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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
8 August 2013 (08.08.2013)

WIPO | PCT

(10) International Publication Number
WO 2013/115822 A1

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

(21) International Application Number:
PCT/US2012/023649

(22) International Filing Date:
2 February 2012 (02.02.2012)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant *(for all designated States except US)*: **HEWLETT-PACKARD DEVELOPMENT COMPANY, L.P.** [US/US]; 11445 Compaq Center Drive W., Houston, Texas 77070 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **LILLIBRIDGE, Mark D.** [US/US]; 1501 Page Mill Road, Palo Alto, California 94304-1100 (US). **ESHGHI, Kave** [GB/US]; 1501 Page Mill Road, Palo Alto, California 94304-1100 (US). **WATKINS, Mark R.** [GB/GB]; Longdown Avenue, Stoke Gifford Bristol BS348QZ (GB).

(74) Agents: **ORTEGA, Arthur S.** et al.; Hewlett-Packard Company, Intellectual Property Administration, 3404 E. Harmony Road, Mail Stop 35, Fort Collins, Colorado 80528 (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, QA, RO, RS, RU, RW, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, 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,

*[Continued on next page]*

(54) Title: SYSTEMS AND METHODS FOR DATA CHUNK DEDUPLICATION

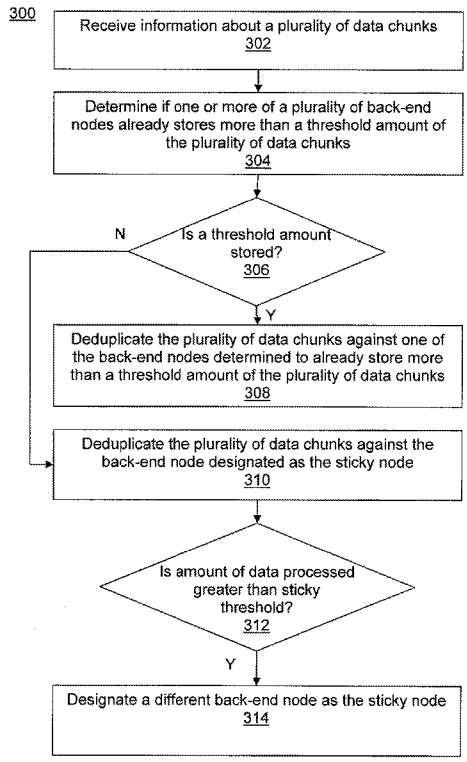



FIG. 3

(57) Abstract: A method includes receiving information about a plurality of data chunks and determining if one or more of a plurality of back-end nodes already stores more than a threshold amount of the plurality of data chunks where one of the plurality of back-end nodes is designated as a sticky node. The method further includes, responsive to determining that none of the plurality of back-end nodes already stores more than a threshold amount of the plurality of data chunks, deduplicating the plurality of data chunks against the back-end node designated as the sticky node. Finally, the method includes, responsive to an amount of data being processed, designating a different back-end node as the sticky node.

WO 2013/115822 A1

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).

**Declarations under Rule 4.17**:

— *as to the identity of the inventor (Rule 4.17(i))*

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

**Published**:

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

# SYSTEMS AND METHODS FOR DATA CHUNK DEDUPLICATION

## BACKGROUND

**[0001]** Data deduplication is a data compression technique for eliminating redundant data and is particularly useful to improve storage utilization, for example when backing up large amounts of data on a regular basis. When using chunk-based inline deduplication for backup, a data stream to be backed up is broken up into smaller pieces (typically on the order of a few kilobytes) called "chunks" using a chunking algorithm and a hash is computed for each chunk (*e.g.*, using an MD5 hash function or SHA hash function). The hash for a chunk to be backed up is looked up in one or more indexes and/or other data structures maintained by the system. If the system determines as a result that it already has a chunk with that hash, then that chunk is a duplicate of data already stored in the system and need not be stored again. Thus, the hash for each incoming chunk is looked up in one or more indexes and/or other data structures maintained by the system.

**[0002]** In order to limit the amount of expensive memory required while maintaining performance, complicated methods of indexing may be used in practice. In one example ("sparse indexing method"), the system maintains a "sparse index," which maps a small subset of hashes called "hooks" (*e.g.*, one out of every 32 or 64 unique hashes) to an index with information about the chunk with that hook as well as chunks that occurred near that chunk in the past.

**[0003]** In another example ("Bloom filter method"), a Bloom filter tracks the hashes of chunks that are stored by the system and a full chunk index maps the hash of every chunk stored by the system to an index with information about the chunk with that hash as well as chunks that occurred near that chunk in the past. The full chunk index is only consulted when the Bloom filter determines that the input chunk has already been stored in order to reduce the number of accesses to the full chunk index.

**[0004]** A single node using one of the above-described methods provides acceptable performance for applications where the amount of data being backed up is low, or where high throughput is not needed. However, for enterprise-type

applications where data backup requirements are much higher, employing multiple-node storage systems may be beneficial. One way to do this is to appropriately route incoming data between a number of mostly autonomous back-end nodes, each of which may be employing one of the above exemplary methods. Each batch of incoming data is deduplicated against only one back-end node under this architecture, so it is important to route similar batches of data to the same back-end node to ensure efficient deduplication to minimize the storage space impact of backing up data.

## BRIEF DESCRIPTION OF THE DRAWINGS

**[0005]** For a detailed description of exemplary implementations, reference will now be made to the accompanying drawings in which:

**[0006]** Figure 1a shows a block diagram of a multi-node data deduplication system in accordance with various examples;

**[0007]** Figure 1b shows an example architecture of a front-end node in accordance with various examples;

**[0008]** Figure 2 shows an example diagram of interleaved data streams in accordance with various examples; and

**[0009]** Figure 3 shows a method flowchart in accordance with various examples.

## NOTATION AND NOMENCLATURE

**[0010]** Certain terms are used throughout the following description and claims to refer to particular system components. As one skilled in the art will appreciate, computer companies may refer to a component by different names. This document does not intend to distinguish between components that differ in name but not function. In the following discussion and in the claims, the terms “including” and “comprising” are used in an open-ended fashion, and thus should be interpreted to mean “including, but not limited to… .” Also, the term “couple” or “couples” is intended to mean either an indirect, direct, optical or wireless electrical connection. Thus, if a first device couples to a second device, that connection may be through a direct electrical connection, through an indirect electrical connection via other devices and connections, through an optical electrical connection, or through a wireless electrical connection.

**[0011]** As used herein, the term "chunk" refers to a subset of a data stream that is produced using a chunking algorithm.

**[0012]** As used herein, the term "segment" refers to a group of chunks that is produced using a segmenting algorithm.

**[0013]** As used herein, the term "hash" refers to an identification of a chunk that is created using a hash function.

**[0014]** As used herein, the term "hook" refers to a hash of a chunk that has been sampled.

**[0015]** As used herein, the term "block" refers to a division of a file or data stream that is interleaved with other files or data streams. For example, interleaved data may comprise 1a, 2a, 3a, 1b, 2b, 1c, 3b, 2c, where 1a is the first block of underlying stream one, 1b is the second block of underlying stream one, 2a is the first block of underlying stream two, etc. In some cases, the blocks may differ in length.

**[0016]** As used herein, the term "deduplicate" refers to the act of logically storing a chunk, segment, or other division of data in a storage system or at a storage node in a way that there is physically only one (or, in non-ideal rare cares, a few) copy of each unique chunk at the system or node. For example, deduplicating ABC, DBC and EBF (where each letter represents a unique chunk) in turn against an initially-empty storage node results in only one physical copy of B but three logical copies.

**[0017]** For example, if a chunk is deduplicated against a storage location and the chunk is not previously stored at the storage location, then the chunk is physically stored at the storage location. However, if the chunk is deduplicated against the storage location and the chunk is already stored at the storage location, then the chunk is not physically stored at the storage location again. As yet another example, if multiple chunks are deduplicated against the storage location and only some of the chunks are already stored at the storage location, then only the chunks not previously stored at the storage location are physically stored at the storage location as part of the deduplication.

**[0018]** As used herein, the term "sticky routing" refers to a new data-storage routing algorithm that tends to store large continuous pieces of new data each at a single back-end node but deduplicates old data in small pieces, each against whichever back-end node it appears most similar to. In some cases, new segments are deduplicated against a currently designated "sticky" storage node, which is changed when enough new data is accumulated at the currently designated "sticky" storage node.

## DETAILED DESCRIPTION

**[0019]** FIG. 1a shows an example multi-node deduplication architecture 100 including a storage system 101. Although described as using the sparse indexing method, the Bloom filter method as well as other methods of indexing may be alternately used. The multi-node architecture 100 comprises a client 102a coupled to a front-end node 104a. Additional clients 102b, 102c, 102d and additional front-end node 104b are shown as an example, although other examples do not require multiple clients or front-end nodes. In the case where there are multiple clients and/or front-end nodes, the clients possess functionality similar to exemplary client 102a and the front-end nodes possess functionality similar to exemplary front-end node 104a; in other words, the functionality described herein may be carried out by multiple clients and front-end nodes operating in parallel. In some cases, the association between clients 102a-d and front-end nodes 104a, 104b may change over time.

**[0020]** The client 102a may comprise a service requester in a typical client-server network and the front-end node 104a may comprise a processing unit and other functional elements of a server that provides resources or services to the client 102a. Additionally, the front-end node 104a may couple to or include a machine-readable storage device 112 (as shown in FIG. 1b), such as a compact disc, floppy disc, flash-based storage, or other non-transitory storage device. The front-end node 104a may execute instructions contained on the machine-readable storage device 112 and, as a result, perform various functions described below.

**[0021]** The front-end node 104a is shown comprising a segmentation module 105a. The segmentation module 105a performs chunking and segmentation

algorithms to chunk and segment incoming data streams received from the clients 102a and 102b. In some cases, the chunking algorithm may define chunks by physical layer constraints (*e.g.*, 4-kilobyte chunk size in Write Anywhere File Layout (WAFL)) while in other cases, chunks may be defined using a "sliding block" algorithm where a window is passed along the data stream to identify more naturally-occurring internal file boundaries. The segmenting algorithm groups chunks into segments and may select segment boundaries in a way that is similar to how a chunking algorithm might select chunk boundaries. For example, segment boundaries may be based on size (*e.g.*, 10-megabyte segments) or based on more naturally-occurring or logical chunk boundaries. In alternate examples, the segmentation module 105a may reside locally or partially locally at the clients 102a-b. In these examples, the clients 102 send chunks, hashes, and/or segments, rather than a byte-oriented data stream, to the front-end node 104a.

**[0022]** The front-end node 104a is coupled to a plurality of back-end nodes 108 through one or more networks 106. The network 106 may comprise a high-speed bus known in the art and the back-end nodes 108 may comprise or be coupled to non-volatile storage devices, such as hard disk drives, solid state drives, and the like. The network 106 allows routing of requests, data streams, chunks, segments and the like from the front-end node 104a to one or more of the back-end nodes 108, for example to backup the data stream. One skilled in the art appreciates that a data stream may comprise files, images, virtual tape images, database backups, documents, and the like.

**[0023]** In some examples, the client 102a desires to deduplicate a data stream produced by a backup agent. The client 102a sends the data stream to the segmentation module 105a of the front-end node 104a, which subsequently chunks and segments the data stream. The front-end node 104a computes hashes for each chunk of each segment and, because this example uses the sparse indexing method, computes the hooks for each segment by sampling the hashes of the chunks of each segment. The sampled hashes that make up the hooks are referred to as "hook hashes." Alternately, the client 102a chunks and segments the data (*e.g.*, using a local segmentation module) and computes

hashes for each chunk of each segment and computes the hooks for each segment, similar to above. Alternately, the client 102a chunks and sends the hashes of each chunk to the front-end node 104a and the front-end node 104a performs the segmentation and hook selection. Other divisions of these tasks may be equivalently used.

**[0024]** As explained above, the storage system 101 uses an architecture where each segment of incoming data is deduplicated against only one relatively autonomous back-end node. Under this architecture, it is important that each segment with a significant number of old chunks be deduplicated against a back-end node that contains most of its old chunks. One way of doing this for segments containing old data is to conduct an auction where each back-end node "bids" based on how likely the back-end node thinks it is to deduplicate the segment well.

**[0025]** Each back-end node 108 maintains an associated sparse index 109 that maps the hooks of segments stored at that particular back-end node to indexes containing information about those hooks (not shown). To ensure that an incoming segment is deduplicated against the most appropriate back-end node 108, the front-end node 104a broadcasts the computed hook hashes for each segment of the data stream to the back-end nodes 108.

**[0026]** Although described with reference to deduplicating one segment, in practice segments can be either deduplicated one after the other or with some degree of parallelism (*e.g.*, a pipeline might be used with one or more segments in each stage). Each back-end node 108 returns a bid to the front-end node 104a for the given segment. The bid may consist of the number of hooks of the given segment that are found in that back-end node's sparse index. The greater the bid, the more likely that the given segment will deduplicate well (*e.g.*, require minimal physical copying) against that back-end node.

**[0027]** Then, based on the bids received from the back-end nodes 108, the front-end node 104a sends the remaining hashes of the given segment to the back-end node 108 that generated the highest bid (for simplicity in the following discussion, assume that 108a has generated the high bid). The bid may be the number of hook hashes that occur in the sparse index 109 of that back-end

nodes 108 or may be another indication of whether the given segment is likely to deduplicate well against that back-end node.

**[0028]** If the Bloom filter method were being used in this example architecture, then the sparse indexes 109 would be replaced by a Bloom filter and a full chunk index. Additionally, the hooks might be all the given segment's hashes and the bid might consist of the number of hooks found in the relevant Bloom filter.

**[0029]** The high bidding back-end node (108a) then determines which of the remaining hashes and the hook hashes it does not already have corresponding chunks stored for and informs the front-end node 104a of the hashes for the chunks that are not stored. This determination may occur, for example, by comparing the hashes to one or more indexes found by looking up the hooks in its sparse index 109. The Bloom filter method would be similar, except that the hooks may be looked up in the full chunk index if they are not found in any available cached indexes.

**[0030]** If the front-end 104a has performed the chunking and segmenting, the front-end node 104a sends the chunks that are indicated as not stored to the back-end node 108a, which then stores those chunks. If the client 102a has performed the chunking and segmenting, the front-end node 104a relays the hashes for the chunks that are not stored at the back-end node 108a to the client 102a and the client 102a sends the chunks that are indicated as not stored to the front-end node 104a, which forwards those chunks to the back-end node 108a for storage. Alternatively, the back-end node 108a may communicate directly with client 102a, bypassing the front-end node 104a.

**[0031]** If no non-zero bids are received, the segment is most likely new data, and thus may be deduplicated against, for example, the least-loaded back-end node 108 (*i.e.*, the back-end node having the most available storage space). Additionally, where bids are equal between back-end nodes 108, the segment may be stored at the least-loaded back-end node 108 of the nodes with equal bids. Furthermore, if the bids are equal and the back-end nodes 108 are equally loaded, the segment may be stored at a randomly-selected or rotating back-end node 108 of the back-end nodes with the highest bids.

**[0032]** The above deduplication scheme is an example of how a data stream from a client 102a may be deduplicated against back-end nodes 108. Each segment of the data stream may be routed to (*i.e.*, deduplicated against) a different back-end node 108. However, other data deduplication schemes may be similarly employed and are applicable to examples given in the present disclosure. Additionally, as explained above, reference is made to the client 102a and the front-end node 104a for simplicity; however, the clients 102b, 102c, 102d and the front-end node 104b possess the same or similar functionality as the client 102a and the front-end node 104a.

**[0033]** FIG. 1b shows an example of a portion of the hardware architecture of the front-end node 104a. The front-end node 104a comprises one or more hardware processors 110 coupled to a machine-readable storage device 112. The machine-readable storage device 112 may comprise a hard drive, a compact disc, floppy disc, flash-based storage, or other non-transitory storage device. The machine-readable storage device 112 includes machine-readable instructions 114 that, when executed by the hardware processor 110, cause the hardware processor 110 to carry out some or all of the various functionality and/or methods (*e.g.*, method 300 of FIG. 3) described herein. Similar hardware architecture may be implemented at the front-end node 104b, the clients 102a-102d, and the back-end nodes 108.

**[0034]** In some cases, each client 102 may generate interleaved or multiplexed data to be backed up at the storage system 101 shown in FIG. 1a (*e.g.*, the data stream that results from dumping an SQL Server® database). Effective data deduplication of interleaved data can be problematic when using solely the bid-based approach explained above.

**[0035]** FIG. 2 shows an example of two interleaved data streams backed up on different days. The shaded portions A-F represent, for example, blocks from different underlying data streams being interleaved by a backup process, which may run on, for example, the client 102a. On day 1, the front-end node 104a receives the first data stream (labeled day 1) and chunks and segments blocks A, B, and a portion of C into Segment 1 and the remainder of block C and blocks D, E, and F into Segment 2. In some cases, the storage system 101 may be

unaware of the block boundaries, which may only be known to the backup process.

**[0036]** To exemplify a potential inefficiency of a pure bid-based approach, assume that the front-end node 104a stores Segment 1 on back-end node 108a and stores Segment 2 on back-end node 108b. Subsequently, on day 2, the underlying data to be backed up remains the same or nearly the same (*e.g.*, the underlying data streams being interleaved have not changed, or have changed only minimally). However, the blocks from the various underlying data streams are interleaved by the backup process in an order that is different from day 1, which is a common occurrence in data interleaving or multiplexing processes. As a result, the front-end node 104a receives the blocks and chunks and segments blocks E, A, and a portion of F into Segment 3 and the remainder of block F and blocks D, C, and B into Segment 4. Thus, Segment 3 and Segment 4 each contain data that has been previously stored across both back-end nodes 108. This results in poor data deduplication when storing Segment 3 on either back-end node 108a or 108b, despite the similarity between the blocks stored on day 1 and the blocks stored on day 2, because only a fraction of that segment is found on either back-end node 108. A similar problem exists with respect to Segment 4.

**[0037]** In accordance with various examples, the front-end node 104a employs a “sticky routing” method, which improves subsequent deduplication performance by grouping large amounts of new contiguous interleaved data on each of the various back-end nodes 108. Referring still to FIG. 2, when Segment 1 and Segment 2 are stored on day 1, it is advantageous to ensure that they are stored on the same back-end node, for example back-end node 108a. If Segments 1 and 2 are initially stored on back-end node 108a, then on day 2 when Segment 3 is to be stored, the above bid-based approach would cause the front-end node 104a to deduplicate Segment 3 against back-end node 108a. This results in no additional chunks being physically stored at back-end node 108a because blocks E, A, and a portion of F are already stored on back-end node 108a. Thus, minimal new data is stored at the back-end node 108a on day 2 when the back-end node 108a deduplicates Segment 3 against the back-end node 108a.

**[0038]** To facilitate sticky routing of incoming data streams, the front-end nodes 104a, 104b have values including a "bid threshold" and a "sticky threshold." The bid threshold is an example of a mechanism used to determine whether a segment is likely to deduplicate poorly against every back-end node 108 (*i.e.*, the segment is likely mostly new data). The sticky threshold is an upper limit on an amount of data that is processed before the back-end node 108 currently designated as the sticky node is changed. The front-end nodes 104a, 104b also may maintain data structures 107a, 107b that store a "sticky node" and an "amount processed" value. The sticky node designates the current back-end node 108 against which new data should be deduplicated.

**[0039]** The amount processed in one example may be the amount of data deduplicated against the current sticky node since it most recently became the current sticky node. Alternatively, in another example, the amount processed may be the amount of data physically added to the current sticky node due to data being deduplicated against it since it most recently became the current sticky node. Alternatively, in another example, the amount processed may simply be the amount of data deduplicated against any back-end node since the current sticky node most recently became the current sticky node. Many other variants are possible; for example, the amount processed may only include new mostly-new segments.

**[0040]** When multiple input data streams are being processed at once, there may be one data structure 107a for each input data stream currently being processed by front-end 104a and the amount processed value associated with a given stream may refer only to data that is part of that stream. Thus, different streams may have different sticky nodes, which are switched at different times. Although generally described as operating on only one stream, the system may normally be operated with many streams being processed at a time.

**[0041]** One skilled in the art appreciates that, in general, the amount of data processed provides an indication of how much new data has been recently stored at the back-end node 108 currently designated as the sticky node and thus provides a way to gauge when a new back-end node 108 should be designated as the current sticky node. Other methods of determining when to designate a

new back-end node 108 as the sticky node are within the scope of the present disclosure.

**[0042]** The bid threshold may be determined experimentally, but generally should be a low number that would indicate whether a particular segment is likely to deduplicate poorly against one of the back-end nodes 108. If the bid threshold is 2, for example, then a bid of 1 or 0 from a back-end node 108 for a segment should indicate that the segment is not likely to deduplicate well against that back-end node 108 and a bid of 3 or more should indicate that the segment is likely to deduplicate at least to some degree against the back-end node 108.

**[0043]** Alternately, rather than being an integer value, the bid threshold may be a percentage value (*i.e.*, a percent threshold of the number of chunks being deduplicated). For purposes of simplicity, the case where the bid is equal to the threshold is not discussed as the actual bid threshold may simply be adjusted to provide for the correct outcome.

**[0044]** The sticky threshold should be larger than the maximum likely distance that two initially adjacent blocks from differently underlying streams will move apart in future days. For example, in practice this value may be less than about 10 gigabytes, so the degree to which an initial segment of interleaved data can "smear out" in size in later days is limited. Thus, the data of a future segment comes from a similarly limited section of the original data. If the sticky threshold is set to be larger than this size (*e.g.*, 50 gigabytes), then it is likely that all or most of the data of the future segment will be originally stored at one back-end node. The bidding process explained above ensures that that back-end node is found successfully in most cases. However, even where the segment is spread across two back-end nodes, bidding helps to ensure that deduplication occurs against the back-end node having a larger amount of the segment. In some cases, the sticky threshold may vary based on the load (*i.e.*, available storage space) of one or more of the back-end nodes 108.

**[0045]** The designated sticky node may change over time, for example as the data processed reaches the sticky threshold. At that point, the designated sticky node may be changed and the amount processed reset to zero. Initially, before

data has been stored at the back-end nodes 108, the sticky node may be arbitrarily set.

**[0046]** FIG. 3 shows an illustrative method 300 for sticky routing of a data stream. In some cases, the method 300 may be performed by one or more hardware processors executing machine-readable instructions as shown in FIG. 1b. Additionally, various steps of the method may be performed by the clients, front-end nodes, and back-end nodes of FIG. 1a and may be performed in an order different than that shown. The method 300 begins with receiving information about a plurality of chunks (block 302). The information may be received from, for example, a segmentation module running on a client 102 or a front-end node 104 and may comprise hashes, hooks, or a combination thereof including other chunk identifying information.

**[0047]** The method 300 continues with determining if one or more of a plurality of back-end nodes 108 already stores more than a threshold amount of the plurality of data chunks (block 304). As explained above, this may be determined by broadcasting the hooks associated with the plurality of data chunks to all of the back-end nodes, receiving bids from each of the back-end nodes, and comparing the bids to a bid threshold. The determination may be made in the affirmative if any of the bids exceeds the bid threshold. In particular, a back-end node 108 may be determined to already store more than a threshold amount of the plurality of data chunks if its bid exceeds the bid threshold. The bid threshold may be absolute (*e.g.*, did the bid exceed 2?) or relative (*e.g.*, were at least 5% of the hooks found?).

**[0048]** Each back-end node 108 may generate its bid by accessing one or more indexes or filters. For example, its bid may consist of the number of the broadcast hooks that it finds in its sparse index or Bloom filter. The bid may be weighted, for example, by how much space that back-end node has left.

**[0049]** In another example, the determination may be made by accessing one or more indexes or filters associated with the plurality of back-end nodes 108 without doing a broadcast. For example, copies of the back-end nodes' Bloom filters might be available on each front-end node 106. In some cases, the determining may be done via heuristics and thus may not always be correct. For

example, storage system 101 may incorrectly conclude that a segment is new because the hashes of its old chunks were not chosen as hooks.

**[0050]** If it is determined that at least a threshold amount of the plurality of data chunks is already stored at a back-end node 108 (block 306), then the plurality of data chunks are deduplicated against one of the back-end nodes 108 that has been determined to already store more than the threshold amount of the plurality of data chunks (block 308) and the method 300 ends. There are many ways to select the back-end node 108 of those exceeding the threshold to deduplicate the given segment against. For example, one of the plurality of back-end nodes 108 that has been determined to already store the most of the plurality of data chunks may be chosen. This might be done by choosing one of the back-end nodes 108 returning the highest bid value.

**[0051]** However, if it is determined that none of the plurality of back-end nodes 108 already stores more than the threshold amount of the plurality of data chunks (block 306), then the plurality of data chunks are mostly new chunks that have not been previously stored. Thus, the method 300 continues with deduplicating the plurality of data chunks against the back-end node 108 designated as the sticky node (block 310).

**[0052]** It is desirable for effective load-balancing that not all new data be sent to the same back-end node 108. Thus, the method 300 continues with comparing the amount of data processed value to the sticky threshold (block 312). If the amount of data processed is less than or equal to the sticky threshold, then the current sticky node should store more new data and thus the method ends.

**[0053]** However, if the amount of data processed is greater than the sticky threshold, then a different back-end node 108 is designated as the sticky node (block 314) and the method ends. Other methods responsive to an amount of data being processed than using a simple sticky threshold for determining when to change the designated sticky node may be similarly employed.

**[0054]** Rather than recalculating the current amount of data processed value each time it is needed, the storage system 101 may maintain the current amount of data processed value as data is processed. That is, as data is processed, the amount of data processed value may be updated appropriately.

**[0055]** The current amount of data processed value may be reset to zero when the current designated sticky node is changed; this causes the amount of data processed to comprise only data processed since the currently designated sticky node last became the designated sticky node. In one example, when data is deduplicated against the currently designated sticky node, the size of that data is added to the amount currently processed value. In this case, the amount of data processed comprises only data that has been deduplicated against the currently designated sticky node. It may in fact be the amount of data that has been deduplicated against the currently designated sticky node since the currently designated sticky node last became the designated sticky node.

**[0056]** In another example, when data is deduplicated against the currently designated sticky node, only the size of the portion of that data that has been physically added to the currently designated sticky node is added to the amount of data processed value. In this case, the amount of data processed comprises only data that has been physically added to the currently designated sticky node. It may in fact be the amount of data that has been physically added to the currently designated sticky node since the currently designated sticky node last became the designated sticky node. In another example, when data is deduplicated against any back-end node 108, the size of that data is added to the amount of data processed value. It may in fact be the amount of data that has been deduplicated since the currently designated sticky node last became the designated sticky node.

**[0057]** There may be multiple back-end nodes beyond the two shown in FIG. 1a. In such a situation, when selecting a new back-end node to be the currently designated sticky node, a choice must be made. The least-loaded back-end node (*e.g.*, the back-end node having the most available storage space) may be selected, or, if there are a number of least-loaded back-end nodes, the updated sticky node may be selected at random from the least loaded back-end nodes. Alternatively, back-end nodes 108 may be selected in a round robin manner, where the back-end node least-recently designated as the sticky node is chosen next. Many other methods exist; generally speaking, the more lightly loaded and least recently used backend nodes should be preferred.

**[0058]** When multiple data streams are being processed at once (*i.e.*, multiple instances of method 300 occurring in parallel), certain additional considerations come into play. In particular, it is useful to load balance across back-end nodes 108 so that not all streams are sending new data to the same back-end node 108 at once. One way to do this is to "penalize" nodes that are currently the designated sticky node for some other data stream; the penalty may depend on how many such streams that node is the currently designated sticky node for.

**[0059]** For example, an effective available storage space could be computed for each back-end node 108 by subtracting from its total available storage space both its storage space already used and the amount of storage space "reserved" on it by existing data streams, where a data stream "reserves" on its currently designated sticky node an amount of space equal to the sticky threshold minus its amount processed value. The storage system 101 might choose the next back-end node 108 to use as a sticky node by choosing among the back-end nodes 108 with the highest effective available storage space and breaking ties by least recently used. In some examples, the same back-end node 108 might be chosen as the new designated sticky node as the old designated sticky node if it is the node with by far the most available space.

**[0060]** The above method 300 enables large contiguous pieces of new data to be stored, each at a single back-end node. In the case where the data being stored is interleaved, the limited reordering of blocks during subsequent data interleaving (*e.g.*, on subsequent days) has a reduced impact on the efficiency of data deduplication because the block portions contained in a segment are likely to be found at a single back-end node 108. More likely, the storage system 101 is likely to determine that the back-end node is the best back-end node to deduplicate that segment against, resulting in enhanced deduplication performance.

**[0061]** As explained above, the data deduplication system may be expanded to include multiple back-end nodes while effectively distributing interleaved data to improve deduplication performance, thereby minimizing the amount of storage space consumed during data backup procedures. Furthermore, bid-based schemes for distributing segments that are not new may be employed in a

manner that is consistent with single back-end node data deduplication and backup systems.

**[0062]** The above discussion is meant to be illustrative of the principles and various examples of the present invention. Numerous variations and modifications will become apparent to those skilled in the art once the above disclosure is fully appreciated. For example, determining whether data is stored at a back-end node may comprise determining whether the data is actually present at the back-end node or, for example in a process using a Bloom filter, which may return a false positive, determining whether it is likely that the data is present at the back-end node. Other methods of determining whether data is or is likely to be stored at a back-end node are apparent to one skilled in the art. It is intended that the following claims be interpreted to embrace all such variations and modifications.

# CLAIMS

What is claimed is:

1. A method, comprising:

receiving information about a plurality of data chunks;

determining if one or more of a plurality of back-end nodes already stores more than a threshold amount of the plurality of data chunks, one of the plurality of back-end nodes designated as a sticky node;

responsive to determining that none of the plurality of back-end nodes already stores more than a threshold amount of the plurality of data chunks, deduplicating the plurality of data chunks against the back-end node designated as the sticky node; and

responsive to an amount of data being processed, designating a different back-end node as the sticky node.

2. The method of claim 1 further comprising accessing one or more indexes or filters associated with the plurality of back-end nodes to determine if one or more of the plurality of back-end nodes already stores more than a threshold amount of the plurality of data chunks.

3. The method of claim 1 further comprising, responsive to an amount of data processed by the sticky node being greater than a sticky threshold, designating a different back-end node as the sticky node.

4. The method of claim 1 wherein designating a different back-end node as the sticky node further comprises designating the back-end node having the greatest available storage space as the sticky node or designating as the sticky node the back-end node least-recently designated as the sticky-node.

5. The method of claim 1 further comprising, responsive to an amount of data processed being greater than a sticky threshold and more than one back-end node having the greatest available storage space, designating one of the back-end nodes having the greatest available storage space as the sticky node.

6. The method of claim 1 further comprising, responsive to determining that one or more of the plurality of back-end nodes already stores more than a threshold amount of the plurality of data chunks, deduplicating the plurality of data chunks against one of the plurality of back-end nodes that has been determined to already store more than the threshold amount of the plurality of data chunks.

7. The method of claim 6 further comprising, responsive to determining that one or more of the plurality of back-end nodes already stores more than a threshold amount of the plurality of data chunks, deduplicating the plurality of data chunks against one of the plurality of back-end nodes that has been determined to store the most of the plurality of data chunks of the plurality of back-end nodes.

8. The method of claim 1 wherein the amount of data processed comprises only data processed since the currently designated sticky node last became the designated sticky node.

9. The method of claim 1 wherein the amount of data processed comprises only data that has been deduplicated against the currently designated sticky node.

10. The method of claim 1 wherein the amount of data processed comprises only data that has been physically added to the currently designated sticky node.

11. A system, comprising:
a front-end node; and
a plurality of back-end nodes coupled to the front-end node by way of a bus;
the front-end node to:
receive information about a plurality of data chunks;
determine if one or more of a plurality of back-end nodes already stores more than a threshold amount of the plurality of data

chunks, one of the plurality of back-end nodes designated as a sticky node;

responsive to a determination that none of the plurality of back-end nodes already stores more than a threshold amount of the plurality of data chunks, deduplicate the plurality of data chunks against the back-end node designated as the sticky node; and

responsive to an amount of data being processed, designate a different back-end node as the sticky node.

12. The system of claim 11 wherein when the front-end node designates a different back-end node as the sticky node, the front-end node designates the back-end node having the greatest available storage space as the sticky node or designates as the sticky node the back-end node least-recently designated as the sticky-node.

13. The system of claim 11 wherein the front-end node is configured to, responsive to an amount of data processed being greater than a sticky threshold and more than one back-end node having the greatest available storage space, designate one of the back-end nodes having the greatest available storage space as the sticky node.

14. The system of claim 11 wherein the front-end node is configured to, responsive to a determination that one or more of the plurality of back-end nodes already stores more than a threshold amount of the plurality of data chunks, deduplicate the plurality of data chunks against one of the plurality of back-end nodes that has been determined to already store more than the threshold amount of the plurality of data chunks.

15. A machine-readable storage device containing machine-readable instructions that, when executed by one or more hardware processors, cause the one or more hardware processors to:

receive information about a plurality of data chunks;

determine if one or more of a plurality of back-end nodes already stores more than a threshold amount of the plurality of data chunks, one of the plurality of back-end nodes designated as a sticky node;

responsive to a determination that none of the plurality of back-end nodes already stores more than a threshold amount of the plurality of data chunks, deduplicate the plurality of data chunks against the back-end node designated as the sticky node; and

responsive to an amount of data being processed, designate a different back-end node as the sticky node.

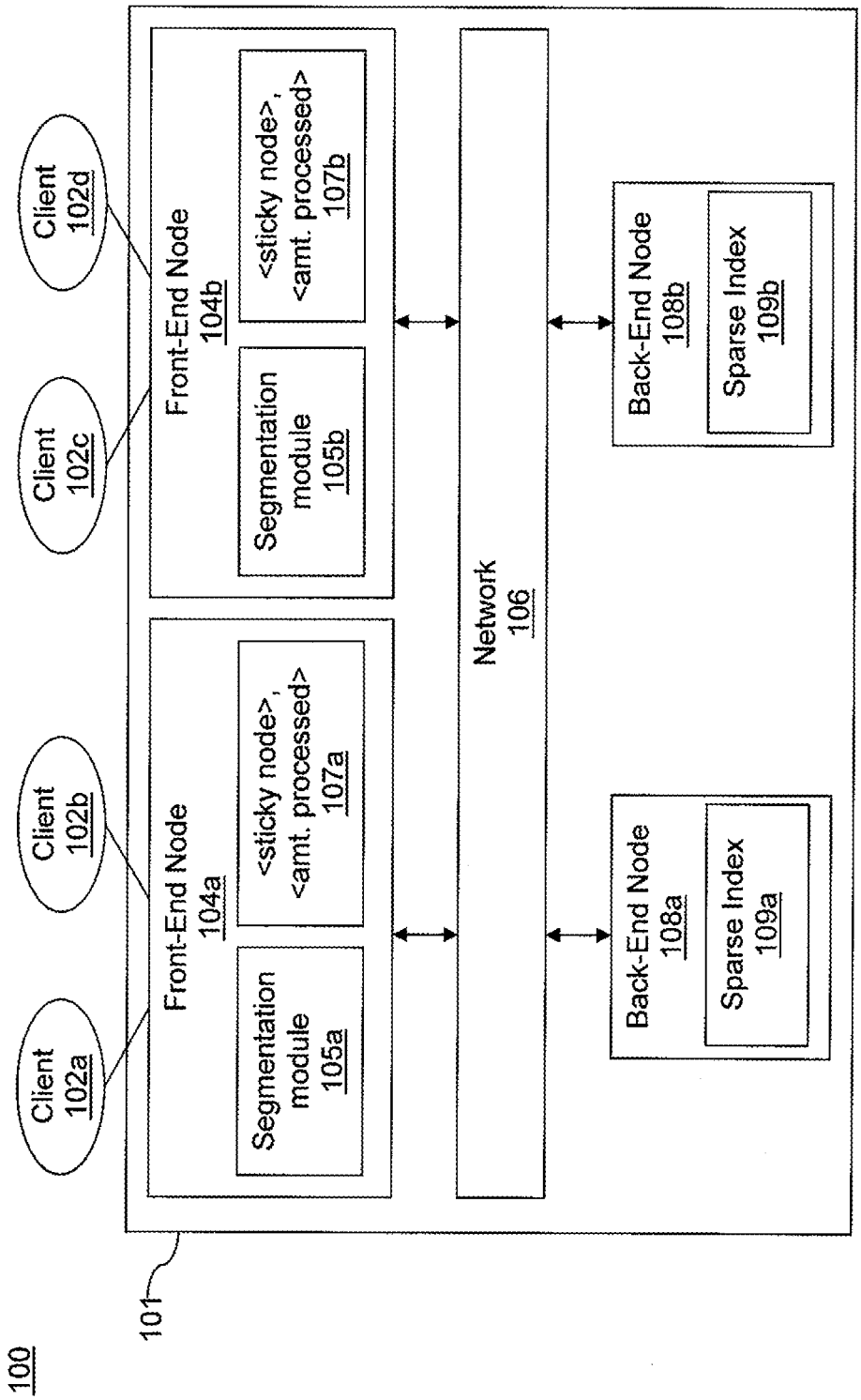

100
101
Client 102a
Client 102b
Client 102c
Client 102d
Front-End Node 104a
Segmentation module 105a
<sticky node>, <amt. processed> 107a
Front-End Node 104b
Segmentation module 105b
<sticky node>, <amt. processed> 107b
Network 106
Back-End Node 108a
Sparse Index 109a
Back-End Node 108b
Sparse Index 109b

FIG. 1a

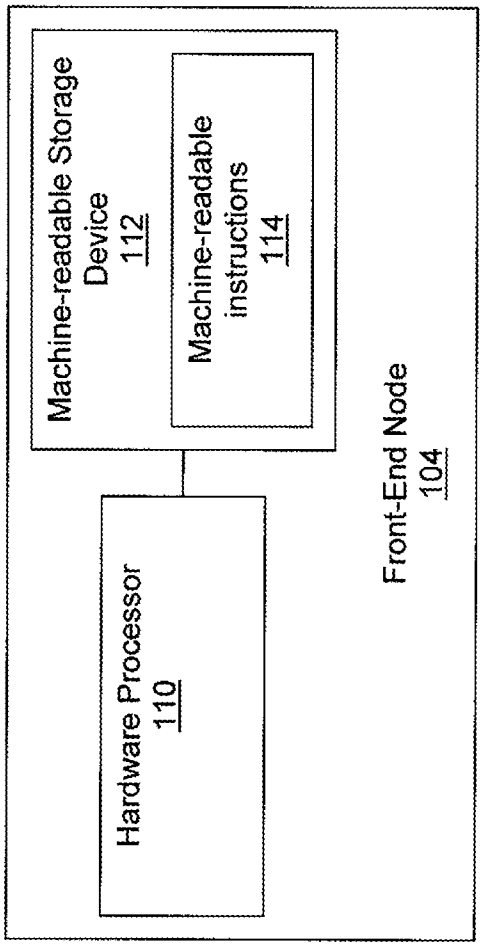

Front-End Node
104
Hardware Processor
110
Machine-readable Storage
Device
112
Machine-readable
instructions
114

FIG. 1b

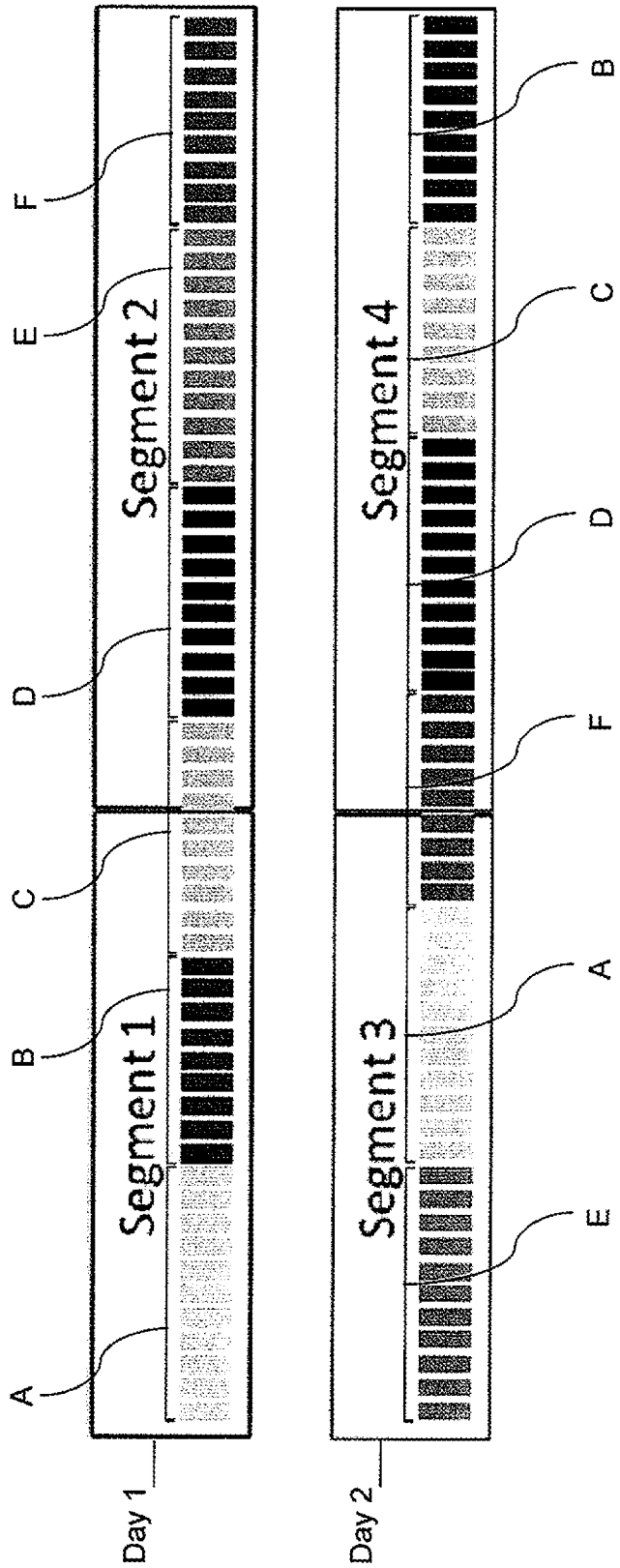

Day 1
Segment 1
Segment 2
A
B
C
D
E
F
Day 2
Segment 3
Segment 4
A
B
C
D
E
F

FIG. 2

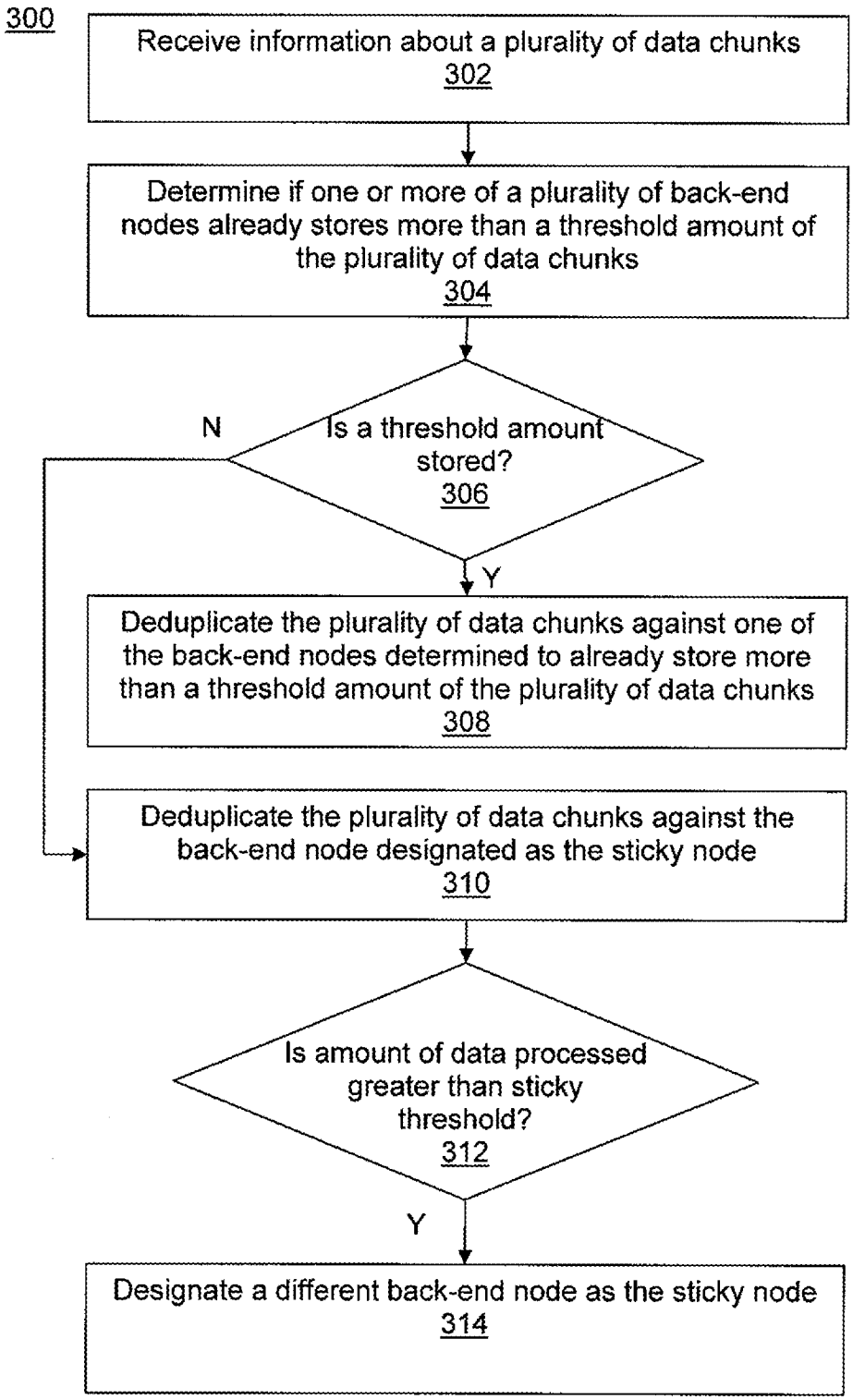

300
Receive information about a plurality of data chunks
302
Determine if one or more of a plurality of back-end nodes already stores more than a threshold amount of the plurality of data chunks
304
Is a threshold amount stored?
306
N
Y
Deduplicate the plurality of data chunks against one of the back-end nodes determined to already store more than a threshold amount of the plurality of data chunks
308
Deduplicate the plurality of data chunks against the back-end node designated as the sticky node
310
Is amount of data processed greater than sticky threshold?
312
Y
Designate a different back-end node as the sticky node
314

FIG. 3

**INTERNATIONAL SEARCH REPORT**

International application No.
**PCT/US2012/023649**

**A. CLASSIFICATION OF SUBJECT MATTER**

***G06F 12/00(2006.01)i***

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)

G06F 12/00; G06F 12/02; G06F 17/30; G06F 12/10

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Korean utility models and applications for utility models

Japanese utility models and applications for utility models

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

eKOMPASS(KIPO internal) & Keywords: data chunk, front-end, back-end, deduplicate, sticky node, threshold

**C. DOCUMENTS CONSIDERED TO BE RELEVANT**

| Category* | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| A | US 2010-0223441 A1 (LILLIBRIDGE, MARK DAVID et al.) 02 September 2010<br>See paragraphs [0029]-[0036]; figure 3. | 1-15 |
| A | US 2010-0235332 A1 (HAUSTEIN, NILS et al.) 16 September 2010<br>See paragraphs [0029]-[0047]; figures 3, 4. | 1-15 |
| A | US 2009-0132619 A1 (ARAKAWA, HIROSHI et al.) 21 May 2009<br>See paragraphs [0009]-[0015]. | 1-15 |
| A | US 2011-0246741 A1 (RAYMOND, ROBERT MICHAEL et al.) 06 October 2011<br>See paragraphs [0046]-[0049]. | 1-15 |

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

☒ See patent family annex.

* 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 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 |
|---|---|
| 22 OCTOBER 2012 (22.10.2012) | **23 OCTOBER 2012 (23.10.2012)** |


Name and mailing address of the ISA/KR

Korean Intellectual Property Office
189 Cheongsa-ro, Seo-gu, Daejeon Metropolitan City, 302-701, Republic of Korea

Facsimile No. 82-42-472-7140

Authorized officer

JANG Ho Keun

Telephone No. 82-42-481-8187

**INTERNATIONAL SEARCH REPORT**

Information on patent family members

International application No.

**PCT/US2012/023649**

| Patent document cited in search report | Publication date | Patent family member(s) | Publication date |
|---|---|---|---|
| US 2010-0223441 A1 | 02.09.2010 | CN 101855619 A | 06.10.2010 |
| | | US 2010-0198792 A1 | 05.08.2010 |
| | | WO 2009-054827 A1 | 30.04.2009 |
| US 2010-0235332 A1 | 16.09.2010 | None | |
| US 2009-0132619 A1 | 21.05.2009 | EP 2063351 A2 | 27.05.2009 |
| | | EP 2063351 A3 | 02.11.2011 |
| | | JP 2009-129441 A | 11.06.2009 |
| | | US 7870105 B2 | 11.01.2011 |
| US 2011-0246741 A1 | 06.10.2011 | None | |