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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
05 April 2018 (05.04.2018)

WIPO | PCT

(10) International Publication Number
WO 2018/064319 A1

(51) International Patent Classification:
G06F 17/30 (2006.01) *G06F 12/0862* (2016.01)

(21) International Application Number:
PCT/US2017/053990

(22) International Filing Date:
28 September 2017 (28.09.2017)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
15/279,694 29 September 2016 (29.09.2016) US
15/279,721 29 September 2016 (29.09.2016) US

(71) Applicant: VERITAS TECHNOLOGIES LLC [US/US]; 500 East Middlefield Road, Mountain View, CA 94043 (US).

(72) Inventors: PATEL, Bhautik; 2455, Ayodhya Nagar, Link Road, Bharuch, Gujurat 392001 (IN). JAMES, Freddy; C/o A F James, Aloor House, Thrissur, Kerala 680581 (IN). KOTHARI, Mitul; 2/1 Mahesh Nagar, Indore, Madhya Pradesh 452002 (IN). BANERJEE, Anindya; A2-603, Rhythm, Kalewadi Phata, Pune, Maharashtra 411057 (IN).

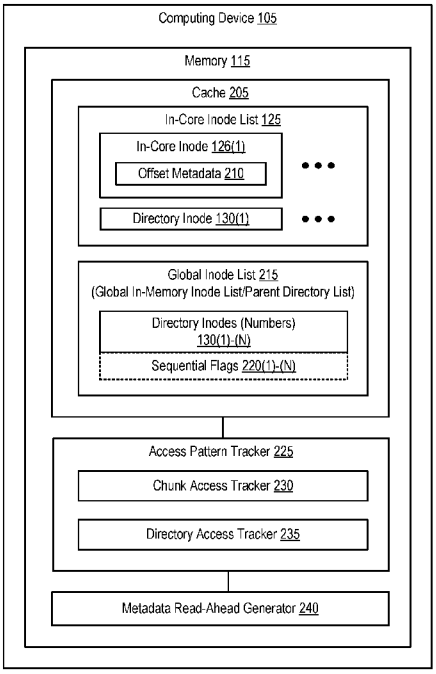
(74) Agent: ANAND, Ashwin; Campbell Stephenson LLP, 11401 Century Oaks Terrace, Building H, Suite 250, Austin, TX 78758 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ,

(54) Title: TRACKING ACCESS PATTERN OF INODES AND PRE-FETCHING INODES

200A

FIG. 2A



(57) Abstract: Disclosed herein are methods, systems, and processes to track access patterns of inodes, and to issue read-ahead instructions to pre-fetch inodes into memory. A location of a unit of metadata in a metadata storage area is determined. Another location in the metadata storage area that corresponds to a current metadata read operation is determined. Whether a metadata read-ahead operation can be performed is determined using the location of the unit of metadata and the another location. In response to a determination that the metadata read-ahead operation can be performed, the metadata-ahead operation is issued. In addition, an inode is accessed and a directory of the inode is determined. Also determined is whether an entry for the directory exists in a global inode list. If the entry exists in the global inode list, whether a file structure of the directory is sequential or non-sequential is determined. If the entry does not exist in the global inode list, a new entry for the directory is added in the global inode list.

WO 2018/064319 A1

[Continued on next page]

OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

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

Published:

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

TRACKING ACCESS PATTERN OF INODES AND PRE-FETCHING INODES

Bhautik Patel
Freddy James
Mitul Kothari
Anindya Banerjee

FIELD OF THE DISCLOSURE

[0001] This disclosure relates to data access. In particular, this disclosure relates to tracking access patterns of inodes, and issuing inode read-ahead instructions to pre-fetch inodes.

DESCRIPTION OF THE RELATED ART

[0002] A file system is used to control how data is stored and retrieved for computing purposes (e.g., for storing and executing applications). A data object in a file system (e.g., a file, a directory, or the like) has one or more inodes. An inode is a data structure that is used to identify data belonging to the data object in the file system. The inode stores attributes (e.g., metadata) and disk block location(s) of the data object's data.

[0003] Accessing a file in a file system requires the file's inode to be read from disk (e.g., from a non-volatile storage unit). Data operations such as backup, periodic scans, administrative operations, and the like, typically access multiple inodes on disk. Reading such "on-disk" inodes from disk can negatively impact application performance. For example, if the underlying disk is slow, reading on-disk inodes from disk can result in unreasonable and/or significant input/output (I/O) wait time before an application can be serviced with the required data.

[0004] A file's contents can be loaded into memory (e.g., Random Access Memory (RAM)) such that when the file is subsequently accessed, the file's contents are read from RAM rather than from disk (e.g., a Hard Disk Drive (HDD)). However, loading a file's contents into memory requires inodes that correspond to the file's contents to be accessed from disk.

SUMMARY OF THE DISCLOSURE

[0005] Disclosed herein are methods, systems, and processes to track access patterns of inodes to accelerate data access by pre-fetching inodes into memory. One such method involves determining a location of a unit of metadata in a metadata storage area that is in a non-volatile storage unit. The method determines another location in the metadata storage area that corresponds to a current metadata read operation, and determines whether a metadata read-ahead operation can be performed using the location of the unit of metadata and the another location. In response to a determination that the metadata read-ahead operation can be performed, the method issues the metadata read-ahead operation.

[0006] In certain embodiments, the metadata storage area includes on-disk inodes. The on-disk inodes include the unit of metadata and are part of a structural file. The structural file is stored in the non-volatile storage unit. The method creates an in-core inode that corresponds to the structural file and stores the location of the unit of metadata in the in-core inode. In this example, the unit of metadata is a last-read chunk of metadata, and the location of the unit of metadata identifies and includes an end offset of the unit of metadata.

[0007] In some embodiments, determining whether the metadata read-ahead operation can be performed includes accessing the end offset of the unit of metadata and determining whether the another location is adjacent to the end offset. In this example, issuing the metadata read-ahead operation includes updating the end offset by replacing the end offset with another end offset of another unit of metadata (e.g., another last-read chunk of metadata) read by the metadata read-ahead operation, and updating the end offset by replacing the end offset with an end offset of a current metadata read operation if the another location is not adjacent to the end offset.

[0008] In other embodiments, the method intercepts a command to read one or more on-disk inodes in response to an input/output (I/O) operation. In this example, the method analyzes an issued metadata read-ahead value in the metadata read-ahead operation by comparing the issued read-ahead value and a chunk total in the command. The method waits for the I/O operation to complete and/or issues an asynchronous metadata read-ahead instruction.

[0009] In certain embodiments, determining whether the I/O operation is complete includes generating a queue if the I/O operation is complete. The queue includes remaining chunks of metadata of the chunk total not included in the asynchronous metadata read-ahead instruction. The method updates the chunk total in the metadata read-ahead operation if the I/O operation is incomplete.

[0010] Some embodiments involve accessing an inode, determining a directory of the inode (e.g., an on-disk inode), and determining whether an entry for the directory exists in a global inode list. If the entry exists in the global inode list, the method determines whether a file structure of the directory is sequential or non-sequential, and if the entry does not exist in the global inode list, the method adds a new entry for the directory in the global inode list.

[0011] In certain embodiments, determining whether the directory exists in a global inode list includes searching the global inode list for the entry. The method determines whether the directory includes a sequential flag if the file structure of the directory is sequential. If the directory includes the sequential flag, the method issues a metadata read-ahead operation for inodes in the directory. If the file structure of the parent directory is non-sequential, the method fetches the inode list of the directory and issues the metadata read-ahead operation for inodes on the inode list. In some embodiments, the method intercepts a command to read one or more on-disk inodes in response to an input/output (I/O) operation. In this example, the method analyzes an issued metadata read-ahead value in the metadata read-ahead operation by comparing the issued metadata read-ahead value and a chunk total in the command. The method waits for the I/O operation to complete and/or issues an asynchronous metadata read-ahead instruction.

[0012] In other embodiments, determining whether the I/O operation is complete includes generating a queue if the I/O operation is complete. The queue includes remaining chunks of metadata of the chunk total not included in the asynchronous metadata read-ahead instruction. The method updates the chunk total in the metadata read-ahead operation if the I/O operation is incomplete.

[0013] The foregoing is a summary and thus contains, by necessity, simplifications, generalizations and omissions of detail; consequently those skilled in the art will appreciate that the summary is illustrative only and is not intended to be in any limiting. Other aspects, inventive features, and advantages of the present disclosure, as defined solely by the claims, will become apparent in the non-limiting detailed description set forth below.

BRIEF DESCRIPTION OF THE DRAWINGS

[0014] The present disclosure may be better understood, and its numerous objects, features and advantages made apparent to those skilled in the art by referencing the accompanying drawings.

[0015] FIG. 1A is a block diagram of a system that tracks the access pattern of inodes and pre-fetches inodes, according to one embodiment of the present disclosure.

[0016] FIG. 1B is a block diagram of a structural file, according to one embodiment of the present disclosure.

[0017] FIG. 1C is a block diagram of a structural file and an in-core inode, according to one embodiment of the present disclosure.

[0018] FIG. 2A is a block diagram of a system that tracks access patterns of inodes and issues metadata read-ahead instructions, according to one embodiment of the present disclosure.

[0019] FIG. 2B is a block diagram of offset metadata of inodes, according to one embodiment of the present disclosure.

[0020] FIG. 2C is a table illustrating the contents of a global inode list/parent directory list, according to one embodiment of the present disclosure.

[0021] FIG. 2D is a block diagram of a directory access tracker, according to one embodiment of the present disclosure.

[0022] FIG. 2E is a block diagram of a metadata read-ahead generator, according to one embodiment of the present disclosure.

[0023] FIG. 3A is a block diagram of a directory with sequential inodes, according to one embodiment of the present disclosure.

[0024] FIG. 3B is a block diagram of a directory with non-sequential inodes, according to one embodiment of the present disclosure.

[0025] FIG. 3C is a block diagram of a cache that implements a global inode list, according to one embodiment of the present disclosure.

[0026] FIG. 4A is a flowchart that illustrates a process for performing inode pre-fetching, according to one embodiment of the present disclosure.

[0027] FIG. 4B is a flowchart that illustrates a process for storing offset metadata associated with on-disk inodes, according to one embodiment of the present disclosure.

[0028] FIG. 5A is a flowchart that illustrates a process for determining the file structure of a directory, according to one embodiment of the present disclosure.

[0029] FIG. 5B is a flowchart that illustrates a process for issuing a metadata read-ahead instruction for on-disk inodes, according to one embodiment of the present disclosure.

[0030] FIG. 6A is a flowchart that illustrates a process for processing input/output (I/O) operations related to inode pre-fetching, according to one embodiment of the present disclosure.

[0031] FIG. 6B is a flowchart that illustrates a process for processing I/O operations, related to inode pre-fetching, according to one embodiment of the present disclosure.

[0032] FIG. 7A is a flowchart that illustrates a process for processing access of on-disk inodes, according to one embodiment of the present disclosure.

[0033] FIG. 7B is a flowchart that illustrates a process for processing a request to access on-disk inodes, according to one embodiment of the present disclosure.

[0034] FIG. 8 is a block diagram of a computing system, illustrating how an access pattern tracker and a metadata read-ahead generator can be implemented in software, according to one embodiment of the present disclosure.

[0035] FIG. 9 is a block diagram of a networked system, illustrating how various computing devices can communicate via a network, according to one embodiment of the present disclosure.

[0036] While the disclosure is susceptible to various modifications and alternative forms, specific embodiments of the disclosure are provided as examples in the drawings and detailed description. It should be understood that the drawings and detailed description are not intended to limit the disclosure to the particular form disclosed. Instead, the intention is to cover all modifications, equivalents and alternatives falling within the spirit and scope of the disclosure as defined by the appended claims.

DETAILED DESCRIPTION

Introduction

[0037] A file system (e.g., a Unix file system) is used to organize data and control how that data is stored and retrieved. A file system is responsible for organizing data objects such as files and directories, and for keeping track of which areas of a storage device (e.g., a Hard Disk Drive (HDD), a Solid State Drive (SSD), and/or the like) belong to which data objects. Typically, each data object in a file system (e.g., a file, a directory, or the like) has a corresponding inode.

[0038] An inode is a data structure used to locate data in a file system. An inode stores attributes (e.g., metadata) and disk block location(s) of the data object's data, and can be identified by an integer number (e.g., called an inode number). Directories can include lists of names assigned to inodes. A directory contains an entry for itself, an entry for the directory's parent, and entries for each of the directory's children.

[0039] Accessing a file in a file system typically involves the file's inode to be read from disk (e.g., from a non-volatile storage unit), for example, to determine changes and/or modifications to the file's contents, and in certain cases, also to verify owner and permission information (e.g., group-id, user-id, permissions, and the like). Therefore, before a file's contents can be accessed, the file's inode (metadata) has to be first read from disk.

[0040] Data operations (e.g., input/output (I/O) operations) typically require the access of multiple inodes on disk, and reading such "on-disk" inodes from disk can negatively impact application performance (e.g., in the form of I/O wait time before a given I/O operation can be completed). Also as previously noted, a file's contents can be "read-ahead" and loaded into memory (e.g., Random Access Memory (RAM)) such that when the file is subsequently accessed, the file's contents are read from RAM rather than from disk (e.g., HDD). Therefore, pre-fetching data in this manner (e.g., to accelerate data access) requires tracking of access pattern(s) of inodes to "read-ahead" these inodes, before data associated with those inodes (e.g., files, directories, and the like) can be preemptively loaded into memory.

[0041] Unfortunately, unlike file data, the efficient tracking of inode metadata access pattern(s) is challenging because of at least two reasons. First, multiple I/O operations (e.g., from multiple applications executing in a cluster) can access the same inode simultaneously. Tracking the inode access pattern(s) of multiple I/O operations can be memory and computing resource intensive, and can result in significant overhead. Second, the tracking of inode access pattern(s) also requires the efficient "read-ahead" of such inodes (e.g., by issuing read-ahead instructions), also without negatively impacting system performance.

[0042] Disclosed herein are methods, systems, and processes capable of tracking access patterns of inodes based on chunk access, sequential access, and non-sequential access, and issuing read-ahead instructions for inodes, among other capabilities.

Example System That Tracks Inode Access Pattern(s) and Issues Read-Ahead

[0043] FIG. 1A is a block diagram of a computing system 100A that is configured to track the access pattern(s) of inodes and pre-fetch inodes, according to one embodiment. As shown in FIG. 1A, computing device 105 includes a processor 110 and a memory 115. Computing device 105 can be any type of computing system including a server, a desktop, a laptop, a tablet, and the like, and is communicatively coupled to storage system 145 via network 185. Network 185 can be any type of network and/or interconnection (e.g., a Local Area Network (LAN), a Wide Area Network (WAN), a Storage Area Network (SAN), the Internet, and the like).

[0044] Storage system 145 can include a variety of different storage devices, including hard disks, compact discs, digital versatile discs, SSD memory such as Flash memory, and the like, or one or more logical storage devices such as volumes implemented on one or more such physical storage devices. Storage system 145 includes one or more of such storage devices (e.g., disk 150). In one embodiment, disk 150 is a non-volatile storage unit. In other embodiments, disk 150 is a HDD or a SSD. Disk 150 includes file system 155. File system 155 can be any type of file system (e.g., a Unix file system, an extent-based file system, and the like).

[0045] An operating system 120, and applications 135 and 140 are stored in memory 115, and executed by operating system 120. Operating system 120 further includes an in-core inode list 125 (e.g., with multiple in-core inodes 126(1)-(N)), including directory inodes 130(1)-(N). Directory inodes 130(1)-(N) are parent directory representations of on-disk inodes in in-core inode list 125, for example (which includes in-core inodes for files, directories, and the like). File system 155 includes a structural file 160, a directory data structure 170, and data 180. Structural file 160 includes on-disk inodes 165(1)-(N), and directory data structure 170 includes an inode list 175. In one embodiment, structural file 160 is an iList file. In this embodiment, the iList file is a file that maintains a listing of on-disk inodes (e.g., on-disk inodes 165(1)-(N)).

[0046] Data objects (e.g., a file, a directory, and/or the like) in file system 155 are associated with corresponding inodes (e.g., on-disk inodes 165(1)-(N)). Each on-disk inode has a specific inode number and is stored on disk 150 (e.g., as on-disk inodes 165(1)-(N), and as part of structural file 160). In-core inode list 125 is an in-memory data structure (or list) for one (or more) on-disk inodes. In-core inode list 125 includes the metadata that is stored as part of on-disk inodes 165(1)-(N), as well as other additional metadata.

[0047] File system 155 includes directory data structure 170. Directory data structure 170 includes inode list 175. The parent of a data object (e.g., a file, a sub-directory, or the like), is a directory of which the given data object is part of. For example, if a given directory (e.g., /home/john) contains four data objects (e.g., file1, file2, file3, and dir1 with pathnames /home/john/file1, /home/john/file2, /home/john/file3, and /home/john/dir1, respectively), then the parent directory of these directory entries (e.g., file1, file2, file3, and dir1) is "john." Each data object also has a corresponding and/or associated on-disk inode (e.g., on-disk inode 165(1) for "john," on-disk inode 165(2) for "file1," on-disk inode 165(3) for "file2," and on-disk inode 165(4) for "file3," and on-disk inode 165(5) for "dir1"). In this scenario, the parent (directory) inode number of on-disk inodes 165(2), 165(3), 165(4), and 165(5) is on-disk inode 165(1) (shown as directory inode 130(1) in in-core inode list 125 for clarity).

[0048] FIG. 1B is a block diagram 100B of a structural file, and FIG. 1C is a block diagram 100C of structural files and in-core inodes, according to some embodiments. A directory in file system 155 includes a list of tuples (e.g., <file name, inode number of the file>). Inode list 175 is one example of such a list of tuples in file system 155. The data portion of on-disk inode 165(1) includes <file1, on-disk inode 165(2)>, <file2, on-disk inode 165(3)>, <file3, on-disk inode 165(4)>, and <dir1, on-disk inode 165(5)>. As shown in FIG. 1B, these "on-disk" inodes are maintained as data of structural file 160 (e.g., an iList file). Because structural file 160 is also a file, structural file 160 also has its own inode with a unique inode number. As shown in FIG. 1C, when structural file 160(1) is brought in-core (e.g., into memory 115), an in-core inode is created for structural file 160(1) (e.g., in-core inode 126(1) in an in-core inode list such as in-core inode list 125). Each structural file and on-disk inode has an associated in-core inode.

[0049] FIG. 2A is a block diagram of a computing system 200A that tracks access patterns of inodes and issues metadata read-ahead instructions, according to one embodiment. As shown in FIG. 2A, computing device 105 includes memory 115. Memory 115 implements a cache 205, an access pattern tracker 225, and a metadata read-ahead generator 240. Cache 205 implements global inode list 215, which includes one or more entries from in-core inode list. In-core inode 126(1) is an in-memory data structure (or list) that is created for structural file 160(1).

[0050] In-core inode 126(1) includes offset metadata 210. Global inode list 215 is a global in-memory inode list. Offset metadata 210 includes location information of on-disk inode access patterns, and global inode list 215 (also called parent directory list) includes directory inodes 130(1)-(N) (e.g., in-core parent directory inode numbers of on-disk inodes) and sequential flags 220(1)-(N). Sequential flags 220(1)-(N) can be stored as part of in-core inodes for parent directories (e.g., parent directories identified in memory by in-core parent directory inode numbers and shown as directory inodes 130(1)-(N) in FIG. 2A). Access pattern tracker 225 includes a chunk access tracker 230 (e.g., to track chunk access of on-disk inodes), and a directory access tracker (e.g., to track sequential and non-sequential access of on-disk inodes in directories). The foregoing elements of FIG. 2A are described in greater detail in connection with FIGs. 2B – 2E.

Example of Tracking Chunk Access Pattern of Inodes

[0051] FIG. 2B is a block diagram 200B of offset metadata, according to one embodiment. Offset metadata 210 includes an end offset of a last read metadata chunk 245 and a starting offset of a current metadata read operation 250. Because file system 155 persistently stores in-core inodes 165(1)-(N) on disk 150, when a particular inode is not found in-core (e.g., in cache 205), chunk access tracker 230, which is part of access pattern tracker 225, reads on-disk inodes in structural file 160 from disk 150 in chunks (e.g., 1 KB, 2 KB, 4 KB, or other appropriate size). In this manner, chunk access tracker 230 can be configured to track access patterns of chunks of metadata (e.g., on-disk inodes), and facilitate a determination as to whether an application (e.g., application 135 or application 140) is accessing on-disk inodes in a sequential (or nearly-sequential) manner.

[0052] In one embodiment, chunk access tracker 230 determines a location of a unit of metadata (e.g., the location/end offset of a 1 KB chunk of on-disk inodes 165(1)-(4) as shown in FIG. 1B), in a metadata storage area (e.g., in structural file 160). Chunk access tracker 230 determines another location (e.g., a starting offset) in the metadata storage area (e.g., in structural file 160) that corresponds to a current metadata read operation. Metadata read-ahead generator 240 then determines whether a metadata read-ahead operation is needed using the location of the data chunk and the another location that corresponds to the current metadata read operation. If the metadata read-ahead operation is needed, metadata read-ahead generator 240 issues the metadata read-ahead operation.

[0053] Chunk access tracker 230 maintains the end offset of the last read metadata chunk 245 from disk 150 in in-core inode 126(1) associated with structural file 160(1) (e.g., the end of a logical offset in structural file 160(1)). For example, if application 135 and/or application 140 attempt to access data 180 in disk 150 that causes (and requires) the reading of a 1 KB chunk of on-disk inodes, then chunk access tracker 230 stores the end offset of the 1 KB chunk of on-disk inodes (e.g., the end offset of the 1 KB chunk of on-disk inode 165(1)-(4) as shown in FIG. 1B) in in-core inode 126(1) of structural file 160(1) as a "stored value."

[0054] In some embodiments, if the starting offset of the current metadata read operation 250 (e.g., on-disk inode 165(5) as shown in FIG. 1B) is adjacent to the stored value (e.g., the end offset of on-disk inodes 165(1)-(4) – the end offset of the last read metadata chunk 245), metadata read-ahead generator 240 issues a metadata read-ahead instruction to fetch the on-disk inodes from 1 KB to 2 KB in structural file 160 (e.g., on-disk inodes 165(5)-(8) as shown in FIG. 1B) into cache 205. Because chunk access tracker 230 determines that on-disk inode access is happening in a sequential manner, metadata read-ahead generator 240 issues a metadata read-ahead instruction to fetch the next 1 KB chunk of on-disk inodes from disk 150 to in-core inode list 125 (e.g., 1 KB to 2 KB), thus accelerating (future) inode access.

[0055] If the metadata read-ahead instruction described above is triggered (e.g., if the starting offset of the current metadata read operation 250 is next to the end offset of the last read metadata chunk 245), then access pattern tracker 225 updates the stored value (e.g., the end offset of on-disk inodes 165(1)-(4) – the end offset of the last read metadata chunk 245) in in-core inode list 125 by replacing the stored value in cache 205 with another end offset of another last read metadata chunk, read by the metadata read-ahead operation (e.g., the end offset of on-disk inodes 165(5)-(8) (e.g., at 2 KB) as shown in FIG. 1B, because the metadata read-ahead operation reads ahead on-disk inodes from 1 KB to 2 KB as a result of the (issued) metadata read-ahead instruction). However, if the starting offset of the current metadata read operation 250 is not next to the end offset of the last read metadata chunk 245, and thus no metadata read-ahead is triggered, access pattern tracker 225 resets the stored value (e.g., the end offset of on-disk inodes 165(1)-(4) – the end offset of the last read metadata chunk 245) with an end offset of the current metadata read operation.

[0056] It will be appreciated that chunk access pattern(s) of inodes can be tracked and metadata read-ahead instructions and/or metadata read-ahead operations can be issued to pre-fetch applicable inodes from disk into memory to accelerate the subsequent access of these inodes. Described next are methods, systems, and processes to track inode access pattern(s) for inodes that are part of directories.

Example of Using a Parent Directory List to Track Access Pattern of Inodes

[0057] It will be appreciated that an inode allocation policy can maintain on-disk inodes in proximity to each other within the same directory, referred to herein as proximate locality. For example, the on-disk inodes of files that are accessed frequently and together can be maintained within the same directory. This proximate locality of on-disk inodes in directories can be used to track directory access of inodes. For example, global inode list 215 (or parent directory list) can be created and maintained to track access of on-disk inodes.

[0058] FIG. 2C is a table 200C illustrating the contents of such a global inode list, and FIG. 2D is a block diagram 200D of directory access tracker 235 that uses a global inode list, according to certain embodiments. Global inode list 215 (which is a parent directory list for directories 265(1)-(N)) includes a directory inode field 255 and a sequential flag field 260. Global inode list 215 is created and maintained in memory, and includes directory inodes 130(1)-(N) and sequential flags 220(1)-(N). Sequential flags can be stored as part of in-core inodes for parent directories. Directory access tracker 235 includes a sequential directory access tracker 270 and a non-sequential directory access tracker 275.

[0059] In one embodiment, an application accesses an inode (e.g., on-disk inode 165(4)). Directory access tracker 235 determines the parent directory of the inode (e.g., directory 265(1)), and whether an entry for the directory exists in global inode list 215. If the entry for the directory exists in global inode list 215, directory access tracker 235 determines whether a file structure of the directory is sequential or non-sequential (e.g., by determining whether the on-disk inodes in that directory are listed, and thus being accessed, in a sequential or non-sequential manner). If the entry for the directory does not exist in global inode list 215, directory access tracker 235 adds a new entry for the parent directory inode in global inode list 215. It should be noted that as shown in FIGs. 1A, 2A, 2C and 3C, directory inodes (e.g., directory inodes 130(1)-(N)) are simply directory-specific representations of on-disk inodes that represent parent directories (e.g., instead of individual files). For example, directory inode 130(1) is an in-memory data structure that represents a parent directory of one or more on-disk inodes.

[0060] To track sequential directory access of on-disk inodes, sequential directory access tracker 270 first finds a parent directory of a given on-disk inode (e.g., finds the parent directory's inode number). For example, if on-disk inode 165(4) is read from disk 150, sequential directory access tracker 270 finds a parent directory of on-disk inode 165(4) (e.g., directory inode 130(1)). Sequential directory access tracker 270 then searches global inode list 215 for an entry of the parent directory (inode) of a given on-disk inode (e.g., whether the parent directory inode number is present in cache 205).

[0061] If the entry of the parent directory (inode) exists in global inode list 215, sequential directory access tracker 270 checks global inode list 215 to determine whether the parent directory has a sequential flag set (e.g., directory inode 130(1), which in this case, is the parent directory inode number, and has the sequential flag set as shown in FIG. 2C). If the sequential flag is set, metadata read-ahead generator 240 issues a metadata read-ahead instruction (e.g., to fetch all remaining on-disk inodes in directory 265(1) because the on-disk inode access is sequential). If sequential directory access tracker 270 does not find the entry of the parent directory in global inode list 215, sequential directory access tracker 270 adds a new entry of the parent directory's inode number to global inode list 215.

[0062] To track non-sequential directory access of on-disk inodes, non-sequential directory access tracker 275 first finds a parent directory of a given on-disk inode (e.g., the parent directory's inode number). Non-sequential directory access tracker 275 then searches global inode list 215 for an existing entry of the parent directory (e.g., whether the parent directory inode number is present in cache 205). If the entry of the parent directory exists in global inode list 215, non-sequential directory access tracker 275 fetches (or retrieves) an inode list of the parent directory (e.g., a portion or a part of inode list 175 applicable to the parent directory in question) from disk 150 into cache 205, and metadata read-ahead generator 240 issues a metadata read-ahead instruction for the on-disk inodes listed on the retrieved inode list (e.g., on-disk inodes that are associated with and part of the parent directory). If the entry of the parent directory does not exist in global inode list 215, non-sequential directory access tracker 275 adds a new entry of the parent directory's inode number to global inode list 215.

Example of Issuing Metadata Read-Ahead Instructions for Inodes

[0063] FIG. 2E is a block diagram 200E of a metadata read-ahead generator, according to one embodiment. Metadata read-ahead generator 240 is implemented by computing device 105 and stores an issued metadata read-ahead value 280 and an asynchronous metadata read-ahead instruction 285, and includes queue generator 290. Although directory access tracker 235 identifies one or more on-disk inodes to pre-fetch, if these on-disk inodes are not pre-fetched, I/O operations associated with these on-disk inodes cannot be completed.

[0064] Therefore, in one embodiment, metadata read-ahead generator 240 intercepts a command to read on-disk inodes in response to an I/O operation (e.g., a read operation or a write operation). An I/O operation to access data can result in (or cause) a command to access and read on-disk inode(s) (e.g., metadata) associated with that data (e.g., to determine when and how the requested data has been modified, and the like). Metadata read-ahead generator 240 analyzes issued metadata read-ahead value 280 in the metadata read-ahead operation by comparing issued metadata read-ahead value 280 and a chunk total in the command.

[0065] Issued metadata read-ahead value 280 includes all the on-disk inodes that should be read ahead (e.g., detected based on sequential/near-sequential chunk access, and/or sequential or non-sequential access of on-disk inodes in directories). A chunk total is the total number of chunks of on-disk inodes to be read ahead (e.g., represented as an integer "N" herein for discussion purposes). For example, chunk access tracker 230 and sequential directory access tracker 270 can identify and determine that a 1 KB chunk of on-disk inodes (e.g., on-disk inodes 165(1)-(4)) or a 2k chunk of on-disk inodes (e.g., on-disk inodes 165(1)-(8)) must be read-ahead (e.g., based on sequential/near-sequential chunk access, and/or sequential or non-sequential access of on-disk inodes in directories). However, as discussed above, non-sequential directory access tracker 275 can identify several non-sequential on-disk inodes (e.g., that can be part of various disparate chunks) to read ahead (e.g., as shown in the case of directory 265(2) in FIG. 3B). Therefore, it will be appreciated that in certain scenarios, issued metadata read-ahead value 280 may or may not be equal to N.

[0066] In some embodiments, based on comparing issued metadata read-ahead value 280 and the chunk total in the command, metadata read-ahead generator 240 either waits for the I/O operation to complete, or issues asynchronous metadata read-ahead instruction 285. If the I/O operation is complete, queue generator 290 generates a queue and includes the remaining chunks of metadata of the chunk total not included in asynchronous metadata read-ahead instruction 285. However, if the I/O operation is incomplete, queue generator 290 updates the chunk total in the metadata read-ahead operation.

[0067] For example, a command (or call) to read an on-disk inode which goes to disk 150 is intercepted after an I/O operation is issued (e.g., by application 130). This command triggers the inode access pattern detection methods described above (e.g., sequential/near-sequential chunk access, and/or sequential or non-sequential access of on-disk inodes in directories). If the inode access pattern detection methods do not trigger a read-ahead of on-disk inodes, metadata read-ahead generator 240 simply waits for the I/O operation to complete. However, if the inode access pattern detection methods do trigger a read-ahead of on-disk inodes, metadata read-ahead generator 240 determines whether the total issued metadata read ahead (e.g., issued metadata read-ahead value 280) is less than or equal to N (e.g., the total number of chunks of on-disk inodes to be read ahead).

[0068] If the total issued metadata read ahead is less than or equal to N, metadata read-ahead generator 240 issues asynchronous metadata read-ahead instruction 285 with the next chunk of the asynchronous metadata read (e.g., the next chunk in issued metadata read-ahead value 280 after N). If the original I/O operation is complete, queue generator 290 generates a separate thread which issues asynchronous metadata read-ahead instruction 285 with the remaining chunks of the asynchronous metadata read (e.g., the remaining chunks in issued metadata read-ahead value 280 after N). If the original I/O operation is not complete, metadata read-ahead generator 240 increments a counter for issued metadata read-ahead value 280, determines whether issued metadata read-ahead value 280 is equal to N, and waits for the original I/O operation to complete.

[0069] It will be appreciated that metadata read-ahead detection and the issuing of asynchronous metadata read-ahead instructions is performed in the context of the blocking thread, while the original I/O operation is waiting to complete in the background. These methodologies decrease inode access pattern detection overhead on system performance, and also avoid the creation and scheduling of separate threads, which can delay the availability of blocks for subsequent reads.

Example of Tracking Sequential and Non-Sequential Directory Access Pattern of Inodes

[0070] FIG. 3A is a block diagram of a directory with a sequential inode structure 300A, FIG. 3B is a block diagram of a directory with a non-sequential inode structure 300B, and FIG. 3C is a block diagram of a cache 300C that implements a global inode list (e.g., a parent directory list or a global in-memory inode list), according to some embodiments. It will be appreciated that a listing of directories is first performed by an application (e.g., by application 135, application 140, or some other application). During the listing of directories, on-disk inode numbers associated with each directory entry are returned. If the on-disk inode numbers within a given directory are sequential (e.g., directory 265(1) as shown in FIG. 3A), then sequential directory access tracker 270 sets a sequential flag for that particular directory in the in-core inode of that directory (e.g., a sequential flag for directory 265(1) is set in global inode list 215 as shown in FIGs. 2C and 3C).

[0071] For example, because the on-disk inodes of directory 265(1) are sequential (e.g., on-disk inodes 165(4)-(9) are listed sequentially), sequential directory access tracker 270 sets a sequential flag in directory inode 130(1) (e.g., in an in-core inode) for directory 265(1) (e.g., indicated by "1" in sequential flag field 260 of global inode list 215 in FIGs. 2C and 3C). On the contrary, because the on-disk inodes of directory 265(2) are non-sequential (e.g., on-disk inodes 165(4), 165(9), 165(15), 165(11), 165(19), and 165(6) are listed non-sequentially), non-sequential directory access tracker 275 does not set a sequential flag in directory inode 130(2) (e.g., in an in-core inode) for directory 265(2)) (e.g., indicated by "0" in sequential flag field 260 of global inode list 215 in FIGs. 2C and 3C).

[0072] As previously noted, a metadata read-ahead instruction to perform a metadata read-ahead operation can be issued after a listing of a directory is performed. In one embodiment, an application performs the listing of directories that are part of disk 150. For example, a listing of directory 265(2) as shown in FIG. 3B, returns file 315(1) with on-disk inode 165(4), file 315(2) with on-disk inode 165(9), file 315(3) with on-disk inode 165(15), file 315(4) with on-disk inode 165(11), file 315(5) with on-disk inode 165(19), and file 315(6) with on-disk inode 165(6). In this scenario, directory access tracker 235 creates and maintains an in-memory data structure (e.g., an in-memory inode list) that includes a list of the foregoing inode numbers (e.g., on-disk inodes 165(4), 165(9), 165(15), 165(11), 165(19), and 165(6)), and associates this in-memory inode list with the in-core inode of directory 265(2) (e.g., directory inode 130(2)).

[0073] For example, if application 130 accesses file 315(1), and hence there is a need to access and read on-disk inode 165(4), non-sequential directory access tracker 275 reads on-disk inode 165(4) from disk 150, and determines that the parent directory inode number of on-disk inode 165(4) is directory inode 130(2). Non-sequential directory access tracker 275 then checks if directory inode 130(2) is present in cache 205, as shown in FIG. 3C. If directory inode 130(2) is not present in cache 205, non-sequential directory access tracker 275 adds directory inode 130(2) (e.g., indicated by bold in FIGs. 2C and 3C) to cache 205.

[0074] Next, if application 130 accesses file 315(2), and thus accesses and reads on-disk inode 165(9) from disk 150, non-sequential directory access tracker 275 determines that the parent directory inode number of on-disk inode 165(9) is also directory inode 130(2). Because directory inode 130(2) has been added to cache 205, metadata read-ahead generator 240 determines that the remaining files (e.g., files 315(3)-(6)) under directory 265(2) can be read ahead. Metadata read-ahead generator 240 then accesses the in-memory inode list and identifies the remaining on-disk inode numbers associated with files 315(3)-(6) (e.g., on-disk inodes 165(15), 165(11), 165(19), and 165(6)), and generates a metadata read-ahead instruction that performs a metadata read-ahead operation to fetch on-disk inodes 165(15), 165(11), 165(19), and 165(6) from disk 150 to memory 115.

[0075] It will be appreciated that directory access tracker 235 tracks chunk access pattern(s) of on-disk inodes, as well as sequential and non-sequential access pattern(s) of on-disk inodes that are part of directories, to identify on-disk inodes that are candidates for a metadata read-ahead operation that accelerates inode and data access.

Processes to Track Inode Access Pattern(s) and Issue Metadata Read-Ahead Instructions

[0076] FIG. 4A is a flowchart 400A that illustrates a process for issuing a read-ahead instruction to pre-fetch on-disk inodes from disk to memory, according to one embodiment. The process begins at 405 by accessing a metadata storage area (e.g., structural file 160). At 410, the process determines the location of a last read chunk of metadata (e.g., end offset of last read metadata chunk 245). At 415, the process determines if a command (or call) (e.g., to read on-disk inodes) has been received. If no command has been received yet, the process loops back to 415. However, if a command has been received, the process, at 420, determines the location of an object of the command in the metadata storage area (e.g., starting offset of current metadata read operation 250).

[0077] At 425, the process determines whether a metadata read-ahead can be performed (or whether a metadata read-ahead is required or feasible). If a metadata read-ahead cannot be performed, the process, at 430, permits normal processing (e.g., no metadata read-ahead operation is performed and on-disk inodes are not pre-fetched from disk into memory). However, if a metadata read-ahead can be performed (and/or is needed and/or feasible), the process, at 435, issues a metadata read-ahead operation (or issues a metadata read-ahead instruction, e.g., using metadata read-ahead generator 240, that causes computing device 105 to perform a metadata read-ahead operation to fetch chunk(s) of on-disk inodes from disk into memory). At 440, the process determines if there is a new command (e.g., to access and/or read on-disk inodes). If there is a new command to access and/or read on-disk inodes, the process loops back to 405. Otherwise, the process ends.

[0078] FIG. 4B is a flowchart 400B that illustrates a process for storing offset metadata associated with on-disk inodes, according to one embodiment. The process begins at 445 by determining an offset location of a chunk of metadata (e.g., end offset of last read metadata chunk 245) in a metadata storage area (e.g., structural file 160). At 450, the process creates an in-core inode (e.g., in-core inode list 125) in memory (e.g., memory 115), and at 455, the process stores the offset location of the chunk of metadata in the in-core inode.

[0079] At 455, the process determines whether a location of a current metadata read operation (e.g., starting offset of current metadata read operation 250) is next to (or adjacent to) the offset location of the chunk of metadata. If the location of the current metadata read operation is not next to the offset location of the chunk of metadata, the process, at 465, stores an offset location of a chunk of metadata read by the current metadata read operation (e.g., an end offset of the current metadata read operation). However, if the location of the current metadata read operation is next to the offset location of the chunk of metadata, the process, at 470, issues a metadata read-ahead operation (or issues a metadata read-ahead instruction), and at 475, stores an offset location of a chunk of metadata read by the metadata read-ahead operation. At 480, the process determines whether there is a new read call (e.g., a command to read on-disk inodes caused by an application I/O operation). If there is a new read call to read on-disk inodes, the process loops back to 460. Otherwise, the process ends.

[0080] It will be appreciated that the processes illustrated in flowchart 400A of FIG. 4A and flowchart 400B of FIG. 4B are examples of tracking the chunk access patterns of on-disk inodes. Because on-disk inodes are persistently stored on disk (e.g., disk 150), chunk access tracker 230 can access structural file 160 to determine the end offset of a last read metadata chunk as well as the starting offset of a current metadata read operation. Because chunk access tracker 230 can save this location information, chunk access tracker 230 can determine whether on-disk inodes are being accessed by application 135 or application 140 in a sequential or nearly-sequential manner. Based on this stored location information, on-disk inodes that are likely to be accessed can be read ahead and pre-fetched from disk into memory, thus accelerating subsequent inode access for those on-disk inodes.

[0081] FIG. 5A is a flowchart 500A that illustrates a process for determining the file structure of a directory, according to one embodiment. The process begins at 505 by accessing a file within a directory (e.g., file 135(1) in directory 265(1) as shown in FIG. 3A or file 135(1) in directory 265(2) as shown in FIG. 3B). At 510, the process searches a global inode list for an entry (e.g., an inode number) of the directory. At 515, the process determines whether the directory exists in the global inode list (e.g., whether directory inode 130(1), which is the inode number of directory 265(1), exists and is listed in global inode list 215).

[0082] If the directory does not exist in the global inode list, the process, at 520, adds a new entry (e.g., adds a parent directory inode number) for an on-disk inode in the global inode list (e.g., shown with respect to directory 265(3) in FIGs. 2C and 3C). If the directory exists in the global inode list, the process, at 525, determines a file structure of the directory (e.g., whether the on-disk inodes in the directory are listed sequentially or non-sequentially). At 530, the process determines whether there is another access of on-disk inodes. If there is another access of on-disk inodes, the process loops back to 505. Otherwise, the process ends.

[0083] FIG. 5B is a flowchart 500B that illustrates a process for issuing a metadata read-ahead instruction for on-disk inodes, according to one embodiment. The process begins at 535 by determining whether on-disk inode numbers of entries within a given directory are sequential or non-sequential (e.g., listed sequentially or non-sequentially as the result of performing a listing of the directory). For example, in FIG. 3A, the on-disk inode numbers of entries are listed sequentially, and in FIG. 3B, the on-disk inode numbers of entries are listed non-sequentially.

[0084] If the on-disk inode numbers of entries are listed non-sequentially, the process, at 540, access a global inode list (e.g., a parent directory list as shown in FIG. 3C). At 545, the process identifies the parent directory inode on the global inode list (e.g., using the on-disk inode's parent directory inode number). At 550, the process fetches an inode list associated with the directory in question (e.g., from disk 150, or from memory because the inode list can be stored and maintained in memory during the listing process), and at 555, the issues a metadata read-ahead instruction for on-disk inodes on the inode list.

[0085] However, if the on-disk inode numbers of entries are listed sequentially, the process, at 560, access the global inode list, and at 565, identifies an in-memory inode of the parent directory of a file that is accessed on the global inode list (e.g., using the on-disk inode's parent directory inode number). At 570, the process verifies that the directory has the sequential flag set, and at 575, issues a metadata read-ahead instruction for the remaining on-disk inodes listed in the directory in question. At 580, the process determines whether there is another access of on-disk inode(s). If there is another access of on-disk inode(s), the process loops back to 535. Otherwise, the process ends.

[0086] It will be appreciated that the processes illustrated in flowchart 500A of FIG. 5A and flowchart 500B of FIG. 5B are examples of using a parent directory list to track access pattern(s) of on-disk inodes when directories are involved. As previously noted, an application can perform a listing of one or more directories to determine whether on-disk inodes in a given directory are listed sequentially or non-sequentially. Because directories are typically stored on disk (e.g., disk 150), determining a directory's data and/or file structure each time a file is accessed in a given directory can consume significant computing resources. However, because both access pattern tracker 225 and the parent directory list are part of memory 115, and because the parent directory list maintains a listing of the data and/or file structure of multiple directories, access pattern tracker 225 can simply use the parent directory list to determine whether a given directory's in-core inodes are sequential or non-sequential, without accessing disk 150 each time a file in a given directory is accessed.

[0087] FIG. 6A is a flowchart 600A that illustrates a process for processing input/output (I/O) operations related to pre-fetching inodes, according to one embodiment. The process begins at 605 by detecting an I/O operation issued for a chunk of metadata associated with an on-disk inode (e.g., a unit of metadata in structural file 160). At 610, the process determines whether the I/O operation is detected. If the I/O operation is not detected, the process loops back to 605. However, if the I/O operation is detected, the process, at 615, intercepts a command (or call) to access and/or read on-disk inode(s).

[0088] At 620, the process accesses a metadata read-ahead value of the metadata read-ahead operation, and at 625, analyzes the metadata read-ahead value by comparing the (total issued) metadata read-ahead value to the total number of chunks of metadata to be read ahead. At 630, the process waits for the I/O operation to complete, and at 635, issues an asynchronous metadata read-ahead instruction (e.g., using metadata read-ahead generator 240). At 640, the process determines if there is another I/O operation. If there is another I/O operation, the process loops to 605. Otherwise, the process ends.

[0089] FIG. 6B is a flowchart that illustrates a process for processing I/O operations related to pre-fetching inodes, according to one embodiment. The process begins at 645 by determining whether a given I/O operation is complete. If the I/O operation is not complete, the process, at 650, updates a chunk total (e.g., a total number of chunks of metadata to be read-ahead or "N"), and proceeds to 625 (in FIG. 6A). However, if the I/O operation is complete, the process, at 660, generates a queue (e.g., a separate thread), and at 665, issues a metadata read-ahead that includes the remaining chunks of metadata not included in the asynchronous metadata read-ahead instruction (e.g., of FIG. 6A). At 670, the process determines if there is another command (e.g., a call to read on-disk inode(s)) to be intercepted. If there is another command to be intercepted, the process loops back to 615 (in FIG. 6A). Otherwise, the process ends.

[0090] It will be appreciated that the processes illustrated in flowchart 600A of FIG. 6A and flowchart 600B of FIG. 6B can be used to identify and pre-fetch on-disk inodes that are part of various disparate chunks of metadata by comparing an issued metadata read-ahead value and a chunk total. Because all or a portion of an inode list can be pre-fetched into memory during a directory listing process, on-disk inodes that are candidates for a metadata read-ahead instruction can be identified in cases where such on-disk are not accessed sequentially. Further, it will also be appreciated that the metadata read-ahead detection and the issuing of asynchronous metadata read-ahead instructions is performed in the context of the blocking thread, while the original I/O operation is waiting to complete in the background. These methodologies decrease inode access pattern detection overhead on system performance, and also avoid the creation and scheduling of separate threads, which can delay the availability of blocks for subsequent reads.

[0091] FIG. 7A is a flowchart that illustrates a process for processing access of on-disk inodes and adding an entry of a directory to a global inode list, according to certain embodiments. The process begins at 705 by determining whether there is an access of on-disk inode(s) (e.g., a read command/call of on-disk inode(s) caused by one or more I/O operations). If there is no access of on-disk inode(s), the process loops back to 705. However, if there is access of on-disk inode(s), the process, at 710, identifies a directory associated with the on-disk inode(s) (e.g., using a parent directory inode number). At 715, the process access a global inode list (e.g., global inode list 215 and/or a parent directory list).

[0092] At 720, the process determines whether the directory is on the global inode list (e.g., whether the parent directory inode number of the inode exists on the global inode list). If the directory is not on the global inode list, the process, at 725, adds the directory to the global inode list (e.g., by adding an entry for the parent directory inode number of the inode to the global inode list). However, if the directory is on the global inode list, the process, at 730, verifies that a sequential flag is set for the directory, and at 735, issues a metadata read-ahead instruction (e.g., for the remaining on-disk inodes that are listed in (or part of) the directory). At 740, the process determines whether there is another access (e.g., of on-disk inode(s)). If there is another access, the process loops back to 705. Otherwise, the process ends.

[0093] FIG. 7B is a flowchart that illustrates a process for processing a request to access on-disk inodes, according to one embodiment. The process begins at 745 by detecting an I/O operation. At 750, the process intercepts a command (or call) to access (or read) on-disk inode(s). At 755, the process determines whether an on-disk inode access pattern (e.g., detected based on chunk access, and sequential or non-sequential access, among other methodologies) triggers, causes, or results in a metadata read-ahead of the on-disk inode(s). If the on-disk inode access pattern does not trigger the metadata read-ahead (e.g., based on chunk access, sequential access, or non-sequential access), the process, at 760, waits for the I/O operation to complete. It should be noted that an example process for waiting for the I/O operation to complete is illustrated in flowchart 600B of FIG. 6B.

[0094] However, if the on-disk inode access pattern does trigger the metadata read-ahead, the process, at 765, determines whether a chunk total (e.g., a total number of chunks of on-disk inode metadata to be read ahead or "N") is less than or equal to a metadata read-ahead value (e.g., issued metadata read-ahead value 280 or a total issued metadata read-ahead value). If the chunk total is not less than or equal to the metadata read-ahead value, the process, at 760, waits for the I/O operation to complete. However, if the chunk total is less than or equal to the metadata read-ahead value, the process, at 770, issues a next chunk of asynchronous metadata read-ahead, and at 775, determines whether the I/O operation is complete. If the I/O operation is not complete, the process, at 780, increments a counter (e.g., for the issued metadata read-ahead), and loops back to 765. However, if the I/O operation is complete, the process, at 785, generates a separate thread and issues the remaining chunks of the asynchronous metadata read-ahead. At 790, the process determines if there is another I/O operation. If there is another I/O operation, the process loops back to 745. Otherwise, the process ends.

[0095] Typically, if an I/O operation is blocked, a thread cannot progress any further. It will be appreciated that because the methods, systems, and processes of inode access pattern detection and metadata read-ahead generation are performed in the context of the blocking thread (and while the original I/O is waiting for completion in the background), inode access pattern detection overhead on system performance is reduced. In addition, permitting the original I/O operation to complete can also avoid the cost of creating and scheduling separate threads (which can delay the availability of data blocks for subsequent read operations) .

[0096] Further, it should be noted that other data structures such as attributes can be associated with nodes. The methods, systems, and processes related to inode access pattern detection and issuing metadata read-ahead instructions described herein can pre-populate these (other) data structures in memory and can initialize various inode locks asynchronously. Therefore, it will be appreciated that the methods, systems, and processes described herein are capable of tracking access patterns of inodes based on chunk access, sequential access, and non-sequential access, and issuing read-ahead instructions for inodes, among other capabilities.

Example Computing Environment

[0097] FIG. 8 is a block diagram of a computing system 800, illustrating how an access pattern tracker and a metadata read-ahead generator can be implemented in software, according to one embodiment. Computing system 800 broadly represents any single or multi-processor computing device or system capable of executing computer-readable instructions. Examples of computing system 800 include, without limitation, any one or more of a variety of devices including workstations, personal computers, laptops, client-side terminals, servers, distributed computing systems, handheld devices (e.g., personal digital assistants and mobile phones), network appliances, storage controllers (e.g., array controllers, tape drive controller, or hard drive controller), and the like. In its most basic configuration, computing system 800 may include at least one processor 110 and a memory 115. By executing the software that implements computing device 105, computing system 800 becomes a special purpose computing device that is configured to track inode access pattern(s) and issue read-ahead instructions for inodes.

[0098] Processor 110 generally represents any type or form of processing unit capable of processing data or interpreting and executing instructions. In certain embodiments, processor 110 may receive instructions from a software application or module. These instructions may cause processor 110 to perform the functions of one or more of the embodiments described and/or illustrated herein. For example, processor 110 may perform and/or be a means for performing all or some of the operations described herein. Processor 110 may also perform and/or be a means for performing any other operations, methods, or processes described and/or illustrated herein.

[0099] Memory 115 generally represents any type or form of volatile or non-volatile storage devices or mediums capable of storing data and/or other computer-readable instructions. Examples include, without limitation, random access memory (RAM), read only memory (ROM), flash memory, or any other suitable memory device. Although not required, in certain embodiments computing system 800 may include both a volatile memory unit and a non-volatile storage device. In one example, program instructions implementing an access pattern tracker and a metadata read-ahead generator may be loaded into memory 115.

[00100] In certain embodiments, computing system 800 may also include one or more components or elements in addition to processor 110 and/or memory 115. For example, as illustrated in FIG. 8, computing system 800 may include a memory controller 820, an Input/Output (I/O) controller 835, and a communication interface 845, each of which may be interconnected via a communication infrastructure 805. Communication infrastructure 805 generally represents any type or form of infrastructure capable of facilitating communication between one or more components of a computing device. Examples of communication infrastructure 805 include, without limitation, a communication bus (such as an Industry Standard Architecture (ISA), Peripheral Component Interconnect (PCI), PCI express (PCIe), or similar bus) and a network.

[00101] Memory controller 820 generally represents any type/form of device capable of handling memory or data or controlling communication between one or more components of computing system 800. In certain embodiments memory controller 820 may control communication between processor 110, memory 115, and I/O controller 835 via communication infrastructure 805. In certain embodiments, memory controller 820 may perform and/or be a means for performing, either alone or in combination with other elements, one or more of the operations or features described and/or illustrated herein.

[00102] I/O controller 835 generally represents any type or form of module capable of coordinating and/or controlling the input and output functions of one or more computing devices such as computing device 105. For example, in certain embodiments I/O controller 835 may control or facilitate transfer of data between one or more elements of computing system 800, such as processor 110, memory 115, communication interface 845, display adapter 815, input interface 825, and/or storage interface 840.

[00103] Communication interface 845 broadly represents any type or form of communication device or adapter capable of facilitating communication between computing system 800 and one or more other devices. Communication interface 845 may facilitate communication between computing system 800 and a private or public network including additional computing systems. Examples of communication interface 845 include, without limitation, a wired network interface

(such as a network interface card), a wireless network interface (such as a wireless network interface card), a modem, and any other suitable interface. Communication interface 845 may provide a direct connection to a remote server via a direct link to a network, such as the Internet, and may also indirectly provide such a connection through, for example, a local area network (e.g., an Ethernet network), a personal area network, a telephone or cable network, a cellular telephone connection, a satellite data connection, or any other suitable connection.

[00104] Communication interface 845 may also represent a host adapter configured to facilitate communication between computing system 800 and one or more additional network or storage devices via an external bus or communications channel. Examples of host adapters include, Small Computer System Interface (SCSI) host adapters, Universal Serial Bus (USB) host adapters, Institute of Electrical and Electronics Engineers (IEEE) 1394 host adapters, Serial Advanced Technology Attachment (SATA), Serial Attached SCSI (SAS), and external SATA (eSATA) host adapters, Advanced Technology Attachment (ATA) and Parallel ATA (PATA) host adapters, Fibre Channel interface adapters, Ethernet adapters, or the like. Communication interface 845 may also allow computing system 800 to engage in distributed or remote computing (e.g., by receiving/sending instructions to/from a remote device for execution).

[00105] As illustrated in FIG. 8, computing system 800 may also include at least one display device 810 coupled to communication infrastructure 805 via a display adapter 815. Display device 810 generally represents any type or form of device capable of visually displaying information forwarded by display adapter 815. Similarly, display adapter 815 generally represents any type or form of device configured to forward graphics, text, and other data from communication infrastructure 805 (or from a frame buffer, as known in the art) for display on display device 810. Computing system 800 may also include at least one input device 830 coupled to communication infrastructure 805 via an input interface 825. Input device 830 generally represents any type or form of input device capable of providing input, either computer or human generated, to computing system 800. Examples of input device 830 include a keyboard, a pointing device, a speech recognition device, or any other input device.

[00106] Computing system 800 may also include storage device 850 (e.g., disk 150) coupled to communication infrastructure 805 via a storage interface 840. Storage device 850 generally represents any type or form of storage devices or mediums capable of storing data and/or other computer-readable instructions. For example, storage device 850 may include a magnetic disk drive (e.g., a so-called hard drive), a floppy disk drive, a magnetic tape drive, an optical disk drive, a flash drive, or the like. Storage interface 840 generally represents any type or form of interface or device for transferring and/or transmitting data between storage device 850, and other components of computing system 800. Storage device 850 may be configured to read from and/or write to a removable storage unit configured to store computer software, data, or other computer-readable information. Examples of suitable removable storage units include a floppy disk, a magnetic tape, an optical disk, a flash memory device, or the like. Storage device 850 may also include other similar structures or devices for allowing computer software, data, or other computer-readable instructions to be loaded into computing system 800. For example, storage device 850 may be configured to read and write software, data, or other computer-readable information. Storage device 850 may also be a part of computing system 800 or may be separate devices accessed through other interface systems.

[00107] Many other devices or subsystems may be connected to computing system 800. Conversely, all of the components and devices illustrated in FIG. 8 need not be present to practice the embodiments described and/or illustrated herein. The devices and subsystems referenced above may also be interconnected in different ways from that shown in FIG. 8. Computing system 800 may also employ any number of software, firmware, and/or hardware configurations. For example, one or more of the embodiments disclosed herein may be encoded as a computer program (also referred to as computer software, software applications, computer-readable instructions, or computer control logic) on a computer-readable storage medium. Examples of computer-readable storage media include magnetic-storage media (e.g., hard disk drives and floppy disks), optical-storage media (e.g., CD- or DVD-ROMs), electronic-storage media (e.g., solid-state drives and flash media), and the like. Such computer programs can also be transferred to computing system 800 for storage in memory via a network such as the Internet or upon a carrier medium.

[00108] The computer-readable medium containing the computer program may be loaded into computing system 800. All or a portion of the computer program stored on the computer-readable medium may then be stored in memory 860 and/or various portions of storage device 850. When executed by processor 110, a computer program loaded into computing system 800 may cause processor 110 to perform and/or be a means for performing the functions of one or more of the embodiments described and/or illustrated herein. Additionally or alternatively, one or more of the embodiments described and/or illustrated herein may be implemented in firmware and/or hardware. For example, computing system 800 may be configured as an application specific integrated circuit (ASIC) adapted to implement one or more of the embodiments disclosed herein.

Example Networking Environment

[00109] FIG. 9 is a block diagram of a networked system 900, illustrating how various devices can communicate via a network, according to one embodiment. In certain embodiments, network-attached storage (NAS) devices may be configured to communicate with computing device 105 and storage system 145 using various protocols, such as Network File System (NFS), Server Message Block (SMB), or Common Internet File System (CIFS), among others.

[00110] Network 185 generally represents any type or form of computer network or architecture capable of facilitating communication between computing device 105 and storage system 145. In certain embodiments, a communication interface, such as communication interface 845 in FIG. 8, may be used to provide connectivity between computing device 105, storage system 145, and network 155. It should be noted that the embodiments described and/or illustrated herein are not limited to the Internet or any particular network-based environment. For example, network 185 can be a Storage Area Network (SAN). Computing device 105 and storage system 145 can be integrated or separate. If separate, for example, computing device 105 and storage system 145 can be coupled by a local connection (e.g., using Bluetooth™, Peripheral Component Interconnect (PCI), Small Computer System Interface (SCSI), or the like), or via one or more networks such as the Internet, a LAN, or a SAN.

[00111] In one embodiment, all or a portion of one or more of the disclosed embodiments may be encoded as a computer program and loaded onto and executed by computing device 105, inode access pattern tracking and metadata read-ahead instruction issuing system 910, inode access pattern tracking system 940, and/or metadata read-ahead instruction generation system 950. All or a portion of one or more of the embodiments disclosed herein may also be encoded as a computer program, stored on computing device 105, inode access pattern tracking and metadata read-ahead instruction issuing system 910, and/or inode access pattern tracking system 940, and distributed over network 185.

[00112] In some examples, all or a portion of computing device 105 may represent portions of a cloud-computing or network-based environment. Cloud-computing environments may provide various services and applications via the Internet. These cloud-based services (e.g., software as a service, platform as a service, infrastructure as a service, etc.) may be accessible through a web browser or other remote interface. Various functions described herein may be provided through a remote desktop environment or any other cloud-based computing environment.

[00113] In addition, one or more of the components described herein may transform data, physical devices, and/or representations of physical devices from one form to another. For example, computing device 105 and/or inode access pattern tracking and metadata read-ahead instruction issuing system 910 may transform the behavior of computing device 105 in order to cause computing device 105 and/or inode access pattern tracking and metadata read-ahead instruction issuing system 910 to track access pattern(s) of inodes and issue read-ahead instructions.

[00114] Although the present disclosure has been described in connection with several embodiments, the disclosure is not intended to be limited to the specific forms set forth herein. On the contrary, it is intended to cover such alternatives, modifications, and equivalents as can be reasonably included within the scope of the disclosure as defined by the appended claims.

WHAT IS CLAIMED IS:

1. A method comprising:
determining a location of a unit of metadata in a metadata storage area, wherein
the metadata storage area is in a non-volatile storage unit;
determining an another location in the metadata storage area, wherein
the another location corresponds to a current metadata read operation;
determining whether a metadata read-ahead operation can be performed, using the
location of the unit of metadata and the another location; and
in response to a determination that the metadata read-ahead operation can be performed,
issuing the metadata read-ahead operation.

2. The method of claim 1, wherein
the metadata storage area comprises a plurality of on-disk inodes, wherein
the on-disk inodes comprise the unit of metadata,
the on-disk inodes are part of a structural file, and
the structural file is stored in the non-volatile storage unit.

3. The method of claim 2, further comprising:
creating an in-core inode, wherein
the in-core inode corresponds to the structural file; and
storing the location of the unit of metadata in the in-core inode, wherein
the unit of metadata is a last-read chunk of metadata, and
the location of the unit of metadata identifies an end offset of the unit of metadata.

4. The method of claim 3, wherein
determining whether the metadata read-ahead operation can be performed comprises
accessing the end offset of the unit of metadata, and
determining whether the another location is adjacent to the end offset.

5. The method of claim 3, wherein
issuing the metadata read-ahead operation comprises
updating the end offset by replacing the end offset with an another end offset of an another unit of metadata read by the metadata read-ahead operation, wherein
the another unit of metadata is an another last-read chunk of metadata.

6. The method of claim 3, further comprising:
updating the end offset by replacing the end offset with an end offset of a current metadata read operation if the another location is not adjacent to the end offset.

7. The method of claim 2, further comprising:
intercepting a command to read one or more on-disk inodes, wherein
the one or more on-disk inodes are part of the plurality of on-disk inodes,
the intercepting is performed in response to an input/output (I/O) operation, and
the I/O operation results in the command;
analyzing an issued metadata read-ahead value in the metadata read-ahead operation, wherein
the analyzing comprises comparing the issued metadata read-ahead value and a chunk total in the command;
waiting for the I/O operation to complete, if the analyzing indicates the I/O operation should complete; and
issuing an asynchronous metadata read-ahead instruction, if the analyzing indicates that an asynchronous metadata read-ahead instruction can be issued.

8. The method of claim 7, further comprising:
determining whether the I/O operation is complete;
generating a queue if the I/O operation is complete, wherein

the queue comprises one or more remaining chunks of metadata of the chunk total, and

the one or more remaining chunks of metadata data are not included in the asynchronous metadata read-ahead instruction; and

updating the chunk total in the metadata read-ahead operation, if the I/O operation is incomplete.

9. A non-transitory computer readable storage medium comprising program instructions executable to:

determine a location of a unit of metadata in a metadata storage area, wherein the metadata storage area is in a non-volatile storage unit;

determine an another location in the metadata storage area, wherein the another location corresponds to a current metadata read operation;

determine whether a metadata read-ahead operation can be performed, using the location of the unit of metadata and the another location; and

in response to a determination that the metadata read-ahead operation can be performed, issuing the metadata read-ahead operation.

10. The non-transitory computer readable storage medium of claim 9, wherein

the metadata storage area comprises a plurality of on-disk inodes, wherein

the on-disk inodes comprise the unit of metadata,

the on-disk inodes are part of a structural file, and

the structural file is stored in the non-volatile storage unit.

11. The non-transitory computer readable storage medium of claim 10, further comprising:

creating an in-core inode, wherein

the in-core inode corresponds to the structural file; and

storing the location of the unit of metadata in the in-core inode, wherein

the unit of metadata is a last-read chunk of metadata, and

the location of the unit of metadata identifies an end offset of the unit of metadata.

12. The non-transitory computer readable storage medium of claim 11, wherein
determining whether the metadata read-ahead operation can be performed comprises
accessing the end offset of the unit of metadata, and
determining whether the another location is adjacent to the end offset.

13. The non-transitory computer readable storage medium of claim 12, wherein
issuing the metadata read-ahead operation comprises
updating the end offset by replacing the end offset with an another end offset of an another unit of metadata read by the metadata read-ahead operation if the another location is adjacent to the end offset, wherein
the another unit of metadata is an another last-read chunk of metadata; and
updating the end offset by replacing the end offset with an end offset of a current metadata read operation if the another location is not adjacent to the end offset.

14. The non-transitory computer readable storage medium of claim 10, further comprising:
intercepting a command to read one or more on-disk inodes, wherein
the one or more on-disk inodes are part of the plurality of on-disk inodes,
the intercepting is performed in response to an input/output (I/O) operation, and
the I/O operation results in the command;
analyzing an issued metadata read-ahead value in the metadata read-ahead operation, wherein
the analyzing comprises comparing the issued metadata read-ahead value and a chunk total in the command;
waiting for the I/O operation to complete, if the analyzing indicates the I/O operation should complete;
issuing an asynchronous metadata read-ahead instruction, if the analyzing indicates that an asynchronous metadata read-ahead instruction can be issued;

determining whether the I/O operation is complete;

generating a queue if the I/O operation is complete, wherein

the queue comprises one or more remaining chunks of metadata of the chunk total, and

the one or more remaining chunks of metadata data are not included in the asynchronous metadata read-ahead instruction; and

updating the chunk total in the metadata read-ahead operation, if the I/O operation is incomplete.

15. A system comprising:

one or more processors; and

a memory coupled to the one or more processors, wherein the memory stores program instructions executable by the one or more processors to:

determine a location of a unit of metadata in a metadata storage area, wherein the metadata storage area is in a non-volatile storage unit;

determine an another location in the metadata storage area, wherein the another location corresponds to a current metadata read operation;

determine whether a metadata read-ahead operation can be performed, using the location of the unit of metadata and the another location; and

in response to a determination that the metadata read-ahead operation can be performed, issuing the metadata read-ahead operation.

16. The system of claim 15, wherein

the metadata storage area comprises a plurality of on-disk inodes, wherein

the on-disk inodes comprise the unit of metadata,

the on-disk inodes are part of a structural file, and

the structural file is stored in the non-volatile storage unit.

17. The system of claim 16, further comprising:

creating an in-core inode, wherein

the in-core inode corresponds to the structural file; and

storing the location of the unit of metadata in the in-core inode, wherein

the unit of metadata is a last-read chunk of metadata, and

the location of the unit of metadata identifies an end offset of the unit of metadata.

18. The system of claim 17, wherein

determining whether the metadata read-ahead operation can be performed comprises

accessing the end offset of the unit of metadata, and

determining whether the another location is adjacent to the end offset.

19. The system of claim 18, wherein

issuing the metadata read-ahead operation comprises

updating the end offset by replacing the end offset with an another end offset of an another unit of metadata read by the metadata read-ahead operation if the another location is adjacent to the end offset, wherein

the another unit of metadata is an another last-read chunk of metadata; and

updating the end offset by replacing the end offset with an end offset of a current metadata read operation if the another location is not adjacent to the end offset.

20. The system of claim 16, further comprising:

intercepting a command to read one or more on-disk inodes, wherein

the one or more on-disk inodes are part of the plurality of on-disk inodes,

the intercepting is performed in response to an input/output (I/O) operation, and

the I/O operation results in the command;

analyzing an issued metadata read-ahead value in the metadata read-ahead operation, wherein

the analyzing comprises comparing the issued metadata read-ahead value and a chunk total in the command;

waiting for the I/O operation to complete, if the analyzing indicates the I/O operation should complete;

issuing an asynchronous metadata read-ahead instruction, if the analyzing indicates that an asynchronous metadata read-ahead instruction can be issued;

determining whether the I/O operation is complete;

generating a queue if the I/O operation is complete, wherein

the queue comprises one or more remaining chunks of metadata of the chunk total, and

the one or more remaining chunks of metadata data are not included in the asynchronous metadata read-ahead instruction; and

updating the chunk total in the metadata read-ahead operation, if the I/O operation is incomplete.

21. A method comprising:

accessing an inode;

determining a directory of the inode;

determining whether an entry for the directory exists in a global inode list;

if the entry exists in the global inode list, determining whether a file structure of the directory is sequential or non-sequential; and

if the entry does not exist in the global inode list, adding a new entry for the directory in the global inode list.

22. The method of claim 21, wherein

determining whether the directory exists in a global inode list comprises

searching the global inode list for the entry.

23. The method of claim 21, further comprising:

determining whether the directory comprises a sequential flag if the file structure of the directory is sequential; and

if the directory comprises the sequential flag, issuing a metadata read-ahead operation for one or more inodes of a plurality of inodes on an inode list.

24. The method of claim 23, further comprising:
fetching the inode list of the plurality of inodes of the directory if the file structure of the parent directory is non-sequential; and
issuing the metadata read-ahead operation for the one or more inodes on the inode list.

25. The method of claim 24, further comprising:
intercepting a command to read one or more inodes, wherein
the one or more inodes are part of the plurality of inodes,
the intercepting is performed in response to an input/output (I/O) operation, and
the I/O operation results in the command.

26. The method of claim of claim 25, further comprising:
analyzing an issued metadata read-ahead value in the metadata read-ahead operation, wherein
the analyzing comprises comparing the issued metadata read-ahead value and a chunk total in the command;
waiting for the I/O operation to complete, if the analyzing indicates the I/O operation should complete; and
issuing an asynchronous metadata read-ahead instruction, if the analyzing indicates that an asynchronous metadata read-ahead instruction can be issued.

27. The method of claim 26, further comprising:
determining whether the I/O operation is complete;
generating a queue if the I/O operation is complete, wherein
the queue comprises one or more remaining chunks of metadata of the chunk total, and

the one or more remaining chunks of metadata are not included in the asynchronous metadata read-ahead instruction; and

updating the chunk total in the metadata read-ahead operation, if the I/O operation is incomplete.

28. The method of claim 21, wherein

the inode is an on-disk inode.

29. A non-transitory computer readable storage medium comprising program instructions executable to:

access an inode;

determine a directory of the inode;

determine whether an entry for the directory exists in a global inode list;

if the entry exists in the global inode list, determine whether a file structure of the directory is sequential or non-sequential; and

if the entry does not exist in the global inode list, add a new entry for the directory in the global inode list.

30. The non-transitory computer readable storage medium of claim 29, wherein

determining whether the directory exists in a global inode list comprises searching the global inode list for the entry, and

the inode is an on-disk inode.

31. The non-transitory computer readable storage medium of claim 29, further comprising:

determining whether the directory comprises a sequential flag if the file structure of the directory is sequential;

if the directory comprises the sequential flag, issuing a metadata read-ahead operation for one or more inodes in the directory;

fetching the inode list of the plurality of inodes of the directory if the file structure of the parent directory is non-sequential; and

issuing the metadata read-ahead operation for the one or more inodes on the inode list.

32. The non-transitory computer readable storage medium of claim 31, further comprising:
intercepting a command to read one or more inodes, wherein
the one or more inodes are part of the plurality of inodes,
the intercepting is performed in response to an input/output (I/O) operation, and
the I/O operation results in the command.

33. The non-transitory computer readable storage medium of claim 32, further comprising:
analyzing an issued metadata read-ahead value in the metadata read-ahead operation, wherein
the analyzing comprises comparing the issued metadata read-ahead value and a chunk total in the command;
waiting for the I/O operation to complete, if the analyzing indicates the I/O operation should complete; and
issuing an asynchronous metadata read-ahead instruction, if the analyzing indicates that an asynchronous metadata read-ahead instruction can be issued.

34. The non-transitory computer readable storage medium of claim 33, further comprising:
determining whether the I/O operation is complete;
generating a queue if the I/O operation is complete, wherein
the queue comprises one or more remaining chunks of metadata of the chunk total, and
the one or more remaining chunks of metadata are not included in the asynchronous metadata read-ahead instruction; and
updating the chunk total in the metadata read-ahead operation, if the I/O operation is incomplete.

35. A system comprising:
one or more processors; and

a memory coupled to the one or more processors, wherein the memory stores program instructions executable by the one or more processors to:

access an inode;

determine a directory of the inode;

determine whether an entry for the directory exists in a global inode list;

if the entry exists in the global inode list, determine whether a file structure of the directory is sequential or non-sequential; and

if the entry does not exist in the global inode list, add a new entry for the directory in the global inode list.

36. The system of claim 35, wherein

determining whether the directory exists in a global inode list comprises searching the global inode list for the entry, and

the inode is an on-disk inode.

37. The system of claim 35, further comprising:

determining whether the directory comprises a sequential flag if the file structure of the directory is sequential;

if the directory comprises the sequential flag, issuing a metadata read-ahead operation for one or more inodes in the directory;

fetching the inode list of the plurality of inodes of the directory if the file structure of the parent directory is non-sequential; and

issuing the metadata read-ahead operation for the one or more inodes on the inode list.

38. The system of claim 37, further comprising:

intercepting a command to read one or more inodes, wherein

the one or more inodes are part of the plurality of inodes,

the intercepting is performed in response to an input/output (I/O) operation, and

the I/O operation results in the command.

39. The system of claim 38, further comprising:

analyzing an issued metadata read-ahead value in the metadata read-ahead operation, wherein

the analyzing comprises comparing the issued metadata read-ahead value and a chunk total in the command;

waiting for the I/O operation to complete, if the analyzing indicates the I/O operation should complete; and

issuing an asynchronous metadata read-ahead instruction, if the analyzing indicates that an asynchronous metadata read-ahead instruction can be issued.

40. The system of claim 39, further comprising:

determining whether the I/O operation is complete;

generating a queue if the I/O operation is complete, wherein

the queue comprises one or more remaining chunks of metadata of the chunk total, and

the one or more remaining chunks of metadata are not included in the asynchronous metadata read-ahead instruction; and

updating the chunk total in the metadata read-ahead operation, if the I/O operation is incomplete.

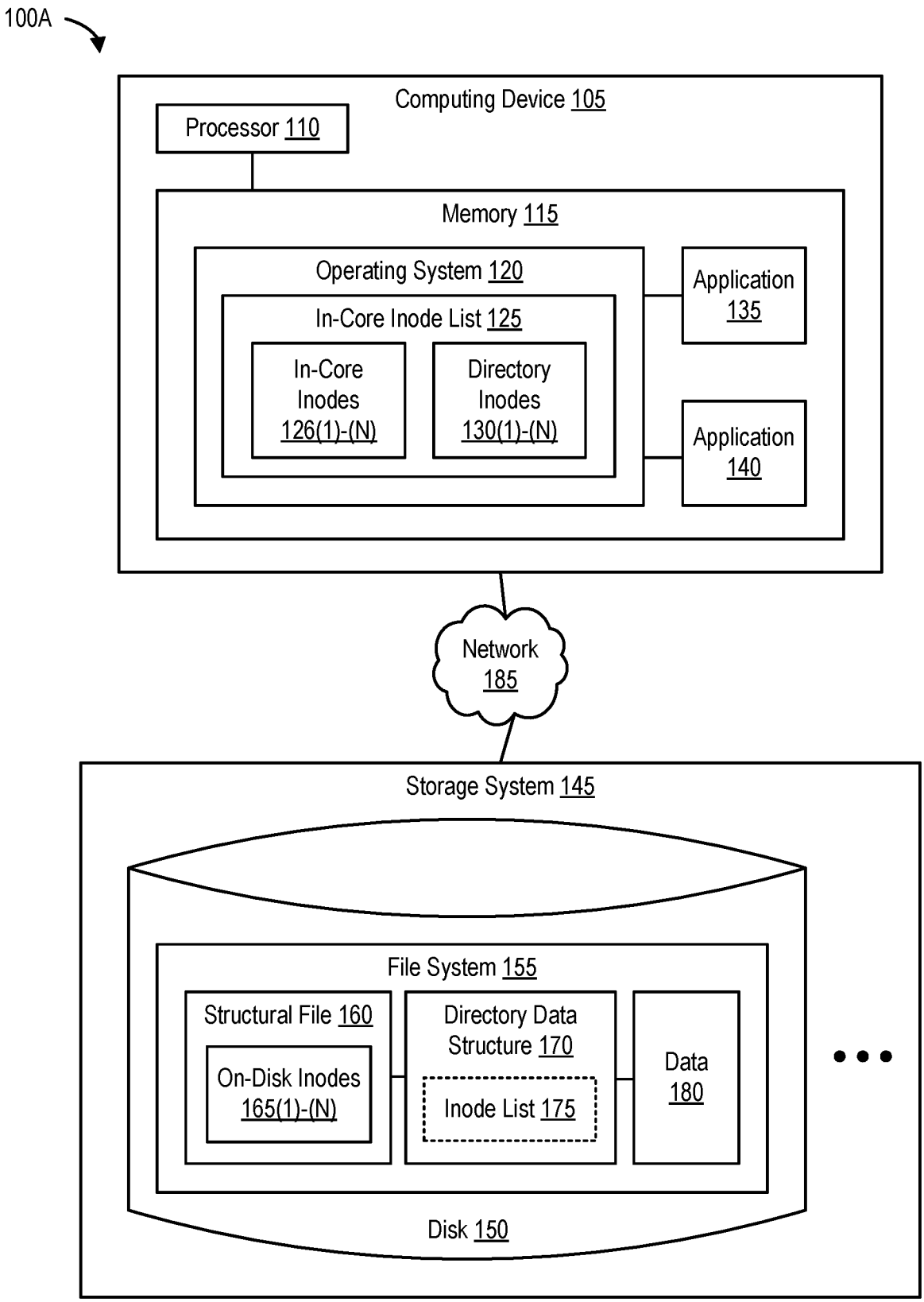

100A
Computing Device 105
Processor 110
Memory 115
Operating System 120
In-Core Inode List 125
In-Core Inodes 126(1)-(N)
Directory Inodes 130(1)-(N)
Application 135
Application 140
Network 185
Storage System 145
File System 155
Structural File 160
On-Disk Inodes 165(1)-(N)
Directory Data Structure 170
Inode List 175
Data 180
Disk 150

FIG. 1A

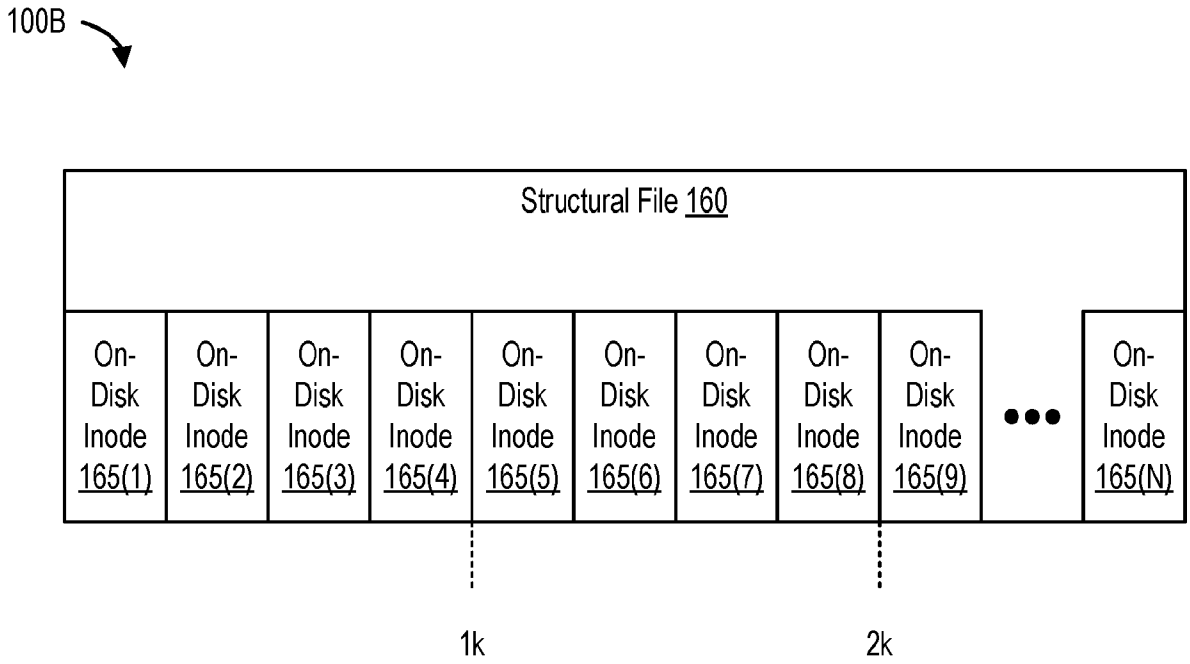

100B
Structural File 160
On-Disk Inode 165(1)
On-Disk Inode 165(2)
On-Disk Inode 165(3)
On-Disk Inode 165(4)
On-Disk Inode 165(5)
On-Disk Inode 165(6)
On-Disk Inode 165(7)
On-Disk Inode 165(8)
On-Disk Inode 165(9)
On-Disk Inode 165(N)
1k
2k

FIG. 1B

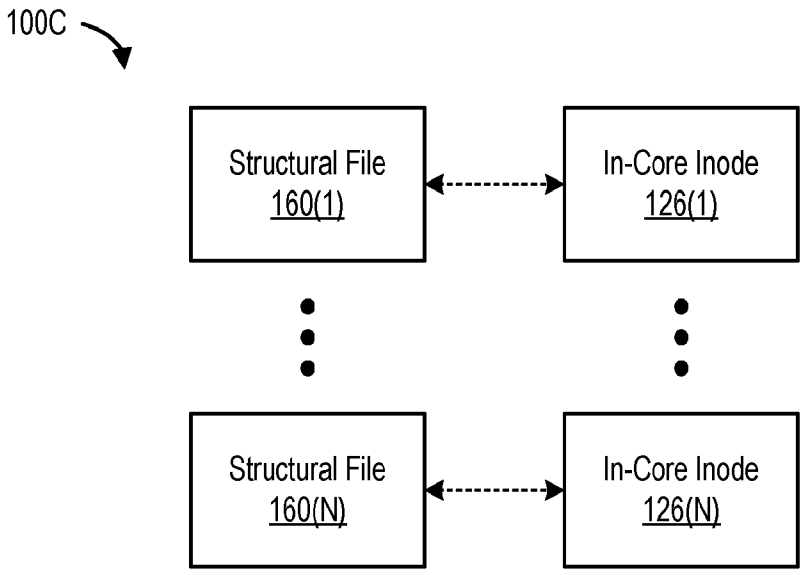

100C
Structural File 160(1)
In-Core Inode 126(1)
Structural File 160(N)
In-Core Inode 126(N)

FIG. 1C

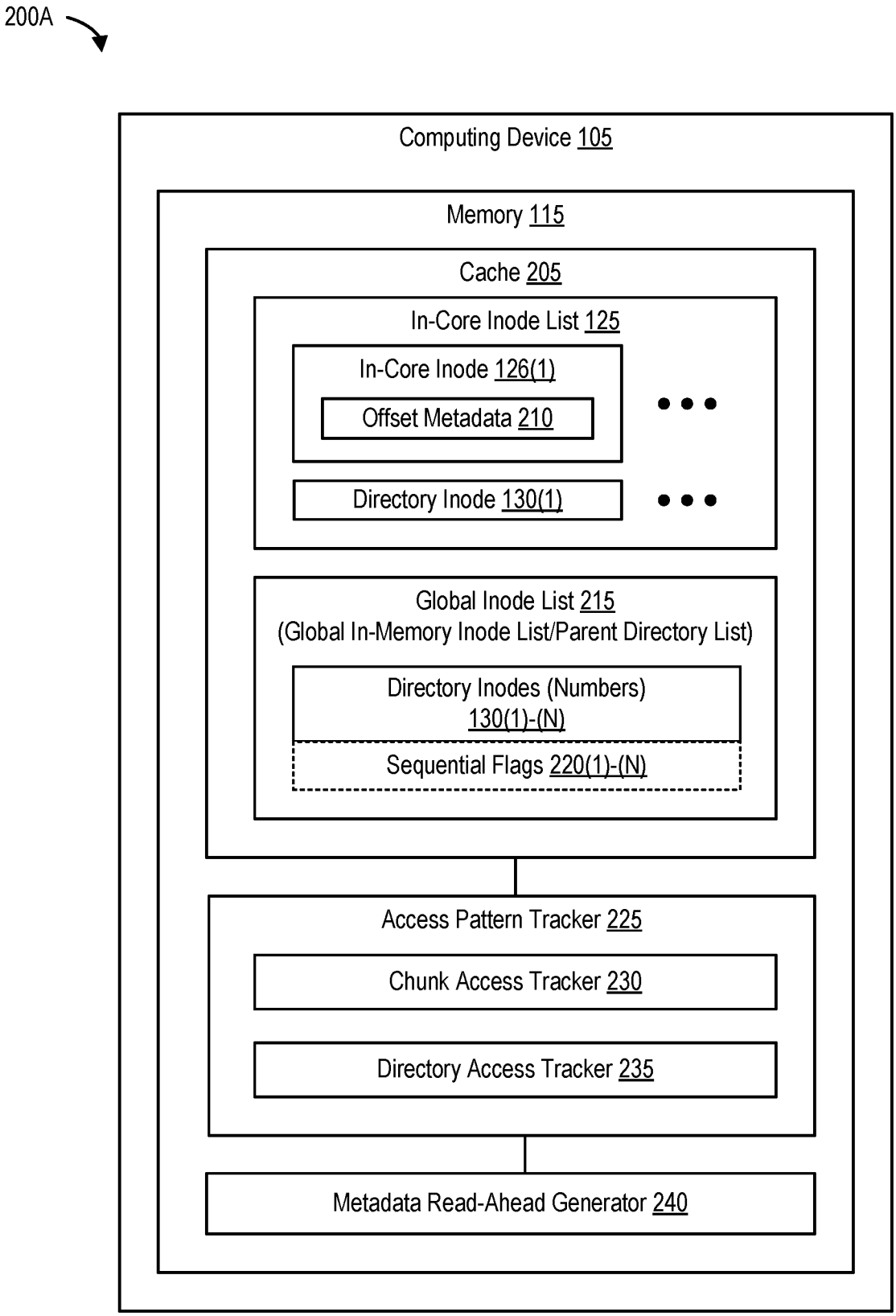

200A
Computing Device 105
Memory 115
Cache 205
In-Core Inode List 125
In-Core Inode 126(1)
Offset Metadata 210
Directory Inode 130(1)
Global Inode List 215
(Global In-Memory Inode List/Parent Directory List)
Directory Inodes (Numbers)
130(1)-(N)
Sequential Flags 220(1)-(N)
Access Pattern Tracker 225
Chunk Access Tracker 230
Directory Access Tracker 235
Metadata Read-Ahead Generator 240

FIG. 2A

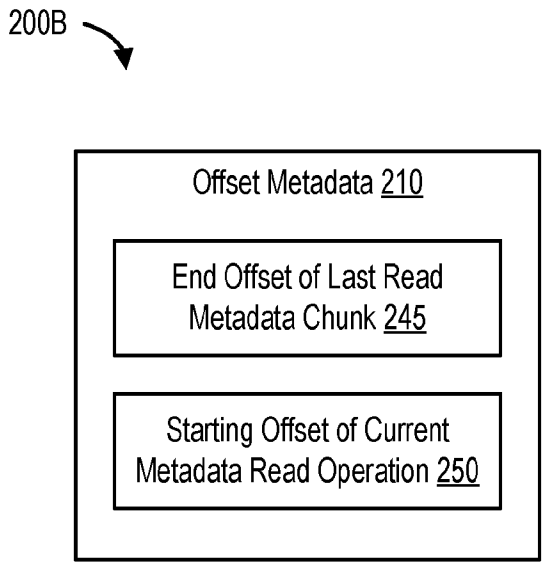


FIG. 2B

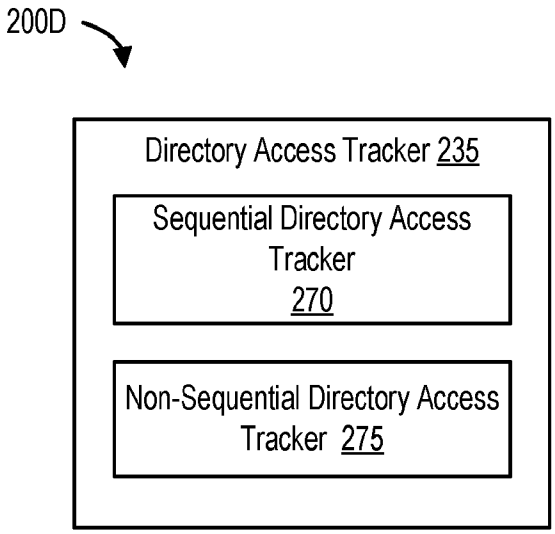


FIG. 2D

200C

Global Inode List 215 (for Directories 265(1)-(N))	
Directory Inode Field 255	Sequential Flag Field 260
Dir. Inode 130(1)	1
Dir. Inode 130(2)	0
Add?	Set?
Dir. Inode 130(4)	1
Dir. Inode 130(5)	0

FIG. 2C

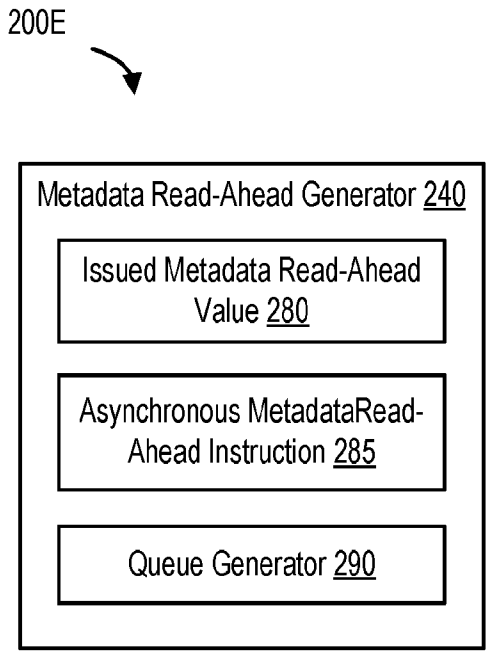


FIG. 2E

Sequential Inode Structure 300A

Directory 265(1)	
File Name Field 305	Inode No. Field 310
File 315(1)	On-Disk Inode 165(4)
File 315(2)	On-Disk Inode 165(5)
File 315(3)	On-Disk Inode 165(6)
File 315(4)	On-Disk Inode 165(7)
File 315(5)	On-Disk Inode 165(8)
File 315(6)	On-Disk Inode 165(9)

FIG. 3A

Non-Sequential Inode Structure 300B

Directory 265(2)	
File Name Field 305	Inode No. Field 310
File 315(1)	On-Disk Inode 165(4)
File 315(2)	On-Disk Inode 165(9)
File 315(3)	On-Disk Inode 165(15)
File 315(4)	On-Disk Inode 165(11)
File 315(5)	On-Disk Inode 165(19)
File 315(6)	On-Disk Inode 165(6)

FIG. 3B

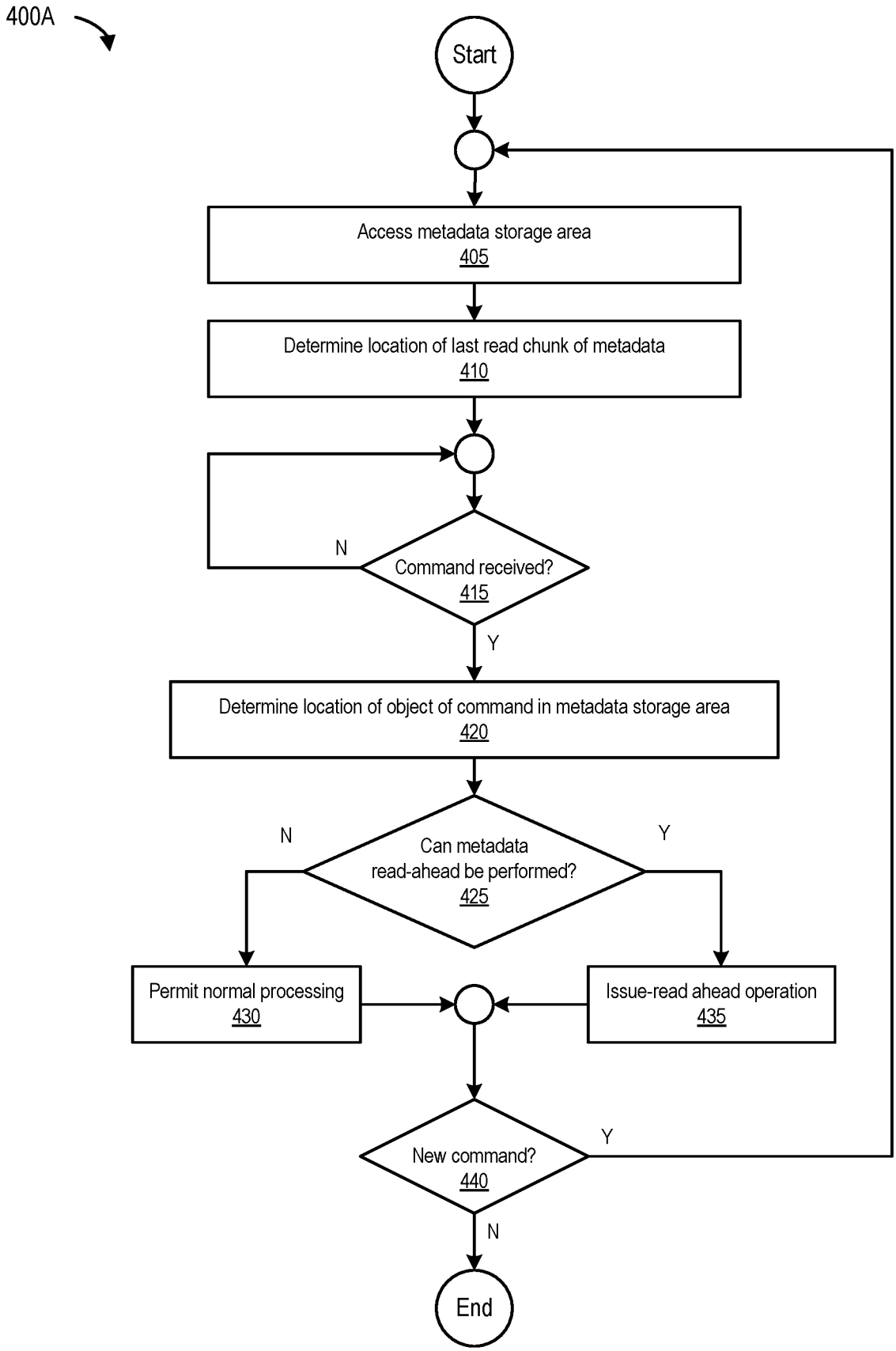
300C

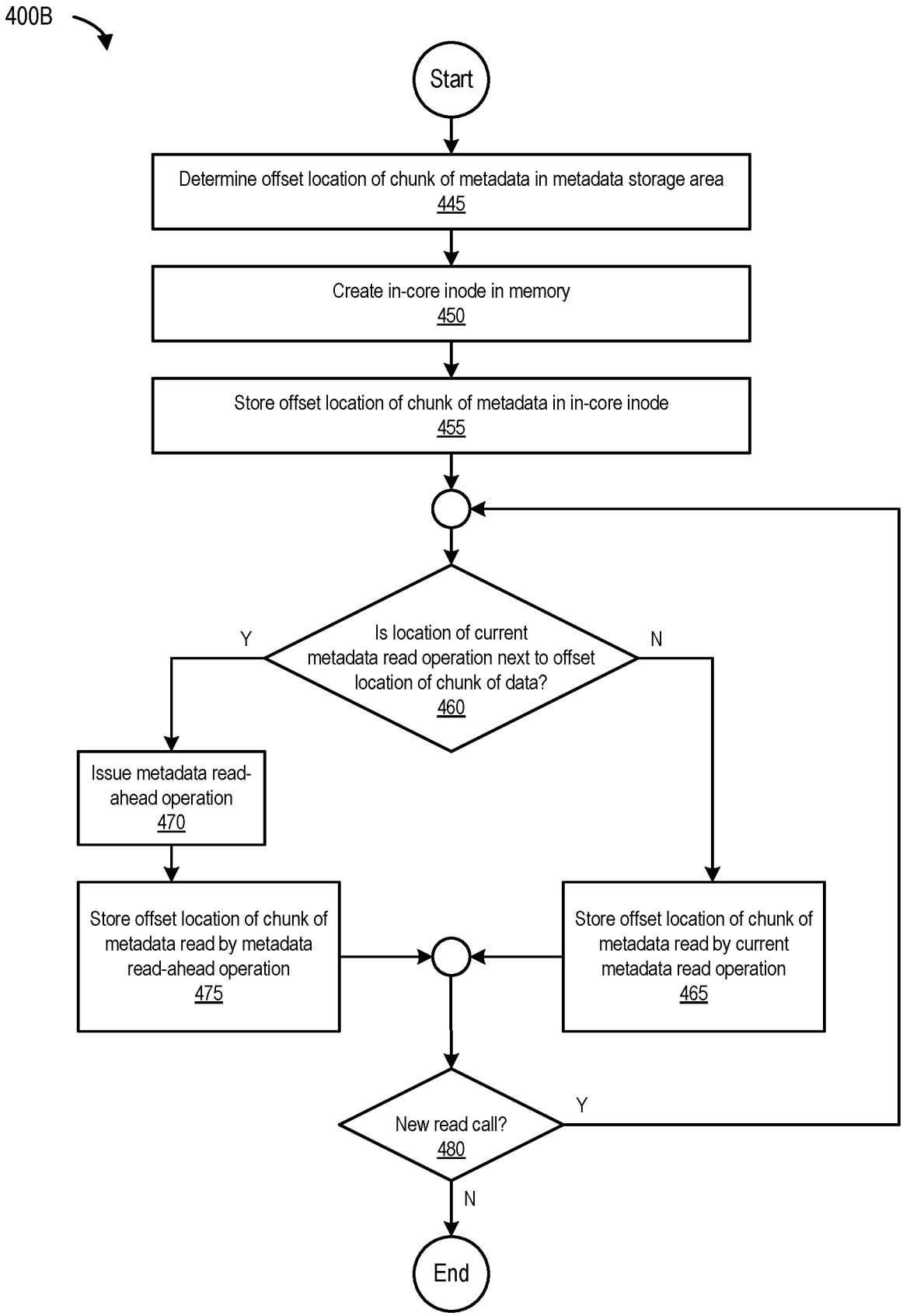
Cache 205

Parent Directory List

Dir. Inode 130(1)	**Dir. Inode 130(2)**	Add?	Dir. Inode 130(4)	Dir. Inode 130(5)
1	0	Set?	1	0

FIG. 3C


400A
Start
Access metadata storage area
405
Determine location of last read chunk of metadata
410
N
Command received?
415
Y
Determine location of object of command in metadata storage area
420
N
Can metadata
read-ahead be performed?
425
Y
Permit normal processing
430
Issue-read ahead operation
435
New command?
440
Y
N
End


400B
Start
Determine offset location of chunk of metadata in metadata storage area
445
Create in-core inode in memory
450
Store offset location of chunk of metadata in in-core inode
455
Is location of current metadata read operation next to offset location of chunk of data?
460
Y
N
Issue metadata read-ahead operation
470
Store offset location of chunk of metadata read by metadata read-ahead operation
475
Store offset location of chunk of metadata read by current metadata read operation
465
New read call?
480
Y
N
End

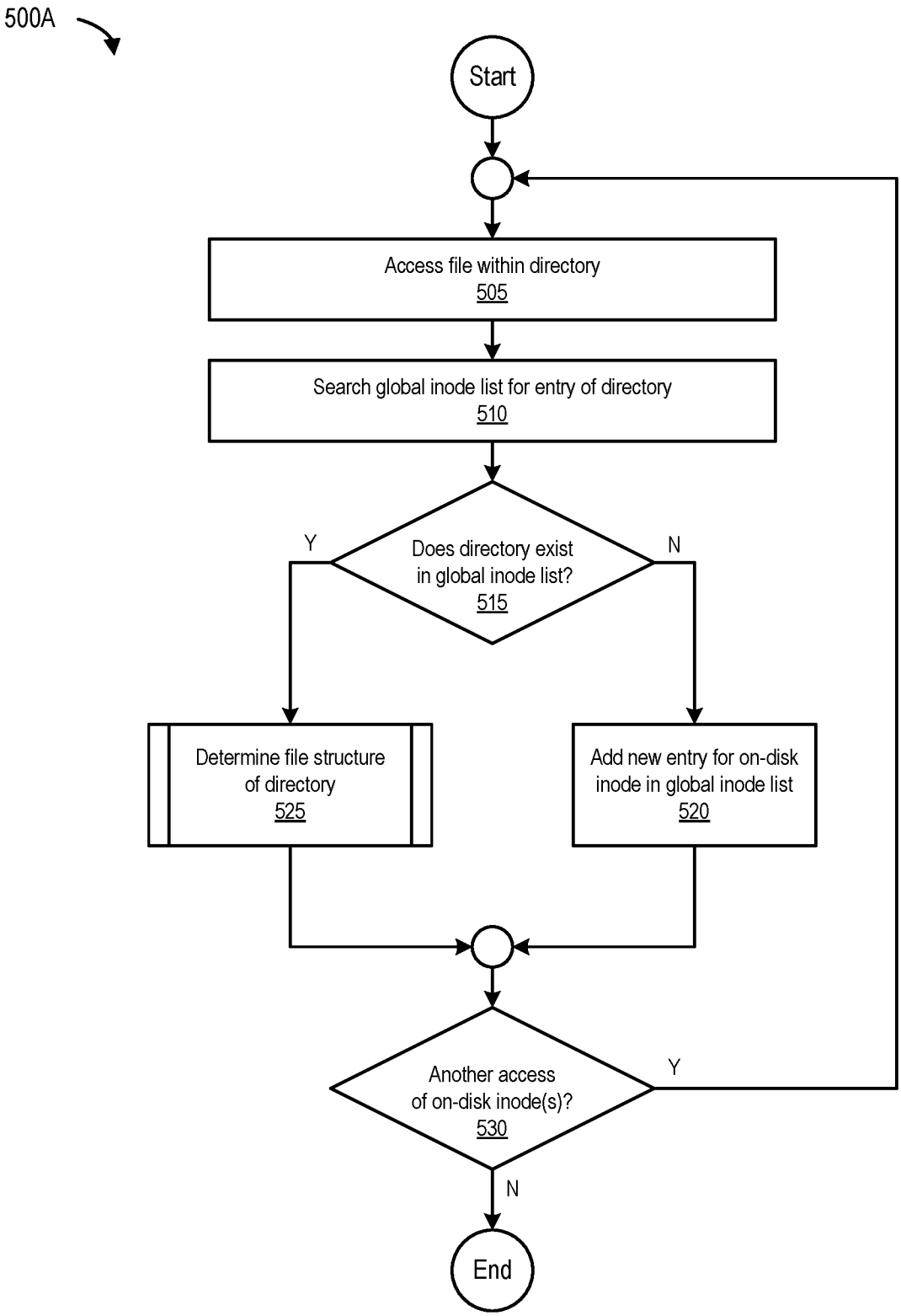
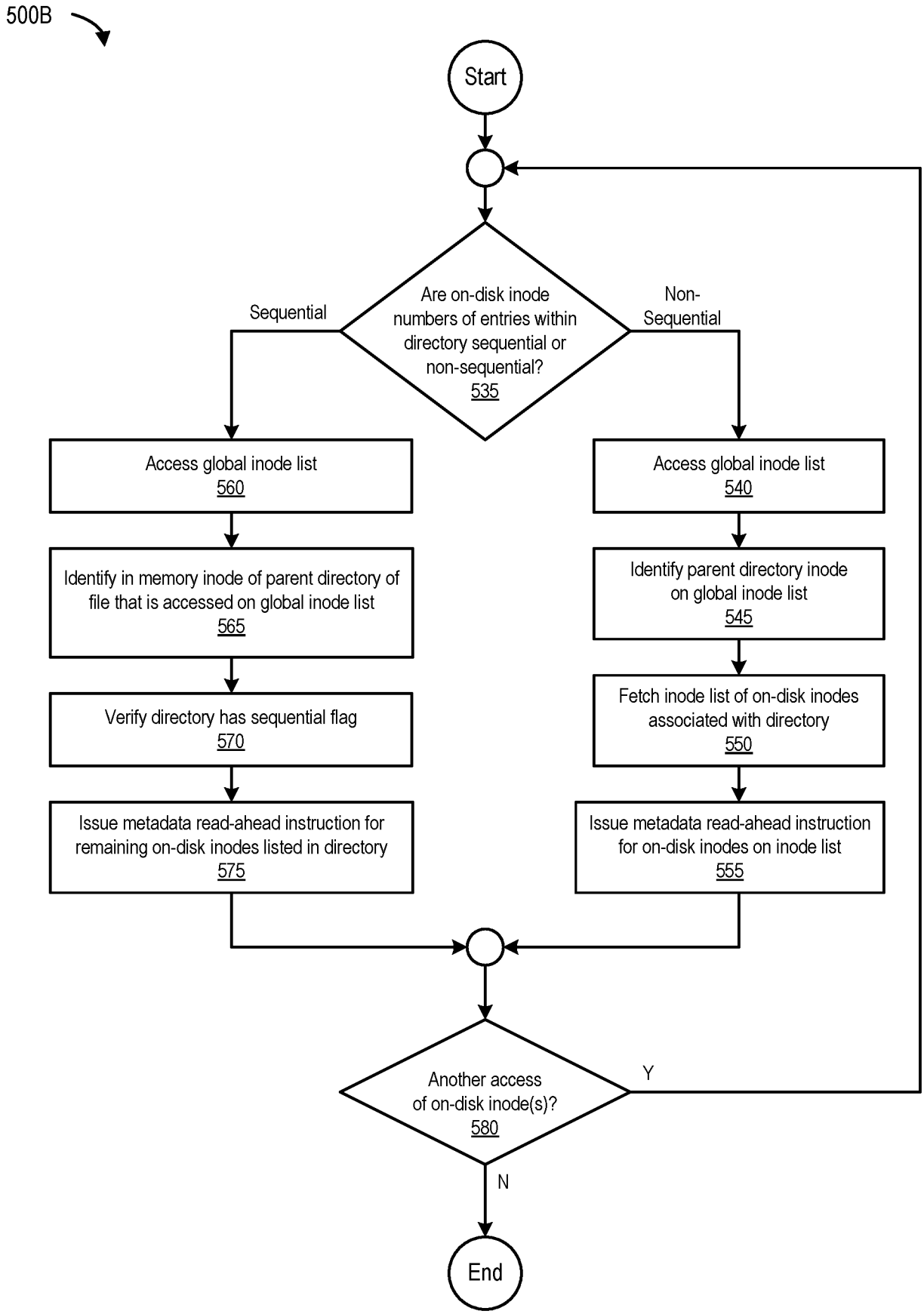
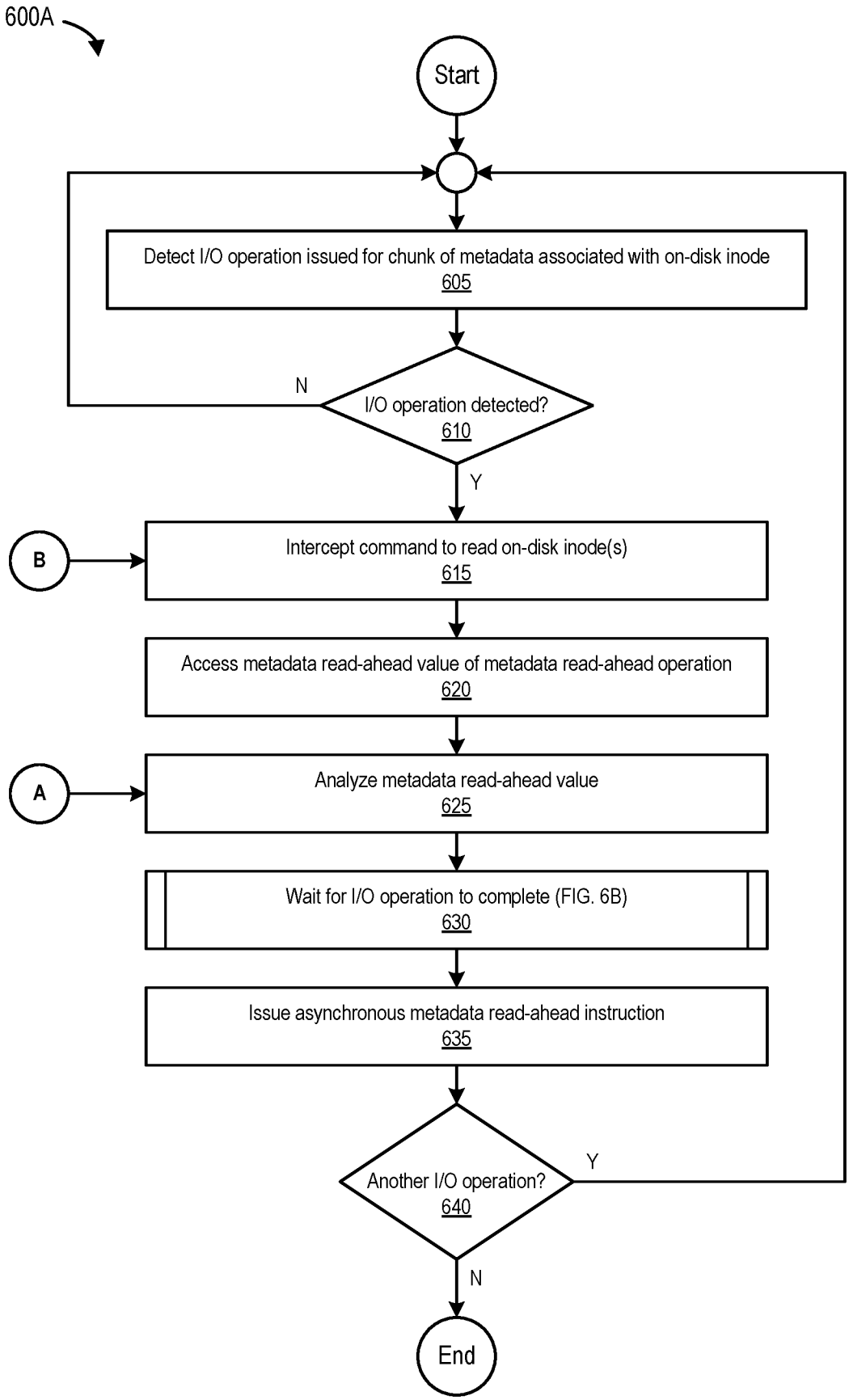

500A
Start
Access file within directory
505
Search global inode list for entry of directory
510
Does directory exist
in global inode list?
515
Y
N
Determine file structure
of directory
525
Add new entry for on-disk
inode in global inode list
520
Another access
of on-disk inode(s)?
530
Y
N
End

FIG. 5A


500B
Start
Are on-disk inode numbers of entries within directory sequential or non-sequential?
535
Sequential
Non-Sequential
Access global inode list
560
Identify in memory inode of parent directory of file that is accessed on global inode list
565
Verify directory has sequential flag
570
Issue metadata read-ahead instruction for remaining on-disk inodes listed in directory
575
Access global inode list
540
Identify parent directory inode on global inode list
545
Fetch inode list of on-disk inodes associated with directory
550
Issue metadata read-ahead instruction for on-disk inodes on inode list
555
Another access of on-disk inode(s)?
580
Y
N
End


600A
Start
Detect I/O operation issued for chunk of metadata associated with on-disk inode
605
N
I/O operation detected?
610
Y
B
Intercept command to read on-disk inode(s)
615
Access metadata read-ahead value of metadata read-ahead operation
620
A
Analyze metadata read-ahead value
625
Wait for I/O operation to complete (FIG. 6B)
630
Issue asynchronous metadata read-ahead instruction
635
Another I/O operation?
640
Y
N
End

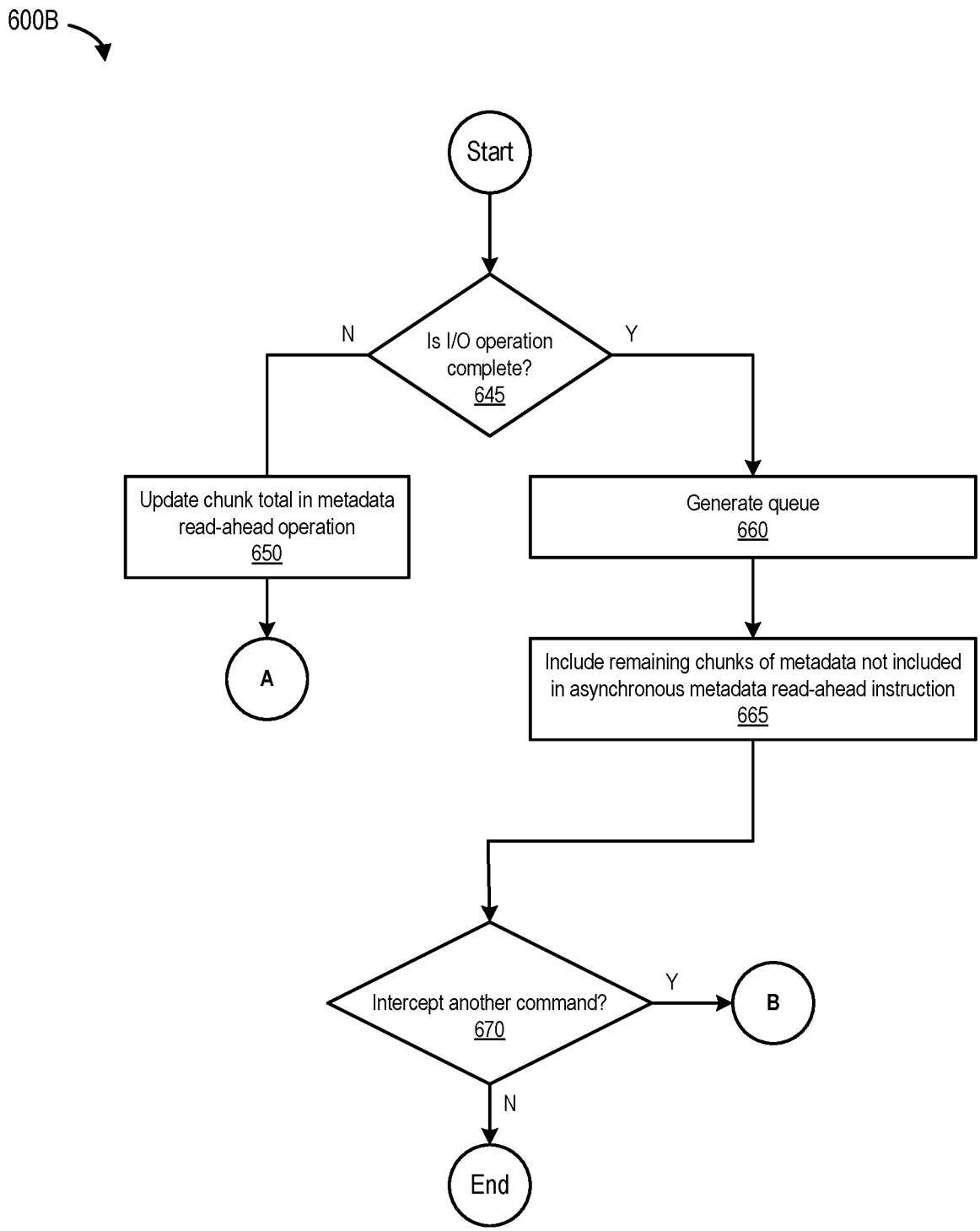
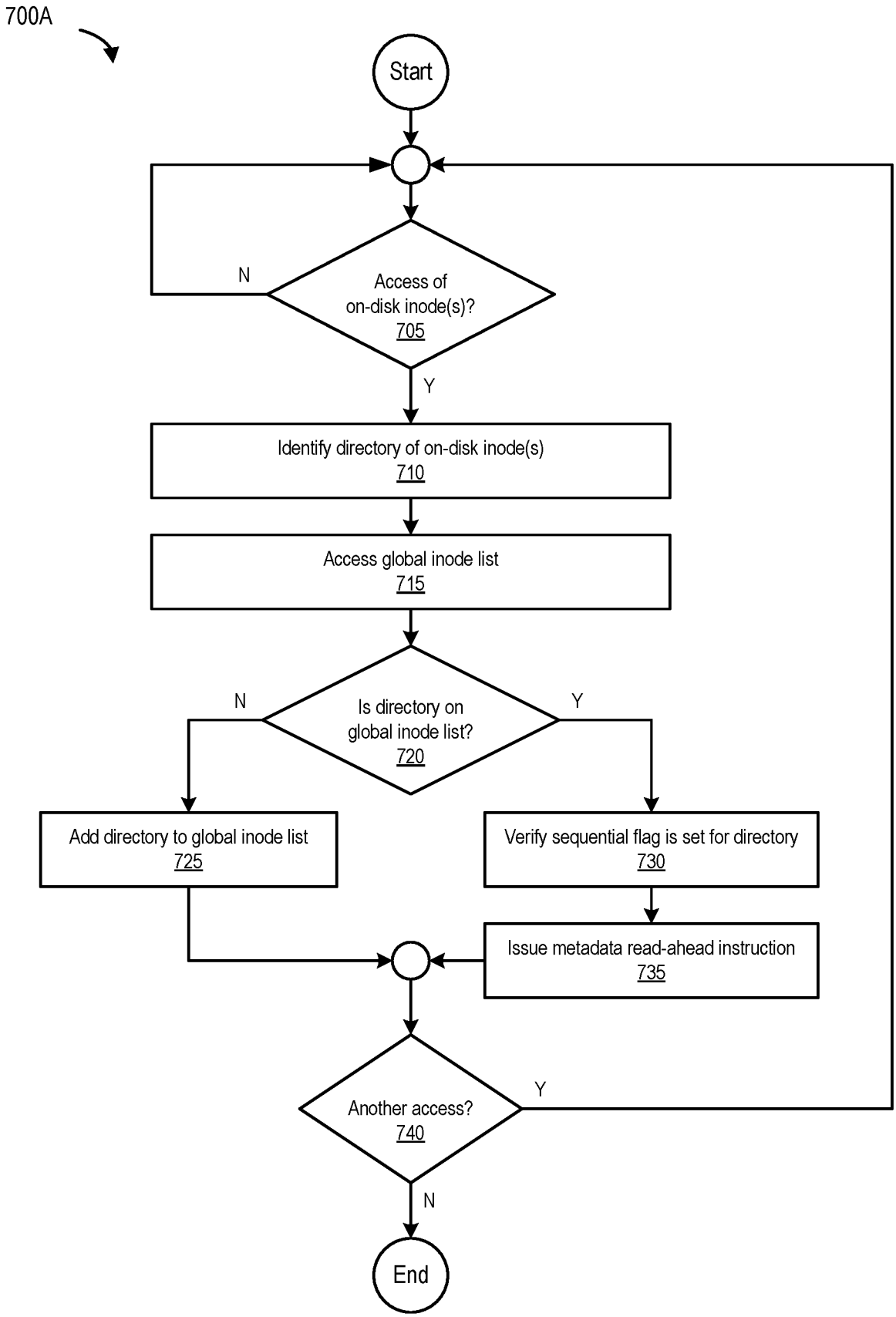

600B
Start
Is I/O operation complete?
645
N
Y
Update chunk total in metadata read-ahead operation
650
A
Generate queue
660
Include remaining chunks of metadata not included in asynchronous metadata read-ahead instruction
665
Intercept another command?
670
Y
B
N
End

FIG. 6B


700A
Start
Access of on-disk inode(s)? 705
N
Y
Identify directory of on-disk inode(s) 710
Access global inode list 715
Is directory on global inode list? 720
N
Y
Add directory to global inode list 725
Verify sequential flag is set for directory 730
Issue metadata read-ahead instruction 735
Another access? 740
Y
N
End

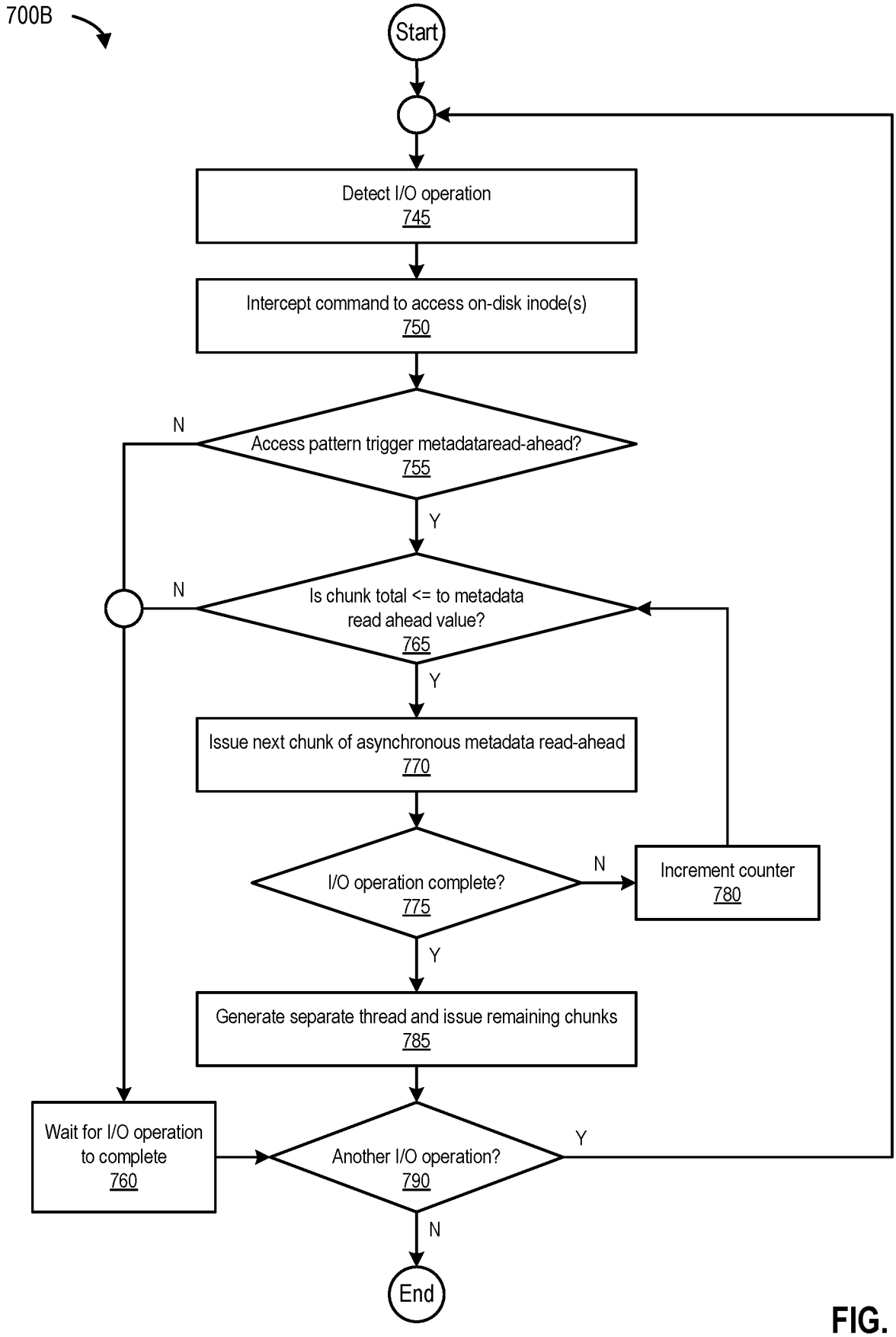

700B
Start
Detect I/O operation
745
Intercept command to access on-disk inode(s)
750
Access pattern trigger metadataread-ahead?
755
N
Y
Is chunk total <= to metadata read ahead value?
765
N
Y
Issue next chunk of asynchronous metadata read-ahead
770
I/O operation complete?
775
N
Increment counter
780
Y
Generate separate thread and issue remaining chunks
785
Wait for I/O operation to complete
760
Another I/O operation?
790
Y
N
End

FIG. 7B

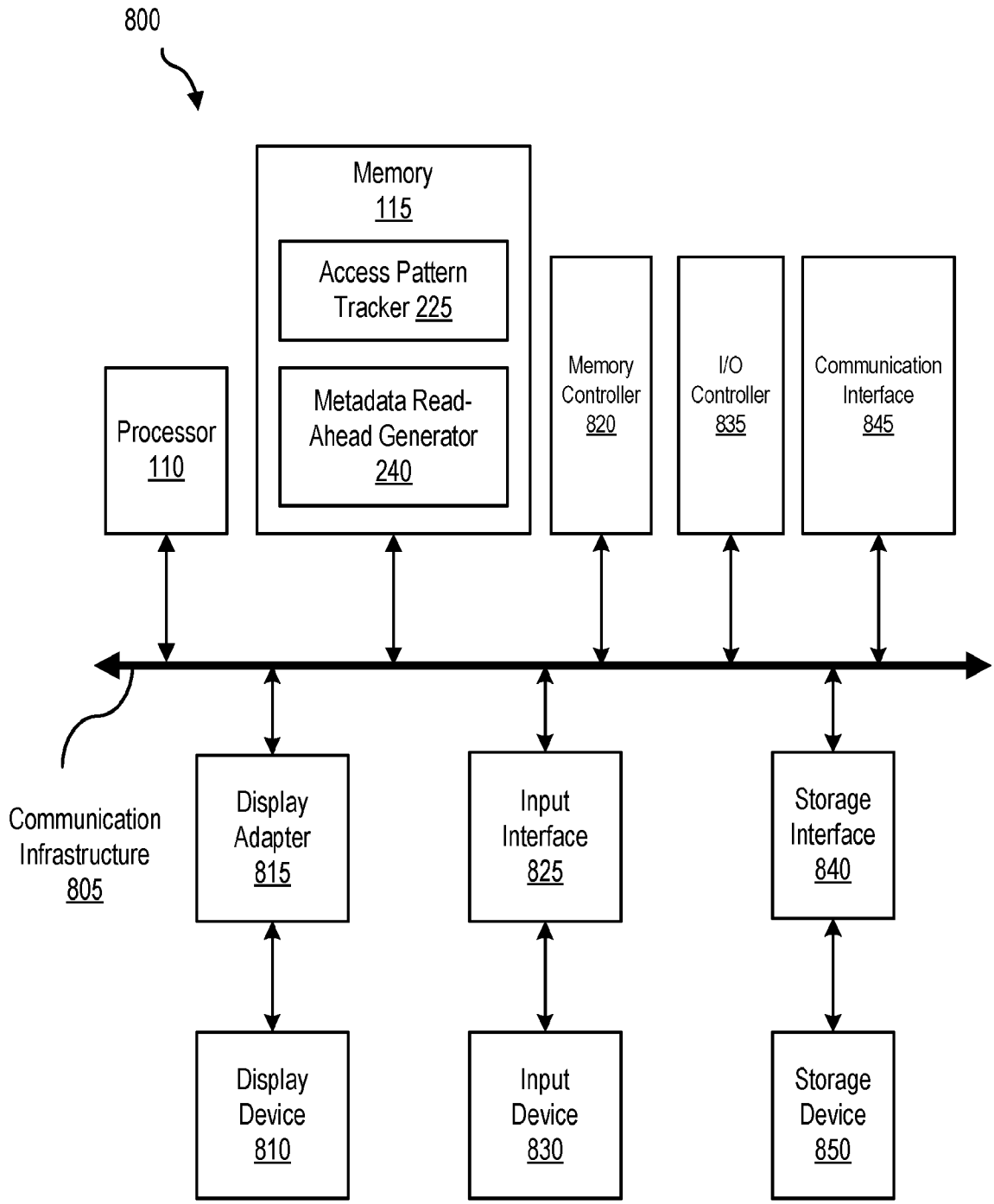

800
Memory
115
Access Pattern
Tracker 225
Metadata Read-
Ahead Generator
240
Processor
110
Memory
Controller
820
I/O
Controller
835
Communication
Interface
845
Communication
Infrastructure
805
Display
Adapter
815
Input
Interface
825
Storage
Interface
840
Display
Device
810
Input
Device
830
Storage
Device
850

FIG. 8

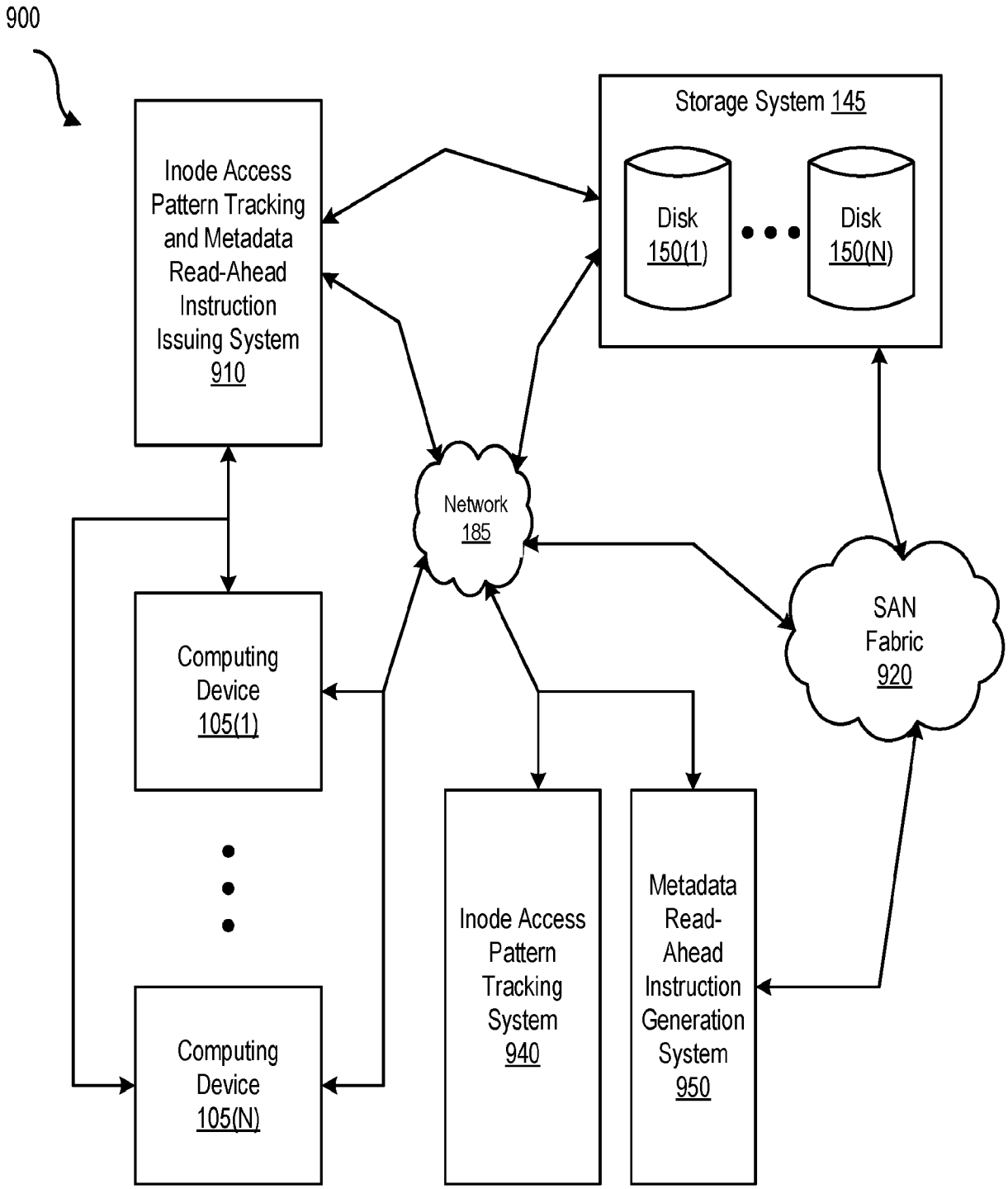

900
Storage System 145
Disk 150(1)
Disk 150(N)
Inode Access Pattern Tracking and Metadata Read-Ahead Instruction Issuing System 910
Network 185
SAN Fabric 920
Computing Device 105(1)
Computing Device 105(N)
Inode Access Pattern Tracking System 940
Metadata Read-Ahead Instruction Generation System 950

FIG. 9

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2017/053990

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F17/30 G06F12/0862
ADD.

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

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

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EPO-Internal, WPI Data, INSPEC, IBM-TDB

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 8 732 406 B1 (PASE DOUGLAS MICHAEL [US] ET AL) 20 May 2014 (2014-05-20) column 1, line 47 - column 2, line 58 column 3, line 56 - column 9, line 25	1-40
X	US 6 973 542 B1 (SCHMUCK FRANK B [US] ET AL) 6 December 2005 (2005-12-06) column 2, line 15 - column 7, line 15	1-40
A	US 2008/270706 A1 (FAIR ROBERT L [US] ET AL) 30 October 2008 (2008-10-30) paragraph [0022] - paragraph [0060]	1-40
A	EP 0 890 916 A2 (IBM [US]) 13 January 1999 (1999-01-13) page 22, line 35 - page 30, line 7 page 35, line 55 - page 39, line 39	1-40

☐ 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 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
23 November 2017	30/11/2017

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Warry, Lawrence

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/US2017/053990

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 8732406 B1	20-05-2014	NONE	
US 6973542 B1	06-12-2005	US 6973542 B1	06-12-2005
		US 2006077894 A1	13-04-2006
		US 2008126299 A1	29-05-2008
US 2008270706 A1	30-10-2008	US 2008270706 A1	30-10-2008
		WO 2008134004 A1	06-11-2008
EP 0890916 A2	13-01-1999	EP 0890916 A2	13-01-1999
		TW 412692 B	21-11-2000
		US 5974424 A	26-10-1999