基于数据位图的滑动分块算法

邓雪峰,孙瑞志,张永瀚,聂娟

(中国农业大学农业部农业信息获取技术重点实验室 北京 100083)
(北京农学院计算机与信息工程学院 北京 100083)
(dxf75@sohu.com)

Sliding blocking algorithm based on data bitmap

Deng Xuefeng, Sun Ruizhi, Zhang Yonghan, Nie Juan
(Key laboratory of Agricultural information acquisition technology（Beijing）,Ministry of Agriculture P.R.China, China Agricultural University, Beijing, 100083)
(College of Computer and Information Engineering in Beijing University of Agriculture, Beijing, 102206 China)

Abstract During similar data synchronization and storage, data blocking is an important step to detect duplication of data. Only after effective data blocking can you find difference between data accurately. This paper first summarizes and analyzes the methods of data blocking, then re-organize data files in the form similar to bitmap based on the sliding blocking algorithm. After that we read data bitmap by column to form a new data chunk and compute fingerprint information of the column. To improve its ability to locate difference in data, fingerprint of column serve as supplement for sliding bolcking algorithm to acquire more accurate information of data difference. Experimental results show that this method is better than sliding blocking algorithm in data duplication detection under the same conditions.

Key words Sliding blocking algorithm; Duplicate data detection; data bitmap; data difference; data synchronization

摘 要 网络中相似的数据文件进行同步与存储的过程中，对数据进行分块，是检测数据重复的重要步骤之一，在有效的对数据分块的基础上才能更准确的定位数据间的差异部分。本文就数据分块方法予以分析总结，在滑动分块算法的基础上，重新将数据文件组织成类似位图的排列形式，对数据位图以列向读取数据信息，形成新的数据分块，并计算列向读取数据的分块指纹信息，以列向数据指纹为补充校正滑动分块算法定位差异数据的能力的不足之处，从而获得更精确的数据差异信息。经实验证明，本方法在同源文件的数据重复检测中效果好于相同条件下的滑动分块方法。

关键词 滑动分块算法；重复数据检测；数据位图；数据差异；数据同步
中图法分类号 TP31；TP39

随着大数据时代的到来，大量的数据将通过网络进行更新与存储。在这些数据文件中存在着大量的同源或类似文件。对同源的数据文件进行同步更新的过程中如何降低网络的占用；对内容大部分相同的文件如何利用更少的存储空间进行存储一直是计算机领域研究的热点问题之一。

同源或内容相近的数据文件进行网络同步过程中，经常采用基于差异的数据传输方法。这种方法是低带宽网络中文件同步的高效方案之一，利用这个技术可以减少网络流量的开销，同时也能提升数据同步更新的速度。利用这种技术为核心定制了相当多的应用系统[1]。该技术在文件的传输过程中，非常重要的一个处理过程就是将数据分块，数据分块的方法是提升查找差异数据精确度的

收稿日期：年-月-日*投稿时不填写此项*；最终修改稿收到日期：年-月-日 *投稿时不填写此项*.
基金项目：中国农业大学研究生科研创新专项基金（2013YJ008）

重要手段。同源或内容相近的文件存储采用的去重技术，也是一个重要的研究方向。存储类似数据文件的方案中，对相同内容的数据去重处理时最重要的步骤也是将数据进行分块，这种数据分块方法不同于基于差异传输数据时的数据分块方案，一般是采用基于内容的数据分块技术。但基于差异的数据传输中用到的核心分块方案，固定数据块大小的滑动分块技术同样也可以应用于该领域。

因此，数据分块技术是对同源或内容相近的数据文件进行同步更新与去重存储的必要技术。本文在分析了各种数据分块技术的基础上，以基于差异的数据传输模型为应用场景，提出了一种基于数据位图的分块方案，这个方案提升了查询内容相近数据文件中相同数据内容的能力，应用于差异数据传输方案中将在不提升数据指纹交换次数的情况下直接达到多轮交换数据指纹的效果，且此方案也可以用于数据存储领域中，用于更精确的检测文件重复信息。

1 相关研究

数据分块技术把数据文件划分为较小的数据块，通过计算这些小数据块的哈希值，将其作为数据指纹，对比不同的内容相似的数据文件中小数据块的指纹信息，从而确定不同的内容相近的数据文件中的重复数据块。利用检测出的结果可以对数据进行基于差异的传输与去重存储等操作，这样可以只传输或存储数据文件中差异部分的数据，减少网络带宽与存储空间的占用。对数据文件进行分块的方法主要有固定分块技术(Fixed-Sized Partition, FSP)，可变分块技术(Variable-Sized Partition, VSP)和滑动块技术(Sliding Block)[2][3]。

固定分块技术（FSP）是把数据流按固定长度的字节数进行切割，然后对固定长度的数据分块计算其数据指纹的方法。

可变分块技术（VSP）主要代表是基于内容可变长度分块(Content-Defined Chunking, CDC)算法。CDC 算法[4]利用一个固定的滑动窗口为边界切分数据文件的数据，形成可变长的数据分块，滑动窗口顺序的从数据文件的起始位置部向后逐一字节滑动，当滑动窗口的 hash 值（CDC 算法中采用的是 Rabin[5]指纹）与预设的值匹配时，就产生一个分块。这个数据块的长度被指定在一个区间范围内获取。

滑动块技术(Sliding Block,SB)是在文献[6]中被提出的，在文中提出了一种用于同步远程内容相近的数据文件的同步算法，即 Rsync 算法，在 Rsync 算法中提出了一种对划分的数据块进行滑动分块的数据分块技术。这种数据分块方法以固定数据分块技术为基础，利用滑动块技术更准确的获得较小的数据分块的位置信息，从而在一定程度上规避了在内容相似文件中对插入、删除等操作对固定分块技术查找重复数据影响较大的问题。

在数据分块的分析中，利用数据分块技术更精确的获取差异数据信息是一个研究的重点内容。在数据文件同步方面对数据分块技术的改进主要集中于多轮(多次交换指纹信息)同步算法[7]~[10]，算法对数据分块采用逐渐减小数据块的粒度的方法，更精确的获取差异数据的在数据文件中的位置，减少传输差异数据的带宽占用。其中固定分块技术与基于内容的可变长分块技术都被利用到其中，用于缩小数据分块粒度，例如文献[7]中将基于内容的分块方法与滑动分块技术结合，形成两轮数据指纹交换，从而确定同源文件的差异信息。文献[11]中对于文件信息交换的次数有一个界定，这个结论说明不可以无限次的交换文件的指纹信息，一般来说多轮同步算法在实际应用中由于信息的交换在网络上进行，出于对系统稳定性的考虑以及具体的节省带宽的效果两方面原因，在实际中应用还比较少。

2 数据分块技术分析

本节对 FSP、VSP 及以 Rsync 算法为原型的 SB 等几种对数据进行分块的方法进行分析，总结利用各种数据分块技术对数据文件分割产生的问题。

FSP 分块方法中新文件示意图如图 1 所示的排列形式，假设新文件为一个顺序排列的字符组成，按数据流顺序排列成图中所示的字符串，固定分块技术将文件按固定的分块

方式拆分成数据块，而旧文件以同样的长度拆分数据，图 1 中，假设在数据中，分块长度为图中示意的间隔长度，新文件分块为 ABCD、EFGH、IJKL……，不妨设待更新的旧文件与新文件有如下差异，在第二块数据由 EF 与 GH 间插入新的数据 23、在第四块 MNOP 前插入 4、在第 7 块 YZ12 中删除 Z、在第 9 块 7890 中删除 0。利用 FSP 分块的方法，在图 1 中所示的情况下，第二块数据由于插入引起数据的改动直接影响到其后的分块，待更新的旧文件中只有第 1 块数据块可以在新文件中找到对应的相同数据分块，新文件中第 2 块数据块其后的数据块均无法在旧文件中找到相同的数据分块。

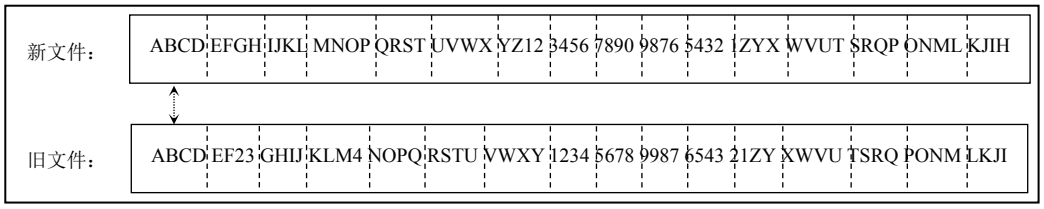

图 1 FSP 数据分块方法示意图

固定分块的方法对于文件的更新，尤其是插入或删除操作是非常敏感的，在数据文件某个位置发生改变后，其后的所有的分块内容都将受到影响，从而使相似文件的匹配度降低。因此在实际应用中利用这种技术实现的系统不常见。

VSP 分块方法示意图如图 2 所示，数据改变方式同前文中 FSP 数据分块方案中的数据变动情况，假设在示意图中（E，F）、（K，L）、（P，Q）、（U，V）、（Z，1）、（6，7）、（6，5）、（1，Z）、（U，T）、（Q，P）、（L，K）等处获得的数据内容指纹相同。分块形式如图 2 所示。

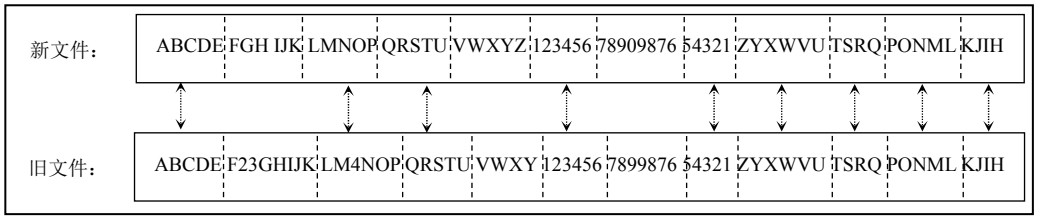

图 2 VSP 数据分块方法示意图

在 VSP 分块方法分割数据时，可以以内容区别分块的边界，这种方案可以把仅被改动过的分块区分出来，并且如果不是修改了边界的内容，可以达到仅识别修改后的区块的目的，检测精确度较固定分块技术有极大的提升，也一定程度上消除了插入及删除的影响。

以 Rsync 算法为原型的 SB 数据分块方法进行数据分块对比结果的示意图如图 3 所示，由于采用了滑动分块的方法查找待更新的旧文件中的区块，可以将固定数据分块的缺点得以消除，在对插入与删除的情况中，利用区块窗口滑动的方法可以消除修改部分对后续分块分割的影响，提升了固定分块技术的现实可用性。

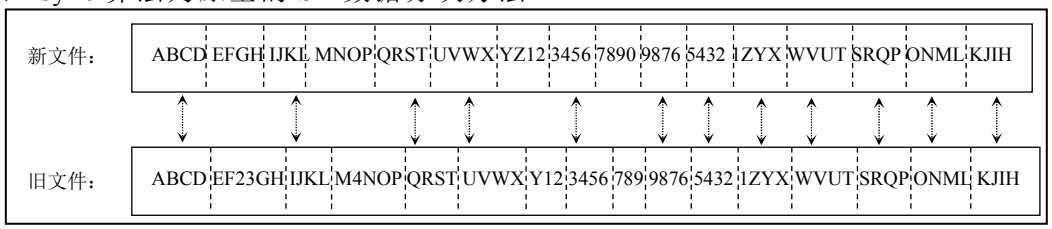

图 3 以 Rsync 算法为原型的 SB 数据分块方法示意图

各种分块方案中，固定分块技术由于受文件修改影响比较大，在具体应用中不常用；

而以内容分块的技术由于文件内容的不确定性，经常会出现区块边界范围超出给定长度的问题，在实际应用中，超过实际长度限制的区块只能被动的被截断[2]，这个问题一直没有得到较好的解决；滑动分块方案中，滑动块可以提升固定分块技术对插入，删除等操作的影响，但是，如何获得更精确的数据块差异，一直是滑动分块方案讨论的问题，例如利用增加同步轮数，逐渐减小固定分块的长度的方案来逐步获取更精确的差异数据[7]~[10]，利用回溯的方法最大化的找到已找到的差异数据块中的相同数据[12]等方案一直是学术领域讨论的问题，但在实际系统中还是以 Rsync 算法为原型的 SB 数据分块方法或和 VSP 分块方案中具有代表性的 CDC 算法应用最为广泛。

3 基于数据位图的分块方法

由于滑动分块技术广泛的应用于远程文件的同步系统中，因此基于滑动分块技术的基础，本文中提出一种其改进策略。

3.1 算法原理分析

在 Rsync 算法中提出了数据滑动分块技术，Rsync 算法的核心思想为把待传输数据进行数据分块，将分块后的数据进行数据指纹的计算，通过网络传送数据指纹到远端，远端系统利用滑动分块技术检测远端数据文件与本地数据文件的差异，最后传送差异部分的数据，完成文件的同步更新。

由本文第 2 节对滑动分块技术的分析中可以看出，在待传输数据的新文件端，对数据文件用固定大小的分块划分数据块，这个过程中，固定端的数据分块大小确定的情况下，可能新文件数据与待更新的旧文件中差异的部分恰好均匀分布于数据文件的不同数据分块中，这时可以造成虽然整个文件的修改非常少的情况下，在待更新的旧文件的一端无法找到大量的相似数据，这样的问题同样存在于其它的分块方法中。所以，在滑动分块的改进方案中，为了能找到更多的相似数据，改进主要的方案是利用多次同步数据，逐渐缩小分块，最后达到更多的检测出相似文件中差异数据的目的，而这样的代价是增加了网络传输的次数。

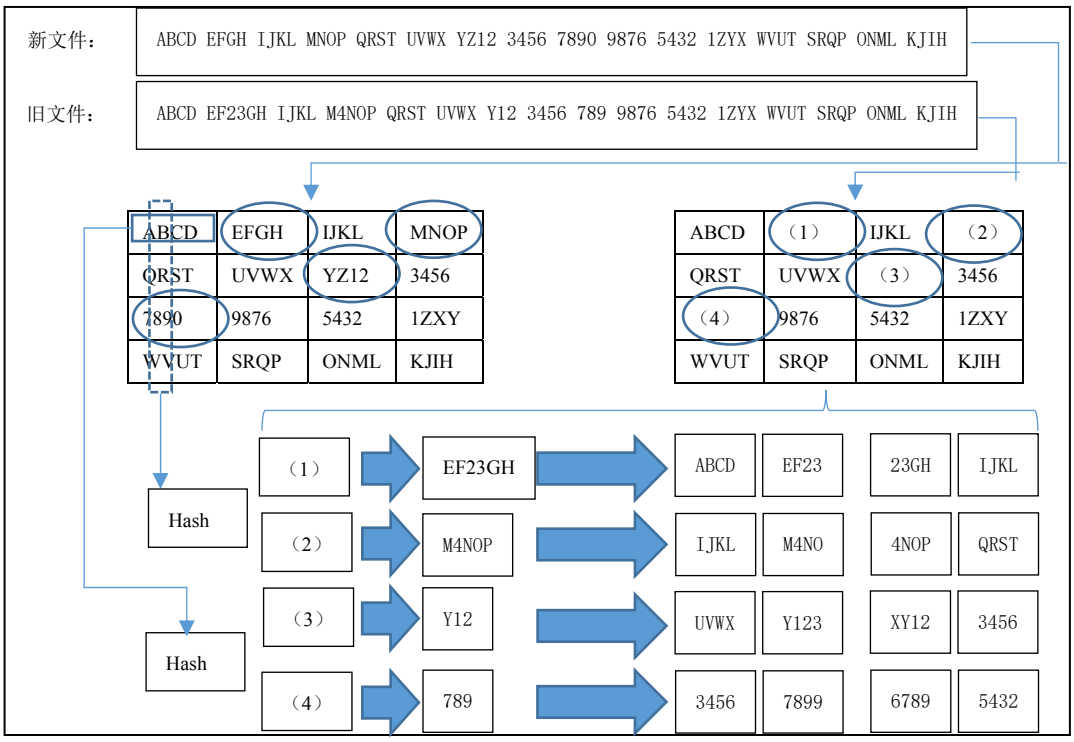

图 4 利用数据位图对数据进行分块示意图

为了减少同步次数并且最大限度的检测文件中的差异部分，本文在数据滑动分块方

案的前提下，将新文件数据经过重新组织成类似位图的数据位图的形式，利用同源数据文件大部分数据相同的关系，充分利用其它相同的数据块中的数据，确定差异块中的数据的准确位置，从而在一次同步的过程中，达到多次确认差异数据块的效果。

本数据分块方案原理如图 4 中所示，这是一个利用数据位图滑动分块方法对数据文件进行分块过程的示意图。图中第一步先将数据分割成 ABCD、EFGH、HIJK……，由于示意图中分块为四个单位，故形成数据位图的列数为四列，为了说明问题方便而设计成这个格式，并生成数据位图；第二步进行列的 hash 值的计算，例如将以列的方式读取的 AO7W、BR8V……等作为数据块，对其进行数据块指纹的计算，保存其指纹及在数据位图中的位置信息；第三步将数据分块的行、列 hash 值都传送给待更新的旧文件端；第四步在待更新的旧文件端先依据相同的 hash 值及文件位图框架信息进行填充，在经过 Rsync 算法检测后，不一致的数据块处暂时不填入信息；第五步对未填入信息部分分别进行两次计算填充，如图中（1）（2）（3）（4）处的位置，填充方式为先跟据被修改数据块之前的相同的数据块，顺序读取其后的数据填充空白处，然后按列计算以这种方式填充后的列数据块指纹，例如，第（1）处由于在 F 与 G 间插入了数据 23，因此，第一次填充在（1）处的数据为 EF23，此时通过列数据分块指纹对比，可以找到待更新的旧文件中 EF23GH 中的 EF 是与新文件中相同的数据，第二次则利用被修改的数据块后的紧邻数据块的前导数据进行填充，此时将 23GH 填充至（1），再次计算列数据指纹，此时可以确认 GH 为与新文件相同数据，经过这两步填充后，可以确认出数据相异部分为插入了数据 23，数据被删除部分操作方式与此理论相同，在此不再重复阐述。经过这个过程，可以确认出新文件数据与待更新的旧文件数据的准确差异。

3.2 算法实现

本文中的算法简称为 SBDB 算法，算法的执行分别在新文件端与待更新的旧文件端进行。算法中主要功能函数说明如下：

Computer_row_chunk(x,y)：文件 x 按固定块大小为 y 的分块方法分割，计算每个数据块的 hash 值；

Create_bitmap(z,y)：创建一个空白位图，z 为数据文件的大小，宽度为([([z/y]+1)/y])+1)×y，高度为 y,位图初始全部用 0 填充；

Fill_bitmap(x,y)：根据 y，填充数据位图，y 为一个顺序的位置信息表，由一个 0、1 组成的掩码组成，对应 1 的位图位置说明此处按顺序填充 x 中相应位置的数据块，对应位置为 0 的位置不填充，y 为空表时，顺序填充 x 中的数据至位图中；

Computer_col_chunk(x)：按 x 中以列方向读取信息，并计算指纹；

Rsync_hash(x,y)：利用类似 rsync 的滑动检测方法，检测出 x 数据分块指纹与 y 中的不同数据，返回一个位置信息掩码；

Compare_hash(x,y)：数据位图 x 中的数据按列方式分块，将这些分块的指纹信息与 y 中存储的新文件中的按列分块的数据块指纹信息进行对比。对比列数据块指纹信息函数中用到了一个填充技术，其描述为对于数据位图中对应的修改过的数据块的位置填充数据按下面的原则，在计算列数据指纹信息时根据最近块最可能相同的原则，将已经确定为不同的数据块的数据进行两次填充，就是先将被修改数据块前紧邻的未修改数据块文件的后续数据填充至修改文件处，第一次用这部分填充数据计算数据指纹，然后再将修改后的数据块的后一个数据块的前边的数据填充至修改处，同样按列进行计算数据指纹。

本算法核心处理过程中，新文件端的处理主要包括数据位图的形成、计算数据位图中对应的行列的数据指纹等操作，具体过程如表 1 描述。

表 1：SBDB 算法——新文件端部分

```
Algorithm SBDB（新文件端）:
Open source file =>file_tmp;
/* L 为新文件的长度,ldc 为数据块长度预设值*/
Computer length of source file=>L;
Read length of data chunk=>ldc;
*hash=NULL;
```

```
file_type=0;
/*hash 为指向指纹数组的指针，初始为空；file_type 指明文件操作是在新文件端还是待更新的旧文件端，0 为新文件端，1 为待更新的旧文件端*/
If ldc>0 and L>0
   hash_row[]=Computer_row_chunk(file_tmp，ldc);
   /*将新文件中的文件按顺序分块，分块长度 ldc,并计算每块的指纹存入 hash 值行信息,存入 hash_row 数组中，其中 hash_row 代表存储行指纹数组*/
   file_bitmap=Create_bitmap(L,ldc);
   /*创建数据位图，位图的宽度为(([L/ldc]+1)/ldc+1) ×ldc，高度为 ldc*/
else
   return 1;
/*此时,file_type=0,hash 指向空*/
if file_type
   hash=hash_row;
else
   hash_tmp=null;
   /*hash_tmp 代表新文件与待更新的旧文件指纹对比后不同一部分。由于当前在新文件端操作，hash_tmp 设置为 null，表明没有文件块需要标记*/
file_bitmap=Fill_bitmap(file_tmp,hash_tmp);
/*填充数据位图，第二个参数为 null 则把文件按顺序填充*/
hash_col[]=Computer_col_chunk(file_bitmap);
/*按列读取位图文件信息，并计算以列为数据分块的数据指纹信息，存入 hash_col 中，hash_col 代表存储以列的方向划分数据分块的指纹信息*/
Trans_hash(hash_row,hash_col);
/*传送行、列指纹信息到待更新的旧文件端*/
Trans_file_msg(L,ldc);
/*传送新文件的长度与分块大小到待更新的旧文件端*/
```

算法在待更新的旧文件端将完成数据指纹的校验过程，在待更新的旧文件端还必须依照新文件端的数据位图框架建立数据位图，从而为完成数据文件的重建过程作准备，其主要操作流程如表 2 所示。

表 2：SBDB 算法——待更新的旧文件端部分

```
Algorithm SBDB（待更新的旧文件端）:
file_type=1;
/*file_type 值说明以下操作步骤在待更新的旧文件端进行*/
Open object file=>file_tmp;
/*此时,file_type=1,hash 指向由新文件传送来的按行分块的数据块的指纹信息*/
if file_type
   hash=hash_row;
hash_tmp=Rsync_hash(file_tmp,hash);
/*利用类似 rsync 算法的流程，用 hash_row 存储的指纹对比分析待更新的旧文件中数据指纹与新文件中生成的 hash_row 中不同的分块，并统计，记录不相同的分块所处的位置，存储于 hash_tmp 中*/
file_bitmap=Create_bitmap(L,ldc);
/*创建待更新的旧文件的位图框架*/
file_bitmap=Fill_bitmap(file_tmp,hash_tmp);
/*填充数据位图，把待更新的旧文件中与新文件中对应位图文件相应位置有相同数指纹的数据块填充至少数据位图中，填充函数第二个参数为 hash_tmp 决定填充方式，相应位置没有相同的 hash_tmp 数据对应时则暂时标记*/
Mask=Compare_hash(file_bitmap,hash_col);
/*对比待更新的旧文件的数据位图与新文件的列数据分块的数据指纹，并标记，记录于 Mask 中*/
Trans_mark(Mask);
/*传送待更新的旧文件被标记的精确的数据差异信息到新文件端*/
```

算法在完成主要的处理流程后，还有部分后续的处理，即完成数据文件在待更新的旧文件端的生成过程，如表 3 所示。

表 3：SBDB 算法——文件合并部分

```
file_type=0;
/*以下操作步骤在新文件端进行*/
Data=Trans_data(file_tmp,Mask);
/*传回差异数据，存储于结构 Data 中，Data 结构包括数据部分与相应所在文件位置信息两部分*/
file_type=1;
/*以下操作步骤在待更新的旧文件端进行*/
Create_file(file_bitmap,Data);
/*根据 Data 中的数据在待更新的旧文件端创建数据文件*/
```

3.3 算法优化分析

假设本算法处理的新文件的长度为 L，数据分块预设值为 l_{dc}，新文件与待更新的旧文件中相同的数据为 b，本算法在基于 SB 数据分块方法的基础上加以改进，不妨设利用 SB 数据分块方法在新文件与旧文件中找到的相同数据块为 c_{sb} 块。在利用 SB 分块技术进行查找数据分块过程中，在新文件端进行的数据指纹哈希运算次数包括生成 32 位的

（adler-32）校验 $h_{sb\_32\_new}$ 及 128 位的（MD5）校验 $h_{sb\_128\_new}$。因此，在 SB 数据分块技术的情况下，而在待更新的旧文件端，查找与新文件端传输到本地的相同的数据指纹过程中，同样要计算旧文件对应的哈希值，不妨设，计算中 32 位的（adler-32）校验 $h_{sb\_32\_old}$ 及 128 位的（MD5）校验 $h_{sb\_128\_old}$。因此，在 SB 数据分块算法的情况下可以找到的新旧文件间可以找到的相同的数据量为

$$l_{dc} \times c_{sb} \quad (1)$$

在利用 SBDB 算法对新文件端处理后，增加了创建新数据位图的过程，同时在新文件端增加了计算以列为单位的哈希校验值的计算，此计算为 128 位的（MD5）校验值，按数据位图的创建方法，可以知道，增加的计算这个检验值的计算次数为 $([[L/l_{dc}]+1)/l_{dc}]+1) \times l_{dc}$，在待更新的旧文件端，增加的数据对比次数增量在最多情况下为

$$([L/l_{dc}]+1-c_{sb}) \times 2 \times l_{dc} \quad (2)$$

此时，所有的新旧文件的有差异的数据块均匀的分布于数据位图的不同数据列中；对比次数增量最少的情况为

$$([([L/l_{dc}]+1-c_{sb})/l_{dc}]) \times 2 \times l_{dc} \quad (3)$$

此时，所有的新旧文件有差异的数据块匀分布于数据位图中相同的列附近。当在待更新的旧文件端对比次数增量为(2)式所示时，如果新旧文件中的通过 SB 数据分块技术找到的相同数据块的数量满足

$$c_{sb} > ([L/l_{dc}]+1) - ([([L/l_{dc}]+1)/l_{dc}]+1) \quad (4)$$

则此时可以达到理论上的找到所有的新旧文件间的相同数据，并且此时用于对比哈希值的对比次数增量为(2)式所示；如果新旧文件中差异数据块都分布于一个列附近，此时找到的相同的数据达到理论上的最小值，此时用于对比哈希值的对比次数增量为(3)式所示，最小值取决所有相异数据块中具有最少的相似数据的数据块。

4 实例分析

本文在提出的 SBDB 算法的基本原型是基于滑动分块技术，主要用来定位内容相近文件的数据差异，从而确定内容相近文件中的相同部分。对于数据文件中的相同的数据块的查找准确度是最重要的指标。

本算法应用场景在文中以文件同步为基础，由 3.3 节的分析可知，其效果应该优于同样的应用场景下的 Rsync 算法，对于其它的数据分块算法的应用场景本文中并未涉及，因此，实验部分主要验证 SBDB 数据分块算法与 SB 数据分块算法在对数据文件分块过程中的差异，从而验证 3.3 节中理论所述的问题。设计实验从两个方面验证，一方面是构建一个特殊案例，证明 3.3 节所述的理论；另一方面用比较普通的文件进行对比分析，说明算法的现实可行性。

4.1 实验环境

算法运行环境为：操作系统 Windows 8，硬件参数为处理器 Intel(R) Core(TM) i5-3337U CPU @1.80GHz；内存（RAM）：8.00GB（7.84GB 可用）。算法采用 java 实现。

4.2 特殊案例实验及理论分析

本部分实验利用一个图像修改的例子，比较 SB 数据分块算法与 SBDB 数据分块算法在数据分块过程中产生的差异。

实验设计如下：在移动终端获取图像信息越来越容易的情况下，对图片信息进行简单的加工、处理后将处理过的图片发送到网络中存储是一个经常要遇到的问题。一般来说，由于手机等客户端的文件存储能力有限，经常用到云存储等存储方式扩展自己的存储能力。在这个过程中，选取两幅相似的位图图像文件进行网络同步。分别将这两个文件存储成不同的彩色位图，由于同一场景的位图有很大的相似性。故可用这个数据来对比算法执行的情况。

所选择的案例如图 5 所示的情况，在一张图片用 windows 画图工具编辑后，存储成 24 位位图形式。利用 SBDB 数据分块算法查找文件中相同的数据与 SB 数据分块算法查找文件中相同的数据的进行对比，数据分块大小为 700 字节，文件大小为 62070 字节。

图 5算法案例分析实例

实验中以第一幅图片为原始图片分别对第二、三幅位图做比较，表 4 为试验结果：

表 4 相似场景位图差异数据对比

		第二幅	第三幅
与第一幅实际相同字节数		61651	61717
查找相同字节数量	SB 算法	700	10500
	SBDB 算法	40028	12562
比较计算时间(ms)	SB 算法	14	11
	SBDB 算法	17	12

图 6 为当数据分块大小分别为 100，300，500，700byte 的情况下，分别对第一、二幅图和第一、三幅图利用 SBDB 数据分块算法与 SB 数据分块算法查重数据分析图，图中柱状图代表查找到的数据块相同数据量，折线图表明分别利用两种算法所消耗的时间。

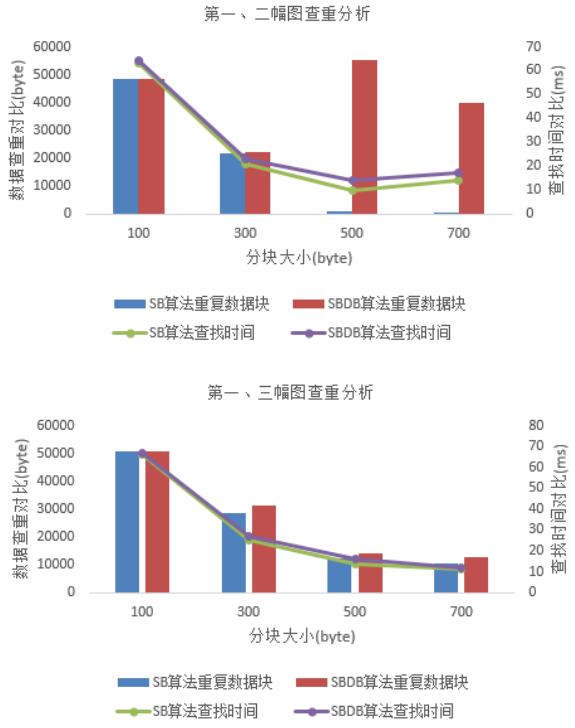

图 6 不同图片文件查重数据分析图

实验结果表明：在数据分块大小为 700byte 的情况下(表 4 中所示数据)，第一幅图片与第二、三幅图片的对比中，SB 数据分块查找相同数据的准确度均落后于 SBDB 数据分块算法。在查找过程中，对比次数相对来说只增加了相当少的计算时间，分别 SBDB 算法仅用了 3ms 和 1ms，时间增量为 21%与 9%，但是在这种情况下，查找相同的数据块有大幅度提升，分别查找的相同的数据块块数为 SB 算法的 57 倍和 1.19 倍。

第二幅图片与第三幅图片虽然都是对第一幅图片有较小的改动，但是第二幅图片的改动集中于一个像素附近，恰好造成了改变的数据均匀分布于新文件的数据分块中，第三幅图片的改动较分散，但由于没有贯穿整个图，在数据区则集中分布于某些数据块中。

第一幅图片与第二幅图片对比时，由于新文件端是固定方式分块方式，SB 数据分块方案能找到的相同数据就非常有限，但是经过 SBDB 数据分块方案处理，此时可以大幅度提升相同数据的查找效果，在进行数据位图以列方式对比时，由于查找的相似的数据块比较多，所以 SBDB 数据分块方案在此时查找的时间也有明显的增加。第一幅图片与第三幅图片对比时，由于数据修改集中于部分数据块中，所以，SB 数据分块方案与 SBDB 数据分块方案找到的相同数据差距不是很明显，由于查找的数据分块方案中，在进行数据位图以列方式对比时，由于查找的相似的数据块相对较少，所以 SBDB 数据分块方案在此时查找的时间也有少量的增加。

以上实例说明，在 3.3 节中分析的，随着数据相似块的分布情况不同，SBDB 数据分块方案与 SB 数据分块方案的查找相同数据能力的提升也是不同的，相异数据越均匀分布于新文件的固定分块中，SBDB 的效果越好，但是查找时间随之增加。

数据分块大小对 SB 数据分块算法与 SBDB 数据分块算法的影响如图 6 中数据所示，随着数据分块的增加，不管是 SB 数据分块算法还是 SBDB 数据分块算法，查找时间都会相对下降，由于查找对比的数据随着数据块的增加，次数会降低。第二幅图片选取的是一个极端的例子，相异数据均匀分布于固定的数据分块中，所以，在数据分块足够小时，SB 数据分块算法与 SBDB 数据分块算法才有较少的差距，数据块增大后，SB 数据分块方案当达到某个临界值后几乎找不到

相同数据，但此时 SBDB 数据分块方案仍然有较好的查找性能。第三幅图片选取的为一个比较数据差异集中在某些固定数据块的情况，在此情况下，SBDB 数据分块算法较 SB 数据分块算法在查找性能提升上较为稳定，数据分块不同的情况下，也表现出稳定的性能提升。

4.3 普通文件实验结果分析

本部分实验用一些普通文件作为实验对象，证实在实际应用中 SBDB 数据分块算法较 SB 数据分块算法的查找相同数据的性能提升以及查找时间的对比。

实验对象选取为二进制文件和源代码文本文件两种情况，分别为开源编辑器 emacs 的二进制安装文件包和 C 语言开源编译软件 gcc 的源代码包。分别对比几种版本在两种算法下查找到相同数据的比例及查找应用的时间，从而证实 4.2 节中分析结果。

具体软件为 emacs-23.1-barebin-i386.zip 与其 23.2、23.3 版本的对比情况及 gcc-3.0.2.tar.gz 与其 3.0.3、3.0.4 版本的对比情况。

图 7 为中四条曲线分别展示利用 SBDB 数据分块算法比 SB 数据分块算法多找到的相同数据与 SB 数据分块算法找到的相同数据的比例关系。

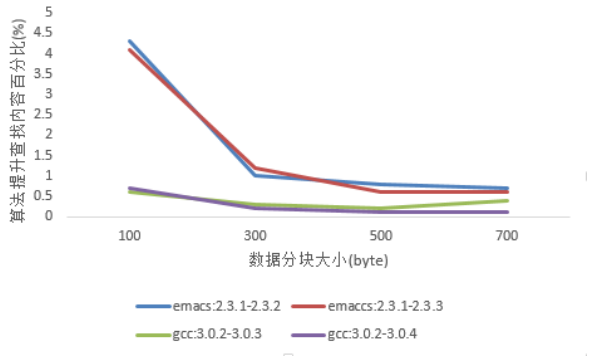

图 7 SBDB 算法比 SB 算法提升查找内容示意图

由图 7 可以看出，利用 SBDB 数据分块算法在对普通文件作处理的过程中，查找到的相同数据较原 SB 数据分块算法均有提升。随着数据分块的减小，SBDB 算法提升的效果比较明显。

图 8 为中四条曲线分别展示利用 SBDB 数据分块算法比 SB 数据分块算法运行时多消耗的查找时间与程序仅运行 SB 数据分块算法的比例关系。

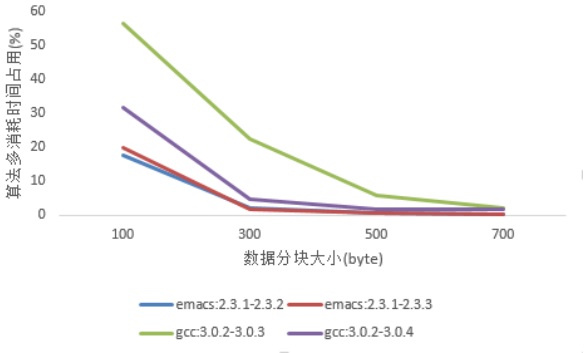

图 8 SBDB 算法比 SB 算法多消耗时间占用图

由图 8 中可以看出，随着数据分块大小的增大，由于 SBDB 算法能够找到的相同数据与 SB 能够找到的相同数据接近的情况下，算法多消耗的时间逐渐减小，而在数据分块较小的情况下，SBDB 算法的消耗时间也占了相当一部分系统运行时间。

实现结果表明：在普通的文件通过数据分块查找文件中的相同数据的过程中，SBDB 数据分块算法与 SB 数据分块算法相比较，随着数据分块的减小，提升查找相同数据的能力逐渐增强，但是查询时间随之提升。

5 总结

本文在滑动分块对数据进行分块的基础上，重新组织了数据的排列形式形成一个数据位图，利用重新组建的数据位图文件，采用基于数据位图的数据分块方法，有效的利用了相似文件的相同的数据的同源效应，利用已经得到的相同数据的数据信息，最大限度的检测出数据的不同部分。在一些极端情况下，对于普通的滑动分块技术已经无法检测出相同的数据块的数据文件，依然有比较好的检测重复数据的效果。本方法可以用于网络文件的同步与数据存储等领域，高效的检测同源数据文件的相似部分，从而降低网络带宽及存储空间占用。

参 考 文 献

[1] Gupta D, Sagar K. Remote file synchronization single-round algorithms[J]. International Journal of Computer Applications, 2010, 4(1): 32-36.

[2] AO Li, SHU Ji-Wu, LI Ming-Qiang. Data Deduplication Techniques[J]. Journal of Software, 2010, 21(5): 916-929.

敖莉, 舒继武, 李明强. 重复数据删除技术[J]. 软件学报, 2010, 21(5): 916-929.

[3] Fu Yinjin，Xiao Nong，Liu Fang. Research and Development on Key Techniques of Data Deduplicati0n[J]. Journal of Computer Research and Development, 2012, 49(1): 12-20.

付印金, 肖侬, 刘芳. 重复数据删除关键技术研究进展[J]. 计算机研究与发展, 2012, 49(1): 12-20.

[4] Bobbarjung D R, Jagannathan S, Dubnicki C. Improving duplicate elimination in storage systems[J]. ACM Transactions on Storage (TOS), 2006, 2(4): 424-448.

[5] Rabin MO. Fingerprinting by random polynomials[R]. CRCT TR-15-81,Boston:Harvard University, 1981.

[6] Tridgell A. Efficient algorithms for sorting and synchronization[M]. Canberra: Australian National University, 1999.

[7] XU Dan, SHENG Yonghong, JU Dapeng, et al. High Effective Two-round Remote File Fast Synchronization Algorithm[J]. Journal of Frontiers of Computer Science and Technology, 2011, (1):38-49.

徐旦, 生拥宏, 鞠大鹏等. 高效的两轮远程文件快速同步算法[J]. 计算机科学与探索, 2011, (1):38-49.

[8] Suel T, Noel P, Trendafilov D. Improved file synchronization techniques for maintaining large replicated collections over slow networks[C].//Proceedings of ICDE 2004, March 2004.

[9] Jain N, Dahlin M, Tewari R.Taper: Tiered approach for eliminating redundancy in replica synchronization[C].//The USENIX Conference on File and Storage, 2005:281−294.

[10] Irmak U, Mihaylov S, Suel T. Improved single-round protocols for remote file synchronization[C]. //Proceedings of IEEE INFOCOM, March 2005.

[11] Cormode G, Paterson M, Sahinalp S C, et al. Communication complexity of document exchange[C]. //Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, 2000: 197-206.

[12] Wang G P, Chen S, Lin M W, et al. SBBS: A sliding blocking algorithm with backtracking sub-blocks for duplicate data detection[J]. Expert Systems with Applications, 2014, 41(5): 2415-2423.

邓雪峰 男，1975 年生，博士研究生，主要研究方向移动互联网、物联网.

孙瑞志 男，1964 年生，教授，主要研究方向农业信息化技术、计算机支持的协同工作的研究.

张永瀚 男，1991 年生，硕士研究生，主要研究方向智能信息处理.

聂 娟 女，1977 年生，博士研究生，主要研究方向农业信息化.