

**PARALLEL PROCESSING IN DATABASE SYSTEMS STORING A TABLE AS KEY VALUE PAIRS IN
MULTIPLE INDEPENDENT SORTED STRUCTURES WITH OVERLAPPING KEYS**

Priority Claim

5 [001] The present patent application is related to and claims the benefit of priority to the co-
pending US provisional patent application entitled, “DYNAMIC INTERLEAVED PARALLEL
PROCESSING IN DATABASE SYSTEMS THAT USE LOG-STRUCTURED MERGE TREE
STORAGE ENGINE”, Serial No.: 63/623,832, Filed: January 23, 2024, which is incorporated in
its entirety herewith to the extent not inconsistent with the description herein.

10

Background of the Disclosure

[002] Technical Field

[003] The present disclosure relates to database systems and more specifically to parallel
processing in database systems storing a table as key value pairs in multiple independent sorted
15 structures with overlapping keys.

[004] Related Art

[005] Database systems are often designed to store data of a table in the form of key-value pairs,
as is well known in the relevant arts. The keys serve as unique identifiers for the retrieval of the
20 values.

[006] To accommodate a large volume of data of a table, the key-value pairs may be stored in
multiple independent sorted structures. Specifically, each structure is sorted by the key, thereby
facilitating efficient retrieval of values. Each structure is stored independently, implying that the
corresponding key-value pairs are stored in a separate storage. An example of such an independent
25 sorted structure is a Sorted String Table (“SSTable”) file, well known in the relevant arts.

[007] The keys in such structures may be overlapping. An overlap implies that the same key is
present in multiple structures, for example, due to the feature that any update to a value of a key,
is stored as a new key-value pair (instead of overwriting the value of the key) potentially in another
independent sorted structure.

30 [008] Parallel processing is desirable in database systems to improve their performance, when
processing data in a constituent table. In parallel processing, the data of a table is split into multiple
distinct (non-overlapping) parts, which are then processed independent of each other using
different CPUs, processes, threads, etc. as is well known in the arts.

[009] Challenges may be presented due to the independent nature of the sorted structures, in
35 combination with potentially overlapping keys. Aspects of the present disclosure are directed to
parallel processing in such database systems.

Brief Description of the Drawings

[010] Example embodiments of the present disclosure will be described with reference to the accompanying drawings briefly described below.

5 [011] Figure 1 is a block diagram illustrating an example environment (computing system) in which several aspects of the present disclosure can be implemented.

[012] Figure 2A is a block diagram of a database system that uses a LSM tree storage engine in one embodiment.

10 [013] Figure 2B is a block diagram illustrating the operation of the LSM tree storage engine in a database system in one embodiment.

[014] Figure 2C illustrates the manner in which parallel processing is desired to be implemented in a database system using a LSM tree storage engine in one embodiment.

15 [015] Figure 3 is a flow chart illustrating the manner in which parallel processing in database systems storing a table as key value pairs in multiple independent sorted structures with overlapping keys is facilitated according to aspects of the present disclosure.

[016] Figure 4A depicts the details of independent sorted structures storing key-value pairs and corresponding SSTable index in one embodiment.

[017] Figure 4B illustrates the manner in which key ranges are determined in one embodiment.

20 [018] Figure 5A is a block diagram depicting the manner in which dynamic generation of key ranges is implemented in a storage layer in a database system using LSM tree storage engine in one embodiment.

[019] Figure 5B illustrates the manner in which key ranges are processed by workers in one embodiment.

25 [020] Figure 6 is a flow chart illustrating the manner in which key ranges are determined according to an aspect of the present disclosure.

[021] Figures 7A-7B together illustrate the manner in which a binary heap is used to generate key ranges in one embodiment.

30 [022] Figure 8 is a block diagram illustrating the details of the digital processing system in which various aspects of the present disclosure are operative by execution of appropriate executable modules.

[023] In the drawings, similar reference numbers generally indicate identical, functionally similar, and/or structurally similar elements. The drawing in which an element first appears is indicated by the leftmost digit(s) in the corresponding reference number.

Detailed Description of the Embodiments of the Disclosure

[024] 1. Definitions

[025] Sorted String Table (“SSTable”) file – An independent sorted structure that stores key-value pairs belonging to a database user table, system table or index. The key-value pairs in each file are stored sorted by the keys in ascending order or descending order (hereinafter “sort order”). The data of a single table/ index is typically stored in one or more SSTable files.

[026] Data block – A portion of a SSTable file that stores a corresponding set of key-value pairs. In an embodiment, data blocks of a single SSTable file may store different number of key-value pairs, but are approximately of the same size in terms of amount of data, number of bytes, etc. It may be appreciated that blocks are ‘approximately’ of same size, for example, due to the size of the last key-value being of variable size.

[027] Data block index key – A key that can be used to identify the set of keys present in a data block according to a convention. The data block index key may be one of the keys such as a first key in the data block or a last key in the data block. Alternatively, data block index key of a current data block may be an arbitrary key that is between the last key of the previous data block and the first key of the current data block (when the keys are stored in ascending order).

[028] SSTable Index – An index that points to the physical location (e.g., byte offset and data block size) of each data block in a SSTable file, along with the data block index key identifying the data block. The index may be part of the SSTable file or may be stored in a separate index file.

[029] Scan of table – A database operation where values are read according to an order (ascending or descending) of the keys. A scan may be performed, for example, as part of processing of a database query or due to an internal requirement (e.g., batch operation) in the database.

[030] Start Scan Key – The start key of a scan of a table.

[031] End Scan Key – The end key of a scan of a table.

[032] Sequence of data block index keys – A set of data block index keys corresponding to the data blocks contained in an SSTable file. The sequence is sorted according to a sort order, in the described embodiments.

[033] Super sequence of data block index keys – A sorted set of data block index keys corresponding to the data blocks contained in one or more SSTable file(s) storing data of a table between a start scan key and an end scan key.

[034] Chunks of data – Sets of data blocks that have distinct (non-overlapping) keys.

[035] Worker – An entity (e.g., CPUs, processes, threads, etc.) that processes a chunk of data. Multiple workers operate in parallel to collectively perform a database operation such as a scan of

the table. Workers may be employed for any database operation where key-value pairs have to be processed in parallel.

[036] Interleaving – A technique in which workers are facilitated to work on different non-contiguous chunks of data.

5 [037] Interleave size – The desirable size of a chunk of data that can be assigned to a worker for processing. In the embodiments below, the interleave size is provided as a number of data blocks since each block is of approximately the same size.

[038] Key Range – A set of keys typically specified by a corresponding pair of boundary keys (start boundary key and end boundary key). The keys in the set between the boundary keys are referred to as intermediate keys.

10 [039] 2. Overview

[040] An aspect of the present disclosure facilitates parallel processing in database systems. In one embodiment, a database system stores data of a table as key-value pairs in a set of independent sorted structures, each independent sorted structure storing corresponding key-value pairs according to a sort order of the keys. Each independent sorted structure consists of a respective set of data blocks, each data block being associated with a corresponding data block index key that identifies a set of keys present in the data block. Upon identifying a start scan key, an end scan key, and an interleave size, the database system determines a next key range specifying a chunk of data approximately equal to the interleave size, with the next key range being contained in a super sequence of data block index keys corresponding to the multiple data blocks between the start scan key and the end scan key in the sort order. The database system sends the next key range to a set of workers for processing the values corresponding to the keys in the next key range. The database system performing the determining and the sending until (in one embodiment, all) the super sequence of data block index keys are identified as having been sent for processing by the set of workers.

25 [041] According to another aspect of the present disclosure, the next key range includes a start boundary key, a set of intermediate keys and an end boundary key, with the database system sending only the start boundary key and the end boundary key to the set of workers, while identifying that the start boundary key, the set of intermediate keys and the end boundary key have been sent for processing by the set of workers.

30 [042] According to one more aspect of the present disclosure, each data block is of approximately a same size, and as such the interleave size specifies a number of data blocks calculated based on the same size.

[043] According to yet another aspect of the present disclosure, the sort order (noted above) is one of ascending order and descending order. The database system associates key iterators

corresponding to each of the set of independent sorted structures, with each key iterator iteratively providing a next key in the sort order from a sequence of data block index keys corresponding to the data blocks contained in the associated independent sorted structure. The database system sets a counter to zero, finds a reference key iterator as a key iterator having a minimum or maximum
5 next key among the key iterators and increments the counter. If the counter equals the interleave size, the database system pushes the next key of the reference iterator as a boundary key into a range boundaries buffer and resets the counter to zero. The database system advances the next key of the reference iterator to a subsequent key in the sequence of data block index keys according to the sort order. The database system repeats the finding, the incrementing, the pushing and the
10 advancing until all the data block index keys in the super sequence of data block index keys are processed.

[044] According to an aspect of the present disclosure, the database system pushes the start scan key into the range boundaries buffer prior to finding the reference key iterator (noted above) and pushes the end scan key into the range boundaries buffer after repeating (noted above). Each
15 worker of the set of workers is designed to retrieve a pair of boundary keys from the range boundaries buffer to form the next key range and delete a first of the pair of boundary keys from the range boundaries buffer. As such, each worker independently processes the data corresponding to the keys in the next key range, with the set of workers processing the data of the table in parallel.

[045] According to an aspect of the present disclosure, the database system constructs a binary
20 heap of the key iterators (noted above), with the binary heap organizing the key iterators as a binary tree with a top key iterator at a root of the binary tree having the minimum or maximum next key among the next keys of the key iterators. As such, the above noted actions of finding is performed by selecting the top key iterator as the reference iterator. The action of advancing is performed by, if the subsequent key is valid, adjusting the binary heap to obtain the top key iterator and otherwise,
25 removing the top key iterator from the binary heap. The action of repeating is performed until the binary heap is empty.

[046] According to another aspect of the present disclosure, the database system receives a retrieval request and finds the (above noted) start scan key and end scan key based on the retrieval request. The database system provides a response to the retrieval request based on the result of
30 processing of the retrieval request by the set of workers.

[047] According to one more aspect of the present disclosure, the database system implements a Log-Structured Merge (LSM) tree designed to add the set of independent sorted structures in response to write requests, with each independent sorted structure being a Sorted String table (SSTable) file.

[048] Several aspects of the present disclosure are described below with reference to examples for illustration. However, one skilled in the relevant art will recognize that the disclosure can be practiced without one or more of the specific details or with other methods, components, materials and so forth. In other instances, well-known structures, materials, or operations are not shown in detail to avoid obscuring the features of the disclosure. Furthermore, the features/aspects described can be practiced in various combinations, though only some of the combinations are described herein for conciseness.

[049] **2. Example Environment**

[050] Figure 1 is a block diagram illustrating an example environment (computing system) in which several aspects of the present disclosure can be implemented. The block diagram is shown containing end user systems 110A-110Z (Z being any natural number), Internet 120, intranet 130, server systems 140A-140C, and database systems 180A-180B. The end user systems are collectively referred to as 110.

[051] Merely for illustration, only representative number/type of systems is shown in the Figure. Many environments often contain many more systems, both in number and type, depending on the purpose for which the environment is designed. Each system/device of Figure 1 is described below in further detail.

[052] Intranet 130 represents a network providing connectivity between server systems 140A-140C, and database systems 180A-180B all provided within an enterprise (shown with dotted boundaries). Internet 120 extends the connectivity of these (and other systems of the enterprise) with external systems such as end user systems 110A-110Z. Each of intranet 140 and Internet 120 may be implemented using protocols such as Transmission Control Protocol (TCP) and/or Internet Protocol (IP), well known in the relevant arts.

[053] In general, in TCP/IP environments, an IP packet is used as a basic unit of transport, with the source address being set to the IP address assigned to the source system from which the packet originates and the destination address set to the IP address of the destination system to which the packet is to be eventually delivered. An IP packet is said to be directed to a destination system when the destination IP address of the packet is set to the IP address of the destination system, such that the packet is eventually delivered to the destination system by networks 120 and 130.

When the packet contains content such as port numbers, which specifies the destination application, the packet may be said to be directed to such application as well. The destination system may be required to keep the corresponding port numbers available/open, and process the packets with the corresponding destination ports. Each of Internet 120 and intranet 130 may be implemented using any combination of wire-based or wireless mediums.

[054] Each of end user systems 110A-110X represents a system such as a personal computer,

workstation, mobile device, computing tablet etc., used by users to generate (user) requests directed to applications executing in server systems 140A-140C. The user requests may be generated using appropriate user interfaces (e.g., web pages provided by an application executing in the server, a native user interface provided by a portion of an application downloaded from the server, etc.). In general, an end user system requests an application for performing desired tasks and receives the corresponding responses (e.g., web pages) containing the results of performance of the requested tasks. The web pages/ responses may then be presented to the user/customer at end user systems 110 by client applications such as the browser.

[055] Each of server systems 140A-140C represents a server, such as a web/application server, constituted of appropriate hardware executing software applications capable of performing tasks requested by end-user systems 110. A server system receives a user request from an end-user system and performs the tasks requested in the user request. A server system may use data stored internally (for example, in a non-volatile storage/hard disk within the server system), external data (e.g., maintained in database system 180A-180B) and/or data received from external sources (e.g., received from a user) in performing the requested tasks. The server system then sends the result of performance of the tasks to the requesting end-user system (one of 110) as a corresponding response to the user request. The results may be accompanied by specific user interfaces (e.g., web pages) for displaying the results to a requesting user.

[056] Each of database systems 180A-180B facilitates storage and retrieval of a collection of data by software applications executing in other systems of the enterprise such as server systems 140A-140C. Each database system receives data requests from the software applications, processes the data requests, and sends the results of the data requests to the requesting software application/system. Each database system typically contains a server constituted of appropriate hardware and one or more non-volatile (persistent) storages. The server executes data processing applications designed to process the received data requests and to store and/or retrieve data from the persistent storages.

[057] Database system 180A represents a system that is implemented to support relational database technologies and accordingly provides storage and retrieval of data using structured queries such as SQL (Structured Query Language). As is well known, a relational database management system (RDBMS) organizes the data into multiple databases, each database containing one or more tables, with each table in turn containing one or more rows and columns. Examples of such relational database systems are PostgreSQL, MySQL, etc.

[058] Database system 180B represents a system that is implemented to store data of a table as key-value pairs in multiple data blocks organized as a set of independent sorted structures (e.g. SSTable files), with each data block storing values associated with a corresponding set of keys

sorted according to a sort order. In one embodiment, database system 180B implements a Log-Structured Merge (LSM) tree designed to add SSTable files in response to write requests received from end user systems 110. In the following disclosure, such a database system (180B) is referred to as using a LSM tree storage engine. An example of such a database system that uses LSM tree storage engine is YugaByte DB available from YugaByteDB, Inc

[059] As noted in the Background section, it may be desirable that database systems 180A-180B support parallel processing. Parallel processing may entail accepting data requests/queries from multiple software applications at the same time and processing each query independent of the others using one or more workers. Such parallelism may also be provided in the processing of a single query by decomposing the single query into parts (commonly referred to as a query plan) which are then processed by multiple workers in parallel.

[060] In normal relational database systems such as database system 180A, data is commonly stored sequentially in pages (blocks) numbered from 1 to N. Therefore, it is possible to distribute these blocks sequentially across workers/threads, scan them in parallel, and then merge the results into single sequential output.

[061] However, in a database system having LSM tree storage engine (such as 180B), there are several challenges to implementing parallel processing. The description is continued with an implementation of a database system using LSM tree storage engine, followed by the challenges to implementing parallel processing in such a database system.

[062] 3. Database System Using a LSM Tree Storage Engine

[063] Figure 2A is a block diagram of a database system that uses a LSM tree storage engine in one embodiment. Database system (DS) 200 corresponds to database system 180B noted above, and is shown containing query layer 210, storage layer 220 and disk/persistent storage 230 (which in turn is shown containing multiple SSTable files). Each of the blocks of the Figure is described in detail below.

[064] Query layer 210 receives (via path 185) requests from software applications (executing in server systems 140A-104D), and processes the received requests by interfacing with storage layer 220. Storage layer 220 provides various interfaces that facilitate storage and retrieval of data from the SSTable files maintained in disk 230. Each SSTable file is shown containing a sequence of key-value pairs, sorted by the keys according to a sort order (ascending order or descending order).

[065] Figure 2B is a block diagram illustrating the operation of the LSM tree storage engine in a database system (200) in one embodiment. The operation is shown being performed in storage layer 220 of database system 200, and illustrates the manner in which write and read requests to the storage are handled by the LSM tree storage engine.

[066] Requests containing writes (insert or updates to the data) 240 are received via path 185,

and first stored in memtable 250 maintained in a volatile random access memory (RAM). Upon the size of memtable 250 exceeding a pre-determined threshold (e.g., 54 MB), the data (based on writes 240) in memtable 250 is flushed/written (in a sorted manner) to a corresponding SSTable file on disk 230. Each SSTable file is stored with an associated index that has zero or more intermediate levels of index blocks, with the bottom most level (leaf-level) pointing to data blocks within the SSTable file. Flushing of memtable 250 entails clearing the memory of the previous writes, thereby enabling future writes (240) to be stored in memtable 250.

[067] Accordingly, multiple SSTable files 260 may be flushed/written to disk 230 at different time instances, with level 0 representing the most recent flush/writes and level N representing the oldest flush/writes. The maintenance of SSTable files at different levels enables the files to be handled differently. For example, compaction/merging of the files may be performed minimally at level 0 and gradually increased with increasing level, with level N files having the maximum compaction.

[068] Requests containing reads 270 are first directed to memtable 250. If the requested data is present in memtable 250, the data is returned as a response to the read 270. If the data is not present in memtable 250, data from one or more of SSTable files 260 are used for processing reads 270. It may be appreciated that the data required to be processed for a single read request may span multiple SSTable files 260.

[069] Figure 2C illustrates the manner in which parallel processing is desired to be implemented in a database system (200) using a LSM tree storage engine in one embodiment. The parallel processing is desired to be implemented in storage layer 220 of database system 200. Broadly, the key-value pairs stored in the multiple SSTable files are required to be organized into disjoint (non-overlapping keys) chunks 280 (1 to 5 shown in the Figure) of approximately the same size, and thereafter assigned for processing to one or more concurrent workers 290 (1 and 2 shown in the Figure).

[070] However, there are several challenges to the above noted implementation of parallel processing in database systems that use LSM tree storage engines (such as 200). The architecture and operation of the LSM tree implies that data is stored across multiple files with overlapping keys. Also, data associated with the same key may be spread across files (because of writes and deletes). These aspects do not allow the easy break-up of data into chunks which can be enumerated and accessed randomly by their numbers. Accordingly, there is a need to come up with a technique which allows data to be broken up into disjoint chunks for parallel processing (particularly for scan).

[071] Aspects of the present disclosure are directed to facilitating parallel processing in database systems (200) storing a table as key value pairs in multiple independent sorted structures (SSTable

files) with overlapping keys, as described below with examples.

[072] 4. Parallel Processing in Database Systems using LSM Tree Storage Engine

[073] Figure 3 is a flow chart illustrating the manner in which parallel processing in database systems storing a table as key value pairs in multiple independent sorted structures (SSTable files) with overlapping keys is facilitated according to aspects of the present disclosure. The flowchart is described with respect to the systems of Figures 1 and 2A-2C, merely for illustration. However, many of the features can be implemented in other environments also without departing from the scope and spirit of several aspects of the present invention, as will be apparent to one skilled in the relevant arts by reading the disclosure provided herein.

[074] In addition, some of the steps may be performed in a different sequence than that depicted below, as suited to the specific environment, as will be apparent to one skilled in the relevant arts. Many of such implementations are contemplated to be covered by several aspects of the present invention. The flow chart begins in step 301, in which control immediately passes to step 310.

[075] In step 310, database system 200 stores key-value pairs in independent sorted structures (e.g., SSTable files), each independent sorted structure containing (one or more) data blocks associated with respective data block index keys. Each independent sorted structure stores corresponding key-value pairs according to a sort order of the keys. As such, the data blocks of an independent sorted structure are also sorted such that a data block in the same independent sorted structure contains keys larger or lesser (depending on sort order) than the keys in a previous data block. Each data block is also associated with a corresponding data block index key that identifies a set of keys present in the data block.

[076] In step 320, database system 200 identifies a start scan key, and end scan key and an interleave size. The start scan key and end scan key represents the end points of the range of keys to be read during a scan. In the case of a full table scan, the start scan key and end scan key may be identified as -inf (infinity) and +inf respectively.

[077] The interleave size is the desirable size (e.g., amount of data, bytes) of a chunk of data (one of chunks 280) that can be assigned to a worker (one or more workers 290) for processing. In one embodiment, since each data block of the multiple data blocks is of approximately the same size, the interleave size specifies a number of data blocks calculated based on the same size.

[078] In step 340, database system 200 determines a next key range specifying a chunk of data approximately equal to the interleave size, the next key range being contained in a super sequence of data block index keys between the start scan key and the end scan key (in the sort order noted above). It should be appreciated that the super sequence is a merged view of all the data block index keys that is dynamically identified, and is not statically pre-generated and stored.

[079] The term “approximately equal” used herein implies that the amount of data in the chunk

is close to or equal to the interleave size. Practically approximate equality is obtained, for example, because of the variable size of the values (and thus the data block), in addition to duplication of keys in multiple independent sorted structures. As a result, a chunk of data containing 3-5 data blocks may be viewed as being approximately equal to an interleave size of 4 data blocks.

5 [080] In step 360, database system 200 sends the next key range to a set of workers (290) for processing the values corresponding to the keys in the next key range. According to an aspect, such sending entails pushing the next key range into a range boundaries buffer, with each worker of the set of workers being designed to retrieve the next key range from the range boundaries buffer and independently process the data corresponding to the keys in the next key range, such that the
10 set of workers processes the data in parallel.

[081] According to one more aspect, the next key range includes a start boundary key, a set of intermediate keys and an end boundary key, with database system 200 sending only the start boundary key and the end boundary key to the set of workers (instead of all the keys in the next key range).

15 [082] In step 380, database system 200 checks whether there are more data block index keys in the super sequence to be sent for processing. The checking may be performed based on whether all the keys in the super sequence have been covered by the previously sent next key ranges, any limit on the number of keys to be sent, etc. In the above aspect, when only the start and end boundary keys of the next key ranges are being sent, database system 200 may identify that the
20 start boundary key, the set of intermediate keys and the end boundary key have been sent for processing by the set of workers.

[083] If there are still keys to be sent, control passes to step 340, wherein a next key range is determined. It may be appreciated that the steps of determining (340) and sending (360) may be performed iteratively or concurrently until the keys in the super sequence have been identified as
25 having been sent for processing. If all the keys are identified to have been sent for processing, control passes to step 399, where the flowchart ends.

[084] Thus, parallel processing is facilitated in database system 200 storing a table as key value pairs in multiple independent sorted structures with overlapping keys. It may be appreciated that when the interleave size is number of data blocks (N), the above noted steps 340-380 may
30 alternatively viewed as sequentially determining every Nth key from the super sequence of data block index keys (limited by the start scan key and end scan key) and pushing the determined key to the range boundaries buffer.

[085] According to an aspect, upon receiving a retrieval request from one or end user systems 110, database system 200 identifies the above noted start scan key and end scan key based on the
35 retrieval request. For example, the start and end scan keys may be determined as the minimum and

maximum key of the range of keys matching one or more conditions specified in the retrieval request. Database system 200 then provides a response to the retrieval request based on the result of processing of the retrieval request by the set of workers. The manner in which database system 200 may provide several aspects of the present disclosure according to the flowchart of Figure 3 is described below with examples.

[086] 5. Illustrative Example

[087] Figures 4A-4C, 5, 6 and 7A-7B together illustrate the manner in which parallel processing in a database system (200) storing a table as key value pairs in multiple independent sorted structures with overlapping keys is facilitated in one embodiment. Each of the Figures is described in detail below.

[088] Figure 4A depicts the details of independent sorted structure storing key-value pairs and corresponding SSTable index in one embodiment. Specifically, the Figure shows the details of two SSTable files 410 and 420 (though database systems typically contain a large number of such SSTable files). For random access effectiveness, data inside each SSTable file is actually stored inside data blocks of approximately the same pre-configured size and each SSTable file has an index mapping keys to data blocks. In the Figure, data is shown stored in 17 data blocks – 9 data blocks in SSTable file 410 and 8 data blocks in SSTable file 420. It may be readily observed the SSTable files 410 and 420 have overlapping keys such as k1, k10, etc.

[089] Each SSTable file is shown associated with a corresponding SSTable index, that contains the data block index key of each data block along with the pointer to corresponding data block inside the SSTable file. Based on implementation, the Stable index may be stored as one block or as multiple blocks or even in multiple levels. In the following description, for illustration, the data block index key of each data block is the last key in the data block.

[090] SSTable index 415 is accordingly shown having the last keys in the 9 data blocks of SSTable file 410 along with their corresponding physical locations, while SSTable index 425 is shown having the last keys in the 8 data blocks of SSTable file 420 along with their corresponding physical locations. As the data block index keys are stored in the indexes of SSTable files, database system 200 may be designed to read the data block index keys from the SSTable indexes, and not from the data blocks in the SSTable files. As such, scanning through data blocks in the SSTable files can be avoided and only indexes need to be scanned which is faster because the indexes have less keys (one data block index key per data block).

[091] It may be appreciated that for implementing parallel processing, the table data shown in Figure 4A is required to be broken into disjoint chunks bounded by key ranges having approximately the same size. These chunks of data may thereafter be assigned to concurrent workers by passing the boundaries of the key ranges to the workers. Each worker may process data

belonging to key ranges assigned to the worker sequentially. In other words, a range of keys [start_scan_key, end_scan_key] (which can be the keys in the whole user/system table data or a narrower range) is required to be split into sequence of continuous disjoint key ranges:

[boundary_key_1, boundary_key_2];
 5 (boundary_key_2, boundary_key_3);
 ...;
 (boundary_key_{n-1}, boundary_key_n],

[092] where boundary_key_1 = start_scan_key, boundary_key_n = end_scan_key and size of data belonging to each key range is approximately the same. It should be noted that using a key
 10 range simplifies the process of communicating the assigned keys to each worker, as only the boundary keys need to be communicated.

[093] Since all data blocks have approximately the same size (B), the problem of breaking data into chunks of approximately the same size (X) could be further reduced to a problem of breaking data into chunks of approximately the same number of data blocks, that is interleave size $N = X /$
 15 B number of data blocks. In the following description, the interleave size N is assumed to be 4 data blocks for illustration. The manner in which key ranges are determined is described in detail below.

[094] Figure 4B illustrates the manner in which key ranges are determined in one embodiment. For illustration, the determination of the key ranges is described below with respect to the two data files and two indexes shown in Figure 4A. Accordingly to an aspect, the key ranges are determined
 20 based on the data block index keys associated with the data blocks.

[095] As such, data portions 430 and 435 indicate the data block index keys (here, last keys) for the data blocks contained in SSTable file 410 and 420 respectively. It may be noted that the data block index keys are in a sort order (here, ascending order). Data portion 440 represents a merged view (super sequence) of the data block index keys in the same sort order. For the interleave size
 25 $N = 4$ data blocks, the caret character “^” indicates the multiples of 4 that form the start and end boundaries of the key ranges.

[096] Data portion 460 illustrates the key ranges determined for the data blocks of SSTable files 410 and 420. The key ranges are shown as per the interval notation, well known in the arts. The start value in the first key range and the end value in the last key range is shown replaced with inf
 30 (infinity) to indicate a full table scan. Data portion 460 also indicates the number of data blocks that is covered by each key range. It may be observed that other than the last key range, the other key ranges are approximately equal to the interleave size.

[097] Portion 470 illustrates the specific portions of the data blocks in the SSTable files 410 and 420 that are specified by each key range. It may be readily observed that the approximation in the
 35 number of data blocks specified by each key range is due to the possibility of duplication of keys

across the multiple SSTables.

[098] Portion 480 illustrate the manner in which the key ranges are processed by two concurrent workers in one embodiment. It may be appreciated that worker 1 is shown processing key ranges 1, 3 and 5, while worker 2 is shown processing key ranges 2 and 4, that is, the processing is performed in an interleaved manner as is well known in the relevant arts. It should be noted that the goal is to utilize all workers as much as possible, as such the objective is to assign the next chunk of data to a worker by the time the worker has completed processing of a current chunk.

[099] Thus, aspects of the present disclosure facilitate parallel processing in database systems using LSM tree storage engine. It may be appreciated that it is generally not desirable to merge all SSTables indexes in advance and generate a super sequence of all required key ranges boundaries upfront. Instead, it may be desirable to have a technique of getting a limited amount of boundary keys (of key ranges) starting at a specified key upon request. In other words, the technique needs to iterate over a “merged” view of the various SST files indexes and dynamically generate the key ranges. The manner in which such dynamic generation of key ranges may be implemented is described below with examples.

[0100] 6. Dynamic Generation of Key Ranges

[0101] Figure 5A is a block diagram depicting the manner in which dynamic generation of key ranges is implemented in a storage layer (220) in a database system (200) using LSM tree storage engine in one embodiment. The block diagram is shown containing storage interface 510, request processor 530, range boundaries generator 550, range boundaries buffer 560 and workers 570. Each of the blocks of the Figure is described below.

[0102] Storage interface 510 facilitates various blocks operating in query layer 210 to interface/communicate with one or more operational blocks (not shown) executing in storage layer 220, thereby facilitating the retrieval of data (key-values pairs stored in SSTable files) from disk 230. It may be appreciated that memtables (250) shown in Figure 2B may contain key-value pairs and that processing of the data of a table may require that the memtable data be read as well. However, for purposes of determining key ranges for parallel processing, it may be sufficient to only take into account the data in SST files and omit memtables data (as assumed in the following description).

[0103] Request processor 530 receives retrieval requests specifying the desired data to be retrieved from SSTable files stored in disk 230. The retrieval requests may correspond to read requests received from end user systems 110 or may be generated internal to database system 200, for example, as part of processing a batch operation, as part of performance of maintenance/replication/backup of the database system, etc. In response to a retrieval request, request processor 530 first determines a range of keys (in particular, a start scan key and an end scan key) matching

the one or more conditions specified in the retrieval request, and then sends the range of keys to range boundaries generator 550, while forwarding the details of the retrieval request to one or more workers 570. It may be noted that the range of keys may correspond to the keys in all of the SSTable files storing the data of a single table in disk 230.

5 [0104] Range boundaries generator 550 receives a start scan key and an end scan key, determines one or more next key ranges contained in the received range of keys, and then pushes the one or more next key ranges to range boundaries buffer 560. According to an aspect, range boundaries generator 550 iteratively processes the range of keys and pushes boundary keys of the key ranges (specifically, the keys marked by “^” in data portion 440) to range boundaries buffer 560. Range
10 boundaries generator 550 also pushes the start scan key and end scan key into range boundaries buffer 560 as described below with respect to Figure 6.

[0105] Range boundaries buffer 560 represents a volatile storage such as a random access memory (RAM) that stores the next key ranges. In one embodiment, range boundaries buffer 560 is implemented as a queue data structure storing the boundary keys of the key ranges. Range
15 boundaries generator 550 adds to one end of the queue (pushes) the start scan key, new boundary keys, and the end scan key as described below with respect to Figure 6. Thus, range boundaries buffer 560 stores a sequence of keys, where two adjacent keys are boundaries of the next key range to be scanned by a worker.

[0106] Workers 570 represent execution entities that can operate independent (in term of CPU
20 usage, memory, etc.) of each other in the processing of different chunks of data (defined by the key ranges). Workers 570 may be implemented as processes executing on different CPUs, different threads, etc. as will be apparent to one skilled in the relevant arts by reading the disclosure herein. Each worker is designed to take a next key range from range boundaries buffer 560, one at a time. That basically means a worker reads the first two keys from the queue and sets them as the
25 boundary keys of the next key range. As soon as a key range is taken, the first key is deleted from the queue (that is range boundaries buffer 560), making room for more keys to be added by range boundaries generator 550.

[0107] It should be noted that the generation of the boundary keys happens concurrently with the scans, except for the very first, which has to be completed before parallel workers 570 can start.
30 As noted above, the queue also includes the boundaries imposed by the conditions (specified in the retrieval request) on the table primary key (missing conditions would mean -inf or +inf boundary) and limits on number of keys (for example, as specified by a LIMIT command in SQL), response size and max returned key size. Limits are based on the buffer size estimated by database system 200 for range boundaries buffer 560.

35 [0108] In the simplest case, all keys generated by range boundaries generator 550 fit into the

queue/buffer. If the next key does not fit, range boundaries generator 550 may wait until workers 570 take some keys, making more room, or discard the rest of the keys in the response. Subsequent requests are made with the lower boundary equal to the last boundary key stored in the buffer. As such, if part of the response is discarded, it may effectively be re-fetched. Database system 200 may readjust the limits on the subsequent requests.

[0109] Again, it is noted that even if part of key-value pairs is stored inside memory in addition to SSTables, range boundaries generator 550, for purposes of simplicity, can ignore the in-memory data since it is not needed for correctness. Every key in the SSTable index covers one data block in its SSTable file. But it should be noted that this can be an approximation in the merged view of the index. For example (k20, k40] in the example above may represent 4 adjacent keys, but they could cover anywhere from 4 to 4*numOfSSTableFiles data blocks if both files have those same data block index keys in their indexes due to overwrites. Probabilistically speaking, the chance of this happening is low. However, this may not be a big issue to solve even if it happens.

[0110] Figure 5B illustrates the manner in which key ranges are processed by workers in one embodiment. For illustration, it is assumed that two concurrent workers (worker 1 and worker 2) process the data chunks determined by the key ranges shown in data portion 460 of Figure 4B. Specifically, 580 illustrates the operation of the two workers in parallel processing the data blocks corresponding to different key ranges retrieved from range boundaries buffer 560. It may be appreciated that the waiting for getting the next M key ranges could be avoided if the key ranges are requested in advance, as illustrated by 585.

[0111] Thus, dynamic generation of key ranges implemented in query layer 210 facilitates parallel processing of retrieval requests received by database system 200. The manner in which range boundaries generator 550 determines the next key ranges to be processed by workers 570 is described below with examples.

[0112] 7. Range Boundaries Generator

[0113] Figure 6 is a flow chart illustrating the manner in which key ranges are determined according to an aspect of the present disclosure. The flowchart is described with respect to the systems of Figure 5, merely for illustration. However, many of the features can be implemented in other environments also without departing from the scope and spirit of several aspects of the present invention, as will be apparent to one skilled in the relevant arts by reading the disclosure provided herein.

[0114] In addition, some of the steps may be performed in a different sequence than that depicted below, as suited to the specific environment, as will be apparent to one skilled in the relevant arts. Many of such implementations are contemplated to be covered by several aspects of the present invention. The flow chart begins in step 601, in which control immediately passes to step 605.

[0115] In step 605, range boundaries generator 550 pushes the start scan key into a buffer (range boundaries buffer 560). In the above noted embodiment, pushing may entail adding the start scan key to the queue data structure. The start scan key may be pushed into the buffer prior to finding the reference iterator, that is, prior to step 630.

5 [0116] In step 610, range boundaries generator 550 associates iterators to the independent sorted structures (SSTables) storing the data for the table sought to be scanned. Each key iterator is designed to iteratively provide a next key in the sort order from a sequence of data block index keys corresponding to the data blocks contained in the independent sorted structure. For example, an iterator 1 associated with SSTable file 410 is designed to iteratively provide the data block index keys shown in data portion 430, while another iterator 2 associated with SSTable file 420 is designed to iteratively provide the data block index keys shown in data portion 435. It should be noted that the next value for iterator 1 is k1, while the next value for iterator 2 is k10.

[0117] In step 615, range boundaries generator 550 sets a counter to zero (0). In step 620, range boundaries generator 550 sets an iterator corresponding to a SSTable file as a reference iterator.

15 The iterator associated with any of the SSTable files may be set as the reference iterator. The description is continued assuming that iterator 1 is set as the reference iterator.

[0118] In step 630, range boundaries generator 550 checks whether there exists another iterator whose next key is less than (for ascending order, greater than for descending order) the next key of the reference iterator. If such another iterator exists, control passes to step 640.

20 [0119] In step 640 range boundaries generator 550 sets the reference iterator to another iterator, and control passes back to step 630. The operation of steps 630 and 640 results in the finding of a reference key iterator that has a minimum next key (when less than) or a maximum next key (when greater than) among all the key iterators. In other words, the next key of the reference iterator would be the next data block index key according to the sort order. In the example noted above, another iterator can only be iterator 2, and accordingly range boundaries generator 550 determines that no such iterator exists since the next value (k10) of iterator 2 is not less than the next value (k1) of the reference iterator. If such another iterator does not exist, control passes to step 650.

25 [0120] In step 650, range boundaries generator 550 increments the counter (adds one to a current value of the counter). It may be appreciated that the operation of steps 630, 640 and 650 ensures that the data block index keys in the reference iterator (here, k1, k5) are continued to be counted until another iterator having a next key < the next key of the reference iterator is found (which happens only when the next key in reference iterator is k75). Once another iterator is set as the reference iterator, keys (k10, k20) from another iterator are counted, until yet another iterator is found. By such an operation, all the keys in the super sequence of keys noted above are sent for processing in the sort order.

[0121] In step 660, range boundaries generator 550 checks whether the counter is equal to the interleave size (in the above example, $N = 4$). Control passes to step 670 if they are equal and to step 680 otherwise. In step 670, range boundaries generator 550 pushes (into the range boundaries buffer 560) the next key (here, k_{20}) of the reference iterator (2) as a boundary key, resets counter to zero and control passes to step 680.

[0122] In step 680, range boundaries generator 550 advances the next key of the reference iterator, specifically, to a subsequent key in the sequence of data block index keys (of the reference SSTable file) according to the sort order. It should be noted that when the next key is the final key in the sequence, the advancing of the next key causes the next key to be set to a large value (e.g., infinity). This ensures that the reference iterator is replaced by any other iterator during further processing of the keys.

[0123] In step 690, range boundaries generator 550 checks whether there are more data block index keys to process. A simple check of whether all iterators have reached the end (that is set to the large value noted above) can be used for such determination (that is, there are more keys to process if there is at least one iterator that has not ended). If there exists at least one key that has yet to be processed, control passes to step 630 for processing of the data block index keys using the corresponding iterators. If all data block index keys (of the table) have been processed, control passes to step 695.

[0124] In step 695, range boundaries generator 550 pushes the end scan key into the buffer (range boundaries buffer 560) after steps 630 through 690 have been repeated until all the data block index keys have been sent for processing. Control then passes to step 699, where the flowchart ends.

[0125] Thus, range boundaries generator 550 implemented as part of database system 200 generates key ranges for processing by multiple workers (570), thereby facilitating parallel processing in database system 200 storing a table as key value pairs in multiple independent sorted structures with overlapping keys. According to an aspect, the above noted steps of Figure 6 may be implemented using a binary heap as described below with examples.

[0126] 8. Binary Heap Based Range Generation

[0127] Figures 7A-7B together illustrate the manner in which a binary heap is used to generate key ranges in one embodiment. Merging iterator implementation based on binary heap is an already existing algorithm used for regular data scan in sequential order over a merged view of SSTable files. It is based on a binary heap. As is well known, a binary heap organizes the key iterators as a binary tree with a top key iterator at a root of the binary tree having the minimum or maximum next key among the next keys of the key iterators.

[0128] In one embodiment described below, the binary heap is considered as a black box and only

the following functions/methods of the heap application programming interface (API) are used:

- push() - add iterator to the heap
- top() - get top iterator (which is positioned to the earliest key across all iterators)
- pop() - remove top iterator from the heap
- 5 • replace_top() – letting the heap know that the top iterator has been moved, so that the heap can update itself if needed
- is_empty() - check whether heap is empty

[0129] In order to sequentially scan all data key-value pairs, an iterator over each SSTable file data is added into a binary heap via push() function. In other words, a data-LSM-heap is constructed. The following algorithm is then used to scan through all key-value pairs:

- A. Get top() iterator from the heap
- B. Process key-value pair from this iterator.
- C. Advance this iterator to the next key.
- D. Check whether the iterator is still valid or the end of the associated SSTable has been reached.
- 15 a. If iterator is still valid - call replace_top() on the heap
- b. If iterator is no longer valid (no more data in the associated SSTable) - call pop() on the heap
- E. If the heap is empty - All key-value pairs have been processed, otherwise go to step A.

20 [0130] Thus, a binary heap is used to process the key-value pairs stored in multiple SSTable files in a desired sort order (in the above scenario, ascending order). It may be appreciated that in a similar way, a heap to merge-scan just the index keys across all the SSTable files (as if the SSTable files had no data blocks) may also be performed. The difference with the data-LSM-heap is that the iterators over each SSTable index (instead of SSTable data) are added into the heap. Such a heap may be called index-LSM-heap.

25 [0131] An index-LSM-heap can be used to get index keys in sequential order upon request starting from the specified key. Simply, by using every N-th of these index keys as boundaries for key ranges, data chunks of approximately the same size can be obtained which can thereafter be processed in parallel. The description is continued with the operation of an index-LSM-heap for illustration.

30 [0132] Referring to Figure 7A, portion 710 depicts the initial index-LSM-heap containing the iterators corresponding to SSTable index 415 and 425 noted above. It is assumed here that key ranges are generated from the first key, not some specific key in the middle (as would be in the case of a retrieval request). Portion 720 depicts the heap after the data block index key k1 (current key of the top iterator) has been processed (step B above) and the top iterator has been advanced

to the next key (step C above). It may be appreciated that at step B, as described above, a counter may be used to keep track of the number of keys processed, and push the key as a boundary key whenever the counter = interleave size.

[0133] Portion 730 depicts the heap after invocation of `replace_top()` (steps D.a) in view of the top iterator still being valid and thereafter performance of steps A, B and C noted above. It should be noted that the function `replace_top()` does nothing since $k5 \leq k10$. At step B, $k5$ is processed, and at step C, the top iterator is advanced to the next key $k75$. Portion 740 depicts the heap after the invocation to `replace_top()` updates the heap since $k75 > k10$. Then, $k10$ is processed (now it is SSTable index 425 iterator) and the top iterator is advanced to the next key.

[0134] Referring to Figure 7B, portion 750 depicts the heap after the invocation to `replace_top()`, which does nothing since $k20 \leq k75$. The steps are repeated the same way and the sequence of data block index keys $k20, k30, k31, k32, k40, k60$ are processed from the same SSTable index 425 iterator and it remains the top iterator. Portion 760 depicts the heap after $k60$ is processed and SSTable index 425 iterator is advanced to the next key. The function `replace_top()` does nothing since $k70 \leq k75$. Then, $k70$ is processed, and the top iterator is advanced to the next index key and SSTable index 425 iterator becomes invalid due to running out of data block index keys. Accordingly, the top iterator is removed from the heap. The heap now contains only SSTable index 415 iterator as shown in portion 770. Then $k75$ is processed from this iterator, advancing it to the next key, getting $k100, k125, k130, k1000, k2000, k2500$ until the SSTable index 415 iterator also runs out of keys.

[0135] It should be further appreciated that the features described above can be implemented in various embodiments as a desired combination of one or more of hardware, software, and firmware. The description is continued with respect to an embodiment in which various features are operative when the software instructions described above are executed.

[0136] 9. Digital Processing System

[0137] Figure 8 is a block diagram illustrating the details of digital processing system (800) in which various aspects of the present disclosure are operative by execution of appropriate executable modules. Digital processing system 800 may correspond to database system 180B/200.

[0138] Digital processing system 800 may contain one or more processors such as a central processing unit (CPU) 810, random access memory (RAM) 820, secondary memory 830, graphics controller 860, display unit 870, network interface 880, and input interface 890. All the components except display unit 870 may communicate with each other over communication path 850, which may contain several buses as is well known in the relevant arts. The components of Figure 8 are described below in further detail.

[0139] CPU 810 may execute instructions stored in RAM 820 to provide several features of the

present disclosure. CPU 810 may contain multiple processing units, with each processing unit potentially being designed for a specific task. Alternatively, CPU 810 may contain only a single general-purpose processing unit.

[0140] RAM 820 may receive instructions from secondary memory 830 using communication path 850. RAM 820 is shown currently containing software instructions constituting shared environment 825 and/or other user programs 826 (such as other applications, DBMS, etc.). In addition to shared environment 825, RAM 820 may contain other software programs such as device drivers, virtual machines, etc., which provide a (common) run time environment for execution of other/user programs.

[0141] Graphics controller 860 generates display signals (e.g., in RGB format) to display unit 870 based on data/instructions received from CPU 810. Display unit 870 contains a display screen to display the images defined by the display signals. Input interface 890 may correspond to a keyboard and a pointing device (e.g., touch-pad, mouse) and may be used to provide inputs. Network interface 880 provides connectivity to a network (e.g., using Internet Protocol), and may be used to communicate with other systems connected to the networks.

[0142] Secondary memory 830 may contain hard drive 835, flash memory 836, and removable storage drive 837. Secondary memory 830 may store the data (e.g., data portions shown in Figures 4A-4C) and software instructions (e.g., for performing the actions of Figures 3 and 6, for implementing the blocks of Figure 5), which enable digital processing system 800 to provide several features in accordance with the present disclosure. The code/instructions stored in secondary memory 830 may either be copied to RAM 820 prior to execution by CPU 810 for higher execution speeds, or may be directly executed by CPU 810.

[0143] Some or all of the data and instructions may be provided on removable storage unit 840, and the data and instructions may be read and provided by removable storage drive 837 to CPU 810. Removable storage unit 840 may be implemented using medium and storage format compatible with removable storage drive 837 such that removable storage drive 837 can read the data and instructions. Thus, removable storage unit 840 includes a computer readable (storage) medium having stored therein computer software and/or data. However, the computer (or machine, in general) readable medium can be in other forms (e.g., non-removable, random access, etc.).

[0144] In this document, the term "computer program product" is used to generally refer to removable storage unit 840 or hard disk installed in hard drive 835. These computer program products are means for providing software to digital processing system 800. CPU 810 may retrieve the software instructions, and execute the instructions to provide various features of the present disclosure described above.

[0145] The term "storage media/medium" as used herein refers to any non-transitory media that

store data and/or instructions that cause a machine to operate in a specific fashion. Such storage media may comprise non-volatile media and/or volatile media. Non-volatile media includes, for example, optical disks, magnetic disks, or solid-state drives, such as storage memory 830. Volatile media includes dynamic memory, such as RAM 820. Common forms of storage media include, for example, a floppy disk, a flexible disk, hard disk, solid-state drive, magnetic tape, or any other magnetic data storage medium, a CD-ROM, any other optical data storage medium, any physical medium with patterns of holes, a RAM, a PROM, and EPROM, a FLASH-EPROM, NVRAM, any other memory chip or cartridge.

[0146] Storage media is distinct from but may be used in conjunction with transmission media.

Transmission media participates in transferring information between storage media. For example, transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise bus 850. Transmission media can also take the form of acoustic or light waves, such as those generated during radio-wave and infra-red data communications.

[0147] Reference throughout this specification to “one embodiment”, “an embodiment”, or similar language means that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one embodiment of the present disclosure. Thus, appearances of the phrases “in one embodiment”, “in an embodiment” and similar language throughout this specification may, but do not necessarily, all refer to the same embodiment.

[0148] Furthermore, the described features, structures, or characteristics of the disclosure may be combined in any suitable manner in one or more embodiments. In the above description, numerous specific details are provided such as examples of programming, software modules, user selections, network transactions, database queries, database structures, hardware modules, hardware circuits, hardware chips, etc., to provide a thorough understanding of embodiments of the disclosure.

[0149] 10. Conclusion

[0150] While various embodiments of the present disclosure have been described above, it should be understood that they have been presented by way of example only, and not limitation. Thus, the breadth and scope of the present disclosure should not be limited by any of the above-described exemplary embodiments, but should be defined only in accordance with the following claims and their equivalents.

[0151] It should be understood that the figures and/or screen shots illustrated in the attachments highlighting the functionality and advantages of the present disclosure are presented for example purposes only. The present disclosure is sufficiently flexible and configurable, such that it may be utilized in ways other than that shown in the accompanying figures.

[0152] Further, the purpose of the following Abstract is to enable the Patent Office and the public generally, and especially the scientists, engineers and practitioners in the art who are not familiar

with patent or legal terms or phraseology, to determine quickly from a cursory inspection the nature and essence of the technical disclosure of the application. The Abstract is not intended to be limiting as to the scope of the present disclosure in any way.

What is claimed is:

1. A method performed in a database system, said method comprising:

storing data of a table as key-value pairs in a set of independent sorted structures, wherein each independent sorted structure stores corresponding key-value pairs according to a sort order of the keys, wherein each independent sorted structure consists of a respective set of data blocks of a plurality of data blocks, each data block being associated with a corresponding data block index key that identifies a set of keys present in the data block;

identifying a start scan key, an end scan key, and an interleave size;

determining a next key range specifying a chunk of data approximately equal to the interleave size, wherein said next key range is contained in a super sequence of data block index keys corresponding to said plurality of data blocks between said start scan key and said end scan key in said sort order;

sending said next key range to a set of workers for processing the values corresponding to the keys in said next key range; and

performing said determining and said sending until said super sequence of data block index keys are identified as having been sent for processing by said set of workers.

2. The method of claim 1, wherein said next key range comprises a start boundary key, a set of intermediate keys and an end boundary key,

wherein said sending sends only said start boundary key and said end boundary key to said set of workers,

wherein said performing identifies that said start boundary key, said set of intermediate keys and said end boundary key have been sent for processing by said set of workers.

3. The method of claim 2, wherein each data block of said plurality of data blocks is of approximately the same size, wherein said interleave size specifies a number of data blocks calculated based on said same size.

4. The method of claim 3, wherein said sort order is one of ascending order and descending order, wherein said performing comprises:

associating key iterators corresponding to each of said set of independent sorted structures, wherein each key iterator iteratively provides a next key in said sort order from a sequence of data block index keys corresponding to the data blocks contained in the associated independent sorted structure;

setting a counter to zero;

finding a reference key iterator as a key iterator having a minimum or maximum next key among said key iterators;

incrementing said counter;

5 if said counter equals said interleave size, pushing said next key of said reference iterator as a boundary key into a range boundaries buffer and resetting said counter to zero;

advancing said next key of said reference iterator to a subsequent key in said sequence of data block index keys according to said sort order; and

repeating said finding, said incrementing, said pushing and said advancing until all the data block index keys in said super sequence of data block index keys are processed.

10

5. The method of claim 4, further comprising pushing said start scan key into said range boundaries buffer prior to said finding and pushing said end scan key into said range boundaries buffer after said repeating,

15 wherein each worker of said set of workers is designed to retrieve a pair of boundary keys from said range boundaries buffer to form said next key range and delete a first of said pair of boundary keys from said range boundaries buffer,

wherein each worker independently processes the data corresponding to the keys in said next key range, wherein said set of workers processes said data in parallel.

20

6. The method of claim 4, wherein said performing further comprises:

constructing a binary heap of said key iterators, wherein said binary heap organizes said key iterators as a binary tree with a top key iterator at a root of said binary tree having the minimum or maximum next key among the next keys of said key iterators,

wherein said finding comprises selecting said top key iterator as said reference iterator,

25 wherein said advancing comprises, if said subsequent key is valid, adjusting said binary heap to obtain said top key iterator and otherwise, removing said top key iterator from said binary heap,

wherein said repeating is performed until said binary heap is empty.

30

7. The method of claim 1, further comprising:

receiving a retrieval request;

finding said start scan key and said end scan key based on said retrieval request; and

providing a response to said retrieval request based on a result of processing of said retrieval request by said set of workers.

35

8. The method of claim 7, wherein said database system implements a Log-Structured Merge (LSM) tree designed to add said set of independent sorted structures in response to write requests, wherein each independent sorted structure is a Sorted String table (SSTable) file.

5 9. A non-transitory machine-readable medium storing one or more sequences of instructions, wherein execution of said one or more instructions by one or more processors contained in a database system causes said database system to perform the actions of:

storing data of a table as key-value pairs in a set of independent sorted structures, wherein each independent sorted structure stores corresponding key-value pairs according to a sort order of the keys, wherein each independent sorted structure consists of a respective set of data blocks of a plurality of data blocks, each data block being associated with a corresponding data block index key that identifies a set of keys present in the data block;

identifying a start scan key, an end scan key, and an interleave size;

15 determining a next key range specifying a chunk of data approximately equal to the interleave size, wherein said next key range is contained in a super sequence of data block index keys corresponding to said plurality of data blocks between said start scan key and said end scan key in said sort order;

sending said next key range to a set of workers for processing the values corresponding to the keys in said next key range; and

20 performing said determining and said sending until said super sequence of data block index keys are identified as having been sent for processing by said set of workers.

10. The non-transitory machine-readable medium of claim 9, wherein said next key range comprises a start boundary key, a set of intermediate keys and an end boundary key,

25 wherein said sending sends only said start boundary key and said end boundary key to said set of workers,

wherein said performing identifies that said start boundary key, said set of intermediate keys and said end boundary key have been sent for processing by said set of workers.

30 11. The non-transitory machine-readable medium of claim 10, wherein each data block of said plurality of data blocks is of approximately the same size, wherein said interleave size specifies a number of data blocks calculated based on said same size.

12. The non-transitory machine-readable medium of claim 3, wherein said sort order is one of ascending order and descending order, wherein said performing comprises one or more instructions for:

5 associating key iterators corresponding to each of said set of independent sorted structures, wherein each key iterator iteratively provides a next key in said sort order from a sequence of data block index keys corresponding to the data blocks contained in the associated independent sorted structure;

setting a counter to zero;

10 finding a reference key iterator as a key iterator having a minimum or maximum next key among said key iterators;

incrementing said counter;

if said counter equals said interleave size, pushing said next key of said reference iterator as a boundary key into a range boundaries buffer and resetting said counter to zero;

15 advancing said next key of said reference iterator to a subsequent key in said sequence of data block index keys according to said sort order; and

repeating said finding, said incrementing, said pushing and said advancing until all the data block index keys in said super sequence of data block index keys are processed.

13. The non-transitory machine-readable medium of claim 12, further comprising one or more instructions for pushing said start scan key into said range boundaries buffer prior to said finding and pushing said end scan key into said range boundaries buffer after said repeating,

wherein each worker of said set of workers is designed to retrieve a pair of boundary keys from said range boundaries buffer to form said next key range and delete a first of said pair of boundary keys from said range boundaries buffer,

25 wherein each worker independently processes the data corresponding to the keys in said next key range, wherein said set of workers processes said data in parallel.

14. The non-transitory machine-readable medium of claim 12, wherein said performing further comprises one or more instructions for:

30 constructing a binary heap of said key iterators, wherein said binary heap organizes said key iterators as a binary tree with a top key iterator at a root of said binary tree having the minimum or maximum next key among the next keys of said key iterators,

wherein said finding comprises selecting said top key iterator as said reference iterator,

wherein said advancing comprises, if said subsequent key is valid, adjusting said binary heap to obtain said top key iterator and otherwise, removing said top key iterator from said binary heap,

wherein said repeating is performed until said binary heap is empty.

5

15. A database system comprising:

one or more processors; and

a random access memory (RAM) to store instructions, wherein said one or more processors retrieve said instructions and execute said instructions, wherein execution of said instructions

10 causes said database system to perform the actions of:

storing data of a table as key-value pairs in a set of independent sorted structures, wherein each independent sorted structure stores corresponding key-value pairs according to a sort order of the keys, wherein each independent sorted structure consists of a respective set of data blocks of a plurality of data blocks, each data block being associated with a corresponding data block index key that identifies a set of keys present in the data block;

15

identifying a start scan key, an end scan key, and an interleave size;

determining a next key range specifying a chunk of data approximately equal to the interleave size, wherein said next key range is contained in a super sequence of data block index keys corresponding to said plurality of data blocks between said start scan key and said end scan key in said sort order;

20

sending said next key range to a set of workers for processing the values corresponding to the keys in said next key range; and

performing said determining and said sending until said super sequence of data block index keys are identified as having been sent for processing by said set of workers.

25

16. The database system of claim 15, wherein said next key range comprises a start boundary key, a set of intermediate keys and an end boundary key,

wherein said sending sends only said start boundary key and said end boundary key to said set of workers,

30

wherein said performing identifies that said start boundary key, said set of intermediate keys and said end boundary key have been sent for processing by said set of workers.

17. The database system of claim 16, wherein each data block of said plurality of data blocks is of approximately the same size, wherein said interleave size specifies a number of data blocks calculated based on said same size.

5 18. The database system of claim 17, wherein said sort order is one of ascending order and descending order, wherein for said performing, said database system performs the actions of:

associating key iterators corresponding to each of said set of independent sorted structures, wherein each key iterator iteratively provides a next key in said sort order from a sequence of data block index keys corresponding to the data blocks contained in the associated independent sorted structure;

10 setting a counter to zero;

finding a reference key iterator as a key iterator having a minimum or maximum next key among said key iterators;

incrementing said counter;

15 if said counter equals said interleave size, pushing said next key of said reference iterator as a boundary key into a range boundaries buffer and resetting said counter to zero;

advancing said next key of said reference iterator to a subsequent key in said sequence of data block index keys according to said sort order; and

20 repeating said finding, said incrementing, said pushing and said advancing until all the data block index keys in said super sequence of data block index keys are processed.

19. The database system of claim 18, further performing the actions of pushing said start scan key into said range boundaries buffer prior to said finding and pushing said end scan key into said range boundaries buffer after said repeating,

25 wherein each worker of said set of workers is designed to retrieve a pair of boundary keys from said range boundaries buffer to form said next key range and delete a first of said pair of boundary keys from said range boundaries buffer,

wherein each worker independently processes the data corresponding to the keys in said next key range, wherein said set of workers processes said data in parallel.

30

20. The database system of claim 18, wherein for said performing, said database system further performs the actions of:

constructing a binary heap of said key iterators, wherein said binary heap organizes said key iterators as a binary tree with a top key iterator at a root of said binary tree having the minimum or maximum next key among the next keys of said key iterators,

35

wherein said finding comprises selecting said top key iterator as said reference iterator,
wherein said advancing comprises, if said subsequent key is valid, adjusting said binary
heap to obtain said top key iterator and otherwise, removing said top key iterator from said binary
heap,
5 wherein said repeating is performed until said binary heap is empty.