

(12) 发明专利

(10) 授权公告号 CN 101379465 B

(45) 授权公告日 2013.03.27

(21) 申请号 200780004129.8

代理人 刘杰 王小衡

(22) 申请日 2007.11.21

(51) Int. Cl.

(30) 优先权数据

G06F 7/24 (2006.01)

11/566,122 2006.12.01 US

(85) PCT申请进入国家阶段日

(56) 对比文件

2008.07.31

US 5990810 A, 1999.11.23,

(86) PCT申请的申请数据

US 20020169934 A1, 2002.11.14,

PCT/US2007/085357 2007.11.21

US 20040225655 A1, 2004.11.11,

(87) PCT申请的公布数据

CN 1790327 A, 2006.06.21,

W02008/067226 EN 2008.06.05

US 6828925 B2, 2004.12.07,

(73) 专利权人 美国日本电气实验室公司

审查员 李昕宇

地址 美国新泽西州

(72) 发明人 C·杜布尼基 K·利乔塔

权利要求书 3 页 说明书 19 页 附图 9 页

E·克鲁斯 C·昂古里努

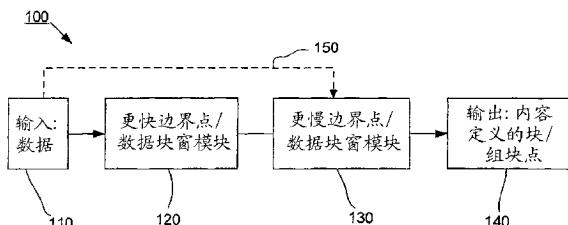
(74) 专利代理机构 中国专利代理(香港)有限公司 72001

(54) 发明名称

利用多种选择标准进行数据管理的方法和系统

(57) 摘要

本发明提供了用于进行数据管理和数据处理的系统和方法。各实施例可以包括涉及到以相当高质量的结果进行快速数据选择的系统和方法，并且可以包括较快数据选择函数和较慢数据选择函数。各实施例可以包括涉及到针对数据集或数据串的数据散列和 / 或数据冗余识别和消除的系统和方法。各实施例可以包括第一选择函数和第二选择函数，所述第一选择函数被用来从数据集或数据流预先选择边界点或数据块 / 窗，所述第二选择函数被用来细化所述边界点或数据块 / 窗。所述第二选择函数在确定所述边界点或数据块 / 窗在所述数据集或数据流中的最佳位置方面可以具有更好的性能。在各实施例中，可以通过第一较快散列函数和较慢但是辨识力更强的第二散列函数来处理数据。



1. 一种计算机实现的数据管理方法,包括以下步骤:

在处理单元处,利用第一内容定义的选择函数预先选择数据流中的多个数据窗当中的部分,所述第一内容定义的选择函数包括矩形波串和函数、乘法线性同余发生器 MLCG 函数或者 rolN-xor 函数;以及

在处理单元处,利用第二内容定义的选择函数选择所述多个数据窗当中的预先选择的所述部分的子集以最大化给定特性,

其中所述第一内容定义的选择函数在选择数据块边界方面比所述第二内容定义的选择函数更快。

2. 权利要求 1 的方法,其中,所述第一内容定义的选择函数在窗选择方面比所述第二内容定义的选择函数更快。

3. 权利要求 1 的方法,其中,所述第一内容定义的选择函数是矩形波串和函数并且所述矩形波串和函数与关于所述矩形波串和的值的选择标准相耦合。

4. 权利要求 1 的方法,其中,所述第二内容定义的选择函数包括 Rabin 指纹、SHA-1 函数或者 CRC32c 函数。

5. 权利要求 1 的方法,其中,所述第一内容定义的选择函数是滚动窗函数。

6. 权利要求 1 的方法,其中,所述多个窗被利用来定义供散列的数据组,并且确定所述数据流中的断点的速度得到提高。

7. 权利要求 1 的方法,还包括以下步骤:

在处理单元处,生成对应于由所述子集的一个或多个组块点确定的数据组块的数据组块尺寸的值,其中所生成的该值表示包含在所述数据组块中的底层数据。

8. 权利要求 7 的方法,还包括以下步骤:

在处理单元处,把所生成的该值与一个或多个先前生成的值进行比较,以便确定所生成的该值是否等于所述一个或多个先前生成的值。

9. 权利要求 8 的方法,还包括以下步骤:

在处理单元处,确定存在数据复制;以及

在处理单元处,停止对被确定为具有复制数据的数据组块中的数据的进一步处理。

10. 权利要求 8 的方法,还包括以下步骤:

如果没有数据复制,则由处理单元存储所生成的该值。

11. 权利要求 1 的方法,还包括以下步骤:在处理单元处,把包含在由所述第二内容定义的选择函数所选择的新数据窗或新数据组块中的底层数据与来自一个或多个先前定义的数据窗或数据组块的先前遇到的底层数据进行比较。

12. 权利要求 11 的方法,还包括以下步骤:

如果包含在所述新数据窗中的底层数据不等于所述先前遇到的底层数据,则由处理单元输出包含在所述新数据窗中的底层数据;以及

如果包含在所述新数据组块中的底层数据等于所述先前遇到的底层数据,则不输出包含在所述新数据组块中的底层数据。

13. 权利要求 1 的方法,还包括以下步骤:在处理单元处,生成对应于数据窗或数据组块的值,该值表示包含在该数据窗或数据组块中的底层数据。

14. 权利要求 13 的方法,还包括以下步骤:在处理单元处,把所生成的该值与一个或多

个先前生成或遇到的值进行比较。

15. 权利要求 14 的方法,还包括以下步骤:

如果所生成的该值不等于所述一个或多个先前生成或遇到的值,则由处理单元输出所生成的该值;以及

如果所生成的该值等于所述一个或多个先前生成或遇到的值,则不输出所生成的该值。

16. 一种确定数据流中的边界点以便进行数据管理的计算机实现的方法,包括以下步骤:

在处理单元处,利用第一内容定义的选择函数预先选择数据流中的多个边界点当中的一部分,所述第一内容定义的选择函数包括矩形波串和函数、乘法线性同余发生器 MLCG 函数或者 ro1N-xor 函数;

在处理单元处,利用第二内容定义的选择函数选择所述多个边界点当中的预先选择的所述部分的子集以最大化给定特性;以及

在处理单元处,生成对应于由所述多个边界点的所述部分的所述子集确定的数据组块的值,

其中所述第一内容定义的选择函数在选择边界点方面比所述第二内容定义的选择函数更快。

17. 权利要求 16 的方法,还包括以下步骤:在处理单元处,把所生成的该值与一个或多个所存储的值进行比较,以便检测复制品或者存储所生成的该值。

18. 权利要求 17 的方法,其中,通过散列来生成值。

19. 权利要求 18 的方法,还包括以下步骤:在处理单元处,消除通过对包含在由一个或多个所述边界点定义的数据组块中的数据进行散列而找到的复制数据。

20. 一种数据处理系统,包括:

处理单元,

第一内容定义的选择函数模块,其被配置成预先选择数据断点或数据窗的第一集合,所述第一内容定义的选择函数模块包括矩形波串和函数、乘法线性同余发生器 MLCG 函数或者 ro1N-xor 函数;以及

第二内容定义的选择函数模块,其被配置成选择所述预先选择的数据断点或数据窗的子集以最大化给定特性,其中所述第一内容定义的选择函数模块在处理所述数据断点或数据窗方面比所述第二内容定义的选择函数模块更快。

21. 权利要求 20 的数据处理系统,其中,所述第二内容定义的选择函数模块包括 Rabin 函数。

22. 权利要求 21 的数据处理系统,其中,所述数据处理系统是散列系统并且输出内容定义的一个或多个数据块或组块点。

23. 一种数据处理系统,包括:

用于使用内容定义的预先选择函数来预先选择数据断点或数据窗的第一集合的装置,所述内容定义的预先选择函数包括矩形波串和函数、乘法线性同余发生器 MLCG 函数或者 ro1N-xor 函数;以及

用于使用内容定义的选择函数选择并输出所述预先选择的数据断点或数据窗的子集

以最大化给定特性的装置,其中所述用于预先选择的装置在处理所述数据断点或数据窗方面比所述用于选择的装置更快。

24. 权利要求 23 的数据处理系统,还包括用于生成对应于所述子集的一个或多个数据断点或数据窗当中的每一个的值的装置。

25. 权利要求 24 的数据处理系统,其中,所述用于预先选择数据断点或数据窗的第一集合的装置执行滚动矩形波串和函数,所述用于选择并输出所述预先选择的数据断点或数据窗的子集的装置执行 Rabin 函数,并且所述用于生成值的装置执行 SHA-1 函数。

26. 权利要求 25 的数据处理系统,其中,所生成的该值是散列值。

27. 权利要求 23 的数据处理系统,其中,所述数据处理系统是散列系统并且输出内容定义的一个或多个数据块或组块点。

利用多种选择标准进行数据管理的方法和系统

[0001] 本专利申请涉及到同样于 2006 年 12 月 1 日提交的授予 Cezary Dubnicki、Erik Kruus 和 Cristian Ungureanu 的标题为“METHODS AND SYSTEMS FOR QUICK AND EFFICIENT DATA MANAGEMENT AND/OR PROCESSING(用于进行快速且高效的数据管理和 / 或处理的方法和系统)”的美国专利申请 No. (TBD) , 其被合并在此以作参考。

[0002] 本公开内容可能包含受到版权保护的信息, 比如这里给出的各种示例性 C++ 代码和伪代码。版权所有者不反对由任何人对本专利内容或者出现在美国专利商标局文件或记录中的本专利进行复制再现, 但是在其他方面保留全部版权权利。

技术领域

[0003] 本发明涉及数据处理和数据管理的领域, 更具体来说, 本发明涉及针对诸如数据散列和 / 或数据冗余消除之类的应用的快速数据处理的方法和系统。

背景技术

[0004] 每天都有越来越多的信息在全世界范围内被产生, 并且所保留及传送的信息量持续以惊人的速度增加, 从而在数据处理和管理方面产生了严重的问题。许多所述信息被电子地产生、处理、保持、传送以及存储。仅仅尝试管理所有这种数据以及相关的数据流和存储的数量就令人吃惊。因此已经开发出多种系统和方法来更加快速地处理数据, 并且通过消除尽可能多的复制数据来存储及传送更少的数据。例如, 已经开发出多种系统和方法以帮助减少针对存储、传送来自各种电子设备的复制数据的需求, 所述电子设备例如是计算机、计算机网络 (比如内联网和因特网) 、诸如电话和 PDA 的移动设备、硬件存储设备等等。此外, 特别在例如数据传输期间需要利用密码术来加密数据。例如, 已经开发出提供强大的 (即密码的) 散列的系统和方法, 并且可以很自然地把这种方法合并在利用数据散列在不安全的通信信道上实现数据冗余消除的应用中。

[0005] 在各种电子数据管理方法和系统中, 已经开发了多种方法以用来散列数据和 / 或例如从数据存储和数据传输中消除冗余数据。这些技术包括各种数据压缩、数据散列和密码方法。在许多文章中公开了一些示例性技术, 其中包括 :Philip Koopman 的“32-Bit Cyclic Redundancy Codes for Internet Applications(用于因特网应用的 32 比特循环冗余码)”(Proceedings of the 2002 Conference on Dependable Systems and Networks, 2002) ;Jonathan Stone 和 Michael Greenwald 的“Performance of Checksums and CRCs over Real Data(校验和及 CRC 在真实数据上的性能)”(IEEE/ACM Transactions on Networking, 1998) ;Val Henson 和 Richard Henderson 的“An Analysis of Compare-by-Hash(散列比较的分析)”(Proceedings of the Ninth Workshop on Hot Topics in Operating Systems, Lihue, Hawaii, 2003 年 5 月, pp. 13-18) ; 以及 Rai Jain 的“AComparison of Hashing Schemes for Address Lookup in Computer Networks(用于计算机网络中的地址查找的散列方案的比较)”(IEEE Transactions on Communications, 1992) 。此外还在多篇美国专利和专利公开中公开了各种示例性技术, 其中包括美国专利

公开 No. 2005/0131939、2006/0047855 和 2006/0112148 以及美国专利 No. 7,103,602 和 6,810,398。

[0006] 然而,所述已知的技术缺少特定有用的能力。一般来说,性能更好的选择技术(例如高数据冗余消除)使用过多的处理时间(花费过长时间),非常快速的数据选择技术可能缺少所期望的数据消除程度。例如,存在多种散列函数方法,其中包括整文件散列、固定尺寸数据块散列以及内容定义的数据组块散列。但是在这些技术当中没有一种技术既相当快速(仅仅使用少量计算时间)又具有识别出数据集中的大部分数据冗余的能力(例如具有高数据冗余消除)。

[0007] 因此,需要一种既具有合理的性能又快速的数据选择技术。例如,需要一种能够快速地执行数据散列和 / 或数据冗余识别及消除并且同时仍然能够识别出数据集中的大部分冗余数据的散列和 / 或数据冗余识别及消除系统和方法。还需要一种更加快速地确定适当的断点和边界以便在内容定义的散列函数中确定数据块或组块的系统和方法。

发明内容

[0008] 本发明总体上针对提供用于数据管理和数据处理的系统和方法。例如,本发明的实施例可以包括用于以性能更好的选择结果(例如高数据冗余消除)进行快速数据选择的系统和方法,并且还可以包括更快的数据选择处理和更慢的数据选择处理。此外,提供了用于对数据集或数据串进行数据散列和 / 或数据冗余识别及消除的示例性系统和方法。本发明可以更为鲁棒并且可以提供在一定程度上把快速散列的速度与确定更适当的断点或边界集合的散列的更加鲁棒的性能特性相组合的系统和方法。本发明可以是计算机实现的发明,其包括用于在不显著降低数据处理结果的质量的情况下提高数据处理速度的软件和硬件。

[0009] 在本发明的各实施例中,例如提供了可以包括第一选择函数并且可以包括第二选择函数的系统和方法,所述第一选择函数被用来从数据集或数据流中预先选择边界点或者数据块 / 窗,所述第二选择函数可以细化来自数据集或数据流的边界点或数据块 / 窗。第一选择函数的计算可以比第二选择函数更快(例如花费更少的处理时间来滚动,所谓滚动即产生对应于来自所述数据集或数据流的后续数据块 / 窗的散列值。)。第二选择函数在确定各边界点或数据块 / 窗在所述数据集或数据流中的适当位置方面可能具有更好的性能。第二选择函数可以是(但不必是)可滚动的。一个示例性第一选择函数可以是使用例如移动窗的基于矩形波串和 (boxcar sum) 的函数。这是一种特别快速的数据选择处理。一个示例性第二选择函数可以是 Rabin 指纹函数。所述第一选择函数特别快速,并且所述第二选择函数具有特别好的比特随机化特性。在各实施例中,第一选择函数和第二选择函数可以被用来生成多个内容定义的边界点、块断点或组块点,以便确定应当在何处把数据集或数据流分离成各数据块、组块或窗。在各实施例中,所述内容定义的边界可以被用来确定数据块、组块或窗尺寸,可以对所述数据块、组块或窗尺寸应用散列函数,并且可以产生所得到的散列值。在各实施例中,可以把所得到的散列值与一个或多个所存储的散列值进行比较,以便确定所述数据块、组块或窗是否是完全复制的数据,或者确定是否应当把该新的散列值存储为独特的数据块、组块或窗。

[0010] 在各实施例中,可以提供一种用于数据散列的内容定义的技术。所述内容定义的

技术可以包括处理系统，该处理系统具有较快散列函数模块（例如由于仅有较少的计算因此花费较少处理时间）和较慢散列函数模块（例如由于有较多的计算因此花费较多处理时间）。所述较快散列函数模块可以接收例如来自数据流的数据，并且可以预先选择可在该处将所述数据流断开成不同的数据块或组块的一个或多个数据边界、断点、块点或组块点。所述较快散列函数在确定最佳的数据边界、断点、块点或组块点时的性能可能略微较低。所述较快散列函数例如可以是矩形波串和函数，其例如使用滚动或移动窗。所述较慢散列函数模块可以仅仅对于已经被所述较快散列函数预先选择的那些数据块或组块执行散列。所述较慢散列函数模块在确定最佳数据边界、断点、块点或分块点以便达到特定目的（例如识别出更多边界）方面的性能可能更好。所述较慢散列函数例如可以是 Rabin 指纹函数、SHA-1 散列、CRC32c 散列等等，并且可以是滚动的或非滚动的。

[0011] 通过研究下面的公开内容及其附图，本领域技术人员将认识到各实施例所包括的其他方面。

附图说明

[0012] 通过考虑下面结合附图对本发明的实施例所做的详细描述，本发明的用途、目的、特征和优点将变得显而易见，在附图中用相同的附图标记表示完全相同的元件，其中：

[0013] 图 1 是根据至少一个实施例的利用多模式数据边界点确定的示例性数据管理或处理系统；

[0014] 图 2 是根据至少一个实施例的利用多模式数据边界点确定的示例性数据管理或处理方法；

[0015] 图 3 是根据至少一个实施例的关于如何利用（多种）多模式数据边界点确定系统和方法进行数据管理或处理的示例性图示；

[0016] 图 4 是根据至少一个实施例的利用多模式数据边界点确定来识别先前在数据集或数据流中遇到的数据的详细示例性数据管理或处理方法；

[0017] 图 5 是根据至少一个实施例的利用内容定义的数据块或组块的示例性散列技术；

[0018] 图 6 是根据至少一个实施例的利用矩形波串和技术的示例性第一选择函数；

[0019] 图 7a 和 7b 提供了根据至少一个实施例的用于利用内容定义的数据块或组块的示例性散列技术的数据窗、块或组块的示例性图示；

[0020] 图 8 是根据至少一个实施例的计算机设备的示例性功能方框图；以及

[0021] 图 9 是示出了根据至少一个实施例的网络的示例性功能方框图。

具体实施方式

[0022] 本发明总体上针对用于进行数据管理和数据处理的系统和方法。各实施例可以包括涉及到多模式数据选择技术的系统和方法。更具体来说，各实施例可以包括涉及以相当高质量的结果进行快速数据选择的系统和方法，并且可以包括较快数据选择处理和较慢数据选择处理。在本发明的各实施例中，例如提供了可以包括第一选择函数并且可以包括第二选择函数的系统和方法，所述第一选择函数被用来从数据集或数据流中预先选择边界点或者数据块 / 窗，所述第二选择函数可以细化来自数据集或数据流的边界点或数据块 / 窗。第一选择函数的计算可以比第二选择函数更快（例如花费更少的处理时间）。第二选择函

数在确定各边界点或数据块 / 窗在所述数据集或数据流中的最佳位置方面可能具有更好的性能。一个示例性第一选择函数可以是例如使用移动窗的基于矩形波串和的函数。这是一种特别快速的数据选择处理。一个示例性第二选择函数可以是 Rabin 指纹函数，其可以是滚动的或非滚动的。具有所述较快 / 较慢特性的其他选择函数同样适用。

[0023] 本发明可以是计算机实现的发明，其包括用于在不显著降低所述数据处理结果的质量的情况下提高数据处理速度的软件和硬件。在至少一个实施例中，可以利用计算设备来实现这里提供的（多个）系统和方法，并且可以在网络内的一个或多个计算机上操作所述系统和方法。在图 8 和图 9 中更加详细地描述了（多个）示例性计算设备和网络的细节。先前提到的那些例子可能有助于更好地理解本发明的各种细节。

[0024] 在任何情况下，为了易于理解，将在用于散列函数和 / 或数据冗余识别和 / 或数据复制消除的情况下解释本发明。但是本领域技术人员将认识到，本发明可以应用于其他数据管理和处理系统和方法，其中包括将要处理或存储数据串的计算机、将要传送数据的无线通信、因特网和内联网应用、数据加密技术等等。特别地，在这里被用来解释本发明的各示例性实施例主要涉及数据散列和数据复制消除。

[0025] 在数据散列和 / 或数据冗余识别和 / 或消除实施例中，本发明可以包括较快散列函数和较慢散列函数。所述较快散列函数可以用于预先选择组块、块或散列点。所述较慢散列函数可以被应用来确定哪些预先选择的组块、块或散列点更适于识别最多数据复制。对应于数据集或数据串的散列函数可以被用来识别大量数据中的完全相同的数据部分。存在三种用于散列的一般技术：整个文件内容散列、固定尺寸块散列以及内容定义的散列。整个文件内容散列适于针对整个数据文件（例如完整的文本或图像文件）产生散列值或校验和。固定尺寸数据块散列可以适于针对不同数据文件的预先确定的固定尺寸部分（例如 1000 个比特）产生散列值或校验和。内容定义的数据块散列可以适于根据其尺寸由所选数据内容标准决定的可变尺寸数据块（例如在所述数据中找到 X 个连续 1 或 0 之后把所述数据分解成数据组块）产生散列值或校验和。

[0026] 在下面几段中，所述 SHA-1 的速度对于所有三种情况可以是几乎相同的；即对于整文件、固定尺寸和可变尺寸分块可以是几乎相同的。在每一种情况下，每一个输入字节都处在唯一组块中，并且必须针对每一个这种组块评估 SHA-1。但是在整文件、固定尺寸、可变尺寸分块之间可能存在的其中一个主要差异是这种 SHA-1 计算的数目。但是无论如何，对于所述 SHA-1 评估来说，总处理时间（例如 CPU 时间）将几乎是相同的，这是因为所述处理时间的大部分将在于所述 SHA-1 计算本身，而不是与存储一个或多个 SHA-1 散列值相关的开销。

[0027] 在整文件散列的情况下，可以通过对于整个文件的所有数据应用某一散列函数来执行散列。例如，可以使用 SHA-1 散列函数并且将其应用于整个数据文件。所述 SHA-1 散列函数的计算复杂度很高，并且相对于某些其他散列函数可能比较慢。无论如何，在本例中，出于识别并且消除复制的目的，找到并且消除最少量数据复制量，这是因为当单一数据比特在文件中发生改变时，所得到的散列值将不同于先前保存的散列值，并且必须传送或保存与该经过修订的文件相关联的完全数据量（例如，当文本文件中的一个字母发生改变时，该文本文件的整个数据表示及其散列值将会改变，从而其将不是该相同文本文件的先前版本的复制品）。另一方面，所述散列较快，这是因为对于整个数据文件仅仅需要操作一次所

述散列函数。

[0028] 如上所述,固定尺寸数据块散列函数对在整个文件中找到的完整数据的各部分或各块执行散列(例如单一文本文件可以被分离成多达10个相同尺寸的10K比特数据块),并且可以在非重叠固定尺寸下设置各数据块。同样地,可以把SHA-1散列函数应用于构成整个文件(例如100K比特)的一组固定尺寸的块当中的每一个(例如10K比特)。在这种情况下可以找到更多复制,这是因为每一次散列的数据块较小,并且在整个文件内的某处的单一比特改变将仅仅导致构成整个文件的多个块的其中之一的改变(例如,10个10K比特块当中的9个将是复制品)。然而,单一字节插入可能会妨碍对大量块的复制品检测(例如,在包含第一次插入的块之后的各块可能成为非复制品)。所述块越小,冗余检测就越好,但是这样做将会得到略微更慢的处理,这是因为所述散列函数(例如SHA-1)对于在所述整个数据文件中找到的相同数据量必须运行更多次。

[0029] 最后,利用所述内容定义的数据组块散列,可以通过应用较慢并且具有略微更好的性能(例如更加精确并且辨识力更强的计算)的散列来执行散列,以便识别出由其内容定义的各数据组块并且生成与之对应的值。在这种情况下,局部化的插入修改和局部化的替换修改都可能会导致对单一内容定义的组块的改变,从而与固定尺寸分块相比会增大所找到的复制量。这种散列函数可以包括Rabin指纹识别与SHA-1散列函数的组合。可以对于所述数据文件中的数据的重叠数据窗(例如滑动窗)应用多次所述Rabin指纹识别,以便基于预定的边界点标准(例如所述函数的值等于以X个0或1结尾的预定数目的比特或字节)确定应当把所述组块边界设置在所述数据文件中的什么位置处,随后可以对每一个所确定的数据块(其尺寸基于所分析的底层数据而改变)应用所述SHA-1散列函数。同样地,如前所述,每一个字节的输入都可以进入某一SHA-1散列计算。但是与固定尺寸分块相比,所述Rabin指纹识别导致附加的计算负担(例如处理时间)。虽然上述方法可以非常好地识别出多得多的数据边界,但是全部所述两种函数都可能比较耗时,二者组合在一起将使得散列和/或数据冗余识别和消除非常耗时。实际上,在尝试优化冗余数据识别和/或数据消除时,所述Rabin指纹识别函数对于识别各数据块切口或散列点应当具体位于何处来说可能特别耗时。

[0030] 为了进行分块,达到所期望的数据消除程度的一种措施是采用选择函数,该选择函数通过内容定义的确定性程序从真实世界数据产生各组块,其中所述程序获得与在给定随机数据输入的情况下所预期的分布紧密匹配的分块尺寸分布。实现上述内容的一种方式是采用某一散列函数,其后是具有已知出现概率的选择标准。一般来说,选择具有对任何数据窗内的比特“加扰”的卓越能力的单一散列函数。按照这种方式,输出值展现出与所述输入窗内的实际数据比特的极低相关性。对于这种比特随机化散列值H,已经被使用的简单的选择标准例如包括确保特定数目的比特等于预定值或者要求 $H^{\wedge}(H-1)$ 的值>阈值、H>阈值或者H与掩码==某值。把比特随机化散列与选择程序标准相耦合可以产生选择函数,其选择输入窗的“看起来随机的”子集。通常希望使得这种选择函数有很高的几率在给定真实生活(例如非随机)数据输入时产生类随机组块尺寸分布。

[0031] 在面对真实生活数据时预期会具有类随机组块尺寸分布的各种选择标准在这里可以被称作具有良好的“比特加扰”能力。本发明的一个性能目标可以包括两个方面:很容易理解,一方面是操作速度;另一方面多少有些抽象,即上面提到的比特加扰行为。本发明

可以提供一种使得数据处理应用大大提高其操作速度同时在可能损失多少比特加扰能力方面仍然保留一定程度的控制的机制。能够很好地满足所述抽象的比特加扰目标的选择函数可能导致得到更能够满足特定于其领域的非抽象目标的应用。例如，在多种应用中，所述目标可能涉及到复制品检测，其中包括：更好的复制品检测、更好的潜在复制品检测、更好的相似性检测、传送更少数据、增大带宽、降低存储需求等等。通过在这种应用中使用本发明可能使得所述应用例如以更快的速度满足非抽象目标。本发明的某些具体应用例如可能集中于（多种）数据存储系统，在这种情况下的具体性能目标可以是速度和复制品消除量。因此，对于真实世界数据产生更好性能（从我们的角度看）的散列函数可能往往在某一时刻或窗处的值与下一时刻或窗处的值之间具有极低的逐比特相关性。在所述窗从输入数据流中的某一时间滚动到下一时间时，所期望的散列函数很可能随机地影响所述散列值的所有比特。

[0032] 可能对于随机数据执行良好但是在真实生活中却失败的选择函数的一个例子是 $H = \text{"对 } 30 \text{ 字节窗内的非字母字符（即不在 [A-Za-z] 内）的数目进行计数"}$ 与选择规则 “ $H = 0$ ” 相耦合。对于随机数据输入， H 将随机地为 0，其概率是 $(204/256)^{**}30 \sim = 0.0011$ ，因此每 908 个窗当中的一个将产生选择，并且将预期到平均大约为 908 字节的分块尺寸的指数分布。不幸的是，如果存储新剪辑的档案，则包括 30 个字母的字的出现次数很有可能比上述预期低很多，因此可能仅得到很少的有用组块点。

[0033] 可能对于真实世界数据产生“更好”性能的散列函数将往往在某一时刻或窗处的值与下一时刻或窗处的值之间具有极低的逐比特相关性。在所述窗从输入数据流中的某一时间滚动到下一时间时，所期望的散列函数很可能会影响所述散列值的所有比特。在输入窗内的单一比特发生改变时，所期望的散列函数也很有可能随机地影响许多比特。类似的概念对应于良好散列函数特性的更为传统的定义，并且满足这些属性的函数可能导致良好的比特加扰能力以用于使用本发明的各种应用的目的。在任何情况下，通常可以假设即使在给出非随机的真实世界数据的情况下，上述散列函数也将很好地执行。

[0034] 所述矩形波串函数例如可能在各输入字节上加扰其散列值的最低有效位方面特别差，所述最低有效位仅仅无关紧要地与输入数据窗的 0 比特的奇偶校验相关。但是所述矩形波串函数对于连续输入数据窗评估散列的速度可能使其在用于本发明时特别有吸引力。此外，在某些实施例中可以修改对于矩形波串散列的选择标准，以避免使用 0 比特来实现所述选择函数。上面提到的伴随专利——即标题为“METHODS AND SYSTEMS FOR QUICK AND EFFICIENT DATA MANAGEMENT AND/OR PROCESSING（用于进行快速且高效的数据管理和 / 或处理的方法和系统）”的美国专利申请——提供了可以通过本发明大大改进与之对应的应用的速度和复制品消除的性能目标的大数据集的一个例子：其中测量到较大的速度提高，同时在 1.1 千兆字节真实生活数据集上验证表明，其复制品消除量与通过例如仅仅利用 Rabin 指纹识别或乘法线性同余发生器（MLCG）散列来实现内容定义的分块的几个实施例所获得的复制品消除量几乎完全相同。

[0035] 参照图 1，其中提供了根据本发明的至少一个实施例的使用多模式数据边界点 / 数据块确定的示例性数据管理或处理系统 100。在该示例性系统中，输入数据 110 可以被提供到较快边界点 / 数据块窗模块 120。该输入数据 110 例如可以是由二进制 1 和 0 构成的数据文件或分组的序列，其需要被处理、传送或存储在诸如微处理器、存储盘、存储器、计算

机系统等等之类的一个或多个电子设备中。所述模块 120 可以更加快速地在所述输入数据 110 中确定散列点、切割点、边界点或断点,这是因为该模块可以使用一种与用于确定良好的散列点、切割点、边界点或断点的传统处理相比花费更少处理时间的方法,例如矩形波串加法处理或函数。虽然模块 120 可能无法提供对所述数据中的散列点、切割点、边界点或断点的最佳选择,但是其速度将对所述较低质量(例如可能导致识别出较少复制数据)的散列点、切割点、边界点或断点做出补偿。在各实施例中,所述模块 120 可以预先选择所有可能的散列点、切割点、边界点或断点的子集。某些示例性的很容易滚动的散列函数可以包括矩形波串和函数、相继地乘以常数及加上字节值(MLCG)的函数、rolN-xor 函数等等。所述较快边界点 / 数据块窗模块 120 可以被耦合到较慢边界点 / 数据块窗模块 130。与模块 120 相比,模块 130 可以对于每次迭代花费相对较长的处理时间,但是该模块 130 能够做出更好的可能的散列点、切割点、边界点或断点确定(例如可能导致识别出更多复制数据)。在各实施例中,模块 130 可以利用可能合并有滚动或非滚动窗的 CRC32c 函数、Rabin 指纹函数、SHA-1 函数等等。在下面的表 I 中示出了各种切割点、断点、散列点、块点函数的表格,其中示出了在仿真期间达到的各个速度。

[0036]

名称	伪代码(非滚动)	速度 MB/s
矩形波串	hash+ = b	360
MLCG	hash = hash*A+b	270
rolN-xor	hash = ROL(hash, N)^b	280
rolN-xor[]	hash = ROL(hash, N)^A[b]	175
xor	hash^ = A[b]	170
xAdler	s1+ = b ; s2+ = s1 ; hash = s1^s2 ; hash^ = hash >> 6 ;	160
Rabin	hash = ((hash << 8) b)^ A[hash >> N]	145

[0037] 表 I

[0038] 在上面的表 I 中提供了一些简单的散列函数。为所述散列指定恒定初始值。对于所述窗内的每一个字节,通过添加下一个字节值 b 来修改所述散列。在这里列出的散列函数具有快速滚动版本。这里的 A 和 N 是特定于散列的常数,并且 ROL 是左旋操作。所述 xAdler 散列是 Adler(/Fletcher) 散列的简化版本。所述速度值表示具有最大数量的编译时间常数的相当优化的代码,并且应当被视为对于所述散列函数速度的合理指示。所述速度测试测量滚动所述散列,并且在几百兆字节的存储器内随机数据上检查终止 0 比特。从表 I 中可以看出,算法速度大致遵循所需要的外部查找的数目,其中矩形波串、MLCG 和

rolN-xor 散列最快,其表查找为 0。所述滚动版本(见下方)或 xor、rolN-xor[] 以及 Rabin 散列需要两次 A[b] 形式的查找。

[0039] 下面将说明对应于散列函数的各种示例性非滚动和相应的滚动版本的 C++ 类代码的特定版本。在所述示例性代码中, ws 是窗尺寸, buf 可以是以字节计的输入数据缓冲器。函数 calc(buf) 是对校验和 csum 的非滚动计算,而 roll(buf) 则是滚动版本,其将在更少的操作内把其值为 calc(buf) 的校验和更新到 calc(& buf[1]) 处的值。可以假设能够作为 rndTable[256] 获得随机常数的全局表,以用于需要把输入字节预先转换成更加随机的 32 比特量的算法。如果所述窗尺寸是编译器时间常数,则几种算法可以更加快速地滚动。实际的代码可以使用各种优化(比如编译器优化选项)来达到更快的速度。

[0040] 下面示出对应于非滚动 calc 和滚动 roll 例程的示例性矩形波串 C++ 类伪代码。

```
[0041] 1 void Boxcar::calc(unsigned char const*const buf) {
[0042] 2   csum = std::accumulate(buf, buf+ws, 0xfffffc03fU) ;
[0043] 3 }
[0044] 4 void Boxcar::roll(unsigned char const* & buf) {
[0045] 5   csum += buf[ws]-buf[0] ;
[0046] 6   ++buf ;
[0047] 7 }
```

[0048] 下面示出对应于非滚动 calc 和滚动 roll 例程的 MLCG C++ 类伪代码的一个例子,其使用了乘法线性同余发生器模块³²。在该选择下,可能不需要在 32 位机器上执行求模运算,这是因为 32 比特算术运算的上溢会隐含地执行这一功能。

```
[0049] 1 MLCG::MLCG(uint32_t windowsize)
[0050] 2   :ws(windowsize)
[0051] 3   , init(4329005U) // 种子值
[0052] 4   , mult(741103597U) // 用于伪随机序列的大素数
[0053] 5   , mult_ws(1UL)//multws
[0054] 6   , magic(int*(1U-mult))// 对应于滚动版本的预先计算
[0055] 7   {
[0056] 8     for(uint32_t i = 0U ;i < ws ;++i)mult_ws* = mult ;
[0057] 9     // (mult ws 实际上被更加高效地计算)
[0058] 10    magic* = mult_ws ;
[0059] 11  }
[0060] 12 void MLCG::calc(unsigned char const*const buf) {
[0061] 13   csum = init ;
[0062] 14   for(unsigned char const*p = buf, *pEnd = & p[ws] ;p != pEnd ;++p)
{
[0063] 15     csum = csum*mult+*p ;
[0064] 16   }
[0065] 17 }
[0066] 18 void MLCG::roll(unsigned char const* & buf) {
```

```
[0067] 19    csum = csum*mult - mult_ws*buf[0]+buf[ws]+magic ;
[0068] 20    ++buf ;
[0069] 21 }
```

[0070] 下面示出对应于非滚动 calc 和滚动 roll 例程的示例性 RolNxor C++ 类伪代码，其使用了 5 比特左旋函数 ro132 并且为简单起见采用 64 字节窗（从而对于所述 roll 函数不需要修正 rol 运算）。

```
[0071] 1 RolNxor::RolNxor(uint32_t windowsize, uint32_t rolbits)
[0072] 2   :ws(windowsize)
[0073] 3   ,N(rolbits)
[0074] 4   ,M((ws-1)*N) & 0x1fU
[0075] 5   ,init(0x356a356aU)// 某一恒定值
[0076] 6 {}
[0077] 7 void RolNxor::calc(unsigned char const*const buf) {
[0078] 8   csum = init ;
[0079] 9   for(unsigned char const*p = buf, *pEnd = &p[ws] ;p != pEnd ;++p) {
[0080] 10     csum = ro132(csum, N)^*p ;
[0081] 11   }
[0082] 12 }
[0083] 13 void RolNxor::roll(unsigned char const* & buf) {
[0084] 14   csum = ro132(csum^ro132(buf[0], M), N)^buf[ws] ;
[0085] 15   ++buf ;
[0086] 16 }
```

[0087] 下面示出对应于非滚动 calc 和滚动 roll 例程的示例性 RolNxor[] C++ 类伪代码，其使用了 5 比特左旋函数 ro132 并且为简单起见采用 64 字节窗（从而对于所述 roll 函数不需要修正 rol 运算）。

```
[0088] 1 RolNxorTable::RolNxorTable(uint32_t windowsize, uint32_t rolbits)
[0089] 2   :ws(windowsize)
[0090] 3   ,N(rolbits)
[0091] 4   ,M((ws-1)*N) & 0x1fU
[0092] 5   ,init(0U)// 某一恒定值
[0093] 6 {}
[0094] 7 void RolNxorTable::calc(unsigned char const*const buf) {
[0095] 8   csum = init ;
[0096] 9   for(unsigned char const*p = buf, *pEnd = &p[ws] ;p != pEnd ;++p) {
[0097] 10     csum = ro132(csum, N)^rndTable[*p] ;
[0098] 11   }
[0099] 12 }
[0100] 13 void RolNxorTable::roll(unsigned char const* & buf) {
[0101] 14   csum = ro132(csum^ro132(rndTable[buf[0]], M) ,
```

```
N)^rndTable[buf[ws]] ;  
[0102] 15  ++buf ;  
[0103] 16 }
```

[0104] 下面示出对应于非滚动 calc 和滚动 roll 例程的示例性 xor 散列 C++ 类伪代码，其与 rollNxor[] 完全相同，但是没有旋转运算。

```
[0105] 1 void XorTable::calc(unsigned char const*const buf) {  
[0106] 2   csum = init ;  
[0107] 3   for(unsigned char const*p = buf, *pEnd = & p[ws] ;p != pEnd ;++p) {  
[0108] 4     csum^ = rndTable[*p] ;  
[0109] 5   }  
[0110] 6 }  
[0111] 7 void XorTable::roll(unsigned char const* & buf) {  
[0112] 8   csum^ = rndTable[buf[0]]^rndTable[buf[ws]] ;  
[0113] 9   ++buf ;  
[0114] 10 }
```

[0115] 下面示出对应于非滚动 calc 和滚动 roll 例程的示例性 xAdler C++ 类伪代码，其利用使用了没有表查找的 Adler(/Fletcher) 散列的经修改的版本。所述例程表明一个滚动由 scramble 函数提供的内部状态的 roll 函数，同时在另一个不同步骤中产生校验和值。或者可以与更标准的 Adler 算法相组合地使用 rndTable[]。

```
[0116] 1 xAdler::xAdler(uint32_t windowsize)  
[0117] 2 :ws(windowsize)  
[0118] 3 , magic(1493U)// 为了略微增大受影响比特的范围，  
[0119] 4           // 其值与 ws 的预期尺寸相关  
[0120] 5 , s1(0U)//s1 和 s2 是内部状态  
[0121] 6 , s2(0U)// 其被用来产生校验和  
[0122] 7 {}  
[0123] 8 // 散列是从所述内部状态生成 csum 的某种方式  
[0124] 9 uint32_t scramble(uint32_t x, uint32_t y) {  
[0125] 10   x^ = y ;  
[0126] 11   x^ = (x >> 6) ;  
[0127] 12   return x ;  
[0128] 13 }  
[0129] 14 void xAdler::calc(unsigned char const*const buf) {  
[0130] 15   s1 = s2 = 0U ;  
[0131] 16   for(unsigned char const*p = buf, *pEnd = & p[ws] ;p != pEnd ;++p)  
{  
[0132] 17     s1+ = *p+magic ;  
[0133] 18     s2+ = s1 ;  
[0134] 19 }
```

```

[0135] 20 // 添加最终比特加扰
[0136] 21 csum = scramble(s1, s2) ;
[0137] 22 }
[0138] 23 void xAdler::roll(unsigned char const* & buf) {
[0139] 24 s1+ = buf[ws]-buf[0] ;
[0140] 25 s2+ = s1-ws*(buf[0]+magic) ;
[0141] 26 ++buf ;
[0142] 27 csum = scramble(s1, s2) ;
[0143] 28 }

```

[0144] 下面针对 CRC 的简单实现方式示出对应于非滚动 calc 和滚动 roll 例程的示例性 Rabin C++ 类伪代码。可以例如遵循以下文献计算常数表 T[256] 和 U[256] :A. Broder 的“Sequences II :Methods in Communications, Security, and Computer Science”, pp. 143152(1993)。

```

[0145] 1 Rabin::Rabin(uint32_t windowsize, uint32_t poly)
[0146] 2 :ws(windowsize)
[0147] 3 , p(poly)// 不可约生成多项式
[0148] 4 {
[0149] 5 // 预先计算与生成多项式相关的
[0150] 6 // 常数表 T[256] 和 U[256]。
[0151] 7 //T 描述如何快速地附加新的字节。
[0152] 8 //U 描述如何快速地添加 / 去除前导字节 (U 取决于窗尺寸 )。
[0153] 9 }
[0154] 10 // 添加字节将如下修改旧的 crc 值 :
[0155] 11 uint32_t append8(uint32_t crc, unsigned char byte) {
[0156] 12 return T[((crc >> 24)^byte) & 0xff]^ (crc << 8) ;
[0157] 13 }
[0158] 14 void Rabin::calc(unsigned char const*constbuf) {
[0159] 15 uint32_t csum = 0ULL ;
[0160] 16 for(unsigned char const*p = buf, *pEnd = & p[ws] ;p != pEnd ;++p) {
[0161] 17 csum = append8(csum, *p) ;
[0162] 18 }
[0163] 19 }
[0164] 20 void Rabin::roll(unsigned char const* & buf) {
[0165] 21 csum = append8(csum^U[buf[0]], buf[ws]) ;
[0166] 22 }

```

[0167] 在分块算法中使用表 I 中的简单散列函数时, 实际的速度可能低很多, 这是因为数据可能是以较小组块被提供的, 并且可能需要额外的缓冲器管理。此外, 分块应用通常可能还要求对于整个组块计算强散列函数。

[0168] 此外, 在各实施例中, 由模块 120 和模块 130 使用的方法可以是内容定义的散列并

且输出内容定义的块 / 组块点 140。一般来说，滚动窗函数比非滚动窗函数更快。如前所述，一般来说，基于内容的散列技术在识别出比文件或固定块尺寸散列技术更多的复制品方面的性能更好。在任何情况下，所述内容定义的块 / 组块点 140 都可以被使用来改进处理时间，同时对于处理输入数据保持合理的质量水平。在各实施例中，这些内容定义的块 / 组块点 140 可以被用来识别复制数据、改进加密、改进数据传输等等。应当注意到，虽然可以把模块 120 的输出馈送给模块 130，但是在这些模块中执行的处理可以在某种程度上并行或同时发生（例如用在双处理器系统中）。模块 130 在某些实施例中可以附加地直接使用输入数据 110。例如，如果确定所述模块 120 反常地无法满足预定标准，则模块 130 可以开始执行一种可能较慢的不那么合乎期望的处理，该处理通过一种用于内容定义的分块的替换机制来处理所述输入数据，如虚线 150 所示。可以通过一种例如使用已知的函数来确定将在步骤 140 中被用作输出的组块点集合的技术来提供这种替换机制。

[0169] 参照图 2，其中提供了根据本发明的至少一个实施例的示例性数据管理或处理方法 200，其利用了多模式数据边界点 / 数据块确定。在该实施例中，在步骤 210 中把数据流或数据作为输入提供到所述处理。接下来，在步骤 220 中执行对边界点 / 数据组块窗的预先选择。在各实施例中，可以利用快速散列函数来执行所述预先选择。一些示例性快速散列函数可以包括矩形波串和函数、rol-5-xor 函数、相继地乘以常数及加上字节值的函数等等。随后，可以在步骤 230 中利用一个更慢但是更好的确定性函数来识别边界点 / 数据组块窗的细化子集，所述确定性函数例如是识别出将最大化某一优选特性（比如对输入数据中的复制数据的识别）的散列点、切割点、边界点或组块点的质量散列函数。对应于所述更慢但是质量更高的散列的一些示例性处理可以例如包括 Rabin 指纹函数、SHA-1 函数等等，其可以合并滚动或非滚动窗。最后，所述处理可以在步骤 240 中输出内容定义的（多个）边界点 / 数据组块窗，其可以被用来例如确定散列值，以便在本发明的某些实施例中确定数据复制和 / 或复制消除。

[0170] 参照图 3，其中提供了根据本发明的至少一个实施例的数据管理或处理的示例性概念图示 300，其利用（多种）多模式数据边界点确定系统和方法进行操作。在该图中示出第一较快选择函数（305）和第二较慢选择函数（310）可以一起操作，以便从可能的数据点 / 窗的较大集合 320 当中更加快速地识别出（即每秒钟处理更多兆字节）所期望的散列点、切割点、边界点或组块点和 / 或窗 340。所述可能的散列点、切割点、边界点或组块点和 / 或窗的较大集合 320 例如可以包括数据点或组块 A(319a)、B(312a)、C(317a)、D(311a)、E(315a)、F(316a)、G(321a)、H(313a)、I(314a) 和 J(318a)。所述第一较快选择函数 305 可以处理所述可能的数据点 / 窗的较大集合 320，以便快速地选择所期望的散列点、切割点、边界点或组块点和 / 或窗的相当好的预先选择的子组 330。在该例中，被预先选择到子组 330 中的数据点和 / 或数据窗包括数据点和 / 或窗 B(312b)、D(311b)、E(315b)、H(313b) 和 I(314b)。所述第一较快选择函数 305 的处理速度例如可以是所述第二较慢选择函数 310 的近似 2 倍。但是与在仅仅由所述第二较慢选择函数 310 分析所述数据集 320 的情况下将由该第二较慢选择函数 310 所选择的那些数据散列点、切割点、边界点或断点和 / 或数据窗相比，由所述第一较快选择函数 305 所选择的数据散列点、切割点、边界点或断点和 / 或数据窗的性能可能较低或者可能不太合乎期望。另一方面，所述第一较快选择函数 305 可能提供合理数目的数据散列点、切割点、边界点或断点和 / 或数据窗 320，从而使得这些点在

统计上很可能包含具有大多数所期望特性的最高质量的许多数据断点和 / 或窗。因此，所述第二较慢选择函数 310 可以处理包括 B(312b)、D(311b)、E(315b)、H(313b) 和 I(314b) 的所述数据点和 / 或窗并且选择组 340，在该组 340 所包括的数据点和 / 或窗中包括具有高质量特性的 B(312c) 和 D(311c)，从而例如可以识别出大多数所期望的数据散列点、切割点、边界点或断点和 / 或数据窗。该处理可以继续重复，直到数据集或数据流中的所有数据都被处理。

[0171] 现在参照图 4，提供了根据本发明的至少一个实施例的用于数据管理或处理 400 的示例性方法，其使用多模式数据边界点确定来识别先前在数据集或数据流中遇到的数据。首先，可以在步骤 405 中提供输入数据以供处理。该输入数据 405 例如可以是由二进制 1 和 0 构成的数据的文件或分组的序列，其需要被处理、传送或存储在诸如微处理器、存储盘、存储器、计算机系统等等之类的一个或多个电子设备中。接下来，在步骤 410 中可以执行第一选择函数。该第一选择函数可以是较快选择函数，其基于预先确定的数据处理目标识别出可以在所述数据集或数据流中的什么位置处设置一个或多个逻辑断点或边界点。在各实施例中，所述第一选择处理 410 可以是相当快的散列函数，其用于识别出所述数据集或数据流中的散列点、切割点、边界点或断点，但是可能仅仅提供减弱的比特加扰能力。所述第一选择可以识别出所有可能的散列点、切割点或断点的一个子集，该子集可以被输入到第二选择函数。较快的第一选择函数可以被定义为能够基于每秒钟处理的数兆字节的输入数据而较快地给出校验和的函数。一些示例性第一选择函数 410 可以包括矩形波串和函数、rol-5-xor 函数、相继地乘以常数及加上字节值的函数等等，其中一些函数已经在上面更加详细地描述了，并且其中一些函数将在下面更加详细地描述。随后，可以在步骤 415 中执行第二选择函数，其在选择所述数据集或数据流中的优选的散列点、切割点、或边界断点方面具有更好的性能。在各实施例中，该第二选择函数可以慢于所述第一选择函数，但是在（基于所期望的特性，比如复制识别和 / 或消除）识别所述数据集或数据流中的优选的散列点、切割点、边界点或断点方面可以提供更好的辨识和 / 或精度。如上所述并且如在下面将更加详细地描述的那样，一些示例性第二选择函数 415 可以包括 CRC32c 函数、Rabin 指纹函数等等。在某些实施例中，所述第二选择函数可以仅仅考虑由所述第一选择标准所识别出的所述数据集或数据流中的散列点、切割点、边界点或断点的所述子集。这样，与单一模式选择技术相比，所述处理可以更加快速地执行并且花费更少处理时间。

[0172] 在其他实施例中，第二选择函数可以注意到所述第一选择函数的输入无法满足预先确定的标准，并且可以（在所述第二选择模块内）提供用来产生所述组块点的替换机制。这种替换机制的操作可以缓慢得多，并且这种替换机制可以优选地仅仅在异常情况下操作。例如，如果注意到由第一选择函数选择了每一个输入窗从而导致大量尺寸为 1 的组块，或者如果注意到第一选择函数正以统计上不可能的低速率（与基于随机输入的预期相比）产生组块，则第二选择函数可以判定第一选择函数是不足够的。在这种情况下，所调用的替换策略可以使用原始数据（如虚线 417 所示），以便例如利用某种分块技术产生最终组块点或断点，其中所述分块技术所使用的散列函数具有比由所述第一（快速）选择函数所提供的更好的比特加扰能力。

[0173] 接下来可以执行步骤 420 或 435 的其中之一或全部二者。在步骤 420 中，可以与先前遇到的底层数据一起识别包含在所述组块中的最近底层数据，从而可以把二者进行比

较。可以在步骤 420 中把所述最近底层数据与所有先前遇到的底层数据进行比较,以便检查是否存在任何匹配。例如,在各实施例中,可以把新的底层数据与先前存储或传送的底层数据进行比较。如果在步骤 420 中存在匹配,则在步骤 425 中不输出与先前遇到的底层数据相匹配的最近底层数据。例如,在各实施例中可以检测到复制品。在这种情况下,可以消除所述复制底层数据。另一方面,如果在步骤 420 中所述最近底层数据不等同于任何先前遇到的底层数据,则在步骤 430 中可以输出所述新的底层数据(例如存储或传送)。

[0174] 在步骤 435 中,可以生成一个所选择的(多个)数据组块或块的值,该值表示包含在所述(多个)数据组块或块中的底层数据。例如,可以生成散列值以表示包含在所述(多个)数据组块或块中的特定数据。用于生成所述值的一个示例性函数例如可以是 SHA-1 函数。接下来,在步骤 440 中可以与先前生成的值一起识别最近从步骤 435 生成的值,从而可以把二者进行比较。把所述最近生成的值与所有先前生成和遇到的值进行比较,以便检查是否存在任何匹配。例如,在各实施例中,可以把新生成的散列值与先前生成和 / 或存储或传送的散列值进行比较。如果在步骤 440 中存在匹配,则在步骤 445 中不能输出与先前生成的值相匹配的所述最近生成的值。例如,在各实施例中可能检测到复制品。在这种情况下,可以消除所述复制数据及其生成值,与所述在前存储的散列值相关的位置指示符表明存储数据的位置以便重新生成所消除的数据。另一方面,如果在步骤 440 中所述新生成的值(例如散列值)不等同于任何先前遇到的值,则在步骤 450 中可以输出所述新生成的值(例如散列值)和位置指示符,并且可以输出(例如存储或传送)其所表示的数据。在散列值的情况下,可以将其存储在散列表中以供将来参考。

[0175] 现在参照图 5,其中提供了根据本发明的至少一个实施例的示例性散列技术,该技术利用了内容定义的数据块或组块。内容定义的散列函数在操作时通过使用预定规则来确定将在数据集或数据流中的哪一点处设置散列点、切割点、边界点或组块点。在把所述规则(例如 X 个连续的 0 或 1)应用于所述数据集或数据流时,将得到随机的散列点、切割点、边界点、组块点或断点以及可变长度的数据组块或数据窗(例如,C₁525、C₂530、C₃535 和 C₄540 分别具有不同的长度),所述数据组块或数据窗可以被散列,以便产生散列值。作为另一个例子,所述内容定义的数据组块可以是基于 Rabin 函数,其中可以针对每一个滑动窗帧计算指纹(其是基于预定计算的对所述底层数据的唯一表示)。如果该计算的值例如与某一常数的 x 个最低有效位相匹配,则可以确定或标记出(例如选择)散列点、切割点、边界点、组块点或断点以供随后用于散列数据块或窗。图 5 的附图标记 500 示出了数据流 502,其中已经利用散列点、切割点、边界点、组块点或断点将该数据流 502 部分地分离成组块 C₁525、C₂530、C₃535 和 C₄540。这可以通过使用滑动窗方法来实现,并且把新的字节 510 加到滑动窗中可以在两部分中实现。所述数据流的窗区域可以是部分 520,其例如可以是 64 字节的窗。在所述滑动窗处理的每一个步骤中,可以减去该滑动窗的最早位置并且加上一个新的位置。可以从所述指纹中减去第一窗 515 中的最早字节 b_{i-64}505 的值,并且随后可以把新字节 b_i510 的值加到所述数据指纹上以便形成第二窗 518。可以通过使得所述指纹像简单的矩形波串函数中那样在加法上换向或者通过特定于为所述滑动窗帧选择的窗尺寸和指纹计算的更加复杂的机制而使得上述方法成为可能。如箭头 560 所示,随着时间的过去把新数据 555 提供到所述处理,所述处理将随着时间继续。在随着时间减去最早字节时,可以通过使用预先计算的表来提高所述计算的速度和性能。如上所述,该 Rabin 指纹可以被用作

第二选择函数以便通过其更好的比特加扰能力而找到性能更好的断点，并且其可以比所述第一选择函数更慢。但是还存在同样可能具有卓越的比特加扰能力并且可以被用作所述第二选择函数的替换函数。例如，可以使用 SHA-1 函数，或者可以使用 Rabin 指纹识别的特定版本，比如 CRC32c 散列。

[0176] 现在参照图 6，其中提供了根据至少一个实施例的示例性第一选择函数 600，其利用了矩形波串和函数。在该例中，矩形波串和函数被用来确定数据集或数据流中的散列点、切割点、边界点、组块点或断点。在这种情况下，同样可以使用滑动窗方法。此外，为了使得所述处理较为快速，所减去的旧数据块以及所加上的新数据块可以简单地是预定数目的数据字节或比特，并且所述计算可以简单地是把每一个所述数据块中的所有数据值加起来的和，从而：矩形波串和 = $\sum N$ 字节（从 $i = 1$ 到 n ）。因此，为了计算新的矩形波串和，可以减去最早一组数据和，并且可以加上最近一组数据求和。如图所示，数据集或数据流 605 被提供以供散列，其由数据组 a、b、c、d 和 e 构成。数据组 a、b、c、d 和 e 可以具有相等的比特或字节长度，例如 64 比特。第一窗 610 可以由数据组 a、b 和 c 构成。数据组“a”可以具有例如 15 的总和值（例如组 a 中的每个比特的每个值的和）。数据组“b”可以具有例如 10 的总和值。数据组“c”可以具有等于 5 的和。数据组“d”可以具有例如 10 的总和值。数据组“e”可以具有例如 5 的总和值。在针对所述数据流或数据集中的各数据组给出这些值的情况下，窗 610 的矩形波串和可以是 $15+10+5 = 30$ 。随后可以把下一个矩形波串和计算为由在前计算的和减去最早数据组 a640 并加上最新数据组 d650； $30-15+10 = 25$ 。随后可以把下一个矩形波串和计算为由在前计算的和减去最早数据组 b655 并且加上最新数据组 e660； $25-10+5 = 20$ 。如果所述函数被设置成在所述和等于 25 的位置处设置所述散列点、切割点、边界点、组块点或断点，则所述散列点、切割点、边界点、组块点或断点子集在该短实例中将包括点 670。可以看出，这可以是执行第一选择函数以便散列数据的一种特别简单且快速的方式，但是由于不够精细，因此所述散列点的质量可能略低。因此，所述矩形波串和方法可以产生潜在的散列点、切割点、边界点、组块点或断点的一个预选子集。

[0177] 现在参照图 7a 和 7b，提供了根据本发明的至少一个实施例的用于示例性散列技术的数据窗、块或组块，所述散列技术利用了内容定义的数据块或组块。图 7a 示出了在所述数据流中的各点处的分块 700，其中对应于各窗的指纹或散列值在所述各点处等于预定值。可以把各种函数与滚动、移动或滑动窗一起使用。该例可以表示涉及到滚动或滑动窗的较慢或较快选择函数，比如所述矩形波串和或 Rabin 指纹函数，并且所述等于预定值的标准可以被具有某一所期望的发生概率的另一种标准所取代。在该例中，滑动窗函数中的各窗（705a-735a）被用黑体示出并且被组织为所述数据集或数据流中的多个数据组，其中所述数据集或数据流包括数据组 a、b、c、d、e、f、g、h、i 和 j。在各时间点处形成一个新的窗并且参照预定值评估该窗，以便确定是否应当设置散列点、切割点、边界点、组块点或断点。在该例中，所述预定值导致在第二窗 710a 的末尾 740 处识别出的散列点、切割点、边界点、组块点或断点。因此，可以由数据块 a、b、c 和 d 的和中的数据得到散列值。随后，在另外两个滑动时间窗（即第三窗 715a 和第四窗 720a）处执行了散列之后，在第四窗 720a 的末尾 750 处识别散列点、切割点、边界点、组块点或断点。因此，可以由数据块 e 和 f 的和中的数据得到散列值。随后，在计算了另外两个滑动时间窗（即第五窗 725a 和第六窗 730a）的各窗中的数据值之后，在第六窗 730a 的末尾 760 处识别散列点、切割点、边界点、组块点或断

点。因此,可以由数据块 g 和 h 的和中的数据得到散列值。如图所示,所述处理继续到第七窗 735a,在该处没有找到另一个值匹配和没有得到散列点、切割点、边界点、组块点或断点。
[0178] 图 7b 示出了在所述数据流中的各点处的分块 765,其中对应于各窗的所述指纹或散列值在所述各点处等于预定值。该例示出了所述散列点、切割点、边界点、组块点或断点将如何由于引入新的数据组 z 而改变,该新数据组 z 已被插入在早前示出的具有数据组 a、b、c、d、e、f、g、h、i 和 j 的数据集或数据流内。从该图中可以看出,引入新的数据组可能导致消除之前在第四窗 720b 中出现的一个散列点、切割点、边界点、组块点或断点;在这种情况下,在 780 处没有选择散列点、切割点、边界点、组块点或断点。在该例中,一个滑动窗函数中的各窗 (705b-735b) 被用黑体示出并且被组织为所述数据集或数据流中的多个数据组,其中所述数据集或数据流具有数据组 a、b、c、d、e、f、g、h、i 和 j,新的数据组 z 被插入在 e 与 f 之间。在各时间点处形成一个新的窗并且参照预定值评估该窗,以便确定是否应当设置散列点、切割点、边界点、组块点或断点。在该例中,所述预定值导致在第二窗 710b 的末尾 770 处识别出的散列点、切割点、边界点、组块点或断点。因此,可以由数据块 a、b、c 和 d 的和中的数据得到一个散列值。随后,在另外两个滑动时间窗(即第三窗 715b 和第四窗 720b) 处执行了散列之后,遇到所述新的数据组 z,并且在第四窗 720b 的末尾 780 处没有识别出散列点、切割点、边界点、组块点或断点。随后,在计算了另外三个(这是由于所述滑动窗覆盖三个数据组)滑动时间窗(即第五窗 725b、第六窗 730b 和第七窗 735b) 的各窗中的数据值之后,在第七窗 735b 的末尾 790 处识别出散列点、切割点、边界点、组块点或断点。因此,可以由数据块 c、z、f、g 和 h 的和中的数据得到一个新的散列值。如图所示,插入新的数据组 z 可能在多达 3 个位置处改变所述选择函数的输出,这是因为所述底层数据的改变被限制到仅仅单一数据组并且所述滑动窗包括三个数据组。在图 7 中,假设其效果是使得所有三个受到影响的窗在选择标准“等于预定值”期间无效。基于对应于滑动窗函数的该例,可以认识到数据的改变或者不同的数据集如何能够导致变化的散列点、切割点、边界点、组块点或断点。在各实施例中,一旦识别出散列点、切割点、边界点、组块点或断点之后,在较快选择函数的情况下可以随后例如利用矩形波串和函数确定散列点、切割点、边界点、组块点或断点的一个子集(其中所述矩形波串和函数具有如上所述的滚动、滑动或移动窗),随后可以使用较慢的但是具有更好性能的第二选择函数。例如随后可以利用被预先选择为 Rabin 指纹函数的起点的所述散列点、切割点、边界点、组块点或断点的子集来执行该 Rabin 指纹函数,以便在保持散列点、切割点、边界点、组块点或断点选择的合理质量的同时减少处理时间。

[0179] 对应于所述第二选择函数的散列函数的一个特别好的例子是 CRC32c。所述 CRC32c 散列函数是 Rabin 指纹的一种特定实现方式,并且例如可以在 J. Stone、R. Stewart 和 D. Otis 的“Stream Control Transmission Protocol (SCTP) Checksum Change(流控制传输协议 (SCTP) 校验和改变)”(The Internet Society, RFC 3309, 2002 年 9 月, 第 1-17 页) 中找到。该文献可以说明所述 CRC32c 并且给出描述该散列函数的一种样本实现方式。所述 CRC32c 推荐实际上是基于多项式除法的散列的一种具体实现方式,因此其在某种程度上可以被视为 Rabin 散列的特定事例。所述 CRC32c 通常是 32 比特代码,并且可以或多或少地指定具体多项式和方法。可以对于 C++ 语言修改这些函数,并且可以将其修改成支持 64 比特多项式。对应于所述第二函数的代码可以是通用的(例如现成的)或者是硬连线的。如

果所述代码是“通用的”，则其可能略微较慢。

[0180] 如上所述，在各实施例中，本发明可以被用于数据识别和复制数据消除。因此，在确定了优选的散列点、切割点、边界点、组块点或断点之后，可以产生对应于所确定的数据组块的散列值并且将其与先前存储的散列值进行比较。在各实施例中，本发明可以特别适用于数据复制消除。此外，如上所述，本发明可以同样适用于各种电子系统，其中包括无线网络、因特网、内联网、计算机系统以及具有被处理、存储和 / 或传送的数据串的网络，其中对散列的使用可以被证明是有用的。下面将描述可以在其上操作本发明的一些系统。

[0181] 如上所述，在各实施例中，可以利用计算设备（比如个人计算机、服务器、小型计算机和 / 或大型计算机）来实现这里提供的（多种）系统和方法，所述计算设备可以被编程来执行指令序列，所述指令序列把计算机配置成执行这里描述的操作。在各实施例中，所述计算设备例如可以是可从任何制造商（比如 Dell Computer of Austin, Texas）获得的个人计算机，其例如运行 Windows™ XP™ 和 Linux 操作系统，并且具有标准外设集合（例如键盘、鼠标、显示器、打印机）。图 8 是计算设备 800 的一个实施例的功能方框图，其可用于充当实现这里描述的（多种）系统和方法的软件应用程序的主机。现在参照图 8，该计算设备 800 可以包括处理单元 805、（多个）通信接口 810、（多个）存储设备 815、用户接口 820、（多个）操作系统指令 835、应用可执行指令 /API840，所有这些都被提供在功能通信中并且例如可以使用数据总线 850。该计算设备 800 还可以包括系统存储器 855、数据和数据可执行代码 865、软件模块 860 以及（多个）接口端口。所述（多个）接口端口 870 可以被耦合到一个或多个输入 / 输出设备 875，比如打印机、（多个）扫描仪、打印机 / 扫描仪 / 传真机一体机等等。所述（多个）处理单元 805 可以是一个或多个微处理器或微控制器，其被配置成执行实现这里描述的函数的软件指令。可以利用计算设备 800 把应用可执行指令 /API840 和操作系统指令 835 存储在所述（多个）存储设备 815 和 / 或系统存储器 855 中，所述存储设备和 / 或系统存储器可以包括易失性存储器和非易失性存储器。应用可执行指令 /API840 可以包括实现本发明的（多种）系统和方法的软件应用程序。操作系统指令 835 可以包括适于控制所述处理器 805 的基本操作和控制的软件指令。在一个实施例中，操作系统指令 835 例如可以包括可从 Microsoft Corporation of Redmond, Washington 获得的 XP™ 操作系统。

[0182] 可以从另一个计算机可读介质（比如存储设备）中把指令读取到主存储器中。这里使用的术语“计算机可读介质”可以指代参与到把指令提供给所述处理单元 805 以供执行的任务中的任何介质。这种介质可以具有许多形式，其中包括（但不限于）非易失性介质、易失性介质和传输介质。非易失性介质例如可以包括光盘或磁盘、拇指或跳跃驱动器以及存储设备。易失性介质可以包括诸如主存储器或高速缓冲存储器的动态存储器。传输介质可以包括同轴电缆、铜线和光纤，其中包括构成所述总线 850 的连接。传输介质还可以具有声波或光波的形式，比如在射频 (RF) 和红外 (IR) 数据通信期间生成的波。计算机可读介质的常见形式例如包括软盘、软磁盘、硬盘、磁带、任何其他磁介质、通用串行总线 (USB) 记忆棒™、CD-ROM、DVD、任何其他光学介质、RAM、ROM、PROM、EPROM、闪速 EEPROM、任何其他存储器芯片或盒、下面描述的载波、或者计算机能够读取的任何其他介质。

[0183] 在把包括一个或多个指令的一个或多个序列载送到所述（多个）处理单元 805 以供执行的过程中可以涉及到多种形式的计算机可读介质。例如，所述指令最初可以处于

(多个)远程计算机 885(例如服务器、PC、大型计算机等等)的磁盘上。所述(多个)远程计算机 885 可以把所述指令加载到其动态存储器中,并且可以例如利用连接到调制解调器的电话线通过一个或多个网络接口 880 发送所述指令,所述调制解调器可以是模拟、数字、DSL 或有线电视调制解调器。所述网络例如可以是因特网、内联网、对等网络等等。所述计算设备 800 可以经由所述通信接口 810 通过(多个)其他计算机的网络发送消息及接收数据(其中包括(多个)程序代码),所述通信接口可以通过(多个)网络接口 880 耦合。服务器可以通过因特网传送所请求的应用程序代码以用于所下载的应用。可以由所述(多个)处理单元 805 在接收到所接收的代码时执行所述代码,并且 / 或者可以将其存储在存储设备 815 或其他非易失性存储装置 855 中以供随后执行。这样,所述计算设备 800 可以获得载波形式的应用代码。

[0184] 本发明的(多种)系统和方法可以驻留在单一计算设备或平台 800 上,或者可以驻留在多个计算设备 800 上,或者不同的应用可以驻留在单独的计算设备 800 上。应用可执行指令 /API840 和操作系统指令 835 可以被加载到计算设备 800 易失性存储器的一个或多个所分配的代码段中以供运行时间执行。在一个实施例中,计算设备 800 可以包括系统存储器 855,比如 512MB 的易失性存储器和 80GB 的非易失性存储装置。在至少一个实施例中,本发明的(多种)系统和方法的软件部分可以例如通过 C 编程语言源代码指令来实现。其他实施例也是可能的。

[0185] 应用可执行指令 /API840 可以包括一个或多个应用程序接口 (API)。本发明的(多种)系统和方法可以使用 API840 来进行处理间通信以及请求和返回应用间调用。例如,可以结合数据库 865 提供 API,以便于例如开发 SQL 脚本,其可用于使得所述数据库根据在所述(多个)脚本中指定的指令来执行特定数据存储或取回操作。一般来说,API 可以被用来促进开发应用程序,所述应用程序被编程来实现这里描述的某些函数。

[0186] 所述(多个)通信接口 810 可以为所述计算设备 800 提供通过因特网传送及接收信息的能力和文件传输能力,其中所述信息包括(但不限于)电子邮件、HTML 或 XML 页面。为此,所述通信接口 810 可以进一步包括 web 浏览器,其例如是(但不限于)由 Microsoft Corporation 提供的 Microsoft Internet ExplorerTM。所述(多个)用户接口 820 可以包括计算机终端显示器、键盘和鼠标设备。还可以包括一个或多个图形用户接口 (GUI) 以提供对包含在交互式 HTML 或 XML 页面中的数据的显示和操纵。

[0187] 现在参照图 9,其中示出了可以在其上操作所述(多种)系统和方法的网络 900。如上所述,本专利申请的(多种)系统和方法可以操作在一个或多个计算机上。所述网络 900 可以包括耦合到一个或多个客户端数据存储库 910 的一个或多个客户端 905。所述一个或多个客户端可以通过通信网络(例如光纤、电话线、无线网络等等)耦合到所述通信框架 930。所述通信框架 230 例如可以是因特网、内联网、对等网络、LAN、自组织计算机对计算机网络等等。所述网络 900 还可以包括耦合到所述通信框架 930 并且耦合到(多个)服务器数据存储库 920 的一个或多个服务器 915。本发明的(多种)系统和方法还可以具有适于操作在所述网络 900 中的一个或多个组件上的部分,以便作为完整的(多种)有效系统和方法进行操作。

[0188] 虽然在上面描述了本发明的实施例,但是本领域技术人员显然可以设想到许多替换方案、修改和变型。一般来说,各实施例可以涉及到在其中执行数据分析的上述和其他事

件处理的自动化。相应地，上面阐述的本发明的实施例应被视为说明性的，而不应被解释成限制本发明的范围。在不背离本发明的精神和范围的情况下可以做出许多修改。相应地，本发明的范围不应由上面说明的实施例决定，而是由所附权利要求书及其等效法律表述决定。

[0189] 这里引用的所有出版物、专利和专利申请都被全文合并在此以作参考。

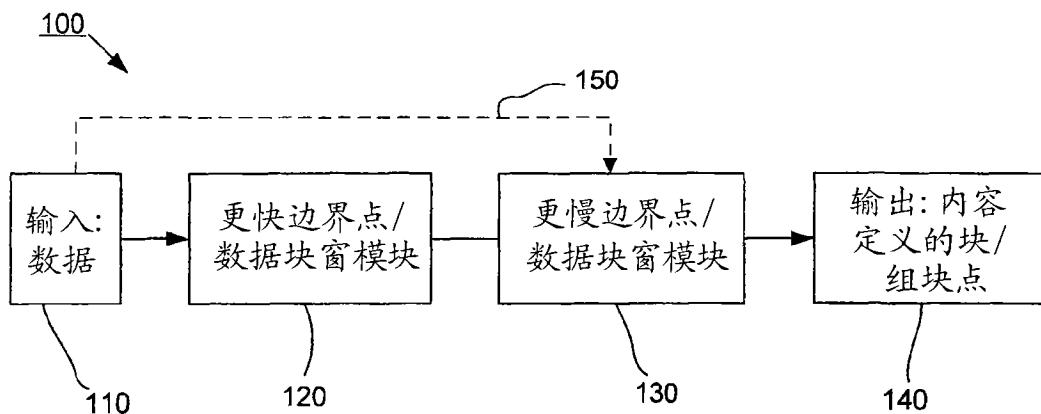


图 1

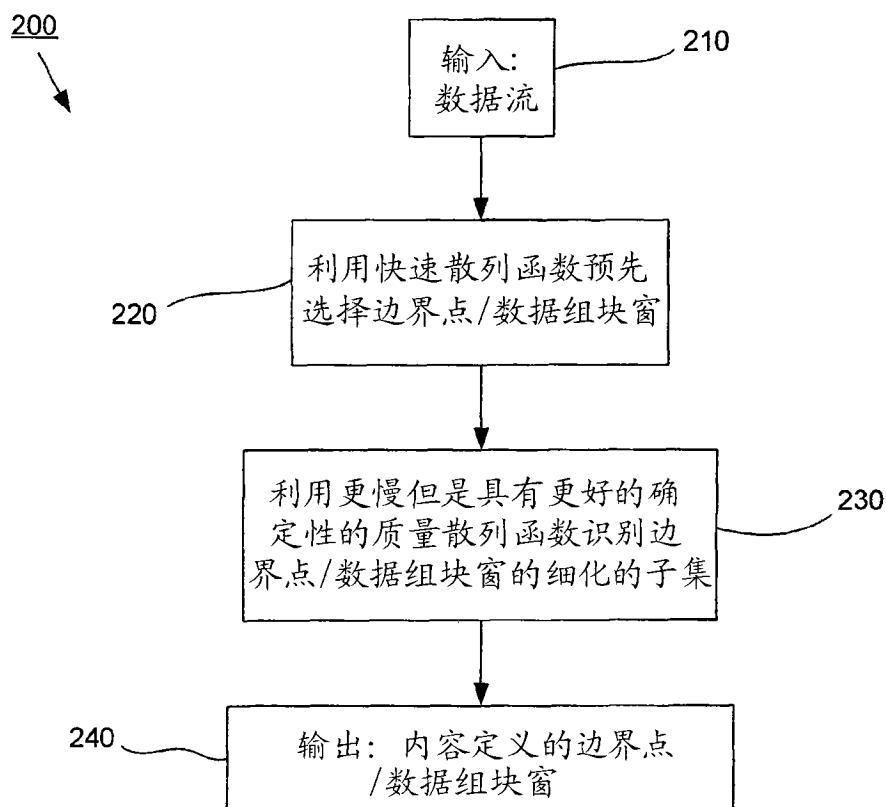


图 2

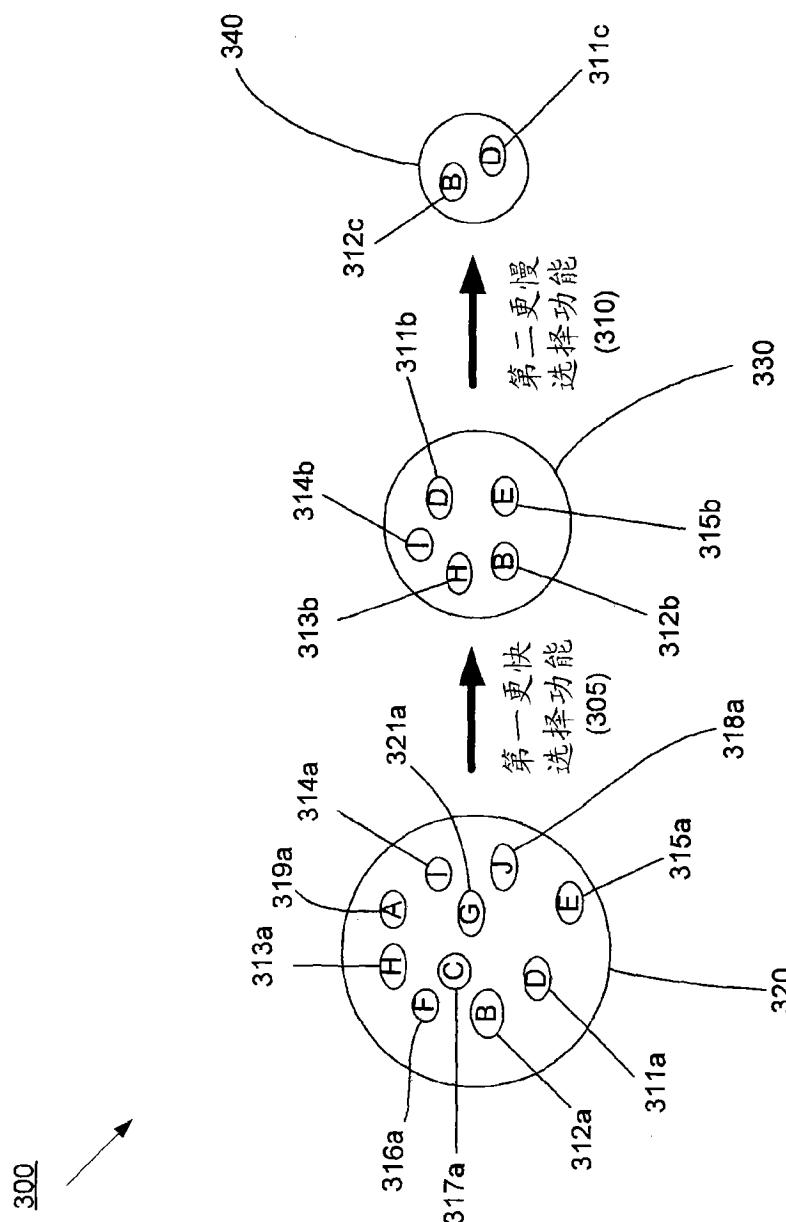


图 3

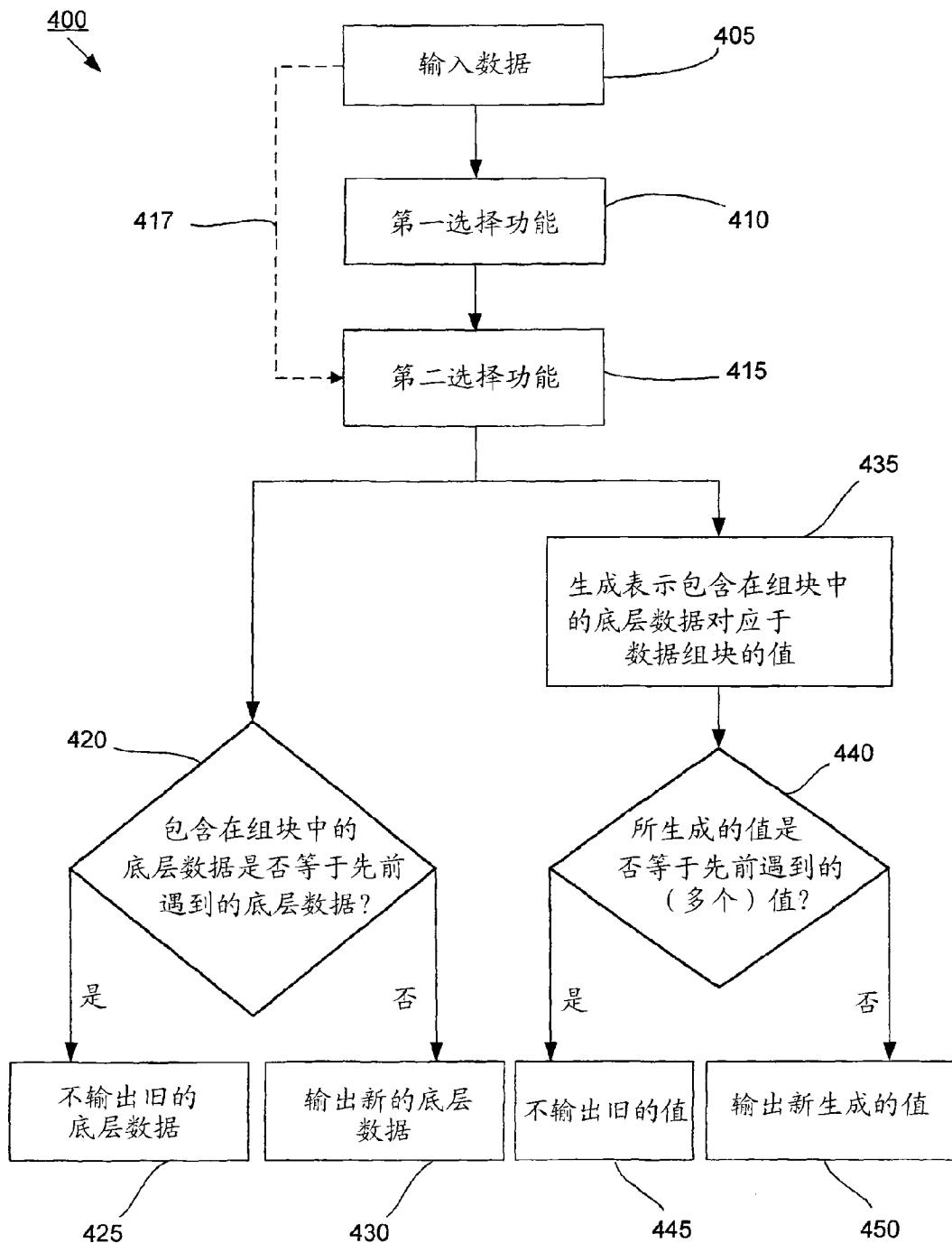


图 4

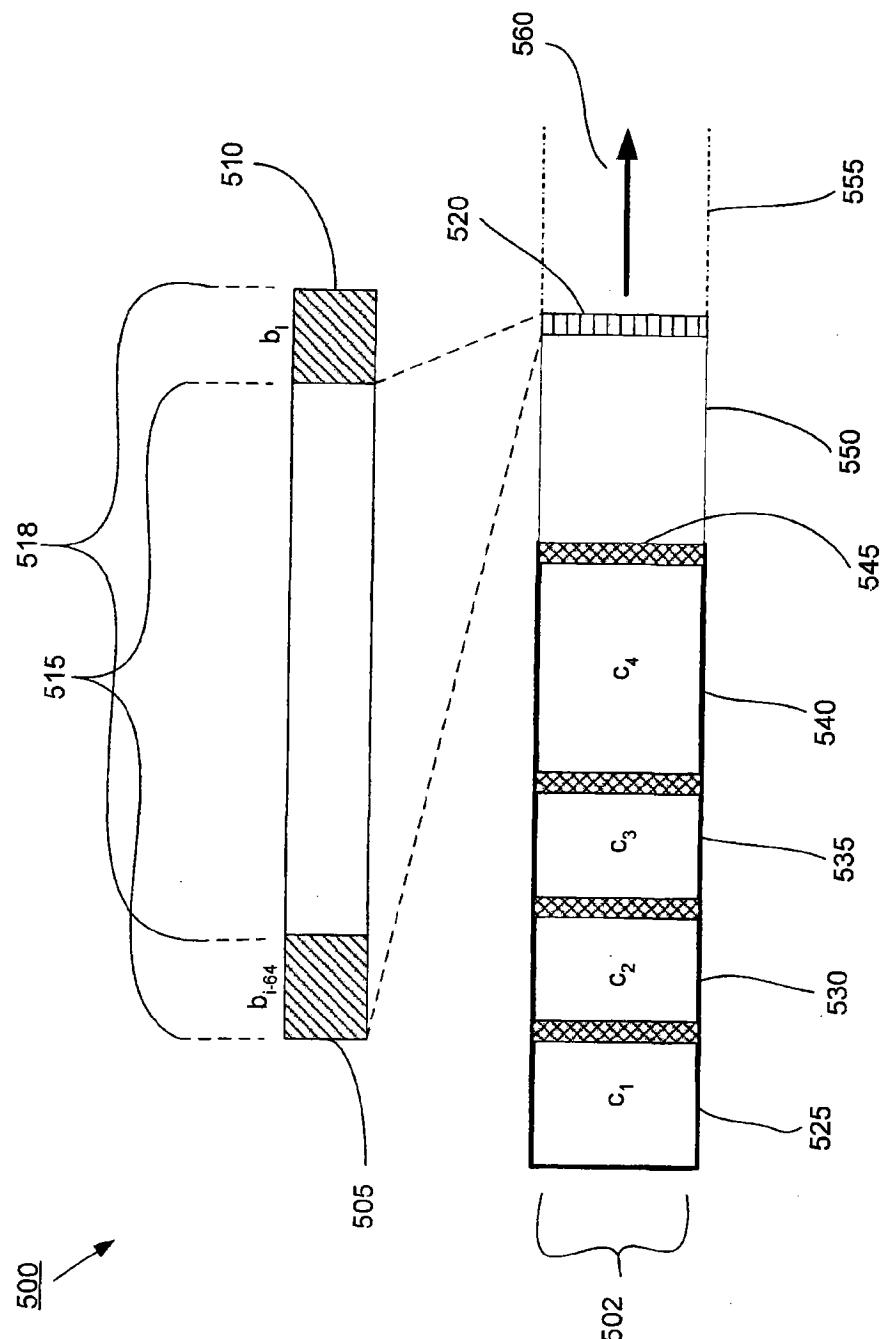


图 5

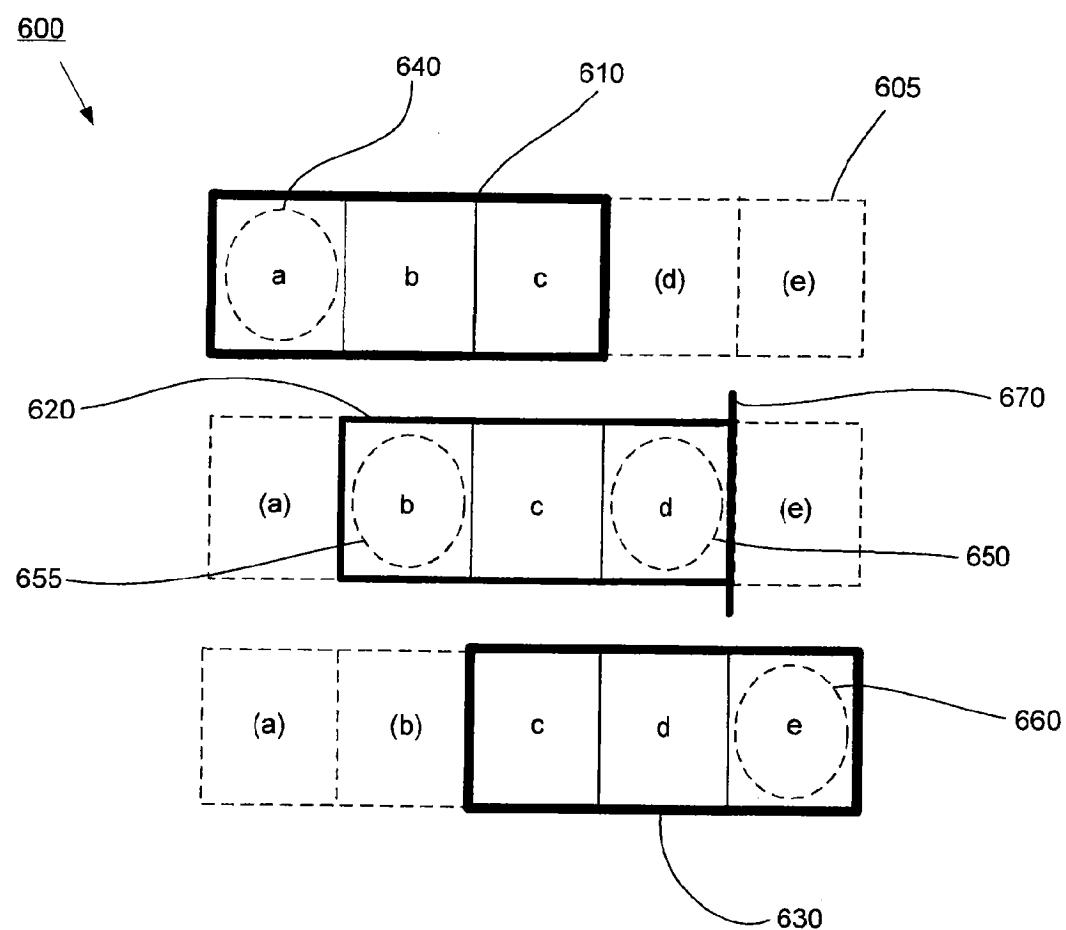


图 6

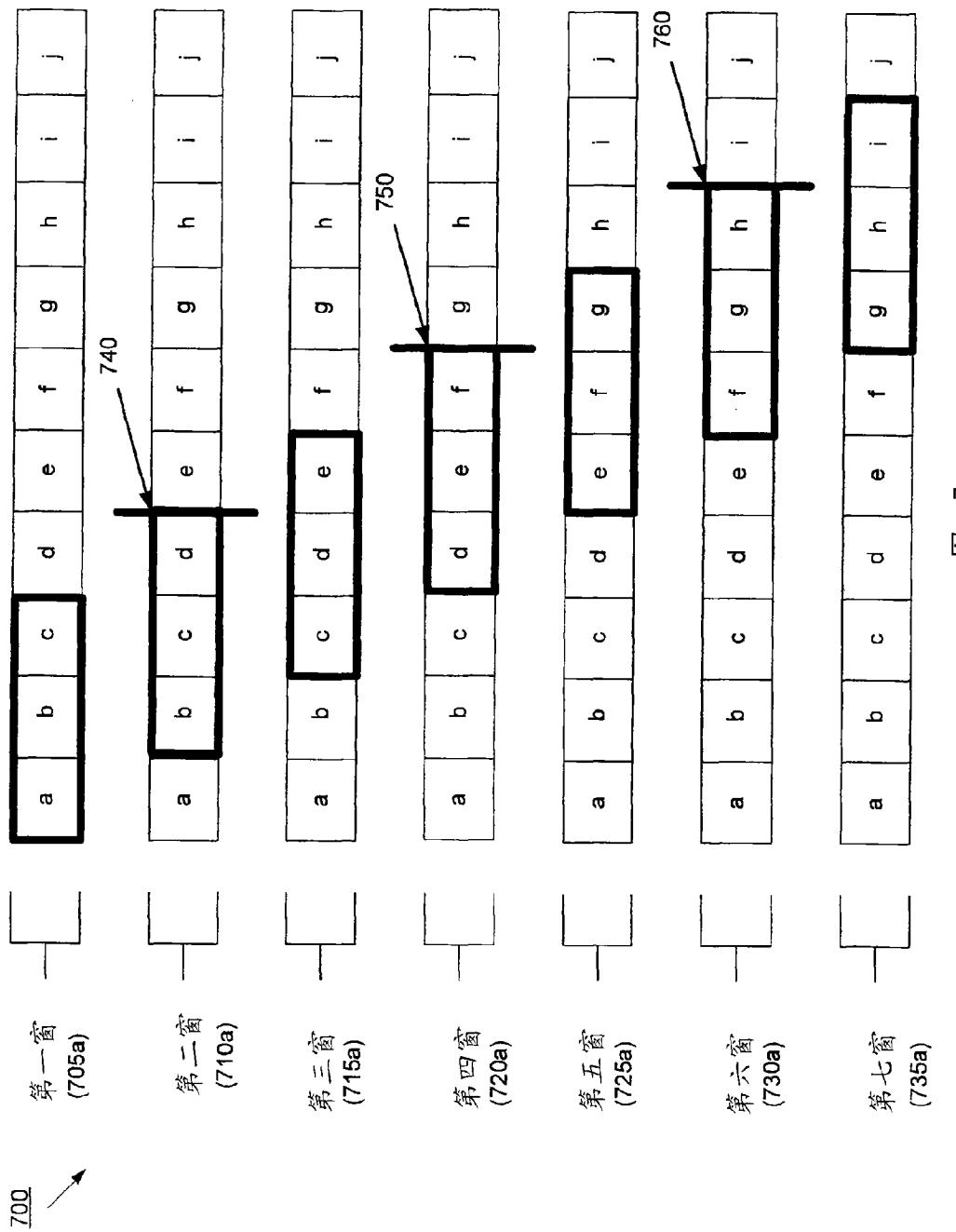
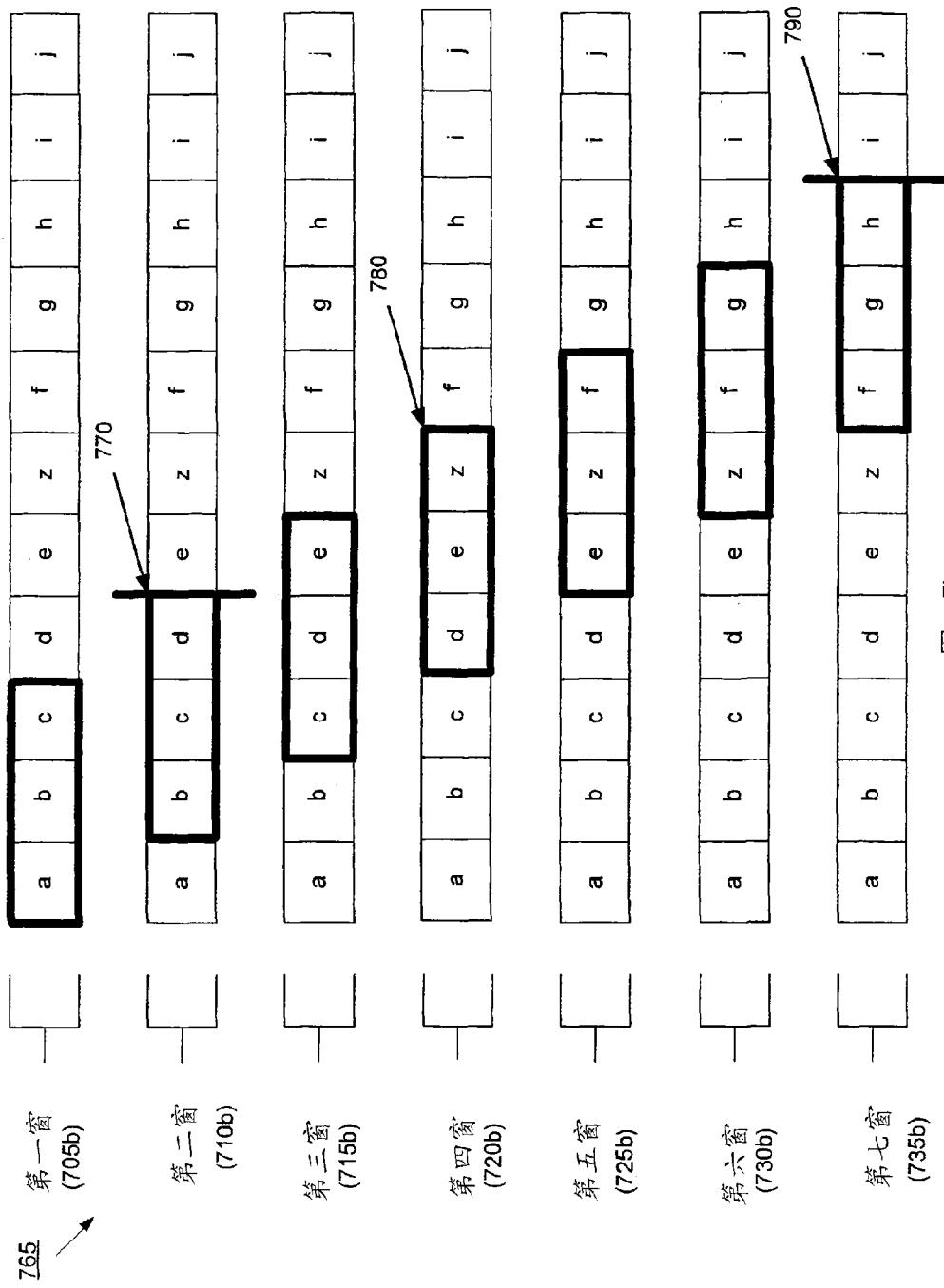


图 7a



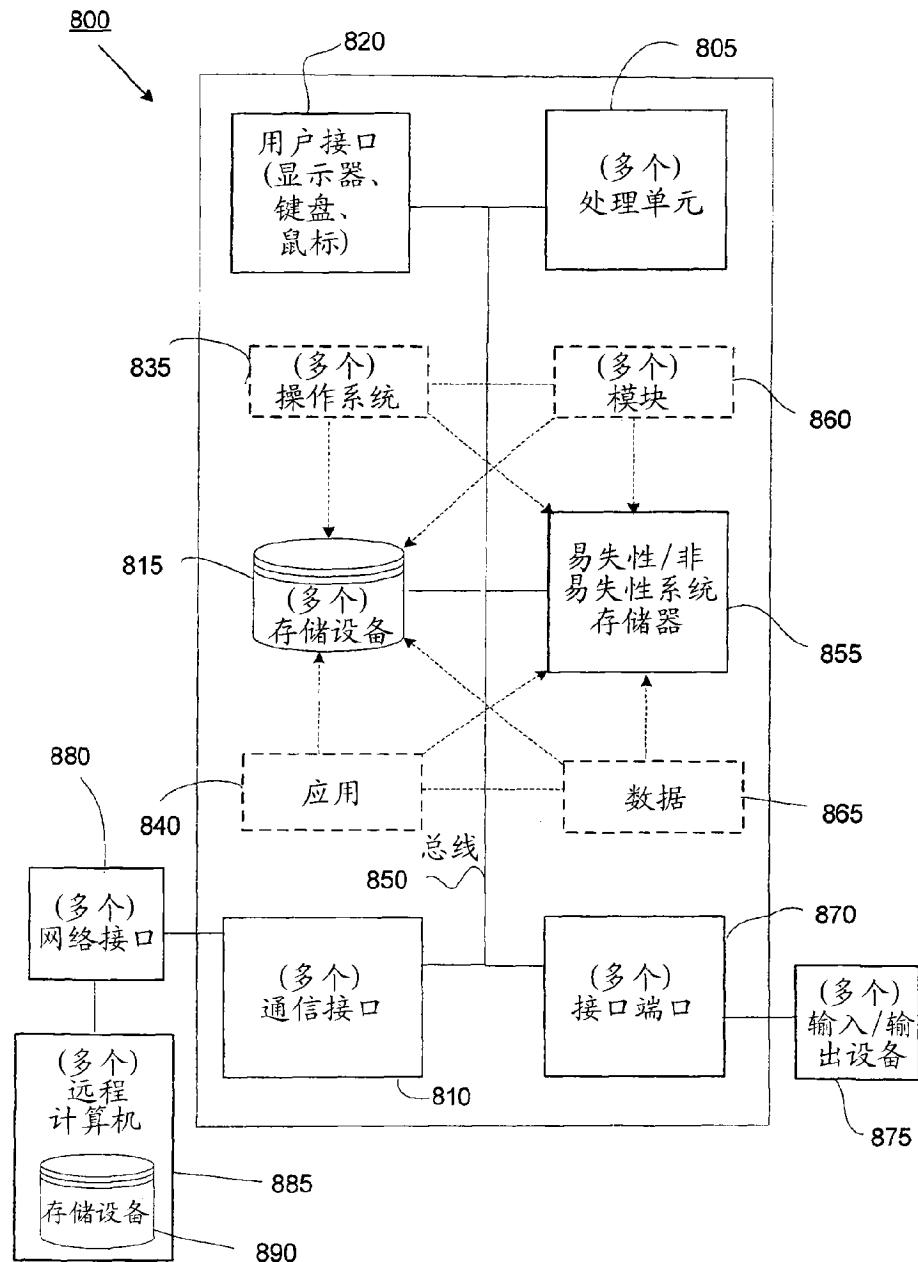


图 8

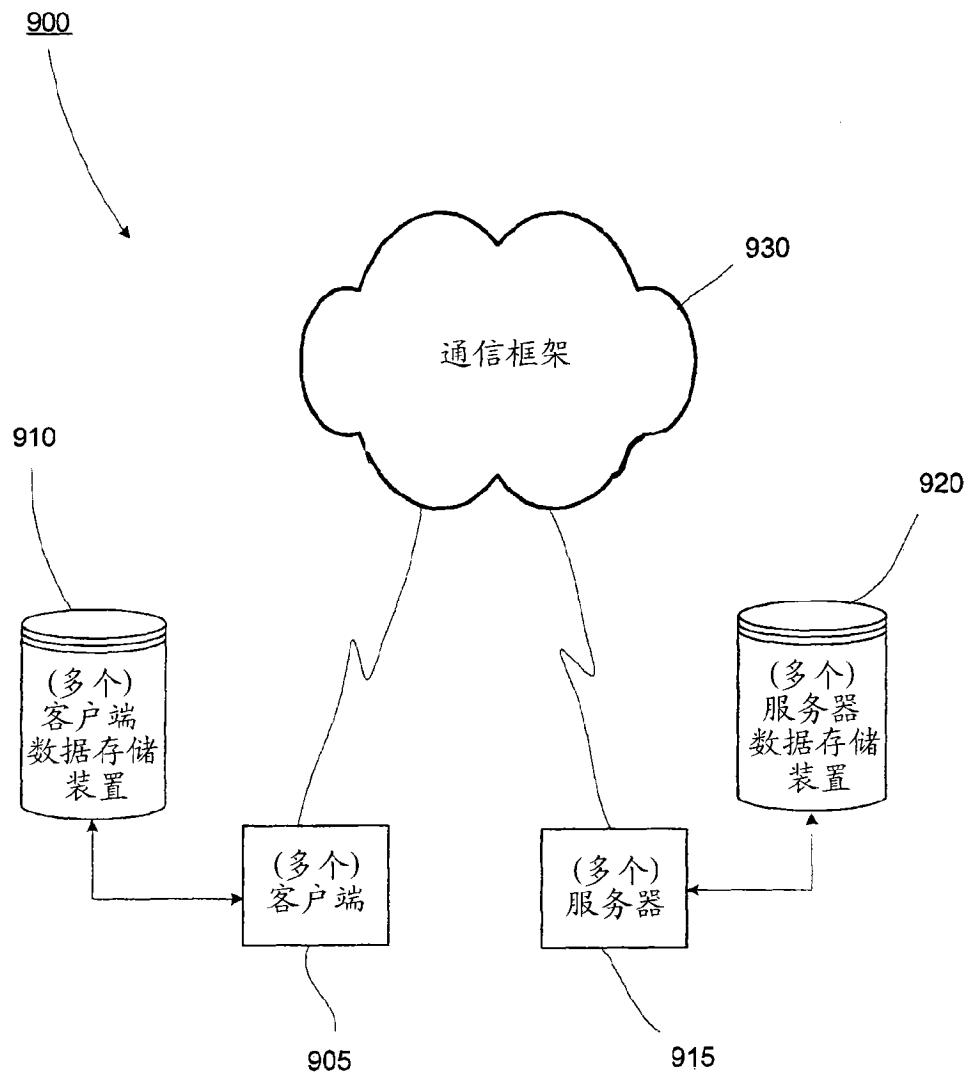


图 9