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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
12 January 2017 (12.01.2017)

WIPO | PCT

(10) International Publication Number
WO 2017/007496 A1

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

(21) **International Application Number**:
PCT/US2015/047351

(22) **International Filing Date**:
28 August 2015 (28.08.2015)

(25) **Filing Language**: English

(26) **Publication Language**: English

(30) **Priority Data**:
3518/CHE/2015 9 July 2015 (09.07.2015) IN

(71) **Applicant**: **HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP** [US/US]; 11445 Compaq Center Drive West, Houston, TX 77070 (US).

(72) **Inventors**: **KARUPPUSAMY, Ramesh Kannan**; Sy.No.192, Whitefield Road, Mahadevapura Post, Bangalore, Karnataka 560048 (IN). **KANNAN, Rajkumar**; Sy.No.192, Whitefield Road, Mahadevapura Post, Bangalore, Karnataka 560048 (IN). **KOOMTHANAM, Annmary Justine**; Sy.No.192, Whitefield Road, Mahadevapura Post, Bangalore, Karnataka 560048 (IN). **SIVASHANMUGAM, Jothivelavan**; Sy.No.192, Whitefield Road, Mahadevapura Post, Bangalore, Karnataka 560048 (IN).

(74) **Agents**: **ORTEGA, Arthur S.** et al.; Hewlett Packard Enterprise, 3404 E. Harmony Road, Mail Stop 79, Fort Collins, CO 80528 (US).

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

(84) **Designated States** *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, 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,

[Continued on next page]

(54) **Title**: MANAGING A DATABASE INDEX FILE

(57) **Abstract**: Some examples relate to managing a database index file. In an example, upon receiving an update for a file, duplicate inode of an inode of the file may be generated, the duplicate inode points to blocks pointed to by the inode of the file. A new block may be allocated to write the update, the new block is linked to last unmodified block of the blocks pointed to by the inode and the duplicate inode. Position of file pointer in the blocks pointed to by the inode and the duplicate inode may be determined. If the pointer is not at an end of a block in the blocks pointed to by the inode and the duplicate inode, unmodified content is copied from the block to the new block till the pointer is at the end of the block. The position of pointer in the new block may be determined.

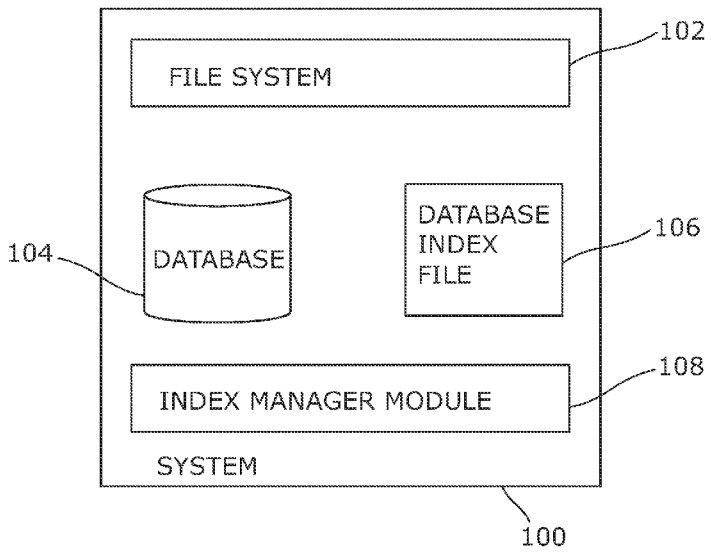

Fig. 1

WO 2017/007496 A1

SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

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

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

— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

Published:

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

MANAGING A DATABASE INDEX FILE

Background

[001] Databases may be part of modern day computing. Whether it is a small start-up or a large enterprise, organizations may have to deal with a vast amount of data these days, which could range from a few terabytes to multiple petabytes of data. Databases provide a useful way of organizing data. Such data may be accessed via a database management system that may allow entry, storage and retrieval of data from a database.

Brief Description of the Drawings

[002] The following detailed description references the drawings, wherein:

[003] FIG. 1 is a block diagram of an example system for managing a database index file;

[004] FIG. 2 is a block diagram of an example computer system for managing a database index file;

[005] FIG. 3 is a flowchart of an example method of managing a database index file;

[006] FIGS. 4A to 4G are conceptual block diagrams of an example method of managing a database index file; and

[007] FIG. 5 is a block diagram of an example computer system for managing a database index file.

Detailed Description

[008] Data management may be used by an organization. Whether it is a private company, a government undertaking, an educational institution, or a new start-up, managing data in an appropriate manner may be important of an enterprise especially in a business environment (for example, e-commerce) where a database may be growing rapidly. This may lead to many challenges in managing a database. Once of such challenges is to manage an index of the columns in a database. With large databases, indexing may be important to quick data access. As the size of a database grows, the size of an index file may also grow in proportion to the database size. This may make updating an index file challenging if the indexed field changes. For example, changing value of one index field may result in rewriting the whole index file as the field may have to be deleted from its old location and placed in another location in order to maintain sort order of the indexed field in the index file. In addition, an index file update may involve acquiring various types of locks to synchronize between various readers and the process that manages the index. This may not only impact performance but also require a complex algorithm to achieve synchronization.

[009] To address this issue, the present disclosure describes various examples for managing a database index file. In an example, upon receiving an update for a database index file, a duplicate inode of an inode of the database index file may be generated, wherein the duplicate inode also points to blocks pointed to by the inode of the database index file. In an instance, a new block may be allocated to write the update, wherein the new block is linked to last unmodified block of the blocks pointed to by the inode and the duplicate inode. The update may be written to the new block. Further, the position of file pointer in the blocks pointed to by the inode and the duplicate inode may be determined. If the file pointer is not at an end of a block in the blocks pointed to by the inode and the duplicate inode, unmodified content may be copied from the block to the new block till the file pointer is at the

end of the block. Furthermore, the position of file pointer in the new block may be determined. If the file pointer is at end of the new block, the blocks that are further to the block in the blocks pointed to by the inode and the duplicate inode may be included in the duplicate inode of the database index file.

[0010] FIG. 1 is a block diagram of a system 100 for managing a database index file, according to an example. In an example, system 100 may represent any type of computing system capable of reading machine-executable instructions. Examples of system 100 may include, without limitation, a server, a desktop computer, a notebook computer, a tablet computer, a thin client, a mobile device, a personal digital assistant (PDA), a phablet, and the like. In an example, system 100 may be a database server.

[0011] In an example, system 100 may include a file system 102, a database 104, a database index file 106, and an index manager module 108. In an example, system 100 may be a distributed computer system. In other words, "system" 100 may include multiple computing devices that may host one or more of the aforementioned components (for example, file system 102) that may interact with each other in order to achieve a common goal. The multiple computing devices may be in communication, for example, via a computer network. Such a computer network may be a wireless or wired network. Such a computer network may include, for example, a Local Area Network (LAN), a Wireless Local Area Network (WAN), a Metropolitan Area Network (MAN), a Storage Area Network (SAN), a Campus Area Network (CAN), or the like. Further, such a computer network may be a public network (for example, the Internet) or a private network (for example, an intranet).

[0012] The term "module" may refer to a software component (machine readable instructions), a hardware component or a combination thereof. A module may include, by way of example, components, such as software components, processes,

tasks, co-routines, functions, attributes, procedures, drivers, firmware, data, databases, data structures, Application Specific Integrated Circuits (ASIC) and other computing devices. A module may reside on a volatile or non-volatile storage medium and configured to interact with a processor of a system (e.g. 100).

[0013] In general, file system 102 may be used for storage and retrieval of data from system 100. In an example, a computer file or "file" may be considered as the basic component of file system 102. Each piece of data on a storage device may be called a "file". A file may contain data, such as text files, image files, video files, and the like, or it may be an executable file or program. Files in file system 102 may be organized by storing related files in a directory or sub-directory.

[0014] In an example, an inode may be associated with each file of the file system 102. An inode is a data structure, which may be used to represent a file system object (for example, directory, file, etc.). Each file or directory may be associated with an inode, which may be identified by an integer number (i.e. an inode number). An inode may store the attributes and disk block location(s) of a file system object's data. In other words, an inode may store information about data blocks associated with a file or directory, or it may point to a data block map that points to the data blocks. In an instance, an "inode block map" may hold page to disk block mapping for a file or directory. In other words, an inode block map may provide mapping information between a page of a file (or directory) and a Logical Block Number (LBN) on a storage system.

[0015] File system 102 may be a local file system or a scale-out file system such as a shared file system or a network file system. Examples of a shared file system may include a Storage Area Network (SAN) file system or a cluster file system. Examples of a network file system may include a distributed file system or a distributed parallel file system. Some non-limiting examples of file systems that may be used on storage device (example, 102) may include FAT (FAT12, FAT16, FAT32), NTFS, HFS

and HFS+, HPFS, UFS, ext2, ext3, and ext4. In an example, file system 102 may be an online file system.

[0016] Database 104 may be a repository that stores an organized collection of data, for instance, in form of data records. System 100 may perform a task or function related to data records stored in database 104. System 100 may, for example, store, modify, retrieve, query, and delete data records in database. In an instance, system may access data records from database 104 by using a query language such as Structured Query Language (SQL). In an example, database 104 may be a relational database, wherein data may be organized into one or more tables of rows and columns. Each entity type described in a database 104 may have its own table. The rows in table may represent instances of that type of entity and the columns may represent values attributed to that instance. In an instance, each data record in database 104 may include a key field, which helps it to be recognized uniquely.

[0017] System 100 may include a database index file 106. In another example, system 100 may include multiple database index files. In an instance, database index file 106 may be generated on data stored in database 104, and may include one or more entries, where each entry may correspond to a data record in database 104. The index entry associated with a data record may be a pair consisting of a value of the key attribute for that record and a pointer to the location where the data record is stored in database. Database index file 106 may be used to efficiently retrieve records from the database 104 based on an attribute(s) on which the indexing may be done. Database index file 106 may help quickly locate data in database without having to search every row in a database table every time a database table is accessed. In an instance, database index file 106 may be created by using one or more columns of a table in database 104.

[0018] Some of the example functionalities performed by file system 102, database index file 106, and index manager module 108 are described in reference to FIG. 2 below.

[0019] FIG. 2 is a block diagram of an example computing system 200 for managing a database index file. In an example, computing system 200 may be similar to system 100 of FIG. 1, in which like reference numerals correspond to the same or similar, though perhaps not identical, components. For the sake of brevity, components or reference numerals of FIG. 2 having a same or similarly described function in FIG. 1 are not being described in connection with FIG. 2. Said components or reference numerals may be considered alike.

[0020] Computing system 200 may be a server, a desktop computer, a notebook computer, a tablet computer, a mobile phone, a personal digital assistant (PDA), and the like.

[0021] In an example, computing system 200 may include a file system 102, a database index file 106, and an index manager module 108.

[0022] Index manager module 108 may generate a duplicate inode of the inode of a database index file (for example, 106). Index manager module 108 may create a copy of the inode of a database index file (for example, 106) upon receiving an update for the database index file (for example, 106). In an instance, each time an update for a database index file (for example, 106) is received, the index manager module 108 may create a copy of the inode of the database index file (for example, 106). The duplicate inode may have the attributes of the original inode. For example, the duplicate inode may also point to the blocks pointed to by the original inode of the database index file (for example, 106).

[0023] File system 102 may provide an interface to the index manager module 108 to create a copy of the inode. For example, the interface may include an ioctl system call or a Remote Procedure Call (RPC). In an instance, file system102 may mark the duplicate inode of the database index file (for example, 106) as a "special" inode to indicate that any update to such inode may require specific processing. Once index manager module 108 creates a copy of the inode of a database index file, file system 102 may allocate a new block to write an update received for the database index file (for example, 106). The new block may be linked to the last unmodified block of the blocks pointed to by the inode and the duplicate inode of the database index file (for example, 106).

[0024] In an instance, prior to allocating a new block for writing the update to the database index file (for example, 106), file system 102 may determine whether a write request for writing the update is a first write request. If the write request for writing the update is the first write request, file system 102 may allocate a new block to write the update. On the other hand, if the write request for writing the update is not the first write request, file system 102 may determine whether a new block is needed for writing the update. In other words, in such case, the file system 102 may determine whether a current write request may be written on a block previously allocated by it. In case, upon such determination, the file system 102 determines that an additional block is needed for writing the update, file system may allocate an additional new block for writing the update. Once the file system 102 makes the appropriate determination i.e. whether to allocate a new block or use a previously allocated block for writing the current update, file system may write the update to the new block or an earlier block, as the case may be. File system 102 may keep allocating new blocks to honor write requests until the update is completed.

[0025] Once the update is complete, file system 102 may determine position of file pointer in the blocks that are pointed to by the inode and the duplicate inode of the

database index file (for example, 106). If the file pointer is not at an end of a block in the blocks pointed to by the inode and the duplicate inode, file system 102 may copy unmodified content from the current file pointer position in the block to the new block (or a previously allocated block, as the case may be) till the file pointer is at the end of the block.

[0026] In an instance, file system 102 may further determine position of file pointer in the new block (or a previously allocated block, as the case may be) that received the update. If the file pointer is at end of the new block (or a previously allocated block, as the case may be) that received the update, file system 102 may include blocks that are further to the block (i.e. block with file pointer at the end) in the blocks pointed to by the inode and the duplicate inode of the database index file, in the duplicate inode of the database index file. In other words, if the file pointer is at end of the new block or a previously allocated block that received the update, file system 102 may reuse the remaining blocks in the blocks pointed to by the inode and the duplicate inode, instead of copying data from the remaining blocks to the duplicate index file. The remaining blocks may be included in the duplicate inode of the database index file. File system 102 may then choose to close the file pointers.

[0027] Index manager module 108 may now determine whether an application is using the inode of the database index file. In other words if there's a "reader" for the original inode of the database index file. If no application is using the inode of the database index file, the index manager module 108 may delete the original inode of the database index file. However, if an application is reading the inode of the database index file, index manager module 108 may mark the inode of the database index file for deletion, and delete the inode of the database index file after the application completes reading the inode of the database index file. In an instance, the original inode of the database index file may be deleted by the application that last reads the original inode. Once the original inode is deleted, file system 102 may

use the duplicate inode of the database index file for reading the database index file in future.

[0028] FIG. 3 is a flowchart of an example method 300 for managing a database index file. The method 300, which is described below, may at least partially be executed on system 100 of FIG. 1 or computing system 200 of FIG. 2. However, other computing devices may be used as well. At block 302, a duplicate inode of an inode of a database index file may be generated upon receiving an update for a database index file, wherein the duplicate inode also points to blocks pointed to by the inode of the database index file. This is illustrated in FIG. 4A, according to an example. FIG. 4A shows a duplicate inode 402 of an inode 404 of a database index file, wherein the duplicate inode 402 points to blocks 406, 408, 410 and 412 pointed to by the inode 404 of the database index file. At block 304, a new block may be allocated to write the update, wherein the new block is linked to last unmodified block of the blocks pointed to by the inode and the duplicate inode. This is illustrated in FIG. 4B, according to an example. FIG. 4B shows that a new block 414 is allocated to write an update, wherein the new block 414 is linked to last unmodified block 406 of the blocks 408, 410, and 412 pointed to by the inode 404 and the duplicate inode 402. At block 306, the update may be written to the new block 414. In an example, additional new blocks 416 and 418 may be allocated, if needed, to complete all write requests related to the update. At block 308, the position of file pointer in the blocks pointed to by the inode and the duplicate inode may be determined. This is illustrated in FIG. 4B, according to an example. FIG. 4B shows the position of file pointer 420 in the blocks pointed to by the inode and the duplicate inode. At block 310, if the file pointer is not at an end of a block in the blocks pointed to by the inode and the duplicate inode, unmodified content may be copied from the block to the new block till the file pointer is at the end of the block. This is illustrated in FIGS. 4C and 4D, according to an example. FIGS. 4C and 4D show that if the file pointer 420 is not at an end of a block in the blocks pointed to by the inode and the duplicate inode, unmodified content may be copied from block 410 to the new block

418 till the file pointer is at the end of the block 410. At block 312, the position of file pointer in the new block 418 may be determined. At block 314, if the file pointer is at the end of the new block 418, blocks that are further to the block in the blocks pointed to by the inode and the duplicate inode may be included in the duplicate inode of the database index file. This is illustrated in FIG. 4E, according to an example. FIG. 4E shows that the file pointer 422 is at the end of the new block 418. In this case, block 412, which is further to the block 410 in the blocks pointed to by the inode and the duplicate inode, may be included in the duplicate inode 404 of the database index file. On the other hand, if the file pointer 422 is not at the end of the last allocated new block 418 then beginning from the existing position of file pointer in the inode, remaining data of the inode may be copied to the duplicate inode of the database index file. The remaining data may be copied to the duplicate inode commencing from the existing position of file pointer in the duplicate inode. This is illustrated in FIGS. 4F and 4G, according to an example. In FIG. 4F, file pointer 422 is not at the end of the last new block 418. In this case, beginning from the existing position of file pointer 420 in the inode 404, remaining data (in blocks 410 and 412) of the inode 404 is copied to the duplicate inode 402 of the database index file. The remaining data may be copied to a new block(s) 418 and 424 of the duplicate inode 402 commencing from the existing position of file pointer 422 (as shown in FIG. 4F) in the duplicate inode 402. FIG. 4G reflects the current position of file pointer 422 in the duplicate inode 402 after the remaining data (in blocks 410 and 412) of the inode 404 is copied to the duplicate inode 402.

[0029] FIG. 5 is a block diagram of an example system 500 for managing a database index file. System 500 includes a processor 502 and a machine-readable storage medium 504 communicatively coupled through a system bus. Processor 502 may be any type of Central Processing Unit (CPU), microprocessor, or processing logic that interprets and executes machine-readable instructions stored in machine-readable storage medium 504. Machine-readable storage medium 504 may be a random access memory (RAM) or another type of dynamic storage device that may store

information and machine-readable instructions that may be executed by processor 502. For example, machine-readable storage medium 504 may be Synchronous DRAM (SDRAM), Double Data Rate (DDR), Rambus DRAM (RDRAM), Rambus RAM, etc. or a storage memory media such as a floppy disk, a hard disk, a CD-ROM, a DVD, a pen drive, and the like. In an example, machine-readable storage medium 504 may be a non-transitory machine-readable medium. Machine-readable storage medium 504 may store instructions 506, 508, 510, 512, 514, 516, and 518.

[0030] In an example, instructions 506 may be executed by processor 502 to create, upon receiving an update for a database index file, a duplicate inode of an inode of the database index file, wherein the duplicate inode also points to data blocks pointed to by the inode of the database index file. Instructions 508 may be executed by processor 502 to allocate a new data block to write the update, wherein the new data block is linked to last unmodified data block of the blocks pointed to by the inode and the duplicate inode. Instructions 510 may be executed by processor 502 to write the update to the new data block. Instructions 512 may be executed by processor 502 to determine position of file pointer in the data blocks pointed to by the inode and the duplicate inode. Instructions 514 may be executed by processor 502 to copy, if the file pointer is not at an end of a current data block in the data blocks pointed to by the inode and the duplicate inode, unmodified content from the current data block to the new data block till the file pointer is at the end of the data block. Instructions 516 may be executed by processor 502 to determine position of file pointer in the new data block. Instructions 518 may be executed by processor 502 to include, if the file pointer is at end of the new data block, data blocks that are further to the current data block in the data blocks pointed to by the inode and the duplicate inode, in the duplicate inode of the database index file.

[0031] For the purpose of simplicity of explanation, the example method of FIG. 3 is shown as executing serially, however it is to be understood and appreciated that the present and other examples are not limited by the illustrated order. The example

systems of FIGS. 1, 2 and 5, and method of FIG. 3 may be implemented in the form of a computer program product including computer-executable instructions, such as program code, which may be run on any suitable computing device in conjunction with a suitable operating system (for example, Microsoft Windows, Linux, UNIX, and the like). Examples within the scope of the present solution may also include program products comprising non-transitory computer-readable media for carrying or having computer-executable instructions or data structures stored thereon. Such computer-readable media can be any available media that can be accessed by a general purpose or special purpose computer. By way of example, such computer-readable media can comprise RAM, ROM, EPROM, EEPROM, CD-ROM, magnetic disk storage or other storage devices, or any other medium which can be used to carry or store desired program code in the form of computer-executable instructions and which can be accessed by a general purpose or special purpose computer. The computer readable instructions can also be accessed from memory and executed by a processor.

[0032] It should be noted that the above-described examples of the present solution is for the purpose of illustration only. Although the solution has been described in conjunction with a specific example thereof, numerous modifications may be possible without materially departing from the teachings and advantages of the subject matter described herein. Other substitutions, modifications and changes may be made without departing from the spirit of the present solution. All of the features disclosed in this specification (including any accompanying claims, abstract and drawings), and/or all of the steps of any method or process so disclosed, may be combined in any combination, except combinations where at least some of such features and/or steps are mutually exclusive.

Claims:

1. A method of managing a database index file, comprising:

generating, upon receiving an update for a database index file, a duplicate inode of an inode of the database index file, wherein the duplicate inode also points to blocks pointed to by the inode of the database index file;

allocating a new block to write the update, wherein the new block is linked to last unmodified block of the blocks pointed to by the inode and the duplicate inode;

writing the update to the new block;

determining position of file pointer in the blocks pointed to by the inode and the duplicate inode;

if the file pointer is not at an end of a block in the blocks pointed to by the inode and the duplicate inode, copying unmodified content from the block to the new block till the file pointer is at the end of the block;

determining position of file pointer in the new block; and

if the file pointer is at end of the new block, including blocks that are further to the block in the blocks pointed to by the inode and the duplicate inode, in the duplicate inode of the database index file.

2. The method of claim 1, further comprising:

if the file pointer is not at the end of the new block, copying unmodified content from the block to the new block till the file pointer is at the end of the block.

3. The method of claim 1, further comprising:

deleting the inode of the database index file; and

using the duplicate inode of the database index file for reading the database index file.

4. The method of claim 1, further comprising:

determining, prior to allocating the new block to write the update, whether a write request for writing the update is a first write request; and

if the write request for writing the update is the first write request, allocating the new block to write the update.

5. The method of claim 4, further comprising:

if the write request for writing the update is not the first write request, determining whether the new block is needed for writing the update; and

if the new block is needed for writing the update, allocating the new block for writing the update.

6. The method of claim 1, further comprising:

determining if an additional new block is needed to write the update;

if the additional new block is needed to write the update, allocating the additional new block to write the update, wherein the additional new block is linked with the new block; and

writing the update to the additional new block.

7. A system for managing a database index file, comprising:

an index manager module to generate, upon receiving an update for a database index file, a duplicate inode of an inode of the database index file, wherein the duplicate inode also points to blocks pointed to by the inode of the database index file; and

a file system to:

allocate one or more new blocks to write the update, wherein first block in the one or more new blocks is linked to last unmodified block of the blocks pointed to by the inode and the duplicate inode;

write the update to the one or more new blocks;

determine position of file pointer in the blocks pointed to by the inode and the duplicate inode;

if the file pointer is not at an end of a block in the blocks pointed to by the inode and the duplicate inode, copying unmodified content from the block to the one or more new blocks till the file pointer is at the end of the block;

determine position of file pointer in the one or more new blocks; and

if the file pointer is at end of the one or more new blocks, include blocks that are further to the block in the blocks pointed to by the inode and the duplicate inode, in the duplicate inode of the database index file.

8. The system of claim 7, wherein the index manager module uses an interface provided by the file system to generate the duplicate inode of the inode of the database index file.

9. The system of claim 7, wherein the file system to use the duplicate inode of the database index file for reading the database index file.

10. The system of claim 7, further comprising a database that includes data for generating the database index file.

11. A non-transitory machine-readable storage medium comprising instructions for managing a database index file, the instructions executable by a processor to:

create, upon receiving an update for a database index file, a duplicate inode of an inode of the database index file, wherein the duplicate inode also points to data blocks pointed to by the inode of the database index file;

allocate a new data block to write the update, wherein the new data block is linked to last unmodified data block of the data blocks pointed to by the inode and the duplicate inode;

write the update to the new data block;

determine position of file pointer in the data blocks pointed to by the inode and the duplicate inode;

if the file pointer is not at an end of a current data block in the data blocks pointed to by the inode and the duplicate inode, copy unmodified content from the current data block to the new data block till the file pointer is at the end of the data block;

determine position of file pointer in the new data block; and

if the file pointer is at end of the new data block, include data blocks that are further to the current data block in the data blocks pointed to by the inode and the duplicate inode, in the duplicate inode of the database index file.

12. The storage medium of claim 11, further comprising instructions to:

determine whether an application is using the inode of the database index file; and

if no application is using the inode of the database index file, delete the inode of the database index file.

13. The storage medium of claim 12, further comprising instructions to use the duplicate inode of the database index file for reading the database index file.

14. The storage medium of claim 11, further comprising instructions to:

if an application is reading the inode of the database index file, mark the inode of the database index file for deletion; and

delete the inode of the database index file after the application completes the reading.

15. The storage medium of claim 11, wherein the inode of the database index file is deleted by the application.

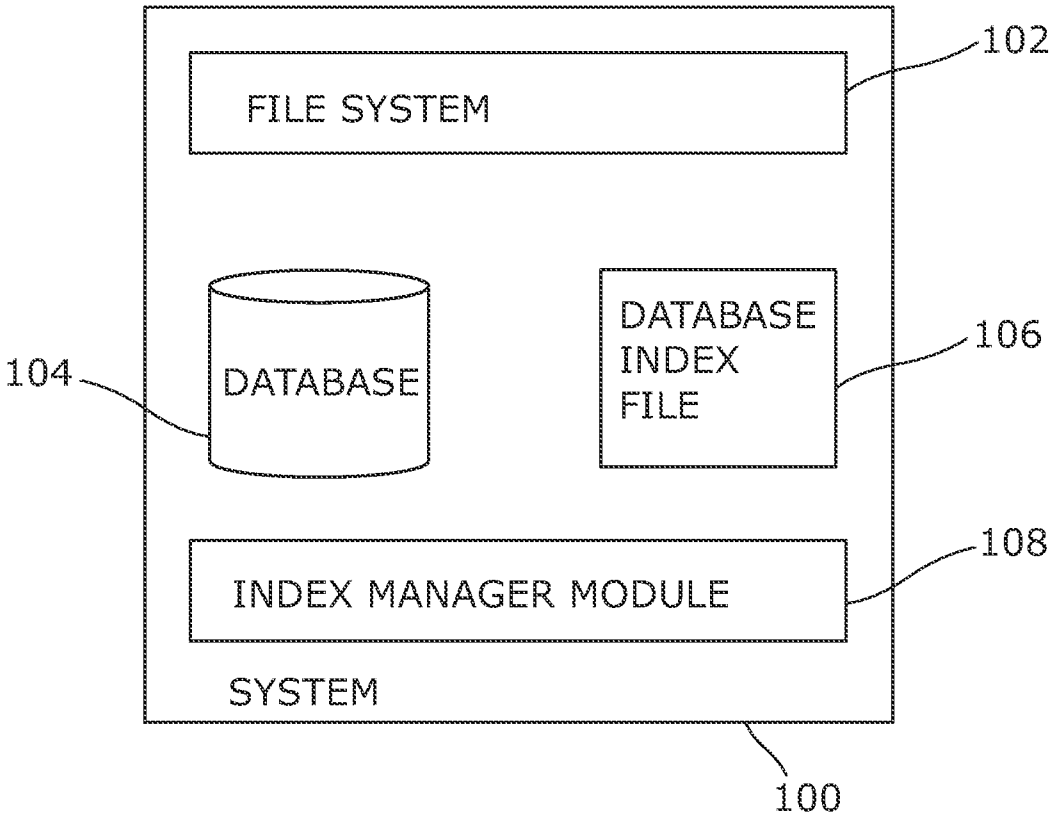
FILE SYSTEM
102
DATABASE
104
DATABASE INDEX FILE
106
INDEX MANAGER MODULE
108
SYSTEM
100

Fig. 1

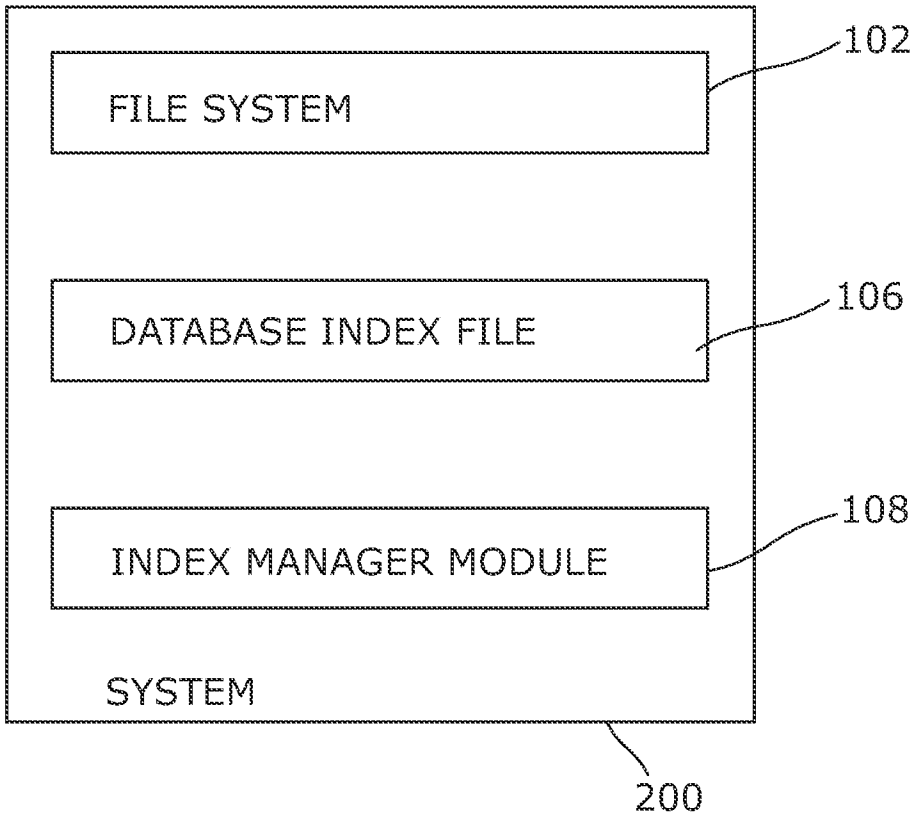
FILE SYSTEM
102
DATABASE INDEX FILE
106
INDEX MANAGER MODULE
108
SYSTEM
200

Fig. 2

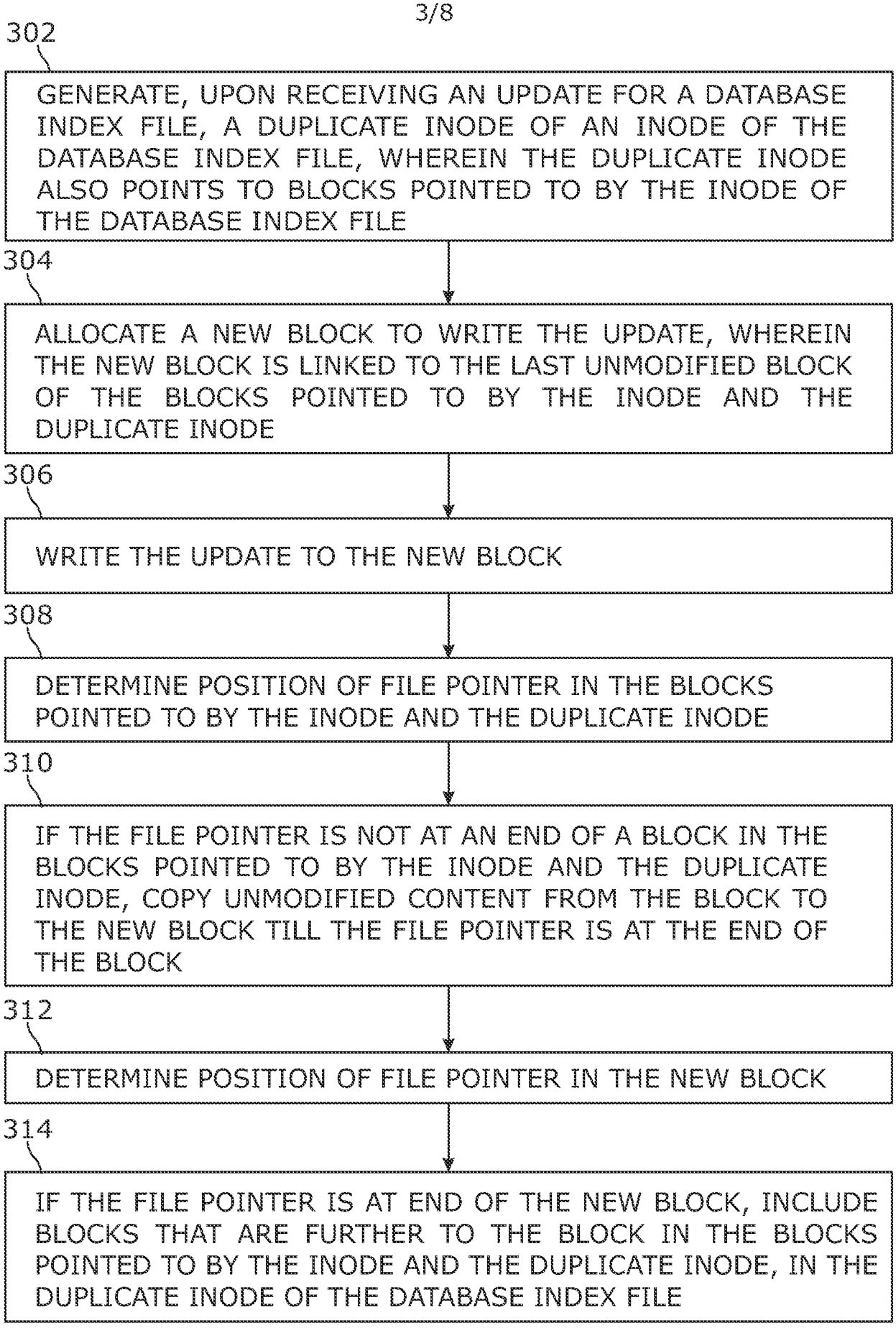
302
GENERATE, UPON RECEIVING AN UPDATE FOR A DATABASE INDEX FILE, A DUPLICATE INODE OF AN INODE OF THE DATABASE INDEX FILE, WHEREIN THE DUPLICATE INODE ALSO POINTS TO BLOCKS POINTED TO BY THE INODE OF THE DATABASE INDEX FILE
304
ALLOCATE A NEW BLOCK TO WRITE THE UPDATE, WHEREIN THE NEW BLOCK IS LINKED TO THE LAST UNMODIFIED BLOCK OF THE BLOCKS POINTED TO BY THE INODE AND THE DUPLICATE INODE
306
WRITE THE UPDATE TO THE NEW BLOCK
308
DETERMINE POSITION OF FILE POINTER IN THE BLOCKS POINTED TO BY THE INODE AND THE DUPLICATE INODE
310
IF THE FILE POINTER IS NOT AT AN END OF A BLOCK IN THE BLOCKS POINTED TO BY THE INODE AND THE DUPLICATE INODE, COPY UNMODIFIED CONTENT FROM THE BLOCK TO THE NEW BLOCK TILL THE FILE POINTER IS AT THE END OF THE BLOCK
312
DETERMINE POSITION OF FILE POINTER IN THE NEW BLOCK
314
IF THE FILE POINTER IS AT END OF THE NEW BLOCK, INCLUDE BLOCKS THAT ARE FURTHER TO THE BLOCK IN THE BLOCKS POINTED TO BY THE INODE AND THE DUPLICATE INODE, IN THE DUPLICATE INODE OF THE DATABASE INDEX FILE

Fig. 3

300

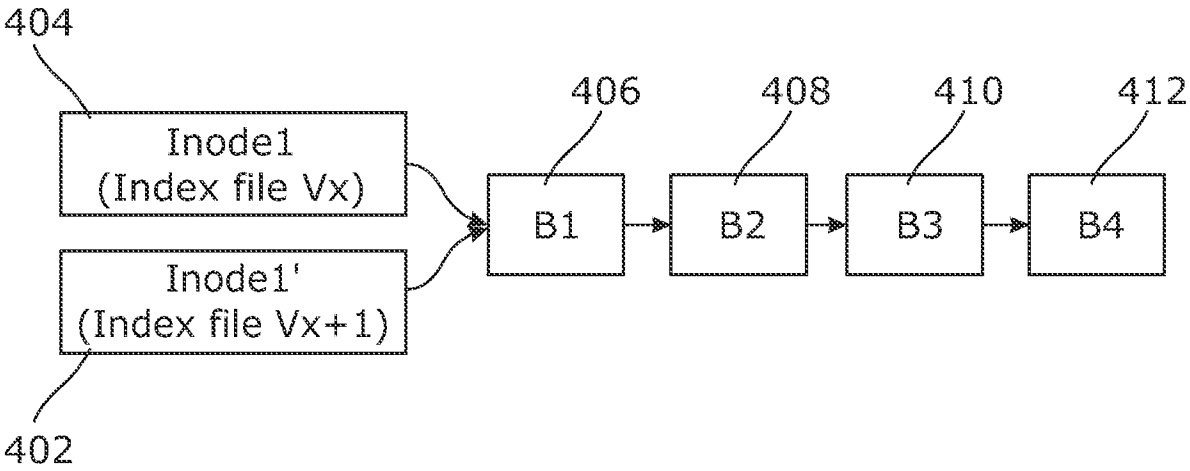
404
406
408
410
412
Inode1
(Index file Vx)
Inode1'
(Index file Vx+1)
B1
B2
B3
B4
402

Fig. 4A

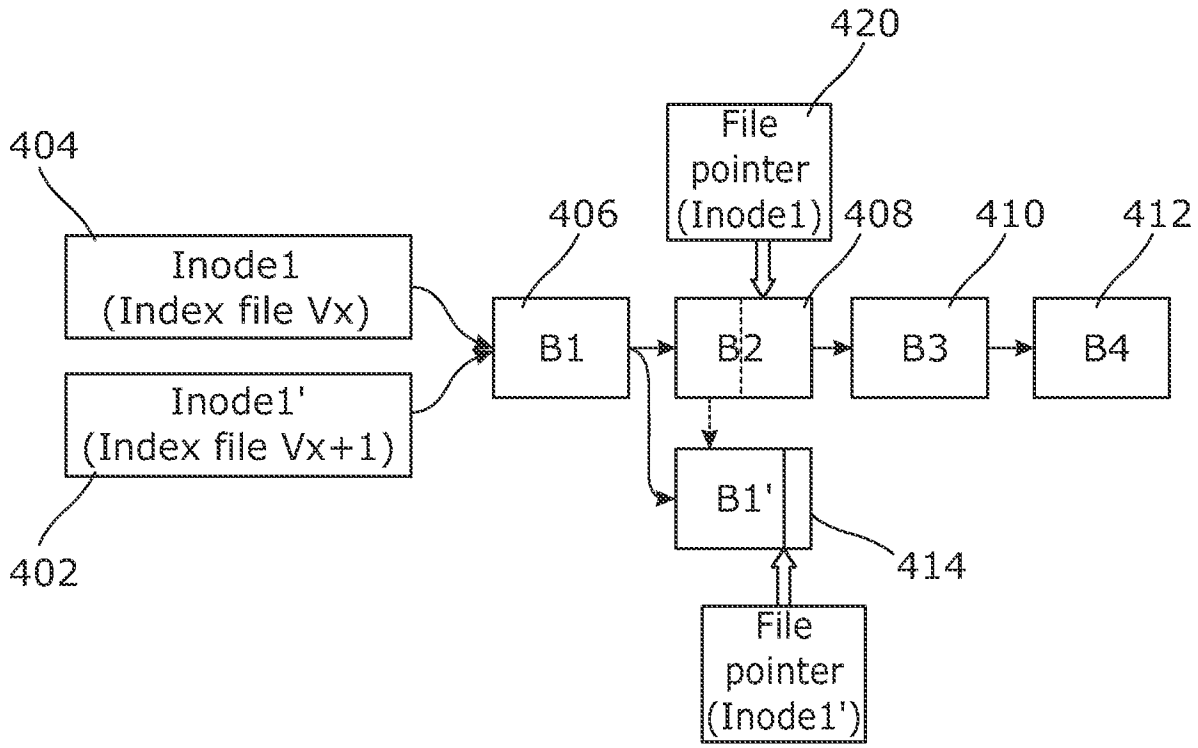
420
File
pointer
(Inode1)
404
406
408
410
412
Inode1
(Index file Vx)
Inode1'
(Index file Vx+1)
B1
B2
B3
B4
B1'
414
File
pointer
(Inode1')
402

Fig. 4B

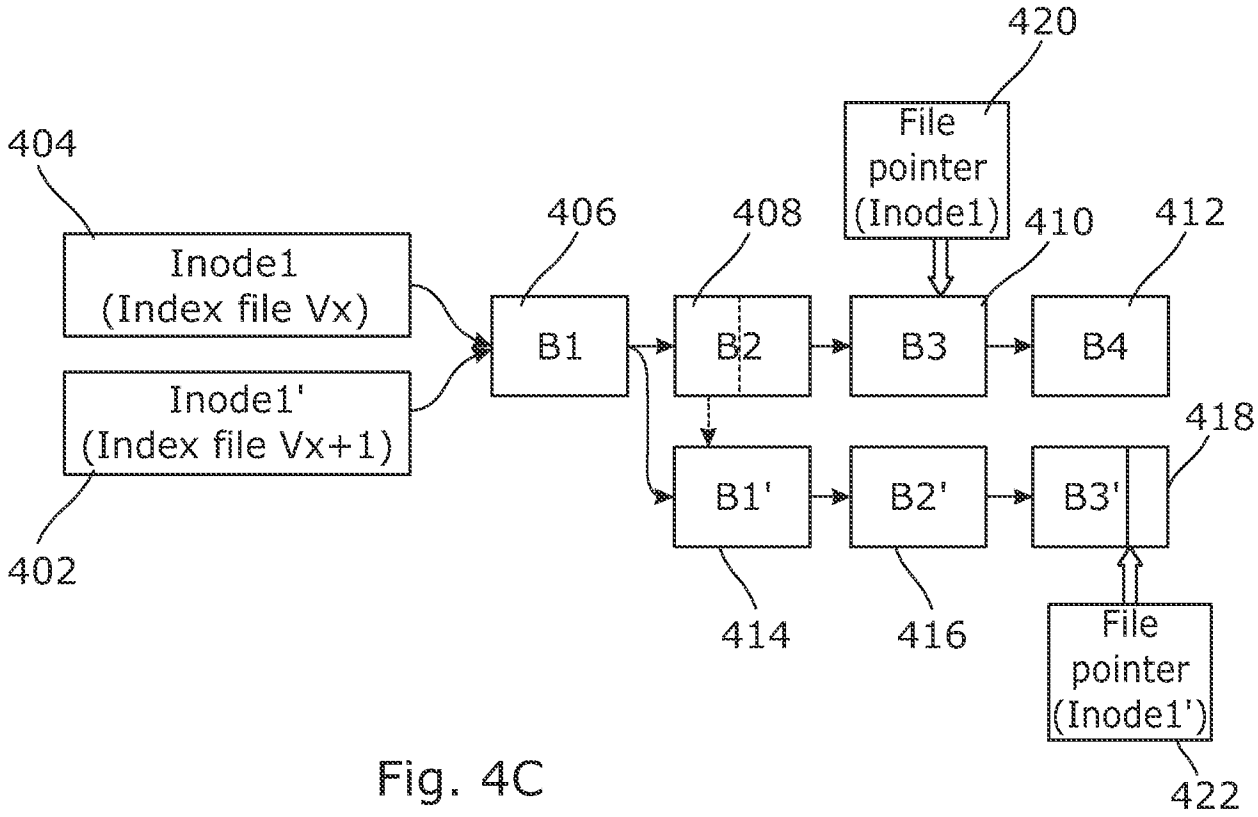
420
File pointer (Inode1)
404
406
408
410
412
Inode1 (Index file Vx)
Inode1' (Index file Vx+1)
B1
B2
B3
B4
418
B1'
B2'
B3'
402
414
416
File pointer (Inode1')
422

Fig. 4C

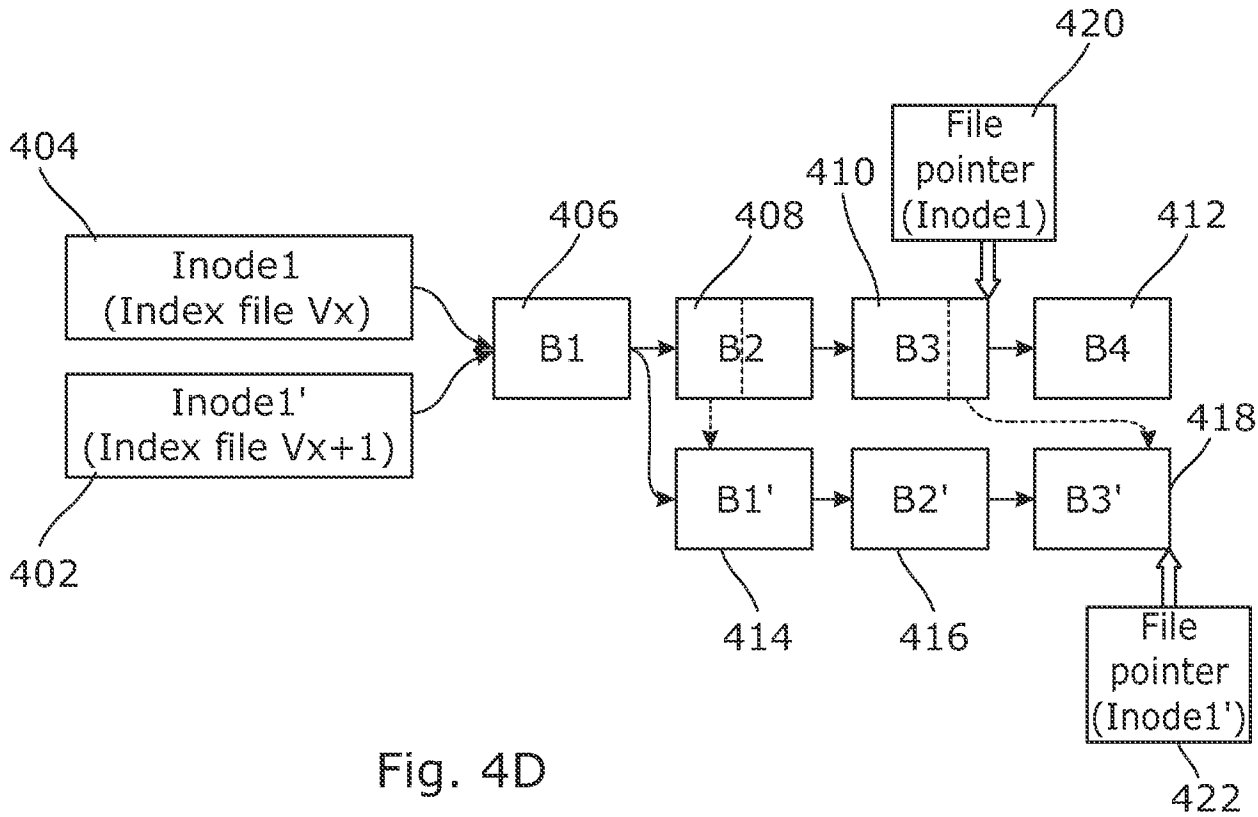
420
File pointer (Inode1)
404
410
406
408
412
Inode1 (Index file Vx)
Inode1' (Index file Vx+1)
B1
B2
B3
B4
418
B1'
B2'
B3'
402
414
416
File pointer (Inode1')
422

Fig. 4D

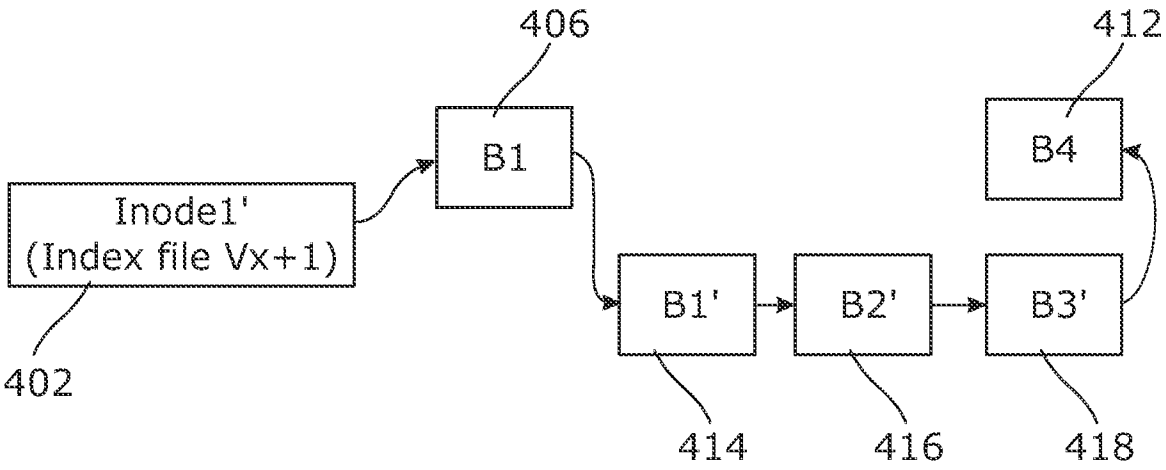
406
412
B1
B4
Inode1'
(Index file Vx+1)
B1'
B2'
B3'
402
414
416
418

Fig. 4E

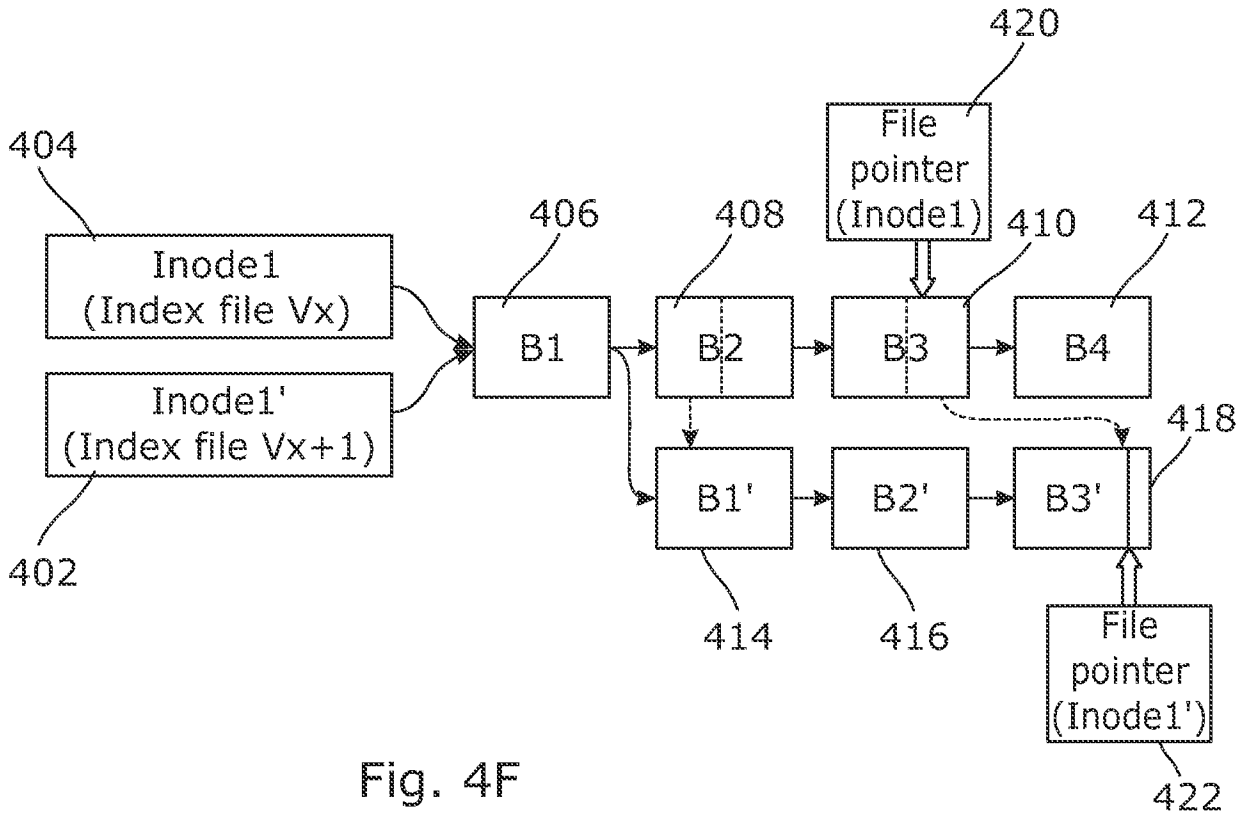
420
File
pointer
(Inode1)
404
406
408
410
412
Inode1
(Index file Vx)
B1
B2
B3
B4
Inode1'
(Index file Vx+1)
418
B1'
B2'
B3'
402
414
416
File
pointer
(Inode1')
422

Fig. 4F

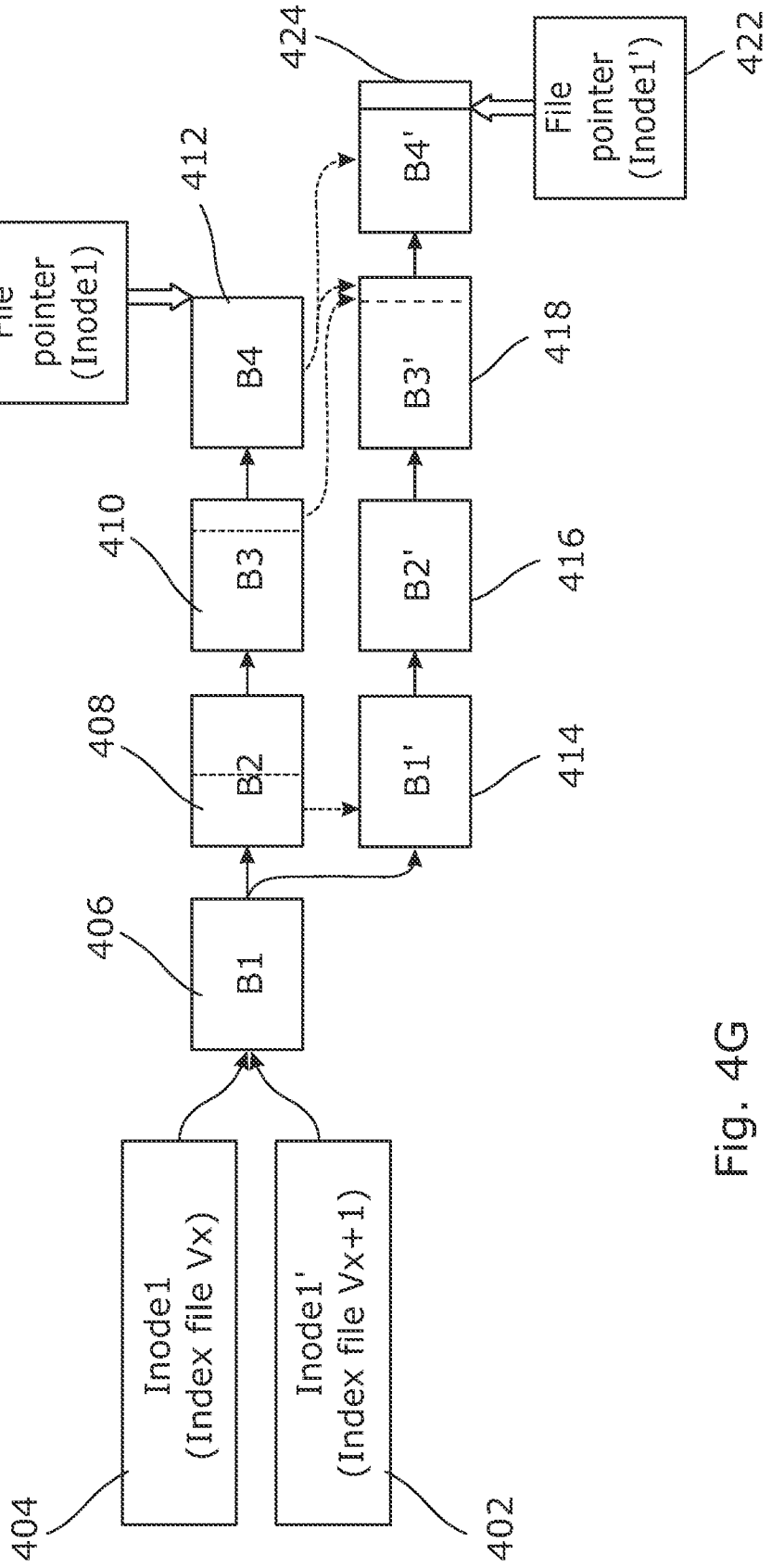
Inode1
(Index file Vx)
Inode1'
(Index file Vx+1)
B1
B2
B3
B4
B1'
B2'
B3'
B4'
File
pointer
(Inode1)
File
pointer
(Inode1')
402
404
406
408
410
412
414
416
418
420
422
424

Fig. 4G

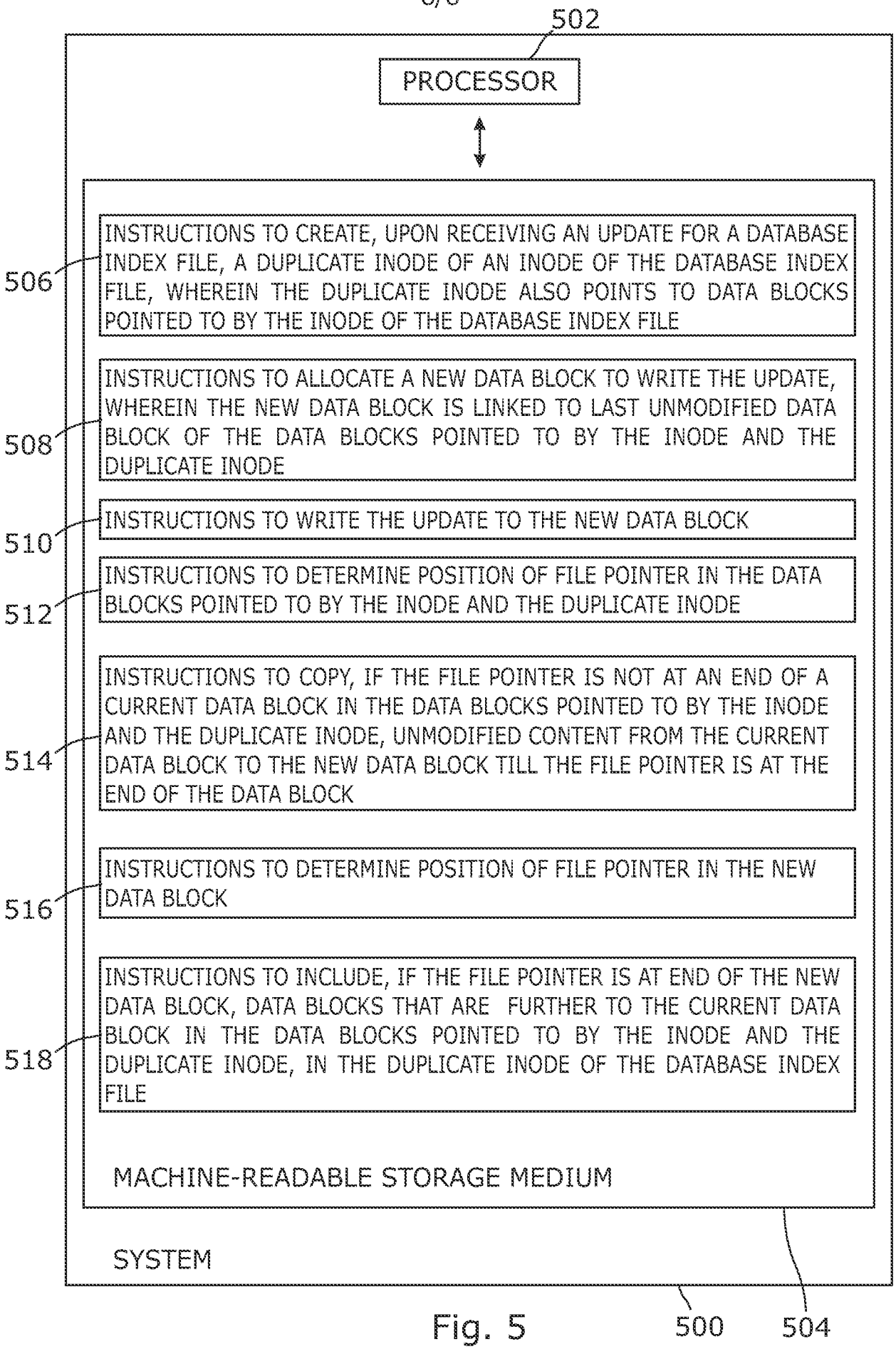
502
PROCESSOR
INSTRUCTIONS TO CREATE, UPON RECEIVING AN UPDATE FOR A DATABASE INDEX FILE, A DUPLICATE INODE OF AN INODE OF THE DATABASE INDEX FILE, WHEREIN THE DUPLICATE INODE ALSO POINTS TO DATA BLOCKS POINTED TO BY THE INODE OF THE DATABASE INDEX FILE
506
INSTRUCTIONS TO ALLOCATE A NEW DATA BLOCK TO WRITE THE UPDATE, WHEREIN THE NEW DATA BLOCK IS LINKED TO LAST UNMODIFIED DATA BLOCK OF THE DATA BLOCKS POINTED TO BY THE INODE AND THE DUPLICATE INODE
508
INSTRUCTIONS TO WRITE THE UPDATE TO THE NEW DATA BLOCK
510
INSTRUCTIONS TO DETERMINE POSITION OF FILE POINTER IN THE DATA BLOCKS POINTED TO BY THE INODE AND THE DUPLICATE INODE
512
INSTRUCTIONS TO COPY, IF THE FILE POINTER IS NOT AT AN END OF A CURRENT DATA BLOCK IN THE DATA BLOCKS POINTED TO BY THE INODE AND THE DUPLICATE INODE, UNMODIFIED CONTENT FROM THE CURRENT DATA BLOCK TO THE NEW DATA BLOCK TILL THE FILE POINTER IS AT THE END OF THE DATA BLOCK
514
INSTRUCTIONS TO DETERMINE POSITION OF FILE POINTER IN THE NEW DATA BLOCK
516
INSTRUCTIONS TO INCLUDE, IF THE FILE POINTER IS AT END OF THE NEW DATA BLOCK, DATA BLOCKS THAT ARE FURTHER TO THE CURRENT DATA BLOCK IN THE DATA BLOCKS POINTED TO BY THE INODE AND THE DUPLICATE INODE, IN THE DUPLICATE INODE OF THE DATABASE INDEX FILE
518
MACHINE-READABLE STORAGE MEDIUM
SYSTEM
500
504

Fig. 5

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2015/047351

A. CLASSIFICATION OF SUBJECT MATTER

G06F 17/30(2006.01)i

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F 17/30; G06F 3/048; G06F 3/06; G06F 7/00; G06F 12/00

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
Korean utility models and applications for utility models
Japanese utility models and applications for utility models

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
eKOMPASS(KIPO internal) & Keywords: database index file, update, inode, duplicate, file pointer, data block, and similar terms.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2005-0066095 A1 (SACHIN MULLICK et al.) 24 March 2005 See paragraphs [0100]-[0141] and [0153]-[0180]; and figures 13-18 and 21-25.	1-15
A	US 2008-0294696 A1 (YUVAL FRANDZEL) 27 November 2008 See paragraphs [0007], [0010], [0013]-[0014], [0029], [0035], and [0049]; and claims 1 and 21-22.	1-15
A	US 2010-0057755 A1 (JAMES P. SCHNEIDER) 04 March 2010 See paragraphs [0005], [0008]-[0009], [0045], [0052], and [0062].	1-15
A	US 2014-0019704 A1 (NETAPP, INC.) 16 January 2014 See paragraphs [0014], [0017], [0056], [0063], and [0088].	1-15
A	US 2012-0239658 A1 (OPHIR FRIEDER et al.) 20 September 2012 See paragraphs [0007], [0080]-[0081], and [0108].	1-15
A	US 2012-0246129 A1 (JEFFREY ROTHSCHILD et al.) 27 September 2012 See paragraphs [0006], [0049], [0063], and [0071].	1-15

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

☒ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of 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
29 March 2016 (29.03.2016)	**29 March 2016 (29.03.2016)**

Name and mailing address of the ISA/KR
International Application Division
Korean Intellectual Property Office
189 Cheongsa-ro, Seo-gu, Daejeon, 35208, Republic of Korea
Facsimile No. +82-42-481-8578

Authorized officer
NHO, Ji Myong
Telephone No. +82-42-481-8528

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.
PCT/US2015/047351

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2005-0066095 A1	24/03/2005	US 7865485 B2	04/01/2011
US 2008-0294696 A1	27/11/2008	EP 2168063 A1	31/03/2010
		US 8315984 B2	20/11/2012
		WO 2008-147516 A1	04/12/2008
US 2010-0057755 A1	04/03/2010	None	
US 2014-0019704 A1	16/01/2014	EP 1745395 A2	24/01/2007
		EP 1745395 B1	18/12/2013
		US 2005-246382 A1	03/11/2005
		US 2008-155220 A1	26/06/2008
		US 2011-225364 A1	15/09/2011
		US 2015-046504 A1	12/02/2015
		US 7430571 B2	30/09/2008
		US 7970770 B2	28/06/2011
		US 8533201 B2	10/09/2013
		US 8903830 B2	02/12/2014
		WO 2005-111803 A2	24/11/2005
		WO 2005-111803 A3	01/06/2006
US 2012-0239658 A1	20/09/2012	US 2009-228462 A1	10/09/2009
		US 8209358 B2	26/06/2012
		US 8626792 B2	07/01/2014
US 2012-0246129 A1	27/09/2012	US 8219562 B1	10/07/2012
		US 8650164 B2	11/02/2014