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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
7 July 2011 (07.07.2011)

PCT

(10) International Publication Number
WO 2011/082123 A1

(51) International Patent Classification:
G06F 7/00 (2006.01) *G06F 17/00* (2006.01)

(21) International Application Number:
PCT/US2010/062126

(22) International Filing Date:
27 December 2010 (27.12.2010)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
61/290,334 28 December 2009 (28.12.2009) US
61/315,392 18 March 2010 (18.03.2010) US
12/942,991 9 November 2010 (09.11.2010) US

(71) Applicant *(for all designated States except US)*: **RIVERBED TECHNOLOGY, INC.** [US/US]; Pierre Keeley, 199 Fremont St., San Francisco, California 94105 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **TALECK, Greg** [US/US]; Riverbed Technology, Inc., 199 Fremont St., San Francisco, California 94105 (US). **PARAB, Nitin** [IN/US]; Riverbed Technology, Inc., 199 Fremont St., San Francisco, California 94105 (US). **MACE, James, E.** [US/US]; Riverbed Technology, Inc., 199 Fremont St., San Francisco, California 94105 (US).

(74) Agent: **HOLLANDER, Jonathan**; Law Office of Jonathan Hollander PC, 660 4th Street #198, San Francisco, California 94107 (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, 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, IS, JP, KE, KG, KM, 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, PE, PG, PH, PL, PT, RO, RS, RU, 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, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, 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, ML, MR, NE, SN, TD, TG).

Published:

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

(54) **Title**: WAN-OPTIMIZED LOCAL AND CLOUD SPANNING DEDUPLICATED STORAGE SYSTEM

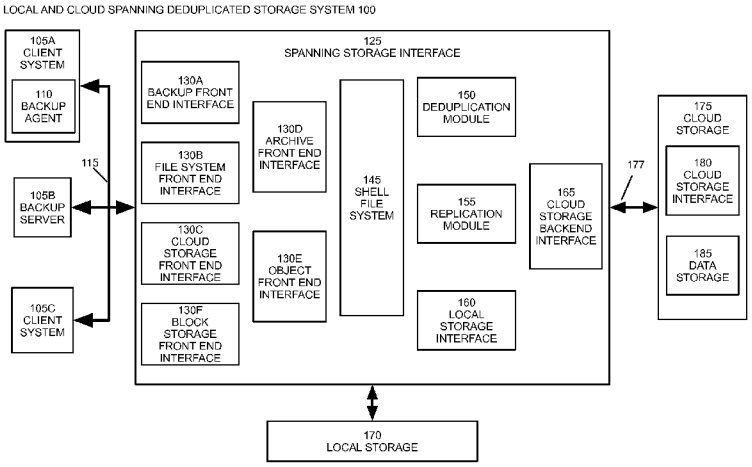

FIG. 1

(57) **Abstract**: A spanning storage interface facilitates the use of cloud storage services by storage clients. The spanning storage interface presents one or more data interfaces to storage clients at a network location, such as file, object, data backup, archival, and storage block based interfaces. The data interfaces allows storage clients to store and retrieve data using non-cloud based protocols. The spanning storage interface may perform data deduplication on data received from storage clients. The spanning storage interface may transfer the deduplicated version of the data to the cloud storage service. The spanning storage interface may include local storage for storing a copy or all or a portion of the data from storage clients. The local storage may be used as a local cache of frequently accessed data, which may be stored data in its deduplicated form.

WO 2011/082123 A1

WAN-OPTIMIZED LOCAL AND CLOUD SPANNING DEDUPLICATED STORAGE SYSTEM

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims priority to U.S. Provisional Patent Application No. 61/315,392, filed March 18, 2010 and entitled "WAN-OPTIMIZED LOCAL AND CLOUD SPANNING DEDUPLICATED STORAGE SYSTEM" and to U.S. Provisional Patent Application No. 61/290,334, filed December 28, 2009 and entitled "DEDUPLICATED OBJECT STORAGE SYSTEM AND APPLICATIONS," which are incorporated by reference herein for all purposes.

BACKGROUND OF THE INVENTION

[0002] The present invention relates generally to data storage systems, and systems and methods to improve storage efficiency, compactness, performance, reliability, and compatibility. In general, data storage systems receive and store all or portions of arbitrary sets or stream of data. Data storage systems also retrieve all or portions of arbitrary sets or streams of data. A data storage system provides data storage and retrieval to one or more storage clients, such as user and server computers. Stored data may be referenced by unique identifiers and/or addresses or indices. In some implementations, the data storage system uses a file system to organize data sets into files. Files may be identified and accessed by a file system path, which may include a file name and one or more hierarchical file system directories.

[0003] Many data storage systems are tasked with handling enormous amounts of data. Additionally, data storage systems often provide data access to large numbers of simultaneous users and software applications. Users and software applications may access the file system via local communications connections, such as a high-speed data bus within a single computer; local area network connections, such as an Ethernet networking or storage area network (SAN) connection; and wide area network connections, such as the Internet, cellular data networks, and other low-bandwidth, high-latency data communications networks.

[0004] Cloud storage services are one type of data storage available via a wide-area network. Cloud storage services provide storage to users in the form of a virtualized storage device available via the Internet. In general, users access cloud storage to store and retrieve data using web services protocols, such as REST or SOAP. Cloud storage service providers manage the operation and maintenance of the physical data storage devices. Users of cloud storage can avoid the initial and ongoing costs associated with buying and maintaining storage devices. Cloud storage services typically charge users for consumption of storage resources, such as storage space and/or transfer bandwidth, on a marginal or subscription basis, with little or no upfront costs. In addition to the cost and administrative advantages, cloud storage services often provide dynamically scalable capacity to meet its users changing needs.

[0005] The term "data deduplication" refers to some process of eliminating redundant data for the purposes of storage or communication. Data deduplicating storage typically compares incoming data with the data already stored, and only stores the portions of the incoming data that do not match data already stored in the data storage system. Data deduplicating storage maintains metadata to determine when portions of data are no longer in use by any files or other data entities.

[0006] The CPU and I/O requirements for supporting an extremely large data deduplicating storage are significant, and are difficult to satisfy through vertical scaling of a single device. As a result, prior spanning storage interface may impose severe throughput, latency, and other performance penalties on storage clients. Additionally, performance considerations limit the amount and types of optimizations and compression applied by prior spanning storage interfaces.

[0007] Additionally, prior spanning storage interfaces have difficulty operating with cloud storage systems. Data deduplication often requires frequent comparisons of incoming data with previously-stored data to identify redundant data. However, cloud data storage is accessible only via a wide-area network, such as the Internet, with significant latency and bandwidth limitations as compared with local-area and storage-area networks. Therefore, prior spanning storage interfaces have poor performance when used with cloud storage systems.

BRIEF DESCRIPTION OF THE DRAWINGS

[0008] The invention will be described with reference to the drawings, in which:

Figure 1 illustrates an example of spanning storage interface according to an embodiment of the invention;

Figure 2 illustrates example data structures used by a spanning storage interface according to an embodiment of the invention;

Figure 3A-3B illustrates a method of converting a data stream into deduplicated data according to an embodiment of the invention;

Figure 4 illustrates a method of retrieving an original data stream from deduplicated data according to an embodiment of the invention;

Figure 5 illustrates a method of deleting a data stream from a spanning storage interface according to an embodiment of the invention;

Figure 6 illustrates a computer system suitable for implementing embodiments of the invention; and

Figure 7 illustrates an example disaster recovery application of a spanning storage interface according to an embodiment of the invention.

SUMMARY

[0009] Embodiments of the invention include a spanning storage interface adapted to facilitate the use of cloud storage services by storage clients. A spanning storage interface presents one or more data interfaces to storage clients at a network location. These data interfaces may include file, object, data backup, archival, and storage block based interfaces. Each of these data interfaces allows storage clients to store and retrieve data using non-cloud based protocols. This allows storage clients to store and retrieve data in the cloud storage service using their native or built-in functions, rather than having to be rewritten and/or reconfigured to operate with a cloud storage service.

[0010] To improve performance of the spanning storage interface, an embodiment of the invention performs data deduplication on data received from storage clients. Once the received data has been deduplicated, the spanning storage interface may transfer the deduplicated version of the data to the cloud storage service. By transferring data in

deduplicated form to and from the cloud storage service, these embodiments of the invention improve storage performance by reducing the time and network bandwidth required to access data, as well as reducing total amount of storage required. If a storage client wishes to access data previously stored in the cloud storage service, the spanning storage interface retrieves the corresponding deduplicated data and reconstructs the original data.

[0011] In an embodiment, the spanning storage interface may include local storage for storing a copy or all or a portion of the data from storage clients. The local storage may be used as a local cache of frequently accessed data. In a further embodiment, the local cache stores data in its deduplicated form.

[0012] The spanning storage interface may operated with multiple cloud storage services to provide storage clients with a range of storage options. In a further embodiment, the spanning storage interface may send different portions of the received data to different cloud storage services based on user specified attributes or criteria, such as all or a portion of the file path associated with the received data.

[0013] In an embodiment, two or more spanning storage interfaces may be used in a disaster recovery application. Disaster recovery application may be used to provide redundant data access to storage clients in the event that the storage clients and/or cloud spanning storage interface at a first network location are disabled, destroyed, or otherwise inaccessible or inoperable. A disaster recovery application includes at least first and second spanning storage interfaces at first and second network locations. The second spanning storage interface is provided for at least disaster recovery operations. The second spanning storage interface includes second local storage for improving data access performance. A copy of the local cache of the first spanning storage interface is transferred to the second local storage while the first network location is operating. In the event of a disaster affecting the first network location, the second spanning storage interface can provide data access to the first network location's data with the improved performance benefit using the copy of local cache in the second local storage.

[0014] Embodiments of the disaster recovery application may use the second network location as a dedicated disaster recovery network location. Alternatively, the second network location may also optionally be used with one or more of its own local storage clients. In this further example, the second spanning storage interface performs data deduplication and facilitates cloud storage for data from storage clients at the second network location in

addition to acting as a disaster recovery system for the first network location. In yet a further embodiment, the first spanning storage interface may act as a disaster recovery system for the second spanning storage interface, just as the second spanning storage interface may act as a disaster recovery system for the first spanning storage interface. This pairing of spanning storage interfaces for disaster recovery may be extended to three or more network locations.

DETAILED DESCRIPTION

[0015] Figure 1 illustrates an example of spanning storage interface 100 according to an embodiment of the invention. An example installation of the spanning storage interface 100 includes one or more client systems 105, which may include client computers, server computers, and standalone network devices. Client systems 105 are connected with a spanning storage interface 125 via a local-area network and/or a storage area network 115. Cloud storage 175 is connected with the spanning storage interface 125 by at least a wide-area network 177 and optionally an additional local area network. Cloud storage 175 includes a cloud storage interface 180 for communicating with the spanning storage interface 125 via wide-area network 177 and at least one physical data storage device 185 for storing data.

[0016] Embodiments of spanning storage interface 100 may support a variety of different storage applications using cloud data storage, including general data storage, data backup, disaster recovery, and deduplicated cloud data storage. In the case of general data storage applications, a client, such as client 105c, may communicate with the spanning storage interface 125 via a file system protocol, such as CIFS or NTFS, or a block-based storage protocol, such as iSCSI or IFCP. Data backup and disaster recovery applications may also use these protocols or specific backup and recovery protocols, such as VTL or OST. For backup applications, a client system 105a may include a backup agent 110 for initiating data backups. The backup agent 110 may communicate directly with the spanning storage interface 125 or a backup server 105b, which in spanning storage interface 100 is equivalent to a client. For cloud storage applications, a client 103c may communicate with the spanning storage interface 125 via a web services protocol, such as SOAP or REST. The web services protocol may present a virtualized storage device to client 103c. The web services protocol used by clients 105 to communicate with the spanning storage interface 125 may be the same

or different than the protocol used by the spanning storage interface 125 to communicate with the cloud storage 175.

[0017] Embodiments of the spanning storage interface 100 may optimize data access to cloud storage 175 in a number of different ways. An embodiment of the spanning storage interface 125 may present clients 105 with a file system, backup device, storage array, or other data storage interface, while transparently storing and retrieving data using the cloud storage 175 via the wide-area network 177. In a further embodiment, the spanning storage interface 125 may perform data deduplication on data received from clients 105, thereby reducing the amount of storage capacity required in cloud storage 175. Additionally, because the bandwidth of the wide-area network is often limited, data deduplication by the spanning storage interface 125 increases the data access performance, as perceived by the clients 125. In still a further embodiment, the spanning storage interface 125 may locally cache a portion of the clients' data using local storage 170. The locally cached data may be accessed rapidly, further improving the perceived data access performance. As described in detail below, the spanning storage interface 125 may use a variety of different criteria for selecting the portion of the clients' data to cache locally and may locally cache data in a deduplicated form to reduce the required capacity of local storage 175.

[0018] An embodiment of spanning storage interface 125 includes one or more front end interfaces 130 for communicating with one or more client systems 105. Examples of front end interfaces 130 include a backup front end interface 130a, a file system front end interface 130b, a cloud storage front end interface 130c, a file archival front end interface 130d, and a object front end interface 130e. An example backup front end interface 130a enables backup applications, such as a backup agent 110 and/or a backup server 105b, to store and retrieve data to and from the cloud storage 175 using data backup and recovery protocols such as VTL or OST. In this example, the backup front end interface 130a allows the spanning storage interface 125 and cloud storage 175 to appear to clients 105 as a backup storage device.

[0019] An example file system front end interface 130b enables clients 105 to store and retrieve data to and from the cloud data storage 175 using a file system protocol, such as CIFS or NTFS, or a block-based storage protocol, such as iSCSI or IFCP. In this example, the file system front end interface 130b allows the spanning storage interface 125 and cloud

storage 175 to appear to clients 105 as one or more storage devices, such as a CIFS or NTFS storage volume or a iSCSI or FibreChannel logical unit number (LUN).

[0020] An example cloud storage front end interface 130c enables clients 105 to store and retrieve data to and from the cloud data storage 175 using a cloud storage protocol or API. Typically, cloud storage protocols or APIs are implemented using a web services protocol, such as SOAP or REST. In this example, the cloud storage front end interface 130c allows the spanning storage interface 125 and cloud storage 175 to appear to clients 105 as one or more cloud storage services. By using spanning storage interface 125 to provide a cloud storage interface to clients 105, rather than letting clients 105 communicate directly with the cloud storage 175, the spanning storage interface 125 may perform data deduplication, local caching, and/or translation between different cloud storage protocols.

[0021] An example file archival front end interface 130d enables clients 105 to store and retrieve file archives. Clients 105 may use the spanning storage interface 125 and the cloud storage 175 to store and retrieve files or other data in one or more archive files. The file archival front end interface 130d allows clients 105 to store archive files using cloud storage 175 using archive file interfaces, rather than a cloud storage interface. Additionally, the spanning storage interface 125 may perform data deduplication and local caching of the file archives.

[0022] An example object front end interface 130e enables clients to store and retrieve data in any arbitrary format, such as object formats and blobs or binary large objects. The object front end interface 130e allows clients 105 to store data in arbitrary formats, such as object formats or blobs, using cloud storage 175 using object protocols, such as object serialization or blob storage protocols, rather than a cloud storage protocol. Additionally, the spanning storage interface 125 may perform data deduplication and local caching of the object or blob data.

[0023] An example block storage protocol front end interface 130f enables clients to store and retrieve data using block-based storage protocols, such as iSCSI. In an embodiment, the block storage protocol front end interface 130f appears to clients 105 as one or more logical storage volumes, such as iSCSI LUNs.

[0024] In an embodiment, spanning storage interface 125 also includes one or more shell file systems 145. Shell file system 145 includes a representation of the entities, such as files, directories, objects, blobs, and file archives, stored by clients 125 via the front end interfaces

130. In an embodiment, the shell file system 145 includes entities stored by the clients 125 in a shell form. In this embodiment, each entity, such as a file or other entity, is a represented by a "shell" entity that does not include the data contents of the original entity. For example, a shell file in the shell file system 145 includes the same name, file path, and file metadata as the original file. However, the shell file does not include the actual file data, which is stored in the cloud storage 175. It should be noted that although the size of the shell file is less than the size of the actual stored file (in either its original or deduplicated format, an embodiment of the shell file system 145 sets the file size metadata attribute of the shell file to the size of the original file. In a further embodiment, each entity in the shell file system 145, such as a file, directory, object, blob, or file archive, may include additional metadata for use by the spanning storage interface 125 to access the corresponding data from the cloud storage 175.

[0025] In an embodiment, storage blocks provided to the spanning storage interface through the block storage protocol front end interface 130f may bypass the shell file system 145. In this embodiment, data received by the spanning storage interface in the form of storage blocks are grouped together, for example in groups of fixed size and in order of receipt. Data deduplication is then applied to each group of storage blocks and the resulting deduplicated data is transferred to the cloud storage service. In this embodiment, the spanning storage interface 125 maintains a table or other data structure that associates storage block addresses or identifiers with corresponding deduplicated storage data, so that the spanning storage interface 125 can retrieve and reconstruct the appropriate data when a storage client requests access to a previously stored storage block.

[0026] An embodiment of the spanning storage interface 125 includes a deduplication module 150 for deduplicating data received from clients 105. Deduplication module 150 analyzes data from clients 105 and compares incoming data with previously stored data to eliminate redundant data for the purposes of storage or communication. Data deduplication reduces the amount of storage capacity used by cloud storage 175 to store clients' data. Also, because wide-area network 177 typically has bandwidth limitations, the reduction of data size due to data deduplication also reduces the amount of time required to transfer data between clients 105 and the cloud storage 175. Additionally, deduplication module 150 retrieves deduplicated data from the cloud storage 175 and converts it back to its original form for use by clients 105.

[0027] In an embodiment, deduplication module 150 performs data deduplication on incoming data and temporarily stores this deduplicated data locally, such as on local storage 170. Local storage 170 may be a physical storage device connected with or integrated within the spanning storage interface 125. Local storage 170 is accessed from spanning storage interface 125 by a local storage interface 160, such as an internal or external data storage interface, or via a local-area network.

[0028] In an embodiment, the cloud storage 175 includes a complete and authoritative version of the clients' data. In a further embodiment, the spanning storage interface 125 may maintain local copies of some or all of the clients' data for the purpose of caching. In this embodiment, the spanning storage interface 125 uses the local storage 170 to cache client data. The spanning storage interface 125 may cache data in its deduplicated format to reduce local storage requirements or increase the effective cache size. In this embodiment, the spanning storage interface 125 may use a variety of criteria for selecting portions of the deduplicated client data for caching. For example, if the spanning storage interface 125 is used for general file storage or as a cloud storage interface, the spanning storage interface may select a specific amount or percentage of the client data for local caching. In another example, the data selected for local caching may be based on usage patterns of client data, such as frequently or recently used data. Caching criteria may be based on elapsed time and/or the type of data. In another example, the spanning storage interface 125 may maintain locally cached copies of the most recent data backups from clients, such as the most recent full backup and the previous week's incremental backups.

[0029] In an embodiment, replication module 155 transfers locally stored deduplicated data from the spanning storage interface 125 to the cloud storage 175. Embodiments of the deduplication module and the replication module 155 may operate in parallel and/or asynchronously, so that the bandwidth limitations of wide-area network 177 do not interfere with the throughput of the deduplication module 150. The operation of embodiments of deduplication module 150 and replication module 155 are described in detail below.

[0030] An embodiment of spanning storage interface 125 includes a cloud storage backend interface 165 for communicating data between the spanning storage interface 125 and the cloud storage 175. Embodiments of the cloud storage backend interface 165 may use cloud storage protocols or API and/or web services protocols, such as SOAP or REST, to store and retrieve data from the cloud storage 175. In an embodiment, the replication module transfers

deduplicated data from local storage 170 to cloud storage 175 using the cloud storage backend interface 165. In an embodiment, the deduplication module retrieves deduplicated data from the cloud storage 175 using the cloud storage backend interface 165.

[0031] An embodiment of the spanning storage interface 125 may be configured to operate with multiple cloud storage services. In an embodiment, the spanning storage interface 125 may transfer all or portions of the dededuplicated data to two or more cloud storage services. In another embodiment, the spanning storage interface 125 may transfer different portions of the deduplicated data to different cloud storage services, such as transferring a first portion of the deduplicated storage data to a first cloud storage service, a second portion of the deduplicated storage data to a second cloud storage service, and so forth.

[0032] Different cloud storage services may have different advantages and/or disadvantages, such as cost, bandwidth, reliability, and replication policies. In this embodiment, a system administrator or other user may identify the different portions of data and designate the cloud storage service to be used to store deduplicated versions of these portions of the data, thereby tailoring the usage of different cloud storage services to data storage needs. The user may identify different portions of data and associated cloud storage services based on file or object name, file or object type, file directory or path, contents of the data, and/or any other criteria or attribute of the data, storage client, cloud storage service, or the spanning storage interface 125.

[0033] In yet a further embodiment, system administrators or other users may specify quotas for cloud storage access based on the total amount of data received from storage clients or the amount of deduplicated data transferred to the one or more cloud storage services. In this embodiment, if a data transfer exceeds or is anticipated to exceed a specified quota, the spanning storage interface 125 may abandon the storage operation and return an error message or other notification to the storage client. Embodiments may allow users to specify quotas for each storage client, a group of two or more storage clients, all of the storage clients at a network location or based on criteria or attributes associated with the cloud storage service, spanning storage interface, and/or data, such as file or object names, file or object types, file directories or paths, contents of the data.

[0034] In an embodiment, the spanning storage interface 125 performs data deduplication by segmenting an incoming data stream to aid data compression. For example, segmentation may be designed to produce many identical segments when the data stream includes

redundant data. Multiple instances of redundant data may be represented by referencing a single copy of this data.

[0035] Additionally, a data stream may be segmented based on data types to aid data compression, such that different data types are in different segments. Different data compression techniques may then be applied to each segment. Data compression may also determine the length of data segments. For example, data compression may be applied to a data stream until segment boundary is reached or the segment including the compressed data reaches a predetermined size, such as 4KB. The size threshold for compressed data segments may be based on optimizing disk or data storage device access.

[0036] Regardless of the technique used to segment data in the data stream, the result is a segmented data stream having its data represented as segments. In some embodiments of the invention, data segmentation occurs in memory and the segmented data stream is not written back to data storage in this form. Each segment is associated with a label. Labels are smaller in size than the segments they represent. The segmented data stream is then replaced with deduplicated data in the form of a label map and segment storage. Label map includes a sequence of labels corresponding with the sequence of data segments identified in the segmented data stream. Segment storage includes copies of the segment labels and corresponding segment data. Using the label map and the data segment storage, a storage system can reconstruct the original data stream by matching in sequence each label in a label map with its corresponding segment data from the data segment storage. In an embodiment, the deduplication module 150 and/or one or more other modules of the spanning storage interface 125 reconstruct all or a portion of the original data stream in response to a data access request from a storage client.

[0037] Embodiments of the invention attempt (but do not always succeed) in assigning a single label to each unique data segment. Because the segmentation of the data stream produces many identical segments when the data stream includes redundant data, these embodiments allow a single label and one copy of the corresponding segment data to represent many instances of this segment data at multiple locations in the data stream. For example, a label map may include multiple instances of a given label at different locations. Each instance of this label represents an instance of the corresponding segment data. Because the label is smaller than the corresponding segment data, representing redundant

segment data using multiple instances of the same label results in a substantial size reduction of the data stream.

[0038] Figures 2, 3A-3B, 4, and 5 illustrate the operation of the deduplication module 150 and the replication module 155 according to an embodiment of the invention. Figure 2 illustrates example data structures 200 used by a spanning storage interface according to an embodiment of the invention. An embodiment of spanning storage interface 200 includes both memory 205, which has high performance but relatively low capacity, and disk storage 210, which has high capacity but relatively low performance.

[0039] Memory 205 includes a slab cache data structure 215. The slab cache 215 is adapted to store a set of labels 220 and a corresponding set of data segments 225. In typical applications, the sets of labels 220 and data segments 225 stored in the slab cache 215 represent only a small fraction of the total number of data segments and labels used to represent stored data. A complete set of the labels and data segments is stored in disk storage 210.

[0040] An embodiment of the slab cache 215 also includes segment metadata 230, which specifies characteristics of the data segments 225. In an embodiment, the segment metadata 230 includes the lengths of the data segments 225; hashes or other characterizations of the contents of the data segments 225; and/or anchor indicators, which indicate whether a particular data segment has been designated as a representative example of the contents of a data segment slab file, as discussed in detail below.

[0041] An embodiment of the slab cache 215 also includes data segment reference count values. The spanning storage interface 200 recognizes that some data segments are used in multiple places in one or more data streams. For at least some of the data segments, an embodiment of the spanning storage interface 200 maintains counts, referred to as reference counts, of the number of times these data segments are used. As discussed in detail below, if a data stream includes a data segment previously defined, an embodiment of the spanning storage interface 200 may increment the reference count value associated with this data segment. Conversely, if a data stream is deleted from the spanning storage interface 200, an embodiment of the spanning storage interface 200 may decrement the reference count values associated with the data segments included in the deleted data stream. If the reference count value of a data segment drops to zero, the data segment and label may be deleted and its storage space reallocated.

[0042] In addition to the slab cache 215, an embodiment of the spanning storage interface 200 includes a reverse map cache 240. In an embodiment, the reverse map cache 240 maps the contents of a data segment to a label, for the labels stored in the slab cache 215. In an embodiment, a hashing or other data characterization technique is applied to segment data. The resulting value is used as an index in the reverse map cache 240 to identify an associated label in the slab cache 215. If the hash or other value derived from the segment data matches an entry in the reverse map cache 240, then this data segment has been previously defined and is stored in the slab cache 215. If the hash or other value derived from the segment data does not match any entry in the reverse map cache 240, then this data segment is not currently stored in the slab cache 215. Because the slab cache 215 only includes a portion of the total number of labels used to represent data segments, a data segment that does not match a reverse map cache entry may either have not been previously defined or may have been previously defined but not loaded into the slab cache 215.

[0043] In an embodiment, memory 205 of the spanning storage interface 200 also includes an anchor cache 245. Anchor cache 245 is similar to reverse map cache 240; however, anchor cache 245 matches the contents of data segments with representative data segments in data segment slab files stored on disk storage 210. A complete set of data segments are stored in one or more data segment slab files in disk storage 210. In an embodiment, one or more representative data segments from each data segment slab file are selected by the spanning storage interface 200. The spanning storage interface 200 determines hash or other data characterization values for these selected representative data segments and stores these values along with data identifying the file or disk storage location including this data segment in the anchor cache 245. In an embodiment, the data identifying the file or disk storage location of a representative data segment may be its associated label. The spanning storage interface 200 uses the anchor cache 245 to determine if a data segment from a data stream matches a data segment from another data stream previously stored in disk storage but not currently stored in the slab cache.

[0044] In an embodiment, potential representative data segments are identified during segmentation of a data stream. As discussed in detail below, when one or more potential representative data segments are later stored in disk storage 210, for example in a data segment slab file, an embodiment of the spanning storage interface 200 selects one or more of these potential representative data segments for inclusion in the anchor cache.

[0045] A variety of criteria and types of analysis may be used alone or together in various combinations to identify representative data segments in data streams and/or in data segment slab files stored in disk storage 210. For example, the spanning storage interface 200 selects the first unique data segment in a data stream as a representative data segment. In another example, the spanning storage interface 200 uses the content of the data stream to identify potential representative data segments. In still another example, the spanning storage interface 200 uses criteria based on metadata such as a file type, data type, or other attributes provided with a data stream to identify potential representative data segments. For example, data segments including specific sequences of data and/or located at specific locations within a data stream of a given type may be designated as representative data segments based on criteria or heuristics used by the spanning storage interface 200. In a further example, a random selection of unique segments in a data stream or a data segment slab file may be designated as representative data segments. In yet a further example, representative data segments may be selected at specific locations of data segment slab files, such as the middle data segment in a slab file.

[0046] Disk storage 210 stores a complete set of data segments and associated labels used to represent all of the data streams stored by spanning storage interface 200. In an embodiment, disk storage 210 may be comprised of multiple physical and/or logical storage devices. In a further embodiment, disk storage 210 may be implementing using a storage area network.

[0047] Disk storage 210 includes one or more data segment slab files 250. Each data segment slab file 250 includes a segment index 255 and a set of data segments 265. The segment index 255 specifies the location of each data segment within the data segment slab file. Data segment slab file 250 also includes segment metadata 260, similar to the segment metadata 230 discussed above. In an embodiment, segment metadata 260 in the data segment slab file 250 is a subset of the segment metadata in the slab cache 215 to improve compression performance. In this embodiment, the spanning storage interface 200 may recompute or recreate the remaining metadata attribute values for data segments upon transferring data segments into the slab cache 215.

[0048] Additionally, data segment slab file 250 may include data segment reference count values 270 for some or all of the data segments 265. In an embodiment, slab file 250 may

include slab file metadata 275, such as a list of data segments to be deleted from the slab file 250.

[0049] Disk storage 210 includes one or more label map container files 280. Each label map container file 280 includes one or more label maps 290. Each of the label maps 290 corresponds with all or a portion of a deduplicated data stream stored by the spanning storage interface 200. Each of the label maps 290 includes a sequence of one or more labels corresponding with the sequence of data segments in all or a portion of a deduplicated data stream. In an embodiment, each label map also includes a label map table of contents providing the offset or relative position of sections of the label map sequence with respect to the original data stream. In one implementation, the label maps are compressed in sections, and the label map table of contents provides offsets or relative locations of sections of the label map sequence relative to the uncompressed data stream. The label map table of contents may be used to allow random or non-sequential access to a deduplicated data stream.

[0050] Additionally, label map container file 280 may include label map container index 285 that specifies the location of each label map within the label map container file.

[0051] In an embodiment, label names are used not only identify data segments, but also to locate data segments and their containing data segment slab files. For example, labels may be assigned to data segments during segmentation. Each label name may include a prefix portion and a suffix portion. The prefix portion of the label name may correspond with the file system path and/or file name of the data segment slab file used to store its associated segment. All of the data segments associated with the same label prefix may be stored in the same data segment slab file. The suffix portion of the label name may be used to specify the location of the data segment within its data segment slab file. The suffix portion of the label name may be used directly as an index or location value of its data segment or indirectly in conjunction with segment index data in the slab file. In this implementation, the complete label name associated with a data segment does not need to be stored in the slab file. Instead, the label name is represented implicitly by the storage location of the slab file and the data segment within the slab file. In a further embodiment, label names are assigned sequentially in one or more namespaces or sequences to facilitate this usage.

[0052] An embodiment similarly uses data stream identifiers to not only identify deduplicated data streams but to locate label maps and their containing label map containers. For example, a data stream identifier is assigned to a data stream during deduplication. Each

data stream identifier name may include a prefix portion and a suffix portion. The prefix portion of the data stream identifier may correspond with the file system path and/or file name of the label map container used to store the label map representing the data stream. The suffix portion of the data stream identifier may be used to directly or indirectly specify the location of the label map within its label map container file. In a further embodiment, data stream identifiers are assigned sequentially in one or more namespaces or sequences to facilitate this usage.

[0053] Embodiments of the spanning storage interface 200 may specify the sizes, location, alignment, and optionally padding of data in data segment slab files 250 and label map container files 280 to optimize the performance of disk storage 210. For example, segment reference counts are frequently updated, so these may be located at the end of the data segment slab file 250 to improve update performance. In another example, data segments may be sized and aligned according to the sizes and boundaries of clusters or blocks in the disk storage 210 to improve access performance and reduce wasted storage space.

[0054] Figure 3A illustrates a method 300 of converting a data stream into deduplicated data according to an embodiment of the invention. An embodiment of method 300 may be executed at least in part by a deduplication module including in a spanning storage interface. Step 305 receives all or a portion of a data stream. The data stream may be any type or format of data, including files and objects. In an embodiment, a deduplicating storage interface client provides the data stream to the spanning storage interface.

[0055] Step 310 uses a segmentation technique to generate one or more data segments from the data stream or portion thereof received by step 305.

[0056] Step 315 determines if any of the generated data segments are referenced by the anchor cache of the spanning storage interface. In an embodiment, step 315 compares a hash or other characterization of the contents of each of the data segments with entries of the anchor cache. If the hash of the data segment matches an entry of the anchor cache, then the data segment is referenced by the anchor cache. In a further embodiment, if the hash of a data segment matches an entry of the anchor cache, step 315 then compares the segment length and/or the contents of the data segment with the corresponding data segment stored in a slab file to verify that the data segment from the data stream and the previously generated instance of the data segment are identical.

[0057] In an embodiment, a copy of only a portion of the data segments used for data deduplication are stored locally. The full and authoritative set of data segments is stored in one or more slab files stored in the cloud storage. Because the cloud storage is accessed via a wide-area network, there are often substantial bandwidth and latency restrictions on accessing slab files from cloud storage. In an embodiment, if a data segment from the data stream matches an entry from the anchor cache, step 315 selects the slab file associated with this anchor cache entry for processing by method 355, as discussed below. In an embodiment, method 355 may retrieve one or more slab files selected by step 315 from the cloud storage in parallel and/or asynchronously with the execution of method 300.

[0058] Step 325 determines if any of the data segments generated in step 310 match a data segment referenced by the reverse map in memory. In an embodiment, step 325 is similar to step 315. Step 325 compares a hash or other characterization of the contents of the data segment with entries of the reverse map. In a further embodiment, if the hash of the data segment matches an entry of the reverse map (and/or previously matched an entry of the anchor cache), step 325 also compares the segment length and/or the contents of the data segment with the corresponding data segment stored in the slab cache to verify that the data segment from the data stream and the cached data segment are identical.

[0059] For each of the data segments from the data stream that match previously generated data segments in the slab cache, step 325 associates these data segments from the data stream with the labels assigned to their counterparts in the slab cache. Step 330 increments the reference counts for these labels based on the number of instances of their associated data segment in the data stream. For example, step 330 increments the reference count by one for each instance of the generated data segment in the data stream.

[0060] Conversely, if one or more the data segments from the data stream are not referenced by the reverse map, then step 335 assigns new labels to these newly generated data segments. These new labels assigned by step 335 are referred to as provisional labels. As discussed below in method 355, method 350 may replace provisional labels assigned by step 335 with previously generated labels corresponding with identical data segments in slab files retrieved from the cloud storage. Step 335 then adds the new data segments and their assigned provisional labels to the slab cache in memory. For each newly added data segment and provisional label, step 335 generates segment metadata adds it to the slab cache. Step 335 also initializes a reference count in the slab cache for each of the newly added data

segments, setting each newly added provisional label's reference count to correspond with the number of currently known instances of the corresponding data segment in the data stream. For example, step 335 may initialize a reference count associated with a new provisional label and data segment to one, if the data segment occurs only once in the data stream or portion thereof received by step 305. In another example, step 335 may initialize the reference count associated with a new provisional label and data segment to a number greater than one of this data segment is used multiple times in the received portion of the data stream. Step 335 also adds the new provisional labels and hashes or other data characterizations of the new data segment to the reverse map in memory.

[0061] Following steps 330 or 335, the slab cache in memory has been updated with all of the data segments generated by step 310 from the received portion of the data stream, either by incrementing the reference counts of previously generated labels or adding new provisional labels and associated data segments to the slab cache. In a further embodiment, the updates to the slab cache in memory are stored in local disk storage for further processing and eventual copying to the cloud storage. In an embodiment, method 300 stores a copy of any new data segments and associated metadata in local disk storage in one or more new slab files. Additionally, any changes to previously-generated data segment metadata, such as updates in reference counts, may be stored in local storage as well.

[0062] Step 340 adds the sequence of labels associated with the data segments generated by step 310 to a label map. The sequence of labels may include both previously generated labels and/or provisional labels, depending upon the contents of the current data stream and any previously processed data streams. Step 340 adds labels to the label map in the same sequence as their corresponding data segments are found in the data stream.

[0063] Decision block 345 determines if all of the data in the data stream has been processed by steps 310 to 340. If all of the data in the data stream has not been processed, method 300 returns to step 305 to receive another portion of the data stream and to generate and process additional data segments.

[0064] If all of the data stream has been processed, method 300 proceeds to step 350. Step 350 adds the completed label map to a label map container file in the local disk storage. Step 350 assigns the data stream and its corresponding label map a data stream identifier. In an embodiment, the data stream identifier specifies the identity and/or the location of the label map container file in the disk storage. Step 350 may store the data stream identifier in the

metadata of the corresponding file in the shell file system, such as in a reparse point in an NTFS file system or a extended attribute in an ext3 file system. Following step 350, the spanning storage interface 125 may delete the original data stream from memory or disk storage, as this data stream is now stored in deduplicated form by the spanning storage interface.

[0065] Figure 3B illustrates a method 350 for transferring deduplicated data from a spanning storage interface to cloud storage. An embodiment of method 350 may be executed by a replication module operating in parallel and/or asynchronously with a deduplication module. As described above, an embodiment of the spanning storage interface includes a local copy of only a portion of the data segments used for data deduplication. The full and authoritative set of data segments is stored in one or more slab files stored in the cloud storage. Thus, this embodiment of the spanning storage interface should copy any newly added data segments or updated segment metadata to the cloud storage as soon as possible, so that the cloud storage includes a complete and authoritative set of the data segments, associated labels, and label metadata, such as reference counts.

[0066] In an embodiment, a complete set of slab files, including at least all of the data segments used to store a deduplicated version of the client's data, is stored in cloud storage. If step 315 in method 300 matches a data segment to an entry of the anchor cache, then the data of this segment has been previously associated with a label. To optimize the data deduplication, this previously associated label should be associated with the new data segment. Additionally, because the anchor cache only includes a representative sample of data segments in the slab file, it is likely that other data segments in the slab file associated with the matching anchor cache entry may also match other recently received data segments. Thus, step 355 retrieves one or more slab files previously selected for retrieval by step 315 in method 300.

[0067] In an embodiment, step 355 retrieves one or more previously selected slab files from cloud storage via the wide-area network. In an embodiment, step 355 uses the label name of the matching anchor cache entry to identify and optionally locate the data segment slab file including the previously generated instance of the data segment. In a further embodiment, copies of some of the slab files may be stored locally. In this embodiment, step 355 determines if any of the selected slab files have local copies. Step 355 then retrieves any selected slab files that do not have copies stored locally from the cloud storage.

[0068] Step 360 processes the selected and retrieved slab files. In an embodiment, step 360 retrieves all of the data segments included in this data segment slab file from disk storage and adds them to the slab cache in memory. Step 360 also retrieves and/or regenerates the labels and segment metadata for these data segments and adds these to the slab cache. Step 360 retrieves the segment reference counts for these data segments from the data segment slab file and adds these to the slab cache in memory. Step 360 also updates the reverse map cache with the labels and hashes or other data characterizations of the retrieved data segments.

[0069] In method 300, data segments that do not match reverse map cache entries are assigned provisional labels. Data segments assigned provisional labels may include data segments matching an anchor cache entry as well as data segments that do not match any anchor cache entries. Step 365 identifies the provisional labels, if any, in one or more newly created label maps and/or label map container files.

[0070] Step 370 compares the data segments associated with the provisional labels with the updated reverse map cache. Step 370 ignores the reverse map cache entries associated with provisional labels in this comparison; instead, step 370 determines if any provisionally labeled data segments are identical to previously generated data segments. In an embodiment, step 370 compares a hash or other characterization of the contents of these provisionally labeled data segments with the non-provisional entries of the reverse map cache, which are cache entries that are not associated with provisional labels. In a further embodiment, if the hash of the data segment matches an entry of the reverse map, step 370 also compares the segment lengths and/or the contents of these provisionally labeled data segments with the corresponding non-provisional data segments stored in the slab cache to verify that the data segment from the data stream and the cached data segment are identical.

[0071] For data segments that do not match cached data segments in the slab cache, an embodiment of step 375 may change their associated labels to non-provisional status. An embodiment of step 375 may update the label map, label map container file, slab file, slab cache and/or reverse map cache with this change in status.

[0072] For data segments that do not match cached data segments in the slab cache, an embodiment of step 380 replaces the associated provisional labels in label maps with the matching non-provisional labels. As a result of step 380, a provisional label referencing a recently created data segment is replaced with a non-provisional label referencing a previously generated segment. However, no data is lost by step 380, because the contents of

the provisional data segment are identical to the previously generated non-provisional data segment, as determined by step 375.

[0073] Step 385 removes data segments and discards data segments associated with provisional labels that match previously generated non-provisional labels. In an embodiment, step 385 removes these provisional data segments from a slab file stored locally by a spanning storage interface. In a further embodiment, step 385 removes the provisional data segment and its associated provisional label from the slab cache and reverse map, respectively. These provisional labels and data segments may be removed because they are duplicative of previously generated data segments and labels. In an embodiment, step 385 updates the previously generated non-provisional label and data segment metadata. For example, if a provisional label is associated with a reference count, which indicates how many times this provisional label is used in one or more label maps; then step 385 may add this reference count to the reference count of the matching previously-generated non-provisional label. As a result, the reference count of this non-provisional label will be equal to the number of total number instances of this segment data, regardless of whether these instances were previously associated with the provisional label or the non-provisional label.

[0074] Step 390 identifies changes in the locally stored label map container files and slab files in comparison with their counterparts (if any) stored in the could storage. The changes identified by step 390 may include new label map container files and new slab files, as well as modified versions of label map container files and slab files previously stored in cloud storage. Step 395 transfers the new and changed label map container files and slab files to the cloud storage. In an embodiment, step 395 only communicates the changed or new data to the cloud storage.

[0075] Following step 395, the cloud storage includes a complete and authoritative version of the label maps and data segments. Thus, the slab files and label map container files stored in the cloud storage may be used to reconstruct any or all of the data previously stored by the clients via the spanning storage interface. In a further embodiment, step 395 may use atomic operations to update or add label map container and slab files in the cloud storage. In this embodiment, new and changed data is first uploaded to the cloud storage and then committed. If the transfer of data is interrupted before the commitment, for example due to a system or network failure, the previous versions of the label map container and slab files stored in the cloud storage will not be corrupted and may be used to restore client data at the same or a

different location. This allows the spanning storage interface to use cloud storage as a deduplicated disaster data recovery facility.

[0076] Following step 395, the spanning storage interface may delete some or all of the local copies of slab files and label map container files. In a further embodiment, the spanning storage interface may maintain local copies of some or all of the slab files and label map container files for the purpose of caching. The local caching may use the local storage associated with the spanning storage interface. The spanning storage interface may cache data in its deduplicated format to reduce local storage requirements or increase the effective cache size. In this embodiment, the spanning storage interface may use a variety of criteria for selecting portions of the deduplicated client data for caching. For example, if the spanning storage interface is used for general file storage or as a cloud storage interface, the spanning storage interface may select a specific amount or percentage of the client data for local caching. In another example, the data selected for local caching may be based on usage patterns of client data, such as frequently or recently used data. Caching criteria may be based on elapsed time and/or the type of data. In another example, the spanning storage interface may maintain locally cached copies of the most recent data backups from clients, such as the most recent full backup and the previous week's incremental backups.

[0077] Figure 4 illustrates a method 400 of retrieving an original data stream from deduplicated data according to an embodiment of the invention. In an embodiment, step 405 receives a data access request from a client.

[0078] Step 410 identifies a label map associated with the requested data. For example, if the data access request is for a file in the shell file system, an embodiment of step 410 retrieves a data stream identifier from the metadata of this shell file. Step 410 then retrieves the label map associated with the data stream identifier from memory, disk storage, or cloud storage. The label map includes a sequence of labels corresponding with a sequence of data segments representing the data stream. In an embodiment, the data stream identifier specifies the identity and/or the location of the label map container file in the disk or cloud storage. For example, a prefix portion of the data stream identifier may correspond with the file system path and/or file name or cloud data identifier of the label map container file used to store the label map representing the data stream. A suffix portion of the data stream identifier may be used to directly or indirectly specify the location of the label map within its label map container file.

[0079] Upon retrieving the label map associated with the data stream identifier, step 415 selects the next label in sequence in the label map. In an embodiment, method 400 may receive the data stream identifier with a request for the entire data stream. In this embodiment, the first iteration of step 415 selects the first label in the label map.

[0080] In another embodiment, method 400 may receive a data stream identifier with a request for only a portion of the data stream. In this embodiment, step 415 selects the first label corresponding with the beginning of the requested portion of the data stream. In an embodiment, each label map includes a label map table of contents providing the offset or relative position of each instance of a label with respect to the original data stream. The label map table of contents may be used to allow random or non-sequential access to a deduplicated data stream. In an embodiment, the requested portion of the data stream is specified with a starting data stream address or offset and/or an ending data stream offset or address. Step 415 uses this label map table of contents to identify the label corresponding with the starting data stream address or offset.

[0081] Decision block 420 determines if the data segment corresponding with the selected label is already stored in the slab cache in memory. In an embodiment, decision block 420 searches for the selected label in the slab cache to make this determination. If the data segment corresponding with the selected label is already stored in the slab cache in memory, then method 400 proceeds to step 430.

[0082] Conversely, if the data segment corresponding with the selected label is not stored in the slab cache in memory, step 425 accesses a slab data file including a previously generated instance of the data segment corresponding with the selected label. In an embodiment, step 425 uses the label name to identify and optionally locate the data segment slab file including the previously generated instance of the data segment. Step 425 may retrieve the slab file from cloud storage. In a further embodiment, step 425 first checks to see if the required slab file is cached locally by the spanning storage interface; if so, then step 425 retrieves the data segment from the local copy of the slab file, rather than from the cloud storage.

[0083] Step 425 retrieves at least the data segment corresponding with the selected label from its data segment slab file and adds it to the slab cache in memory. In an embodiment, step 425 retrieves all of the data segments included in this data segment slab file from local storage or cloud storage and adds them to the slab cache in memory. Step 425 also retrieves

and/or generates the labels and segment metadata for the retrieved data segments and adds these to the slab cache. Step 425 retrieves the segment reference counts for these data segments from the data segment slab file and adds these to the slab cache in memory. Step 425 also updates the reverse map cache with the labels and hashes or other data characterizations of the retrieved data segments.

[0084] Step 430 retrieves the data segment corresponding with the selected label from the slab cache. Step 435 adds all or a portion of this data segment to a data stream buffer or other data structure used to reconstruct the requested data stream. In an embodiment, steps 430 and 435 decompress the contents of the data segment prior to adding it to the data stream buffer. In another embodiment, data segments are decompressed upon being initially added to the slab cache. In still another embodiment, one or more data segments are decompressed after being added to the data stream buffer.

[0085] In an embodiment, method 400 may receive a request for only a portion of the data stream. In this embodiment, step 435 may need to remove the beginning of a data segment if the data segment is the first data segment in the requested portion of the data stream, such that the beginning of the data stream buffer matches the beginning of the requested portion of the data stream. Similarly, step 435 may need to remove the end of a data segment if the data segment is the last data segment in the requested portion of the data stream, such that the end of the data stream buffer matches the end of the requested portion of the data stream.

[0086] Decision block 440 determines if all of the labels corresponding with the requested data in the data stream have been processed by steps 410 to 435. If all of the labels corresponding with the requested data in the data stream have not been processed, method 400 returns to step 415 to process additional labels from the label map associated with the data stream.

[0087] Once all of the labels associated with the requested portion of the data stream have been processed, method 400 proceeds to step 445. Step 445 returns the data stream to the deduplicating storage interface client or other entity providing the data stream. Embodiments of method 400 may output the data stream in its entirety in step 445 or output portions of the requested portion of the data stream in step 445 in parallel with performing the other steps of method 400 to reconstruct other portions of the requested portion of the data stream. For example, step 425 may be performed asynchronously with other steps of method 400 so that

slab files may be retrieved from the cloud storage in the background while the spanning storage interface processes other labels in the label map.

[0088] Figure 5 illustrates a method 500 of deleting a data stream from a spanning storage interface according to an embodiment of the invention. In an embodiment, step 505 receives a data stream identifier from a deduplicating storage interface client.

[0089] Step 510 retrieves the label map associated with the data stream identifier from memory or disk storage. The label map includes a sequence of labels corresponding with a sequence of data segments representing the data stream. In an embodiment, the data stream identifier specifies the identity and/or the location of the label map container file in the disk storage. For example, a prefix portion of the data stream identifier may correspond with the file system path and/or file name of the label map container used to store the label map representing the data stream. A suffix portion of the data stream identifier may be used to directly or indirectly specify the location of the label map within its label map container file.

[0090] Upon retrieving the label map associated with the data stream identifier, step 515 selects the next label in sequence in the label map. In an embodiment, the first iteration of step 515 selects the first label in the label map.

[0091] Decision block 520 determines if the data segment corresponding with the selected label is already stored in the slab cache in memory. In an embodiment, decision block 520 searches for the selected label in the slab cache to make this determination. If the data segment corresponding with the selected label is already stored in the slab cache in memory, then method 500 proceeds to step 530.

[0092] Conversely, if the data segment corresponding with the selected label is not stored in the slab cache in memory, step 525 accesses a slab data file including a previously generated instance of the data segment corresponding with the selected label. In an embodiment, step 525 uses the label name to identify and optionally locate the data segment slab file including the previously generated instance of the data segment.

[0093] Step 525 retrieves at least the data segment corresponding with the selected label from its data segment slab file and adds it to the slab cache in memory. In an embodiment, step 525 retrieves all of the data segments included in this data segment slab file from disk storage or cloud storage and adds them to the slab cache in memory. Step 525 also retrieves and/or generates the labels and segment metadata for the retrieved data segments and adds

these to the slab cache. Step 525 retrieves the segment reference counts for these data segments from the data segment slab file and adds these to the slab cache in memory. Step 525 also updates the reverse map cache with the labels and hashes or other data characterizations of the retrieved data segments.

[0094] Step 530 decrements the reference count in the slab cache associated with the selected label. In an embodiment, if the reference count of a label is decremented to zero, then the label and its data segment are marked for deletion from the slab cache and its data segment slab file.

[0095] Decision block 535 determines if all of the labels in the label map have been processed by steps 510 to 530. If all of the labels corresponding with the requested data in the data stream have not been processed, method 500 returns to step 515 to process additional labels from the label map associated with the data stream.

[0096] Once all of the labels associated with the label map have been processed, method 500 proceeds to step 540. Step 540 updates the data segment slab files including any data segments affected by the deletion operation. In an embodiment, step 540 writes the updated and decremented reference counts for data segments associated with the label map back to their respective data segment slab files. In an embodiment, if the reference count of a data segment has been decremented to zero, an embodiment of step 540 marks this data segment for deletion from the data segment slab file. In a further embodiment, a garbage collection process removes unneeded data segments and associated reference counts and segment metadata from data segment slab files. An embodiment of step 540 transfers the updated slab files to the cloud storage.

[0097] Step 545 updates the label map container file to remove the label map associated with the data stream identifier. In an embodiment, if the disk storage supports sparse files, the label map may be deleted directly without rewriting the label map container file. In another embodiment, if sparse files are not supported by the disk storage, then unneeded label maps are marked for deletion. A garbage collection process, similar to that used by embodiments of step 540, may be used to remove unnecessary label maps by rewriting label map container files when the number or proportion of label maps marked for deletion exceeds a threshold. An embodiment of step 545 transfers the updated label map container files to the cloud storage.

[0098] In an embodiment, steps 525, 540, and 545 may perform transfers to and from the cloud storage via the wide-area network in parallel and/or asynchronously with other steps of method 500. Similarly to step 390 above, steps 540 and 545 may identify changes in the locally stored label map container files and slab files in comparison with their counterparts (if any) stored in the could storage. Steps 540 and 545 transfer the changed label map container files and slab files to the cloud storage. In an embodiment, steps 540 and 545 only communicates the changed or new data to the cloud storage.

[0099] Embodiments of method 500 may return a deletion confirmation to the deduplicating storage interface client or other entity. In one embodiment, the deletion confirmation is provided following the successful retrieval of the label map corresponding with the data stream identifier in step 510. The remainder of method 500 may be performed as a background or low priority process by the deduplication and/or replication modules without impacting the performance of the client. In another embodiment, the deletion confirmation is returned to the client following the completion of method 500.

[0100] A further embodiment of method 500 may allow for deletion of a specified portion of data from a data stream. In this embodiment, for data segments that are partially contained within the specified portion of the data stream, the data from these data segments is retrieved and truncated so that only data outside of the specified portion of the data stream remains. This modified data is then re-encoded as one or more revised data segments and corresponding labels, which may be new to the spanning storage interface or may match previously created data segments, as described above. The labels representing data segments contained wholly or partially within the specified portion of a data stream are removed from the label map. The reference counts of these data segments are updated accordingly. The label map is rewritten to remove unused labels and to add labels for revised data segments.

[0101] In an embodiment, one or more garbage collection processes removes unneeded data segments, labels, and metadata from caches and files. Embodiments of the garbage collection process or processes may be performed independently of the above methods, for example as a background or low-priority processes. Alternatively, some or all of the garbage collection processes may be performed as part of the above methods in creating or updating the slab and/or label map container files on disk storage and/or the slab cache and anchor caches in memory.

[0102] For example, a garbage collection process may remove unneeded data segments and associated reference counts and segment metadata from the data segment slab files. In an embodiment, the garbage collection process determines if the number or proportion of data segments marked for deletion in a data segment slab file exceeds a threshold. If this threshold is exceeded, then the entire data segment slab file is rewritten, with the data segments marked for deletion omitted from the rewritten data segment slab file.

[0103] In another example, a garbage collection process removes labels from the anchor cache after the corresponding data segments have been loaded into the slab cache. In an embodiment, a garbage collection process uses label metadata attributes to identify labels in the slab cache corresponding with representative data segments and then compares these identified labels with the labels in the anchor cache. If a label in the anchor cache matches a label in the slab cache, the garbage collection process removes this label from the anchor cache, as this data segment is now loaded into memory in the slab cache.

[0104] In many applications, some data segments may be used more frequently than other data segments. Typical frequently-used data segments can include data corresponding to repeating data patterns, such as data segments consisting entirely of null values or other data or file-format specific motifs.

[0105] To improve performance, an embodiment of the deduplicating data storage system stores frequently-used data segments separately from less-used data segments. In an embodiment, the deduplicating data storage system monitors the reference counts associated with data segments. When the reference count of a data segment is increased above a threshold value, that data segment is designated as a frequently-used data segment. An embodiment moves or copies this data segment to separate slab file reserved for frequently-used data segments. The frequently-used data segment is relabeled as it is transferred to the frequently-used data segment slab file.

[0106] In an embodiment, the frequently-used data segment slab file is similar to other data segment slab files, such as data segment slab file 250 discussed above. In still a further embodiment, data segment reference counts are not maintained or updated for frequently-used data segments; accordingly, data segment reference counts may be omitted from the frequently-used data segment slab file.

[0107] Embodiments of the invention may store frequently-used data segments in memory for improved performance using a variety of different techniques. In a first embodiment, all

of the frequently-used data segments and their associated labels and metadata from one or more frequently-used data segment slab files may be loaded into the slab cache or a separate frequently-used data segment cache during the initialization of the deduplication data storage system. In another embodiment, hashes or other data characterizations of all of the frequently-used data segments and their associated labels from one or more frequently-used data segment slab files are initially loaded into the anchor cache or a separate, similar cache. In this embodiment, the data associated with a frequently-used data segment is loaded into the slab cache as needed, in a similar manner as with other data segments as described above.

[0108] In an embodiment, frequently-used data segments stored in the slab cache are accessed for deduplicating additional data streams and retrieving deduplicated data in a similar manner as other data segments, as described above. However, in an embodiment, data segment reference counts are not maintained or updated in memory for frequently-used data segments. Therefore, an embodiment of the deduplicating data storage system does not increment an associated data segment reference count when a frequently-used data segment is used to deduplicate an additional data stream and does not decrement an associated data segment reference count when a data stream including a frequently-used data segment is deleted.

[0109] Embodiments of the deduplicating data storage system may be used in a variety of data storage applications to store files, objects, databases, or any other type or arrangement of data in a deduplicated form.

[0110] Figure 6 illustrates a computer system suitable for implementing embodiments of the invention. Figure 6 is a block diagram of a computer system 2000, such as a personal computer or other digital device, suitable for practicing an embodiment of the invention. Embodiments of computer system 2000 may include dedicated networking devices, such as wireless access points, network switches, hubs, routers, hardware firewalls, WAN and LAN network traffic optimizers and accelerators, network attached storage devices, storage array network interfaces, and combinations thereof.

[0111] Computer system 2000 includes a central processing unit (CPU) 2005 for running software applications and optionally an operating system. CPU 2005 may be comprised of one or more processing cores. Memory 2010 stores applications and data for use by the CPU 2005. Examples of memory 2010 include dynamic and static random access memory. Storage 2015 provides non-volatile storage for applications and data and may include fixed or

removable hard disk drives, flash memory devices, ROM memory, and CD-ROM, DVD-ROM, Blu-ray, HD-DVD, or other magnetic, optical, or solid state storage devices.

[0112] In a further embodiment, CPU 2005 may execute virtual machine software applications to create one or more virtual processors capable of executing additional software applications and optional additional operating systems. Virtual machine applications can include interpreters, recompilers, and just-in-time compilers to assist in executing software applications within virtual machines. Additionally, one or more CPUs 2005 or associated processing cores can include virtualization specific hardware, such as additional register sets, memory address manipulation hardware, additional virtualization-specific processor instructions, and virtual machine state maintenance and migration hardware.

[0113] Optional user input devices 2020 communicate user inputs from one or more users to the computer system 2000, examples of which may include keyboards, mice, joysticks, digitizer tablets, touch pads, touch screens, still or video cameras, and/or microphones. In an embodiment, user input devices may be omitted and computer system 2000 may present a user interface to a user over a network, for example using a web page or network management protocol and network management software applications.

[0114] Computer system 2000 includes one or more network interfaces 2025 that allow computer system 2000 to communicate with other computer systems via an electronic communications network, and may include wired or wireless communication over local area networks and wide area networks such as the Internet. Computer system 2000 may support a variety of networking protocols at one or more levels of abstraction. For example, computer system may support networking protocols at one or more layers of the seven layer OSI network model. An embodiment of network interface 2025 includes one or more wireless network interfaces adapted to communicate with wireless clients and with other wireless networking devices using radio waves, for example using the 802.11 family of protocols, such as 802.11a, 802.11b, 802.11g, and 802.11n.

[0115] An embodiment of the computer system 2000 may also include one or more wired networking interfaces, such as one or more Ethernet connections to communicate with other networking devices via local or wide-area networks.

[0116] The components of computer system 2000, including CPU 2005, memory 2010, data storage 2015, user input devices 2020, and network interface 2025 are connected via one or more data buses 2060. Additionally, some or all of the components of computer system

2000, including CPU 2005, memory 2010, data storage 2015, user input devices 2020, and network interface 2025 may be integrated together into one or more integrated circuits or integrated circuit packages. Furthermore, some or all of the components of computer system 2000 may be implemented as application specific integrated circuits (ASICS) and/or programmable logic.

[0117] Figure 7 illustrates an example disaster recovery application 700 of a spanning storage interface according to an embodiment of the invention. Disaster recovery application 700 may be used to provide redundant data access to storage clients in the event that the storage clients and/or cloud spanning storage interface at a first network location are disabled, destroyed, or otherwise inaccessible or inoperable.

[0118] In example disaster recovery application 700, a first network location A 705 includes a first spanning storage interface 710. Spanning storage interface 710 provides storage access to one or more storage clients, such as storage client 720A and backup server 720B, via a local area network and/or a storage area network. Spanning storage interface 710 deduplicates data received from storage clients and transfers the deduplicated data via the wide area network 780 to one or more cloud storage services, such as cloud storage services 770 and 775, for storage. The spanning storage interface 710 may also retrieve deduplicated data via the wide area network 780 from one or more cloud storage services and reconstruct this data in its original form to provide to storage clients.

[0119] As discussed above, the spanning storage interface 710 includes local storage 715 to improve data access performance. Local storage 715 includes a local cache A 725 of a portion of the storage data provided by storage clients at network location A 705.

[0120] To provide disaster recovery, example application 700 includes a second network location B 735. Network location B 735 includes a second spanning storage interface 740. Spanning storage interface 740 is provided for disaster recovery operations and may be used to access the data associated with the first network location A 705 in the event that network location A 705 is disabled, destroyed, or otherwise inaccessible or inoperable.

[0121] To provide disaster recovery data access, the second spanning storage interface 740 can access deduplicated data stored in one or more of the cloud storage services 770 and/or 775 via wide-area network 780. The second spanning storage interface 740 reconstructs the original data from the retrieved deduplicated data and provides it to storage clients.

[0122] The second spanning storage interface 740 includes local storage B 745 for improving data access performance. In an embodiment, a copy 760 of some or all or the local cache A 725 used by the first spanning storage interface 710 is transferred to the local storage B 745 while the first network location 705 is operating. In the event of a disaster affecting the first network location 705, the second spanning storage interface 740 can provide data access to the first network location's data with the improved performance benefit provided by the copy of local cache A 760 in its local storage B 745.

[0123] Network location B 735 may be a dedicated disaster recovery network location. Alternatively, network location B may also optionally be used with one or more local storage clients, such as storage clients 750A and backup server 750B. In this further example, the second spanning storage interface B 740 performs data deduplication and facilitates cloud storage for data from storage clients 750. Like the first spanning storage interface 710, the second spanning storage interface B 740 in this example deduplicates second data received from storage clients at network location B 735 and transfers this second deduplicated data via the wide area network 780 to one or more cloud storage services, such as cloud storage services 770 and 775, for storage. The second spanning storage interface 740 may also retrieve second deduplicated data via the wide area network 780 from one or more cloud storage services and reconstruct this second data in its original form to provide to storage clients at the second network location B 735. To improve the performance of the second spanning storage interface 740, its local storage B 745 may include a local cache B 765, which includes a portion of the storage data provided by storage clients at network location B 735.

[0124] In yet a further embodiment, spanning storage interfaces 710 and 740 can operate in a paired disaster recovery configuration. For example, the second spanning storage interface 740 at network location B 735 may act as disaster recovery for the first spanning storage interface 710 at the first network location A 705. As described above, the local storage B 745 at the second network location B 735 may include a copy 760 of the local cache A 725 used by the first spanning storage interface 710. The copy 760 of local cache A in local storage B 745 improves the initial performance of the second spanning storage interface 740 in the event that it is required to substitute for the first spanning storage interface 710.

[0125] Similarly, in the paired disaster recovery configuration, first spanning storage interface 710 may act as disaster recovery for the second spanning storage interface 740. In

the event that the second spanning storage interface 740 is destroyed, disabled, or otherwise available to its storage clients, the first spanning storage interface 710 may provide access to storage data associated with the network location 735. Additionally, the local storage A 715 includes a copy 730 of the local cache B 765 used by the second spanning storage interface 740. The copy 730 of the local cache B 765 is transferred to the local storage A 715 while the second spanning storage interface 740 is operating. The copied version of local cache B 730 in local storage A 715 improves the initial performance of the first spanning storage interface 710 in the event that it is required to substitute for the second spanning storage interface 740.

[0126] In an further embodiment, the paired disaster recovery configuration can be extended to include additional network locations, with local storage at each network location including a copy of at least one (and possibly more than one) local cache from other spanning storage interfaces.

[0127] In an embodiment, copies of local caches of spanning storage interfaces may be transferred directly between network locations. For example, spanning storage interfaces at different network locations may communicate with each other to transfer and update copies of their local caches at other network locations. In another embodiment, a spanning storage interface can retrieve a portion of the deduplicated data from a cloud storage service to recreate a copy of a local cache of another spanning storage interface.

[0128] Further embodiments can be envisioned to one of ordinary skill in the art. In other embodiments, combinations or sub-combinations of the above disclosed invention can be advantageously made. The block diagrams of the architecture and flow charts are grouped for ease of understanding. However it should be understood that combinations of blocks, additions of new blocks, re-arrangement of blocks, and the like are contemplated in alternative embodiments of the present invention.

[0129] The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense. It will, however, be evident that various modifications and changes may be made thereunto without departing from the broader spirit and scope of the invention as set forth in the claims.

WHAT IS CLAIMED IS:

1. A local and cloud spanning storage interface comprising:

a front-end interface adapted to communicate with at least one storage client via a local-area network;

a back-end interface adapted to communicate with at least one cloud storage service via a wide-area network;

a data deduplication module adapted to reduce data redundancy in data received from the storage client to produce deduplicated data; and

a replication module adapted to transfer deduplicated data to the cloud storage service using the back-end interface.

2. The local and cloud spanning storage interface of claim 1, wherein:

the data deduplication module includes:

slab cache adapted to store a subset of labels and associated data segments in memory;

a reverse map cache adapted to store associations in memory between each of the subset of labels in the slab cache with a portion of a data stream; and

an anchor cache adapted to store associations in memory between portions of the data stream and a second subset of labels and data segments not stored in the slab cache; and

a local data storage adapted to store a copy of at least a portion of the data received from the storage client, wherein the local data storage includes:

a copy of a data segment slab file adapted to store in non-volatile storage the set of labels and associated data segments; and

a copy of a label map container file adapted to store in non-volatile storage at least one label map specifying an arrangement of labels corresponding with the data stream.

3. The local and cloud spanning storage interface of claim 2, wherein the data segment slab file and the label map container file are stored using the cloud storage service.

4. The local and cloud spanning storage interface of claim 2, wherein the copy of the data segment slab file and the copy of the label map container file are selected

from a plurality of data segment slab files and label map container files stored using the cloud storage service.

5. The local and cloud spanning storage interface of claim 4, wherein the copy of the data segment slab file and the copy of the label map container file are selected from a plurality of data segment slab files and label map container files based on a cache criteria.

6. The local and cloud spanning storage interface of claim 1, comprising:

a shell file system module adapted to present a shell file system representing the data to the storage client.

7. The local and cloud spanning storage interface of claim 1, wherein the front-end interface includes a file system interface adapted to communicate with the storage client using a file system protocol.

8. The local and cloud spanning storage interface of claim 1, wherein the front-end interface includes a backup system interface adapted to communicate with the storage client using a backup system protocol.

9. The local and cloud spanning storage interface of claim 1, wherein the front-end interface includes a cloud storage interface adapted to communicate with the storage client using a cloud storage protocol.

10. The local and cloud spanning storage interface of claim 1, wherein the front-end interface includes an archival interface adapted to communicate with the storage client using an archival protocol.

11. The local and cloud spanning storage interface of claim 1, wherein the front-end interface includes an object interface adapted to communicate with the storage client using a binary large object protocol.

12. The local and cloud spanning storage interface of claim 1, wherein the front-end interface includes a block storage interface adapted to communicate with the storage client using a block storage protocol.

13. The local and cloud spanning storage interface of claim 12, wherein the block storage protocol includes iSCSI.

14. The local and cloud spanning storage interface of claim 1, wherein the replication module is adapted to communicate a first portion of the deduplicated data to a first cloud storage service and a second portion of the deduplicated data to a second cloud storage service.

15. The local and cloud spanning storage interface of claim 14, wherein the first and second portions of the deduplicated data determined from a specification provided by a user.

16. The local and cloud spanning storage interface of claim 14, wherein the first and second portions of the deduplicated data are determined at least in part on a file path.

17. The local and cloud spanning storage interface of claim 1, wherein the replication module is adapted to abandon the transfer of the deduplicated data to the cloud storage service in response to a quota being exceeded.

18. The local and cloud spanning storage interface of claim 17, wherein the quota is associated with a storage client.

19. A method of deduplicating a data stream, the method comprising:

receiving a data stream;

generating new data segments from the data stream;

determining if the new data segments match previously generated data segments stored only on a cloud storage service;

in response to the determination that at least one of new data segments matches one of the previously generated data segments stored only on the cloud storage service, retrieving a set of data segments including the matching previously generated data segment from the cloud storage service;

assigning provisional labels to at least a portion of the new data segments not matching locally cached data segments; and

adding the provisional labels to the label map associated with the data stream.

20. The method of claim 19, comprising:

comparing the new data segments assigned to the provisional labels with the set of data segments retrieved from the cloud storage service;

in response to the determination that the new data segment matches one of the set of data segments, discarding the provisional label assigned to the new data segment and assigning a previously generated label associated with the matching one of the set of data segments to the new data segment.

21. The method of claim 19, comprising:

determining a set of changes to a locally stored label map container file and a locally stored slab file; and

transferring the set of changes to the cloud storage service to update a full and authoritative set of data segments and label maps.

22. The method of claim 21, wherein transferring the set of changes includes performing an atomic operation to update the full and authoritative set of data segments and label maps.

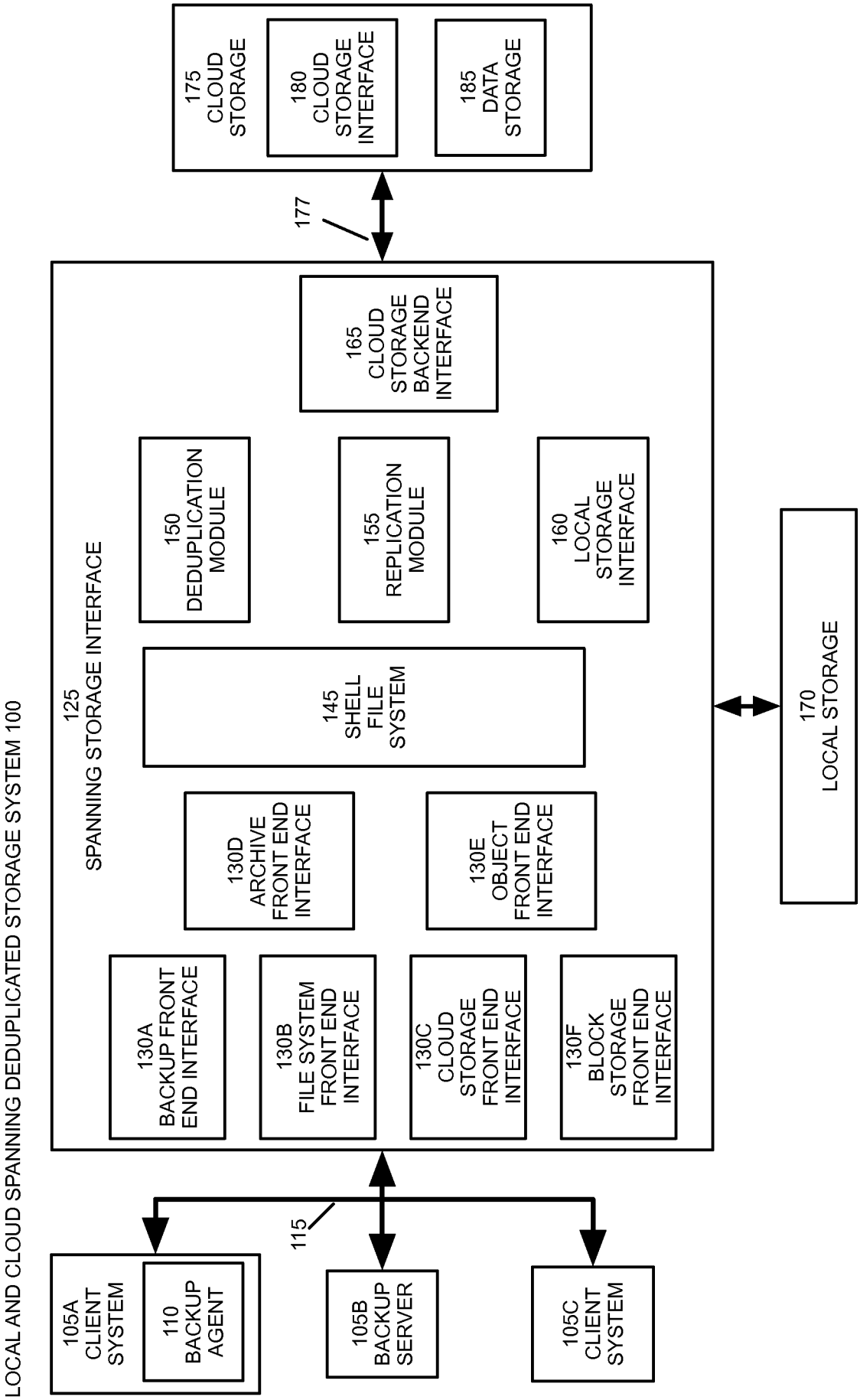
LOCAL AND CLOUD SPANNING DEDUPLICATED STORAGE SYSTEM 100
105A CLIENT SYSTEM
110 BACKUP AGENT
105B BACKUP SERVER
105C CLIENT SYSTEM
115
125 SPANNING STORAGE INTERFACE
130A BACKUP FRONT END INTERFACE
130B FILE SYSTEM FRONT END INTERFACE
130C CLOUD STORAGE FRONT END INTERFACE
130F BLOCK STORAGE FRONT END INTERFACE
130D ARCHIVE FRONT END INTERFACE
130E OBJECT FRONT END INTERFACE
145 SHELL FILE SYSTEM
150 DEDUPLICATION MODULE
155 REPLICATION MODULE
160 LOCAL STORAGE INTERFACE
165 CLOUD STORAGE BACKEND INTERFACE
170 LOCAL STORAGE
177
175 CLOUD STORAGE
180 CLOUD STORAGE INTERFACE
185 DATA STORAGE

FIG. 1

DEDUPLICATING DATA STORAGE SYSTEM 200

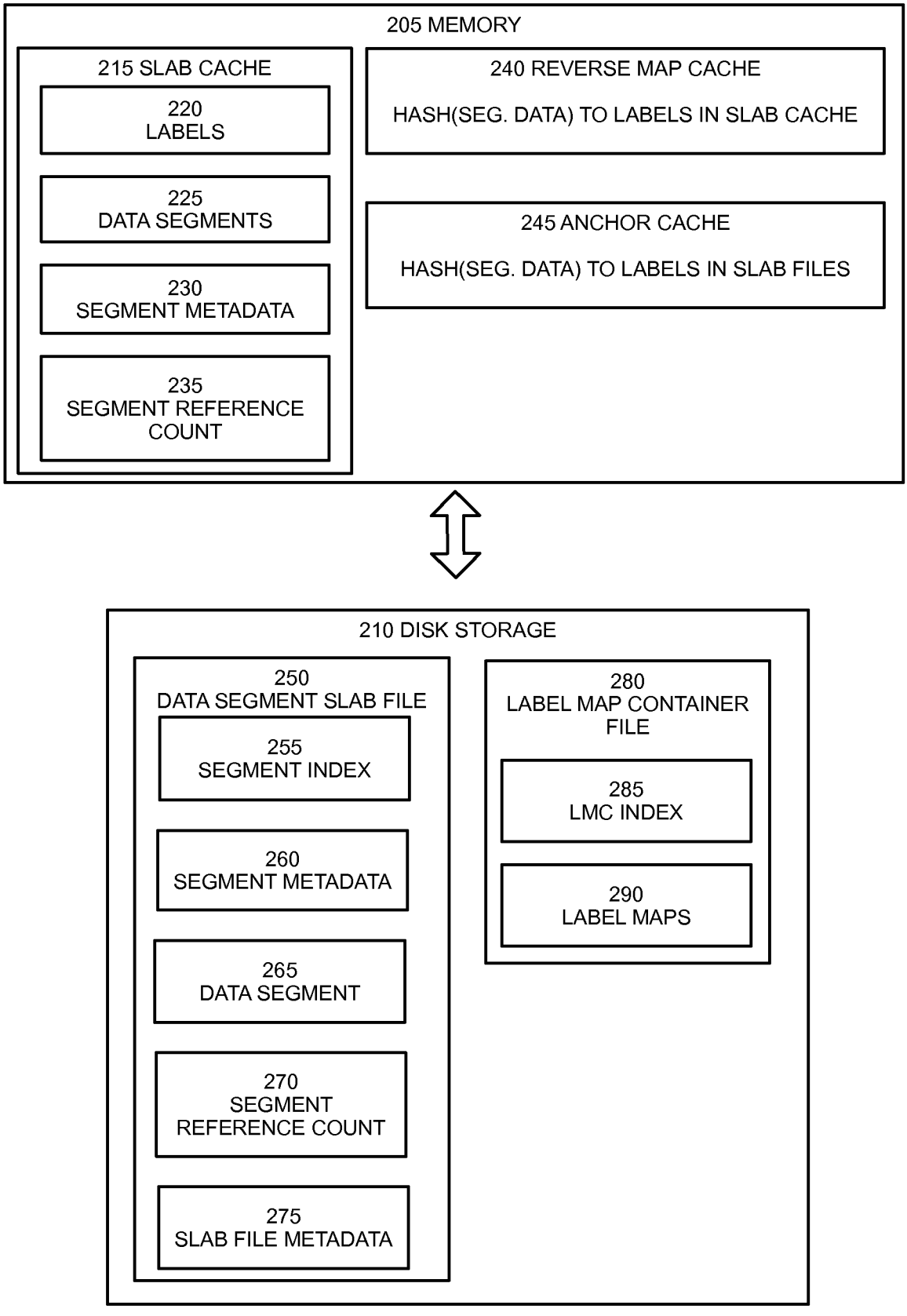

FIG. 2

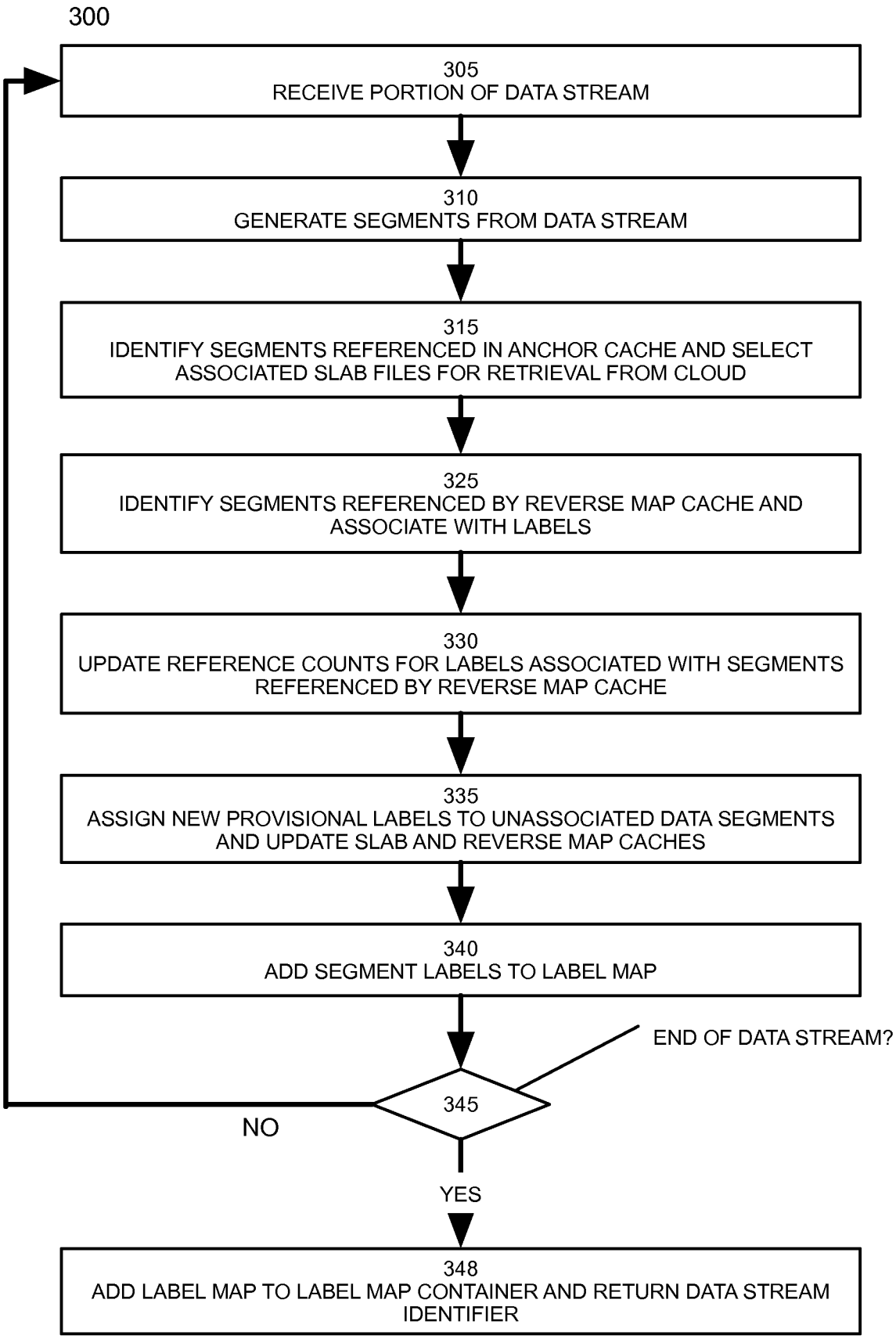
300
305
RECEIVE PORTION OF DATA STREAM
310
GENERATE SEGMENTS FROM DATA STREAM
315
IDENTIFY SEGMENTS REFERENCED IN ANCHOR CACHE AND SELECT ASSOCIATED SLAB FILES FOR RETRIEVAL FROM CLOUD
325
IDENTIFY SEGMENTS REFERENCED BY REVERSE MAP CACHE AND ASSOCIATE WITH LABELS
330
UPDATE REFERENCE COUNTS FOR LABELS ASSOCIATED WITH SEGMENTS REFERENCED BY REVERSE MAP CACHE
335
ASSIGN NEW PROVISIONAL LABELS TO UNASSOCIATED DATA SEGMENTS AND UPDATE SLAB AND REVERSE MAP CACHES
340
ADD SEGMENT LABELS TO LABEL MAP
END OF DATA STREAM?
345
NO
YES
348
ADD LABEL MAP TO LABEL MAP CONTAINER AND RETURN DATA STREAM IDENTIFIER

FIG. 3A

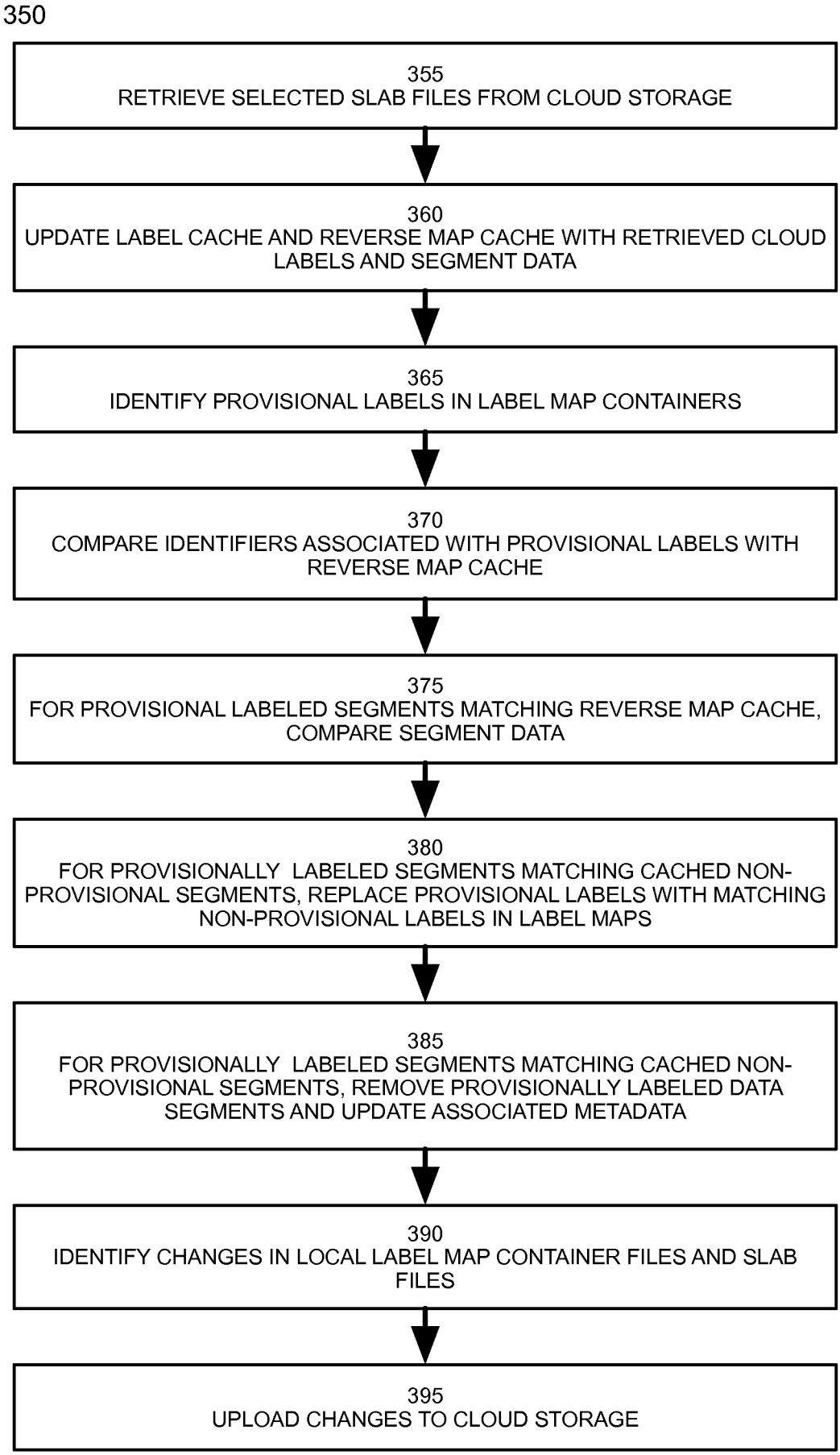
350
355
RETRIEVE SELECTED SLAB FILES FROM CLOUD STORAGE
360
UPDATE LABEL CACHE AND REVERSE MAP CACHE WITH RETRIEVED CLOUD LABELS AND SEGMENT DATA
365
IDENTIFY PROVISIONAL LABELS IN LABEL MAP CONTAINERS
370
COMPARE IDENTIFIERS ASSOCIATED WITH PROVISIONAL LABELS WITH REVERSE MAP CACHE
375
FOR PROVISIONAL LABELED SEGMENTS MATCHING REVERSE MAP CACHE, COMPARE SEGMENT DATA
380
FOR PROVISIONALLY LABELED SEGMENTS MATCHING CACHED NON-PROVISIONAL SEGMENTS, REPLACE PROVISIONAL LABELS WITH MATCHING NON-PROVISIONAL LABELS IN LABEL MAPS
385
FOR PROVISIONALLY LABELED SEGMENTS MATCHING CACHED NON-PROVISIONAL SEGMENTS, REMOVE PROVISIONALLY LABELED DATA SEGMENTS AND UPDATE ASSOCIATED METADATA
390
IDENTIFY CHANGES IN LOCAL LABEL MAP CONTAINER FILES AND SLAB FILES
395
UPLOAD CHANGES TO CLOUD STORAGE

FIG. 3B

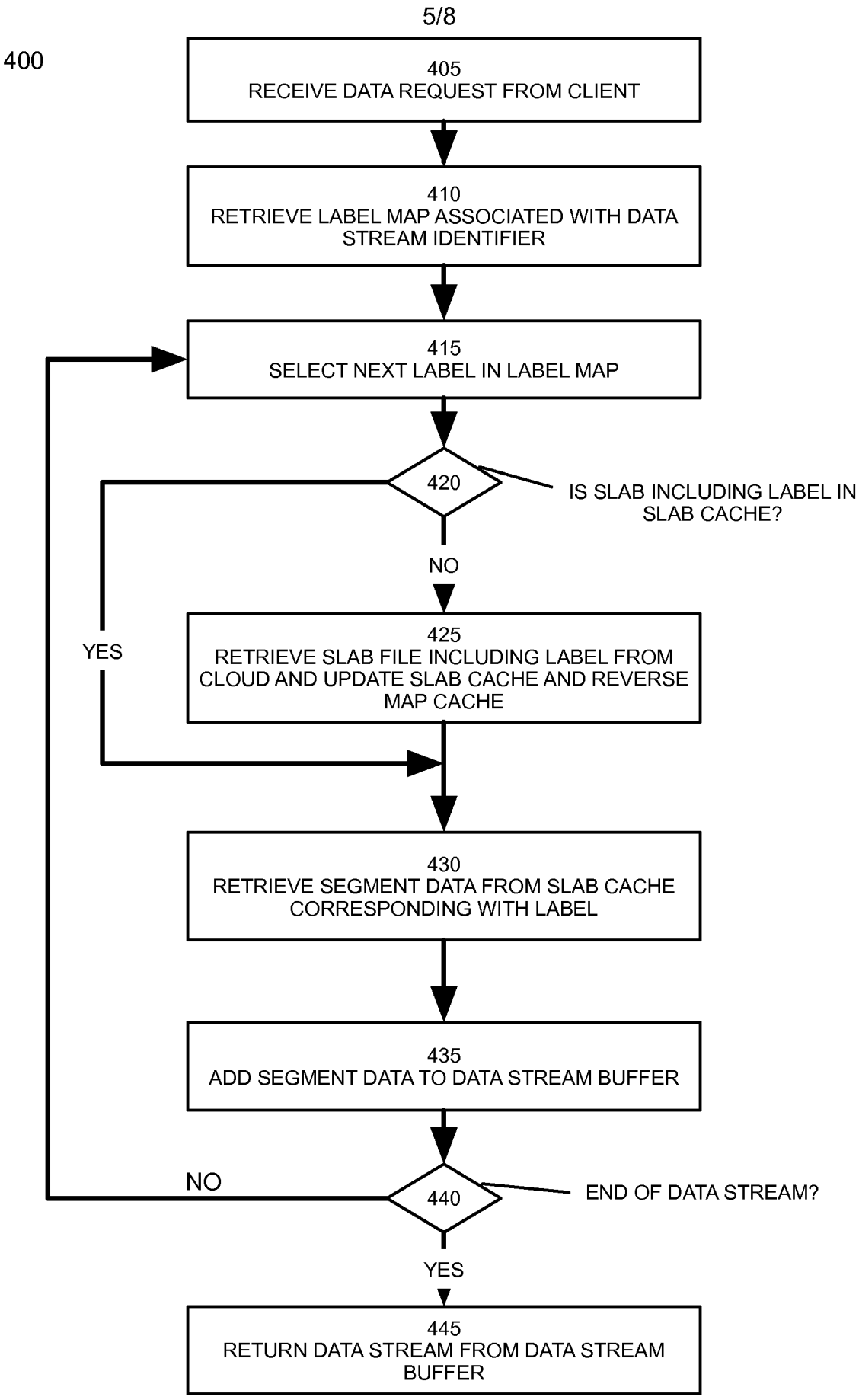
400
405
RECEIVE DATA REQUEST FROM CLIENT
410
RETRIEVE LABEL MAP ASSOCIATED WITH DATA STREAM IDENTIFIER
415
SELECT NEXT LABEL IN LABEL MAP
420
IS SLAB INCLUDING LABEL IN SLAB CACHE?
NO
YES
425
RETRIEVE SLAB FILE INCLUDING LABEL FROM CLOUD AND UPDATE SLAB CACHE AND REVERSE MAP CACHE
430
RETRIEVE SEGMENT DATA FROM SLAB CACHE CORRESPONDING WITH LABEL
435
ADD SEGMENT DATA TO DATA STREAM BUFFER
NO
440
END OF DATA STREAM?
YES
445
RETURN DATA STREAM FROM DATA STREAM BUFFER

FIG. 4

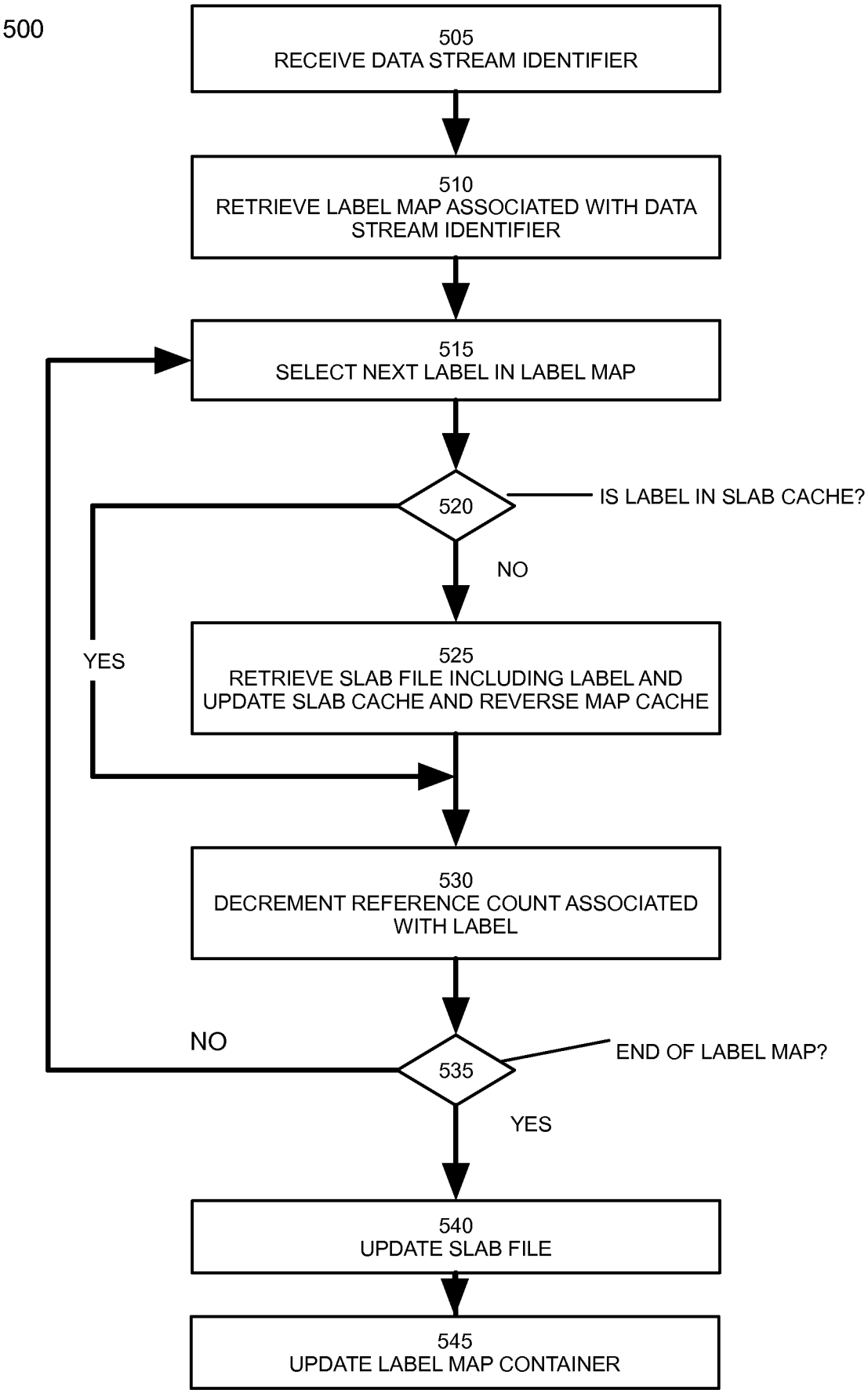
500
505
RECEIVE DATA STREAM IDENTIFIER
510
RETRIEVE LABEL MAP ASSOCIATED WITH DATA STREAM IDENTIFIER
515
SELECT NEXT LABEL IN LABEL MAP
520
IS LABEL IN SLAB CACHE?
NO
YES
525
RETRIEVE SLAB FILE INCLUDING LABEL AND UPDATE SLAB CACHE AND REVERSE MAP CACHE
530
DECREMENT REFERENCE COUNT ASSOCIATED WITH LABEL
NO
535
END OF LABEL MAP?
YES
540
UPDATE SLAB FILE
545
UPDATE LABEL MAP CONTAINER

FIG. 5

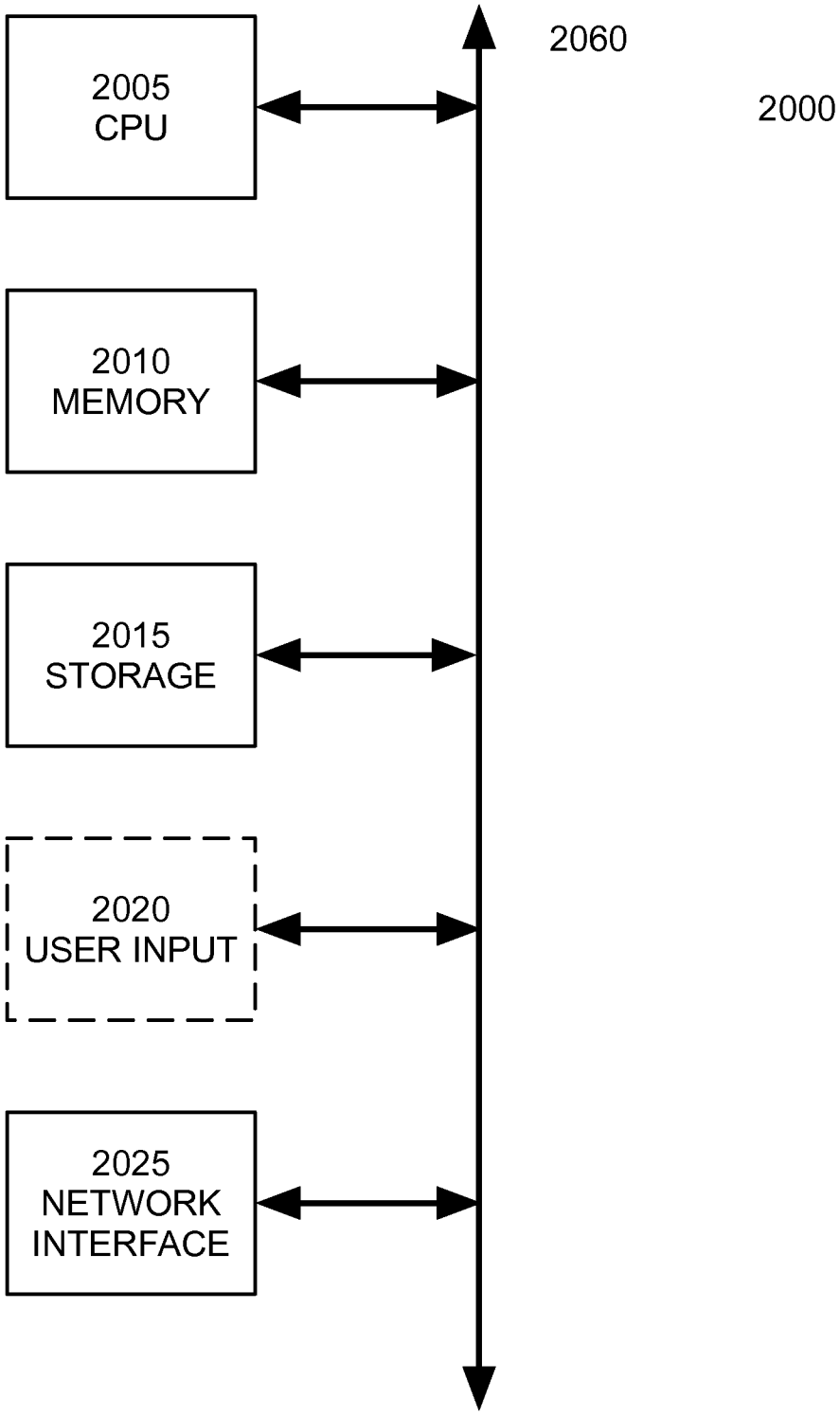
2060
2000
2005
CPU
2010
MEMORY
2015
STORAGE
2020
USER INPUT
2025
NETWORK
INTERFACE

FIG. 6

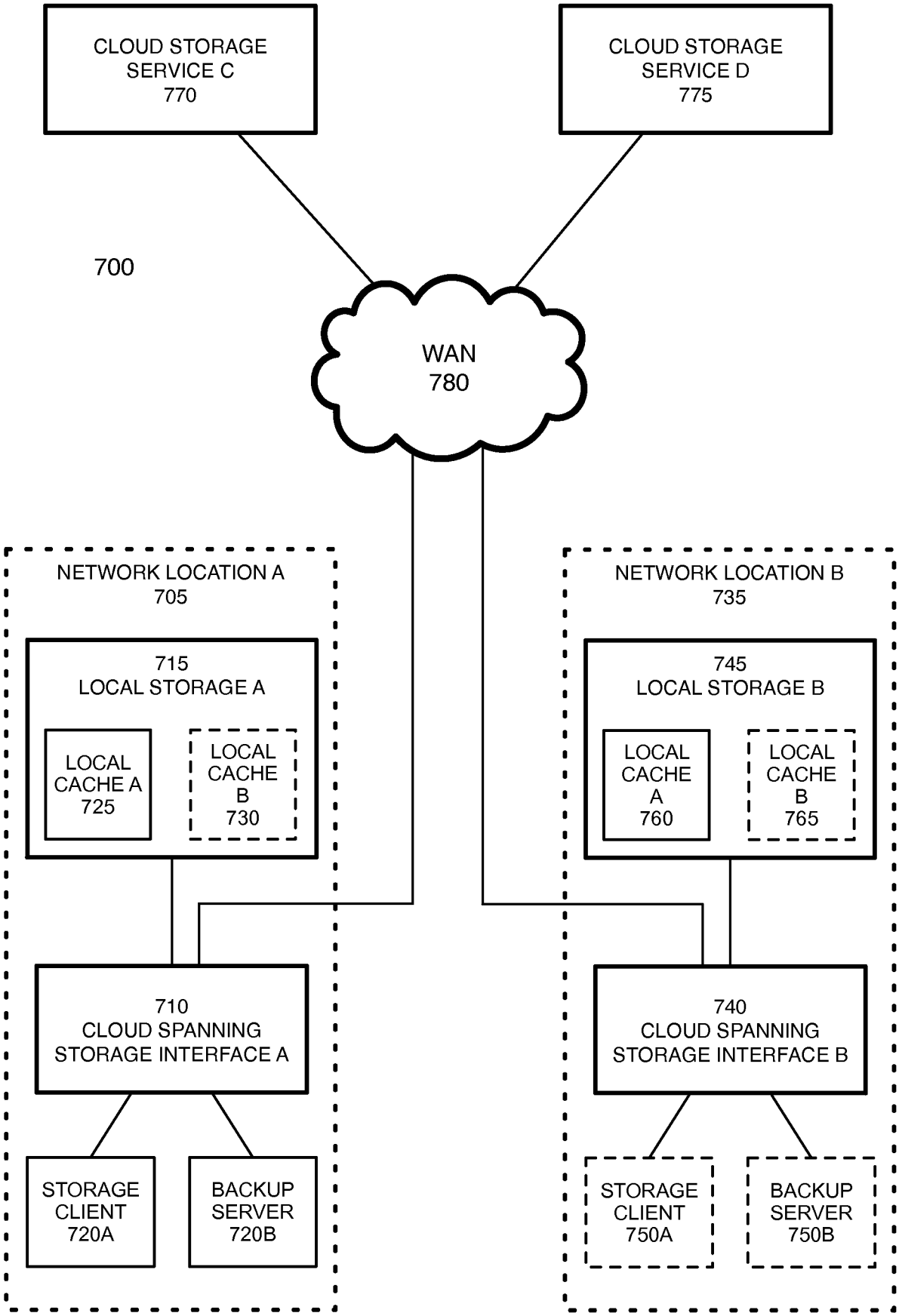
CLOUD STORAGE SERVICE C 770
CLOUD STORAGE SERVICE D 775
700
WAN 780
NETWORK LOCATION A 705
715 LOCAL STORAGE A
LOCAL CACHE A 725
LOCAL CACHE B 730
710 CLOUD SPANNING STORAGE INTERFACE A
STORAGE CLIENT 720A
BACKUP SERVER 720B
NETWORK LOCATION B 735
745 LOCAL STORAGE B
LOCAL CACHE A 760
LOCAL CACHE B 765
740 CLOUD SPANNING STORAGE INTERFACE B
STORAGE CLIENT 750A
BACKUP SERVER 750B

FIG. 7

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US 10/62126

A. CLASSIFICATION OF SUBJECT MATTER
IPC(8) - G06F 7/00, G06F 17/00 (2011.01)
USPC - 707/692
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC(8): G06F 7/00, G06F 17/00 (2011.01)
USPC: 707/692

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
USPC: 707/609, 664, 687, 693, E17.055; 709/230; 711/1, 6, E12.016 (keyword limited; terms below)

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
PubWEST(PGPB, USPT, EPAB, JPAB); Google; Search Terms: interfac storage database storehouse Internet cloud network storage database storehouse memory span extend bridg cross travers storage database storehouse front-end frontend front forward foreground leading back-end backend back rear reverse deduplicat de-duplicat aggregat fusion data informat

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X --- Y	US 7,620,775 B1 (WAXMAN) 17 November 2009 (17.11.2009), entire document, especially col 1, ln 54-60; col 2, ln 4-13; col 3, ln 8-10; col 4, ln 21-23, 39-40; col 5, ln 1-29, 55-57; col 6, ln 4-12; col 7, ln 23-27, 62-67; col 8, ln 2-7, 18-20; col 9, ln 60-63; col 11, ln 49-67; col 12, ln 5-7; col 13, ln 2-7, 26-29; col 14, ln 19-23, 38-41; col 15, ln 18-30; col 16, ln 51-53; col 20, ln 13-15; col 21, ln 55-58	1, 7, 9, 11-13, 19, 21, 22 --- 2-6, 8, 10, 14-18, 20
Y	US 2009/0182953 A1 (MERKEY et al.) 16 July 2009 (16.07.2009), entire document, especially para. [0005], [0008], [0055], [0056], [0058], [0067], [0069], [0070], [0081], [0087], [0088], [0092], [0113], [0115], [0116], [0159], [0162], [0168], [0176]	2-6, 8, 10, 20
Y	US 2009/0204718 A1 (LAWTON et al.) 13 August 2009 (13.08.2009), entire document, especially para. [0007], [0043], [0083], [0091]	14-16
Y	US 2009/0276771 A1 (NICKOLOV et al.) 05 November 2009 (05.11.2009), entire document, especially para. [0951], [2220], [2926], [2930]	17, 18

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

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

Date of the actual completion of the international search	Date of mailing of the international search report
02 February 2011 (02.02.2011)	18 FEB 2011
Name and mailing address of the ISA/US Mail Stop PCT, Attn: ISA/US, Commissioner for Patents P.O. Box 1450, Alexandria, Virginia 22313-1450 Facsimile No. 571-273-3201	Authorized officer: Lee W. Young PCT Helpdesk: 571-272-4300 PCT OSP: 571-272-7774

Form PCT/ISA/210 (second sheet) (July 2009)