



- (51) **International Patent Classification:**
G06F 17/30 (2006.01) *G06F 9/44* (2006.01)
G06F 9/38 (2006.01)
- (21) **International Application Number:**
PCT/US2014/020937
- (22) **International Filing Date:**
5 March 2014 (05.03.2014)
- (25) **Filing Language:** English
- (26) **Publication Language:** English
- (71) **Applicant:** HEWLETT-PACKARD DEVELOPMENT COMPANY, L.P. [US/US]; 11445 Compaq Center Drive W., Houston, Texas 77070 (US).
- (72) **Inventors:** MORREY III, Charles B; 1501 Page Mill Rd., Palo Alto, California 94304-1100 (US). KEETON, Kimberly; 1501 Page Mill Rd., Palo Alto, California 94304-1100 (US). VEITCH, Alistair; 2420 Whitney Drive, Mountain View, California 94043 (US).
- (74) **Agents:** ORTEGA, Arthur et al.; Hewlett-Packard Company, Intellectual Property Administration, 3404 E. Harmony Road, Mail Stop 35, Fort Collins, Colorado 80528 (US).
- (81) **Designated States** (*unless otherwise indicated, for every kind of national protection available*): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, 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, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK,

[Continued on next page]

(54) **Title:** PROCESSING PRIMARY KEY MODIFICATIONS

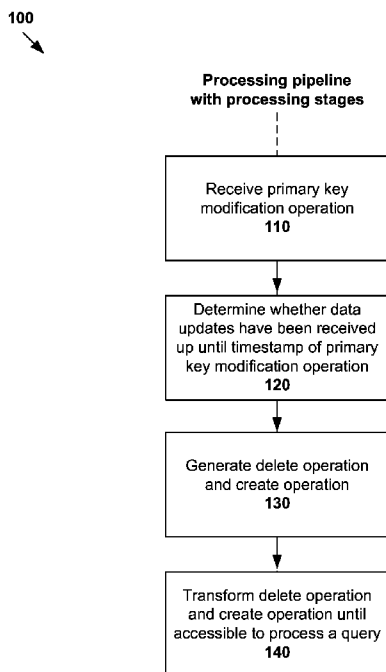


Fig. 1

(57) **Abstract:** A method for processing a primary key modification using a processing pipeline is provided. The processing pipeline includes a first processing stage and a second processing stage to process data updates from a plurality of update sources. The first processing stage receives a primary key modification operation, the primary key modification operation being from one of the plurality of update sources to modify a source primary key to a destination primary key. The primary key modification operation is associated with a timestamp. In response to determination that data updates from all of the plurality of update sources up until the timestamp of the primary key modification operation have been received by the processing pipeline, the second processing stage applies the primary key modification operation by generating a delete operation to delete the source primary key and a create operation to create the destination primary key. The second processing stage also transforms the delete operation and create operation into an output data structure that is accessible to process a query.



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).

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

Published:

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

Declarations under Rule 4.17:

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

Processing Primary Key Modifications

Background

[0001] Data mining, analysis, and search often make up a substantial portion of enterprise application workloads. Examples of data that are the subject of data mining, analysis, and search include purchase transactions, news updates, web search results, email notifications, hardware or software monitoring observations, and so forth.

Brief Description of Drawings

[0002] By way of non-limiting examples, the present disclosure will be described with reference to the following drawings, in which:

[0003] Fig. 1 is a flowchart of a process for processing a primary key modification using a processing pipeline with a plurality of processing stages, according to examples of the present disclosure;

[0004] Fig. 2 is a schematic diagram of a system for processing a primary key modification, according to examples of the present disclosure;

[0005] Fig. 3 is a schematic diagram of the system in Fig. 2, further illustrating data updates at various stages of a processing pipeline, according to examples of the present disclosure;

[0006] Fig. 4 is a schematic diagram illustrating arrival of a primary key modification operation and other data updates over time, according to examples of the present disclosure;

[0007] Fig. 5 is a schematic diagram of the system in Fig. 2, further illustrating processing of a primary key modification operation using a priority processing approach, according to examples of the present disclosure;

[0008] Fig. 6 is a flowchart of a detailed process for processing a primary key modification operation using a priority processing approach, according to examples of the present disclosure;

[0009] Fig. 7 is a schematic diagram of the system in Fig. 2, further illustrating processing of a primary key modification operation using an aggregation approach, according to examples of the present disclosure;

[0010] Fig. 8 is a flowchart of a detailed process for processing a primary key modification using an aggregation approach, according to examples of the present disclosure; and

[0011] Fig. 9 is a schematic diagram of a structure of a device capable of acting as a server system supporting a processing pipeline, according to examples of the present disclosure.

Detailed Description

[0012] To improve the ability to locate the content of various data stored across an organization, metadata associated with such data from many information sources may be uploaded to a server system (or multiple server systems) to allow users to submit queries against the server system(s) to locate data based on the metadata. Examples of metadata include metadata computed based on content of the data, including hashes (e.g., produced by applying hash functions on data), term vectors (e.g., containing terms in the data), fingerprints, feature vectors. Other examples of metadata include filesystem metadata, such as file owners or creators, file size and security attributes, or information associated with usage of the data, such as access frequency statistics.

[0013] The server system is designed to support data updates from multiple sources across the organization (e.g., up to hundreds of thousands or even

millions for a large organization). A “data update” may generally refer to a creation, modification, and/or deletion of data. Since there may be a relatively large amount of data updates to upload to the server system, it may take a relatively long period of time before the data updates are available for access by queries submitted to the server system.

[0014] To improve efficiency, the server system may implement a “processing pipeline” with multiple processing stages for performing respective data processing. After one processing stage has completed, the processing stage sends processed data updates to another processing stage for further processing. The processing stages are arranged to sequentially apply processing of data updates that pass through the processing pipeline, thereby allowing the process to be parallelized.

[0015] Fig. 1 is a flowchart of a process 100 for processing a primary key modification using a processing pipeline with a first processing stage and a second processing stage to process data updates from a plurality of update sources, according to examples of the present disclosure. The second device supports the following techniques.

[0016] At 110, the first processing stage receives a primary key modification operation. The primary key modification operation is from one of the plurality of update sources to modify a source primary key to a destination primary key. The primary key modification operation is associated with a timestamp.

[0017] At 120, it is determined whether data updates from all of the plurality of update sources up until the timestamp of the primary key modification operation have been received by the processing pipeline.

[0018] At 130, in response to an affirmative determination, the second processing stage applies the primary key modification operation by generating

a delete operation to delete the source primary key and a create operation to create the destination primary key.

[0019] At 140, the second processing stage transforms the delete operation and create operation into an output data structure that is accessible to process a query.

[0020] In general, it is difficult to support data updates relating to primary key modifications using a processing pipeline having multiple processing stages. For example, the processing pipeline may process data updates in batches, each batch being self-contained. However, primary key modifications, by their very nature, do not result in self-contained updates because the new primary key may affect all other data updates in the current batch and following batches. As such, simple read-modify-write operations are not supported by the processing pipeline because of the asynchronous processing nature of the processing stages.

[0021] According to examples of the present disclosure, a “complete prefix” of data updates is observed prior to applying the primary key modification operation. The complete prefix, which includes data updates up until the timestamp of the primary key modification operation, is observed to ensure correct processing of the data updates. Once the complete prefix has been observed, delete operation and create operation are generated for the primary key modification and uploaded to the processing pipeline. Data updates in the complete prefix may include data updates associated with the source primary key, even if they are still being processed by the processing pipeline.

[0022] Any suitable system with a processing pipeline and multiple update sources may implement process 100 for processing primary key modifications, such as a distributed file system, etc. For example in a distributed file system, there is a possibility of out-of-order data update observations, i.e. data updates having an earlier timestamp being observed at a later time than the primary key

modification operation. If the primary key modification operation is applied before the earlier data updates are observed, subsequent processing of the earlier data updates will be incorrect because the source primary key has been modified. Any suitable primary key modification may be performed using process 100 in Fig. 1, such as renaming of file name, and directory name, modification of a unique object ID in a catalogue of objects, modification a social security ID in a personnel file, etc.

[0023] Process 100 may also be implemented in other systems, such as a distributed object-based storage system, etc. In this case, the update sources may be nodes storing the data for the objects. In another example, process 100 may be implemented in sensor data collection environments, where the update sources are sensors that generate periodic readings.

[0024] Process 100 involves “inline processing” that may be implemented within stages of the processing pipeline. This approach does not require, for example, an external device to query the processing pipeline and determine whether the complete prefix of data updates have been received by the processing pipeline before the primary key modification operation is applied.

[0025] Distributed file system

[0026] Fig. 2 illustrates a distributed file system 200 in which primary key modifications may be processed, according to examples of the present disclosure. Distributed file system 200 may be implemented using any suitable file system.

[0027] Distributed file system 200 is generally scalable and includes multiple nodes 210-1 and 210-2 (collectively referred to as “nodes 210” or individually as a generic “node 210”, two shown for simplicity) that monitor file system events relating to data updates to be performed. Nodes 210 may include or otherwise access a journal 220 that records data updates as journal entries.

[0028] Each node 210 may support “segments” 212, which are logically contiguous pieces of storage built out of any combination of physical storage. A “file system” is generally a concatenation of segments 212 supported by nodes 210. For example, if there are 10 nodes 210 supporting two segments 212 each, there will be a total of 20 segments in the file system. Multiple nodes 210 may also form a “cluster” in the distributed file system 200. Individual nodes 210 may be logically or physically distinct. In distributed file system 200, node 210, or more specifically segment 212 of node 210, may be an “update source” (also known as “event source”) from which data updates (or events) originate.

[0029] Nodes 210 may include kernel level logic (not shown for simplicity) to detect kernel level events that correspond to intent to perform or initiation of kernel level operations. Examples of kernel level operations include delete, read and write, etc. Kernel level events generally identify parameters that are relevant to the corresponding operation, such as file name, timestamp, number of bytes read, user, etc.

[0030] Nodes 210 may further include user level logic (also not shown for simplicity) to detect user level events that correspond to the node initiating user level operations. User level events may also identify parameters relevant to corresponding operations, such as file name, user-defined tag, user name and timestamp, etc.

[0031] Journal entries in journal 220 are outputs from the kernel of node 210. In a storage system for archive data, a journal entry is part of an “archive journal file” or “AJ file”. Journal 220 may be implemented using any suitable journal, e.g. a Journaling Block Device (JBD) that acts as the journal for the Third Extended File System (Ext3), etc. Entries in the journal 220 may be metadata (instead of file content) representing a corresponding data update. Individual entries in the journal 220 may be marked “uncommitted” until

confirmation is received that the contents of the operations represented by the entries have been written to non-volatile storage (e.g., hard disk) within file system 200. Different types of data updates may be recorded in journal 220. If a system crash or power failure occurs, journal 220 may be used as a “checkpoint” to recover unsaved information.

[0032] Distributed file system 200 further includes upload entity 230 to retrieve data updates 232 recorded in journal entries 220 and upload them to processing pipeline 244. It should be noted that “upload entity” 230 may be a physical or logical entity that is on the same physical device as an update source, or separately on a different physical device. For example, upload entity 230 may be a journal scanning or aggregation component (also referred to as “Archive Journal (AJ) Scanner”) that transforms journal entries 220 into a format readable by server system 240. Upload entity 230 also orders data updates based on timestamps and aggregates them into a batch of updates.

[0033] A batch of data updates 232 may form a “self-consistent update (SCU)” that represents a single atomic unit of data updates. An SCU is not considered “durable” until all the individual updates in the SCU are written to stable (persistent) storage. Atomic application of data updates 232 of an SCU to stable storage means that all data updates 232 of the SCU are applied, or none is applied at all. Atomicity prevents partial data updates of the SCU, which can cause data consistency problems. Processing pipeline 244 supports primary key modification operations because it is capable of applying batches of data updates atomically, thereby continuously maintaining a consistent view of the data.

[0034] Each SCU may contain insert, update and delete operations. Upload entity 230 may form an SCU until either a sufficient amount of time has passed (e.g., based on a timeout threshold) or a sufficient amount of data has been collected (e.g., based on some predefined size).

[0035] Processing pipeline

[0036] In the example in Fig. 2, server system 240 includes coordinator 242 and processing pipeline 244 with multiple processing stages: ingest stage 248, sorting stage 250 and merging stage 252. In general, stages 248-252 are logically distinct and may be run on the same physical device or different physical devices. If run on different physical devices, server system 240 represents a logical system that supports the stages 248-252 of processing pipeline 244.

[0037] Coordinator 242 manages or controls different stages of processing pipeline 244. Coordinator 242 also schedules each SCU to be processed by the next stage. When each stage completes processing, it sends a message to coordinator 242. This allows coordinator 242 to track which upload entities 230 in the system hold which SCUs and what stages of processing they have completed.

[0038] Each stage 248, 250, 252 of processing pipeline 244 may be individually and independently scalable. Parallelism in each stage 248, 250, 252 may be enhanced by implementing the stage using more processors. Data updates 232 in SCUs from upload entities 230 are processed by processing pipeline 244 in stages as follows.

[0039] Ingest stage 248 durably stores incoming data updates 232 from upload entities 230 to generate at least one unsorted SCU. When stored “durably”, data updates of the unsorted SCU are not lost upon some error condition or power failure. Unsorted SCUs may be stored in an update data structure associated with ingest stage 248.

[0040] Sorting stage 250 is a transformation stage that sorts unsorted SCUs to generate sorted SCUs, such as according to a primary key. Sorted SCUs may be stored in an update data structure associated with sorting stage 250.

[0041] Merging stage 252 is a further transformation stage that combines multiple sorted SCUs into a merged SCU, which generally represents a single set of authority data structure 256 that is accessible to process a query 252 from a query client device 260.

[0042] By merging sorted SCUs together, query cost of retrieving fresher results may be reduced because there are fewer SCUs that must be examined by the query 262. Any suitable merging process may be used, such as a tree-based merging process, etc. In this case, SCUs are placed into a tree as leaf nodes and once a sufficient span of SCUs is available or a sufficient time has passed, they are merged together.

[0043] Throughout the present disclosure, the term “authority” 254 (used also known as “authority data structure”, “authority table”, “base table” and “output data structure”) generally refers to a data structure or set of data structures that is searched in response to data queries 262 from query client devices 260. For example, authority 254 may be organized as a table of named, typed columns, where the columns are described in the table schema and each row of the table provides a distinct instance. The authority 254 may represent the minimal amount of data that must be queried to retrieve a result. Data store 256 may store multiple authority tables 254.

[0044] Examples of authority data structure 254 may include the following:

[0045] Data structure to store metadata associated with File Lifetime (FL), such as create time and delete time associated with a file, etc.

[0046] Data structure to store metadata associated with Change Log (CL), such as last activity time and last activity reason, etc.

[0047] Data structure to store metadata associated with File Delete Time (FDT), such as timestamp of deletion, etc.

[0048] Data structure to store metadata associated with File Objects (FO), such as modification date, size and any custom metadata, etc. For example, a row in the data structure is created for each file in the distributed file system 200.

[0049] Server system 210 may also maintain update data structures at various stages 248, 250, 252 of processing pipeline 244. Similar to authority 254, an update data structure may be organized as a table of named, typed columns, where the columns are described in the table schema and each row of the table provides a distinct instance. An update data structure may have the same schema as the associated authority 254, as well as additional columns to indicate the type of operation and a timestamp. Data in the update data structure is to be applied to authority 254 after being processed through processing pipeline 244.

[0050] Processing pipeline 244 provides the ability to trade data freshness for query performance. "Data freshness" refers to how up-to-date data should be for a response to a query. In some applications, a query client 260 may want a relatively quick response 264 to a query 262, but it may be willing to accept data that is out-of-date (e.g., out-of-date by a certain time period, such as 12 hours, one day, etc.). On the other hand, another query client 260 may want an up-to-date data, but it may be willing to accept a slower response time to a query 262. Through the use of a pipelined architecture, different stages of processing pipeline 244 may be selectively accessed depending on user's requirements of freshness and/or query performance.

[0051] To process a query 262, server system 210 may selectively access authority data structure 254 or update data structure of any stage 248, 250, 252 of processing pipeline 244. The selection is a trade-off between data freshness and query performance. Accessing authority data structure 254 provides better query performance, but the response 264 retrieved may not be up-to-date

(since there may be various data updates being processed in the different stages of processing pipeline 244).

[0052] On the other hand, accessing stages 248, 250, 252 would increase the amount of time to process the query 242 but provides fresher (i.e. more up-to-date data). The amount of time depends upon which stage is accessed; accessing a later stage involves less query processing time compared with an earlier one. For example, accessing content of sorted and merged update tables provided by sorting stage 250 and merging stage 252 stages takes less time than accessing unsorted update tables maintained by ingest stage 248.

[0053] Although not shown in Fig. 2, processing pipeline 244 may further include an ID (identifier) remapping stage between ingest stage 248 and sorting stage 250 to transform initial IDs of SCUs into global IDs to generate remapped SCUs. In particular, an ID remapping stage maps an ID in a first space to an ID in a second space that is searchable. Initial IDs used by ingest stage 248 are assigned to each entity (e.g., file names, etc.) as those entities are processed and remapped to global ID space later to achieve enhanced pipeline performance when text fields like file system pathnames are extremely long. Substituting short numeric binary fields for long strings requiring lengthy string processing reduces overhead at some steps.

[0054] Fig. 3 is a schematic diagram of the system in Fig. 2, further illustrating data updates at various stages 248, 250, 252 of processing pipeline 244, according to examples of the present disclosure. In this example, Node 1 and Node 2 support a total of four segments (labelled Segment 1 to Segment 4) that form File System 1. Node 1 and Node 2 output journal entries that are stored in respective archive journal files in per-segment directories in journal 220, such as *AJFile_1* to *AJFile_3* in directory /S_1/ for segment 1, *AJFile_1* to *AJFile_5* in directory /S_2/ for segment 2, and so on.

[0055] Upload entity 230 (e.g. AJ Scanner) sorts journal entries 220 according to their timestamps and convert them from journal entry format to a format readable by server system 240. The converted journal entries are uploaded to ingest stage 248 of processing pipeline 244 in the form of SCUs 232.

[0056] Uploaded SCUs 232 are durably stored at ingest stage 248, before being transformed by sorting stage 250 and then merging stage 252 into authority. For example, at 310 in Fig. 3, ingest stage 248 durably stores *SCU_57* to *SCU_65*. At 320 in Fig. 3, sorting stage 250 sorts the SCUs into sorted SCUs according to at least one key, producing sorted *Table A* and *Table B* of *SCU_57*, etc. At 330 in Fig. 3, tables of sorted SCUs *SCU_57* to *SCU_65* are merged during merging stage 252 into *SCU_57_65/Table A*, *SCU_57_65/Table B* and *SCU_57_65/Table C*.

[0057] Complete prefix of data updates

[0058] Referring to Fig. 1 again, a primary key modification operation is applied after it is determined that data updates have been observed by processing pipeline 244 from all update sources (e.g., segments 212 of the file system 200) up until the timestamp of the primary key modification at blocks 120 and 130. As will be explained below, this is to ensure correct processing of data updates.

[0059] Fig. 4 is a schematic diagram illustrating arrival of a primary key modification operation and other data updates at processing pipeline 244 over time, according to examples of the present disclosure. For simplicity, the file system has three segments represented by boxes labelled "Segment S1" (i.e. Segment 1), "Segment S2" (i.e. Segment 2) and "Segment S3" (i.e. Segment 3). Each data update (indicated generally at 402 for simplicity) is associated with a timestamp indicating when the data update occurs, according to the originating segment.

[0060] Timelines 410, 412 and 414 indicate various data updates from S1, S2 and S3 respectively that occur over time. Timestamps *TS6* to *TS12* indicate when the data updates occur at the originating segment, according to the originating segment. The data updates occur according to their timestamp labelling order, for example data update with *TS6* occurs before *TS7*, *TS8* and so on. In practice, timestamps from the same source may be compared directly, but comparison of timestamps from different sources may be based on a clock skew among the sources. Timestamps within that clock skew may be considered as occurring “at the same time.”

[0061] In this example, journal 220 includes two data updates from S1 with timestamps *TS9* and *TS11* respectively, three from S2 with timestamps *TS7*, *TS9* and *TS10* respectively and three from S3 with timestamps *TS6*, *TS7* and *TS12* respectively. Data update 420 from S2 is a primary key modification operation (i.e. move *DirA DirB*) to rename *DirA* as *DirB* that is associated with timestamp *TS9*. At upload entity 230, data updates with different timestamps from a single segment are batched together, such as *SCU_1281* with data updates from S1, *SCU_1280* from S2 and *SCU_1279* from S3.

[0062] Since system 200 in Fig. 2 is a distributed file system, data updates from various segments may arrive at processing pipeline 244 in an out-of-order manner. For example, timeline 430 indicates different arrival times of *SCU_1281*, *SCU_1280* and *SCU_1279* at processing pipeline 244, i.e. *SCU_1279* is first uploaded, followed by *SCU_1280* and then *SCU_1281*. Although data update with timestamp *TS12* from S3 occurs at a later time, it is uploaded as part of *SCU_1279* before data updates with earlier timestamps (i.e. *TS7* to *TS11*) from segments S1 and S2. Similarly, although the data updates with timestamp *TS8* from S1 occurs earlier, it is uploaded at a later time in the primary key modification operation 420 with timestamp *TS9* from S2.

[0063] If primary key modification operation 420 with timestamp $TS9$ from $S2$ is applied before the (earlier) data update with timestamp $TS8$ from $S1$ is observed, it may adversely affect the $TS8$ data update because source primary key $DirA$ will be deleted. To determine whether a complete prefix of data updates have been observed from all $S1$, $S2$ and $S3$ up until the timestamp of the primary key modification (i.e. $TS9$), a value called “minimum freshness” is used.

[0064] For each segment, server system 240 records a freshness timestamp representing the time the latest data update for the segment. The minimum of these freshness timestamps across all segments, i.e. $\min(\text{freshness})$, represents the segment for which the system has the oldest information. Referring to 440 in the example in Fig. 4, the freshness timestamps and minimum freshness change over time as SCUs are uploaded, i.e. at 442 (before SCU_1279 is uploaded), 444 (after SCU_1279 is uploaded but before SCU_1280), 446 (after SCU_1279 and SCU_1280 are uploaded but before SCU_1281) and 448 (after all SCUs are uploaded).

[0065] At 442 in Fig. 4, before SCU_1279 is uploaded, the freshness timestamps for $S1$, $S2$ and $S3$ are $TS7$, $TS4$ and $TS5$ respectively, in which case the $\min(\text{freshness}) = \min(TS7, TS4, TS5) = TS4$. At 444 in Fig. 4, after SCU_1279 is uploaded, the freshness timestamp of $S3$ is updated from $TS5$ to $TS12$. Since data updates from $S1$ and $S2$ have not been uploaded, their freshness timestamp is unchanged. The minimum freshness across $S1$, $S2$ and $S3$ is therefore $\min(\text{freshness}) = \min(TS7, TS4, TS12) = TS4$.

[0066] At 446 in Fig. 4, after SCU_1280 is uploaded, the freshness timestamp of $S2$ changes from $TS4$ to $TS10$. The minimum freshness across $S1$, $S2$ and $S3$ is therefore $\min(\text{freshness}) = \min(TS7, TS10, TS12) = TS7$. Since the minimum freshness of $TS7 < TS9$ (i.e. less than the freshness of $TS9$ of

primary key modification 420), this indicates that data updates up until TS9 have not been observed.

[0067] At 448 in Fig. 4, after SCU_1281 is uploaded, the freshness timestamp of S1 changes from TS7 to TS11. The minimum freshness across S1, S2 and S3 is therefore $\min(\text{freshness}) = \min(TS11, TS10, TS12) = TS10$. Since minimum freshness $TS10 > TS9$, this indicates that data updates up until TS9 have been observed.

[0068] The freshness timestamp of each segment may be stored in a data structure, such as a freshness table indicated generally at 450 in Fig. 4. The minimum freshness timestamp changes to a newer timestamp whenever there has been an authority merge at processing pipeline 244, and updates from the oldest segment (the one which defines the minimum freshness) arrive and update the freshness of that segment to a fresher value. At this point, entries of the update data structure that are older than the new minimum freshness timestamp cannot have any outstanding updates with timestamps older than the minimum freshness timestamp.

[0069] The minimum freshness also represents a time boundary at which the system has a complete knowledge of all data updates that may have happened across all segments. The minimum freshness may also be defined as follows:

$$\min(\text{freshness}) = \min(ST_{\max} \text{ for source's uploaded SCU}).$$

[0070] STmax is the largest timestamp present in any uploaded SCU from a given source (e.g., segment S1, S2 or S3 in Fig. 4). An uploaded SCU is an SCU that has been ingested, but not transformed (e.g., sorted, merged, etc.) by processing pipeline 244.

[0071] In the example in Fig. 4, primary key modification operation 420 should be applied by deleting source primary key *DirA* and creating destination primary

key *DirB* once the minimum freshness value is greater than *TS9*, thereby indicating that a complete prefix of data updates up until *TS9* should have been observed from all segments by processing pipeline 244.

[0072] Primary key modification

[0073] According to process 100 in Fig. 1, once a complete prefix of data updates from all segments have been observed for a period of time, the primary key modification operation is applied within the processing pipeline. Any suitable implementation may be used to “wait” for the complete prefix within the processing pipeline, such as the following.

[0074] Priority processing approach, in which a primary key modification operation is separated into a first processing batch and other data updates that do not involve primary key modification into a second processing batch at ingest stage 248. The first processing batch is given a higher priority and processed first, and once a complete prefix is observed, the second processing batch is processed. This approach involves modifications to coordinator 242, ingest stage 248, sorting stage 250 and merging stage 252 of processing pipeline 244. See examples in Fig. 5 and Fig. 6.

[0075] Aggregation approach, in which data updates (including the primary key operation) are aggregated together. Once a complete prefix is observed, the aggregated data updates are processed by sorting stage 250 and merging stage 252 as a single processing batch. This approach involves addition of a new aggregating stage in addition to modifications to coordinator 242, ingest stage 248, sorting stage 250 and merging stage 252 of processing pipeline 244. See examples in Fig. 7 and Fig. 8.

[0076] It will be appreciated that the above “inline” approaches are expected to perform better than other approaches that require an external device to apply the primary key modification. However, both inline approaches introduce

processing delays, as they accumulate their complete prefix of operations to apply. In the aggregation approach, the accumulation occurs at the beginning of processing pipeline 244, which delays further processing in sorting 250 and merging 252 stages until the prefix is completed. In the priority processing approach, data updates not relating to primary key modification are held back (roughly by one processing iteration) until the prefix is completed. The data updates are processed in a time-ordered manner.

[0077] Priority processing approach (Fig. 5 and Fig. 6)

[0078] Fig. 5 illustrates the system in Fig. 2 processing a primary key modification operation using a priority processing approach, according to examples of the present disclosure. Fig. 6, which will be explained along with Fig. 5, is a flowchart of a detailed process 600 for processing a primary key modification operation using the priority processing approach, according to examples of the present disclosure. Blocks 510 to 595 in Fig. 5 correspond to blocks 610 to 695 in Fig. 6, respectively.

[0079] At 510 in Fig. 5 and 610 in Fig. 6, upload entity 230 retrieves from journal 220 an entry associated with primary key modification operation “*move DirA DirB*” in *AJFile_325*. Timestamp *TS9* of the primary key modification operation is also recorded in the journal entry. As previously explained with reference to Fig. 2 and Fig. 3, upload entity 230 also converts the format of *AJFile_325* into a format readable by processing pipeline 244.

[0080] At 520 in Fig. 5 and 620 in Fig. 6, upload entity 230 generates an SCU (i.e. *SCU_1280*) to record the primary key modification operation in a log data structure called “primary key modification log”. The purpose of this block is to allow upload entity 230 and processing pipeline 244 to keep track of all primary key modification operations from all segments of the file system. In this example, the log data structure stores metadata about “*move DirA DirB*”,

including source primary key *DirA*, destination primary key *DirB* and timestamp *TS9*.

[0081] At 530 in Fig. 5 and 630 in Fig. 6, upload entity 230 uploads the SCU (i.e. *SCU_1280*) to ingest stage 248 of processing pipeline 244.

[0082] At 540 in Fig. 5 and 640 in Fig. 6 (related to 110 in Fig. 1), ingest stage 248 receives *SCU_1280* that includes primary key modification operation to move source primary key *DirA* to destination primary key *DirB*. Meanwhile, ingest stage 248 also receives other data updates (not shown for simplicity).

[0083] At 550 and 552 in Fig. 5 and 660 in Fig. 6, ingest stage 248 separates primary key modification operation from other updates. In the example in Fig. 5, the result is two SCUs or processing batches: *SCU_1280* that includes the primary key modification operation and *SCU_1281* include all other data updates. The purpose is to allow separate processing of *SCU_1280* and *SCU_1281*, since the primary key modification should not be applied until data updates from all segments of the file system of timestamp *TS9* have been observed by processing pipeline 244 as explained with reference to Fig. 4. Although not shown, *SCU_1280* may also include other primary key modification operations that are separated from the other (non-primary key modification) data updates.

[0084] At 560 in Fig. 5 and 660 in Fig. 6 (related to 120 in Fig. 1), ingest stage 248 also inspects each data update (primary key modification and others) to extract their timestamp information. Once the data updates are durably stored, ingest stage 248 sends the extracted timestamp information of data updates in unprocessed SCUs to coordinator 242. For example, as explained with reference to Fig. 4, freshness of the data updates may be determined based on their timestamp information. When an SCU from a particular segment is ingested, ingest stage 248 captures the freshness of the segment and sends it to coordinator 242.

[0085] At 570 in Fig. 5 and 670 in Fig. 6 (related to 120 in Fig. 1), coordinator 242 determines whether data updates from all segments in the file system up until the timestamp of the primary key modification operation have been observed. As explained with reference to Fig. 4, coordinator 242 may determine whether the minimum freshness of data updates from all segments of the file system has exceeded timestamp TS_9 as follows:

[0086] Coordinator 242 analyses timestamp information provided by ingest stage 248 to determine the minimum freshness value of data updates from all segments.

[0087] At 570, when the minimum freshness timestamp is greater than a timestamp of the primary key modification operation (i.e. $\min(\text{freshness}) > TS_9$ in Fig. 5), coordinator 242 determines that data updates from all segments of the file system up until the timestamp of the primary key modification operation have been up observed.

[0088] At 580, 582 and 584 in Fig. 5 and 680 in Fig. 6 (related to 120 in Fig. 1), coordinator 242 schedules processing of the primary key modification operation in SCU_1280 and other updates in SCU_1281 . In one example, SCU_1280 is pushed through the processing pipeline 244 while SCU_1281 are held back until a complete prefix of data updates has been observed.

[0089] At 580 in Fig. 5, coordinator 242 schedules processing of primary key modification operation in SCU_1280 . Sorting stage 250 and merging stage 252 transforms SCU_1280 until it is in authority 254 and accessible to process a query.

[0090] At 582 in Fig. 5, after being pushed through processing pipeline 244, the primary key modification operation is recorded in a log data structure called "primary key modification log". The purpose of this block is to allow processing pipeline 244 to keep track of all primary key modification operations from all

segments of the file system and apply them at sorting stage 250 and merging stage 252 of processing pipeline 244 once a complete prefix of updates is available.

[0091] At 584 in Fig. 5, coordinator 242 schedules processing of other data updates in *SCU_1281*. In particular, when $\min(\text{freshness}) > TS9$, *SCU_1281* is processed by sorting stage 250 and merging stage 252 in subsequent blocks.

[0092] At 590 and 592 in Fig. 5 and 690 in Fig. 6 (related to 130 and 140 in Fig. 1), sorting stage 250 of processing pipeline 244 performs sorting and applies primary key modification to other data updates in *SCU_1281*. Sorting stage 250 takes primary key modification into account during sorting, to account for possible operations on the same primary key that straddle a primary key modification. Otherwise, there might be information loss and inconsistent system after SCUs are merged together. In one example, sorting stage 250 performs the following:

[0093] At 590 in Fig. 5, as sorting stage 250 reads input *SCU_1281*, it retrieves a primary key modification operation from primary key modification log (see 582) that satisfies $ST_{\min} < \text{timestamp of primary key modification operation} < ST_{\max}$. $ST_{\max}$ is the largest timestamp present in *SCU_1281*, while $ST_{\min}$ is its smallest timestamp. In other words, if the primary key modification occurs after an event in the SCU, then the event in the SCU could not possibly be affected by the primary key modification. On the other hand, if the primary key modification occurs prior to an event in the SCU, then that event must be checked against the primary key modification even to determine if it needs to be updated.

[0094] At 592 in Fig. 5, as *SCU_1281* is sorted, sorting stage 250 generates a new *SCU_1281A* ("rename SCU") to apply the primary key modification by generating a delete operation for the source primary key (i.e. DirA) and a create

operation for the destination primary key (i.e. *DirB*). In general, sorted *SCU_1281A* at 592 is logically attached to sorted *SCU_1281* at 590.

[0095] For each record in a data structure that is associated with source primary key *DirA*, sorting stage 250 generates a delete operation to delete source primary key *DirA* and a create operation to create destination primary key *DirB* in the record. See 592 again. Directories *DirA* and *DirB* may be relative or absolute paths. Table 1 sets out the pseudo-code to perform the above, according to examples of the present disclosure. The “rename key” represents the source primary key.

Table 1: Pseudo-code for sorting stage 250

```

If (record does not have deleted flag) {
    For (each rename with timestamp >= record timestamp) {
        If (record key matches rename key) {
            Insert a delete record into the rename SCU for the
            source key. Use timestamp of rename;
            Insert a create record into the rename SCU for the
            destination key. Use timestamp of rename. The record contents
            are defined by calling a per-table user defined function on the
            original record;
        }
    }
}

```

[0096] According to Table 1, newly defined record contents are defined by calling a user-defined function on record contents. The function may be defined as part of a schema, should take a record, and generate a new record in response.

[0097] The data structure from which affected records are retrieved may be data structure associated with File Objects, Change Log, File Delete Time, File Lifetime tables, etc. For most tables (e.g., File Objects table), the function may be an identity, as the desired contents should be identical in the new records and old records. For other tables (e.g., Change Log, File Delete Time, File Lifetime, etc.), their contents may be updated in other, table-specific ways.

[0098] At 595 and 596 in Fig. 5 and 695 in Fig. 6 (related to 130 and 140 in Fig. 1), merging stage 252 of processing pipeline 244 performs merging and applies primary key modification to other data updates in *SCU_1281*.

[0099] At 595 in Fig. 5, analogous to sorting stage 250, merging stage 252 applies primary key modification operation with a timestamp greater than $ST_{\max}$ before it is written out to a merged file. The primary key modification operation may be retrieved from log data structure 582 ("primary key modification log") at the end of pipeline 244.

[00100] At 596 in Fig. 5, as *SCU_1281* is merged, merging stage 252 generates a new *SCU_1281A* ("rename SCU") to apply the primary key modification by generating a delete operation for the source primary key (i.e. *DirA*) and a create operation for the destination primary key (i.e. *DirB*). In general, merged *SCU_1281A* at 596, is logically attached to merged *SCU_1281* at 595.

[00101] Similar to the logic in Table 1, for each record in a data structure that is associated with source primary key *DirA*, merging stage 252 generates a delete operation to delete source primary key *DirA* and a create operation to create destination primary key *DirB* in the record. Similar to sorting stage 250, the data structure from which affected records may be retrieved may be data structure storing metadata, such as File Objects, Change Log, File Delete Time, File Lifetime tables, etc. Table 2 sets out the pseudo-code to perform the above, according to examples of the present disclosure.

Table 2: Pseudo-code for merging stage 252

```

If (record does not have deleted flag) {
  For (each rename with timestamp >= record timestamp) {
    If (record key matches rename key) {
      Insert a delete record into the rename SCU for the
      source key. Use timestamp of rename;
      Insert a create record into the rename SCU for the
      destination key. Use timestamp of rename. The record contents

```

```
are defined by calling a per-table user defined function on the
original record;
        }
    }
}
```

[00102] Once the primary key modification operation in primary key modification log data structure 582 has been processed, the SCU created as a result of the merging stage should be sorted and de-duplicated before they are merged, effectively stalling processing of the merge until these operations can be completed. Once sorted and de-duplicated, the merge can proceed as usual.

[00103] The end result, after sorting and merging, is stored in authority 254 (i.e. output data structure) that is accessible to process a query.

[00104] Although not shown in Fig. 5 and Fig. 6, primary key modification operations stored in “primary key modification log” data structure 582 may be removed after they are applied. For example, once all SCUs with ST_{min} less than the oldest log entry have gone through a final authority merge, the log entry may be removed. The process of checking and deleting entries may be performed by coordinator 242, or an external entity (not shown for simplicity).

[00105] Aggregation approach (Fig. 7 and Fig. 8)

[00106] Fig. 7 illustrates the system in Fig. 2 processing a primary key modification operation using an aggregation approach, according to examples of the present disclosure. Fig. 8, which will be explained along with Fig. 7, is a flowchart of a detailed process 800 for processing a primary key modification operation using the aggregation approach, according to examples of the present disclosure. Blocks 710 to 795 in Fig. 7 correspond to blocks 810 to 895 in Fig. 8, respectively.

[00107] Compared to Fig. 2 and Fig. 5, processing pipeline 244 in Fig. 7 includes a new processing stage, named “aggregating stage” 702. If a primary key modification is ingested into ingest stage 248, multiple subsequent SCUs are aggregated by aggregating stage 702 until a complete prefix is observed.

[00108] At 710 to 730 in Fig. 7 and 810 to 830 in Fig. 8, upload entity 230 retrieves primary key modification operation from journal 220, generates *SCU_1280* and uploads it to processing pipeline 244. For brevity, the description for blocks 510 to 530 in Fig. 5 and blocks 610 and 630 in Fig. 6 are also applicable and will not be repeated here.

[00109] At 740 in Fig. 7 and 840 in Fig. 8 (related to 110 in Fig. 1), ingest stage 248 receives *SCU_1280* that includes primary key modification operation to move source primary key *DirA* to destination primary key *DirB* (indicated at 742). Ingest stage 248 also receives a number of other SCUs, such as *SCU_1281* to *SCU_1299* (indicated at 744 and 746).

[00110] At 750 and 760 in Fig. 7 and corresponding 850 and 860 in Fig. 8 (related to 120 in Fig. 1), aggregating stage 702 determines whether data updates from all segments in the file system up until the timestamp of the primary key modification operation have been observed. If yes, aggregating stage 702 aggregates the data updates into a single, larger batch of updates before they are sorted and merged in subsequent sorting 250 and merging 252 stages.

[00111] At 750 in Fig. 7, aggregating stage 702 monitors freshness timestamp of each segment, in particular, whether the minimum freshness timestamp across all segments is less than the timestamp of the primary key modification operation (i.e. $\min(\text{freshness}) > TS9$).

[00112] At 760 in Fig. 7, when $\min(\text{freshness}) > TS9$, aggregating stage 702 aggregates the primary key modification operation with other data updates. For

example, unprocessed SCUs *SCU_1280* to *SCU_1299* are aggregated into a larger “mega SCU”. In other words, aggregation does not start until $\min(\text{freshness}) > TS9$.

[00113] Coordinator 242 is capable of communicating with aggregating stage 702 to keep track of which SCUs are aggregated.

[00114] At 770 and 775 in Fig. 7 and 870 in Fig. 8 (related to 130 and 140 in Fig. 1), sorting stage 250 of processing pipeline 244 performs sorting and applies primary key modification to the aggregated SCU (i.e. *SCU_1280* to *SCU_1299*).

[00115] At 770, sorting stage 250 sorts the aggregated SCU. At 775, as the aggregated SCU is sorted, sorting stage 250 also generates a new SCU (e.g. “rename SCU”) to apply the primary key modification by generating a delete operation for the source primary key (i.e. *DirA*) and a create operation for the destination primary key (i.e. *DirB*).

[00116] The processes discussed with reference to 590 in Fig. 5 and 690 in Fig. 6, such as Table 1, are also applicable here and will not be repeated for brevity.

[00117] At 780 and 785 in Fig. 7 and 880 in Fig. 8 (related to 130 and 140 in Fig. 1), merging stage 252 of processing pipeline 244 performs merging and applies primary key modification to the aggregated SCU (i.e. *SCU_1280* to *SCU_1299*).

[00118] At 780, merging stage 250 merges the aggregated SCU. At 785, as the aggregated SCU is merged, merging stage 250 also applies the primary key modification by generating a delete operation for the source primary key (i.e. *DirA*) and a create operation for the destination primary key (i.e. *DirB*).

[00119] The processes discussed with reference to 595 in Fig. 5 and 695 in Fig. 6, such as Table 2, are also applicable here and will not be repeated for brevity.

[00120] The end result, after sorting and merging, is stored in authority 254 (i.e. output data structure) that is accessible to process a query.

[00121] Additional processing

[00122] Further, some data structures in authority 254 may require additional processing that goes beyond preserving original values in records associated with the source primary key. Some examples of additional processing are provided below.

[00123] In a data structure storing metadata associated with File Lifetime (FL), updating a delete time attribute of the source primary key to a timestamp of the primary key modification operation and a create time attribute of the destination primary key as a create time of the source primary key.

[00124] In a data structure storing metadata associated with Change Log (CL), updating a last activity time attribute of the source primary key to a timestamp of the primary key modification operation.

[00125] The above additional processing may be performed by server system 240. To improve scalability and performance, the above additional processing may be implemented within processing pipeline 244. For example, the pseudo-codes in Table 1 and Table 2 may further include functions that are specific to the different data structure types and executed during sorting 250 and merging stage 252. In this case, it is not necessary to generate an additional SCU for the additional processing; the updates are simply applied along with the primary key modification.

[00126] Further, upload entity 230 (e.g. AJ Scanner) may be modified to differentiate between two types of primary key modification operation: file rename and directory rename. If the entity being renamed is a directory, upload entity 230 uploads an additional rename operation with an appropriate wildcard character (*) to processing pipeline 244 at 530 in Fig. 5 and 730 in Fig. 7.

[00127] The additional operation ensures children of the directory are also renamed. For example, *"move DirA DirB"* would result in the directory rename itself, plus rename of its children, i.e. *"move DirA/* DirB/*"*. When the primary key modification is to rename a file name or directory name, the destination primary key should inherit metadata or attributes of the source primary key, including non-physical inode metadata and any other custom attribute.

[00128] Example Devices 900

[00129] The above examples can be implemented by hardware, software or firmware or a combination thereof. Referring to Fig. 8, example device 900 is capable of acting as upload entity 230, server system 240 or client device 260. Device 900 includes processor 910, memory 920 and network interface device 940 that communicate with each other via bus 930. Although not shown device 900 may include a persistent storage (e.g., disk), in case memory (e.g., dynamic random-access memory (DRAM)) does not have sufficient storage for processing. Processor 910 is to perform processes described herein with reference to Fig. 1 to Fig. 8.

[00130] Memory 920 may store any necessary data 922 for facilitating processing of primary key modifications. Memory 920 may store machine-readable instructions 924 executable by the processor 910 to cause the processor 910 to perform processes described herein with reference to Fig. 1 to Fig. 8.

[00131] The methods, processes and units described herein may be implemented by hardware (including hardware logic circuitry), software or firmware or a combination thereof. The term "processor" is to be interpreted broadly to include a processing unit, application-specific integrated circuit (ASIC), logic unit, or programmable gate array etc. The processes, methods and functional units may all be performed by the one or more processors 910; reference in this disclosure or the claims to a "processor" should thus be interpreted to mean "one or more processors".

[00132] Further, the processes, methods and functional units described in this disclosure may be implemented in the form of a computer software product. The computer software product is stored in a storage medium and comprises a plurality of instructions for making a processor to implement the methods recited in the examples of the present disclosure.

[00133] The figures are only illustrations of an example, wherein the units or procedure shown in the figures are not necessarily essential for implementing the present disclosure. Those skilled in the art will understand that the units in the device in the example can be arranged in the device in the examples as described, or can be alternatively located in one or more devices different from that in the examples. The units in the examples described can be combined into one module or further divided into a plurality of sub-units.

[00134] Although the flowcharts described show a specific order of execution, the order of execution may differ from that which is depicted. For example, the order of execution of two or more blocks may be changed relative to the order shown. Also, two or more blocks shown in succession may be executed concurrently or with partial concurrence. All such variations are within the scope of the present disclosure.

[00135] Throughout the present disclosure, the word "comprise", or variations such as "comprises" or "comprising", will be understood to imply the inclusion

of a stated element, integer or step, or group of elements, integers or steps, but not the exclusion of any other element, integer or step, or group of elements, integers or steps.

[00136] It will be appreciated by persons skilled in the art that numerous variations and/or modifications may be made to the above-described embodiments, without departing from the broad general scope of the present disclosure. The present embodiments are, therefore, to be considered in all respects as illustrative and not restrictive.

Claims

1. A method for processing a primary key modification using a processing pipeline with a first processing stage and a second processing stage to process data updates from a plurality of update sources, the method comprising:

the first processing stage receiving a primary key modification operation, wherein the primary key modification operation is from one of the plurality of update sources to modify a source primary key to a destination primary key and the primary key modification operation is associated with a timestamp; and

in response to determination that data updates from all of the plurality of update sources up until the timestamp of the primary key modification operation have been received by the processing pipeline, the second processing stage:

applying the primary key modification operation by generating a delete operation to delete the source primary key and a create operation to create the destination primary key; and

transforming the delete operation and create operation into an output data structure that is accessible to process a query.

2. The method of claim 1, wherein the determination comprises:

determining a minimum freshness timestamp of the data updates from all of the plurality of update sources; and

when the minimum freshness timestamp is greater than a timestamp of the primary key modification, determining that data updates from all of the plurality of update sources up until the timestamp of the primary key modification operation have been received by the processing pipeline.

3. The method of claim 2, wherein the determination further comprises:

the first processing stage providing, as the primary key modification operation and other data updates are received, timestamp information of the primary key modification operation and other data updates to a coordinator of the processing pipeline;

based on the timestamp information, the coordinator determining the minimum freshness timestamp; and

when the minimum freshness timestamp is greater than a timestamp of the primary key modification, the coordinator determining that data updates from all of the plurality of update sources up until the timestamp of the primary key modification operation have been received by the processing pipeline.

4. The method of claim 3, wherein after receiving the primary key modification operation and other data updates, the method further comprises:

the first processing stage separating the primary key modification operation into a first processing batch and the other data updates into a second processing batch; and

the coordinator scheduling the first processing stage to provide the first processing batch to a sorting stage and a merging stage of the processing pipeline.

5. The method of claim 4, wherein applying the primary key modification operation comprises:

when the minimum freshness timestamp is greater than a timestamp of the primary key modification operation, the coordinator scheduling the first processing stage to provide the second processing batch to the sorting stage and merging stage of the processing pipeline.

6. The method of claim 2, wherein the second processing stage comprises an aggregating stage and the determination further comprises:

the aggregating stage determining the minimum freshness timestamp based on timestamp information of the primary key modification operation and other data updates received by the first processing stage; and

when the minimum freshness timestamp is greater than a timestamp of the primary key modification, the aggregating stage determining that data

updates from all of the plurality of update sources up until the timestamp of the primary key modification operation have been observed.

7. The method of claim 6, further comprising:

the aggregating stage aggregating the primary key modification operation with other data updates received by the first processing stage; and

when the minimum freshness timestamp is greater than a timestamp of the primary key modification, the aggregating stage providing the aggregated primary key modification operation with other data updates to a sorting stage and a merging stage of the processing pipeline.

8. The method of claim 1, wherein the second processing stage further comprises a sorting stage to perform sorting and applying the primary key modification operation comprises:

the sorting stage retrieving records of a data structure that are associated with the source primary key; and

the sorting stage generating, for each record associated with the source primary key, a delete operation to delete the source primary key and a create operation to create the destination primary key in the record.

9. The method of claim 1, wherein the second processing stage further comprises a merging stage to perform merging and applying the primary key modification operation comprises:

the merging stage retrieving records of a data structure that are associated with the source primary key; and

the merging stage generating, for each record associated with the source primary key, a delete operation to delete the source primary key and a create operation to create the destination primary key in the record.

10. The method of claim 9, wherein the data structure, from which records associated with the source primary key are retrieved, stores metadata associated with objects in the system.

11. The method of claim 9, further comprising the processing pipeline processing at least one of the following:

- data update to update, in a data structure storing metadata associated with File Lifetime, a delete time attribute of the source primary key to a timestamp of the primary key modification operation and a create time attribute of the destination primary key as a create time of the source primary key;

- data update to update, in a data structure storing metadata associated with Change Log, a last activity time attribute of the source primary key to a timestamp of the rename operation;

- data update to record, in a data structure storing metadata associated with File Delete Time, the source primary key and timestamp of the primary key modification operation; and

- data update to record, in a data structure storing metadata associated with File Rename Time, the source primary key, destination primary key and timestamp of the primary key modification operation.

12. The method of claim 1, wherein the plurality of update sources are segments of a distributed file system.

13. The method of claim 1, wherein when the source primary key and destination primary key are:

- hierarchical primary key names, wherein the primary key modification operation further comprises an operation to rename children of the source primary key as children of the destination primary key.

14. A computer-readable medium storing instructions, which when executed by a processor of a device supporting a processing pipeline with a first

processing stage and a second processing stage to process data updates from a plurality of update sources, cause the processor to perform processing of a primary key modification using the processing pipeline, comprising:

the first processing stage receiving a primary key modification operation, wherein the primary key modification operation is from one of the plurality of update sources to modify a source primary key to a destination primary key and the primary key modification operation is associated with a timestamp; and

in response to determination that data updates from all of the plurality of update sources until the timestamp of the primary key modification operation have been received by the processing pipeline, the second processing stage:

applying the primary key modification operation by generating a delete operation to delete the source primary key and a create operation to create the destination primary key; and

transforming the delete operation and create operation into an output data structure that is accessible to process a query.

15. A device to perform processing of a primary key modification using a processing pipeline with a first processing stage and a second processing stage to process data updates from a plurality of update sources, comprising a processor to:

receive, by the first processing stage, a primary key modification operation, wherein the primary key modification operation is from one of the plurality of update sources to modify a source primary key to a destination primary key and the primary key modification operation is associated with a timestamp; and

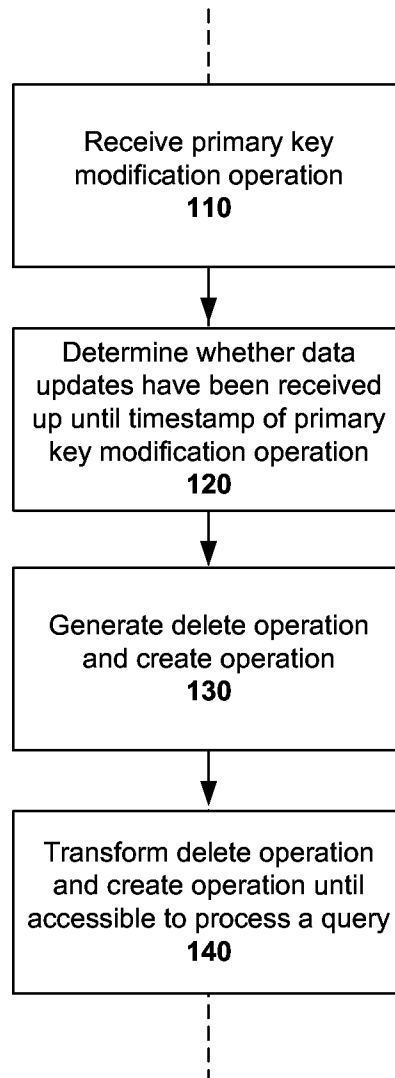
in response to determination that data updates from all of the plurality of update sources up until the timestamp of the primary key modification operation have been received by the processing pipeline:

apply, by the second processing stage, the primary key modification operation by generating a delete operation to delete the

source primary key and a create operation to create the destination primary key; and

transform, by the second processing stage, the delete operation and create operation into an output data structure that is accessible to process a query.

1/9

100
↓**Processing pipeline
with processing stages****Fig. 1**

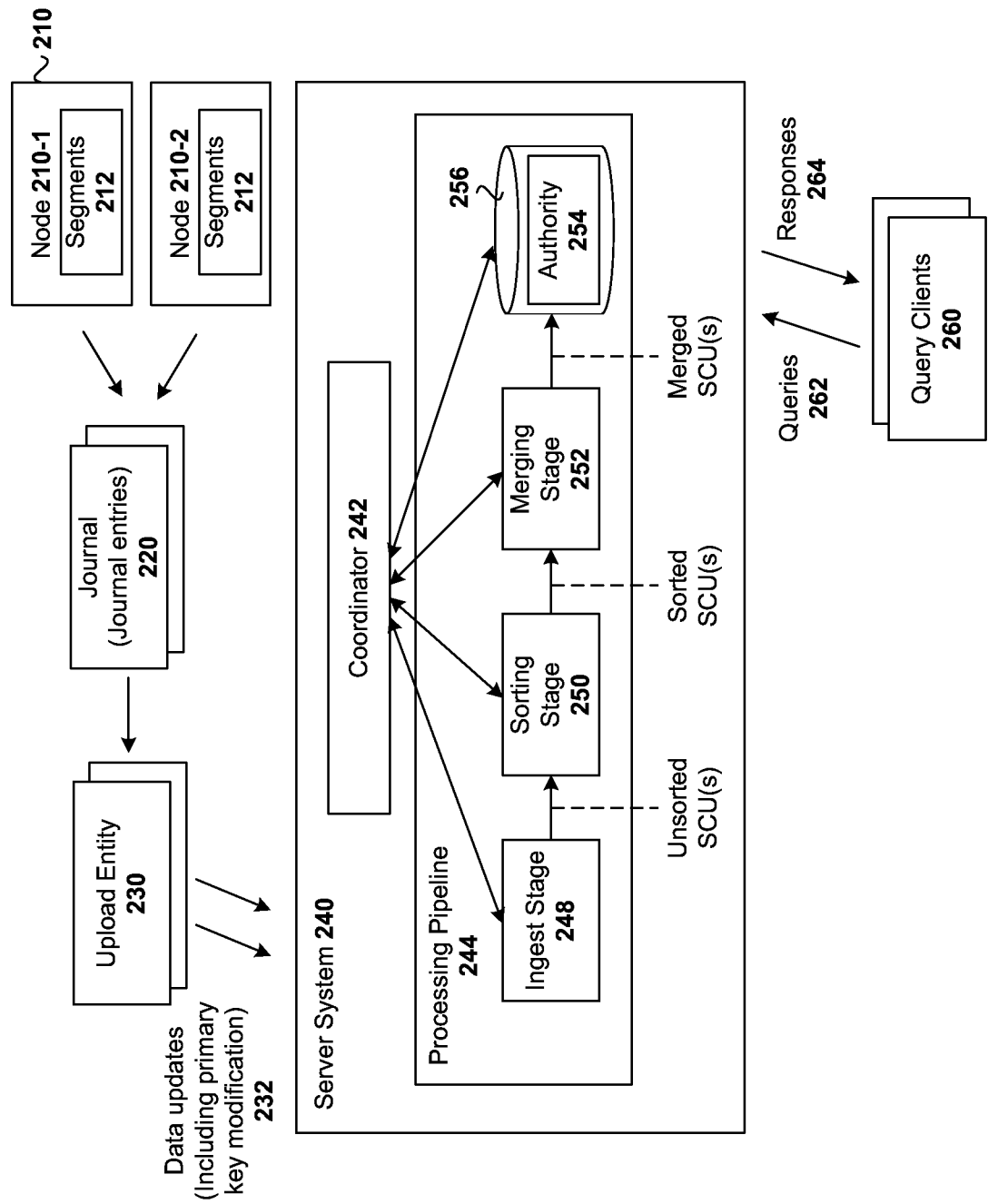


Fig. 2

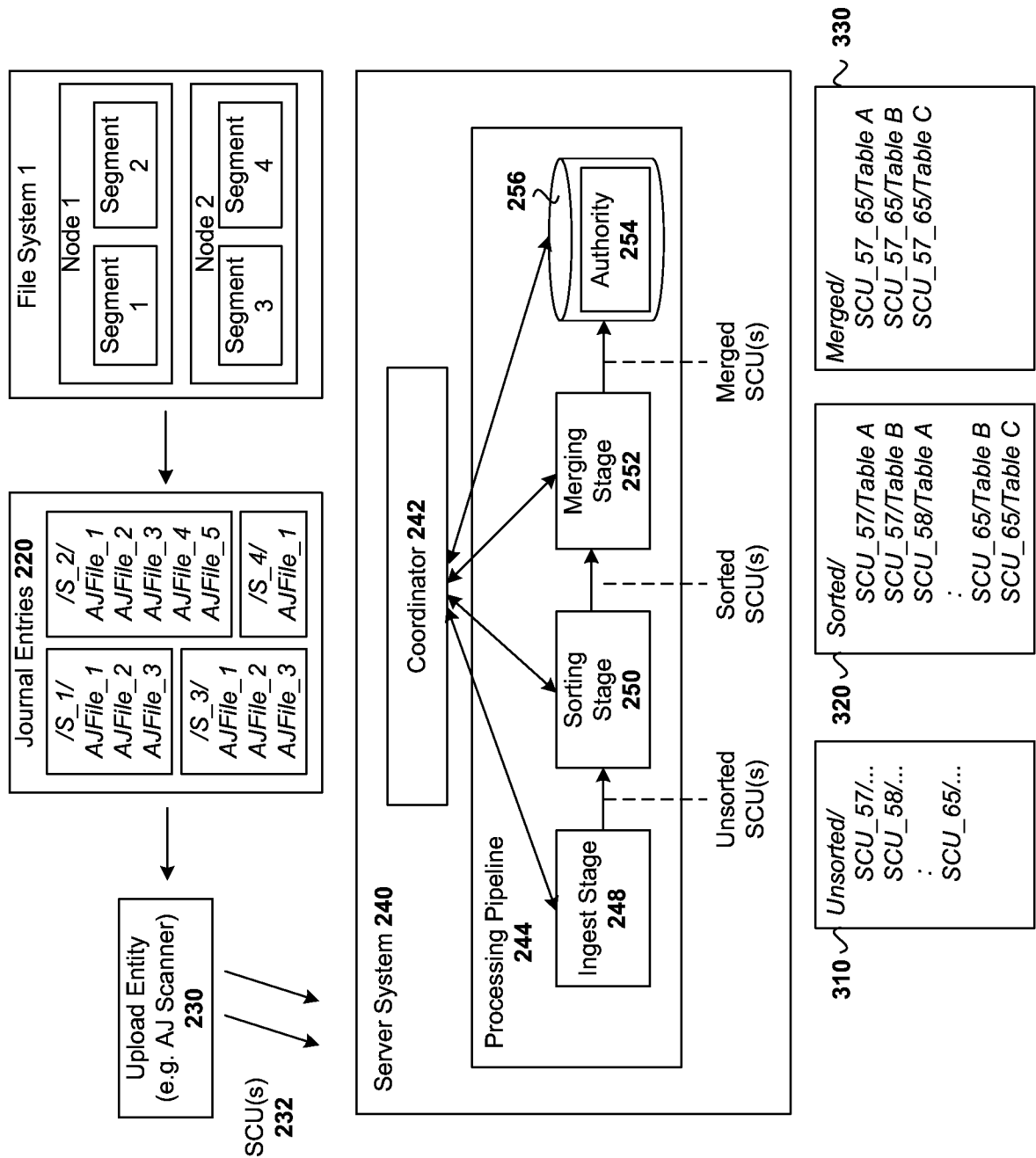


Fig. 3

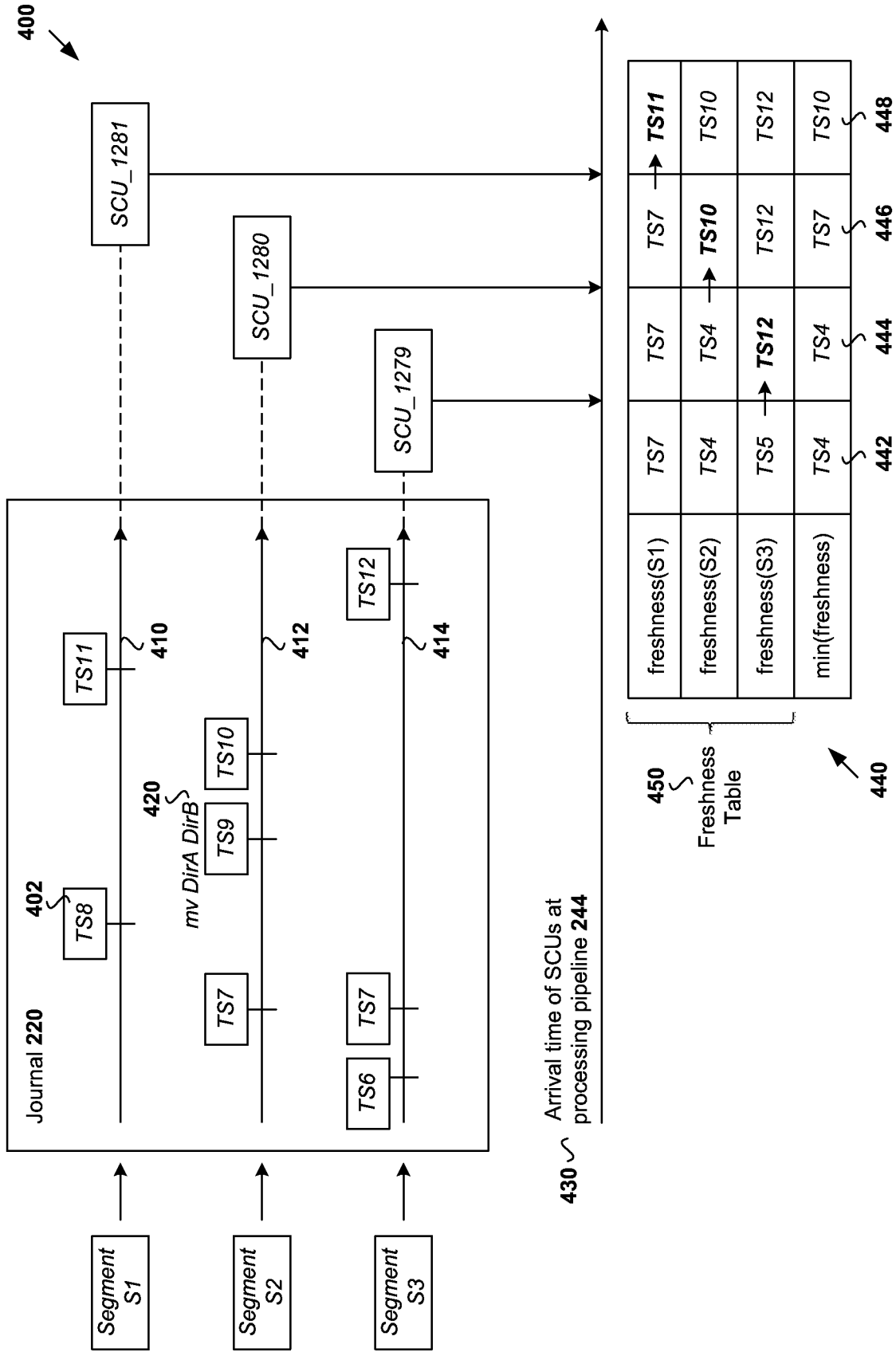


Fig. 4

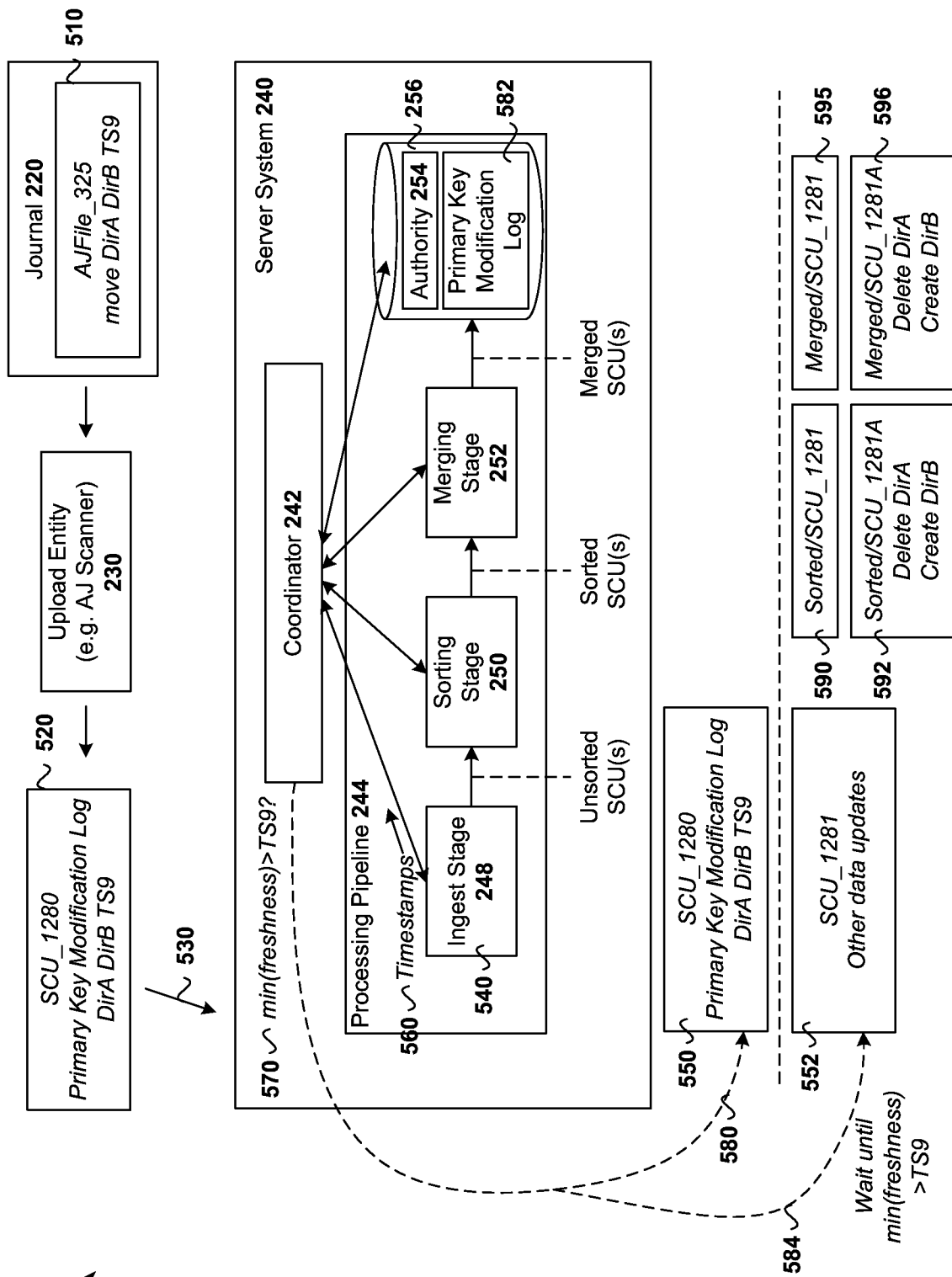


Fig. 5

6/9

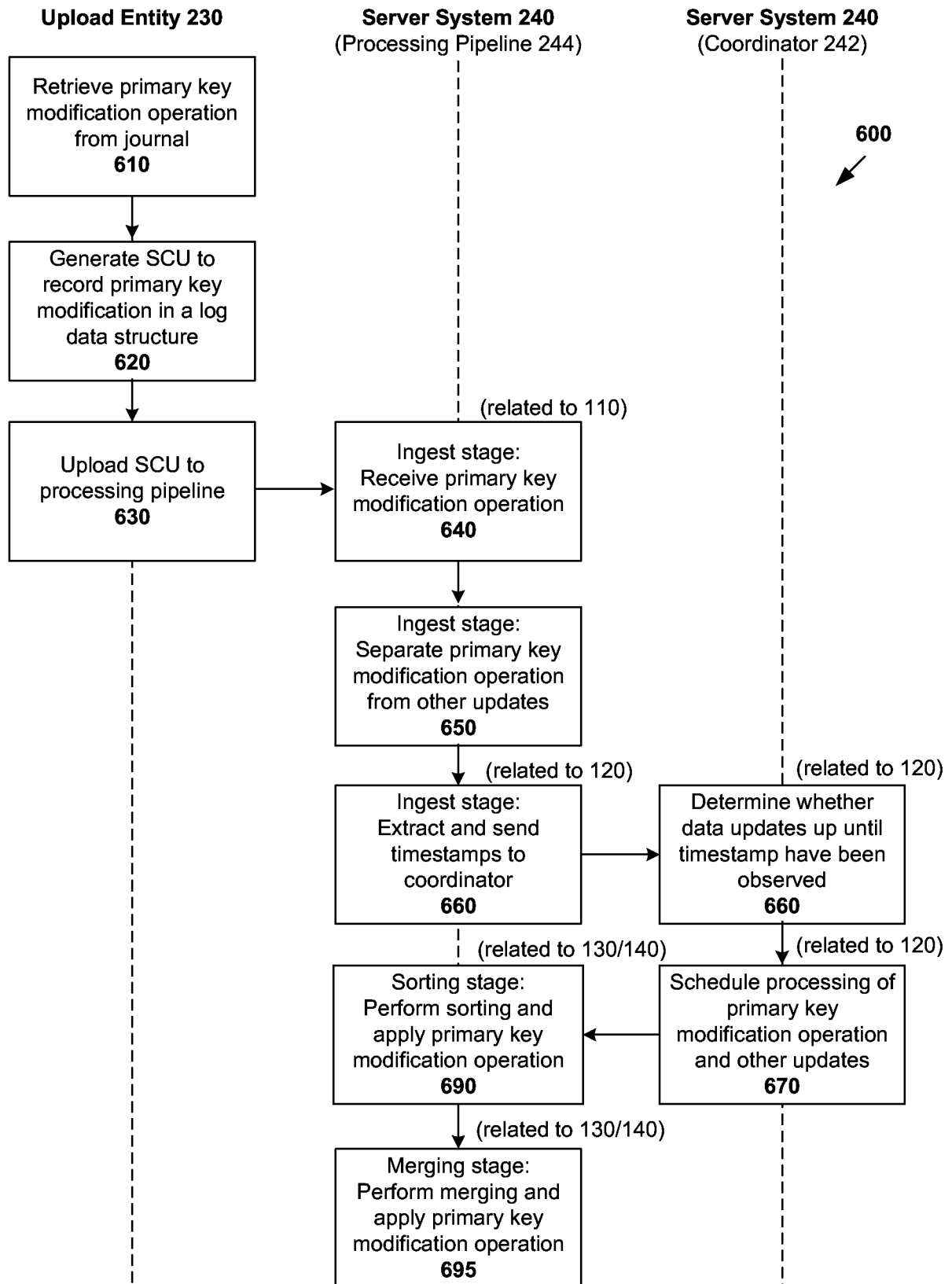


Fig. 6

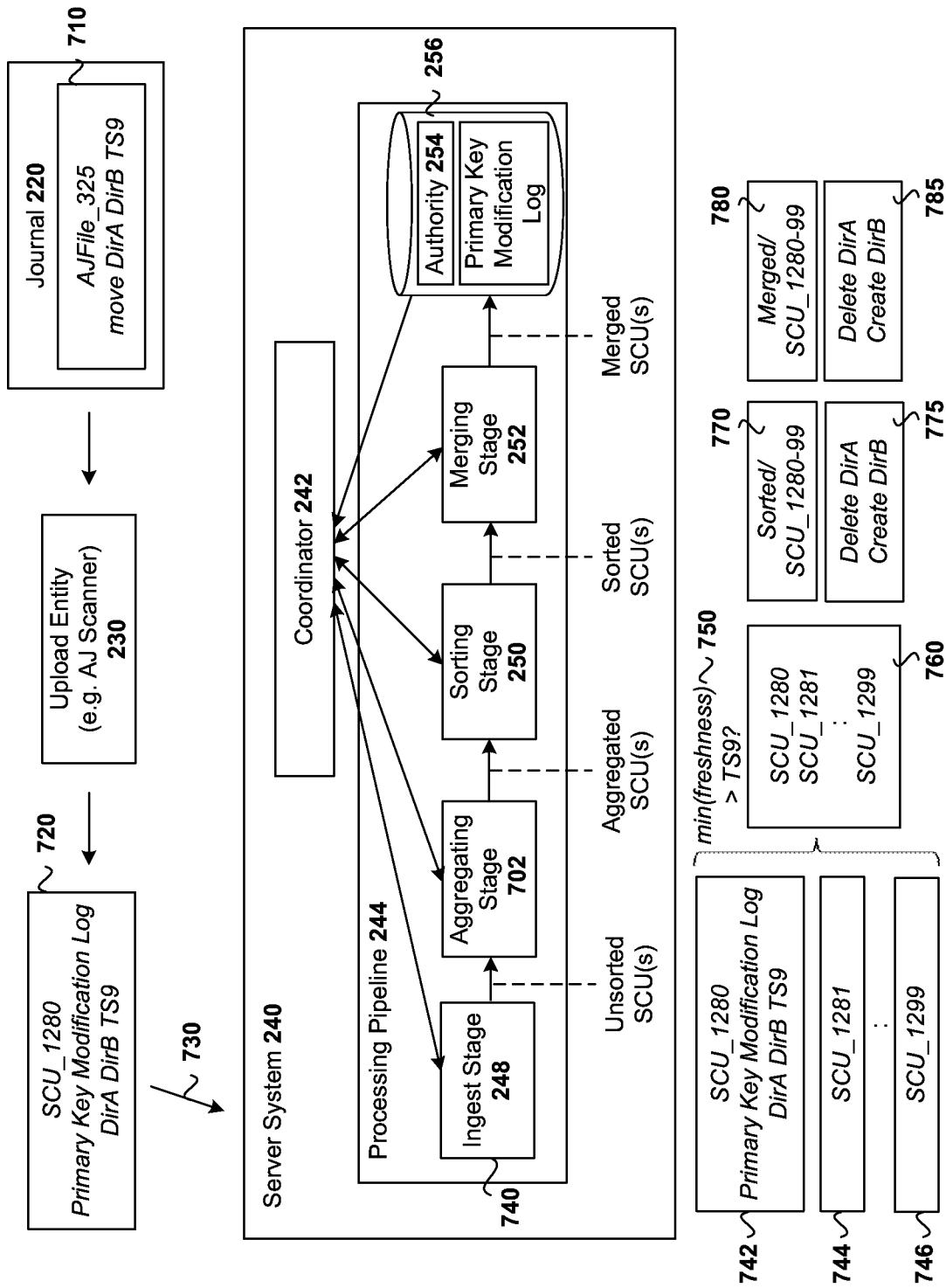


Fig. 7

8/9

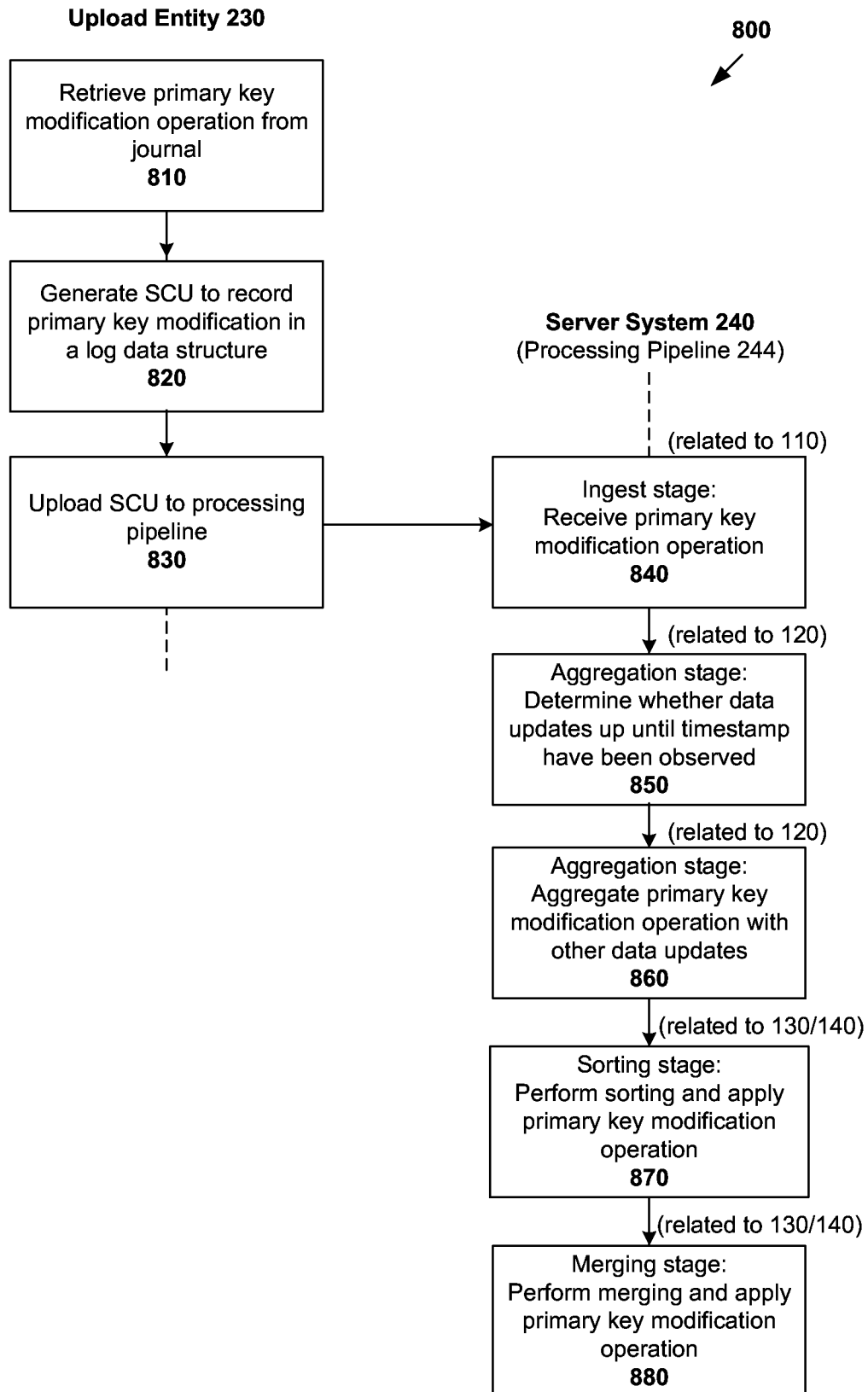


Fig. 8

9/9

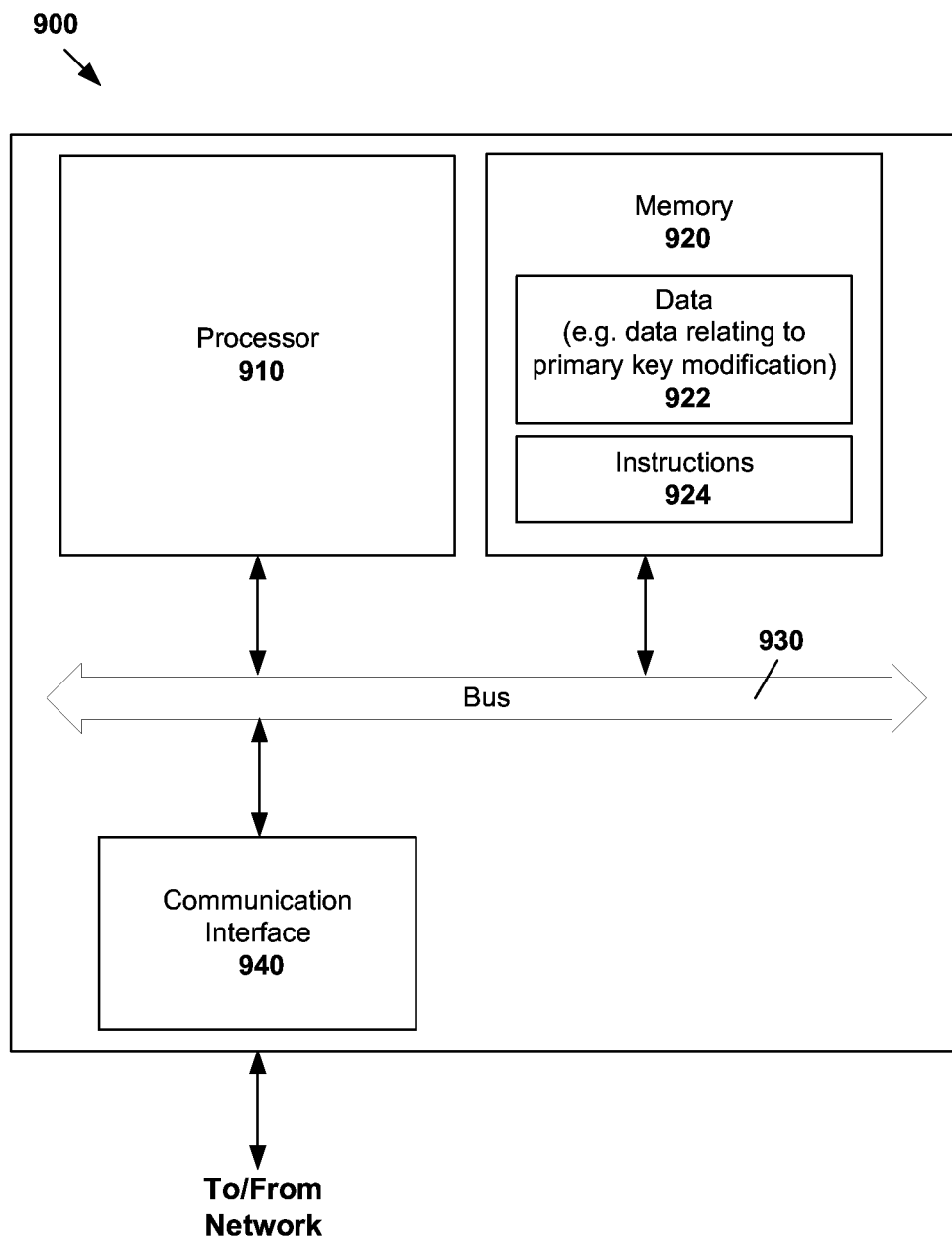


Fig. 9

A. CLASSIFICATION OF SUBJECT MATTER**G06F 17/30(2006.01)i, G06F 9/38(2006.01)i, G06F 9/44(2006.01)i**

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

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F 17/30; G06F 9/38; G06F 9/44

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

Korean utility models and applications for utility models

Japanese utility models and applications for utility models

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

eKOMPASS(KIPO internal) & Keywords: processing pipeline, updates, primary key modification, timestamp, stage, freshness, and similar terms.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	JOHNSON, CHARLES et al., "From Research to Practice: Experiences Engineering a Production Metadata Database for a Scale Out File System," In: Proceedings of the 12th USENIX conference on File and Storage Technologies (FAST '14), pp. 191-198, 17-20 February 2014 (https://www.usenix.org/conference/fast14/technical-sessions/presentation/johnson) See pages 191-195 and 197; and figure 1.	1-15
A	CIPAR, JAMES et al., "LazyBase: Trading Freshness for Performance in a Scalable Database," In: Proceedings of the 7th ACM european conference on Computer Systems, pp. 169-182, 2012 (http://dl.acm.org/citation.cfm?id=2168854) See pages 172-173 and figure 1.	1-15
A	US 8,311,982 B2 (SOULES, CRAIG A. N. et al.) 13 November 2012 See column 2, line 48 - column 6, line 15; and figures 1-2.	1-15
A	KEETON, KIMBERLY et al., "LazyBase: Freshness vs. performance in information management," ACM SIGOPS Operating Systems Review, Vol. 44, No. 1, pp. 15-19, January 2010 (http://dl.acm.org/citation.cfm?id=1740395) See pages 16-17 and figure 1.	1-15



Further documents are listed in the continuation of Box C.



See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search

28 November 2014 (28.11.2014)

Date of mailing of the international search report

28 November 2014 (28.11.2014)

Name and mailing address of the ISA/KR

International Application Division
Korean Intellectual Property Office
189 Cheongsu-ro, Seo-gu, Daejeon Metropolitan City, 302-701,
Republic of Korea

Facsimile No. +82-42-472-7140

Authorized officer

NHO, Ji Myong

Telephone No. +82-42-481-8528



INTERNATIONAL SEARCH REPORT

International application No.

PCT/US2014/020937

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT		
Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2009-0089334 A1 (MOHAMED, AHMED et al.) 02 April 2009 See paragraph [0041] and figure 6.	1-15

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2014/020937Patent document
cited in search reportPublication
datePatent family
member(s)Publication
date

US 8311982 B2

13/11/2012

US 2011-196880 A1

11/08/2011

US 2009-0089334 A1

02/04/2009

US 7779045 B2

17/08/2010

WO 2009-042920 A2

02/04/2009

WO 2009-042920 A3

22/05/2009