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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
18 June 2020 (18.06.2020)

WIPO | PCT

(10) International Publication Number
WO 2020/119050 A1

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

(21) International Application Number:
PCT/CN2019/091397

(22) International Filing Date:
14 June 2019 (14.06.2019)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
62/777,972 11 December 2018 (11.12.2018) US

(71) Applicant: **HUAWEI TECHNOLOGIES CO., LTD.** [CN/CN]; Huawei Administration Building, Bantian, Longgang District, Shenzhen, Guangdong 518129 (CN).

(72) Inventors: **CHEN, Jun**; 19800 Vallco Parkway #541, Santa Clara, California 95014 (US). **CAI, Le**; 833 Alderbrook Ln, Cupertino, California 95014 (US). **PEI, Chunfeng**; 2330 Central Expressway, Santa Clara, California 95050 (US). **DIMITRIJEVIC, Marko**; 272 Palm Valley Boulevard #272, San Jose, California 95123 (US). **CHEN, Jianjun**; 10139 North Blaney Avenue, Cupertino, California 95014 (US). **CHEN, Yu**; 3266 Stimson Way, San Jose, California 95135 (US). **SUN, yang**; 2330 Central Expressway, Santa Clara, California 95050 (US). **DU, Xiaolin**; 2330 Central Expressway, Santa Clara, California 95050 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JO, JP, KE, KG, KH, KN, KP,

(54) Title: WRITE-WRITE CONFLICT DETECTION FOR MULTI-MASTER SHARED STORAGE DATABASE

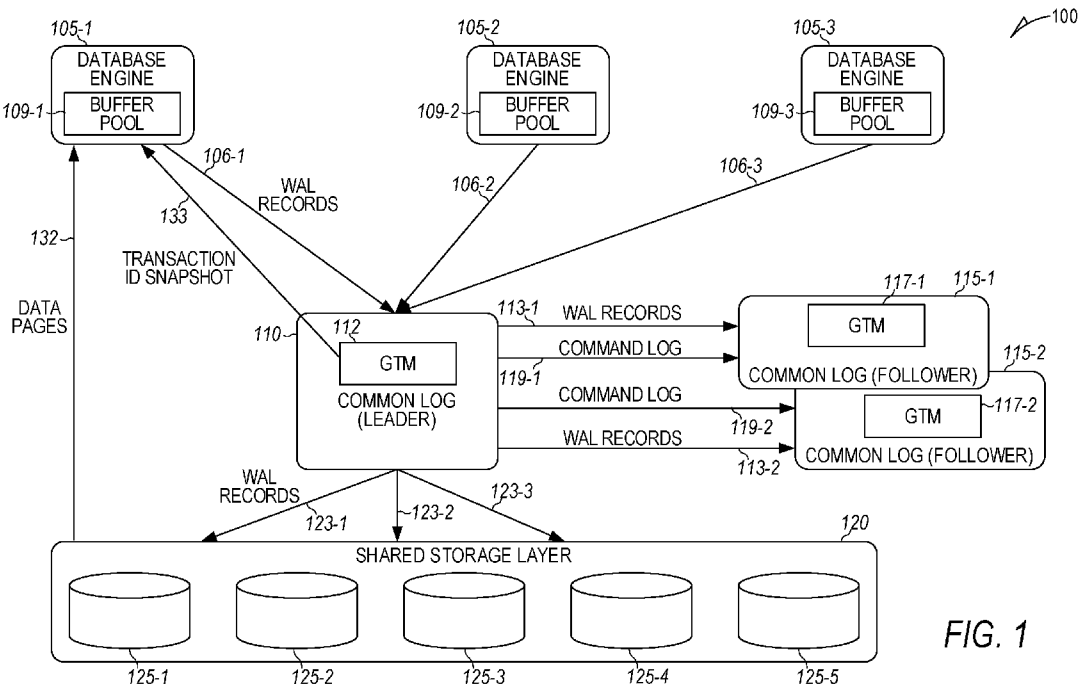

FIG. 1

(57) **Abstract:** Systems and methods are provided to produce an efficient architecture and methodology for multiple database engines to write to a shared data storage to eliminate global locking in the write process. Such systems and methods can use a common log layer between a shared data storage and computing database nodes in which write conflict detection is implemented using a write ahead log record and a log sequence number. A write-write conflict check on a write ahead log record received from a database engine of multiple database engines can be performed by conducting a comparison of a log sequence number, received with the write ahead log record from the database engine, with a global log sequence number in a hash table in the common log. The write ahead log record can be sent to the shared data storage in response to the write ahead log record passing the write-write conflict check.

WO 2020/119050 A1

[Continued on next page]

KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

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

Declarations under Rule 4.17:

— *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))*

WRITE-WRITE CONFLICT DETECTION FOR MULTI-MASTER SHARED STORAGE DATABASE

Field of the Invention

[0001] This application claims priority to and benefit of U.S. Provisional Application No. 62/777,972, filed on December 11, 2018, entitled "Write-Write Conflict Detection for Multi-Master Shared Storage Database," which application is hereby incorporated by reference.

Field of the Invention

[0002] The present disclosure is related to data communications and, in particular, storage of data in shared stored databases.

Background

[0003] Some enterprise multi-master database systems can allow multiple database instances to access a shared storage to read from and write to the shared storage and to generate transaction write ahead log (WAL) records. The multiple database instances can generate transaction write ahead log records so that when any of the database instances fails, the system is still available for read-write access through other instances. When multiple database instances conduct read-write access to shared data, conflicts that arise need to be detected and resolved to maintain data consistency. For example, with respect to database technology, a write–write conflict is a computational occurrence associated with interleaved execution of transactions. Such an interleaved execution can be from a write operation to the same data from different write sources. A write–write conflict, which can be referred to as overwriting uncommitted data, where the act of committing data is making a set of tentative changes permanent.

[0004] Centralized or distributed global locking facilities are often adopted to avoid conflicting access to the shared data. Acquire lock and release lock operations require communication with a global lock manager. These lock acquire and release operations are on a critical path of transaction execution, increasing the latency and producing low throughput.

Summary

[0005] It is an object of various embodiments to provide an efficient architecture and methodology for multiple database engines to write to shared data storage to eliminate global locking in a write process of multiple masters to the shared data storage and to bring the benefit of optimistic concurrency control to multi-master shared storage database systems. Optimistic concurrency control (OCC) is a concurrency control method typically applied to transactional systems, such as relational database management systems and software transactional memory, where concurrency methods are methods to make certain that correct results are generated for concurrent operations, while attaining the results as fast as possible. OCC assumes that multiple transactions can frequently complete without interfering with each other. In OCC, while running operations, data resources are used in transactions without acquiring locks on the resources, and before committing, each transaction can be verified such that no other transaction has modified the data it has read. This object is achieved by the features of the independent claims. Further embodiments of the invention are apparent from the dependent claims, the description and the figures.

[0006] Embodiments are based on using a common log layer between a shared data storage and computing nodes in which write conflict detection is implemented using a write ahead log record and a log sequence number. The detection of a write conflict results from a write conflict check. The write conflict check (write conflict detection) may be referred to as a write-write conflict check (write-write confliction detection) since it is a check of conflict from different write operations. The common log layer can be arranged with other common log layers between storage and computing nodes in an architecture providing high availability. The conflict check can be implemented as a page-level check or tuple-level conflict check. The locks used by the conflict check are local in the common log, without the use of global locking. The locking facility provided by the common log doesn't require any network communications related to lock acquire or release.

[0007] According to a first aspect, an embodiment relates to a computer-implemented method of writing to data storage shared among multiple database engines, the computer-implemented method comprising: performing, in a

common log using one or more processors, a write conflict check on a write ahead log record received from a database engine of the multiple database engines, in which the write conflict check includes conducting a comparison of a log sequence number, received with the write ahead log record from the database engine, with a global log sequence number in a hash table in the common log; and sending the write ahead log record to the data storage shared among multiple database engines, in response to the write ahead log record passing the write conflict check. In this manner, elimination of network communication related to lock acquire and release can be realized.

[0008] This approach removes main bottlenecks from global locking-based conflict resolution. It in essence brings the benefit of optimistic concurrency control (OCC) to a multi-master shared data system. In the manner of OCC, each master runs transactions on data in its local buffer pool without waiting for locks being held by transactions in other masters. At a group commit time, the master can flush log records to the common log, which can perform validation. The term flush to an entity refers to storing to the entity.

[0009] In a first implementation form of the computer-implemented method according to the first aspect as such, conducting the comparison comprises using a tuple identification or a page identification as a key in the hash table, the key associated with an entry in a form of a master identification and a global log sequence number value, the master identification being an identification of a database engine of the multiple database engines.

[0010] In a second implementation form of the computer-implemented method according to the first aspect as such or any preceding implementation form of the first aspect, passing the write conflict check includes the log sequence number being greater than the global log sequence number.

[0011] In a third implementation form of the computer-implemented method according to the first aspect as such or any preceding implementation form of the first aspect, the method comprises, in response to passing the write conflict check, updating the global log sequence number to equal the log sequence number.

[0012] In a fourth implementation form of the computer-implemented method according to the first aspect as such or any preceding implementation form of the first aspect, the method comprises, in response to passing the write conflict

check: inserting the write ahead log record into a group flush write ahead log buffer; and saving all write ahead log records in the group flush write ahead log buffer to a persistent log in the common log.

[0013] In a fifth implementation form of the computer-implemented method according to the first aspect as such or any preceding implementation form of the first aspect, the method comprises replicating the write ahead log record to one or more follower common logs, the one or more follower common logs structured as back-ups to the common log.

[0014] In a sixth implementation form of the computer-implemented method according to the first aspect as such or any preceding implementation form of the first aspect, the write ahead log record received from a database engine is extracted from a batch of write ahead log records received from the database engine having the log sequence number for one or more transactions between the database engine and the data storage.

[0015] In a seventh implementation form of the computer-implemented method according to the first aspect as such or any preceding implementation form of the first aspect, the method comprises extracting another write ahead log record from another batch of write ahead log records received from another database engine of the multiple database engines having another log sequence number for one or more other transactions between the other database engine and the data storage.

[0016] In an eighth implementation form of the computer-implemented method according to the first aspect as such or any preceding implementation form of the first aspect, the method comprises maintaining in a command log all operations and commands that modify an internal state of the common log.

[0017] According to a second aspect, an embodiment relates to a system comprising a memory storage comprising instructions; and one or more processors in communication with the memory storage, wherein the one or more processors execute the instructions to: perform, in a common log, a write conflict check on a write ahead log record received from a database engine of multiple database engines, in which the write conflict check includes a comparison of a log sequence number, received with the write ahead log record from the database engine, with a global log sequence number in a hash table in the common log; and send the write ahead log record to a data storage shared among the multiple

database engines, in response to the write ahead log record passing the write conflict check. In this system, elimination of network communication related to lock acquire/release can be realized.

[0018] In a first implementation form of the system according to the second aspect as such, the comparison comprises using a tuple identification or a page identification as a key in the hash table, the key associated with an entry in a form of a master identification and a global log sequence number value, the master identification being an identification of a database engine of the multiple database engines

[0019] In a second implementation form of the system according to the second aspect as such or any preceding implementation form of the second aspect, passing the write conflict check includes the log sequence number being greater than the global log sequence number.

[0020] In a third implementation form of the system according to the second aspect as such or any preceding implementation form of the second aspect, the one or more processors, in response to passing the write conflict check, update the global log sequence number to equal the log sequence number.

[0021] In a fourth implementation form of the system according to the second aspect as such or any preceding implementation form of the second aspect, the one or more processors, in response to passing the write conflict check, execute: an insertion of the write ahead log record into a group flush write ahead log buffer; and a save operation of all write ahead log records in the group flush write ahead log buffer to a persistent log in the common log.

[0022] In a fifth implementation form of the system according to the second aspect as such or any preceding implementation form of the second aspect, the one or more processors execute replication of the write ahead log record to one or more follower common logs, the one or more follower common logs structured as back-ups to the common log.

[0023] In a sixth implementation form of the system according to the second aspect as such or any preceding implementation form of the second aspect, the one or more processors extract the write ahead log record from a batch of write ahead log records received from the database engine having the log sequence number for one or more transactions between the database engine and the data storage.

[0024] In a seventh implementation form of the system according to the second aspect as such or any preceding implementation form of the second aspect, the one or more processors extract another write ahead log record from another batch of write ahead log records received from another database engine of the multiple database engines having another log sequence number for one or more other transactions between the other database engine and the data storage.

[0025] In an eighth implementation form of the system according to the second aspect as such or any preceding implementation form of the second aspect, the one or more processors maintain in a command log all operations and commands that modify an internal state of the common log.

[0026] In a ninth implementation form of the system according to the eighth implementation form of the second aspect, the system includes the multiple database engines, the data storage shared among the multiple database engines, and one or more follower common logs in addition to the common log.

[0027] The computer-implemented method can be performed by the system. Further features of the computer-implemented method directly result from the functionality of the system.

[0028] The explanations provided for the first aspect and its implementation forms apply equally to the second aspect and the corresponding implementation forms.

[0029] According to a third aspect, embodiments relate to a non-transitory computer-readable medium storing computer instruction, which when executed by one or more processors, cause the one or more processors to perform the steps of any of the computer-implemented methods according to the first aspect or any of its implementation forms when executed on a computer. Thus, the computer-implemented methods can be performed in an automatic and repeatable manner.

[0030] The computer instructions stored by the non-transitory computer-readable medium can be performed by the system. The system can be programmably-arranged to perform the computer instructions.

[0031] According to a fourth aspect, an embodiment relates to a system comprising a means for performing a write conflict check on a write ahead log record received from a database engine of multiple database engines, in which the write conflict check includes a comparison of a log sequence number, received with the write ahead log record from the database engine, with a global

log sequence number in a hash table in a common log, the means for performing the write conflict check including the common log and operationally arranged between the multiple database engines and a shared data storage, the shared data storage being shared among the multiple database engines; and a means for sending the write ahead log record to the shared data storage, in response to the write ahead log record passing the write conflict check.

[0032] In a first implementation form of the system according to the fourth aspect as such, the comparison comprises using a tuple identification or a page identification as a key in the hash table, the key associated with an entry in a form of a master identification and a global log sequence number value, the master identification being an identification of a database engine of the multiple database engines

[0033] In a second implementation form of the system according to the fourth aspect as such or any preceding implementation form of the fourth aspect, passing the write conflict check includes the log sequence number being greater than the global log sequence number.

[0034] In a third implementation form of the system according to the fourth aspect as such or any preceding implementation form of the fourth aspect, the means for performing the write conflict check, in response to passing the write conflict check, updates the global log sequence number to equal the log sequence number.

[0035] In a fourth implementation form of the system according to the fourth aspect as such or any preceding implementation form of the fourth aspect, the means for performing the write conflict check, in response to passing the write conflict check, executes: an insertion of the write ahead log record into a group flush write ahead log buffer; and a save operation of all write ahead log records in the group flush write ahead log buffer to a persistent log in the common log.

[0036] In a fifth implementation form of the system according to the fourth aspect as such or any preceding implementation form of the fourth aspect, the means for performing the write conflict check executes replication of the write ahead log record to one or more follower common logs, the one or more follower common logs structured as back-ups to the common log.

[0037] In a sixth implementation form of the system according to the fourth aspect as such or any preceding implementation form of the fourth aspect, the

means for performing the write conflict check extracts the write ahead log record from a batch of write ahead log records received from the database engine having the log sequence number for one or more transactions between the database engine and the data storage.

[0038] In a seventh implementation form of the system according to the fourth aspect as such or any preceding implementation form of the fourth aspect, the means for performing the write conflict check extracts another write ahead log record from another batch of write ahead log records received from another database engine of the multiple database engines having another log sequence number for one or more other transactions between the other database engine and the data storage.

[0039] In an eighth implementation form of the system according to the fourth aspect as such or any preceding implementation form of the fourth aspect, the means for performing the write conflict check maintains in the command log all operations and commands that modify an internal state of the common log.

[0040] In a ninth implementation form of the system according to the eighth implementation form of the fourth aspect, the system includes the multiple database engines, the data storage shared among the multiple database engines, and one or more follower common logs in addition to the means for performing the write conflict check and the means for sending the write ahead log record to the shared data storage.

[0041] The explanations provided for the first aspect and its implementation forms apply equally to the fourth aspect and the corresponding implementation forms.

[0042] Embodiments of the invention can be implemented in hardware, software or in any combination thereof. Any one of the foregoing examples may be combined with any one or more of the other foregoing examples to create a new embodiment within the scope of the present disclosure.

Brief Description of the Drawings

[0043] Figure 1 is a block diagram of an example system having an architecture with multiple master databases with computation-storage separation, according to an example embodiment.

[0044] Figure 2 depicts example operational components inside the common log of Figure 1 that can facilitate write-write conflict detection, according to an example embodiment.

[0045] Figure 3 is a flow diagram of an example cycle of a write-write conflict detection thread, according to an example embodiment.

[0046] Figures 4 - 10 illustrate challenges and specific solutions of write conflict for a PostgreSQL system, which is an object-relational database management system, as an example, according to an example embodiment.

[0047] Figure 11 is a flow diagram of features of an example method of writing to data storage shared among multiple database engines, according to an example embodiment.

[0048] Figure 12 is a block diagram illustrating circuitry for devices for implementing algorithms and performing methods of providing write-write conflict detection for multi-master shared storage database, according to an example embodiment.

Detailed Description

[0049] In the following description, reference is made to the accompanying drawings that form a part hereof, and in which is shown by way of illustration specific embodiments which may be practiced. These embodiments are described in sufficient detail to enable those skilled in the art to practice the embodiments, and it is to be understood that other embodiments may be utilized, and that structural, logical and electrical changes may be made. The following description of example embodiments is, therefore, not to be taken in a limited sense.

[0050] The functions or algorithms described herein may be implemented in software in an embodiment. The software may comprise computer executable instructions stored on computer readable media or computer readable storage device such as one or more non-transitory memories or other type of hardware-based storage devices, either local or networked. Further, such functions correspond to modules, which may be software, hardware, firmware or any combination thereof. Multiple functions may be performed in one or more modules as desired, and the embodiments described are merely examples. The software may be executed on a digital signal processor, ASIC, microprocessor,

or other type of processor operating on a computer system, such as a personal computer, server or other computer system, turning such computer system into a specifically programmed machine.

[0051] Computer-readable non-transitory media includes all types of computer readable media, including magnetic storage media, optical storage media, and solid state storage media and specifically excludes signals. It should be understood that the software can be installed in and sold with the devices that handle event streams as taught herein. Alternatively, the software can be obtained and loaded into such devices, including obtaining the software via a disc medium or from any manner of network or distribution system, including, for example, from a server owned by the software creator or from a server not owned but used by the software creator. The software can be stored on a server for distribution over the Internet, for example.

[0052] In various embodiments, a system can be implemented with an architecture in which a common log is operated between master nodes and a data storage layer. The master nodes can be realized as database nodes. The common log can be arranged as a write-write detection layer. A write-write detection can also be referred to as a write-write conflict check in which a determination is made as to whether a write transaction from one source conflicts with a related write transaction from another source. The system can be a multi-master database system designed with an architecture in which database instances write log records to storage. Storage nodes can replay, that is, apply the log records to construct data pages.

[0053] In such architectures, master databases can flush log records to a common log, where the common log can conduct a page-level or tuple-level conflict check. A database table can be divided into multiple pages. Each page can have a certain number of records. A tuple is an individual record of a database table. A table can be divided into a smaller unit such as pages and each page can have multiple tuples. All locks used by conflict check can be local in the common log such that there is no global locking. The common log persists and only log records that pass the conflict check are forward to the storage nodes from the common log. This architecture can allow for a high availability (HA) common log layer between storage nodes and compute nodes, where the compute nodes can be databases. This HA can be provided by having multiple

common logs at a conflict detection layer between the storage nodes and compute nodes with one of the multiple common logs being the primary common log with the other common logs of the multiple common logs being duplicates of the primary common log. This duplication defines the HA of the common log layer.

[0054] An architecture with the common log layer can be implemented with novel write-write conflict detection algorithms based on a WAL and log sequence numbers (LSNs), which can be performed on each log record. A log sequence number is an identification number given a WAL record indicating its position in a sequence of records for a transaction. The write-write detection can be a page-level conflict detection or a tuple-level conflict detection. Such approaches to conflicts at a tuple-level or page-level can provide fine grained locks for good parallelism in that, in the common log, conflict checks run on multiple worker threads in parallel. A thread is a sequence of instructions that may execute in parallel with sequences of instructions. The associated locking facility does not require any network communications. Additionally, read operations do not participate in write-write conflict detection, which boosts overall system throughput. These write-write conflict detection techniques can eliminate global locking-based conflict prevention and can bring the benefit of OCC to multi-master shared data/storage database systems.

[0055] Figure 1 is a block diagram of an embodiment of an example system 100 having an architecture with multiple master databases with computation-storage separation. The system 100 can include a number of databases that communicate with a common log 110 to write data to a shared storage layer 120. In this illustration, there are three databases having database engines 105-1, 105-2, and 105-3. Though three database engines, for three databases, are shown, the system 100 can have less than or more than three databases and, hence, can have less than or more than three database engines. Each database engine can include one or more processors and storage devices having instructions stored thereon, which instructions are executable by the one or more processors to perform operations of the database for which it is a component. Such operations of each database can include storing data from a communication with a client of the respective database in the shared storage layer 120 through operation of the common log 110. Each database represented by a database engine can be

arranged as a database node that is a master, which receives structured query language (SQL) queries or update/insert/delete/create requests from users such as client devices to the databases. Data from users to database nodes managed by database engines 105-1, 105-2, and 105-3 can be served through buffer pools 109-1, 109-2, 109-3 of database engines 105-1, 105-2, and 105-3, respectively, or served through persistent storage nodes, if buffer pool is unavailable or missing from a database. When a transaction modifies a tuple, it creates a log record and the log record of the modified tuple is flushed to the common log 110. The common log 110 performs a conflict check on the log record and distributes, if the log record is without conflict, the log record to the shared storage layer 120, where a new version of data is created by performing a log apply operation.

[0056] Each database instance is a master in the architecture of the system 100. Each of database engines 105-1, 105-2, and 105-3 can receive SQL queries from clients and can start transactions. Each of database engines 105-1, 105-2, and 105-3 can flush WAL records to common log 110. The control of loading data pages to the shared storage layer 120 from started transactions in the database engines 105-1, 105-2, and 105-3 is provided by the common log 110 in response to successful completion of conflict checks. However, data pages from the shared storage layer 120 can be loaded by each of database engines 105-1, 105-2, and 105-3. For example, data pages can be loaded along path 132 from the shared storage layer by the database engine 105-1. Appropriate data can be loaded to each of database engines 105-2 and 105-3 in a similar manner.

[0057] The common log 110 can include one or more processors and storage devices having instructions stored thereon, which instructions are executable by the one or more processors to perform operations of the common log 110. The common log 110 can perform a number of functions. It can receive different WAL records from database engines 105-1, 105-2, and 105-3, operating as masters, on paths 106-1, 106-2, and 106-3, respectively. It can conduct write-write conflict detection and can send WAL records that pass their conflict check to the shared storage layer 120. In the architecture of the system 100, the common log 110 operates as a primary common log node in a HA arrangement. As the primary common log node, the common log 110 is a leader common log node that replicates WAL records that pass conflict check to follower common logs 115-1 and 115-2. WAL records can be sent along a path 113-1 to the

follower common log 115-1 and WAL records can be sent along a path 113-2 to the follower common log 115-2. The common log 110, as the leader common log node of the system 100, can replicate its internal state to follower common log 115-1 and follower common log 115-2 by shipping a command log to these follower common logs. The command log can be sent along a path 119-1 to the follower common log 115-1 and the command log can be sent along a path 119-2 to the follower common log 115-2. The command log can keep all the operations, which can be identified as commands, that modify the internal state of the common log 110.

[0058] Though Figure 1 shows two follower common logs, system 100 can have less than or more than two follower common logs in a common log layer between masters and shared storage layer. The implementation of the common log 110 with follower common logs provides an architecture of a HA common log layer between storage and compute nodes. The HA characteristic of such an architecture of a system increases with the number of follower common logs disposed in the common log layer of the system. When the leader common log fails, a follower common log in the HA common log layer becomes the leader common log. The sequencing of which follower common log becomes the leader common log can be predefined or implemented at the time of failure of the leader common log using conventional techniques to elect a leader node among a set of peer nodes.

[0059] The common log 110 can include a global transaction manager (GTM) 112 residing in the common log 110. The GTM 112 can generate transaction identifications (IDs). These transaction IDs can be generated in ascending order. The GTM 112 can maintain a snapshot of active transaction to a database engine, for example path 133 to database engine 105-1. The snapshot can be implemented as a list of active transactions. For the follower common logs 115-1 and 115-2 to be able to replace the common log 110, the follower common logs 115-1 and 115-2 include a GTM 117-1 and GTM 117-2, respectively, that can perform the functions of GTM 112.

[0060] The shared storage layer 120 can be implemented as a shared storage server having storage nodes 125-1, 125-2, 125-3, 125-4, and 125-5. The shared storage server can include one or more processes that control operation of the shared storage sever by execution of instructions stored in the shared storage

layer 120. Though five storage nodes are shown, the shared storage layer 120 can include less than or more than five storage nodes. Each storage node can include one or more storage devices. The storage nodes 125-1, 125-2, 125-3, 125-4, and 125-5 can be structured as distributed storage nodes. The shared storage layer 120 can receive different WAL records from the common log 110 along a number of paths such as paths 123-1, 123-2, and 123-3.

[0061] Figure 2 depicts an embodiment of example operational components inside the common log 110 of Figure 1 that can facilitate write-write conflict detection. Similar or identical operational components are disposed in follower common logs 115-1 and 115-2 to perform operations as a common log leader when a selected follower common log is changed from a follower common log to a leader common log. The operational components can be implemented with storage devices controlled by the one or more processors of the common log 110. In the example shown in Figure 2, these operational components residing in the common log 110 can be implemented as buffers 252 and 254 for write-write conflict detection threads, a hash table 255, a group flush WAL buffer (GFWB) 256, and a persistent log 258. For ease of presentation, the common log 110 is shown receiving a batch of write transaction log records from only two masters 105-1 and 105-2. The shown components can be expanded to handle communication of common log 110 with more than two masters.

[0062] The write-write conflict detection threads in buffers 252 and 254 can be dedicated worker threads that execute write-write conflict detection on buffers of transaction log records in parallel. The dedicated threads in buffers 252 and 254 can run conflict checks on batches of WAL records concurrently. In Figure 2 with respect to threads 1 and 2 associated with buffers 252 and 254, respectively, $W_{ij}(x)$ represents a write transaction log record, where W_{ij} stands for a j^{th} write operation (write transaction log) record from an i^{th} transaction (T_i). The parameter x is either a tuple ID or a page ID depending on whether the conflict detection is a tuple level or a page level conflict detection. With each write transaction log record that passes the write-write conflict check being assigned a global LSN in the common log 110, the term reader LSN represents the latest global LSN known by a transaction. In the example of Figure 2, data for thread 1 includes a reader LSN equal to seven for transactions 1 and 2 for a batch of write transaction records with commit transaction C_2 and C_1. The commit

transactions are from the perspective of the master, which does not perceive any inconsistencies with performing the given commit transaction. The data is received in time order staring on the left and proceeding to the right. In this example, the commit for transaction 2 occurs before the commit for transaction 1, though the write transaction log records for transaction 1 are received before the records for transaction 2. Data for thread 2 includes a reader LSN equal to seven for transactions 3 and 4 for a batch of write transaction records with commit transactions C_3 and C_4. Different from the data for thread 1, the commit for transaction 3 occurs before the commit for transaction 4, along with the write transaction log records for transaction 3 being received before the records for transaction 4.

[0063] The hash table 255 is a write-write conflict detection hash table 255 having entries for a tuple ID or a page ID 261, a master ID and LSN 262, and a bucket (also referred to as a slot) number 263. For a tuple level conflict check, a key for the write-write conflict detection hash table 255 is a tuple ID, and for a page level conflict check, the key is a page ID. The value of the key is in the form {master ID, LSN}, which includes the latest global LSN of the WAL record that modifies the tuple or page and the master ID is the ID of the master that sends the WAL record. Whenever a log record passed the conflict check in the common log 110, it is assigned with a new global LSN. The LSN in {master ID, LSN} represents this global LSN, and master ID is the master that sends the log record to the common log 110. The global LSN is only generated for a WAL record that passes the conflict check. For N masters, the master IDs can be the integers 1 … 10 with each integer assigned to one of the N masters different from the other masters of the N masters. Write-write conflict detection hash table 255 has fixed number of buckets, where each bucket has its own lock. Bucket locks are either tuple lock or page lock depending on whether the conflict check is tuple-level or page-level.

[0064] In common log 110, when a WAL record passes the conflict check, the WAL record is inserted to GFWB 256. Once a WAL record enters GFWB 256, it is assigned a new global LSN. When the GFWB 256 is full or a timer goes off, all the log records in the GFWB 256 are flushed to the persistent log 258 of the common log 110. The persistent log 258 can be implemented as a disk. All log records that pass conflict check are eventually flushed to the persistent log 258

and are also sent to the storage nodes 125-1, 125-2, 125-3, 125-4, and 125-5 of the shared storage layer 120 of Figure 1. Use of the components of the common log 110 is illustrated in Figure 3.

[0065] Figure 3 is a flow diagram 300 of an embodiment of an example cycle of a write-write conflict detection thread. The cycle can be implemented in a common log, such as the common log 110 of Figures 1 and 2, using one or more processors to execute instructions stored in a memory storage, for example at a common log layer of a system. At 305, a batch of WAL records are received at the common log. For example, the WAL records of the write-write conflict detection thread 1 can be received in a buffer, such as buffer 252. Buffer 252 can include many log records and is not limited to four log records. At 310, a WAL record $W_{ij}(x)$ is extracted from the buffer. In a sequence of processing WAL records, the extracted $W_{ij}(x)$ is the next WAL record in the buffer to be conflict checked. At 315, a transaction Id is extracted from $W_{ij}(x)$. At 320, a determination is made as to whether $W_{ij}(x)$ is a write operation record, an abort record, or a commit record. At 325, if the $W_{ij}(x)$ is an abort record, the transaction ID is aborted and a goto 310 operation is performed to extract the next WAL record.

[0066] At 330, if the $W_{ij}(x)$ is a commit record, a determination is made as to whether the transition ID is marked with "has conflict" or other manner of identifying the occurrence of a conflict. At 335, with the transaction ID marked as having a conflict, abort the transaction ID is aborted and a goto 310 is preformed to extract the next WAL record. At 340, with the transaction ID not marked as having a conflict, the transaction ID is committed and the procedure goes to 310 to extract the next WAL record.

[0067] At 345, with the $W_{ij}(x)$ from the determination at 320 being a write operation record, a page ID or a tuple ID from the $W_{ij}(x)$ is extracted and the reader LSN from the $W_{ij}(x)$ is extracted. At 350, a look up in a hash table, such as a write-write conflict detection hash table 255 of Figure 2, by the extracted page ID or the extracted tuple ID is preformed, and the entry having this extracted page ID or the extracted tuple ID is obtained. At 355, a determination is made as to whether the master ID of the extracted entry is not equal to the master ID of the sender of the $W_{ij}(x)$ and whether the LSN of the entry extracted from the hash table is larger than the reader LSN extracted from the $W_{ij}(x)$. At

360, with the condition at 355 met, the transaction ID is marked a "has a conflict" or other equivalent identifier and the procedure goes to 310 to extract the next WAL record. The current condition arises since the received $W_{ij}(x)$ identifies the latest global LSN known by the transaction, represented by the reader LSN, as being less than the current global LSN identifying other write operations have occurred such that the data is not in the expected state.

[0068] At 365, with a finding of no conflict at 355, several actions are taken. The $W_{ij}(x)$ is inserted into the GCWB with the first LSN following the WAL records currently in the GCWB. The LSN of the hash table entry is updated to the LSN from the insertion of the $W_{ij}(x)$ into the GCWB. The master ID of the hash table entry is updated to equal the master ID of the $W_{ij}(x)$. The GCWB is then flushed if full or when a timer reaches a set time. Internal state changes are persisted in a command log that is also shipped to the common log replicas, such as follower common logs 115-1 and 115-2 of Figure 1. Each write-write conflict detection thread can run the above conflict detection algorithm continuously on batches of WAL records.

[0069] Multiple write-write conflict detection threads work concurrently, providing parallelism for a system. Each thread can access the write-write conflict detection hash table of a common log using bucket level locks (close to tuple level locks or page level locks depending on whether the write-write conflict detection is performed on a page level or a tuple level). The write-write conflict detection hash table can be pre-allocated with a fixed number of buckets. Each bucket can have its own lock, where the lock can be derived from the index of the bucket.

[0070] To provide common log HA, all log records that pass conflict check can be propagated to follower common logs. The follower common logs can be referred to as standby common logs. Also, the state of the write-write conflict detection hash table and the GFWB can be replicated to the standby common logs by shipping a command log to each standby common log. The command log stores all the operations/commands that modifies the write-write conflict detection hash table and the GFWB.

[0071] Compared to page-level write-write conflict check, tuple-level write-write conflict check can reduce the number of conflicts and can produce better throughput. At the same time, tuple-level write-write conflict check poses more

challenges for implementation. In the next sections with respect to Figures 4-10, the challenges and the solutions specific are described for PostgreSQL page and tuple layout, as an example. PostgreSQL is an object-relational database management system (ORDBMS), where an ORDBMS is a database management system (DBMS) similar to a relational database, but with an object-oriented database model in which objects, classes and inheritance are directly supported in database schemas and in the query language. PostgreSQL is open-source and compliant with atomicity, consistency, isolation, durability (ACID) principles, which are a set of properties of database transactions intended to guarantee validity even in the event of errors, power failures, etc. PostgreSQL manages concurrency through a system known as multi-version concurrency control (MVCC), which gives each transaction a snapshot of the database, allowing changes to be made without being visible to other transactions until the changes are committed.

[0072] In PostgreSQL, a data file can be referred to as a heap, where the heap can have associated with it a heap tuple (HTUP), a page having a page number (PAGE_NO), and line pointer (LP) with a line pointer number (LP_NO). Also associated with the data file (heap) is an index. An index is a specific structure that organizes a reference to the data that makes it easier to look up. In PostgreSQL, an index can be a copy of the item to be indexed combined with a reference to the actual data location. Associated with an index is an index tuple (ITUP) and LP_NO. In Figures 4-10, the reference labels that being with H, I, and HI refer to heap, index, and heap or index, respectively, and tup refers to a tuple.

[0073] Inserts should not conflict with each other. But PostgreSQL's heap insert and index insert implementation can lead to conflicts between inserts in a multi-master system. Figure 4 is an example of log records for inserts that lead to conflict. If a common log receives the log records from master 405-1 and master 405-2, only those from one master can win. This can lead to a large number of insert conflicts. Master 405-1 submits a transaction log (XLOG) record 1 to insert a HTUP, which is labelled HTUP1. The XLOG record 1 includes the content of HTUP1 with the information that the PAGE_NO is equal to HX and the LP_NO is equal to one. Master 405-1 also submits a XLOG record 2 to insert an ITUP, which is labelled ITUP1. The XLOG record 2

includes the content of ITUP1with the information that the PAGE_NO for XLOG record 2 is equal to IY and the Lp_No is equal to one. The content of ITUP 1 is the PAGE_NO = HX and LP_NO = 1. Master 405-2 submits a transaction log (XLOG) record 1 to insert a HTUP, which is labelled HTUP2. The XLOG record 1 includes the content of HTUP2 with the information that the PAGE_NO is equal to HX and the LP_NO is equal to one. Master 405-2 also submits a XLOG record 2 to insert an ITUP, which is labelled ITUP2. The XLOG record 2 includes the content of ITUP2 with the information that the PAGE_NO for XLOG record 2 is equal to iy and the LP_NO is equal to one. The content of ITUP 2 is the PAGE_NO = HX and LP_NO = 1, which leads to conflict in XLOG record 2 from master 405-1.

[0074] The following is a way to eliminate this kind of conflicts. First, when inserting a heap tuple, instead of just picking an unused LP, a master checks to determine if the LP_NO can be equal to its own master Id. In the example of Figure 4, master 405-1(ID=1) can select LP1 on page HX, and master 405-2 (ID=2) can select LP2 on page HX. Second, when inserting an index tuple, each master can simply pick the next unused LP. But the LP_NO should not appear in the log record. This way, when the log record is applied on the storage node, an overall next LP can be figured out, where the LPS of index tuples are ordered according to the order of index keys.

[0075] Figure 5 illustrates the use of these principles of operation of a common log to provide modified log records for insertions that eliminate conflict associated with Figure 4. As shown in Figure 5, XLOG record 1 from master 405-1, having master ID=1, has a LP_NO = 1 for inserting HTUP1 and XLOG record 2 from master 405-1 has a LP_NO = ϕ (next unused LP), while XLOG record 1 from master 405-2, having master ID = 2, has a LP_NO = 2 and XLOG record 2 from master 405-2 has a LP_NO = ϕ (next unused LP). The associated common log for these two masters can let both masters commit, where the different LP_NOS are not conflicting.

[0076] Inserts from different writers can also conflict with each other when a heap or an index (referred to below as HI) page is close to be full. Figure 6 illustrates an example of inserts with respect to a full page. For example, a master 605-1 sees page heap or index X (HIX) has room for one tuple, and it inserts heap or index tuple (HITUP) 6 (HITUP6) with an unused LP_NO=5 or

next unused LP_NO = ϕ. The associated common log receives XLOG record from master 605-1 for the heap or index TUPLE6 (HITUP6) with the XLOG record having PAGE_NO for the heap or the index, PAGE_NO = X, LP_NO =5 (heap) or next unused LP_NO = ϕ (index), and content of tuple 6 (heap or index). A master 605-2 also sees page HIX has room for one tuple, and master 605-2 inserts HITUP7 with LP_NO = 6 or next unused LP_NO = ϕ. The associated common log receives XLOG record from master 605-2 for the heap or index TUPLE7 (HITUP7) with the XLOG record having PAGE_NO for the heap or the index, PAGE_NO = X, LP_NO = 6 (heap) or next unused LP_NO = ϕ (index), and content of TUPLE 7 (heap or index). When the common log receives the two log records for HITUP6 and HITUP7, it sees no conflicts and commits both of them. However, when these two log records are applied, the storage sees there is no room on page HIX, but the transactions have committed. To detect page full conflicts, the common log maintains free space map (FSM) pages. Each heap and index relation can have a FSM to keep track of available space in the relation. When an insert related log record has no conflict and is ready to commit, the common log also checks whether the increase will cause page size overflow. If so, the common log aborts the transaction. An update adding new version of a tuple is handled similarly.

[0077] Inserts can cause index pages to split. When one writer splits an index page while another writer inserts index tuple to the same page, either the split or the insert should abort. Figure 7 is an example of index page splitting. In Figure 7, master 705-1 inserts ITUP5 making index page IX full. Master 705-1 then tries to insert ITUP6 to page IX. Page IX splits, where the first half of ITUPs stay at page IX, and the second half go to a new page INEW. Both ITUP5 and ITUP6 go to INEW because their index keys are larger. Master 705-2 inserts ITUP7 on page IX supposing ITUP7 has the largest key.

[0078] Figure 8 illustrates roughly log records generated by the two masters of Figure 7 from index page splitting. The master 705-1 generates a XLOG 1 about the index tuple ITUP5, which includes PAGE_NO = IX (index page X), LP_NO = ϕ, and ITUP5 content (for example, PAGE_NO = HX and LP_NO = 1). The master 705-1 generates a XLOG 1 about the index tuple ITUP6 and page IX split, which includes LP_NO of the splitting point, the new PAGE_NO for the new page, and the content of the new page containing ITUP5 and ITUP6. The master

705-2 generates a XLOG 1 about index tuple TUP7, which includes PAGE_NO = IX (index page x), LP_NO = ϕ, and ITUP7 content (for example, PAGE_NO = HX and LP_NO = 2).

[0079] The common log maintains PAGE_NO(s) of recently split index pages and their latest commit LSNs in a write-write conflict detection hash table of the common log. Suppose the common log commits changes of the master 705-1 first, and the common log receives from the master 705-2 the log record about inserting ITUP7 on page IX. The common log sees PAGE_NO = IX just split and the split commit LSN is later than the reader LSN of the log record of master 705-2, and it aborts the change of master 705-2. Similarly, suppose the common log commits the change of master 705-2 first, and the common log receives the log record of master 705-1 about the split of page IX. The reader LSN of the split is earlier than the commit LSN of the insertion by master 705-2 on page IX, and the common log aborts the split of page IX.

[0080] Figure 9 illustrates handling an update-update conflict. Handling updates can be relatively straight forward. Suppose master 905-1 and master 905-2 try to update HTUP0 at {PAGE_NO = HX, LP_NO = 0}. The newer version on master 905-1 is HTUP0' at {PAGE_NO = HX, LP_NO = 1} with the content of HTUP0' and the maximum pages (xmax) equal to the transaction ID (TID1) for master 905-1. The newer version on master 905-2 is HTUP0" at {PAGE_NO = HX, LP_NO = 2} with the content of HTUP0'' and the maximum pages (XMAX) equal to the transaction ID (TID2) for master 905-2. The LP_NO selected on heap page satisfies LP_NO = master id. If the common log commits the log record of master 905-1 first, the common log will reject the change by master 905-2 when it sees the latest commit LSN of HTUP0 at {PAGE_NO = HX, LP_NO = 0} is larger than the reader LSN of the log record of master 905-2 that also contains HTUP0 at {PAGE_NO = HX, LP_NO = 0}. The same reasoning holds for the situation where the common log commits the log record of master 905-2 first.

[0081] Figure 10 illustrates handling an update-delete conflict. Suppose master 1005-1 deletes HTUP0 at {PAGE_NO = HX, LP_NO = 0}, and master 1005-2 updates HTUP0 to HTUP0'. The master 1005-2 puts HTUP0' at {PAGE_NO = HX, LP_NO = 2}. If the associated common log commits the log record of master 1005-1 first, it will reject the change of master 1005-2 when it

sees the latest commit LSN of HTUP0 at {PAGE_NO = HX, LP_NO = 0} is larger than the reader LSN of the log record of master 1005-2 that also contains HTUP0 at {PAGE_NO = HX, LP_NO = 0}. Similar reasoning for the situation where the common log commits the log record of the master 1005-2 first.

[0082] Figure 11 is a flow diagram of features of an embodiment of an example method 1100 of writing to data storage shared among multiple database engines. The method 1100 can be implemented as a computer-implemented method. At 1110, in a common log using one or more processors, a write-write conflict check is performed on a write ahead log record received from a database engine of the multiple database engines, in which the write-write conflict check includes conducting a comparison of a log sequence number, received with the write ahead log record from the database engine, with a global log sequence number in a hash table in the common log. Conducting the comparison can comprise using a tuple identification or a page identification as a key in the hash table, the key associated with an entry in a form of a master identification and a global log sequence number value, where the master identification can be an identification of a database engine of the multiple database engines.

[0083] At 1120, the write ahead log record is sent to the data storage shared among multiple database engines, in response to the write ahead log record passing the write-write conflict check. Passing the write conflict check can include the log sequence number being greater than the global log sequence number. In response to passing the write conflict check, method 1100 or a method similar to method 1100 can include updating the global log sequence number to equal the log sequence number. In response to passing the write conflict check, method 1100 or a method similar to method 1100 can include inserting the write ahead log record into a group flush write ahead log buffer and saving all write ahead log records in the group flush write ahead log buffer to a persistent log in the common log.

[0084] Variations of the method 1100 or methods similar to the method 1100 can include a number of different embodiments that may be combined depending on the application of such methods and/or the architecture of systems in which such methods are implemented. Such methods can comprise replicating the write ahead log record to one or more follower common logs, the one or more follower common logs structured as back-ups to the common log. Such methods

can comprise maintaining in a command log all operations and commands that modify an internal state of the common log.

[0085] In method 1100 or methods similar to the method 1100, the write ahead log record received from a database engine can be extracted from a batch of write ahead log records received from the database engine having the log sequence number for one or more transactions between the database engine and the data storage. Another write ahead log record can be extracted from another batch of write ahead log records received from another database engine of the multiple database engines having another log sequence number for one or more other transactions between the other database engine and the data storage.

[0086] In various embodiments, a non-transitory machine-readable storage device, such as computer-readable non-transitory media, can comprise instructions stored thereon, which, when executed by components of a machine, cause the machine to perform operations, where the operations comprise one or more features similar to or identical to features of methods and techniques described with respect to method 1100, flow diagram 300, variations thereof, and/or features of other methods taught herein such as associated with Figures 1-11. The physical structures of such instructions may be operated on by one or more processors. For example, executing these physical structures can cause the machine to perform operations comprising: performing, in a common log using the one or more processors, a write-write conflict check on a write ahead log record received from a database engine of the multiple database engines, in which the write-write conflict check includes conducting a comparison of a log sequence number, received with the write ahead log record from the database engine, with a global log sequence number in a hash table in the common log; and sending the write ahead log record to the data storage shared among multiple database engines, in response to the write ahead log record passing the write-write conflict check. Conducting the comparison can comprise using a tuple identification or a page identification as a key in the hash table, the key associated with an entry in a form of a master identification and a global log sequence number value, the master identification being an identification of a database engine of the multiple database engines.

[0087] Using the one or more processors to execute the instructions stored on the machine-readable storage device can comprise operations in which passing

the write-write conflict check can include the log sequence number being greater than the global log sequence number. In response to passing the write-write conflict check, the executable operations can include updating the global log sequence number to equal the log sequence number. In response to passing the write-write conflict check, the executable operations can include inserting the write ahead log record into a group flush write ahead log buffer and saving all write ahead log records in the group flush write ahead log buffer to a persistent log in the common log.

[0088] The operations can include the write ahead log record received from a database engine being extracted from a batch of write ahead log records received from the database engine having the log sequence number for one or more transactions between the database engine and the data storage. The operations can include extracting another write ahead log record from another batch of write ahead log records received from another database engine of the multiple database engines having another log sequence number for one or more other transactions between the other database engine and the data storage.

[0089] The operations can include replicating the write ahead log record to one or more follower common logs, where the one or more follower common logs is structured as back-ups to the common log. The operations can include maintaining in a command log all operations and commands that modify an internal state of the common log.

[0090] Figure 12 is a block diagram illustrating circuitry for devices for implementing algorithms and performing methods of providing write-write conflict detection for multi-master shared storage database, according to the teachings herein. Figure 12 depicts a device 1200 having a non-transitory memory storage 1201 storing instructions, a cache 1207, and a processing unit 1202, coupled to a bus 1220. Processing unit 1202 can include one or more processors operatively in communication with non-transitory memory storage 1201 and cache 1207. The one or more processors can be structured to execute the instructions to operate device 1200 as a database engine, a common log, or a shared data storage according to any of the methods taught herein.

[0091] Device 1200 can include a communication interface 1216 operable to communicate among devices and systems associated with an architecture such as the architecture of Figure 1. One or more of multiple databases, common logs,

and shared data storage can be implemented in a cloud to which device 1200 can be associated. Typically, the term "cloud" refers to data processing on a number of virtual servers as opposed to directly on physical machines. A cloud could span across a wide area network (WAN). WAN is also commonly referred to the public Internet or sometimes also a network of leased optic fiber links that inter-connect multiple branch offices of an Enterprise. Or alternately, a cloud could be entirely resident within a private datacenter within an internal local area network. Datacenters of clouds, meaning datacenters that host virtual compute or services, can also provide services for network traffic management from one location on a network to another location on the network or across networks spanning far flung locations over WAN (or Internet). In addition, the term "cloud computing" refers to the software and services executed for users by these servers virtually (over a hypervisor), and typically the user is unaware of the physical location of the servers or datacenter. Further, the datacenter may be a distributed entity. Cloud computing can provide shared computer processing resources and data to computers and other devices on demand over the associated networks. The communication interface 1216 may be part of a data bus that can be used to receive the data traffic for processing.

[0092] Non-transitory memory storage 1201 may be realized as machine-readable media, such as computer-readable media, and may include volatile memory 1214 or non-volatile memory 1208. Device 1200 may include or have access to a computing environment that includes a variety of machine-readable media including as computer-readable media, such as volatile memory 1214, non-volatile memory 1208, removable storage 1211, or non-removable storage 1222. Such machine-readable media may be used with instructions in one or more programs 1218 executed by device 1200. Cache 1207 may be realized as a separate memory component or part of one or more of volatile memory 1214, non-volatile memory 1208, removable storage 1211, or non-removable storage 1222. Memory storage can include random access memory (RAM), read only memory (ROM), erasable programmable read-only memory (EPROM), electrically erasable programmable read-only memory (EEPROM), flash memory or other memory technologies, compact disc read-only memory (CD ROM), Digital Versatile Disks (DVD) or other optical disk storage, magnetic

cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium capable of storing computer-readable instructions.

[0093] Device 1200 may include or have access to a computing environment that includes input interface 1226 and output interface 1224. Output interface 1224 may include a display device, such as a touchscreen, that also may serve as an input device. Input interface 1226 may include one or more of a touchscreen, touchpad, mouse, keyboard, camera, one or more device-specific buttons, one or more sensors integrated within or coupled via wired or wireless data connections to device 1200, and other input devices.

[0094] Device 1200 may operate in a networked environment using a communication connection to connect to one or more other devices that are remote. Such remote devices may be identical or similar to device 1200 or may be different types of devices having features similar or identical to features of device 1200 or other features, as taught herein, to handle procedures associated with providing write-write conflict detection for multi-master shared storage database, according to the teachings herein. The remote devices may include computers, such as database servers. Such remote computers may include a personal computer (PC), server, router, network PC, a peer device or other common network node, or the like. The communication connection may include a Local Area Network (LAN), a Wide Area Network (WAN), cellular, WiFi, Bluetooth, or other networks.

[0095] Machine-readable instructions, such as computer-readable instructions stored on a computer-readable medium, are executable by the processing unit 1202 of the device 1200. A hard drive, CD-ROM, and RAM are some examples of articles including a non-transitory computer-readable medium such as a storage device. The terms machine-readable medium, computer-readable medium, and storage device do not include carrier waves to the extent carrier waves are deemed transitory. Storage can also include networked storage such as a storage area network (SAN).

[0096] Device 1200 can be realized as a computing device that may be in different forms in different embodiments, as part of a network such as a SDN/IoT network. For example, device 1200 may be a smartphone, a tablet, smartwatch, other computing device, or other types of devices having wireless communication capabilities, where such devices include components to engage

in the distribution and storage of items of content, as taught herein. Devices, such as smartphones, tablets, smartwatches, and other types of device having wireless communication capabilities, are generally collectively referred to as mobile devices or user equipment. In addition, some of these devices may be considered as systems for the functions and/or applications for which they are implemented. Further, although the various data storage elements are illustrated as part of the device 1200, the storage may also or alternatively include cloud-based storage accessible via a network, such as the Internet or server based storage.

[0097] In an example embodiment, the device 1200 includes a conflict check module performing, in a common log using one or more processors, a write-write conflict check on a write ahead log record received from a database engine of the multiple database engines, in which the write-write conflict check includes conducting a comparison of a log sequence number, received with the write ahead log record from the database engine, with a global log sequence number in a hash table in the common log, and a transmission module sending the write ahead log record to the data storage shared among multiple database engines, in response to the write ahead log record passing the write-write conflict check. In some embodiments, the device 1200 may include other or additional modules for performing any one of or combination of steps described in the embodiments. Further, any of the additional or alternative embodiments or aspects of the method, as shown in any of the figures or recited in any of the claims, are also contemplated to include similar modules.

[0098] Further, machine-readable storage devices, such as computer-readable non-transitory media, herein, are physical devices that stores data represented by physical structure within the respective device. Such a physical device is a non-transitory device. Examples of machine-readable storage devices can include, but are not limited to, read only memory (ROM), random access memory (RAM), a magnetic disk storage device, an optical storage device, a flash memory, or other electronic, magnetic, and/or optical memory devices. The machine-readable device may be a machine-readable medium such as memory 1201 of Figure 12. Terms such as "memory," "memory module," "machine-readable medium," "machine-readable device," and similar terms should be taken to include all forms of storage media, either in the form of a single

medium (or device) or multiple media (or devices), in all forms. For example, such structures can be realized as centralized database(s), distributed database(s), associated caches, and servers; one or more storage devices, such as storage drives (including but not limited to electronic, magnetic, and optical drives and storage mechanisms), and one or more instances of memory devices or modules (whether main memory; cache storage, either internal or external to a processor; or buffers). Terms such as "memory," "memory module," "machine-readable medium," and "machine-readable device," shall be taken to include any tangible non-transitory medium which is capable of storing or encoding a sequence of instructions for execution by the machine and that cause the machine to perform any one of the methodologies taught herein. The term “non-transitory” used in reference to a "machine-readable device," “medium,” “storage medium,” “device,” or “storage device” expressly includes all forms of storage drives (optical, magnetic, electrical, etc.) and all forms of memory devices (e.g., DRAM, Flash (of all storage designs), SRAM, MRAM, phase change, etc., as well as all other structures designed to store data of any type for later retrieval.

[0099] In various embodiments, a system can be implemented to enable write-write conflict detection for multi-master shared storage database. Such a system can comprise a memory storage comprising instructions and one or more processors in communication with the memory storage. The one or more processors can execute the instructions to: perform, in a common log, a write conflict check on a write ahead log record received from a database engine of multiple database engines, in which the write conflict check includes a comparison of a log sequence number, received with the write ahead log record from the database engine, with a global log sequence number in a hash table in the common log; and send the write ahead log record to a data storage shared among the multiple database engines, in response to the write ahead log record passing the write conflict check. The comparison can comprise using a tuple identification or a page identification as a key in the hash table, where the key is associated with an entry in a form of a master identification and a global log sequence number value, with the master identification being an identification of a database engine of the multiple database engines.

[00100] Variations of such a system or similar systems can include a number of different embodiments that may be combined depending on the application of

such systems and/or the architecture in which such systems are implemented. Passing the write conflict check can include the log sequence number being greater than the global log sequence number. The one or more processors, in response to passing the write conflict check, can update the global log sequence number to equal the log sequence number. The one or more processors, in response to passing the write conflict check, can execute an insertion of the write ahead log record into a group flush write ahead log buffer, and execute a save operation of all write ahead log records in the group flush write ahead log buffer to a persistent log in the common log.

[0100] Such systems can include the one or more processors configured to execute instructions to execute replication of the write ahead log record to one or more follower common logs, with the one or more follower common logs structured as back-ups to the common log. The one or more processors can extract the write ahead log record from a batch of write ahead log records received from the database engine having the log sequence number for one or more transactions between the database engine and the data storage. The one or more processors can extract another write ahead log record from another batch of write ahead log records received from another database engine of the multiple database engines having another log sequence number for one or more other transactions between the other database engine and the data storage. The one or more processors can maintain in a command log all operations and commands that modify an internal state of the common log.

[0101] Such a system or similar systems can include the multiple database engines, the data storage shared among the multiple database engines, and one or more follower common logs in addition to the common log. The multiple database engines can be separate structural units separate from the common logs and the shared data storage. The multiple database engines can communicate with a leader common log of the follower common logs and the shared data storage to transfer data using conventional communicate techniques such as but not limited to transmission control protocol (TCP) and internet protocol (IP). Such system or similar systems can be structured according to any permutation of the features taught herein for write-write conflict detection for multi-master shared storage database.

[0102] In various embodiments, a system can be implemented to enable write-write conflict detection for multi-master shared storage database. Such a system can comprise a means for performing a write conflict check on a write ahead log record received from a database engine of multiple database engines, in which the write conflict check includes a comparison of a log sequence number, received with the write ahead log record from the database engine, with a global log sequence number in a hash table in a common log, the means for performing the write conflict check including the common log and operationally arranged between the multiple database engines and a shared data storage, the shared data storage being shared among the multiple database engines; and a means for sending the write ahead log record to the shared data storage, in response to the write ahead log record passing the write conflict check. The comparison can comprise using a tuple identification or a page identification as a key in the hash table, the key associated with an entry in a form of a master identification and a global log sequence number value, the master identification being an identification of a database engine of the multiple database engines.

[0103] Variations of such a system or similar systems, having a means for performing a write conflict check on a write ahead log record received from a database engine of multiple database engines, can include a number of different embodiments that may be combined depending on the application of such systems and/or the architecture in which such systems are implemented. Passing the write conflict check can include the log sequence number being greater than the global log sequence number. The means for performing the write conflict check, in response to passing the write conflict check, can update the global log sequence number to equal the log sequence number. The system means for performing the write conflict check, in response to passing the write conflict check, can execute an insertion of the write ahead log record into a group flush write ahead log buffer, and execute a save operation of all write ahead log records in the group flush write ahead log buffer to a persistent log in the common log. Such a system or similar systems can include the means for performing the write conflict check structured to execute replication of the write ahead log record to one or more follower common logs, the one or more follower common logs structured as back-ups to the common log.

[0104] The means for performing the write conflict check can extract the write ahead log record from a batch of write ahead log records received from the database engine having the log sequence number for one or more transactions between the database engine and the data storage. The means for performing the write conflict check can extract another write ahead log record from another batch of write ahead log records received from another database engine of the multiple database engines having another log sequence number for one or more other transactions between the other database engine and the data storage. The means for performing the write conflict check can maintain in the command log all operations and commands that modify an internal state of the common log.

[0105] A system or similar systems, having a means for performing a write conflict check on a write ahead log record, can include the multiple database engines, the data storage shared among the multiple database engines, and one or more follower common logs in addition to the means for performing the write conflict check and the means for sending the write ahead log record to the shared data storage. Such system or similar systems can be structured according to any permutation of the features taught herein for write-write conflict detection for multi-master shared storage database.

[0106] Conflict prevention based on global locking incurs significant network communications and blocking, which yields low performance and throughput. The methods and structures as taught herein uses WAL records to determine the existence of conflicts. This approach eliminates global locking and brings the benefit of optimistic concurrency control to multi-master shared storage database systems. Such an approach can be a key technology to build large scale on-cloud multi-master systems.

[0107] Although the present disclosure has been described with reference to specific features and embodiments thereof, it is evident that various modifications and combinations can be made thereto without departing from scope of the disclosure. The specification and drawings are, accordingly, to be regarded simply as an illustration of the disclosure as defined by the appended claims, and are contemplated to cover any and all modifications, variations, combinations or equivalents that fall within the scope of the present disclosure.

Claims

What is claimed is:

1. A computer-implemented method of writing to data storage shared among multiple database engines, the computer-implemented method comprising:

performing, in a common log using one or more processors, a write-write conflict check on a write ahead log record received from a database engine of the multiple database engines, in which the write-write conflict check includes conducting a comparison of a log sequence number, received with the write ahead log record from the database engine, with a global log sequence number in a hash table in the common log; and

sending the write ahead log record to the data storage shared among multiple database engines, in response to the write ahead log record passing the write-write conflict check.

2. The computer-implemented method of claim 1, wherein conducting the comparison comprises using a tuple identification or a page identification as a key in the hash table, the key associated with an entry in a form of a master identification and a global log sequence number value, the master identification being an identification of a database engine of the multiple database engines.

3. The computer-implemented method of claim 1 or claim 2, wherein passing the write-write conflict check includes the log sequence number being greater than the global log sequence number.

4. The computer-implemented method of any of claims 1-3, comprising, in response to passing the write-write conflict check, updating the global log sequence number to equal the log sequence number.

5. The computer-implemented method of any of claims 1-4, comprising, in response to passing the write-write conflict check:

inserting the write ahead log record into a group flush write ahead log buffer; and

saving all write ahead log records in the group flush write ahead log buffer to a persistent log in the common log.

6. The computer-implemented method of any of claims 1-5, comprising replicating the write ahead log record to one or more follower common logs, the one or more follower common logs structured as back-ups to the common log.

7. The computer-implemented method of any of claims 1-6, wherein the write ahead log record received from a database engine is extracted from a batch of write ahead log records received from the database engine having the log sequence number for one or more transactions between the database engine and the data storage.

8. The computer-implemented method of any of claims 1-7, comprising extracting another write ahead log record from another batch of write ahead log records received from another database engine of the multiple database engines having another log sequence number for one or more other transactions between the other database engine and the data storage.

9. The computer-implemented method of any of claims 1-8, comprising maintaining in a command log all operations and commands that modify an internal state of the common log.

10. A non-transitory computer-readable medium storing computer instructions, that when executed by one or more processors, cause the one or more processors to perform the steps of any of claims 1-9.

11. A system comprising:

a memory storage comprising instructions; and

one or more processors in communication with the memory storage, wherein the one or more processors execute the instructions to:

perform, in a common log, a write-write conflict check on a write ahead log record received from a database engine of multiple database engines, in which the write-write conflict check includes a comparison of

a log sequence number, received with the write ahead log record from the database engine, with a global log sequence number in a hash table in the common log; and

send the write ahead log record to a data storage shared among the multiple database engines, in response to the write ahead log record passing the write-write conflict check.

12. The system of claim 11, wherein the comparison comprises using a tuple identification or a page identification as a key in the hash table, the key associated with an entry in a form of a master identification and a global log sequence number value, the master identification being an identification of a database engine of the multiple database engines.

13. The system of claim 11 or claim 12, wherein passing the write-write conflict check includes the log sequence number being greater than the global log sequence number.

14. The system of any of claims 11-13, wherein the one or more processors, in response to passing the write-write conflict check, update the global log sequence number to equal the log sequence number.

15. The system of any of claims 11-14, wherein the one or more processors, in response to passing the write-write conflict check, execute:

an insertion of the write ahead log record into a group flush write ahead log buffer; and

a save operation of all write ahead log records in the group flush write ahead log buffer to a persistent log in the common log.

16. The system of any of claims 11-15, wherein the one or more processors execute replication of the write ahead log record to one or more follower common logs, the one or more follower common logs structured as back-ups to the common log.

17. The system of any of claims 11-16, wherein the one or more processors

extract the write ahead log record from a batch of write ahead log records received from the database engine having the log sequence number for one or more transactions between the database engine and the data storage.

18. The system of any of claims 11-17, wherein the one or more processors extract another write ahead log record from another batch of write ahead log records received from another database engine of the multiple database engines having another log sequence number for one or more other transactions between the other database engine and the data storage.

19. The system of any of claims 11-18, wherein the one or more processors maintain in a command log all operations and commands that modify an internal state of the common log.

20. The system of any of claims 11-19, wherein the system includes the multiple database engines, the data storage shared among the multiple database engines, and one or more follower common logs in addition to the common log.

21. A system comprising:

a means for performing a write conflict check on a write ahead log record received from a database engine of multiple database engines, in which the write conflict check includes a comparison of a log sequence number, received with the write ahead log record from the database engine, with a global log sequence number in a hash table in a common log, the means for performing the write conflict check including the common log and operationally arranged between the multiple database engines and a shared data storage, the shared data storage being shared among the multiple database engines; and

a means for sending the write ahead log record to the shared data storage, in response to the write ahead log record passing the write conflict check.

22. The system of claim 21, wherein the comparison comprises using a tuple identification or a page identification as a key in the hash table, the key associated with an entry in a form of a master identification and a global log sequence number value, the master identification being an identification of a

database engine of the multiple database engines.

23. The system of claim 21 or claim 22, wherein passing the write conflict check includes the log sequence number being greater than the global log sequence number.

24. The system of any of claims 21-23, wherein the means for performing the write conflict check, in response to passing the write conflict check, updates the global log sequence number to equal the log sequence number.

25. The system of any of claims 21-24, wherein the means for performing the write conflict check, in response to passing the write conflict check, executes:

an insertion of the write ahead log record into a group flush write ahead log buffer; and

a save operation of all write ahead log records in the group flush write ahead log buffer to a persistent log in the common log.

26. The system of any of claims 21-25, the means for performing the write conflict check executes replication of the write ahead log record to one or more follower common logs, the one or more follower common logs structured as back-ups to the common log.

27. The system of any of claims 21-26, wherein the means for performing the write conflict check extracts the write ahead log record from a batch of write ahead log records received from the database engine having the log sequence number for one or more transactions between the database engine and the data storage.

28. The system of any of claims 21-27, wherein the means for performing the write conflict check extracts another write ahead log record from another batch of write ahead log records received from another database engine of the multiple database engines having another log sequence number for one or more other transactions between the other database engine and the data storage.

29. The system of any of claims 21-28, wherein the means for performing the write conflict check maintains in a command log all operations and commands that modify an internal state of the common log.

30. The system of any of claims 21-29, wherein the system includes the multiple database engines, the data storage shared among the multiple database engines, and one or more follower common logs in addition to the means for performing the write conflict check and the means for sending the write ahead log record to the shared data storage.

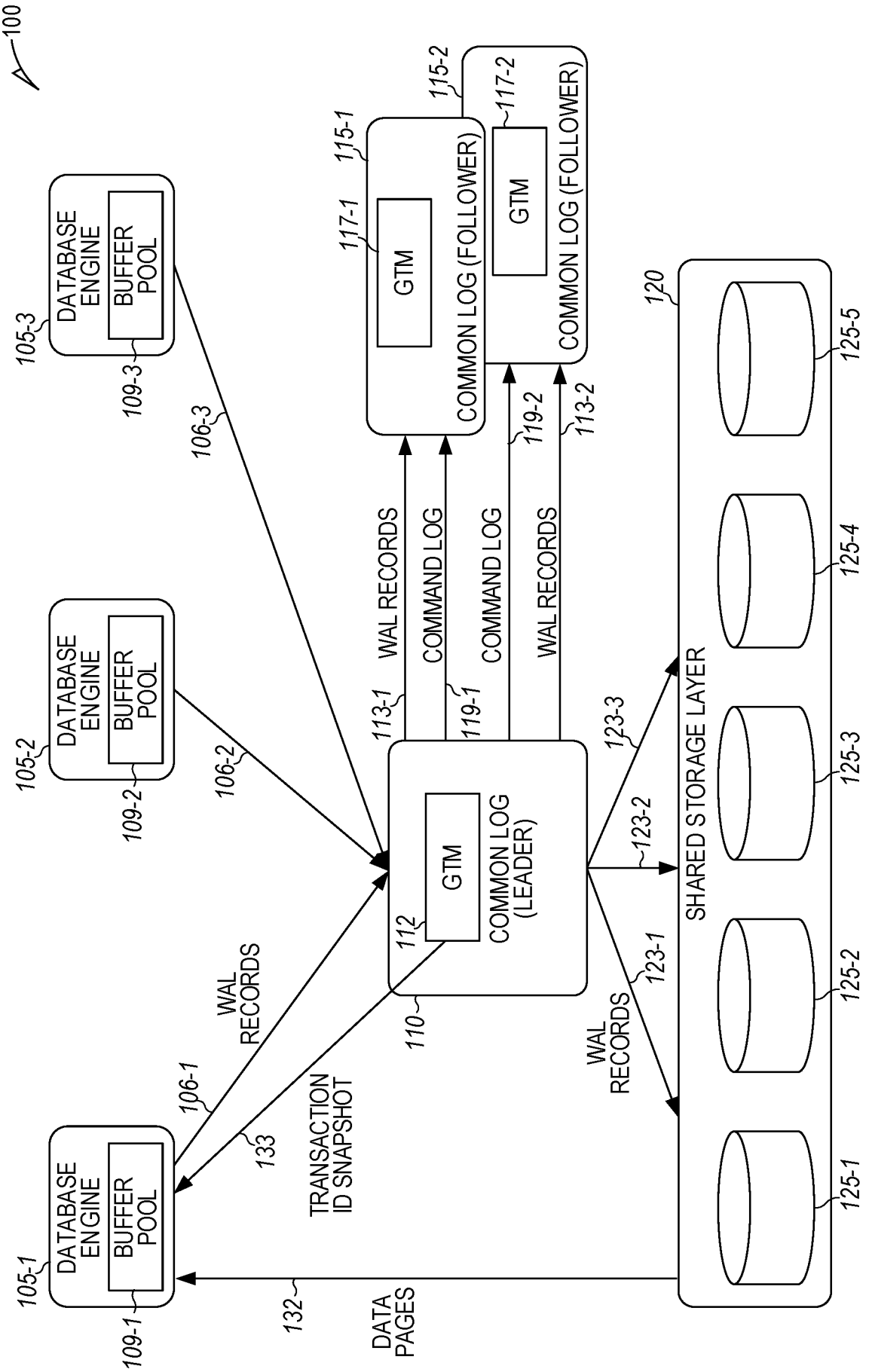
100
DATABASE ENGINE
BUFFER POOL
105-1
109-1
105-2
109-2
105-3
109-3
106-1
106-2
106-3
WAL RECORDS
TRANSACTION ID SNAPSHOT
133
132
DATA PAGES
COMMON LOG (LEADER)
GTM
112
110
WAL RECORDS
COMMAND LOG
113-1
119-1
COMMON LOG (FOLLOWER)
115-1
117-1
115-2
117-2
WAL RECORDS
COMMAND LOG
119-2
113-2
123-1
123-2
123-3
SHARED STORAGE LAYER
120
125-1
125-2
125-3
125-4
125-5

FIG. 1

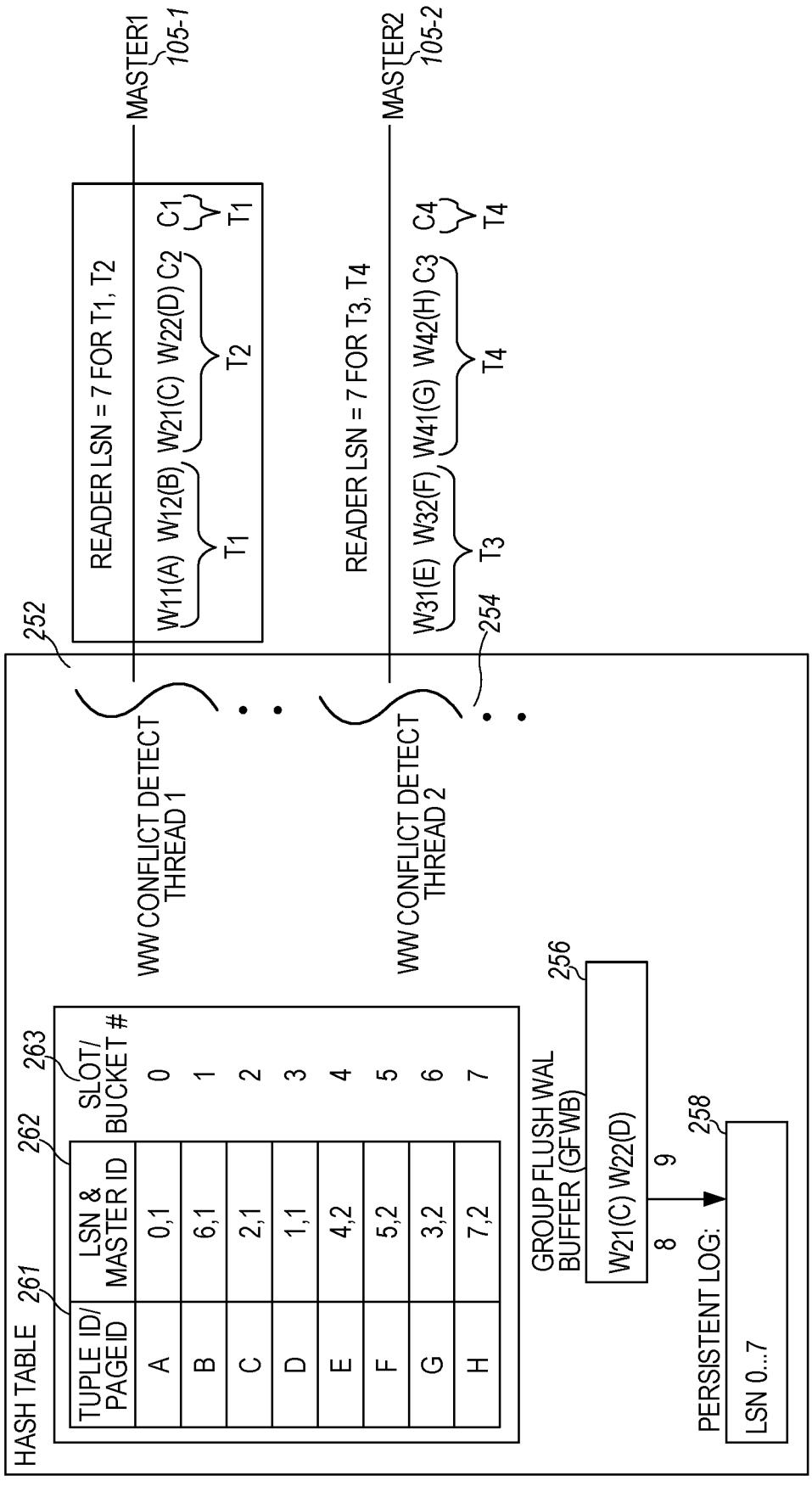
MASTER1
105-1
MASTER2
105-2
READER LSN = 7 FOR T1, T2
W11(A) W12(B) W21(C) W22(D) C2 C1
T1 T2 T1
READER LSN = 7 FOR T3, T4
W31(E) W32(F) W41(G) W42(H) C3 C4
T3 T4 T4
252
254
WW CONFLICT DETECT THREAD 1
WW CONFLICT DETECT THREAD 2
HASH TABLE 261
262
263
TUPLE ID/ PAGE ID
LSN & MASTER ID
SLOT/ BUCKET #
A 0,1 0
B 6,1 1
C 2,1 2
D 1,1 3
E 4,2 4
F 5,2 5
G 3,2 6
H 7,2 7
GROUP FLUSH WAL BUFFER (GFWB)
256
W21(C) W22(D)
8 9
PERSISTENT LOG:
258
LSN 0...7

FIG. 2

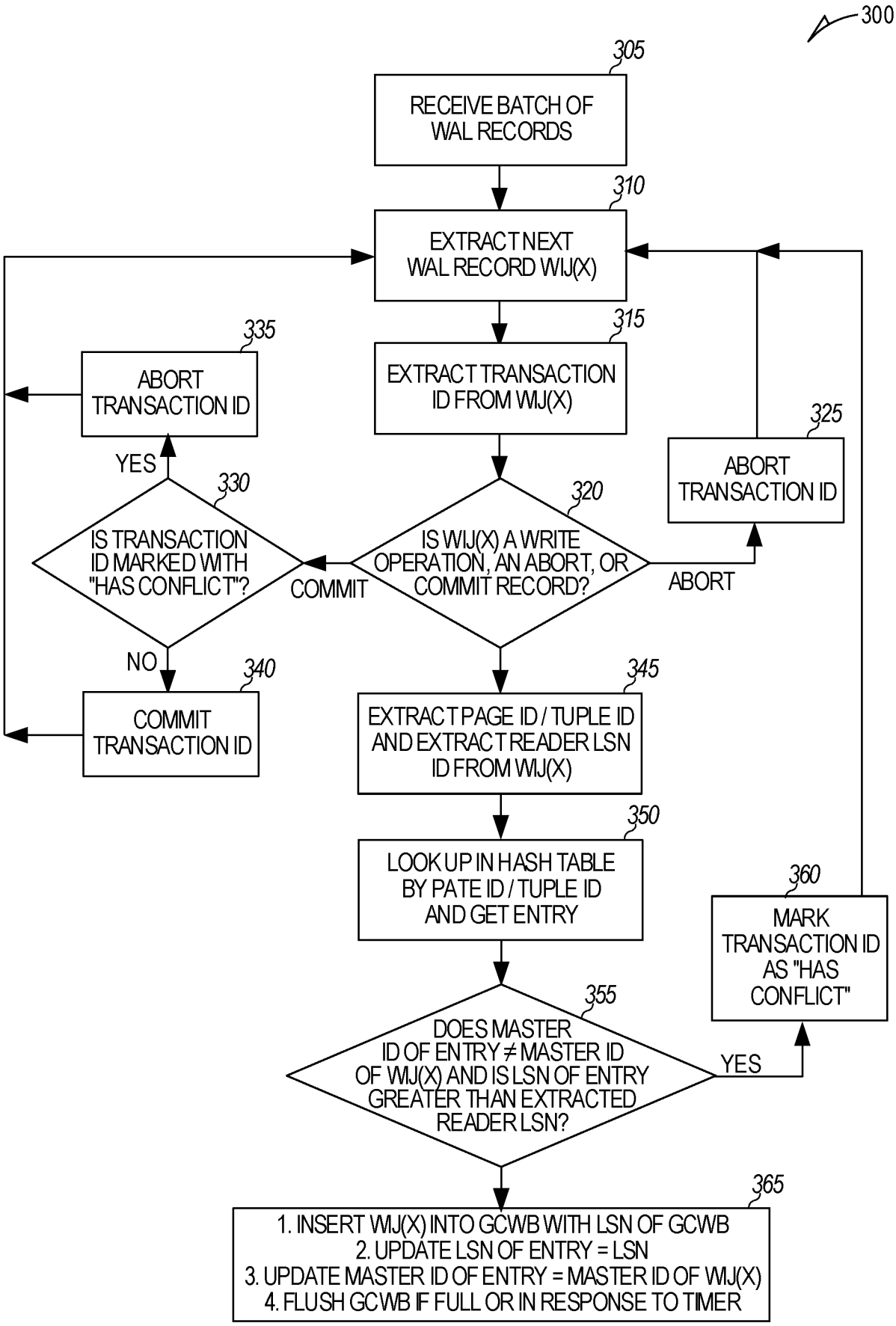
300
305
RECEIVE BATCH OF WAL RECORDS
310
EXTRACT NEXT WAL RECORD WIJ(X)
315
EXTRACT TRANSACTION ID FROM WIJ(X)
320
IS WIJ(X) A WRITE OPERATION, AN ABORT, OR COMMIT RECORD?
COMMIT
ABORT
325
ABORT TRANSACTION ID
330
IS TRANSACTION ID MARKED WITH "HAS CONFLICT"?
YES
NO
335
ABORT TRANSACTION ID
340
COMMIT TRANSACTION ID
345
EXTRACT PAGE ID / TUPLE ID AND EXTRACT READER LSN ID FROM WIJ(X)
350
LOOK UP IN HASH TABLE BY PATE ID / TUPLE ID AND GET ENTRY
355
DOES MASTER ID OF ENTRY ≠ MASTER ID OF WIJ(X) AND IS LSN OF ENTRY GREATER THAN EXTRACTED READER LSN?
YES
360
MARK TRANSACTION ID AS "HAS CONFLICT"
365
1. INSERT WIJ(X) INTO GCWB WITH LSN OF GCWB
2. UPDATE LSN OF ENTRY = LSN
3. UPDATE MASTER ID OF ENTRY = MASTER ID OF WIJ(X)
4. FLUSH GCWB IF FULL OR IN RESPONSE TO TIMER

FIG. 3

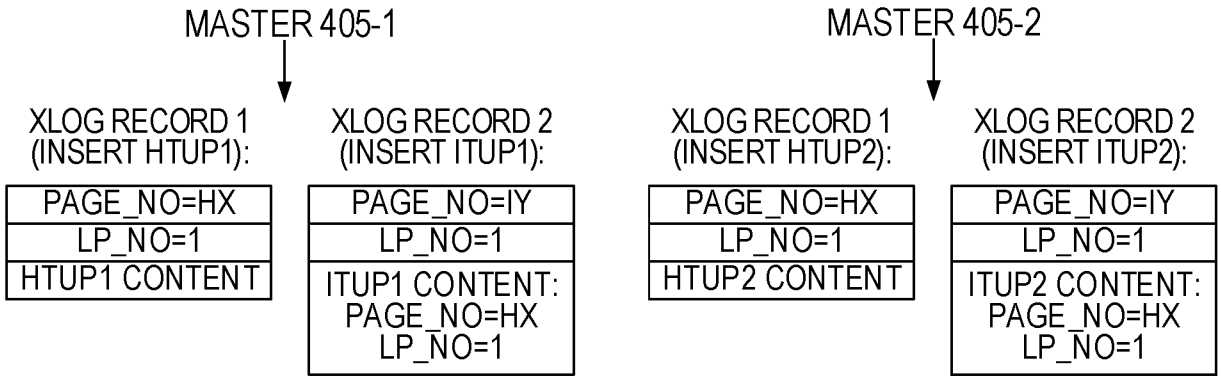
MASTER 405-1
XLOG RECORD 1 (INSERT HTUP1):
PAGE_NO=HX
LP_NO=1
HTUP1 CONTENT
XLOG RECORD 2 (INSERT ITUP1):
PAGE_NO=IY
LP_NO=1
ITUP1 CONTENT: PAGE_NO=HX LP_NO=1
MASTER 405-2
XLOG RECORD 1 (INSERT HTUP2):
PAGE_NO=HX
LP_NO=1
HTUP2 CONTENT
XLOG RECORD 2 (INSERT ITUP2):
PAGE_NO=IY
LP_NO=1
ITUP2 CONTENT: PAGE_NO=HX LP_NO=1

FIG. 4

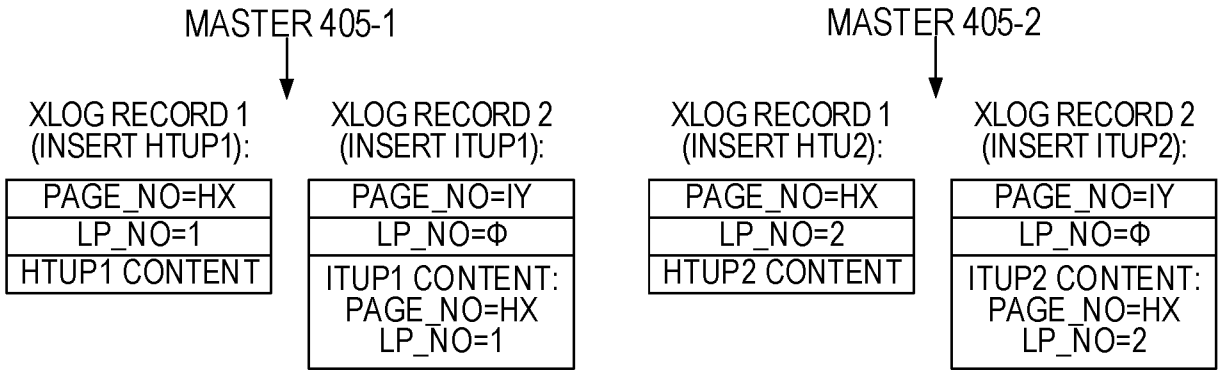
MASTER 405-1
XLOG RECORD 1 (INSERT HTUP1):
PAGE_NO=HX
LP_NO=1
HTUP1 CONTENT
XLOG RECORD 2 (INSERT ITUP1):
PAGE_NO=IY
LP_NO=Φ
ITUP1 CONTENT: PAGE_NO=HX LP_NO=1
MASTER 405-2
XLOG RECORD 1 (INSERT HTU2):
PAGE_NO=HX
LP_NO=2
HTUP2 CONTENT
XLOG RECORD 2 (INSERT ITUP2):
PAGE_NO=IY
LP_NO=Φ
ITUP2 CONTENT: PAGE_NO=HX LP_NO=2

FIG. 5

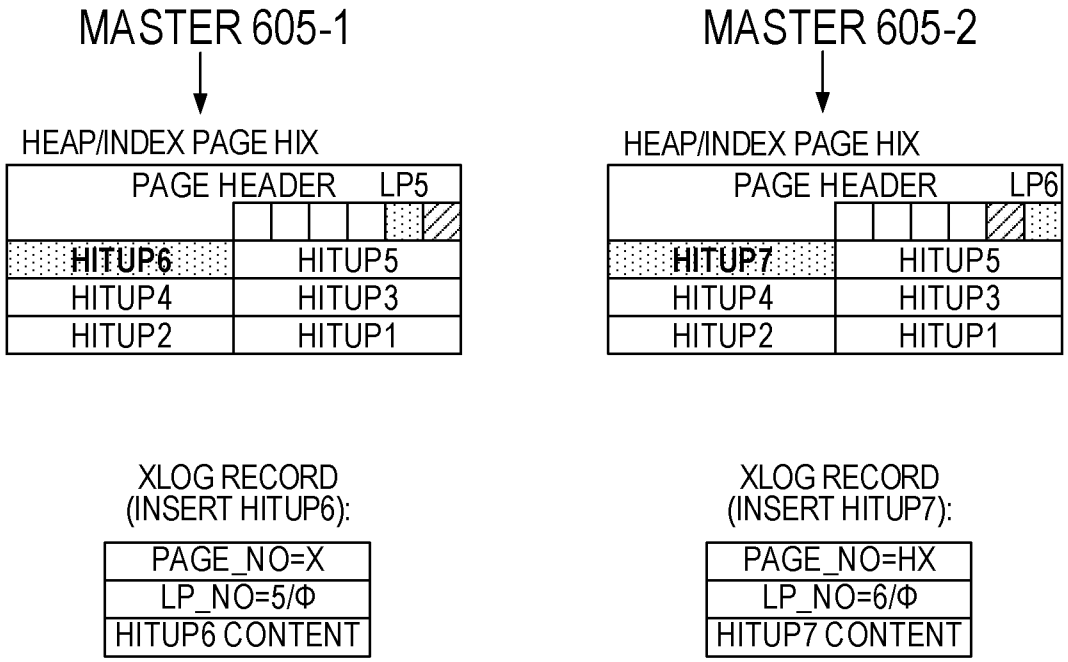
MASTER 605-1
HEAP/INDEX PAGE HIX
PAGE HEADER
LP5
HITUP6
HITUP5
HITUP4
HITUP3
HITUP2
HITUP1
MASTER 605-2
HEAP/INDEX PAGE HIX
PAGE HEADER
LP6
HITUP7
HITUP5
HITUP4
HITUP3
HITUP2
HITUP1
XLOG RECORD
(INSERT HITUP6):
PAGE_NO=X
LP_NO=5/Φ
HITUP6 CONTENT
XLOG RECORD
(INSERT HITUP7):
PAGE_NO=HX
LP_NO=6/Φ
HITUP7 CONTENT

FIG. 6

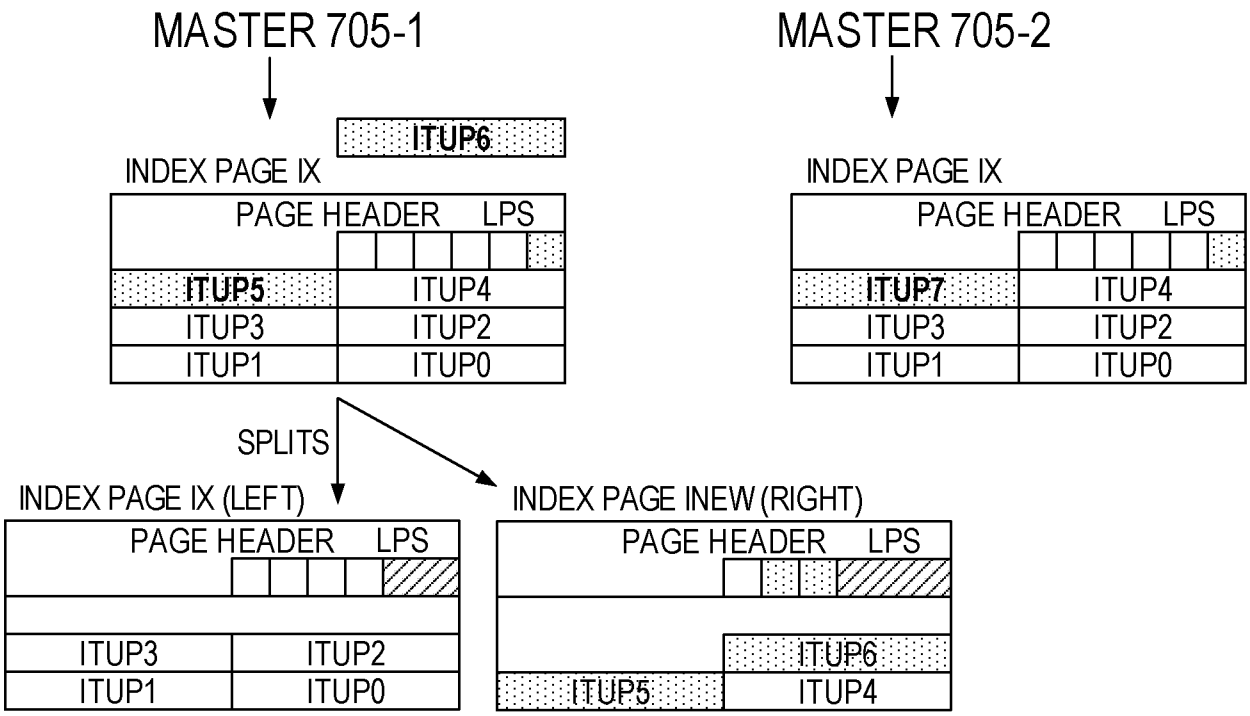
MASTER 705-1
ITUP6
INDEX PAGE IX
PAGE HEADER
LPS
ITUP5
ITUP4
ITUP3
ITUP2
ITUP1
ITUP0
MASTER 705-2
INDEX PAGE IX
PAGE HEADER
LPS
ITUP7
ITUP4
ITUP3
ITUP2
ITUP1
ITUP0
SPLITS
INDEX PAGE IX (LEFT)
PAGE HEADER
LPS
ITUP3
ITUP2
ITUP1
ITUP0
INDEX PAGE INEW (RIGHT)
PAGE HEADER
LPS
ITUP6
ITUP5
ITUP4

FIG. 7

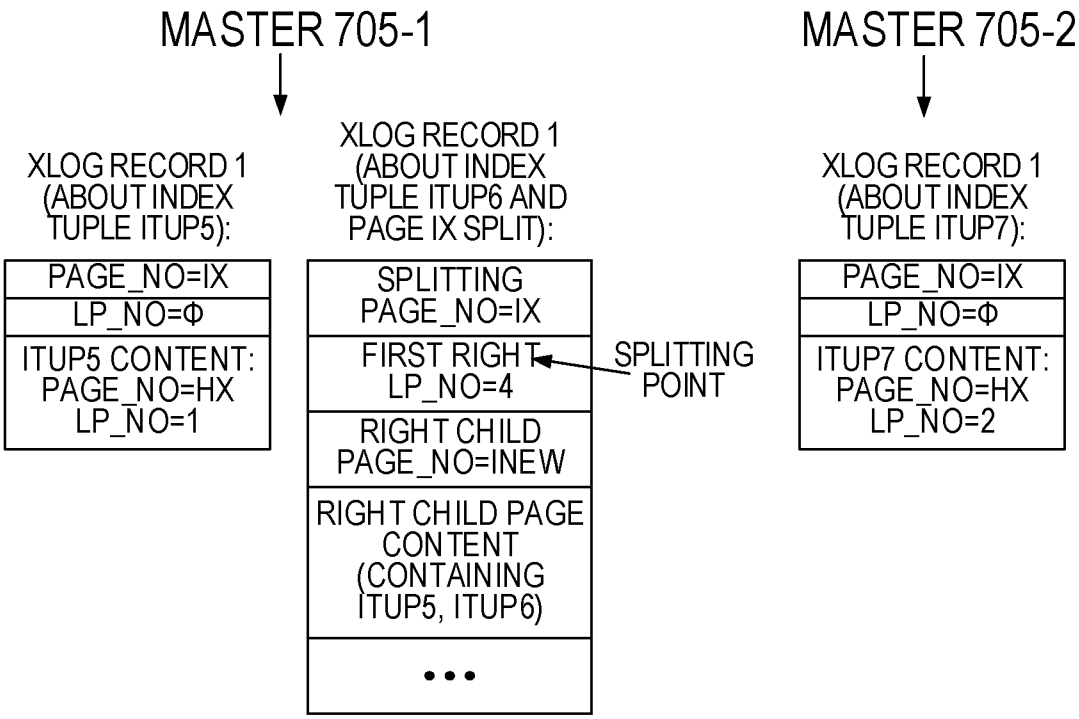
MASTER 705-1
MASTER 705-2
XLOG RECORD 1 (ABOUT INDEX TUPLE ITUP5):
XLOG RECORD 1 (ABOUT INDEX TUPLE ITUP6 AND PAGE IX SPLIT):
XLOG RECORD 1 (ABOUT INDEX TUPLE ITUP7):
PAGE_NO=IX
LP_NO=Φ
ITUP5 CONTENT: PAGE_NO=HX LP_NO=1
SPLITTING PAGE_NO=IX
FIRST RIGHT LP_NO=4
SPLITTING POINT
RIGHT CHILD PAGE_NO=INEW
RIGHT CHILD PAGE CONTENT (CONTAINING ITUP5, ITUP6)
• • •
PAGE_NO=IX
LP_NO=Φ
ITUP7 CONTENT: PAGE_NO=HX LP_NO=2

FIG. 8

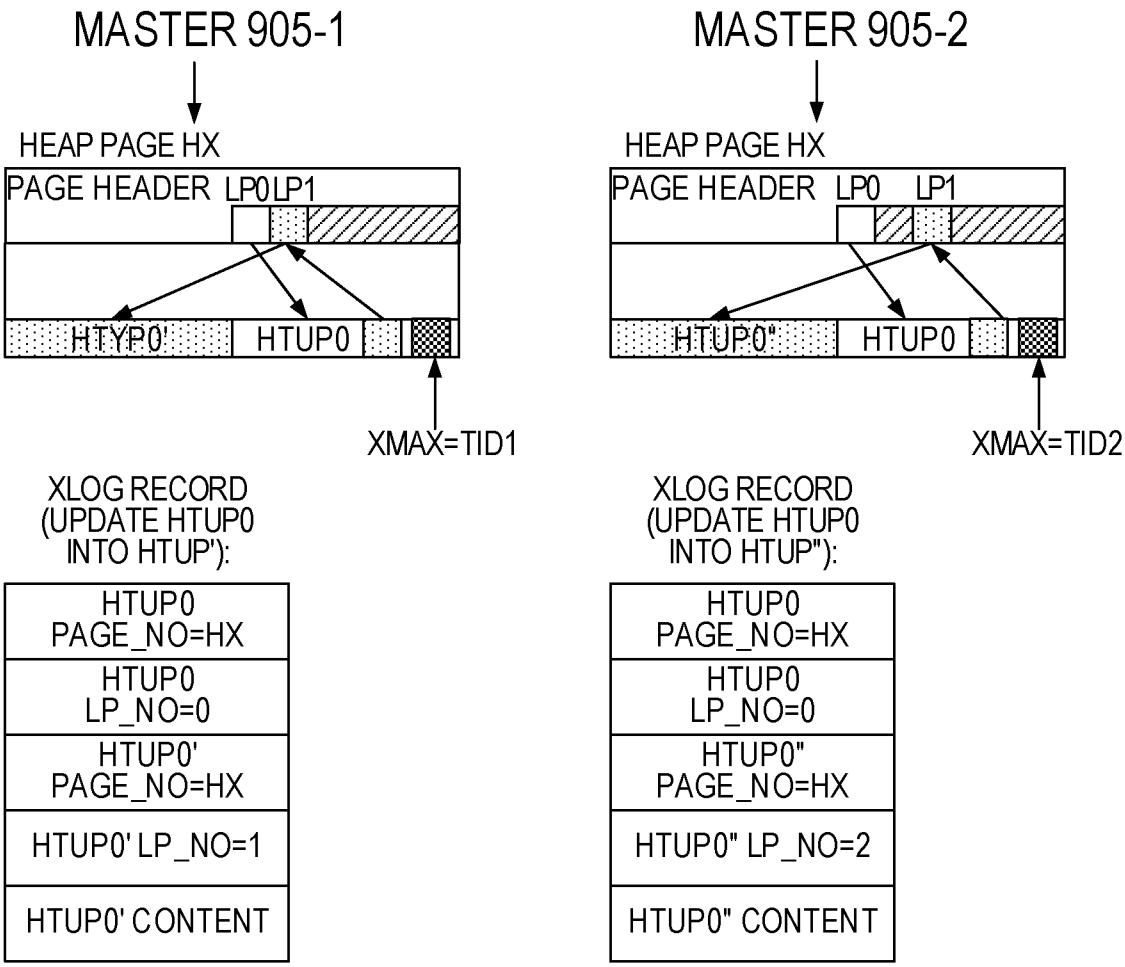
MASTER 905-1
HEAP PAGE HX
PAGE HEADER LP0 LP1
HTYP0'
HTUP0
XMAX=TID1
XLOG RECORD (UPDATE HTUP0 INTO HTUP'):
HTUP0 PAGE_NO=HX
HTUP0 LP_NO=0
HTUP0' PAGE_NO=HX
HTUP0' LP_NO=1
HTUP0' CONTENT
MASTER 905-2
HEAP PAGE HX
PAGE HEADER LP0 LP1
HTUP0"
HTUP0
XMAX=TID2
XLOG RECORD (UPDATE HTUP0 INTO HTUP"):
HTUP0 PAGE_NO=HX
HTUP0 LP_NO=0
HTUP0" PAGE_NO=HX
HTUP0" LP_NO=2
HTUP0" CONTENT

FIG. 9

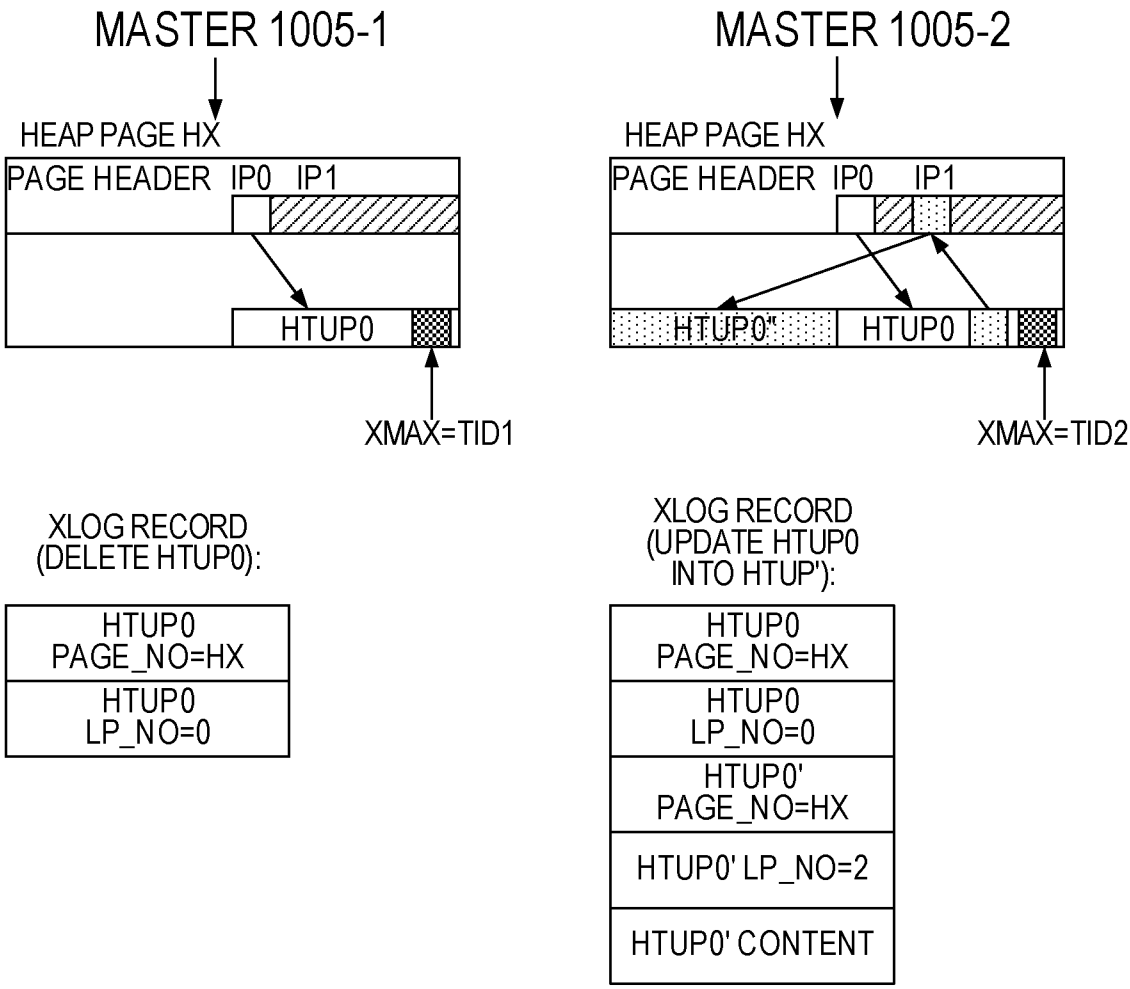
MASTER 1005-1
HEAP PAGE HX
PAGE HEADER IP0 IP1
HTUP0
XMAX=TID1
MASTER 1005-2
HEAP PAGE HX
PAGE HEADER IP0 IP1
HTUP0"
HTUP0
XMAX=TID2
XLOG RECORD
(DELETE HTUP0):
HTUP0
PAGE_NO=HX
HTUP0
LP_NO=0
XLOG RECORD
(UPDATE HTUP0
INTO HTUP'):
HTUP0
PAGE_NO=HX
HTUP0
LP_NO=0
HTUP0'
PAGE_NO=HX
HTUP0' LP_NO=2
HTUP0' CONTENT

FIG. 10

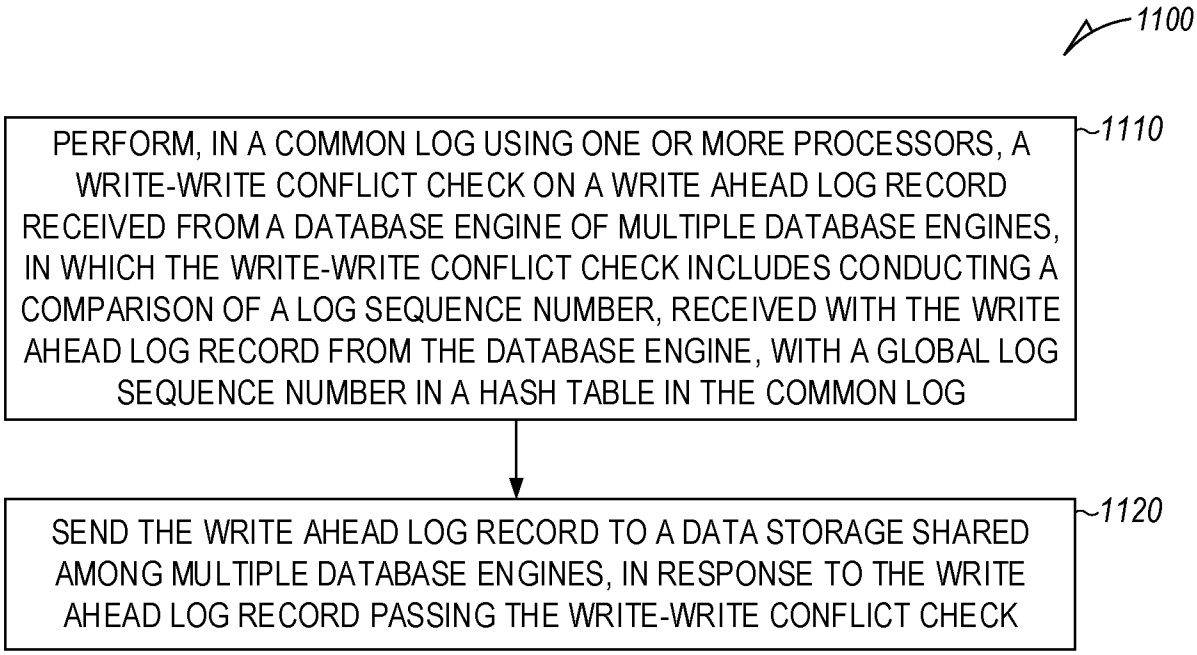
1100
PERFORM, IN A COMMON LOG USING ONE OR MORE PROCESSORS, A WRITE-WRITE CONFLICT CHECK ON A WRITE AHEAD LOG RECORD RECEIVED FROM A DATABASE ENGINE OF MULTIPLE DATABASE ENGINES, IN WHICH THE WRITE-WRITE CONFLICT CHECK INCLUDES CONDUCTING A COMPARISON OF A LOG SEQUENCE NUMBER, RECEIVED WITH THE WRITE AHEAD LOG RECORD FROM THE DATABASE ENGINE, WITH A GLOBAL LOG SEQUENCE NUMBER IN A HASH TABLE IN THE COMMON LOG
1110
SEND THE WRITE AHEAD LOG RECORD TO A DATA STORAGE SHARED AMONG MULTIPLE DATABASE ENGINES, IN RESPONSE TO THE WRITE AHEAD LOG RECORD PASSING THE WRITE-WRITE CONFLICT CHECK
1120

FIG. 11

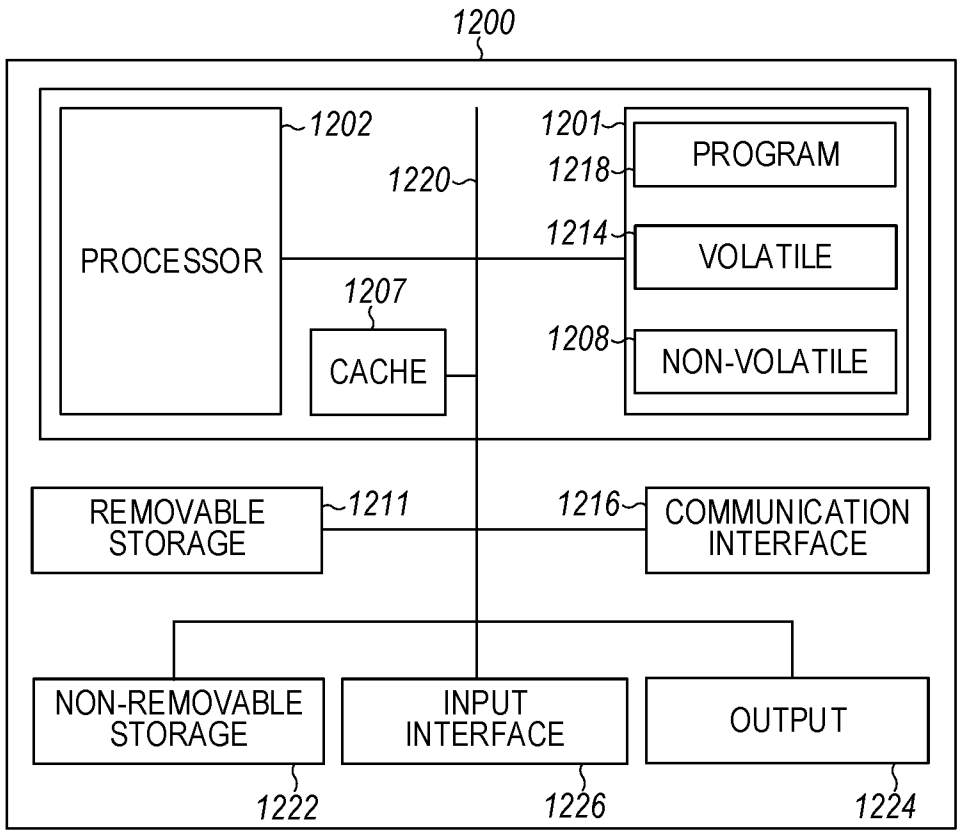
1200
PROCESSOR
1202
1220
1201
PROGRAM
1218
1214
VOLATILE
1207
CACHE
1208
NON-VOLATILE
REMOVABLE STORAGE
1211
1216
COMMUNICATION INTERFACE
NON-REMOVABLE STORAGE
INPUT INTERFACE
OUTPUT
1222
1226
1224

FIG. 12

INTERNATIONAL SEARCH REPORT

International application No.
PCT/CN2019/091397

A. CLASSIFICATION OF SUBJECT MATTER

G06F 12/00(2006.01)i

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

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F H04L

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)

CNPAT, WPI, EPODOC, CNKI, GOOGLE: write, conflict, detect, check, determine, sequence, order, number, identification, ID, log, common, global, local

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 7475201 B1 (TEPLIN APPLICATION LIMITED LIABILITY CO.) 06 January 2009 (2009-01-06) description, column 3 line 20 to column 7 line 35	1-30
A	US 2016267009 A1 (MARGULIS, OLEG ET AL.) 15 September 2016 (2016-09-15) the whole document	1-30
A	US 2018322158 A1 (HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP) 08 November 2018 (2018-11-08) the whole document	1-30
A	EP 0716382 A1 (XEROX CORPORATION) 12 June 1996 (1996-06-12) the whole document	1-30
A	US 2014040554 A1 (POHLACK MARTIN T. ET AL.) 06 February 2014 (2014-02-06) the whole document	1-30
A	US 2011179230 A1 (SUN MICROSYSTEMS, INC.) 21 July 2011 (2011-07-21) the whole document	1-30
A	US 2015378777 A1 (INTERNATIONAL BUSINESS MACHINES CORPORATION) 31 December 2015 (2015-12-31) the whole document	1-30

☑ 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
02 August 2019	**03 September 2019**

Name and mailing address of the ISA/CN	Authorized officer
National Intellectual Property Administration, PRC **6, Xitucheng Rd., Jimen Bridge, Haidian District, Beijing 100088** **China**	**CHENG,Jiali**
Facsimile No. **(86-10)62019451**	Telephone No. **86-10-53961625**

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	WO 2016086342 A1 (HUAWEI TECHCONOLOGY CO., LTD.) 09 June 2016 (2016-06-09) the whole document	1-30
A	CN 105045563 A (SHANNXI UNIVERSITY OF SCIENCE AND TECHNOLOGY) 11 November 2015 (2015-11-11) the whole document	1-30
A	CN 107148617 A (AMAZON TECHNOLOGIES INC.) 08 September 2017 (2017-09-08) the whole document	1-30

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/CN2019/091397

Patent document cited in search report	Publication date (day/month/year)	Patent family member(s)	Publication date (day/month/year)
US 7475201 B1	06 January 2009	US 2009193216 A1	30 July 2009
US 2016267009 A1	15 September 2016	None	
US 2018322158 A1	08 November 2018	None	
EP 0716382 A1	12 June 1996	US 5581754 A	03 December 1996
		JP H08241234 A	17 September 1996
		DE 69528338 D1	31 October 2002
US 2014040554 A1	06 February 2014	None	
US 2011179230 A1	21 July 2011	None	
US 2015378777 A1	31 December 2015	US 2015378631 A1	31 December 2015
		US 2017010929 A1	12 January 2017
WO 2016086342 A1	09 June 2016	CN 105849688 A	10 August 2016
CN 105045563 A	11 November 2015	None	
CN 107148617 A	08 September 2017	EP 3195117 A1	26 July 2017
		WO 2016044763 A1	24 March 2016