US 20160246842A1

(19) United States

(12) Patent Application Publication

LI et al.

(10) Pub. No.: US 2016/0246842 A1

(43) Pub. Date: Aug. 25, 2016

(54) QUERY OPTIMIZATION ADAPTIVE TO SYSTEM MEMORY LOAD FOR PARALLEL DATABASE SYSTEMS

(71) Applicant: **Futurewei Technologies, Inc.**, Plano, TX (US)

(72) Inventors: **Huaizhi LI**, Belmont, CA (US); **Guogen ZHANG**, San Jose, CA (US)

(21) Appl. No.: **14/631,074**

(22) Filed: **Feb. 25, 2015**

Publication Classification

(51) **Int. Cl.**

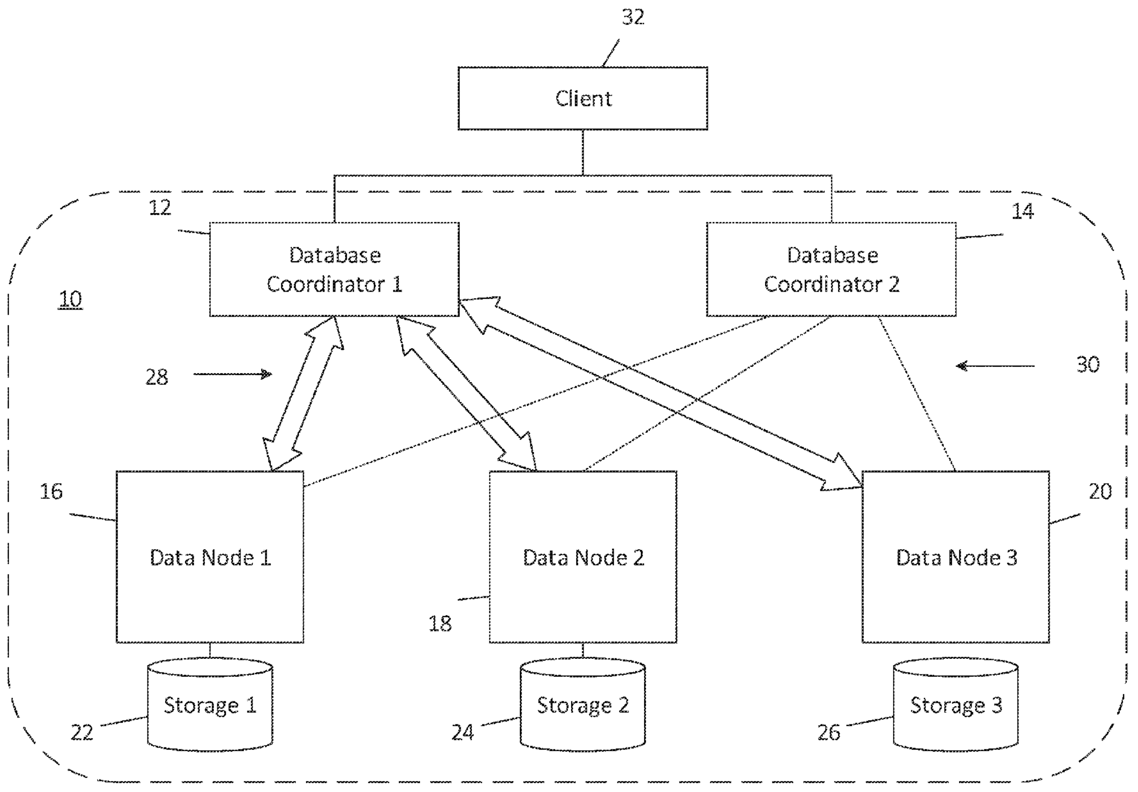
G06F 17/30	(2006.01)
H04L 29/08	(2006.01)

(52) **U.S. Cl.**

CPC ***G06F 17/30463*** (2013.01); ***G06F 17/30598*** (2013.01); ***G06F 17/30864*** (2013.01); ***H04L 67/10*** (2013.01)

(57) **ABSTRACT**

A method for adaptively generating a query execution plan for a parallel database distributed among a cluster of data nodes includes receiving memory usage data from a multiple data nodes including network devices, calculating a representative memory load corresponding to the data nodes based on the memory usage data, categorizing a memory mode corresponding to the data nodes based on the calculated representative memory load, calculating an available work memory corresponding to the data nodes based on the memory mode, and generating the query execution plan for the data nodes based on the available work memory, wherein the memory usage data is based on monitored individual memory loads associated with the data nodes and the query execution plan corresponds to the currently available work memory.



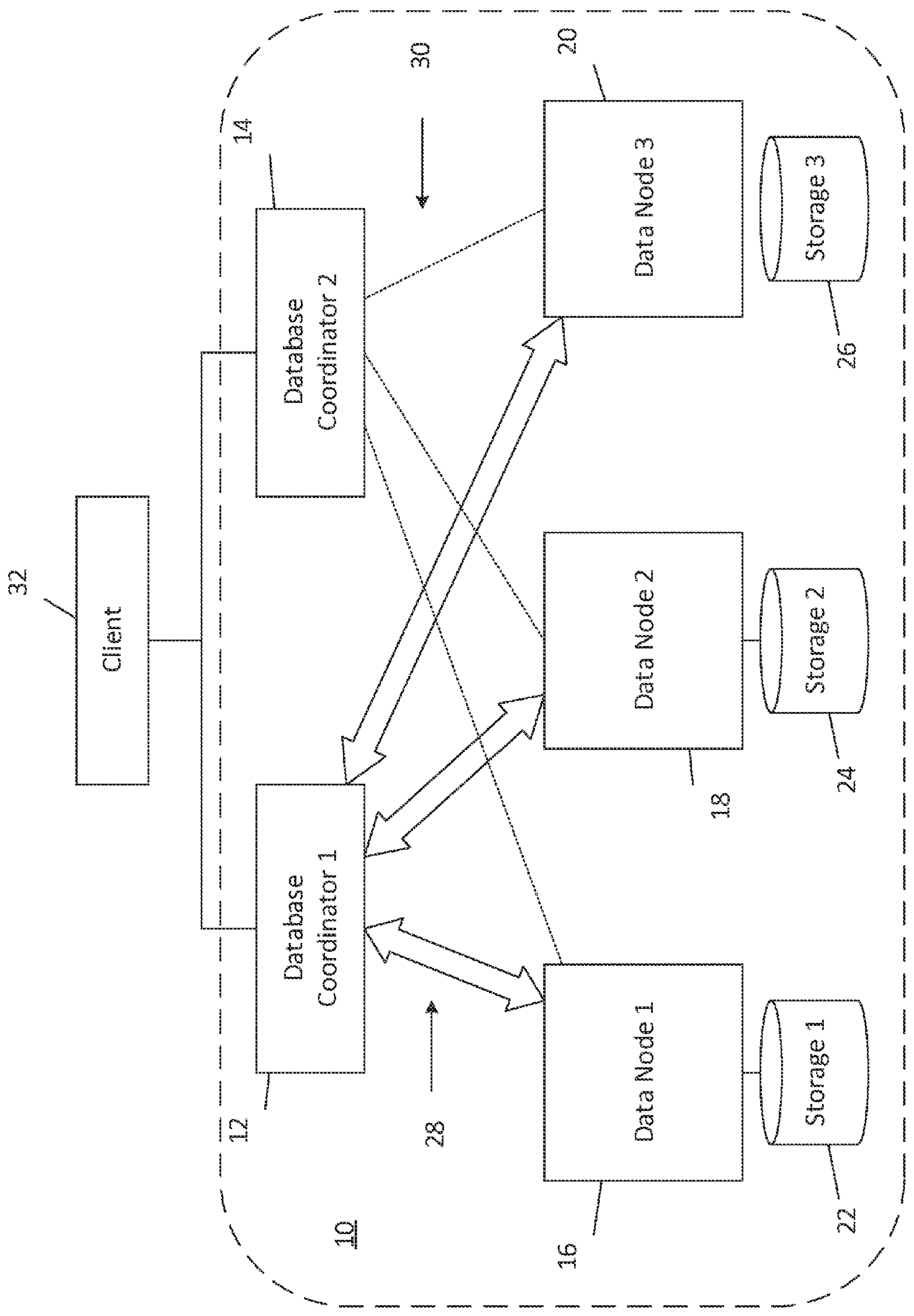

32
Client
14
Database Coordinator 2
30
20
Data Node 3
Storage 3
26
Data Node 2
Storage 2
18
24
Database Coordinator 1
12
28
Data Node 1
Storage 1
16
22
10

FIG. 1

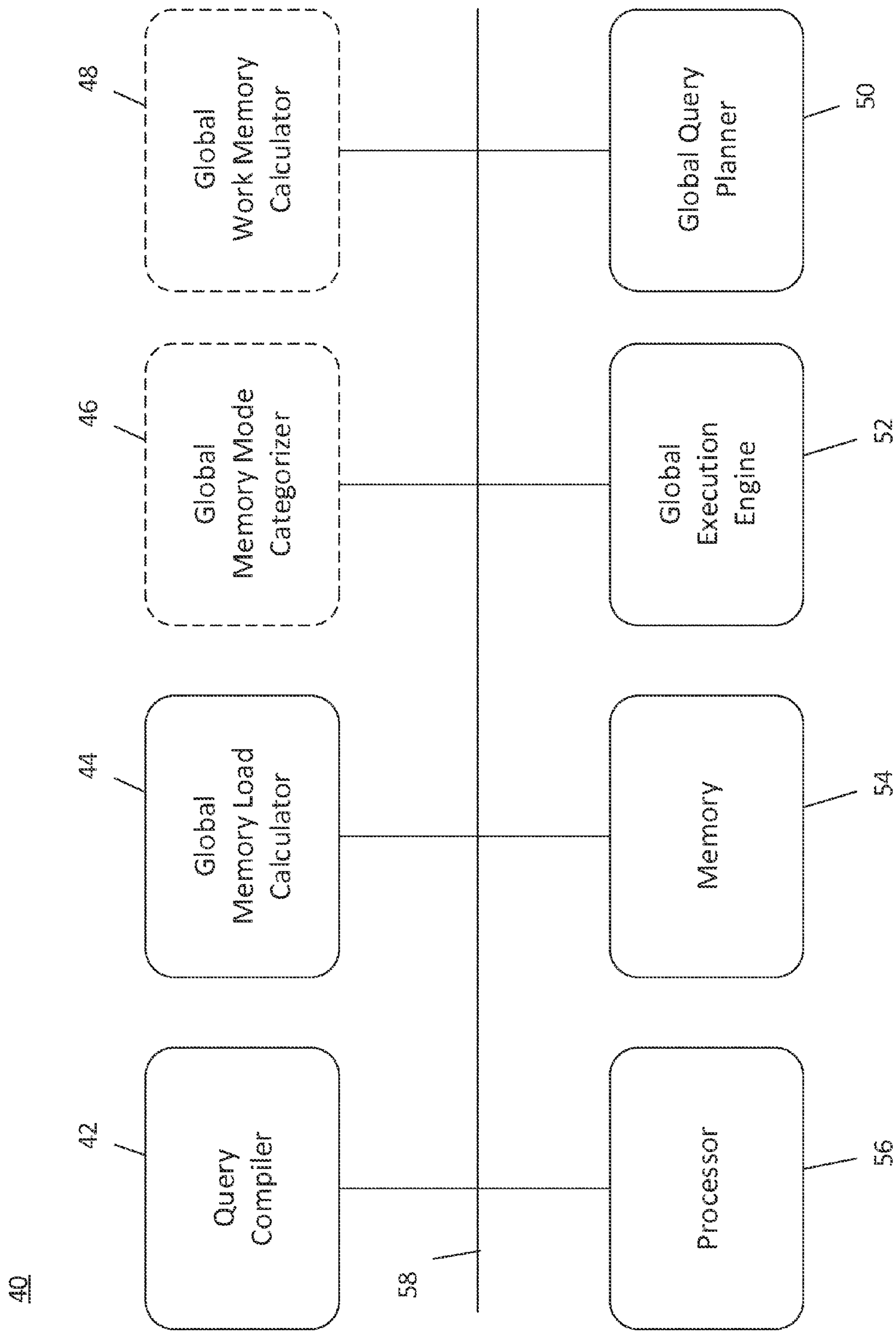

40
Query Compiler
42
Global Memory Load Calculator
44
Global Memory Mode Categorizer
46
Global Work Memory Calculator
48
Global Query Planner
50
Global Execution Engine
52
Memory
54
Processor
56
58

FIG. 2

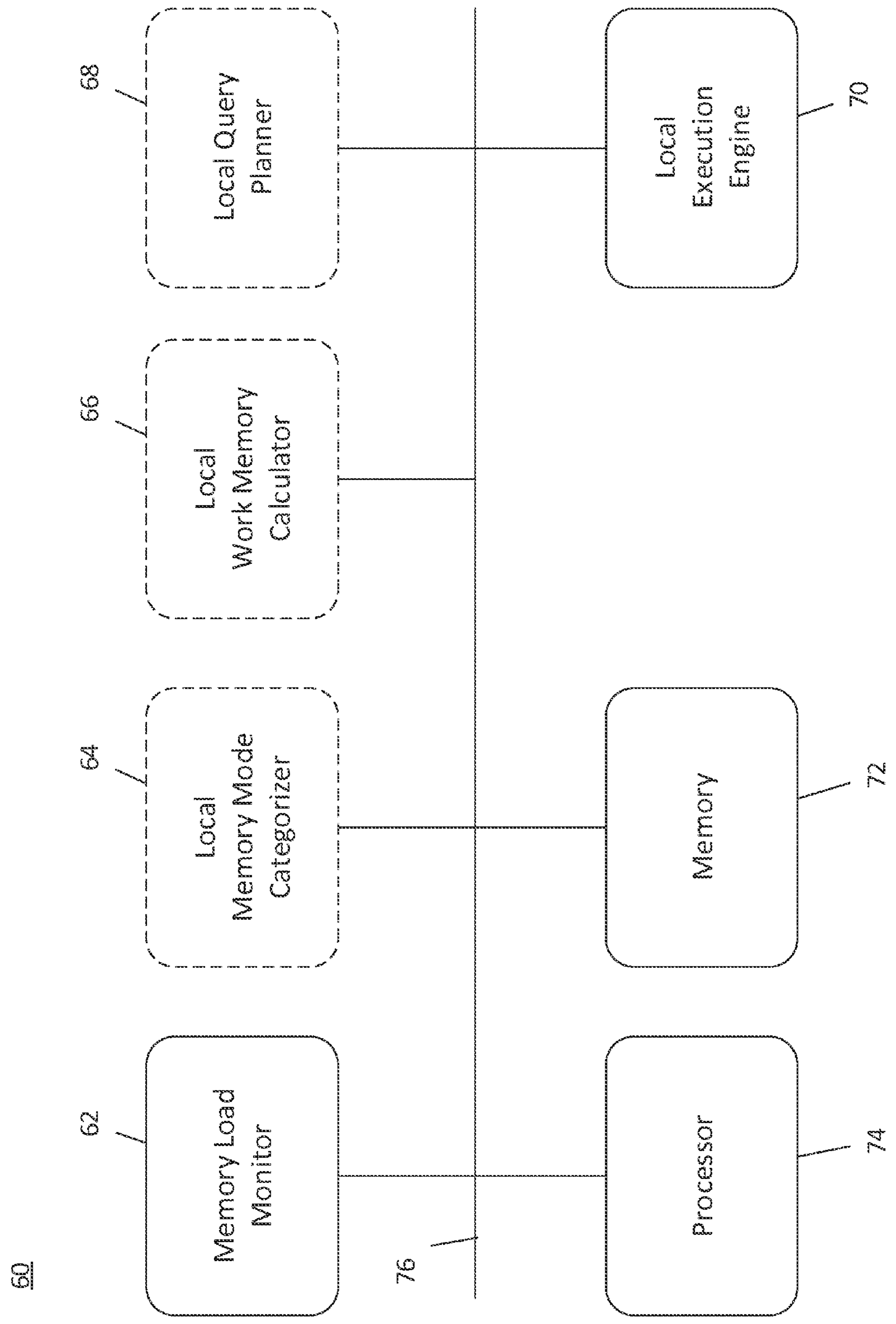

60
62
Memory Load Monitor
64
Local Memory Mode Categorizer
66
Local Work Memory Calculator
68
Local Query Planner
76
Processor
74
Memory
72
Local Execution Engine
70

FIG. 3

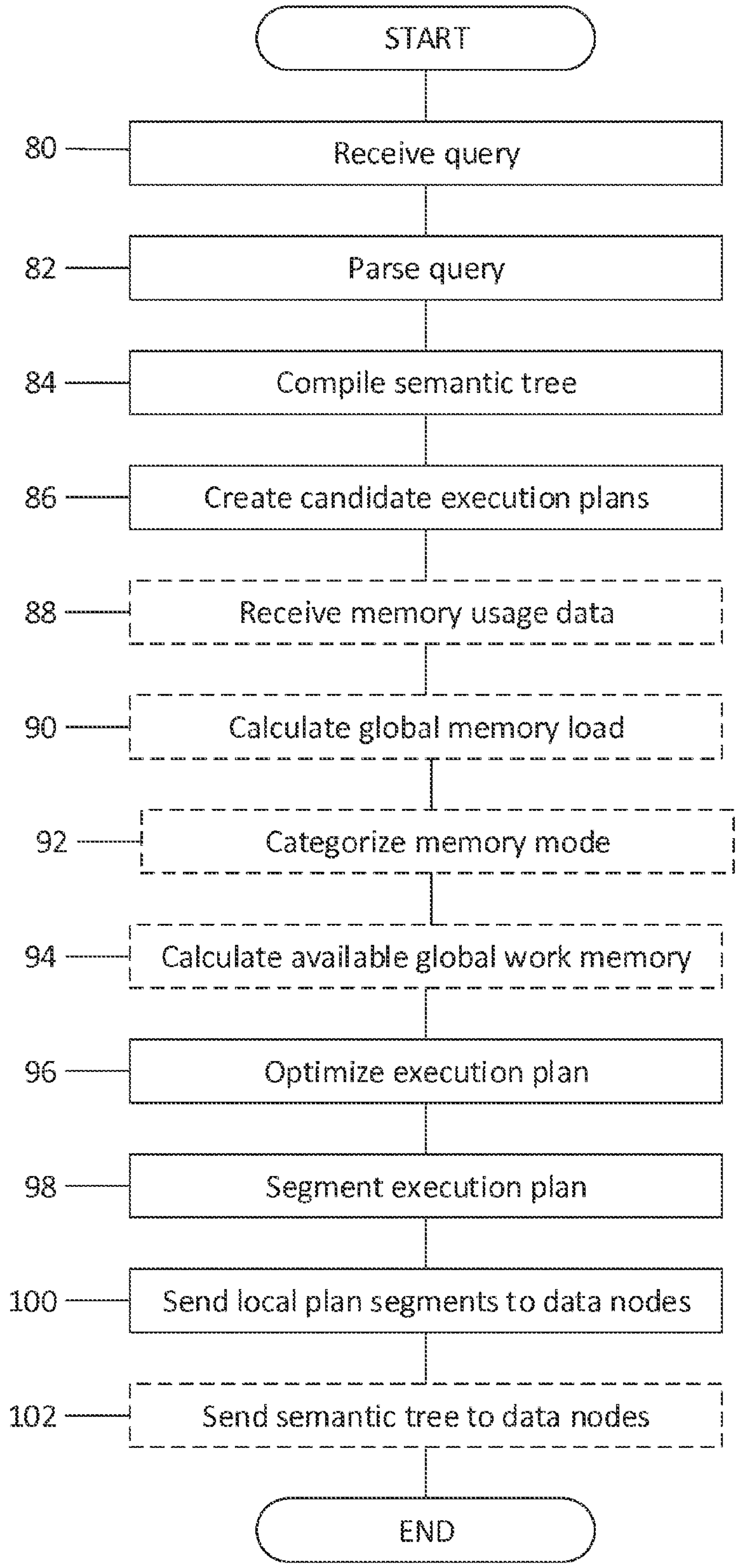

START
80
Receive query
82
Parse query
84
Compile semantic tree
86
Create candidate execution plans
88
Receive memory usage data
90
Calculate global memory load
92
Categorize memory mode
94
Calculate available global work memory
96
Optimize execution plan
98
Segment execution plan
100
Send local plan segments to data nodes
102
Send semantic tree to data nodes
END

FIG. 4

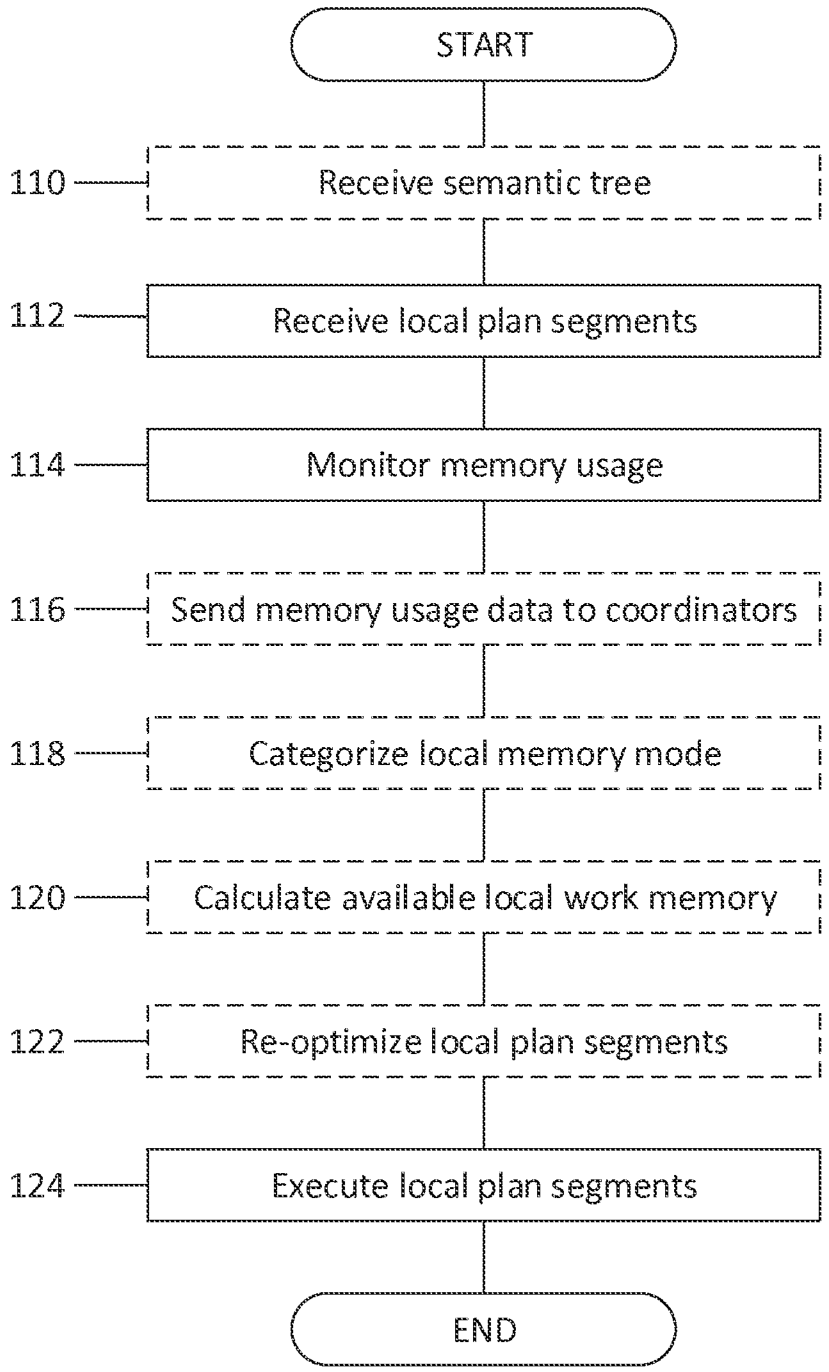

START
110
Receive semantic tree
112
Receive local plan segments
114
Monitor memory usage
116
Send memory usage data to coordinators
118
Categorize local memory mode
120
Calculate available local work memory
122
Re-optimize local plan segments
124
Execute local plan segments
END

FIG. 5

QUERY OPTIMIZATION ADAPTIVE TO SYSTEM MEMORY LOAD FOR PARALLEL DATABASE SYSTEMS

TECHNICAL FIELD

[0001] This description relates generally to databases, and more particularly to adaptively optimizing parallel database query execution plans based on system memory load.

BACKGROUND

[0002] Database systems are used to store information and relationship data that can be queried to find individual pieces of information, related pieces of information or relations between pieces of information. A typical parallel database system includes a coordinator node, or multiple coordinator nodes, along with multiple data processing nodes interconnected by a network.

[0003] In general, the coordinator nodes form the front end of the system that interfaces with client systems by way of the same or another network, and coordinates with the data processing nodes. Typically, parallel database clients submit queries to the coordination nodes, or coordinators, which in turn dispatch the queries to the data nodes for execution.

[0004] In some existing distributed parallel database systems, for example, massively parallel processing (MPP) database systems, multiple coordinator nodes and multiple data nodes together form a cluster of computing systems. In distributed database systems the tables of a database typically are divided into multiple sections, or partitioned, and the resulting partitions reside on multiple data nodes in the cluster.

[0005] In general, in both traditional, single-node, non-distributed relational database management systems and distributed relational database management systems, when a database receives a query, such as a structured query language (SQL) query, from a client the database system compiles the query, creates and optimizes a query execution plan, and executes the query execution plan. The database system then generates query results and sends the results back to the client.

[0006] In typical parallel database systems, the query plan compilation and optimization is carried out by the coordinator node, and the query is executed in parallel on all the nodes. Upon receiving a query, a coordinator invokes a query compiler to create a semantic tree based on the query. The query is parsed using aggregated statistics in the global catalog as if the database were running on single computer. The coordinator then invokes a query planner that processes the semantic tree, creates and compares all possible query execution plans, and outputs an optimal query execution plan.

[0007] The query plan typically is subdivided into segments and parallelized for the number of distributed data nodes or data partitions in system. Some query segments are executed on the coordinator nodes, and other query segments are executed on the data nodes. Thus, the coordinator sends the latter query plan segments to the various data nodes in the cluster for execution. Typically, the coordinator node passes the same query plan segment, or segments, to each of the individual data nodes, all of which execute the same query execution plan segment, or segments, against the various stored data partitions.

[0008] With regard to any particular query, the query planner considers multiple candidate query execution plans, any one of which the parallel database system is capable of processing and generating the results. For example, a typical query execution plan consists of database operators such as join, sort and aggregation operators. As an example, with regard to the join operator there are different join algorithms, including hash join, nested loop join and sort-merge join.

[0009] Since each operator has differing efficiencies, even though all of the candidate plans are able to determine the appropriate final query output, the cost of executing each of the plans varies substantially. The query planner takes into consideration system resources, such as memory and table partitions statistics, when optimizing the algorithms for database operators. The optimizer function of the query planner on the coordinator node determines the optimal plan, for example, making a choice between an external merge sort operation and a quick sort operation, or deciding between a hash join operation and a nested loop join operation.

[0010] In some existing solutions, the concept of work memory, the amount of system memory area or space currently available for use regarding the query, drives the determination of the optimal execution plan. In general, existing solutions apply the concept of a fixed work memory to optimize query plans, without taking into consideration the discrepancies between loading of different data nodes over time. As a result, all of the data nodes typically execute the same plan segment, which is not always the optimal plan with respect to each of the data nodes.

[0011] Thus, due to factors such as non-uniform distribution of database table partitions across the various data nodes and the dynamic change of memory availability on different data nodes over time, the fixed work memory configuration sometimes results in a non-optimal query plan being selected for the data nodes. For example, given a system with substantial available memory, if the predetermined work memory is too small the query planner selects an external sort for a sorting operation, even though a quick sort operation under the circumstances could be more efficient.

[0012] Such optimization errors can result in general database performance degradation. As a result, some existing query optimization methodologies can have drawbacks when used in distributed parallel database systems, since database query performance is of relatively high importance.

SUMMARY

[0013] According to one general aspect, a method for adaptively generating a query execution plan for a parallel database distributed among a cluster of data nodes includes receiving memory usage data from multiple data nodes including network devices, calculating a representative memory load corresponding to the data nodes based on the memory usage data, categorizing a memory mode corresponding to the data nodes based on the calculated representative memory load, calculating an available work memory corresponding to the data nodes based on the memory mode, and generating the query execution plan for the data nodes based on the available work memory. The memory usage data is based on monitored individual memory loads associated with the data nodes and the query execution plan is adapted to the currently available work memory.

[0014] According to another general aspect, a device for adaptively generating a query execution plan for a parallel database distributed among a cluster of data nodes includes an individual data node that includes an individual network device associated with the cluster configured to store at least a portion of data corresponding to the database and to receive a query execution plan segment, a memory load monitor associated with the individual data node and configured to monitor a memory load associated with the individual data node, and a local execution engine configured to execute the query execution plan segment.

[0015] The details of one or more embodiments of the invention are set forth in the accompanying drawings and the description below. Other features, objects, and advantages of the invention will be apparent from the description and drawings, and from the claims.

DESCRIPTION OF THE DRAWINGS

[0016] FIG. **1** is a schematic drawing depicting a system for adaptively generating a query execution plan for a parallel database distributed among a cluster of data nodes.

[0017] FIG. **2** is a block diagram of an exemplary coordinator device implemented in a system for adaptively generating a query execution plan for a parallel database distributed among a cluster of data nodes.

[0018] FIG. **3** is a block diagram of an exemplary data node implemented in a system for adaptively generating a query execution plan for a parallel database distributed among a cluster of data nodes.

[0019] FIG. **4** is a flowchart representing a method of adaptively generating a query execution plan for a parallel database distributed among a cluster of data nodes.

[0020] FIG. **5** is a flowchart representing another method of adaptively generating a query execution plan for a parallel database distributed among a cluster of data nodes.

DETAILED DESCRIPTION

[0021] This disclosure describes a query plan optimization strategy for use in distributed relational database management systems in which query execution plans are adaptively determined based on current system memory availability. Instead of assuming a fixed work memory configuration, as in existing prior art technologies, the methods and devices described in this disclosure monitor the system load and memory availability on the distributed data processing nodes associated with the database cluster on a current and ongoing basis.

[0022] In an embodiment, a coordinator node determines the global work memory configuration using memory usage data received from memory load monitors on each of the data nodes and generates a query plan that is optimized for the current aggregate work memory available on the data nodes. In an alternative embodiment, each data node determines the local work memory configuration depending on the current memory usage and availability monitored at that node and modifies or re-optimizes the query plan for the current local work memory available on the data node. In the former embodiment the query plan is tailored to the cluster of data nodes, and in the latter embodiment the query plan is tailored for each of the individual data nodes.

[0023] As illustrated in FIG. **1**, a system **10** for adaptively generating a query execution plan for a parallel database distributed among a cluster of data nodes includes a pair of database coordinator nodes, or coordinators, **12**, **14** and three data processing nodes, or data nodes, **16**, **18**, **20** having three storage devices **22**, **24**, **26**, respectively. In various embodiments, the storage devices **22**, **24**, **26** is either integrated into or peripherally connected to the data nodes **16**, **18**, **20**. The coordinator nodes **12**, **14** are interconnected with each of the data nodes by data links **28**, **30**, including, for example, a communications network.

[0024] The storage devices **22**, **24**, **26** at the data nodes **16**, **18**, **20** each have stored a partition, or multiple partitions, of a distributed database table. Together, the storage devices **22**, **24**, **26** contain the information data for the complete database table.

[0025] In operation, the coordinator nodes receive query requests from a client node, or client, **32**. As an example, referring still to FIG. **1**, the coordinator node **12** receives a query request from the client **32**. In response, the coordinator **12** compiles the query and create a query plan. The coordinator **12** further subdivides the query plan into segments and send the query plan segments to each of the data nodes **16**, **18**, **20** by way of the data links **28**, such as a network, for local execution on each of the data nodes **16**, **18**, **20**.

[0026] As a result of the working environment, including factors such as data skew or input/output (I/O), the memory usage and availability at the various data nodes **16**, **18**, **20** sometimes is uneven. In an embodiment, each of the data nodes **16**, **18**, **20** monitors the memory usage at the individual data node **16**, **18**, **20** and sends memory usage data to each of the coordinators **12**, **14**. The coordinators **12**, **14** use the memory usage data from all of the data nodes **16**, **18**, **20** to determine an aggregate work memory that represents the average amount of memory currently available on each of the data nodes **16**, **18**, **20** to be dedicated to locally executing the query plan on the data nodes **16**, **18**, **20**. The coordinators **12**, **14** optimize the query plans, or the query plan segments, for globally optimal execution performance on all the data nodes **16**, **18**, **20**, and send the same query plan, or query plan segments, to all of the data nodes **16**, **18**, **20**.

[0027] Similarly, in an alternative embodiment each of the data nodes **16**, **18**, **20** monitors the memory usage at the individual data node **16**, **18**, **20**. However, each of the individual data nodes **16**, **18**, **20** determines a local work memory that indicates the amount of memory currently available on the individual data node **16**, **18**, **20** to be dedicated to locally executing the query plan on the data node **16**, **18**, **20**. Each of the individual data nodes **16**, **18**, **20** further performs localized query planning to adapt query plan segments received from one of the coordinators **12**, **14** for optimal execution performance on the individual data node **16**, **18**, **20**.

[0028] These implementations provide advantages with respect to existing solutions, which typically do not take actual system load or memory usage and availability variation among the data nodes into consideration, but rather presume a fixed work memory. By determining a more accurate work memory instead of a predetermined value, the implementations described in this disclosure can generate a more efficient query plan dynamically tailored to the actual working environment of the data nodes, and thus improve the overall performance of the distributed parallel database system.

[0029] Referring to FIG. **2**, a coordinator node, or coordinator, **40** implemented in the system **10** of FIG. **1** includes a query compiler **42**, an optional global memory load calculator **44**, an optional global memory mode categorizer **46**, an optional global work memory calculator **48**, a global query planner **50**, a global execution engine **52**, a memory **54** and a processor **56**, all of which are interconnected by a data link **58**. The coordinator **40** is configured to receive a query request, such as a structured query language (SQL) query request, from a client. Components shown with dashed lines in FIG. **2** are optional items that are not included in all implementations.

[0030] The query compiler **42** is configured to parse the received query request and create a semantic tree that corresponds to the query request. The global memory load calculator **44** optionally calculates a global memory load that represents, for example, the average current memory load on the data nodes that form the cluster using memory usage data received from all the data nodes.

[0031] The global memory mode categorizer **46** optionally assigns a global category, or mode, that indicates the approximate level of current memory usage or availability among the data nodes that form the cluster. The global memory mode categorizer **46** in some implementations maps the current average memory load among the data nodes to one of three categories, for example, LIGHT, NORMAL and HEAVY, according to how heavy the current global memory load is throughout the system.

[0032] For example, the global memory mode categorizer **46** assigns the LIGHT mode when average memory usage among all the data nodes is below thirty percent (30%) of the total system memory capacity, assign the NORMAL mode when average memory usage among all the data nodes is from thirty percent (30%) to seventy percent (70%) of the total system memory capacity, and assign the HEAVY mode when average memory usage among all the data nodes is above seventy percent (70%) of the total system memory capacity.

[0033] Based on the currently assigned memory mode, the global work memory calculator **48** optionally calculates the current global work memory for use in optimizing the query plan. The current global work memory corresponds to the average memory space available on each of the data nodes that form the cluster. For example, the global work memory calculator **48** in some implementations uses a memory load factor corresponding to the current memory mode, or category, to compute the available global work memory according to the following formula:

$$\text{work\_memory} = \text{system\_memory\_for\_query} \times \text{memory\_load\_factor}$$

where

$$\text{system\_memory\_for\_query} = \frac{\text{system\_memory} - \text{memory\_for\_bufferpool} - \text{other\_memory\_overhead}}{\text{connection\_number}}$$

using the following definition for the memory load factor:

```
if memory__load == HEAVY
        memory__load__factor = 0.3;
if memory__load == LIGHT
        memory__load__factor = 0.9;
if memory__load == NORMAL
{
    if query is JOIN
        memory__load__factor = 0.6;
    else
        memory__load__factor = 0.5;
}
```

in addition to the following definitions:

[0034] system_memory_for_query is the amount of memory available for query operations for each connection;

[0035] system_memory is the total amount of memory on an individual data node;

[0036] memory_for_bufferpool is the amount of memory currently used for bufferpool;

[0037] other_memory_overhead is the amount of memory currently used for log file caching, thread creation, and so on; and

[0038] connection_number is the recent average number of connections to the database.

[0039] As a result, when the memory mode is LIGHT, query plans are generated based on a larger work memory suitable for doing relatively memory-intensive operations like building a hash table or a sort operation. This can be desirable, because even though query execution plans computed with larger work memory will likely to consume more memory resources, the query plans generally will execute with a faster response. Conversely, when the memory mode is HEAVY, query plans are computed based on a smaller work memory.

[0040] On the other hand, when the memory mode is NORMAL, queries are differentiated based on one or more features of the query. That is, a query with a higher probability of being relatively memory-intensive will be assigned a larger size work memory for query planning, and a query with a lower chance of being relatively memory-intensive will be assigned a smaller size work memory for query planning. Accordingly, the optimizer adaptively plans queries based on the current memory load situation, achieving dynamic memory utilization and better execution performance.

[0041] The global query planner **50** creates multiple alternative candidate query plans and determine the optimal plan using the calculated global work memory. The selected query plan generally results in improved query execution performance with respect to fixed work memory solutions, because the calculated global work memory more accurately reflects the system resources currently available on the distributed data nodes.

[0042] The global query planner **50** further divides the query plan into multiple segments to be forwarded to the data nodes, and then send one or more of the optimized query plan segments to each of the data nodes to be locally executed on the data nodes. The global execution engine executes portions of the query plan segments on the coordinator node **40**.

[0043] Referring to FIG. **3**, a data processing node, or data node, **60** implemented in the system **10** of FIG. **1** includes a memory load monitor **62**, an optional local memory mode categorizer **64**, an optional local work memory calculator **66**, an optional local query planner **68**, a local execution engine **70**, a memory **72** and a processor **76**, all of which are interconnected by a data link **76**. The data node **60** is configured to receive a query execution plan segment, or segments, from one of the coordinator nodes. Components shown with dashed lines in FIG. **2** are optional items that are not included in all implementations.

[0044] The memory load monitor **62** monitors system memory usage and availability in the data node **60**. In an implementation, the data node **60** periodically sends memory usage and availability information to all the coordinator nodes. As described above with regard to FIG. **2**, the coordinators use the memory usage and availability data to compute the average memory load of all the data nodes in the database cluster and map the memory load to a memory mode. The coordinator further calculates the work memory, as described above, and generate a query plan for all the data nodes.

[0045] In an alternative implementation, referring again to FIG. **3**, the local memory mode categorizer **64** optionally assigns a local category, or mode, that indicates the approximate level of current memory usage or availability on the data node **60**. The local memory mode categorizer **64** in some implementations maps the current memory load of the data node **60** to one of three categories, for example, LIGHT, NORMAL and HEAVY, according to how heavy the current local memory load at the data node **60**.

[0046] For example, the local memory mode categorizer **64** assigns the LIGHT mode when memory usage on the data node **60** is below thirty percent (30%) of the data node **60** memory capacity, assign the NORMAL mode when memory usage on the data node **60** is from thirty percent (30%) to seventy percent (70%) of the data node **60** memory capacity, and assign the HEAVY mode when memory usage on the data node **60** is above seventy percent (70%) of the data node **60** memory capacity.

[0047] Based on the currently assigned local memory mode, the local work memory calculator **66** optionally calculates the current local work memory for use in adapting the plan segment to the current work environment at the data node **60**. The current local work memory corresponds to the memory space available on the data node **60**. For example, the local work memory calculator **66** in some implementations uses a memory load factor corresponding to the current memory mode, or category, to compute the available local work memory according to the following formula:

$$\text{work\_memory}=\text{system\_memory\_for\_query}\times\text{memory\_load\_factor}$$

where

$$\text{system\_memory\_for\_query}=\frac{\text{system\_memory}-\text{memory\_for\_bufferpool}-\text{other\_memory\_overhead}}{\text{connection\_number}}$$

using the following definition for the memory load factor:

```
if memory__load == HEAVY
        memory__load__factor = 0.3;
if memory__load == LIGHT
        memory__load__factor = 0.9;
if memory__load == NORMAL
{
    if query is JOIN
        memory__load__factor = 0.6;
    else
        memory__load__factor = 0.5;
}
```

in addition to the following definitions:

[0048] system_memory_for_query is the amount of memory available for query operations for each connection;

[0049] system_memory is the amount of memory on the data node **60**;

[0050] memory_for_bufferpool is the amount of memory currently used for bufferpool;

[0051] other_memory_overhead is the amount of memory currently used for log file caching, thread creation, and so on; and

[0052] connection_number is the recent average number of connections to the database.

[0053] The local query planner **68** modifies or re-optimizes the query execution plan segment, or segments, using the calculated local work memory in order to adapt the plan segment, or segments, to the current local work environment. The modified or re-optimized query plan segment generally results in improved query execution performance with respect to fixed work memory solutions, because the calculated local work memory more accurately reflects the system resources currently available on the data node **60**. In any embodiment, the local execution engine **70** executes the query execution plan segment, or segments, on the data node **60**.

[0054] With regard to FIGS. **1-3**, the coordinator nodes **12**, **14**, **40** and the data processing nodes **16**, **18**, **40**, **60** includes a general computing device, and the memory **54**, **42** and processor **56**, **54** is integral components of a general computing device, such as a personal computer (PC), a workstation, a server, a mainframe computer, or the like. Peripheral components coupled to the general computing device further includes programming code, such as source code, object code or executable code, stored on a computer-readable medium that can be loaded into the memory **54**, **52** and executed by the processor **56**, **54** in order to perform the functions of the system **10**.

[0055] Thus, in various embodiments, the functions of the system **10** is executed on any suitable processor, such as a server, a mainframe computer, a workstation, a PC, including, for example, a note pad or tablet, a PDA, a collection of networked servers or PCs, or the like. Additionally, as modified or improved versions of the system **10** are developed, for example, in order to revise or add a template or country-specific information, software associated with the processor is updated.

[0056] In various embodiments, the system **10** is coupled to a communication network, which can include any viable combination of devices and systems capable of linking computer-based systems, such as the Internet; an intranet or extranet; a local area network (LAN); a wide area network (WAN); a direct cable connection; a private network; a public network; an Ethernet-based system; a token ring; a value-added network; a telephony-based system, including, for example, T1 or E1 devices; an Asynchronous Transfer Mode (ATM) network; a wired system; a wireless system; an optical system; a combination of any number of distributed processing networks or systems or the like.

[0057] The system **10** is coupled to the communication network by way of the local data links **58**, **56**, which in various embodiments incorporates any combination of devices—as well as any associated software or firmware-configured to couple processor-based systems, such as modems, access points, network interface cards, serial buses, parallel buses, LAN or WAN interfaces, wireless or optical interfaces and the like, along with any associated transmission protocols, as desired or required by the design.

[0058] An embodiment of the present invention communicates information to the user and request user input, for example, by way of an interactive, menu-driven, visual display-based user interface, or graphical user interface (GUI). The user interface is executed, for example, on a personal computer (PC) or terminal with a mouse and keyboard, with which the user interactively inputs information using direct manipulation of the GUI. Direct manipulation can include the use of a pointing device, such as a mouse or a stylus, to select from a variety of windows, icons and selectable fields, including selectable menus, drop-down menus, tabs, buttons, bullets, checkboxes, text boxes, and the like. Nevertheless, various embodiments of the invention incorporates any number of additional functional user interface schemes in place of this interface scheme, with or without the use of a mouse or buttons or keys, including for example, a trackball, a touch screen or a voice-activated system.

[0059] In an exemplary implementation of the system **10** of FIG. **1**, the coordinator nodes **12**, **14** includes the query compiler **42**, the global memory load calculator **44**, the global memory mode categorizer **46**, the global work memory calculator **48**, the global query planner **50**, the global execution engine **52**, the memory **54** and the processor **56**, while the data processing nodes **16**, **18**, **20** includes the memory load monitor **62**, the local execution engine **70**, the memory **72** and the processor **74**. The data nodes **16**, **18**, **20** periodically send memory usage data monitored at the data nodes **16**, **18**, **20** to all the coordinator nodes **12**, **14**, and the coordinator nodes **12**, **14** calculates the average memory load and global work memory, and generate and optimize query execution plan segments to be sent to and carried out on each of the data nodes **16**, **18**, **20**.

[0060] As an example, memory load monitors associated with each of the data nodes **16**, **18**, **20** of FIG. **1** at a particular point in time determines that the data nodes **16**, **18**, **20** are currently operating at approximately ninety percent (90%), twenty-five percent (25%) and fifty percent (50%), respectively. The data nodes **16**, **18**, **20** subsequently passes this information on to both coordinator nodes **12**, **14**. Then, when one of the coordinator nodes **12**, **14**, say, for example, coordinator **12**, processes a query request that has been received at the coordinator **12**, the coordinator **12** computes the average memory load of the system as fifty-five percent (55%) and assign the current memory mode to the NORMAL category. The coordinator **12** further computes the available global work memory for the data nodes in accordance with the NORMAL memory mode and generate the same optimized plan segments for all the data nodes in light of the current work environment.

[0061] In an alternative implementation of the system **10** of FIG. **1**, the coordinator nodes **12**, **14** includes the query compiler **42**, the global query planner **50**, the global execution engine **52**, the memory **54** and the processor **56**, while the data processing nodes **16**, **18**, **20** includes the memory load monitor **62**, the local memory mode categorizer **64**, the local work memory calculator **66**, the local query planner **68**, the local execution engine **70**, the memory **72** and the processor **74**. The coordinator nodes **12**, **14** generates global query execution plan segments and send these to all of the data nodes **16**, **18**, **20**. The data nodes **16**, **18**, **20** monitor memory usage at the individual data nodes **16**, **18**, **20**, calculate the local work memory, and modify or optimize the query execution plan segments for execution on the individual data nodes **16**, **18**, **20**.

[0062] As an example, memory load monitors associated with each of the data nodes **16**, **18**, **20** of FIG. **1** at a particular point in time determines that the data nodes **16**, **18**, **20** are currently operating at approximately ninety percent (90%), twenty-five percent (25%) and fifty percent (50%), respectively. The data nodes **16**, **18**, **20** subsequently receive a query execution plan segment from one of the coordinator nodes **12**, **14**. The data node **16** assigns the current local memory mode to the HEAVY category, the data node **18** assigns the current local memory mode to the LIGHT category, and the data node **20** assigns the current local memory mode to the NORMAL category. Each of the data nodes **16**, **18**, **20** further computes the available local work memory in accordance with the HEAVY, LIGHT and NORMAL memory modes, respectively, and re-optimize the query plan segment in parallel for the each of the individual data nodes **16**, **18**, **20** in light of the current work environment at the corresponding individual data nodes **16**, **18**, **20**. As a result, the query plan segments executed at each of the data nodes **16**, **18**, **20** differs.

[0063] Referring now to FIG. **4**, a process flow is illustrated that is performed, for example, by the coordinator node **40** of FIG. **2** to implement the method described in this disclosure for adaptively generating a query execution plan for a parallel database distributed among a cluster of data nodes. Blocks shown with dashed lines in FIG. **4** are optional actions, or events, that are not performed in all implementations. The process begins at block **80**, where a query request, such as a structured query language (SQL) query is received, for example, from a client node.

[0064] In block **82**, the received query is parsed, and in block **84** a semantic tree corresponding to the query is compiled. Multiple candidate query execution plans are created, in block **86**, based on the semantic tree. Current memory usage or availability information regarding the individual data nodes are received in block **88**, and in block **90** the current global memory load is calculated as described above using the received memory usage or availability data. In block **92**, the memory mode is assigned to an appropriate category, as described above, corresponding to the current global memory load. The available global work memory is computed as described above, in block **94**, and used in block **96** to optimize the query execution plan selected from among the candidate plans, as described above.

[0065] In block **98**, the query execution plan is divided into multiple segments for distribution to the data nodes, and in block **100** the same query execution plan segment, or segments, is transmitted to all of the data nodes in the database cluster. Additionally, the compiled semantic tree is forwarded to the data nodes in block **102**.

[0066] Referring now to FIG. **5**, a process flow is illustrated that is performed, for example, by the data processing node **60** of FIG. **3** to implement the method described in this disclosure for adaptively generating a query execution plan for a parallel database distributed among a cluster of data nodes. Blocks shown with dashed lines in FIG. **5** are optional actions, or events, that is performed in all implementations. The process begins at block **110**, where a query execution plan segment, or segments, are received. In block **112**, a compiled semantic tree also is received.

[0067] In block **114**, the current memory usage or availability of an individual data node is monitored. Optionally, in block **116** memory usage or availability information periodically is sent, for example, to all coordinator nodes. In block **118**, the local memory mode is optionally assigned to a category, as described above, corresponding to the current memory usage or availability

[0068] The available local work memory is computed as described above, in block **120**, and used in block **122** to modify or re-optimize the query execution plan segment, or segments, as described above. In block **124**, the query plan segment, or segments, is executed on the data node.

[0069] In an exemplary implementation of the system **10** of FIG. **1**, the coordinator nodes **12**, **14** performs the actions or events described in blocks **80** through **102** of FIG. **4**, while the data nodes **16**, **18**, **20** performs the actions or events described in blocks **112**, **114**, **116**, and **124** of FIG. **5**. Thus, the same query execution plan segment, or segments, which is optimized according to the dynamically-determined global work memory configuration across all the data nodes, is sent to all of the data nodes in the cluster.

[0070] In an alternative implementation of the system **10** of FIG. **1**, the coordinator nodes **12**, **14** performs the actions or events described in blocks **80** through **86**, and blocks **96** through **100** of FIG. **4**, while the data nodes **16**, **18**, **20** performs the actions or events described in blocks **110** through **114**, and blocks **118** through **124** of FIG. **5**. Thus, each data node throughout the cluster individually re-optimizes the query execution plan segment, or segments, in parallel using the dynamically-determined local work memory configuration corresponding to each individual data node.

[0071] As an example, the following query request is received by one of the coordinator nodes **12**, **14** of FIG. **1**, say, for example, by the coordinator **12**:

[0072] select count(*) from lineitem,part where l_partkey=p_partkey group by l_partkey;

In response, the coordinator **12** generates the following query execution plan segment and send the segment to the three data nodes **16**, **18**, **20** of FIG. **1**:

QUERY PLAN

```
GroupAggregate
    -> GATHER
        Node/s: All datanodes
        -> GroupAggregate
            -> Join
                Condition: (lineitem.l_partkey = part.p_partkey)
```

[0073] The first three lines of the query plan segment are executed on the coordinator **12**, while the aggregation and join operations are executed on each of the data nodes **16**, **18**, **20** in accordance with the current local memory mode category assigned to each of the data nodes **16**, **18**, **20** in light of the current work environment at the corresponding individual data nodes **16**, **18**, **20**. Thus, for example, if the local memory mode of data node **16** currently is assigned to the HEAVY category, the data node **16** re-optimizes the query plan to carry out a sort-based aggregation operation and a nested loop join operation. At the same time, if the local memory modes of the data node **18** and the data node **20** currently are assigned to the LIGHT and NORMAL categories, respectively, the data nodes **18**, **20** each instead re-optimizes the query plan to carry out a hash aggregation operation and a hash join operation.

[0074] Use of the adaptive query planning methodology described in this disclosure, which implements a dynamically calculated work memory configuration reflecting the current system load, results in improved query execution efficiency or performance with respect to solutions using fixed work memory configuration. By using the more accurate work memory configuration, rather than a predetermined, or fixed, value, the adaptive query planner can generate a modified or optimized query plan tailored to the current work environment at the data nodes, resulting in improved performance of the distributed parallel database system, reduced query response time, improved memory resource utilization and reduced data spilling.

[0075] Aspects of this disclosure are described herein with reference to flowchart illustrations or block diagrams, in which each block or any combination of blocks can be implemented by computer program instructions. The instructions are provided to a processor of a general purpose computer, special purpose computer, or other programmable data processing apparatus to effectuate a machine or article of manufacture, and when executed by the processor the instructions create means for implementing the functions, acts or events specified in each block or combination of blocks in the diagrams.

[0076] In this regard, each block in the flowchart or block diagrams corresponds to a module, segment, or portion of code that including one or more executable instructions for implementing the specified logical function(s). It should also be noted that, in some alternative implementations, the functionality associated with any block can occur out of the order noted in the figures. For example, two blocks shown in succession can, in fact, be executed substantially concurrently, or blocks can sometimes be executed in reverse order.

[0077] A person of ordinary skill in the art will appreciate that aspects of this disclosure can be embodied as a device, system, method or computer program product. Accordingly, aspects of this disclosure, generally referred to herein as circuits, modules, components or systems, can be embodied in hardware, in software (including firmware, resident software, micro-code, etc.), or in any combination of software and hardware, including computer program products embodied in a computer-readable medium having computer-readable program code embodied thereon.

[0078] In this respect, any combination of one or more computer readable media can be utilized, including, but not limited to, an electronic, magnetic, optical, electromagnetic, infrared, or semiconductor system, apparatus, or device, or any suitable combination of these. More specific examples of computer readable storage media would include the following non-exhaustive list: a portable computer diskette, a hard disk, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM), a Flash memory, a portable compact disc read-only memory (CD-ROM), an optical storage device, network-attached storage (NAS), a storage area network (SAN), magnetic tape, or any suitable combination of these. In the context of this disclosure, a computer readable storage medium can include any tangible medium that is capable of containing or storing program instructions for use by or in connection with a data processing system, apparatus, or device.

[0079] Computer program code for carrying out operations regarding aspects of this disclosure can be written in any combination of one or more programming languages, including object oriented programming languages such as Java, Smalltalk, C++, or the like, as well as conventional procedural programming languages, such as the "C," FORTRAN, COBOL, Pascal, or the like. The program code can execute entirely on an individual personal computer, as a stand-alone software package, partly on a client computer and partly on a remote server computer, entirely on a remote server or computer, or on a cluster of distributed computer nodes. In general, a remote computer, server or cluster of distributed computer nodes can be connected to an individual (user) computer through any type of network, including a local area network (LAN), a wide area network (WAN), an Internet access point, or any combination of these.

[0080] It will be understood that various modifications can be made. For example, useful results still could be achieved if steps of the disclosed techniques were performed in a different order, and/or if components in the disclosed systems were combined in a different manner and/or replaced or supplemented by other components. Accordingly, other implementations are within the scope of the following claims.

What is claimed is:

1. A method for adaptively generating a query execution plan for a parallel database distributed among a cluster of data nodes, comprising:

receiving, with a processor, memory usage data from a plurality of data nodes comprising a plurality of network devices;

calculating a representative memory load corresponding to the data nodes based on the memory usage data;

categorizing a memory mode corresponding to the data nodes based on the calculated representative memory load;

calculating an available work memory corresponding to the data nodes based on the memory mode; and

generating the query execution plan for the data nodes based on the available work memory, wherein the memory usage data is determined from a plurality of monitored individual memory loads associated with the data nodes and the query execution plan corresponds to the currently available work memory.

2. The method of claim **1**, further comprising:

receiving first memory usage data from a first data node associated with the cluster; and

receiving second memory usage data from a second data node associated with the cluster.

3. The method of claim **1**, wherein the representative memory load is calculated as a statistical mean based on the memory usage data corresponding to all of the data nodes associated with the cluster.

4. The method of claim **1**, wherein the memory mode is categorized in a first category when the representative memory load is below a first predetermined percentage of a system capacity, in a second category when the representative memory load is from the first predetermined percentage to a second predetermined percentage of the system capacity, or in a third category when the representative memory load is above the second predetermined percentage of the system capacity, wherein the system capacity corresponds to an aggregate capacity of the data nodes.

5. The method of claim **4**, wherein the available work memory is calculated based on a multiple corresponding to the memory mode, the multiple selected from a first multiple greater than one-half corresponding to the first category, a second multiple between three-tenths and seven-tenths corresponding to the second category if the query execution plan includes a relatively memory-intensive operator, a third multiple between four-tenths and eight-tenths if the query execution plan does not include a relatively memory-intensive operator, and a fourth multiple between one-tenth and five-tenths corresponding to the third category, wherein the first multiple is greater than the second multiple, the second multiple is greater than the third multiple, and the third multiple is greater than the fourth multiple.

6. The method of claim **1**, wherein the available work memory is calculated based on a multiple corresponding to the memory mode.

7. The method of claim **6**, wherein the available work memory is calculated based on an aggregate system memory size, an aggregate buffer memory area size, an aggregate additional overhead memory area size, and an average number of client connections associated with the database.

8. The method of claim **1**, further comprising:

receiving a query request from a client device interconnected with the database by a network associated with the cluster, wherein generating the query execution plan further comprises:

compiling a semantic tree based on the query request;

creating a plurality of candidate query execution plans based on the semantic tree;

substantially optimizing the query execution plan based on at least one of the candidate query execution plans and the available work memory;

segmenting the query execution plan into a plurality of query execution plan segments; and

sending at least one of the query execution plan segments to each of the data nodes.

9. The method of claim **1**, further comprising executing a global portion of the query execution plan.

10. A method for adaptively generating a query execution plan for a parallel database distributed among a cluster of data nodes, comprising:

monitoring, with a processor, a memory load associated with a data node comprising a network device;

categorizing a memory mode corresponding to the data node based on the memory load;

calculating an available work memory corresponding to the data node based on the memory mode;

receiving a query execution plan segment; and

adapting the query execution plan segment for the data node based on the available work memory, wherein the data node is associated with the cluster and the query execution plan segment corresponds to the current available work memory.

11. The method of claim **10**, further comprising:

monitoring an additional memory load associated with an additional data node comprising an additional network device;

categorizing an additional memory mode corresponding to the additional data node based on the additional memory load;

calculating an additional available work memory corresponding to the additional data node based on the additional memory mode; and

adapting the query execution plan segment for the additional data node based on the additional available work memory, wherein the additional data node is associated with the cluster and the query execution plan segment is corresponds to the current additional available work memory.

12. The method of claim **10**, wherein the memory mode is categorized in a first category when the representative memory load is below a first predetermined percentage of a system capacity, in a second category when the representative memory load is from the first predetermined percentage to a second predetermined percentage of the system capacity, or in a third category when the representative memory load is above the second predetermined percentage of the system capacity, wherein the system capacity corresponds to an individual capacity of the data node.

13. The method of claim **12**, wherein the available work memory is calculated based on a multiple corresponding to the memory mode, the multiple selected from a first multiple greater than one-half corresponding to the first category, a second multiple between three-tenths and seven-tenths corresponding to the second category if the query execution plan includes a relatively memory-intensive operator, a third multiple between four-tenths and eight-tenths if the query execution plan does not include a relatively memory-intensive operator, and a fourth multiple between one-tenth and five-tenths corresponding to the third category, wherein the first multiple is greater than the second multiple, the second multiple is greater than the third multiple, and the third multiple is greater than the fourth multiple.

14. The method of claim **10**, wherein the available work memory is calculated based on a multiple corresponding to the memory mode.

15. The method of claim **14**, wherein the available work memory is calculated based on a system memory size, an overhead memory buffer memory area size, an additional area size, and an average number of client connections associated with the database.

16. The method of claim **10**, wherein adapting the query execution plan further comprises:

receiving a semantic tree based on a query request from a client device interconnected with the database by a network associated with the cluster; and

substantially optimizing the received query execution plan segment based on at least the available work memory.

17. The method of claim **10**, further comprising executing the query execution plan segment.

18. A device for adaptively generating a query execution plan for a parallel database distributed among a cluster of data nodes, comprising:

an individual network device associated with the cluster, the individual network device comprising:

a memory that stores data corresponding to the database;

a memory load monitor that monitors a memory load associated with the individual network device; and

a processor that receives a query execution plan segment, modifies the query execution plan segment to create a modified query execution plan segment corresponding to the memory load, and executes the modified query execution plan segment.

19. The device of claim **18**, further comprising:

a database coordinator configured to receive a query request from a client device coupled to the database coordinator by a network associated with the cluster, receive memory usage data, including the memory load associated with the individual network device, from a plurality of network devices including the individual network device, and send at least one of a plurality of query execution plan segments to each of the network devices; and

one or more circuits for executing:

a query compiler configured to compile a semantic tree based on the query request;

a global memory load calculator configured to calculate a representative memory load corresponding to the network devices based on the memory usage data;

a global memory mode categorizer configured to categorize a memory mode corresponding to the network devices based on the calculated representative memory load;

a global work memory calculator configured to calculate an available work memory corresponding to the network devices based on the memory mode;

a global query planner configured to create a plurality of candidate query execution plans based on the semantic tree, generate and substantially optimize the query execution plan for the network devices based on at least one of the candidate query execution plans and the available work memory, and segment the query execution plan into the query execution plan segments; and

a global execution engine configured to execute a global portion of the query execution plan, wherein the query execution plan is corresponds to the currently available work memory.

20. The device of claim **18**, further comprising one or more circuits for executing:

a local memory mode categorizer configured to categorize a memory mode corresponding to the individual network device based on the memory load;

a local work memory calculator configured to calculate an available work memory corresponding to the individual network device based on the memory mode; and

a local query planner configured to substantially optimize the received query execution plan segment for the individual network device based on at least the available work memory, wherein the individual network device is further configured to receive a semantic tree based on a query request from a client device interconnected with the database by a network associated with the cluster and the query execution plan segment is corresponds to the current available work memory.

* * * * *