知,Rsync 滚动校验和算法能够从文件的任意位置开始计算滚动校验值,并且后续校验值可以由当前校验值通过递推关系获得,所以,可以高效地计算连续数据块的校验值,大幅减少窗口计算量。

SBBW 算法采用 MD5 滚动校验和算法作为检测重叠块和匹配失败数据段的强哈希函数。MD5 算法是以 512 bit 分组来处理输入的信息,算法输出是由 4 个 32 bit 分组级联后生成的一个 128 bit 散列值,散列冲突的概率极低。

4.3　SBBW 算法流程及其性能分析

SBBW 算法首先计算并存储原始文件的每个数据块以及数据块的各 1/4 子块的校验值。然后利用一个长度与原始文件中数据块长度相同的窗口按照滑动与滚动结合的移动模式计算被测文件中每个重叠块的哈希值,若窗口检测出 B_{i+1} 为重复数据块,则利用回溯窗口来检测匹配失败数据段 B_i。回溯窗口的长度是原始文件中数据块长度的 1/4,滑距为 1 Byte,依据上述移动模式从 B_{i-1} 的下界移动到 B_{i+1} 的上界,计算匹配失败数据段中数据段的哈希值。窗口在移动过程中都要先计算 Rsync 滚动校验值,并与先前存储的值比较,若匹配,则进一步计算

MD5 值;若依然匹配,则为重复数据块(段);否则窗口继续移动。由此可知,算法在处理重叠块和匹配失败数据段的流程相同,SBBW 算法检测匹配失败数据段的伪代码具体如下:

输入 匹配失败数据段 B_i

输出 0 个 ~ 3 个数据块 1/4 长度的重复数据段

```
Begin
1. 新建数据块 1/4 长度且滑距为 1 Byte 的回溯窗口;
2. 回溯窗口的上界寻找数据块 B_{i-1} 的下界;
while(窗口的下界小于等于 B_{i+1} 的上界){
3. 计算窗口内数据段的 Rsync 滚动校验值;
4. 与原始文件中 B_i 的 1/4 子块的 Rsync 滚动校验值进行比较;
if(相等){
5. 计算窗口内数据段的 MD5 值;
6. 与原始文件中 B_i 的 1/4 子块的 MD5 值比较;
if(相等){
7. 失败数据段 B_i 中该 1/4 数据段匹配成功;
8. 窗口直接滚动到下一个 1/4 数据段;
}else{
9. 窗口向下滑动 1 Byte;
}
}else{
10. 窗口向下滑动 1 Byte;
}
}
End
```

滑动分块算法对原始文件数据块的划分长度直接决定了数据块数量、窗口大小以及被测文件中滑动窗口的计算量。SBBW 算法不仅如此,而且还决定了检测匹配失败数据段时的回溯窗口长度以及回溯窗口的计算量。SBBW 算法对匹配失败数据段的处理流程如图 7 所示。

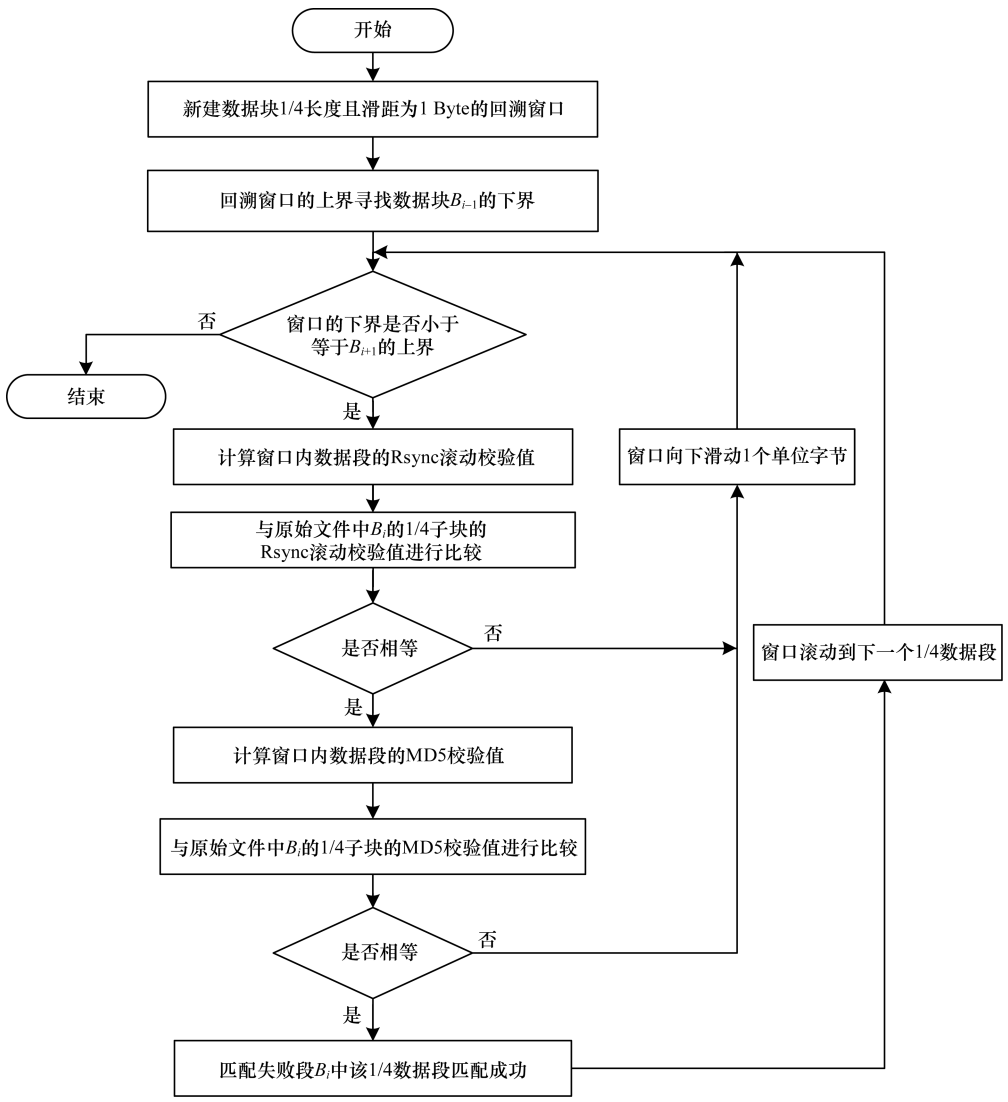


图 7 SBBW 算法对匹配失败数据段的处理流程

在 4.1 节中已经讨论了 SBBW 算法在不考虑匹配失败数据段时的计算复杂度,本节主要讨论检测匹配失败数据段时的计算复杂度。设原始文件中数据块的长度为 *size*,插入或删除数据段的长度为 d($d < size/4$),具体分析如下:

(1)若 B_i 有 3/4 个子块匹配成功,至多增加($size/4$) $\pm d + 2$ 次 Rsync 滚动校验值的计算、3 次 MD5 值的计算。

(2)若 B_i 有 1/2 个子块匹配成功,至多增加($size/2$) $\pm 2d + 1$ 次 Rsync 滚动校验值的计算、2 次 MD5 值的计算。

(3)若 B_i 有 1/4 个子块匹配成功,至多增加($3size/4$) $\pm 3d$ 次 Rsync 滚动校验值的计算、1 次 MD5 值的计算。

(4)若 B_i 所有子块匹配失败,增加 $size \pm 4d$ 次 Rsync 滚动校验值的计算。

对于约 8 KB 匹配失败数据段,当插入或删除的数据段接近原始边界时,SBBW 算法将至少匹配约 4 KB的重复数据段,而 SBBS 算法却至多匹配约 4 KB。在计算量上,由于滑动分块算法的窗口是按字节滑动的,需要进行约 8 000 次的哈希计算,而 SBBW 算法只需要在低于 8 000 次的基础上进行平均 $0.625size + 2.5d + 2.25$ 次哈希计算就可以匹配至少 4 KB 的重复数据段。

5　实验结果与分析

5.1　实验环境

实验将 SBBW 算法、SBBS 算法、滑动分块算法作比较,评估 SBBW 算法的检测精度和算法效率。硬件环境:主机为 CPU Intel Pentium Dual 2.0 GHz,内存 2 GB,Windows XP。测试环境:Eclipse3.7,JDK1.6 版本,数据库为 Oracle11g。实验对象:将大小为 25 165 KB 的日志文件作为原始文件,通过在其中插入、删除、修改原始文件中的记录,得到 2 个修改文件,其信息如表 1 所示。

表 1　修改文件的相关信息

场景	修改文件大小/KB	修改文件中重复数据/KB	重复数据段分布情况
场景 1	50 762	28 526	2 KB ~ 32 KB 均匀分布
场景 2	51 028	30 106	2 KB ~ 32 KB 非均匀分布

5.2　查全率对比

设 3 种算法的窗口滑距全部设为 1 Byte,并且 SBBW 算法回溯时的窗口滑距也为 1 Byte,依次选择 2 KB,4 KB,8 KB,16 KB,32 KB 作为数据块的划分长度,观察 3 种算法在重复数据段均匀分布和非均匀分布时的查全率。查全率的计算公式如下:

$$\text{查全率} = \frac{\text{正确识别的重复数据量}}{\text{实际的重复数据量}} \tag{7}$$

从图 8 可以看出,在重复数据段均匀分布时,3 种算法的查全率呈现平稳下降趋势。SBBW 算法的查全率比 SBBS 算法平均高约 4.32%,比滑动分块算法平均高约 9.28%。

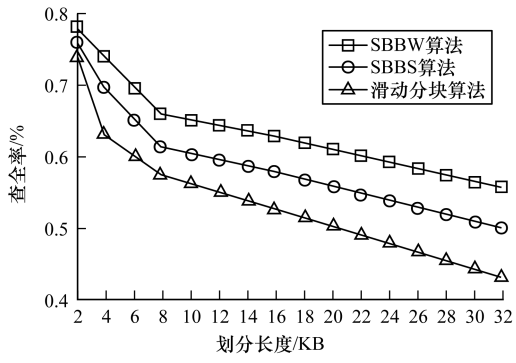


图 8　3 种算法在重复数据段均匀分布时的查全率

从图 9 可以看出,在重复数据段非均匀分布时,3 种算法的查全率呈现不规则下降趋势。SBBW 算法的查全率比 SBBS 算法平均高约 5.28%,比滑动分块算法平均高约 9.77%。

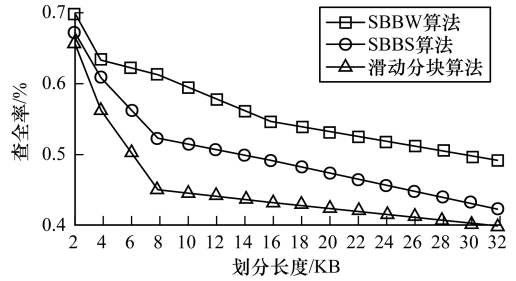


图 9　3 种算法在重复数据段非均匀分布时的查全率

5.3　额外存储开销对比

设原始文件的数据量为 $S_{\text{original\_file}}$,由原始文件计算出哈希值的数据量为 $S_{\text{strong\_hash}} + S_{\text{weak\_hash}}$;强哈希值位数为 $B_{\text{strong\_hash}}$,弱哈希值位数为 $B_{\text{weak\_hash}}$;原始文件中数据块划分长度为 *size*,数据块数量为 n;每个数据块下子块数量为 m,子块的强哈希值位数为 $B'_{\text{strong\_hash}}$,子块的弱哈希值位数为 $B'_{\text{weak\_hash}}$。额外的存储开销计算公式如下:

$$\begin{aligned}\text{额外存储开销} &= \frac{S_{\text{strong\_hash}} + S_{\text{weak\_hash}}}{S_{\text{original\_file}}} \\ &= \frac{n \times [B_{\text{strong\_hash}} + B_{\text{weak\_hash}} + (B'_{\text{strong\_hash}} + B'_{\text{weak\_hash}}) \times m]}{n \times size} \\ &= \frac{B_{\text{strong\_hash}} + B_{\text{weak\_hash}} + (B'_{\text{strong\_hash}} + B'_{\text{weak\_hash}}) \times m}{size}\end{aligned} \tag{8}$$

由此可知,3 个算法的存储开销与原始文件的数据块数量无关,图 10 适用于均匀分布和非均匀分布 2 个场景。由于 SBBW 算法与 SBBS 算法选择的哈

希算法所计算出的哈希值的位数相同,并且所划分的子块数量 m 均为 4,因此 SBBW 算法和 SBBS 算法的额外存储开销也是相同的。

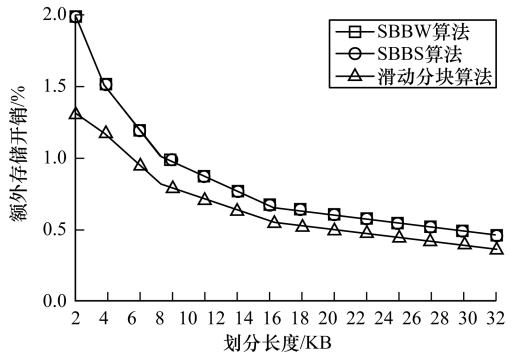


图 10 3 种算法的额外存储开销

从图 10 可以看出,3 种算法的额外存储开销之间的差距在不断缩小。当划分长度在[2 KB,32 KB]时,SBBW 算法、SBBS 算法与滑动分块算法的存储开销差距也从 0.668% 下降至 0.098% 。

实验结果表明,随着数据块划分长度的不断增加,在与滑动分块算法的额外存储开销的差距不断缩小的同时,SBBW 算法与另外 2 种算法的查全率的差距却在扩大。虽然具有相同的额外存储开销,但是 SBBW 算法的查全率要高于 SBBS 算法。

6 结束语

滑动分块算法虽然能够很好地解决插入问题和删除问题,但是无法检测由插入、删除问题导致的匹配失败数据段中的重复数据段,而 SBBS 算法在插入或删除的字节接近匹配失败数据段的两侧时会导致回溯功能的局部甚至完全失效。为此,本文提出的 SBBW 算法在窗口的移动模式和计算模式上对 2 个算法进行改进,不仅可提高检测精度,而且解决了SBBS算法所存在的问题。同时对由插入或删除少量字节导致的匹配失败数据段的重复数据检测具有一定的参考作用。

参考文献

[1] 叶 鸥,张 璟,李军怀. 中文数据清洗研究综述[J]. 计算机工程与应用,2012,48(14):121-129.

[2] 叶焕倬,吴 迪. 相似重复记录清理方法研究综述[J]. 情报分析与研究,2010,(9):56-66.

[3] 赵晓永,杨 扬,王 宁. 基于声学指纹的海量 MP3 文件近似去重方法[J]. 计算机工程,2013,39(7):73-75.

[4] 李振兴,刘 波. 基于 Hadoop 平台的 XML 文档重复数据检测[J]. 计算机系统应用,2013,22(11):195-199.

[5] 张 敏. 海量数据的 MapReduce 相似度检测[J]. 实验室研究与探索,2014,33(9):132-136.

[6] 孙 娜,吴兰兰. 一种节点加权的相似重复 XML 数据检测算法[J]. 计算机光盘软件与应用,2014,17(2):99-100.

[7] Ripon K S N, Rahman A. A Domain-independent Data Cleaning Algorithm for Detecting Similar-duplicates[J]. Journal of Computers,2010,5(12):1800-1809.

[8] 敖 莉,舒继武,李明强. 重复数据删除技术[J]. 软件学报,2013,21(5):916-929.

[9] 孙爱玲,冉禄纯. 一种基于重复数据删除的网络文件备份系统设计与实现[J]. 计算机应用与软件,2014,31(10):86-90.

[10] Lee Woo-joong, Park C. An Adaptive Chunking Method for Personal Data Backup and Sharing[C]//Proceedings of the 8th USENIX Conference on File and Storage Technologies. San Antionio, USA: USENIX Association, 2010:758-762.

[11] Wang Guiping, Chen Shuyu, Lin Mingwei. SBBS: A Sliding Blocking Algorithm with Backtracking Subblocks for Duplicate Data Detection[J]. Expert Systems with Applications,2014,41(5):2415-2423.

[12] Tridgell A. Efficient Algorithms for Sorting and Synchronization[D]. Canberra, Australia: Australian National University, 1999.

编辑 陆燕菲

(上接第 37 页)

[9] Chu C K, Chow S S M, Tzeng W G, et al. Key-aggregate Cryptosystem for Scalable Data Sharing in Cloud Storage[J]. IEEE Transactions on Parallel and Distributed Systems,2014,25(2):468-477.

[10] Boneh D, Waters B, Zhandry M. Low Overhead Broadcast Encryption from Multilinear Maps[C]//Proceedings of Cryptology-CRYPTO'14. Berlin, Germany: Springer, 2014: 206-223.

[11] Boneh D, Silverberg A. Applications of Multilinear Forms to Cryptography[J]. Contemporary Mathematics, 2003,324:71-90.

[12] Garg S, Gentry C, Halevi S. Candidate Multilinear Maps from Ideal Lattices[C]//Proceedings of Cryptology-EUROCRYPT'13. Berlin, Germany: Springer, 2013: 1-17.

[13] Kate K, Potdukhe S D. Data Sharing in Cloud Storage with Key-aggregate Cryptosystem[J]. International Journal of Engineering Research and General Science,2014,2(6): 882-886.

[14] 赖俊祚. 可证安全的公钥加密和无证书公钥加密的研究[D]. 上海:上海交通大学,2009.

[15] 杨坤伟,李顺东. 一种基于身份的匿名广播加密方案[J]. 计算机工程,2014,40(7):97-101.

[16] Canetti R, Halevi S, Katz J. Chosen-ciphertext Security from Identity-based Encryption[C]//Proceedings of Cryptology-Eurocrypt'04. Berlin, Germany: Springer, 2004:207-222.

编辑 陆燕菲