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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
26 May 2017 (26.05.2017)

WIPO | PCT

(10) International Publication Number
WO 2017/086987 A1

(51) International Patent Classification:
G06F 12/08 (2006.01)

(21) International Application Number:
PCT/US2015/061843

(22) International Filing Date:
20 November 2015 (20.11.2015)

(25) Filing Language: English

(26) Publication Language: English

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

(72) Inventors: **LI, Jun**; 1501 Page Mill Road, Palo Alto, California 94304-1100 (US). **VOLOS, Haris**; 1501 Page Mill Road, Palo Alto, California 94304-1100 (US).

(74) Agents: **HARTMANN II, Kenneth R.** et al.; Hewlett Packard Enterprise, 3404 E. Harmony Road, Mail Stop 79, 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, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

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

Declarations under Rule 4.17:

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

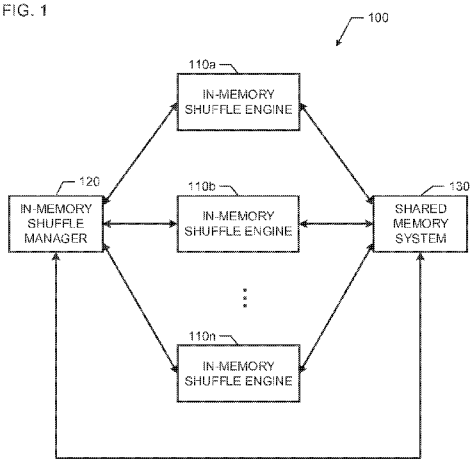
Published:

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

WO 2017/086987 A1

(54) Title: IN-MEMORY DATA SHUFFLING

FIG. 1



(57) Abstract: Examples herein involve in-memory data shuffling. In examples herein, in-memory shuffle engines use a mapper to sort shuffle data into buckets of a shared memory and an index to map the sorted shuffle data to locations within the shared memory. The in-memory shuffle engines further use reducers to retrieve location information from the index and merge the sorted shuffle data using merge engines. Examples herein may provide processed shuffle data that is sorted/merged based on key/value pairs of the shuffle data.

IN-MEMORY DATA SHUFFLING

BACKGROUND

[0001] Data shuffling involves processing data based on key/value pairs. The shuffle data is sorted by a source (e.g., a mapper or data producer) and merged at a destination (e.g., a reducer or data consumer). Accordingly, the shuffle data including a plurality of keys and corresponding values, may be processed to provide data that is organized based on the keys.

BRIEF DESCRIPTION OF THE DRAWINGS

[0002] FIG. 1 illustrates an example in-memory data shuffle system, including in-memory shuffle engines that may be implemented in accordance with an aspect of this disclosure.

[0003] FIG. 2 is a block diagram of an example in-memory shuffle engine that may be implemented by the in-memory data shuffle system of FIG. 1.

[0004] FIG. 3 illustrates an example representation of shuffle data being sorted/merged within the in-memory data shuffle system of FIG. 1 utilizing in-memory mappers and in-memory reducers of in-memory shuffle engines that may be implemented by the in-memory shuffle engine of FIG. 2.

[0005] FIG. 4 is a flowchart representative of example machine readable instructions that may be executed to implement the in-memory shuffle engine of FIG. 2.

[0006] FIG. 5 is a flowchart representative of example machine readable instructions that may be executed to implement the in-memory data shuffle system of FIG. 1.

[0007] FIG. 6 is a block diagram of an example processor platform capable of executing the instructions of FIGS. F1 and/or F2 to implement the A1 of FIG. B.

[0008] Wherever possible, the same reference numbers will be used throughout the drawing(s) and accompanying written description to refer to the same or like parts.

DETAILED DESCRIPTION

[0009] Examples disclosed herein involve an in-memory data shuffle system to shuffle data within a shared memory fabric, such as a configuration dynamic random access memory (DRAM) and/or a non-volatile memory (NVM), of a multi-compute node system. In examples herein, in-memory mappers of in-memory shuffle engines partition and index shuffle data in a shared memory region while in-memory reducers of the in-memory shuffle engines retrieve and reduce the shuffle data using the index buckets that map locations of the buckets in the shared-memory.

[0010] In data shuffling technologies, a plurality of mappers may process data based on key/value pairs (e.g., organize based on keys) and reducers may merge the processed data (retrieve all data having a certain key). Accordingly, when the reducers are to merge the shuffle data, the reducers may have to contact all mappers to retrieve data having a certain key, which may cause great delay and processing resources. This may be compounded in systems utilizing disk storage, solid-state drive (SSD) storage, etc. to shuffle the data as communication protocols (e.g., Transmission Control Protocol/Internet Protocol) in such systems introduce a lot of overhead, including the overhead from an operating system of the systems.

[0011] An example shared memory fabric (e.g., a configuration of dynamic random access memory (DRAM) or a non-volatile memory (NVM)) of a multi-compute node system provides relatively low latency and high bandwidth in comparison to other storage technologies (e.g., disk, solid-state drive (SSD), etc.) used in combination with TCP/IP communication. In examples herein, in-memory shuffle engines utilize shared memory to shuffle data between nodes (e.g., in-memory shuffle engines) of the multi-compute node system via direct memory accesses. Accordingly, shuffle data may be shuffled from sources (e.g., mappers) to destinations (e.g., reducers) using relatively low-latency, high

bandwidth communication via the direct memory access to increase speed and efficiency of shuffling data in the example in-memory shuffle systems. Direct memory access between in-memory mappers and in-memory reducers is possible as the in-memory mappers and the in-memory reducers are located within a same load/store domain (e.g., a collection of processors or computing devices that can access a same collection of fabric attached memory via load/store processor operations).

[0012] Examples herein involve in-memory data shuffling. In examples herein, in-memory shuffle engines use a mapper to sort shuffle data into buckets of a shared memory and an index to map the sorted shuffle data to locations within the shared memory. As used herein, a bucket or data bucket is any block, portion, or memory space of a memory that stores data. The in-memory shuffle engines further use reducers to retrieve bucket location information from index buckets in the shared memory region and process mapped shuffle data using (e.g., via merge engines, pass through, a hash-map, etc.). Examples herein may provide processed shuffle data that is partitioned/sorted/merged based on key/value pairs of the shuffle data.

[0013] FIG. 1 is a schematic diagram of an example in-memory data shuffle system 100 including *n* in-memory shuffle engines 110(a), 110(b),...,110(n) (which may be referred to herein collectively as the in-memory shuffle engines 110 or individually as an example in-memory shuffle engine 110) that may be implemented in accordance with an aspect of this disclosure, where *n* represents the number of in-memory shuffle engines 110 in the in-memory shuffle system 100. The in-memory data shuffle system 100 of FIG. 1 includes an in-memory shuffle manager 120, the *n* in-memory shuffle engines 110, and a shared memory system 130. In examples herein, the in-memory shuffle engines 110 may shuffle data within a shared memory region of the shared memory system 130 that is accessible by any or all of the in-memory shuffle engines 110. The example shared memory system 130 may include a memory manager and memory or storage. For example, the example shared memory system 130 may include a shared non-volatile memory (NVM), a

shared non-volatile storage, a shared dynamic random access memory (DRAM), etc.

[0014] The in-memory shuffle manager 120 manages the in-memory shuffle engines 110 of the in-memory data shuffle system 100 of FIG. 1. The in-memory shuffle manager 120 (e.g., a shuffle scheduler) may provide shuffle data (e.g., from a processing pipeline) to the in-memory shuffle engines 110 for shuffling. For example, the in-memory shuffle manager 120 may include an interface (e.g., a remote procedure call interface) to receive data and/or instructions to process data. The example in-memory shuffle manager 120 may maintain communication between the in-memory shuffle engines 110. Such communication may include facilitating exchanges of index buckets indicating locations of buckets including mapped shuffle data stored in a shared memory region of the shared memory system 130, as further described below. The example in-memory shuffle manager 120 provides global pointers to index buckets in a shared NVM (or shared DRAM) of the shared memory system 130. Accordingly, the in-memory shuffle manager 120 may serve as a master node of the in-memory shuffle engines 110 to facilitate shuffling of shuffle data between in-memory mappers of in-memory shuffle engines and in-memory reducers of the in-memory shuffle engines.

[0015] The in-memory shuffle engines 110 of FIG. 1 process shuffle data (e.g., key/value pairs) in a shared memory region of the shared memory system 130 in accordance with the teachings of this disclosure. The in-memory shuffle engines 110 may be implemented by virtual machine(s) (which may be implemented by hardware or hardware and executable instructions), processor(s), etc. In examples herein, the in-memory shuffle engines 110 shuffle data by redistributing the data from a source stage (e.g., a map stage) to a destination stage (e.g., a reduce stage). For example, the in-memory shuffle engines 110 may support a variety of shuffle operations, such as *groupBy* (to group together values sharing a same key or key family), *reduceBy* (to apply a reduce function on grouped values sharing a same key or key family), *partitionBy* (to move keys into different partitions), and *sortBy* (to sort keys with global ordering), etc. The example in-memory shuffle engines 110 may partition

the shuffle data and/or sort the shuffle data and store the sorted shuffle data into buckets of a shared memory region (e.g., of an NVM or DRAM) based on key/value pairs within the shuffle data. An example implementation of an in-memory shuffle engine that may be used to implement the in-memory shuffle engines 110 of FIG. 1 is disclosed below in connection with FIG. 2.

[0016] Example buckets of the shared memory region of a shared memory of the shared memory system 130 may be allocated via a memory manager of the shared memory system 130 (e.g., in response to requests from the in-memory shuffle engines 110). The memory manager of the shared memory system 130 may be implemented by a memory broker (MB) of a shared-memory system 130. The example memory manager may include an allocation interface (e.g., a malloc/free interface) that is called by the in-memory shuffle engines 110 to allocate shared memory to the in-memory shuffle engines 110. The example memory manager may implement any suitable allocation schemes, offsets, address translations (e.g., virtual to physical, etc.) to enable the in-memory shuffle engines 110 to access the shared memory region of the shared memory system 130.

[0017] In examples herein, the memory manager of the shared memory system 130 may partition a shared memory region into fixed sized zones. During shuffling or processing of shuffle data, a zone may be acquired/allocated (e.g., via the in-memory shuffle manager 120). In some examples, the zones may be allocated to support locality (e.g., via a *libnuma* operation). In other words, the zones may be allocated such that their proximity to a location of a corresponding in-memory shuffle engine 110 using the zone is relatively lessened (e.g., minimized, lowered, etc.). In examples herein, after processing of the shuffle data (e.g., after a shuffle stage), blocks of the zones may be cleared (e.g., via a single bulk free operation) to enable use of the block for future/subsequent processes/shuffling.

[0018] FIG. 2 is a block diagram of an example in-memory shuffle engine 200, which may be used to implement the in-memory shuffle engines 110 of FIG. 1. The example in-memory shuffle engine 200 includes an in-memory mapper 210 with an indexer 212, a partitioner 214, and a map operator 216,

and an in-memory reducer 220 with an index retriever 222 and a reduce operator 224. In examples herein, in-memory mappers, similar to the in-memory mapper 210, of the in-memory shuffle engines 110 of FIG. 1 may partition and/or sort shuffle data from a processing pipeline and in-memory reducers, similar to the in-memory reducer 220, may sort-based merge, pass through, or hash-map based merge shuffle data from in-memory mappers 210 of the in-memory shuffle engines 110 of FIG. 1 in accordance with the teachings of this disclosure. For the sake of readability, the following refers to a single example of the in-memory shuffle engine of FIG. 2, though all or some of the in-memory shuffle engines 110 of FIG. 1 may operate in the same or similar manner as the in-memory shuffle engine 110 of FIG. 2.

[0019] The example in-memory mapper 210 of FIG. 2 includes an indexer 212, a partitioner 214, and map operator 216. The in-memory mapper 210 may also include a buffer (not shown) to receive shuffle data (e.g., key/value pairs). The example buffer may be implemented by a dynamic random access memory (DRAM) within or in communication with the in-memory data shuffle system 100 of FIG. 1. As such, shuffle data having a designated type or identifier (e.g., *integers, longs, floats, doubles, strings, byte[]*, etc.) may be written directly to the buffer without serialization. In some examples, the in-memory mapper 210 may utilize separate data structures for certain types of keys (e.g., *integers, longs, floats, strings*, etc.) of the shuffle data to enable suitable sorting of the different types.

[0020] In examples herein, the in-memory mapper 210 receives shuffle data (e.g., via a buffer of the in-memory mapper 210) from a processing pipeline (or a stage of a processing pipeline) the in-memory data shuffle system 100 of FIG. 1. In examples herein, the partitioner 214 encodes a partitioning strategy on received key/value pairs based on a value of the key to load shuffle data into data buckets of the shared memory the key/value pairs are to be assigned. In examples herein, each of the in-memory mappers 210 may map data to plurality of data buckets in a same or similar manner as one another, based on a partitioning strategy implemented by the partitioner 214, such that similar data buckets (e.g., data buckets having a same name or identifier) receive similar

key/value pairs based on the keys. The in-memory mappers 210 may maintain an index bucket that maps the locations of the buckets in the shared memory regions of the in-memory mappers 210. For example, one partition strategy is a hash partitioning, which is implemented as follows:

$$p = \text{hash(key)}/N \quad (1)$$

where p is a partition number (e.g., an identifier of the bucket), hash() is a hash function that converts the key value into an integer, and N is a number of the in-memory reducers 220 for the shuffle.

[0021] In some examples, the map operator 216 of the in-memory mapper 210 sorts the received shuffle data and store shuffle data in the data buckets, for example, in order of the keys of the key/value pairs in the buckets. In some examples (for non-ordered operations such as groupBy, reduceBy, and partitionBy, etc.), the map operator 216 may pass the partitioned key/value pairs, such that the in-memory reducers may have access to the buckets without ordering of the key/value pairs within the buckets.

[0022] The indexer 212 may provide map status information of the in-memory mapper 210 corresponding to the mapped shuffle data to the in-memory shuffle manager 120 of FIG. 1. For example, the indexer 212 may provide a map identifier and a global pointer to the corresponding index bucket of the in-memory mapper 210 in which the partitioner 214 filled the received key/value pairs. The in-memory shuffle manager 120 may then gather the map statuses of all in-memory mappers 210 of the in-memory data shuffle system 100, and pass the map status information to the in-memory reducers 220 of the in-memory shuffle data system 100. The in-memory shuffle manager 120 may provide the map status information via a global pointer to shuffle data of each in-memory mapper 210. The global pointer may include an offset to the bucket index and a region identifier of the shared memory region of the in-memory mapper 210. Accordingly, the in-memory shuffle manager 120 may facilitate or handle communication (e.g., on a dedicated channel) of the index information and/or provide such index information to in-memory reducers of the in-memory shuffle engines 110 of FIG. 1 (e.g., via a global pointer).

[0023] As mentioned above, the indexer 212 provides a map identifier and a global pointer to the corresponding index bucket of the in-memory mapper 210. Each bucket of the in-memory mapper 210 may be represented as a <start offset, size> entry in the index bucket. In examples herein, when the size of the bucket is 0, the data bucket may be considered empty.

[0024] The example in-memory reducer 220 in the example of FIG. 2 includes an index retriever 222 and a reduce operator 224. The example in-memory reducer 220 may also include a buffer to receive shuffle data (e.g., sorted key/value pairs) for processing. The example buffer of the in-memory reducer 220 may be implemented by a DRAM (e.g., the same DRAM or a different DRAM that implements the buffer of the in-memory mapper 210) of the in-memory data shuffle system 100, though the buffer of the in-memory reducer 220 may be separate from a buffer of the in-memory mapper 210. In examples herein, the in-memory reducer 220 receives/retrieves shuffle data (e.g., via a buffer of the in-memory reducer 220) that has been sorted by in-memory mappers of the in-memory shuffle engines 110 of FIG. 1 (which may include shuffle data sorted by the in-memory mapper 210 of FIG. 2). The in-memory reducers 220 of the in-memory shuffle engines 110 merge data based on keys of the shuffle data (e.g., via a priority queue). For example, the reduce operator 224 (e.g., a sort-based merge engine such as a priority queue, a hash-map based merge engine, or a direct pass-through engine) of the in-memory reducer 220 may merge or pass mapped shuffle data from a particular bucket.

[0025] The index retriever 222 of FIG. 2 may retrieve/receive index information (e.g., global pointers to the index buckets of the in-memory mappers 210 of the in-memory shuffle engines 110 of FIG. 1) from the in-memory shuffle manager 120 of FIG. 1 that includes location information of buckets. The in-memory reducer 220 may retrieve shuffle data based on the index information from the in-memory shuffle manager 120. Using the index information, the in-memory reducer 220 retrieves data from non-empty buckets of the shared memory region (e.g., the buckets that did not receive sorted data from the in-memory mapper 210). As such, the in-memory reducer 220 may not necessarily retrieve data from each in-memory mapper of the in-memory data

shuffle system 100 of FIG. 1 nor spend time attempting to fetch data from empty buckets. The processed keys of the shuffle data from the in-memory reducer 220 and corresponding values (e.g., values in a buffer of the in-memory reducer 220) may then be provided to in-memory shuffle engine 110 of FIG. 1 as output to go back to a processing pipeline (e.g., for a subsequent stage of the processing pipeline).

[0026] In some examples, if the in-memory reducer 220 determines that the shuffle data is located in a same load/store domain as the in-memory reducer 220, the in-memory reducer 220 may directly access the buckets via a direct memory access (e.g., to merge keys, merge values of keys, etc.). Accordingly, the in-memory shuffle engine 200 may utilize a zero-copy technique to efficiently process shuffle data in accordance with the teachings of this disclosure.

[0027] Accordingly, in examples herein, a plurality of in-memory shuffle engines 200 may communicatively work together (e.g., via the in-memory shuffle manager 110) to process shuffle data using shared memory of the in-memory data shuffle system 100.

[0028] While an example manner of implementing the in-memory shuffle engines 110 of FIG. 1 is illustrated by the example in-memory shuffle engine 200 of FIG. 2, at least one of the elements, processes and/or devices illustrated in FIG. 2 may be combined, divided, re-arranged, omitted, eliminated and/or implemented in any other way. Further, the in-memory mapper, including the indexer 212, the partitioner 214, and the map operator 216, or the in-memory reducer 220, including the index retriever 222 and the reduce operator 224, and/or more generally, the example in-memory shuffle engine 200 of FIG. 2 may be implemented by hardware and/or any combination of hardware and executable instructions (e.g., software and/or firmware). Thus, for example, any of the in-memory mapper, including the indexer 212, the partitioner 214, and the map operator 216, or the in-memory reducer 220, including the index retriever 222 and the reduce operator 224, and/or more generally, the example in-memory shuffle engine 200 could be implemented by at least one of an analog or digital circuit, a logic circuit, a programmable processor, an application

specific integrated circuit (ASIC), a programmable logic device (PLD) and/or a field programmable logic device (FPLD). When reading any of the apparatus or system claims of this patent to cover a purely software and/or firmware implementation, at least one of the in-memory mapper 210, the indexer 212, the partitioner 214, the map operator 216, the in-memory reducer 220, the index retriever 222, or the reduce operator 224 is/are hereby expressly defined to include a tangible machine readable storage device or storage disk such as a memory, a digital versatile disk (DVD), a compact disk (CD), a Blu-ray disk, etc. storing the executable instructions. Further still, the example in-memory shuffle engine 200 of FIG. 2 may include at least one element, process, and/or device in addition to, or instead of, those illustrated in FIG. 2, and/or may include more than one of any or all of the illustrated elements, processes and devices.

[0029] FIG. 3 illustrates an example representation of shuffle data being processed within the in-memory data shuffle system 100 of FIG. 1. In the illustrated example of FIG. 3, a plurality of in-memory mappers 310a, 310b, 310c, 310d (which may be collectively referred to as in-memory mappers 310) and in-memory reducers 320a, 320b, 320c (which may be collectively referred to as in-memory reducers 320) process shuffle data. The example in-memory mappers 310 and in-memory reducers 320 of FIG. 3 may be implemented by the in-memory mapper 210 and in-memory reducer 220 of FIG. 2, respectively. Additionally, the in-memory mappers 310 and in-memory reducers 320 of FIG. 3 may be managed and/or controlled via an in-memory shuffle manager (such as the in-memory shuffle manager 120 of FIG. 1). In examples herein, any one of the in-memory mappers 310 may be included within a same in-memory shuffle engine (e.g., the in-memory shuffle engine 200 of FIG. 2) as any one of the in-memory reducers 320. In other words, the in-memory mapper 310a may be included within a same in-memory shuffle engine 110 as the in-memory reducer 320a or the in-memory mapper 310a may be included within a same in-memory shuffle engine 110 as the in-memory reducer 320b, etc.

[0030] In the illustrated example of FIG. 3, the in-memory mappers 310 and in-memory reducers 320 shuffle data via a shared memory region 330. The example shared memory region 330 may be implemented by a non-volatile

memory fabric (e.g., a memristor array, a phase-change memory, etc.) and/or a dynamic random access memory. In the illustrated example of FIG. 3, shuffle data is represented in data buckets with the bucket identifiers being the numerals 0, 1, and 2. For example, the 0, 1, and 2 of the shuffle data may represent data buckets including keys resulting from a partitioning strategy (e.g., such as the strategy of Equation (1) above) of key/value pairs in the shuffle data. The example shuffle data may be received by the in-memory mappers 310 in a random order (e.g., as received in a processing pipeline) via a buffer of the in-memory mappers 310. It is noted that the shuffle data received by the in-memory mappers 310 may also include additional or alternative data than the data represented by of the data buckets 0, 1, and 2 in FIG. 3.

[0031] The example in-memory mappers 310 may sort the shuffle data and store the sorted shuffle data into buckets of the shared memory region 330 and provide index information corresponding to addresses/locations of the buckets within the shared memory region 330 to a master node or shuffling manager in communication with the in-memory mappers 310 (e.g., the in-memory shuffle manager 120 of FIG. 1). For example, key/value pairs may be sorted and stored into buckets of the shared memory region 330 based on the key resulting from a partitioning strategy. For example, using the above strategy of Equation 1, a first bucket includes keys having a remainder 0, a second bucket includes keys having a remainder 1, and a third bucket includes keys having a remainder 2 .

[0032] The example shuffled data is received/retrieved by the in-memory reducers 320 and merged as shown (all data buckets labelled with the identifier of 0 merged by the in-memory reducer 320a, all data buckets labelled with the identifier of 1 merged by the in-memory reducer 320b, and all data buckets labeled with the identifier of 2 merged by the in-memory reducer 320c). In examples herein, the in-memory reducers 320 may receive/retrieve index information (e.g., a global pointer to an index bucket of the in-memory mappers 310) corresponding to the bucket locations (e.g., a beginning address of a bucket in the shared memory region 330 and merge corresponding values from buffers of the in-memory reducers 320. For example, the in-memory reducers

320 may refer to an index (e.g., index buckets of the in-memory mappers 310 in the shared memory region 330) to determine start locations (e.g., a beginning address) or start offsets of the data buckets 0, 1, 2. The in-memory reducers 320 may then retrieve the shuffle data from the data buckets for processing. For example, reduce operators of the in-memory reducers 320 may load the shuffle data into a merge engine (e.g., a priority queue), may perform a hash-map based merge of the shuffle data, or a pass-through of the shuffle data to the processing pipeline. The reducer operator pulls corresponding values corresponding to the keys of the shuffle data from a buffer of the in-memory reducers 320. The processing may then pull the processed shuffle data for the next stage.

[0033] As illustrated in FIG. 3, the in-memory mappers 310 and in-memory reducers 320 work together to shuffle data using the shared memory region 330. Accordingly, the in-memory reducers 320 may directly access data sorted by the in-memory mappers 310 within the shared memory region 330 without copying data, thus reducing processing overhead/resources.

[0034] A flowchart representative of example machine readable instructions for implementing the in-memory shuffle engine 200 of FIG. 2 is shown in FIG. 4. In this example, the machine readable instructions comprise a program/process for execution by a processor such as the processor 612 shown in the example processor platform 600 discussed below in connection with FIG. 6. The program/process may be embodied in executable instructions (e.g., software) stored on a tangible machine readable storage medium such as a CD-ROM, a floppy disk, a hard drive, a digital versatile disk (DVD), a Blu-ray disk, or a memory associated with the processor 612, but the entire program/process and/or parts thereof could alternatively be executed by a device other than the processor 612 and/or embodied in firmware or dedicated hardware. Further, although the example program is described with reference to the flowchart illustrated in FIG. 4, many other methods of implementing the example in-memory shuffle engine 200 may alternatively be used. For example, the order of execution of the blocks may be changed, and/or some of the blocks described may be changed, eliminated, or combined.

[0035] The process 400 of FIG. 4 begins with an initiation of the in-memory shuffle engine 200 or a plurality of in-memory shuffle engines 200 (e.g., upon startup, upon instructions from a user, upon startup of a system implementing the in-memory shuffle engine 200 (e.g., the in-memory data shuffle system 100), etc.). The example process 400 of FIG. 4 may be executed to implement a single in-memory shuffle engine 200 or the plurality of in-memory shuffle engines 100. For example, some blocks may be executed to implement a first shuffle engine and some blocks may be executed to implement a second shuffle engine in accordance with the examples herein. Furthermore, the example process 400 may be iteratively executed by a single or multiple in-memory shuffle engines to process shuffle data until all of shuffle data in a set of shuffle data is processed. In some examples, if shuffle data is missing from data buckets identified by the memory manager 120 via the index buckets, the in-memory shuffle engines 110 may notify the in-memory shuffle manager 120, which may instruct in-memory mappers 210 associated with the missing shuffle data to re-execute mapping of the shuffle data.

[0036] At block 410 of FIG. 4, the partitioner 214 of an in-memory mapper 210 partitions shuffle data into buckets of a shared memory. For example, at block 410, the in-memory mapper 210 may distribute the shuffle data based on keys of key/value pairs, such that the key/value pairs are allocated to the buckets so that each bucket comprises a same partition number (e.g., a same remainder from Equation 1 above) from the shuffle data. At block 420, the indexer 212 of the in-memory mapper 210 indexes location information of the sorted shuffle data in an index. The example indexer 212 may index beginning addresses of buckets of the in-memory mapper 210 in the shared memory region.

[0037] At block 430, the index retriever 222 of an in-memory reducer 220 of an in-memory shuffle engine 200 (which may be the same in-memory shuffle engine 200 as in block 410 or a different in-memory shuffle engine 200 than that in block 410) determines a location in the shared memory region of a portion of the shuffle data based on the location information in the index. For example, the portion of the shuffle data may refer to shuffle data having a certain key of

key/value pairs. Accordingly, the index retriever 222 may identify the keys in the index which maps the keys to corresponding locations of buckets in the shared memory region. Using the determined location, the in-memory reducer 220 may access the portion of the shuffle data from the determined location of the shared memory. For example, the in-memory reducer 220 may access the shuffle data by first retrieving the keys from the buckets to the reduce operator and then loading corresponding values of the keys from the bucket into a buffer of the in-memory reducer 220. Accordingly, after block 440, the shuffle data from the reduce operator 224 and the buffer of the in-memory reducer may be output to the processing pipeline of the next stage. After block 440, the example process 400 ends.

[0038] A flowchart representative of example machine readable instructions for implementing the in-memory data shuffle system 100 of FIG. 1 is shown in FIG. 5. In this example, the machine readable instructions comprise a program/process for execution by a processor such as the processor 612 shown in the example processor platform 600 discussed below in connection with FIG. 6. The program/process may be embodied in executable instructions (e.g., software) stored on a tangible machine readable storage medium such as a CD-ROM, a floppy disk, a hard drive, a digital versatile disk (DVD), a Blu-ray disk, or a memory associated with the processor 612, but the entire program/process and/or parts thereof could alternatively be executed by a device other than the processor 612 and/or embodied in firmware or dedicated hardware. Further, although the example program is described with reference to the flowchart illustrated in FIG. 5, many other methods of implementing the example in-memory data shuffle system 100 may alternatively be used. For example, the order of execution of the blocks may be changed, and/or some of the blocks described may be changed, eliminated, or combined.

[0039] The process 500 of FIG. 5 begins with an initiation of the in-memory data shuffle system 100 (e.g., upon startup, upon instructions from a user, upon startup of a device implementing the in-memory data shuffle system 100 (e.g., a server, a computer, etc.), etc.). At block 510, the shared memory system 130 (e.g., via a memory broker or memory manager), allocates a shared

region of a non-volatile memory for data shuffling, which is accessible to the plurality of in-memory shuffle engines 110.

[0040] At block 520, the in-memory shuffle manager 120 provides shuffle data to the plurality of shuffle engines 110. At block 520, the shuffle engines are to process the shuffle data using a shared memory region of a shared memory (e.g., a DRAM, a NVM, etc.) and an index. For example, at block 520, in-memory mappers (e.g., similar to the in-memory mapper 210 of FIG. 2) of the shuffle engines are to sort shuffle data and store the sorted shuffle data into the shared memory region of the shared memory system 130 and index location information in an index (e.g., an index bucket of the shared memory system 130), and in-memory reducers (e.g., similar to the in-memory reducer 220 of FIG. 2) retrieve and merge the sorted data using the index.

[0041] At block 530, the processed shuffle data based on key/value pairs of the shuffle data is outputted to the processing pipeline in the next stage. The example processed shuffle data from the in-memory manager 120 may include shuffle data from the plurality of in-memory shuffle engines. After block 530, the example process 500 ends.

[0042] As mentioned above, the example processes of FIGS. 4 and 5 may be implemented using coded instructions (e.g., computer and/or machine readable instructions) stored on a tangible machine readable storage medium such as a hard disk drive, a flash memory, a read-only memory (ROM), a compact disk (CD), a digital versatile disk (DVD), a cache, a random-access memory (RAM) and/or any other storage device or storage disk in which information is stored for any duration (e.g., for extended time periods, permanently, for brief instances, for temporarily buffering, and/or for caching of the information). As used herein, the term tangible machine readable storage medium is expressly defined to include any type of machine readable storage device and/or storage disk and to exclude propagating signals and to exclude transmission media. As used herein, "computer readable storage medium" and "machine readable storage medium" are used interchangeably. Additionally or alternatively, the example processes of FIGS. 4 and 5 may be implemented using coded instructions (e.g., computer and/or machine readable instructions)

stored on a non-transitory computer and/or machine readable medium such as a hard disk drive, a flash memory, a read-only memory, a compact disk, a digital versatile disk, a cache, a random-access memory and/or any other storage device or storage disk in which information is stored for any duration (e.g., for extended time periods, permanently, for brief instances, for temporarily buffering, and/or for caching of the information). As used herein, the term non-transitory machine readable medium is expressly defined to include any type of machine readable storage device and/or storage disk and to exclude propagating signals and to exclude transmission media. As used herein, when the phrase "at least" is used as the transition term in a preamble of a claim, it is open-ended in the same manner as the term "comprising" is open ended. As used herein the term “a” or “an” may mean “at least one,” and therefore, “a” or “an” do not necessarily limit a particular element to a single element when used to describe the element. As used herein, when the term “or” is used in a series, it is not, unless otherwise indicated, considered an “exclusive or.”

[0043] FIG. 6 is a block diagram of an example processor platform 600 capable of executing the instructions of FIGS. 4 and/or 5 to implement the shuffle engine 200 of FIG. 2 and/or the in-memory data shuffle system 100 of FIG. 1. The example processor platform 600 may be any type of apparatus or may be included in any type of apparatus, such as a server, a personal computer, or any other type of computing device.

[0044] The processor platform 600 of the illustrated example of FIG. 6 includes a processor 612. The processor 612 of the illustrated example is hardware. For example, the processor 612 can be implemented by at least one integrated circuit, logic circuit, microprocessor or controller from any desired family or manufacturer.

[0045] The processor 612 of the illustrated example includes a local memory 613 (e.g., a cache). The processor 612 of the illustrated example is in communication a volatile memory 614 and a non-volatile memory 616 via a bus 618. The volatile memory 614 may be implemented by Synchronous Dynamic Random Access Memory (SDRAM), Dynamic Random Access Memory (DRAM), RAMBUS Dynamic Random Access Memory (RDRAM) and/or any

other type of random access memory device. The non-volatile memory 616 may be implemented by flash memory, a memristor memory fabric, and/or any other desired type of fast non-volatile memory device. Access to the main memory 614, 616 may be controlled by a memory controller.

[0046] The processor platform 600 of the illustrated example also includes an interface circuit 620. The interface circuit 620 may be implemented by any type of interface standard, such as an Ethernet interface, a universal serial bus (USB), and/or a peripheral component interconnect (PCI) express interface.

[0047] In the illustrated example, at least one input device 622 is connected to the interface circuit 620. The input device(s) 622 permit(s) a user to enter data and commands into the processor 612. The input device(s) may be implemented by, for example, an audio sensor, a microphone, a camera (still or video), a keyboard, a button, a mouse, a touchscreen, a track-pad, a trackball, and/or a voice recognition system.

[0048] At least one output device 624 is also connected to the interface circuit 620 of the illustrated example. The output device(s) 624 may be implemented, for example, by display devices (e.g., a light emitting diode (LED), an organic light emitting diode (OLED), a liquid crystal display, a cathode ray tube display (CRT), a touchscreen, a tactile output device, a light emitting diode (LED), a printer and/or speakers). The interface circuit 620 of the illustrated example, thus, may include a graphics driver card, a graphics driver chip or a graphics driver processor.

[0049] The interface circuit 620 of the illustrated example also includes a communication device such as a transmitter, a receiver, a transceiver, a modem and/or network interface card to facilitate exchange of data with external machines (e.g., computing devices of any kind) via a network 626 (e.g., an Ethernet connection, a digital subscriber line (DSL), a telephone line, coaxial cable, a cellular telephone system, etc.).

[0050] The processor platform 600 of the illustrated example also includes at least one mass storage device 628 for storing executable instructions (e.g., software) and/or data. Examples of such mass storage

device(s) 628 include floppy disk drives, hard drive disks, compact disk drives, Blu-ray disk drives, RAID systems, and digital versatile disk (DVD) drives.

[0051] The coded instructions 632 of FIGS. 4 and/or 5 may be stored in the mass storage device 628, in the local memory 613 in the volatile memory 614, in the non-volatile memory 616, and/or on a removable tangible machine readable storage medium such as a CD or DVD.

[0052] From the foregoing, it is to be appreciated that the above disclosed methods, apparatus and articles of manufacture provide an in-memory data shuffle system that uses a shared memory fabric, such as a non-volatile memory, and an index to sort/merge shuffle data. In examples herein, in-memory mappers of shuffle engines and in-memory reducers of the shuffle engines have access to a same shared memory location and can therefore directly access the same data from the shared memory fabric rather than copying from separate storage locations, retrieving shuffle data via TCP/IP, etc.

[0053] Although certain example methods, apparatus and articles of manufacture have been disclosed herein, the scope of coverage of this patent is not limited thereto. On the contrary, this patent covers all methods, apparatus and articles of manufacture fairly falling within the scope of the claims of this patent.

CLAIMS

What is claimed is:

1. A method comprising:

partitioning shuffle data via a mapper of a first shuffle engine into buckets of a shared memory;

indexing location information mapping the partitioned shuffle data in an index;

determining a location of a portion of the shuffle data in the shared memory based on the location information in the index; and

accessing, via a reducer of a second shuffle engine, the portion of the shuffle data from the determined location of the shared memory.

2. The method as defined in claim 1, wherein the shuffle data comprises key/value pairs and accessing the portion of the shuffle data further comprises:

merging keys of the key/value pairs via a priority queue and loading values of the corresponding keys in a buffer of a reducer.

3. The method as defined in claim 2, further comprising:

outputting the merged keys from the merge engine and the corresponding values from the buffer to a processing pipeline.

4. The method as defined in claim 1, wherein the index maps locations of buckets in the shared memory, the buckets comprising key/value pairs corresponding to bucket identifiers of the buckets.

5. The method as defined in claim 1, wherein the first shuffle engine is a different shuffle engine than the second shuffle engine.

6. The method as defined in claim 1, wherein the first shuffle engine comprises the second shuffle engine.

7. A system comprising:

an in-memory mapper of an in-memory shuffle engine to:

partition first shuffle data into a plurality of buckets stored in a shared memory based on key/value pairs of the shuffle data, and

index the first shuffle data and corresponding location information of the buckets stored in the shared memory in an index bucket; and

an in-memory reducer of the in-memory shuffler node to:

retrieve second shuffle data comprising a first key from a bucket of the plurality of buckets stored in the shared memory based on the index bucket, and

reduce the second shuffle data based on key/value pairs of the second shuffle data.

8. The system of claim 7, wherein the shared memory comprises a non-volatile memory.

9. The system of claim 7, wherein the shared memory comprises a dynamic random access memory (DRAM).

10. The system of claim 7, further comprising an in-memory shuffle manager, the in-memory mapper to provide the location information via a global pointer to a corresponding index bucket of the in-memory mapper.

11. The system of claim 10, wherein the in-memory shuffle manager is to provide a global pointer to the in-memory reducer indicating a location of the index bucket.

12. A non-transitory machine readable storage medium comprising instructions that, when executed, cause a machine to at least:

allocate a shared region of a shared memory for data shuffling, the shared region accessible by a plurality of shuffle engines;

provide shuffle data to the plurality of shuffle engines for processing, the shuffle engines to process the shuffle data using the shared region of the shared memory and an index, the index to map the shuffle data to location information of the shared memory region; and

output the processed shuffle data based on key/value pairs of the shuffle data.

13. The non-transitory machine readable storage medium of claim 12, wherein the index maps the shuffle data to location information by indicating a location of buckets comprising the shuffle data, the buckets comprising key/value pairs.

14. The non-transitory machine readable storage medium of claim 13, wherein the key/value pairs are assigned to the buckets based on a partitioning strategy of in-memory mappers of the plurality of shuffle engines.

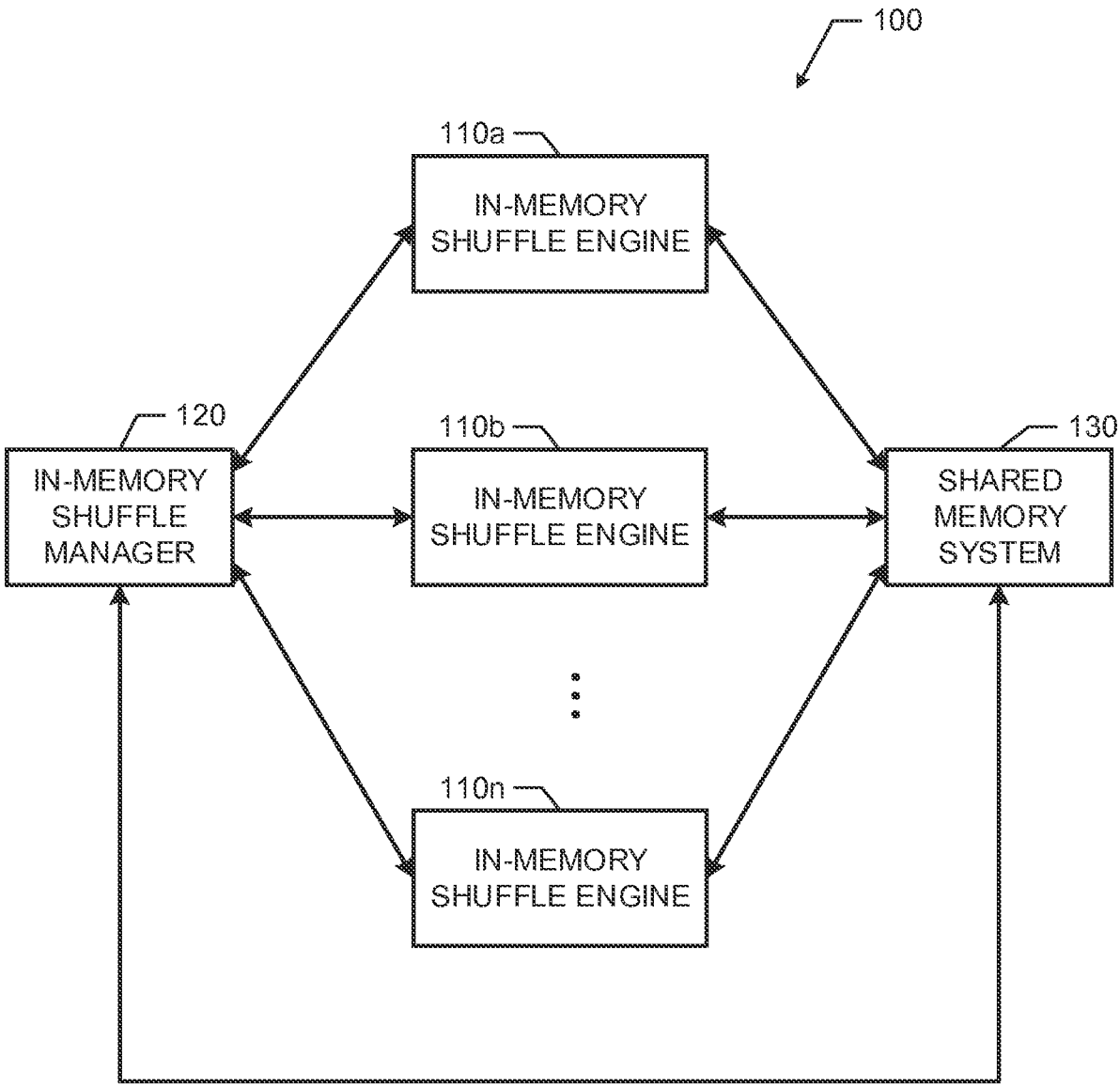

100
110a
IN-MEMORY SHUFFLE ENGINE
120
IN-MEMORY SHUFFLE MANAGER
110b
IN-MEMORY SHUFFLE ENGINE
130
SHARED MEMORY SYSTEM
110n
IN-MEMORY SHUFFLE ENGINE

FIG. 1

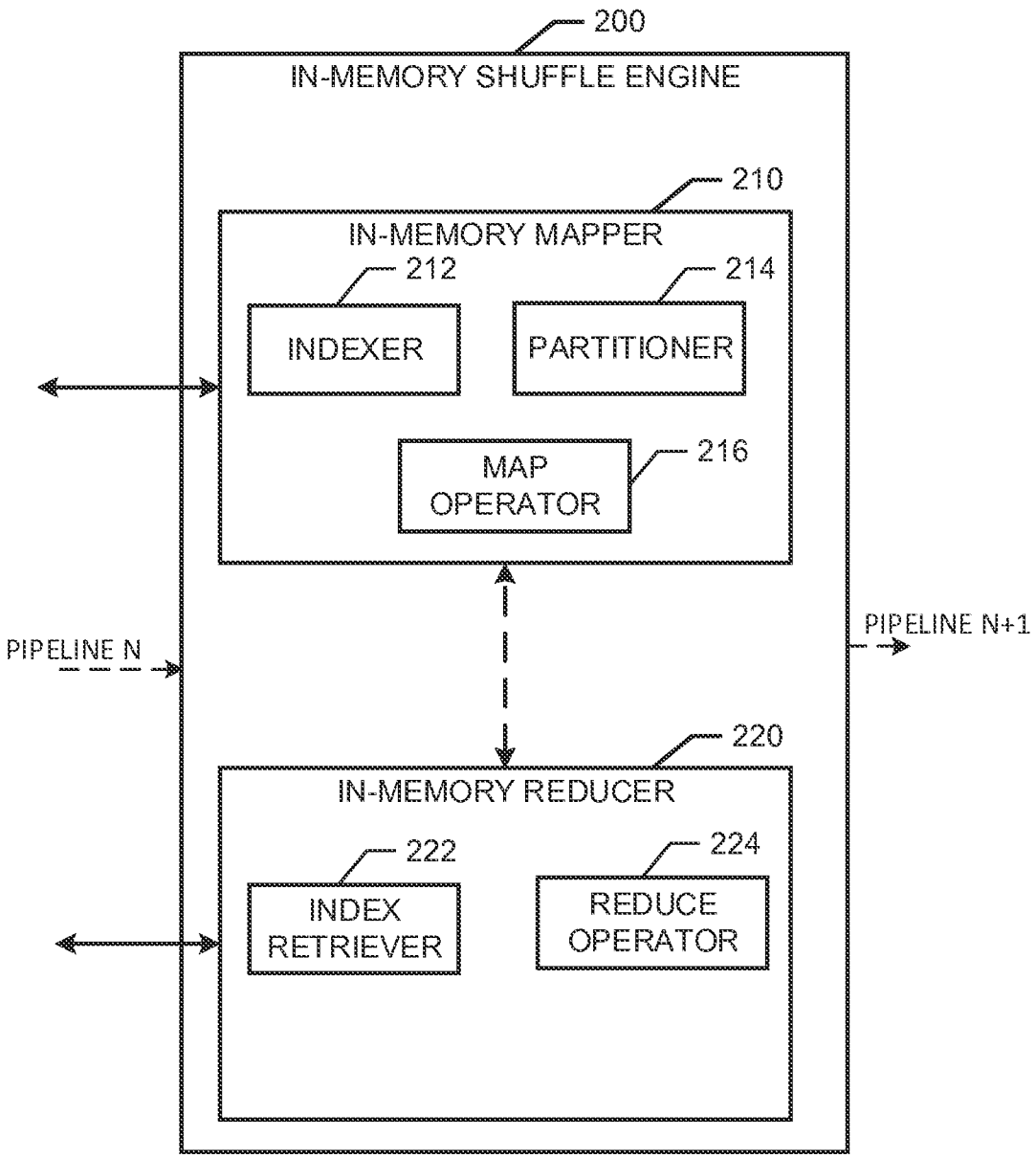

200
IN-MEMORY SHUFFLE ENGINE
210
IN-MEMORY MAPPER
212
INDEXER
214
PARTITIONER
MAP OPERATOR
216
PIPELINE N
PIPELINE N+1
220
IN-MEMORY REDUCER
222
INDEX RETRIEVER
224
REDUCE OPERATOR

FIG. 2

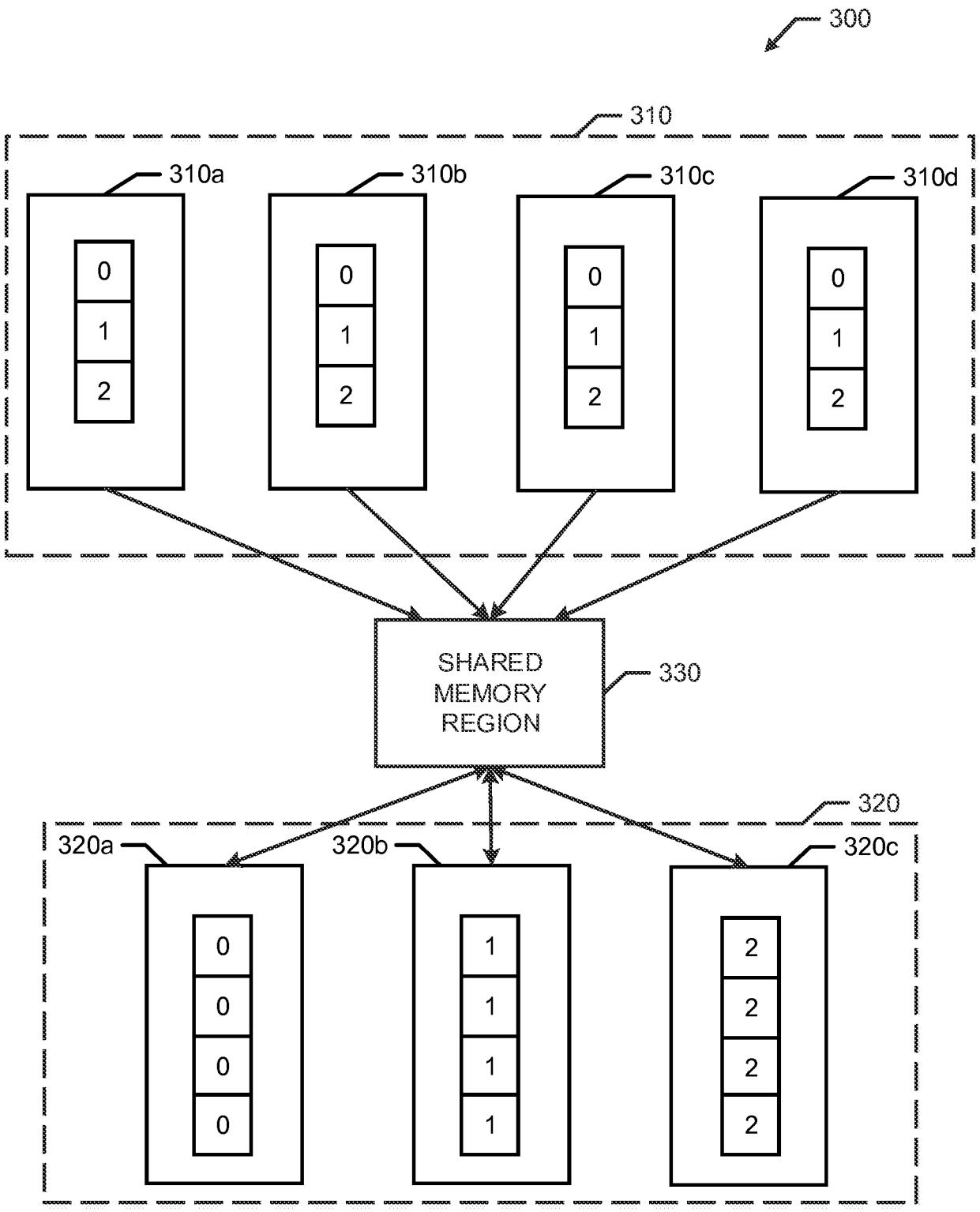

300
310
310a
310b
310c
310d
0
1
2
SHARED MEMORY REGION
330
320
320a
320b
320c
0
0
0
0
1
1
1
1
2
2
2
2

FIG. 3

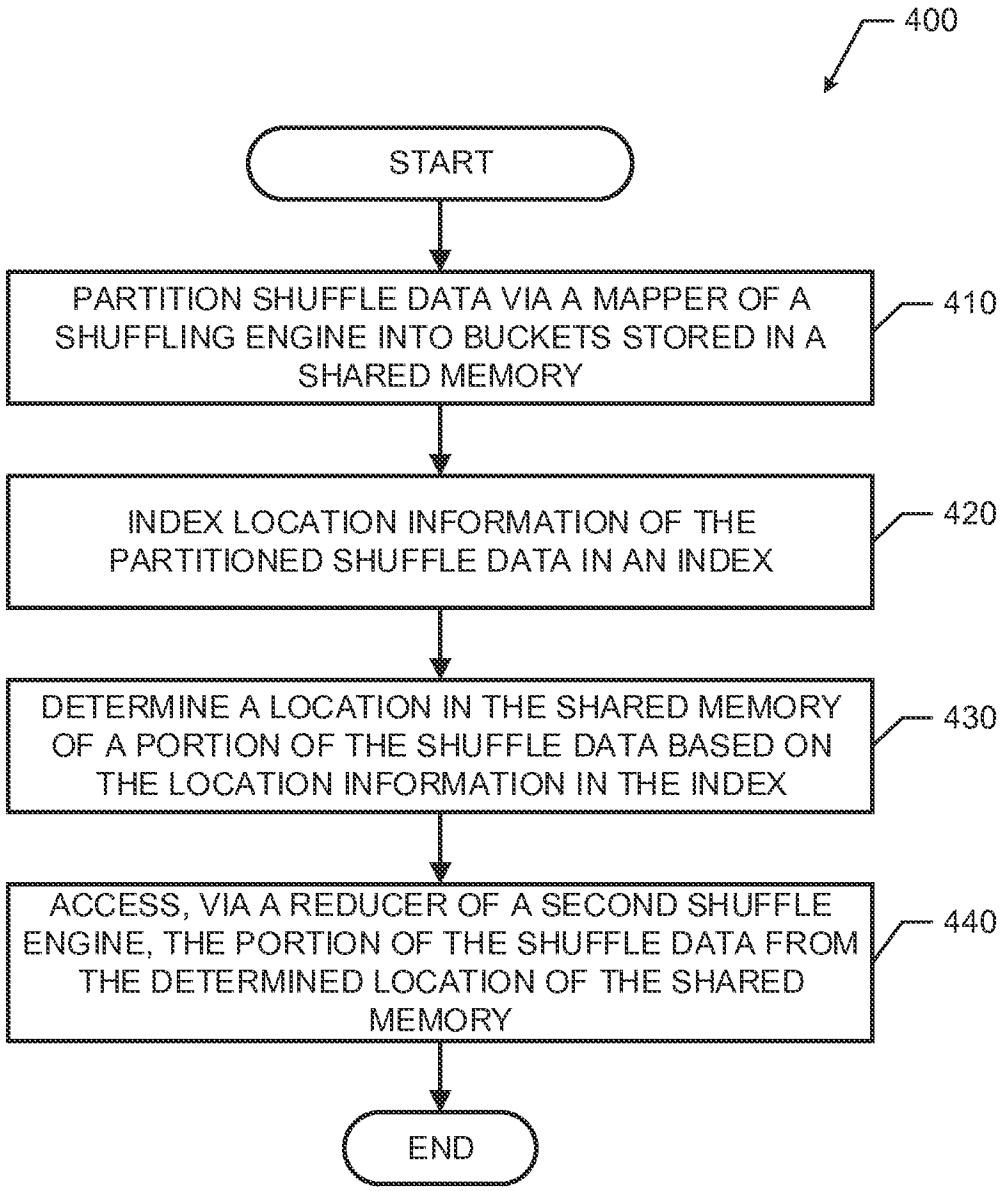

400
START
PARTITION SHUFFLE DATA VIA A MAPPER OF A SHUFFLING ENGINE INTO BUCKETS STORED IN A SHARED MEMORY
410
INDEX LOCATION INFORMATION OF THE PARTITIONED SHUFFLE DATA IN AN INDEX
420
DETERMINE A LOCATION IN THE SHARED MEMORY OF A PORTION OF THE SHUFFLE DATA BASED ON THE LOCATION INFORMATION IN THE INDEX
430
ACCESS, VIA A REDUCER OF A SECOND SHUFFLE ENGINE, THE PORTION OF THE SHUFFLE DATA FROM THE DETERMINED LOCATION OF THE SHARED MEMORY
440
END

FIG. 4

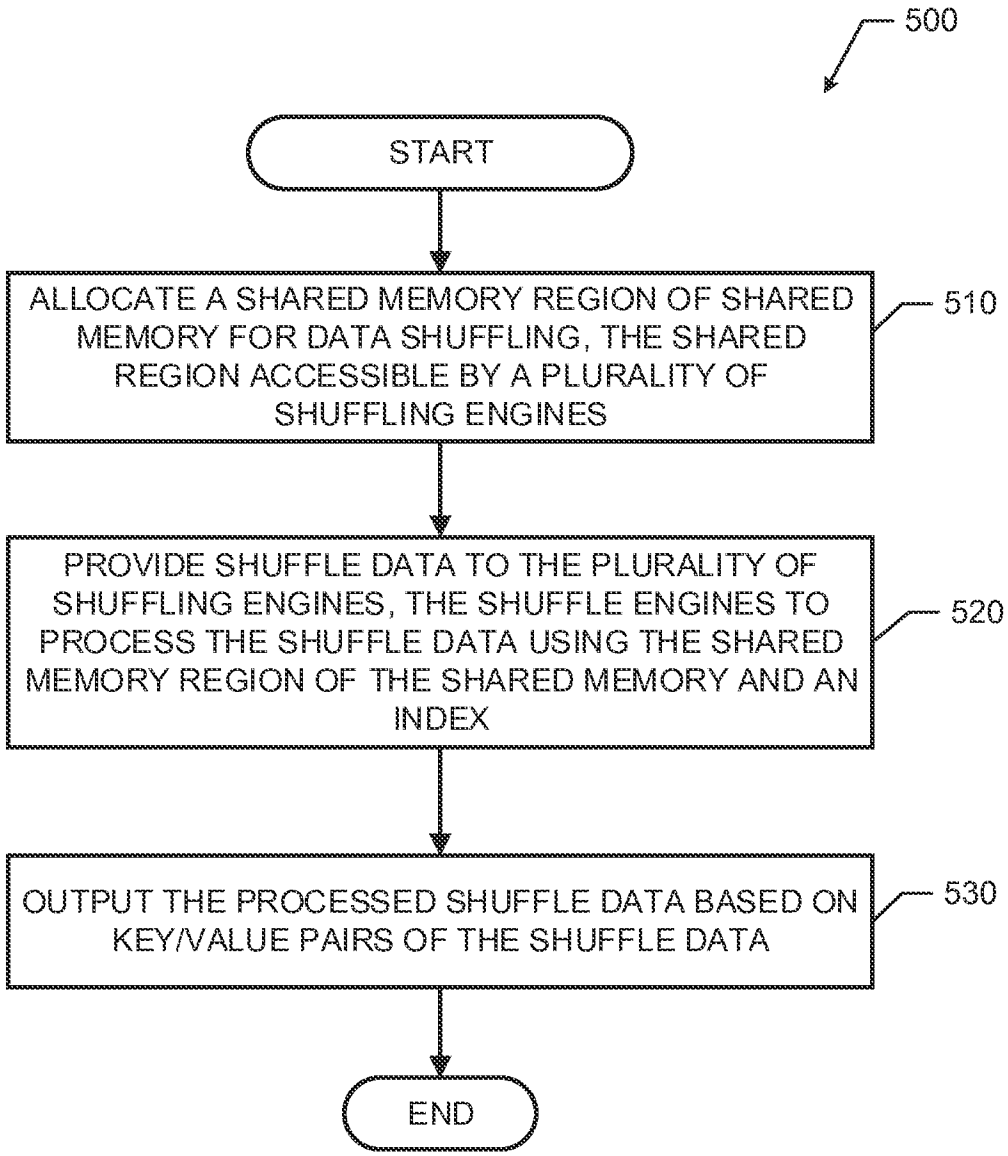

500
START
ALLOCATE A SHARED MEMORY REGION OF SHARED MEMORY FOR DATA SHUFFLING, THE SHARED REGION ACCESSIBLE BY A PLURALITY OF SHUFFLING ENGINES
510
PROVIDE SHUFFLE DATA TO THE PLURALITY OF SHUFFLING ENGINES, THE SHUFFLE ENGINES TO PROCESS THE SHUFFLE DATA USING THE SHARED MEMORY REGION OF THE SHARED MEMORY AND AN INDEX
520
OUTPUT THE PROCESSED SHUFFLE DATA BASED ON KEY/VALUE PAIRS OF THE SHUFFLE DATA
530
END

FIG. 5

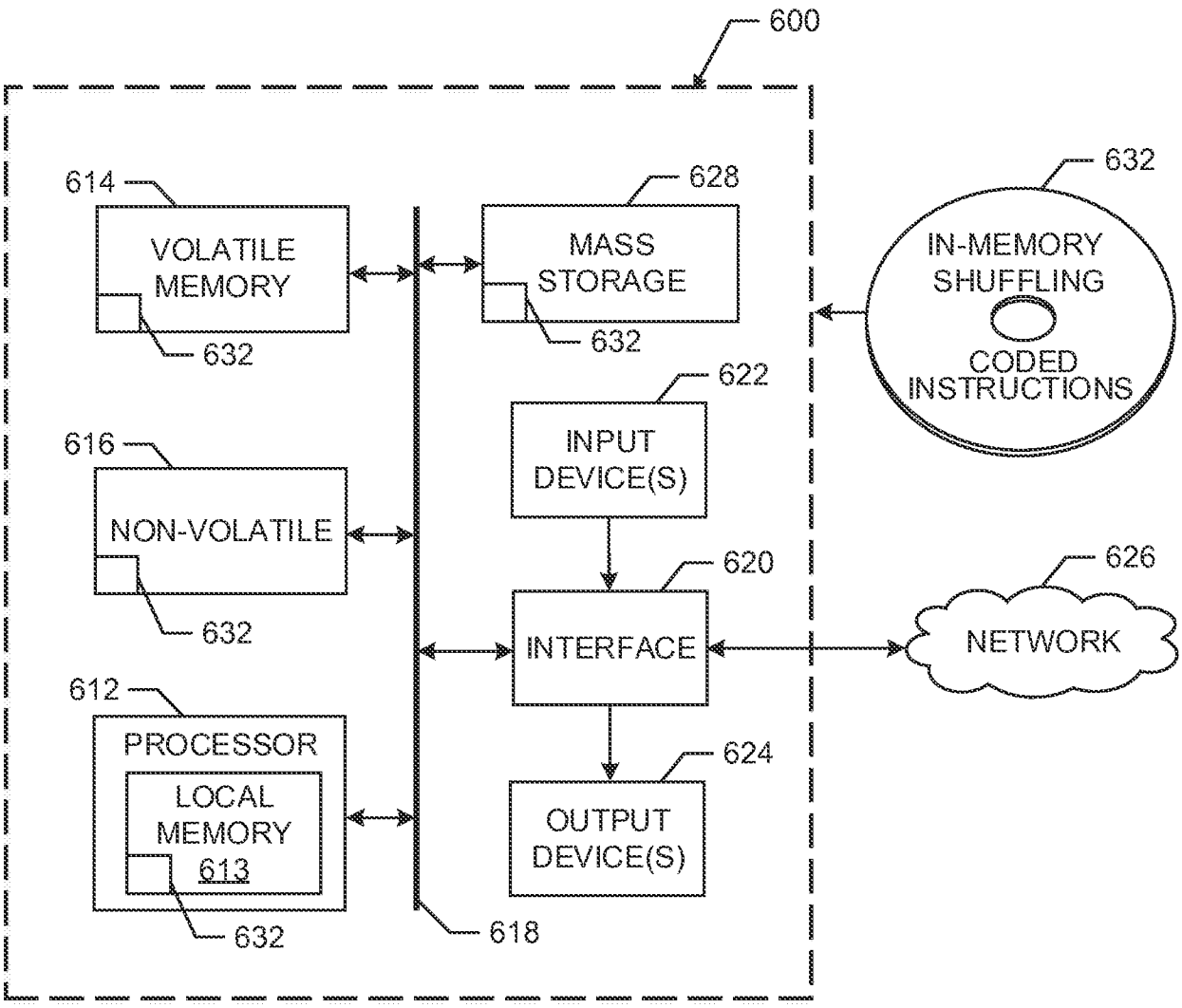

600
614
VOLATILE MEMORY
632
628
MASS STORAGE
632
632
IN-MEMORY SHUFFLING
CODED INSTRUCTIONS
622
INPUT DEVICE(S)
616
NON-VOLATILE
632
620
INTERFACE
626
NETWORK
612
PROCESSOR
LOCAL MEMORY
613
632
624
OUTPUT DEVICE(S)
618

FIG. 6

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2015/061843

A. CLASSIFICATION OF SUBJECT MATTER

G06F 12/08(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 12/08; G06F 9/48; G06F 17/30; G06F 9/46; G06F 17/27

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: partitioning, shuffle, engine, mapper, shared, memory, index, location, information, reducer, key, value, pair, in-memory

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2014-0059552 A1 (DAVID CUNNINGHAM et al.) 27 February 2014 See paragraphs [0003]-[0004], [0007], [0017], [0039], [0044], [0058], [0070], [0101], [0105], [0118], [0135]; and figures 1-2, 4-5, 11.	1-9,12-14
A		10-11
A	US 2014-0365463 A1 (DIGITALGLOBE INC.) 11 December 2014 See paragraph [0082]; and figure 7.	1-14
A	US 2013-0006612 A1 (PENG XU et al.) 03 January 2013 See paragraphs [0022]-[0030]; and figure 1.	1-14
A	US 2012-0323920 A1 (ALFREDO ALBA et al.) 20 December 2012 See paragraphs [0022]-[0029]; and figure 1.	1-14
A	US 2015-0150018 A1 (INTERNATIONAL BUSINESS MACHINES CORPORATION) 28 May 2015 See paragraphs [0028]-[0039]; and figure 1.	1-14

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

☒ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents,such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
18 August 2016 (18.08.2016)	**19 August 2016 (19.08.2016)**

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

Authorized officer
CHIN, Sang Bum
Telephone No. +82-42-481-8398

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2015/061843

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2014-0059552 A1	27/02/2014	US 2014-055496 A1	27/02/2014
		US 9147373 B2	29/09/2015
US 2014-0365463 A1	11/12/2014	WO 2014-197684 A1	11/12/2014
US 2013-0006612 A1	03/01/2013	CN 103650033 A	19/03/2014
		EP 2727103 A2	07/05/2014
		EP 2727103 B1	31/12/2014
		EP 2851895 A2	25/03/2015
		EP 2851895 A3	06/05/2015
		KR 10-2014-0041735 A	04/04/2014
		US 2013-006623 A1	03/01/2013
		US 8494850 B2	23/07/2013
		US 8959014 B2	17/02/2015
		WO 2013-003772 A2	03/01/2013
		WO 2013-003772 A3	28/02/2013
US 2012-0323920 A1	20/12/2012	US 2012-179684 A1	12/07/2012
		US 9104749 B2	11/08/2015
		US 9146983 B2	29/09/2015
US 2015-0150018 A1	28/05/2015	US 2015-150017 A1	28/05/2015