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

(19) **World Intellectual Property Organization**
International Bureau

(43) **International Publication Date**
20 June 2024 (20.06.2024)

(10) **International Publication Number**
WO 2024/129644 A1

(51) **International Patent Classification:**
G06F 12/02 (2006.01)

(21) **International Application Number:** PCT/US2023/083495

(22) **International Filing Date:** 12 December 2023 (12.12.2023)

(25) **Filing Language:** English

(26) **Publication Language:** English

(30) **Priority Data:**
18/064,373 12 December 2022 (12.12.2022) US

(71) **Applicant: GOOGLE LLC** [US/US]; 1600 Amphitheatre Parkway, Mountain view, California 94043 (US).

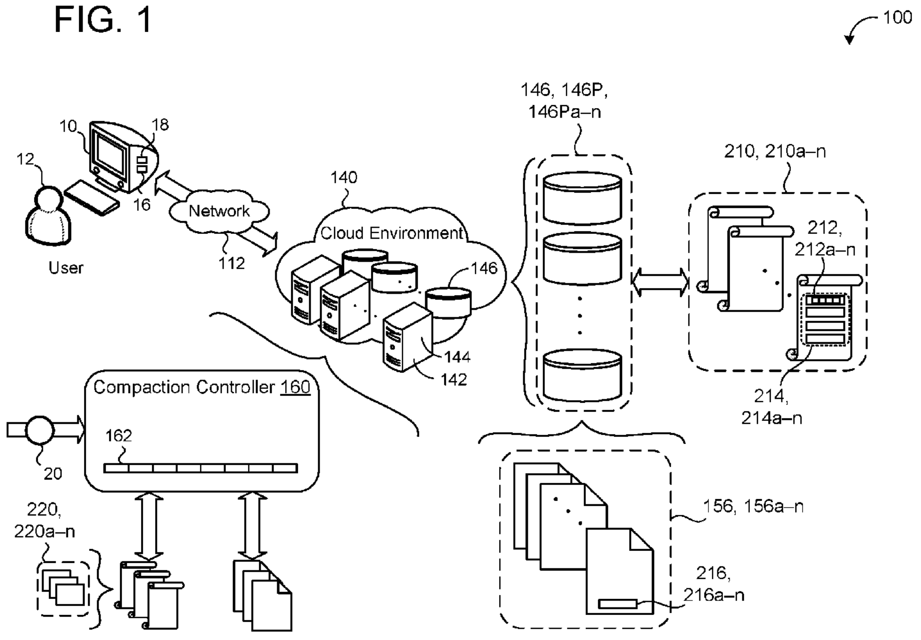
(72) **Inventors: DEISS, Olivier**; 1600 Amphitheatre Parkway, Mountain View, California 94043 (US). **LYAKHOVITSKIY, Grigory Borisovich**; 1600 Amphitheatre Parkway, Mountain View, California 94043 (US). **KIMMEL, Dan**; 1600 Amphitheatre Parkway, Mountain View, California 94043 (US).

(74) **Agent: KRUEGER, Brett A.**; 200 Ottawa Ave. NW, Suite 700, Grand Rapids, Michigan 49503 (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, CV, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IQ, IR, IS, IT, JM, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, MG, MK, MN, MU, 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, ST, SV, SY, TH,

(54) **Title:** GARBAGE COLLECTION IN LOG-BASED BLOCK DEVICES WITH SNAPSHOTS

FIG. 1



(57) **Abstract:** A method (400) for garbage-collection includes obtaining a request (20) to compact a plurality of log files (210) of a log-structured volume (146P). Each log file includes fresh block runs (214) in use and stale block runs no longer in use. The log-structured volume includes a plurality of snapshots (156). For each respective snapshot, the method includes determining, using a plurality of interval maps (162), the fresh block runs of the plurality of log files used by the respective snapshot. For each respective log file, the method includes writing the fresh block runs of the respective log file to a respective compacted log file and generating a respective per-log diff file (220). The method includes, for each respective snapshot, generating a respective checkpoint (216) based on respective per-log diff files (220) and deleting each respective log file of the plurality of log files.

WO 2024/129644 A1

[Continued on next page]

TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, CV, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SC, 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, ME, 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))*
— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments (Rule 48.2(h))*

Garbage Collection in Log-based Block Devices with Snapshots

TECHNICAL FIELD

[0001] This disclosure relates to garbage collection in log-based block devices with snapshots.

BACKGROUND

[0002] Log-based storage devices store data and metadata in one or more logs. That is, data is written to disk sequentially rather than scanning blocks across the disk. Because data is written sequentially, log-based storage devices may transform groups of small, synchronous writes into a few large, asynchronous sequential write operations. This can greatly increase write bandwidth by decreasing seek overhead. However, log-structured volumes and file systems must include careful management of free space. As the log begins to fill, garbage collection must be undertaken to free space in the log by deleting data no longer in use.

SUMMARY

[0003] One aspect of the disclosure provides a computer-implemented method for garbage collection of log-based block devices that, when executed by data processing hardware, causes the data processing hardware to perform operations. The operations include obtaining a request to compact a plurality of log files of a log-structured volume. The log-structured volume includes a plurality of snapshots. Each snapshot of the plurality of snapshots is representative of a state of the log-structured volume at a different point in time. For each respective snapshot of the plurality of snapshots of the log-structured volume, the operations include determining fresh block runs of the plurality of log files used by the respective snapshot. For each respective log file of the plurality of log files, the operations include writing the fresh block runs of the respective log file to a respective compacted log file and generating a respective per-log diff file mapping a location of the written fresh block runs at the respective log file to a location of the written fresh block runs at the respective compacted log file. For each respective snapshot of the plurality of snapshots of the log-structured volume, the operations include generating a respective checkpoint based on respective per-log diff files and deleting each respective log file of the plurality of log files.

[0004] Implementations of the disclosure may include one or more of the following optional features. In some implementations, determining the fresh block runs includes using a plurality of interval maps where each respective interval map of the plurality of interval maps specifies the fresh block runs of a respective log file for each snapshot of the plurality of snapshots and the operations further include generating, using the plurality of log files and the plurality of snapshots, the plurality of interval maps. In some of these implementations, generating the plurality of interval maps includes scanning, for each snapshot of the plurality of snapshots, each log file of the plurality of log files.

[0005] Optionally, each fresh block run and each stale block run includes a tuple comprising a log file name, an offset, and a byte count. The operations further include, in some examples, selecting the plurality of log files from a set of log files based on a ratio of a sum of a size of the fresh block runs of each respective log file over a total size of each respective log file. In some of these examples, the operations further include, for each respective log file of the set of log files, determining the ratio for each respective log file using a respective interval map of the plurality of interval maps.

[0006] In some implementations, each log file of the plurality of log files is frozen. Obtaining the request to compact the plurality of log files may include determining a threshold amount of time has passed since a previous compaction. Each respective per-log diff file optionally includes an ordered list of tuples, each tuple representing a respective fresh block run and including an offset, a log file name, and a byte count. In some examples, the operations further include, for each deleted respective log file, deleting the respective per-log diff file. In some examples, each log file of the plurality of log files includes the fresh block runs representing contiguous blocks of data in use by the log-structured volume and stale block runs representing contiguous blocks of data no longer in use by the log-structured volume.

[0007] Another aspect of the disclosure provides a system for garbage collection of log-based block devices. The system includes data processing hardware and memory hardware in communication with the data processing hardware. The memory hardware stores instructions that when executed on the data processing hardware cause the data processing hardware to perform operations. The operations include obtaining a request to compact a plurality of log files of a log-structured volume. The log-structured volume includes a plurality of snapshots. Each snapshot of the plurality of snapshots is representative of a state of the log-structured volume at a different point in time. For each respective snapshot of the plurality of snapshots of the log-structured volume, the operations include determining fresh block runs of the plurality of log files used by the respective snapshot. For each respective log file of the plurality of log files, the operations include writing the fresh block runs of the respective log file to a respective compacted log file and generating a respective per-log diff file mapping a location of the written fresh block runs at the respective log file to a location of the written fresh block runs at the respective compacted log file. For each respective snapshot of the plurality of snapshots of the log-structured volume, the operations include generating a respective checkpoint based on respective per-log diff files and deleting each respective log file of the plurality of log files.

[0008] This aspect may include one or more of the following optional features. In some implementations, the operations further include generating, using the plurality of log files and the plurality of snapshots, the plurality of interval maps. In some of these implementations, generating the plurality of interval maps includes scanning, for each snapshot of the plurality of snapshots, each log file of the plurality of log files.

[0009] Optionally, each fresh block run and each stale block run includes a tuple comprising a log file name, an offset, and a byte count. The operations further include, in some examples, selecting the plurality of log files from a set of log files based on a ratio of a sum of a size of the fresh block runs of each respective log file over a total size of each respective log file. In some of these examples, the operations further include, for each respective log file of the set of log files, determining the ratio for each respective log file using a respective interval map of the plurality of interval maps.

[0010] In some implementations, each log file of the plurality of log files is frozen. Obtaining the request to compact the plurality of log files may include determining a threshold amount of time has passed since a previous compaction. Each respective per-log diff file optionally includes an ordered list of tuples, each tuple representing a respective fresh block run and including an offset, a log file name, and a byte count. In some examples, the operations further include, for each deleted respective log file, deleting the respective per-log diff file. In some examples, each log file of the plurality of log files includes the fresh block runs representing contiguous blocks of data in use by the log-structured volume and stale block runs representing contiguous blocks of data no longer in use by the log-structured volume.

[0011] The details of one or more implementations of the disclosure are set forth in the accompanying drawings and the description below. Other aspects, features, and advantages will be apparent from the description and drawings, and from the claims.

DESCRIPTION OF DRAWINGS

[0012] FIG. 1 is a schematic view of an example system for performing garbage collection on log-based block devices with snapshots.

[0013] FIGS. 2A–2D are schematic views of exemplary log files during a compaction event.

[0014] FIG. 3 is a schematic view of an exemplary snapshot statistics table.

[0015] FIG. 4 a flowchart of an example arrangement of operations for a method of performing garbage collection on log-based block devices with snapshots.

[0016] FIG. 5 is a schematic view of an example computing device that may be used to implement the systems and methods described herein.

[0017] Like reference symbols in the various drawings indicate like elements.

DETAILED DESCRIPTION

[0018] Log-based storage devices and file systems store data and metadata in one or more logs. That is, data is written to disk sequentially rather than scattering blocks across the disk like most conventional file systems. Because data is written sequentially, log-based storage devices may transform groups of small, synchronous writes into a few large, asynchronous sequential write operations. This can greatly increase write bandwidth by decreasing seek overhead. However, log-structured volumes and file systems must include careful management of free space. As the log begins to fill, garbage collection must be undertaken to free space in the log by deleting data no longer in use.

[0019] Garbage collection of log-based block devices may be performed by background workers. These workers first load a checkpoint representing a point-in-time view of the device, and proceed to read the entirety of the device (e.g., all of the log files representing the device). These log files are filled with both fresh data that is still in use and stale data that is no longer in use. Next, the worker writes all of the fresh data out to new log files. After this process, the new log files only contain in-use data and the original input log files can be deleted.

[0020] The block device may support "snapshots." A snapshot is a state of the block device at a particular point in time. Snapshots serve many important functions, such as data protection and availability. However, these snapshots increase the complexity of garbage collection. Without performing garbage collection on the log files associated with the snapshots, the amount of space consumed on the device will rapidly increase. However, performing garbage collection on the snapshots is challenging due to the extra storage and processing costs for the worker, especially when the worker is designed with limited memory as a background process.

[0021] A naive implementation for garbage collection for such a system would include opening and reading the log files for each snapshot, writing out a new log file, and generating a new checkpoint for the snapshot. Each checkpoint defines, for each snapshot, the location in the log file of respective block runs (i.e., contiguous ranges of data blocks). However, building checkpoints in this manner requires a significant amount of memory as the worker must track the location of many data blocks. Because it is likely many snapshots "share" the same data (i.e., there may be little changes to the data between snapshots), using such large amounts of memory to generate very similar checkpoints is an inefficient use of resources.

[0022] Implementations herein are directed toward a compaction controller of a computing system (e.g., a distributed system) that enables efficient garbage collection or compaction of log-structured volumes (e.g., persistent disks) that employ snapshots. The compaction controller performs a scan to determine, using an interval map, which blocks of the log files are currently fresh or in use. The compaction controller writes all fresh blocks to new log files, and each time a block is moved from a source log file to a new log file, a record is added to a per-log diff file that indicates where the new data is now stored. The compaction controller merges the compacted log files into the system by writing a new checkpoint for each snapshot using the per-log diff files. The compaction controller, instead of tracking all of the new locations of the new data blocks in each of the different checkpoints, generates the per-log diff files per log file. Thus, each log file that the compaction controller compacts has an associated per-log diff file and when building the checkpoints, the compaction controller can go to the same per-log diff file for each snapshot that uses the respective log file, thus leveraging the limited differences in data between snapshots to greatly decrease the amount of resources required to generate the checkpoints.

[0023] Referring to FIG. 1, in some implementations, an example data compaction system 100 includes a remote system 140 in communication with one or more user devices 10 via a network 112. The remote system 140 may be a single computer, multiple computers, or a distributed system (e.g., a cloud environment) having scalable / elastic resources 142 including computing resources 144 (e.g., data processing hardware) and/or storage resources 146 (e.g., memory hardware). A data store (i.e., a remote storage device) may be overlain on the storage resources 146 to allow scalable use of the storage resources 146 by one or more of the clients (e.g., the user device 10) or the computing resources 144.

[0024] The storage resources 146 include one or more persistent disks 146, 146P, 146Pa–n. Each persistent disk 146 represents a durable network storage device that instances (e.g., virtual machines (VMs)) may access like a physical disk. Each persistent disk 146P may be distributed across several physical disks (e.g., solid-state drives (SSDs), hard disks, etc.). Each persistent disk 146P is a block-based log-structured volume. That is, each persistent disk 146P stores data via one or more log files 210, 210a–n. The persistent disks 146P may be divided into one or more segments of any size (e.g., 32 gigabytes per segment).

[0025] As the remote system 140 receives block runs 214, 214a–n to write to the persistent disks 146P (e.g., from user 12, the remote system 140 itself, a VM, or any other remote entity), the remote system 140 writes the block runs 214 to respective log files

210. Each block run 214 represents or is used to describe a location of a contiguous range of data blocks 212, 212a–n (e.g., 4k blocks) of the log files 210 and/or persistent disk 146P. In some examples, each block run 214 is a tuple that includes a log file name (i.e., a reference to which log file 210 the data blocks 212 are at), an offset (i.e., a location with the respective log file 210), and a byte count (i.e., a length or size of the range of data blocks 212). In some examples, depending on the file system in use on the persistent disk 146P, the block runs 214 may represent some or all of a data file. Periodically, or whenever certain conditions are satisfied, the remote system 140 captures a snapshot 156, 156a–n of the persistent disk(s) 146P. Each snapshot 156 captures or represents a state of the persistent disk 146P at the time of the snapshot 156 (i.e., the state of the block runs 214 at the point in time the snapshot 156 is taken). As used herein, snapshots 156 may also be referred to as "generations." Using the snapshots, a user 12 or the remote system 140 may recall the state of the persistent disk 146P at the point in time the snapshot 156 was taken for various reasons (e.g., data recovery). The remote system 140 may capture any number of snapshots 156 at any intervals (e.g., after a threshold period of time has passed since the last snapshot 156 was captured, whenever a threshold amount of data has been written to the log files 210, etc.). The snapshots 156, which represent a state of the block runs 214 at some point in the past, are contrasted against the "live device," which refers to the state of the block runs 214 of the persistent disk 146P at the current point in time. That is, the user 12 may query the live device of the persistent disk 146P to determine a current state of the block runs 214 stored at the persistent disk 146P or query a snapshot 156 to determine a state of the block runs 214 stored at the persistent disk 146P for a particular point in time in the past.

[0026] To track the locations of the block runs 214 in the log files 210, each snapshot 156 (and the live device) may include one or more checkpoints 216, 216a–n (i.e., checkpoint regions) that include pointers to locations within the log files 210 relevant to the locations of the block runs 214. In some examples, each checkpoint 216 is a mapping of a logical block address to a location in a respective log file 210. Optionally, there is a separate log file 210 per segment (e.g., per 32 gigabyte segment). When the remote system 140 receives a read request to retrieve a particular block run 214 (e.g., from the live device or a particular snapshot 156), the remote system 140 may reference the appropriate checkpoint(s) 216 to determine the one or more log files 210 that include the requested block run 214 and the location (e.g., an offset) within with the log files 210 that include the block run 214 or points to the block run 214. Some or all of the checkpoints 216 may be cached to volatile memory of the remote system 140 to decrease access times of reads and/or writes. In some examples, the remote system 140 uses other options to reconstruct a device (e.g., tail scanning) when checkpoints 216 are unavailable.

[0027] As the log files 210 accumulate data from written block runs 214, a capacity of the associated persistent disk 146P decreases. Without garbage collection (i.e., compaction), each persistent disk 146P will become unavailable for new data once the log files 210 of the persistent disk 146P reach the end of the device's capacity (e.g., a capacity allotted to the persistent disk 146P by the remote system and/or a capacity available to the remote system 140). To this end, the remote system 140 executes a compaction controller 160 that manages garbage collection and/or compaction of the block runs 214 stored within the log files 210.

[0028] The compaction controller 160 obtains a compaction request 20 to compact one or more of the log files 210 of the remote system 140. The request may come from a user 12, via another process or module of the remote system 140, or even generated by the compaction controller 160. The request may be in response to any number of compaction triggers being satisfied (e.g., a threshold amount of time having passed since a previous compaction, an amount of data written, free capacity of the persistent disks 146P, deletion of data or a snapshot 156, etc.). The compaction request 20 may request that all log files 210 be evaluated for compaction or, alternatively, only a subset of the log files 210 be evaluated for compaction. For example, the compaction request 20 may restrict the compaction to log files 210 associated with a particular user 12, a particular process/module, particular persistent disks 146P located within a particular geographical region, etc.

[0029] For each respective snapshot 156 included by the compaction request 20, the compaction controller 160 determines, using one or more interval maps 162, fresh block runs 214, 214F of the log files 210 used by the snapshot 156. Each interval map 162 specifies the fresh block runs 214F of a respective log file 210 for each snapshot 156. More specifically, in some implementations, each interval map 162 is a mapping from an offset in a respective log file 210 to data (e.g., metadata) about fresh block runs 214F. The compaction controller 160 may maintain a separate interval map 162 for each log file 210. The compaction controller 160 may generate and/or update the interval maps 162. For example, the compaction controller 160 scans, during a compaction event, for each snapshot 156, each log file 210 to generate the interval maps 162.

[0030] The fresh block runs 214F refer to block runs 214 that are in use. That is, fresh block runs 214F refer to block runs 214 that have not been modified (i.e., overwritten). This is in contrast to stale block runs 214, 214S, which refer to block runs 214 that are no longer in use. For example, a user 12 writes a block run 214 "foo.txt" to a persistent disk 146P. The remote system 140 writes the block run 214 to an appropriate log file 210 for the persistent disk 146P. This block run 214, at this point in time, is a fresh block run 214F. At a later point in time, the user 12 modifies "foo.txt." The remote system writes the modified block run 214 to a log file 210. The new, modified block run 214 is a fresh block run 214F, however the original block run 214 is now a stale block run 214S. That is, "foo.txt" now exists twice within the log files 210, with the first one being stale (because the second one "overwrote" its data) and the second one being fresh.

[0031] After determining the fresh block runs 214F of the log files 210, the compaction controller 160, for each respective log file 210, writes the fresh block runs 214F of the respective log file 210 to one or more respective new or compacted log files 210. The fresh block runs 214F from one respective log file 210, in some implementations, are spread across multiple new or compacted log files 210. As explained in more detail below, the new log files 210 include only the fresh block runs 214F and purposefully exclude any stale block runs 214S. Thus, the new log files 210 include only fresh block runs 214F from the original log files 210. The compaction controller 160 also, for each respective log file 210, generates a respective per-log diff file 220, 220a–n. Each respective per-log diff file 220 maps a location of the written fresh block runs 214F at the original log file 210 to a location of the written fresh block runs 214F at the new compacted log file 210.

[0032] The compaction controller 160, for each respective snapshot 156 (and for the live device), generates a respective new checkpoint 216 for the snapshot 156 based on or using the per-log diff file 220. That is, instead of reading or scanning the entirety of the snapshot 156, which may take a substantial amount of memory and other computational resources, the compaction controller 160 leverages the per-log diff file 220 when updating or modifying the original checkpoints 216 or generating the new checkpoints 216 for the snapshots 156. In some implementations, the compaction controller 160 generates the checkpoints 216 sequentially for the snapshots 156 by starting with the oldest snapshot 156 and finishing with the live device. After the compaction controller 160 generates all of the checkpoints 216, the new checkpoints 216 are merged into the snapshots 156 and the live device (i.e., the live device (or controlling virtual machine, etc.) is aware of the new mappings), the compaction controller 160 may delete the original, non-compacted log files 210, thus freeing space for the persistent disk 146P. In some implementations, the compaction controller 160 automatically deletes any log files 210 that are not referenced by any checkpoint 216. For example, once the updated checkpoints 216 merge, the original log files 210 are no longer referenced, and the compaction controller 160 may safely delete the unreferenced log files 210. The checkpoints 216 may merge under any number of conditions. For example, a VM regularly generates new checkpoints 216 (e.g., due to data written to the live device) and merges the checkpoints 216 from the compaction controller 160 simultaneously. The checkpoints 216 may merge during “offline compactions” (e.g., when the controlling VM is offline). The checkpoints 216 may merge immediately upon generation or after a period of time if certain conditions are satisfied.

[0033] Referring now to FIG. 2A, a schematic view 200a includes a simplified example of a persistent disk 146P represented by two log files 210a–b (i.e., a “log1.log” first log file 210a and a “log2.log” second log file 210b). The example also includes two checkpoints 216a–b that includes a “live device” first checkpoint 216a and a “snapshot 1” second checkpoint 216b. The first log file 210a includes four block runs 214 using six data blocks 212. The block runs 214 include three fresh block runs 214Fa–c and one stale block run 214Sa. The second log file 210b includes four block runs 214 that use five data blocks 212. These block runs 214 include three fresh block runs 214Fd–f and one stale block run 214Sb. Thus, the two log files 210a–b store a total of six fresh block runs 214Fa–f and two stale block runs 214Sa–b. The stale block runs 214S represent an opportunity for compaction or garbage collection.

[0034] Both checkpoints 216a–b include six regions 218, with each region 218 mapping to one of the six fresh block runs 214F. In this example, there were no changes to the block runs 214 of the log files 210a–b in the time between when the snapshot checkpoint 216b was created and the current live device checkpoint 216a (which represents the current state of the live device). Accordingly, each of the corresponding regions 218 between the two checkpoints 216a–b map to the same fresh block run 214F.

[0035] Referring now to FIG. 2B, a schematic view 200b continues the example of FIG. 2A. Here, two additional data blocks 212 have been written to the second log file 210b such that the first log file 210a still has six data blocks 212 of data while the second log file 210b now has seven data blocks 212 of data. Due to the addition of the new data, the live device checkpoint 216a updates accordingly to include the mapping to the new data of the log files 210. Due to the update, the mappings between the first checkpoint 216a (i.e., for the live device, which updated) and the second checkpoint 216b (i.e., for the snapshot 156, which did not update) are no longer the same. Specifically, when an area 250A of the live device checkpoint 216a mutated (i.e., updated), the snapshot checkpoint 216b gained an area 250B that maps to two locations in the log files 210a–b that the live device checkpoint 216a does not map to. While in a conventional log-structured volume without snapshots, these two locations (i.e., fresh block runs 214Ff, 214Fg) would be marked stale and be compacted, in this example, the snapshot 156 continues to use the fresh block runs 214Ff, 214Fg even after the live device updates, thus necessitating that the fresh block runs 214Ff, 214Fg stay fresh. As shown in schematic view 200C of FIG. 2C, after the live device checkpoint 216a mutates in response to the two new data blocks written to the second log file 210b, of the thirteen data blocks 212 of the log files 210a–b, two are stale and may be compacted, two are in use only by the snapshot 156, and the remaining nine are in use by the live device and/or the snapshot 156.

[0036] Referring now to FIG. 2D, a schematic view 200D continues the examples of FIGS. 2A–2C with an example of compacting the log files 210a–b. For clarity, each the individual files of the log files 210 are not labeled. Here, the original log files 210a–b include two stale block runs 214Sa–b. To compact these log files 210a–b, the compaction controller 160, in this example, generates two new log files 210c–d (i.e., "log3.log" third log file 210c and "log3.log" fourth log file 210d) that include all of the fresh block runs 214F of the original two log files 210a–b that the live device checkpoint 216a references. Notably, the compaction controller 160 does not write the stale block runs 214Sa–b to the new log files 210c–d. The compaction controller 160 must also maintain the two fresh block runs 214Ff–g that the live device checkpoint 216a does not reference but that the snapshot checkpoint 216b does reference. While the compaction controller 160 may write these fresh block runs 214Ff–g to the log files 210c–d, in this example, the compaction controller 160 generates a third new log file 210e (i.e., "log5.log" fifth log file 210e) that includes only the fresh block runs 214Ff–g not referenced by the live device checkpoint 216a. By segregating these fresh block runs 214Ff–g into a separate log file 210e, future compactions may be simpler, as when the snapshot 156 associated with the snapshot checkpoint 216b is deleted, no other checkpoints 216 will reference the associated log file 210e, and the log file 210e may be deleted without any other compaction required.

[0037] While generating the new log files 210c–e, the compaction controller 160 also generates a corresponding per-log diff file 220a–b for each of the original log files 210a–b. Each per-log diff file 220 tracks movement of blocks still referenced by the live device checkpoint 216 or any snapshot checkpoint 216 from the compacted log file 210 to the new destination (e.g., a newly generated log file 210). In some implementations, each per-diff log file 220 includes an ordered list of tuples (e.g., a tuple includes an offset element, a log file name element, and a byte count element). In some examples, each tuple includes {offset, block run} elements. Here, the offset element refers to the offset of the first data block 212 of the block run 214, and the block run 214 refers to a location of a contiguous range of data blocks 212. Each block run 214 may define the data blocks 212 used by the persistent disk 146P. The block run 214, in some examples, includes a tuple with <filename, offset, byte_count> elements.

[0038] In the example of FIG. 2D, two log files 210a–b are compacted, and accordingly, the compaction controller 160 generates two per-log diff files 220a–b. Here, the first per-log diff file 220a corresponds to the first log file 210a and the second per-log diff file 220b corresponds to the second log file 210b. The first per-log diff file 220a maps each of the block runs 214 (or block runs) of the first log file 210a to their destination locations in the new log files 210c–d. For example, the block run 214Fa located at data block ‘0’ in the first log file 210a is now located at data block ‘3’ of the first new log file 210c. Similarly, the second per-log diff file 220b maps the moved block runs 214 and/or block runs of the second log file 210b to the new logs files 210c–d. For example, the second per-log diff file 220b maps the fresh block run 214Fd located at data block ‘1’ of the second log file 210b to data block ‘0’ of the first new log file 210c.

[0039] Optionally, the compaction controller 160 periodically deletes the per-log diff files 220. For example, when the log file 210 the per-log diff file 220 is associated with is deleted (i.e., because the live device and snapshots 156 do not reference the log file 210), the per-diff log file 220 is also deleted.

[0040] The per-log diff files 220 summarize the shuffling of data that happens for each log file 210 during compaction. The per-log diff files 220 are relatively small (e.g., less than 10 mebibytes). Use of per-log diff files 220 instead of using one checkpoint 216 per compacted snapshot 156 is advantageous, as the per-log diff files 220 allow the compaction controller 160 to scale compaction to any number of snapshots 156 without requiring similarly scaling memory requirements. When using checkpoints 216 instead of per-log diff files 220, when one block 212 is referenced by all snapshots 156, the movement that occurs during the compaction of this block must be duplicated in all of the in-memory maps (i.e., one per snapshot 156), which greatly increases memory usage.

[0041] Referring now to FIG. 3, in some implementations, the compaction controller 160 determines when to run a compaction (e.g., generate a compaction request 20) and/or a type of compaction to execute (e.g., a standard compaction that only compacts the live device versus a heavy compaction that compacts the live device and at least some snapshots 156). In some implementations, the compaction controller 160 executes a heavy compaction (i.e., compacts the snapshots 156) whenever a snapshot 156 is deleted. However, in some scenarios (e.g., when snapshot 156 churn is high), this technique may generate a large number of compactions. Additionally or alternatively, the compaction controller 160 determines a frequency and/or a type of compaction based on an “IS-density” of the live device and/or one or more snapshots 156. As used herein, the IS-density refers to a ratio of bytes used by the live device or snapshot 156 over a total size of the log file 210. Optionally, the compaction controller 160, during a compaction event, selects the log files 210 eligible for compaction based on the IS-density of the respective log file 210 (i.e., based on a ratio of a sum of a size of the fresh block runs of each respective log file over a total size of each respective log file). In this way, only a subset of available log files 210 may be compacted during any particular compaction event. As described in more detail below, the compaction controller 160, in some implementations, determines the IS-density using a respective interval map 162. In some implementations, only "frozen" log files 210 associated with snapshots 156 are eligible for compaction. Frozen log files 210 refer to log files 210 that are restricted from having new data written. That is, the remote system 140 will not write data to frozen log files 210.

[0042] To determine or track the IS-density, in some examples, the compaction controller 160 maintains a snapshot statistics table 300 for each log file 210. The compaction controller 160 may generate a new snapshot statistics table 300 periodically (e.g., whenever the compaction controller 160 executes a heavy compaction) and otherwise will maintain the snapshot statistics table 300 until a new snapshot statistics table 300 is generated.

[0043] In some implementations, the snapshot statistics table 300 stores a number of statistics 302. For example, the snapshot statistics table 300 stores a span statistic 302, 302A; a garbage statistic 302, 302B; and a new blocks statistic 302, 302C. The span statistic 302A defines, per respective snapshot 156 (i.e. per "generation"), a number of bytes in the log file 210 that are mapped or referenced by the respective snapshot 156 and each previous snapshot 156 (i.e., snapshots captured at an earlier point in time relative to the respective snapshot 156). The garbage statistic 302B defines, per respective snapshot 156, an amount of garbage bytes (i.e., bytes that are not referenced by any checkpoint 216) if the respective snapshot 156 and all previous snapshots 156 (i.e., snapshots captured at an earlier point in time relative to the respective snapshot 156) were deleted. The new block statistic 302C defines, per snapshot 156, a number of bytes in the log file 210 that are new relative to the previous snapshot 156.

[0044] Turning to the exemplary snapshot statistics table 300 of FIG. 3, here, the snapshot statistics table 300 includes statistics for five snapshots 156 (i.e., generations 1–5) and for seventeen bytes (or, alternatively, blocks). The first snapshot 156 (i.e., "Gen

1") includes ten new data bytes (indicated by the cross-hatch) written to the associated log file 210 (i.e., at byte locations 1–5, 8–10, 13, and 14) and seven byte locations are still empty. Accordingly, the span statistic 302A and the new block statistic 302C for the first snapshot 156 are ten. The garbage statistic 302B (which is four in this example), requires knowledge from the next snapshot 156 to determine. The second snapshot includes seven data bytes written to the associated log file (i.e., at byte locations 3–6, 9, 10, and 13). Four of the data bytes used by the first snapshot 156 are no longer used by the second snapshot, and therefore the garbage statistic 302B for the first snapshot 156 is four. The second snapshot 156 uses one additional data byte not used by the first snapshot 156, so the span statistic 302A for the second snapshot (i.e., "Gen 2") grows to eleven while the new block statistic 302C is set to one. A third snapshot 156 includes data bytes at byte locations 4, 5, 9, 11, 12, 15, and 16. The four new bytes added (i.e., at byte locations 11, 12, 15, and 16) set the span statistic 302A for the third snapshot 156 to fifteen and the new block statistic 302C to four. The snapshot statistics table 300 continues in this fashion for each snapshot 156 that references the associated log file 210. In some examples, the compaction controller 160 builds each snapshot statistics table 300 when performing a compaction (e.g., a heavy compaction) using the corresponding interval maps 162 and checkpoints 216.

[0045] At the beginning of a compaction, the compaction controller 160 may determine a number of checkpoints 216 and determine which checkpoints 216 have been deleted. However, storing all possibilities in the snapshot statistics table 300 generally requires significant resources as the number of snapshots 156 grows (i.e., in order to have data in case any subset of snapshots 156 are deleted). Instead, in some examples, the snapshot statistics table 300 is ideal for rolling schedules of snapshots 156 where the older snapshots 156 are deleted first. For example, if all snapshots 156 with generation a lower than N are removed, the compaction controller 160 determines a minimum number of garbage bytes based on the garbage statistic 302B from the N-1 snapshot 156. This is a minimum, as there may be potentially more garbage bytes from live device activity or from other deleted snapshots 156 with a generation larger than N. Accordingly, the compaction controller 160 may determine an upper bound of the IS-density of a respective log with

$$ISD = \frac{L-garbage[N-1]}{L} \quad (1)$$

[0046] Here, *ISD* refers to the IS-density, *L* refers to a size (e.g., in bytes) of the respective log file 210, and *garbage*[*N-1*] refers to the garbage statistic 302B of the previous snapshot 156.

[0047] Using the new block statistic 302C from the snapshot statistics table 300, the compaction controller 160, in some examples, determines major changes in bytes mapped by different snapshots 156. In the example snapshot statistics table 300 of FIG. 3, it is clear that, while the third snapshot 156 (i.e., "Gen 3") reuses a lot of blocks used by previous snapshots 156, the snapshot 156 also introduces a lot of fresh data. When two adjacent snapshots 156 both introduce a lot of fresh data, deletion of the older snapshot 156 may indicate an appropriate time to execute a heavy compaction. The new block statistic 302C can correctly be summed for adjacent snapshots. For example, when generations [X, Y] are deleted and their summed new bytes (determined from the new block statistic 302C) is significant (e.g., above a threshold amount), while the new block statistic 302C for the snapshot 156 [Y+1] is also significant, then a compaction may also yield significant byte savings.

[0048] FIG. 4 is a flowchart of an exemplary arrangement of operations for a method 400 of garbage collection in log-based block devices with snapshots. The computer-implemented method 400, when executed by data processing hardware 144, causes the data processing hardware 144 to perform operations. The method 400, at operation 402, includes obtaining a request 20 to compact a plurality of log files 210 of a log-structured volume 146P. Each log file 210 of the plurality of log files 210 may include fresh block runs 214F in use by the log-structured volume 146P and stale block runs 164S no longer in use by the log-structured volume 146P. The log-structured volume 146P includes a plurality of snapshots 156. Each snapshot 156 of the plurality of snapshots 156 is representative of a state of the log-structured volume 146P at a different point in time.

[0049] At operation 404, the method 400 includes, for each respective snapshot 156 of the plurality of snapshots 156 of the log-structured volume 146P, determining the fresh block runs 214F of the plurality of log files 210 used by the respective snapshot 156. Optionally, the method 400 includes using a plurality of interval maps 162 to determine the fresh block runs 214F. Each respective interval map 162 of the plurality of interval maps 162 specifies the fresh block runs 214F of a respective log file 210 for each snapshot 156 of the plurality of snapshots 156. For each respective log file 210 of the plurality of log files 210, the method 400, at operation 406, includes writing the fresh block runs 214F of the respective log file 210 to a respective compacted log file 210 and generating a respective per-log diff file 220 mapping a location of the written fresh block runs 214F at the respective log file 210 to a location of the written fresh block runs 214F at the respective compacted log file 210.

[0050] At operation 408, the method 400 includes, for each respective snapshot 156 of the plurality of snapshots 156 of the log-structured volume 146P, generating a respective checkpoint 216 based on respective per-log diff files 220. At operation 410, the method 400 includes deleting each respective log file 210 of the plurality of log files 210.

[0051] FIG. 5 is a schematic view of an example computing device 500 that may be used to implement the systems and methods described in this document. The computing device 500 is intended to represent various forms of digital computers, such as laptops, desktops, workstations, personal digital assistants, servers, blade servers, mainframes, and other appropriate computers. The components shown here, their connections and relationships, and their functions, are meant to be exemplary only, and are not meant to limit implementations of the inventions described and/or claimed in this document.

[0052] The computing device 500 includes a processor 510, memory 520, a storage device 530, a high-speed interface/controller 540 connecting to the memory 520 and high-speed expansion ports 550, and a low speed interface/controller 560 connecting to a low speed bus 570 and a storage device 530. Each of the components 510, 520, 530, 540, 550, and 560, are interconnected using various busses, and may be mounted on a common motherboard or in other manners as appropriate. The processor 510 can process instructions for execution within the computing device 500, including instructions stored in the memory 520 or on the storage device 530 to display graphical information for a graphical user interface (GUI) on an external input/output device, such as display 580 coupled to high speed interface 540. In other implementations, multiple processors and/or multiple buses may be used, as appropriate, along with multiple memories and types of memory. Also, multiple computing devices 500 may be connected, with each device providing portions of the necessary operations (e.g., as a server bank, a group of blade servers, or a multi-processor system).

[0053] The memory 520 stores information non-transitorily within the computing device 500. The memory 520 may be a computer-readable medium, a volatile memory unit(s), or non-volatile memory unit(s). The non-transitory memory 520 may be physical devices used to store programs (e.g., sequences of instructions) or data (e.g., program state information) on a temporary or permanent basis for use by the computing device 500. Examples of non-volatile memory include, but are not limited to, flash memory and read-only memory (ROM) / programmable read-only memory (PROM) / erasable programmable read-only memory (EPROM) / electronically erasable programmable read-only memory (EEPROM) (e.g., typically used for firmware, such as boot programs). Examples of volatile memory include, but are not limited to, random access memory (RAM), dynamic random access memory (DRAM), static random access memory (SRAM), phase change memory (PCM) as well as disks or tapes.

[0054] The storage device 530 is capable of providing mass storage for the computing device 500. In some implementations, the storage device 530 is a computer-readable medium. In various different implementations, the storage device 530 may be a floppy disk device, a hard disk device, an optical disk device, or a tape device, a flash memory or other similar solid state memory device, or an array of devices, including devices in a storage area network or other configurations. In additional implementations, a computer program product is tangibly embodied in an information carrier. The computer program product contains instructions that, when executed, perform one or more methods, such as those described above. The information carrier is a computer- or machine-readable medium, such as the memory 520, the storage device 530, or memory on processor 510.

[0055] The high speed controller 540 manages bandwidth-intensive operations for the computing device 500, while the low speed controller 560 manages lower bandwidth-intensive operations. Such allocation of duties is exemplary only. In some implementations, the high-speed controller 540 is coupled to the memory 520, the display 580 (e.g., through a graphics processor or accelerator), and to the high-speed expansion ports 550, which may accept various expansion cards (not shown). In some implementations, the low-speed controller 560 is coupled to the storage device 530 and a low-speed expansion port 590. The low-speed expansion port 590, which may include various communication ports (e.g., USB, Bluetooth, Ethernet, wireless Ethernet), may be coupled to one or more input/output devices, such as a keyboard, a pointing device, a scanner, or a networking device such as a switch or router, e.g., through a network adapter.

[0056] The computing device 500 may be implemented in a number of different forms, as shown in the figure. For example, it may be implemented as a standard server 500a or multiple times in a group of such servers 500a, as a laptop computer 500b, or as part of a rack server system 500c.

[0057] Various implementations of the systems and techniques described herein can be realized in digital electronic and/or optical circuitry, integrated circuitry, specially designed ASICs (application specific integrated circuits), computer hardware, firmware, software, and/or combinations thereof. These various implementations can include implementation in one or more computer programs that are executable and/or interpretable on a programmable system including at least one programmable processor, which may be special or general purpose, coupled to receive data and instructions from, and to transmit data and instructions to, a storage system, at least one input device, and at least one output device.

[0058] A software application (i.e., a software resource) may refer to computer software that causes a computing device to perform a task. In some examples, a software application may be referred to as an "application," an "app," or a "program." Example applications include, but are not limited to, system diagnostic applications, system management applications, system maintenance applications, word processing applications, spreadsheet applications, messaging applications, media streaming applications, social networking applications, and gaming applications.

[0059] These computer programs (also known as programs, software, software applications or code) include machine instructions for a programmable processor, and can be implemented in a high-level procedural and/or object-oriented programming language, and/or in assembly/machine language. As used herein, the terms “machine-readable medium” and “computer-readable medium” refer to any computer program product, non-transitory computer readable medium, apparatus and/or device (e.g., magnetic discs, optical disks, memory, Programmable Logic Devices (PLDs)) used to provide machine instructions and/or data to a programmable processor, including a machine-readable medium that receives machine instructions as a machine-readable signal. The term “machine-readable signal” refers to any signal used to provide machine instructions and/or data to a programmable processor.

[0060] The processes and logic flows described in this specification can be performed by one or more programmable processors, also referred to as data processing hardware, executing one or more computer programs to perform functions by operating on input data and generating output. The processes and logic flows can also be performed by special purpose logic circuitry, e.g., an FPGA (field programmable gate array) or an ASIC (application specific integrated circuit). Processors suitable for the execution of a computer program include, by way of example, both general and special purpose microprocessors, and any one or more processors of any kind of digital computer. Generally, a processor will receive instructions and data from a read only memory or a random access memory or both. The essential elements of a computer are a processor for performing instructions and one or more memory devices for storing instructions and data. Generally, a computer will also include, or be operatively coupled to receive data from or transfer data to, or both, one or more mass storage devices for storing data, e.g., magnetic, magneto optical disks, or optical disks. However, a computer need not have such devices. Computer readable media suitable for storing computer program instructions and data include all forms of non-volatile memory, media and memory devices, including by way of example semiconductor memory devices, e.g., EPROM, EEPROM, and flash memory devices; magnetic disks, e.g., internal hard disks or removable disks; magneto optical disks; and CD ROM and DVD-ROM disks. The processor and the memory can be supplemented by, or incorporated in, special purpose logic circuitry.

[0061] To provide for interaction with a user, one or more aspects of the disclosure can be implemented on a computer having a display device, e.g., a CRT (cathode ray tube), LCD (liquid crystal display) monitor, or touch screen for displaying information to the user and optionally a keyboard and a pointing device, e.g., a mouse or a trackball, by which the user can provide input to the computer. Other kinds of devices can be used to provide interaction with a user as well; for example, feedback provided to the user can be any form of sensory feedback, e.g., visual feedback, auditory feedback, or tactile feedback; and input from the user can be received in any form, including acoustic, speech, or tactile input. In addition, a computer can interact with a user by sending documents to and receiving documents from a device that is used by the user; for example, by sending web pages to a web browser on a user's client device in response to requests received from the web browser.

[0062] A number of implementations have been described. Nevertheless, it will be understood that various modifications may be made without departing from the spirit and scope of the disclosure. Accordingly, other implementations are within the scope of the following claims.

WHAT IS CLAIMED IS:

1. A computer-implemented method (400) executed by data processing hardware (144) that causes the data processing hardware (144) to perform operations comprising:

obtaining a request (20) to compact a plurality of log files (210) of a log-structured volume (146P), the log-structured volume (146P) comprising a plurality of snapshots (156), each snapshot (156) of the plurality of snapshots (156) representative of a state of the log-structured volume (146P) at a different point in time;

for each respective snapshot (156) of the plurality of snapshots (156) of the log-structured volume (146P), determining fresh block runs (214) of the plurality of log files (210) used by the respective snapshot (156);

for each respective log file (210) of the plurality of log files (210):

writing the fresh block runs (214) of the respective log file (210) to one or more respective compacted log files (210); and

generating a respective per-log diff file mapping a location of the written fresh block runs (214) at the respective log file (210) to a location of the written fresh block runs (214) at the one or more respective compacted log files (210);

for each respective snapshot (156) of the plurality of snapshots (156) of the log-structured volume (146P), generating a respective checkpoint (216) based on respective per-log diff files (220); and

deleting each respective log file (210) of the plurality of log files (210).

2. The method (400) of claim 1, wherein:

determining the fresh block runs (214) comprises using a plurality of interval maps (162), each respective interval map (162) of the plurality of interval maps (162) specifying the fresh block runs (214) of a respective log file (210) for each snapshot (156) of the plurality of snapshots (156); and

the operations further comprise generating, using the plurality of log files (210) and the plurality of snapshots (156), the plurality of interval maps (162).

3. The method (400) of claim 2, wherein generating the plurality of interval maps (162) comprises scanning, for each snapshot (156) of the plurality of snapshots (156), each log file (210) of the plurality of log files (210).

4. The method (400) of any of claims 1–3, wherein each fresh block (212) run and each stale block (212) run comprises a tuple comprising a log file name, an offset, and a byte count.

5. The method (400) of any of claims 1–4, wherein the operations further comprise selecting the plurality of log files (210) from a set of log files (210) based on a ratio of a sum of a size of the fresh block runs (214) of each respective log file (210) over a total size of each respective log file (210).

6. The method (400) of claim 5, wherein the operations further comprise, for each respective log file (210) of the set of log files (210), determining the ratio for each respective log file (210) using a respective interval map (162) specifying the fresh block runs (214) of a respective log file (210) for each snapshot (156) of the plurality of snapshots (156).

7. The method (400) of any of claims 1–6, wherein each log file (210) of the plurality of log files (210) is frozen.

8. The method (400) of any of claims 1–7, wherein obtaining the request (20) to compact the plurality of log files (210) comprises determining a threshold amount of time has passed since a previous compaction.

9. The method (400) of any of claims 1–8, wherein each respective per-log diff file (220) comprises an ordered list of tuples, each tuple representing a respective fresh block (212) run and comprising an offset, a log file name, and a byte count.

10. The method (400) of any of claims 1–9, wherein the operations further comprise, for each deleted respective log file (210), deleting the respective per-log diff file (220).

11. The method (400) of any of claims 1–10, wherein each log file (210) of the plurality of log files (210) comprises the fresh block runs (214) representing contiguous blocks of data in use by the log-structured volume (146P) and stale block runs (214) representing contiguous blocks of data no longer in use by the log-structured volume (146P).

12. A system (100) comprising:

data processing hardware (144); and

memory hardware (146) in communication with the data processing hardware (144), the memory hardware (146) storing instructions that when executed on the data processing hardware (144) cause the data processing hardware (144) to perform operations comprising:

obtaining a request (20) to compact a plurality of log files (210) of a log-structured volume (146P), the log-structured volume (146P) comprising a plurality of snapshots (156), each snapshot (156) of the plurality of snapshots (156) representative of a state of the log-structured volume (146P) at a different point in time;

for each respective snapshot (156) of the plurality of snapshots (156) of the log-structured volume (146P), determining fresh block runs (214) of the plurality of log files (210) used by the respective snapshot (156);

for each respective log file (210) of the plurality of log files (210):

writing the fresh block runs (214) of the respective log file (210) to one or more respective compacted log files (210); and

generating a respective per-log diff file mapping a location of the written fresh block runs (214) at the respective log file (210) to a location of the written fresh block runs (214) at the one or more respective compacted log files (210);

for each respective snapshot (156) of the plurality of snapshots (156) of the log-structured volume (146P), generating a respective checkpoint (216) based on respective per-log diff files (220); and deleting each respective log file (210) of the plurality of log files (210).

13. The system (100) of claim 12, wherein:

determining the fresh block runs (214) comprises using a plurality of interval maps (162), each respective interval map (162) of the plurality of interval maps (162) specifying the fresh block runs (214) of a respective log file (210) for each snapshot (156) of the plurality of snapshots (156); and

the operations further comprise generating, using the plurality of log files (210) and the plurality of snapshots (156), the plurality of interval maps (162).

14. The system (100) of claim 13, wherein generating the plurality of interval maps (162) comprises scanning, for each snapshot (156) of the plurality of snapshots (156), each log file (210) of the plurality of log files (210).

15. The system (100) of any of claims 12–14, wherein each fresh block (212) run and each stale block (212) run comprises a tuple comprising a log file name, an offset, and a byte count.

16. The system (100) of any of claims 12–15, wherein the operations further comprise selecting the plurality of log files (210) from a set of log files (210) based on a ratio of a sum of a size of the fresh block runs (214) of each respective log file (210) over a total size of each respective log file (210).

17. The system (100) of claim 16, wherein the operations further comprise, for each respective log file (210) of the set of log files (210), determining the ratio for each respective log file (210) using a respective interval map (162) specifying the fresh block runs (214) of a respective log file (210) for each snapshot (156) of the plurality of snapshots (156).

18. The system (100) of any of claims 12–17, wherein each log file (210) of the plurality of log files (210) is frozen.

19. The system (100) of any of claims 12–18, wherein obtaining the request (20) to compact the plurality of log files (210) comprises determining a threshold amount of time has passed since a previous compaction.

20. The system (100) of any of claims 12–19, wherein each respective per-log diff file (220) comprises an ordered list of tuples, each tuple representing a respective fresh block (212) run and comprising an offset, a log file name, and a byte count.

21. The system (100) of any of claims 12–20, wherein the operations further comprise, for each deleted respective log file (210), deleting the respective per-log diff file (220).

22. The system (100) of any of claims 12–21, wherein each log file (210) of the plurality of log files (210) comprises the fresh block runs (214) representing contiguous blocks of data in use by the log-structured volume (146P) and stale block runs (214) representing contiguous blocks of data no longer in use by the log-structured volume (146P).

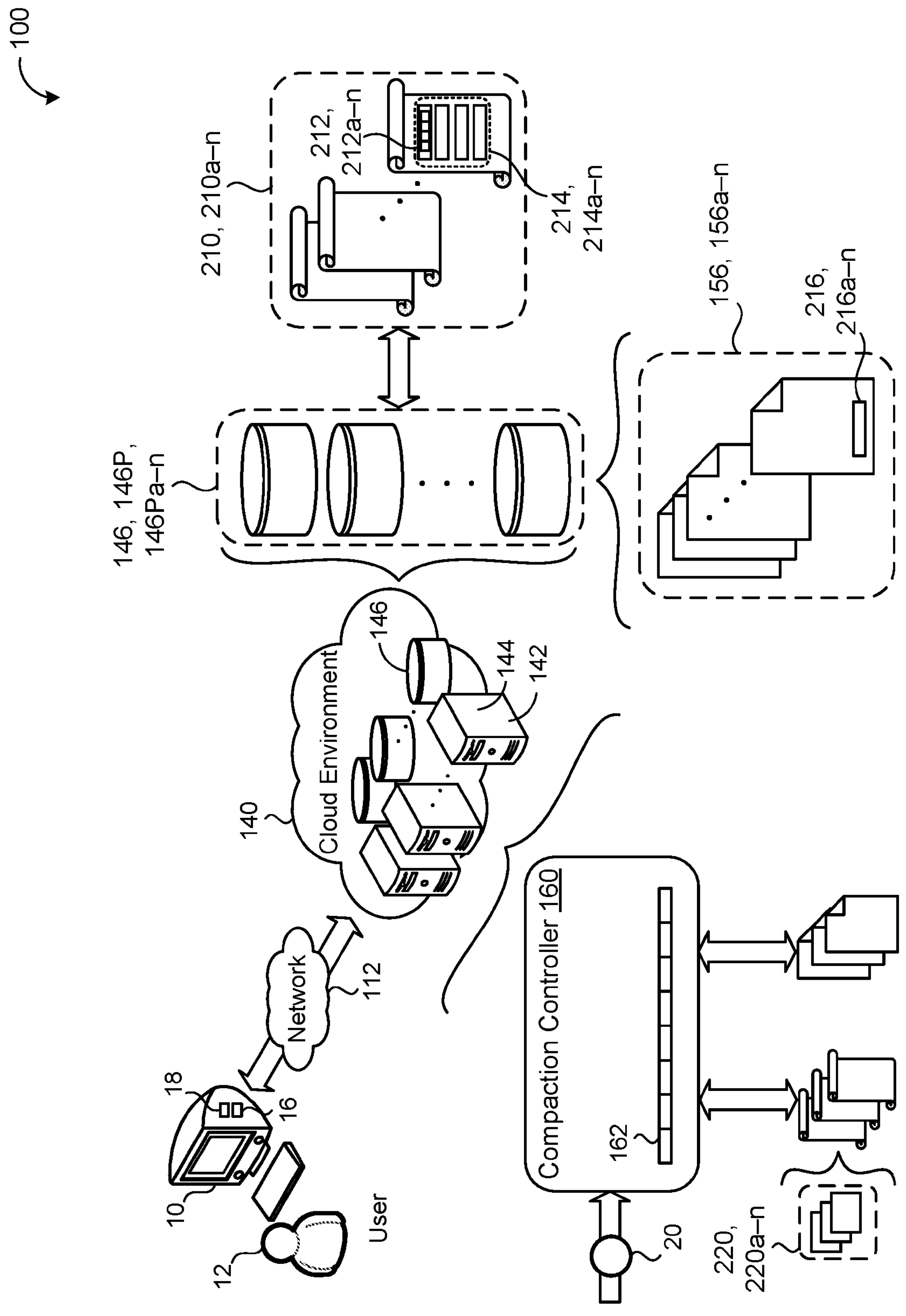

100
210, 210a–n
212, 212a–n
214, 214a–n
146, 146P, 146Pa–n
156, 156a–n
216, 216a–n
Cloud Environment
140
146
144
142
Network
112
18
16
10
12
User
Compaction Controller 160
162
20
220, 220a–n

FIG. 1

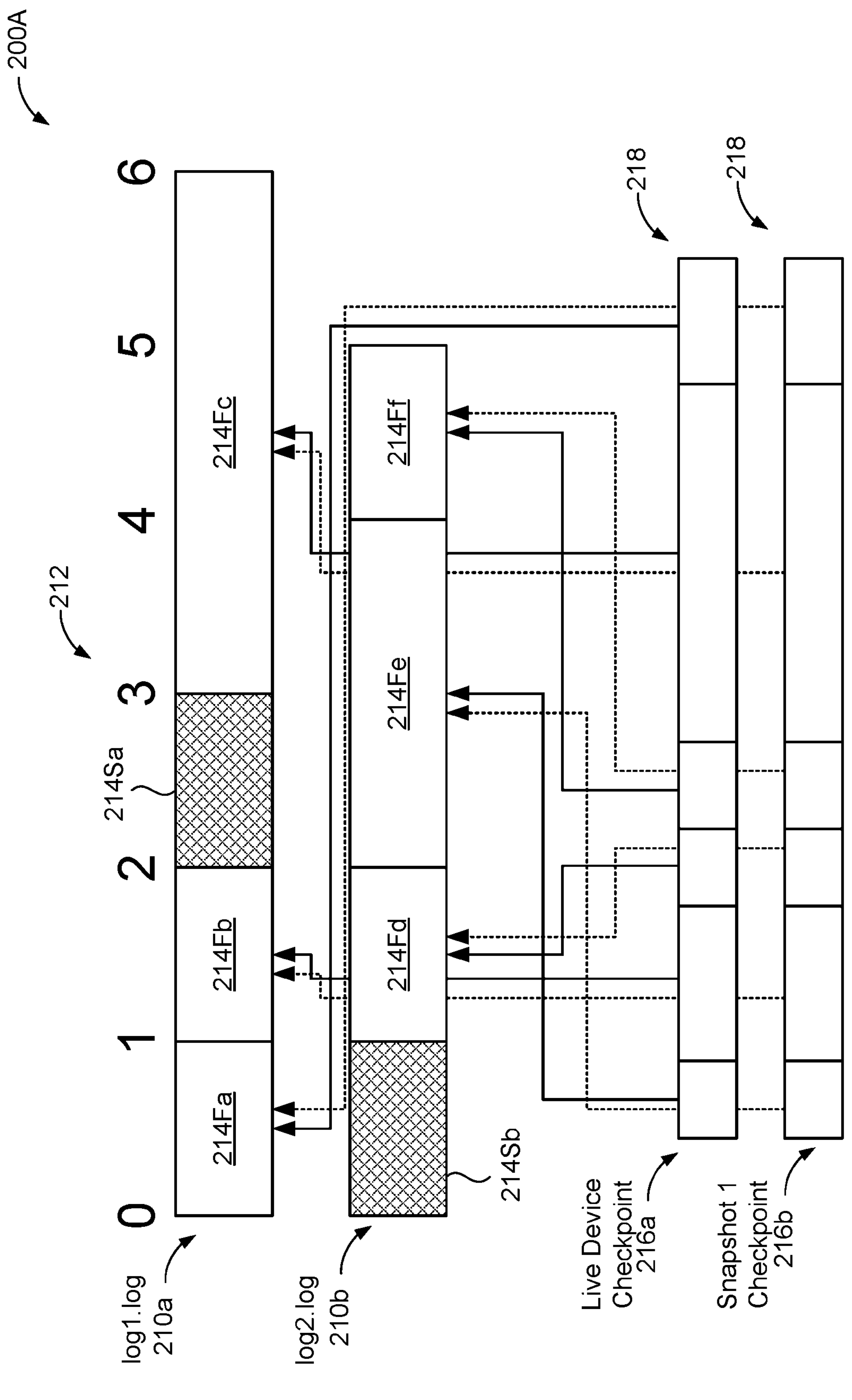

200A
212
0
1
2
3
4
5
6
214Fa
214Fb
214Sa
214Fc
214Sb
214Fd
214Fe
214Ff
218
218
log1.log 210a
log2.log 210b
Live Device Checkpoint 216a
Snapshot 1 Checkpoint 216b

FIG. 2A

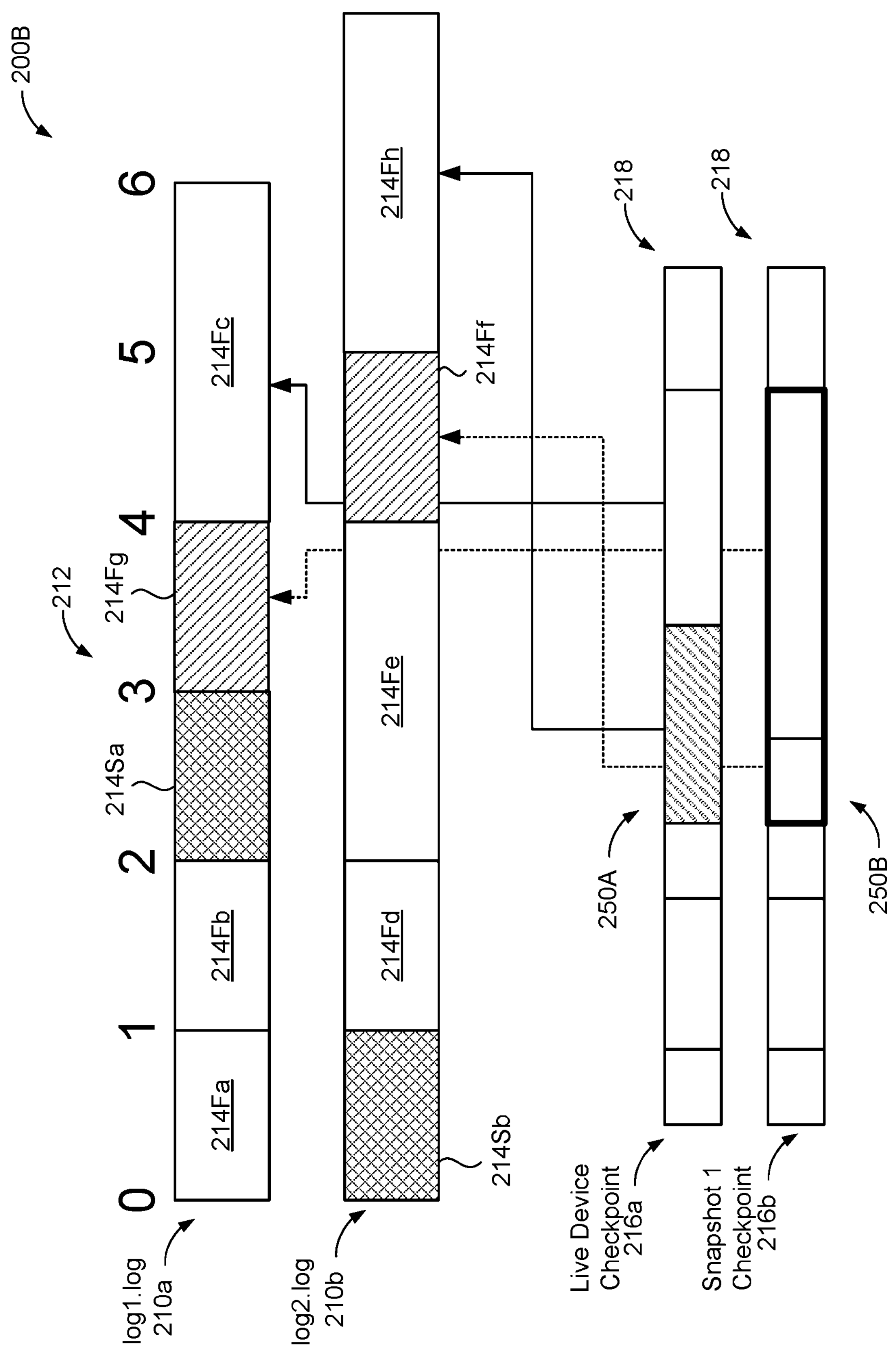

200B
212
0
1
2
3
4
5
6
log1.log
210a
214Fa
214Fb
214Sa
214Fg
214Fc
log2.log
210b
214Sb
214Fd
214Fe
214Ff
214Fh
Live Device
Checkpoint
216a
Snapshot 1
Checkpoint
216b
218
218
250A
250B

FIG. 2B

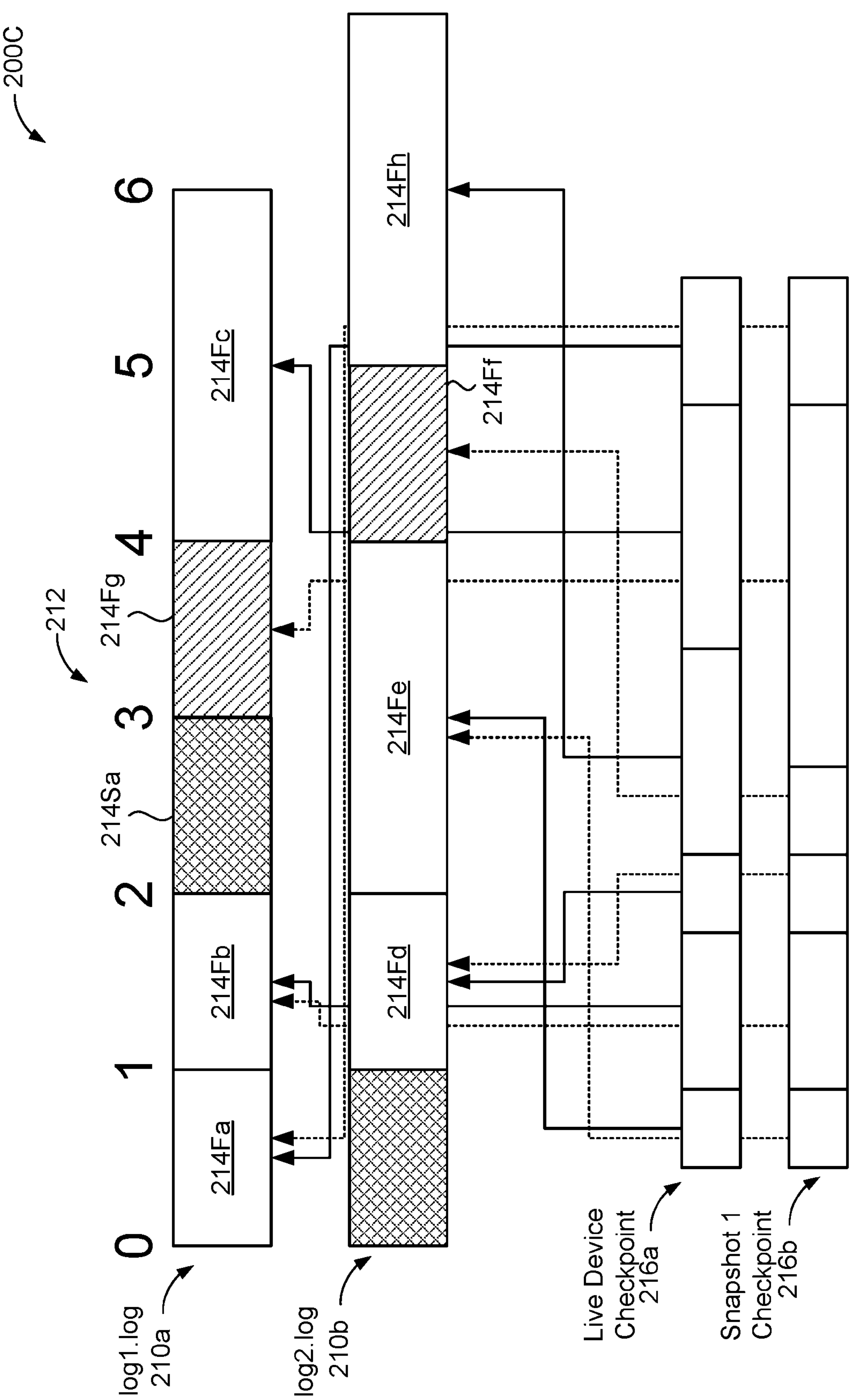

200C
212
0
1
2
3
4
5
6
214Fa
214Fb
214Sa
214Fg
214Fc
214Fd
214Fe
214Ff
214Fh
log1.log
210a
log2.log
210b
Live Device
Checkpoint
216a
Snapshot 1
Checkpoint
216b

FIG. 2C

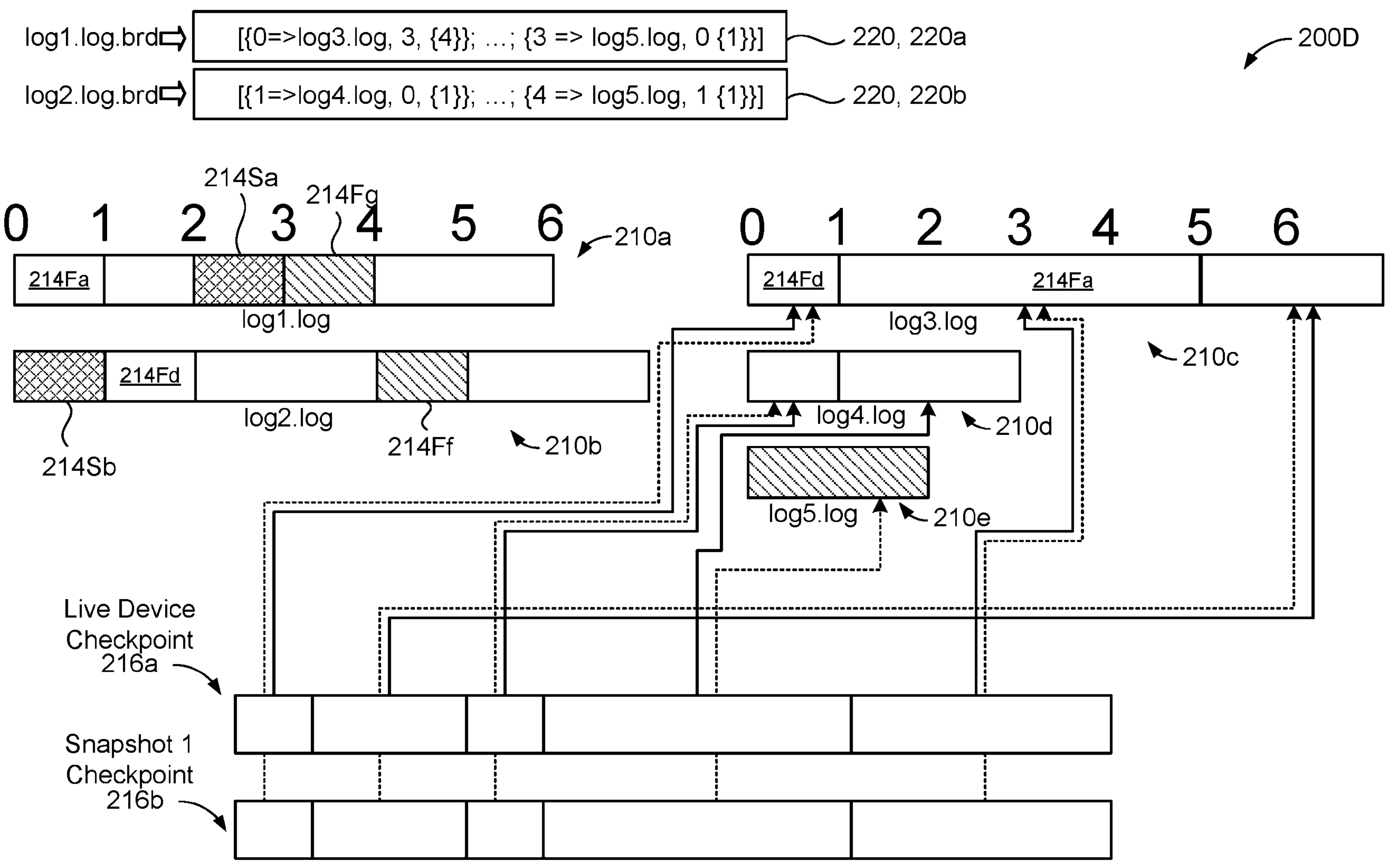
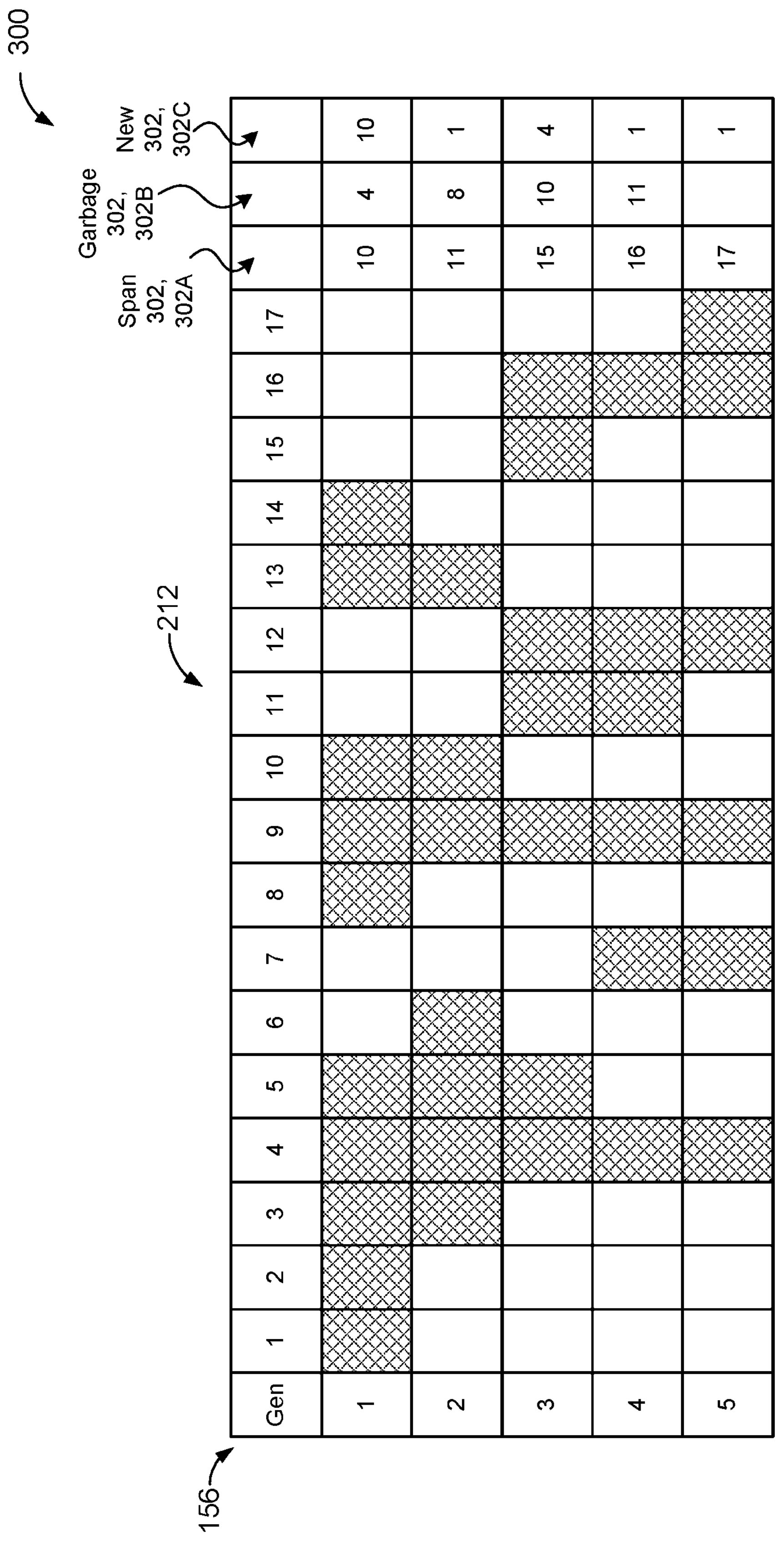

200D
log1.log.brd
[{0=>log3.log, 3, {4}}; …; {3 => log5.log, 0 {1}}]
220, 220a
log2.log.brd
[{1=>log4.log, 0, {1}}; …; {4 => log5.log, 1 {1}}]
220, 220b
214Sa
214Fg
0 1 2 3 4 5 6
210a
214Fa
log1.log
214Fd
log2.log
214Sb
214Ff
210b
0 1 2 3 4 5 6
214Fd
214Fa
log3.log
210c
log4.log
210d
log5.log
210e
Live Device Checkpoint 216a
Snapshot 1 Checkpoint 216b

FIG. 2D



Gen	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17			
1	▨	▨	▨	▨	▨			▨	▨	▨			▨	▨				10	4	10
2			▨	▨	▨	▨			▨	▨			▨					11	8	1
3				▨	▨				▨		▨	▨			▨	▨		15	10	4
4				▨			▨		▨		▨	▨				▨		16	11	1
5				▨			▨		▨			▨				▨	▨	17		1

FIG. 3

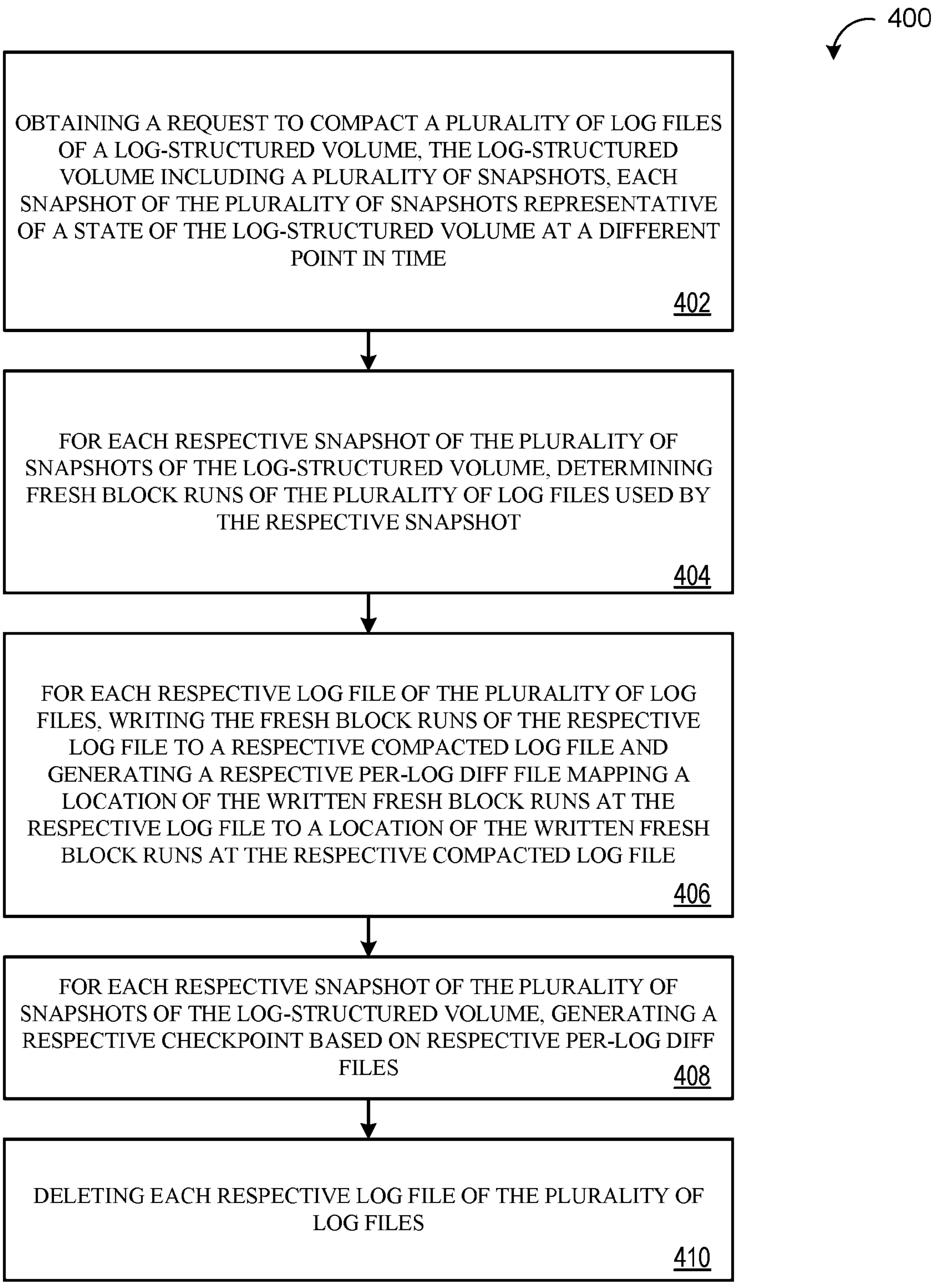

400
OBTAINING A REQUEST TO COMPACT A PLURALITY OF LOG FILES OF A LOG-STRUCTURED VOLUME, THE LOG-STRUCTURED VOLUME INCLUDING A PLURALITY OF SNAPSHOTS, EACH SNAPSHOT OF THE PLURALITY OF SNAPSHOTS REPRESENTATIVE OF A STATE OF THE LOG-STRUCTURED VOLUME AT A DIFFERENT POINT IN TIME
402
FOR EACH RESPECTIVE SNAPSHOT OF THE PLURALITY OF SNAPSHOTS OF THE LOG-STRUCTURED VOLUME, DETERMINING FRESH BLOCK RUNS OF THE PLURALITY OF LOG FILES USED BY THE RESPECTIVE SNAPSHOT
404
FOR EACH RESPECTIVE LOG FILE OF THE PLURALITY OF LOG FILES, WRITING THE FRESH BLOCK RUNS OF THE RESPECTIVE LOG FILE TO A RESPECTIVE COMPACTED LOG FILE AND GENERATING A RESPECTIVE PER-LOG DIFF FILE MAPPING A LOCATION OF THE WRITTEN FRESH BLOCK RUNS AT THE RESPECTIVE LOG FILE TO A LOCATION OF THE WRITTEN FRESH BLOCK RUNS AT THE RESPECTIVE COMPACTED LOG FILE
406
FOR EACH RESPECTIVE SNAPSHOT OF THE PLURALITY OF SNAPSHOTS OF THE LOG-STRUCTURED VOLUME, GENERATING A RESPECTIVE CHECKPOINT BASED ON RESPECTIVE PER-LOG DIFF FILES
408
DELETING EACH RESPECTIVE LOG FILE OF THE PLURALITY OF LOG FILES
410

FIG. 4

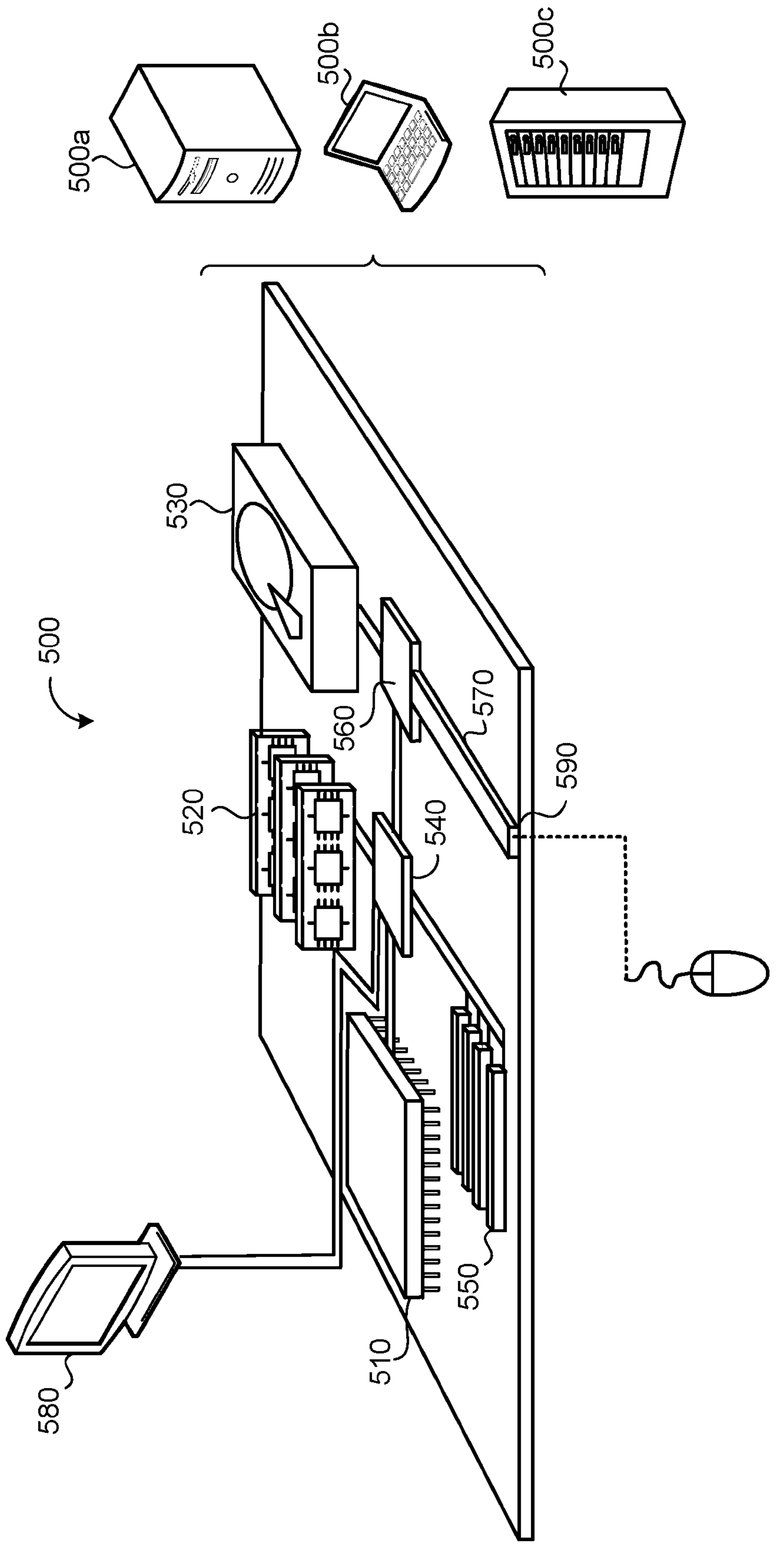


FIG. 5

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2023/083495

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F12/02
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

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2022/066883 A1 (WANG WENGUANG [US] ET AL) 3 March 2022 (2022-03-03)	1-6, 8-17, 19-22
Y	paragraph [0012] - paragraph [0086] claims 1-21; figures 1-7	7,18
Y	CN 115 421 648 A (UNIV TSINGHUA) 2 December 2022 (2022-12-02) paragraph [0016] - paragraph [0115]	7,18
A	US 10 885 022 B1 (TIAN SHIKUN [CN]) 5 January 2021 (2021-01-05) column 10, line 43 - column 38, line 42 claims 1-23; figures 1-15	1-22

-/--

[✓] 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 12 April 2024	Date of mailing of the international search report 26/04/2024
Name and mailing address of the ISA/ European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Authorized officer Alvado Cárcel, Lucía

1

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2023/083495

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	LEE KWANGJIN ET AL: "Validity Tracking Based Log Management for In-Memory Databases", IEEE ACCESS, IEEE, USA, vol. 9, 9 August 2021 (2021-08-09), pages 111493-111504, XP011871767, DOI: 10.1109/ACCESS.2021.3103862 [retrieved on 2021-08-12] page 111496 - page 111499 figures 1-13 -----	1-22
A	US 7 925 856 B1 (GREENE CHRISTOPHER [US]) 12 April 2011 (2011-04-12) column 4, line 45 - column 4, line 62 -----	1-22

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No

PCT/US2023/083495

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2022066883 A1	03-03-2022	US 2022066883 A1	03-03-2022
		US 2023251997 A1	10-08-2023
CN 115421648 A	02-12-2022	NONE	
US 10885022 B1	05-01-2021	CN 115398874 A	25-11-2022
		EP 3695586 A2	19-08-2020
		SG 11202002732T A	29-04-2020
		TW 202111564 A	16-03-2021
		US 10885022 B1	05-01-2021
		WO 2019228569 A2	05-12-2019
US 7925856 B1	12-04-2011	NONE	