DATA QUERY METHOD, DEVICE, DATABASE SYSTEM AND STORAGE MEDIUM

[0001] The present disclosure claims priority to Chinese Patent Application No. 202311786596.8, filed with the China National Intellectual Property Administration on December 22, 2023, and entitled "DATA QUERY METHOD, DEVICE, DATABASE SYSTEM AND STORAGE MEDIUM", which is incorporated herein by reference in its entirety.

TECHNICAL FIELD

[0002] The present disclosure relates to the field of database technologies, and in particular, to a data query method, a device, a database system, and a storage medium.

BACKGROUND

[0003] Partitioning is a database optimization technique that can divide a large logical table into a plurality of small physical tables according to a certain rule, thereby improving query and maintenance efficiency. In a partitioning process, a database allocates data to different partitions according to a partitioning rule, and indexes and other optimization technologies may be used in the partitions to improve query efficiency. The large logical table is a partitioned table, and the small physical tables are partitions. Each partition independently organizes and manages data and indexes on a storage engine. When join query is performed, if two tables are both partitioned tables, and a partition key is the same as or related to a join key, a partition-wise join can be implemented.

[0004] Currently, the partition-wise join technology has been combined with a multi-node parallel query architecture. Query task split and scheduling are performed based on partition information, so that a plurality of nodes only need to be independently responsible for join operations on partitions allocated to themselves, thereby reducing unnecessary data transmission power consumption between nodes.

[0005] However, a single node in the multi-node parallel query architecture usually further starts a plurality of worker processes (workers) to process in parallel subtasks allocated to the node. Within a single node, fetched partitions have been downgraded to ordinary data tables, and the node needs to rely on a shuffle (redistribution) mechanism for intra-node multi-thread scheduling. As is well known, the shuffle mechanism incurs significant performance overheads, resulting in

poor query execution efficiency within the node.

SUMMARY

[0006] Various aspects of the present disclosure provide a data query method, a device, a database system, and a storage medium, to improve query execution efficiency in a multi-node parallel query scenario.

[0007] An embodiment of the present disclosure provides a query method, where the method is applicable to any execution node among a plurality of execution nodes that process a query task in parallel, and a query object of the query task is a target partitioned table. The method includes:

after a subtask split from the query task is received, determining target partitions allocated to the subtask from the target partitioned table;

configuring a scheduling relationship between the target partitions and worker processes corresponding to the subtask; and

after data corresponding to the target partitions is fetched, scheduling, based on the scheduling relationship, the fetched data to the worker processes associated with the partitions to which the fetched data belongs, to cause the worker processes to perform query.

[0008] An embodiment of the present disclosure further provides an execution node, including a memory, a processor, and a communication component, where

the memory is configured to store one or more computer instructions; and

the processor is coupled to the memory and the communication component, and configured to execute the one or more computer instructions, to perform the foregoing query method.

[0009] An embodiment of the present disclosure further provides a database system, including a plurality of execution nodes that process a query task in parallel, where the plurality of execution nodes respectively receive subtasks split from the query task, a single execution node starts a plurality of parallel worker processes for a subtask received by the single execution node, and the single execution node is configured to perform the foregoing query method.

[0010] An embodiment of the present disclosure further provides a computer-readable storage medium storing computer instructions, where the computer instructions, when executed by one or more processors, cause the one or more processors to perform the foregoing query method.

[0011] In embodiments of the present disclosure, an improved query solution is provided under a multi-node parallel query architecture. After subtasks are distributed to a plurality of parallel execution nodes, the execution nodes can respectively determine target partitions allocated to the subtasks from a target partitioned table, such that partition information in the target

partitioned table can be transparently transmitted to the execution nodes. Based on this, within the execution node, a scheduling relationship between allocated partitions and worker processes started for an allocated subtask within the execution node can be configured, and fetched data is scheduled to corresponding worker processes based on the scheduling relationship. That is, within the execution node, intra-node scheduling can be performed by fully using the obtained partition information. Therefore, a scheduling process within the execution node no longer needs to rely on a shuffle (redistribution) mechanism, so that performance overheads in the intra-node scheduling process can be effectively reduced, thereby improving query execution efficiency in a multi-node parallel query scenario.

BRIEF DESCRIPTION OF DRAWINGS

[0012] The accompanying drawings described herein are intended to provide a further understanding of the present disclosure, and form part of the present disclosure. Schematic embodiments of the present disclosure and descriptions thereof are intended to explain the present disclosure, and do not constitute any inappropriate limitation to the present disclosure.

[0013] FIG. 1 is a schematic structural diagram of a database system according to an exemplary embodiment of the present disclosure.

[0014] FIG. 2 is a schematic logic diagram of an exemplary solution for inter-node scheduling.

[0015] FIG. 3 is a schematic diagram of an internal logical structure of an execution node according to an exemplary embodiment of the present disclosure.

[0016] FIG. 4 is a schematic logic diagram of an implementation solution for configuring a scheduling relationship according to an exemplary embodiment of the present disclosure.

[0017] FIG. 5 is a schematic logic diagram of an improved query solution according to an exemplary embodiment of the present disclosure.

[0018] FIG. 6 is a schematic flowchart of a query method according to another exemplary embodiment of the present disclosure.

[0019] FIG. 7 is a schematic structural diagram of an execution node according to still another exemplary embodiment of the present disclosure.

DESCRIPTION OF EMBODIMENTS

[0020] To make the objectives, technical solutions, and advantages of the present disclosure clearer, the technical solutions of the present disclosure are clearly and completely described below with reference to specific embodiments of the present disclosure and corresponding accompanying

drawings. Clearly, the described embodiments are only some embodiments rather than all the embodiments of the present disclosure. Based on the embodiments of the present disclosure, all other embodiments obtained by a person of ordinary skill in the art without creative efforts fall within the scope of protection of the present disclosure.

[0021] Before the technical solutions provided in the embodiments of the present disclosure are described in detail, several technical concepts involved in the present disclosure are briefly explained below.

[0022] Partitioning: It is a database optimization technique that can divide a large table into a plurality of small tables according to a certain rule, thereby improving query and maintenance efficiency. In a partitioning process, a database allocates data to different partitions according to a partitioning rule, and indexes and other optimization technologies may be used in the partitions to improve query efficiency.

[0023] Partitioned table: In a database, a partitioned table can be used to implement partitioning. A partitioned table refers to splitting a large table into a plurality of small tables, and each small table is referred to as a partition. The partitioned table includes a plurality of partitions, and each partition may correspond to an independent physical storage space.

[0024] Partition-wise join (PartitionWise Joins): Also translated as a partition-based join, it may be understood as breaking a join between partitioned tables into a join between mutually matching partitions according to a join condition (a join key). The partition-based join can be applied when two tables being joined are partitioned on a join key.

[0025] Multi-node parallel query: It may be understood as that a plurality of execution nodes process a same query task in parallel. During actual application, the query task can be split into a plurality of subtasks, which are allocated to a plurality of execution nodes, and the plurality of execution nodes can execute in parallel the subtasks allocated to themselves. Output results of the plurality of subtasks may be aggregated, to generate an output result of the query task. A single execution node can provide a plurality of worker processes to process an allocated subtask in parallel.

[0026] Shuffle (redistribution): It may be simply understood as calculating a hash value for each data record in a data table, and then determining which node or process the data record should be scheduled to based on the hash value. When a query task involves a plurality of data tables, shuffle needs to be performed on all of the plurality of data tables. Shuffle is also involved in the hash join operator. General logic is as follows: full-table hash-value calculation is respectively performed for two data tables on which hash join needs to be performed, and for each data record from one of the tables, full-table probing is performed in the other table, to identify whether a join condition is satisfied. It is well known that the performance overheads of the shuffle are usually

very high.

[0027] As described in the Background, currently, partition-wise join (PartitionWise Joins) mainly uses partition information in a task scheduling process among a plurality of execution nodes. However, scheduling within a single execution node needs to rely on a shuffle (redistribution) mechanism, causing excessive performance overheads within the execution nodes and poor query execution efficiency.

[0028] Therefore, the present disclosure provides an improved query solution under a multi-node parallel query architecture, to optimize intra-node scheduling, thereby improving query execution efficiency.

[0029] The following describes the technical solutions provided in embodiments of the present disclosure in detail with reference to the accompanying drawings.

[0030] FIG. 1 is a schematic structural diagram of a database system according to an exemplary embodiment of the present disclosure. As shown in FIG. 1, the database system includes a plurality of execution nodes. In this embodiment, the database system may use a storage-compute separation architecture, for example, a database such as PolarDB. Referring to FIG. 1, the database system further includes a plurality of storage nodes, forming a storage layer. The execution nodes in this embodiment may be understood as computing nodes, forming a computing layer. A database engine may run in each execution node, to be responsible for computing operations. Data tables are stored in the storage layer, and the execution nodes may fetch needed data from the storage layer according to a computing requirement. The storage-compute separation architecture is not further described herein.

[0031] In this embodiment, a query service is provided in the database system by using a multi-node parallel query architecture. As explained above for the concept of multi-node parallel query, after a user initiates a query task, the query task may be split into a plurality of subtasks, which are allocated to a plurality of execution nodes. In addition, in this embodiment, a data table on which the query task is performed is a partitioned table. In this way, a process of allocating subtasks among the plurality of execution nodes may be implemented according to a scheduling mechanism provided in the foregoing partition-wise join (PartitionWise Joins) technology. In this embodiment, this scheduling process is not limited. FIG. 2 is a schematic logic diagram of an exemplary solution for inter-node scheduling. Referring to FIG. 2, in this exemplary solution, based on a logical structure of a database system, after a user initiates a query statement, a query optimizer may parse the query statement. Subsequently, a database scheduler generates a physical execution plan (as a query task) for the query statement according to a parsing result, and obtains a plurality of shard plans (as subtasks) through splitting based on the physical execution plan. The shard plans have consistent task logic, and are responsible for processing different partitions of a target partitioned

table specified by the query statement. Then, the scheduler may allocate the shard plans to a plurality of executors. A scheduling rule may be load balancing or the like. This is not specifically exemplified herein. The executors are provided by the execution nodes in this embodiment. In this way, a plurality of subtasks split from a query task are allocated to the execution nodes.

[0032] In an existing partition-wise join (PartitionWise Joins) technology, from the perspective of a single execution node, after receiving a subtask, the execution node starts to process the subtask as an ordinary query task. The execution node is no longer aware of partition information of a target partitioned table, but relies on a shuffle mechanism for intra-node scheduling.

[0033] In this embodiment, the intra-node scheduling manner relying on the shuffle mechanism is discarded, and a new intra-node scheduling concept is provided: partition information of a partitioned table is transparently transmitted to the execution node, and the execution node performs intra-node scheduling based on the partition information, without relying on the shuffle mechanism. This concept is explained in detail below.

[0034] For ease of description, this embodiment describes this concept from the perspective of a single execution node. It should be understood that, all execution nodes that process a same query task in parallel may implement intra-node scheduling according to this concept. FIG. 3 is a schematic diagram of an internal logical structure of an execution node according to an exemplary embodiment of the present disclosure. Referring to FIG. 3, in this embodiment, an executor in an execution node may be modified. Specifically, a data distribution component (for example, an exchange operator) in the executor may be modified, so that the execution node performs intra-node scheduling according to the inventive concept provided in this embodiment, thereby improving query execution efficiency.

[0035] Referring to FIG. 3, when a query object of a query task is a target partitioned table, in this embodiment, after receiving a subtask, the execution node can determine target partitions allocated to the subtask from the target partitioned table. During actual application, an identifier of the target partitioned table and identifiers of the target partitions allocated to this execution node from the target partitioned table may be carried in the subtask. In this way, the execution node can determine, by parsing the subtask, the target partitions allocated from the target partitioned table. Certainly, the scheduler in FIG. 2 may alternatively push, to the execution node, the identifiers of the target partitions allocated to the subtask from the target partitioned table. In this embodiment, an implementation of how the execution node learns about the target partitions allocated from the target partitioned table is not limited.

[0036] The target partitioned table is a partitioned table that needs to be queried by the query task. Similarly, each subtask split from the query task also uses the target partitioned table as a query object. However, it should be understood that the subtask does not need to query the entire

target partitioned table, but only needs to query target partitions in the target partitioned table. In addition, the number of target partitioned tables is related to a task type of the query task. For example, if the query task is a join task, there are usually two or more target partitioned tables. If the query task is an aggregation task (namely an agg task), there is usually one target partitioned table. It should be understood that, the query task corresponds to the physical plan mentioned above, and the essence thereof may be understood as a plan tree. The task type herein is defined based on a root node in the plan tree. For example, if the root node in the plan tree is a join operator, the query task is described as a join task. For another example, if the root node in the plan tree is an agg operator, the query task is described as an aggregation task. In this embodiment, other nodes in the plan tree are not limited. For example, if the root node is a join operator, two subnodes are connected to the root node at the layer immediately below the root node, and the two subnodes may each be connected to one or more subnodes. Usually, leaf nodes of the plan tree are scan nodes (scan operators), configured to fetch data for the query task. Types of operators on the subnodes are not limited in this embodiment. In addition, it should be understood that task types of the subtask and the query task are generally consistent. The subtask is also essentially a physical plan. Furthermore, plan logic of the subtask is generally also consistent with that of the query task. However, the subtask may only undertake query of some data in the data that needs to be queried by the query task.

[0037] In this embodiment, there may be one or more target partitions allocated to this execution node from the target partitioned table, which is determined by an inter-node scheduling process. This is not interfered with in this embodiment.

[0038] In this way, partition information of the target partitions is transparently transmitted to the execution node. Based on this, it is proposed in this embodiment that the execution node uses obtained partition information as a basis for intra-node scheduling, and configures a scheduling relationship between the target partitions and worker processes started for the subtask within this execution node. The scheduling relationship herein is used to represent which worker process is responsible for a query operation on which target partition.

[0039] In this embodiment, a policy according to which the scheduling relationship is configured within the execution node based on the partition information is not limited. The basic concept herein is that intra-node scheduling is performed within the execution node with a partition as a unit, and the intra-node scheduling with a data record as a unit as in a conventional shuffle mechanism is no longer performed.

[0040] For example, if the subtask is a join task that instructs data records with a same id in a partitioned table A and a partitioned table B to be joined and six target partitions with id=0, id=2, id=4, id=6, id=8, and id=10 are allocated to the execution node, the execution node may configure,

during intra-node scheduling, a scheduling relationship between these six target partitions and three worker processes started for the subtask within the execution node. For example, each worker process may be responsible for two of the target partitions.

[0041] Based on this, the execution node may perform query according to an execution plan of the subtask. As mentioned above, there is a scan node in the execution plan, and the scan node is configured to fetch data in target partitions. Based on that the scheduling relationship between the target partitions and the worker processes started for the subtask within this execution node has been configured, the execution node may schedule fetched data to a worker process associated with a partition to which the fetched data belongs. Following the foregoing example, if there is a scheduling relationship between the target partition with id=0 and the worker process 0, after data from the target partition with id=0 is fetched, the execution node may automatically schedule the data to the worker process 0 to perform query, without the need for shuffle.

[0042] During actual application, data fetching is usually performed in batches, that is, data is fetched batch by batch. In this embodiment, each time the execution node completes fetching of a batch of data, the execution node may schedule the batch of data to the appropriate worker process(es), or certainly, the execution node may perform a data scheduling operation after data needed by the subtask is completely fetched in a plurality of batches. The timing of execution of the data scheduling operation is not limited in this embodiment. Preferably, the data scheduling operation may be performed in batches, to improve query execution efficiency.

[0043] To sum up, in this embodiment, an improved query solution is provided under the multi-node parallel query architecture. After the subtasks are distributed to a plurality of parallel execution nodes, the execution nodes can respectively determine the target partitions allocated to the subtasks from a target partitioned table, such that the partition information in the target partitioned table can be transparently transmitted to the execution nodes. Based on this, within the execution node, a scheduling relationship between the allocated partitions and the worker processes started for the allocated subtask within this node can be configured, and the fetched data is scheduled to the corresponding worker processes based on the scheduling relationship. That is, within the execution node, intra-node scheduling can be performed by fully using the obtained partition information. Therefore, the scheduling process within the execution node no longer needs to rely on a shuffle (redistribution) mechanism, so that performance overheads in the intra-node scheduling process can be effectively reduced, thereby improving query execution efficiency in the multi-node parallel query scenario.

[0044] FIG. 4 is a schematic logic diagram of an implementation solution for configuring a scheduling relationship according to an exemplary embodiment of the present disclosure. In the foregoing or following embodiments, a plurality of implementations may be used to support an

execution node to configure a scheduling relationship between target partitions and worker processes started for an allocated subtask within this execution node. FIG. 4 provides a preferred implementation.

[0045] Referring to FIG. 4, in this preferred implementation, an execution node may: obtain identifiers of target partitions; map the identifiers of the target partitions to partition numbers within this execution node; and establish a scheduling relationship between the partition numbers within this execution node and worker processes within this execution node, where a consecutive numbering mechanism is adopted for the partition numbers within this execution node.

[0046] The identifiers of the target partitions are identifiers generated when the partitions are defined in the target partitioned table. For example, "id=1" in the foregoing example may be used as an identifier of a target partition. It is proposed in this implementation to define intra-node partition numbers for an intra-node scheduling process. It should be understood that, an executor within the execution node is specifically the data distribution component mentioned above, and what are perceived by the data distribution component in the intra-node scheduling process are the intra-node partition numbers proposed in this embodiment, rather than original identifiers of the target partitions.

[0047] In this way, by the use of unified partition numbers, the data distribution component can deal with different subtasks according to a general scheduling rule. Moreover, the consecutive numbering mechanism can also prevent a problem such as data skew in the intra-node scheduling process, which would otherwise cause excessive partitions to be allocated to some worker processes.

[0048] It should be understood that in this preferred implementation, partitions in the target partitioned table are not destroyed, but the original partition identifiers of the target partitions are converted into self-defined partition numbers within the execution node, and the execution node performs intra-node scheduling based on the partition numbers.

[0049] Referring to FIG. 4, the target partitioned table includes four partitions, namely "split0: partition0", "split1: partition1", "split2: partition2", and "split3: partition3". After the inter-node scheduling process, two target partitions, namely "split0: partition0" and "split2: partition2" are allocated to node0 (execution node 0). Based on this, in node0, the foregoing mapping operation may be performed, so that "split0: partition0" is mapped to a partition number 0, and "split2: partition2" is mapped to a partition number 1. A similar mapping operation is also performed in node1. Based on this, in node0, the partition number 0 and the partition number 1 may be allocated to parallel worker processes within the node, to establish a scheduling relationship between the partition numbers within this execution node and the worker processes within this execution node.

[0050] It can be learned from the example in FIG. 4 that, in this preferred implementation, the

use of the consecutive numbering mechanism can further bring another technical advantage, that is, the total number of allocated target partitions within the execution node can be reflected. This indicator can be used as one of parameters of intra-node scheduling, to more properly establish a scheduling relationship between the partition numbers within this execution node and the worker processes within this execution node.

[0051] Based on this, further, in this preferred implementation solution, the execution node may determine the total number of partitions within this execution node based on the partition numbers generated within this execution node due to the mapping operation, and respectively determine, for the worker processes within this execution node, partition numbers for which the worker processes are responsible, according to the total number of partitions and a degree of parallelism of the worker processes, to generate the scheduling relationship between the partition numbers within this execution node and the worker processes within this execution node. Due to the use of the consecutive numbering mechanism, after the mapping operation is completed, a difference between the maximum partition number and the minimum partition number plus one is the total number of partitions within this execution node. During actual application, after the total number of partitions is determined, division calculation may be performed between the total number of partitions and the degree of parallelism of the worker processes within this execution node, to determine how many target partitions can be allocated to each worker process, and then a needed number of partition numbers may be sequentially selected for the worker processes.

[0052] For example, if partition numbers generated in the execution node are 0, 1, 2, 3, 4, and 5 and the degree of parallelism of the worker processes started for a subtask in the execution node is 3, two partition numbers may be allocated to each worker process. Then, according to a sequential number-taking scheme, the partition numbers 0 and 1 may be allocated to the worker process 0, the partition numbers 2 and 3 may be allocated to the worker process 1, and the partition numbers 4 and 5 may be allocated to the worker process 2, to establish a scheduling relationship between the partition numbers within this execution node and the worker processes within this execution node.

[0053] It should be understood that, in this preferred implementation solution, other mechanisms may alternatively be used to establish, for the subtask, the scheduling relationship between the partition numbers within the execution node and the worker processes within the execution node, and is not limited to the foregoing mechanism based on the total number of partitions and the degree of parallelism. No more examples are provided herein.

[0054] In addition, in this preferred implementation solution, if the subtask is a join task, as mentioned above, there are a plurality of target partitioned tables to which the subtask points. In this case, in a process of performing the foregoing mapping operation, same identifiers of partitions

allocated to this execution node from the plurality of target partitioned tables are mapped to a same partition number within this execution node.

[0055] For example, the subtask executes a partitioned table A and a partitioned table B. The two partitioned tables use id as a partition key. The partitioned table A includes several partitions with id=0, id=1, id=2, id =3, id =4, and id=5. The partitioned table B includes several partitions with id=0, id=1, id=2, id =3, id =4, and id=5. If the target partitions allocated to the execution node are partitions with id=0, id=1, and id=2, the partition with id=0 in the partitioned table A and the partition with id=0 in the partitioned table B need to be mapped to a same partition number in the execution node, for example, mapped to the partition number 0. Similarly, other same identifiers of partitions in the partitioned table A and the partitioned table B should also be mapped according to this principle. This can ensure that when there are a plurality of target partitioned tables, the partition numbers after the mapping still maintain an original pairing relationship between partitions.

[0056] It should be understood that, the foregoing implementation in which the execution node is supported by the mapping operation to configure the scheduling relationship between the target partitions and the worker processes started for the subtask in this execution node is preferable. In this embodiment, other implementations may alternatively be used to support configuration of this scheduling relationship, and this embodiment is not limited thereto. For example, allocation may be directly performed according to identifiers of target partitions, and no more examples are provided herein.

[0057] To sum up, in this embodiment, various implementations may be used to support the configuration within the execution node for a scheduling relationship between target partitions and worker processes started for the subtask in this execution node, so that intra-node scheduling is implemented with a partition as a unit, and target partitions that are allocated to these worker processes can be non-overlapping, thereby performing the subtask in parallel more efficiently and accurately.

[0058] FIG. 5 is a schematic logic diagram of an improved query solution according to an exemplary embodiment of the present disclosure. Referring to FIG. 5, after intra-node scheduling is completed, each worker process may perform, according to an execution plan, a query operation on partition data scheduled to the worker process.

[0059] In addition to the foregoing modifications to the data exchange component (exchange operator) in the execution node, it is further proposed in this embodiment to also modify a hash join operator in the execution node, to optimize performance of the hash join operator. In a conventional multi-node parallel query scenario, a hash join operator within an execution node can only perform a hash join operation according to the shuffle mechanism explained above, which

incurs excessive performance overheads, resulting in poor execution efficiency of the hash join operator.

[0060] Referring to the improved query solution provided in FIG. 5, in this embodiment, the hash join operator is modified, so that the hash join operator can also make full use of partition information, to support independent hash join operations respectively performed for different partitions. In this way, a hash join operation performed for a single partition no longer needs to involve data within other partitions, and a full-table shuffle can be avoided, thereby greatly reducing performance overheads of the hash join operator, and optimizing hash join operation efficiency.

[0061] Based on this, if a query subplan is a hash join task, partition information may be fully used by a modified hash join operator, to optimize hash join operation efficiency.

[0062] In this embodiment, it is supported that a single worker process is independently responsible for a single target partition, and it is also supported that a plurality of worker processes are jointly responsible for a single target partition. The two cases are described separately below.

[0063] When a single worker process is independently responsible for a single target partition, from the perspective of the single worker process, the single worker process can be independently responsible for one or more target partitions. When the single worker process is independently responsible for a plurality of target partitions, a modified hash join operator may be used to respectively perform hash join operations on different target partitions for which the worker process is responsible. The hash join operations performed for different target partitions do not interfere with each other.

[0064] For example, with a target process as an example, if the target process is independently responsible for two target partitions with partition numbers 0 and 1 in each of the partitioned table A and the partitioned table B, the target process may respectively build independent hash tables for the two target partitions in the partitioned table A by using a build side of a modified hash join operator. For example, the hash tables are marked as a hash table 0 and a hash table 1. Then, by using a probe side in the hash join operator, the target partition with the partition number 0 in the partitioned table B may be probed based on the hash table 0; and the target partition with the partition number 1 in the partitioned table B may be probed based on the hash table 1. Clearly, the modified hash join operator no longer needs to perform full-table probing as in a conventional shuffle mechanism.

[0065] Still referring to FIG. 5, considering that partition information needs to be used in the hash join operator, in this embodiment, it is further proposed to also modify an operator that does not disrupt partitions, such as a project operator and/or a filter operator in the executor. The main modification is as follows. When data passes through such operators, partition numbers of

remaining data are retained, so that the remaining data still carries the partition numbers when transferred to the hash join operator, thereby ensuring that the hash join operator can still separately perform hash join according to partitions.

[0066] During research, the inventors have found that, before the hash join operator, there may be some special operators that can disrupt partitions, for example, an aggregation (agg) operator, a sort operator, and other join operators. After the data passes through such special operators, the data no longer conforms to an original partition rule. In view of this, in this embodiment, after the data passes through such special operators, related partition numbers may be discarded, and the hash join operator returns to a conventional shuffle mechanism to perform a hash join operation, so as to ensure accuracy of an output result.

[0067] When a plurality of worker processes are jointly responsible for a single target partition, a modification solution for the hash join operator is also proposed in this embodiment. A target partition shared by a plurality of worker processes is described as a shared partition herein. During research, the inventors have found that in a database system, generally, a build process of hash join is a blocking process, that is, the "probe" certainly occurs after the "build" is completed. In the probe process, no update or deletion operations are performed on a hash table, and therefore, this process is actually a complete read-only process. Based on this, the probe process does not involve the problem in multi-process concurrency control. Therefore, in this embodiment, the modification focuses on the build process. That is, a modification solution for the hash join operator is provided, so as to solve the problem in multi-process concurrency control in this case.

[0068] A conventional solution to solve the problem in multi-process concurrency control is usually implemented based on a lock mechanism and the like, and the existing solution is not described in detail herein. In this embodiment, a new solution is implemented by modifying the hash join operator:

obtaining, by using the hash join operator in the target process, the total number of times of batch processing corresponding to the shared partition and the total number of worker processes responsible for the shared partition;

determining, by using the hash join operator, a data batch allocated to the target process, according to a preset allocation rule, the total number of times of batch processing, and the total number of worker processes; and

determining, by using the hash join operator, whether the current data batch is the data batch allocated to the target process, and if so, starting a build side of the hash join operator to build, for the current data batch, a hash table for supporting hash join calculation, to cause the worker processes jointly responsible for the shared partition to cooperatively build a hash table corresponding to the shared partition.

[0069] In this new solution, the preset allocation rule based on the total number of times of batch processing and the total number of worker processes is defined in hash join operators. In this way, each hash join operator of the hash join operators jointly responsible for the shared partition may determine, according to the preset allocation rule, data batches allocated to this hash join operator, after obtaining the total number of times of batch processing corresponding to the shared partition and the total number of worker processes responsible for the shared partition. As mentioned above, in an execution process of the subtask, data is fetched in batches. The total number of times of batch processing corresponding to the shared partition may be pre-generated by a scheduler inside the execution node or pre-generated in a data distribution component, and is notified to the hash join operators jointly responsible for the shared partition. Similarly, the total number of worker processes responsible for the shared partition may also be notified by the data distribution component to the hash join operators jointly responsible for the shared partition. In this way, each hash join operator of the hash join operators jointly responsible for the shared partition can obtain the two parameters, and then determine data batches that should be allocated to this hash join operator.

[0070] Based on this, from the perspective of a single hash join operator, the single hash join operator can determine whether the current data batch is a data batch that should be allocated to the single hash join operator, and if so, start a build side of the hash join operator to build, for the current data batch, a hash table used to support hash join calculation. This can not only ensure that hash tables respectively built by the hash join operators jointly responsible for the shared partition do not overlap, but also ensure that no data in the shared partition is missed, thereby effectively solving the problem in multi-process concurrency control.

[0071] In this new solution, a specific mechanism in the preset allocation rule is not limited. For example, a load balancing mechanism may be used, to allocate a substantially equal amount of data batches to each hash join operator. Certainly, a preemption mechanism may alternatively be used. The hash join operators perform preemption on the current data batch, and the hash join operator that successfully preempts the current data batch is responsible for building a hash table for the current data batch. No more examples of mechanisms are provided herein, and this embodiment is not limited thereto. Subsequently, the hash tables respectively built by the hash join operators jointly responsible for the shared partition may be merged, to generate a hash table corresponding to the shared partition.

[0072] To sum up, in this embodiment, modification solutions for more operators other than the data distribution component (the exchange operator) in the execution node are provided, so that after intra-node scheduling is completed by using the partition information, the partition information can continue to be fully used, to optimize execution performance of a plurality of

operators in the execution plan, thereby improving execution efficiency for the subtask.

[0073] During research, the inventors have further found that in the conventional multi-node parallel query architecture, in addition to the foregoing intra-node scheduling process and hash join operators, a process of data interaction between nodes also needs to rely on a shuffle mechanism. That is, the execution node needs to determine, by using the shuffle mechanism, which data needs to be transmitted to other nodes. This also causes a large amount of performance overheads in the execution node, and also reduces query execution efficiency.

[0074] In view of this, in this embodiment, an optimization solution is further proposed for an inter-node data exchange process. The data distribution component (for example, an exchange operator) in the execution node is usually responsible for an inter-node data exchange operation. Therefore, this optimization solution may be implemented by further modifying the data distribution component (for example, the exchange operator) in the execution node.

[0075] It is proposed in this optimization solution that the database scheduler for distributing subtasks as shown in FIG. 2 may construct a partition pruning list for the target partitioned table, and send the partition pruning list to the execution nodes where the subtasks reside. The partition pruning list includes a list of partitions in the target partitioned table that do not affect an output result after being pruned. The database scheduler may complete construction and sending of the partition pruning list before distributing the subtasks, to ensure that the execution nodes where the subtasks reside can obtain the partition pruning list in time.

[0076] During actual application, the database scheduler may identify, according to a parsing result corresponding to a query statement of the query task, partitions in the target partitioned table that are designated as needing to be pruned, identify partitions with null data in the target partitioned table by using attribute description information corresponding to the target partitioned table, and record identifiers of the two types of partitions in the partition pruning list. Certainly, the partition pruning list may further record a list of other partitions that do not affect an output result after being pruned, which are not limited to the two types exemplarily shown herein, and no more examples are provided herein.

[0077] Based on this, in this optimization solution, the execution node may: obtain a partition pruning list constructed for the target partitioned table; if this execution node needs to exchange data with another node, perform, according to the partition pruning list, pruning on the target partitions allocated to this execution node from the target partitioned table, and then construct exchange data for the another node based on remaining partitions; and use the exchange data constructed based on the remaining partitions as response data when a data exchange request is received. That is, the partitions in the partition pruning list that are among the target partitions allocated to this execution node are pruned, and exchange data can be constructed under remaining

partitions among the target partitions allocated to this execution node, whereas there is no need to construct exchange data for the pruned target partitions.

[0078] As mentioned above, if the subtask is a join task, there are usually a plurality of target partitioned tables. In this case, when applying the partition pruning list, the execution node may perform synchronous pruning in a plurality of target partitioned tables. In other words, if there are a plurality of target partitioned tables and to-be-pruned partitions in the partition pruning list are present among the target partitions, the execution node may prune all to-be-pruned partitions allocated to this execution node from the plurality of target partitioned tables.

[0079] For example, if the subtask points to partitions with id=0 and id=1 in the partitioned table A and the partitioned table B, because data in the partition with id=0 in the partitioned table A is null, the partition with id=0 is included in the partition pruning list. Based on this, after the execution node fetches the partitions with id=0 and id=1 from the partitioned table A and the partitioned table B, the partition with id=0 in the partitioned table A may be pruned. In addition, although data in the partition with id=0 in the partitioned table B is not null, the execution node still prunes the partition with id=0 in the partitioned table B.

[0080] To sum up, in this preferred solution, a “logic link of constructing exchange data” inside the execution node remains unchanged. However, before initiating the “logic link of constructing exchange data”, the execution node performs pruning on the target partitions allocated to this execution node from the target partitioned table based on the partition pruning list, and the remaining partitions after the pruning serve as an operation range for the “logic link of constructing exchange data”. This can effectively reduce the operation range of performing the “logic link of constructing exchange data” inside the execution node. Therefore, a data transmission amount in a process of data exchange between nodes can be effectively reduced, thereby further improving query execution efficiency.

[0081] FIG. 6 is a schematic flowchart of a query method according to another exemplary embodiment of the present disclosure. The method is applicable to any execution node among a plurality of execution nodes that process a query task in parallel, and a query object of the query task is a target partitioned table. The method may be implemented by an executor in the execution node. Referring to FIG. 6, the method may include the following steps.

[0082] Step 600: After a subtask split from the query task is received, determine target partitions allocated to the subtask from the target partitioned table.

[0083] Step 601: Configure a scheduling relationship between the target partitions and worker processes started for the subtask within the current execution node.

[0084] Step 602: After data corresponding to the target partitions is fetched, schedule, based on the scheduling relationship, the fetched data to the worker processes associated with the

partitions to which the data belongs, to cause the worker processes to perform query.

[0085] In an optional embodiment, the configuring a scheduling relationship between the target partitions and worker processes started for the subtask within the current execution node includes:

obtaining identifiers of the target partitions;

mapping the identifiers of the target partitions to partition numbers within the current execution node; and

establishing a scheduling relationship between the partition numbers within the current execution node and worker processes within the current execution node;

where a consecutive numbering mechanism is adopted for the partition numbers within the current execution node.

[0086] In an optional embodiment, in a case where the subtask is a join query task, there are a plurality of target partitioned tables, and the mapping the identifiers of the target partitions to partition numbers within the current execution node includes:

mapping same identifiers of partitions allocated to the current execution node from the plurality of target partitioned tables to a same partition number within the current execution node.

[0087] In an optional embodiment, the establishing a scheduling relationship between the partition numbers within the current execution node and worker processes within the current execution node includes:

determining a total number of partitions within the current execution node based on the partition numbers generated within the current execution node due to the mapping operation; and

respectively determining, for the worker processes within the current execution node, partition numbers for which the worker processes are responsible, according to the total number of partitions and a degree of parallelism of the worker processes, to generate the scheduling relationship between the partition numbers within the current execution node and the worker processes within the current execution node.

[0088] In an optional embodiment, if the subtask is a hash join task and a target process is independently responsible for a plurality of target partitions, the method further includes:

if a project operator and/or a filter operator is present before a hash join operator in an execution plan of the target process, retaining partition numbers of remaining data after data scheduled to the target process passes through the project operator or the filter operator; and

performing, in the hash join operator, hash join operations respectively on different target partitions for which the hash join operator is independently responsible;

where the target process is any worker process within the current execution node.

[0089] In an optional embodiment, the method further includes:

if a special operator capable of disrupting partitions is present before the hash join operator in the execution plan of the target process, discarding related partition numbers after data passes through the special operator, to cause the hash join operator to perform the hash join operations according to a redistribution (shuffle) mechanism.

[0090] In an optional embodiment, if the target process and other worker processes are jointly responsible for a shared partition, the method further includes:

obtaining, by using the hash join operator in the target process, a total number of times of batch processing corresponding to the shared partition and a total number of worker processes responsible for the shared partition;

determining, by using the hash join operator, a data batch allocated to the target process, according to a preset allocation rule, the total number of times of batch processing, and the total quantity of worker processes; and

determining, by using the hash join operator, whether the current data batch is the data batch allocated to the target process, and if so, starting a build side of the hash join operator to build, for the current data batch, a hash table for supporting hash join calculation, to cause worker processes jointly responsible for the shared partition to cooperatively build a hash table corresponding to the shared partition.

[0091] In an optional embodiment, the preset allocation rule includes performing allocation according to a load balancing mechanism.

[0092] In an optional embodiment, the method further includes:

obtaining a partition pruning list constructed for the target partitioned table, where the partition pruning list includes a list of partitions that do not affect a query result after being pruned;

if the current execution node needs to exchange data with another node, performing, according to the partition pruning list, pruning on the target partitions allocated to the current execution node from the target partitioned table, and then constructing exchange data for the another node based on remaining partitions; and

using the exchange data constructed based on the remaining partitions as response data when a data exchange request is received.

[0093] In an optional embodiment, the performing, according to the partition pruning list, pruning on the target partitions allocated to the current execution node from the target partitioned table includes:

if there are a plurality of target partitioned tables and to-be-pruned partitions in the partition pruning list are present among the target partitions, pruning all to-be-pruned partitions allocated to the current execution node from the plurality of target partitioned tables.

[0094] In an optional embodiment, the partition pruning list is generated by a database

scheduler distributing the subtask, and the database scheduler is configured to identify, according to a parsing result corresponding to a query statement of the query task, partitions in the target partitioned table that are designated as needing to be pruned, and identify partitions with null data in the target partitioned table by using attribute description information corresponding to the target partitioned table, to construct the partition pruning list.

[0095] It should be noted that, some processes described in the foregoing embodiments and accompanying drawings include a plurality of operations presented in a specific order. However, it should be clearly understood that these operations may be performed out of the order presented herein or in parallel. Sequence numbers of the operations, such as 601 and 602, are merely used to distinguish between different operations, and the sequence numbers do not indicate any execution order.

[0096] FIG. 7 is a schematic structural diagram of an execution node according to still another exemplary embodiment of the present disclosure. As shown in FIG. 7, the execution node is any one of a plurality of execution nodes that process a query task in parallel, and a query object of the query task is a target partitioned table. The execution node may include: a memory 70, a processor 71, and a communication component 72.

[0097] The processor 71 is coupled to the memory 70 and the communication component 72, and is configured to execute a computer program in the memory 70, so as to:

after a subtask split from the query task is received, determine target partitions allocated to the subtask from the target partitioned table;

configure a scheduling relationship between the target partitions and worker processes started for the subtask within the current execution node; and

after data corresponding to the target partitions is fetched, schedule, based on the scheduling relationship, the fetched data to the worker processes associated with the partitions to which the data belongs, to cause the worker processes to perform query.

[0098] In an optional embodiment, when configuring the scheduling relationship between the target partitions and the worker processes started for the subtask within the current execution node, the processor 71 may be specifically configured to:

obtain identifiers of the target partitions;

map the identifiers of the target partitions to partition numbers within the current execution node; and

establish a scheduling relationship between the partition numbers within the current execution node and worker processes within the current execution node;

where a consecutive numbering mechanism is adopted for the partition numbers within the current execution node.

[0099] In an optional embodiment, in a case where the subtask is a join query task, there are a plurality of target partitioned tables, and when mapping the identifiers of the target partitions to the partition numbers within the current execution node, the processor 71 may be specifically configured to:

map same identifiers of partitions allocated to the current execution node from the plurality of target partitioned tables to a same partition number within the current execution node.

[0100] In an optional embodiment, when establishing the scheduling relationship between the partition numbers within the current execution node and the worker processes within the current execution node, the processor 71 may be specifically configured to:

determine a total number of partitions within the current execution node based on the partition numbers generated within the current execution node due to the mapping operation; and

respectively determine, for the worker processes within the current execution node, partition numbers for which the worker processes are responsible, according to the total number of partitions and a degree of parallelism of the worker processes, to generate the scheduling relationship between the partition numbers within the current execution node and the worker processes within the current execution node.

[0101] In an optional embodiment, if the subtask is a hash join task and a target process is independently responsible for a plurality of target partitions, the processor 71 may be further configured to:

if a project operator and/or a filter operator is present before a hash join operator in an execution plan of the target process, retain partition numbers of remaining data after data scheduled to the target process passes through the project operator or the filter operator; and

perform, in the hash join operator, hash join operations respectively on different target partitions for which the hash join operator is independently responsible;

where the target process is any worker process within the current execution node.

[0102] In an optional embodiment, the processor 71 may be further configured to:

if a special operator capable of disrupting partitions is present before the hash join operator in the execution plan of the target process, discard related partition numbers after data passes through the special operator, to cause the hash join operator to perform the hash join operations according to a redistribution (shuffle) mechanism.

[0103] In an optional embodiment, if the target process and other worker processes are jointly responsible for a shared partition, the processor 71 may be further configured to:

obtain, by using the hash join operator in the target process, a total number of times of batch processing corresponding to the shared partition and a total number of worker processes responsible for the shared partition;

determine, by using the hash join operator, a data batch allocated to the target process, according to a preset allocation rule, the total number of times of batch processing, and the total number of worker processes; and

determine, by using the hash join operator, whether the current data batch is the data batch allocated to the target process, and if so, starting a build side of the hash join operator to build, for the current data batch, a hash table for supporting hash join calculation, to cause worker processes jointly responsible for the shared partition to cooperatively build a hash table corresponding to the shared partition.

[0104] In an optional embodiment, the preset allocation rule includes performing allocation according to a load balancing mechanism.

[0105] In an optional embodiment, the processor 71 may be further configured to:

obtain a partition pruning list constructed for the target partitioned table, where the partition pruning list includes a list of partitions that do not affect a query result after being pruned;

if the current execution node needs to exchange data with another node, perform, according to the partition pruning list, pruning on the target partitions allocated to the current execution node from the target partitioned table, and then construct exchange data for the another node based on remaining partitions; and

use the exchange data constructed based on the remaining partitions as response data when a data exchange request is received.

[0106] In an optional embodiment, when performing, according to the partition pruning list, pruning on the target partitions allocated to the current execution node from the target partitioned table, the processor 71 may be specifically configured to:

if there are a plurality of target partitioned tables and to-be-pruned partitions in the partition pruning list are present among the target partitions, prune all to-be-pruned partitions allocated to the current execution node from the plurality of target partitioned tables.

[0107] In an optional embodiment, the partition pruning list is generated by a database scheduler distributing the subtask, and the database scheduler is configured to identify, according to a parsing result corresponding to a query statement of the query task, partitions in the target partitioned table that are designated as needing to be pruned, and identify partitions with null data in the target partitioned table by using attribute description information corresponding to the target partitioned table, to construct the partition pruning list.

[0108] Further, as shown in FIG. 7, the execution node further includes other components such as a power supply component 73. Only some components are schematically provided in FIG. 7, which does not mean that the execution node includes only the components shown in FIG. 7.

[0109] It should be noted that, for technical details of the foregoing embodiments of the

execution node, reference may be made to related descriptions of the execution node in the foregoing system embodiments. To save space, details are not described herein, but this should not cause a loss to the scope of protection of the present disclosure.

[0110] Correspondingly, an embodiment of the present disclosure further provides a computer-readable storage medium storing a computer program. When the computer program is executed, the steps performed in the foregoing method embodiments can be implemented.

[0111] An embodiment of the present disclosure further provides a computer program. The computer program, when executed on a computer, causes the computer to perform the steps performed in the foregoing method embodiments.

[0112] The memory in FIG. 7 is configured to store a computer program, and may be configured to store various other data to support operations on a computing platform. Examples of the data include instructions for any application or method used to operate on the computing platform, contact data, telephone directory data, messages, pictures, videos, and the like. The memory may be implemented by any type of volatile or nonvolatile storage devices or a combination thereof, such as a static random access memory (SRAM), an electrically erasable programmable read-only memory (EEPROM), an erasable programmable read-only memory (EPROM), a programmable read-only memory (PROM), a read-only memory (ROM), a magnetic memory, a flash memory, a disk, or an optical disc.

[0113] The communication component in FIG. 7 is configured to facilitate communication in a wired or wireless manner between a device where the communication component is located and another device. The device where the communication component is located may access a wireless network based on a communication standard, such as a mobile communication network such as WiFi, 2G, 3G, 4G/LTE, or 5G, or a combination thereof. In an exemplary embodiment, the communication component receives a broadcast signal or broadcast-related information from an external broadcast management system through a broadcast channel. In an exemplary embodiment, the communication component further includes a near field communication (NFC) module, to promote short range communication. For example, the NFC module may be implemented based on a radio frequency identification (RFID) technology, an infrared data association (IrDA) technology, an ultra-wideband (UWB) technology, a Bluetooth (BT) technology, and other technologies.

[0114] The power supply component in FIG. 7 provides power to various component of a device where the power supply component is located. The power supply component may include a power management system, one or more power supplies, and other components associated with power generation, management, and distribution of the device where the power supply component is located.

[0115] A person skilled in the art should understand that the embodiments of the present disclosure may be provided as a method, a system, or a computer program product. Therefore, the present disclosure may use a form of hardware-only embodiments, software-only embodiments, or embodiments combining software and hardware. In addition, the present disclosure may use a form of a computer program product implemented on one or more computer-usable storage media (including but not limited to a disk memory, a CD-ROM, an optical memory, and the like) including computer-usable program code.

[0116] The present disclosure is described with reference to flowcharts and/or block diagrams of the method, device (system), and computer program product in the embodiments of the present disclosure. It should be understood that computer program instructions can implement each flow and/or block in the flowcharts and/or block diagrams and a combination of flows and/or blocks in the flowcharts and/or block diagrams. These computer program instructions may be provided to a general-purpose computer, a special-purpose computer, an embedded processor, or processors of other programmable data processing devices to generate a machine, so that an apparatus configured to implement functions specified in one or more flows in the flowcharts and/or one or more blocks in the block diagrams is generated by using instructions executed by the computer or the processors of other programmable data processing devices.

[0117] These computer program instructions may alternatively be stored in a computer-readable memory that can instruct a computer or other programmable data processing devices to work in a specific manner, so that the instructions stored in the computer-readable memory generate an artifact that includes an instruction apparatus. The instruction apparatus implements functions specified in one or more flows in the flowcharts and/or in one or more blocks in the block diagrams.

[0118] These computer program instructions may also be loaded onto a computer or other programmable data processing devices, so that a series of operation steps are performed on the computer or the other programmable devices, thereby generating computer-implemented processing. Therefore, the instructions executed on the computer or the other programmable devices provide steps for implementing functions specified in one or more flows in the flowcharts and/or in one or more blocks in the block diagrams.

[0119] It should be further noted that the terms “include”, “comprise”, or any variants thereof are intended to cover a non-exclusive inclusion. Therefore, a process, method, article, or device that includes a series of elements not only includes such elements, but also includes other elements not specified expressly, or includes inherent elements of the process, method, article, or device. Unless otherwise specified, an element limited by “include a/an/one...” does not exclude other same elements existing in the process, method, article, or device that includes the element.

[0120] It should be noted that user information (including but not limited to user device information, user personal information, etc.) and data (including but not limited to data for analysis, stored data, displayed data, etc.) involved in the present disclosure are information and data for which an authorization is obtained from the user or a full authorization is obtained from each party, and the collection, use, and processing of relevant data need to comply with relevant laws, regulations, and standards of relevant countries and regions, for which corresponding operation entries are provided for the user to choose to authorize or deny.

[0121] The foregoing descriptions are merely embodiments of the present disclosure and are not intended to limit the present disclosure. For a person skilled in the art, various modifications and variations can be made to the present disclosure. Any modifications, equivalent substitutions, improvements, and the like made within the spirit and principle of the present disclosure should fall within the scope of protection of the present disclosure.

CLAIMS

1. A data query method, wherein the method is applicable to any execution node among a plurality of execution nodes that process a query task in parallel, a query object of the query task is a target partitioned table, and the method comprises:

after a subtask split from the query task is received, determining target partitions allocated to the subtask from the target partitioned table;

configuring a scheduling relationship between the target partitions and worker processes corresponding to the subtask; and

after data corresponding to the target partitions is fetched, scheduling, based on the scheduling relationship, the fetched data to the worker processes associated with the partitions to which the data belongs, to cause the worker processes to perform query.

2. The method according to claim 1, wherein the configuring a scheduling relationship between the target partitions and worker processes corresponding to the subtask comprises:

obtaining identifiers of the target partitions;

mapping the identifiers of the target partitions to partition numbers within the execution node; and

establishing a scheduling relationship between the partition numbers within the execution node and worker processes within the execution node;

wherein a consecutive numbering mechanism is adopted for the partition numbers within the execution node.

3. The method according to claim 2, wherein in a case where the subtask is a join query task, there are a plurality of target partitioned tables, and the mapping the identifiers of the target partitions to partition numbers within the execution node comprises:

mapping same identifiers of partitions allocated to the execution node from the plurality of target partitioned tables to a same partition number within the execution node.

4. The method according to claim 2, wherein the establishing a scheduling relationship between the partition numbers within the execution node and worker processes within the execution node comprises:

determining a total number of partitions within the execution node based on the partition numbers to which the identifiers of the target partitions are mapped within the execution node; and

respectively determining, for the worker processes within the execution node, partition numbers for which the worker processes are responsible, according to the total number of partitions and a degree of parallelism of the worker processes, to generate the scheduling relationship

between the partition numbers within the execution node and the worker processes within the execution node.

5. The method according to any one of claims 1 to 4, wherein if the subtask is a hash join task and a target process is independently responsible for a plurality of target partitions, the method further comprises:

if a project operator and/or a filter operator is present before a hash join operator in an execution plan of the target process, retaining partition numbers of remaining data after data scheduled to the target process passes through the project operator or the filter operator; and

performing, in the hash join operator, hash join operations respectively on different target partitions for which the hash join operator is independently responsible;

wherein the target process is any worker process within the execution node.

6. The method according to claim 5, further comprising:

if a special operator capable of disrupting partitions is present before the hash join operator in the execution plan of the target process, discarding related partition numbers after data passes through the special operator, to cause the hash join operator to perform the hash join operations according to a shuffle mechanism.

7. The method according to claim 5, wherein if the target process and one or more other worker processes are jointly responsible for a shared partition, the method further comprises:

obtaining, by using the hash join operator in the target process, a total number of times of batch processing corresponding to the shared partition and a total number of worker processes responsible for the shared partition;

determining, by using the hash join operator, a data batch allocated to the target process, according to a preset allocation rule, the total number of times of batch processing, and the total number of worker processes; and

determining, by using the hash join operator, whether a current data batch is the data batch allocated to the target process, and if so, starting a build side of the hash join operator to build, for the current data batch, a hash table for supporting hash join calculation, to cause worker processes jointly responsible for the shared partition to cooperatively build a hash table corresponding to the shared partition.

8. The method according to claim 7, wherein the preset allocation rule comprises performing allocation according to a load balancing mechanism.

9. The method according to any one of claims 1 to 8, further comprising:

obtaining a partition pruning list constructed for the target partitioned table, wherein the partition pruning list comprises a list of partitions that do not affect a query result after being pruned;

if the execution node needs to exchange data with another node, performing, according to the partition pruning list, pruning on the target partitions allocated to the subtask from the target partitioned table, and then constructing exchange data for the another node based on remaining partitions; and

using the exchange data constructed based on the remaining partitions as response data when a data exchange request is received.

10. The method according to claim 9, wherein the performing, according to the partition pruning list, pruning on the target partitions allocated to the subtask from the target partitioned table comprises:

if there are a plurality of target partitioned tables and to-be-pruned partitions in the partition pruning list are present among the target partitions, pruning all to-be-pruned partitions allocated to the subtask from the plurality of target partitioned tables.

11. The method according to claim 9, wherein the partition pruning list is generated by a database scheduler distributing the subtask, and the database scheduler is configured to identify, according to a parsing result corresponding to a query statement of the query task, partitions in the target partitioned table that are designated as needing to be pruned, and identify partitions with null data in the target partitioned table by using attribute description information corresponding to the target partitioned table, to construct the partition pruning list.

12. An execution node, comprising a memory, a processor, and a communication component, wherein

the memory is configured to store one or more computer instructions; and

the processor is coupled to the memory and the communication component, and configured to execute the one or more computer instructions, to perform the data query method according to any one of claims 1 to 11.

13. A database system, comprising a plurality of execution nodes that process a query task in parallel, wherein the plurality of execution nodes respectively receive subtasks split from the query task, a single execution node starts a plurality of parallel worker processes for a subtask received by the single execution node, and the single execution node is configured to perform the data query method according to any one of claims 1 to 11.

14. A computer-readable storage medium storing computer instructions, wherein the computer instructions, when executed by one or more processors, cause the one or more processors to perform the data query method according to any one of claims 1 to 11.

15. A computer program, wherein the computer program, when executed on a computer, causes the computer to perform the data query method according to any one of claims 1 to 11.