(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
7 August 2014 (07.08.2014)

WIPO | PCT

(10) International Publication Number
WO 2014/121102 A2

(51) International Patent Classification:
G06F 12/16 (2006.01)

(21) International Application Number:
PCT/US2014/014209

(22) International Filing Date:
31 January 2014 (31.01.2014)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

13/756,921	1 February 2013 (01.02.2013)	US
13/797,093	12 March 2013 (12.03.2013)	US
13/908,239	3 June 2013 (03.06.2013)	US

(72) Inventor; and
(71) Applicant : **IGNOMIRELO, Brian** [US/US]; 40 Manor Road, Colts Neck, NJ 07722 (US).

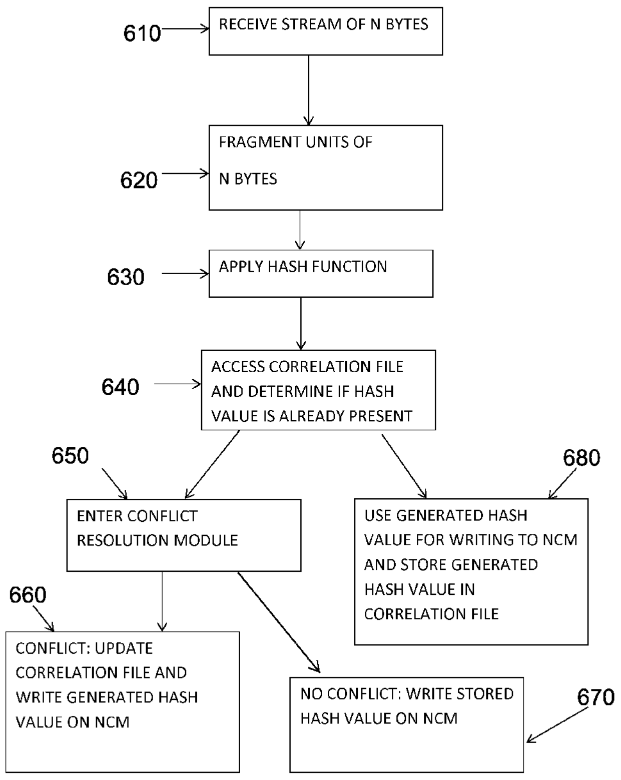
(74) Agent: **LOCKE, Scott, D.**; Dorf & Nelson LLP, The International Corporate Center, 555 Therdore Fremd Ave., Rye, NY 10580 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

[Continued on next page]

(54) Title: METHODS AND SYSTEMS FOR STORING AND RETRIEVING DATA


610
RECEIVE STREAM OF N BYTES
620
FRAGMENT UNITS OF N BYTES
630
APPLY HASH FUNCTION
640
ACCESS CORRELATION FILE AND DETERMINE IF HASH VALUE IS ALREADY PRESENT
650
ENTER CONFLICT RESOLUTION MODULE
680
USE GENERATED HASH VALUE FOR WRITING TO NCM AND STORE GENERATED HASH VALUE IN CORRELATION FILE
660
CONFLICT: UPDATE CORRELATION FILE AND WRITE GENERATED HASH VALUE ON NCM
NO CONFLICT: WRITE STORED HASH VALUE ON NCM
670
Figure 6

(57) Abstract: Through use of the technologies of the present invention, one is able to store and to retrieve data efficiently. In various embodiments, one may realize these efficiencies by coding the data and storing coded data that is of a smaller size than original data.

Methods and Systems for Storing and Retrieving Data

[0001] Field of the Invention

[0002] The present invention relates to the field of data storage and retrieval.

[0003] Background of the Invention

[0004] The twenty-first century has witnessed an exponential growth in the amount of digitized information that people and companies generate and store. This information is composed of electronic data that is typically stored on magnetic surfaces such as disks, which contain small regions that are sub-micrometer in size and are capable of storing individual binary pieces of information.

[0005] Because of the large amount of data that many entities generate, the data storage industry has turned to network-based storage systems. These types of storage systems may include at least one storage server that forms or is part of a processing system that is configured to store and to retrieve data on behalf of one or more entities. The data may be stored and retrieved as storage objects, such as blocks and/or files.

[0006] One system that is used for storage is a <u>N</u>etwork <u>A</u>ttached <u>S</u>torage (NAS) system. In the context of NAS, a storage server operates on behalf of one or more clients to store and to manage file-level access to data. The files may be stored in a storage system that includes one or more arrays of mass storage devices, such as magnetic or optical disks or tapes. Additionally, this data storage model may employ <u>R</u>edundant <u>A</u>rray of <u>I</u>ndependent <u>D</u>isks (RAID) technology.

[0007] Another system that is used for storage is a <u>S</u>torage <u>A</u>rea <u>N</u>etwork (SAN). In a SAN system, typically a storage server provides clients with block-level access to stored data, rather than file-level access to it. However, some storage servers are capable of providing clients with both file-level access and block-level access.

[0008] Regardless of whether one uses NAS or SAN, all industries that generate data must consider the cost of storing and retrieving that data. Therefore, there is a need for new technologies for economically storing and retrieving data.

[0009] Summary of the Invention

[0010] The present invention provides methods, systems and computer program products for improving the efficiency of storing and retrieving data. By using various embodiments of the present invention, one can efficiently store and access data that optionally has been converted or encoded. Additionally or alternatively, various embodiments of the present invention use a mediator that facilitates efficient storage of data and access to data.

[0011] Various embodiments of the present invention work with raw data and/or separate metadata from raw data. Consequently, there is no limitation based on the type of file that can be stored and/or retrieved in connection with the present invention. Examples of file types that may be used include, but are not limited to, JPEGs, PDFs, WORD documents, MPEGs and TXT documents. Through various embodiments of the present invention one may transform data and/or change the physical devices on which transformed or converted data is stored. These embodiments may be carried out through automated processes that employ a computer that comprises or is operably coupled to a computer program product that when executed causes the performance of the steps of the methods or processes of the present invention. These methods or processes may, for example, be embodied in or comprise a computer algorithm or script and optionally be carried out by a system of the present invention.

[0012] According to a first embodiment, the present invention is directed to a method for storing data on a recording medium comprising: (i) receiving a plurality of digital binary signals, wherein the digital binary signals are organized in a plurality of chunklets, wherein each chunklet is N bits long, wherein N is an integer number greater than 1 and wherein the chunklets have an order; (ii) dividing each chunklet into subunits of a uniform size and assigning a marker to each subunit from a set of X markers to form a set of a plurality of markers, wherein X equals the number of different combinations of bits within a subunit, identical subunits are assigned the same marker and at least one marker is smaller than the size of a subunit; and (iii) storing the set of the plurality of markers on a non-transitory recording medium in either an order that corresponds to the order of the chunklets or another configuration that permits recreation of the order of the chunklets.

[0013] According to a second embodiment, the present invention is directed to a method for retrieving data from a recording medium comprising: (i) accessing a recording medium, wherein the recording medium stores a plurality of markers in an order; (ii) translating the plurality of markers into a set of chunklets, wherein each chunklet is N bits long, wherein N is an integer number greater than 1 and wherein the chunklets have an order that corresponds to the order of the plurality of markers and wherein the translating is accomplished by accessing a bit marker table, wherein within the bit marker table each unique marker is identified as corresponding to a unique string of bits; and (iii) generating an output that comprises the set of chunklets. The markers may or may not be stored in an order that is the same as the order of the chunklets, but regardless of the order in which they are stored, one can recreate the order of the chunklets and any file from which they were derived.

[0014] According to a third embodiment, the present invention is directed to a method for storing data on a recording medium comprising: (i) receiving a plurality of digital binary signals, wherein the digital binary signals are organized in chunklets, wherein each chunklet is N bits long, each chunklet has a first end and a second end, N is an integer number greater than 1, and the chunklets have an order; (ii) dividing each chunklet into a plurality of subunits, wherein each subunit is A bits long; (iii) analyzing each subunit to determine if the bit at the second end has value 0 and if the bit at the second end has a value 0, removing the bit at the second end and all bits that have the value 0 and form a contiguous string of bits with the bit at the second end, thereby forming a revised chunklet for any chunklet that has a 0 at the second end; and (iv) on a non-transitory recording medium, storing each revised subunit and each subunit that is A bits long and has a 1 at its second end in a manner that permits reconstruction of the chunklets in the order. For example, the revised subunits (and any subunits that were not revised) may be organized in an order that corresponds to the order of the subunits within each chunklet prior to being revised, and storage of the data may be in the same order as the corresponding chunklets within the original file.

[0015] According to a fourth embodiment, the present invention provides a method for storing data on a recording medium comprising: (i) receiving a plurality of digital binary signals, wherein the digital binary signals are organized in chunklets, wherein each chunklet is N bits long, each chunklet has a first end and a second end, N is an integer number greater than 1, and the chunklets have an order; (ii) analyzing each chunklet to determine if the bit at the first end has a value 0 and if the bit at the first end has a value 0, removing the bit at the first end and all bits that have the value 0 and form a contiguous string of bits with the bit at the first end, thereby forming a first revised chunklet for any chunklet that has a 0 at the first end; (iii) analyzing each chunklet to determine if the bit at the second end has a value 0 and if the bit at the second end has a value 0, removing the bit at the second end and all bits that have the value 0 and form a contiguous string of bits with the bit at the second end, thereby forming a second revised chunklet for any chunklet that has a 0 at the second end; (iv) for each chunklet (a) if the sizes of the first revised chunklet and the second revised chunklet are the same, storing the first revised chunklet or the second revised chunklet, (b) if the first revised chunklet is smaller than the second revised chunklet, storing the first revised chunklet, (c) if the second revised chunklet is smaller than the first revised chunklet, storing the second revised chunklet, (d) if there are no revised chunklets, storing the chunklet, (e) if there is no first revised chunklet, but there is a second revised chunklet, then storing the second revised chunklet, (f) if there is no second revised chunklet, but there is a first revised chunklet, then storing the first revised chunklet, wherein each revised chunklet that is stored, is stored with information that indicates if one or more bits were removed from the first end or the second end. The information that indicates if one or more bits were removed from the first end or the second end may, for example, be in the form of the uniqueness of the subunit. When generating each of the first and second revised chunklets to be compared in any comparison, preferably 0's have been removed from either end, but not both ends.

[0016] According to fifth embodiment, the present invention provides a method for storing data on a recording medium comprising: (i) receiving a plurality of digital binary signals, wherein the digital binary signals are organized in chunklets, wherein each chunklet is N bits long, each chunklet has a first end and a second end, N is an integer number greater than 1, and the chunklets have an order; (ii) dividing each chunklet into a plurality of subunits, wherein each subunit is A bits long; (iii) analyzing each subunit to determine if the bit at the first end has a value 0 and if the bit at the first end has a value 0, removing the bit at the first end and all bits that have the value 0 and form a contiguous string of bits with the bit at the first end, thereby forming a first revised subunit for any subunit that has a 0 at the first end; (iv) analyzing each subunit to determine if the bit at the second end has value 0 and if the bit at the second end has a value 0, removing the bit at the second end and all bits that have the value 0 and form a contiguous string of bits with the bit at the second end, thereby forming a second revised subunit for any subunit that has a 0 at the second end; and (v) for each subunit (a) if the sizes of the first revised subunit and the second revised subunit are the same, storing the first revised subunit or the second revised subunit, (b) if the first revised subunit is smaller than the second revised subunit, storing the first revised subunit, (c) if the second revised subunit is smaller than the first revised subunit, storing the second revised subunit, (d) if there are no revised subunits, storing the subunit, (e) if there is no first revised subunit, but there is a second revised subunit, storing the second revised subunit, (f) if there is no second revised subunit, but there is a first revised subunit, storing the first revised subunit, wherein each revised subunit that is stored is stored with information that indicates if one or more bits were removed from the first end or the second end. The information that indicates if one or more bits were removed from the first end or the second end may, for example, be in the form of the uniqueness of the subunit.

[0017] According to a sixth embodiment, the present invention provides a method for retrieving data from a recording medium comprising: (i) accessing a recording medium, wherein the recording medium stores a plurality of data units in a plurality of locations, wherein each data unit contains a plurality of bits and the maximum size of the data unit is N bits, at least one data unit contains fewer than N bits and the data units have an order; (ii) retrieving the data units and adding one or more bits at an end of any data unit that is fewer than N bits long to generate a set of chunklets that corresponds to the data units, wherein each chunklet contains the same number of bits; and (iii) generating an output that comprises the set of chunklets in an order that corresponds to the order of the data units.

[0018] According to a seventh embodiment, the present invention is directed to a method for storing electronic data, said method comprising: (i) receiving a set of parameters, wherein the parameters comprise one or more of file system information, bootability information and partition information; (ii) receiving metadata; (iii) receiving one or more files, wherein each file has a file name; (iv) storing the parameters and metadata on a mediator; (v) storing each of the files on a non-cache medium at a location; and (vi) storing on the mediator, a correlation of each file name with a location on the non-cache medium. The mediators of the present invention may serve one or more of the following purposes: (1) storing a protocol for encoding data; (2) allocating physical space on recording media; (3) acting as a central point for a host initiator's disk geometry; (4) adding security; (5) allowing system internals to log, to read, and to interact with one or two reserves (R_1 and R_2); (6) providing frameworks for new ways to take snapshots (*i.e.,* freeze the data stored at a particular time) and/or to clone disks; and (7) to provide metadata. The realization of one or more if not all of these features can contribute to the efficiency of methods for storing data, protecting data from unauthorized access and/or retrieving data.

[0019] According to an eighth embodiment, the present invention is directed to a method for backing up data, said method comprising: (i) on a first mediator, correlating a plurality of file names with a plurality of locations of data files, wherein the locations of the data files correspond to locations on a first non-cache medium and the first mediator is configured to permit a user who identifies a specific file name to retrieve a data file that corresponds to the specific file name; (ii) copying the plurality of data files to a second non-cache medium; (iii) generating a second mediator, wherein the second mediator is a copy of the first mediator at time T1 and within the second mediator the locations of a plurality of data files on the second non-cache medium are correlated with the file names; (iv) receiving instructions to save revisions to a data file; and (v) at time T2, which is after T1, on the first non-cache medium saving the revisions to the data file. Preferably, the revisions are not saved in the corresponding data file on the second non-cache medium.

[0020] According to a ninth embodiment, the present invention provides a data storage and retrieval system comprising: (i) a non-cache data storage medium; (ii) a mediator, wherein the mediator is stored remotely from the non-cache data storage medium, and the mediator comprises: (a) a first set of tracks; (b) a second set of tracks; (c) a third set of tracks; and (d) a fourth set of tracks; and (ii) a manager, wherein the manager is configured: (a) to store data comprising one or more of file system information, bootability information and partition information in the first set of tracks; (b) to store metadata in the third set of tracks; (c) to store one or more files on the non-cache medium, wherein the one or more files are stored on the non-cache medium without any of file system information, bootability information and partition information; (d) to store in the fourth set of tracks the location of each file in the non-cache medium; and (e) to store a correlation of the location of each file in the non-cache medium with a host name for a file. Thus, the manager may cause storage of information on the mediator that may not be stored on the manager itself.

[0021] According to a tenth embodiment, the present invention is directed to a method for storing data on a non-cache recording medium comprising: (i) receiving an I/O stream of N Bytes (using for example an I/O protocol); (ii) fragmenting the N Bytes into fragmented units of X Bytes; (iii) applying a cryptographic hash function (which is a value algorithm) to each fragmented unit of X Bytes to form a generated hash function value for each fragmented unit of X Bytes; (iv) accessing a correlation file, wherein the correlation file associates a stored hash function value of Y bits with each of a plurality of stored sequences of X Bytes and (a) if the generated hash function value for a fragmented unit of X Bytes is in the correlation file, using the stored hash function value of Y bits for storage on a non-cache recording medium; and (b) if the generated hash function value for the fragmented unit of X Bytes is not in the correlation file, then storing the generated hash function value of Y bits with the fragmented unit of X Bytes in the correlation file and using the generated hash function value for storage on the non-cache recording medium.

[0022] According to an eleventh embodiment, the present invention is directed to a method for storing data on a non-cache recording medium comprising: (i) receiving an I/O stream of N Bytes; (ii) fragmenting the N Bytes into fragmented units of X Bytes; (iii) associating a cryptographic hash function value with each fragmented unit of X Bytes, wherein said associating comprises accessing a correlation file, wherein the correlation file associates a stored hash function value of Y bits with each of a plurality of stored sequences of X Bytes and (a) if the sequence of a fragmented unit of X Bytes is in the correlation file, using the stored hash function value of Y bits for storage on a non-cache recording medium; and (b) if the sequence of a fragmented unit of X Bytes is not in the correlation file, then storing a new generated hash function value of Y bits with the fragmented unit of X Bytes in the correlation file and using the new generated hash function value of Y bits for storage on the non-cache recording medium.

[0023] According to a twelfth embodiment, the present invention is directed to a method for storing data on a non-cache recording medium comprising: (i) receiving an I/O stream of N Bytes; (ii) applying a cryptographic hash function to each unit of N Bytes to form a generated hash function value for each unit of N Bytes; (iii) accessing a correlation file, wherein the correlation file associates a stored hash function value of Y bits with each of a plurality of stored sequences of N Bytes and (a) if the generated hash function value for the unit of N Bytes is in the correlation file, using the stored hash function value of Y bits for storage on a non-cache recording medium; and (b) if the generated hash function value for the unit of N Bytes is not in the correlation file, then storing the generated hash function value of Y bits with the unit of N Bytes in the correlation file and using the generated hash function value for storage on the non-cache recording medium.

[0024] Various methods of the present invention may, for example, be used in connection with the following method for configuring storage systems to store electronic data: (i) receiving a set of parameters, wherein the parameters comprise one or more, if not all of file system information, bootability information and partition information; (ii) receiving metadata; and (iii) storing the parameters and metadata on a mediator. After configuration of the storage system, the system may be ready to receive one or more files, wherein each file has a file name; to store each of the files (optionally as processed through one of the aforementioned methods for translating) on a non-cache medium at a location; and to store on the mediator, a correlation of each file name with a location on the non-cache medium.

[0025] When employing certain methods of the present invention, instructions may be stored on a computer program product that is encoded on a non-transitory computer-readable medium, operable to cause a data processing apparatus to perform operations comprising: (a) receiving, from a server, an I/O stream of N Bytes through an I/O protocol; (b) obtaining a hash function value for each unit of N Bytes or for a plurality of fragmented units of N Bytes; and (c) causing data to be stored that comprises either a plurality of hash values or a plurality of converted hash values. By converting the hash values into converted hash values, there may be heightened protection against unauthorized access to information within a stored file. Optionally, the computer program product further comprises a conflict resolution module that permits identification of the same hash function value being assigned to different units of N

Bytes or different fragmented units of N Bytes, and causes different data to be stored that corresponds to each of the conflicting hash function values. The computer program products, when executed, can cause initiation and automatic execution of the aforementioned features. These instructions may be stored in one module or in a plurality of separate modules that are operably coupled to one another.

[0026] Additionally, various embodiments of the present invention may also be implemented on systems. An example of a system of the present invention comprises: (a) a non-cache storage medium (which may be a persistent or non-transitory storage device) and (b) one or more processors operable to interact with the non-cache storage medium. Optionally, the system further comprises one or more user interfaces (*e.g.*, graphic use interfaces) that are capable of allowing a user to interact with the one or more processors. The one or more processors are further operable to carry out one or more of the methods of the present invention, which may, for example, be stored on a computer program product that is stored in a non-transitory medium.

[0027] In some embodiments, the system comprises: (i) a mediator, wherein the mediator is stored remotely from the non-cache data storage medium, and the mediator comprises: (a) a first set of tracks; (b) a second set of tracks; (c) a third set of tracks; and (d) a fourth set of tracks; (ii) a non-cache medium; and (iii) a manager, wherein the manager is configured: (a) to cause storage of data comprising one or more of file system information, bootability information and partition information in the first set of tracks; (b) to cause storage of metadata in the third set of tracks; (c) to cause storage of one or more files on the non-cache medium, wherein the one or more files are stored on the non-cache medium without any of file system information, bootability information and partition information; (d) to cause storage in the fourth set of tracks of the location of each file in the non-cache medium; and (e) to cause storage of a correlation of the location of each file in the non-cache medium with a host name and/or address(es) for a file.

[0028] Through the various embodiments of the present invention, one can increase the efficiency of storing and retrieving data. The increased efficiency may be realized by using less storage space than is used in commonly used methods and investing less time and effort in the activity of storing information. In some embodiments, one can also increase protection against unauthorized retrieval of data files. These benefits may be realized when storing data either remotely or locally, and the various embodiments of the present invention may be used in conjunction with or independent of RAID technologies.

[0029] Brief Description of the Figures

[0030] Figure 1 is a representation of an overview of a system of the present invention.

[0031] Figure 2 is a representation of a mediator and non-cache medium (NCM).

[0032] Figure 3 is a representation of a system for storing information using a mediator.

[0033] Figure 4 is a representation of a system for using two mediators to back up information that is stored.

[0034] Figure 5 is a representation of a binary tree.

[0035] Figure 6 is a flow chart that represents a method of the present invention.

[0036] Detailed Description of the Invention

[0037] Reference will now be made in detail to various embodiments of the present invention, examples of which are illustrated in the accompanying figures. In the following detailed description, numerous specific details are set forth in order to provide a thorough understanding of the present invention. However, unless otherwise indicated or implicit from context, the details are intended to be examples and should not be deemed to limit the scope of the invention in any way. Additionally, headers are used for the convenience of the reader but are not intended to limit the scope of any of the embodiments of the present invention or to suggest that any features are limited to sections within a particular header.

[0038] *Definitions*

[0039] Unless otherwise stated or implicit from context, the following terms and phrases have the meanings provided below.

[0040] The term "bit" refers to a binary digit. It can have one of two values, which can be represented by either 0 or 1.

[0041] The term "block" refers to a sequence of bytes or bits of data having a predetermined length.

[0042] The phrases "bootability code," "bootability information" and "bootability feature" refer to information that provides the means by which to enter a bootable state and may be stored on a boot sector. A boot sector may contain machine code that is configured to be loaded into RAM (random access memory) by firmware, which in turn allows the boot process to load a program from or onto a storage device. By way of example, a master boot record may contain code that locates an active partition and invokes a volume boot record, which may contain code to load and to invoke an operating system or other standalone program.

[0043] The term "byte" refers to a sequence of eight bits.

[0044] The term "cache" refers to the location in which data is temporarily stored in order for future requests for the data to be served faster or for the purposes of buffering. The L1 cache (level 1 cache) refers to a static memory that is, for example, integrated with a processor core. The L1 cache may be used to improve data access speed in cases in which the CPU (central processing unit) accesses the same data multiple times. The L2 cache (level 2 cache) is typically larger than the L1 cache, and if a data file is sought but not found in a L1 cache, a search may be made of a L2 cache prior to looking to external memory. In some embodiments, the L1 cache is not within a central processing unit. Instead, it may be located within a DDR, DIMM or DRAM. Additionally or alternatively, L2 cache may be part of PCI2.0/3.0, which goes into a motherboard. Thus, each of L1 cache and L2 cache may be in separate parts of a motherboard. With respect to size, in some embodiments of the present invention L1 cache is between 2 gigabytes and 128 terabytes or between 2 gigabytes and 4 terabytes; and L2 cache is between 16 gigabytes and 1 petabyte or between 16 gigabytes and 3.2 terabytes. In some embodiments, when the methods of the present invention are implemented, one or more of a bit marker table, a frequency converter or a hash value table resides in L2 cache.

[0045] The term "chunklet" refers to a set of bits that may correspond to a sector cluster. The size of a chunklet is determined by the storage system and may have a chunklet size. Traditionally, the chunklet size was derived by the CHS scheme, which addressed blocks by means of a tuple that defines the cylinder, head and sector at which they appeared on hard disks. More recently, the chunklet size has been derived from the LBA measurement, which refers to logical block addressing, and is another means for specifying the location of blocks of data that are stored on computer storage devices. By way of example, the chunklet size may be 512B, 1K, 2K, 4K, 8K, 16K, 32K, 64K or 1MB. As persons of ordinary skill in the art are aware 1K = 1024B. Chunklets may be received as raw data from a host.

[0046] A “file” is a collection of related bytes or bits having a length. A file may be smaller than a chunklet, the same size as a chunklet or larger than a chunklet.

[0047] The phrase “file name” refers to a notation or code that permits a computer to identify a specific file and to distinguish that file from other files.

[0048] The phrase “file system” refers to an abstraction that is used to store, to retrieve and to update a set of files. Thus, the file system is the tool that is used to manage access to the data and the metadata of files, as well as the available space on the storage devices that contain the data. Some file systems may, for example, reside on a server. Examples of file systems include but are not limited to the Unix file system and its associated directory tables and inodes, Windows FAT16 and FAT32 file systems (FAT refers to File Allocation Table), Windows NTFS, which is based on master file tables, and Apple Mac OSX, which uses HFS or HFS plus.

[0049] The phrases “hash function,” “cryptographic hash function value algorithm” and “hash function value algorithm” refer to an algorithm or subroutine that maps large data sets (optionally of variable length) to smaller data sets that have a fixed length for a particular hash function. A “hash function value” refers to the output that is returned after application of a hash function algorithm. The values that the algorithm returns may also be called hash values, hash codes, hash sums, checksums or hashes. When, for example, using MD5, the output is 128 bits, whereas when using SHA-1, the output is 160 bits.

[0050] The terms “host” and “initiator” may be used interchangeably and refer to the entity or system that sends data for storage to the data storage and retrieval mediation system of the present invention. The host may send data that corresponds to one or more types of documents or files and received data. Preferably, within any input/output ("I/O") stream, the data corresponds to a file of a single document type. For convenience the phrase "I/O stream" is used to refer that data is transmitted from on entity to another. What is output for a first entity may be input for a second entity.

[0051] The abbreviation "LBA" refer to logical block addressing. LBA is a linear addressing scheme and is a system that is used for specifying the location of blocks of data that are stored in certain storage media, *e.g.*, hard disks. In a LBA scheme, blocks are located by integer numbers and only one number is used to address data. Typically, the first block is block 0.

[0052] The abbreviation "LUN" refers to a logical unit number and is a number that is used to identify a logical unit. LUNs are commonly used to manage block storage arrays that are shared over a SAN.

[0053] The term "manager" refers to a computer program product, *e.g.*, code that may be stored in a non-transitory medium and that causes one or more other actions to be taken, *e.g.*, receiving, transmitting, storing or processing data. A manager may be stored on hardware, software or a combination thereof. In some embodiments, the manager may be part of a computer and/or system that is configured to permit the manager to carry out its intended function.

[0054] The term "mediator" refers to a computer program product that may be stored on hardware, software or a combination thereof, and that correlates one or more units of storage space within at least one non-cache medium with a file name. A mediator may be orders of magnitude smaller than the non-cache medium to which it points. For example, it may be approximately as small as about 0.2% of the size of a typical cylinder. In some embodiments, the mediator may exist in a computing cloud, whereas in other embodiments, it exists in a non-transitory tangible recording medium. The mediator may be able to organize, to convert, to translate and to control the storage of data in locations that hosts perceive as being in certain tracks of recording media while actually occurring in different tracks of recording media or it may be operably coupled to a manager that serves one or more if not all of these functions. Furthermore, the mediator may comprise a sector map, a table or other organization of data that may be located within a physical device or structure, and thus the contents of the mediator may cause the physical device or structure to have certain geometry. In some embodiments, the mediator resides permanently on L2 cache. In other embodiments it reside on a solid state drive.

[0055] The term "metadata" refers to the administration information about containers of data. Examples of metadata include, but are not limited to, the length or byte count of files that are being read; information pertaining to the last time files were modified; information that describes file types and access permissions; and LUN QoS, VM and WORM. Other types of metadata include operating system information, auto-initialization information, group permissions, and frequency of bits within the document type. In some embodiments, stored metadata may, for example, be used to permit efficient contraction or expansion of storage space for an initiator as the number and size of documents that it seeks to store shrinks or grows.

[0056] The abbreviation "NAS" refers to network area storage. In a NAS system, a disk array may be connected to a controller that gives access to a local area network transport.

[0057] The phrase "operably coupled" means that systems, devices and/or modules are configured to communicate with each other or one another and are able to carry out their intended purposes when in communication or after having communicated.

[0058] The phrase "operating system" refers to the software that manages computer hardware resources. Examples of operating systems include, but are not limited to, Microsoft Windows, Linux, and Mac OS X.

[0059] The term "partition" refers to formats that divide a storage medium, *e.g.*, a disk drive into units. Thus, the partition may also be referred to as a disk partition. Examples of partitions include, but are not limited to, a GUID partition table and an Apple partition map.

[0060] The abbreviation "RAID" refers to a redundant array of independent disks. To the relevant server, this group of disks may look like a single volume. RAID technologies improve performance by pulling a single strip of data from multiple disks and are built on one or multiple premise types such as: (1) mirroring of data; (2) striping of data; or (3) a combination of mirroring and striping of data.

[0061] The phrase "recording medium" refers to a non-transitory tangible computer readable storage medium in which one can store magnetic signals that correspond to bits. By way of example, a recording medium includes but is not limited to a non-cache medium such as hard disks and solid state drives. As persons of ordinary skill in the art know, solid state drives also have cache and do not need to spin. Examples of non-transitory tangible computer readable storage media include, but are not limited to, a hard drive, a hard disk, a floppy disk, a computer tape, ROM, EEPROM, nonvolatile RAM, CD-ROM and a punch card.

[0062] The abbreviation "SAN" refers to a storage area network. This type of network can be used to link computing devices to disks, tape arrays and other recording media. Data may, for example, be transmitted over a SAN in the form of blocks.

[0063] The abbreviation "SAP" refers to a system assist processor, which is an I/O (input/output) engine that is used by operating systems.

[0064] The abbreviation "SCSI" refers to a small computer systems interface.

[0065] The term "sector" refers to a subdivision of a track on a disk, for example, a magnetic disk. Each sector stores a fixed amount of data. Common sector sizes for disks are 512 bytes (512B), 2048 bytes (2048B), and 4096 bytes (4K). If a chunklet is 4K in size and each sector is 512B in size, then each chunklet corresponds to 8 sectors (4*1024/512 = 8). Sectors have tracks and are located on platters. Commonly, two or four platters make up a cylinder, and 255 cylinders make up hard disk and media devices.

[0066] The phrase "sector map" refers to the tool that receives calls from a host and correlates locations in a storage device where a file is stored. A sector map may, for example, operate under parameters that are defined by a SCSI protocol. In some embodiments of the present invention, the sector map may be located in a bit field of a mediator.

[0067] The term "track" refers to a circular unit within a disk that transverses all sectors. A "track sector" is a track within any one sector. A "track cluster" spans more than one sector.

[0068] ***Preferred embodiments***

[0069] The present invention provides methods for storing data on a non-cache recording media, computer program products for carrying out these methods, and systems that are configured to carry out these methods. Through various embodiments of the present invention, one can efficiently store and retrieve data.

[0070] Bit Markers

[0071] According to one embodiment, the present invention is directed to a method for storing data on a recording medium that uses bit markers as representations of strings of bits within raw data. Thus, raw data is translated into a series of markers that represent the raw data, and the method provides for receiving a file or stream that contains data that will be converted into a set of signals, which may be referred to as bit markers, for storage and storing those signals.

[0072] The file may be received from a person or entity that is referred to as a host. Preferably, the host will send the signals in the form of raw data. For example, the host may send one or more chunklets that individually or collectively form one or more files such as a JPEG, PDF, TIFF or WORD document. Various embodiments of the present invention commence upon receipt of chunklets, receipt of subunits of chunklets or receipt of one or more files to be converted into chunklets. Preferably, the chunklets or subunits of chunklets that are received are all of a uniform size. Optionally, the method verifies uniformity of size and if non-uniformity is detected, *e.g.,* one or more chunklets has too few bits, the method causes the addition of zeroes until all chunklets are of a uniform size.

[0073] By way of example, data may be received in chunklets that are N bits long, wherein N is an integer greater than one. For example, N may be from 128 Bytes to 16K, *e.g.,* 4K, which corresponds to 4096B.

[0074] The chunklets are received in an order. For example, a file may contain ten chunklets that are received by the system serially. Alternatively, a plurality of chunklets for a given file could be transmitted in parallel or together if they were to contain information that allows for their being re-associated with one another in a manner that allows for recreation and use of the file by the host's operating system. Thus, in some embodiments, the methods of the present invention store markers in the same order in which the chunklets are received. Accordingly, when a host calls for retrieval of a file, the corresponding retrieval methodologies would call the encoded data back in the same order, and decode it into chunklets in the appropriate order.

[0075] Optionally, prior to encoding, the system may divide the chunklets into groups of bits, also referred to as subunits, each of which is A bits long. If the system divides the chunklets into subunits, the subunits may be compared to a bit marker table. If the system does not divide the chunklets into subunits, then each chunklet may be compared to a bit marker table.

[0076] The bit marker table correlates unique set of bits with unique markers. In some embodiments, the bit marker table contains a marker for each unique string of bits of size A when subunits are used or of size N when subunits are note used. Thus, under this method a computer program may receive a set of chunklets as input. It may then divide each chunklet into Y subunits that are the same size and that are each A bits long, wherein A/8 is an integer. For each unique A, there may be a marker within the table.

[0077] Thus, through an automated protocol, after receipt of the chunklets, a computer program product causes the bit marker table to be accessed. Accordingly, each chunklet or subunit may serve as an input, and each bit marker may serve as an output, thereby forming an output set of markers. The output set of markers may be referred to as translated, coded or encoded data. In embodiments in which each chunklet is not subdivided, then each chunklet would receive one marker. If the chunklet is divided into two subunits, it would be translated or encoded into two markers. Thus, a computer program product uses a bit marker table that correlates markers with input, to assign at least one marker that corresponds to each chunklet. The computer program product may be designed such that a different output is generated that corresponds to each individual marker, a different output is generated that contains a set of markers that corresponds to each chunklet or a different output is generated that contains the set of markers that corresponds to a complete file.

[0078] The bit marker table contains X markers. In some embodiments, X equals either the number of different combinations of bits within a chunklet of length N, if the method does not divide the chunklets into subunits, or the number of different combinations of bits within a subunit of length A, if the method divides the chunklets. If documents types are known or expected to have fewer than all of the combinations of bits for a given length subunit or chunklet, X (the number of markers) can be smaller than the actual number of possible combinations of bits. For example, in some embodiments, all of the bit markers are the same size, and the number of bit markers within the bit marker table is equal to the number of combinations of bits within a string of bits of size N or A. In other embodiments, all of the bit markers are the same size, and the number of bit markers within the bit marker table is less than 90%, less than 80%, less than 70% or less than 60% of the number of combinations of bits within a string of bits of size N or A.

[0079] By way of example, in some embodiments, each chunklet is assigned a code (*i.e.,* a marker) that consists of a plurality of 0s and/or 1s. In other embodiments, each chunklet is divided into a plurality of subunits that are each assigned a code (*i.e.,* a marker) that consists of a plurality of 0s and 1s. The subunits may be defined by a length A, wherein N/A =Y and Y is an integer. If any subunit does not have that number of bits, *e.g.,* one or more subunits have a smaller number of bits than the system is configured to receive as input, the system may add bits, *e.g.,* zeroes, until all subunits are the same size. This step may, for example, be performed after the chunklets are divided into subunits and in the absence of first checking to see if all of the chunklets are the same size. Alternatively, and as described above, it may be performed on the chunklet level prior to dividing the chunklets into subunits.

[0080] As the above-description suggests, the algorithm may be configured to translate strings of bits into a set of coded data, and the algorithm may be designed such that the strings of bits correspond either to the chunklets or to the subunits of the chunklets. Preferably, the set of coded data is smaller than the file as received from the host or client. However, regardless of whether the set of coded data is smaller than the original data, it is capable of being converted back into the chunklets of the file. As persons of ordinary skill in the art will recognize, the data that is received from the host for storage will be raw data, and thus can correspond to any document type.

[0081] The encoding can serve two independent purposes. First, by encoding the data for storage, there is increased security. Only a person or entity that knows the code (*i.e.,* has access to the bit marker table) will be able to decode it and to reconstruct the document. Second, if the code is created using fewer bits than the original document, then less storage space will be needed and there can be a cost savings. If the amount of storage space for one or more files is reduced, then the NCM can store data that corresponds to a greater number of files and/or larger files.

[0082] For at least a plurality of the unique combination of bits within the table, preferably if the system does not divide the chunklets into subunits the marker is smaller than chunklet length N or if the system does divide the chunklets into subunits, smaller than subunit length A. Preferably if the system does not divide the chunklets into subunits, no markers are larger than chunklet length N, or if the system does divide the chunklets into subunits, no markers are larger than subunit length A. In some embodiments, all markers are smaller than N or smaller than A. Additionally, in some embodiments, each marker may be the same size or two or more markers may be different sizes. When there are markers of different sizes, these different sized markers may, for example, be in the table. Alternatively, within the table all markers are the same size, but prior to storage all 0s are removed from one or both ends of the markers.

[0083] After the computer program product translates the chunklets into a plurality of markers, it causes the plurality of markers (with or without having had 0's removed from an end) to be stored on a non-transitory recording medium in an order that corresponds to the order of the chunklets or from which the order of the chunklets may otherwise be recreated. Ultimately, the markers are to be stored in a non-transitory medium that is a non-cache-medium. However, optionally, they may first be sent to a cache medium, *e.g.*, L1 and/or L2. Thus, so that no information is lost, the bit marker table may be stored in persistent memory, but when in use, a copy may be in *e.g.*, L2 cache, and upon booting or rebooting populating of the table in the L2 cache may be from the persistent memory.

[0084] In one embodiment, a storage device stores a plurality of bit markers in a non-cache medium that correspond to a given file. The bit markers are of a size range X to Y, wherein X is less than Y and at least two markers have different sizes. As or after the bit markers are retrieved, a computer algorithm adds 0's to one end of all bit markers that are smaller than a predetermined size of Z, wherein Z is greater or equal to Y. A look up table may be consulted in which each marker of size Z is translated into strings of bits of length A, wherein A is greater than or equal to Z. In a non-limiting example, X= 4, Y = 20, Z= 24, A=32. In some embodiments, A is at least 50% larger than Z. The string of bits that correspond to A may be subunits that are combined into chunklets or they may be chunklets themselves.

[0085] Frequency Converters

[0086] As described above, a bit marker table may assign markers to strings of bits in a random or non-random manner to raw data, and the bit markers may be of a uniform or non-uniform size. However, instead of a bit marker table as described above, one may use a frequency converter.

[0087] As persons of ordinary skill in the art will recognize, Table I and Table II in the examples section of this disclosure assign bit markers (*i.e.,* converted bits) in a manner that is independent of the frequency of the string of bits in the raw data. However, as mentioned above and explained in example 3 below, one could assign smaller markers to raw data that is expected to appear more frequently in a document type or set of documents. This strategy takes advantage of the fact that approximately 80% of all information is contained within approximately the top 20% of the most frequent subunits. In other words, the subunits that correspond to data are highly repetitive.

[0088] To further illustrate how a frequency converter can use strings of bits of different sizes, reference may be made to **figure 5**. **Figure 5** shows a binary tree for all binary sequences that are five units long and begin with the number 1. As the tree shows, there are 16 paths to create sequences of five digits in length, beginning at the top row and moving down the tree to the bottom. However, when moving down the tree, one can move down a branch to for example, the third, fourth or fifth rows. Accordingly, the method may be designed such that for one piece of data, it assigns a code that corresponds to moving partially down the tree, whereas for other pieces of data, the method assigns a code that corresponds to moving along a different branch a greater number of units. By way of a non-limiting example, for particular pieces of converted data, one may stop at the third row, *e.g.,* generating a sequence of 101, whereas for all other pieces of data, the first three digits are not 101. Thus, the methods would never allow use of: 1010, 1011, 10100, 10101, 10110 and 10111 (which are within the box in **figure 5**). Upon retrieval of data from memory, the system would read digits until it recognizes them as unique, and upon recognition of them as unique, use the unique string as input into a program that allows for decoding, which would allow recreation of the raw data file for the host.

[0089] The various methods of the present invention may, during or after retrieval call for reading a minimum number of bits and after that minimum number (which corresponds to the shortest unique code), check for uniqueness by comparing to a data file or table, and if the code is not unique within the data file or table continue to extend the number of bits and check for uniqueness of the extended code after addition of each individual bit until the sequence is recognized as unique. The aforementioned example is described in terms of strings of 3, 4 or 5 bits, but those sizes are used for illustrative purposes only. In some embodiments, the shortest sequences of unique bits for use in connection with a frequency converter are 10 to 15 bits long and the longest sequences of bits are from 25 to 40 bits long.

[0090] In some embodiments, for every plurality of converted strings of bits that are of different sizes there is a first converted string of bits that is A bits long and a second converted string of bits that is B bits long, wherein $A < B$, and the identity of the A bits of the first converted string of bits is not the same as the identity of the first A bits of the second converted string of bits.

[0091] As Tables I, II and III (see examples) show, information may be converted and the output code can be configured to take up less space than the input because markers are used to represent groups of bits. Thus, preferably within a table, at least one, a plurality, at least 50%, at least 60%, at least 70%, at least 80%, at least 90%, or at least 95% of the markers are smaller in size than the subunits. However, there is no technological impediment to having the converted data be the same size or larger than the data received from the host or as generated from a hash function value algorithm.

[0092] Fragmentation

[0093] As noted above, after receipt of the stream of data, in some embodiments the method calls for fragmenting data that is received. If the data that is received is in the form of a large file, it may be fragmented into chunklets. If the data that is received is in the form of chunklets, the chunklets may be fragmented into subunits. A large file may first be fragmented into chunklets and then those chunklets may be fragmented into subunits. Thus, for any input from a host, there may be zero, one or two fragmentation steps.

[0094] By fragmenting an I/O stream, the method divides each I/O stream into smaller units (which also may be referred to as blocks) of predetermined and preferably uniform size. As persons of ordinary skill in the art will recognize, one benefit of using smaller block sizes is that small blocks allow for easier processing of data. By way of a non-limiting example, an I/O stream may be 4096 Bytes long, and the method may fragment that stream into fragmented units that are 1024, 512, 256 or 128 Bytes long. After the data is received or as the data is being received, an algorithm may be executed that divides chunklets into subunits of, for example, 32 bits.

[0095] The size of the subunits is a choice of the designer of the system that receives the data from the host. However, the size of the subunits should be selected such that the chunklets are divided into subunits of a consistent size, and the subunits can easily be used in connection with consultation of a bit marker table or a frequency converter.

[0096] If any of the chunklets are smaller than the others, optionally, upon receipt of that chunklet of the smaller size, the algorithm adds zeroes in order to render the smaller chunklet to be the same size as the other chunklets. Alternatively, the system may divide the chunklets into subunits and upon obtaining a subunit that is smaller than the desired length, add zeroes to an end of that subunit.

[0097] Preprocessing

[0098] During the translation process (which also may be referred to as an encoding process) the string of bits (*i.e.*, the chunklets or subunits) that the algorithm uses as input for the bit marker table or frequency converter may be preprocessed. Each of these strings of bits may be defined by a first end and a second end, and prior to assigning a marker the method further comprises analyzing each string of bits to determine if the bit at the second end has a value of 0. If the bit at the second end has a value of 0, the method may remove the bit at the second end and all subsequent bits that have a value of 0 and form a contiguous string of bits with the bit at the second end, thereby forming a string of bits of reduced size. A benefit of the pre-processing step is that a smaller bit marker table or frequency converter can be used. For example, Table II could be used instead of Table I to produce the same coded data.

As persons of ordinary skill in the art will recognize, this preprocessing can be accomplished by searching and removing 1s instead of zeroes from the second end.

[0099] Truncation

[00100] In another embodiment of the present invention, rather than translate through the use of a bit marker table or a frequency converter, one could store the truncated subunits in the same order that they exist within the chunklets (or if subunits are not used, then the chunklets could be truncated and stored). Thus, instead of having strings that were shortened function as preprocessed data for use as input in *e.g.,* a bit marker table, they themselves may be stored.

[00101] Thus, in some embodiments, there is another method for storing data on a recording medium. According to this method, one receives a plurality of digital binary signals, wherein the digital binary signals are organized in chunklets that are in a format as described above. Optionally, each chunklet may be divided into subunits as provided above.

[00102] Each chunklet or subunit may be defined by its length and each chunklet or subunit has a first end and a second end. One may analyze each chunklet or subunit to determine if the bit at the second end has value 0 and if the bit at the second end has a value 0, remove the bit at the second end and all bits that both have the value 0 and form a contiguous string of bits with that bit at the second end, thereby forming a revised chunklet or a revised subunit for any chunklet or subunit that has a 0 at the second end. Because these revised strings of bits are shorter than the original strings, they may be referred to as truncated.

[00103] After the chunklets or subunits are truncated, one may store the truncated information in a non-transitory recording medium. By storing truncated information, fewer bits are used for storing the same information that otherwise would have been stored in strings of bits that were not truncated.

[00104] In various methods described above one can remove the digit(s) from the first end or the second end of each subunit or of each chunklet, but not both. However, it is within the scope of various embodiments of the present invention to practice methods in which one considers removing digits from the first end of each subunit or chunklet, one separately considers removing digits from the second end of each subunit or chunklet, for each subunit or chunklet one analyzes whether truncation occurs at either, one or both of the first end and the second end, and if it occurs at only one end, saving the truncated chunklet or subunit, and if it occurs at both ends, then saving the smaller of the truncated units. It is also within the scope of the present invention to practice methods in which digits could be removed from both ends of a chunklet or subunit.

[00105] Thus, one may receive a plurality of digital binary signals. The binary signals may be received in units, *e.g.,* chunklets or subunits of chunklets. Each unit may be the same number of bits long, and each unit has a first end and a second end. The number of bits within a unit is an integer number greater than 1, and the bits have an order within the units, and the units have an order.

[00106] One may then analyze each unit in order to determine if the bit at the first end has a value 0 and if the bit at the first end has a value 0, removing the bit at the first end and all bits that both have the value 0 and form a contiguous string of bits with that bit, thereby forming a first revised unit for any unit that has a 0 at the first end.

[00107] One may also analyze each unit to determine if the bit at the second end has value 0 and if the bit at the second end has a value 0, removing the bit at the second end and all bits that both have the value 0 and form a contiguous string of bits with that bit, thereby forming a second revised unit for any unit that has a 0 at the second end.

[00108] For each unit, the following decision tree may be applied: (a) if the sizes of the first revised unit and the second revised unit are the same, storing the first revised unit (or the second revised subunit); (b) if the first revised unit is smaller than the second revised unit, storing the first revised unit; (c) if the second revised unit is smaller than the first revised unit, storing the second revised unit; (d) if there are no revised units, storing the unit; (e) if there is no first revised unit, but there is a second revised unit storing the second revised unit; and (f) if there is no second revised unit, but there is a first revised unit storing the first revised unit.

[00109] One may also store information that indicates if one or more bits were removed from the first end or the second end or one could use a first bit marker table for units for which bits are removed from the first end and a second bit marker table for units for which bit markers are removed from the second end, and between the two bit marker tables, there are no duplications of bit markers. These two different bit marker tables can be organized as sections of the same table and include bit markers for units that are not revised. In the table or tables, there are no duplications of the bit markers for first revised units, second revised units and any units that are not revised because, for example, they have 1s at both ends.

[00110] As persons of ordinary skill in the art will recognize, although the method described above is described in connection with removing zeroes, the system could instead remove ones. Additionally, these methods are described as being used as an alternative to the use of a bit marker table or frequency converter. However, it is within the scope of the present invention to use these methods in conjunction with a bit marker table or frequency converter, *e.g.*, by truncating the markers that are the outputs of those protocols, so long as they are stored in a manner that permits retrieval and reconstitution of the original file.

[00111] **Hash Values**

[00112] In certain of the embodiments described above, one accesses a bit market table or a frequency converter to encode data. In another embodiment, the chunklets or subunits serve as input for a cryptographic hash function value algorithm.

[00113] Thus, for each input, there is an output that is smaller in size than the input. For example, if there are subunits of X Bytes, which also may be referred to as fragmented unit of X Bytes, the output is a generated hash function value for each fragmented unit of X Bytes. Examples of hash function value algorithms include, but are not limited to, MD5 hash (also referred to as a message digest algorithm), MD4 hash and SHA-1. The value that is output from a hash function value algorithm may be referred to as a checksum or sum. In some embodiments, the sum is 64, 128, 160 or 256 bits in size. Because of the highly repetitive nature of data within I/O streams, the probability of generating conflicting sums, *i.e.* sums that are the same but correspond to different fragmented units, is relatively low.

[00114] In some embodiments, each hash value is a string of bits or a code that consists of a plurality of 0s and/or 1s, preferably of the same length. These hash values may be used for storage on a non-cache recording medium.

[00115] In one embodiment, a system may, for example, execute a method for storing data on a non-cache recording medium comprising: (i) receiving an I/O stream of N Bytes; (ii) fragmenting the N Bytes into fragmented units of X Bytes; (iii) associating a cryptographic hash function value with each fragmented unit of X Bytes, wherein the associating comprises accessing a correlation file, wherein the correlation file correlates a stored hash function value of Y bits with each of a plurality of stored sequences of X Bytes and (a) if the sequence of fragmented X Bytes is in the correlation file, using the stored hash function value of Y bits for storage on a non-cache recording medium; and (b) if the sequence of fragmented X Bytes is not in the correlation file, then generating and storing a new hash function value of Y bits with the fragmented sequence of X Bytes in the correlation file and using the new hash function value for storage on the non-cache recording medium.

[00116] The above described methods call for fragmenting the I/O stream of N Bytes. However, the methods could be applied by applying a cryptographic hash function value algorithm to the I/O stream and not fragmenting it. Thus, an alternative method comprises: (i) receiving an I/O stream of N Bytes; (ii) associating a cryptographic hash function value with each unit of N Bytes, wherein said associating comprises accessing a correlation file, wherein the correlation file correlates a stored hash function value of Y bits with each of a plurality of stored sequences of N Bytes and (a) if the sequence of N Bytes is in the correlation file using the stored hash function value of Y bits for storage on a non-cache recording medium; and (b) if the sequence of N Bytes is not in the correlation file, then generating and storing a new hash function value of Y bits with the N Bytes in the correlation file and using the new hash function value for storage on the non-cache recording medium. Additionally, although various methods are for illustrative purposes described as being applied to a set of N Bytes or X Bytes, a person of ordinary skill in the art will appreciate that the methods may be and preferably applied for all N Bytes or X Bytes of a file.

[00117] Additionally, the above described method applies the algorithm only if the string of Bytes is not already within the correlation file. In another embodiment, the present invention provides a method for storing data on a non-cache recording medium comprising: (i) receiving an I/O stream of N Bytes; (ii) applying a cryptographic hash function value algorithm to each unit of N Bytes to form a generated hash function value for each unit of N Bytes; (iii) accessing a correlation file, wherein the correlation file correlates a stored hash function value of Y bits with each of a plurality of stored sequences of N Bytes and (a) if the generated hash function value for the unit of N Bytes is in the correlation file, using the stored hash function value of Y bits for storage on a non-cache recording medium; and (b) if the generated hash function value for the unit of N Bytes is not in the correlation file, then storing the generated hash function value of Y bits with the fragmented unit of N Bytes in the correlation file and using the generated hash function value for storage on the non-cache recording medium. This embodiment may also be modified to add steps of fragments the N Bytes to form fragmented units and applying the cryptographic hash function value algorithm to the fragmented units instead of to the units of N Bytes.

[00118] In certain of the embodiments described above, after a checksum is obtained for each chunklet, fragmented unit, or subunit, one accesses a correlation file. These methods may obtain checksums according to a first in first out ("FIFO") protocol and either begin accessing the correlation file while the I/O streams are being received, while checksums are being generated, or after all I/O streams have been received, fragmented and subjected to the hash function value algorithm.

[00119] As noted above, the correlation file associates a different stored hash function value of Y bits with each of a plurality of stored sequences of X Bytes. Accordingly, in these cases in which there is a generated hash function value: (a) if the generated hash function value for a unit of X Bytes is in the correlation file, the method causes the stored hash function value of Y bits to be used for storage on a non-cache recording medium; and (b) if the generated hash function value for the unit of X Bytes is not in the correlation file, the method causes the generated hash function value of Y bits to be stored with the sequence of X Bytes in the correlation file and uses the generated hash function value for storage on the non-cache recording medium.

[00120] Conflict Resolution Modules

[00121] The probability of conflicting hash values being generated as a result of application of the hash value algorithm is low. Thus, in some embodiments, one may choose to employ methods described herein without the use of a conflict resolution module. However, in various embodiments, the present invention applies a conflict resolution module in order to address circumstances in which a conflict might arise.

[00122] As shown in **figure 6**, a system may receive a stream of sets of N Bytes **610**. It may then cause fragmentation of the N Bytes into units of a desirable size **620** in order to apply a hash function and to generate a hash function value **630**, for each fragmented unit. Next the system queries whether each hash value is already within the correlation file **640**.

[00123] In these embodiments, there is a conflict resolution module that is activated whenever for a fragmented unit of X Bytes, a hash value is generated that is the same as a hash value in a correlation file **650**. This module then queries whether the fragmented unit of X Bytes is the same as the already stored value of X Bytes for that hash value. If the conflict resolution module determines that for a fragmented unit of X Bytes, a hash function value is generated that is the same as a stored hash function value in the correlation file, but the fragmented unit of X Bytes is different from the X Bytes associated with the stored hash function value, then it is necessary to update the correlation file and store the generated hash value therein, as well as writing to the NCM **660**. For example, in some embodiments as described below, the module causes there to be different Z bits associated with the stored hash function value and the generated hash function value.

[00124] If the generated hash value (with any associated Z value if present) is unique, then it may be used for storage on the NCM and the correlation file may be updated to include the association **680**. Further, if the hash value is already in the correlation file and the received N Bytes are the same as those within the correlation file, then the stored hash value (with any associated Z value if present) may be written to the NCM and the system need not update the correlation file **670**. Thus, the conflict module may be accessed as a check after the correlation file is accessed, and only if the newly generated hash value is already within the correlation file. The conflict

module would then determine if there is a conflict or if both the checksum and the fragmented units from the received file are already associated with each other in the correlation file.

[00125] The aforementioned embodiment describes a method in which each fragment's unit of bits is subjected to a cryptographic hash function value algorithm prior to accessing the correlation file. However, alternatively one could check the correlation file first for the presence of the subunit or chunklet, and apply a cryptographic hash function value algorithm only if the fragmented units are not already in the table. Then, if the chunklet or subunit is already in the correlation file, use the stored checksum value. If the subunit or chunklet is not in the file, then one would apply the hash value algorithm and enter the protocol for checking for conflicts. If there is a true conflict, *i.e.,* the same hash value being associated with different subunits or chunklets, then one would update the correlation file to address this issue.

[00126] One means by which to address the issue of a true conflict is for all hash function values for which no conflict exists, Z bits are associated and the Z bits are a uniform length of *e.g.,* 8 to 16 zeroes. By way of a non-limiting example N=4096 bytes (chunklet size), X =512 bytes (subunit size), Y=128 or 256 bits (hash value size), and Z=8 or 16 bits (hash value extension size). The method may, for example, associate 8 zeroes at the end of a checksum when the checksum does not conflict with a previously stored checksum. Upon identification of a conflict, (*e.g.,* different fragmented units being associated with the same checksum,) the newest checksum may be assigned a different Z value. Thus, if the Z value as stored in the correlation file is 00000000, the Z value for the first conflicting checksum may be 00000001 and should there be another conflicting checksum 00000010. If there were additional conflicting checksums, each conflicting checksum may be assigned the next Z value as the conflicting checksum is identified.

[00127] Upon obtaining checksums for each fragmented unit, and following application of the conflict module if present, a plurality of hash function values (and in some embodiments with Z values) may be written to a non-cache recording medium for storage.

[00128] Combined Use of Hash Values and Bit Markers or Frequency Converters

[00129] In some embodiments, instead of storing the hash value, the method further comprises converting (also referred to as encoding) each hash function value for storage through use of a bit marker table to generate a bit marker for each hash function value for storage and storing each bit marker for the respective hash function value (and in some embodiments with Z values) in the non-cache recording medium. By using a bit marker table, one may convert or code the hash values. Thus, the hash values may serve as input subunits when accessing the bit marker table, and the associated markers may serve as output.

[00130] The subunits may be defined by a length A, wherein the length of the hash value divided by A is an integer. If any string of bits does not have that number of bits, *e.g.,* one or more string of bits have a smaller number of bits than the number within the subunits that are configured to be converted, the system may add bits, *e.g.,* zeroes, until all subunits are the same size. Alternatively, the addition of 0s may be performed on the hash values level prior to dividing the hash values into subunits. Furthermore, if the hash values (and in some embodiments as combined with the Z values) are smaller than the subunit size, they may be combined to form subunits or combined and be fragmented to form subunits of the appropriate size.

[00131] The conversion of hash values into coded data may be carried out by an algorithm or computer program. Execution of the algorithm or computer program may, for example, be controlled by a manager that also controls the application of the hash function value algorithm. The coded data, which also may be referred to as converted data, is also comprised of binary signals, and it is coded and stored in a manner that permits it to be converted back into the hash values of the file. Thus, information is retained during the encoding process that permits decoding without a loss of information.

[00132] The encoding can serve two independent purposes. First, by converting the data for storage through the bit marker table, there is increased security. Only a person or entity that knows the code will be able to decode it and to reconstruct the document. Second, if the code is created using fewer bits than the hash values that correspond to the document, then both the use of converted data and the hash values will cause less storage space to be needed and there can be further cost savings.

[00133] In some embodiments, rather than using a bit marker table, the method further comprises encoding each hash function value for storage through use of a frequency converter. In these methods, for each hash function value for storage, a converted string of bits is generated, wherein for hash values of the same size, the converted strings of bits that are output are of different lengths. The frequency converter may associate each of a plurality of converted string of bits with each of a plurality of hash function values, and the plurality of converted string of bits in the frequency converter are sufficiently distinct that when read back the converted string of bits can be correlated with the appropriate hash values.

[00134] By combining the use of hash value algorithms with either: (1) bit marker table applications; or (2) frequency converter applications, one is able to introduce increased security and/or cost-savings. Additionally, as described above the hash values are input for the bit marker table or frequency converter. However, the order could be reversed and the chunklets or subunits could enter a protocol that uses a bit marker table or frequency converter to generate markers, and those markers could serve as input for a hash function algorithm according to any of the methods described herein, including but not limited to methods that use conflict resolution modules.

[00135] Mediators and Managers

[00136] Any or each of a bit marker table, frequency converter, hash value correlation file and programs for accessing and using the aforementioned files and protocols, may be stored within a mediator, a manager or elsewhere. Whenever any one or more of the aforementioned protocols or files is used in connection with an embodiment of the present invention and not stored within a mediator or manager, preferably, it is operably coupled to a mediator, a manager or both. Methods and systems for communicating with files and programs that are stored locally or remotely are well known to persons of ordinary skill in the art.

[00137] In various embodiments, a manager controls the methods of the present invention. For example, after an I/O (output from the host, input to the system) is received from a host, a manager may cause acknowledging receipt of the I/O stream to the host.

[00138] The host may record the file of the I/O stream as being stored at a first storage address. However, the converted string of bits may in fact be stored in a non-cache recording medium at a second storage address, wherein the first storage address is not the same as the second storage address. Further, the first storage address may show (or be associated with a denotation of) the file as having a first file size, while the second storage address may correspond to a second file size, wherein the first file size is at least two times as large or at least four times as large as the second file size.

[00139] In some embodiments, the second storage address is stored on a mediator. Within the mediator, the second storage address may be stored in a bit field. The mediator described herein may be used independent of or in connection with the methods described for translating data through one or more of a bit marker table, frequency converter and hash value algorithm.

[00140] In some embodiments, the mediator is hardware, software or a combination thereof that comprises: (i) a first set of tracks, wherein the first set of tracks comprises or contains information that corresponds to file system information, bootability information and partition information; (ii) a second set of tracks, wherein the second set of tracks comprises or contains information that corresponds to a copy of the first set of tracks; (iii) a third set of tracks, wherein the third set of tracks comprises or contains information that corresponds to metadata other than file system information, bootability information and partition information; and (iv) a fourth set of tracks, wherein the fourth set of tracks comprises or contains information that corresponds to the bit field. By way of example, the mediator may reside in L2 cache and any one or more of the bit marker tables, frequency converters or hash value tables may reside in RAM of a server when in use and also exist in persistent storage of a solid state device that may be the same as or different from the NCM on which markers or hash values are stored. If updated, the updates are made to the table in RAM and also to the persistent memory.

[00141] In addition to using a mediator, various embodiments of the present invention call for the use of a manager. The manager, which may comprise one or more modules and reside on a local computer, on a network or in a cloud. The manager is configured to coordinate receipt of or to receive certain information itself and to transfer this information to a mediator, or to control receipt of the information directly by the mediator. Thus, the methods can be designed such that information from the initiator flows through the manager to mediator or that the manager only directs the flow of information to other components of the system, *e.g.,* to the host or NCM.

[00142] In various embodiments of the present invention, the manager may apply or cause to be applied, the hash function algorithm or other protocol, any conflict module if present and any conversion module if present and cause the flow of data directly to the mediator. The manager also may control storage of information through use of the mediator and retrieval and transmission of information. When storing the information, an LBA number may be identified and the data to be stored may be sent to a buffer in order to avoid or to reduce bottlenecking. Further, L1 and/or L2 cache may be used during storage.

[00143] Similarly, when retrieving data, the method may call the data from the NCM, fill a buffer, obtain the checksum values, and then recreate the fragmented units and reassemble, or if not fragmented, then directly reassemble to form information in a form that a host may receive and review. If the stored data is converted data, prior to obtaining the checksum values, the data may be decoded. These steps may be coordinated and control through the manager, which executes the requisite protocols.

[00144] In some embodiments, a manager may control, communicate with and coordinate the activities of one or a plurality of mediators. For each mediator, the manager receives (or coordinates receipt of) a set of parameters. These parameters may comprise, consist essentially of or consist of one, two or all three of file system information, bootability information and partitioning information. The manager causes this information to be stored in a first set of tracks on the mediator, which may be referred to as reserve 1 or R_1. The file system will dictate how the reserve blocks are to be used. For example, when using NTFS, sectors 1-2 may be for a MBR (master boot record) and sector 3 may be for \$MFT. Optionally, these tracks may be copied into a second set of tracks, which may be referred to as reserve 2 or R_2.

[00145] The manager may also receive metadata in addition to the parameters described in the preceding paragraph. The metadata is stored in a third set of tracks on the mediator. At the time that the manager receives the parameters and metadata, or at a later time, it may also receive one or more files for storage on a non-cache medium. Each file is received with a file name. The file name is generated by a host that transmits the file and may be defined by the host's file system. The manager, which may, for example, be or be a part of a SAN or NAS or combination thereof, upon receipt of the file with a file name, can automatically execute the steps described herein for storage.

[00146] The systems of the present invention may be designed such that the hash function value algorithm and algorithm for conversion are either stored within the mediator, or the manager, or within other hardware and/or software that are operably coupled to the mediator or manager. Either or both of the algorithms may also cause the file name to be stored in a mediator. There are no limitations as to where the mediator is physically located. However, preferably, it is configured to communicate with a host or a computer that is capable of communicating with a host that preferably is located remote from the mediator. The mediator is also configured to communicate, directly or indirectly (*e.g.*, through the manager), with a recording medium, *e.g.*, a non-cache medium where the coded set of data is stored, which optionally is remote from the mediator, any manager and the host. As noted above, the mediator permits a user who identifies the file name to retrieve the set of coded data from the non-cache storage medium.

[00147] Cache

[00148] As noted above, in some embodiments, upon receipt of the raw data, the methods of the present invention may cause a confirmation of receipt to be automatically returned to the host. In one QoS (quality of service) protocol, a data file is received through an I/O and immediately sent to L1 cache. Upon receipt, an acknowledgement is sent from L1 cache back through the I/O. From L1 cache, the data file may be sent to L2 cache, which transmits an acknowledgement back to L1 cache. The L2 cache may also send the data file to a non-cache medium (NCM) for long term storage. The NCM may in turn send an acknowledgement back to L2 cache.

[00149] In some embodiments, the mediator may reside in or be operably coupled to a heap (dynamically allocated memory) within L1 cache. Alternatively, the mediator may reside within a card, or be part of or be operably coupled to L2 cache.

[00150] As one of ordinary skill in the art knows, the decision to place the mediator in L1 versus L2 will be impacted by factors such as the frequency of use of the stored data. Thus, L1 cache is used to store data that is used frequently by the system or an end user, while L2 caches may be used for data that is accessed somewhat frequently.

[00151] In another QoS protocol, through the I/O, a data file is received by L1 cache. The data file is transferred to both L2 cache and the NCM from L1 cache. Each of L2 cache and the NCM send acknowledgments to L1 cache. Either before or after receiving acknowledgments from one or both of L2 cache and the NCM, L1 cache sends an acknowledgement through the I/O.

[00152] File Correlation

[00153] In various embodiments of the present invention, the host will understand each file to be stored at a first storage address. The first storage address may be stored by the host in a sector map and correspond to a LUN. It may also include the start and, either implicitly or explicitly, the end of the units, sectors or blocks that correspond to a file. The first storage address will correspond to where the host believes that the file is located within a storage device or storage area network. The host will use this first address to keep track of its stored documents or files and to retrieve them. The first storage address is a virtual address *i.e.*, it does not correspond to where the data is actually stored.

[00154] As persons of ordinary skill in the art will recognize, methods and systems may be used in which the host generates the first storage address and sends it along to the systems of the present invention with SCSI commands and optionally associated sector or LBA number(s). The mediator may correlate the file name, what the host thinks of as the location of the file and the storage size of the file as received from the host, *i.e.,* the raw data and any header or footer data, with a second storage address, which is the actual storage address of the data, which may contain information from which the file can be recreated, but that has been converted through for example one or more of a hash value algorithm, bit marker table and frequency converter. Alternatively, the mediator may store only the file name, and optionally, it may not receive the first storage address for a file. As noted above, because storage addresses are based on a linear organization of data, they may implicitly contain the size of the stored information by reciting only the first user perceived LBA or explicitly reciting all locations.

[00155] Although the paragraphs above describe that the host will provide what it believes to be the first storage address, the information could be generated by another entity that either is a conduit through which the host communicates directly or indirectly with the mediator, a module within the host or operably coupled to the host, or a module within or operably coupled to the mediator and/or manager. As persons of ordinary skill in the art will recognize, the stored information that identifies a location of a data file on a storage device may be referred to as a pointer. For a file, the pointer may direct retrieval of data from different locations (*e.g.,* blocks) than the locations on the NCM at which the user believes the data to be located.

[00156] Optionally, one may associate the file with a file type. The file type will direct the recipient of the data to know which operating system should be used to open it. In some embodiments, the association with a file type is done at the initiator or client or host.

[00157] Reserve tracks

[00158] As noted above, the mediator may comprise a first reserve set of tracks (R_1) and a second reserve set of tracks (R_2). In some embodiments, the second reserve set of tracks (R_2) is a copy of the first reserve set of tracks (R_1). Additionally, in some embodiments, one may use the second reserve set of tracks (R_2) to check for errors in the first reserve set of tracks (R_1).

[00159] R_1 may be configured to function as the central point for host initiation. Thus, the host may select the parameters to send to R_1. The mediator may receive this information directly from the host or indirectly through the manager. R_2 is preferably never exposed to the host. Thus, only the mediator itself or the manager can cause information to be stored in R_2. Each of R_1 and R_2 may, for example, contain sixteen sectors and be filled with real data such as host modifiers. By convention, numbering may start at 0. Thus, R_1 may, for example, contain sectors (or tracks) 0 -15, and R_2 may contain sectors (or tracks) 16 -31. However, the mediator may be constructed so as to allow for expansion of each of R_1 and R_2 beyond the initial size of 16 tracks.

[00160] In some embodiments, R_1 contains unique reserve sector information and partition information. Within the partition information, one may store the file system information.

[00161] By way of a non-limiting example and as persons of ordinary skill in the art are aware, when formatting a volume with an NFTS file system, one creates metadata files such as $MFT (Master File Table), $Bitmap, $Log File and others. This metadata contains information about all of the files and folders on an NFTS volume. The first information on an NTFS volume may be a Partition Boot Sector ($Boot metadata file), and be located at sector 0. This file may describe the basic NTFS volume information and a location of the main metadata file $MFT.

[00162] The formatting program allocates the first sixteen sectors for the $Boot metadata file. The first sector is a boot sector with a bootstrap code, and the following fifteen sectors are the boot sector's IPL (initial program loader).

[00163] In addition to the tracks of R_1 and R_2, the mediator may store additional metadata. This metadata may, for example, correspond to information that allows the execution of thin provisioning strategies, which correspond to visualization technology that allows a device to give the appearance of having more physical resources than are actually available, and it may, for example, be contained in the eight tracks after R_2, which would be tracks 32-39. The metadata may also provide for features such as LUN QoS, VM and WORM.

[00164] In some embodiments the system may contain a SAN indexer. The SAN indexer may check what is in R_1 and R_2, and extract that information. This information can be put into a database that may readily be searched by, for example, text searching.

[00165] **Bit Field**

[00166] The mediator may also comprise or contain information that corresponds to a bit field. The bit field contains the information that indicates where the data is physically stored within a storage medium, and if the metadata is located in tracks 32-39, the sector number of the bit field may begin at track 40. It is within the bit field of the mediator that correlation between the file name of the host and the location of the data is stored. Thus, it may comprise, consist essentially of or consist of a sector map. This information from the bit field component of the mediator may be used to determine the actual space saving on any device. For example, the percentage of space saved = 1- [(space actually used)/(space as mapped by host)]. In some embodiments, the space saved by converting data according to the methods of the present invention is at least 50%, at least 60%, at least 70% or at least 80%. These savings may be per file or averaged over all files on a device.

[00167] As a matter of practice, preferably the mediator is not located on the disk or recording medium on which the file data is stored. Additionally, preferably the mediator requires only about 0.1-0.2% of the total memory of the corresponding disk or recording medium.

[00168] In addition to providing economic value from saving of space, various embodiments of the present invention open the door for increased efficiencies when looking to protect the integrity of data. Accordingly, various embodiments of the present invention provide new and non-obvious technologies for backing-up data.

[00169] Because in many embodiments, the stored file will be smaller than the raw data file as received from the host, less storage space is needed for it. Thus, the data needed to recreate the file can be stored in a smaller location than the host perceives is being used and than the first storage address suggests. The actual location in which the file is stored, may be referred to as a second storage address. Thus, for each file there will be a first storage address, which is where the host believes that the file is stored, and a second storage address, which is where the coded file is actually stored.

[00170] It is possible that for a given file, which may correspond to one or more blocks, a first storage address and a second storage address are located at the same block within a storage device or one or more overlapping set of blocks. However, preferably for at least one, at least 50%, at least 60%, at least 70%, at least

80%, at least 90% or 100% of the files there is no overlap of blocks within the first storage address and the second storage address. Additionally even if the host perceives the same storage address as the mediator perceives, when data is coded the host cannot recreate the file without first decoding the data. In some embodiments, the host is unaware of the code, and thus is not capable of decoding the stored data.

[00171] In some embodiments, the mediator may receive the chunklet(s) that correspond to a file and temporarily store them in a L1 or a L2 cache. If the L2 cache is present, the L2 cache may acknowledge receipt to the host, and optionally provide or confirm the first storage address to the host. As persons of ordinary skill in the art will recognize, the acknowledgement of receipt and transmission of the first storage address may be done prior to storage at the second storage address and if encoding is performed, then prior to or after the encoding. Regardless of when the acknowledgement is sent, correlation of the actual storage address and virtual storage address of a file is preferably tracked within the bit field.

[00172] Backing-up of data

[00173] In certain embodiments, one may use two mediators to facilitate backing-up data. For example, in a first mediator one may correlate a data file that is stored in a first sector or first sector cluster on a first recording medium with a file name. As described above, the first mediator is configured to permit a user or entity that identifies the file name to retrieve the data file from the recording medium.

[00174] A data protection protocol may be executed that generates a second mediator. The second mediator will be an exact copy of the first mediator at a time T1. Thus, at T1, both the first mediator and the second mediator will point to the same sectors or sector clusters (and locations therein) on the first recording medium.

[00175] After time T1, for example at T2, the host may seek to update a file that is stored in a given location on a given sector or sector cluster. The host will not change the data stored at the first storage address. Rather than causing the information on the sector or sector cluster to be written over, the first mediator may cause the new information to be written to a third storage address that corresponds to a location in a different sector or sector cluster and correlate the file name and the first storage address with this new storage address.

[00176] Thus, the first mediator will point to a new sector or sector cluster even though the host believes that the information is being overwritten at a particular storage address. Accordingly, the host will not need to update its address for the sector cluster.

[00177] The second mediator will not be updated, and it will continue to correlate the file name with the first location at which the file was stored. This use of the two mediators will permit one to provide a snapshot of the data as it existed at T1, without causing the host to need to update its file system to indicate that the file as it existed both at T1 and at T2 are being stored. Thus, the snapshot locks all data files that are stored at time T1 and prevents anyone from deleting or writing over those physical files. However, if the host wishes to revise those files, it can work under the impression that it is doing so, when in fact new files are stored. This method is described in connection with sectors and sector clusters. However, it will also work with non-cache media that are not arranged in sectors or sector clusters. For example, they may be organized by LBAs in LUNs.

[00178] As suggested above, this method may be implemented by a system that comprises a first mediator, a second mediator and a non-cache storage medium. Each of the first mediator, the second mediator and the recording medium may be stored on or be formed from separate devices that comprise, consist essentially of or consist of non-transitory media. The afore-described system recites the use of different sectors of the same recording medium but could also be used by writing to different sectors of different recording media. Additionally, within the system the mediators and the recording media are operably coupled to one another and optionally to one or more computers or CPUs that store instructions to cause them to carry out their intended functions and to communicate through one or more portals over a network to one or more hosts. Still further, although this embodiment is described in connection with the use of two mediators, one could implement the system using two sections of the same mediator rather than two separate mediators.

[00179] The system may further be configured with a locking module. The locking module may prevent revision, overwriting or deletion at one or more blocks that have been written as of a certain time. The locking module may also be designed to allow for the writing of new blocks and revision of those new blocks that have not been locked. Thus, the locking module may be configured to permit a host, a user or a system administrator to select certain blocks that have been written as of a certain time or to select all blocks that have been written as of a certain time not to be overwritten.

[00180] Furthermore, there may be a selection module that by default sends all requests for retrieval of files and revision, overwriting or deletion through the first mediator. The selection module may also be configured to allow recovery of what a host may believe are older versions of one or more files as of the times at which the locking technology was applied. Optionally, access to the entire selection module may be restricted to persons who have authorization, *e.g.*, a system administrator.

[00181] The aforementioned system for backing-up data is described in the context of two mediators. However, more than two mediators could be used to capture a history of stored files or versions of files. For example, at least three, at least four, at least five, at least ten mediators, *etc.*, may be used. Additionally, hosts may have mediators take snapshots at regular intervals, *e.g.*, weekly, monthly, quarterly or yearly, or irregular intervals, *e.g.*, on-demand.

[00182] According to another method for backing up data, a clone of the non-cache media may be made. In this method, in a first mediator, one correlates a plurality of file names with a plurality of locations of data that are stored on a non-cache storage medium. The first mediator is configured to permit a user who identifies a specific file name to retrieve a data file from the first non-cache storage medium that corresponds to the specific file name. Part or the entire specific file may be stored in a first sector or sector cluster.

[00183] One may make a copy of the plurality of data files (or all data files of a first non-cache storage medium) to a second non-cache storage medium and a second mediator. The second mediator is a copy of the first mediator at time T1 and is operably coupled to the second non-cache storage medium. At time T2, which is after T1, the system may save revisions to a data file that is stored in said first sector or sector cluster on the first non-cache storage medium. However, no changes would be made to the corresponding location on the second non-cache medium. As a user requests a file after T2, he or she would go through the first mediator and retrieve the most recent stored version of the file. However, the system administrator would have access to an earlier version, which would be stored on the second non-cache medium and could retrieve it by going through the second mediator.

[00184] This method may be implemented by a system that comprises a first mediator, a second mediator, a first non-cache storage medium and a second non-cache storage medium. Each of the first mediator, the second mediator and the first and second recording media for storing data files may be stored on separate devices that comprise, consist essentially of or consist of non-transitory media. Additionally, the first mediator correlates a file name that is derived from a host with a first LUN of the first recording medium and the second mediator correlates the same file name with a second LUN on the second recording medium. In some embodiments, the most recent file, which is stored in the first non-cache medium, has the same LUN that the legacy file has within the second non-cache medium.

[00185] Data retrieval

[00186] According to any of the methods of the present invention, any data that is stored in a converted form is capable of being retrieved and decoded before returning it to a host. Through the use of one or more algorithms that permit the retrieval of the converted data, the accessing of the reference table or frequency converter described above and the conversion back into a uniform string of bits and chunklets, files can be recreated for hosts. By way of a non-limiting example, the data may have been converted and stored in a format that contains an indication where one marker ends *e.g.,* use of unique strings of bits or through the use of uniform sizes of markers.

[00187] Retrieval of the data as stored may be through processes and technologies that are now known or that come to be known and that a person of ordinary skill in the art would appreciate as being of use in connection with the present invention. Optionally, a manager coordinates storage and retrieval of files. In some embodiments, data from the NCM such as markers are retrieved through parallel processing.

[00188] In one method, one begins by accessing a recording medium. The recording medium stores a plurality of markers in an order, and from these markers, one can recreate a file. Access may be initiated by host requesting retrieval of a file and transmitting the request to a storage area network or by the administrator of the storage area network.

[00189] After the data is retrieved from a recording medium from locations identified by a mediator if present, if the data has been converted, then one translates the plurality of markers (or data that has been converted through the frequency converter) into bits that may be used to form chunklets or hash values to be converted back into the I/O stream or the fragmented units of the I/O stream. The markers may be stored such that each marker corresponds to a chunklet or each marker corresponds to a subunit and a plurality of subunits may be combined to form a chunklet. In the stored format, the markers are arranged in an order that permits recreation of bits within chunklets (or hash values) and recreation of the order of chunklets (or hash values), in a manner that allows for recreation of the desired document or file.

[00190] If the data is converted, then in order to translate the markers into chunklets, one may access a bit marker table or a frequency converter. Within the bit marker table or frequency converter, there may be a unique marker that is associated with each unique string of bits or within each unique string of bits within the file. If the table is organized in a format similar to Table II, after translation, zeroes may be added in order to have each subunit and chunklet be the same size. When decoding, one uses the bit maker table or frequency converter in the opposite way that one would use this for coding. Optionally, instead of using the same table and reversing the input and output, one could use separate tables.

[00191] As with other embodiments, each chunklet may be N bits long, wherein N is an integer number greater than 1 and each subunit may be A bits long, wherein A is an integer. In order to translate the markers into chunklets, one may access a bit marker table or a frequency converter.

[00192] After the chunklets are formed, one will have an output that corresponds to binary data from which a document can be reconstituted. Optionally, one may associate the file with a file type. For example the host may keep track of the MIME translator and re-associate it with the file upon return. The file type will direct the recipient of the data to know which operating system should be used to open it. As a person of ordinary skill in the art will recognize, the storage area network needs not keep track of the file type, and in some embodiments does not.

[00193] After the chunklets are formed, in some embodiments, one will have an output that corresponds to binary data from which a file can be reconstituted. In other embodiments, the converted data corresponds to hash values, and the hash values must first be used to create fragmented units of the I/O stream or the I/O stream itself.

[00194] With respect to truncated data, even in the absence of needing to avail oneself of the bit marker table or a frequency converter, one may retrieve the data. One may do so by accessing a recording medium, wherein the recording medium stores a plurality of data units in a plurality of location, wherein each data unit contains a plurality of bits and the maximum size of the data unit is a first number of bits, at least one data unit contains a second number of bits, wherein the second number of bits is smaller than the first number of bits.

[00195] Next one may retrieve the data units and add one or more bits *e.g.,* 0s at an end of any data unit that is fewer than N bits long to generate a set of chunklets that corresponds to the data units, wherein each chunklet contains the same number of bits; and generate an output that comprises the set of chunklets in an order that corresponds to the order of the data units. If the truncated data were formed by removing zeroes, then when retrieving the data, one would add the zeroes back. Additionally, if the stored data units were subunits of chunklets, the system may first add back zeroes to truncated subunits in order to generate subunits of a uniform size and then combine the subunits to form the chunklets.

[00196] When a look-up table is used, preferably it is stored in the memory of a computing device. In some embodiments, the look-up table is static and the markers are pre-determined. Thus, when storing a plurality of documents of one or more different document types over time, the same table may be used. Optionally, it could be stored at the location of a host or as part of a storage area network.

[00197] **Receipt of Data from Host**

[00198] In order to illustrate the various embodiments further and to provide context, reference is made below to specific hardware that one may use, which may be combined to form a system to implement the methods of the present invention. The hardware may be configured to allow for the above described process to be automated and controlled by one or more processors upon receipt of one or more data files from a host.

[00199] In some embodiments, a host may generate documents and computer files in any manner at a first location. The documents will be generated by the host's operating system and organized for storage by the host's file system. The host's operating system may locally store in its memory, the file name. The present invention is not limited by the type of operating system or file system that a host uses. By way of a non-limiting example, the host may comprise a computer or set of computers within a network having one or more of the following hardware components: memory, storage, an input device, an output device, a graphic user interface, one or more communication portals and a central processing unit.

[00200] At that first location a SAP executes a protocol for storing the data that correlates to documents or files. The SAP formats the data into I/O streams or chunklets that are for example, 4K in size.

[00201] The data may be sent over a SAN to a computer or network that has one or more modules or to a computer or set of computers that are configured to receive the data. This receiving computer may comprise one or more of the following hardware components: memory, storage, an input device, an output device, a graphic user interface, a central processing unit and one or more communication portals that are configured to permit the communication of information with one or more hosts and one or more storage devices locally and/or over a network.

[00202] Additionally, there may be a computer program product that stores an executable computer code on hardware, software or a combination of hardware and software. The computer program product may be divided into or able to communicate with one or more modules that are configured to carry out the methods of the present invention and may be stored in one or more non-transitory media.

[00203] For example there may be a level 1 (L1) cache and a level 2 cache (L2). As persons of ordinary skill in the art are aware, the use of cache technology has traditionally allowed for one to increase efficiency in storing data. In the present invention, by way of an example, the data may be sent over a SAN to a cache and the data may be sent to the cache prior to accessing a hash function algorithm, prior to consulting a bit marker table, prior to consulting a frequency converter, and prior to truncating bits, and/or after consulting a hash function algorithm, after consulting a bit marker table, after consulting a frequency converter, and after truncating bits.

[00204] .Assuming that the sector size is 512B, for each chunklet that is 4K in size, the host will expect 8 sectors of storage to be used.

[00205] Systems

[00206] Various embodiments of the present invention provide data storage and retrieval systems. In one embodiment, the system comprises a non-cache data storage medium, a mediator and a manager. Communication among these elements and optionally with the initiator may be over a network that is wired, wireless or a combination thereof.

[00207] The non-cache data storage medium may, for example, comprise, consist essentially of or consist of one or more disks or solid state drives. When in use, the non-cache data storage medium may store files with a space savings of at least 50%, at least 60%, at least 70% or at least 80% when compared to files that have not been processed according to one or more of the methods of the present invention.

[00208] The mediator may comprise, consist essentially of or consist of four sets of tracks: a first set of tracks, a second set of tracks, a third set of tracks and a fourth set of tracks. The mediator is preferably stored on a non-transitory medium and is located remotely from the non-cache data storage medium. Thus, the mediator and the non-cache data storage medium are preferably not part of the same device.

[00209] The system may also contain a manager. The manager may provide the control of the receipt, processing storage and retrieval and transmission of data through the mediator. Thus, preferably, it is operably coupled to the host and the mediator and optionally operably coupled to the non-cache data storage medium. Furthermore, in some embodiments it is located remotely from each of the mediator, the non-cache medium and the host.

[00210] The manager may be configured to carry out one or more of the following features: (a) to store data comprising one or more of file system information, bootability information and partition information in the first set of tracks of the mediator; (b) to store metadata in the third set of tracks of the mediator; (c) to store one or more files on the non-cache medium, wherein the one or more files are stored on the non-cache medium without any of file system information, bootability information or partition information (thus in some embodiments, only raw data is on the non-cache medium); (d) to store in the fourth set of tracks of the mediator the location of each file in the non-cache medium; and (e) to store on the mediator a correlation of the location of each file in the non-cache medium with a host name for a file. Preferably, the correlation of the location of each file in the non-cache medium with a host name for a file is stored in the fourth set of tracks, which corresponds to a bit field. Additionally, the manager may comprise or be operably coupled to a computer program product that contains executable instructions that are capable of executing one or more of the methods of the present invention.

[00211] Within various systems of the present invention, a SAN, according to directions stored in a computer program product, may access a bit marker table or frequency converter. These resources correlate a bit marker with each of the subunits and generate an output. Because most, if not all, of the bit markers are smaller in size than the subunits, the output is a data file that is smaller than the input file that was received from the host. Thus, whereas a file as received from the host may be a size R, the actual data as saved by the SAN may be S, wherein R>S. Preferably, R is at least twice as large as S, and more preferably R is at least three times as large as S.

[00212] The SAN takes the output file and stores it in a non-transitory storage medium, *e.g.*, non-cache media. Preferably, the SAN correlates the file as stored with the file as received from the host such that the host can retrieve the file.

[00213] For purposes of further illustration, reference may be made to **figure 1**, which shows a system for implementing methods of the present invention. In the system **10**, the host **1**, transmits files to a storage area network, **6**, that contains a processor **3** that is operably coupled to memory **4.** Optionally, the storage area network confirms receipt back to the host.

[00214] Within the memory is stored a computer program product that is designed to take the chunklets and to divide the data contained therein into subunits. The memory may also contain or be operably coupled to a reference table **5.** The table contains bit markers or frequency converters or correlation files for one or more of the subunits, and the computer program product creates a new data file that contains one or more of the markers in place of the original subunits.

[00215] The processor next causes storage of the bit markers on a recording medium, such as a non-cache medium, which may, for example, be a disk **2**. In some embodiments, initially all of the markers are the same size. However, in some embodiments, prior to storing them, one or more, preferably at least 25%, at least 50%, or at least 75% are truncated prior to storage.

[00216] **Figure 2** shows a system **100** with a mediator **70** that contains R_1 **40** and R_2 **50**, as well as a space for a bit field **60** and metadata files **30.** The representation of the mediator is for illustrative purposes only and places no limitations on the structure of the mediator or organization within it. Also shown is a non-cache medium (NCM) **20**. The non-cache medium is shown in the proximity of the mediator, but they are separate structures. In some embodiments, the mediator is located a separate device from the NCM, and one or more of the devices of the present invention are portable. The markers or other converted raw data will be stored on the NCM.

[00217] **Figure 3** shows another system **200.** In this system, the initiator (I^n) **270** transmits chunklets to a cache manager **230**, which optionally arranges for coding of data files and transmits them to the mediator **210.** Examples of hosts include but are not limited to computers or computer networks that run Microsoft Windows Server and Desktop, Apple OS X, Linux RHEL and SUSE Oracle Solaris, IBM AIX, HP UX and VM ESX and ESXi. The information that corresponds to data files is initially sent to R_1 **240**, which previously was populated with parameters that the initiator defined. The mediator may itself translate the information through use of a bit marker table or a frequency converter (not shown) or it may communicate with a remote encoder (which also may be referred to as a remote converter and is not shown), and the mediator will store within R_1 as well as within R_2 **250** copies of a file name that is received from the host. After the data has been converted, and a smaller size file has been created, within a sector map of the bit field **260** is recorded a location or locations where the file will be stored in the disk **220**. The coded data may be stored at location **285**. Prior to or instead of coding data, a hash value algorithm may be accessed and the checksums may be used for storage or conversion.

[00218] **Figure 4** shows another system **300** that is a variation of the embodiment of the system of **figure 3** and that provides for back-up of storage. In this system the initiator **370** transmits chunklets to the cache manager **330**, which forwards information to the first mediator **310** that contains data to revise the same file that was sent for **figure 3**. Either prior to receipt of the revised file or after receipt of it, but before storage of it in the non-cache media, a second mediator **380** is created from the first mediator **310**. The second mediator is an exact copy of the first mediator at the time that it was created and for the file name, at that time, points to the same sector (or sector cluster) **385** within the non-cache medium **320**.

[00219] The first revised file is received at R_1 **340** of the first mediator. The first mediator will again either translate the information through use of a bit marker table or a frequency converter (not shown) or communicate with a remote encoder. The mediator will continue to store within R_1 as well as within R_2 **350** copies of the file name that is received from the host. After the data has been converted, and a smaller size file has been created, within a sector map of the bit field **360** of the first mediator, is recorded a location that the file will be stored in the disk **320**. However, the revised file will be stored in a different sector **395**. Thus, the changes to the first mediator will not be made to the second mediator.

[00220] The host is by default in communication with the first mediator. Thus, when it wants to recall the file from storage, the first mediator will call back the data from sector **395.** Should the host or a system administrator wish to obtain a previous version of the data, it could submit the file name to the second mediator, which would look to sector **385**.

[00221] According to another embodiment of the present invention, an initiator that has previously provided metadata to the system of the present invention (*e.g.*, operating system information, bootability information, partition information, document type information QoS information, *etc.*) sends bits that correspond to a document to a cache manager. The bits may, for example, be organized in chunklets.

[00222] The cache manager may send the information to L1 cache, to L2 cache and to a mediator. The cache manager may also send an acknowledgement of receipt to the initiator.

[00223] The mediator, which may already have the relevant metadata stored on it, sends the bits that correspond to the document, to a converter, which converts the bits into coded information. Associated with or part of the converter may also be a calculator, which determines the size of the converted bits. Conversion may, for example, be through use of a bit marker table or a frequency converter.

[00224] The converter may then tell the mediator of the size of the converted file, and the mediator may determine the location at which the converted file will be stored in a non-cache-medium. Following this step, the mediator may cause storage at that location.

[00225] Any of the features of the various embodiments described in this specification can be used in conjunction with features described in connection with any other embodiments disclosed unless otherwise specified. Thus, features described in connection with the various or specific embodiments are not to be construed as not suitable in connection with other embodiments disclosed herein unless such exclusivity is explicitly stated or implicit from context.

[00226] **Examples:**

[00227] **Example 1: Bit Marker Table (Prophetic)**

[00228] Within a reference locator table each unique marker is identified as corresponding to unique strings of bits. The table may be stored in any format that is commonly known or that comes to be known for storing tables and that permits a computer algorithm to obtain an output that is assigned to each input.

[00229] Table I below provides an example of excerpts from a bit marker table where the subunits are 8 bits long.

[00230] Table I

Bit Marker (as stored)	Subunit = 8 bits (input)
0101	00000001
1011	00000010
1100	00000011
1000	00000100
1010	00000101
11111101	11111101

[00231] By way of example and using the subunits identified in Table I, if the input were 00000101 00000100 00000101 00000101 00000001, the output would be: 1010 1000 1010 1010 0101. When the bit marker output is smaller than the subunit input, it will take up less space on a storage medium, and thereby conserve both storage space and the time necessary to store the bits.

[00232] As a person of ordinary skill in the art will recognize, in a given bit marker table such as that excerpted to produce Table I, if all combinations of bits are to be used there will need to be 2^N entries, wherein N corresponds to the number of bits within a subunit. When there are 8 bits, there are 256 entries needed. When there are 16 bits in a subunit one needs 2^{16} entries, which equals 65,536 entries. When there are 32 bits in a subunit, one needs 2^{32} entries, which equals 4,294,967,296 entries. If one knows that certain strings of bits will not be used in files, then the table may allocate markers starting with the smallest ones.

[00233] Example 2: Bit Marker Table For Pre-processed Subunits (Prophetic)

[00234] Because as the subunit size gets larger the table becomes more cumbersome, in some embodiments, the table may be configured such that all zeroes from one end of the subunit column are missing and prior to accessing the table, all zeroes from that end of each subunit are removed. Thus, rather than a table from which Table I is excerpted, a table from which Table II is excerpted could be configured.

[00235] **Table II**

Bit Marker (output)	Pre-processed Subunit
0101	00000001
1011	0000001
1100	00000011
1000	000001
1010	00000101
11111101	11111101

[00236] As one can see, in the second and fourth lines, after the subunits were pre-processed, they had fewer than eight bits. However, the actual subunits in the raw data received from the host all had eight bits. Because the system in which the methods are implemented can be designed to understand that the absence of a digit implies a zero and all absences of digits are at the same end of any truncated subunits, one can use a table that takes up less space and that retains the ability to assign unique markers to unique subunits. Thus, the methods permit the system to interpret 00000001 (seven zeroes and a one) and 0000001 (six zeroes and a one) as different.

[00237] In order to implement this method, one may deem each subunit (or each chunklet if subunits are not used) to have a first end and a second end. The first end can be either the right side of the string of bits or the left side, and the second end would be the opposite side. For purposes of illustration, one may think of the first end as being the leftmost digit and the second end as being the rightmost digit. Under this method one then analyzes one or more bits within each subunit of each chunklet to determine if the bit at the second end has a value 0. This step may be referred to as preprocessing, and the subunits after they are preprocessed appear in the right column of Table II. If the bit at the second end has a value 0, the method may remove the bit at the second end and all bits that have the value 0 and form a contiguous string of bits with that bit, thereby forming a revised subunit (pre-processed subunit in the table) for any subunit that originally had a 0 at the second end.

[00238] One may use a computer algorithm that reviews each subunit to determine whether at the second end there is a 0 and if so removes the 0 to form the pre-processed subunit, which also may be referred to as a revised subunit with a revised second end at a position that was adjacent to the second end of the subunit. Next, the algorithm reviews the revised subunit to determine whether at its now revised second end there is a 0 and if so removing the 0 to form a further revised second end. In this method, the revised second end would be the location that was previously adjacent to the bit at the second end. Any further revised second end would have been two or more places away from the second end of the subunit. Thus, the term “revised” means a shortened or truncated second end. The algorithm may repeat this method for the revised subunit until a shortened chunklet is generated that has a 1 at its second end.

[00239] As persons of ordinary skill in the art will recognize, the aforementioned method is described as being applied by removing zeroes from the second end until a 1 is at the revised second end or further revised second end. The methods could be designed in reverse so that the system removes 1s from the second end until a 0 is at a revised second end or further revised second end. Additionally, with the present disclosure a person of ordinary skill in the art could remove bits from the first end instead of the second end and use a table created to convert those revised subunits into bit markers.

[00240] Example 3: Frequency Exchange (Prophetic)

[00241] Based on empirical analysis, one can determine the frequency of each subunit within a type of document or a set of documents received from a particular host or from within a set of documents that have been received within a given timeframe, *e.g.*, the past year or past two years. With this information, rather than look to a table as illustrated in Table I or Table II in which the subunits are organized in numerical order, one could look to a frequency converter in which the smaller bit markers are associated with subunits that are predicted most likely to appear within a file, within a type of document or within a set of documents as received from a particular host. Thus, within the frequency converter, the markers are a plurality of different sizes and markers of a smaller size are correlated with higher frequency subunits.

[00242] Table III: Frequency Converter

Bit Marker (output)	Frequency	Subunit = 8 bits (input)
0101	16%	00000001
1000	15%	00000010
11011	10%	00000011
10011101	0.00001%	00000100
10111110	0.00001%	00000101
1100	15%	11111101

[00243] Table III is an example of an excerpt from a frequency converter that uses the same subunits as Table I. However, one will note that the bit markers are not assigned in sequence, and instead larger bit markers are assigned to lower frequency subunits. As the table illustrates, the marker that is assigned to subunit 00000011 is twenty five percent larger than that assigned to subunit 00000001, and for subunit 11111101, despite being of high numerical value, it receives a smaller bit marker because it appears more frequently in the types of files received from the particular host. Thus, if one used Table I and the subunit 11111101 appears in 10,000 places, it would correspond to 111,111,010,000 bits. However, if one used Table III, only 11,000,000 bits would need to be used for storage purposes for the same information. Although not shown in this method, the subunits could be preprocessed to remove zeroes from one end or the other, and the table could be designed to contain the correlating truncated subunits.

[00244] As noted above, frequency converters can be generated based on analyses of a set of files that are deemed to be representative of data that is likely to be received from one or more hosts. In some embodiments, the algorithm that processes the information could perform its own quality control and compare the actual frequencies of subunits for documents from a given time period with those on which the allocation of marker in the frequency converter are based. Using statistical analyses it may then determine if for future uses a new table should be created that reallocates how the markers are associated with the subunits. As a person of ordinary skill in the art will recognize, Table III is a simplified excerpt of a frequency converter. However, in practice one may choose a hexadecimal system in order to obtain the correlations. Additionally, the recitation of the frequencies on which the table is based is included for the convenience of the reader, and they need not be included in the table as accessed by the various embodiments of the present invention.

[00245] **Example 4: Allocation of Space in a Mediator (Prophetic)**

[00246] In a hypothetical recording medium that is 1 MB in size, a person of ordinary skill in the art may map the sectors as follows:

[00247] The 1 MB recording medium has 1,024,000 Bytes, which corresponds to 250 sectors. (1,024,000/4096= 250). The geometry of the recording medium may be summarized as follows: Volume = (c * h * spt * ss), wherein

[00248] c (number of cylinder) = 7;

[00249] h (number of heads) = 255;

[00250] spt (sectors per track) = 63; and

[00251] ss (sector size in bytes) = 4096.

[00252] Within the mediator, the sectors may be allocated as follows:

[00253] Table IV

Address	**Actual Non-cache-media LBA**
0-15	mediator <<Reserved 1>> “Boot Sector 0” +15
16-31	mediator location <<Reserved 2>> Sys_Internal Only
32-35	mediator_Metadata
36	Map Data “LBA-nnnnnnnnnnnn”
37	Map Data “LBA-nnnnnnnnnnnn”
. . .	Map Data “LBA-nnnnnnnnnnnn”
250	Map Data “LBA-nnnnnnnnnnnn”

[00254] Example 5: Space Saving

[00255] A system of the present invention received 42.5 million blocks of data in I/O streams of 4096 Bytes. It applied the MD5 hash algorithm to generate 7.8 million blocks of hash value that corresponded to the 42.5 million blocks in the I/O stream.

[00256] This translated into use of only 18.5% of the space that would have been needed to store the original 42.5 million blocks. A conflict module was applied, and it verified that no conflicts existed, *i.e.*, no duplication of hash values were generated for different blocks.

Claims

I claim:

1. A method for storing data on a non-cache recording medium comprising:
    i. receiving an I/O stream of N Bytes;
    ii. fragmenting the N Bytes into fragmented units of X Bytes;
    iii. applying a cryptographic hash function to each fragmented unit of X Bytes to form a generated hash function value for each fragmented unit of X Bytes;
    iv. accessing a correlation file, wherein the correlation file associates a stored hash function value of Y bits with each of a plurality of stored sequences of X Bytes and for each fragmented unit
        (a) if the generated hash function value for a fragmented unit of X Bytes is in the correlation file, using the stored hash function value of Y bits for storage on a non-cache recording medium; and
        (b) if the generated hash function value for the fragmented unit of X Bytes is not in the correlation file, then storing the generated hash function value of Y bits with the fragmented unit of X Bytes in the correlation file and using the generated hash function value for storage on the non-cache recording medium.

2. The method according to claim 1, wherein in (iii) the algorithm is the MD5 Message-Digest Algorithm.

3. The method according to claim 1 further comprising applying a conflict resolution module, wherein if the conflict resolution module determines that for a fragmented unit of X Bytes, a hash function value is generated that is same as a stored hash function value in the correlation file, but the fragmented unit of X Bytes is different from the X Bytes associated with the stored hash function value, then

associating different Z bits with the stored hash function value and the generated hash function value.

4. The method according to claim 3, wherein in the correlation file, for all hash function values for which no conflict exists Z bits are associated and the Z bits are 8 to 16 zeroes.

5. The method according to claim 4, wherein N=4096 and X =512.

6. The method according to claim 5, wherein Y=128 or 256.

7. The method according to claim 1 further comprising storing a plurality of hash function values on the non-cache recording medium.

8. The method according to claim 1 further comprising encoding each hash function value for storage through use of a bit marker table to generate a bit marker for each hash function value for storage and storing each bit marker in the non-cache recording medium.

9. The method according to claim 1 further comprising encoding each hash function value for storage through use of a frequency converter, wherein for each hash function value for storage, a converted string of bits is generated, wherein for at least two different strings of Y bits, the converted strings of bits that are output are of different lengths.

10. The method according to claim 9, wherein for every plurality of converted strings of bits that are of different sizes there is a first converted string of bits that is A bits long and a second converted string of bits that is B bits long, wherein $A < B$, and the identity of the A bits of the first converted string of bits is not the same as the identity of the first A bits of the second converted string of bits.

11. The method according to claim 3 further comprising encoding each hash function value for storage through use of a frequency converter, wherein for each hash function value for storage, a converted string of

bits is generated, wherein for at least two different strings of Y bits, the converted strings of bits that are output are of different lengths.

12. The method according to claim 1, wherein the I/O is received from a host and said method further comprises acknowledging receipt of the I/O stream to the host.

13. The method according to claim 12, wherein the host records the I/O as being stored at a first storage address and the converted string of bits are stored in the non-cache recording medium at a second storage address, wherein the first storage address is not the same as the second storage address.

14. The method according to claim 13, wherein the first storage address corresponds to a file of a first file size and the second storage address corresponds to a file of a second file size, wherein the first file size is at least four times as large as the second file size.

15. The method according to claim 13 further comprising storing the second storage address on a mediator, wherein the mediator is a non-transitory storage medium.

16. The method according to claim 15, wherein the second storage address is stored in a bit field.

17. The method according to claim 16, wherein the mediator comprises:
    i. a first set of tracks, wherein the first set of tracks comprises file system information, bootability information and partition information;
    ii. a second set of tracks, wherein the second set of tracks comprises a copy of the first set of tracks;
    iii. a third set of tracks, wherein the third set of tracks comprises metadata other than file system information, bootability information and partition information; and

iv. a fourth set of tracks, wherein the fourth set of tracks comprise the bit field.

18. A method for storing data on a non-cache recording medium comprising:
    i. receiving an I/O stream of N Bytes;
    ii. fragmenting the N Bytes into fragmented units of X Bytes;
    iii. associating a cryptographic hash function value with each fragmented unit of X Bytes, wherein said associating comprises accessing a correlation file, wherein the correlation file associates a stored hash function value of Y bits with each of a plurality of stored sequences of X Bytes and
        (a) if the sequence of a fragmented unit of X Bytes is in the correlation file, then using the stored hash function value of Y bits for storage on a non-cache recording medium; and
        (b) if the sequence of a fragmented unit of X Bytes is not in the correlation file, then storing a new hash function value of Y bits with the fragmented unit of X Bytes in the correlation file and using the new hash function value of Y bits for storage on the non-cache recording medium.

19. A method for storing data on a non-cache recording medium comprising:
    i. receiving an I/O stream of N Bytes;
    ii. applying a cryptographic hash function value algorithm to each unit of N Bytes to form a generated hash function value for each unit of N Bytes;
    iii. accessing a correlation file, wherein the correlation file associates a stored hash function value of Y bits with each of a plurality of stored sequences of N Bytes and
        (a) if the generated hash function value for the unit of N Bytes is in the correlation file, using the stored hash

function value of Y bits for storage on a non-cache recording medium; and

(b) if the generated hash function value for the unit of N Bytes is not in the correlation file, then storing the generated hash function value of Y bits with the unit of N Bytes in the correlation file and using the generated hash function value for storage on the non-cache recording medium.

20. A method for storing data on a recording medium comprising:

i. receiving a plurality of digital binary signals, wherein the digital binary signals are organized in a plurality of chunklets, wherein each chunklet is N bits long, wherein N is an integer number greater than 1 and wherein the chunklets have an order;

ii. dividing each chunklet into subunits of a uniform size and assigning a marker to each subunit from a set of X markers to form a set of a plurality of markers, wherein X is less than or equal to the number of different combinations of bits within a subunit, identical subunits are assigned the same marker and at least one marker is smaller than the size of a subunit; and

iii. storing the set of the plurality of markers on a non-transitory recording medium in an order that corresponds to the order of the chunklets.

21. The method according to claim 20, wherein said assigning comprises accessing a bit marker table, wherein within the bit marker table each unique marker is identified as corresponding to a unique string of bits.

22. The method according to claim 21, wherein each subunit has a first end and a second end and prior to assigning said marker, the method further comprises analyzing one or more bits within each subunit of each chunklet to determine if the bit at the second end has a value 0 and if the bit at the second end has a value 0, removing the bit at the second end and all bits that have the value 0 and form a contiguous

string of bits with the bit at the second end, thereby forming a revised subunit for any subunit that has a 0 at the second end.

23. The method according to claim 22, wherein a computer algorithm:

(a) reviews each subunit to determine whether at the second end there is a 0 and if so removes the 0 to form a revised subunit with a revised second end at a position that was adjacent to the second end of the subunit;
(b) reviews each revised subunit to determine whether at the revised second end there is a 0 and if so removing the 0 to form a further revised second end; and
(c) repeating (b) for each revised subunit until a shortened subunit is generated that has a 1 at its second end.

24. The method according to claim 21, wherein each subunit has a first end and a second end and prior to assigning said marker, the method further comprises analyzing one or more bits within each subunit of each chunklet to determine if the bit at the second end has a value 1 and if the bit at the second end has a value 1, removing the bit at the second end and all bits that have the value 1 and form a contiguous string of bits with the bit at the second end, thereby forming a revised subunit for any subunit that has a 1 at the second end.

25. The method according to claim 24, wherein a computer algorithm:

(a) reviews each subunit to determine whether at the second end there is a 1 and if so removes the 1 to form a revised subunit with a revised second end at a position that was adjacent to the second end of the subunit;
(b) reviews each revised subunit to determine whether at the revised second end there is a 1 and if so removing the 1 to form a further revised second end; and
(c) repeating (b) for each revised subunit until a shortened subunit is generated that has a 0 at its second end.

26. The method according to claim 21, wherein the markers are stored in a frequency converter, the markers are a plurality of different sizes and markers of a smaller size are correlated with higher frequency subunits.

27. The method according to claim 20, wherein a plurality of different markers are formed from different numbers of bits.

28. A method for retrieving data from a recording medium comprising:
    i. accessing a recording medium, wherein the recording medium stores a plurality of markers in an order;
    ii. translating the plurality of markers into a set of chunklets, wherein each chunklet is N bits long, wherein N is an integer number greater than 1 and wherein the chunklets have an order that corresponds to the order of the plurality of markers and wherein the translating is accomplished by accessing a bit marker table, wherein within the bit marker table each unique marker is identified as corresponding to a unique string of bits; and
    iii. generating an output that comprises the set of chunklets.

29. The method according to claim 28, wherein the plurality of markers as stored on the recording medium have sizes from X to Y wherein Y>X and at least one marker has a size X and at least one marker has a size Y.

30. The method according to claim 29, wherein said translating comprises rendering all of the markers that are smaller than length Z into markers of a length Z by adding 0's to a first end of the markers, wherein Z is greater than or equal to Y and translating the markers of length Z into chunklets, wherein the chunklets are larger than length Z.

31. The method according to claim 30, wherein said translating the markers of length Z into chunklets comprises translating the markers of length Z into subunits and combining the subunits into chunklets.

32. A method for retrieving a document from storage comprising the method of claim 28, and further comprising associating the output with a file type and transmitting the output to an operating system that is capable of converting the chunklets into a document of said file type.

33. A method for storing data on a recording medium comprising:

    i. receiving a plurality of digital binary signals, wherein the digital binary signals are organized in chunklets, wherein each chunklet is N bits long, each chunklet has a first end and a second end, N is an integer number greater than 1, and the chunklets have an order;
    ii. dividing each chunklet into a plurality of subunits, wherein each subunit is A bits long;
    iii. analyzing each subunit to determine if the bit at the second end has value 0 and if the bit at the second end has a value 0, removing the bit at the second end and all bits that have the value 0 and form a contiguous string of bits with the bit at the second end, thereby forming a revised chunklet for any chunklet that has a 0 at the second end; and
    iv. on a non-transitory recording medium, storing in said order each revised subunit and each subunit that is A bits long and has a 1 at its second end.

34. A method for storing data on a recording medium comprising:

    i. receiving a plurality of digital binary signals, wherein the digital binary signals are organized in chunklets, wherein each chunklet is N bits long, each chunklet has a first end and a second end, N is an integer number greater than 1, and the chunklets have an order;

ii. dividing each chunklet into a plurality of subunits, wherein each subunit is A bits long;

iii. analyzing each subunit to determine if the bit at the first end has a value 0 and if the bit at the first end has a value 0, removing the bit at the first end and all bits that have the value 0 and form a contiguous string of bits with the bit at the first end, thereby forming a first revised subunit for any subunit that has a 0 at the first end;

iv. analyzing each subunit to determine if the bit at the second end has value 0 and if the bit at the second end has a value 0, removing the bit at the second end and all bits that have the value 0 and form a contiguous string of bits with the bit at the second end, thereby forming a second revised subunit for any subunit that has a 0 at the second end; and

v. for each subunit

   (a) if the sizes of the first revised subunit and the second revised subunit are the same, storing the first revised subunit or the second revised subunit,

   (b) if the first revised subunit is smaller than the second revised subunit, storing the first revised subunit,

   (c) if the second revised subunit is smaller than the first revised subunit, storing the second revised subunit,

   (d) if there are no revised subunits, storing the subunit,

   (e) if there is no first revised subunit, but there is a second revised subunit, storing the second revised subunit, and

   (f) if there is no second revised subunit, but there is a first revised subunit, storing the first revised subunit,

   wherein each revised subunit that is stored is stored with information that indicates if one or more bits were removed from the first end or the second end.

35. A method for retrieving data from a recording medium comprising:

i. accessing a recording medium, wherein the recording medium stores a plurality of data units in a plurality of locations, wherein each data unit contains a plurality of bits and the maximum size of the data unit is N bits, at least one data unit contains fewer than N bits and the data units have an order;

ii. retrieving the data units and adding one or more bits at an end of any data unit that is less than N bits long to generate a set of chunklets that correspond to the data units, wherein each chunklet contains the same number of bits; and

iii. generating an output that comprises the set of chunklets in an order that corresponds to the order of the data units.

36. The method according to claim 35, wherein in (ii) bits of value 0 are added.

37. A method for retrieving a document from storage comprising the method of claim 36, and further comprising associating the output with a file type and transmitting the output to an operating system that is capable of converting the chunklets into a document of said file type.

38. A method for storing electronic data, said method comprising:

i. receiving a set of parameters, wherein the parameters comprise one or more of file system information, bootability information and partition information;

ii. receiving metadata;

iii. receiving one or more files, wherein each file has a file name;

iv. storing the parameters and metadata on a mediator;

v. storing each of the files on a non-cache medium at a location; and

vi. storing on the mediator a correlation of each file name with a location on the non-cache medium.

39. The method according to claim 38 further comprising encoding the file prior to storing the file on the non-cache medium.

40. The method according to claim 39, wherein the encoding comprises using a bit marker table to create a converted file.

41. The method according to claim 39, wherein the encoding comprises using a frequency converter to create a converted file.

42. The method according to claim 40, wherein the converted file does not contain any of file system information, bootability information or partition information.

43. The method according to claim 38, wherein the parameters are stored in a plurality of reserve tracks.

44. The method according to claim 38, wherein the plurality of reserve tracks are a first set of reserve tracks and the method further comprises copying the parameters into a second set of reserve tracks.

45. The method according to claim 44 further comprising using the second set of reserve tracks to check for errors in the first set of reserve tracks.

46. The method according to claim 45, wherein the metadata corresponds to instructions for thin-provisioning.

47. The method according to claim 38, wherein the file is received from a host that records the file as being stored at a virtual address and the virtual address is not the same as the location of the file.

48. A method for backing up data, said method comprising the method according to claim 38, wherein the mediator is a first mediator and the location is a first location, said method further comprises generating a second mediator, wherein the second mediator is a copy of the first mediator at time T1, and at time T2, which is after T1:

    i. receiving instructions to save a revised file that is located at the first location;

ii. storing a new file at a second location, wherein the new file corresponds to the revised file; and

iii. updating the first mediator to correlate the file name with the second location, wherein on the second mediator, the file name is correlated with the first location.

49. A method for backing up data, said method comprising:

i. on a first mediator, correlating a plurality of file names with a plurality of locations of data files, wherein the locations of the data files correspond to locations on a first non-cache medium and the first mediator is configured to permit a user who identifies a specific file name to retrieve a data file that corresponds to the specific file name;

ii. copying the plurality of data files to a second non-cache medium;

iii. generating a second mediator, wherein the second mediator is a copy of the first mediator at time T1 and within the second mediator the locations of a plurality of data files on the second non-cache medium are correlated with the file names;

iv. receiving instructions to save revisions to a data file; and

v. at time T2, which is after T1, in the first non-cache medium saving the revisions to the data file.

50. A data storage and retrieval system comprising:

i. a non-cache data storage medium;

ii. a mediator, wherein the mediator is stored remotely from the non-cache data storage medium, and the mediator comprises:

(a) a first set of tracks;

(b) a second set of tracks;

(c) a third set of tracks; and

(d) a fourth set of tracks; and

iii. a manager, wherein the manager is configured:

(a) to store data comprising one or more of file system information, bootability information and partition information in the first set of tracks.
(b) to store metadata in the third set of tracks;
(c) to store one or more files on the non-cache medium, wherein the one or more files are stored on the non-cache medium without any of file system information, bootability information and partition information;
(d) to store in the fourth set of tracks the location of each file in the non-cache medium; and
(e) to store a correlation of the location of each file in the non-cache medium with a host name for a file.

51. A system of claim 50, wherein the manager is further configured to copy the information in the first set of tracks into the second set of tracks.

52. The system of claim 50, wherein the location of each file in the non-cache medium is not the same as the location at which the host believes that the file is located.

53. The system according to claim 50, wherein the one or more files are converted to form converted files, wherein the converted files take up less space than the files from which they were converted.

54. The system according to claim 50, wherein the mediator is a first mediator and the system further comprises a second mediator and a module for copying the information within the first mediator into the second mediator.

55. The system according to claim 54, wherein the first mediator and the second mediator correlate the same file name with different locations within the non-cache data storage medium.

56. The system according to claim 54, wherein the non-cache data storage medium is a first non-cache data storage medium and the system further comprises a second non-cache data storage medium, and the first mediator correlates a file name with a location within the first non-cache data storage medium and the second mediator correlates the file name with a location within the second non-cache data storage medium.

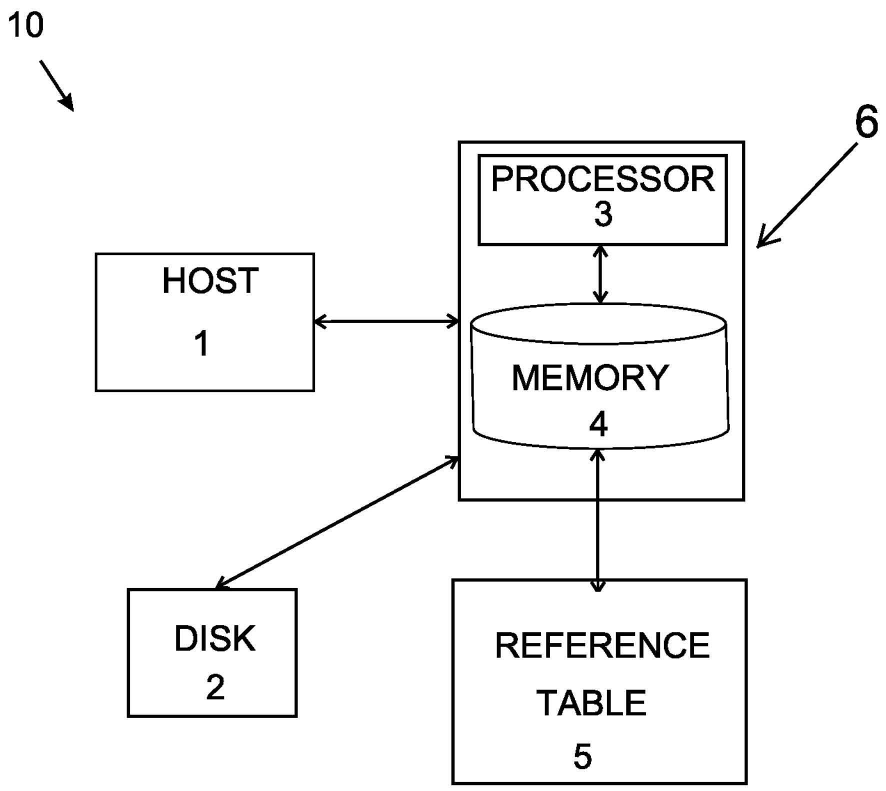


Figure 1

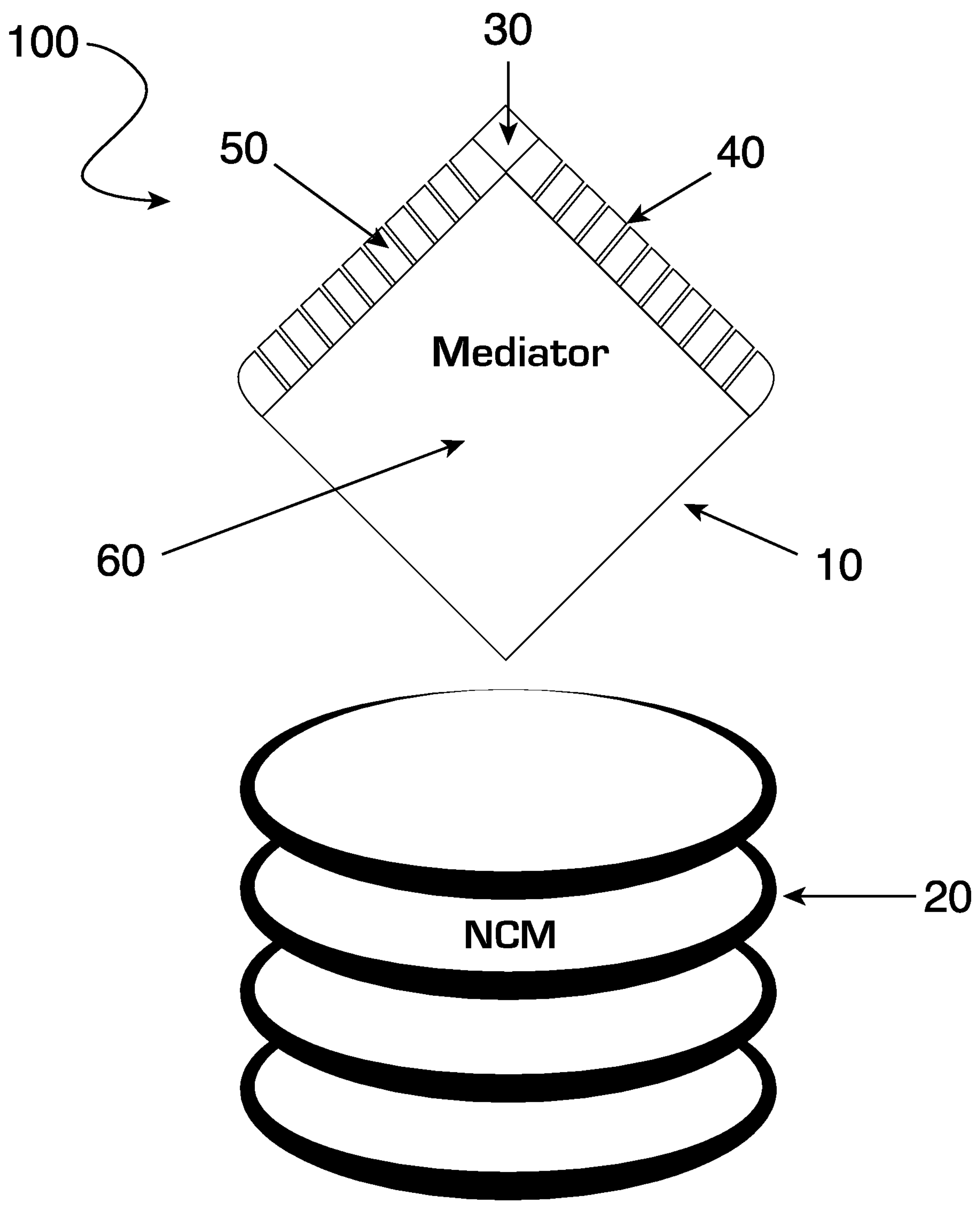


Figure 2

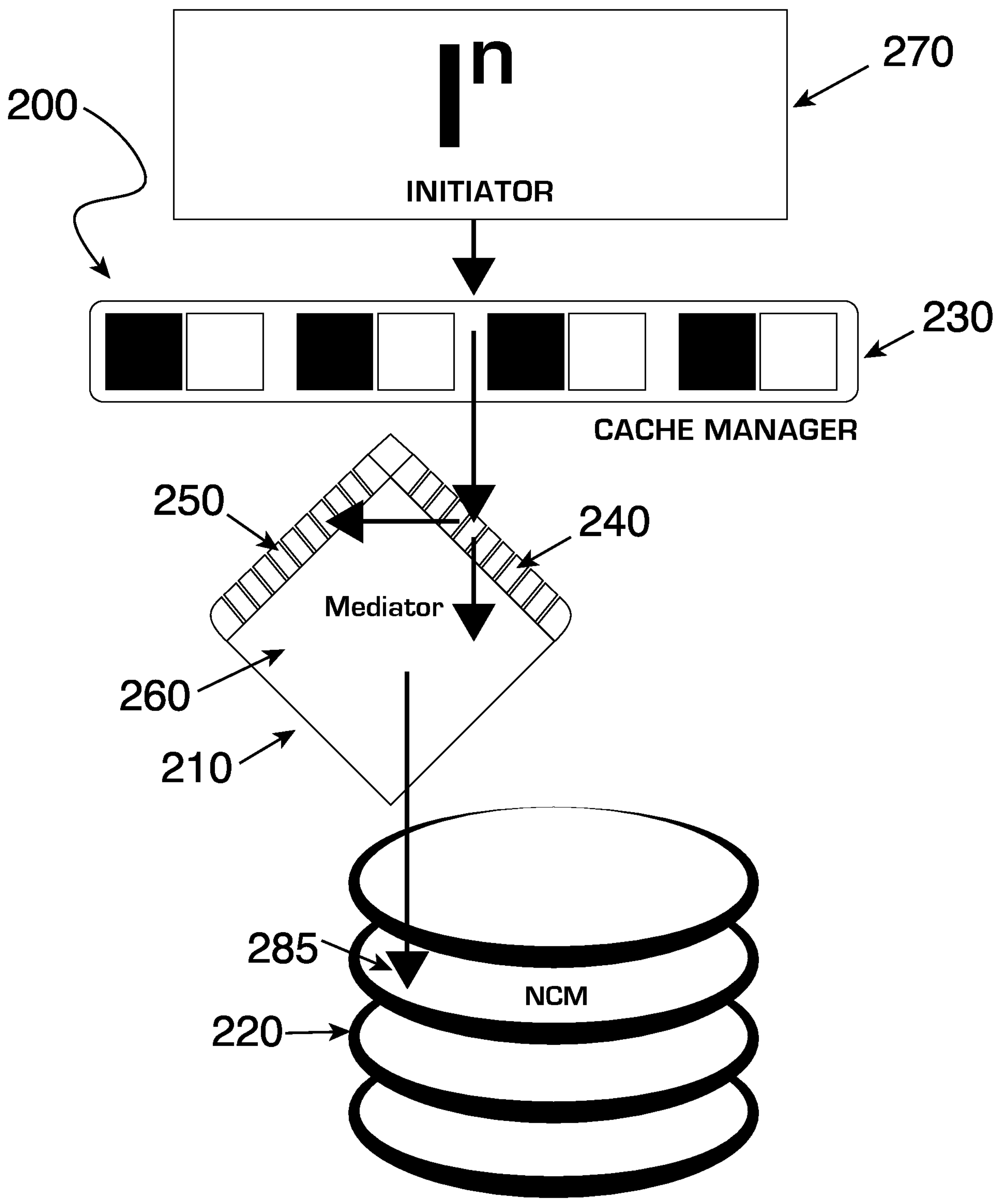

I^n
INITIATOR
270
200
230
CACHE MANAGER
250
240
Mediator
260
210
285
NCM
220

Figure 3

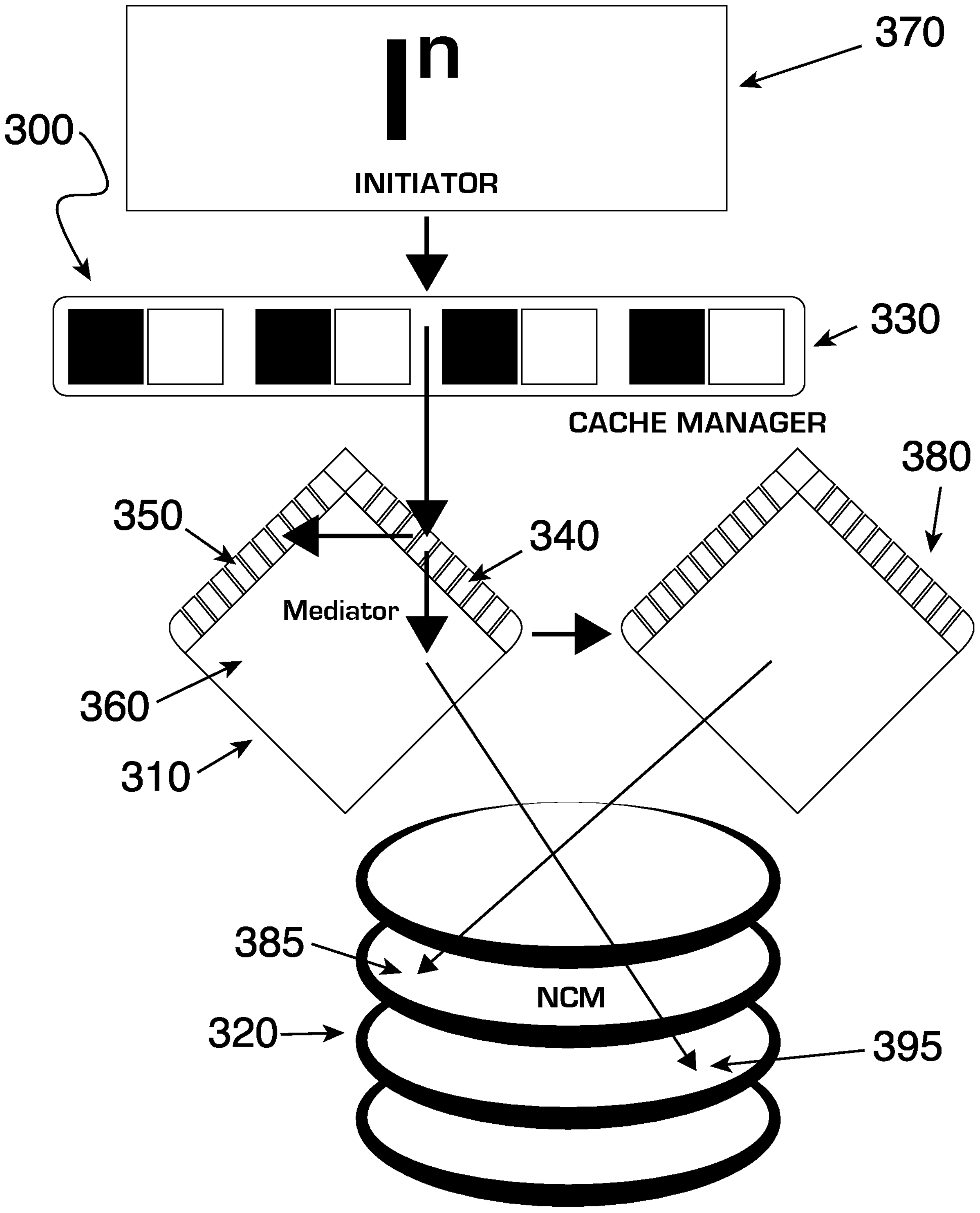

I^n
INITIATOR
370
300
330
CACHE MANAGER
380
350
340
Mediator
360
310
385
NCM
320
395

Figure 4

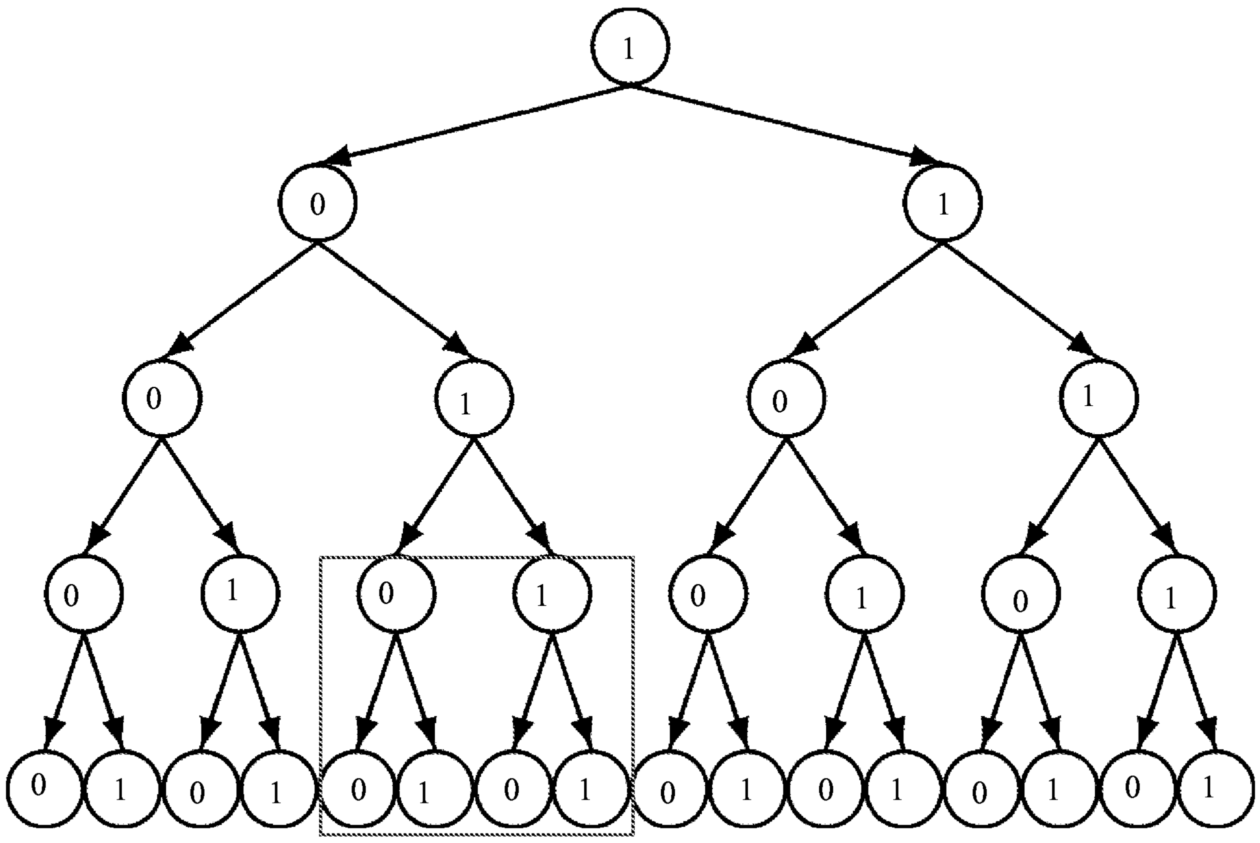


Figure 5

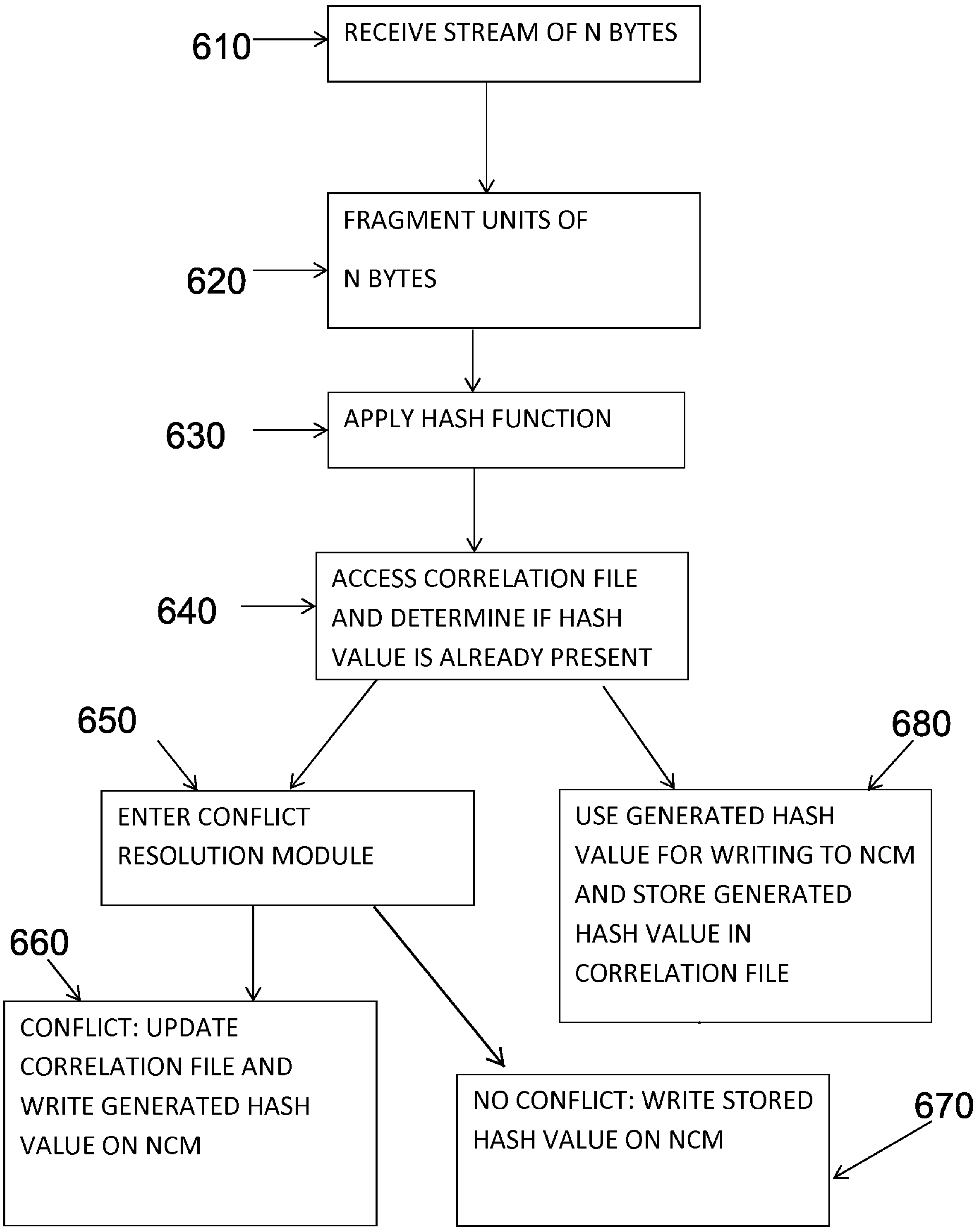

610
RECEIVE STREAM OF N BYTES
620
FRAGMENT UNITS OF N BYTES
630
APPLY HASH FUNCTION
640
ACCESS CORRELATION FILE AND DETERMINE IF HASH VALUE IS ALREADY PRESENT
650
ENTER CONFLICT RESOLUTION MODULE
680
USE GENERATED HASH VALUE FOR WRITING TO NCM AND STORE GENERATED HASH VALUE IN CORRELATION FILE
660
CONFLICT: UPDATE CORRELATION FILE AND WRITE GENERATED HASH VALUE ON NCM
NO CONFLICT: WRITE STORED HASH VALUE ON NCM
670

Figure 6

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
7 August 2014 (07.08.2014)

(10) International Publication Number
WO 2014/121102 A3

(51) International Patent Classification: *G06F 12/00* (2006.01)

(21) International Application Number: PCT/US2014/014209

(22) International Filing Date: 31 January 2014 (31.01.2014)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

13/756,921	1 February 2013 (01.02.2013)	US
13/797,093	12 March 2013 (12.03.2013)	US
13/908,239	3 June 2013 (03.06.2013)	US

(72) Inventor; and
(71) Applicant : **IGNOMIRELO, Brian** [US/US]; 40 Manor Road, Colts Neck, NJ 07722 (US).

(74) Agent: **LOCKE, Scott, D.**; Dorf & Nelson LLP, The International Corporate Center, 555 Therdore Fremd Ave., Rye, NY 10580 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

[Continued on next page]

(54) Title: METHODS AND SYSTEMS FOR STORING AND RETRIEVING DATA

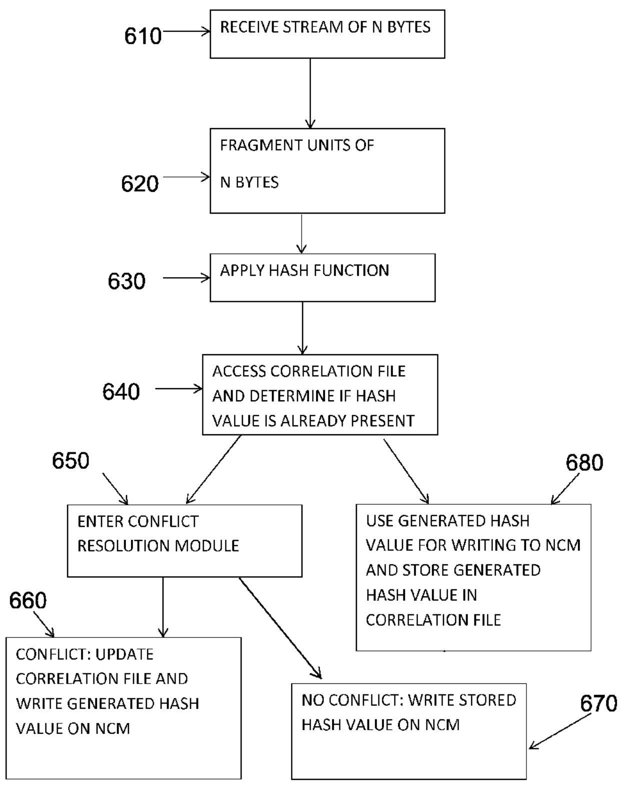

610
RECEIVE STREAM OF N BYTES
620
FRAGMENT UNITS OF N BYTES
630
APPLY HASH FUNCTION
640
ACCESS CORRELATION FILE AND DETERMINE IF HASH VALUE IS ALREADY PRESENT
650
680
ENTER CONFLICT RESOLUTION MODULE
USE GENERATED HASH VALUE FOR WRITING TO NCM AND STORE GENERATED HASH VALUE IN CORRELATION FILE
660
CONFLICT: UPDATE CORRELATION FILE AND WRITE GENERATED HASH VALUE ON NCM
NO CONFLICT: WRITE STORED HASH VALUE ON NCM
670

Figure 6

(57) **Abstract**: Through use of the technologies of the present invention, one is able to store and to retrieve data efficiently. In various embodiments, one may realize these efficiencies by coding the data and storing coded data that is of a smaller size than original data. The present invention relates to the field of data storage and retrieval.

WO 2014/121102 A3

Declarations under Rule 4.17:

— *of inventorship (Rule 4.17(iv))*

Published:

— *with international search report (Art. 21(3))*

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments (Rule 48.2(h))*

(88) Date of publication of the international search report:
9 October 2014

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2014/014209

A. CLASSIFICATION OF SUBJECT MATTER
IPC(8) - G06F12/00 ((2014.01)
USPC - 711/100
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC(8) - G06F12/00; G06F17/30; G06F17/00 (2014.01)
USPC - 711/100,162; 707/999.006

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
CPC - G06F11/1453; G06F3/0641; G06F3/0608 (2014.02)

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
PatBase, Google Patents, Google Scholar,

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 2011/0276771 A1 (TAJIMA et al.) 10 November 2011 (10.11.2011) entire document	1-19
Y	US 2008/0133835 A1 (ZHU et al.) 05 June 2008 (05.06.2008) entire document	1-19
A	US 6,560,599 B1 (BOA et al.) 06 May 2003 (06.05.2003) entire document	1-19
A	US 2007/0101074 A1 (PATTERSON) 03 May 2007 (03.05.2007) entire document	1-19
A	US2010/0250501 (MANDAGERE et al.) 30 September 2010 (30.09.2010) entire document	1-19

☐ Further documents are listed in the continuation of Box C. ☐

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search 31 July 2014	Date of mailing of the international search report 20 AUG 2014
Name and mailing address of the ISA/US Mail Stop PCT, Attn: ISA/US, Commissioner for Patents P.O. Box 1450, Alexandria, Virginia 22313-1450 Facsimile No. 571-273-3201	Authorized officer: Blaine R. Copenheaver PCT Helpdesk: 571-272-4300 PCT OSP: 571-272-7774

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2014/014209

Box No. II Observations where certain claims were found unsearchable (Continuation of item 2 of first sheet)

This international search report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box No. III Observations where unity of invention is lacking (Continuation of item 3 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

See Extra Sheet

1. ☐ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.

2. ☐ As all searchable claims could be searched without effort justifying additional fees, this Authority did not invite payment of additional fees.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☒ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:
1-19

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest and, where applicable, the payment of a protest fee.

☐ The additional search fees were accompanied by the applicant's protest but the applicable protest fee was not paid within the time limit specified in the invitation.

☐ No protest accompanied the payment of additional search fees.

Continuation of Box No. III:

This application contains the following inventions or groups of inventions which are not so linked as to form a single general inventive concept under PCT Rule 13.1. In order for all inventions to be examined, the appropriate additional examination fees must be paid.

Group I, claims 1-19, drawn to a method of storing data on a non-cache recording medium based on a generated hash function value for each fragmented unit of X bytes.
Group II, claims 20-37, drawn to a method for storing data on a recording medium and retrieving data from a recording medium wherein digital binary signals are organized in a plurality of chunklets of N bits long that correspond to the data units.
Group III, claims 38-56, drawn to a method for storing electronic data, backing up data, and a data storage and retrieval system by using parameters comprising one or more of file system information, bootability information and partition information and correlating a plurality of file names with a plurality of locations of data files.

The inventions listed as Groups I, II and III do not relate to a single general inventive concept under PCT Rule 13.1 because, under PCT Rule 13.2, they lack the same or corresponding special technical features for the following reasons: the special technical feature of the Group I invention: if the generated hash function value for a fragmented unit of X Bytes is in the correlation file, using the stored hash function value of Y bits for storage on a non-cache recording medium, and if the generated hash function value for the fragmented unit of X Bytes is not in the correlation file, then storing the generated hash function value of Y bits with the fragmented unit of X Bytes in the correlation file and using the generated hash function value for storage on the non-cache recording medium as claimed therein is not present in the invention of Groups II and III. The special technical feature of the Group II invention: generating a set of chunklets that are N bits long and correspond to the data units, dividing each chunklet into a plurality of subunits, and analyzing each subunit to determine if the bit at the second end has value 0 and if the bit at the second end has a value 0, removing the bit at the second end and all bits that have the value 0 and form a contiguous string of bits with the bit at the second end, thereby forming a revised chunklet for any chunklet that has a 0 at the second end as claimed therein is not present in the invention of Groups I or III. The special technical feature of the Group III invention: storing the parameters and metadata on a mediator, storing each of the files on a non-cache medium at a location, and storing on the mediator a correlation of each file name with a location on the non-cache medium as claimed therein is not present in the invention of Groups I or II.

Groups I, II and III lack unity of invention because even though the inventions of these groups require the technical feature of storing and retrieving data on and from a non-cache recording medium where data is partitioned into N- bit units and correlation file is used, this technical feature is not a special technical feature as it does not make a contribution over the prior art. Specifically,
US 2010/0250501 A1 to Mandagere et al. 30 September 2010 (30.09.2010) discloses storage management through adaptive duplication (para. 0001) and teaches storing and retrieving data on and from a non-cache recording medium (para . 0015, retrieve first portion of a plurality of stored objects from at least one storage device; para. 0030, retrieving a portion of many stored objects (called training objects) from a storage device; claim 4, storing the second plurality of deduplication chunk portions in a storage sub-system) where data is partitioned into N- bit units (para. 0050, for data of type 2, sub file chunking algorithms are selected) and correlation file is used (para. 0059, to correlate each of these classes With metadata of the files (e. g. file system metadata, Hidden, System and Compressed) that are contained in them).

Since none of the special technical features of the Group I, II or III inventions are found in more than one of the inventions, unity of invention is lacking.

(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 105339904 A
(43)申请公布日 2016.02.17

(21)申请号 201480016823.1

(22)申请日 2014.01.31

(30)优先权数据
13/756,921 2013.02.01 US
13/797,093 2013.03.12 US
13/908,239 2013.06.03 US

(85)PCT国际申请进入国家阶段日
2015.09.18

(86)PCT国际申请的申请数据
PCT/US2014/014209 2014.01.31

(87)PCT国际申请的公布数据
WO2014/121102 EN 2014.08.07

(71)申请人 辛博立科伊奥公司
地址 美国新泽西州

(72)发明人 B·伊格诺米瑞罗

(74)专利代理机构 北京市铸成律师事务所 11313
代理人 孟锐

(51)Int.Cl.
G06F 12/00(2006.01)

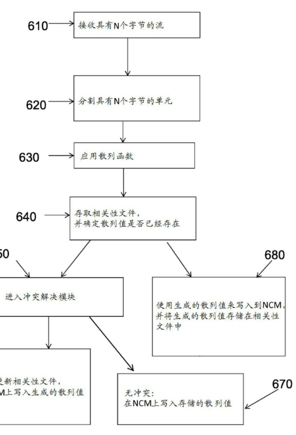
权利要求书7页 说明书32页 附图6页

(54)发明名称

用于存储和检索数据的方法和系统

(57)摘要

通过使用本发明的技术,相关人员能够有效地存储和检索数据。在各种实施方案中,相关人员可以通过编码所述数据以及存储大小上小于起初数据的编码数据,而实现这些效率。



1. 一种用于将数据存储在非高速缓存器记录介质上的方法，所述方法包括：

i. 接收具有 N 个字节的 I/O 流；

ii. 将所述 N 个字节分割成具有 X 个字节的片段单元；

iii. 将加密散列函数应用到具有 X 个字节的每个片段单元，以便针对具有 X 个字节的每个片段单元来形成生成的散列函数值；

iv. 存取相关性文件，其中所述相关性文件使所存储的具有 Y 个比特的散列函数值与多个 X 字节存储序列中的每个序列关联，并且对于每个片段单元而言：

(a) 如果针对具有 X 个字节的片段单元的所述生成的散列函数值是处于所述相关性文件中，那么便使用所述所存储的具有 Y 个比特的散列函数值来在非高速缓存器记录介质上进行存储；以及

(b) 如果针对所述具有 X 个字节的片段单元的所述生成的散列函数值并未处于所述相关性文件中，那么便将所述具有 Y 个比特的生成的散列函数值与所述具有 X 个字节的片段单元一起存储在所述相关性文件中，并使用所述生成的散列函数值来在所述非高速缓存器记录介质上进行存储。

2. 根据权利要求 1 所述的方法，其中在 (iii) 中，所述算法为 MD5 消息摘要算法。

3. 根据权利要求 1 所述的方法，其还包括应用冲突解决模块，其中，如果所述冲突解决模块确定对于具有 X 个字节的片段单元而言，生成与所述相关性文件中的所存储的散列函数值相同的散列函数值，但是所述具有 X 个字节的片段单元不同于与所述所存储的散列函数值关联的所述 X 个字节，那么便将不同的 Z 个比特与所述所存储的散列函数值以及所生成的散列函数值相关联。

4. 根据权利要求 3 所述的方法，其中在所述相关性文件中，对于不存在冲突的所有散列函数值而言，关联 Z 个比特，并且所述 Z 个比特为 8 至 16 个 0。

5. 根据权利要求 4 所述的方法，其中 $N = 4096$ 并且 $X = 512$。

6. 根据权利要求 5 所述的方法，其中 $Y = 128$ 或 256。

7. 根据权利要求 1 所述的方法，其还包括将多个散列函数值存储在所述非高速缓存器记录介质上。

8. 根据权利要求 1 所述的方法，其还包括：通过使用比特标记符表来编码用于存储的每个散列函数值，以便针对用于存储的每个散列函数值而生成比特标记符；以及，将每个比特标记符存储在所述非高速缓存器记录介质中。

9. 根据权利要求 1 所述的方法，其还包括通过使用频次转换器来编码用于存储的每个散列函数值，其中，针对用于存储的每个散列函数值而言，生成转换比特串，其中，对于至少两个不同的 Y 比特串而言，输出的转换比特串具有不同长度。

10. 根据权利要求 9 所述的方法，其中，对于具有不同大小的每批多个转换比特串而言，存在长度为 A 个比特的第一转换比特串以及长度为 B 个比特的第二转换比特串，其中 $A<B$，并且所述第一转换比特串中的所述 A 个比特的身份与所述第二转换比特串中的前 A 个比特的身份并不相同。

11. 根据权利要求 3 所述的方法，其还包括通过使用频次转换器来编码用于存储的每个散列函数值，其中，针对用于存储的每个散列函数值而言，生成转换比特串，其中，对于至少两个不同的 Y 比特串而言，输出的转换比特串具有不同长度。

12.根据权利要求 1 所述的方法，其中所述 I/O 是从主机接收的，并且所述方法还包括向所述主机确认所述 I/O 流的接收。

13.根据权利要求 12 所述的方法，其中所述主机将所述 I/O 记录为存储在第一存储地址上，并且所述转换比特串是存储在所述非高速缓存器记录介质中的第二存储地址上，其中所述第一存储地址与所述第二存储地址并不相同。

14.根据权利要求 13 所述的方法，其中所述第一存储地址对应于具有第一文件大小的文件，并且所述第二存储地址对应于具有第二文件大小的文件，其中所述第一文件大小是所述第二文件大小的至少四倍大。

15.根据权利要求 13 所述的方法，其还包括将所述第二存储地址存储在中介器上，其中所述中介器为非暂时性存储介质。

16.根据权利要求 15 所述的方法，其中所述第二存储地址存储在比特字段中。

17.根据权利要求 16 所述的方法，其中所述中介器包括 :

i. 第一组磁道，其中所述第一组磁道包括文件系统信息、可启动性信息和分区信息 ;

ii. 第二组磁道，其中所述第二组磁道包括所述第一组磁道的复本 ;

iii. 第三组磁道，其中所述第三组磁道包括不同于文件系统信息、可启动性信息和分区信息的元数据 ;以及

iv. 第四组磁道，其中所述第四组磁道包括所述比特字段。

18.一种用于将数据存储在非高速缓存器记录介质上的方法，所述方法包括 :

i. 接收具有 N 个字节的 I/O 流 ;

ii. 将所述 N 个字节分割成具有 X 个字节的片段单元 ;

iii. 使加密散列函数值与具有 X 个字节的每个片段单元关联，其中所述关联包括存取相关性文件，其中所述相关性文件使所存储的具有 Y 个比特的散列函数值与多个 X 字节存储序列中的每个序列关联，并且

(a) 如果所述具有 X 个字节的片段单元的序列是处于所述相关性文件中，那么便使用所述所存储的具有 Y 个比特的散列函数值来在非高速缓存器记录介质上进行存储 ;以及

(b) 如果所述具有 X 个字节的片段单元的序列并未处于所述相关性文件中，那么便将具有 Y 个比特的新散列函数值与所述具有 X 个字节的片段单元一起存储在所述相关性文件中，并使用所述具有 Y 个比特的新散列函数值来在所述非高速缓存器记录介质上进行存储。

19.一种用于将数据存储在非高速缓存器记录介质上的方法，所述方法包括 :

i. 接收具有 N 个字节的 I/O 流 ;

ii. 将加密散列函数值算法应用到具有N个字节的每个单元，以便针对具有N个字节的每个单元来形成生成的散列函数值 ;

iii. 存取相关性文件，其中所述相关性文件使所存储的具有 Y 个比特的散列函数值与多个 N 字节存储序列中的每个序列关联，并且

(a) 如果针对所述具有 N 个字节的单元的所述生成的散列函数值是处于所述相关性文件中，那么便使用所述所存储的具有 Y 个比特的散列函数值来在非高速缓存器记录介质上进行存储 ;以及

(b) 如果针对所述具有 N 个字节的单元的所述生成的散列函数值并未处于所述相关性

文件中，那么便将所述具有 Y 个比特的生成的散列函数值与所述具有 N 个字节的单元一起存储在所述相关性文件中，并使用所述生成的散列函数值来在所述非高速缓存器记录介质上进行存储。

20. 一种用于将数据存储在记录介质上的方法，所述方法包括：

i. 接收多个数字二进制信号，其中所述数字二进制信号是组织在多个小盘中，其中每个小盘的长度为 N 个比特，其中 N 是大于 1 的整数并且其中所述小盘具有某个顺序；

ii. 将每个小盘划分为具有统一大小的子单元，并从一组 X 个标记符中指配标记符给每个子单元，以便形成一组多个标记符，其中 X 小于或等于子单元内不同的比特组合的数目，同样的子单元指配相同的标记符，并且至少一个标记符小于子单元的所述大小；以及

iii. 将所述组的所述多个标记符以与所述小盘的所述顺序相对应的顺序，存储在非暂时性记录介质上。

21. 根据权利要求 20 所述的方法，其中所述指配包括存取比特标记符表，其中在所述比特标记符表内，每个独特标记符识别为对应于独特比特串。

22. 根据权利要求 21 所述的方法，其中每个子单元具有第一末端和第二末端，并且在指配所述标记符之前，所述方法还包括分析每个小盘的每个子单元内的一个或多个比特，以便确定所述第二末端处的比特是否具有数值 0，并且，如果所述第二末端处的所述比特具有数值 0，那么便移除所述第二末端处的所述比特以及具有所述数值 0 且与所述第二末端处的所述比特形成连续比特串的所有比特，从而针对在所述第二末端处具有 0 的任何子单元来形成修正子单元。

23. 根据权利要求 22 所述的方法，其中计算机算法：

(a) 核查每个子单元以便确定在所述第二末端处是否存在 0，且如果存在，便移除所述 0，以便形成具有在邻近于所述子单元的所述第二末端的位置处的修正的第二末端的修正子单元；

(b) 核查每个修正子单元以便确定在所述修正的第二末端处是否存在 0，且如果存在，便移除所述 0，以便形成另外的修正第二末端；以及

(c) 针对每个修正子单元来重复 (b)，直到生成在其第二末端处具有 1 的缩短子单元。

24. 根据权利要求 21 所述的方法，其中每个子单元具有第一末端和第二末端，并且在指配所述标记符之前，所述方法还包括分析每个小盘的每个子单元内的一个或多个比特，以便确定所述第二末端处的所述比特是否具有数值 1，并且，如果所述第二末端处的所述比特具有数值 1，那么便移除所述第二末端处的所述比特以及具有所述数值 1 且与所述第二末端处的所述比特形成连续比特串的所有比特，从而针对在所述第二末端处具有 1 的任何子单元来形成修正子单元。

25. 根据权利要求 24 所述的方法，其中计算机算法：

(a) 核查每个子单元以便确定在所述第二末端处是否存在 1，且如果存在，便移除所述 1，以便形成具有在邻近于所述子单元的所述第二末端的位置处的修正的第二末端的修正子单元；

(b) 核查每个修正子单元以便确定在所述修正的第二末端处是否存在 1，且如果存在，便移除所述 1，以便形成另外的修正第二末端；以及

(c) 针对每个修正子单元来重复 (b)，直到生成在其第二末端处具有 0 的缩短子单元。

26.根据权利要求 21 所述的方法，其中所述标记符存储在频次转换器中，所述标记符具有多个不同大小，并且大小上更小的标记符与更高频次子单元相互关联。

27.根据权利要求 20 所述的方法，其中多个不同标记符是由不同数目的比特形成。

28.一种用于从记录介质检索数据的方法，所述方法包括：

i.访问记录介质，其中所述记录介质以某个顺序存储多个标记符；

ii.将所述多个标记符转译成一组小盘，其中每个小盘的长度为 N 个比特，其中 N 是大于 1 的整数，并且其中所述小盘具有与所述多个标记符的所述顺序相对应的顺序，而且其中所述转译是通过存取比特标记符表来完成，其中在所述比特标记符表内，每个独特标记符识别为对应于独特比特串；以及

iii.生成包括所述组小盘的输出。

29.根据权利要求 28 所述的方法，其中存储在所述记录介质上的所述多个标记符具有 X 至 Y 的大小，其中 Y>X，并且，至少一个标记符具有大小 X 且至少一个标记符具有大小 Y。

30.根据权利要求 29 所述的方法，其中所述转译包括通过将 0 添加到小于长度 Z 的所有标记符的第一末端，而将所述标记符渲染成具有长度 Z 的标记符，其中 Z 大于或等于 Y，并且将所述具有长度 Z 的标记符转译成小盘，其中所述小盘大于长度 Z。

31.根据权利要求 30 所述的方法，其中所述将所述具有长度 Z 的标记符转译成小盘包括，将所述具有长度 Z 的标记符转译成子单元并将所述子单元组合成小盘。

32.一种用于从储存器检索文档的方法，其包括权利要求 28 所述的方法，而且还包括将所述输出与文件类型关联，并将所述输出传输给操作系统，所述操作系统能够将所述小盘转换成所述文件类型的文档。

33.一种用于将数据存储在记录介质上的方法，所述方法包括：

i.接收多个数字二进制信号，其中所述数字二进制信号组织在小盘中，其中每个小盘的长度为 N 个比特，每个小盘具有第一末端和第二末端，N 是大于 1 的整数，并且所述小盘具有某个顺序；

ii.将每个小盘划分为多个子单元，其中每个子单元的长度为 A 个比特；

iii.分析每个子单元以便确定所述第二末端处的比特是否具有数值 0，并且，如果所述第二末端处的所述比特具有数值 0，那么便移除所述第二末端处的所述比特以及具有所述数值 0 且与所述第二末端处的所述比特形成连续比特串的所有比特，从而针对在所述第二末端处具有 0 的任何小盘来形成修正小盘；以及

iv.在非暂时性记录介质上，以所述顺序来存储每个修正子单元以及长度为 A 个比特且在其第二末端处具有 1 的每个子单元。

34.一种用于将数据存储在记录介质上的方法，所述方法包括：

i.接收多个数字二进制信号，其中所述数字二进制信号组织在小盘中，其中每个小盘的长度为 N 个比特，每个小盘具有第一末端和第二末端，N 是大于 1 的整数，并且所述小盘具有某个顺序；

ii.将每个小盘划分为多个子单元，其中每个子单元的长度为 A 个比特；

iii.分析每个子单元以便确定所述第一末端处的比特是否具有数值 0，并且，如果所述第一末端处的所述比特具有数值 0，那么便移除所述第一末端处的所述比特以及具有所述数值 0 且与所述第一末端处的所述比特形成连续比特串的所有比特，从而针对在所述第

一末端处具有 0 的任何子单元来形成第一修正子单元；

iv. 分析每个子单元以便确定所述第二末端处的比特是否具有数值 0，并且，如果所述第二末端处的所述比特具有数值 0，那么便移除所述第二末端处的所述比特以及具有所述数值 0 且与所述第二末端处的所述比特形成连续比特串的所有比特，从而针对在所述第二末端处具有 0 的任何子单元来形成第二修正子单元；以及

v. 对于每个子单元而言：

(a) 如果所述第一修正子单元和所述第二修正子单元的大小是相同的，那么便存储所述第一修正子单元或所述第二修正子单元，

(b) 如果所述第一修正子单元小于所述第二修正子单元，那么便存储所述第一修正子单元，

(c) 如果所述第二修正子单元小于所述第一修正子单元，那么便存储所述第二修正子单元，

(d) 如果不存在修正子单元，那么便存储所述子单元，

(e) 如果不存在第一修正子单元，但是存在第二修正子单元，那么便存储所述第二修正子单元，以及

(f) 如果不存在第二修正子单元，但是存在第一修正子单元，那么便存储所述第一修正子单元，

其中所存储的每个修正子单元是与指示是否从所述第一末端或所述第二末端移除了一个或多个比特的信息一起存储。

35. 一种用于从记录介质检索数据的方法，所述方法包括：

i. 访问记录介质，其中所述记录介质在多个位置上存储多个数据单元，其中每个数据单元含有多个比特且所述数据单元的最大大小为 N 个比特，至少一个数据单元含有少于 N 个比特，并且所述数据单元具有某个顺序；

ii. 检索所述数据单元，并在长度少于 N 个比特的任何数据单元的末端处添加一个或多个比特，以便生成一组对应于所述数据单元的小盘，其中每个小盘含有相同数目的比特；以及

iii. 以与所述数据单元的所述顺序相对应的顺序来生成包括所述组小盘的输出。

36. 根据权利要求 35 所述的方法，其中在 (ii) 中添加具有数值 0 的比特。

37. 一种用于从储存器检索文档的方法，其包括权利要求 36 所述的方法，而且还包括将所述输出与文件类型关联，并将所述输出传输给操作系统，所述操作系统能够将所述小盘转换成所述文件类型的文档。

38. 一种用于存储电子数据的方法，所述方法包括：

i. 接收一组参数，其中所述参数包括文件系统信息、可启动性信息和分区信息中的一个或多个；

ii. 接收元数据；

iii. 接收一个或多个文件，其中每个文件具有文件名；

iv. 将所述参数和元数据存储在中介器上；

v. 将所述文件中的每个文件存储在非高速缓存器介质上的某个位置中；以及

vi. 在所述中介器上，存储每个文件名与所述非高速缓存器介质上的位置的相关性。

39. 根据权利要求 38 所述的方法，其还包括在将所述文件存储在所述非高速缓存器介质上之前，编码所述文件。

40. 根据权利要求 39 所述的方法，其中所述编码包括使用比特标记符表来创建转换文件。

41. 根据权利要求 39 所述的方法，其中所述编码包括使用频次转换器来创建转换文件。

42. 根据权利要求 40 所述的方法，其中所述转换文件并不含有文件系统信息、可启动性信息或分区信息中的任何信息。

43. 根据权利要求 38 所述的方法，其中所述参数存储在多个预留磁道中。

44. 根据权利要求 38 所述的方法，其中所述多个预留磁道为第一组预留磁道，并且所述方法还包括将所述参数复制到第二组预留磁道中。

45. 根据权利要求 44 所述的方法，其还包括使用所述第二组预留磁道来检查所述第一组预留磁道中的错误。

46. 根据权利要求 45 所述的方法，其中所述元数据与用于自动精简配置的指令相对应。

47. 根据权利要求 38 所述的方法，其中所述文件是从主机接收的，所述主机将所述文件记录为存储在虚拟地址上，并且所述虚拟地址与所述文件的所述位置并不相同。

48. 一种用于备份数据的方法，所述方法包括根据权利要求 38 所述的方法，其中所述中介器为第一中介器且所述位置为第一位置，所述方法还包括生成第二中介器，其中所述第二中介器为所述第一中介器在时间 T1 上的复本，并且，在 T1 之后的时间 T2 上：

i. 接收指令来保存位于所述第一位置处的修正文件；

ii. 将新文件存储在第二位置处，其中所述新文件对应于所述修正文件；以及

iii. 更新所述第一中介器以便使所述文件名与所述第二位置相互关联，其中在所述第二中介器上，所述文件名与所述第一位置相互关联。

49. 一种用于备份数据的方法，所述方法包括：

i. 在第一中介器上，使多个文件名与数据文件的多个位置相互关联，其中所述数据文件的所述位置对应于第一非高速缓存器介质上的位置，并且所述第一中介器被配置成允许识别特定文件名的用户来检索对应于所述特定文件名的数据文件；

ii. 将所述多个数据文件复制到第二非高速缓存器介质；

iii. 生成第二中介器，其中所述第二中介器是所述第一中介器在时间 T1 上的复本，并且在所述第二中介器内，所述第二非高速缓存器介质上多个数据文件的位置与所述文件名相互关联；

iv. 接收指令来保存数据文件的修正；以及

v. 在 T1 之后的时间 T2 上，在所述第一非高速缓存器介质中保存所述数据文件的所述修正。

50. 一种数据存储和检索系统，所述系统包括：

i. 非高速缓存器数据存储介质；

ii. 中介器，其中所述中介器是从所述非高速缓存器数据存储介质远程存储，并且所述中介器包括：

(a) 第一组磁道；

(b) 第二组磁道；

(c) 第三组磁道；和

(d) 第四组磁道；以及

iii. 管理器，其中所述管理器被配置来：

(a) 将包括文件系统信息、可启动性信息和分区信息中的一个或多个的数据存储在所述第一组磁道中。

(b) 将元数据存储在所述第三组磁道中；

(c) 将一个或多个文件存储在所述非高速缓存器介质上，其中所述一个或多个文件存储在不含有文件系统信息、可启动性信息和分区信息中的任何信息的所述非高速缓存器介质上；

(d) 将所述非高速缓存器介质中每个文件的位置存储在所述第四组磁道中；以及

(e) 存储所述非高速缓存器介质中每个文件的位置与文件的主机名的相关性。

51. 根据权利要求 50 所述的系统，其中所述管理器还被配置来将所述第一组磁道中的所述信息复制到所述第二组磁道中。

52. 根据权利要求 50 所述的系统，其中所述非高速缓存器介质中每个文件的位置与所述主机认为所述文件所处于的位置并不相同。

53. 根据权利要求 50 所述的系统，其中所述一个或多个文件经过转换而形成转换文件，其中所述转换文件比它们所转自的所述文件占据更少的空间。

54. 根据权利要求 50 所述的系统，其中所述中介器为第一中介器，并且所述系统还包括第二中介器以及用于将所述第一中介器内的所述信息复制到所述第二中介器中的模块。

55. 根据权利要求 54 所述的系统，其中所述第一中介器和所述第二中介器使同一文件名与所述非高速缓存器数据存储介质内的不同位置相互关联。

56. 根据权利要求 54 所述的系统，其中所述非高速缓存器数据存储介质为第一非高速缓存器数据存储介质，且所述系统还包括第二非高速缓存器数据存储介质，并且所述第一中介器使文件名与所述第一非高速缓存器数据存储介质内的位置相互关联，而所述第二中介器使所述文件名与所述第二非高速缓存器数据存储介质内的位置相互关联。

用于存储和检索数据的方法和系统

发明领域

[0001]　本发明涉及数据存储和检索领域。

[0002]　发明背景

[0003]　二十一世纪已经见证了人们和公司生成和存储的数字化信息的量成指数级增长。这类信息由通常存储在磁性表面（如磁盘）上的电子数据组成，所述磁性表面含有尺寸为亚微米并且能够存储数条单独二进制信息的较小区域。

[0004]　由于许多实体生成的大量数据，因此数据存储行业已经转向基于网络的存储系统。这些类型的存储系统可以包括形成处理系统或作为处理系统一部分的至少一个存储服务器，所述处理系统被配置成代表一个或多个实体来存储和检索数据。数据可以作为存储对象（如区块和／或文件）进行存储和检索。

[0005]　用于存储的一个系统是网络附加存储（NAS）系统。在NAS的情境中，存储服务器代表一个或多个客户端进行运作来存储数据和管理数据的文件级存取。文件可以存储在存储系统中，所述存储系统包括海量存储装置（如磁盘或光盘或磁带）的一个或多个阵列。另外，这个数据存储模型可以使用独立磁盘冗余阵列（RAID）技术。

[0006]　用于存储的另一系统是存储区域网络（SAN）。在SAN系统中，通常存储服务器向客户端提供所存储数据的区块级存取，而不是所存储数据的文件级存取。然而，一些存储服务器能够向客户端提供文件级存取和区块级存取。

[0007]　无论相关人员使用NAS还是SAN，所有生成数据的行业都必须考虑存储和检索所述数据的成本。因此，需要新的技术来节约性地存储和检索数据。

发明内容

[0008]　本发明提供用于改善数据存储和检索效率的方法、系统以及计算机程序产品。通过使用本发明的各种实施方案，相关人员可以有效地存储和存取已经可选地转换或编码的数据。另外或替代地，本发明的各种实施方案使用会促进数据有效存储和数据有效存取的中介器（mediator）。

[0009]　本发明的各种实施方案是针对原始数据和／或将元数据与原始数据分离。因此，基于可以结合本发明进行存储和／或检索的文件类型而言，不存在任何限制。可以使用的文件类型的示例包括但不限于JPEG、PDF、WORD文档、MPEG以及TXT文档。借助本发明的各种实施方案，相关人员可以转变数据和／或更改转变后或转换后数据所存储的物理装置。这些实施方案可以借助使用计算机的自动化过程来执行，所述计算机包括或者可操作性地耦接到计算机程序产品，所述计算机程序产品在运行时会使得执行本发明的方法或过程的步骤。这些方法或过程可以（例如）体现在计算机算法或脚本中或包括所述计算机算法或脚本，且可选地由本发明的系统加以执行。

[0010]　根据第一实施方案，本发明针对一种用于将数据存储在记录介质上的方法，所述方法包括：(i) 接收多个数字二进制信号，其中所述数字二进制信号是组织在多个小盘（chunklet）中，其中每个小盘的长度为N个比特，其中N是大于1的整数并且其中所述小

盘具有某个顺序；(ii) 将每个小盘划分为具有统一大小的子单元，并从一组 X 个标记符中指配标记符给每个子单元，以便形成一组多个标记符，其中 X 等于子单元内不同的比特组合的数目，同样的子单元指配相同的标记符，并且至少一个标记符小于子单元的大小；以及 (iii) 将所述组的所述多个标记符以与所述小盘的所述顺序相对应的顺序，或者以允许重新创建所述小盘的所述顺序的另一配置而存储在非暂时性记录介质上。

[0011] 根据第二实施方案，本发明针对一种用于从记录介质检索数据的方法，所述方法包括：(i) 访问记录介质，其中所述记录介质以某个顺序存储多个标记符；(ii) 将所述多个标记符转译成一组小盘，其中每个小盘的长度为 N 个比特，其中 N 是大于 1 的整数，并且其中所述小盘具有与所述多个标记符的顺序相对应的顺序，而且其中所述转译是通过存取比特标记符表来完成，其中在所述比特标记符表内，每个独特标记符识别为对应于独特比特串；以及 (iii) 生成包括所述组小盘的输出。所述标记符可以按照或可以不按照与所述小盘的顺序相同的顺序来存储，但是无论它们存储的顺序如何，相关人员都可以重新创建所述小盘的顺序以及从中获得这些标记符的任何文件。

[0012] 根据第三实施方案，本发明针对一种用于将数据存储在记录介质上的方法，所述方法包括：(i) 接收多个数字二进制信号，其中所述数字二进制信号组织在小盘中，其中每个小盘的长度为 N 个比特，每个小盘具有第一末端和第二末端，N 是大于 1 的整数，并且所述小盘具有某个顺序；(ii) 将每个小盘划分为多个子单元，其中每个子单元的长度为 A 个比特；(iii) 分析每个子单元以便确定所述第二末端处的比特是否具有数值 0，并且，如果所述第二末端处的所述比特具有数值 0，那么便移除所述第二末端处的所述比特以及具有所述数值 0 且与所述第二末端处的所述比特形成连续比特串的所有比特，从而针对在所述第二末端处具有 0 的任何小盘来形成修正小盘；以及 (iv) 在非暂时性记录介质上，以允许按照所述顺序重新构建所述小盘的方式，存储每个修正子单元以及长度为 A 个比特且在其第二末端处具有 1 的每个子单元。举例而言，所述修正子单元（以及任何未修正的子单元）可以在修正之前，按照与每个小盘内的子单元的顺序相对应的顺序进行组织，并且所述数据的存储可以与原始文件内的对应小盘处于相同的顺序中。

[0013] 根据第四实施方案，本发明提供一种用于将数据存储在记录介质上的方法，所述方法包括：(i) 接收多个数字二进制信号，其中所述数字二进制信号组织在小盘中，其中每个小盘的长度为 N 个比特，每个小盘具有第一末端和第二末端，N 是大于 1 的整数，并且所述小盘具有某个顺序；(ii) 分析每个小盘以便确定所述第一末端处的比特是否具有数值 0，并且，如果所述第一末端处的所述比特具有数值 0，那么便移除所述第一末端处的所述比特以及具有所述数值 0 且与所述第一末端处的所述比特形成连续比特串的所有比特，从而针对在所述第一末端处具有 0 的任何小盘来形成第一修正小盘；(iii) 分析每个小盘以便确定所述第二末端处的比特是否具有数值 0，并且，如果所述第二末端处的所述比特具有数值 0，那么便移除所述第二末端处的所述比特以及具有所述数值 0 且与所述第二末端处的所述比特形成连续比特串的所有比特，从而针对在所述第二末端处具有 0 的任何小盘来形成第二修正小盘；(iv) 对于每个小盘而言：(a) 如果所述第一修正小盘和所述第二修正小盘的大小是相同的，那么便存储所述第一修正小盘或所述第二修正小盘，(b) 如果所述第一修正小盘小于所述第二修正小盘，那么便存储所述第一修正小盘，(c) 如果所述第二修正小盘小于所述第一修正小盘，那么便存储所述第二修正小盘，(d) 如果不存在修正小盘，那么

便存储所述小盘，(e) 如果不存在第一修正小盘，但是存在第二修正小盘，那么便存储所述第二修正小盘，(f) 如果不存在第二修正小盘，但是存在第一修正小盘，那么便存储所述第一修正小盘，其中所存储的每个修正小盘是与指示是否从所述第一末端或所述第二末端移除了一个或多个比特的信息一起存储。指示是否从所述第一末端或所述第二末端移除了一个或多个比特的所述信息可以（例如）呈所述子单元的独特性的形式。当生成有待于在任何比较方式中进行比较的所述第一与第二修正小盘中的每个小盘时，优选的是 0 已经从任一末端移除，而不是从两个末端移除。

[0014] 根据第五实施方案，本发明提供一种用于将数据存储在记录介质上的方法，所述方法包括：(i) 接收多个数字二进制信号，其中所述数字二进制信号组织在小盘中，其中每个小盘的长度为 N 个比特，每个小盘具有第一末端和第二末端，N 是大于 1 的整数，并且所述小盘具有某个顺序；(ii) 将每个小盘划分为多个子单元，其中每个子单元的长度为 A 个比特；(iii) 分析每个子单元以便确定所述第一末端处的比特是否具有数值 0，并且，如果所述第一末端处的所述比特具有数值 0，那么便移除所述第一末端处的所述比特以及具有所述数值 0 且与所述第一末端处的所述比特形成连续比特串的所有比特，从而针对在所述第一末端处具有 0 的任何子单元来形成第一修正子单元；(iv) 分析每个子单元以便确定所述第二末端处的比特是否具有数值 0，并且，如果所述第二末端处的所述比特具有数值 0，那么便移除所述第二末端处的所述比特以及具有所述数值 0 且与所述第二末端处的所述比特形成连续比特串的所有比特，从而针对在所述第二末端处具有 0 的任何子单元来形成第二修正子单元；以及 (v) 对于每个子单元而言：(a) 如果所述第一修正子单元和所述第二修正子单元的大小是相同的，那么便存储所述第一修正子单元或所述第二修正子单元，(b) 如果所述第一修正子单元小于所述第二修正子单元，那么便存储所述第一修正子单元，(c) 如果所述第二修正子单元小于所述第一修正子单元，那么便存储所述第二修正子单元，(d) 如果不存在修正子单元，那么便存储所述子单元，(e) 如果不存在第一修正子单元，但是存在第二修正子单元，那么便存储所述第二修正子单元，(f) 如果不存在第二修正子单元，但是存在第一修正子单元，那么便存储所述第一修正子单元，其中所存储的每个修正子单元是与指示是否从所述第一末端或所述第二末端移除了一个或多个比特的信息一起存储。指示是否从所述第一末端或所述第二末端移除了一个或多个比特的所述信息可以（例如）呈所述子单元的独特性的形式。

[0015] 根据第六实施方案，本发明提供一种用于从记录介质检索数据的方法，所述方法包括：(i) 访问记录介质，其中所述记录介质在多个位置上存储多个数据单元，其中每个数据单元含有多个比特且所述数据单元的最大大小为 N 个比特，至少一个数据单元含有少于 N 个比特，并且所述数据单元具有某个顺序；(ii) 检索所述数据单元，并在长度少于 N 个比特的任何数据单元的末端处添加一个或多个比特，以便生成一组对应于所述数据单元的小盘，其中每个小盘含有相同数目的比特；以及 (iii) 以与所述数据单元的所述顺序相对应的顺序来生成包括所述组小盘的输出。

[0016] 根据第七实施方案，本发明针对一种用于存储电子数据的方法，所述方法包括：(i) 接收一组参数，其中所述参数包括文件系统信息、可启动性信息和分区信息中的一个或多个；(ii) 接收元数据；(iii) 接收一个或多个文件，其中每个文件具有文件名；(iv) 将所述参数和元数据存储在中介器上；(v) 将所述文件中的每个文件存储在非高速缓存器介质

上的某个位置中；以及 (vi) 在中介器上，存储每个文件名与所述非高速缓存器介质上的位置的相关性。本发明的中介器可以服务下述目的中的一个或多个目的：(1) 存储用于数据编码的协议；(2) 在记录介质上分配物理空间；(3) 充当主机发起器的磁盘几何排列的中心点；(4) 增加安全性；(5) 允许系统内部构件来登记、读取一个或两个预留 (R_1和 R_2) 并与其进行互动；(6) 提供框架以便采用新的方式来获取快照（也就是，冻结特定时间上所存储的数据）和 / 或克隆磁盘；以及 (7) 提供元数据。即使没有实现所有这些特征，但只要实现其中的一个或多个特征，也可以有助于提升用于存储数据、从未授权存取中保护数据和 / 或检索数据的方法的效率。

[0017] 根据第八实施方案，本发明针对一种用于备份数据的方法，所述方法包括：(i) 在第一中介器上，使多个文件名与数据文件的多个位置相互关联，其中所述数据文件的所述位置对应于第一非高速缓存器介质上的位置，并且所述第一中介器被配置成允许识别特定文件名的用户来检索对应于所述特定文件名的数据文件；(ii) 将所述多个数据文件复制到第二非高速缓存器介质；(iii) 生成第二中介器，其中所述第二中介器是所述第一中介器在时间 T1 上的复本；并且在所述第二中介器内，所述第二非高速缓存器介质上多个数据文件的位置与所述文件名相互关联；(iv) 接收指令来保存数据文件的修正；以及 (v) 在 T1 之后的时间 T2 上，在所述第一非高速缓存器介质上保存所述数据文件的所述修正。优选地，所述修正并非保存在所述第二非高速缓存器介质上的对应数据文件中。

[0018] 根据第九实施方案，本发明提供一种数据存储和检索系统，所述系统包括：(i) 非高速缓存器数据存储介质；(ii) 中介器，其中所述中介器从所述非高速缓存器数据存储介质远程存储，并且所述中介器包括：(a) 第一组磁道；(b) 第二组磁道；(c) 第三组磁道；和 (d) 第四组磁道；以及 (ii) 管理器，其中所述管理器被配置来：(a) 将包括文件系统信息、可启动性信息和分区信息中的一个或多个的数据存储在所述第一组磁道中；(b) 将元数据存储在所述第三组磁道中；(c) 将一个或多个文件存储在所述非高速缓存器介质上，其中所述一个或多个文件存储在不含有文件系统信息、可启动性信息和分区信息中的任何信息的所述非高速缓存器介质上；(d) 将所述非高速缓存器介质中每个文件的位置存储在所述第四组磁道中；以及 (e) 存储所述非高速缓存器介质中每个文件的位置与文件的主机名的相关性。因此，所述管理器可以使得将信息存储在所述中介器上，所述信息可以不存储在所述管理器本身上。

[0019] 根据第十实施方案，本发明针对一种用于将数据存储在非高速缓存器记录介质上的方法，所述方法包括：(i) 接收具有 N 个字节的 I/O 流（通过使用（例如）I/O 协议）；(ii) 将所述 N 个字节分割成具有 X 个字节的片段单元；(iii) 将加密散列函数（其为数值算法）应用到具有 X 个字节的每个片段单元，以便针对具有 X 个字节的每个片段单元来形成生成的散列函数值；(iv) 存取相关性文件，其中所述相关性文件使所存储的具有 Y 个比特的散列函数值与多个 X 字节存储序列中的每个序列关联，并且：(a) 如果针对具有 X 个字节的片段单元的所述生成的散列函数值是处于所述相关性文件中，那么便使用所存储的具有 Y 个比特的散列函数值来在非高速缓存器记录介质上进行存储；以及 (b) 如果针对所述具有 X 个字节的片段单元的所述生成的散列函数值并未处于所述相关性文件中，那么便将所述具有 Y 个比特的生成的散列函数值与所述具有 X 个字节的片段单元一起存储在所述相关性文件中，并使用所述生成的散列函数值来在所述非高速缓存器记录介质上进行存储。

[0020] 根据第十一实施方案，本发明针对一种用于将数据存储在非高速缓存器记录介质上的方法，所述方法包括：(i) 接收具有 N 个字节的 I/O 流；(ii) 将所述 N 个字节分割成具有 X 个字节的片段单元；(iii) 使加密散列函数值与具有 X 个字节的每个片段单元关联，其中所述关联包括存取相关性文件，其中所述相关性文件使所存储的具有 Y 个比特的散列函数值与多个 X 字节存储序列中的每个序列关联，并且：(a) 如果具有 X 个字节的片段单元的序列是处于所述相关性文件中，那么便使用所存储的具有 Y 个比特的散列函数值来在非高速缓存器记录介质上进行存储；以及 (b) 如果具有 X 个字节的片段单元的序列并未处于所述相关性文件中，那么便将具有 Y 个比特的新生成的散列函数值与所述具有 X 个字节的片段单元一起存储在所述相关性文件中，并使用所述具有 Y 个比特的新生成的散列函数值来在所述非高速缓存器记录介质上进行存储。

[0021] 根据第十二实施方案，本发明针对一种用于将数据存储在非高速缓存器记录介质上的方法，所述方法包括：(i) 接收具有 N 个字节的 I/O 流；(ii) 将加密散列函数应用到具有 N 个字节的每个单元，以便针对具有 N 个字节的每个单元来形成生成的散列函数值；(iii) 存取相关性文件，其中所述相关性文件使所存储的具有 Y 个比特的散列函数值与多个 N 字节存储序列中的每个序列关联，并且：(a) 如果针对所述具有 N 个字节的单元的所述生成的散列函数值是处于所述相关性文件中，那么便使用所存储的具有 Y 个比特的散列函数值来在非高速缓存器记录介质上进行存储；以及 (b) 如果针对所述具有 N 个字节的单元的所述生成的散列函数值并未处于所述相关性文件中，那么便将所述具有 Y 个比特的生成的散列函数值与所述具有 N 个字节的单元一起存储在所述相关性文件中，并使用所述生成的散列函数值来在所述非高速缓存器记录介质上进行存储。

[0022] 本发明的各种方法可以（例如）与下述方法结合使用以便配置存储系统来存储电子数据：(i) 接收一组参数，其中所述参数包括文件系统信息、可启动性信息和分区信息中的一个或多个（即便不包括所有这些）；(ii) 接收元数据；以及 (iii) 将所述参数和元数据存储在中介器上。在配置所述存储系统之后，所述系统可以准备：接收一个或多个文件，其中每个文件都具有文件名；将所述文件中的每个文件（可选的是借助上述方法中的一种方法进行处理，以便进行转译）存储在非高速缓存器介质上的位置中；以及，将每个文件名与所述非高速缓存器介质上的位置的相关性存储在所述中介器上。

[0023] 当使用本发明的某些方法时，指令可以存储在编码于非暂时性计算机可读介质上的计算机程序产品上，所述指令可以经过操作而致使数据处理设备来执行操作，所述操作包括：(a) 借助 I/O 协议而从服务器接收具有 N 个字节的 I/O 流；(b) 针对具有 N 个字节的每个单元或者针对多个具有 N 个字节的片段单元来获取散列函数值；以及 (c) 致使存储包括多个散列值或多个转换散列值的数据。通过将所述散列值转换成转换散列值，便可以针对存储文件内信息的未授权存取来提供加强保护。可选地，所述计算机程序产品还包括冲突解决模块，所述冲突解决模块允许识别指配给具有 N 个字节的不同单元或具有 N 个字节的不同片段单元的相同散列函数值，并且致使存储对应于所述冲突散列函数值中的每个值的不同数据。当被运行时，所述计算机程序产品可以致使上述特征的启用和自动运行。这些指令可以存储在一个模块中，或者存储在可操作性地彼此耦接的多个单独模块中。

[0024] 另外，本发明的各种实施方案也可以实施在系统上。本发明的系统的示例包括：(a) 非高速缓存器存储介质（其可以是永久性或非暂时性存储装置）；以及 (b) 一个或多个

处理器，其可经过操作而与所述非高速缓存器存储介质进行互动。可选地，所述系统还包括一个或多个用户界面（例如，图形用户界面），所述用户界面能够允许用户与所述一个或多个处理器进行互动。所述一个或多个处理器还可经过操作而执行本发明的方法中的一个或多个方法，所述方法可以（例如）存储在计算机程序产品上，所述计算机程序产品存储在非暂时性介质上。

[0025] 在一些实施方案中，所述系统包括：(i) 中介器，其中所述中介器是从所述非高速缓存器数据存储介质远程存储，并且所述中介器包括：(a) 第一组磁道；(b) 第二组磁道；(c) 第三组磁道；和 (d) 第四组磁道；(ii) 非高速缓存器介质；以及 (iii) 管理器，其中所述管理器被配置来：(a) 致使将包括文件系统信息、可启动性信息和分区信息中的一个或多个的数据存储在所述第一组磁道中；(b) 致使将元数据存储在所述第三组磁道中；(c) 致使将一个或多个文件存储在所述非高速缓存器介质上，其中所述一个或多个文件是存储在在不含有文件系统信息、可启动性信息和分区信息中的任何信息的所述非高速缓存器介质上；(d) 致使将所述非高速缓存器介质中每个文件的位置存储在所述第四组磁道中；以及 (e) 致使存储所述非高速缓存器介质中每个文件的位置与文件的主机名和 / 或地址的相关性。

[0026] 借助本发明的各种实施方案，相关人员可以提高存储和检索数据的效率。可以通过比普遍使用的方法使用更小的存储空间以及在存储信息的活动中投入更少的时间和努力，而实现效率的提高。在一些实施方案中，相关人员也可以针对数据文件的未授权检索来提高保护。这些益处可以在远程或本地存储数据时加以实现，并且本发明的各种实施方案可以与 RAID 技术结合使用或者独立于所述 RAID 技术来使用。

[0027] 附图简述

[0028] 图 1 为本发明系统的概况的表示图。

[0029] 图 2 为中介器和非高速缓存器介质 (NCM) 的表示图。

[0030] 图 3 为使用中介器来存储信息的系统的表示图。

[0031] 图 4 为使用两个中介器来备份所存储的信息的系统的表示图。

[0032] 图 5 为二进制树的表示图。

[0033] 图 6 为表示本发明的方法的流程图。

[0034] 发明详述

[0035] 现将详细参考本发明的实施方案，其示例在附图中予以说明。在以下详细描述中，阐述了许多具体细节以提供对本发明的充分理解。然而，除非上下文中另有指示或暗示，否则这些细节都是作为示例，并且不应该视为以任何方式来限制本发明的范围。另外，标题是为了方便读者而使用的，而不是意在限制本发明实施方案中的任何实施方案的范围，也不是意在表明任何特征是限于特定标题内的章节。

[0036] 定义

[0037] 除非上下文中另有陈述或暗示，否则下述术语和短语都具有下文所提供的意义。

[0038] 术语“比特”是指二进制数字。其可以具有两个数值中的一个数值，这个数值可以由 0 或 1 表示。

[0039] 术语“区块”是指具有预定长度的数据字节或比特序列。

[0040] 短语“可启动性代码”、“可启动性信息”和“可启动性特征”是指提供相应方式来

进入可启动状态并且可以存储在启动扇区上的信息。启动扇区可以含有机器代码，所述机器代码被配置成由固件加载到RAM（随机存取存储器）中，这又允许启动进程来从存储装置加载程序或将程序加载到存储装置上。举例而言，主启动记录可以含有定位现用分区并调用卷启动记录的代码，其可以含有代码来加载和调用操作系统或其它独立程序。

[0041]　术语“字节”是指八比特序列。

[0042]　术语“高速缓存器”是指为了将来更快提供数据的请求或出于缓冲的目的而临时存储数据的位置。L1 高速缓存器（1 级高速缓存器）是指（例如）与处理器核心集成的静态存储器。L1 高速缓存器可以用来在 CPU（中央处理单元）多次存取相同数据的情况下提升数据存取速度。L2 高速缓存器（2 级高速缓存器）通常比 L1 高速缓存器更大，并且，如果在 L1 高速缓存器中寻找但并未发现数据文件，那么便可以在查看外部存储器之前对 L2 高速缓存器进行搜索。在一些实施方案中，L1 高速缓存器并不处于中央处理单元内。相反，其可以位于 DDR、DIMM 或 DRAM 内。另外或替代地，L2 高速缓存器可以是 PCI2.0/3.0 的一部分，所述 PCI2.0/3.0 并入到主板中。因此，L1 高速缓存器和 L2 高速缓存器中的每个高速缓存器可以处于主板的单独部分中。就大小而言，在本发明的一些实施方案中，L1 高速缓存器介于 2GB 与 128TB 之间或者介于 2GB 与 4TB 之间；而 L2 高速缓存器介于 16GB 与 1PB 之间或者介于 16GB 与 3.2TB 之间。在一些实施方案中，当实施本发明的方法时，比特标记符表、频次转换器（frequency converter）或散列数值表存在于 L2 高速缓存器中。

[0043]　术语“小盘”是指可以对应于扇区群集的一组比特。小盘的大小由存储系统确定，并且可以具有某个小盘大小。传统上，小盘大小是通过 CHS 方案得出，所述 CHS 方案会经由元组来定址区块，所述元组定义所述区块在硬盘上出现的柱面、磁头和扇区。更近地，小盘大小已经从 LBA 测量中得出，所述 LBA 测量是指逻辑区块定址，并且作为另一种方式用以指定存储在计算机存储装置上的数据区块的位置。举例而言，小盘大小可以是 512B、1K、2K、4K、8K、16K、32K、64K 或 1MB。正如本领域普通技术人员所知晓的，1K ＝ 1024B。小盘可以从主机作为原始数据进行接收。

[0044]　“文件”是具有某个长度的相关字节或比特的集合。文件可以小于小盘、与小盘具有相同大小或者大于小盘。

[0045]　短语“文件名”是指允许计算机识别特定文件并将所述文件与其它文件加以区别的名称或代码。

[0046]　短语“文件系统”是指用来存储、检索和更新一组文件的抽象概念。因此，文件系统是用来管理数据和文件的元数据以及含有数据的存储装置上的可用空间的存取的工具。一些文件系统可以（例如）存在于服务器上。文件系统的示例包括但不限于 Unix 文件系统和其关联的目录表和索引节点、Windows FAT16 和 FAT32 文件系统（FAT 是指文件分配表）、Windows NTFS（其基于主文件表）以及 Apple Mac OSX（其使用 HFS 或 HFS plus）。

[0047]　短语“散列函数”、“加密散列函数值算法”和“散列函数值算法”是指针对特定散列函数而将较大数据组（可选地具有可变长度）映射到具有固定长度的更小数据组的算法或子程序。“散列函数值”是指在应用散列函数算法之后返回的输出。所述算法返回的数值也可以称为散列值、散列代码、散列和、校验和或散列数据。当（例如）使用 MD5 时，输出为 128 个比特，然而，当使用 SHA-1 时，输出便为 160 个比特。

[0048]　术语“主机”和“发起器”可以互换使用，并且是指发送数据以便存储到本发明的

数据存储和检索中介系统上的实体或系统。主机可以发送与一种或多种类型的文档或文件以及接收的数据相对应的数据。优选地，在任何输入 / 输出（“I/O”）流内，数据对应于单个文档类型的文件。出于便利性目的，短语“I/O 流”用来指代数据从一个实体传输到另一实体。第一实体的输出内容可以是第二实体的输入。

[0049] 缩写“LBA”是指逻辑区块定址。LBA 是线性定址方案，并且是用来指定存储在某些存储介质（例如，硬盘）中的数据区块位置的系统。在 LBA 方案中，区块是通过整数来定位，并且只有一个数字用来定址数据。通常，第一区块为区块 0。

[0050] 缩写“LUN”是指逻辑单元数字，并且是用来识别逻辑单元的数字。LUN 通常用来管理 SAN 上共享的区块存储阵列。

[0051] 术语“管理器”是指可以存储在非暂时性介质中并且致使采取一个或多个其它动作（例如，接收、传输、存储或处理数据）的计算机程序产品，例如，代码。管理器可以存储在硬件、软件或其组合上。在一些实施方案中，管理器可以是计算机和 / 或系统的一部分，所述计算机和 / 或系统被配置成允许管理器执行其预期功能。

[0052] 术语“中介器”是指可以存储在硬件、软件或其组合上并且使至少一个非高速缓存器介质内存储空间的一个或多个单元与文件名相互关联的计算机程序产品。中介器可以是比其所指向的非高速缓存器介质更小的量级。举例而言，中介器可以大约小到典型柱面大小的约 0.2％。在一些实施方案中，中介器可以存在于计算云中，然而在其它实施方案中，其存在于非暂时性有形记录介质中。中介器能够组织、转换、转译并控制主机察觉到是处于记录介质的某些磁道中而实际上出现在记录介质的不同磁道中的位置中的数据存储，或者中介器可以可操作性地耦接到管理器，所述管理器即便不会提供所有这些功能，也会提供其中的一个或多个功能。此外，中介器可以包括可位于物理装置或结构内的扇区图、表或其它数据组织，且因此，中介器的内容可以致使物理装置或结构具有某种几何排列。在一些实施方案中，中介器永久性地存在于 L2 高速缓存器上。在其它实施方案中，其存在于固态驱动器上。

[0053] 术语“元数据”是指关于数据容器的管理信息。元数据的示例包括但不限于：正在读取的文件的长度或字节计数；与文件修改的最后一次有关的信息；描述文件类型和存取许可的信息；以及 LUN QoS、VM 和 WORM。其它类型的元数据包括操作系统信息、自动初始化信息、分组许可以及文档类型内比特的频次。在一些实施方案中，存储的元数据可以（例如）用来允许有效地紧缩或扩展发起器的存储空间，因为发起器力图存储的文档的数目和大小在缩减或增加。

[0054] 缩写“NAS”是指网络区域存储。在 NAS 系统中，磁盘阵列可以连接到针对局域网络传送而给予访问权限的控制器。

[0055] 短语“可操作性地耦接”意味着系统、装置和 / 或模块被配置成彼此或互相通信，并且能够在通信时或通信之后执行它们的预期目的。

[0056] 短语“操作系统”是指管理计算机硬件资源的软件。操作系统的示例包括但不限于 Microsoft Windows、Linux 和 Mac OS X。

[0057] 术语“分区”是指将存储介质（例如，磁盘驱动器）划分为多个单元的格式。因此，分区也可以称为磁盘分区。分区的示例包括但不限于 GUID 分区表和 Apple 分区图。

[0058] 缩写“RAID”是指独立磁盘冗余阵列。对于相关服务器而言，这组磁盘可能看上去

像单个卷。RAID 技术通过从多个磁盘牵引单条数据来提升性能，并且建立在一种或多种前提类型上，诸如 :(1) 数据的镜像 ;(2) 数据的条状化 ;或者 (3) 数据镜像和条状化的组合。

[0059]　术语“记录介质”是指非暂时性有形计算机可读存储介质，在其中相关人员可以存储对应于比特的磁性信号。举例而言，记录介质包括但不限于非高速缓存器介质，如硬盘和固态驱动器。正如本领域普通技术人员所知道的，固态驱动器也具有高速缓存器，并且不需要自旋。非暂时性有形计算机可读存储介质的示例包括但不限于硬驱、硬盘、软盘、计算机磁带、ROM、EEPROM、非易失性 RAM、CD-ROM 以及穿孔卡。

[0060]　缩写“SAN”是指存储区域网络。这种类型的网络可以用来将计算装置链接到磁盘、磁带阵列以及其它记录介质。数据可以（例如）在 SAN 上以区块形式进行传输。

[0061]　缩写“SAP”是指系统辅助处理器，其是操作系统所使用的 I/O（输入 / 输出）引擎。

[0062]　缩写“SCSI”是指小型计算机系统接口。

[0063]　术语“扇区”是指磁盘（例如，磁性磁盘）上磁道的细分。每个扇区存储固定量的数据。针对磁盘而言的普通扇区大小为 512 个字节 (512B)、2048 个字节 (2048B) 和 4096 个字节 (4K)。如果小盘的大小为 4K 且每个扇区的大小为 512B，那么每个小盘对应于 8 个扇区 (4*1024/512 ＝ 8)。扇区具有磁道并且位于盘片上。通常，两个或四个盘片构成一个柱面，并且 255 个柱面构成硬盘和介质装置。

[0064]　短语“扇区图”是指从主机接收调用信号并关联存储装置中文件所存储的位置的工具。扇区图可以（例如）在 SCSI 协议定义的参数下进行运作。在本发明的一些实施方案中，扇区图可以位于中介器的比特字段中。

[0065]　术语“磁道”是指磁盘内横跨所有扇区的圆形单元。“磁道扇区”为任何一个扇区内的磁道。“磁道群集”跨越一个以上扇区。

[0066]　优选实施方案

[0067]　本发明提供用于将数据存储在非高速缓存器记录介质上的方法、用于执行这些方法的计算机程序产品以及被配置来执行这些方法的系统。借助本发明的各种实施方案，相关人员可以有效地存储和检索数据。

[0068]　比特标记符

[0069]　根据一个实施方案，本发明针对一种用于将数据存储在记录介质上的方法，所述方法使用比特标记符作为原始数据内比特串的表示形式。因此，原始数据被转译成一系列表示所述原始数据的标记符，并且所述方法提供接收含有会被转换成一组信号（可以称为比特标记符）而用于存储的数据的文件或流，并且存储这些信号。

[0070]　文件可以从称为主机的个人或实体进行接收。优选地，主机将以原始数据的形式来发送信号。举例而言，主机可以发送单独或共同形成一个或多个文件（如 JPEG、PDF、TIFF 或 WORD 文档）的一个或多个小盘。本发明的各种实施方案开始于接收小盘、接收小盘的子单元或者接收有待转换成小盘的一个或多个文件。优选地，所接收的小盘或小盘的子单元都具有统一大小。可选地，所述方法验证大小的统一性，并且如果检测到不统一性，例如，一个或多个小盘具有过少的比特，那么所述方法便添加 0，直到所有小盘都具有统一大小。

[0071]　举例而言，数据可以在长度为 N 个比特的小盘中进行接收，其中 N 是大于 1 的整数。举例而言，N 可以是 128 个字节至 16K，例如，4K，这对应于 4096B。

[0072]　小盘是按顺序接收的。举例而言，文件可以含有系统连续接收的十个小盘。或者，

针对给定文件的多个小盘可以并行或一起传输，条件是它们含有允许其以合适方式彼此重新关联的信息，所述方式允许通过主机的操作系统重新创建和使用所述文件。因此，在一些实施方案中，本发明的方法以接收小盘的相同顺序来存储标记符。因此，当主机想要检索文件时，对应的检索方法论将会以相同顺序来调回编码数据，并按照合适顺序将所述编码数据解码成小盘。

[0073]　可选地，在编码之前，系统可以将小盘划分为比特群组，也称为子单元，其中每个子单元的长度为 A 个比特。如果系统将小盘划分为子单元，那么这些子单元可以与比特标记符表进行比较。如果系统并未将小盘划分为子单元，那么每个小盘便可以与比特标记符表进行比较。

[0074]　比特标记符表使独特组的比特与独特标记符相互关联。在一些实施方案中，比特标记符表含有标记符而用于使用子单元时具有大小 A 的每个独特比特串，或不使用子单元时具有大小 N 的每个独特比特串。因此，在这种方法下，计算机程序可以接收一组小盘作为输入。所述计算机程序随后可以将每个小盘划分为具有相同大小且每个的长度为 A 个比特的 Y 个子单元，其中 A/8 是整数。对于每个独特的 A 而言，可以在所述表内存在标记符。

[0075]　因此，借助自动化协议，在接收小盘之后，计算机程序产品会致使存取比特标记符表。因此每个小盘或子单元可以充当输入，并且每个比特标记符可以充当输出，从而形成输出的标记符组。所述输出的标记符组可以称为转译数据、译码数据或编码数据。在每个小盘并未再分的实施方案中，每个小盘会接收一个标记符。如果小盘划分为两个子单元，那么所述小盘会转译或编码成两个标记符。因此，计算机程序产品使用会将标记符与输入相互关联的比特标记符表，以便指配对应于每个小盘的至少一个标记符。计算机程序产品可以经过设计，从而使得生成对应于每个单独标记符的不同输出、生成含有一组对应于每个小盘的标记符的不同输出或者生成含有对应于完整文件的所述组标记符的不同输出。

[0076]　比特标记符表含有 X 个标记符。在一些实施方案中，X 等于长度为 N 的小盘内不同的比特组合的数目，条件是所述方法并未将小盘划分为子单元，或者 X 等于长度为 A 的子单元内不同的比特组合的数目，条件是所述方法划分小盘。如果文档类型已知或预期具有比针对给定长度子单元或小盘的所有比特组合少，那么 X（标记符的数目）可以小于可能的比特组合的实际数目。举例而言，在一些实施方案中，所有比特标记符都具有相同大小，并且比特标记符表内的比特标记符的数目等于大小为 N 或 A 的比特串内的比特组合的数目。在其它实施方案中，所有比特标记符都具有相同大小，并且比特标记符表内的比特标记符的数目小于大小为 N 或 A 的比特串内的比特组合数目的 90％、小于其 80％、小于其 70％或者小于其 60％。

[0077]　举例而言，在一些是实施方案中，每个小盘指配有由多个 0 和 / 或 1 组成的代码（也就是标记符）。在其它实施方案中，每个小盘划分为多个子单元，其中每个子单元指配有由多个 0 和 1 组成的代码（也就是标记符）。子单元可以由长度 A 界定，其中 N/A ＝ Y 且 Y 为整数。如果任何子单元并不具有此数目的比特，例如，一个或多个子单元比系统经过配置而接收为输入的比特具有更小数目的比特，那么系统便可以添加比特，例如 0，直到所有子单元具有相同大小。此步骤可以（例如）在小盘划分为子单元之后，并且在没有首先进行检查来查看是否所有小盘都具有相同大小的情况下执行。或者，且正如上文所描述的，此步骤可以在将小盘划分为子单元之前，在小盘层面上执行。

[0078] 正如上文的描述所表明的，所述算法可以被配置来将比特串转译成一组编码数据，并且所述算法可以经过设计，从而使得比特串对应于小盘或对应于小盘的子单元。优选地，所述组编码数据小于从主机或客户端所接收的文件。然而，无论所述组编码数据是否小于起初数据，所述组编码数据都能够转回成文件的小盘。正如本领域普通技术人员会认识到的，从主机接收而用于存储的数据将是原始数据，并且因此可以对应于任何文档类型。

[0079] 编码可以服务于两个独立目的。第一，通过编码用于存储的数据，会增强安全性。只有知道代码（也就是对比特标记符表具有访问权限）的个人或实体才能够对所述数据进行解码并且重新构建文档。第二，如果使用比起初文档更少的比特来创建代码，那么便会需要更少的存储空间，并且可以节约成本。如果用于一个或多个文件的存储空间量减小，那么NCM便可以存储对应于更大数目文件和/或更大文件的数据。

[0080] 对于表内的至少多个独特比特组合而言，优选地，如果系统并未将小盘划分为子单元，那么标记符便小于小盘长度N，或者，如果系统确实将小盘划分为子单元，那么标记符便小于子单元长度A。优选地，如果系统并未将小盘划分为子单元，那么不会有任何标记符大于小盘长度N，或者，如果系统确实将小盘划分为子单元，那么不会有任何标记符大于子单元长度A。在一些实施方案中，所有标记符都小于N或者小于A。另外，在一些实施方案中，每个标记符都可以具有相同大小，或者两个或两个以上标记符可以具有不同大小。当存在具有不同大小的标记符时，这些不同大小的标记符可以（例如）处于所述表中。或者，在所述表内，所有标记符都具有相同大小，但是在存储之前，从这些标记符的一个或两个末端移除所有的0。

[0081] 在计算机程序产品将小盘转译成多个标记符之后，其会致使将所述多个标记符（已经或尚未从末端移除0）以与小盘的顺序相对应的顺序存储在非暂时性记录介质上，或者以另一顺序存储在非暂时性记录介质上，其中可以根据此顺序按照其他方式重新创建小盘的顺序。最后，标记符将会存储在非暂时性介质中，所述介质为非高速缓存器介质。然而，可选地，这些标记符可以首先发送给高速缓存器介质，例如L1和/或L2。因此，为了不损失任何信息，比特标记符表可以存储在永久性存储器中，但是在使用时，复本可以处于（例如）L2高速缓存器中，并且在启动或重启后，L2高速缓存器中表的填写可以来自永久性存储器。

[0082] 在一个实施方案中，存储装置将对应于给定文件的多个比特标记符存储在非高速缓存器介质中。这些比特标记符的大小范围为X至Y，其中X小于Y并且至少两个标记符具有不同大小。在检索比特标记符时或之后，计算机算法将0添加到所有小于预定大小Z的比特标记符的一个末端，其中Z大于等于Y。可以查询查找表，其中大小为Z的每个标记符都转译成长度为A的比特串，其中A大于或等于Z。在非限制性示例中，X＝4、Y＝20、Z＝24、A＝32。在一些实施方案中，A至少比Z大50％。对应于A的比特串可以是组合成小盘的子单元，或者它们本身可以是小盘。

[0083] 频次转换器

[0084] 正如上文所描述的，比特标记符表可以针对原始数据以随机或非随机的方式将标记符指配给比特串，并且比特标记符可以具有统一或不统一的大小。然而，相关人员可以不使用如上文所述的比特标记符表，而使用频次转换器。

[0085] 正如本领域普通技术人员会认识到的，本公开的示例性章节中的表I和表II以独

立于原始数据中比特串的频次的方式来指配比特标记符（也就是转换比特）。然而，正如上文所提及的以及下文在示例 3 中所解释的，相关人员可以将更小的标记符指配给预期更为频繁地出现在文档类型或一组文档中的原始数据。这个策略利用如下事实：全部信息的大约 80％是包含在最频繁子单元的大约最靠前的 20％内。换句话来说，对应于数据的子单元是高度重复的。

[0086] 为了进一步说明频次转换器可以如何使用具有不同大小的比特串，可以参考图 5。图 5 展示针对长度为五个单元并且开始于数字 1 的所有二进制序列的二进制树。正如所述树展示的，存在 16 个路径来创建长度为五个数字的序列，这些路径开始于顶行并沿着树向下移动到底部。然而，当沿着树向下移动时，相关人员可以沿着分支向下移动到（例如）第三、第四或第五行。因此，所述方法可以经过设计，从而使得对于一条数据而言，所述方法指配对应于部分沿着树向下移动的代码，然而，对于其它条的数据而言，所述方法指配对应于沿着不同分支移动更大数目单元的代码。举一个非限制性示例，对于特定条的转换数据而言，相关人员可以停止于第三行上，例如，生成序列 101，然而，对于所有其它条的数据而言，前三个数字不是 101。因此，所述方法从不允许使用：1010、1011、10100、10101、10110 和 10111（这些处于图 5 中的方框内）。在从存储器检索数据后，系统会读取数字，直到其将这些数字辨识为独特的，并且在将它们辨识为独特后，所述系统会使用独特串作为容许解码的程序的输入，这会允许为主机重新创建原始数据文件。

[0087] 本发明的各种方法可以在检索过程中或检索之后要求读取最小数目的比特，并在这个最小数目（其对应于最短独特代码）之后，通过与数据文件或表进行比较来检查独特性，而且，如果代码在数据文件或表内不是独特的，那么所述方法便可以继续扩展比特的数目，并且在添加每个单独比特之后检查所扩展代码的独特性直到序列被识别为独特的。前述示例是就 3 比特、4 比特或 5 比特串来描述的，但是这些大小仅仅用于说明性目的。在一些实施方案中，与频次转换器结合使用的最短独特比特序列的长度为 10 至 15 个比特，并且最长比特序列的长度为 25 至 40 个比特。

[0088] 在一些实施方案中，对于具有不同大小的每批多个转换比特串而言，存在长度为 A 个比特的第一转换比特串以及长度为 B 个比特的第二转换比特串，其中 $A<B$，并且，第一转换比特串的 A 个比特的身份与第二转换比特串的前 A 个比特的身份并不相同。

[0089] 正如表 I、表 II 和表 III（参看示例）所展示的，信息可以进行转换，并且输出代码可以被配置成比输入占据更少的空间，原因在于标记符用来表示比特群组。因此，优选地在表内，标记符中的至少一个标记符、多个标记符、至少 50％、至少 60％、至少 70％、至少 80％、至少 90％或至少 95％在大小上小于子单元。然而，不存在任何技术障碍来使得转换数据与从主机所接收的数据或根据散列函数值算法而产生的数据具有相同大小或比其更大。

[0090] 分割

[0091] 如上文所阐明的，在接收数据流之后，在一些实施方案中所述方法需要分割所接收的数据。如果所接收的数据呈较大文件的形式，那么所述数据可以分割成小盘。如果所接收的数据呈小盘的形式，那么小盘可以分割成子单元。较大文件可以首先分割成小盘，且然后这些小盘可以分割成子单元。因此，对于来自主机的任何输入而言，可能存在零个、一个或两个分割步骤。

[0092] 通过分割 I/O 流，所述方法将每个 I/O 流划分为具有预定大小且优选地具有统一大小的更小单元（其也可以称为区块）。正如本领域普通技术人员会认识到的，使用更小区块大小的一个益处在于，较小区块容许更轻易地进行数据处理。举一个非限制性示例，I/O 流的长度可以是 4096 个字节，并且所述方法可以将这个流分割成长度为 1024、512、256 或 128 个字节的片段单元。在接收数据之后或者数据正被接收时，可以运行某个算法，所述算法将小盘划分为（例如）具有 32 个比特的子单元。

[0093] 子单元的大小是从主机接收数据的系统的设计者的一个选择。然而，子单元的大小应经过选择，使得将小盘划分为具有一致大小的子单元，并且子单元可以轻易地与比特标记符表或频次转换器的查询结合使用。

[0094] 如果小盘中的任何小盘小于其它小盘，那么可选地，在接收具有更小大小的那个小盘后，所述算法便添加 0，从而将所述更小的小盘渲染成与其它小盘具有相同大小。或者，系统可以将小盘划分为子单元，并且在获取小于期望长度的子单元后，将 0 添加到这个子单元的末端。

[0095] 预处理

[0096] 在转译过程（其也可以称为编码过程）期间，算法针对比特标记符表或频次转换器而用作输入的比特串（也就是小盘或子单元）可以进行预处理。这些比特串中的每个比特串都可以由第一末端和第二末端界定，并且在指配标记符之前，所述方法还包括分析每个比特串，以便确定第二末端处的比特是否具有数值 0。如果第二末端处的比特具有数值 0，那么所述方法便可以移除第二末端处的比特以及具有数值 0 且与第二末端处的比特形成连续比特串的所有后续比特，从而形成大小上减小的比特串。预处理步骤的益处在于，可以使用更小的比特标记符表或频次转换器。举例而言，可以不使用表 I，而是使用表 II 来产生相同的编码数据。正如本领域普通技术人员会认识到的，这个预处理可以通过从第二末端搜索和移除 1 而不是 0 来完成。

[0097] 截断

[0098] 在本发明的另一实施方案中，相关人员不是通过使用比特标记符表或频次转换器来转译，而是可以按照截断子单元在小盘内存在的相同顺序来存储截断子单元（或者，如果不使用子单元，那么可以截断和存储小盘）。因此，不是使缩短的比特串充当预处理数据来用作（例如）比特标记符表中的输入，而是可以存储比特串本身。

[0099] 因此，在一些实施方案中，存在另一种用于将数据存储在记录介质上的方法。根据这种方法，相关人员接收多个数字二进制信号，其中所述数字二进制信号是组织在呈上文所述格式的小盘中。可选地，每个小盘可以划分为子单元，正如上文所提供的。

[0100] 每个小盘或子单元可以由其长度界定，并且每个小盘或子单元具有第一末端和第二末端。相关人员可以分析每个小盘或子单元以便确定所述第二末端处的比特是否具有数值 0，并且，如果第二末端处的比特具有数值 0，那么便移除第二末端处的比特以及具有数值 0 且与第二末端处的该比特形成连续比特串的所有比特，从而针对在第二末端处具有 0 的任何小盘或子单元来形成修正小盘或修正子单元。因为这些修正比特串比起初比特串更短，所以它们可以称为是截断的。

[0101] 在截断小盘或子单元之后，相关人员可以将截断的信息存储在非暂时性记录介质中。通过存储截断的信息，使用更少的比特来存储相同的信息，所述信息会以其它方式存储

在未截断的比特串中。

[0102] 在上文所描述的各种方法中，相关人员可以从每个子单元或每个小盘的第一末端或第二末端移除数字，但是不会从这两个末端同时移除。然而，实践下述方法是处于本发明的各种实施方案的范围内：相关人员考虑从每个子单元或小盘的第一末端移除数字；相关人员单独考虑从每个子单元或小盘的第二末端移除数字；对于每个子单元或小盘而言，相关人员分析截断发生在第一末端和第二末端中的任一末端、一个末端还是两个末端处，并且，如果截断只发生在一个末端处，那么便保存截断的小盘或子单元，而如果截断发生在两个末端处，那么便保存截断单元中更小的单元。实践可以从小盘或子单元的两个末端处移除数字的方法，也是处于本发明的范围内。

[0103] 因此，相关人员可以接收多个数字二进制信号。二进制信号可以在单元（例如，小盘或者小盘的子单元）中进行接收。每个单元的长度可以是相同数目的比特，并且每个单元具有第一末端和第二末端。单元内的比特数目为大于 1 的整数，并且这些比特在单元内具有某个顺序，而这些单元也具有某个顺序。

[0104] 相关人员随后可以分析每个单元，从而确定第一末端处的比特是否具有数值 0，并且，如果第一末端处的比特具有数值 0，那么便移除第一末端处的比特以及具有数值 0 且与这个比特形成连续比特串的所有比特，从而针对在第一末端处具有 0 的任何单元来形成第一修正单元。

[0105] 相关人员也可以分析每个单元以便确定第二末端处的比特是否具有数值 0，并且，如果第二末端处的比特具有数值 0，那么便移除第二末端处的比特以及具有数值 0 且与这个比特形成连续比特串的所有比特，从而针对在第二末端处具有 0 的任何单元来形成第二修正单元。

[0106] 对于每个单元而言，可以应用下述决策树：(a) 如果第一修正单元和第二修正单元的大小是相同的，那么便存储第一修正单元（或所述第二修正子单元）；(b) 如果第一修正单元小于第二修正单元，那么便存储第一修正单元；(c) 如果第二修正单元小于第一修正单元，那么便存储第二修正单元；(d) 如果不存在修正单元，那么便存储所述单元；(e) 如果不存在第一修正单元，但是存在第二修正单元，那么便存储第二修正单元；以及 (f) 如果不存在第二修正单元，但是存在第一修正单元，那么便存储第一修正单元。

[0107] 相关人员也可以存储指示是否从第一末端或第二末端移除了一个或多个比特的信息，或者，相关人员可以将第一比特标记符表用于从第一末端移除比特的单元，并将第二比特标记符表用于从第二末端移除比特标记符的单元，而且在这两个比特标记符表之间，不存在比特标记符的任何复制形式。这两个不同的比特标记符表可以组织为同一张表的区段，并且包括用于未修正的单元的比特标记符。在所述一张或多张表中，不存在比特标记符的任何复制形式来用于第一修正单元、第二修正单元以及任何未修正的单元，因为（例如）它们在两个末端上都具有数值 1。

[0108] 正如本领域普通技术人员会认识到的，尽管上文所描述的方法是结合移除 0 来描述的，但是系统也可以替代移除 1。另外，这些方法被描述成是用作使用比特标记符表或频次转换器的方法的替代方法。然而，将这些方法与比特标记符表或频次转换器结合使用也是处于本发明的范围内，例如，通过截断作为那些协议的输出的标记符，只要它们是以允许检索和重构起初文件的方式存储的。

[0109]　散列值

[0110]　在上文所描述的实施方案中的某些实施方案中，相关人员存取比特标记符表或频次转换器来编码数据。在另一实施方案中，小盘或子单元充当加密散列函数值算法的输入。

[0111]　因此，对于每个输入而言，存在大小上小于所述输入的输出。举例而言，如果存在具有X个字节的子单元（其也可以称为具有X个字节的片段单元），那么输出便是针对每个具有X个字节的片段单元的生成的散列函数值。散列函数值算法的示例包括但不限于MD5散列算法（也称为消息摘要算法）、MD4散列算法和SHA-1。从散列函数值算法中输出的值可以称为校验和或和。在一些实施方案中，所述和大小为64、128、160或256个比特。由于I/O流内数据高度重复的特性，生成冲突和（也就是，相同的但对应于不同片段单元的和）的可能性便相对较低。

[0112]　在一些实施方案中，每个散列值是由多个0和/或1组成的、优选地具有相同长度的比特串或代码。这些散列值可以用来在非高速缓存器记录介质上进行存储。

[0113]　在一个实施方案中，系统可以（例如）执行一种用于将数据存储在非高速缓存器记录介质上的方法，所述方法包括：(i)接收具有N个字节的I/O流；(ii)将所述N个字节分割成具有X个字节的片段单元；(iii)使加密散列函数值与具有X个字节的每个片段单元关联，其中所述关联包括存取相关性文件，其中所述相关性文件使所存储的具有Y个比特的散列函数值与多个X字节存储序列中的每个序列相互关联，并且：(a)如果具有分割的X个字节的序列是处于所述相关性文件中，那么便使用所存储的具有Y个比特的散列函数值来在非高速缓存器记录介质上进行存储；以及(b)如果具有分割的X个字节的序列并未处于所述相关性文件中，那么便生成具有Y个比特的新散列函数值，并将其与所述具有X个字节的片段序列一起存储在所述相关性文件中，并使用所述新散列函数值来在所述非高速缓存器记录介质上进行存储。

[0114]　上述方法要求分割具有N个字节的I/O流。然而，所述方法可以通过将加密散列函数值算法应用到I/O流并且不将其分割而加以应用。因此，一种替代方法包括：(i)接收具有N个字节的I/O流；(ii)使加密散列函数值与具有N个字节的每个单元关联，其中所述关联包括存取相关性文件，其中所述相关性文件使所存储的具有Y个比特的散列函数值与多个N字节存储序列中的每个序列相互关联，并且：(a)如果具有N个字节的序列是处于所述相关性文件中，那么便使用所存储的具有Y个比特的散列函数值来在非高速缓存器记录介质上进行存储；以及(b)如果具有N个字节的序列并未处于所述相关性文件中，那么便生成具有Y个比特的新散列函数值，并将其与所述N个字节一起存储在所述相关性文件中，并使用所述新散列函数值来在所述非高速缓存器记录介质上进行存储。另外，尽管各种方法是出于说明性目的而描述成是应用到一组N个字节或X个字节，但是本领域普通技术人员将会了解，所述方法可以且优选地应用于文件的所有N个字节或X个字节。

[0115]　另外，上述方法只是在字节串已经不处于相关性文件内时才应用所述算法。在另一实施方案中，本发明提供一种用于将数据存储在非高速缓存器记录介质上的方法，所述方法包括：(i)接收具有N个字节的I/O流；(ii)将加密散列函数值算法应用到具有N个字节的每个单元，以便针对具有N个字节的每个单元来形成生成的散列函数值；(iii)存取相关性文件，其中所述相关性文件使所存储的具有Y个比特的散列函数值与多个N字节存储序列中的每个序列相互关联，并且：(a)如果针对所述具有N个字节的单元的所述生成的散

列函数值是处于所述相关性文件中，那么便使用所存储的具有 Y 个比特的散列函数值来在非高速缓存器记录介质上进行存储；以及（b）如果针对所述具有 N 个字节的单元的所述生成的散列函数值并未处于所述相关性文件中，那么便将所述具有 Y 个比特的生成的散列函数值与所述具有 N 个字节的片段单元一起存储在所述相关性文件中，并使用所述生成的散列函数值来在所述非高速缓存器记录介质上进行存储。这个实施方案也可以修改成增加下述步骤：分割所述 N 个字节，以便形成片段单元；以及将所述加密散列函数值算法应用到所述片段单元，而不是应用到所述具有 N 个字节的单元。

[0116]　在上文所描述的实施方案中的某些实施方案中，在针对每个小盘、片段单元或子单元获取校验和之后，相关人员存取相关性文件。这些方法可以根据先进先出（“FIFO”）协议来获取校验和，并且在 I/O 流正在接收时、校验和正在生成时或者在所有 I/O 流已经接收、分割并经受散列函数值算法之后，开始存取相关性文件。

[0117]　如上文说阐明的，相关性文件使具有 Y 个比特的不同存储散列函数值与多个 X 字节存储序列中的每个序列关联。因此，在存在生成的散列函数值的这些情况下：（a）如果针对具有 X 个字节的单元的生成的散列函数值是处于相关性文件中，那么所述方法便致使利用所存储的具有 Y 个比特的散列函数值来在非高速缓存器记录介质上进行存储；以及（b）如果针对具有 X 个字节的单元的生成的散列函数值并未处于相关性文件中，那么所述方法便致使将具有 Y 个比特的生成的散列函数值与具有 X 个字节的序列存储在相关文件中，并使用生成的散列函数值来在非高速缓存器记录介质上进行存储。

[0118]　冲突解决模块

[0119]　作为应用散列值算法的结果而生成冲突散列值的可能性是较低的。因此，在一些实施方案中，相关人员可以选择使用本文中所描述的方法，而不使用冲突解决模块。然而，在各种实施方案中，本发明应用冲突解决模块，从而解决可能发生冲突的情形。

[0120]　如图 6 中所示，系统可以接收几组 N 个字节的流 610。系统随后可以致使将所述 N 个字节分割成具有合意大小的单元 620，从而针对每个片段单元来应用散列函数并生成散列函数值 630。接着，系统查询是否每个散列值已经处于相关性文件内 640。

[0121]　在这些实施方案中，存在冲突解决模块，每当对于具有 X 个字节的片段单元而言，生成与相关性文件中的散列值相同的散列值时，便会激活所述冲突解决模块 650。这个模块随后查询对于所述散列值而言具有 X 个字节的片段单元是否与已经存储的 X 字节数值相同。如果冲突解决模块确定对于具有 X 个字节的片段单元而言，生成与相关性文件中的存储散列函数值相同的散列函数值，但是具有 X 个字节的片段单元不同于与所存储的散列函数值关联的 X 个字节，那么便有必要更新相关性文件并将生成的散列值存储在其中，以及写入到 NCM 660。举例而言，在下文所描述的一些实施方案中，所述模块会致使存在不同的 Z 个比特与所存储的散列函数值以及生成的散列函数值关联。

[0122]　如果生成的散列值（与任何关联的 Z 值（若存在）一起）是独特的，那么其便可以用来在 NCM 上进行存储，并且相关性文件可以被更新成包括关联性 680。此外，如果散列值已经处于相关性文件中，并且所接收的 N 个字节与相关性文件中的那些字节相同，那么所存储的散列值（与任何关联的 Z 值（若存在）一起）便可以写入到 NCM，并且系统不需要更新相关性文件 670。因此，可以在存取相关性文件之后并且只有在新生成的散列值已经处于相关性文件内的情况下，才会作为检查来访问冲突模块。冲突模块随后会确定是否存在

冲突，或者来自所接收文件的校验和以及片段单元是否都已经在相关性文件中彼此关联。

[0123]　前述实施方案描述一种方法，在这种方法中，存取相关性文件之前，每个片段比特单元经受加密散列函数值算法。然而，作为替代地，相关人员可以首先检查相关性文件中子单元或小盘的存在，并且只有在片段单元已经不处于表中的情况下，才会应用加密散列函数值算法。然后，如果小盘或子单元已经处于相关性文件中，那么便使用所存储的校验和值。如果子单元或小盘并未处于文件中，那么相关人员将会应用散列值算法并进入协议来检查冲突。如果存在真实冲突，也就是，相同的散列值与不同的子单元或小盘关联，那么相关人员将会更新相关性文件来解决这个问题。

[0124]　一种用以解决真实冲突问题的方式是，对于不存在冲突的所有散列函数值而言，Z 个比特进行关联，并且所述 Z 个比特具有（例如）8 至 16 个 0 的统一长度。举一个非限制性示例，N ＝ 4096 字节（小盘大小），X ＝ 512 字节（子单元大小），Y ＝ 128 或 256 比特（散列值大小），并且 Z ＝ 8 或 16 比特（散列值扩展大小）。所述方法可以（例如）在校验和与之前存储的校验和并不冲突时，在校验和的末端处关联 8 个 0。在识别冲突后（例如，不同的片段单元与相同的校验和关联），最新的校验和可以指配有不同的 Z 值。因此，如果存储在相关性文件中的 Z 值是 00000000，那么针对第一冲突校验和的 Z 值可以是 00000001，并且应存在另一冲突校验和 00000010。如果存在额外的冲突校验和，那么每个冲突校验和可以随着冲突校验和被识别而指配有下一个 Z 值。

[0125]　在针对每个片段单元获取校验和后，并且紧接着冲突模块的应用（若存在）之后，多个散列函数值（且在一些实施方案中与 Z 值一起）可以写入到非高速缓存器记录介质而便于存储。

[0126]　散列值和比特标记符或频次转换器的组合使用

[0127]　在一些实施方案中，不是存储散列值，而是所述方法还包括：通过使用比特标记符表来转换（也称为编码）用于存储的每个散列函数值，以便针对用于存储的每个散列函数值来生成比特标记符；以及，将针对各别散列函数值的每个比特标记符（并且在一些实施方案中与 Z 值一起）存储在非高速缓存器记录介质中。通过使用比特标记符表，相关人员可以转换或编码散列值。因此，在存取比特标记符表时散列值可以充当输入子单元，并且关联的标记符可以充当输出。

[0128]　子单元可以由长度 A 界定，其中散列值的长度除以 A 是整数。如果任何比特串都不具有这个数目的比特，例如，一个或多个比特串比被配置来进行转换的子单元内的数目具有更小数目的比特，那么系统便可以添加比特，例如 0，直到所有子单元具有相同大小。或者，可以在将散列值划分为子单元之前，在散列值层面上执行 0 的添加。此外，如果散列值（并且在一些实施方案中与Z值组合在一起）小于子单元大小，那么它们便可以进行组合来形成子单元，或者进行组合和分割来形成具有适当大小的子单元。

[0129]　散列值向编码数据的转换可以由算法或计算机程序来执行。算法或计算机程序的运行可以（例如）由管理器进行控制，所述管理器也控制散列函数值算法的应用。编码数据（其也可以称为转换数据）也是由二进制信号组成，并且所述编码数据是以允许其被转回成文件的散列值的方式进行编码和存储。因此，在编码过程期间保持信息，所述编码过程允许在不损失信息的情况下进行解码。

[0130]　编码可以服务于两个独立目的。第一，通过比特标记符表来转换用于存储的数据，

便会增强安全性。只有知道代码的个人或实体才能够将所述数据解码以及重构文档。第二，如果使用比对应于文档的散列值更少的比特来创建代码，那么使用转换数据和散列值将会致使需要更少的存储空间，并且可以进一步节约成本。

[0131] 在一些实施方案中，不是使用比特标记符表，而是所述方法还包括，通过使用频次转换器来编码用于存储的每个散列函数值。在这些方法中，针对用于存储的每个散列函数值而言，生成转换比特串，其中，对于具有相同大小的散列值而言，所输出的转换比特串具有不同长度。频次转换器可以将多个转换比特串中的每个转换比特串与多个散列函数值中的每个散列函数值关联，并且频次转换器中的所述多个转换比特串是足够地截然不同，从而使得在回读时，转换比特串可以与适当的散列值相互关联。

[0132] 通过将散列值算法的使用与 (1) 比特标记符表应用或 (2) 频次转换器应用组合，相关人员便能够带来安全性的加强和 / 或成本的节约。另外，正如上文所描述的，散列值是针对比特标记符表或频次转换器而输入的。然而，顺序可以颠倒，且小盘或子单元可以进入使用比特标记符表或频次转换器来生成标记符的协议，并且这些标记符可以根据本文所描述的方法中的任何方法来充当散列函数算法的输入，这些方法包括但不限于使用冲突解决模块的方法。

[0133] 中介器和管理器

[0134] 比特标记符表、频次转换器、散列值相关性文件以及用以存取和使用前述文件和协议的程序中任何一者或每一者都可以存储在中介器内、管理器内或其它地方。每当前述协议或文件中的任何一个或多个协议或文件与本发明的实施方案结合使用，而且并未存储在中介器或管理器内时，优选地，其可操作性地耦接至中介器、管理器或这两者。用以与本地或远程存储的文件和程序进行通信的方法和系统对于本领域普通技术人员而言是熟知的。

[0135] 在各种实施方案中，管理器控制本发明的方法。举例而言，在 I/O(从主机输出、输入到系统) 从主机接收之后，管理器可以致使向主机确认 I/O 流的接收。

[0136] 主机可以将 I/O 流的文件记录成是存储在第一存储地址上。然而，转换比特串事实上可以存储在非高速缓存器记录介质中的第二存储地址上，其中第一存储地址与第二存储地址并不相同。此外，第一存储地址可以将文件展示为具有第一文件大小 (或者与文件的名称关联)，而第二存储地址可以对应于第二文件大小，其中第一文件大小至少是第二文件大小的两倍大或者至少是四倍大。

[0137] 在一些实施方案中，第二存储地址存储在中介器上。在中介器内，第二存储地址可以存储在比特字段中。本文中所描述的中介器可以与所描述的通过比特标记符表、频次转换器和散列值算法中的一者或多者来转译数据的方法独立使用或结合使用。

[0138] 在一些实施方案中，中介器为硬件、软件或其组合，所述中介器包括 :(i) 第一组磁道，其中所述第一组磁道包括或含有与文件系统信息、可启动性信息和分区信息相对应的信息 ;(ii) 第二组磁道，其中所述第二组磁道包括或含有与第一组磁道的复本相对应的信息 ;(iii) 第三组磁道，其中所述第三组磁道包括或含有与不同于文件系统信息、可启动性信息和分区信息的元数据相对应的信息 ;以及 (iv) 第四组磁道，其中所述第四组磁道包括或含有对应于比特字段的信息。举例而言，中介器可以存在于 L2 高速缓存器中，并且比特标记符表、频次转换器或散列值表中的任何一者或多者可以在使用时存在于服务器的

RAM 中，且也可以存在于固态装置的永久性储存器中，所述固态装置可以与上面存储标记符或散列值的 NCM 相同或不同。如果进行更新，那么会对 RAM 中的表做出更新，并且也会对永久性存储器做出更新。

[0139] 除了使用中介器之外，本发明的各种实施方案要求使用管理器。管理器可以包括一个或多个模块，并且可以存在于本地计算机上、网络上或云中。管理器被配置来协调某个信息的接收或接收某个信息本身，以及将这个信息传递给中介器，或直接通过中介器来控制信息的接收。因此，所述方法可以经过设计，从而使得来自发起器的信息经过管理器而流向中介器，或者管理器只指引信息流动到系统的其它部件，例如主机或 NCM。

[0140] 在本发明的各种实施方案中，管理器可以应用或致使应用散列函数算法或其它协议、任何冲突模块（若存在）以及任何转换模块（若存在），并且致使数据直接流向中介器。管理器也可以通过使用中介器来控制信息的存储以及信息的检索和传输。当存储信息时，LBA 编号可以进行识别，并且有待存储的数据可以发送给缓冲器，从而避免或减少瓶颈效应。此外，在存储过程中可以使用 L1 和 / 或 L2 高速缓存器。

[0141] 类似地，当检索数据时，所述方法可以从 NCM 调用数据、填充缓冲器、获取校验和值且然后重新创建片段单元和进行重组，或者如果没有进行分割，便直接重组来形成呈主机可以接收和核查的形式的信息。如果所存储的数据是转换数据，那么在获取校验和值之前，所述数据可以解码。这些步骤可以借助管理器来协调和控制，所述管理器会执行必要的协议。

[0142] 在一些实施方案中，管理器可以控制一个或多个中介器、与其进行通信并协调其活动。对于每个中介器而言，管理器接收一组参数（或协调这组参数的接收）。这些参数可以包括文件系统信息、可启动性信息和分区信息中的一者、两者或所有三者，基本上由其组成或者由其组成。管理器致使这个信息存储在中介器上的第一组磁道中，所述第一组磁道可以称为预留 1 或 R_1。文件系统将规定如何使用预留区块。举例而言，当使用 NTFS，扇区 1 至 2 可以用于 MBR（主启动记录），而扇区 3 可以用于 $MFT。可选地，这些磁道可以复制到第二组磁道中，所述第二组磁道可以称为预留 2 或 R_2。

[0143] 除了前一段落中所描述的参数之外，管理器也可以接收元数据。元数据存储在中介器上的第三组磁道中。在管理器接收参数和元数据的时间上或者在稍后时间上，管理器也可以接收一个或多个文件以便存储在非高速缓存器介质上。每个文件都是在带有文件名的情况下接收的。文件名由传输文件的主机生成，并且可以由主机的文件系统进行定义。可以（例如）作为 SAN 或 NAS 或其组合，或者作为 SAN 或 NAS 或其组合的一部分的管理器，在接收具有文件名的文件后，可以自动执行本文中所描述的步骤来进行存储。

[0144] 本发明的系统可以经过设计，从而使得散列函数值算法和转换算法存储在中介器内、或管理器内，或者存储在可操作性地耦接至中介器或管理器的其它硬件和 / 或软件内。所述算法中的任一算法或两个算法也可以致使文件名存储在中介器内。对中介器物理上所位于的地方没有任何限制。然而，优选地，中介器被配置成与主机或计算机通信，所述计算机能够与主机通信，所述主机优选地位于远离中介器的地方上。中介器也被配置成直接或间接（例如，借助管理器）与记录介质（例如，存储有所述组编码数据的非高速缓存器介质）通信，所述记录介质可选地远离中介器、任何管理器和主机。正如上文所提到的，中介器允许识别文件名的用户来从非高速缓存器存储介质检索所述组编码数据。

[0145]　高速缓存器

[0146]　正如上文所阐明的,在一些实施方案中,在接收原始数据后,本发明的方法可以致使自动向主机返回接收的确认。在一个 QoS(服务质量)协议中,数据文件是借助 I/O 来接收,并立即发送给 L1 高速缓存器。在接收后,借助 I/O 而从 L1 高速缓存器发回确认。数据文件可以从 L1 高速缓存器发送给 L2 高速缓存器,所述 L2 高速缓存器将确认回传给 L1 高速缓存器。L2 高速缓存器也可以将数据文件发送给非高速缓存器介质 (NCM),以便长期存储。NCM 又可以将确认发回给 L2 高速缓存器。

[0147]　在一些实施方案中,中介器可以存在于 L1 高速缓存器内的堆积(动态分配存储器)中,或者可操作性地耦接至所述堆积。或者,中介器可以存在于卡片内,或者作为 L2 高速缓存器的一部分或可操作性地耦接至 L2 高速缓存器。

[0148]　正如本领域普通技术人员所知悉的,将中介器放置在 L1 与 L2 中的决策将受到多个因素的影响,如存储数据的使用频次。因此,L1 高速缓存器用来存储系统或最终用户频繁使用的数据,而 L2 高速缓存器可以用于略微频繁存取的数据。

[0149]　在另一 QoS 协议中,借助 I/O,数据文件由 L1 高速缓存器加以接收。数据文件从 L1 高速缓冲器传递给 L2 高速缓存器和 NCM。L2 高速缓存器和 NCM 中的每一者都将确认发送给 L1 高速缓存器。在从 L2 高速缓存器和 NCM 中的一者或两者接收确认之前或之后,L1 高速缓存器借助 I/O 来发送确认。

[0150]　文件相关性

[0151]　在本发明的各种实施方案中,主机将会了解有待存储在第一存储地址上的每个文件。第一存储地址可以由主机存储在扇区图中,并且对应于 LUN。第一存储地址也可以包括对应于文件的单元、扇区或区块的起点并且暗示地或明确地包括其末端。第一存储地址将对应于主机认为文件在存储装置或存储区域网络内所处于的位置。主机将使用这个第一地址来跟踪其存储文档或文件,并且对它们进行检索。第一存储地址是虚拟地址,也就是,其并不对应于数据实际存储的位置。

[0152]　正如本领域普通技术人员会认识到的,可使用多个方法和系统,其中主机生成第一存储地址,并将其与 SCSI 命令以及可选的是关联扇区或 LBA 编号一起发送给本发明的系统。中介器可以将文件名、主机认为是作为从主机接收的文件的位置和文件的存储大小的信息(也就是原始数据)以及任何页眉或页脚数据与第二存储地址相互关联,所述第二存储地址是数据的实际存储地址,这个地址可以含有可供重新创建文件的信息,但是所述信息已经借助(例如)散列值算法、比特标记符表和频次转换器中的一者或多者进行了转换。或者,中介器可以只存储文件名,并且可选地,中介器可以不接收用于文件的第一存储地址。正如上文所提到的,因为存储地址是基于数据的线性组织,所以它们可以通过只列举第一用户感知 LBA 或明确地列举所有位置,而暗示地含有存储信息的大小。

[0153]　尽管上面的段落描述了主机将会提供其认为是第一存储地址的信息,但是所述信息可以由另一实体生成,所述实体是供主机直接或间接与中介器通信的通道、处于主机内或可操作性地耦接至主机的模块,或者是处于中介器和 / 或管理器内或可操作性地耦接至中介器和 / 或管理器的模块。正如本领域普通技术人员会认识到的,识别存储装置上数据文件位置的存储信息可以称为指针。对于文件而言,指针可以从与 NCM 上用户认为数据所处的位置不同的位置(例如,区块)来指引数据的检索。

[0154] 可选地，相关人员可以使文件与文件类型关联。文件类型将会指引数据的接收者来知悉应使用哪个操作系统来打开所述数据。在一些实施方案中，与文件类型的关联是在发起器或客户端或主机处完成的。

[0155] 预留磁道

[0156] 正如上文所提到的，中介器可以包括第一组预留磁道（R_1）和第二组预留磁道（R_2）。在一些实施方案中，第二组预留磁道（R_2）是第一组预留磁道（R_1）的复本。另外，在一些实施方案中，相关人员可以使用第二组预留磁道（R_2）来检查第一组预留磁道（R_1）中的错误。

[0157] R_1可以被配置来充当主机发起的中心点。因此，主机可以选择参数来发送给 R_1。中介器可以直接从主机或者间接地借助管理器来接收这个信息。R_2优选的是从不暴露给主机。因此，只有中介器本身或管理器可以致使将信息存储在 R_2中。R_1和 R_2中的每一者可以（例如）含有 16 个扇区，并且填充有真实数据，如主机修改符。按照惯例，编号可以从 0 开始。因此，R_1可以（例如）含有扇区（或磁道）0 至 15，而 R_2可以含有扇区（或磁道）16 至 31。然而，中介器可以经过构建，从而允许将 R_1和 R_2中的每一者扩展成超过 16 个磁道的初始大小。

[0158] 在一些实施方案中，R_1含有独特的预留扇区信息和分区信息。在分区信息内，相关人员可以存储文件系统信息。

[0159] 举一个非限制性示例，并且正如本领域普通技术人员所意识到的，当利用 NFTS 文件系统格式化某个卷时，相关人员会创建元数据文件，如 $MFT（主文件表）、$Bitmap、$Log 文件以及其他文件。这个元数据含有与 NFTS 卷上所有文件和文件夹相关的信息。NFTS 卷上的第一信息可以是分区启动扇区（$Boot 元数据文件）信息，并且可以位于扇区 0 处。这个文件可以描述基本的 NTFS 卷信息和主要元数据文件 $MFT 的位置。

[0160] 格式化程序会为 $Boot 元数据文件分配前 16 个扇区。第一扇区为具有引导程序代码的启动扇区，并且随后的 15 个扇区为启动扇区的 IPL（初始程序加载程序）。

[0161] 除了 R_1和 R_2磁道之外，中介器还可以存储额外的元数据。这个元数据可以（例如）对应于允许运行自动精简配置策略的信息，所述自动精简配置策略对应于允许装置看上去比实际可用资源具有更多物理资源的可视化技术，并且，这个元数据可以（例如）包含在 R_2之后的八个磁道中，所述八个磁道是磁道 32 至 39。元数据也可以提供有多个特征，如 LUN QoS、VM 和 WORM。

[0162] 在一些实施方案中，系统可以含有 SAN 索引器。SAN 索引器可以检查 R_1和 R_2中存在什么信息，并且提取所述信息。这个信息可以放入到数据库中，可以通过（例如）文本搜索而轻易地搜索所述数据库。

[0163] 比特字段

[0164] 中介器也可以包括或含有对应于比特字段的信息。比特字段含有指示数据在存储介质内物理上存储的位置的信息，并且如果元数据位于磁道 32 至 39 中，那么比特字段的扇区编号可以开始于磁道 40。主机的文件名与数据的位置之间的相关性也是存储在中介器的比特字段内。因此，比特字段可以包括扇区图、基本上由扇区图组成或者由扇区图组成。来自中介器的比特字段成分的这个信息可以用来确定任何装置上的实际空间节约。举例而言，所节约空间的百分比＝ 1-[（实际使用的空间）/（主机所映射的空间）]。在一些实施方

案中，根据本发明的方法来转换数据而节约的空间至少是50％、至少是60％、至少是70％或至少是80％。这些节约可以针对每个文件或者对于装置上的所有文件取平均值。

[0165] 作为惯例，优选地中介器并不位于存储文件数据的磁盘或记录介质上。另外，优选地中介器只需要对应磁盘或记录介质的总内存的约0.1％至0.2％。

[0166] 除了从空间节约方面提供经济价值之外，本发明的各种实施方案在寻求保护数据完整性时为提高效率打开了方便之门。因此，本发明的各种实施方案为备份数据提供新颖且非显而易见的技术。

[0167] 因为在许多实施方案中，存储文件将会小于从主机所接收的原始数据文件，所以需要更少的存储空间来用于所述存储文件。因此，重新创建文件所需要的数据可以存储在比主机察觉到正在使用的位置以及第一存储地址所表明的位置更小的位置中。文件所存储的实际位置可以称为第二存储地址。因此，对于每个文件而言，将会存在第一存储地址和第二存储地址，其中第一存储地址是主机认为文件所存储的地方，而第二存储地址则是编码文件实际存储的地方。

[0168] 可能的是，对于给定文件（其可对应于一个或多个区块）而言，第一存储地址和第二存储地址位于存储装置内的同一区块处或者一组或多组重叠区块处。然而，优选地，对于文件中的至少一个文件、至少50％、至少60％、至少70％、至少80％、至少90％或100％而言，第一存储地址和第二存储地址内不存在区块重叠。另外，即使主机与中介器察觉到相同的存储地址，但是当数据编码时，主机也无法在未首先解码数据的情况下来重新创建文件。在一些实施方案中，主机并不知道代码，且因此不能够解码存储数据。

[0169] 在一些实施方案中，中介器可以接收对应于文件的小盘，并将它们临时存储在L1或L2高速缓存器中。如果存在L2高速缓存器，那么L2高速缓存器便可以向主机确认接收，并且可选地向主机提供或确认第一存储地址。正如本领域普通技术人员会认识到的，第一存储地址接收和传输的确认可以在第二存储地址上进行存储之前完成，并且，如果执行编码，那么便在编码之前或之后完成。无论何时发送确认，文件的实际存储地址与虚拟存储地址的相关性，优选的是在比特字段内跟踪。

[0170] 数据的备份

[0171] 在某些实施方案中，相关人员可以使用两个中介器来促进数据备份。举例而言，在第一中介器中，相关人员可以使存储在第一记录介质上的第一扇区或第一扇区群集中的数据文件与文件名相互关联。正如上文所描述的，第一中介器被配置成允许识别文件名的用户或实体来从记录介质检索数据文件。

[0172] 可以执行数据保护协议，从而生成第二中介器。第二中介器将会是第一中介器在时间T1上的精确复本。因此，在时间T1上，第一中介器和第二中介器都将指向第一记录介质上的相同扇区或扇区群集（以及其中的位置）。

[0173] 在时间T1之后，例如在时间T2上，主机可以试图更新存储在给定扇区或扇区群集上的给定位置中的文件。主机将不会更改第一存储地址上所存储的数据。不是致使改写扇区或扇区群集上的信息，而是第一中介器可以致使将新的信息写入到对应于不同扇区或扇区群集中的位置的第三存储地址，并使文件名以及第一存储地址与这个新存储地址相互关联。

[0174] 因此，即使主机认为正在特定存储地址上重写信息，但是第一中介器也将指向新

的扇区或扇区群集。因此，主机将不需要针对扇区群集来更新其地址。

[0175] 第二中介器将不会被更新，并且其将继续使文件名与文件所存储的第一位置相互关联。这样使用两个中介器将会允许相关人员提供数据存在于时间 T1 上时的快照，而不会使得主机需要更新其文件系统来指示存在于时间 T1 和 T2 上时的文件正在被存储。因此，快照会锁定时间 T1 上存储的所有数据文件，并且防止任何人删除或改写这些物理文件。然而，如果主机想要修正这些文件，那么主机可以在持有正在修正文件的印象下工作，事实上这时会存储新的文件。这种方法是结合扇区和扇区群集来描述的。然而，所述方法也会与并非布置在扇区或扇区群集中的非高速缓存器介质一同使用。举例而言，它们可以由 LUN 中的 LBA 加以组织。

[0176] 正如上文所表明的，这种方法可以由包括第一中介器、第二中介器和非高速缓存器存储介质的系统来实施。第一中介器、第二中介器和记录介质中的每一者可以存储在单独装置上或者由单独装置形成，所述单独装置包括非暂时性介质、基本上由非暂时性介质组成或者由非暂时性介质组成。前述系统列举同一记录介质的不同扇区的使用，但是也可以通过写入到不同记录介质的不同扇区来使用。另外，在系统内，中介器和记录介质可操作性地彼此耦接，并且可选地耦接至一个或多个计算机或 CPU，所述一个或多个计算机或 CPU 存储指令以便致使它们执行它们的预期功能，并在网络上借助一个或多个门户向一个或多个主机进行通信。此外，尽管这个实施方案是结合两个中介器的使用来描述的，但是相关人员也可以使用同一中介器的两个区段来实施系统，而不是使用两个单独中介器。

[0177] 系统可以进一步配置有锁定模块。锁定模块可以防止在从某个时间起已经写入的一个或多个区块上进行修正、重写或删除。锁定模块也可以经过设计而容许写入新区块以及修正尚未锁定的这些新区块。因此，锁定模块可以配置成允许主机、用户或系统管理员来选择从某个时间起已经写入的某些区块，或者选择从某个时间起已经写入的所有区块都不进行重写。

[0178] 此外，可以存在选择模块，所述选择模块默认借助第一中介器来发送所有的文件检索以及修正、重写或删除请求。选择模块也可以被配置成允许恢复主机可能认为从应用锁定技术的时间起是一个或多个文件的更老版本的文件。可选地，整个选择模块的访问权限可以限制给具有授权的人员，例如，系统管理员。

[0179] 用于备份数据的前述系统是在两个中介器的情境中加以描述的。然而，也可以使用两个以上的中介器来捕获存储文件或文件版本的历史数据。举例而言，可以使用至少三个、至少四个、至少五个、至少十个中介器等等。另外，主机可以使得中介器在有规律间隔上（例如，每周、每月、每个季度或每年）或不规律间隔上（例如根据需要）获取快照。

[0180] 根据另一种用于备份数据的方法，可以制作非高速缓存器介质的克隆形式。在这种方法中，在第一中介器中，相关人员使多个文件名与存储在非高速缓存器存储介质上的数据的多个位置相互关联。第一中介器被配置成允许识别特定文件名的用户来从第一非高速缓存器存储介质检索对应于所述特定文件名的数据文件。特定文件的一部分或整个特定文件可以存储在第一扇区或扇区群集中。

[0181] 相关人员可以将所述多个数据文件（或第一非高速缓存器存储介质的所有数据文件）复制到第二非高速缓存器存储介质和第二中介器。第二中介器是第一中介器在时间 T1 上的复本，并且可操作性地耦接至第二非高速缓存器存储介质。在时间 T1 之后的时间 T2

上，系统可以保存第一非高速缓存器存储介质上的所述第一扇区或扇区群集中所存储的数据文件的修正文件。然而，不会对第二非高速缓存器介质上的对应位置做出任何更改。用户在时间 T2 之后请求文件时，他或她将会仔细查看第一中介器并检索文件的最近存储版本。然而，系统管理员会对存储在第二非高速缓存器介质上的早先版本具有存取权限，并且可以通过仔细查看第二中介器来检索早先版本。

[0182] 这种方法可以由包括第一中介器、第二中介器、第一非高速缓存器存储介质以及第二非高速缓存器存储介质的系统来实施。用于存储数据文件的第一中介器、第二中介器以及第一和第二记录介质中的每一者可以存储在单独装置上，所述单独装置包括非暂时性介质、基本上由非暂时性介质组成或者由非暂时性介质组成。另外，第一中介器使从主机获得的文件名与第一记录介质的第一 LUN 相互关联，并且第二中介器使同一文件名与第二记录介质上的第二 LUN 相互关联。在一些实施方案中，存储在第一非高速缓存器介质中的最近的文件与第二非高速缓存器介质内的遗留文件具有相同的 LUN。

[0183] 数据检索

[0184] 根据本发明的方法中的任何方法，以转换形式存储的任何数据都能够在将其返回给主机之前进行检索和解码。通过使用允许检索转换数据、存取上述参考表或频次转换器以及转回成统一比特串和小盘的一个或多个算法，便可以为主机重新创建文件。举一个非限制性示例，数据可能已经按照某个格式进行了转换和存储，所述格式含有一个标记符结束位置的指示，例如，使用独特的比特串或者通过使用统一大小的标记符。

[0185] 所存储数据的检索可以借助现在已知的或者将要知悉的并且本领域普通技术人员了解到会与本发明结合使用的过程和技术。可选地，管理器协调文件的存储和检索。在一些实施方案中，来自 NCM 的数据（如标记符）是借助并行处理来检索。

[0186] 在一个方法中，相关人员从访问记录介质开始。记录介质以某个顺序存储多个标记符，并且相关人员可以根据这些标记符来重新创建文件。访问可以由请求文件的检索并将请求传输给存储区域网络的主机发起，或者由存储区域网络的管理员发起。

[0187] 在从中介器（若存在）识别的位置上来从记录介质检索数据之后，如果数据已经转换，那么相关人员便将所述多个标记符（或已经借助频次转换器进行转换的数据）转译成可以用来形成有待于转回成 I/O 流或 I/O 流片段单元的小盘或散列值的比特。标记符可以经过存储，从而使得每个标记符对应于小盘或者每个标记符对应于子单元，并且多个子单元可以组合来形成小盘。在存储格式中，以容许重新创建期望文档或文件的方式、按照允许重新创建小盘（或散列值）内的比特和重新创建小盘（或散列值）顺序的顺序来布置标记符。

[0188] 如果数据经过转换，那么为了将标记符转译成小盘，相关人员可以存取比特标记符表或频次转换器。在比特标记符表或频次转换器内，可以存在与每个独特比特串关联的，或者处于文件内的每个独特比特串内的独特标记符。如果所述表是组织在类似于表 II 的格式中，那么在转译之后，可以添加 0，从而使得每个子单元和小盘具有相同大小。在解码时，相关人员以与其用于编码的方式相反的方式来使用比特标记符表或频次转换器。可选地，不是使用同一个表以及颠倒输入和输出，而是相关人员可以使用单独的表。

[0189] 就其它实施方案而言，每个小盘的长度可以是 N 个比特，其中 N 为大于 1 的整数，并且每个子单元的长度可以是 A 个比特，其中 A 为整数。为了将标记符转译成小盘，相关人

员可以存取比特标记符表或频次转换器。

[0190] 在形成小盘之后，相关人员将会具有对应于二进制数据的输出，可以根据所述二进制数据来重新组建文档。可选地，相关人员可以使文件与文件类型关联。举例而言，主机可以跟踪MIME转译程序，并在返回后将其与文件重新关联。文件类型将会指引数据的接收者来知悉应使用哪个操作系统打开所述数据。正如本领域普通技术人员会认识到的，存储区域网络不需要跟踪文件类型，并且在一些实施方案中也并未跟踪文件类型。

[0191] 在一些实施方案中，形成小盘之后，相关人员将会具有对应于二进制数据的输出，可以根据所述二进制数据来重新组建文件。在其它实施方案中，转换数据对应于散列值，并且散列值必须首先用来创建I/O流的片段单元或I/O流本身。

[0192] 相对于截断数据而言，甚至在不需要利用比特标记符表或频次转换器的情况下，相关人员也可以检索数据。相关人员可以通过访问记录介质来检索数据，其中所述记录介质将多个数据单元存储在多个位置中，其中每个数据单元含有多个比特并且数据单元的最大大小为第一比特数目，至少一个数据单元含有第二比特数目，其中第二比特数目小于第一比特数目。

[0193] 接着，相关人员可以检索数据单元，并且在长度小于N个比特的任何数据单元的末端添加一个或多个比特(例如0)，以便生成一组对应于数据单元的小盘，其中每个小盘含有相同数目的比特；以及，以与数据单元的顺序相对应的顺序来生成包括所述组小盘的输出。如果截断数据是通过移除0来形成的，那么在检索数据时，相关人员会将0加回。另外，如果存储的数据单元为小盘的子单元，那么系统可以首先将0加回到截断子单元，从而生成具有统一大小的子单元，且然后组合子单元来形成小盘。

[0194] 当使用查找表时，优选地，所述查找表是存储在计算装置的存储器中。在一些实施方案中，查找表是静态的，并且标记符是预定的。因此，当随着时间的推移来存储一种或多种不同文档类型的多个文档时，可以使用同一个表。可选地，这个表可以存储在主机的位置处，或者作为存储区域网络的一部分加以存储。

[0195] 从主机接收数据

[0196] 为了进一步说明各种实施方案并提供情境，下文将参考相关人员可以使用的特定硬件，所述硬件可以组合来形成系统以便实施本发明的方法。所述硬件可以被配置成容许在从主机接收一个或多个数据文件时，通过一个或多个处理器将上述过程自动并进行控制。

[0197] 在一些实施方案中，主机可以按照任何方式在第一位置处生成文档和计算机文件。文档将由主机的操作系统生成，并且经过组织而由主机的文件系统进行存储。主机的操作系统可以将文件名本地存储在其存储器中。本发明并不受限于主机所使用的操作系统或文件系统的类型。举一个非限制性示例，主机可以包括网络内的计算机或一组计算机，所述计算机或所述组计算机具有下述硬件部件中的一个或多个部件：存储器、储存器、输入装置、输出装置、图形用户界面、一个或多个通信门户以及中央处理单元。

[0198] 在所述第一位置处，SAP执行协议而用以存储与文档或文件相互关联的数据。SAP将数据格式化成大小为(例如)4K的I/O流或小盘。

[0199] 数据可以在SAN上发送给具有一个或多个模块的计算机或网络，或者发送给被配置来接收数据的计算机或一组计算机。这个接收计算机可以包括下述硬件部件中的一个或

多个部件：存储器、储存器、输入装置、输出装置、图形用户界面、中央处理单元以及被配置成允许在本地和／或在网络上与一个或多个主机和一个或多个存储装置进行信息通信的一个或多个通信门户。

[0200] 另外，可以存在将可执行计算机代码存储于硬件、软件或硬件与软件的组合上的计算机程序产品。所述计算机程序产品可以划分为一个或多个被配置来执行本发明方法的模块或者能够与所述一个或多个模块通信，并且可以存储在一个或多个非暂时性介质中。

[0201] 举例而言，可以存在 1 级 (L1) 高速缓存器和 2 级 (L2) 高速缓存器。正如本领域普通技术人员所知晓的，高速缓存器技术的使用在传统上容许相关人员提高数据存储的效率。在本发明中，举例而言，数据可以在 SAN 上发送给高速缓存器，并且数据可以在存取散列函数算法之前、在查询比特标记符表之前、在查询频次转换器之前以及在将比特截断之前，和／或在查询散列函数算法之后、在查询比特标记符表之后、在查询频次转换器之后以及在将比特截断之后，发送给高速缓存器。

[0202] 假设扇区大小为 512B，那么对于大小为 4K 的每个小盘而言，主机将期望使用 8 个存储扇区。

[0203] 系统

[0204] 本发明的各种实施方案提供数据存储和检索系统。在一个实施方案中，所述系统包括非高速缓存器数据存储介质、中介器和管理器。这些元件之间的通信以及可选地与发起器之间的通信可以在有线网络、无线网络或其组合上进行。

[0205] 非高速缓存器数据存储介质可以（例如）包括一个或多个磁盘或固态驱动器、基本上由所述一个或多个磁盘或固态驱动器组成或者由所述一个或多个磁盘或固态驱动器组成。在使用时，与尚未根据本发明方法中的一个或多个方法进行处理的文件相比而言，非高速缓存器数据存储介质可以在节约至少 50％、至少 60％、至少 70％或至少 80％的空间的情况下来存储文件。

[0206] 中介器可以包括四组磁道、基本上由这四组磁道组成或者由这四组磁道组成：第一组磁道、第二组磁道、第三组磁道和第四组磁道。中介器优选地存储在非暂时性介质上，并且位于远离非高速缓存器数据存储介质的地方。因此，中介器和非高速缓存器数据存储介质优选地不是同一装置的部分。

[0207] 系统也可以含有管理器。管理器可以借助中介器来对数据的接收、处理存储以及检索和传输提供控制。因此，优选地，管理器可操作性地耦接至主机和中介器，并且可选的是可操作性地耦接至非高速缓存器数据存储介质。此外，在一些实施方案中，管理器位于远离中介器、非高速缓存器介质和主机中的每一者的地方。

[0208] 管理器可以被配置来执行下述特征中的一个或多个特征：(a) 将包括文件系统信息、可启动性信息和分区信息中的一个或多个信息的数据存储在中介器的第一组磁道中；(b) 将元数据存储在中介器的第三组磁道中；(c) 将一个或多个文件存储在非高速缓存器介质上，其中所述一个或多个文件存储在不具有文件系统信息、可启动性信息和分区信息中的任何信息的非高速缓存器介质上（因此在一些实施方案中，只有原始数据处于非高速缓存器介质上）；(d) 将非高速缓存器介质中每个文件的位置存储在中介器的第四组磁道中；以及 (e) 将非高速缓存器介质中每个文件的位置与文件的主机名的相关性存储在中介器上。优选地，非高速缓存器介质中每个文件的位置与文件的主机名的相关性是存储在第

四组磁道中，所述第四组磁道对应于比特字段。另外，管理器可以包括或可操作性地耦接至计算机程序产品，所述计算机程序产品含有能够执行本发明方法中的一个或多个方法的可执行指令。

[0209] 在本发明的各种系统中，根据存储在计算机程序产品中的指令，SAN 可以存取比特标记符表或频次转换器。这些资源使比特标记符与子单元中的每个子单元相互关联，并生成输出。因为即便不是全部，也会有大部分比特标记符在大小上小于子单元，所以所述输出便为比从主机接收的输入文件更小的数据文件。因此，虽然从主机接收的文件可以具有大小 R，但是 SAN 所保存的实际数据可以是 S，其中 R>S。优选地，R 至少是 S 的两倍大，并且更优选地，R 至少是 S 的三倍大。

[0210] SAN 取得输出文件并将其存储在非暂时性存储介质（例如，非高速缓存器介质）中。优选地，SAN 使所存储的文件与从主机接收的文件相互关联，从而使得主机可以检索文件。

[0211] 出于进一步说明的目的，可以参考图 1，其展示用于实施本发明方法的系统。在系统 10 中，主机 1 将文件传输给存储区域网络 6，所述存储区域网络 6 含有可操作性地耦接至存储器 4 的处理器 3。可选地，存储区域网络确认接收并回复给主机。

[0212] 在存储器内存储有计算机程序产品，所述计算机程序产品经过设计而取得小盘并将其中含有的数据划分为子单元。存储器也可以含有或可操作性地耦接至参考表 5。所述表含有针对子单元中的一个或多个子单元的比特标记符或频次转换器或相关性文件，并且计算机程序产品创建新数据文件，所述新数据文件含有取代原始子单元的标记符中的一个或多个标记符。

[0213] 处理器接着致使将比特标记符存储在记录介质（如非高速缓存器介质）上，所述记录介质可以（例如）是磁盘 2。在一些实施方案中，最初所有标记符都具有相同大小。然而，在一些实施方案中，在存储这些标记符之前，一个或多个标记符、优选的是至少 25％、至少 50％或至少 75％在存储之前被截断。

[0214] 图 2 展示具有中介器 70 的系统 100，所述中介器 70 含有 $R_1$40 和 $R_2$50 以及用于比特字段 60 和元数据文件 30 的空间。中介器的表示形式仅仅出于说明性目的，并且不会对中介器的结构或其内的组织施加任何限制。也在图中展示的是非高速缓存器介质(NCM)20。非高速缓存器介质展示为处于中介器的附近，但是它们是单独的结构。在一些实施方案中，中介器位于与 NCM 相分离的装置上，并且本发明的装置中的一个或多个装置是便携式的。标记符或其它转换原始数据将存储在 NCM 上。

[0215] 图 3 展示另一系统 200。在这个系统中，发起器 (I^n)270 将小盘传输给高速缓存管理器 230，所述管理器 230 可选地会安排数据文件的编码并将数据文件传输给中介器 210。主机的示例包括但不限于运行 Microsoft Windows Server 和 Desktop、Apple OS X、Linux RHEL 和 SUSE Oracle Solaris、IBM AIX、HP UX 以及 VM ESX 和 ESXi 的计算机或计算机网络。对应于数据文件的信息最初发送给 $R_1$240，$R_1$240 之前填写有发起器所定义的参数。中介器可以通过使用比特标记符表或频次转换器（未图示）而亲自转译信息，或者中介器可以与远程编码器（其也可以称为远程转换器，而且未图示）通信，并且中介器会将从主机接收的文件名的复本存储在 R_1内以及 $R_2$250 内。在数据已经转换并且更小大小的文件已经创建之后，在比特字段 260 的扇区图内记录文件将在磁盘 220 中所存储的某个位置或某些位

置。编码数据可以存储在位置 285 处。在数据编码之前或者代替数据编码，可以存取散列值算法，并且校验和可以用于存储或转换。

[0216] 图 4 展示另一系统 300，所述系统 300 是图 3 系统的实施方案的变型并且提供存储的备份。在这个系统中，发起器 370 将小盘传输给高速缓存管理器 330，高速缓存管理器 330 将信息转发给第一中介器 310，第一中介器 310 含有数据来修正针对图 3 所发送的同一文件。在接收修正文件之前或者在接收修正文件之后，但是在将修正文件存储于非高速缓存器介质中之前，从第一中介器 310 创建第二中介器 380。第二中介器是第一中介器在其创建时间上的精确复本，并且在这个时间上，对于文件名而言，第二中介器指向非高速缓存器介质 320 内的同一扇区（或扇区群集）385。

[0217] 第一修正文件是在第一中介器的 $R_1$340 处接收。第一中介器将再次通过使用比特标记符表或频次转换器（未图示）来转译信息，或者与远程编码器通信。中介器会继续将从主机接收的文件名的复本存储在 R_1内以及 $R_2$350 内。在数据已经转换并且更小大小的文件已经创建之后，在第一中介器的比特字段 360 的扇区图内记录文件将在磁盘 320 中所存储的位置。然而，修正文件将会存储在不同的扇区 395 中。因此，对第一中介器的更改不会针对第二中介器做出。

[0218] 主机默认与第一中介器通信。因此，当主机想要从储存器重新调用文件时，第一中介器将会从扇区 395 调回数据。如果主机或系统管理员希望获取数据的前一个版本，那么它可以将文件名提交给第二中介器，所述第二中介器会查看扇区 385。

[0219] 根据本发明的另一实施方案，之前已经将元数据（例如，操作系统信息、可启动性信息、分区信息、文档类型信息、QoS 信息等）提供给本发明系统的发起器，将对应于文档的比特发送给高速缓存管理器。所述比特可以（例如）组织在小盘中。

[0220] 高速缓存管理器可以将信息发送给 L1 高速缓存器、L2 高速缓存器和中介器。高速缓存管理器也可以向发起器发送接收确认。

[0221] 上面可能已经存储有相关元数据的中介器，将对应于文档的比特发送给转换器，所述转换器将比特转换成编码信息。计算器也可以与转换器关联或者作为转换器的一部分，所述计算器确定转换比特的大小。可以（例如）通过使用比特标记符表或频次转换器来进行转换。

[0222] 转换器随后可以将转换文件的大小告知给中介器，并且中介器可以确定转换文件将在非高速缓存器介质中所存储的位置。紧接着这个步骤之后，中介器可以致使在所述位置上进行存储。

[0223] 本说明书中所描述的各种实施方案的特征中的任何特征，都可以与结合所公开的任何其它实施方案来描述的特征联合使用，除非另有说明。因此，结合各种或特定实施方案来描述的特征不应解释为不适于结合本文所公开的其它实施方案，除非上下文中明确陈述或暗含此类排他性。

[0224] 示例：

[0225] 示例 1：比特标记符表（预示性的）

[0226] 在参考定位符表内，每个独特标记符被识别为对应于独特比特串。所述表可以按照普遍熟知的或将要知悉的用于表存储的，并且允许计算机算法来获取指配给每个输入的输出的任何格式来存储。

[0227] 下面的表 I 提供来自比特标记符表的节选的示例，其中子单元的长度为 8 个比特。

[0228] 表 I

[0229]

比特标记符(所存储的)	子单元= 8 比特 (输入)
0101	00000001
1011	00000010
1100	00000011
1000	00000100
1010	00000101
11111101	11111101

[0230] 举例而言，并且在使用表 I 中所识别的子单元的情况下，如果输入为 00000101 00000100 00000101 00000101 00000001，那么输出便为 :1010 1000 1010 1010 0101。当比特标记符输出小于子单元输入时，所述比特标记符输出将在存储介质上占据更少的空间，并且因而节省存储比特所需要的存储空间和时间。

[0231] 正如本领域普通技术人员会认识到的，在给定比特标记符表 (如经过节选而产生表 I) 中，如果要使用所有比特组合，那么便需要 2^N 个条目，其中 N 对应于子单元内的比特数目。当存在 8 个比特时，便需要 256 个条目。当子单元中存在 16 个比特时，相关人员需要 2^{16} 个条目，这等于是 65536 个条目。当子单元中存在 32 个比特时，相关人员需要 2^{32} 个条目，这等于是 4294967296 个条目。如果相关人员知悉某些比特串将不在文件中使用，那么所述表便可以从最小的标记符开始来分配标记符。

[0232] 示例 2 :针对预处理子单元的比特标记符表 (预示性的)

[0233] 因为随着子单元大小变大，表也会变得更繁琐，所以在一些实施方案中，表可以经过配置，从而使得来自子单元列的一个末端的所有 0 都是缺失的，并且在存取所述表之前，从每个子单元的所述末端移除所有 0。因此，不是配置可供节选成表 I 的表，而是可以配置可供节选成表 II 的表。

[0234] 表 II

[0235]

比特标记符(输出)	预处理子单元
0101	00000001
1011	0000001
1100	00000011
1000	000001
1010	00000101
11111101	11111101

[0236] 正如相关人员可以看出的，在第二和第四行中，在子单元预处理之后，它们会具有少于八个比特。然而，从主机接收的原始数据中的实际子单元都具有八个比特。因为实施所述方法的系统可以经过设计而理解数字的不存在便意味着是 0，而且数字的所有不存在都是处于任何截断子单元的相同末端，所以相关人员可以使用占据更少空间并且仍然保持向独特子单元指配独特标记符的能力的表。因此，所述方法允许系统将 00000001(七个 0 和一个 1) 与 0000001(六个 0 和一个 1) 解译为是不同的。

[0237] 为了实施这个方法，相关人员可以将每个子单元 (或如果不使用子单元便是将每

个小盘）视为具有第一末端和第二末端。第一末端可以是比特串的右侧或左侧，而第二末端则是相反的一侧。出于说明性目的，相关人员可以认为第一末端是最左侧数字而第二末端是最右侧数字。在这个方法下，相关人员随后分析每个小盘的每个子单元内的一个或多个比特，以便确定第二末端处的比特是否具有数值 0。这个步骤可以称为预处理，并且在子单元预处理之后，它们会出现在表 II 的右列。如果第二末端处的比特具有数值 0，那么所述方法便可以移除第二末端处的比特以及具有数值 0 且与这个比特形成连续比特串的所有比特，从而针对起初在第二末端处具有 0 的任何子单元来形成修正子单元（表中的预处理子单元）。

[0238] 相关人员可以使用计算机算法，所述计算机算法核查每个子单元以便确定在第二末端处是否存在 0，并且如果存在，便移除所述 0 来形成预处理子单元，所述预处理子单元也可以称为修正子单元，其中修正的第二末端处在邻近于子单元第二末端的位置上。接着，所述算法核查修正子单元以便确定在其现在修正的第二末端处是否存在 0，并且如果存在，便移除所述 0 来形成另外的修正第二末端。在这个方法中，修正的第二末端会是之前邻近于第二末端处的比特的位置。任何另外的修正第二末端都会是远离子单元第二末端的两个或两个以上的地方。因此，术语“修正的”意味着缩短或截断的第二末端。所述算法可以对修正单元重复这个方法，直到生成在其第二末端处具有 1 的缩短小盘。

[0239] 正如本领域普通技术人员会认识到的，前述方法被描述成是通过如下方式加以应用的：从第二末端移除 0，直到 1 处于修正的第二末端或另外的修正第二末端处。所述方法可以反向设计，从而使得系统从第二末端移除 1，直到 0 处于修正的第二末端或另外的修正第二末端处。另外，在利用本公开的情况下，本领域普通技术人员可以从第一末端移除比特而不是从第二末端来移除，并使用所创建的表来将这些修正子单元转换成比特标记符。

[0240] 示例 3：频次互换（预示性的）

[0241] 基于经验性分析，相关人员可以确定从特定主机接收的一类文档或一组文档内或已经在给定时间范围内（例如，过去的一年或过去的两年）接收的一组文档内的每个子单元的频次。利用这个信息，而不是查看如表 I 或 II 所示出的、子单元按照数字顺序组织的表，相关人员可以查看频次转换器，其中更小的比特标记符是与预期最有可能出现在从特定主机接收的文件内、一类文档内或一组文档内的子单元关联。因此，在频次转换器内，标记符具有多个不同大小，并且具有更小大小的标记符是与更高频次的子单元相互关联。

[0242] 表 III：频次转换器

[0243]

比特标记符(输出)	频次	子单元=8 比特(输入)
0101	16%	00000001
1000	15%	00000010
11011	10%	00000011
10011101	0.00001%	00000100
10111110	0.00001%	00000101

[0244]

1100	15%	11111101

[0245] 表 III 是来自与表 I 使用相同子单元的频次转换器的节选的示例。然而，相关人员

将会注意到，比特标记符并不是依次指配的，并且实际上更大的比特标记符指配给更低频次的子单元。正如表所示出的，指配给子单元 00000011 的标记符比指配给子单元 00000001 的标记符大 25％，并且对于子单元 11111101 而言，尽管具有较高数值，但是其仍然接收更小的比特标记符，因为所述子单元在从特定主机接收的这些类型文件中更为频繁地出现。因此，如果相关人员使用表 I 并且子单元 11111101 出现在 10000 个地方，那么所述子单元将会对应于 111111010000 个比特。然而，如果相关人员使用表 III，那么只需要 11000000 个比特来用于相同信息的存储目的。尽管未展示在这个方法中，但是子单元也可以经过预处理来从一个末端或另一末端移除 0，并且所述表可以被设计成含有关联的截断子单元。

[0246] 正如上文所提到的，频次转换器可以基于被视为代表可能从一个或多个主机接收的数据的一组文件的分析来生成。在一些实施方案中，处理信息的算法可以执行其自身的品质控制，并且将用于来自给定时期的文档的子单元的实际频次，与频次转换器中标记符的分配所基于的那些频次进行比较。在使用统计分析的情况下，所述算法随后可以确定，对于将来的使用而言，是否应创建新表来重新安排标记符如何与子单元关联。正如本领域普通技术人员会认识到的，表 III 是频次转换器的简化节选。然而，在实践中，相关人员可以选择十六进制系统，从而获得相关性。另外，是为了方便读者而包括表所基于的频次的列举，并且这些频次不需要包括在本发明的各种实施方案所存取的表中。

[0247] 示例 4：中介器中空间的分配（预示性的）

[0248] 在大小为 1MB 的假设记录介质中，本领域普通技术人员可以按照如下方式来映射扇区：

[0249] 1MB 记录介质具有 1024000 个字节，这对应于 250 个扇区。(1024000/4096 = 250)。记录介质的几何特性可以按照如下方式汇总：容量 = (c*h*spt*ss)，其中：

[0250] c（柱面数目）= 7；

[0251] h（磁头数目）= 255；

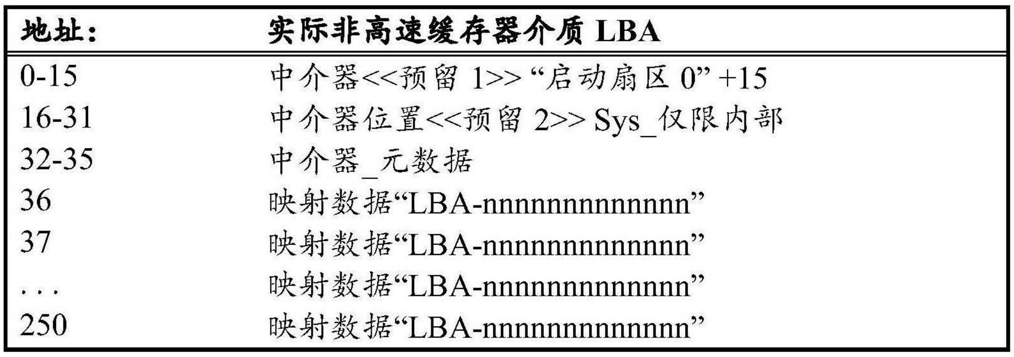
[0252] spt（每个磁道的扇区数）= 63；以及

[0253] ss（扇区大小，单位：字节）= 4096。

[0254] 在中介器内，扇区可以按照如下方式进行分配：

[0255] 表 IV

[0256]



地址:	实际非高速缓存器介质 LBA
0-15	中介器<<预留 1>> "启动扇区 0" +15
16-31	中介器位置<<预留 2>> Sys_仅限内部
32-35	中介器_元数据
36	映射数据"LBA-nnnnnnnnnnnn"
37	映射数据"LBA-nnnnnnnnnnnn"
...	映射数据"LBA-nnnnnnnnnnnn"
250	映射数据"LBA-nnnnnnnnnnnn"

[0257] 示例 5：空间节约

[0258] 本发明的系统接收具有 4096 个字节的 I/O 流中的 42500000 个数据区块。所述系统应用 MD5 散列算法来生成对应于 I/O 流中的 42500000 个区块的 7800000 个散列值区块。

[0259] 这会体现为仅仅使用存储起初的 42500000 个区块所需要的空间的 18.5％。应用冲突模块，并且所述冲突模块会验证不存在任何冲突，也就是不会针对不同区块来生成散列值的复制值。

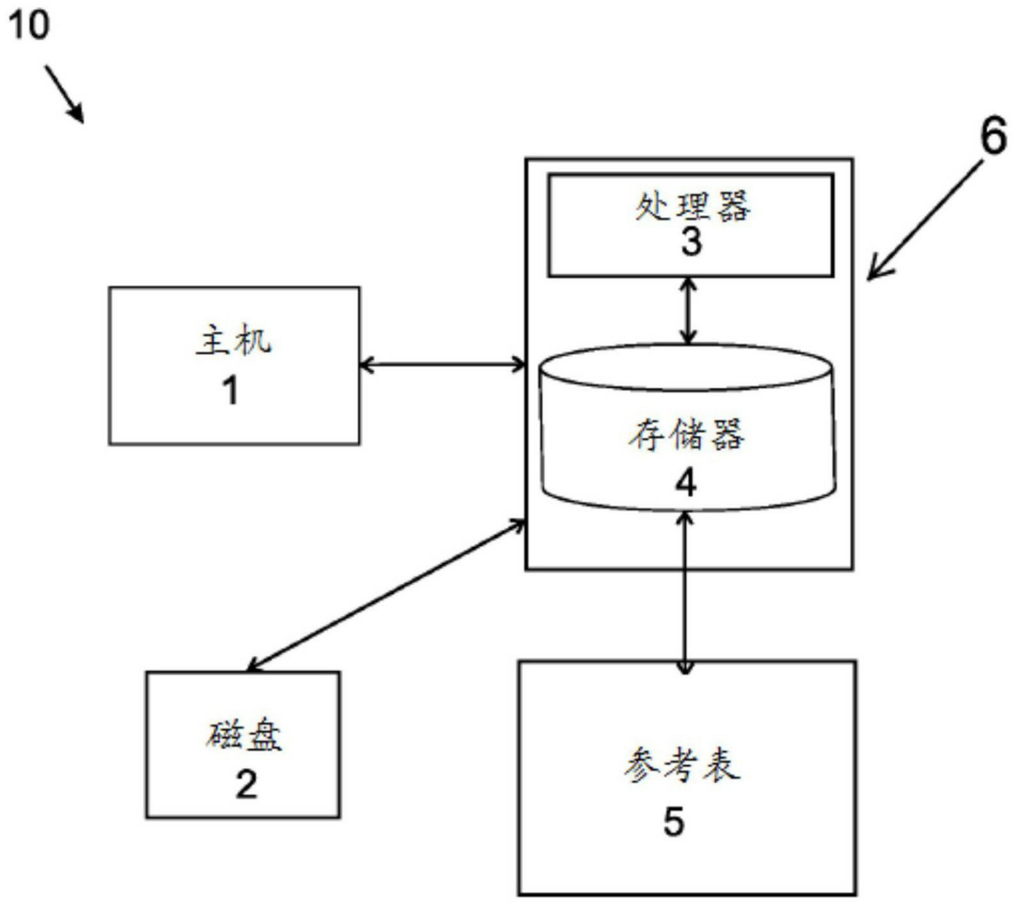


图 1

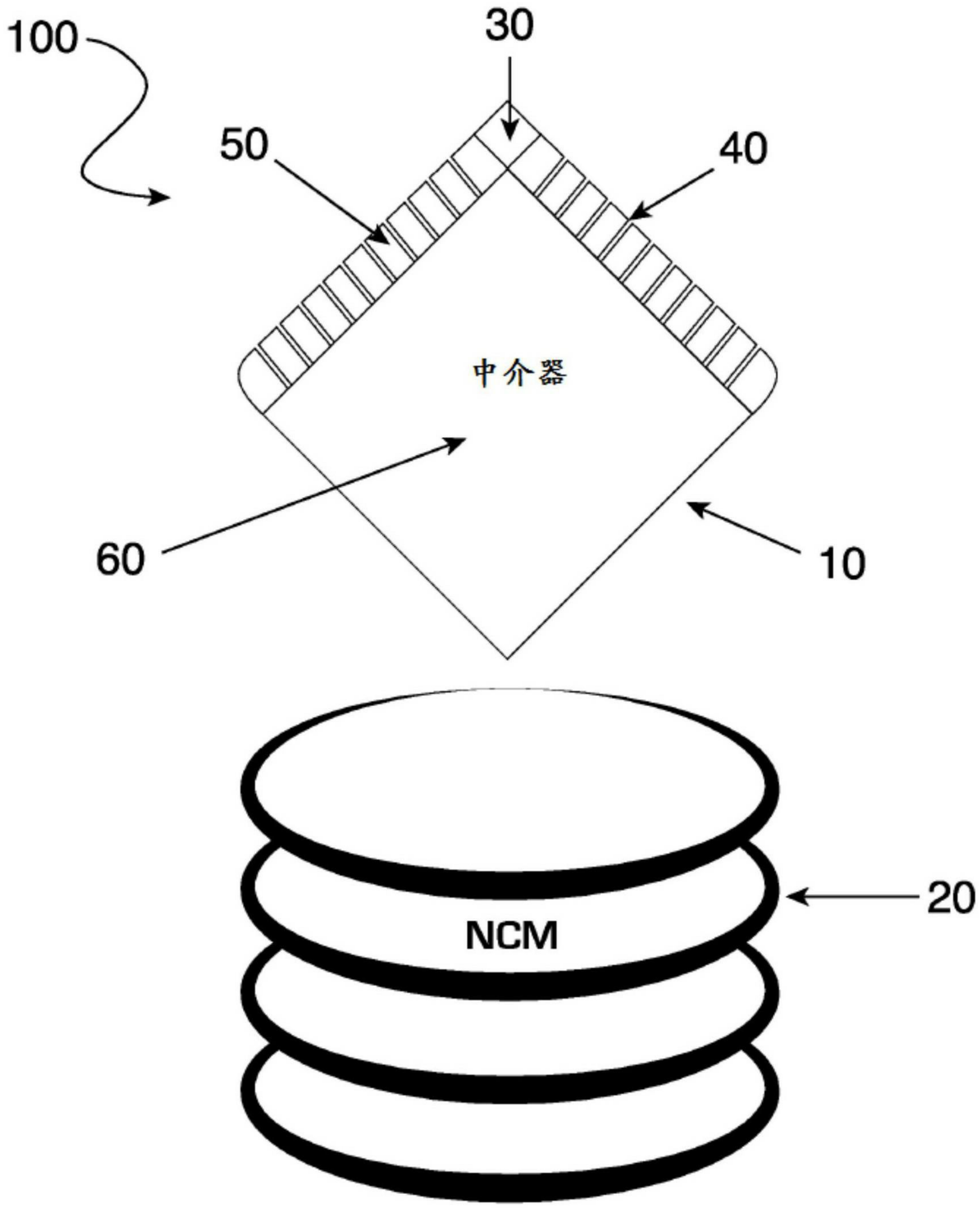


图 2

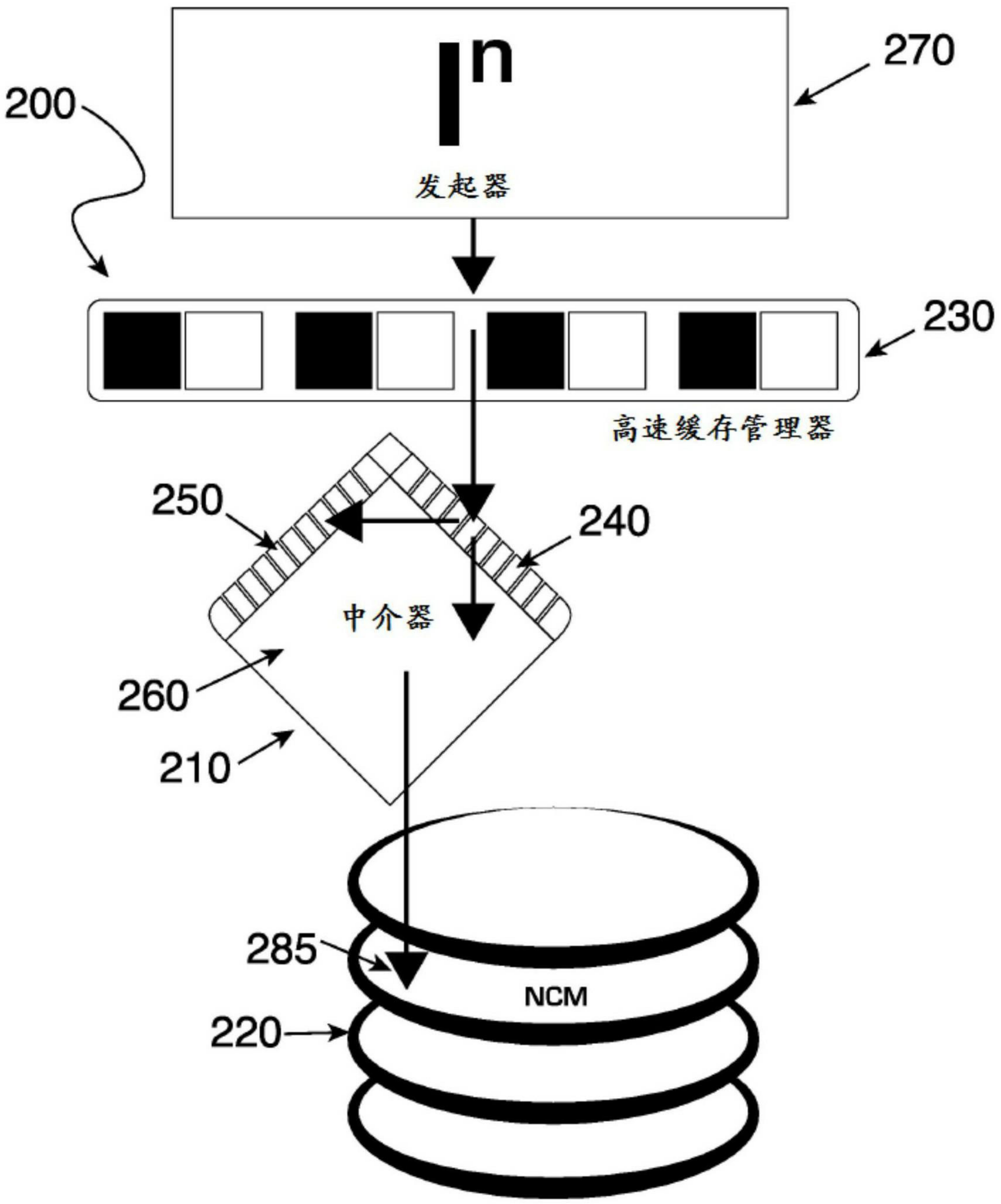
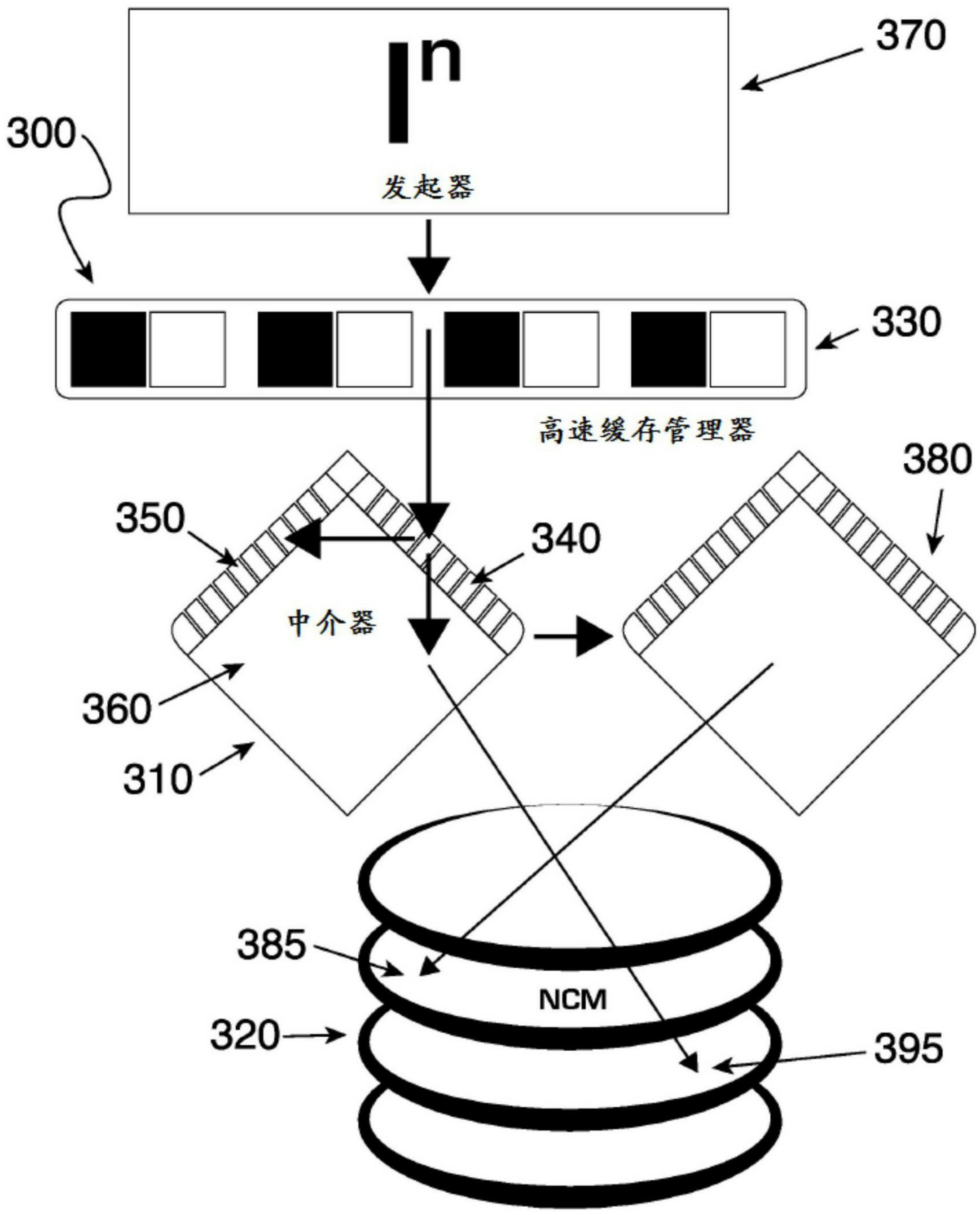


图 3



370
I^n
发起器
300
330
高速缓存管理器
380
350
340
中介器
360
310
385
NCM
320
395

图 4

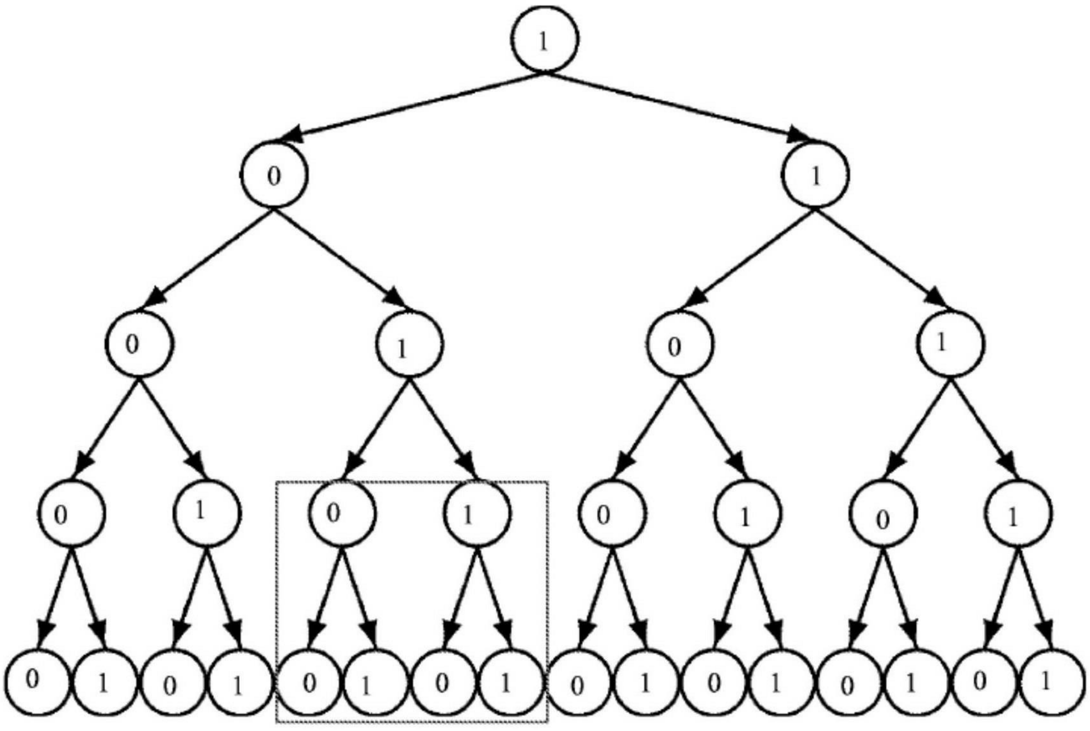


图 5

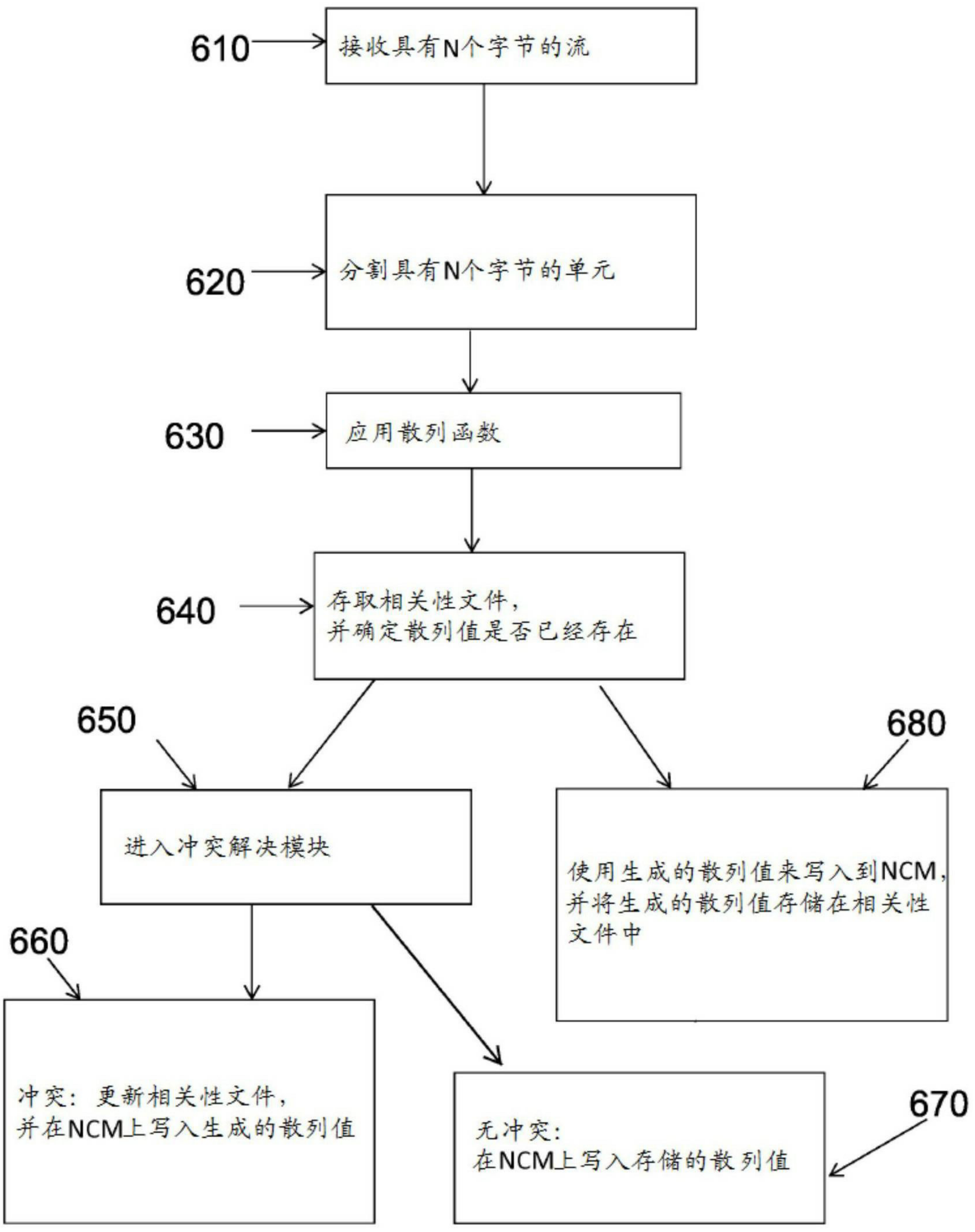

610
接收具有N个字节的流
620
分割具有N个字节的单元
630
应用散列函数
640
存取相关性文件，
并确定散列值是否已经存在
650
进入冲突解决模块
680
使用生成的散列值来写入到NCM，
并将生成的散列值存储在相关性
文件中
660
冲突：更新相关性文件，
并在NCM上写入生成的散列值
670
无冲突：
在NCM上写入存储的散列值

图 6