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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
24 September 2015 (24.09.2015)

WIPO | PCT

(10) International Publication Number
WO 2015/139670 A1

(51) International Patent Classification:
G06F 17/30 (2006.01)

(21) International Application Number:
PCT/CN2015/074819

(22) International Filing Date:
21 March 2015 (21.03.2015)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

61/968.793	21 March 2014 (21.03.2014)	US
14/663,210	19 March 2015 (19.03.2015)	US

(71) Applicant: **HUAWEI TECHNOLOGIES CO., LTD.** [CN/CN]; Yu Chan B1-3A Intellectual Property Dept., Huawei Administration Building, Bantian,Longgang District, Shenzhen, Guangdong 518129 (CN).

(72) Inventors: **KU, Chi young**; 531 Huckleberry Way, San Ramon, CA California 94582 (US). **HU, Ron-Chung**; 37 Erstwild Ct., Palo Alto, CA California 94303 (US). **CHEN, Mengmeng**; 130 Descanso Dr.Unit 459, San Jose, CA California 95134 (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,

[Continued on next page]

(54) Title: SYSTEM AND METHOD FOR COLUMN-SPECIFIC MATERIALIZATION SCHEDULING

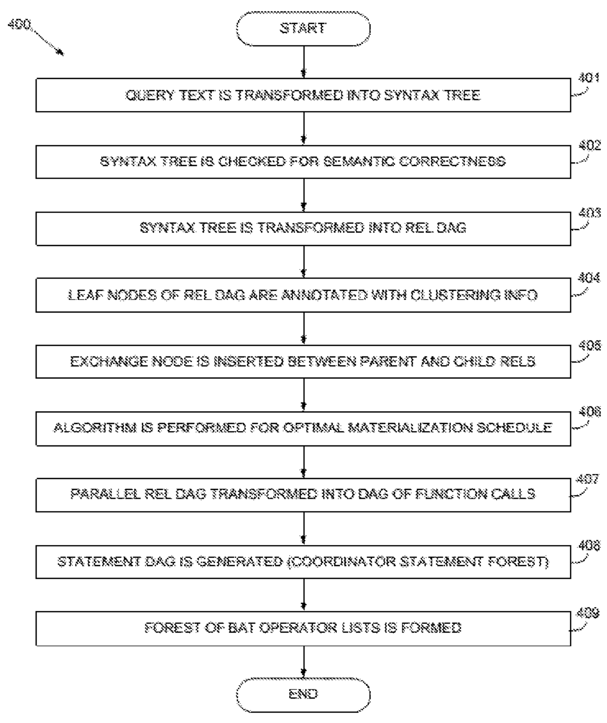
400
START
QUERY TEXT IS TRANSFORMED INTO SYNTAX TREE
401
SYNTAX TREE IS CHECKED FOR SEMANTIC CORRECTNESS
402
SYNTAX TREE IS TRANSFORMED INTO REL DAG
403
LEAF NODES OF REL DAG ARE ANNOTATED WITH CLUSTERING INFO
404
EXCHANGE NODE IS INSERTED BETWEEN PARENT AND CHILD RELS
405
ALGORITHM IS PERFORMED FOR OPTIMAL MATERIALIZATION SCHEDULE
406
PARALLEL REL DAG TRANSFORMED INTO DAG OF FUNCTION CALLS
407
STATEMENT DAG IS GENERATED (COORDINATOR STATEMENT FOREST)
408
FOREST OF BAT OPERATOR LISTS IS FORMED
409
END

FIG. 4

(57) Abstract: A method of dynamically computing an optimal materialization schedule for each column in a column oriented RDBMS. Dynamic column-specific materialization scheduling in a distributed column oriented RDBMS is optimized by choosing a materialization strategy based on execution cost including central processing unit (CPU), disk, and network costs for each individual exchange operator. The dynamic programming approach is computationally feasible because the optimal schedule for a sub-plan is path independent.

WO 2015/139670 A1

SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

Published:

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

SYSTEM AND METHOD FOR COLUMN-SPECIFIC MATERIALIZATION SCHEDULING

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims priority to U.S. Provisional Application No. 61/968,793, filed on March 21, 2014 and entitled "COLUMN-SPECIFIC MATERIALIZATION SCHEDULING" and U.S. Non-Provisional Application No. 14/663,210, filed on March 19, 2015 and entitled "SYSTEM AND METHOD FOR COLUMN-SPECIFIC MATERIALIZATION SCHEDULING". Each of these applications is incorporated herein by reference as if reproduced in its entirety.

TECHNICAL FIELD

[0002] The present disclosure is generally directed to relational database management systems (RDBMSs), and more specifically to a system and method for column-specific materialization in a column oriented RDBMS.

BACKGROUND

[0003] A column oriented RDBMS is a DBMS that stores data tables as sections of columns of data, rather than as rows of data. During query execution in a column oriented RDBMS, it is often necessary to stitch together multiple columns of a record. Some of the columns are added to intermediate results during the query execution. This process is called materialization. How columns are materialized is an important factor in determining query performance in a column oriented RDBMS. Existing column oriented RDBMSs typically employ either fixed early materialization or fixed late materialization. In early materialization, columns referenced in a query are fetched at the leaf nodes of an operator graph and they are transmitted from a child operator to a parent operator if required by up-stream operators. In late materialization, columns needed by an operator are fetched from their sources just before processing and discarded afterwards. For most column oriented RDBMSs, the column materialization strategy is hard coded.

SUMMARY

[0004] This disclosure is directed to determining an optimal materialization schedule for each column in a query execution in a column oriented RDBMS.

[0005] One example embodiment includes a method of dynamically establishing a materialization schedule in a RDBMS. The method includes receiving a query text, transforming the query text into a Rel directed acyclic graph (DAG), performing a bottom-up transversal of the Rel DAG to create a parallel Rel DAG, and computing a column specific materialization schedule of the parallel Rel DAG. The parallel Rel DAG is transformed into a DAG of function calls and data re-shuffling actions to create a parallel statement forest. A coordinator statement forest is generated that invokes the function calls and the data re-shuffling actions according to the parallel statement forest. The parallel statement forest and the coordinator statement forest are transformed into a forest of binary association table (BAT) operator lists to compute an optimal materialization schedule for each column of a table.

[0006] In another example embodiment, a RDBMS is configured to dynamically establish a materialization schedule.

[0007] In another example embodiment, a relational data base management system (RDBMS) is configured to dynamically establish a materialization schedule. The RDBMS includes: a receiving means configured to receive a query text, a transforming means configured to transform the query text into a Rel directed acyclic graph (DAG), a performing means configured

to perform a bottom-up transversal of the Rel DAG to create a parallel Rel DAG, and a computing means configured to compute a column specific materialization schedule of the parallel Rel DAG. The parallel Rel DAG is transformed into a DAG of function calls and data re-shuffling actions to create a parallel statement forest. A coordinator statement forest is generated that invokes the function calls and the data re-shuffling actions according to the parallel statement forest. The parallel statement forest and the coordinator statement forest are transformed into a forest of binary association table (BAT) operator lists to compute an optimal materialization schedule for each column of a table.

BRIEF DESCRIPTION OF THE DRAWINGS

[0008] For a more complete understanding of the present disclosure, and the advantages thereof, reference is now made to the following descriptions taken in conjunction with the accompanying drawings, wherein like numbers designate like objects, and in which:

[0009] FIGURE 1 illustrates a table schema from the Transaction Processing Performance Council (TPC) Benchmark H (TPC-H);

[0010] FIGURE 2 illustrates a syntax tree of a query plan for a structured query language (SQL) statement;

[0011] FIGURE 3 illustrates an example column-specific materialization algorithm in accordance with this disclosure;

[0012] FIGURE 4 illustrates an example method for parallel query optimization in accordance with this disclosure;

[0013] FIGURE 5 illustrates an example parallel statement forest;

[0014] FIGURE 6 illustrates an example coordinator statement forest; and

[0015] FIGURE 7 illustrates an example of a computing device for parallel query optimization according to this disclosure.

DETAILED DESCRIPTION

[0016] FIGURES 1 through 7, discussed below, and the various embodiments used to describe the principles of the present invention in this patent document are by way of illustration only and should not be construed in any way to limit the scope of the invention. Those skilled in the art will understand that the principles of the invention may be implemented in any type of suitably arranged device or system.

[0017] In a given query in a column oriented RDBMS, different columns are accessed by different sets of operators. Therefore, using a single materialization approach for all columns in the query would likely result in some columns not being materialized in an optimal fashion. Embodiments of this disclosure provide a method and apparatus for dynamic column-specific materialization scheduling in a distributed column oriented RDBMS. The materialization schedule is optimized by selecting a materialization strategy based on an execution cost including central processing unit (CPU), disk, and network costs for each individual exchange operator. The disclosed embodiments use dynamic programming techniques to determine the optimal materialization schedule. Dynamic programming is computationally feasible for the disclosed embodiments because the optimal schedule for a sub-plan is path independent.

[0018] As described earlier, conventional materialization schemes include early materialization and late materialization. To better illustrate these materialization schemes, examples of each will now be described.

[0019] To illustrate an example of early materialization, consider the following example Query 1, which is based on existing tables in the Transaction Processing Performance Council (TPC) Benchmark H (TPC-H) table schema shown in FIGURE 1.

[0020] Example Query 1:

```
SELECT l_suppkey from lineitem, part
 WHERE l_partkey = p_partkey
   AND l_shipdate > '2008-01-01';
```

[0021] For the purposes of this example, it is assumed that the table PART is partitioned by the column p_partkey, and that the table LINEITEM is partitioned by the column l_orderkey. Based on the known data in the PART and LINEITEM tables, it can be shown that, for Query 1, the join selectivity is approximately 50%. As known in the art, join selectivity is a measure of how much variation (i.e., how many different values) exists between records in a join result. Low selectivity means that there is not a lot of variation in the values in a column, while high selectivity means there is substantial variation in the values in the column. Before a shuffle of the records in the table LINEITEM, the join selectivity can be examined to determine the cost of early materialization and late materialization. After computing the cost, it is found that early materialization (i.e., stitch l_suppkey with l_partkey, and then shuffle) is better for Query 1. This is because the communication/CPU cost of sending 50% of Row IDs and l_suppkey column data would be more costly than re-shuffling the entire l_suppkey column data.

[0022] To illustrate an example of late materialization, consider the following example Query 2, which is also based on the TPC-H table schema shown in FIGURE 1.

[0023] Example Query 2:

```
SELECT l_suppkey from lineitem, part
```

```
 WHERE l_partkey = p_partkey
   AND l_shipdate > '2014-10-01';
```

[0024] Once again, it is assumed that the table PART is partitioned by the column p_partkey, and that the table LINEITEM is partitioned by the column l_orderkey. Because of the different shipdate value in Query 2, it can be shown that the join selectivity for Query 2 is approximately 1%. The cost to shuffle the records in the table LINEITEM using early materialization and late materialization can be determined. After computing the cost, it is found that late materialization (i.e., join first, and then fetch the useful l_suppkey data) is better for Query 2. This is because the communication/CPU cost of sending 1% of Row IDs and l_suppkey column data separately would be less costly than re-shuffling the entire l_suppkey column with l_partkey data.

[0025] For some queries that have multiple table joins, a mixed materialization scheme in accordance with this disclosure can be used. For example, consider the example Query 3, which is also based on the TPC-H table schema shown in FIGURE 1.

[0026] Example Query 3:

```
SELECT l_suppkey from lineitem, part a, part b
 WHERE l_partkey = a.p_partkey
   AND l_suppkey = b.p_partkey
   AND l_shipdate > '2008-10-01';
```

[0027] Once again, it is assumed that the table PART is partitioned by the column p_partkey, and that the table LINEITEM is partitioned by the column l_orderkey. In Query 3, there are two table joins. For the known data in the tables, it can be shown that the join selectivity for "l_partkey = a.p_partkey" (the first table join) is 50%, and that the join selectivity for "l_suppkey = b.p_partkey" (the second table join) is 1%. After computing costs to shuffle the records in the table

LINEITEM, it is found that mixed materialization is better for Query 3. That is, the optimal materialization scheme for Query 3 is to use early materialization in the first table join, and use late materialization in the second table join.

[0028] FIGURE 2 illustrates a syntax tree of a query plan for example Query 3. As shown in FIGURE 2, the syntax tree 200 includes a node for each operator in Query 3. The syntax tree 200 is a directed acyclic graph (DAG) that includes a plurality of nodes 201-207. The node 201 represents the LINEITEM table in Query 3. The node 201 is at the bottom of the syntax tree 200 because LINEITEM is the first operator in Query 3. The node 202 is an exchange node that represents a data shuffling or redistribution operation in Query 3. Here, the data in the LINEITEM table is shuffled before its join to the PART table. The node 203 represents the first instance (instance A) of the PART table. The node 204 represents the join of the shuffled LINEITEM table and instance A of the PART table, as well as the SELECT statement. The node 205 is another exchange node that represents a data shuffling of the result set from the node 204. The node 206 represents the second instance (instance B) of the PART table. The node 207 represents the join of shuffled result set from the node 205 and instance B of the PART table.

[0029] **Materialization Cost**

[0030] In the embodiments disclosed herein, a column-specific materialization algorithm is part of a parallel query optimization compilation process that transforms a structured query language (SQL) statement into a parallel execution plan. In accordance with the disclosed embodiments, a decision to using early materialization or late materialization can be based on the following recursive reasoning.

[0031] If the parallel execution plan is represented as a DAG of exchange nodes (such as the syntax tree 200 shown in FIGURE 2), the best way to materialize a column C at an exchange node E (e.g., the exchange node 205) depends on whether the column is materialized at E's child exchange node, E_1 (e.g., the exchange node 202).

[0032] For example, if column C is materialized at exchange node E_1, the cost to materialize column C at exchange node E would be the cost to materialize column C at exchange node E_1 plus the communication/CPU cost to re-shuffle column C at exchange node E. If column C is not materialized at exchange node E_1, the cost would be the cost of late materialization at exchange node E, which is the communication/CPU cost of sending Row IDs and column C's data.

[0033] Based on the preceding reasoning, the problem of computing the best materialization schedule for a column C, given the exchange node E and the parallel execution exchange node DAG, could be summarized as:

[0034] Choose the materialization schedule M so as to minimize the following cost at each level L of Exchange node E in a recursive way:

Minimize (Cost of transferring C's column data based on M[L-1]

+ the cost of materializing C at E[L-1] according to M[L-1])

where M[L-1] is the materialization choice at level L-1, and E[L-1] is the Exchange node at level L-1.

[0035] Turning again to FIGURE 2 and Query 3, it can be shown based on the preceding reasoning, that the best way to materialize l_suppkey at the node 202 (Exchange_1) is early materialization, while the best way to materialize l_suppkey at the node 205 (Exchange_2) is late materialization.

[0036] In accordance with this disclosure, a method for computing the optimal materialization schedule for a column is provided. The disclosed method assumes that the distributed execution plan is represented by a DAG of exchange nodes and relational operators, such as the syntax tree 200 shown in FIGURE 2. When the parent exchange node needs to materialize a column, the parent exchange node materializes this column if it has a lower cost than fetching the column from Row ID in the parent exchange node. The method can be summarized in the following outline:

1. A choice is made at each exchange node whether to materialize the column. This is based on cost comparisons.
    a. If a column is materialized at the previous exchange node:
        i. If the column is materialized at the next exchange node:
            1. If the column is materialized at the current exchange node:

cost of materialization at the previous node =
the transfer cost of the column from next node to current node +
the cost of materialization at the next node.

2. If the column is not materialized at the current exchange node:

cost of materialization at the previous node =
the cost of sending the Row ID to the source nodes +
transfer cost of the resulting column.

ii. If the column is not materialized at the next exchange node:

cost of materialization at the previous node =
the cost of sending the Row ID to the source nodes +
transfer cost of the resulting column.

As a result of (a), the column is materialized at the current exchange node if it is required at the current exchange node or if the cost at (a.i.1) is smaller than cost at (a.i.2).

b. If the column is not materialized at the previous exchange node:

i. If the column is materialized at the next exchange node:

1. If the column is materialized at the current exchange node:

cost of not materializing at the previous node = 0.

2. If the column is not materialized at the current exchange node:

cost of not materializing at the previous node = 0.

ii. If the column is not materialized at the next exchange node:

cost of not materializing at the previous node = 0.

As a result of (b), the column is not materialized at the current exchange node.

[0037] Materialization Schedule Algorithm

[0038] FIGURE 3 illustrates an example column-specific materialization algorithm 300 in accordance with this disclosure. The algorithm 300 is pseudocode of an acyclic algorithm that can represent (or be represented by) a DAG. The algorithm 300 can be part of a parallel query optimization compilation process that transforms a structured query language (SQL) statement into a parallel execution plan. In particular embodiments, the algorithm 300 can be used to perform a SQL operation that may include mixed materialization, such as Query 3. The algorithm 300 may be performed using a computing device capable of RDBMS operations, such as the computing device 700 of FIGURE 7 (described below).

[0039] The algorithm 300 includes three inputs: Exchange Node E, level L, and column C. As described earlier, the exchange node is a database operator that is used to shuffle records in one or more tables. The level L is provided by the system and is used to identify a level in a query tree. Here, the levels in the query tree are numbered such that the lowest levels of the query tree are shown or indicated at the bottom of the tree, and the highest levels of the query tree are shown or indicated at the top of the tree.

[0040] Array K in the algorithm 300 contains materialization costs for the different levels. That is, each element of the array K corresponds to a materialization cost for one level. The

IF-THEN-ELSE argument indicated at 301 in the algorithm 300 determines if an early schedule or a late schedule will be used for level 1 based on the materialization cost for that level. Thus, the schedule is determined first for level 1. The SET COST operation indicated at 302 is a recursive function that calls the Materialization Schedule algorithm 300 to be performed on a next lower level using Exchange Node E's child as in input. For example, if a query tree includes four levels, and the algorithm 300 is being performed for level 4, then the SET COST operation 302 is used to call the algorithm 300 to be performed for level 3. The SET SCHEDULE operation indicated at 303 is used to set the schedule (early schedule or late schedule) for levels other than level 1 by selecting the minimum cost between (a) the cost of the next lower level + the early materialization cost, and (b) the cost of the next lower level + the late materialization cost. Then, the operation indicated at 304 sets the cost at level L based on the cost of the next lower level (L – 1) and the materialization cost at level L.

[0041] **Dynamic Programming**

[0042] The algorithm 300 is based on dynamic programming principles. Dynamic programming is a technique for solving complex problems by breaking them down into simpler sub-problems. Dynamic programming is often used in mathematics, computer science, economics, and in other fields. One classic example of a complex problem for which dynamic programming is frequently used is determining the shortest path between two cities or locations on a map, taking into account the different roads and intermediate points available in the area.

[0043] In order to be able to use dynamic programming to solve a complex problem, the complex problem itself must possess certain properties. First, the complex problem must include overlapping sub-problems. Second the complex problem must have an optimal substructure. If a problem does not possess these properties, then use of dynamic programming to solve the problem may either be impossible or lead to a sub-optimal solution.

[0044] The materialization algorithm 300 includes overlapping sub-problems. For example, the optimal schedule at level L is determined based on the optimal schedule at level L-1, while the optimal schedule at level L-1 is determined based on the optimal schedule at level L-2, and so on. Thus, the determinations of the different levels can be considered to overlap.

[0045] Similarly, the materialization algorithm 300 includes an optimal substructure. For example, the IF-THEN-ELSE argument at 301 in the algorithm 300 determines an optimal schedule for level 1 based on the materialization cost for that level, and then the algorithm 300 determines the optimal schedule for higher levels based on the schedule for the next lower level. Thus, materialization algorithm 300 includes an optimal substructure based on a lowest performance cost (i.e., a fastest execution time).

[0046] Use of dynamic programming has been shown to find a globally optimal solution for a complex problem. Dynamic programming is different from a greedy algorithm. A greedy algorithm may find a local optimal solution to a sub-problem, but often may arrive at a globally sub-optimal solution. For example, considering the shortest path between two cities problem, a greedy algorithm may find a locally optimal solution to a traffic jam at one intersection, but the local solution may be optimal for only that intersection, and may result in a sub-optimal route overall when the total route between the two cities is considered as a global solution.

[0047] As described above, the algorithm 300 can be part of a parallel query optimization compilation process that transforms a SQL statement into a parallel execution plan. One approach to this query-text to-parallel-plan transformation process to anchor the context within which the column specific algorithm 300 is performed can be summarized in the following method described in FIGURE 4.

[0048] FIGURE 4 illustrates an example method for parallel query optimization in accordance with this disclosure. For ease of explanation, the method 400 is described as being used with the algorithm 300 of FIGURE 3. However, the method 400 could be used with any suitable algorithm and in any suitable system. The method 400 may be performed using a computing device capable of RDBMS operations, such as the computing device 700 of FIGURE 7 (described below), or may be performed by another suitable device or system.

[0049] In operation 401, a query text is transformed into a syntax tree.

[0050] In operation 402, the syntax tree is checked for semantic correctness.

[0051] In operation 403, the syntax tree is transformed into a DAG of relational operators (rels), which may be referred to as a Rel DAG, as known in the art.

[0052] In operation 404, the leaf nodes of the Rel DAG are annotated with clustering information.

[0053] In operation 405, using a bottom-up traversal of the Rel DAG, an exchange node is inserted between a parent Rel and a child Rel when the clustering properties of the output of the child Rel is incompatible with the clustering properties of the input of the parent node. The resulting DAG is called a parallel Rel DAG. The parallel Rel DAG may be similar to the DAG 200 shown in FIGURE 2.

[0054] In operation 406, a column-specific materialization algorithm (e.g., the algorithm 300) is performed to compute the optimal materialization schedule for each column.

[0055] In operation 407, the parallel Rel DAG is transformed into a DAG of function calls and data re-shuffling actions according to the following details. Each function corresponds to a fragment of the parallel Rel DAG between two adjacent exchange nodes. Each data re-shuffling action corresponds to an exchange node. Each function is transformed into a statement forest, where a statement represents a logical BAT operator. The logical BAT operator produces an expression based on expressions produced by its children statements.

[0056] To produce the expression, the logical BAT operator makes a depth-first traversal of the Rel DAG fragment. Then, for each Rel, for each expression exported by the Rel, and for each combination of the source tables' partitions, a statement DAG is generated for the expression.

[0057] Each function takes the columns' data exported from its children exchange nodes as an input. The outputs of each function are expressions exported by the top Rel of the function. The output of the function becomes the input of the data re-shuffling action of its parent exchange node. Note that Row IDs are always exported by a Rel.

[0058] Each data re-shuffling action re-shuffles columns to be materialized at this exchange node. Columns to be materialized but not exported by a child exchange node are fetched by using Row IDs. Each data re-shuffling action is transformed into a statement forest containing one statement DAG for each re-shuffled column. The resulting statement forest is called the parallel statement forest. An example parallel statement forest 500 is shown in FIGURE 5.

[0059] Then, in operation 408, a statement DAG is generated that invokes the functions and data re-shuffling actions according to the depth-first traversal sequence of the parallel statement forest. The resulting statement DAG is called the coordinator statement forest. An example coordinator statement forest 600 is shown in FIGURE 6.

[0060] In operation 409, the parallel statement forest and the coordinator statement forest are transformed into a forest of BAT operator lists. Each list corresponds to a function, a data re-shuffling action, or the coordinator program.

[0061] Although FIGURE 4 illustrates one example of a method 400 for parallel query optimization, various changes may be made to FIGURE 4. For example, while shown as a series of steps, various steps in FIGURE 4 could overlap, occur in parallel, occur in a different order, or occur any number of times.

[0062] FIGURE 7 illustrates an example of a computing device 700 for performing the materialization algorithm 300 of FIGURE 3 or the parallel query optimization method 400 of FIGURE 4. As shown in FIGURE 7, the computing device 700 includes a computing block 703 with a processing block 705 and a system memory 707. The processing block 705 may be any type of programmable electronic device for executing software instructions, but will conventionally be one or more microprocessors. The system memory 707 may include both a read-only memory (ROM) 709 and a random access memory (RAM) 711. As will be appreciated by those of skill in the art, both the read-only memory 709 and the random access memory 711 may store software instructions for execution by the processing block 705.

[0063] The processing block 705 and the system memory 707 are connected, either directly or indirectly, through a bus 713 or alternate communication structure, to one or more peripheral devices. For example, the processing block 705 or the system memory 707 may be directly or indirectly connected to one or more additional memory storage devices 715. The memory storage devices 715 may include, for example, a “hard” magnetic disk drive, a solid state disk drive, an optical disk drive, and a removable disk drive. The processing block 705 and the system memory 707 also may be directly or indirectly connected to one or more input devices 717 and one or more output devices 719. The input devices 717 may include, for example, a keyboard, a pointing device (such as a mouse, touchpad, stylus, trackball, or joystick), a touch screen, a scanner, a camera, and a microphone. The output devices 719 may include, for example, a display device, a printer and speakers. Such a display device may be configured to display video images. With various examples of the computing device 700, one or more of the peripheral devices 715-719 may be internally housed with the computing block 703. Alternately, one or more of the peripheral devices 715-719 may be external to the housing for the computing block 703 and connected to the bus 713 through, for example, a Universal Serial Bus (USB) connection or a digital visual interface (DVI) connection.

[0064] With some implementations, the computing block 703 may also be directly or indirectly connected to one or more network interfaces cards (NIC) 721, for communicating with other devices making up a network. The network interface cards 721 translate data and control signals from the computing block 703 into network messages according to one or more communication protocols, such as the transmission control protocol (TCP) and the Internet protocol (IP). Also, the network interface cards 721 may employ any suitable connection agent (or combination of agents) for connecting to a network, including, for example, a wireless transceiver, a modem, or an Ethernet connection.

[0065] It should be appreciated that the computing device 700 is illustrated as an example only, and it not intended to be limiting. Various embodiments of this disclosure may be implemented using one or more computing devices that include the components of the computing device 700 illustrated in FIGURE 7, or which include an alternate combination of components, including components that are not shown in FIGURE 7. For example, various embodiments of the invention may be implemented using a multi-processor computer, a plurality of single and/or multiprocessor computers arranged into a network, or some combination of both.

[0066] The algorithm described in this disclosure computes the best materialization schedule for each column on every exchange operator within a query. This is advantageous over existing materialization scheduling algorithms that employ either fixed early materialization or fixed late materialization for all exchange operators in a query. The algorithm disclosed herein can be implemented by traversing the parallel execution graph from the top down, identifying columns that have not been scheduled. For each such column, dynamic programming is applied to compute the materialization schedule in a recursive (or bottom up) fashion. The minimum materialization costs at level L-1 do not change with the choice of materialization at levels greater or equal to L. The computation complexity is linearly proportional to the height of the parallel execution graph and number of columns.

[0067] Embodiments of this disclosure has been demonstrated in simulation tests to reduce the interconnect bandwidth requirement for distributed query processing by an average of 10% - 30%. Assuming that the inter-node communication cost is about 25% of the total query processing cost, this reduces the total cost of distributed query processing by 2.5% - 7.5%.

[0068] In some embodiments, some or all of the functions or processes of the one or more of the devices are implemented or supported by a computer program that is formed from computer readable program code and that is embodied in a computer readable medium. The phrase "computer readable program code" includes any type of computer code, including source code, object code, and executable code. The phrase "computer readable medium" includes any type of medium capable of being accessed by a computer, such as read only memory (ROM), random access memory (RAM), a hard disk drive, a compact disc (CD), a digital video disc (DVD), or any other type of memory.

[0069] It may be advantageous to set forth definitions of certain words and phrases used throughout this patent document. The terms "include" and "comprise," as well as derivatives thereof, mean inclusion without limitation. The term "or" is inclusive, meaning and/or. The phrases “associated with” and “associated therewith,” as well as derivatives thereof, mean to include, be included within, interconnect with, contain, be contained within, connect to or with, couple to or with, be communicable with, cooperate with, interleave, juxtapose, be proximate to, be bound to or with, have, have a property of, or the like.

[0070] While this disclosure has described certain embodiments and generally associated methods, alterations and permutations of these embodiments and methods will be apparent to those skilled in the art. Accordingly, the above description of example embodiments does not define or constrain this disclosure. Other changes, substitutions, and alterations are also possible without departing from the spirit and scope of this disclosure, as defined by the following claims.

WHAT IS CLAIMED IS:

1. A method of establishing a materialization schedule using a dynamic programming technique in a relational data base management system (RDBMS), the method comprising:

receiving a query text;

transforming the query text into a Rel directed acyclic graph (DAG);

performing a bottom-up transversal of the Rel DAG to create a parallel Rel DAG;

computing a column specific materialization schedule of the parallel Rel DAG;

transforming the parallel Rel DAG into a DAG of function calls and data re-shuffling actions to create a parallel statement forest;

generating a coordinator statement forest that invokes the function calls and the data re-shuffling actions according to the parallel statement forest; and

transforming the parallel statement forest and the coordinator statement forest into a forest of binary association table (BAT) operator lists to compute an optimal materialization schedule for each column of a table.

2. The method as specified in Claim 1, wherein:

each function call corresponds to a fragment of the parallel Rel DAG between two adjacent exchange nodes; and

each data re-shuffling action corresponds to an exchange node.

3. The method as specified in Claim 1 or Claim 2, wherein leaf nodes of the Rel DAG are annotated with clustering information.

4. The method as specified in any one of Claims 1 - 3, wherein the bottom-up traversal is performed by inserting an exchange node between a parent Rel node and a child Rel node when a clustering property of an output of the child Rel node is incompatible with a clustering property of an input of the parent Rel node.

5. The method as specified in any one of Claims 1 - 4, wherein each function call is transformed into a statement forest, wherein a statement represents a BAT operator, to produce an expression produced by a child statement of the function call.

6. The method as specified in Claim 5, wherein the expression is produced by:

making a depth-first traversal of a corresponding Rel DAG fragment, and

for each Rel, for each expression exported by the Rel, and for each combination of a source tables' partitions, generate a statement DAG for the expression.

7. The method as specified in any one of Claims 1 - 6, wherein:

each function call takes columns' data exported from its child Rel nodes as an input,

an output of each function call is expression exported by a top Rel of the function call, and

the output of the function call is the input of the data re-shuffling action of its parent node, wherein Row IDs are exported by a Rel.

8. The method as specified in Claim 4, wherein each data re-shuffling action re-shuffles columns to be materialized at a respective Rel node, and columns materialized but not exported by a child Rel node are fetched by using Row IDs.

9. The method as specified in Claim 8, wherein each data re-shuffling action is transformed into a statement forest containing one statement DAG for each re-shuffled column to create the parallel statement forest.

10. The method as specified in any one of Claims 1 - 9, further comprising generating a statement DAG that invokes the function calls and data re-shuffling actions according to a depth-first traversal sequence of the parallel statement forest to create the coordinator statement forest.

11. A relational data base management system (RDBMS) configured to:

receive a query text;

transform the query text into a Rel directed acyclic graph (DAG);

perform a bottom-up transversal of the Rel DAG to create a parallel Rel DAG;

compute a column specific materialization schedule of the parallel Rel DAG;

transform the parallel Rel DAG into a DAG of function calls and data re-shuffling actions to create a parallel statement forest;

generate a coordinator statement forest that invokes the function calls and the data re-shuffling actions according to the parallel statement forest; and

transform the parallel statement forest and the coordinator statement forest into a forest of binary association table (BAT) operator lists to compute an optimal materialization schedule for each column of a table.

12. The RDBMS as specified in Claim 11, wherein:

each function call corresponds to a fragment of the parallel Rel DAG between two adjacent exchange nodes; and

each data re-shuffling action corresponds to an exchange node.

13. The RDBMS as specified in Claim 11 or claim 12, wherein leaf nodes of the Rel DAG are configured to be annotated with clustering information.

14. The RDBMS as specified in any one of Claims 11 - 13, wherein the bottom-up traversal is performed by inserting an exchange node between a parent Rel node and a child Rel

node when a clustering property of an output of the child Rel node is incompatible with a clustering property of an input of the parent Rel node.

15. The RDBMS as specified in any one of Claims 11 - 14, wherein each function call is configured to be transformed into a statement forest, wherein a statement represents a BAT operator, to produce an expression produced by a child statement of the function call.

16. The RDBMS as specified in Claim 15, wherein the expression is configured to be produced by:

making a depth-first traversal of a corresponding Rel DAG fragment, and

for each Rel, for each expression exported by the Rel, and for each combination of a source tables' partitions, generating a statement DAG for the expression.

17. The RDBMS as specified in any one of Claims 11 - 16, wherein:

each function call takes columns' data exported from its child Rel nodes as an input,

an output of each function call is expression exported by a top Rel of the function call, and

the output of the function call is the input of the data re-shuffling action of its parent node, wherein Row IDs are exported by a Rel.

18. The RDBMS as specified in Claim 14, wherein each data re-shuffling action re-shuffles columns to be materialized at a respective Rel node, and columns materialized but not exported by a child Rel node are fetched by using Row IDs.

19. The RDBMS as specified in Claim 18, wherein each data re-shuffling action is transformed into a statement forest containing one statement DAG for each re-shuffled column to create the parallel statement forest.

20. A method of establishing a materialization schedule using a dynamic programming technique in a relational database management system (RDBMS), the method comprising:

receiving a query text;

transforming the query text into a Rel directed acyclic graph (DAG);

performing a bottom-up transversal of the Rel DAG to create a parallel Rel DAG;

computing a column specific materialization schedule of the parallel Rel DAG;

transforming the parallel Rel DAG into a DAG of function calls and data re-shuffling actions to create a parallel statement forest, wherein:

each function call takes columns' data exported from its child Rel nodes as an input,

output of each function call is expression exported by a top Rel of the function call, and

the output of the function call is the input of a data re-shuffling action of its parent node, wherein Row IDs are exported by a Rel;

generating a coordinator statement forest that invokes the function calls and the data re-shuffling actions according to the parallel statement forest; and

transforming the parallel statement forest and the coordinator statement forest into a forest of binary association table (BAT) operator lists to compute an optimal materialization schedule for each column of a table.

21. A relational data base management system (RDBMS) configured to dynamically establish a materialization schedule, wherein the RDBMS comprises:

a receiving means, configured to receive a query text;

a transforming means, configured to transform the query text into a Rel directed acyclic graph (DAG);

a performing means, configured to perform a bottom-up transversal of the Rel DAG to create a parallel Rel DAG;

a computing means, configured to compute a column specific materialization schedule of the parallel Rel DAG;

the transforming means, configured to transform the parallel Rel DAG into a DAG of function calls and data re-shuffling actions to create a parallel statement forest;

a generating means, configured to generate a coordinator statement forest that invokes the function calls and the data re-shuffling actions according to the parallel statement forest; and

the transforming means, configured to transform the parallel statement forest and the coordinator statement forest into a forest of binary association table (BAT) operator lists to compute an optimal materialization schedule for each column of a table.

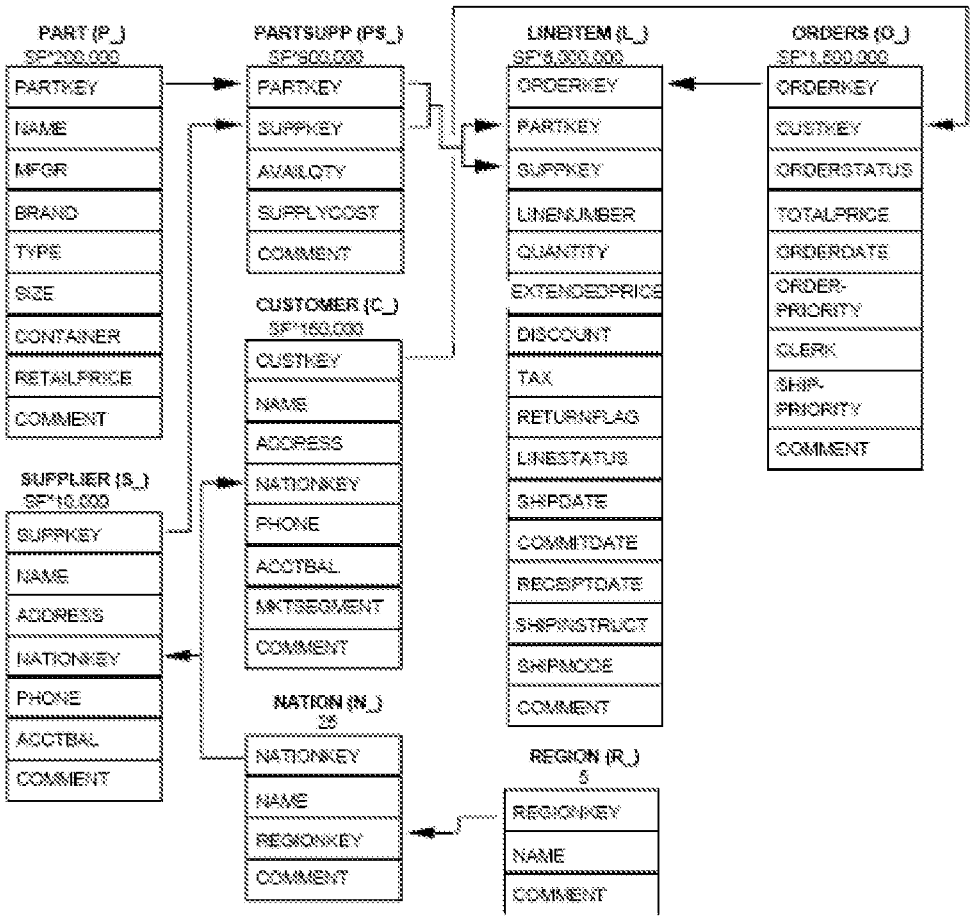
PART (P_)
SF*200,000
PARTKEY
NAME
MFGR
BRAND
TYPE
SIZE
CONTAINER
RETAILPRICE
COMMENT
PARTSUPP (PS_)
SF*800,000
PARTKEY
SUPPKEY
AVAILQTY
SUPPLYCOST
COMMENT
LINEITEM (L_)
SF*6,000,000
ORDERKEY
PARTKEY
SUPPKEY
LINENUMBER
QUANTITY
EXTENDEDPRICE
DISCOUNT
TAX
RETURNFLAG
LINESTATUS
SHIPDATE
COMMITDATE
RECEIPTDATE
SHIPINSTRUCT
SHIPMODE
COMMENT
ORDERS (O_)
SF*1,500,000
ORDERKEY
CUSTKEY
ORDERSTATUS
TOTALPRICE
ORDERDATE
ORDER-PRIORITY
CLERK
SHIP-PRIORITY
COMMENT
CUSTOMER (C_)
SF*150,000
CUSTKEY
NAME
ADDRESS
NATIONKEY
PHONE
ACCTBAL
MKTSEGMENT
COMMENT
SUPPLIER (S_)
SF*10,000
SUPPKEY
NAME
ADDRESS
NATIONKEY
PHONE
ACCTBAL
COMMENT
NATION (N_)
25
NATIONKEY
NAME
REGIONKEY
COMMENT
REGION (R_)
5
REGIONKEY
NAME
COMMENT

FIG. 1

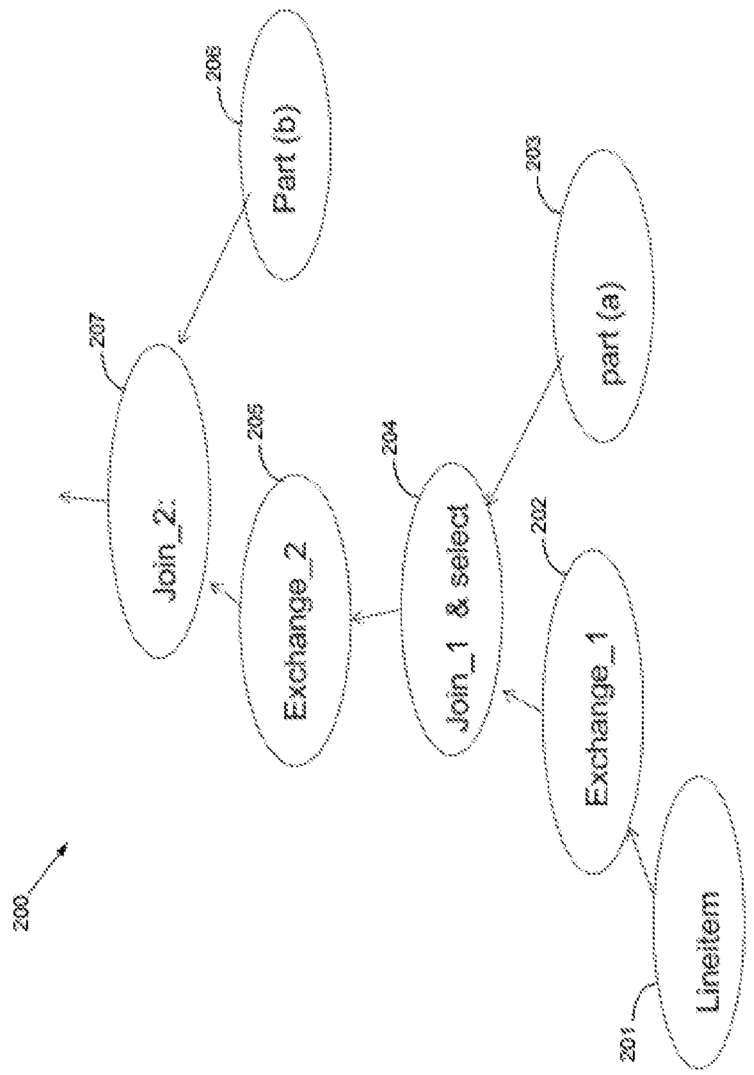
200
201
Lineitem
202
Exchange_1
203
part (a)
204
Join_1 & select
205
Exchange_2
206
Part (b)
207
Join_2

FIG. 2

300

```
Procedure  MaterializationSchedule( Exchange Node E, level L, column C)
    // Array K contains materialisation costs for various levels.
    if (L == 1) then {                                                    ⎫
        if (early schedule cost < late schedule cost)                     ⎪
            then return early schedule M[1] with cost K[1];               ⎬ 301
            else return late schedule M[1] with cost K[1];                ⎪
    }                                                                     ⎭

    set cost K[L-1] by calling MaterializationSchedule(E's child, L-1, C); } 302

    Set schedule M[L] to the lower cost strategy:                         ⎫
        minimize { K[L-1] + early materialisation cost,                   ⎬ 303
                   K[L-1] + late materialisation cost};                   ⎭

    K[L] = K[L-1] + materialization cost at level L;                      } 304
End Procedure
```

FIG. 3

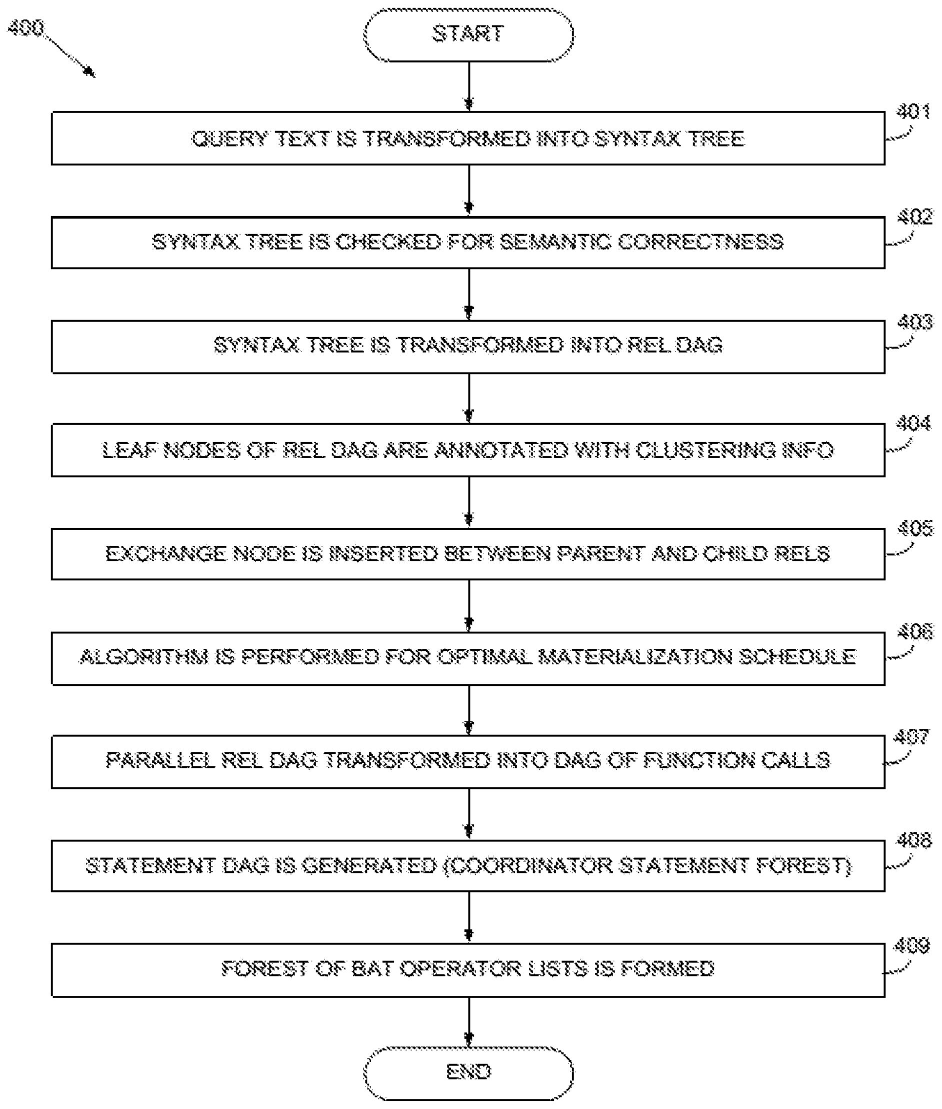
400
START
QUERY TEXT IS TRANSFORMED INTO SYNTAX TREE
401
SYNTAX TREE IS CHECKED FOR SEMANTIC CORRECTNESS
402
SYNTAX TREE IS TRANSFORMED INTO REL DAG
403
LEAF NODES OF REL DAG ARE ANNOTATED WITH CLUSTERING INFO
404
EXCHANGE NODE IS INSERTED BETWEEN PARENT AND CHILD RELS
405
ALGORITHM IS PERFORMED FOR OPTIMAL MATERIALIZATION SCHEDULE
406
PARALLEL REL DAG TRANSFORMED INTO DAG OF FUNCTION CALLS
407
STATEMENT DAG IS GENERATED (COORDINATOR STATEMENT FOREST)
408
FOREST OF BAT OPERATOR LISTS IS FORMED
409
END

FIG. 4

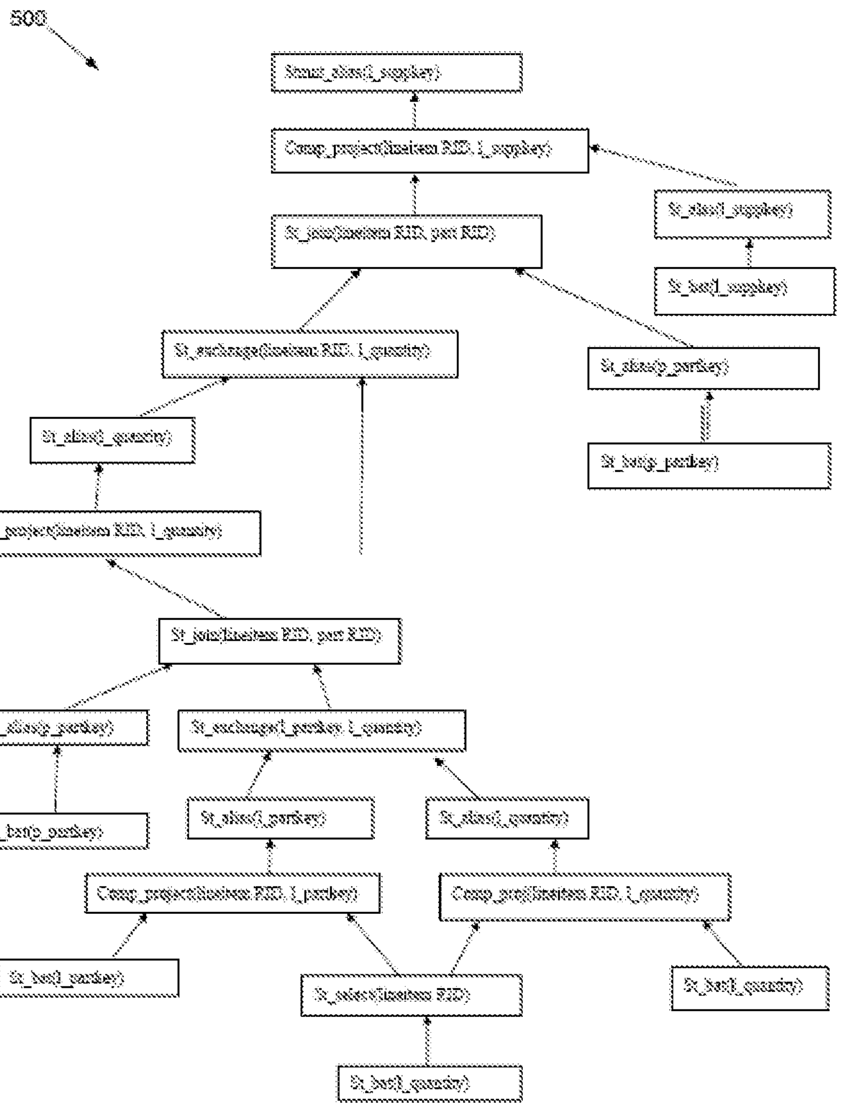
500
Stmat_alias(l_suppkey)
Comp_project(lineitem RID, l_suppkey)
St_alias(l_suppkey)
St_join(lineitem RID, part RID)
St_bat(l_suppkey)
St_exchange(lineitem RID, l_quantity)
St_alias(p_partkey)
St_alias(l_quantity)
St_bat(p_partkey)
Comp_project(lineitem RID, l_quantity)
St_join(lineitem RID, part RID)
St_alias(p_partkey)
St_exchange(l_partkey, l_quantity)
St_bat(p_partkey)
St_alias(l_partkey)
St_alias(l_quantity)
Comp_project(lineitem RID, l_partkey)
Comp_proj(lineitem RID, l_quantity)
St_bat(l_partkey)
St_select(lineitem RID)
St_bat(l_quantity)
St_bat(l_quantity)

FIG. 5

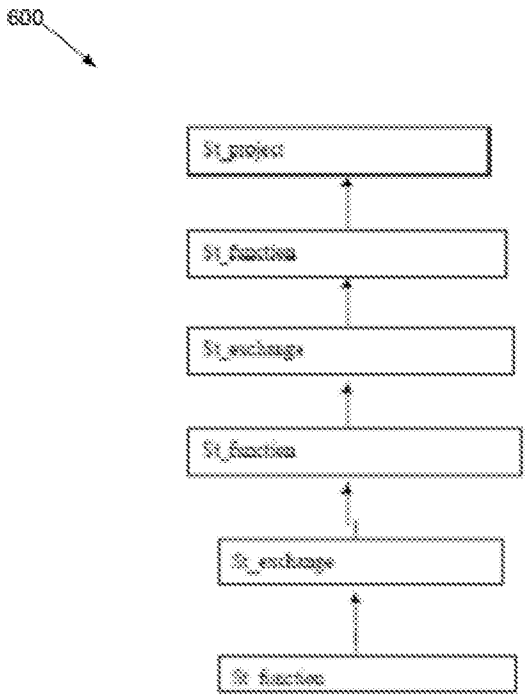
600
St_project
St_function
St_exchange
St_function
St_exchange
St_function

FIG. 6

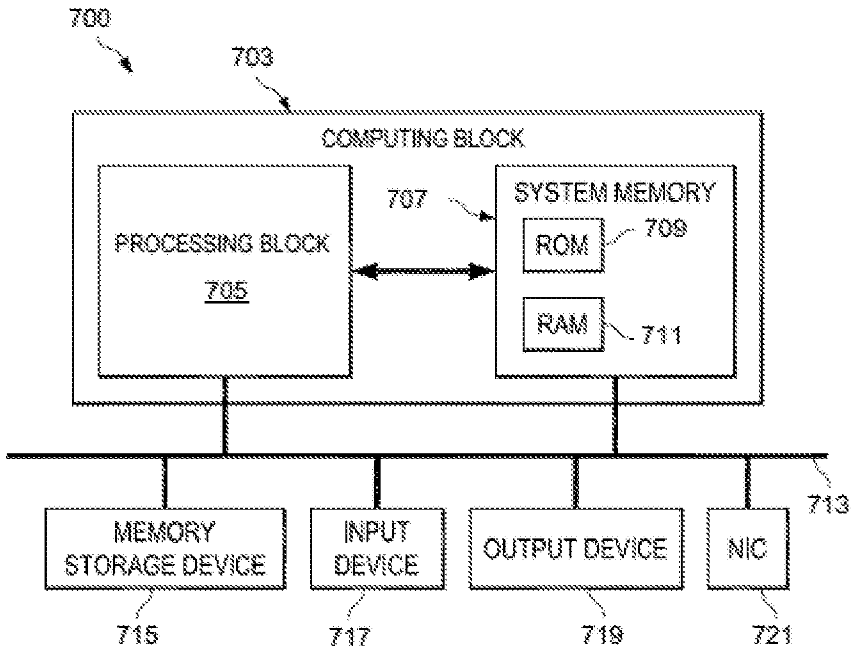
700
703
COMPUTING BLOCK
PROCESSING BLOCK
705
707
SYSTEM MEMORY
ROM
709
RAM
711
713
MEMORY STORAGE DEVICE
INPUT DEVICE
OUTPUT DEVICE
NIC
715
717
719
721

FIG. 7

INTERNATIONAL SEARCH REPORT	International application No. **PCT/CN2015/074819**

A. CLASSIFICATION OF SUBJECT MATTER

G06F 17/30(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

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

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

CPRSABS,CNKI,VEN: establish+, creat+, materializ+, schedul+, relational data base management system, RDBMS, receiv+, query, Rel DAG, DAG, directed acyclic graph, transversal, parallel, column, specific, oriented, comput+, transform, re-shuffl+, statement, forest, BAT, optimal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2014075350 A1 (GAUTHIER, A. ET AL.) 13 March 2014 (2014-03-13) the whole document	1-21
A	EP 2040180 A1 (HASSO PLATTNER INST FUER SOFTWARESYSTEMTECHNIK GMBH) 25 March 2009 (2009-03-25) the whole document	1-21
A	US 2009150413 A1 (ORACLE INTERNATIONAL CORP.) 11 June 2009 (2009-06-11) the whole document	1-21

☐ 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	"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
"E"	earlier application or patent but published on or after the international filing date	"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
"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)	"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
"O"	document referring to an oral disclosure, use, exhibition or other means	"&"	document member of the same patent family
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search **06 June 2015**	Date of mailing of the international search report **19 June 2015**
Name and mailing address of the ISA/CN **STATE INTELLECTUAL PROPERTY OFFICE OF THE P.R.CHINA 6, Xitucheng Rd., Jimen Bridge, Haidian District, Beijing 100088, China**	Authorized officer **YUAN,Cui**
Facsimile No. **(86-10)62019451**	Telephone No. **(86-10)62089566**

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/CN2015/074819

Patent document cited in search report			Publication date (day/month/year)	Patent family member(s)			Publication date (day/month/year)
US	2014075350	A1	13 March 2014	CN	103678452	A	26 March 2014
				EP	2706494	A1	12 March 2014
EP	2040180	A1	25 March 2009		None		
US	2009150413	A1	11 June 2009	US	8078652	B2	13 December 2011
				US	2012036111	A1	09 February 2012