(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 2014/121109 A2

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

(21) International Application Number:
PCT/US2014/014225

(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

(71) Applicant: **SYMBOLIC IO CORPORATION** [US/US]; 379 Thornall Street, 10th Floor, Edison, NJ 08837 (US).

(72) Inventors: **IGNOMIRELLO, Brian**; 40 Manor Road, Colts Neck, NJ 07722 (US). **LIANG, S'uihong**; 64 Hillcrest Road, Martinsville, NJ 08836 (US).

(74) Agent: **LOCKE, Scott, D.**; Dorf & Nelson LLP, The International Corporate Center, 555 Theodore 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).

Published:

— *without international search report and to be republished upon receipt of that report (Rule 48.2(g))*

WO 2014/121109 A2

(54) Title: REDUCED REDUNDANCY IN STORED DATA

(57) Abstract: Through use of the technologies of the present invention, one is able to store data efficiently by the use of new and non-obvious methods, systems and computer program products that minimize the need to store duplicative data. Optionally, user data can be preprocessed in order to encode the data before entering a protocol for reducing the number of times that one stores redundant data.

Reduced Redundancy in Stored Data

[0001] Field of the Invention

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

[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 typically is stored on magnetic surfaces such as disks. These disks contain small regions that are sub-micrometer in size and are capable of storing individual binary pieces of data.

[0005] Within the vast amount of data that any given entity stores, often there is significant duplication of information. For example, the same company letterhead may appear in thousands of documents, and each file that corresponds to this data will contain the bits that code for the letterhead. Historically, many entities have accepted that this type of duplication exists in their files, and that the inefficiency of redundant storage of the same information is a cost of doing business.

[0006] As the cost for storage has been increasing and the availability of storage has been decreasing, entities have begun to explore means by which to store fewer than all of the duplicative information within or among files. In theory, entities that seek to avoid the storage of duplicative information or to minimize the number of times that duplicative information is stored could seek to identify unique bit or byte patterns within their data set and store the unique bit or byte patterns a minimal number of times. In order to carry out these methods, as new files are being prepared for storage, information within those files would be compared to reference sets of already stored information, and only if the bit or byte pattern that is being considered is unique, would it be stored. If it were not unique, then the redundant data would be replaced with a reference that is smaller in size than the data that points to the stored data of which it is a duplicate.

[0007] The goal of reducing the number of times that duplicative information is stored presents a number of challenges, including but not limited to: (1) maintaining a

sufficient speed by which to check for redundancy; (2) maintaining a sufficient speed in data reconstitution for retrieval; (3) ensuring that data is not lost during the processes of either checking for redundancy or storing information that corresponds to the original file; (4) protecting against unauthorized access to the stored information; and (5) providing efficient technologies and methods that may be used in connection with one or more if not all of taking snapshots of data, cloning data and restoring data. Various embodiments of the present invention are directed to overcoming one or more of these challenges.

[0008] Summary of the Invention

[0009] The present invention provides methods, systems and computer program products for improving the efficiency of storing and retrieving data, while minimizing the degree to which redundant data is unnecessarily stored a plurality of times. By using various embodiments of the present invention, one can efficiently store and access data. Through these various embodiments of the present invention one may transform data and/or change the physical devices on which transformed or converted data is stored. This may be accomplished through automated processes that employ a computer that comprises or is operably coupled to a computer program product that when executed carries out one or more 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 through one or more modules.

[0010] According to a first embodiment, the present invention is directed to a method for storing data on a non-cache recording medium, the method comprising: (i) receiving instructions to write data to a non-cache recording medium, wherein the instructions comprise a user perceived logical block address ("LBA") and a user supplied buffer, wherein the user supplied buffer consists of, for example, from 512 Bytes to 2 megabytes or from 512 Bytes to 64K; (ii) dividing the user supplied buffer into user supplied buffer units and applying a cryptographic hash function to the each of the user supplied buffer units, thereby generating a generated hash value; (iii) activating a computer program product that comprises an algorithm that causes the computer program product to access a hash value table and to determine whether the

generated hash value is duplicative of a stored hash value within the hash value table, wherein the hash value table correlates each of a plurality of stored hash values with a different stored buffer unit and a true logical block address; and (A) if the generated hash value is not within the hash value table, then writing the user supplied buffer unit to a block in a non-cache recording medium, updating the hash value table to include a correlation of the user supplied buffer unit, the generated hash value and a true logical block address at which the user supplied buffer unit is stored, and writing on a mediator the true logical block address that corresponds to where the user supplied buffer unit has been written and the user perceived logical block address (or addresses) for the user supplied buffer, and (B) if the generated hash value is duplicative of a stored hash value within the hash value table, querying whether there is a conflict, wherein a conflict is defined as the circumstance in which the same hash value is associated with a stored buffer unit and the current user supplied buffer unit and the two buffer units have different contents, and (a) if there is a conflict, writing the user supplied buffer unit to a block in the non-cache recording medium, rendering inactive or deleting an association within the hash value table between the stored buffer unit and the stored hash value, updating the hash value table to include a correlation of the user supplied buffer unit, the generated hash value and a true logical block address at which the user supplied buffer unit is stored, and writing on the mediator the true logical block address that corresponds to where the user supplied buffer unit has been written and the user perceived logical block address; and (b) if there is no conflict, writing on the mediator the true logical block address of a buffer unit stored on the non-cache recording medium that is the same as the user generated buffer unit and correlating it with the user perceived logical block address for the user supplied buffer without writing the user supplied buffer unit on the non-cache recording medium.

[0011] In step (A) when writing the user supplied buffer unit, persons of ordinary skill in the art will appreciate that the methods call for writing the user supplied buffer unit that has been determined not to be associated with a hash value in the table.

[0012] Often, the user will supply data (the user supplied buffer) that is in a stream or in units that are larger than the user supplied buffer units for which the hash value algorithm is configured to accept as input. In these cases, the user supplied buffer units may be formed by fragmenting (also referred to as breaking) the raw data sent

by a host into smaller units, which may be deemed the user supplied buffer units, regardless of whether the host fragments them or the systems or methods of the present invention do so. Thus, these fragmented user supplied buffer units may server as the input for the cryptographic hash function. By way of non-limiting example, the user supplied data may be from 16K to 2MB and each fragmented user supplied buffer unit is from 512 Bytes to 4K, *e.g.,* 512 Bytes or 4K. Thus, in some embodiments, the fragmented user supplied buffer unit is no more than ¼th or no more than 1/16th or no more than 1/64th the size of the user supplied buffer prior to fragmentation. If the step of fragmentation is used prior to entering the hash value algorithm, then the hash value table would contain a correlation of the user supplied buffer units and hash values, the writings to the storage device would be fragmented user supplied buffer units and not the larger data buffer units and the mediator would correlate the user perceived address (or addresses) of the user supplied buffer units with the plurality of fragmented user supplied buffer units.

[0013] The various steps of the methods of the present invention may be stored in one or more modules, *e.g.,* a receipt of buffer and user perceived logical block address module, a fragmentation module, a hash value search module, a duplication of a hash value analysis module, a conflict model, and a writing module. Similarly, there may be a reading and reconstitution of files module. These modules may be stored in a non-transitory medium in the form of executable code.

[0014] According to a second embodiment, the present invention provides a system for storing data, wherein the system comprises: (a) persistent memory, wherein the persistent memory stores a hash value table that is configured to associate a stored buffer unit with a stored hash value and a true logical block address; (b) a central processing unit that comprises or is operably coupled to a computer program product that is stored in a non-transitory medium, wherein the computer program product comprises executable code that when executed, automatically, (i) applies a hash value algorithm to each of one or more user supplied buffer units to generate a generated hash value; and (ii) determines whether the generated hash value is a duplicate of a stored hash value within the hash value table that is associated with a stored buffer unit, and if so, determines whether a conflict exists, wherein the conflict is defined as a hash value being associated with two different buffer units and if a conflict exists, updating the hash value table to cause the hash value within the table to be associated

with the user supplied buffer unit and not the stored buffer unit; (c) a non-cache recording medium, wherein the non-cache recording medium is configured for block level storage; and (d) a mediator, wherein the mediator stores a correlation of a true logical block address with a user perceived logical block address.

[0015] According to a third embodiment, the present invention provides a computer program product comprising a non-transitory computer useable medium including a computer readable program, wherein, the computer readable program when executed on a computer causes the computer to implement a method for de-duplicating and managing data blocks within a file system comprising any of the methods of the present invention. These methods may be organized in one or more modules.

[0016] Through the various embodiments of the present invention, one can increase the efficiency of storing and retrieving data, because in most circumstances, large buffer units that are duplicative of previously written data will not need to be rewritten to a non-cache recording medium (NCM). Instead, a pointer on a mediator that is smaller in size than and is within a structure that is physically separate from the storage unit that houses the data to which it points will direct the computer to a previously stored copy of that data. The increased efficiency may be realized by using less storage space than is used in commonly applied methods and investing less time and effort in the activity of storing and/or retrieving information. Further in some embodiments, the present invention leads to increased speed in storing and retrieving documents. Thus, the technologies and methodologies of the present invention help to reduce the total amount of physical storage that is required to store data. This is accomplished by minimizing the number of times that duplicative data is written and is stored and, in the cases in which there is duplicative data, using a mediator to point to previously stored data.

[0017] Brief Description of the Figures

[0018] **Figure 1** is a representation of a method for writing data according to an embodiment of the present invention.

[0019] **Figure 2** is a representation of a protocol for resolving a conflict according to a method of the present invention.

[0020] **Figure 3** is a representation of a method for reading information according to an embodiment of the present invention.

[0021] **Detailed Description of the Invention**

[0022] 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.

[0023] *Definitions*

[0024] Unless otherwise stated or implicit from context the following terms and phrases have the meanings provided below.

[0025] The term "bit" refers to a binary digit. It can have one of two values. Each value may be represented by either 0 or 1.

[0026] The term "block" refers to a sequence of bytes or bits of data having a predetermined length. On a recording medium, the physical media may be divided into units that are defined by a block size. Each block on a recording medium may be identified by a logical block address. In the industry, currently, 512 bytes is the standard size of a block. However, there is a movement to using 4096 bytes as a standard. Additionally, as persons of ordinary skill in the art will appreciate, frequently the phrases "block size" and "sector size" are used interchangeably by persons of ordinary skill in the art.

[0027] 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.

[0028] The phrase "buffer unit" refers to a series of bits that are of a size that is compatible for use as input into a hash value algorithm. The buffer unit may be the same size as a chunklet. However, in some embodiments, it may be a fraction of the size of a chunklet or a multiple of a size of a chunklet.

[0029] The term "byte" refers to a sequence of eight bits.

[0030] 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. In some embodiments, when the methods of the present invention are implemented, a hash value table resides in L2 cache.

[0031] 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 defined the cylinder, head and sector at which they appeared on hard disks. More recently, the chunklet size has been derived from the logical block address (LBA) measurement. 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.

[0032] The term "conflict" refers to the occurrence of the generation of the same output, *e.g.*, hash value by a function, such as a hash value algorithm, for different inputs, *e.g.*, buffer units.

[0033] A "file" is a collection of related bytes or bits that combine to provide a file of a size with a length that may be measured in bits or bytes. A file may be smaller than a chunklet, the same size as a chunklet or larger than a chunklet.

[0034] 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.

[0035] 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.

[0036] The phrases "hash function," "cryptographic hash function," "cryptographic hash function value algorithm" and "hash function value algorithm" refer to an algorithm or subroutine that maps large data sets (of the same or variable lengths) 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. Thus, in some embodiments the hash value is from 32 -512 bits in length.

[0037] The terms "host," "user" 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 receive data. Preferably, within any input/output ("I/O") stream, the data corresponds to a file of a single document type.

[0038] The terms "including" and "comprising" are used in an open-ended fashion and thus should be interpreted to mean "including but not limited to."

[0039] The abbreviation "LBA" refers to "logical block addressing" or a "logical block address." 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. A user may believe that data is stored on a particular LBA. The location at which a user perceives data to be stored is a "user perceived logical block address." This may be different from where the data is actually stored. The location at which data is actually stored on a NCM may be referred to as a "true logical block address."

[0040] 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.

[0041] 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 or processing data. It 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.

[0042] 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. Thus, it may correlate a user perceived LBA with a true LBA. 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 exists 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 translate, to convert 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 on L2 cache.

[0043] 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.

[0044] The abbreviation "NCM" refers to a non-cache recording medium. Examples of NCMs include, but are not limited to, hard disks and solid state drives. An NCM may, for example, be configured to hold 100 terabytes bytes of data. The NCM stores the unique buffer units. In some embodiments, the NCM also stores a digest map, which contains the associations of the buffer units and stored hash values. These stored associations may be used to populate the hash value table in the RAM of a server. By populating the RAM with this information from the persistent storage of the NCM, fast loading may be accomplished. Additionally or alternatively, the NCM stores a bit map of 1 bit per unit block, which may be used to track the storage savings of the present invention. As a matter of practicality, the storage of the digest map and bit map require a small overhead of between 5 and 10 bytes, *e.g.,* 8.125 bytes per block of stored data on the NCM. Alternatively, the digest map and bit map may be stored on different storage media than store the buffer units that are written as a results of the methods or uses of systems of the present invention.

[0045] The phrase "operably coupled" is used interchangeably with the term "coupled" and 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. The phrase and term include indirect, direct, optical, wired or wireless connections. Thus, if a first device is operably coupled to a second device, that connection may be through a direct electrical connection, through an indirect electrical connection via other devices and connections, through an optical connection or through a wireless connection or a combination thereof.

[0046] 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.

[0047] 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.

[0048] 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 NCM, such as a hard drive, a hard disk, a floppy disk, a computer tape, ROM, EEPROM, nonvolatile RAM, CD-ROM and a punch card.

[0049] 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.

[0050] 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 an iSCSI (internet small computer system interface) protocol. In some embodiments of the present invention, the sector map may be located in a bit field of a mediator.

[0051] 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.

[0052] *Preferred embodiments*

[0053] 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 by decreasing the number of times that duplicative data is stored.

[0054] According to one embodiment, the present invention provides a method for storing data on a non-cache recording medium. In some embodiments, the data that is received is comprised of, consists essentially of, or consists of a user defined logical block address (LBA) and a user supplied buffer. The size of the user supplied buffer may, for example, be from 512 Bytes to 64K or even higher, *e.g.,* up to 2 megabytes. The data that is received may be in the format of (LBA_x, $buffer_x$), wherein x=1 to n and n= the number of buffer that the user sends. The buffers are raw data and thus they may correspond to any document type, *e.g.,* JPEGs, PDFs, WORD documents, MPEGs and TXT documents.

[0055] The total stream that a user sends may be in the form N Bytes. The stream may be received over a network that is wired or wireless and through known methods and technologies for transmitting I/O streams. The user may format the data in the format: (LBA_x, $buffer_x$) or the user may send the data to a server that formats the data into this format. Thus, N may be larger than the size of a buffer unit. Preferably, the user transmits the data with a file name and/or file identifier.

[0056] When N is larger than the size of the buffer unit, after receipt of the stream of data, the server may fragment the N Bytes into buffer units. Thus, a buffer is a stream of data that may be of any size, but a buffer unit is of a fixed size and corresponds to the size of the storage units on the NCM. For example, a user may send a single buffer of 1024 Bytes with a designation for the file to begin a LBA10. If the buffer unit size of the implementation of the invention is 512 Bytes, the data may be stored at LBA20 and LBA21. Notably, in some embodiments, a user may transmit only the starting LBA and its system will perceive the file to be stored on the NCM beginning at that address and stretching to consecutive blocks so that there is sufficient room for the file.

[0057] This fragmentation may be performed on the total stream or on chunklets. For example, if the method is configured to receive buffer units of 4K in size, but the user transmits chunklets that are 16K in size, after or as the server receives the chunklets, it will initiate a protocol for dividing each chunklet into four buffer units that are each 4K in size. Thus, according to some methods, one receives the data in the form of (LBA_x, $buffer_x$) or converts the data that it receives into this form. Regardless of whether the user supplies the data in packets that are configured to be of the requisite buffer unit size, or are converted to that size, each of the buffer units

that serves as input for the hash value algorithm is referred to as a "user supplied buffer unit."

[0058] After the data is received in (or converted into) the correct form, a cryptographic hash function value algorithm is applied to each buffer unit to form a generated hash value for that buffer unit. The hash value that is generated may be referred to as a generated hash value. The cryptographic hash value algorithm may, for example, be in the form of a computer program product or protocol within a computer program product that is stored in a non-transitory storage medium. Examples of these types of 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 hash value, a checksum or a sum. In some examples, the hash value is 64, 128 or 256 bits or 8 Bytes in size or any value in between. Because of the highly repetitive nature of data within I/O streams, the probability of generating conflicting hash values, *i.e.* hash values that are the same but correspond to different buffer units, is relatively low. The method may obtain hash values according to a first in first out ("FIFO") protocol and either begins accessing the correlation file while an I/O stream is being received, while hash values are being generated, or after all I/O streams have been received, fragmented, if necessary, and subjected to the hash function value algorithm.

[0059] After a generated hash value is obtained for a user supplied buffer unit, a different computer program product or a different module within the same computer program product that produced the generated hash value is accessed. This computer program product accesses a hash value table. The hash value table may, for example, be stored in persistent memory, and called into L2 cache for access and use. Within the hash value table, a plurality of stored hash values are each associated with a different stored buffer unit within a set of stored buffer units and a true LBA of the buffer unit on a NCM. The phrase, "stored hash value" is used to contrast a hash value that is generated by the hash value algorithm for a particular user supplied buffer unit, which may be referred to as a generated hash value.

[0060] A hash value table may initially be populated with a set of known hash values and buffer units as associated by a particular hash value algorithm. These known values may have been determined based on empirical prior uses of the algorithm, *e.g.*, those previously generated for a host's files or the files of a host in a similar industry.

Alternatively, it may initially be empty and the first user supplied buffer unit is compared to a null set of associations. When in use, the hash value table may reside in RAM on for example, a server. Thus, the hash value table or the data to populate it may reside in persistent storage, such as on the NCM that stores buffer units or elsewhere in a format that can be accessed for repopulation of RAM upon booting or rebooting of the system. When updated, the updates are made to the table in RAM and also to the persistent memory.

[0061] In some embodiments, at any time, within the table, a given stored hash value is actively associated with no more than one buffer unit. When a buffer unit is within the hash value table and is associated with a stored hash value, the buffer unit is a "stored buffer unit." The buffer unit that is associated with a particular hash value within a hash value table may change over time, and in the case of a conflict as described below, the most recent buffer unit, hash value association determined by the hash value algorithm will be, after application of the methods described herein, the active association within the table. The phrases "active association" and "actively associated" refer to the condition by which the table provides, configures or denotes data to be extracted for comparison to a generated hash value. The active association may include the true logical block address on the NCM that most recently received the buffer unit that led to generation of the hash value upon application of the hash value algorithm. Once a hash value becomes a stored hash value, it will remain in the table; however, over time, the buffer unit with which it is associated may change. Thus, a stored buffer unit might not remain a stored buffer unit.

[0062] Persons of ordinary skill in the art will recognize that in certain embodiments, rather than using a hash value table, one may use a multimap. When using a multimap, a hash value may be associated with more than one buffer unit and thus previously stored hash values need not be removed when there is a conflict.

[0063] Because a hash value algorithm can generate the same hash value for a plurality of different buffer units, a procedure is needed to determine how to treat these occurrences within the table. When a hash value has been generated for a user supplied buffer unit that is the same as a stored hash value, one of three procedures may be used: (1) the pre-existing association may be written over with the new association; (2) the pre-existing association may be deleted, and a new entry may be made in the table at for example a new location; or (3) the pre-existing association

may be moved to an inactive file or otherwise modified to denote that it is inactive, but the information is retained in, for example, an archive file, and through appropriately designed computer programs, can be accessed at a later time. Conditions (1) or (3) are examples of rendering an association inactive. As persons of ordinary skill in the art will recognize, because data retrieval from the NCM does not require access to the hash value table, any archived information is not needed for implementation of various embodiments of the present invention. However, this information may be used to review the degree to which the same hash value is generated for different user supplied buffer units or fragmented user supplied buffer units.

[0064] Due to the highly repetitive nature of data, the probability of conflicting hash values being generated as a result of application of the hash value algorithm is low. For example, SHA-1's 160 bit hash has a probability of randomly generating the same hash value for different patterns of 1 in 10^{24}. The present invention makes use of this feature of hash value algorithms in order to minimize the extent to which duplicative buffer units are to be stored. In order to accomplish this, the methods provide for the allocation of two modules or two separate computer program algorithms that query: (1) is the newly generated hash value the same as a stored hash value for which there is a stored buffer unit?; and if so, (2) is there a conflict such that the stored buffer unit for that hash value differs from the user supplied buffer unit?

[0065] The outcomes of the queries discussed above determine what new information is to be stored on the NCM and under what circumstances, as well as what is written to the mediator. In the simplest case, the protocols of the present invention determine that there is not a duplication of hash values. The absence of duplication of hash values indicates that the user supplied buffer unit is not within the hash value table. Consequently, the buffer unit is written in a new block of the NCM, and an update is made to the hash value table so that should the user later submit the same buffer unit as part of a different data stream or data packet or later within the same data stream, the methodologies will be able to detect that it is the same as the data that is already stored on the NCM. The hash value table is also updated to include the true LBA at which the buffer unit is written. Upon writing a generated hash value and a user supplied buffer unit to the hash value table, they become a stored hash value and a stored buffer unit, respectively.

[0066] If the methodologies determine that there is a duplication of hash values, then the second query is made. As noted above, in this second query, the method considers whether application of the algorithm to the user supplied buffer unit leads to production of the same hash value with which an already stored buffer unit is associated, even though the two buffer units are different.

[0067] As persons of ordinary skill in the art will readily recognize, this two tier approach introduces efficiency into methods for reducing the number of times that duplicative data is stored on a NCM. In the first step, hash values are compared. These values are smaller than the buffer units, *e.g.,* at least 2x, at least 10x, at least 100x or at least 1000x smaller than the buffer units, and thus, are easier to compare than the buffer unit themselves. Only if those values indicate a duplication of hash values, does the system compare the actual buffer units. Thus, by weeding out the buffer units that are associated with hash values that are not already in the hash value table prior to checking the buffer units against each other, the system is efficient.

[0068] In the second tier of the queries, one compares two buffer units to each other. If there is no conflict, *i.e.,* there is a true identity of buffer units (stored and current user supplied); then no additional step of writing to the NCM is needed. Instead, there need only be a notation made to a mediator of the block at which the buffer unit was previously stored. Thus, there is increased efficiency because these buffer units will not need to be written again. In order to keep track of the file to which it belongs, within the mediator, the LBA is associated with the user perceived LBA and optionally, the user generated file name or file system.

[0069] The other outcome of the second tier of queries occurs when a stored hash value (as associated with a stored buffer unit) is the same as a generated hash value but there is a conflict because the stored buffer unit and user supplied buffer unit are different. In these cases, the methods of the present invention deem the user supplied buffer unit as needing to be written to the NCM. They also cause the hash value table to be changed so that it associates the common hash value with the user supplied buffer unit. The previous association of the common hash value is rendered inactive or deleted. Consequently, in subsequent queries that search for duplicative buffer units, only the more recently stored hash value, buffer unit association is considered. In these methods, no extension of hash values is ever needed in the case of conflicts.

[0070] In various embodiments, for each LBA_x, the method causes the mediator to store an LBA_y with the LBA_x, wherein LBA_y refers to the actual location of the buffer unit that is identical to the user supplied buffer unit. Because the LBA_x is where the user believes that a buffer unit is stored and the LBA_y is where the buffer unit is actually stored, the LBA_y is also stored within the hash value table. However, the LBA_x may be omitted from the hash value table and in some embodiments, only exist on the mediator. Optionally, the mediator also stores the user created file name and/or file identifier and associates that file name with one or more user perceived logical block addresses.

[0071] As noted above, most buffer units are stored on an NCM only once. Thus, for a plurality of user files, there are correlations of user perceived logical block addresses and true logical block addresses, and within a plurality of the correlations (on the mediator) that correspond to different user files, there are one or more of the same true logical block addresses but different user perceived logical block addresses. The most highly repetitive buffer units in actual user files will appear in the greatest number of different correlations. Additionally, because a user may supply a buffer and believe that it is being stored at consecutive sites on an NCM, it may record association of a single LBAx with that buffer and view the data as being stored at consecutive LBAs beginning at LBAx. However, because storage is actually on the buffer unit level and frequently for a given buffer won't be stored at consecutive locations, the mediator may store a single user perceived LBA (or implicitly or explicitly) a plurality of consecutive LBAs with a plurality of true LBAs that are not-consecutive. For example, the user may supply a buffer of size 4096B and a user perceived LBA of 10. If the buffer units are 512B in size, the user may implicitly believe that its data is at eight consecutive storage sites beginning at LBA 10. However, in reality, they may be at LBA4, LBA3, LBA2, LBA2, LBA3, LBA3, LBA 9, LBA4, and the mediator would point to those locations. Notably, for most if not all duplicative buffer units that correspond to data within a buffer that a user sends, the mediator would point to the same LBA on the NCM. Thus, on the mediator, there may in some embodiments be correlation of all true locations of data (*e.g.,* all true LBAs) with the user perceived location or locations as received from the user.

[0072] Thus in these methods, rather than using extensions to take into account the possibility of the hash value algorithm generating the same hash value, the above

described use of the most recent occurrence of the association writes a particular supplied buffer unit to a NCM only when: (i) the comparable hash values are not already within the hash value table as associated with any buffer unit; or (ii) for any given hash value, there is a conflict. In these latter cases, there may be some writing of the same data to the NCM that has previously been written. However, as a matter of practicality, this will only rarely occur.

[0073] In some embodiments, writing to the NCM of buffer units is contiguous, *i.e.*, each user supplied buffer unit that is to be written, may be written in the next contiguous block on the NCM. Therefore, there is minimal or no scattering of data on the NCM, which improves read/write performance and permits a savings of storage space. Additionally, read/write performance is improved because less is actually written to the NCM. This in turn enables an operating system's cache to function better. Furthermore, there can be increased security of the data because the data on the NCM cannot be used to reconstitute a file in the absence of the mediator and the hash value table.

[0074] Further benefits of the present invention may be appreciated if one considers how data is read. A user may send a request to read a file. The request includes a file identifier and information pertaining to one or more user perceived logical block addresses. By accessing the relevant mediator, one can determine the actual logical block address(es) and retrieve the relevant data. Notably, in contrast to writing protocols, no hash value algorithm or table is needed during the retrieval and reading steps. Instead, the reader retrieves data from the sites on the NCM to which the mediator points, and the mediator may point to one or more buffer units a plurality of times for one or more files.

[0075] The present invention also provides systems for the efficient storage of data. These systems may comprise: (a) persistent memory; (b) a central processing unit ("CPU"); (c) a non-cache recording medium; and (d) a mediator. Each of the components may be operably coupled to one or more other components in order to carry out their designated functions.

[0076] The persistent memory comprises the hash value table and may be part of or distinct from the non-cache recording medium described below. The hash value table associates each of a plurality of stored hash values with a different stored buffer unit.

The hash value is determined by the application of a hash value algorithm to a buffer unit. Each stored buffer unit is also associated with a true logical block address. Persons of ordinary skill in the art will recognize that the hash value table may contain associations among the three types of data: stored hash value, a stored buffer unit and true logical block address, in one table. Alternatively, the hash value table may, in one table, contain an association of only the first two types of data and in another table, in the same or a different memory device (*e.g.*, the mediator) there may be stored a table that correlates the true LBA and the stored buffer unit. The persistent memory may be stored within the CPU or operably coupled to the CPU.

[0077] The central processing unit is comprised of hardware or a combination of hardware and software. The CPU is configured to access the persistent memory and to search the hash value table. The CPU is also configured to execute one or more of the methods of the present invention, including but not limited to writing to the NCM and mediator. Furthermore, the CPU is configured to communicate with one or more remote users through a wireless or wired network, *e.g.,* through a server.

[0078] The NCM is configured for block level storage. The NCM may be separate from or part of the CPU.

[0079] In some embodiments of the above described systems, each of the mediators, the hash value table, CPU and the non-cache recording medium are stored remotely from one another. They may be in separate structures that are within the same housing or in different housings.

[0080] As noted above, the system of the present invention (which may be controlled through a server) can, after consulting the mediator and without accessing the hash value table, recreate the file and transmit the data to a user. Thus, in one embodiment, a system of the present invention comprises a retrieval module, wherein the retrieval module is configured to reconstitute a data file by accessing the mediator and recombining a plurality of buffer units in an order dictated by the mediator without accessing a hash value table and wherein the order dictated by the mediator is different from the order of the buffer units on the non-cache recording medium.

[0081] The various methods of the present invention may be controlled automatically by a manager. The manager may comprise one or more modules and reside on a local computer, on a network or in a cloud or for example, in the CPU.

The manager may be configured to coordinate receipt of or to receive 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 for the de-duplication methods of the present invention and to a mediator or to other components of the system at the direction of the manager, but it does not flow through the manager.

[0082] 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.

[0083] The mediator, may for example, comprise: (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. The manager causes file system information, bootability information and partitioning information to be stored in a first set of tracks on the mediator, which may be referred to as reserve 1 or R_1. This information may include an identification of file system information, which 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.

[0084] In these embodiments, the manager may also receive metadata in addition to the parameters described in the preceding paragraph. The metadata may be 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 and one or more user perceived LBAs. 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, including stores the true and user perceived LBAs in a bit field of the fourth set of tracks.

[0085] 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 system or a part of a system that executes one or more of the embodiments of the present invention, after going through the de-duplication protocols of the present invention, and writes to a mediator and in some cases writes to a non-cache medium (NCM) for long term storage. The NCM may in turn send an acknowledgement back to L2 cache.

[0086] 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 or be on a solid state drive or any block device for storage.

[0087] As persons of ordinary skill in the art know, 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.

[0088] 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.

[0089] 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).

[0090] R_1 may be configured to function as the central point for host initiation. Thus, prior to any of the de-duplication methods of the present invention, 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. Preferably, R_2 is 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.

[0091] In some embodiments, R_1 contains unique reserve sector information and partition information. Within the partition information, one may store the file system information.

[0092] 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.

[0093] The formatting program allocates the first 16 sectors for the $Boot metadata file. The first sector is a boot sector with a bootstrap code, and the following 15 sectors are the boot sector's IPL (initial program loader).

[0094] 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.

[0095] Finally, the mediator may also comprise 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 begins 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.

[0096] As a matter of practice, preferably the mediator is not located on the disk or recording medium on which the buffer unit 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.

[0097] For purposes of further illustration, reference may be made to the figures. **Figure 1** is a representation of a method of the present invention, the instructions for which may be stored in persistent storage in a non-transitory recording medium. For illustrative purposes, the method shown in **figure 1** from step **130** to the end is shown for a single user supplied buffer unit; however, the method may be repeated a plurality of times for a given buffer that is fragmented into a plurality of user supplied buffer units. When the associations of the user perceived LBA and the true LBA are written to the mediator, they associations are grouped together and denoted as corresponding to a particular file. As persons of ordinary skill in the art will recognize, if a user supplied buffer is the same size as a user supplied buffer unit, it need not be fragmented and the various embodiments of the present invention can be configured to check for this condition. Various embodiments of the present invention may also be configured to confirm that all buffers are capable of being divided into buffer units of equal size and if not, then adding 0's to the end of the string of bits of the buffer or what would be the last buffer unit to render them all of the same size.

[0098] As the figure shows, instructions may be received to write information to a storage medium. These instructions may be in the form of a user perceived logical block address (LBA) or (LBA_x) and a user supplied buffer unit **110**.

[0099] As described above, the user perceived LBA is the location at which the user believes that his or her data will be stored. The data within the user supplied buffer unit is typically going to be of the size of the blocks on the device on which it will be stored. If the user submits data that is larger than the block size on the device, the data that the user submits may be pre-processed so that it contains a plurality of buffer units, each of the size of the block. If the buffer unit is smaller than the block size, then the computer program product may add all zeroes to one end of the buffer unit until it is the size of a block.

[00100] The received instructions are thus in a form or converted to a form that may be represented by (LBA, buffer). For the user supplied buffer unit, the computer program product fragments the data as received, if necessary, and calculates a hash value **120**. When a plurality of instructions is received, a hash value is calculated for each buffer unit.

[00101] After the algorithm generates a hash value, it queries whether the hash value is duplicative of a stored hash value that already exists within a hash value table **130**. If the table does not contain the generated hash value, the algorithm concludes that the buffer unit is new to the NCM and writes the buffer unit to a block within the NCM that has not previously been used **140**. The algorithm also causes the hash value table that is stored in memory to be updated to include an association of this user generated buffer unit that has been newly written to the NCM with its generated hash value **150**. This generated hash value becomes a stored hash value, and the user supplied buffer unit becomes a stored buffer unit. In some embodiments, the hash value table also identifies the true logical block address at which the buffer unit is stored on the NCM, whereas in other embodiments, the hash value table excludes this information, and it is stored elsewhere, *e.g.*, on the mediator or in another data file.

[00102] After the hash value table has been updated, the user perceived LBA and the true LBA at which the user supplied buffer unit is stored are stored on a mediator and correlated with each other **160**.

[00103] Returning back to step **130,** within **figure 1**, if the query results in a conclusion that the newly calculated hash value is duplicative of a stored hash value within the hash value table, then the methods initiate a protocol that asks whether there is a conflict **170**.

[00104] If there is no conflict, then the protocol causes the mediator to store a correlation of the location of the buffer unit data that is already on the NCM and the same as the user supplied buffer unit **160** with the user perceived location. Thus, for this duplicative data, no new information will be written to the NCM.

[00105] Returning to the query as to whether there is a conflict **170**, when the response is yes, meaning that the same hash value is assigned to different buffer units, the protocol calls for writing of the user supplied buffer unit to the NCM **140** and updating of the hash value table in memory to a hash value actively associated with

the newly written buffer unit. Additionally, the mediator will be updated to contain the new correlations of the user perceived location(s) and true location(s).

[00106] **Figure 2** further illustrates this step. Similar to **figure 1** upon entry into a conflict determination protocol **270**, if there is a yes output, then the unique data of the buffer unit is written to a new block on the NCM **240**. Following writing to the new block, the algorithm updates the hash value table.

[00107] When updating the hash value table, the method disassociates the previously stored hash value with a buffer unit, associates the stored hash value with the more recently received buffer unit **251**, and removes or inactivates the association of the hash value with the previously stored buffer unit **256**. After the hash value table has been updated, the protocol writes the true logical block address of the most recently received buffer unit to the mediator and associates it with the user perceived logical block address **260**.

[00108] **Figure 3** represents a reading instruction. The system may receive instructions to read a block, *i.e.,* retrieve information of: (LBA, buffer) **310**. The request may come in the form of a request for a file, by a file name, which is the user's system association with one or more user perceived LBAs.

[00109] A protocol causes accessing of a correlation table within a mediator and reading at each LBA location that corresponds to the buffer or portion of it **320**. Because the correlation table correlates user perceived LBAs with true LBAs, after the information for an LBA (or LBAs) is obtained, the protocol reads the NCM at the specified actual LBA or LBAs, retrieves the block information and fills the buffer parameter **330**. With this information, the system will have the raw data in the correct order and be able to send it to the host for reconstitution of the requested file through the host's operating system.

[00110] Notably, during the reading steps, there is no need to access a hash value algorithm or table. Instead, the LBAs as stored on the NCM may be read to retrieve the buffer units for which a user call. If any one or more buffer units are still within either L1 or L2 cache at the time of reading, then they may be read from the appropriate cache rather than from the NCM.

[00111] By way of further example, one can consider the following table:

Table 1

User Perceived LBA	User Supplied Buffer Unit	Hash Value	True LBA Location
1	A	x	100
2	B	y	110
3	C	z	111
4	D	a	112
5	E	b	113
6	F	c	114
7	G	x	115
8	H	d	116
9	I	e	117
10	J	y	110
11	K	f	118
12	L	x	115
13	M	g	119
14	N	h	120
15	O	b	113
16	P	z	111
17	Q	y	121
18	R	x	122

Assume: A≠G

G=L

B≠Q

B=J

E=O

C=P

A=R

[00112] As the user submits data for storage, the user perceives the LBAs to be 1-18 for 18 blocks each which is the size of a buffer unit, *e.g.,* 512 Bytes. When transmitting the data for storage, the user treats each buffer unit as unique, assuming that it is being stored at a different block.

[00113] Upon application of the hash value algorithm, a few duplicative hash values are generated. As can be seen, hash value x applies to buffer units A, G, L and R; hash value y applies to buffer units B, J and Q; hash value z applies buffer units C and P; and hash value b applies to buffer units E and O. However, the true duplications of data are located between A and R, G and L, B and J, E and O and C and P. Thus, conflicts arise between A and G; G and R; B and Q; and Q and J.

[00114] As the fourth column shows, when writing the data, for true duplicates, no new blocks are written. See *e.g.,* the actual locations of where the user perceived LBA 2 and 10 are stored. Both are stored at block 110. If one considers the third and fourth columns, one sees that after the first seven user supplied buffer units have been analyzed, there is the first duplication hash value generated (the x value). However, because A≠G, G must be written to a block on the NCM. After this occurs, for purposes of further conflict analysis, within the hash value table that is stored in memory, x is associated with G and not A.

[00115] After the tenth buffer unit J has been analyzed, one sees that a duplicative hash value has been generated y. However, because the buffer units are the same, there is no conflict. Therefore, no update is needed to the hash value table, and no new block needs to be written to the NCM. Instead, the mediator will be updated to point to the true LBA 110**,** and to correlate it with the user perceived LBA. A similar true duplication is revealed after the twelfth user perceived LBA is analyzed, and the fifteenth and sixteenth are analyzed.

[00116] After buffer unit Q is analyzed, hash value y is generated. However, because Q≠B, there is a conflict. Consequently, a new block is written at LBA 121, and the hash value table is updated to associate y with Q and no longer associates y with B (which is the same as J).

[00117] After the user supplied buffer unit R is analyzed, a hash value x is generated. R=A. However, A is not the most recent association with x within the hash value table. Consequently, the protocol treats R as if it were the first time that it saw R and

must both: (1) save a new correlation in the hash value table or write over an old one, (but it will not reinstate an association stored in memory); and (2) store the data of a new block in the NCM (here block 122). Thus, the NCM, in the example, will contain some duplication.

[00118] Table 1 is for illustrative purposes and in some embodiments, only the first and fourth columns are part of the mediator and they are not part of the hash value table. The size requirement of the mediator is small. For example, in some embodiments, a mediator needs on 8 or 16 Bytes for each buffer unit that is stored on the NCM of size of 512 Bytes or 4K.

[00119] Through the various embodiments of the present invention, physical storage space of NCMs can be greatly reduced. For example, they can be reduced by at least 50%, at least 100%, at least 200%, at least 500% or at least 1000%. Thus, in some embodiments, the amount of storage space that is needed is 50 -150 times less than would be need under standard conditions. For example, only 5-10 Bytes would be needed for storage on a mediator of what corresponds to a buffer unit of *e.g.*, 512 Bytes to 4K. Thus, by reducing the need to store the same buffer units (or fragmented buffer units) multiple times, an 8GB USB stick, which is an example of an NCM of the present invention, can be used to store what corresponds to 1.53TB of data.

[00120] The methods, systems and computer program products of the present invention have been described assuming that the buffer units that are stored are the same, even if fragmented, as those that are received from a user. In these embodiments, because they are stored at different LBAs than the user perceives, and in an order that the user cannot perceive the mediator is necessary for retrieving the information, and the user cannot retrieve data without it. Thus, it provides a degree of security.

[00121] Different users who make use of the same hash value algorithm will generate the same hash values. Therefore, their hash value tables will be similar, except for the locations of the buffer unit on the NCM. Additionally, the correlation information in the mediator will be different. As an additional level of security, prior to entering the methods of the present invention, a user may desire to code or to convert the data. These actions may be referred to as preprocessing.

[00122] In some embodiments, prior to entering the protocols or systems described above, a user's data is first converted through the use of a bit marker table or frequency converter or other hash value algorithm to generate data that is smaller in size and/or encoded. Methods, systems, and computer program products for carrying out these technologies are disclosed in U.S. 13/756,921, filed February 2, 2013, entitled, Bit Markers and Frequency Converters; U.S. 13/797,003, filed March 12, 2013, entitled Data Storage and Retrieval Mediation Systems and Methods for Using Same; and U.S. 13/908,239, filed June 3, 2013, entitled Methods and Systems for Storing and Retrieving Data. The entire disclosures of the aforementioned applications are incorporated by reference in their entireties. The output of these methods may be used to generate the buffer units for reducing duplications.

[00123] These applications describe methodologies that may be incorporated into the present invention as preprocessing steps, *i.e.,* prior to the de-duplication strategies of the present embodiments. In these cases, the buffer units would correspond to the output of the preprocessing methods by which data is converted through the use of a bit marker table or frequency converter.

[00124] Thus, in one embodiment, this preprocessing step comprises: (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) using the markers as buffer units. This preprocessing step makes use of a bit marker table that may be stored in the same or in different persistent memory than the hash value table. When reading data, these preprocessing steps may be carried out in the reverse order, and after data is retrieved from the NCM and the buffer units on the NCM are recombined in the manner dictated by the mediator. When using the marker as the buffer unit, one may combine or divide markers in order to form buffer units of the necessary size.

[00125] A bit marker table may contain makers that are all the same size or of different sizes. When of different sizes, the sizes may, as described below, be determined by the predicted frequency of string of bits or bytes.

[00126] By way of further example, when the preprocessing steps makes use of a bit marker table, raw data is translated into a series of markers that represent the raw data. The raw data corresponds to the data received from the host, and thus, may for example, be one or more chunklets that individually or collectively form one or more files such as a JPEG, PDF, TIFF or WORD document.

[00127] 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 generate 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.

[00128] 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.

[00129] The bit marker table correlates unique sets 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.

[00130] 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 in order 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.

[00131] As noted above, 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.

[00132] 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.

[00133] 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. The output of markers may be in an order that allows them to be combined to form user supplied buffer units for input into a hash value algorithm as described above.

[00134] 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.

[00135] 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.

[00136] 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. Thus, 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.

[00137] 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. When using markers, either from the bit marker table or frequency converter, when they are of a different size, they must be formatted that the system can know where one marker ends and the next begins. This may, for example, be accomplished through setting a minimal marker size and a read analysis that queries whether each string of the minimal size is unique within the table or converter and if not, continuing to grow the string by reading additional bit(s) and repeating the query for each additional bit.

[00138] Information may be converted and the output code can be configured to be smaller 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 being the same size or larger than the data received from the host or as generated from a hash function value algorithm.

[00139] According to another embodiment, the preprocessing step comprises: (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 (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 as the user supplied buffer unit; 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 as the user supplied buffer unit.

[00140] When using this preprocessing hash value algorithm, one needs to address the possibility of their being conflicts. As, an alternative to the methods described above for de-duplication in which the most recent hash value association is maintained and stored, during this optional preprocessing step, one may use a conflict resolution module that, in the case of generation of a hash function value that is the same as a stored hash function value in the correlation file but for which the user supplied chunklet is different from the stored chunklet, a method causes there to be different Z bits associated with the stored hash function value and the generated hash function value. This technique is described in U.S. patent application serial number 13/908,239, filed June 3, 2013, the entire disclosure of which is incorporated by reference.

[00141] Thus, this preprocessing step may make use of a first hash value table, and 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, the method may associate 8 zeroes at the end of a checksum of 8 Bytes 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. 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. These extension files may be used as an alternative to replacing or overwriting stored hash value associations with stored buffer units. These checksums with extensions may be combined to be the necessary size to form input as the user supplied buffer units.

[00142] In any case in which the preprocessing step uses a hash value algorithm, the first hash value algorithm to be applied may be referred to as a first hash value algorithm or a preprocessing hash value algorithm that makes use of a first hash value table, and the second hash value algorithm may be referred to as a de-duplication hash value algorithm or second hash value algorithm that makes use of second hash value table. When both a preprocessing hash value algorithm and a second hash value algorithm are used, as noted above, preferably, in order to address conflicts when using de-duplication hash value algorithm the corresponding hash value table decides between conflicting associations by choosing in favor of the most recent association, whereas in order to address conflicts when using preprocessing hash value algorithm the corresponding hash value table uses the extension method as described above.

[00143] When using preprocessing techniques, the outputs, *e.g.,* the bit markers may be of the same size as the buffer units. In some embodiments, the bit markers may be larger than the size of the buffer units. In these cases, the system may fragment them to form the buffer units, or contain a default module that rejects any bit markers that are larger than the data for which they code, and instead use the original raw data to form the buffer units, thereby bypassing the access to the bit marker table. In other embodiments, they may be smaller and need to be combined to form the buffer units or to form a buffer to be fragmented into buffer units. These steps may be carried out on a server, in a cloud or by a CPU according to a module within a computer program product.

[00144] If preprocessing of data is part of the writing process, then post-processing of data must be part of the reading process. Unlike in the reading of the de-duplication steps of the present invention, the post-processing steps are symmetrical to the preprocessing steps but carried out in the reverse order.

[00145] Additionally, the various embodiments of the present invention may be used in combination with other methods for protecting data against loss. 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 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.

[00146] 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 LBAs on the first recording medium.

[00147] After time T1, for example at T2, the host may seek to update a file that it believes is stored in a given location *e.g.,* on a given sector or sector cluster. The host will not change the data stored at the first storage address(es). Rather than causing the information on the NCM to be written over, the first mediator may generate a new correlation entry that corresponds to what the host believes is an updated file. Because most if not all of the buffer units that are written on the NCM are unique entries, the new correlation on the mediator will differ from the original correlation only for the buffer units that are different from those in the original correlation. Thus, at T0 for file(A), a first mediator may correlate the following true LBAs: 200, 201, 202, 203, 204, 205, 206. At T1, a copy of the mediator may be made. At T2, the user may try to update file(A). On the first mediator, a new correlation may be saved that points to the following true LBAs: 200, 201, 310, 203, 204, 205, 206. However, the second mediator would not change. Thus, they would differ by to where they point. The earlier saved correlation may be rendered inactive on the first mediator or deleted or overwritten.

[00148] 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 only new portions of the file are stored, and a new mediator entry is made.

[00149] 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. 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.

[00150] 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.

[00151] 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.

[00152] 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 user may direct the system to save revisions to a data file that is stored in said first sector or sector cluster on the first non-cache storage medium. Only new buffer units (or buffer units that are not actively associated with a hash value) would be added to the first non-cache storage medium and there would be no writing over of data on the first non-cache storage medium. Instead a new correlation would be written on the first mediator. No changes would be made to the second mediator or second non-cache storage 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.

[00153] 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. 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.

[00154] 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.

Claims

We claim:

1. A method for storing data on a non-cache recording medium, the method comprising:
    i. receiving instructions to write data to a non-cache recording medium, wherein the instructions comprise a user perceived logical block address and a user supplied buffer;
    ii. dividing the user supplied buffer into user supplied buffer units;
    iii. applying a cryptographic hash function to each user supplied buffer unit, thereby generating a generated hash value for each user supplied buffer unit;
    iv. activating a computer program product that comprises an algorithm that causes the computer program product to access a hash value table and to determine whether each generated hash value is duplicative of a hash value within the hash value table, wherein the hash value table associates each of a plurality of stored hash values with a different stored buffer unit, and a true logical block address; and
        A. if the generated hash value is not within the hash value table, then writing the user supplied buffer unit to a block in a non-cache recording medium, updating the hash value table to include a correlation of the user supplied buffer unit, the generated hash value and a true logical block address at which the user supplied buffer unit is stored, and correlating on a mediator the true logical block address that corresponds to where the user supplied buffer has been written and the user perceived logical block address for the user supplied buffer; and
        B. if the generated hash value is duplicative of a stored hash value within the hash value table, querying whether there is a conflict, wherein a conflict is defined as the circumstance in which the same hash value is associated with a stored buffer unit and the user

supplied buffer unit, and the stored buffer unit and the stored buffer use and the user supplied buffer unit have different values, and

a. if there is a conflict, writing the user supplied buffer unit to a block in the non-cache recording medium, rendering inactive or deleting an association within the hash value table between the stored buffer unit and the stored hash value, updating the hash value table to include a correlation of the user supplied buffer unit, the generated hash value and a true logical block address at which the user supplied buffer unit is stored, and writing on the mediator the true logical block address that corresponds to where the user supplied buffer unit has been written and the user perceived logical block address, and

b. if there is no conflict, writing on the mediator the true logical block address of a buffer unit stored on the non-cache recording medium that is the same as the user generated buffer unit and correlating it with the user perceived logical block address for the user supplied buffer without writing the user supplied buffer unit on the non-cache medium.

2. The method according to claim 1 further comprising repeating (iii) – (iv) for each of a plurality of user supplied buffer units, wherein collectively said plurality of user supplied buffer units correspond to a file.

3. The method according to claim 1, wherein the user supplied buffer consists of from 512 Bytes to 2MB.

4. The method according to claim 1, wherein for a user perceived logical block address, the mediator correlates the same true logical block address a plurality of times.

5. The method according to claim 1, wherein the buffer unit is 512B or 4K in size.

6. The method according to claim 5, wherein the hash value is 8 Bytes or 16 Bytes in size.

7. The method according to claim 1, wherein the user supplied buffer units are formed by preprocessing, wherein said preprocessing comprises: (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) assigning a marker to each chunklet from a set of X markers to form a set of a plurality of markers, wherein X is equal to or less than the number of different combinations of bits within a chunklet, identical subunits are assigned the same marker; and (iii) using the markers as buffer units.

8. The method according to claim 1, wherein the user supplied buffer units are formed by preprocessing, wherein said preprocessing comprises: (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 equal to or less than the number of different combinations of bits within a subunit, identical subunits are assigned the same marker; and (iii) using the markers as buffer units.

9. The method according to claim 8, wherein at least one marker is smaller in size than a subunit.

10. The method according to claim 7, wherein step (ii) comprises accessing a bit marker table.

11. The method according to claim 8, wherein step (ii) comprises accessing a bit marker table.

12. The method according to claim 7, wherein step (ii) comprises accessing a frequency converter.

13. The method according to claim 8, wherein step (ii) comprises accessing a frequency converter.

14. A system for storing data, wherein the system comprises:

(a) persistent memory, wherein the persistent memory stores a hash value table that is configured to associate a stored buffer unit with a stored hash value and a true logical block address;

(b) a central processing unit that comprises or is operably coupled to a computer program product that is stored in a non-transitory medium, wherein the computer program product comprises executable code that when executed, automatically,

i. applies a hash value algorithm to each of one or more user supplied buffer units to generate a generated hash value; and

ii. determines whether the generated hash value is a duplicate of a stored hash value within the hash value table that is associated with a stored buffer unit, and if so, determines whether a conflict exists, wherein the conflict is defined as

a hash value being associated with two different buffer units and if a conflict exists, updating the hash value table to cause the hash value within the hash value table to be associated with the user supplied buffer unit and not the stored buffer unit;

(c) a non-cache recording medium, wherein the non-cache recording medium is configured for block level storage; and

(d) a mediator, wherein the mediator stores a correlation of a true logical block address with a user perceived logical block address.

15. The system of claim 14, wherein the computer program writes the user supplied buffer unit to the non-cache recording medium only if:

i. either the generated hash value is not a duplicate of the hash value within the hash value table; or

ii. the generated hash value is a duplicate of a stored hash value, and there is a conflict.

16. The system of claim 14 wherein for a plurality of user files, there are correlations of user perceived logical block addresses and true logical block addresses, and within a plurality of correlations for different files there are one or more of the same true logical block addresses but different user perceived logical block addresses.

17. The system of any of claims 14-16 further comprising a retrieval module, wherein the retrieval module is configured to reconstitute a data file by accessing the mediator and recombining a plurality of buffer units in an order dictated by the mediator without accessing a hash value table and wherein the order dictated by the mediator is different from the order of the buffer units on the non-cache recording medium.

18. A computer program product comprising a non-transitory computer useable medium including a computer readable program, wherein, the computer readable program when executed on a computer causes the computer to implement a method for de-duplicating and managing data blocks within a file system comprising the method of any of claims 1-13.

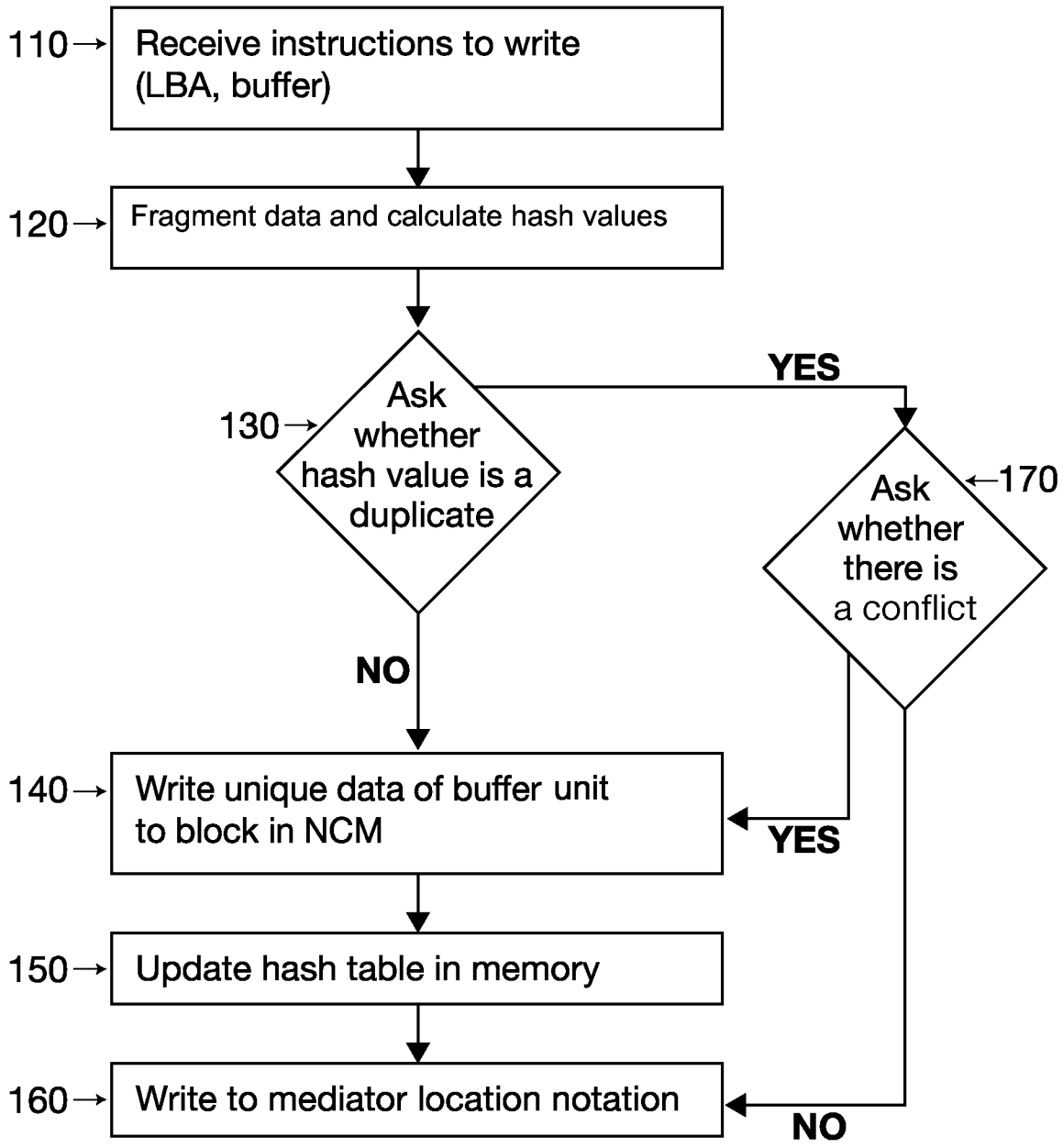

110
Receive instructions to write (LBA, buffer)
120
Fragment data and calculate hash values
130
Ask whether hash value is a duplicate
YES
170
Ask whether there is a conflict
NO
140
Write unique data of buffer unit to block in NCM
YES
150
Update hash table in memory
160
Write to mediator location notation
NO

Figure 1

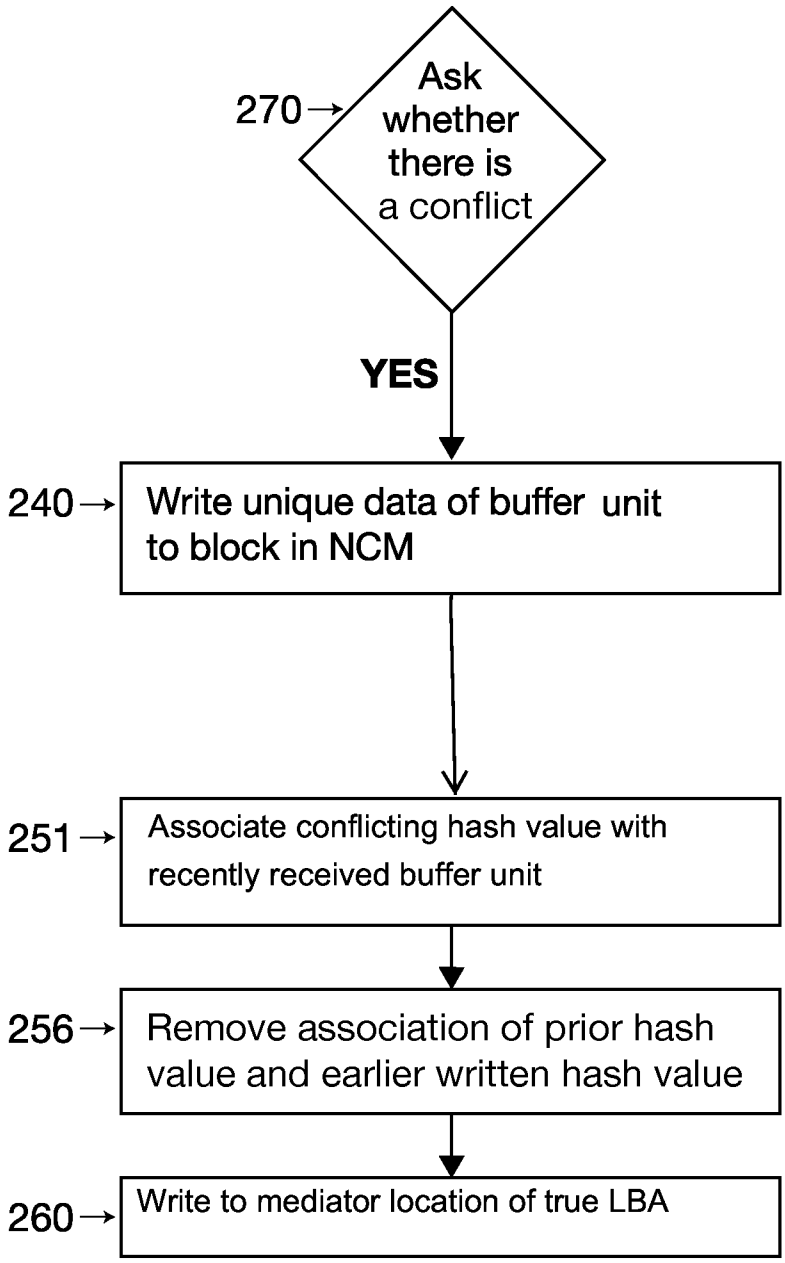

270
Ask whether there is a conflict
YES
240
Write unique data of buffer unit to block in NCM
251
Associate conflicting hash value with recently received buffer unit
256
Remove association of prior hash value and earlier written hash value
260
Write to mediator location of true LBA

Figure 2

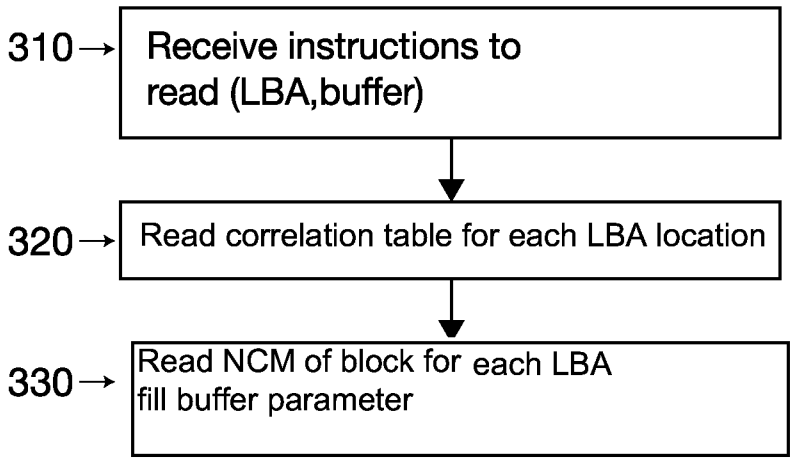

310
Receive instructions to read (LBA,buffer)
320
Read correlation table for each LBA location
330
Read NCM of block for each LBA fill buffer parameter

Figure 3

(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 2014/121109 A3

(51) International Patent Classification:
G06F 12/16 (2006.01) *G06F 12/14* (2006.01)

(21) International Application Number:
PCT/US2014/014225

(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

(71) Applicant: **SYMBOLIC IO CORPORATION** [US/US]; 379 Thornall Street, 10th Floor, Edison, NJ 08837 (US).

(72) Inventors: **IGNOMIRELLO, Brian**; 40 Manor Road, Colts Neck, NJ 07722 (US). **LIANG, S'uihong**; 64 Hillcrest Road, Martinsville, NJ 08836 (US).

(74) Agent: **LOCKE, Scott, D.**; Dorf & Nelson LLP, The International Corporate Center, 555 Theodore 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,

[Continued on next page]

(54) Title: REDUCED REDUNDANCY IN STORED DATA

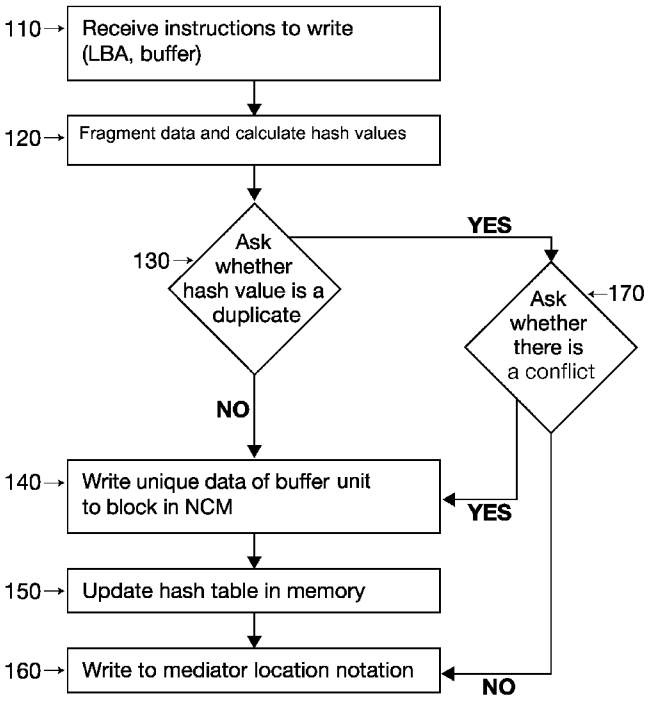


Figure 1

(57) Abstract: Through use of the technologies of the present invention, one is able to store data efficiently by the use of new and non-obvious methods, systems and computer program products that minimize the need to store duplicative data. Optionally, user data can be preprocessed in order to encode the data before entering a protocol for reducing the number of times that one stores redundant data.

WO 2014/121109 A3

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).

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:
19 March 2015

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US 14/14225

A. CLASSIFICATION OF SUBJECT MATTER
IPC(8) - G06F 12/16 (2014.01); G06F 12/14 (2014.01)
CPC - G06F 12/1408
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)
CPC: G06F12/1408; IPC(8): G06F 12/16 (2014.01); G06F12/14 (2014.01); USPC: 713/193

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
CPC: G06F12/1408, G06F21/10, G11B20/00086, G11B20/0021, G06F2221/2107, H04L9/08; USPC: 713/193; 707/791, 707/781, 707/802, 707/758, 707/706, 707/736; 711/216, 711/217, 711/141 (keyword limited; terms below)

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
PatBase; Google Scholar
Search Terms: non-cache, write, buffer, hash, crypto, encrypt, table, index, database, value, digest, logical block address, LBA, duplicate, conflict, collision, dedup, de-dup, file, record, data, 512, byte, KB, MB, chunk, mark, frequency converter, reconstitute

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 2012/0124282 A1 (FRANK et al.) 17 May 2012 (17.05.2012), entire document, especially abstract and para	1-18
Y	US 2013/0013618 A1 (HELLER et al.) 10 January 2013 (10.01.2013), entire document, especially abstract and para	1-18
Y	US 2012/0159282 A1 (ITO) 21 June 2012 (21.06.2012), entire document, especially abstract and para [0013], [0045], [0053], [0060], [0131], [0184], [0188].	10-13
A	US 2012/0239860 A1 (ATKISSON et al.) 20 September 2012 (20.09.2012), entire document.	1-18

☐ 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	Date of mailing of the international search report
17 December 2014 (17.12.2014)	16 JAN 2015
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: Lee W. Young PCT Helpdesk: 571-272-4300 PCT OSP: 571-272-7774

(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 105190573 A
(43)申请公布日 2015.12.23

(21)申请号 201480016699.9

(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/US2014/014225 2014.01.31

(87)PCT国际申请的公布数据
WO2014/121109 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 S·梁

(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 12/16(2006.01)
G06F 12/14(2006.01)

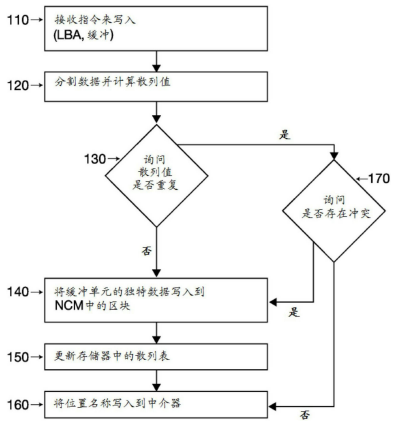
权利要求书3页 说明书22页 附图3页

(54)发明名称

存储数据的减少冗余

(57)摘要

通过使用本发明的技术，相关人员能够通过使用最小化重复数据存储需求的新颖且非显而易知的方法、系统和计算机程序产品，而有效地存储数据。可选地，用户数据可以进行预处理，从而在进入协议之前编码所述数据，以便减少相关人员存储冗余数据的次数。



CN 105190573 A

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. 接收指令来将数据写入到非高速缓存器记录介质,其中所述指令包括用户感知逻辑区块地址和用户供应缓冲 ;

ii. 将所述用户供应缓冲划分为用户供应缓冲单元 ;

iii. 将加密散列函数应用于每个用户供应缓冲单元,从而针对每个用户供应缓冲单元来生成生成散列值 ;

iv. 激活包括算法的计算机程序产品,所述算法致使所述计算机程序产品存取散列值表并确定每个生成散列值是否与所述散列值表内的散列值重复,其中所述散列值表使多个存储散列值中的每个存储散列值与不同存储缓冲单元以及真实逻辑区块地址相关联 ;以及

A. 如果所述生成散列值并不处于所述散列值表内,那么便将所述用户供应缓冲单元写入到非高速缓存器记录介质中的区块,更新所述散列值表以便包括所述用户供应缓冲单元、所述生成散列值以及所述用户供应缓冲单元所存储的真实逻辑区块地址的相关性,并在中介器上使与所述用户供应缓冲已经写入的位置相对应的所述真实逻辑区块地址和用于所述用户供应缓冲的所述用户感知逻辑区块地址相互关联 ;并且

B. 如果所述生成散列值是与所述散列值表内的存储散列值重复,那么便查询是否存在冲突,其中冲突定义为同一散列值与存储缓冲单元以及所述用户供应缓冲单元相关联且所述存储缓冲单元和所述存储缓冲用途以及所述用户供应缓冲单元具有不同值的情形,而且

a. 如果存在冲突,那么便将所述用户供应缓冲单元写入到所述非高速缓存器记录介质中的区块,渲染不活动的或删除所述散列值表内所述存储缓冲单元与所述存储散列值之间的关联,更新所述散列值表以便包括所述用户供应缓冲单元、所述生成散列值和所述用户供应缓冲单元所存储的真实逻辑区块地址的相关性,并在所述中介器上写入与所述用户供应缓冲单元已经写入的位置相对应的所述真实逻辑区块地址以及所述用户感知逻辑区块地址,以及

b. 如果不存在冲突,那么便在所述中介器上写入存储在所述非高速缓存器记录介质上的、与所述用户生成缓冲单元相同的缓冲单元的真实逻辑区块地址,并使所述真实逻辑区块地址与用于所述用户供应缓冲的所述用户感知逻辑区块地址相互关联,而并不在所述非高速缓存器介质上写入所述用户供应缓冲单元。

2.根据权利要求 1 所述的方法,其还包括针对多个用户供应缓冲单元中的每个单元来重复 (iii) 至 (iv),其中所述多个用户供应缓冲单元共同对应于文件。

3.根据权利要求 1 所述的方法,其中所述用户供应缓冲由 512 个字节至 2MB 字节组成。

4.根据权利要求 1 所述的方法,其中对于用户感知逻辑区块地址而言,所述中介器多次关联同一个真实逻辑区块地址。

5.根据权利要求 1 所述的方法,其中所述缓冲单元大小为 512B 或 4K。

6.根据权利要求 5 所述的方法,其中所述散列值大小为 8 个字节或 16 个字节。

7.根据权利要求 1 所述的方法,其中所述用户供应缓冲单元是通过预处理来形成,其中所述预处理包括 :(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 的整数且其中所述小盘具有某个顺序 ;(ii) 从一组 X 个标记符中指配标记符给每个小盘,以便形成一组多个标记符,其中 X 等于或小于小盘内不同比特组合的数目,同样的子单元指配有相同标记符 ;以及 (iii) 使

用所述标记符作为缓冲单元。

8.根据权利要求 1 所述的方法,其中所述用户供应缓冲单元是通过预处理来形成,其中所述预处理包括 :(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 等于或小于子单元内不同比特组合的数目,同样的子单元指配有相同标记符 ;以及 (iii) 使用所述标记符作为缓冲单元。

9.根据权利要求 8 所述的方法,其中至少一个标记符在大小上小于子单元。

10.根据权利要求 7 所述的方法,其中步骤 (ii) 包括存取比特标记符表。

11.根据权利要求 8 所述的方法,其中步骤 (ii) 包括存取比特标记符表。

12.根据权利要求 7 所述的方法,其中步骤 (ii) 包括存取频次转换器。

13.根据权利要求 8 所述的方法,其中步骤 (ii) 包括存取频次转换器。

14.一种用于存储数据的系统,其中所述系统包括 :

(a) 永久性存储器,其中所述永久性存储器存储散列值表,所述散列值表被配置来使存储缓冲单元与存储散列值以及真实逻辑区块地址相关联 ;

(b) 中央处理单元,所述中央处理单元包括或可操作性地耦接至存储在非暂时性介质中的计算机程序产品,其中所述计算机程序产品包括可执行代码,所述可执行代码在运行时自动进行下述操作,

i. 将散列值算法应用于一个或多个用户供应缓冲单元中的每个单元,以便生成生成散列值 ;和

ii. 确定所述生成散列值是否为所述散列值表内的、与存储缓冲单元相关联的存储散列值的重复,并且如果是,便确定是否存在冲突,其中所述冲突定义为散列值是与两个不同缓冲单元相关联,而且如果存在冲突,便更新所述散列值表,以便致使所述散列值表内的所述散列值与所述用户供应缓冲单元相关联,而不与所述存储缓冲单元相关联 ;

(c) 非高速缓存器记录介质,其中所述非高速缓存器记录介质被配置用于区块级存储 ;以及

(d) 中介器,其中所述中介器存储真实逻辑区块地址与用户感知逻辑区块地址的相关性。

15.根据权利要求 14 所述的系统,其中只有出现如下情况,所述计算机程序才会将所述用户供应缓冲单元写入到所述非高速缓存器记录介质 :

i. 所述生成散列值不是所述散列值表内的所述散列值的重复 ;或

ii. 所述生成散列值是存储散列值的重复,并且存在冲突。

16.根据权利要求 14 所述的系统,其中对于多个用户文件而言,存在用户感知逻辑区块地址与真实逻辑区块地址的相关性,并且在针对不同文件的多个相关性内,存在相同的真实逻辑区块地址中的一个或多个,但是存在不同的用户感知逻辑区块地址。

17.根据权利要求 14 至 16 中任一项所述的系统,其还包括检索模块,其中所述检索模块被配置成通过访问所述中介器并按照所述中介器所传达的顺序重新组合多个缓冲单元来重建数据文件,而不存取散列值表,并且其中所述中介器所传达的所述顺序不同于所述非高速缓存器记录介质上的所述缓冲单元的顺序。

18. 一种计算机程序产品，所述计算机程序产品包括非暂时性计算机可用介质，所述非暂时性计算机可用介质包括计算机可读程序，其中所述计算机可读程序在计算机上运行时，会致使所述计算机来实施用于消重和管理文件系统内数据区块的方法，所述方法包括权利要求 1 至 13 中任一项所述的方法。

存储数据的减少冗余

发明领域

[0001] 本发明涉及数据的存储。

[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。这些磁盘含有尺寸为亚微米并且能够存储数条单独二进制数据的较小区域。

[0004] 在任何给定实体所存储的巨量数据内，经常存在信息的明显重复。举例而言，相同的公司信头可以出现在数千个文档中，并且对应于这个数据的每个文件将含有针对信头进行编码的比特。历史上，许多实体已经接受这类重复存在于它们的文件中，以及相同信息的冗余存储的无效是开展业务的成本。

[0005] 因为存储成本不断增加以及存储的可用性不断降低，所以众多实体已经开始探究某些方式来供存储比文件内或文件之间所有重复信息更少的信息。理论上，力图避免重复信息的存储或最小化重复信息存储的次数的实体，可以力图识别其数据集内的独特比特或字节样式，并且以最小的次数来存储这些独特比特或字节样式。为了执行这些方法，在准备新的文件用于存储时，这些文件内的信息将会与已存储的参考信息集进行比较，并且，只有正在考虑的比特或字节样式是独特的，其才会被存储。如果所述比特或字节样式不是独特的，那么便会用参考数据来取代冗余数据，所述参考数据在大小上小于指向所存储数据的数据，所述数据是存储数据的重复。

[0006] 减少重复信息存储的次数的目标面临很多挑战，包括但不限于：(1) 保持检查冗余的足够速度；(2) 保持数据重建而便于检索的足够速度；(3) 确保在检查冗余或存储对应于最初文件的信息的过程中数据不会丢失；(4) 针对存储信息未经授权的存取进行保护；以及 (5) 提供可以与获取数据快照、克隆数据和恢复数据中的一个或多个操作（即便不是所有这些操作）结合使用的有效技术和方法。本发明的各种实施方案的意图在于克服这些挑战中的一个或多个挑战。

[0007] 发明概述

[0008] 本发明提供方法、系统和计算机程序产品，用以改善数据存储和检索的效率，而同时最小化冗余数据多次不必要存储的程度。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数据。借助本发明的这些各种实施方案，相关人员可以变换数据和/或更改变换后或转换后数据所存储的物理装置。这可以借助使用计算机的自动化过程来完成，所述计算机包括或可操作性地耦接至计算机程序产品，在运行时，所述计算机程序产品执行本发明方法或过程中的一个或多个方法或过程。这些方法或过程可以（例如）体现在计算机算法或脚本中或者包括所述计算机算法或脚本，并且可选地由系统借助一个或多个模块来执行。

[0009] 根据第一实施方案，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) 接收指令来将数据写入到非高速缓存器记录介质，其中所述指令包括用户感知逻辑区块地址（“LBA”）和用户供应缓冲 (user supplied buffer)，其中

所述用户供应缓冲由（例如）512 个字节至 2MB 字节或 512 个字节至 64K 字节组成；(ii) 将所述用户供应缓冲划分为用户供应缓冲单元，并将加密散列函数应用到所述用户供应缓冲单元中的每个单元，从而生成一个生成散列值；(iii) 激活包括算法的计算机程序产品，所述算法致使所述计算机程序产品存取散列值表并确定所述生成散列值是否与所述散列值表内的存储散列值重复，其中所述散列值表使多个存储散列值中的每个存储散列值与不同存储缓冲单元以及真实逻辑区块地址相互关联；以及 (A) 如果所述生成散列值并不处于所述散列值表内，那么便将所述用户供应缓冲单元写入到非高速缓存器记录介质中的区块，更新所述散列值表以便包括所述用户供应缓冲单元、所述生成散列值以及所述用户供应缓冲单元所存储的真实逻辑区块地址的相关性，并在中介器上写入与所述用户供应缓冲单元已经写入的位置相对应的所述真实逻辑区块地址以及用于所述用户供应缓冲的所述用户感知逻辑区块地址（或多个用户感知逻辑区块地址），并且，(B) 如果所述生成散列值是与所述散列值表内的存储散列值重复，那么便查询是否存在冲突，其中冲突定义为同一散列值与存储缓冲单元以及当前用户供应缓冲单元相关联且所述两个缓冲单元具有不同内容的情形，而且，(a) 如果存在冲突，那么便将所述用户供应缓冲单元写入到所述非高速缓存器记录介质中的区块，渲染不活动的或删除所述散列值表内所述存储缓冲单元与所述存储散列值之间的关联，更新所述散列值表以便包括所述用户供应缓冲单元、所述生成散列值和所述用户供应缓冲单元所存储的真实逻辑区块地址的相关性，并在所述中介器上写入与所述用户供应缓冲单元已经写入的位置相对应的所述真实逻辑区块地址以及所述用户感知逻辑区块地址；以及 (b) 如果不存在冲突，那么便在所述中介器上写入存储在所述非高速缓存器记录介质上的、与所述用户生成缓冲单元相同的缓冲单元的真实逻辑区块地址，并使所述真实逻辑区块地址与用于所述用户供应缓冲的所述用户感知逻辑区块地址相互关联，而并不在所述非高速缓存器记录介质上写入所述用户供应缓冲单元。

[0010] 在步骤 (A) 中，当写入所述用户供应缓冲单元时，本领域普通技术人员将会了解，所述方法要求写入已经确定为并未与所述表中的散列值相关联的用户供应缓冲单元。

[0011] 通常，用户会供应处于流中或者处在比所述散列值算法被配置成接受为其输入的用户供应缓冲单元更大的单元中的数据（所述用户供应缓冲）。在这些情况下，所述用户供应缓冲单元可以通过将主机所发送的原始数据分割（也称为分裂）成更小单元来形成，这些更小单元可以视为所述用户供应缓冲单元，而无论所述主机是否将它们分割或者本发明的系统或方法是否进行这样的分割。因此，这些分割的用户供应缓冲单元可以充当所述加密散列函数的输入。举一个非限制性示例，所述用户供应数据可以是 16K 至 2MB，而每个分割的用户供应缓冲单元为 512 个字节至 4K，例如，512 个字节或 4K。因此，在一些实施方案中，所述分割的用户供应缓冲单元不大于分割之前的所述用户供应缓冲的大小的 1/4 或不大于 1/16 或不大于 1/64。如果在进入所述散列值算法之前使用所述分割步骤，那么所述散列值表便会含有所述用户供应缓冲单元和散列值的相关性，向所述存储装置的写入便是分割的用户供应缓冲单元而不是更大的数据缓冲单元，并且所述中介器会使所述用户供应缓冲单元的所述用户感知地址（或多个用户感知地址）与所述多个分割的用户供应缓冲单元相互关联。

[0012] 本发明方法的各种步骤可以存储在一个或多个模块中，例如，缓冲和用户感知逻辑区块地址接收模块、分割模块、散列值搜索模块、散列值重复分析模块、冲突模块以及写

入模块。类似地，可以存在文件读取和重建模块。这些模块可以按照可执行代码的形式存储在非暂时性介质中。

[0013] 根据第二实施方案，本发明提供一种用于存储数据的系统，其中，所述系统包括：(a) 永久性存储器，其中所述永久性存储器存储散列值表，所述散列值表被配置来使存储缓冲单元与存储散列值以及真实逻辑区块地址相关联；(b) 中央处理单元，所述中央处理单元包括或可操作性地耦接至存储在非暂时性介质中的计算机程序产品，其中所述计算机程序产品包括可执行代码，所述可执行代码在运行时自动进行下述操作：(i) 将散列值算法应用于一个或多个用户供应缓冲单元中的每个单元，以便生成一个生成散列值；和 (ii) 确定所述生成散列值是否为所述散列值表内的、与存储缓冲单元相关联的存储散列值的重复，并且如果是，便确定是否存在冲突，其中冲突定义为散列值是与两个不同缓冲单元相关联，而且如果存在冲突，便更新所述散列值表以便致使所述表内的所述散列值与所述用户供应缓冲单元相关联，而不与所述存储缓冲单元相关联；(c) 非高速缓存器记录介质，其中所述非高速缓存器记录介质被配置用于区块级存储；以及 (d) 中介器，其中所述中介器存储真实逻辑区块地址与用户感知逻辑区块地址的相关性。

[0014] 根据第三实施方案，本发明提供一种计算机程序产品，所述计算机程序产品包括非暂时性计算机可用介质，所述非暂时性计算机可用介质包括计算机可读程序，其中所述计算机可读程序在计算机上运行时，会致使所述计算机来实施用于消重和管理文件系统内数据区块的方法，所述方法包括本发明方法中的任何方法。这些方法可以组织在一个或多个模块中。

[0015] 借助本发明的各种实施方案，相关人员可以提高数据存储和检索的效率，因为在大多数情形中，与之前写入的数据重复的较大缓冲单元将不需要重新写入到非高速缓存器记录介质 (NCM)。相反，中介器上的、就大小而言小于与容纳所指向数据的存储单元物理分离的结构且处于所述结构内的指针，会将所述计算机定向到这个数据的之前存储复本。可以通过比普遍应用的方法中使用的使用更小的存储空间以及在存储和 / 或检索信息的活动中投入更少的时间和努力，而实现效率的提高。此外，在一些实施方案中，本发明使得提高文档存储和检索的速度。因此，本发明的技术和方法论有助于减少存储数据所需要的物理存储总量。这通过最小化重复数据写入和存储的次数而完成，并且在存在重复数据的情况下，通过使用中介器来指向到之前存储数据而完成。

[0016] 附图简述

[0017] 图 1 为根据本发明实施方案的用于写入数据的方法的表示图。

[0018] 图 2 为根据本发明方法的用于解决冲突的协议的表示图。

[0019] 图 3 为根据本发明实施方案的用于读取信息的方法的表示图。

[0020] 发明详述

[0021] 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。

[0022] 定义

[0023] 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。

[0024] 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。每个数值可

以由 0 或 1 表示。

[0025] 术语“区块”是指具有预定长度的数据字节或比特序列。在记录介质上，物理介质可以划分为由区块大小所界定的单元。记录介质上的每个区块可以通过逻辑区块地址来识别。在行业中，目前 512 个字节是区块的标准大小。然而，存在使用 4096 个字节作为标准的渐进趋势。另外，正如本领域普通技术人员将会了解的，短语“区块大小”和“扇区大小”常常被本领域普通技术人员互换使用。

[0026] 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。

[0027] 短语“缓冲单元”是指具有兼容用作散列值算法输入的大小的一系列比特。缓冲单元可以与小盘具有相同大小。然而，在一些实施方案中，缓冲单元可以是小盘大小的几分之一或者是小盘大小的倍数。

[0028] 术语“字节”是指八比特序列。

[0029] 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。在一些实施方案中，当实施本发明的方法时，散列值表存在于 L2 高速缓存器中。

[0030] 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，小盘大小已经从逻辑区块地址 (LBA) 测量中得出。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。

[0031] 术语“冲突”是指针对不同输入（例如，缓冲单元）出现了生成相同输出的情况，例如，通过函数（如散列值算法）生成相同的散列值。

[0032] “文件”是相关字节或比特的集合，这些相关字节或比特组合来提供具有某个大小的文件，而且长度可以用比特或字节来度量。文件可以小于小盘、与小盘具有相同大小或者大于小盘。

[0033] 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。

[0034] 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)。

[0035] 短语“散列函数”、“加密散列函数”、“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（具有相同或可变长度）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。因此，在一些实施方案中，散列值的长度为 32 至 512 个比特。

[0036] 术语“主机”、“用户”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相对应的数据并且接收数据。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。

[0037] 术语“包括 (including)”和“包括 (comprising)”以开放的方式使用，且因此应解释为表示“包括但不限于”的意思。

[0038] 缩写“LBA”是指“逻辑区块寻址”或“逻辑区块地址”。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。用户可以认为数据是存储在特定 LBA 上。用户感知到的数据存储位置是“用户感知逻辑区块地址”。这可以不同于数据实际存储的位置。数据在 NCM 上实际存储的位置可以称为“真实逻辑区块地址”。

[0039] 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。

[0040] 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、传输或处理数据）的计算机程序产品，例如，代码。它可以存储在硬件、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。

[0041] 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。因此，中介器可以使用户感知 LBA 与真实 LBA 相互关联。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、转译、转换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，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，中介器存在于 L2 高速缓存器上。

[0042] 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。

[0043] 缩写“NCM”是指非高速缓存器记录介质。NCM 的示例包括但不限于硬盘和固态驱动器。NCM 可以（例如）被配置来存有 100TB 字节的数据。NCM 存储独特的缓冲单元。在一些实施方案中，NCM 也存储摘要图，所述摘要图含有缓冲单元和存储散列值的关联形式。这些存储的关联形式可以用来填写服务器的 RAM 中的散列值表。通过向 RAM 中填写来自 NCM 永久性储存器的这个信息，可以完成快速加载。另外或替代地，NCM 存储每个单元区块 1 比特的比特图，所述比特图可以用来跟踪本发明的存储保存内容。作为惯例，摘要图和比特图的存储需要介于 5 个字节与 10 个字节之间的较小开销，例如，在 NCM 上每个区块的存储数据需要 8.125 个字节。或者，摘要图和比特图可以与作为本发明方法或系统用途的结果而写入的缓冲单元，存储在不同的存储介质上。

[0044] 短语“可操作性地耦接”与术语“耦接”可互换使用，而且意味着系统、装置和 / 或模块被配置成彼此或互相通信，并能够在通信时或通信之后执行它们的预期目的。这个短语和术语包括间接、直接、光学、有线或无线连接。因此，如果第一装置可操作性地耦接至第二装置，那么这个连接可以借助直接电气连接、经由其它装置和连接而借助间接电气连接、借助光学连接或借助无线连接，或者借助这些连接的组合。

[0045] 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。

[0046] 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。

[0047] 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 NCM，如硬驱、硬盘、软盘、计算机磁带、ROM、EEPROM、非易失性 RAM、CD-ROM 以及穿孔卡。

[0048] 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。

[0049] 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 iSCSI（互联网小型计算机系统接口）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。

[0050] 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。

[0051] 优选实施方案

[0052] 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，相关人员可以通过降低重复数据存储的次数来有效地存储和检索数据。

[0053] 根据一个实施方案，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。在一些实施方案中，所接收的数据包括用户定义逻辑区块地址（LBA）和用户供应缓冲、基本上由用户定义逻辑区块地址（LBA）和用户供应缓冲组成，或者由用户定义逻辑区块地址（LBA）和用户供应缓冲组成。用户供应缓冲的大小可以是（例如）512 个字节至 64K，或者甚至更高，例如，高达 2MB。所接收的数据可以呈（LBA_x，缓冲 $_x$）格式，其中 x ＝ 1 至 n 且 n ＝用户所发送的缓冲的数目。缓冲是原始数据，且因此它们可以对应于任何文档类型，例如，JPEG、PDF、WORD 文档、MPEG 以及 TXT 文档。

[0054] 用户所发送的整个流可以呈 N 个字节的形式。所述流可以在有线或无线网络上并且借助已知的用于传输 I/O 流的方法和技术来接收。用户可以将数据格式化成（LBA_x，缓冲 $_x$）格式，或者用户可以将数据发送给服务器，所述服务器将数据格式化成这个格式。因此，N 可以大于缓冲单元的大小。优选地，用户传输具有文件名和 / 或文件识别符的数据。

[0055] 当 N 大于缓冲单元的大小时，在接收数据流之后，服务器可以将所述 N 个字节分割成缓冲单元。因此，缓冲是可以具有任何大小的数据流，但是缓冲单元具有固定大小并且对应于 NCM 上存储单元的大小。举例而言，用户可以发送具有文件名称的单个 1024 字节缓冲，以便开始于 LBA10。如果本发明的实施方式的缓冲单元大小为 512 个字节，那么数据可以存储在 LBA20 和 LBA21 处。值得注意地，在一些实施方案中，用户可以仅仅传输起始 LBA，并且其系统将感知到有待于存储在 NCM 上的文件是开始于所述地址处且延展到连贯区块，从而使得存在足够的空间用于所述文件。

[0056] 这个分割可以针对整个流或小盘执行。举例而言，如果所述方法被配置来接收大小为 4K 的缓冲单元，但是用户传输大小为 16K 的小盘，那么在服务器接收小盘之后或在服务器接收小盘时，服务器会发起协议来将每个小盘划分为四个大小各自为 4K 的缓冲单元。因此，根据一些方法，相关人员以（LBA_x，缓冲 $_x$）的形式来接收数据，或者将其所接收的数据转换成这个形式。无论用户是否以被配置成具有必要的缓冲单元大小或被转换成这个大小的封包来供应数据，充当散列值算法的输入的缓冲单元中的每个单元都称为“用户供应缓冲单元”。

[0057] 在数据以正确形式接收（或转换成正确形式）之后，将加密散列函数值算法应用于每个缓冲单元，以便针对所述缓冲单元来形成一个生成散列值。所生成的散列值可以称为生成散列值。加密散列值算法可以（例如）呈计算机程序产品的形式或者呈存储在非暂时性存储介质中的计算机程序产品内的协议的形式。这些类型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散列值、校验和或者总和。在一些实例中，散列值大小为 64、128 或 256 个比特或 8 个字节，或者是介于其间的任何值。由于 I/O 流内数据高度重复的特性，所以生成冲突散列值（也就是，相同的但对应于不同缓冲单元的散列值）的可能性便相对较低。所述方法可以根据先进先出（“FIFO”）协议来获取散列值，并且在正接收 I/O 流时、在正生成散列值时或者在所有 I/O 流已经接收、分割（必要时）以及经受散列函数值算法之后，开始存取相关性文件。

[0058] 在针对用户供应缓冲单元获取生成散列值之后，访问不同的计算机程序产品或者同一计算机程序产品内产生所述生成散列值的不同模块。这个计算机程序产品存取散列值表。散列值表可以（例如）存储在永久性存储器中，并调用到 L2 高速缓存器中而便于存取

和使用。在散列值表内，多个存储散列值各自与 NCM 上一组存储缓冲单元内的不同存储缓冲单元以及所述存储缓冲单元的真实 LBA 相关联。短语“存储散列值”用来与通过散列值算法而针对特定用户供应缓冲单元来生成的散列值（其可以称为生成散列值）形成对照。

[0059] 散列值表最初可以填写有通过特定散列值算法而关联的一组已知散列值和缓冲单元。这些已知的值可能已经基于算法的经验先前使用加以确定，例如，之前针对类似行业中主机文件或主机的文件所生成的那些值。或者，散列值表最初可以是空的，并且将第一用户供应缓冲单元与一组无效关联性进行对比。在使用时，散列值表可以存在于（例如）服务器上的 RAM 中。因此，散列值表或者将要填入其中的数据可以存在于永久性储存器中，如存在于存储缓冲单元的 NCM 上，或者按照可以在系统启动或重启时进行存取而用于 RAM 的重新填写的格式存在于其它地方。在更新时，会对 RAM 中的表并且也会对永久性存储器做出更新。

[0060] 在一些实施方案中，任何时候，在所述表内，给定存储散列值与不超过一个缓冲单元主动进行关联。当缓冲单元处于散列值表内且与存储散列值相关联时，缓冲单元为“存储缓冲单元”。与散列值表内的特定散列值相关联的缓冲单元可以随着时间发生改变，并且，在发生下文所述的冲突的情况下，散列值算法所确定的最近缓冲单元、散列值关联将在应用本文中所述的方法之后，成为表内的主动关联。短语“主动关联性”和“主动进行关联”是指所述表赖以提供、配置或表示有待提取的数据而与生成散列值进行比较的条件。主动关联性可以包括在 NCM 上最近接收缓冲单元的真实逻辑区块地址，所述缓冲单元使得在应用散列值算法后生成散列值。一旦散列值变成存储散列值，那么其将保留在所述表中；然而，随着时间的推移，这个散列值所关联的缓冲单元可以发生改变。因此，存储缓冲单元可以不再是存储缓冲单元。

[0061] 本领域普通技术人员将会认识到，在某些实施方案中，相关人员可以不使用散列值表，而是使用多重映射。当使用多重映射时，散列值可以与一个以上缓冲单元关联，且因此之前存储的散列值无须在存在冲突时进行移除。

[0062] 因为散列值算法可以针对多个不同缓冲单元而生成相同散列值，所以需要某个操作程序来确定如何处置表内的这些发生情况。当已经针对用户供应缓冲单元生成与存储散列值相同的散列值时，可以使用下述三个操作程序中的一个：(1) 可以用新的关联性来改写预先存在的关联性；(2) 可以删除预先存在的关联性，并且可以在表内（例如）新的位置处制作新的条目；或者 (3) 可以将预先存在的关联性移动到不活动文件，或者以其它方式进行修改以便表示其是不活动的，但是信息保持在（例如）归档文件中，并且借助适当设计的计算机程序，信息可以在稍后时间上进行存取。条件 (1) 或 (3) 是将关联性渲染为不活动的示例。正如本领域普通技术人员将会认识到的，因为从 NCM 中进行的数据检索并不需要存取散列值表，所以不需要任何归档信息来实施本发明的各种实施方案。然而，这个信息可以用来核查针对不同用户供应缓冲单元或分割用户供应缓冲单元来生成相同散列值的程度。

[0063] 由于数据的高度重复特性，因此作为应用散列值算法的结果而生成冲突散列值的机率较低。举例而言，SHA-1 的 160 比特散列算法针对不同样式随机生成相同散列值的机率为 10^{24}之一。本发明利用散列值算法的这个特征，从而最小化将要存储重复缓冲单元的范围。为了完成这个操作，所述方法使得分配两个模块或两个单独计算机程序算法，所述模

块或算法查询：(1) 新生成的散列值与存在存储缓冲单元的存储散列值相同？并且如果相同，(2) 存在冲突，从而使得针对所述散列值的存储缓冲单元不同于用户供应缓冲单元？

[0064] 上文所论述的查询的结果确定什么样的新信息将会存储在 NCM 上并且是在什么情形下，以及什么内容会写入到中介器。在最简单的情况下，本发明的协议确定不存在散列值的重复。不存在散列值的重复则指示用户供应缓冲单元不处于散列值表内。因此，缓冲单元写入在 NCM 的新区块中，并且对散列值表做出更新，从而使得如果用户随后提交相同的缓冲单元来作为不同数据流或数据包的一部分，或者随后在相同数据流内提交相同的缓冲单元，所述方法论将能够检测到这个缓冲单元是与 NCM 上已经存储的数据相同。散列值表也会进行更新，以便包括缓冲单元所写入的真实 LBA。在将生成散列值和用户供应缓冲单元写入到散列值表后，它们便分别变成存储散列值和存储缓冲单元。

[0065] 如果所述方法论确定存在散列值的重复，那么便进行第二查询。正如上文所提到的，在这个第二查询中，所述方法考虑将所述算法应用于用户供应缓冲单元是否会致使产生已存储缓冲单元所关联的相同散列值，即使这两个缓冲单元是不同的。

[0066] 正如本领域普通技术人员将会轻易认识到的，这个两级方式将效率带入到方法中，从而减少在 NCM 上存储重复数据的次数。在第一步骤中，比较散列值。这些值小于缓冲单元，例如，比缓冲单元小至少 2 倍、至少 10 倍、至少 100 倍或至少 1000 倍，且因此比缓冲单元本身更容易进行比较。只有在这些值指示散列值重复的情况下，系统才会比较实际缓冲单元。因此，通过在将与已经不处于散列值表内的散列值关联的缓冲单元进行彼此检查之前除去这些缓冲单元，系统是有效的。

[0067] 在查询的第二级中，相关人员会将两个缓冲单元彼此进行比较。如果不存在冲突，也就是存在缓冲单元（存储缓冲单元和当前用户供应缓冲单元）的真实身份，那么便不需要写入到 NCM 的额外步骤。相反，只需要针对缓冲单元之前所存储的区块的中介器进行标记。因此，效率便得以提高，因为这些缓冲单元将不需要再次写入。为了跟踪区块所从属的文件，在中介器内，LBA 与用户感知 LBA 进行关联，并且可选地与用户生成文件名或文件系统相关联。

[0068] 当（与存储缓冲单元相关联的）存储散列值与生成散列值相同，但是因为存储缓冲单元和用户供应缓冲单元不同而存在冲突时，便出现第二级查询的另一结果。在这些情况下，本发明的方法将用户供应缓冲单元视为需要写入到NCM。这些方法也致使改变散列值表，从而使得其将共同散列值与用户供应缓冲单元相关联。共同散列值的之前关联性被渲染为不活动的或被删除。因此，在搜索重复缓冲单元的后续查询中，只考虑最近存储的散列值、缓冲单元关联性。在这些方法中，从不需要在发生冲突的情况下扩展散列值。

[0069] 在各种实施方案中，对于每个 LBA_x 而言，所述方法致使中介器将 LBA_y 与 LBA_x 一起存储，其中 LBA_y 是指与用户供应缓冲单元相同的缓冲单元的实际位置。因为 LBA_x 是用户认为缓冲单元所存储的位置而 LBA_y 是缓冲单元实际存储的位置，所以 LBA_y 也存储在散列值表内。然而，LBA_x 可以从散列值表中省略，并且在一些实施方案中只存在于中介器上。可选地，中介器也存储用户创建的文件名和 / 或文件识别符，并且使这个文件名与一个或多个用户感知逻辑区块地址相关联。

[0070] 正如上文所提到的，大多数缓冲单元只存储在 NCM 上一次。因此，对于多个用户文件而言，存在用户感知逻辑区块地址与真实逻辑区块地址的相关性，并且在（中介器上）对

应于不同用户文件的多个相关性内，存在一个或多个相同的真实逻辑区块地址，但是存在不同的用户感知逻辑区块地址。实际用户文件中最高度重复的缓冲单元将出现在最大数目的不同相关性中。另外，因为用户可以供应缓冲并且认为所述缓冲正存储在 NCM 上的连续位点处，所以可以记录单个 LBAx 与所述缓冲的关联性，并将数据视为存储在连续 LBA 处，开始于 LBAx。然而，因为存储实际是在缓冲单元级别上，并且常常对于给定缓冲而言不会存储在连续位置上，所以中介器可以存储单个用户感知 LBA，（或者暗中地或明确地）将多个连续 LBA 与多个不连续的真实 LBA 一起存储。举例而言，用户可以供应大小为 4096B 的缓冲和用户感知 LBA10。如果缓冲单元大小为 512B，那么用户可以暗中地认为其数据是处于八个连续存储位点上，开始于 LBA10。然而，事实上，它们可以处于 LBA4、LBA3、LBA2、LBA2、LBA3、LBA3、LBA9、LBA4 处，并且中介器会指向这些位置。值得注意地，对于与用户所发送的缓冲内的数据相对应的大多数重复缓冲单元（即便不是全部）而言，中介器会指向 NCM 上的同一 LBA。因此，在中介器上，在一些实施方案中可以存在所有真实数据位置（例如，所有真实 LBA）与从用户接收的用户感知位置或多个用户感知位置的相关性。

[0071] 因此，在这些方法中，不是使用扩展来虑及多个散列值算法生成相同散列值的机率，而是关联性的最近发生情况的上述使用，仅仅是在下述情形下才会将特定供应缓冲单元写入到 NCM：(i) 可比散列值已经不处于与任何缓冲单元关联的散列值表内；或者 (ii) 对于任何给定散列值而言，存在冲突。在这些后者情况下，可以将之前已经写入的相同数据部分地写入到 NCM。然而，作为惯例，这个很少发生。

[0072] 在一些实施方案中，将缓冲单元写入到 NCM 是连续的，也就是，每个将要写入的用户供应缓冲单元可以写入在 NCM 上的下一个连续区块中。因此，在 NCM 上存在最小的数据分散或不存在数据分散，这会改善读取 / 写入性能并允许节约存储空间。另外，读取 / 写入性能得以改善，是因为实际上向 NCM 写入更少数据。这又会使得操作系统的高速缓存器能够更好地发挥作用。此外，可以提高数据的安全性，因为 NCM 上的数据在不没有中介器和散列值表的情况下无法用来重建文件。

[0073] 如果相关人员考虑到数据是如何读取的，便会了解本发明的其它益处。用户可以发送请求来读取文件。所述请求包括文件识别符以及与一个或多个用户感知逻辑区块地址相关的信息。通过访问相关中介器，相关人员可以确定实际逻辑区块地址（或多个实际逻辑区块地址）并检索相关数据。值得注意地，与写入协议相比而言，在检索和读取步骤中不需要散列值算法或表。相反，读者从中介器所指向的、NCM 上的位点检索数据，并且中介器可以针对一个或多个文件而多次指向一个或多个缓冲单元。

[0074] 本发明也提供用于数据有效存储的系统。这些系统可以包括：(a) 永久性存储器；(b) 中央处理单元（“CPU”）；(c) 非高速缓存器记录介质；以及 (d) 中介器。这些部件中的每个部件可操作性地耦接至一个或多个其它部件，从而执行它们的指定功能。

[0075] 永久性存储器包括散列值表，并且可以是下文所述的非高速缓存器记录介质的一部分，或者截然不同于所述非高速缓存器记录介质。散列值表使多个存储散列值中的每个散列值与不同存储缓冲单元相关联。通过将散列值算法应用于缓冲单元来确定散列值。每个存储缓冲单元也与真实逻辑区块地址相关联。本领域普通技术人员将会认识到，散列值表可以在一个表中含有下述三种类型数据之间的关联性：存储散列值、存储缓冲单元和真实逻辑区块地址。或者，散列值表可以在一个表中仅仅含有前两种类型数据的关联性，并

且，在同一或不同存储器装置（例如，中介器）中的另一个表中，可以存储使真实 LBA 和存储缓冲单元相互关联的表。永久性存储器可以存储在 CPU 内或者可操作性地耦接至 CPU。

[0076] 中央处理单元包含硬件或硬件与软件的组合。CPU 被配置来访问永久性存储器并搜索散列值表。CPU 也被配置来执行本发明方法中的一个或多个方法，包括但不限于向 NCM 和中介器进行写入。此外，CPU 被配置成借助无线或有线网络（例如，借助服务器）来与一个或多个远程用户通信。

[0077] NCM 被配置用于区块级存储。NCM 可以与 CPU 相分离或者作为 CPU 的一部分。

[0078] 在上述系统的一些实施方案中，中介器、散列值表、CPU 和非高速缓存器记录介质中的每一者彼此都是远程地存储。它们可以处于同一外壳内或不同外壳中的单独结构中。

[0079] 正如上文所提到的，本发明的系统（其可以借助服务器进行控制）可以在咨询中介器之后且在不存取散列值表的情况下，重新创建文件并向用户传输数据。因此，在一个实施方案中，本发明的系统包括检索模块，其中所述检索模块被配置成通过访问所述中介器并按照所述中介器所传达的顺序重新组合多个缓冲单元来重建数据文件，而不存取散列值表，并且其中所述中介器所传达的所述顺序不同于非高速缓存器记录介质上的缓冲单元的顺序。

[0080] 本发明的各种方法可以由管理器自动控制。管理器可以包括一个或多个模块，并且存在于本地计算机上、网络上或云中或（例如）CPU 中。管理器可以被配置来协调信息的接收或亲自接收信息，并将这个信息传递给中介器，或者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流过管理器而用于本发明的消重方法并且流向中介器，或者在管理器的方向上流向系统的其它部件，但其并不流过管理器。

[0081] 在一些实施方案中，管理器可以控制一个或多个中介器的活动、与所述一个或多个中介器通信，并且协调所述一个或多个中介器的活动。对于每个中介器而言，管理器接收一组参数（或者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个信息、两个信息或全部三个信息，基本上由所述一个信息、两个信息或全部三个信息组成，或者由所述一个信息、两个信息或全部三个信息组成。

[0082] 中介器可以（例如）包括：(a) 第一组磁道；(b) 第二组磁道；(c) 第三组磁道；以及 (d) 第四组磁道。管理器致使将文件系统信息、可启动性信息和分区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。这个信息可以包括文件系统信息的识别，所述识别信息将传达如何使用预留区块。举例而言，当使用 NTFS 时，扇区 1 至 2 可以用于 MBR(主启动记录)，并且扇区 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。

[0083] 在这些实施方案中，除了前面段落中所描述的参数之外，管理器还可以接收元数据。元数据可以存储在中介器上的第三组磁道中。在管理器接收所述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和一个或多个用户感知 LBA 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来定义。可以（例如）是 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，在接收带有文件名的文件后，可以自动执行本文中所述的步骤来进行存储，包括将真实和用户感知 LBA 存储在第四组磁道的比

特字段中。

[0084] 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将接收确认自动返回给主机。在一个 QoS（服务品质）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，L2 高速缓存器会将确认传回给 L1 高速缓存器。L2 高速缓存器也可以在经受本发明的消重协议之后，将数据文件发送给执行本发明实施方案中的一个或多个实施方案的系统或系统的一部分，并写入到中介器，而且在一些情况下写入到非高速缓存器介质 (NCM) 以便长期存储。NCM 又可以将确认发回给 L2 高速缓存器。

[0085] 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（动态分配存储器）中或者可操作性地耦接至所述堆。或者，中介器可以存在于卡片内，或者可以是 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，或者可以处于固态驱动器上或任何用于存储的区块装置上。

[0086] 正如本领域普通技术人员所知晓的，将中介器放置在 L1 与 L2 中的决策将受到诸如存储数据的使用频次等因素的影响。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稍微频繁存取的数据。

[0087] 在另一 QoS 协议中，数据文件是由 L1 高速缓存器借助 I/O 来接收。数据文件从 L1 高速缓存器传递给 L2 高速缓存器和 NCM。L2 高速缓存器和 NCM 中的每一者都向 L1 高速缓存器发送确认。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。

[0088] 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。

[0089] R_1可以被配置来充当主机发起的中心点。因此，在本发明的消重方法中的任何一个方法之前，主机可选择参数来发送给 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。优选地，R_2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。

[0090] 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。

[0091] 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。

[0092] 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）。

[0093] 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。

[0094] 最后，中介器也可以包括比特字段。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便开始于磁道 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。

[0095] 作为惯例，优选地中介器并不位于存储缓冲单元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％。

[0096] 出于进一步说明的目的，可以参考附图。图 1 为本发明方法的表示图，用于本发明方法的指令可以存储在非暂时性记录介质中的永久性储存器中。出于说明性目的，图 1 中从步骤 130 至结束所示的方法是针对单个用户供应缓冲单元来展示的；然而，所述方法可以针对被分割成多个用户供应缓冲单元的给定缓冲而多次重复。当用户感知 LBA 和真实 LBA 的关联性写入到中介器时，它们的关联性一起进行分组并表示为对应于特定文件。正如本领域普通技术人员将会认识到的，如果用户供应缓冲与用户供应缓冲单元具有相同大小，那么它便不需要进行分割，并且本发明的各种实施方案可以被配置来检测这个条件。本发明的各种实施方案也可以被配置来确认所有缓冲能够被划分为具有同等大小的缓冲单元，并且，如果不能够被划分，那么便将 0 添加到缓冲的比特串的末端或者添加到原本作为最后一个缓冲单元的数据的比特串末端，以便将它们全部渲染为具有相同大小。

[0097] 如图中所展示的，可以接收指令来将信息写入到存储介质。这些指令可以呈用户感知逻辑区块地址 (LBA) 或 (LBA_x) 和用户供应缓冲单元的形式 110。

[0098] 正如上文所描述的，用户感知 LBA 是用户认为其数据将会存储的位置。用户供应缓冲单元内的数据通常将会具有其所存储的装置上的区块的大小。如果用户提交大于所述装置上的区块大小的数据，那么用户所提交的数据便可以进行预处理，从而使得所述数据含有多个缓冲单元，每个缓冲垫单元具有区块的大小。如果缓冲单元小于区块大小，那么计算机程序产品便可以将所有的 0 添加到缓冲单元的一个末端，直到它具有区块的大小。

[0099] 因此，所接收的指令呈可以由 (LBA，缓冲）表示的形式，或者转换为这个形式。对于用户供应缓冲单元而言，计算机程序产品在必要时会分割所接收的数据，并计算散列值 120。当接收多个指令时，便针对每个缓冲单元来计算散列值。

[0100] 在算法生成散列值之后，算法会查询散列值是否与已经存在于散列值表内的存储散列值重复 130。如果所述表并不含有生成散列值，那么算法便断定缓冲单元对于 NCM 而言是新的，并且将缓冲单元写入到 NCM 内之前尚未使用的区块 140。算法也会致使更新存储器中所存储的散列值表，以便包括新写入到 NCM 的这个用户生成缓冲单元与其生成散列值的关联性 150。这个生成散列值变成存储散列值，并且用户供应缓冲单元变成存储缓冲单元。在一些实施方案中，散列值表也识别缓冲单元在 NCM 上所存储的真实逻辑区块地址，然而在其它实施方案中，散列值表不包括这个信息，并且这个信息是存储在其它地方，例如，存储在中介器上或另一数据文件中。

[0101] 在更新散列值表之后，用户感知 LBA 和用户供应缓冲单元所存储的真实 LBA 会存储在中介器上，并彼此相互关联 160。

[0102] 返回到步骤 130，在图 1 内，如果所述查询得出结论是，新计算的散列值与散列值表内的存储散列值重复，那么所述方法便发起询问是否存在冲突的协议 170。

[0103] 如果不存在冲突，那么所述协议便致使中介器来存储已经处于 NCM 上且与用户供应缓冲单元相同的缓冲单元数据的位置与用户感知位置的相关性 160。因此，对于这个重复数据而言，不会将新信息写入到 NCM。

[0104] 返回到查询是否存在冲突 170，当回应为“是”时，意味着相同散列值被指配给不同缓冲单元，所述协议便会要求将用户供应缓冲单元写入到 NCM 140，并将存储器内的散列值表更新成与新写入的缓冲单元主动进行关联的散列值。另外，将会更新中介器以便含有用户感知位置和真实位置的新相关性。

[0105] 图 2 进一步说明这个步骤。类似于图 1，在进入冲突确定协议 270 后，如果存在“是”输出，那么便将缓冲单元的独特数据写入到 NCM 上的新区块 240。在写入到新区块之后，算法会更新散列值表。

[0106] 当更新散列值表时，所述方法会使之前存储的散列值与缓冲单元不再关联，使存储散列值与更近接收的缓冲单元相关联 251，并移除或去活散列值与之前存储的缓冲单元的关联性 256。在更新散列值表之后，所述协议会将最近接收的缓冲单元的真实逻辑区块地址写入到中介器，并使其与用户感知逻辑区块地址相关联 260。

[0107] 图 3 表示读取指令。系统可以接收指令来读取区块，也就是检索 (LBA，缓冲) 的信息 310。请求可以是通过文件名来请求文件的形式，所述文件名是用户与一个或多个用户感知 LBA 的系统关联性。

[0108] 协议会致使存取中介器内的相关性表，并在对应于缓冲或缓冲一部分的每个 LBA 位置上进行读取 320。因为相关性表使用户感知 LBA 与真实 LBA 相互关联，所以在获取 LBA(或多个 LBA) 的信息之后，所述协议会在指定的实际 LBA 或多个 LBA 上读取 NCM、检索区块信息并填充缓冲参数 330。在利用这个信息的情况下，系统将会使原始数据处于正确顺序中，并且能够将其发送给主机，以便借助主机的操作系统来重建请求文件。

[0109] 值得注意地，在读取步骤中，不需要存取散列值算法或表。相反，存储在 NCM 上的 LBA 可以进行读取来检索用户所调用的缓冲单元。如果任何一个或多个缓冲单元在读取的时候仍然处于 L1 或 L2 高速缓存器内，那么所述缓冲单元便可以从适当的高速缓存器中读取，而不是从 NCM 中读取。

[0110] 再举例而言，相关人员可以考虑下述表 ：

[0111] 表 1

[0112]

用户感知 LBA	用户供应缓冲单元	散列值	真实 LBA 位置
1	A	x	100
2	B	y	110

3	C	z	111
4	D	a	112
5	E	b	113
6	F	c	114
7	G	x	115
8	H	d	116
9	I	e	117
10	J	y	110
11	K	f	118
12	L	x	115
13	M	g	119
14	N	h	120
15	O	b	113
16	P	z	111
17	Q	y	121
18	R	x	122

[0113] 假设 :A ≠ G

[0114] G = L

[0115] B ≠ Q

[0116] B = J

[0117] E = O

[0118] C = P

[0119] A = R

[0120] 在用户提交用于存储的数据时,用户感知到对于 18 个区块而言 LBA 为 1 至 18,其中每个区块具有缓冲单元的大小,例如,512 个字节。当传输用于存储的数据时,用户将每个缓冲单元视为独特的,假设其是存储在不同区块处。

[0121] 在应用散列值算法后,会生成几个重复散列值。正如可以看出的,散列值 x 适用于缓冲单元 A、G、L 和 R ;散列值 y 适用于缓冲单元 B、J 和 Q ;散列值 z 适用于缓冲单元 C 和 P ;以及,散列值 b 适用于缓冲单元 E 和 O。然而,数据的真实副本位于 A 与 R、G 与 L、B 与 J、E

与 O 以及 C 与 P 之间。因此，冲突会发生在 A 与 G、G 与 R、B 与 Q 以及 Q 与 J 之间。

[0122] 正如第四列所展示的，当写入数据时，对于真实副本而言，不写入新区块。参见（例如）用户感知 LBA2 和 LBA10 所存储的实际位置。这两者都存储在区块 110 处。如果相关人员考虑第三和第三列，那么其便会看到，在分析前七个用户供应缓冲单元之后，便生成第一重复散列值（x 值）。然而，因为 A ≠ G，所以 G 必须写入到 NCM 上的区块。在这个发生之后，出于进一步冲突分析的目的，在存储于存储器中的散列值表内，x 与 G 相关联，而不与 A 相关联。

[0123] 在分析第十个缓冲单元 J 之后，相关人员会看到生成重复散列值 y。然而，因为缓冲单元是相同的，所以不存在冲突。因此，不需要对散列值表进行更新，并且不需要将新的区块写入到 NCM。相反，将会更新中介器以便指向真实 LBA110，并使其与用户感知 LBA 相互关联。在分析第 12 个用户感知 LBA 并且分析第 15 和第 16 个用户感知 LBA 之后，便会显露类似的真实副本。

[0124] 在分析缓冲单元 Q 之后，便生成散列值 y。然而，因为 Q ≠ B，所以存在冲突。因此，在 LBA121 处写入新区块，并且更新散列值表以便使 y 与 Q 相关联，而不再使 y 与 B（其与 J 相同）相关联。

[0125] 在分析用户供应缓冲单元 R 之后，便生成散列值 x。R = A。然而，A 并非是与散列值表内的 x 的最近关联性。因此，所述协议会将 R 视为其是第一次看到 R，并且必须执行下述两个操作：(1) 将新的相关性保存在散列值表中或者改写旧的相关性（但是所述协议不会复原存储器中所存储的关联性）；以及 (2) 将新区块的数据存储在 NCM 中（这里是区块 122)。因此，示例中的 NCM 将含有某个副本。

[0126] 表 1 是出于说明性的目的，并且在一些实施方案中，只有第一列和第四列是中介器的一部分，而且它们不是散列值表的一部分。中介器的大小要求是较小的。举例而言，在一些实施方案中，对于存储在 NCM 上、大小为 512 字节或 4K 的每个缓冲单元而言，中介器需要 8 个或 16 个字节。

[0127] 借助本发明的各种实施方案，可以大大减小 NCM 的物理存储空间。举例而言，它们可以减小至少 50％、至少 100％、至少 200％、至少 500％或至少 1000％。因此，在一些实施方案中，所需要的存储空间量比标准条件下所需要的存储空间量小 50 至 150 倍。举例而言，在中介器上仅仅需要 5 至 10 个字节来存储对应于例如为 512 字节至 4K 的缓冲单元的数据。因此，通过降低多次存储相同缓冲单元（或分割缓冲单元）的需求，8GB USB 棒（其是本发明的 NCM 的示例）可以用来存储与 1.53TB 数据对应的内容。

[0128] 已经在假设所存储的缓冲单元与从用户接收的缓冲单元相同（即使进行分割）的情况下，描述了本发明的方法、系统和计算机程序产品。在这些实施方案中，因为所述缓冲单元是存储在与用户所感知的 LBA 不同的 LBA 处，并且是以用户无法感知的顺序存储的，所以需要中介器来用于检索信息，并且在没有中介器的情况下用户无法检索数据。因此，中介器提供一定程度的安全性。

[0129] 使用相同散列值算法的不同用户将会生成相同的散列值。因此，它们的散列值表将是类似的，除了 NCM 上缓冲单元的位置。另外，中介器中的相关性信息将会是不同的。作为额外的安全性级别，在进入本发明的方法之前，用户可能希望编码或转换数据。这些动作可以称为预处理。

[0130] 在一些实施方案中，在进入上文所述的协议或系统之前，用户的数据首先通过使用比特标记符表或频次转换器或其它散列值算法进行转换，以便生成大小上更小和 / 或编码的数据。用于执行这些技术的方法、系统和计算机程序产品公开于下述申请中 :2013 年 2 月 2 日提交的、标题为 Bit Markers and Frequency Converters 的 U.S.13/756,921 ; 2013 年 3 月 12 日提交的、标题为 Data Storage and Retrieval Mediation Systems and Methods for Using Same 的 U.S.13/797,003 ;以及，2013 年 6 月 3 日提交的、标题为 Methods and Systems for Storing and Retrieving Data 的 U.S.13/908,239。上述申请的所有公开内容全部以引用的方式并入本文中。这些方法的输出可以用来生成缓冲单元以便减少重复。

[0131] 这些申请描述了可以作为预处理步骤 (也就是，在目前实施方案的消重策略之前) 并入到本发明中的方法论。在这些情况下，缓冲单元将会对应于预处理方法的输出，依靠所述预处理方法、通过使用比特标记符表或频次转换器来转换数据。

[0132] 因此，在一个实施方案中，这个预处理步骤包括 :(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比特组合的数目，同样的子单元指配有相同标记符，并且至少一个标记符小于子单元的大小 ;以及 (iii) 使用所述标记符作为缓冲单元。这个预处理步骤利用比特标记符表，所述比特标记符表可以与散列值表存储在同一个或不同的永久性存储器中。当读取数据时，这些预处理步骤可以按照相反顺序执行，并且是在从 NCM 中检索数据且 NCM 上的缓冲单元以中介器所传达的方式重新组合之后执行。当使用标记符作为缓冲单元时，相关人员可以组合或划分标记符，从而形成具有必要大小的缓冲单元。

[0133] 比特标记符表可以含有全部具有相同大小或全部具有不同大小的标记符。当具有不同大小时，正如下文所述的，这些大小可以由比特串或字节串的预测频次来决定。

[0134] 再举例而言，当预处理步骤利用比特标记符表时，原始数据会转译成一系列表示原始数据的标记符。原始数据对应于从主机所接收的数据，且因此可以 (例如) 是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。

[0135] 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生成标记符。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。

[0136] 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。

[0137] 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。

[0138] 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，从而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。

[0139] 正如上文所提到的，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％。

[0140] 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。

[0141] 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将会认识到的，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。标记符的输出可以按照允许它们进行组合来形成用户供应缓冲单元以便输入到散列值算法中的顺序，正如上文所描述的。

[0142] 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。

[0143] 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。

[0144] 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。因此，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。

[0145] 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。当使用来自比特标记符表或频次转换器的标记符时，在它们具有不同大小的情况下，它们必须进行格式化，从而使得系统可以知道一个标记符结束的地方以及下一个标记符开始的地方。这可以（例如）通过下述操作来完成：设置最小标记符大小，并进行读取分析来查询每个具有最小大小的比特串在所述表或转换器内是否为独特的，而且，如果不是独特的，便继续通过读取额外比特而使比特串增大，并重复对每个额外比特的查询。

[0146] 信息可以进行转换，并且输出代码可以被配置成小于输入，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生成的数据，具有相同大小或比其更大。

[0147] 根据另一实施方案，所述预处理步骤包括：(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（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一个生成散列函数值；(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作为用户供应缓冲单元；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，并使用所述生成散列函数值作为用户供应缓冲单元。

[0148] 当使用这个预处理散列值算法时，相关人员需要解决它们发生冲突的可能性。作为上文所述的用于消重的方法（其中保持和存储最近的散列值关联性）的替代方法，在这个可选预处理步骤中，相关人员可以使用冲突解决模块，从而使得，在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，但对于这个散列函数值而言用户供应小盘不同于存

储小盘的情况下，某个方法会致使存在不同的 Z 个比特与所述存储散列函数值和所述生成散列函数值相关联。这个技术在 2013 年 6 月 3 日提交的美国专利申请第 13/908,239 号中加以描述，所述申请的全部内容以引用的方式并入本文中。

[0149] 因此，这个预处理步骤可以利用第一散列值表，并且，对于不存在冲突的所有散列函数值而言，关联 Z 个比特且所述 Z 个比特是（例如）8 至 16 个 0 的统一长度。举一个非限制性示例，当具有 8 个字节的校验和并未与之前存储的校验和冲突时，所述方法可以在所述校验和的末端处关联 8 个 0。在识别冲突（例如，不同的片段单元与相同校验和关联）后，最新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 值为 00000000，那么针对第一冲突校验和的 Z 值便可以为 00000001，并且应当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便可以在冲突校验和被识别时指配有下一个 Z 值。因此，可以在存取相关性文件之后且只有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，或者所述校验和与来自接收文件的片段单元是否已经在相关性文件内彼此关联。这些扩展文件可以用作替代方式来替换或重写存储散列值与存储缓冲单元的关联性。这些带有扩展的校验和可以组合成必要大小来形成作为用户供应缓冲单元的输入。

[0150] 在预处理步骤使用散列值算法的任何情况下，将要应用的第一散列值算法可以称为利用第一散列值表的第一散列值算法或预处理散列值算法，并且第二散列值算法可以称为利用第二散列值表的消重散列值算法或第二散列值算法。当预处理散列值算法和第二散列值算法都使用时，正如上文所提到的，优选地，为了解决在使用消重散列值算法时的冲突，对应散列值表通过优先选择最近的关联性而在冲突关联性之间做出抉择，然而，为了解决在使用预处理散列值算法时的冲突，对应散列值表使用上文所述的扩展方法。

[0151] 当使用预处理技术时，输出（例如，比特标记符）可以与缓冲单元具有相同大小。在一些实施方案中，比特标记符可以大于缓冲单元的大小。在这些情况下，系统可以将它们分割来形成缓冲单元，或者包含默认模块，所述默认模块拒绝任何大于编码针对的数据的比特标记符，而是使用起初的原始数据来形成缓冲单元，从而绕过对比特标记符表的存取。在其它实施方案中，比特标记符可以更小，并且需要组合来形成缓冲单元或形成缓冲以便分割成缓冲单元。这些步骤可以根据计算机程序产品内的模块，而在服务器上执行、在云中执行或者通过 CPU 来执行。

[0152] 如果数据预处理是写入过程的一部分，那么数据后处理必须是读取过程的一部分。与本发明的消重步骤的读取不同，后处理步骤与预处理步骤是对称的，但是以相反的顺序执行。

[0153] 另外，本发明的各种实施方案可以与其它用于保护数据免受损失的方法组合使用。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上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。

[0154] 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 LBA。

[0155] 在时间 T1 之后，例如在时间 T2 上，主机可以试图更新其认为是存储在（例如）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 NCM 上的信息，而是第一中介器可以生成与主机认为是更新后文件的数据相对应的新相关性条目。因为写入在 NCM 上的大多数缓冲单元（即便不是全部）都是独特条目，所以中介器上的新相关性将会仅仅针对与起初相关性中的缓冲单元不同的缓冲单元，而不同于起初相关性。因此，在针对文件 (A) 的时间 T0 上，第一中介器可以关联下述真实 LBA :200、201、202、203、204、205、206。在 T1 上，可以制作中介器的复本。在 T2 上，用户可以尝试更新文件 (A)。在第一中介器上，可以保存新相关性，所述新相关性指向下述真实 LBA :200、201、310、203、204、205、206。然而，第二中介器将不会改变。因此，这两个中介器就其所指向的位置而言是不同的。早先保存的相关性可以在第一中介器上渲染为不活动的，或者删除或重写。

[0156] 使用这两个中介器，将会允许相关人员提供数据在 T1 上存在时的快照，而不会致使主机需要更新其文件系统来指示正存储 T1 和 T2 上都存在过的文件。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，事实上这时仅仅存储文件新的部分，并且制作新的中介器条目。

[0157] 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。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其预期功能，并在网络上借助一个或多个门户来与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。

[0158] 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）或者在不规律间隔上（例如根据需要）获取快照。

[0159] 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。

[0160] 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，用户可以指引系统来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。仅仅新的缓冲单元（或未与散列值主动进行关联的缓冲单元）会添加到第一非高速缓存器存储介质，并且不会改写第一非高速缓存器存储介质上的数据。相反，新相关性会写入在第一中介器上。不会对第二中介器或第二非高速缓存器存

储介质做出任何更改。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。

[0161] 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。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。

[0162] 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。

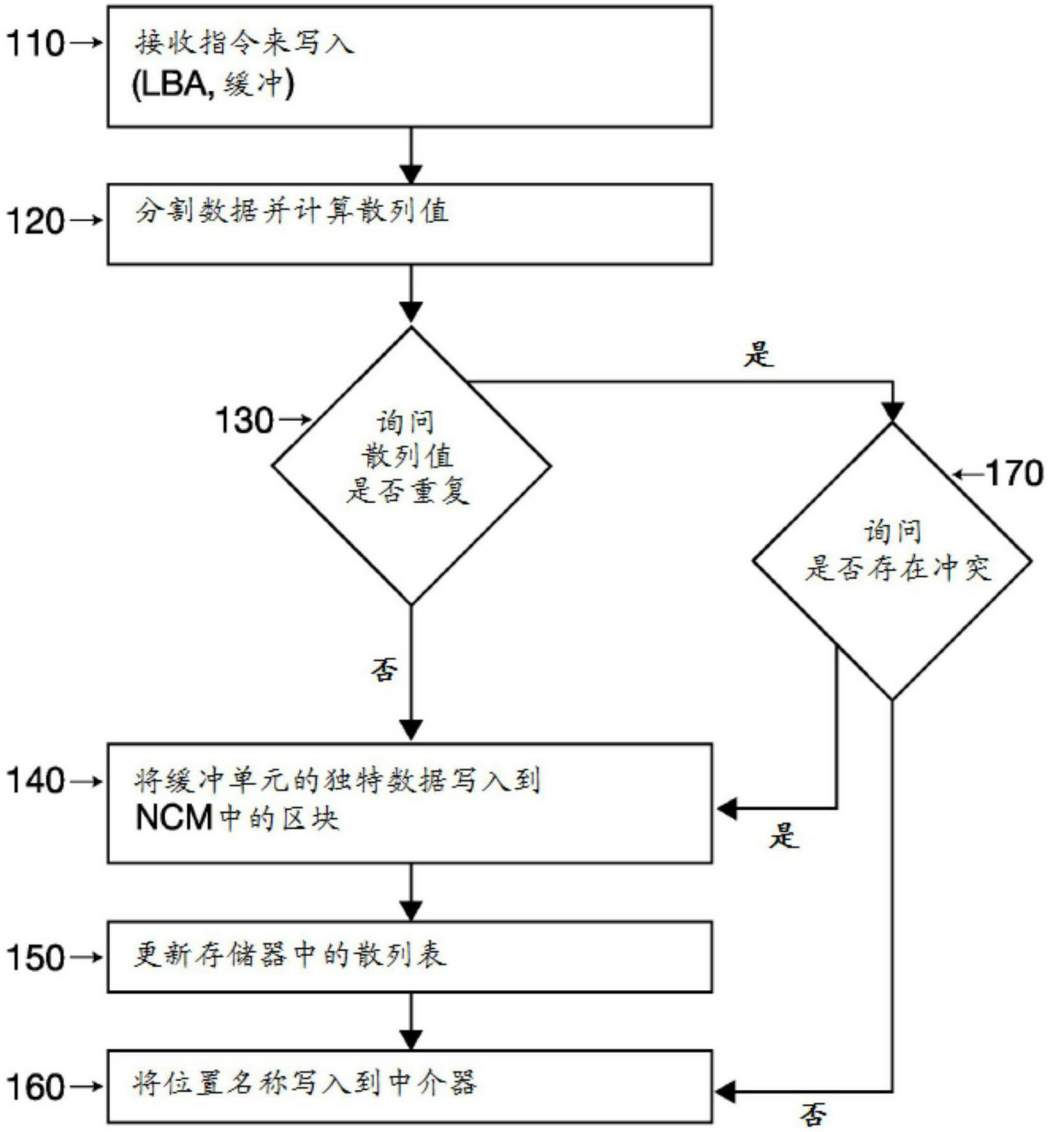

110
接收指令来写入
(LBA, 缓冲)
120
分割数据并计算散列值
130
询问
散列值
是否重复
是
170
询问
是否存在冲突
否
140
将缓冲单元的独特数据写入到
NCM中的区块
是
150
更新存储器中的散列表
160
将位置名称写入到中介器
否

图 1

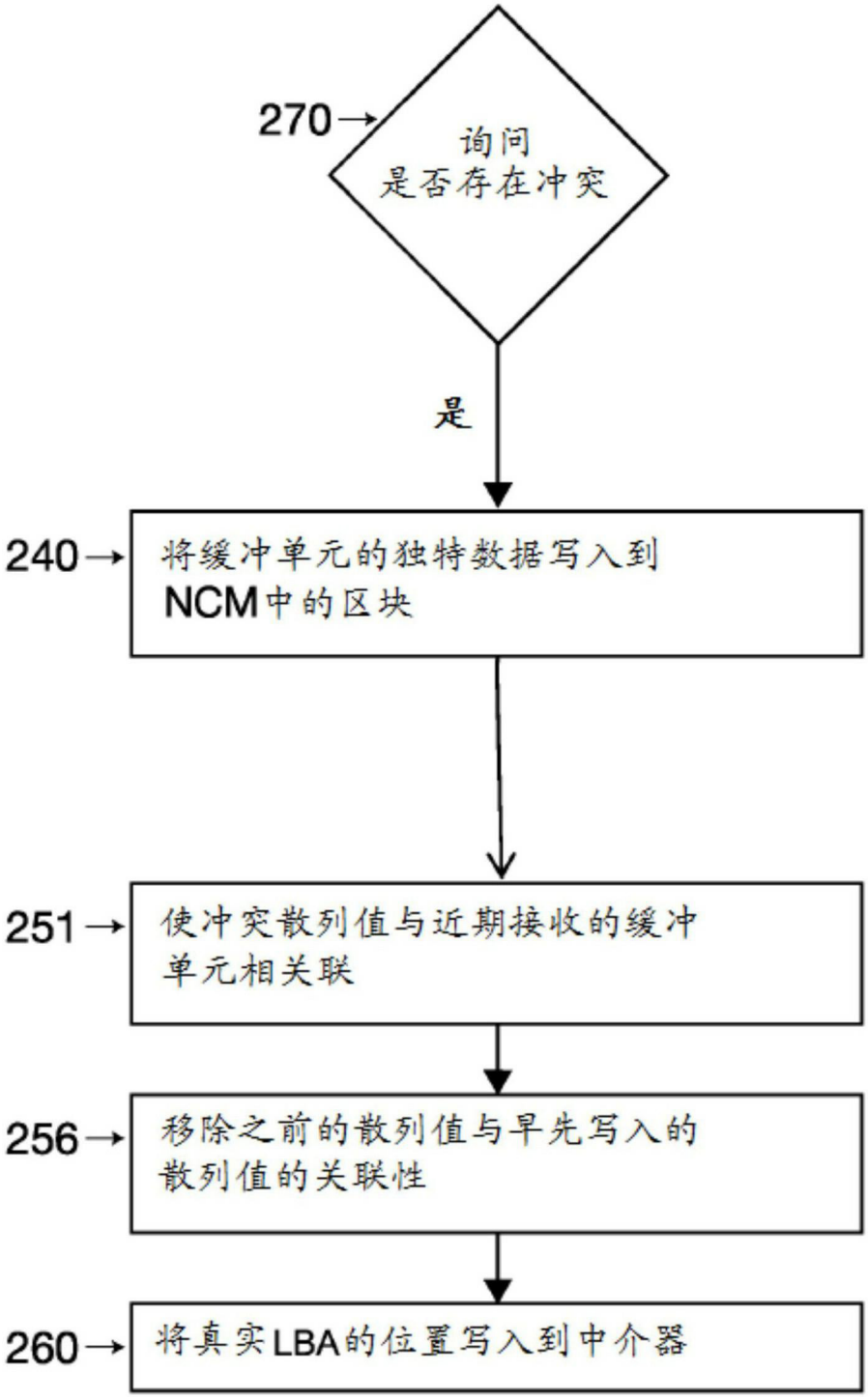

270
询问
是否存在冲突
是
240
将缓冲单元的独特数据写入到
NCM中的区块
251
使冲突散列值与近期接收的缓冲
单元相关联
256
移除之前的散列值与早先写入的
散列值的关联性
260
将真实LBA的位置写入到中介器

图 2

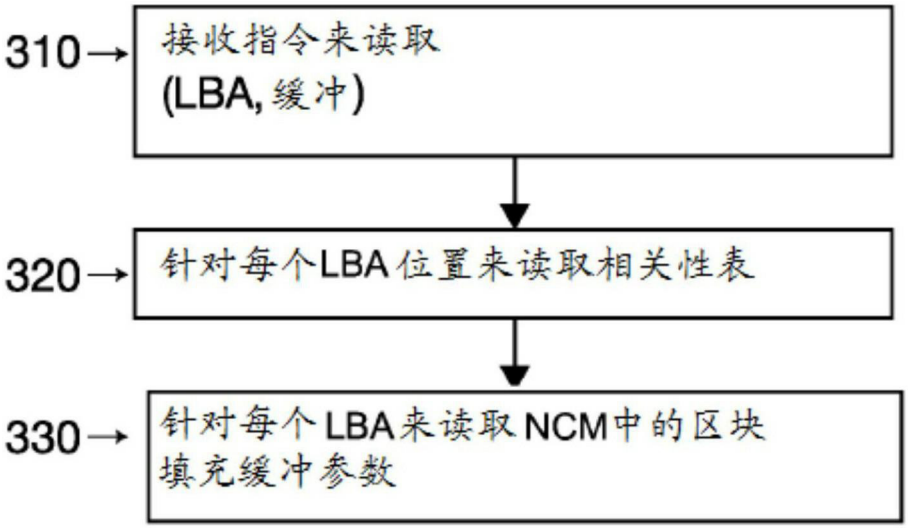

310
接收指令来读取
(LBA, 缓冲)
320
针对每个LBA位置来读取相关性表
330
针对每个LBA来读取NCM中的区块
填充缓冲参数

图 3