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

(19) World Intellectual Property Organization
International Bureau

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

WIPO | PCT

(10) International Publication Number
WO 2014/121102 A2

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

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

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

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

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

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

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

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

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

[Continued on next page]

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

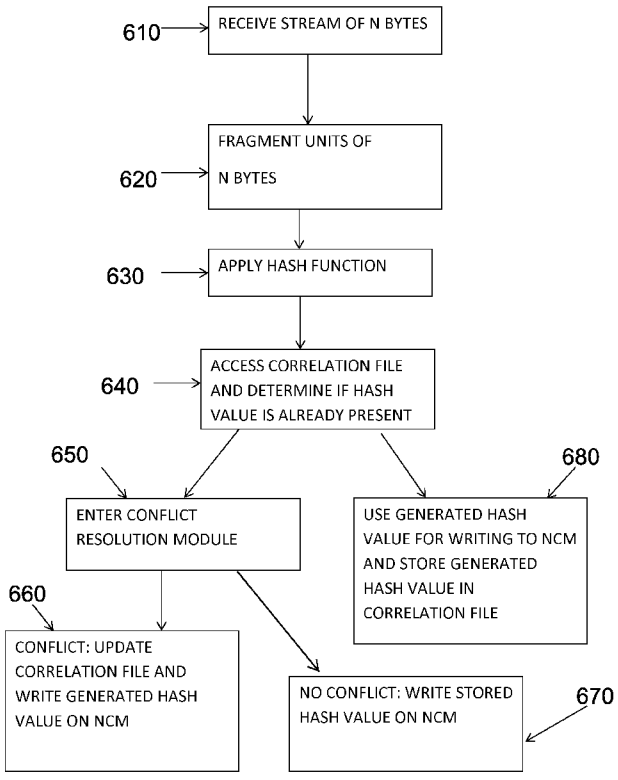

Figure 6

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

WO 2014/121102 A2

Declarations under Rule 4.17:

— *of inventorship (Rule 4.17(iv))*

Published:

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

Methods and Systems for Storing and Retrieving Data

[0001] Field of the Invention

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

[0003] Background of the Invention

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

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

[0006] One system that is used for storage is a Network Attached Storage (NAS) system. In the context of NAS, a storage server operates on behalf of one or more clients to store and to manage file-level access to data. The files may be stored in a storage system that includes one or more arrays of mass storage devices, such as magnetic or optical disks or tapes. Additionally, this data storage model may employ Redundant Array of Independent Disks (RAID) technology.

[0007] Another system that is used for storage is a Storage Area Network (SAN). In a SAN system, typically a storage server provides clients with block-level access to stored data, rather than file-level access to it. However, some storage servers are capable of providing clients with both file-level access and block-level access.

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

[0009] Summary of the Invention

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

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

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

either an order that corresponds to the order of the chunklets or another configuration that permits recreation of the order of the chunklets.

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

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

[0015] According to a fourth embodiment, the present invention provides a method for storing data on a recording medium comprising: (i) receiving a plurality of digital

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

[0016] According to fifth embodiment, the present invention provides a method for storing data on a recording medium comprising: (i) receiving a plurality of digital binary signals, wherein the digital binary signals are organized in chunklets, wherein each chunklet is N bits long, each chunklet has a first end and a second end, N is an integer number greater than 1, and the chunklets have an order; (ii) dividing each chunklet into a plurality of subunits, wherein each subunit is A bits long; (iii) analyzing each subunit to determine if the bit at the first end has a value 0 and if the

bit at the first end has a value 0, removing the bit at the first end and all bits that have the value 0 and form a contiguous string of bits with the bit at the first end, thereby forming a first revised subunit for any subunit that has a 0 at the first end; (iv) analyzing each subunit to determine if the bit at the second end has value 0 and if the bit at the second end has a value 0, removing the bit at the second end and all bits that have the value 0 and form a contiguous string of bits with the bit at the second end, thereby forming a second revised subunit for any subunit that has a 0 at the second end; and (v) for each subunit (a) if the sizes of the first revised subunit and the second revised subunit are the same, storing the first revised subunit or the second revised subunit, (b) if the first revised subunit is smaller than the second revised subunit, storing the first revised subunit, (c) if the second revised subunit is smaller than the first revised subunit, storing the second revised subunit, (d) if there are no revised subunits, storing the subunit, (e) if there is no first revised subunit, but there is a second revised subunit, storing the second revised subunit, (f) if there is no second revised subunit, but there is a first revised subunit, storing the first revised subunit, wherein each revised subunit that is stored is stored with information that indicates if one or more bits were removed from the first end or the second end. The information that indicates if one or more bits were removed from the first end or the second end may, for example, be in the form of the uniqueness of the subunit.

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

[0018] According to a seventh embodiment, the present invention is directed to a method for storing electronic data, said method comprising: (i) receiving a set of parameters, wherein the parameters comprise one or more of file system information, bootability information and partition information; (ii) receiving metadata; (iii)

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

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

[0020] According to a ninth embodiment, the present invention provides a data storage and retrieval system comprising: (i) a non-cache data storage medium; (ii) a mediator, wherein the mediator is stored remotely from the non-cache data storage medium, and the mediator comprises: (a) a first set of tracks; (b) a second set of tracks; (c) a third set of tracks; and (d) a fourth set of tracks; and (ii) a manager, wherein the manager is configured: (a) to store data comprising one or more of file system information, bootability information and partition information in the first set of tracks; (b) to store metadata in the third set of tracks; (c) to store one or more files

on the non-cache medium, wherein the one or more files are stored on the non-cache medium without any of file system information, bootability information and partition information; (d) to store in the fourth set of tracks the location of each file in the non-cache medium; and (e) to store a correlation of the location of each file in the non-cache medium with a host name for a file. Thus, the manager may cause storage of information on the mediator that may not be stored on the manager itself.

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

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

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

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

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

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

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

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

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

embodiments of the present invention may be used in conjunction with or independent of RAID technologies.

[0029] Brief Description of the Figures

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

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

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

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

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

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

[0036] Detailed Description of the Invention

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

[0038] *Definitions*

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

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

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

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

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

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

[0045] The term "chunklet" refers to a set of bits that may correspond to a sector cluster. The size of a chunklet is determined by the storage system and may have a

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

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

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

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

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

[0050] The terms "host" and "initiator" may be used interchangeably and refer to the entity or system that sends data for storage to the data storage and retrieval mediation system of the present invention. The host may send data that corresponds to one or more types of documents or files and received data. Preferably, within any

input/output ("I/O") stream, the data corresponds to a file of a single document type. For convenience the phrase "I/O stream" is used to refer that data is transmitted from on entity to another. What is output for a first entity may be input for a second entity.

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

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

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

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

certain geometry. In some embodiments, the mediator resides permanently on L2 cache. In other embodiments it reside on a solid state drive.

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

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

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

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

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

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

[0061] The phrase "recording medium" refers to a non-transitory tangible computer readable storage medium in which one can store magnetic signals that correspond to bits. By way of example, a recording medium includes but is not limited to a non-

cache medium such as hard disks and solid state drives. As persons of ordinary skill in the art know, solid state drives also have cache and do not need to spin. Examples of non-transitory tangible computer readable storage media include, but are not limited to, a hard drive, a hard disk, a floppy disk, a computer tape, ROM, EEPROM, nonvolatile RAM, CD-ROM and a punch card.

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

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

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

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

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

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

[0068] ***Preferred embodiments***

[0069] The present invention provides methods for storing data on a non-cache recording media, computer program products for carrying out these methods, and

systems that are configured to carry out these methods. Through various embodiments of the present invention, one can efficiently store and retrieve data.

[0070] Bit Markers

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

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

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

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

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

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

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

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

markers within the bit marker table is equal to the number of combinations of bits within a string of bits of size N or A. In other embodiments, all of the bit markers are the same size, and the number of bit markers within the bit marker table is less than 90%, less than 80%, less than 70% or less than 60% of the number of combinations of bits within a string of bits of size N or A.

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

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

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

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

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

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

[0085] Frequency Converters

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

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

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

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

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

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

[0092] Fragmentation

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

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

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

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

[0097] Preprocessing

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

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

[0099] Truncation

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

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

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

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

[00104] In various methods described above one can remove the digit(s) from the first end or the second end of each subunit or of each chunklet, but not both. However, it is within the scope of various embodiments of the present invention to practice methods in which one considers removing digits from the first end of each subunit or chunklet, one separately considers removing digits from the second end of

each subunit or chunklet, for each subunit or chunklet one analyzes whether truncation occurs at either, one or both of the first end and the second end, and if it occurs at only one end, saving the truncated chunklet or subunit, and if it occurs at both ends, then saving the smaller of the truncated units. It is also within the scope of the present invention to practice methods in which digits could be removed from both ends of a chunklet or subunit.

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

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

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

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

[00109] One may also store information that indicates if one or more bits were removed from the first end or the second end or one could use a first bit marker table for units for which bits are removed from the first end and a second bit marker table

for units for which bit markers are removed from the second end, and between the two bit marker tables, there are no duplications of bit markers. These two different bit marker tables can be organized as sections of the same table and include bit markers for units that are not revised. In the table or tables, there are no duplications of the bit markers for first revised units, second revised units and any units that are not revised because, for example, they have 1s at both ends.

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

[00111] **Hash Values**

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

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

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

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

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

[00117] Additionally, the above described method applies the algorithm only if the string of Bytes is not already within the correlation file. In another embodiment,

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

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

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

[00120] **Conflict Resolution Modules**

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

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

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

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

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

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

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

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

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

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

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

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

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

the hash values will cause less storage space to be needed and there can be further cost savings.

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

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

[00135] **Mediators and Managers**

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

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

received from a host, a manager may cause acknowledging receipt of the I/O stream to the host.

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

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

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

[00141] In addition to using a mediator, various embodiments of the present invention call for the use of a manager. The manager, which may comprise one or more modules and reside on a local computer, on a network or in a cloud. The

manager is configured to coordinate receipt of or to receive certain information itself and to transfer this information to a mediator, or to control receipt of the information directly by the mediator. Thus, the methods can be designed such that information from the initiator flows through the manager to mediator or that the manager only directs the flow of information to other components of the system, *e.g.,* to the host or NCM.

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

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

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

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

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

[00147] **Cache**

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

long term storage. The NCM may in turn send an acknowledgement back to L2 cache.

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

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

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

[00152] **File Correlation**

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

[00154] As persons of ordinary skill in the art will recognize, methods and systems may be used in which the host generates the first storage address and sends it along to the systems of the present invention with SCSI commands and optionally associated sector or LBA number(s). The mediator may correlate the file name, what the host thinks of as the location of the file and the storage size of the file as received

from the host, *i.e.*, the raw data and any header or footer data, with a second storage address, which is the actual storage address of the data, which may contain information from which the file can be recreated, but that has been converted through for example one or more of a hash value algorithm, bit marker table and frequency converter. Alternatively, the mediator may store only the file name, and optionally, it may not receive the first storage address for a file. As noted above, because storage addresses are based on a linear organization of data, they may implicitly contain the size of the stored information by reciting only the first user perceived LBA or explicitly reciting all locations.

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

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

[00157] Reserve tracks

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

[00159] R_1 may be configured to function as the central point for host initiation. Thus, the host may select the parameters to send to R_1. The mediator may receive this information directly from the host or indirectly through the manager. R_2 is preferably

never exposed to the host. Thus, only the mediator itself or the manager can cause information to be stored in R_2. Each of R_1 and R_2 may, for example, contain sixteen sectors and be filled with real data such as host modifiers. By convention, numbering may start at 0. Thus, R_1 may, for example, contain sectors (or tracks) 0 -15, and R_2 may contain sectors (or tracks) 16 -31. However, the mediator may be constructed so as to allow for expansion of each of R_1 and R_2 beyond the initial size of 16 tracks.

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

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

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

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

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

[00165] **Bit Field**

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

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

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

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

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

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

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

[00172] **Backing-up of data**

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

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

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

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

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

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

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

a system administrator to select certain blocks that have been written as of a certain time or to select all blocks that have been written as of a certain time not to be overwritten.

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

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

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

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

access to an earlier version, which would be stored on the second non-cache medium and could retrieve it by going through the second mediator.

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

[00185] **Data retrieval**

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

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

[00188] In one method, one begins by accessing a recording medium. The recording medium stores a plurality of markers in an order, and from these markers, one can recreate a file. Access may be initiated by host requesting retrieval of a file

and transmitting the request to a storage area network or by the administrator of the storage area network.

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

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

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

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

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

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

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

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

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

[00198] In order to illustrate the various embodiments further and to provide context, reference is made below to specific hardware that one may use, which may be combined to form a system to implement the methods of the present invention. The hardware may be configured to allow for the above described process to be

automated and controlled by one or more processors upon receipt of one or more data files from a host.

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

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

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

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

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

consulting a bit marker table, prior to consulting a frequency converter, and prior to truncating bits, and/or after consulting a hash function algorithm, after consulting a bit marker table, after consulting a frequency converter, and after truncating bits.

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

[00205] **Systems**

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

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

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

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

[00210] The manager may be configured to carry out one or more of the following features: (a) to store data comprising one or more of file system information, bootability information and partition information in the first set of tracks

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

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

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

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

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

of the subunits, and the computer program product creates a new data file that contains one or more of the markers in place of the original subunits.

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

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

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

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

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

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

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

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

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

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

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

[00226] **Examples:**

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

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

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

[00230] **Table I**

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

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

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

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

[00234] Because as the subunit size gets larger the table becomes more cumbersome, in some embodiments, the table may be configured such that all zeroes

from one end of the subunit column are missing and prior to accessing the table, all zeroes from that end of each subunit are removed. Thus, rather than a table from which Table I is excerpted, a table from which Table II is excerpted could be configured.

[00235] **Table II**

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

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

[00237] In order to implement this method, one may deem each subunit (or each chunklet if subunits are not used) to have a first end and a second end. The first end can be either the right side of the string of bits or the left side, and the second end would be the opposite side. For purposes of illustration, one may think of the first end as being the leftmost digit and the second end as being the rightmost digit. Under this method one then analyzes one or more bits within each subunit of each chunklet to

determine if the bit at the second end has a value 0. This step may be referred to as preprocessing, and the subunits after they are preprocessed appear in the right column of Table II. If the bit at the second end has a value 0, the method may remove the bit at the second end and all bits that have the value 0 and form a contiguous string of bits with that bit, thereby forming a revised subunit (pre-processed subunit in the table) for any subunit that originally had a 0 at the second end.

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

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

[00240] **Example 3: Frequency Exchange (Prophetic)**

[00241] Based on empirical analysis, one can determine the frequency of each subunit within a type of document or a set of documents received from a particular host or from within a set of documents that have been received within a given timeframe, *e.g.*, the past year or past two years. With this information, rather than

look to a table as illustrated in Table I or Table II in which the subunits are organized in numerical order, one could look to a frequency converter in which the smaller bit markers are associated with subunits that are predicted most likely to appear within a file, within a type of document or within a set of documents as received from a particular host. Thus, within the frequency converter, the markers are a plurality of different sizes and markers of a smaller size are correlated with higher frequency subunits.

[00242] Table III: Frequency Converter

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

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

zeroes from one end or the other, and the table could be designed to contain the correlating truncated subunits.

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

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

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

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

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

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

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

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

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

[00253] **Table IV**

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

[00254] **Example 5: Space Saving**

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

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

Claims

I claim:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

18. A method for storing data on a non-cache recording medium comprising:

i. receiving an I/O stream of N Bytes;

ii. fragmenting the N Bytes into fragmented units of X Bytes;

iii. associating a cryptographic hash function value with each fragmented unit of X Bytes, wherein said associating comprises accessing a correlation file, wherein the correlation file associates a stored hash function value of Y bits with each of a plurality of stored sequences of X Bytes and

(a) if the sequence of a fragmented unit of X Bytes is in the correlation file, then using the stored hash function value of Y bits for storage on a non-cache recording medium; and

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

19. A method for storing data on a non-cache recording medium comprising:

i. receiving an I/O stream of N Bytes;

ii. applying a cryptographic hash function value algorithm to each unit of N Bytes to form a generated hash function value for each unit of N Bytes;

iii. accessing a correlation file, wherein the correlation file associates a stored hash function value of Y bits with each of a plurality of stored sequences of N Bytes and

(a) if the generated hash function value for the unit of N Bytes is in the correlation file, using the stored hash

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

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

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

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

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

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

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

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

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

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

(a) reviews each subunit to determine whether at the second end there is a 0 and if so removes the 0 to form a revised subunit with a revised second end at a position that was adjacent to the second end of the subunit;

(b) reviews each revised subunit to determine whether at the revised second end there is a 0 and if so removing the 0 to form a further revised second end; and

(c) repeating (b) for each revised subunit until a shortened subunit is generated that has a 1 at its second end.

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

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

(a) reviews each subunit to determine whether at the second end there is a 1 and if so removes the 1 to form a revised subunit with a revised second end at a position that was adjacent to the second end of the subunit;

(b) reviews each revised subunit to determine whether at the revised second end there is a 1 and if so removing the 1 to form a further revised second end; and

(c) repeating (b) for each revised subunit until a shortened subunit is generated that has a 0 at its second end.

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

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

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

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

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

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

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

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

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

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

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

iv. on a non-transitory recording medium, storing in said order each revised subunit and each subunit that is A bits long and has a 1 at its second end.

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

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

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

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

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

v. for each subunit

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

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

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

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

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

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

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

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

i. accessing a recording medium, wherein the recording medium stores a plurality of data units in a plurality of locations,

wherein each data unit contains a plurality of bits and the maximum size of the data unit is N bits, at least one data unit contains fewer than N bits and the data units have an order;

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

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

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

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

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

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

ii. receiving metadata;

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

iv. storing the parameters and metadata on a mediator;

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

50. A data storage and retrieval system comprising:

i. a non-cache data storage medium;

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

(a) a first set of tracks;

(b) a second set of tracks;

(c) a third set of tracks; and

(d) a fourth set of tracks; and

iii. a manager, wherein the manager is configured:

(a) to store data comprising one or more of file system information, bootability information and partition information in the first set of tracks.

(b) to store metadata in the third set of tracks;

(c) to store one or more files on the non-cache medium, wherein the one or more files are stored on the non-cache medium without any of file system information, bootability information and partition information;

(d) to store in the fourth set of tracks the location of each file in the non-cache medium; and

(e) to store a correlation of the location of each file in the non-cache medium with a host name for a file.

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

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

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

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

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

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

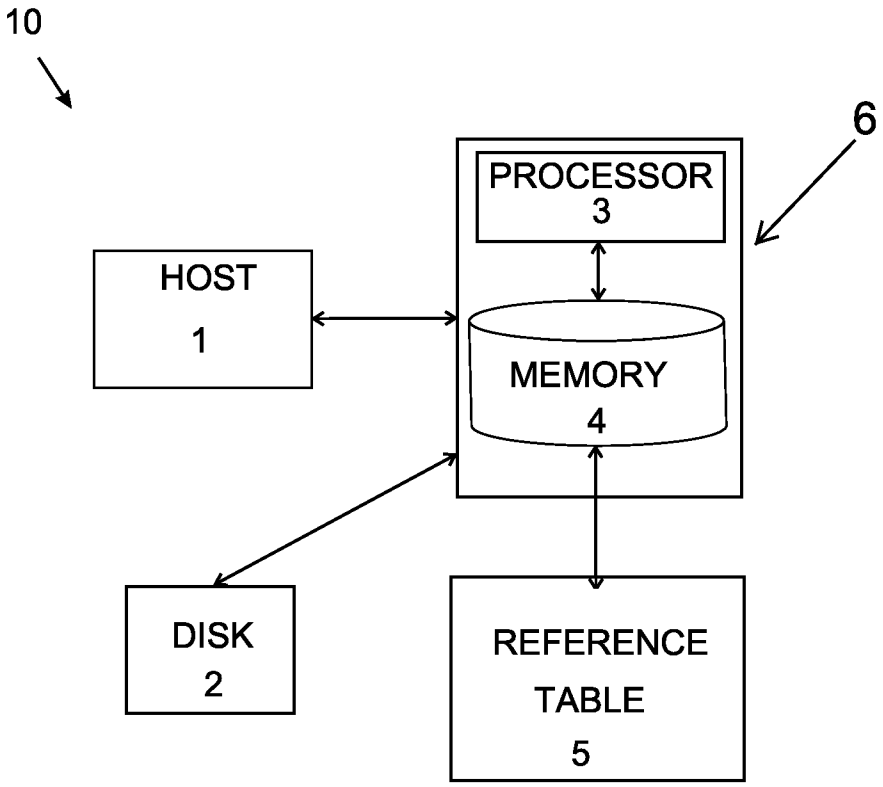

Figure 1

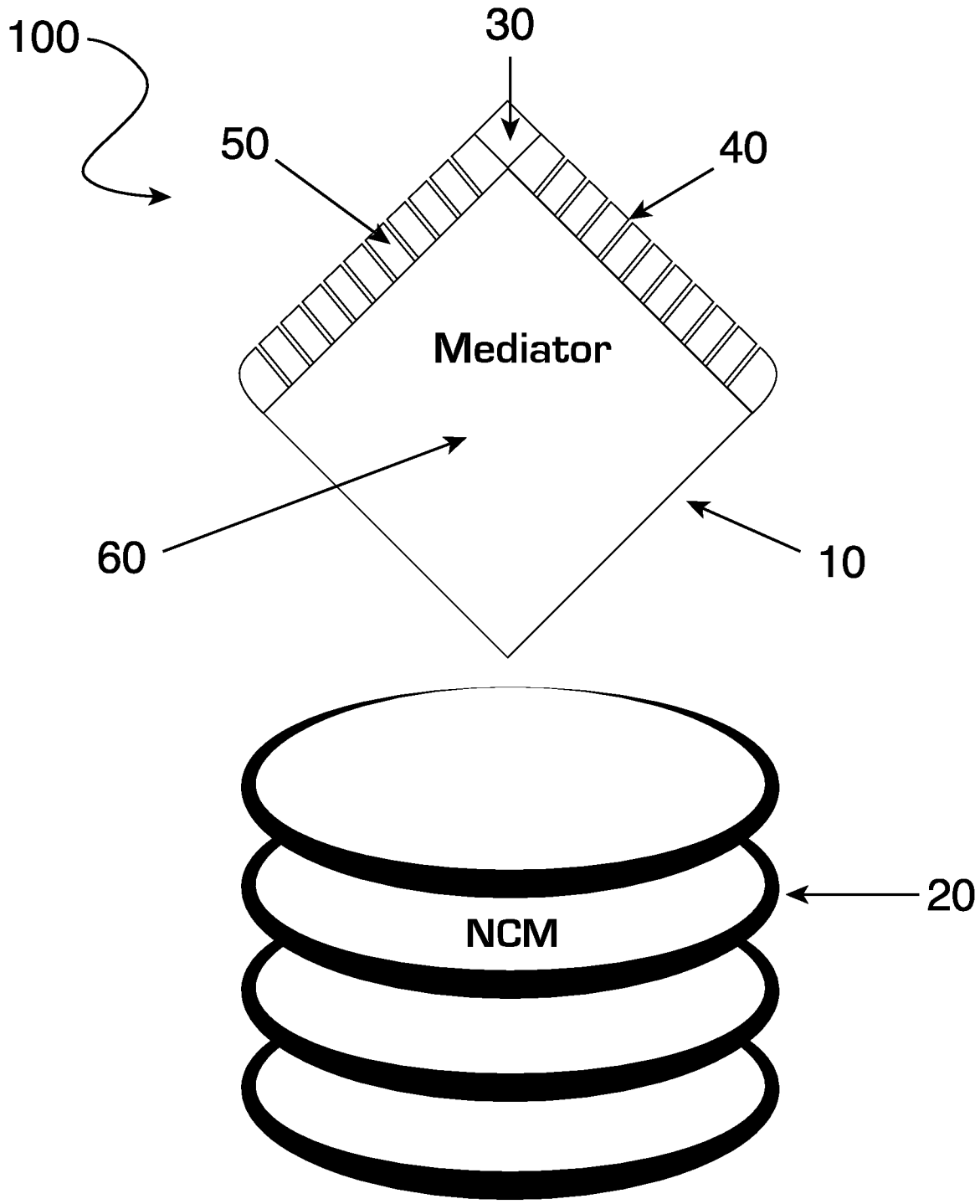

Figure 2

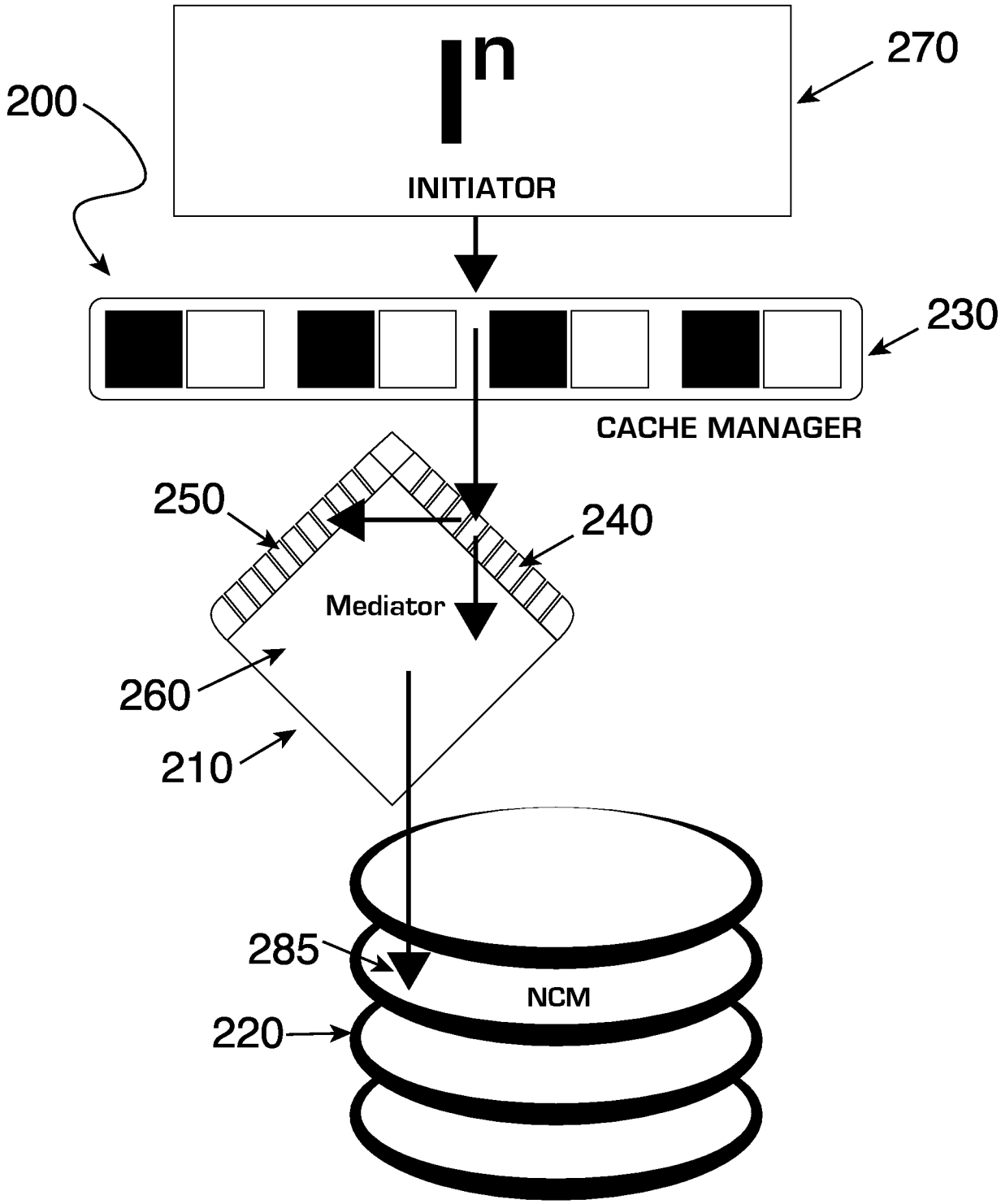
200
270
INITIATOR
230
CACHE MANAGER
250
240
Mediator
260
210
285
NCM
220

Figure 3

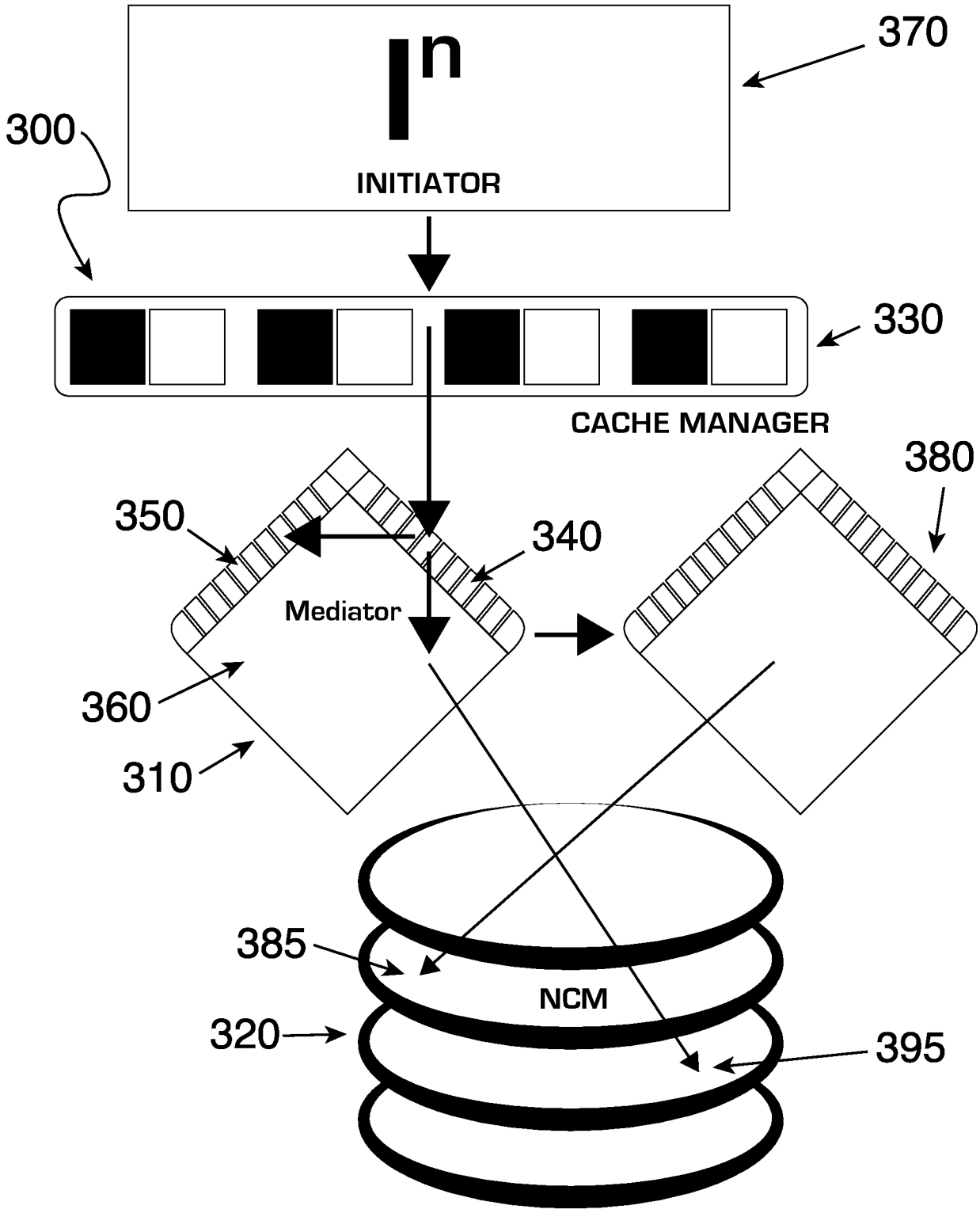
In
INITIATOR
370
300
330
CACHE MANAGER
380
350
340
Mediator
360
310
385
NCM
320
395

Figure 4

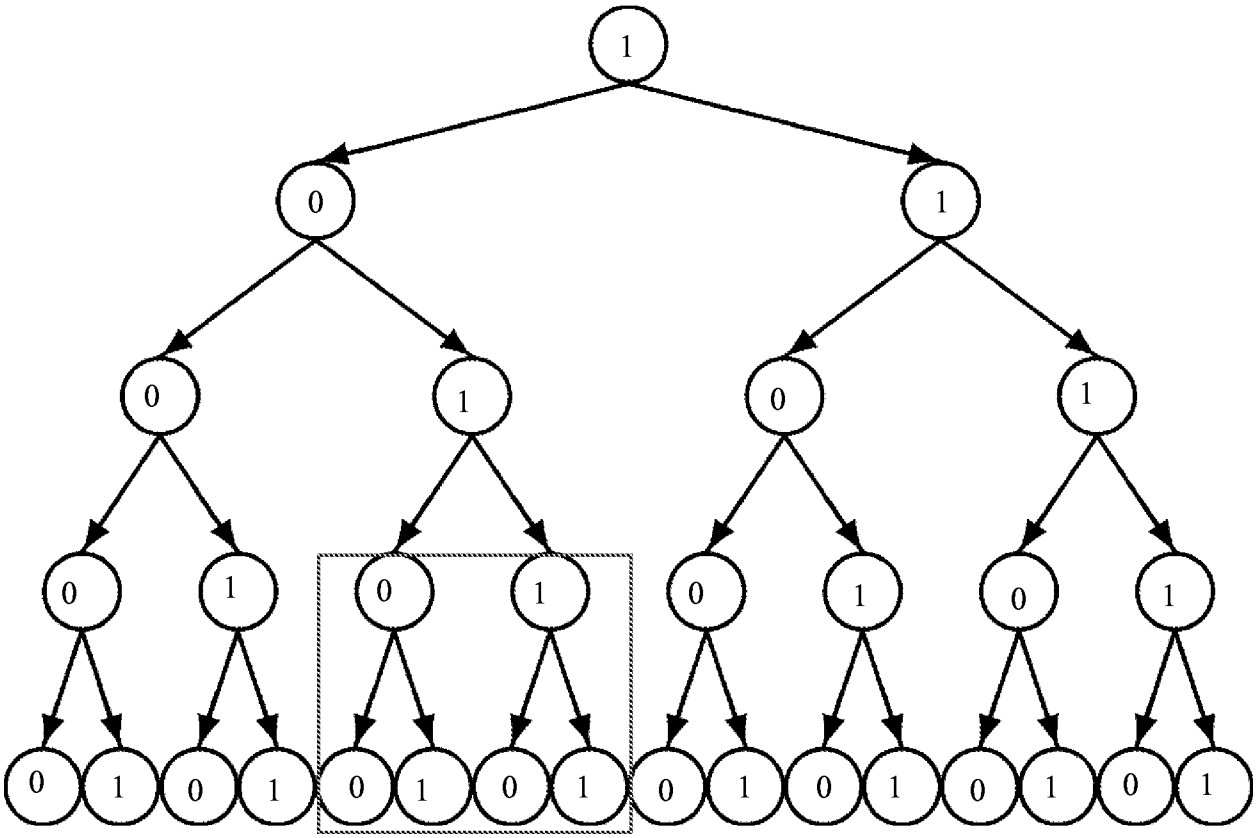

Figure 5

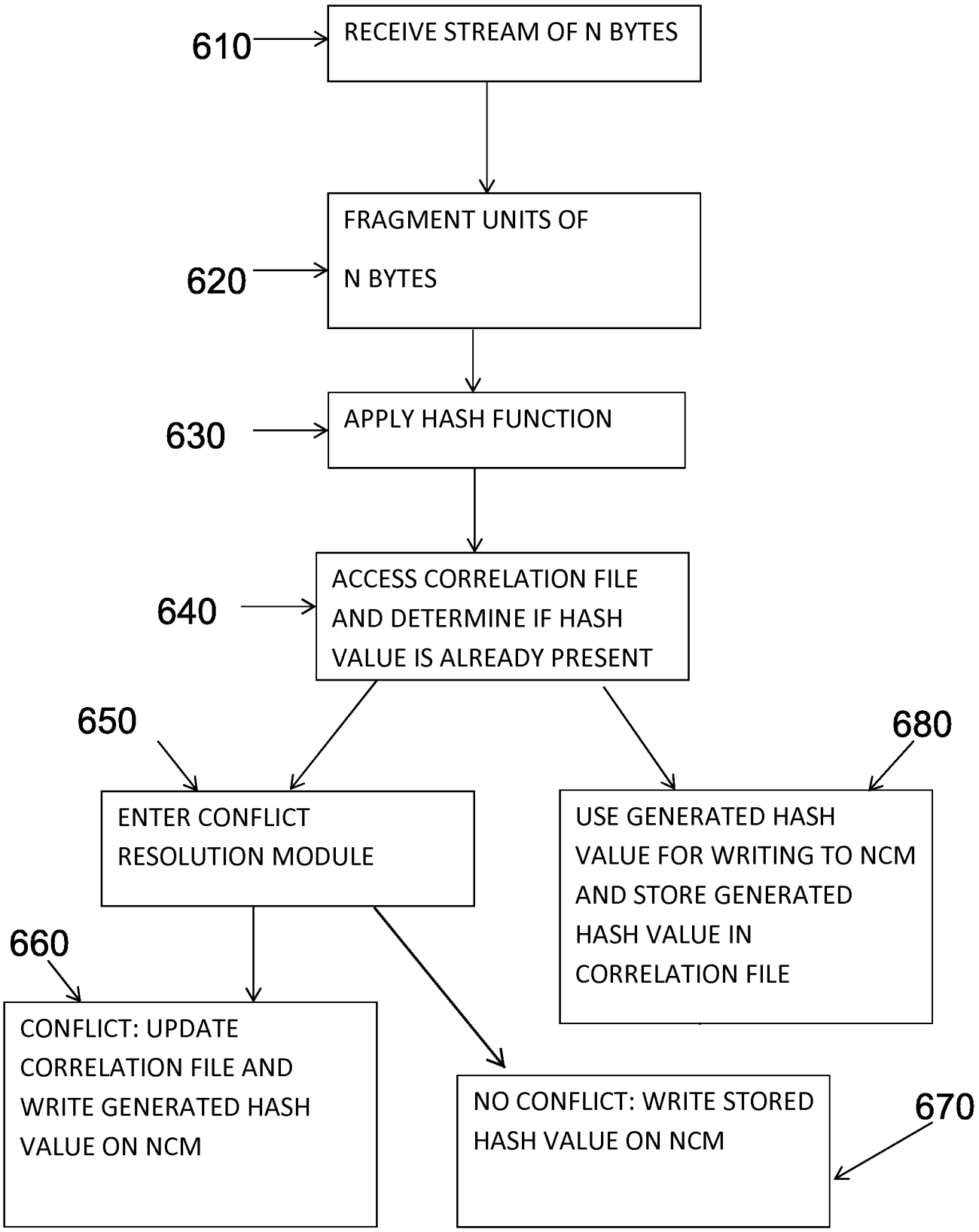
610
RECEIVE STREAM OF N BYTES
620
FRAGMENT UNITS OF N BYTES
630
APPLY HASH FUNCTION
640
ACCESS CORRELATION FILE AND DETERMINE IF HASH VALUE IS ALREADY PRESENT
650
ENTER CONFLICT RESOLUTION MODULE
660
CONFLICT: UPDATE CORRELATION FILE AND WRITE GENERATED HASH VALUE ON NCM
680
USE GENERATED HASH VALUE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

Figure 6