Office de la Propriété Intellectuelle du Canada
Un organisme d'Industrie Canada

Canadian Intellectual Property Office
An agency of Industry Canada

CA 2939906 A1 2015/08/27

(21) 2 939 906

(12) DEMANDE DE BREVET CANADIEN
CANADIAN PATENT APPLICATION

(13) A1

(86) **Date de dépôt PCT/PCT Filing Date:** 2015/02/18
(87) **Date publication PCT/PCT Publication Date:** 2015/08/27
(85) **Entrée phase nationale/National Entry:** 2016/08/16
(86) **N° demande PCT/PCT Application No.:** US 2015/016418
(87) **N° publication PCT/PCT Publication No.:** 2015/126968
(30) **Priorités/Priorities:** 2014/02/19 (US61/941,986); 2014/10/20 (US14/518,873)

(51) **Cl.Int./Int.Cl.** ***G06F 17/00*** (2006.01), ***G06F 17/30*** (2006.01), ***G06F 9/46*** (2006.01)

(71) **Demandeur/Applicant:**
SNOWFLAKE COMPUTING INC., US

(72) **Inventeurs/Inventors:**
DAGEVILLE, BENOIT, US;
CRUANES, THIERRY, US;
ZUKOWSKI, MARCIN, US

(74) **Agent:** SMART & BIGGAR

(54) **Titre : SYSTEMES ET PROCEDES DE GESTION DE DONNEES**
(54) **Title: DATA MANAGEMENT SYSTEMS AND METHODS**

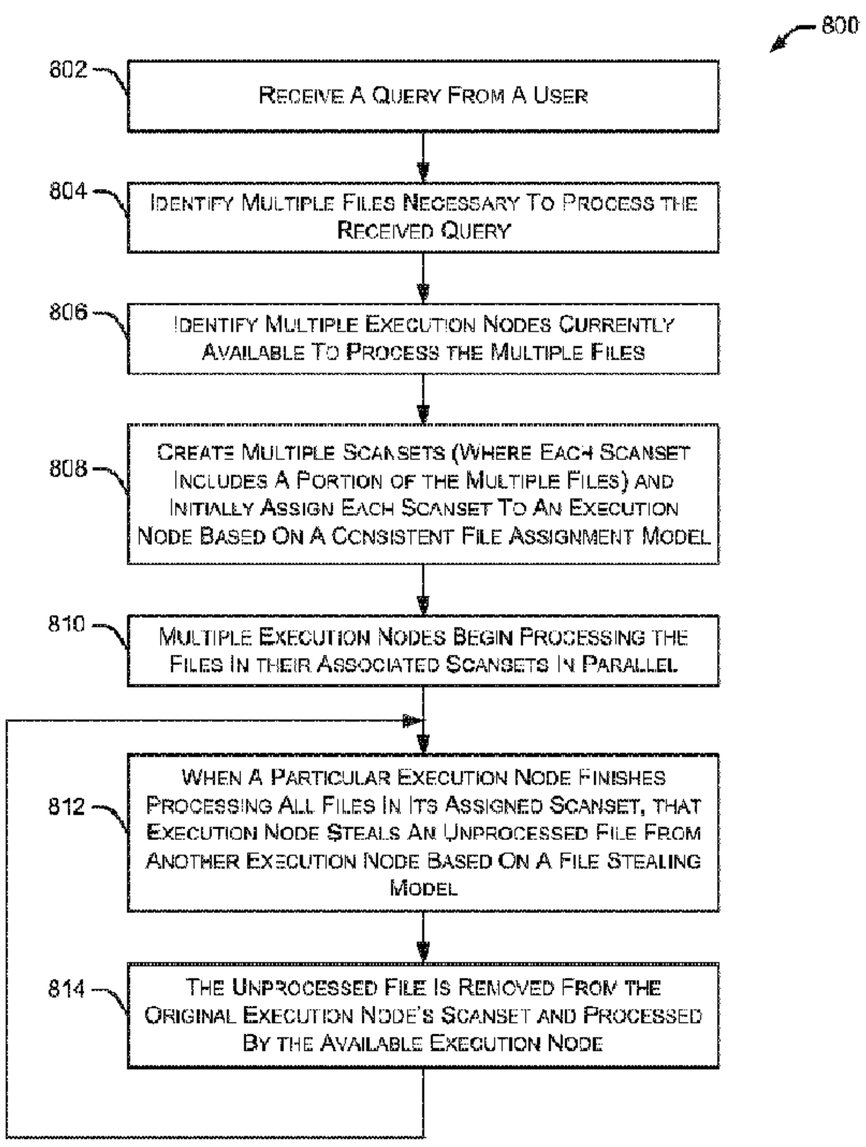


FIG. 8

(57) **Abrégé/Abstract:**
Example data management systems and methods are described. In one implementation, a method identifies multiple files to process based on a received query and identifies multiple execution nodes available to process the multiple files. The method initially creates multiple scansets, each including a portion of the multiple files, and assigns each scanset to one of the execution nodes based on a file assignment model. The multiple scansets are processed by the multiple execution nodes. If the method determines that a particular execution node has finished processing all files in its assigned scanset, an unprocessed file is reassigned from another execution node to the particular execution node.

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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
27 August 2015 (27.08.2015)

WIPO | PCT

(10) International Publication Number
WO 2015/126968 A3

(51) International Patent Classification:
G06F 15/16 (2006.01)

(21) International Application Number:
PCT/US2015/016418

(22) International Filing Date:
18 February 2015 (18.02.2015)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
61/941,986 19 February 2014 (19.02.2014) US
14/518,873 20 October 2014 (20.10.2014) US

(71) Applicant: **SNOWFLAKE COMPUTING INC.** [US/US]; 101 S Ellsworth Ave #350, San Mateo, CA 94401 (US).

(72) Inventors: **DAGEVILLE, Benoit**; 101 S Ellsworth Ave #350, San Mateo, CA 94401 (US). **CRUANES, Thierry**; 101 S Ellsworth Ave #350, San Mateo, CA 94401 (US). **ZUKOWSKI, Marcin**; 101 S Ellsworth Ave #350, San Mateo, CA 94401 (US).

(74) Agent: **STEVENS, David, R.**; Stevens Law Group, 1754 Technology Drive, Suite 226, San Jose, CA 95110 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

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

Published:

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

(88) Date of publication of the international search report:
15 October 2015

(54) Title: DATA MANAGEMENT SYSTEMS AND METHODS

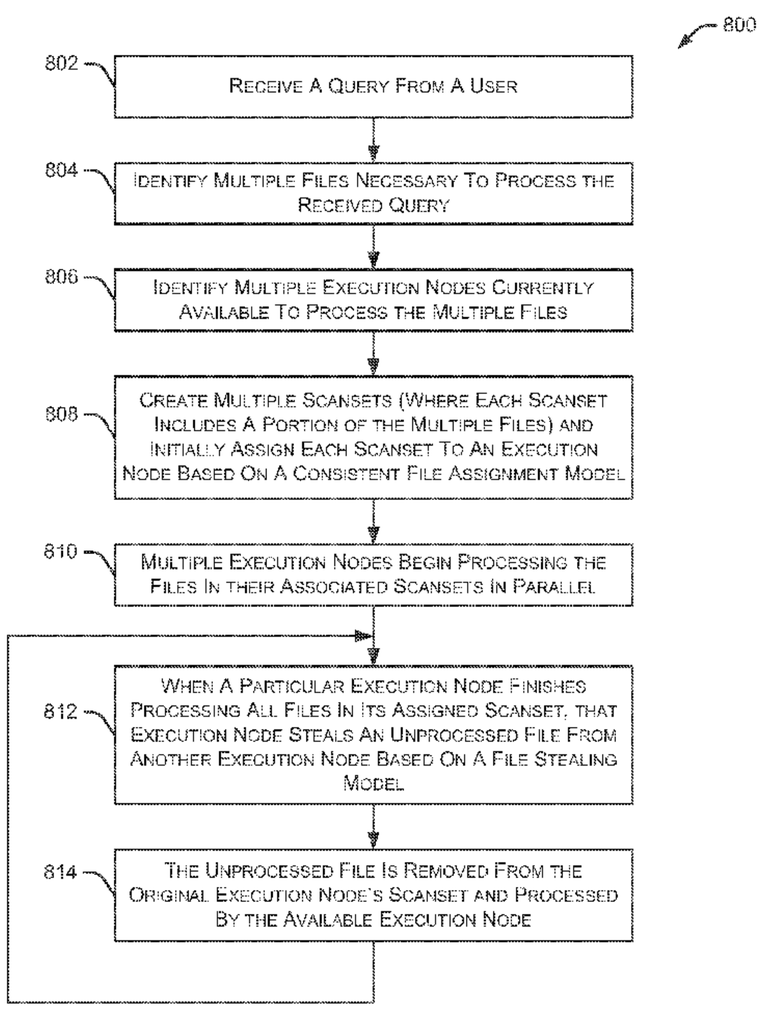


FIG. 8

(57) Abstract: Example data management systems and methods are described. In one implementation, a method identifies multiple files to process based on a received query and identifies multiple execution nodes available to process the multiple files. The method initially creates multiple scansets, each including a portion of the multiple files, and assigns each scanset to one of the execution nodes based on a file assignment model. The multiple scansets are processed by the multiple execution nodes. If the method determines that a particular execution node has finished processing all files in its assigned scanset, an unprocessed file is reassigned from another execution node to the particular execution node.

WO 2015/126968 A3

DATA MANAGEMENT SYSTEMS AND METHODS

CROSS REFERENCE TO RELATED APPLICATIONS

[0001] This application claims the benefit of United States Provisional Application Serial No. 61/941,986, entitled "Apparatus and method for enterprise data warehouse data processing on cloud infrastructure," filed February 19, 2014, the disclosure of which is incorporated herein by reference in its entirety.

TECHNICAL FIELD

[0002] The present disclosure relates to resource management systems and methods that manage the processing of data and other tasks.

BACKGROUND

[0003] Many existing data storage and retrieval systems are available today. For example, in a shared-disk system, all data is stored on a shared storage device that is accessible from all of the processing nodes in a data cluster. In this type of system, all data changes are written to the shared storage device to ensure that all processing nodes in the data cluster access a consistent version of the data. As the number of processing nodes increases in a shared-disk system, the shared storage device (and the communication links between the processing nodes and the shared storage device) becomes a bottleneck that slows data read and data write operations. This bottleneck is further aggravated with the addition of more processing nodes. Thus, existing shared-disk systems have limited scalability due to this bottleneck problem.

[0004] Another existing data storage and retrieval system is referred to as a "shared-nothing architecture." In this architecture, data is distributed across multiple processing nodes such that each node stores a subset of the data in the entire database. When a new processing node is added or removed, the shared-nothing architecture must rearrange data across the multiple processing nodes. This rearrangement of data can be time-consuming and disruptive to data read and write operations executed during the data rearrangement. And, the affinity of data to a particular node can create "hot spots" on the data cluster for popular data. Further, since each processing node performs also the storage function, this architecture requires at least one processing node to store data. Thus, the shared-nothing architecture fails to store data if all processing nodes are removed. Additionally, management of data in a shared-nothing architecture is complex due to the distribution of data across many different processing nodes.

[0005] The systems and methods described herein provide an improved approach to data storage and data retrieval that alleviates the above-identified limitations of existing systems.

BRIEF DESCRIPTION OF THE DRAWINGS

[0006] Non-limiting and non-exhaustive embodiments of the present disclosure are described with reference to the following figures, wherein like reference numerals refer to like parts throughout the various figures unless otherwise specified.

[0007] FIG. 1 is a block diagram depicting an example embodiment of the systems and methods described herein.

[0008] FIG. 2 is a block diagram depicting an embodiment of a resource manager.

[0009] FIG. 3 is a block diagram depicting an embodiment of an execution platform.

[0010] FIG. 4 is a block diagram depicting an example operating environment with multiple users accessing multiple databases through multiple virtual warehouses.

[0011] FIG. 5 is a block diagram depicting another example operating environment with multiple users accessing multiple databases through a load balancer and multiple virtual warehouses contained in a virtual warehouse group.

[0012] FIG. 6 is a block diagram depicting another example operating environment having multiple distributed virtual warehouses and virtual warehouse groups.

[0013] FIG. 7 is a flow diagram depicting an embodiment of a method for managing data storage and retrieval operations.

[0014] FIG. 8 is a flow diagram depicting an embodiment of a method for managing the processing of multiple files by multiple execution nodes.

[0015] FIG. 9 is a flow diagram depicting an embodiment of a method for managing the stealing of files from an execution node.

[0016] FIGs. 10A-10D depict example embodiments of assigning files to execution nodes using consistent hashing.

[0017] FIG. 11 is a block diagram depicting an example computing device.

DETAILED DESCRIPTION

[0018] The systems and methods described herein provide a new platform for storing and retrieving data without the problems faced by existing systems. For example, this new platform supports the addition of new nodes without the need for rearranging data files as required by the shared-nothing architecture. Additionally, nodes can be added to the platform without creating bottlenecks that are common in the shared-disk system. This new platform is always available

for data read and data write operations, even when some of the nodes are offline for maintenance or have suffered a failure. The described platform separates the data storage resources from the computing resources so that data can be stored without requiring the use of dedicated computing resources. This is an improvement over the shared-nothing architecture, which fails to store data if all computing resources are removed. Therefore, the new platform continues to store data even though the computing resources are no longer available or are performing other tasks.

[0019] In the following description, reference is made to the accompanying drawings that form a part thereof, and in which is shown by way of illustration specific exemplary embodiments in which the disclosure may be practiced. These embodiments are described in sufficient detail to enable those skilled in the art to practice the concepts disclosed herein, and it is to be understood that modifications to the various disclosed embodiments may be made, and other embodiments may be utilized, without departing from the scope of the present disclosure. The following detailed description is, therefore, not to be taken in a limiting sense.

[0020] Reference throughout this specification to "one embodiment," "an embodiment," "one example" or "an example" means that a particular feature, structure or characteristic described in connection with the embodiment or example is included in at least one embodiment of the present disclosure. Thus, appearances of the phrases "in one embodiment," "in an embodiment," "one example" or "an example" in various places throughout this specification are not necessarily all referring to the same embodiment or example. In addition, it should be appreciated that the figures provided herewith are for explanation purposes to persons ordinarily skilled in the art and that the drawings are not necessarily drawn to scale.

[0021] Embodiments in accordance with the present disclosure may be embodied as an apparatus, method or computer program product. Accordingly, the present disclosure may take

the form of an entirely hardware-comprised embodiment, an entirely software-comprised embodiment (including firmware, resident software, micro-code, etc.) or an embodiment combining software and hardware aspects that may all generally be referred to herein as a "circuit," "module" or "system." Furthermore, embodiments of the present disclosure may take the form of a computer program product embodied in any tangible medium of expression having computer-usable program code embodied in the medium.

[0022] Any combination of one or more computer-usable or computer-readable media may be utilized. For example, a computer-readable medium may include one or more of a portable computer diskette, a hard disk, a random access memory (RAM) device, a read-only memory (ROM) device, an erasable programmable read-only memory (EPROM or Flash memory) device, a portable compact disc read-only memory (CDROM), an optical storage device, and a magnetic storage device. Computer program code for carrying out operations of the present disclosure may be written in any combination of one or more programming languages. Such code may be compiled from source code to computer-readable assembly language or machine code suitable for the device or computer on which the code will be executed.

[0023] Embodiments may also be implemented in cloud computing environments. In this description and the following claims, "cloud computing" may be defined as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned via virtualization and released with minimal management effort or service provider interaction and then scaled accordingly. A cloud model can be composed of various characteristics (e.g., on-demand self-service, broad network access, resource pooling, rapid

elasticity, and measured service), service models (e.g., Software as a Service ("SaaS"), Platform as a Service ("PaaS"), and Infrastructure as a Service ("IaaS")), and deployment models (e.g., private cloud, community cloud, public cloud, and hybrid cloud).

[0024] The flow diagrams and block diagrams in the attached figures illustrate the architecture, functionality, and operation of possible implementations of systems, methods, and computer program products according to various embodiments of the present disclosure. In this regard, each block in the flow diagrams or block diagrams may represent a module, segment, or portion of code, which comprises one or more executable instructions for implementing the specified logical function(s). It will also be noted that each block of the block diagrams and/or flow diagrams, and combinations of blocks in the block diagrams and/or flow diagrams, may be implemented by special purpose hardware-based systems that perform the specified functions or acts, or combinations of special purpose hardware and computer instructions. These computer program instructions may also be stored in a computer-readable medium that can direct a computer or other programmable data processing apparatus to function in a particular manner, such that the instructions stored in the computer-readable medium produce an article of manufacture including instruction means which implement the function/act specified in the flow diagram and/or block diagram block or blocks.

[0025] The systems and methods described herein provide a flexible and scalable data warehouse using a new data processing platform. In some embodiments, the described systems and methods leverage a cloud infrastructure that supports cloud-based storage resources, computing resources, and the like. Example cloud-based storage resources offer significant storage capacity available on-demand at a low cost. Further, these cloud-based storage resources may be fault-tolerant and highly scalable, which can be costly to achieve in private data storage

systems. Example cloud-based computing resources are available on-demand and may be priced based on actual usage levels of the resources. Typically, the cloud infrastructure is dynamically deployed, reconfigured, and decommissioned in a rapid manner.

[0026] In the described systems and methods, a data storage system utilizes an SQL (Structured Query Language)-based relational database. However, these systems and methods are applicable to any type of database, and any type of data storage and retrieval platform, using any data storage architecture and using any language to store and retrieve data within the data storage and retrieval platform. The systems and methods described herein further provide a multi-tenant system that supports isolation of computing resources and data between different customers/clients and between different users within the same customer/client.

[0027] FIG. 1 is a block diagram depicting an example embodiment of a new data processing platform 100. As shown in FIG. 1, a resource manager 102 is coupled to multiple users 104, 106, and 108. In particular implementations, resource manager 102 can support any number of users desiring access to data processing platform 100. Users 104-108 may include, for example, end users providing data storage and retrieval requests, system administrators managing the systems and methods described herein, and other components/devices that interact with resource manager 102. Resource manager 102 provides various services and functions that support the operation of all systems and components within data processing platform 100. As used herein, resource manager 102 may also be referred to as a "global services system" that performs various functions as discussed herein.

[0028] Resource manager 102 is also coupled to metadata 110, which is associated with the entirety of data stored throughout data processing platform 100. In some embodiments, metadata 110 includes a summary of data stored in remote data storage systems as well as data

available from a local cache. Additionally, metadata 110 may include information regarding how data is organized in the remote data storage systems and the local caches. Metadata 110 allows systems and services to determine whether a piece of data needs to be accessed without loading or accessing the actual data from a storage device.

[0029] Resource manager 102 is further coupled to an execution platform 112, which provides multiple computing resources that execute various data storage and data retrieval tasks, as discussed in greater detail below. Execution platform 112 is coupled to multiple data storage devices 116, 118, and 120 that are part of a storage platform 114. Although three data storage devices 116, 118, and 120 are shown in FIG. 1, execution platform 112 is capable of communicating with any number of data storage devices. In some embodiments, data storage devices 116, 118, and 120 are cloud-based storage devices located in one or more geographic locations. For example, data storage devices 116, 118, and 120 may be part of a public cloud infrastructure or a private cloud infrastructure. Data storage devices 116, 118, and 120 may be hard disk drives (HDDs), solid state drives (SSDs), storage clusters, Amazon S3™ storage systems or any other data storage technology. Additionally, storage platform 114 may include distributed file systems (such as Hadoop Distributed File Systems (HDFS)), object storage systems, and the like.

[0030] In particular embodiments, the communication links between resource manager 102 and users 104-108, metadata 110, and execution platform 112 are implemented via one or more data communication networks. Similarly, the communication links between execution platform 112 and data storage devices 116-120 in storage platform 114 are implemented via one or more data communication networks. These data communication networks may utilize any communication protocol and any type of communication medium. In some embodiments, the

data communication networks are a combination of two or more data communication networks (or sub-networks) coupled to one another. In alternate embodiments, these communication links are implemented using any type of communication medium and any communication protocol.

[0031] As shown in FIG. 1, data storage devices 116, 118, and 120 are decoupled from the computing resources associated with execution platform 112. This architecture supports dynamic changes to data processing platform 100 based on the changing data storage/retrieval needs as well as the changing needs of the users and systems accessing data processing platform 100. The support of dynamic changes allows data processing platform 100 to scale quickly in response to changing demands on the systems and components within data processing platform 100. The decoupling of the computing resources from the data storage devices supports the storage of large amounts of data without requiring a corresponding large amount of computing resources. Similarly, this decoupling of resources supports a significant increase in the computing resources utilized at a particular time without requiring a corresponding increase in the available data storage resources.

[0032] Resource manager 102, metadata 110, execution platform 112, and storage platform 114 are shown in FIG. 1 as individual components. However, each of resource manager 102, metadata 110, execution platform 112, and storage platform 114 may be implemented as a distributed system (e.g., distributed across multiple systems/platforms at multiple geographic locations). Additionally, each of resource manager 102, metadata 110, execution platform 112, and storage platform 114 can be scaled up or down (independently of one another) depending on changes to the requests received from users 104-108 and the changing needs of data processing platform 100. Thus, in the described embodiments, data processing platform 100 is dynamic and supports regular changes to meet the current data processing needs.

[0033] During typical operation, data processing platform 100 processes multiple queries (or requests) received from any of the users 104-108. These queries are managed by resource manager 102 to determine when and how to execute the queries. For example, resource manager 102 may determine what data is needed to process the query and further determine which nodes within execution platform 112 are best suited to process the query. Some nodes may have already cached the data needed to process the query and, therefore, are good candidates for processing the query. Metadata 110 assists resource manager 102 in determining which nodes in execution platform 112 already cache at least a portion of the data needed to process the query. One or more nodes in execution platform 112 process the query using data cached by the nodes and, if necessary, data retrieved from storage platform 114. It is desirable to retrieve as much data as possible from caches within execution platform 112 because the retrieval speed is typically much faster than retrieving data from storage platform 114.

[0034] As shown in FIG. 1, data processing platform 100 separates execution platform 112 from storage platform 114. In this arrangement, the processing resources and cache resources in execution platform 112 operate independently of the data storage resources 116-120 in storage platform 114. Thus, the computing resources and cache resources are not restricted to specific data storage resources 116-120. Instead, all computing resources and all cache resources may retrieve data from, and store data to, any of the data storage resources in storage platform 114. Additionally, data processing platform 100 supports the addition of new computing resources and cache resources to execution platform 112 without requiring any changes to storage platform 114. Similarly, data processing platform 100 supports the addition of data storage resources to storage platform 114 without requiring any changes to nodes in execution platform 112.

[0035] FIG. 2 is a block diagram depicting an embodiment of resource manager 102. As shown in FIG. 2, resource manager 102 includes an access manager 202 and a key manager 204 coupled to a data storage device 206. Access manager 202 handles authentication and authorization tasks for the systems described herein. Key manager 204 manages storage and authentication of keys used during authentication and authorization tasks. For example, access manager 202 and key manager 204 manage the keys used to access data stored in remote storage devices (e.g., data storage devices in storage platform 114). As used herein, the remote storage devices may also be referred to as "persistent storage devices." A request processing service 208 manages received data storage requests and data retrieval requests (e.g., database queries). For example, request processing service 208 may determine the data necessary to process the received data storage request or data retrieval request. The necessary data may be stored in a cache within execution platform 112 (as discussed in greater detail below) or in a data storage device in storage platform 114. Request processing service 208 may be implemented using a "request processing module." A management console service 210 supports access to various systems and processes by administrators and other system managers. Additionally, management console service 210 may receive requests from users 104-108 to issue queries and monitor the workload on the system. Management console service 210 may be implemented using a "management console module." In some embodiments, a particular user may issue a request to monitor the workload that their specific query places on the system.

[0036] Resource manager 102 also includes an SQL compiler 212, an SQL optimizer 214 and an SQL executor 210. SQL compiler 212 parses SQL queries and generates the execution code for the queries. SQL optimizer 214 determines the best method to execute queries based on the data that needs to be processed. SQL optimizer 214 also handles various data pruning

operations and other data optimization techniques to improve the speed and efficiency of executing the SQL query. SQL executor 216 executes the query code for queries received by resource manager 102.

[0037] A query scheduler and coordinator 218 sends received queries to the appropriate services or systems for compilation, optimization, and dispatch to execution platform 112. For example, queries may be prioritized and processed in that prioritized order. In some embodiments, query scheduler and coordinator 218 identifies or assigns particular nodes in execution platform 112 to process particular queries. A virtual warehouse manager 220 manages the operation of multiple virtual warehouses implemented in execution platform 112. As discussed below, each virtual warehouse includes multiple execution nodes that each include a cache and a processor.

[0038] Additionally, resource manager 102 includes a configuration and metadata manager 222, which manages the information related to the data stored in the remote data storage devices and in the local caches (i.e., the caches in execution platform 112). As discussed in greater detail below, configuration and metadata manager 222 uses the metadata to determine which data files need to be accessed to retrieve data for processing a particular query. A monitor and workload analyzer 224 oversees the processes performed by resource manager 102 and manages the distribution of tasks (e.g., workload) across the virtual warehouses and execution nodes in execution platform 112. Monitor and workload analyzer 224 also redistributes tasks, as needed, based on changing workloads throughout data processing platform 100. Configuration and metadata manager 222 and monitor and workload analyzer 224 are coupled to a data storage device 226. Data storage devices 206 and 226 in FIG. 2 represent any data storage device within data processing platform 100. For example, data storage devices 206 and 226 may represent

caches in execution platform 112, storage devices in storage platform 114, or any other storage device.

[0039] Resource manager 102 also includes a transaction management and access control module 228, which manages the various tasks and other activities associated with the processing of data storage requests and data access requests. For example, transaction management and access control module 228 provides consistent and synchronized access to data by multiple users or systems. Since multiple users/systems may access the same data simultaneously, changes to the data must be synchronized to ensure that each user/system is working with the current version of the data. Transaction management and access control module 228 provides control of various data processing activities at a single, centralized location in resource manager 102. In some embodiments, transaction management and access control module 228 interacts with SQL executor 216 to support the management of various tasks being executed by SQL executor 216.

[0040] FIG. 3 is a block diagram depicting an embodiment of an execution platform 112. As shown in FIG. 3, execution platform 112 includes multiple virtual warehouses 302, 304, and 306. Each virtual warehouse includes multiple execution nodes that each include a data cache and a processor. Virtual warehouses 302, 304, and 306 are capable of executing multiple queries (and other tasks) in parallel by using the multiple execution nodes. As discussed herein, execution platform 112 can add new virtual warehouses and drop existing virtual warehouses in real time based on the current processing needs of the systems and users. This flexibility allows execution platform 112 to quickly deploy large amounts of computing resources when needed without being forced to continue paying for those computing resources when they are no longer needed. All virtual warehouses can access data from any data storage device (e.g., any storage device in storage platform 114).

[0041] Although each virtual warehouse 302-306 shown in FIG. 3 includes three execution nodes, a particular virtual warehouse may include any number of execution nodes. Further, the number of execution nodes in a virtual warehouse is dynamic, such that new execution nodes are created when additional demand is present, and existing execution nodes are deleted when they are no longer necessary.

[0042] Each virtual warehouse 302-306 is capable of accessing any of the data storage devices 116-120 shown in FIG. 1. Thus, virtual warehouses 302-306 are not necessarily assigned to a specific data storage device 116-120 and, instead, can access data from any of the data storage devices 116-120. Similarly, each of the execution nodes shown in FIG. 3 can access data from any of the data storage devices 116-120. In some embodiments, a particular virtual warehouse or a particular execution node may be temporarily assigned to a specific data storage device, but the virtual warehouse or execution node may later access data from any other data storage device.

[0043] In the example of FIG. 3, virtual warehouse 302 includes three execution nodes 308, 310, and 312. Execution node 308 includes a cache 314 and a processor 316. Execution node 310 includes a cache 318 and a processor 320. Execution node 312 includes a cache 322 and a processor 324. Each execution node 308-312 is associated with processing one or more data storage and/or data retrieval tasks. For example, a particular virtual warehouse may handle data storage and data retrieval tasks associated with a particular user or customer. In other implementations, a particular virtual warehouse may handle data storage and data retrieval tasks associated with a particular data storage system or a particular category of data.

[0044] Similar to virtual warehouse 302 discussed above, virtual warehouse 304 includes three execution nodes 326, 328, and 330. Execution node 326 includes a cache 332 and a

processor 334. Execution node 328 includes a cache 336 and a processor 338. Execution node 330 includes a cache 340 and a processor 342. Additionally, virtual warehouse 306 includes three execution nodes 344, 346, and 348. Execution node 344 includes a cache 350 and a processor 352. Execution node 346 includes a cache 354 and a processor 356. Execution node 348 includes a cache 358 and a processor 360.

[0045] In some embodiments, the execution nodes shown in FIG. 3 are stateless with respect to the data the execution nodes are caching. For example, these execution nodes do not store or otherwise maintain state information about the execution node or the data being cached by a particular execution node. Thus, in the event of an execution node failure, the failed node can be transparently replaced by another node. Since there is no state information associated with the failed execution node, the new (replacement) execution node can easily replace the failed node without concern for recreating a particular state.

[0046] Although the execution nodes shown in FIG. 3 each include one data cache and one processor, alternate embodiments may include execution nodes containing any number of processors and any number of caches. Additionally, the caches may vary in size among the different execution nodes. The caches shown in FIG. 3 store, in the local execution node, data that was retrieved from one or more data storage devices in storage platform 114 (FIG. 1). Thus, the caches reduce or eliminate the bottleneck problems occurring in platforms that consistently retrieve data from remote storage systems. Instead of repeatedly accessing data from the remote storage devices, the systems and methods described herein access data from the caches in the execution nodes which is significantly faster and avoids the bottleneck problem discussed above. In some embodiments, the caches are implemented using high-speed memory devices that

provide fast access to the cached data. Each cache can store data from any of the storage devices in storage platform 114.

[0047] Further, the cache resources and computing resources may vary between different execution nodes. For example, one execution node may contain significant computing resources and minimal cache resources, making the execution node useful for tasks that require significant computing resources. Another execution node may contain significant cache resources and minimal computing resources, making this execution node useful for tasks that require caching of large amounts of data. Yet another execution node may contain cache resources providing faster input-output operations, useful for tasks that require fast scanning of large amounts of data. In some embodiments, the cache resources and computing resources associated with a particular execution node are determined when the execution node is created, based on the expected tasks to be performed by the execution node.

[0048] Additionally, the cache resources and computing resources associated with a particular execution node may change over time based on changing tasks performed by the execution node. For example, a particular execution node may be assigned more processing resources if the tasks performed by the execution node become more processor intensive. Similarly, an execution node may be assigned more cache resources if the tasks performed by the execution node require a larger cache capacity.

[0049] Although virtual warehouses 302-306 are associated with the same execution platform 112, the virtual warehouses may be implemented using multiple computing systems at multiple geographic locations. For example, virtual warehouse 302 can be implemented by a computing system at a first geographic location, while virtual warehouses 304 and 306 are implemented by another computing system at a second geographic location. In some

embodiments, these different computing systems are cloud-based computing systems maintained by one or more different entities.

[0050] Additionally, each virtual warehouse is shown in FIG. 3 as having multiple execution nodes. The multiple execution nodes associated with each virtual warehouse may be implemented using multiple computing systems at multiple geographic locations. For example, a particular instance of virtual warehouse 302 implements execution nodes 308 and 310 on one computing platform at a particular geographic location, and implements execution node 312 at a different computing platform at another geographic location. Selecting particular computing systems to implement an execution node may depend on various factors, such as the level of resources needed for a particular execution node (e.g., processing resource requirements and cache requirements), the resources available at particular computing systems, communication capabilities of networks within a geographic location or between geographic locations, and which computing systems are already implementing other execution nodes in the virtual warehouse.

[0051] Execution platform 112 is also fault tolerant. For example, if one virtual warehouse fails, that virtual warehouse is quickly replaced with a different virtual warehouse at a different geographic location.

[0052] A particular execution platform 112 may include any number of virtual warehouses 302-306. Additionally, the number of virtual warehouses in a particular execution platform is dynamic, such that new virtual warehouses are created when additional processing and/or caching resources are needed. Similarly, existing virtual warehouses may be deleted when the resources associated with the virtual warehouse are no longer necessary.

[0053] In some embodiments, virtual warehouses 302, 304, and 306 may operate on the same data in storage platform 114, but each virtual warehouse has its own execution nodes with independent processing and caching resources. This configuration allows requests on different virtual warehouses to be processed independently and with no interference between the requests. This independent processing, combined with the ability to dynamically add and remove virtual warehouses, supports the addition of new processing capacity for new users without impacting the performance observed by the existing users.

[0054] FIG. 4 is a block diagram depicting an example operating environment 400 with multiple users accessing multiple databases through multiple virtual warehouses. In environment 400, multiple users 402, 404, and 406 access multiple databases 414, 416, 418, 420, 422, and 424 through multiple virtual warehouses 408, 410, and 412. Although not shown in FIG. 4, users 402, 404, and 406 may access virtual warehouses 408, 410, and 412 through resource manager 102 (FIG. 1). In particular embodiments, databases 414-424 are contained in storage platform 114 (FIG. 1) and are accessible by any virtual warehouse implemented in execution platform 112. In some embodiments, users 402-406 access one of the virtual warehouses 408-412 using a data communication network, such as the Internet. In some implementations, each user 402-406 specifies a particular virtual warehouse 408-412 to work with at a specific time. In the example of FIG. 4, user 402 interacts with virtual warehouse 408, user 404 interacts with virtual warehouse 410, and user 406 interacts with virtual warehouse 412. Thus, user 402 submits data retrieval and data storage requests through virtual warehouse 408. Similarly, users 404 and 406 submit data retrieval and data storage requests through virtual warehouses 410 and 412, respectively.

[0055] Each virtual warehouse 408-412 is configured to communicate with a subset of all databases 414-424. For example, in environment 400, virtual warehouse 408 is configured to communicate with databases 414, 416, and 422. Similarly, virtual warehouse 410 is configured to communicate with databases 416, 418, 420, and 424. And, virtual warehouse 412 is configured to communicate with databases 416, 422, and 424. In alternate embodiments, one or more of virtual warehouses 408-412 communicate with all of the databases 414-424. The arrangement shown in FIG. 4 allows individual users to send all data retrieval and data storage requests through a single virtual warehouse. That virtual warehouse processes the data retrieval and data storage tasks using cached data within one of the execution nodes in the virtual warehouse, or retrieves (and caches) the necessary data from an appropriate database. The mappings between the virtual warehouses is a logical mapping, not a hardware mapping. This logical mapping is based on access control parameters related to security and resource access management settings. The logical mappings are easily changed without requiring reconfiguration of the virtual warehouse or storage resources.

[0056] Although environment 400 shows virtual warehouses 408-412 configured to communicate with specific subsets of databases 414-424, that configuration is dynamic. For example, virtual warehouse 408 may be reconfigured to communicate with a different subset of databases 414-424 based on changing tasks to be performed by virtual warehouse 408. For instance, if virtual warehouse 408 receives requests to access data from database 418, virtual warehouse 408 may be reconfigured to also communicate with database 418. If, at a later time, virtual warehouse 408 no longer needs to access data from database 418, virtual warehouse 408 may be reconfigured to delete the communication with database 418.

[0057] FIG. 5 is a block diagram depicting another example operating environment 500 with multiple users accessing multiple databases through a load balancer and multiple virtual warehouses contained in a virtual warehouse group. Environment 500 is similar to environment 400 (FIG. 4), but additionally includes a virtual warehouse resource manager 508 and multiple virtual warehouses 510, 512, and 514 arranged in a virtual warehouse group 516. Virtual warehouse resource manager 508 may be contained in resource manager 102. In particular, multiple users 502, 504, and 506 access multiple databases 518, 520, 522, 524, 526, and 528 through virtual warehouse resource manager 508 and virtual warehouse group 516. In some embodiments, users 502-506 access virtual warehouse resource manager 508 using a data communication network, such as the Internet. Although not shown in FIG. 5, users 502, 504, and 506 may access virtual warehouse resource manager 508 through resource manager 102 (FIG. 1). In some embodiments, virtual warehouse resource manager 508 is implemented within resource manager 102.

[0058] Users 502-506 may submit data retrieval and data storage requests to virtual warehouse resource manager 508, which routes the data retrieval and data storage requests to an appropriate virtual warehouse 510-514 in virtual warehouse group 516. In some implementations, virtual warehouse resource manager 508 provides a dynamic assignment of users 502-506 to virtual warehouses 510-514. When submitting a data retrieval or data storage request, users 502-506 may specify virtual warehouse group 516 to process the request without specifying the particular virtual warehouse 510-514 that will process the request. This arrangement allows virtual warehouse resource manager 508 to distribute multiple requests across the virtual warehouses 510-514 based on efficiency, available resources, and the availability of cached data within the virtual warehouses 510-514. When determining how to

route data processing requests, virtual warehouse resource manager 508 considers available resources, current resource loads, number of current users, and the like.

[0059] In some embodiments, fault tolerance systems create a new virtual warehouses in response to a failure of a virtual warehouse. The new virtual warehouse may be in the same virtual warehouse group or may be created in a different virtual warehouse group at a different geographic location.

[0060] Each virtual warehouse 510-514 is configured to communicate with a subset of all databases 518-528. For example, in environment 500, virtual warehouse 510 is configured to communicate with databases 518, 520, and 526. Similarly, virtual warehouse 512 is configured to communicate with databases 520, 522, 524, and 528. And, virtual warehouse 514 is configured to communicate with databases 520, 526, and 528. In alternate embodiments, virtual warehouses 510-514 may communicate with any (or all) of the databases 518-528.

[0061] Although environment 500 shows one virtual warehouse group 516, alternate embodiments may include any number of virtual warehouse groups, each associated with any number of virtual warehouses. The number of virtual warehouse groups in a particular environment is dynamic and may change based on the changing needs of the users and other systems in the environment.

[0062] FIG. 6 is a block diagram depicting another example operating environment 600 having multiple distributed virtual warehouses and virtual warehouse groups. Environment 600 includes resource manager 102 that communicates with virtual warehouse groups 604 and 606 through a data communication network 602. Warehouse group 604 includes two virtual warehouses 608 and 610, and warehouse group 606 includes another two virtual warehouses 614

and 616. Resource manager 102 also communicates with virtual warehouse 612 (which is not part of a virtual warehouse group) through data communication network 602.

[0063] Virtual warehouse groups 604 and 606 as well as virtual warehouse 612 communicate with databases 620, 622, and 624 through a data communication network 618. In some embodiments data communication networks 602 and 618 are the same network. Environment 600 allows resource manager 102 to coordinate user data storage and retrieval requests across the multiple virtual warehouses 608-616 to store and retrieve data in databases 620-624. Virtual warehouse groups 604 and 606 can be located in the same geographic area, or can be separated geographically. Additionally, virtual warehouse groups 604 and 606 can be implemented by the same entity or by different entities.

[0064] The systems and methods described herein allow data to be stored and accessed as a service that is separate from computing (or processing) resources. Even if no computing resources have been allocated from the execution platform, data is available to a virtual warehouse without requiring reloading of the data from a remote data source. Thus, data is available independently of the allocation of computing resources associated with the data. The described systems and methods are useful with any type of data. In particular embodiments, data is stored in a structured, optimized format. The decoupling of the data storage/access service from the computing services also simplifies the sharing of data among different users and groups. As discussed herein, each virtual warehouse can access any data to which it has access permissions, even at the same time as other virtual warehouses are accessing the same data. This architecture supports running queries without any actual data stored in the local cache. The systems and methods described herein are capable of transparent dynamic data movement, which moves data from a remote storage device to a local cache, as needed, in a manner that is

transparent to the user of the system. Further, this architecture supports data sharing without prior data movement since any virtual warehouse can access any data due to the decoupling of the data storage service from the computing service.

[0065] FIG. 7 is a flow diagram depicting an embodiment of a method 700 for managing data storage and retrieval operations. Initially, method 700 receives a statement, request or query from a user at 702. A statement is any request or command to perform a data-related operation. Example statements include data retrieval requests, data storage requests, data transfer requests, data queries, and the like. In some embodiments, the statement is implemented as an SQL statement. A resource manager creates a query coordinator at 704 to manage the received statement. For example, the query coordinator manages the various tasks necessary to process the received statement, including interacting with an execution platform and one or more data storage devices. In some embodiments, the query coordinator is a temporary routine created specifically to manage the received statement.

[0066] Method 700 continues as the resource manager determines multiple tasks necessary to process the received statement at 706. The multiple tasks may include, for example, accessing data from a cache in an execution node, retrieving data from a remote storage device, updating data in a cache, storing data in a remote storage device, and the like. The resource manager also distributes the multiple tasks to execution nodes in the execution platform at 708. As discussed herein, the execution nodes in the execution platform are implemented within virtual warehouses. Each execution node performs an assigned task and returns a task result to the resource manager at 710. In some embodiments, the execution nodes return the task results to the query coordinator. The resource manager receives the multiple task results and creates a statement result at 712, and communicates the statement result to the user at 714. In some

embodiments, the query coordinator is deleted after the statement result is communicated to the user.

[0067] FIG. 8 is a flow diagram depicting an embodiment of a method 800 for managing the processing of multiple files by multiple execution nodes. In particular embodiments, method 800 is performed by resource manager 102. Initially, method 800 receives (or identifies) a query from a user at 802, and identifies multiple files necessary to process the received query at 804. The files needed to process a particular query will vary from one query to the next. The query parameters and query instructions indicate the data to be processed and, accordingly, indicate the files necessary to access the data. For example, if a query is associated with financial records for a particular company in a particular date range, the necessary files include all files that contain data for the particular company and within the particular date range. To process the multiple files at substantially the same time, the multiple files are distributed to multiple execution nodes. To accomplish this, method 800 identifies multiple execution nodes that are currently available to process the multiple files at 806.

[0068] Method 800 continues at 808 by creating multiple scansets, where each scanset includes a portion of the multiple files. A scanset is any collection of one or more files. The union of all scansets includes all files necessary to process the received query. Different scansets may contain different numbers of files. Each scanset is initially assigned to a particular execution node based on a consistent file assignment model. This is an initial assignment of scanset files because certain files may subsequently be reassigned to a different execution node, as discussed below. The consistent file assignment model defines an approach for assigning files to execution nodes, and is used each time files related to a query are assigned to execution nodes for processing. By repeatedly using the same file assignment model, most files are assigned to

the same execution nodes for processing, thereby increasing the likelihood that the assigned file is already in the execution node's cache which maintains a high cache hit ratio. The files in each scanset are arranged (or ordered) based on the consistent file assignment model. In some embodiments, the files in each scanset are arranged based on their size such that the arrangement is the same if the arrangement is repeated multiple times (e.g., for multiple queries accessing similar files). For example, files in the scansets may be arranged from largest to smallest, or from smallest to largest. As discussed below, the assignment of scansets to the execution nodes may use, for example, a consistent hashing approach.

[0069] In some embodiments, any algorithm may be used to assign scansets to the execution nodes. The goal of the algorithm is to assign the scansets in a manner that maximizes the probability that an execution node will find the file in its cache. This can be accomplished by using the same algorithm or approach to assigning scansets to execution nodes. By consistently assigning scansets in the same manner, an execution node is more likely to have a necessary file in its cache. Additionally, consistently ordering the files assigned to each execution node (i.e., the order in which the execution node processes the files) will increase the likelihood that the first files processed are in the cache. In some embodiments, the files are ordered in a manner that files which are least likely to be in the cache are processed last, which increases the likelihood they will be stolen by another execution node, as discussed herein.

[0070] Multiple execution nodes begin processing the files in their associated scansets in parallel at 810. The files in a particular scanset are processed by an execution node in the arrangement (or order) previously determined using the consistent file assignment model. When a particular execution node finishes processing all files in its assigned scanset, that execution node steals an unprocessed file from another execution node based on a file stealing model at

812. As used herein, "stealing" a file refers to reassigning the file from a first execution node to a second execution node. The terms "stealing" and "reassigning" are used interchangeably herein. The file stealing model defines an approach for stealing files between execution nodes. When stealing a file for a particular execution node, that execution node is assigned a file that it would have received originally if the other execution node (i.e., the execution node with the unprocessed file) was not available when the scansets were created. In some embodiments, stolen files are selected in reverse order within the scanset (e.g., stolen files are selected from the bottom of the ordered list of files in the scanset).

[0071] Based on the file stealing model, a particular unprocessed file is selected for processing by the available execution node. This unprocessed file is removed from the original execution node's scanset and processed by the available execution node at 814. Method 800 continues by identifying other execution nodes that have finished processing all files, and instructs the execution nodes to steal unprocessed files. This continues until all files in all scansets have been processed. In some embodiments, the file stealing model uses a consistent hashing algorithm, as discussed herein. In other embodiments, the file stealing model may use any algorithm or process that provides a consistent selection of files to be stolen (or reassigned) from one execution node to another execution node. This consistent selection of files generally increases the cache hit ratio.

[0072] The stealing process improves overall system performance by fully utilizing all execution node resources. Instead of allowing one execution node to remain idle while other execution nodes have files waiting to be processed, the idle execution node can process the waiting files in parallel with other execution node processes. Additional details regarding stealing (or reassigning) unprocessed files are discussed below.

[0073] In some embodiments, the file stealing process also uses the consistent hashing model. In these embodiments, each execution node has ownership of all files, but at different levels. For example, a highest level of ownership (level 0) indicates that the file was assigned to that execution node initially. After a particular execution node has processed all of its initially assigned files (level 0 files), the execution node then proceeds to process level 1 files, then level 2 files, and so forth. A level 1 file indicates that the file would have been initially assigned to the execution node if the immediately adjacent execution node was not present. In some embodiments, when one execution node steals a file from another execution node, the "stealing" execution node is the only one able to steal the file because it is the only adjacent execution node (i.e., adjacent to the other execution node from which the file is stolen). This prevents any competition between multiple execution nodes trying to steal the same file.

[0074] In particular implementations, query scheduler and coordinator 218 (FIG. 2) is responsible for managing these types of file activities. In some embodiments, query scheduler and coordinator 218 includes a file manager that manages the distribution of files among the various virtual warehouses and execution nodes within the virtual warehouses. This file manager also manages the stealing (or reassignment) of files between different execution nodes.

[0075] Use of the consistent hashing algorithm discussed herein allows peer-to-peer operation, thereby eliminating the need for centralized logic to manage each of the individual execution nodes. Instead, every execution node knows of all other execution nodes in the ring. When stealing a file, the "stealing" execution node asks the adjacent execution node if it has any files available to be stolen. If no level 1 files remain to be stolen in the adjacent execution node, the stealing execution node will move to the next execution node in the ring. If no level 1 files

remain in any of the execution nodes, the stealing execution node will move to level 2 files, and so forth.

[0076] FIG. 9 is a flow diagram depicting an embodiment of a method 900 for managing the stealing of files from an execution node. In particular embodiments, method 900 is performed by resource manager 102. Initially, a first execution node is instructed to steal a file from a second execution node at 902. As discussed above with respect to FIG. 8, this file stealing may occur, for example, when a particular execution node has processed all files in its assigned scanset, but additional files (in other nodes' scansets) remain unprocessed.

[0077] The first execution node begins retrieving the file from a remote storage device at 904. Method 900 determines whether the file retrieval (from the remote storage device) by the first execution node is complete at 906. If the file retrieval is complete, the second execution node is instructed not to process the file at 908. In some embodiments, in addition to instructing the second execution node not to process the file, the second execution node is instructed to remove the file from its scanset.

[0078] If the first execution node has not completed retrieving the file from the remote storage device at 906, method 900 determines whether the second execution node has become available to process the file at 910. If the second execution node is not available to process the file, method 900 returns to check for completion of the file retrieval by the first execution node at 906.

[0079] If the second execution node is available to process the file at 910, method 900 determines whether the file is available in the second execution node's cache at 912. If the file is not in the second execution node's cache, method 900 branches to 918 where the first execution node continues retrieving and processing the file, while the second execution node is instructed

not to process the file. In this situation, since the second execution node has not cached the file, the first execution node is allowed to continue retrieving the file from the remote storage device since that will likely be faster than starting a new file retrieval process by the second execution node.

[0080] If the file is available in the second node's cache at 912, the first execution node is instructed not to process the file at 914. Additionally, the second execution node is instructed to process the cached file at 916. In this situation, since the second execution node is available to process the file, and the file is already in the second execution node's cache, the second execution node can process the file faster than the first execution node, which is still retrieving the file from the remote storage device.

[0081] In some embodiments, one execution node can steal a file from another execution node by copying the file directly from the other execution node's cache. Thus, rather than having the execution node retrieve the stolen file from a remote storage system, it may be faster to retrieve the file from the other execution node's cache.

[0082] In some embodiments, consistent hashing is used as the underlying model to initially assign files to an execution node for processing, and to re-assign (or steal) files when an execution node has processed all of its initially assigned files. In one embodiment, consistent hashing performs a hash for each server in a cluster (e.g., using the physical server identifier) into a large hash space, such as a 64 bit hash space. To initially assign a file to an execution node, the file is hashed the same way using the file's unique identifier. The execution node associated with that file is the first execution node that appears in the hash space after the file hash is performed. This approach "wraps back" to zero when the maximum hash value is reached.

[0083] Using this consistent hashing approach supports the addition or removal of servers (and execution nodes) without causing a significantly different initial assignment of files. For example, in a system with 10 servers, the addition of one server will cause the reassignment of approximately 10% of the files. Thus, approximately 90% of the files are not reassigned. To minimize the risk of skew (some execution nodes perform more file processing work than other execution nodes), especially when the number of execution nodes is small, some embodiments create multiple replicas of each execution node in the hash space using multiple hash functions. In particular implementations, a system may create 128 replicas for each execution node and use a 64 bit hash value for the hash space.

[0084] FIGs. 10A-10D depict example embodiments of assigning files to execution nodes using consistent hashing. In particular embodiments, this assigning of files to execution nodes is performed by resource manager 102. FIG. 10A illustrates the allocation of 10 files to three execution nodes. Starting at the top of the circle in FIG. 10A and moving clockwise, File6 and File3 are assigned to the next execution node (Execution Node 3), then Files 1, 8, 4, and 5 are assigned to the next execution node (Execution Node 2), and finally Files 7, 10, 9, and 2 are assigned to the next execution node (Execution Node 1).

[0085] FIG. 10B shows the resulting file allocation after adding another execution node (Execution Node 4). In this example, the files associated with Execution Node 2 and Execution Node 3 are unchanged from FIG. 10A. The files associated with Execution Node 1 in FIG. 10A are shared between Execution Node 1 and Execution Node 4 in FIG. 10B. Thus, only a few files are reassigned as a result of adding Execution Node 4.

[0086] The examples shown in FIGs. 10A and 10B may have problems with skew (the situation where some execution nodes perform more file processing work than other execution

nodes) because only three execution nodes are available. To reduce the likelihood of skew, multiple replicas of each execution node are provided in the hash space.

[0087] FIG. 10C shows an example similar to FIG. 10A, but using eight replicas for each of the three execution nodes. This approach provides a more uniform allocation of files among the execution nodes.

[0088] FIG. 10D shows the resulting file allocation after adding another execution node (Execution Node 4). In this example, eight replicas of Execution Node 4 are added to the hash space. As shown in FIG. 10D, File 1 is moved from Execution Node 3 to the new Execution Node 4, and File 5 is moved from Execution Node 2 to Execution Node 4. Thus, files are moved from two different nodes rather than both moving from the same node as illustrated in FIGs. 10A and 10B.

[0089] The consistent hashing examples shown in FIGs. 10A-10B are useful in the initial assignment of scansets (or individual files) to execution nodes as well as reassigning (e.g., stealing) files from one execution node to another execution node. In both instances, the consistent hashing approach increases the likelihood of a cache hit for the files being processed by the execution nodes.

[0090] In some embodiments, when an available execution node is ready to steal a file, the consistent hashing approach is used to identify unprocessed files that would have been assigned to the available execution node if the other execution node (the execution node to which the file was initially assigned) was not available when the initial file assignments were performed. This approach increases the likelihood that the available execution node will be stealing a file that is already cached on the available execution node.

[0091] In some implementations, the same file is cached by multiple execution nodes at the same time. This multiple caching of files helps with load balancing (e.g., balancing data processing tasks) across multiple execution nodes. Additionally, caching a file in multiple execution nodes helps avoid potential bottlenecks when significant amounts of data are trying to pass through the same communication link. This implementation also supports the parallel processing of the same data by different execution nodes.

[0092] The systems and methods described herein take advantage of the benefits of both shared-disk systems and the shared-nothing architecture. The described platform for storing and retrieving data is scalable like the shared-nothing architecture once data is cached locally. It also has all the benefits of a shared-disk architecture where processing nodes can be added and removed without any constraints (e.g., for 0 to N) and without requiring any explicit reshuffling of data.

[0093] FIG. 11 is a block diagram depicting an example computing device 1100. In some embodiments, computing device 1100 is used to implement one or more of the systems and components discussed herein. For example, computing device 1100 may allow a user or administrator to access resource manager 102. Further, computing device 1100 may interact with any of the systems and components described herein. Accordingly, computing device 1100 may be used to perform various procedures and tasks, such as those discussed herein. Computing device 1100 can function as a server, a client or any other computing entity. Computing device 1100 can be any of a wide variety of computing devices, such as a desktop computer, a notebook computer, a server computer, a handheld computer, a tablet, and the like.

[0094] Computing device 1100 includes one or more processor(s) 1102, one or more memory device(s) 1104, one or more interface(s) 1106, one or more mass storage device(s)

1108, and one or more Input/Output (I/O) device(s) 1110, all of which are coupled to a bus 1112. Processor(s) 1102 include one or more processors or controllers that execute instructions stored in memory device(s) 1104 and/or mass storage device(s) 1108. Processor(s) 1102 may also include various types of computer-readable media, such as cache memory.

[0095] Memory device(s) 1104 include various computer-readable media, such as volatile memory (e.g., random access memory (RAM)) and/or nonvolatile memory (e.g., read-only memory (ROM)). Memory device(s) 1104 may also include rewritable ROM, such as Flash memory.

[0096] Mass storage device(s) 1108 include various computer readable media, such as magnetic tapes, magnetic disks, optical disks, solid state memory (e.g., Flash memory), and so forth. Various drives may also be included in mass storage device(s) 1108 to enable reading from and/or writing to the various computer readable media. Mass storage device(s) 1108 include removable media and/or non-removable media.

[0097] I/O device(s) 1110 include various devices that allow data and/or other information to be input to or retrieved from computing device 1100. Example I/O device(s) 1110 include cursor control devices, keyboards, keypads, microphones, monitors or other display devices, speakers, printers, network interface cards, modems, lenses, CCDs or other image capture devices, and the like.

[0098] Interface(s) 1106 include various interfaces that allow computing device 1100 to interact with other systems, devices, or computing environments. Example interface(s) 1106 include any number of different network interfaces, such as interfaces to local area networks (LANs), wide area networks (WANs), wireless networks, and the Internet.

[0099] Bus 1112 allows processor(s) 1102, memory device(s) 1104, interface(s) 1106, mass storage device(s) 1108, and I/O device(s) 1110 to communicate with one another, as well as other devices or components coupled to bus 1112. Bus 1112 represents one or more of several types of bus structures, such as a system bus, PCI bus, IEEE 1394 bus, USB bus, and so forth.

[00100] For purposes of illustration, programs and other executable program components are shown herein as discrete blocks, although it is understood that such programs and components may reside at various times in different storage components of computing device 1100, and are executed by processor(s) 1102. Alternatively, the systems and procedures described herein can be implemented in hardware, or a combination of hardware, software, and/or firmware. For example, one or more application specific integrated circuits (ASICs) can be programmed to carry out one or more of the systems and procedures described herein.

[00101] Although the present disclosure is described in terms of certain preferred embodiments, other embodiments will be apparent to those of ordinary skill in the art, given the benefit of this disclosure, including embodiments that do not provide all of the benefits and features set forth herein, which are also within the scope of this disclosure. It is to be understood that other embodiments may be utilized, without departing from the scope of the present disclosure.

CLAIMS

1. A method comprising:

identifying a plurality of files to process based on a received query;

identifying a plurality of execution nodes available to process the plurality of files;

initially creating a plurality of scansets and assigning each of the scansets to one of the plurality of execution nodes based on a file assignment model, wherein each scanset includes a portion of the plurality of files;

processing the multiple scansets by the plurality of execution nodes;

determining whether a particular execution node has finished processing all files in its assigned scanset; and

responsive to determining that a particular execution node has finished processing all files in its assigned scanset:

reassigning an unprocessed file from a second execution node; and

assigning the unprocessed file to the particular execution node for processing.

2. The method of claim 1, further comprising arranging the files in each scanset based on the size of each file.

3. The method of claim 1, further comprising arranging the files in each scanset to prioritize files cached by the assigned execution node.

4. The method of claim 1, wherein the file assignment model uses a consistent hashing model.

5. The method of claim 1, further comprising removing the unprocessed file from the scanset of the originally assigned execution node during the reassigning.

6. The method of claim 1, wherein reassigning an unprocessed file from another execution node includes identifying a file to reassign that is already cached by the execution node that finished processing all files in its assigned scanset.

7. The method of claim 1, wherein reassigning an unprocessed file from another execution node includes selecting the unprocessed file to reassign based on a file stealing model.

8. The method of claim 7, wherein the file stealing model uses consistent hashing at different ownership levels.

9. The method of claim 8, wherein the different ownership levels determine an order in which files are processed by each of the plurality of execution nodes.

10. The method of claim 1, wherein the particular execution node retrieves the unprocessed file from a remote storage device.

11. The method of claim 10, further comprising:

determining whether the second execution node becomes available to process the unprocessed file;

determining whether the second execution node has cached the unprocessed file;

determining whether the particular execution node has finished retrieving the unprocessed file from the remote storage device; and

instructing the particular execution node to stop processing the unprocessed file responsive to determining that the second execution node becomes available, the second execution node has cached the unprocessed file, and the particular execution node has not finished retrieving the unprocessed file from the remote storage device.

12. The method of claim 11, further comprising instructing the second execution node to process the unprocessed file responsive to determining that the second execution node becomes available, the second execution node has cached the unprocessed file, and the particular execution node has not finished retrieving the unprocessed file from the remote storage device.

13. An apparatus comprising:

a request processing module configured to identify a plurality of files associated with a received query;

a virtual warehouse manager configured to identify a plurality of execution nodes available to process the plurality of files; and

a transaction management module configured to create a plurality of scansets and assign each of the scansets to one of the plurality of execution nodes based on a file assignment model, the transaction management module further configured to determine whether a particular execution node has finished processing all files in its assigned scanset and, responsive to determining that a particular execution node has finished processing all files in its assigned

scanset, reassign an unprocessed file from a second node and assign the unprocessed file to the particular execution node for processing.

14. The apparatus of claim 13, wherein each scanset includes a portion of the plurality of files.

15. The apparatus of claim 13, wherein the transaction management module is further configured to arrange the files in each scanset based on the size of each file.

16. The apparatus of claim 13, wherein the file assignment model uses a consistent hashing model.

17. The apparatus of claim 13, wherein the transaction management module is further configured to select the unprocessed file based on a file stealing model.

18. The apparatus of claim 17, wherein the file stealing model uses a consistent hashing model.

19. An apparatus comprising:

means for processing a request to identify a plurality of files associated with a received query;

means for managing a virtual warehouse to identify a plurality of execution nodes available to process the plurality of files; and

management means for creating a plurality of scansets and assigning each of the scansets to one of the plurality of execution nodes based on a file assignment model, the management

means further for determining whether a particular execution node has finished processing all files in its assigned scanset and, responsive to determining that a particular execution node has finished processing all files in its assigned scanset, reassigning an unprocessed file from a second node and assigning the unprocessed file to the particular execution node for processing.

20. The apparatus of claim 19, wherein the file assignment model uses a consistent hashing model.

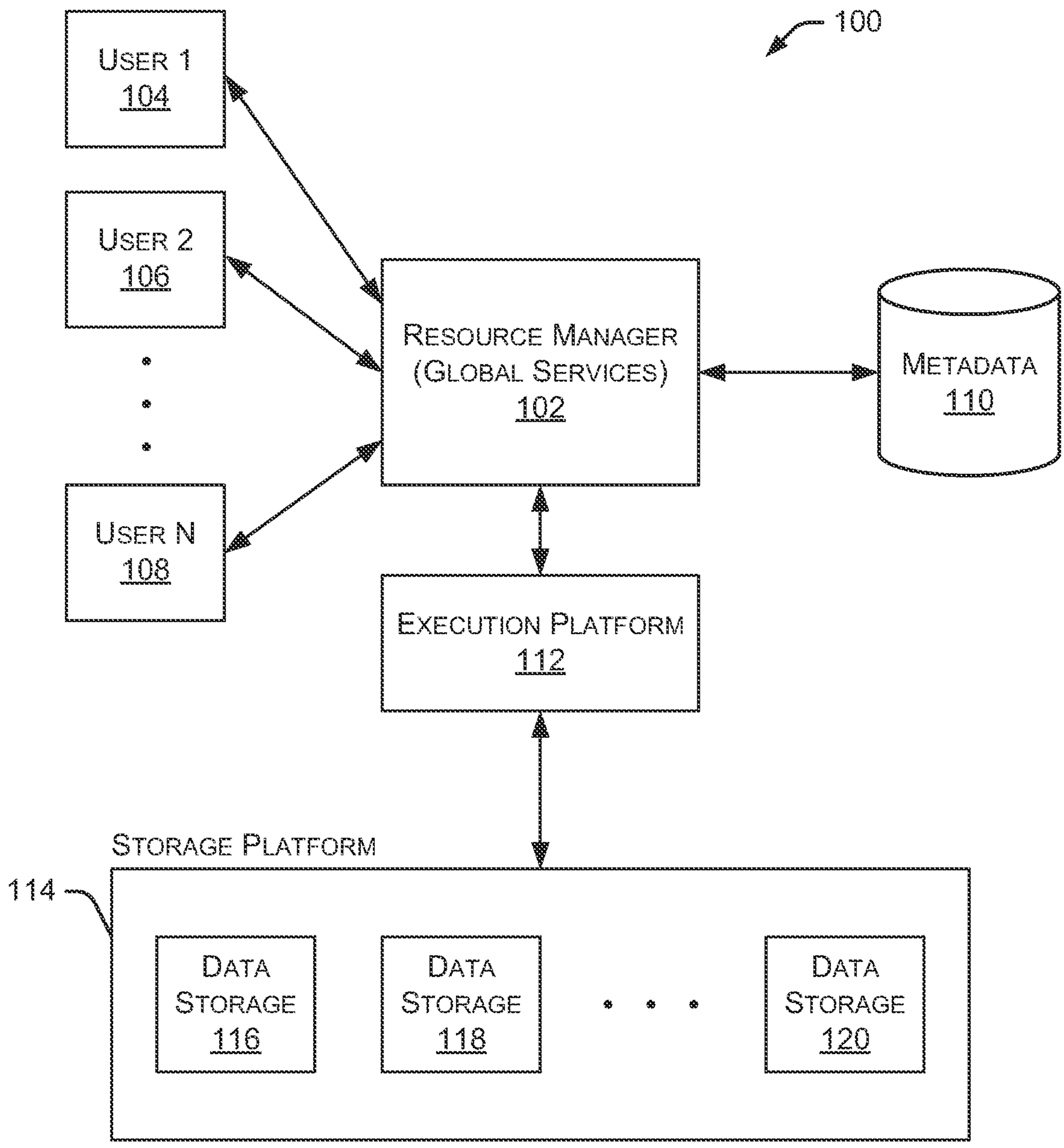

100
USER 1
104
USER 2
106
USER N
108
RESOURCE MANAGER
(GLOBAL SERVICES)
102
METADATA
110
EXECUTION PLATFORM
112
STORAGE PLATFORM
114
DATA
STORAGE
116
DATA
STORAGE
118
DATA
STORAGE
120

FIG. 1

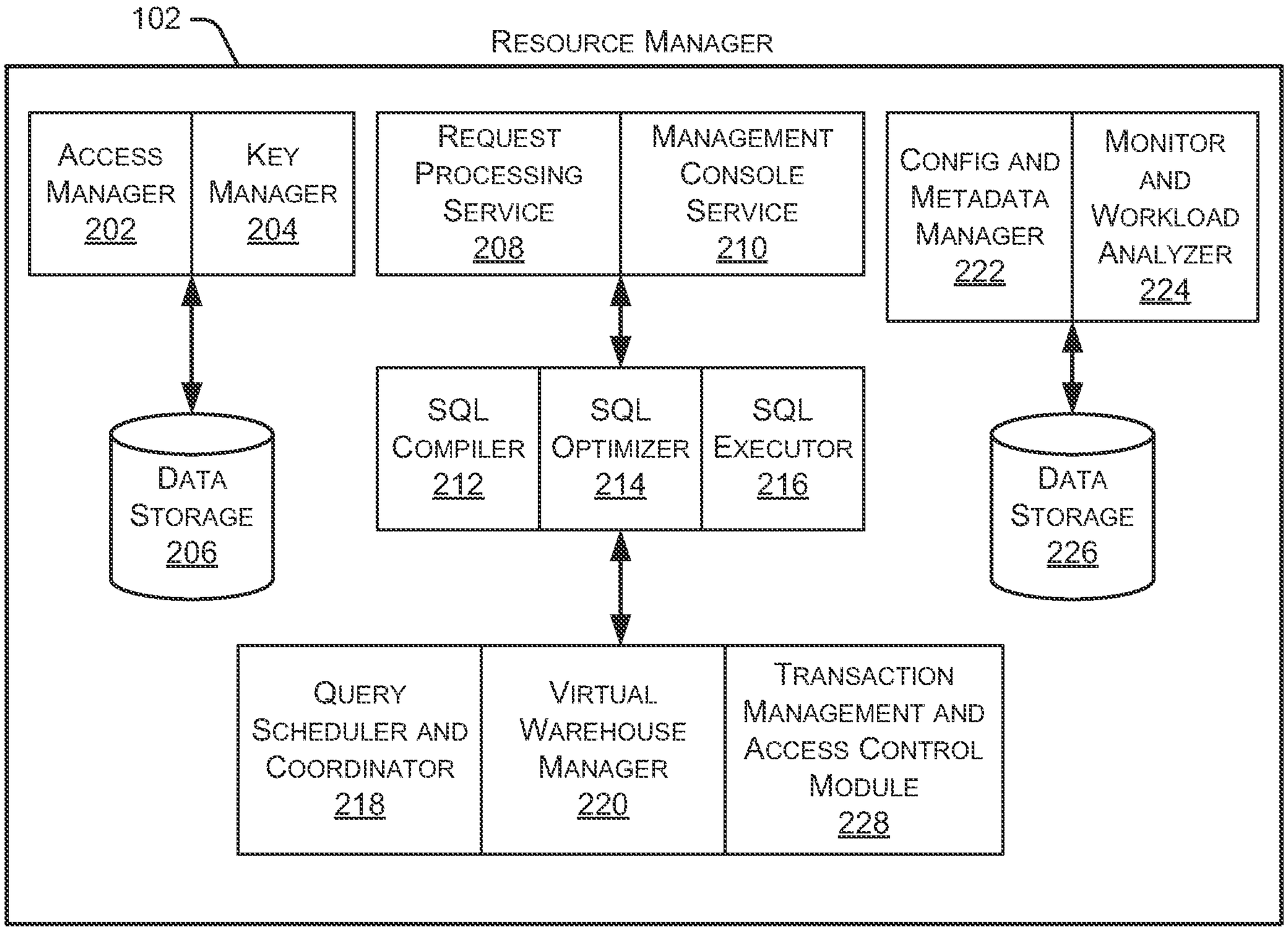

102
RESOURCE MANAGER
ACCESS MANAGER 202
KEY MANAGER 204
REQUEST PROCESSING SERVICE 208
MANAGEMENT CONSOLE SERVICE 210
CONFIG AND METADATA MANAGER 222
MONITOR AND WORKLOAD ANALYZER 224
DATA STORAGE 206
SQL COMPILER 212
SQL OPTIMIZER 214
SQL EXECUTOR 216
DATA STORAGE 226
QUERY SCHEDULER AND COORDINATOR 218
VIRTUAL WAREHOUSE MANAGER 220
TRANSACTION MANAGEMENT AND ACCESS CONTROL MODULE 228

FIG. 2

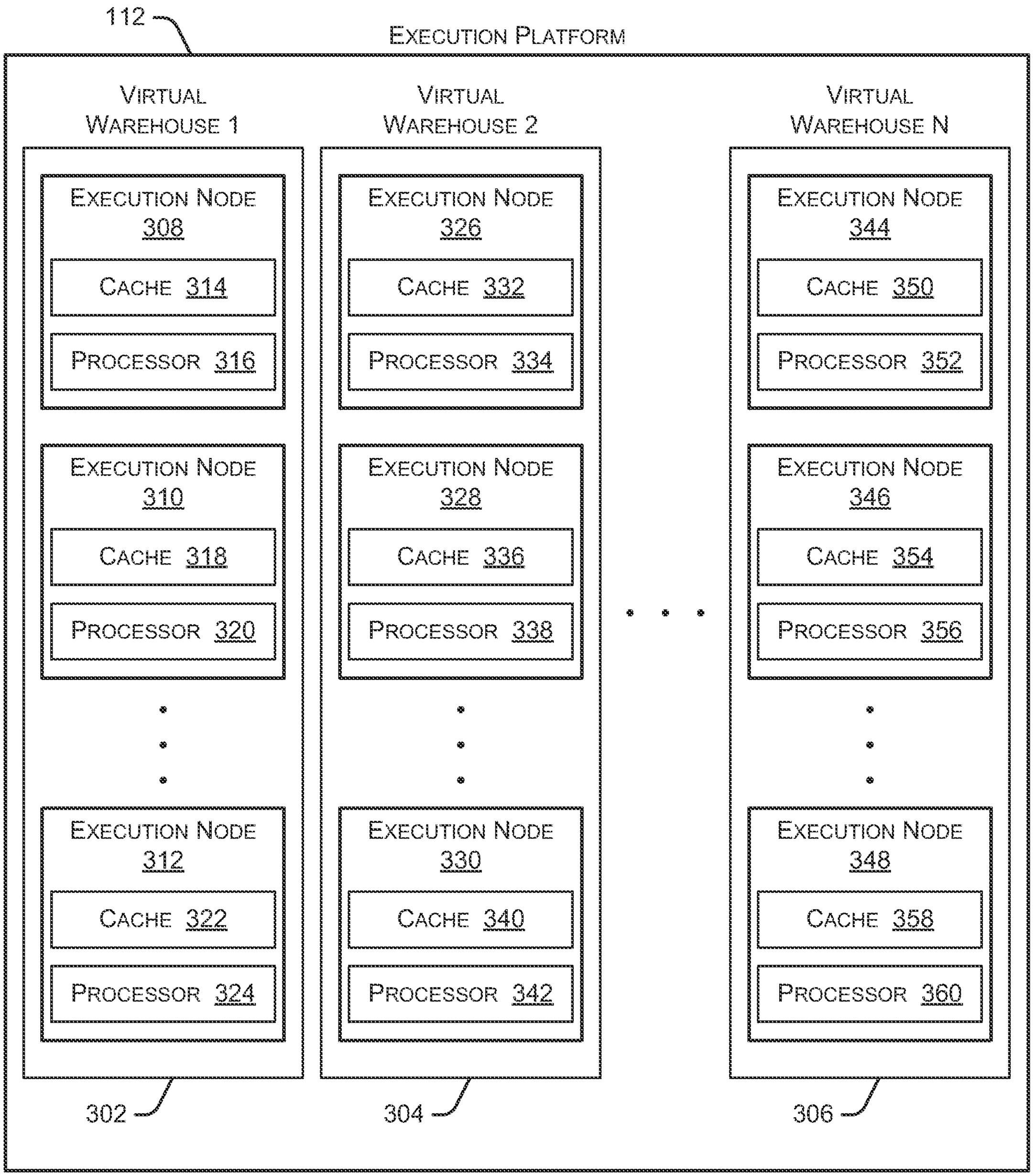

112
EXECUTION PLATFORM
VIRTUAL WAREHOUSE 1
VIRTUAL WAREHOUSE 2
VIRTUAL WAREHOUSE N
EXECUTION NODE 308
CACHE 314
PROCESSOR 316
EXECUTION NODE 310
CACHE 318
PROCESSOR 320
EXECUTION NODE 312
CACHE 322
PROCESSOR 324
EXECUTION NODE 326
CACHE 332
PROCESSOR 334
EXECUTION NODE 328
CACHE 336
PROCESSOR 338
EXECUTION NODE 330
CACHE 340
PROCESSOR 342
EXECUTION NODE 344
CACHE 350
PROCESSOR 352
EXECUTION NODE 346
CACHE 354
PROCESSOR 356
EXECUTION NODE 348
CACHE 358
PROCESSOR 360
302
304
306

FIG. 3

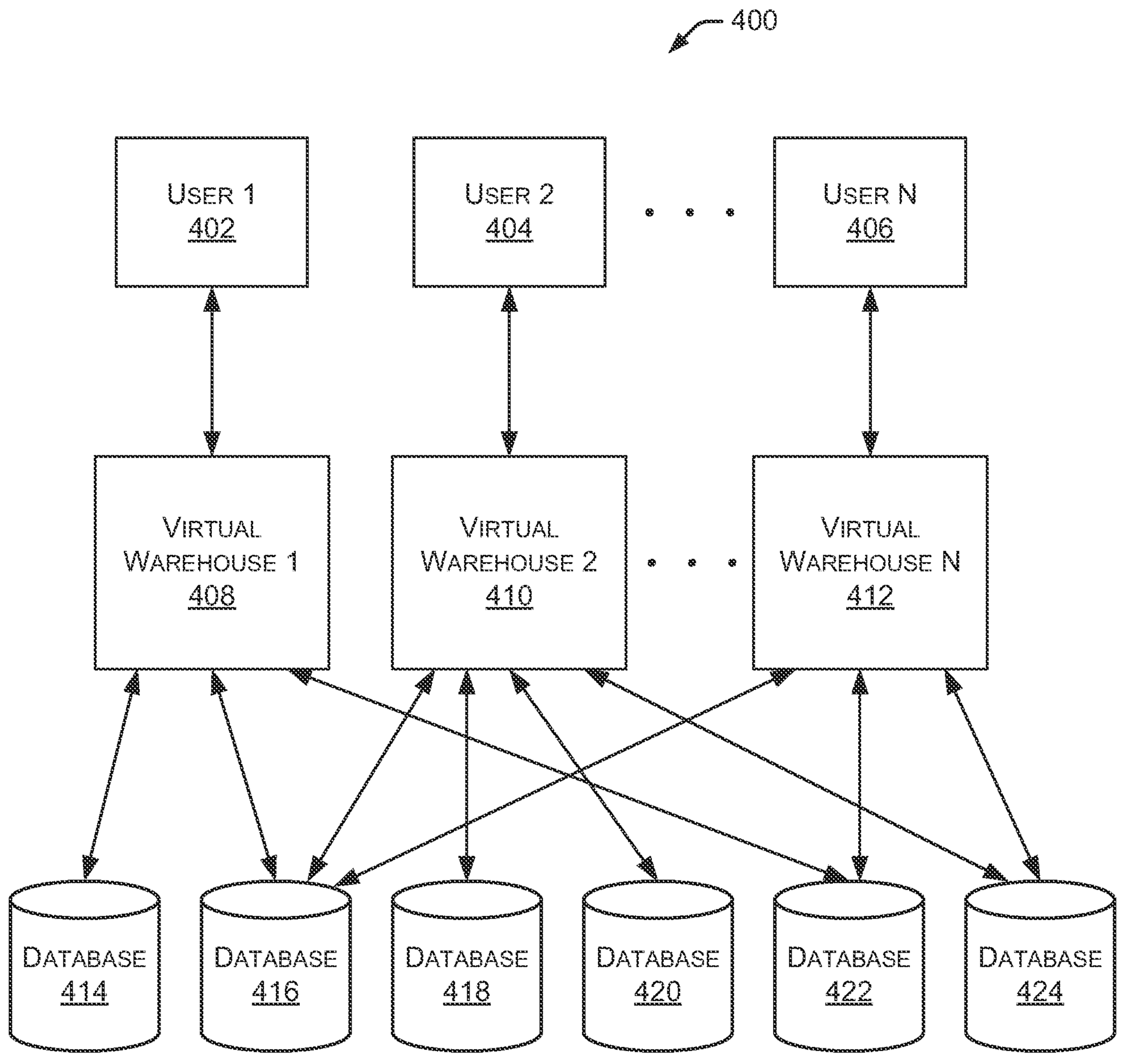

400
USER 1
402
USER 2
404
USER N
406
VIRTUAL WAREHOUSE 1
408
VIRTUAL WAREHOUSE 2
410
VIRTUAL WAREHOUSE N
412
DATABASE
414
DATABASE
416
DATABASE
418
DATABASE
420
DATABASE
422
DATABASE
424

FIG. 4

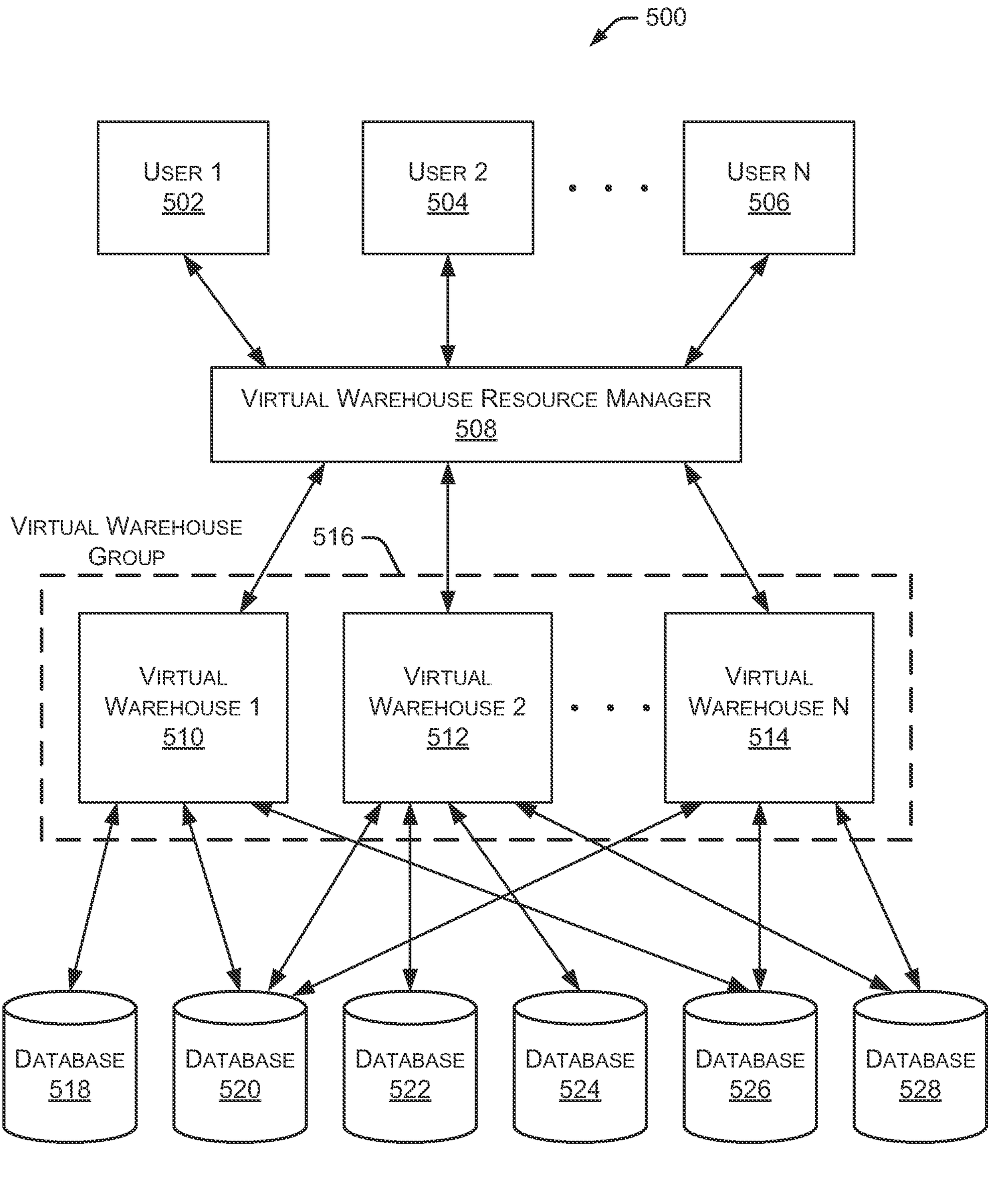

500
USER 1
502
USER 2
504
USER N
506
VIRTUAL WAREHOUSE RESOURCE MANAGER
508
VIRTUAL WAREHOUSE GROUP
516
VIRTUAL WAREHOUSE 1
510
VIRTUAL WAREHOUSE 2
512
VIRTUAL WAREHOUSE N
514
DATABASE
518
DATABASE
520
DATABASE
522
DATABASE
524
DATABASE
526
DATABASE
528

FIG. 5

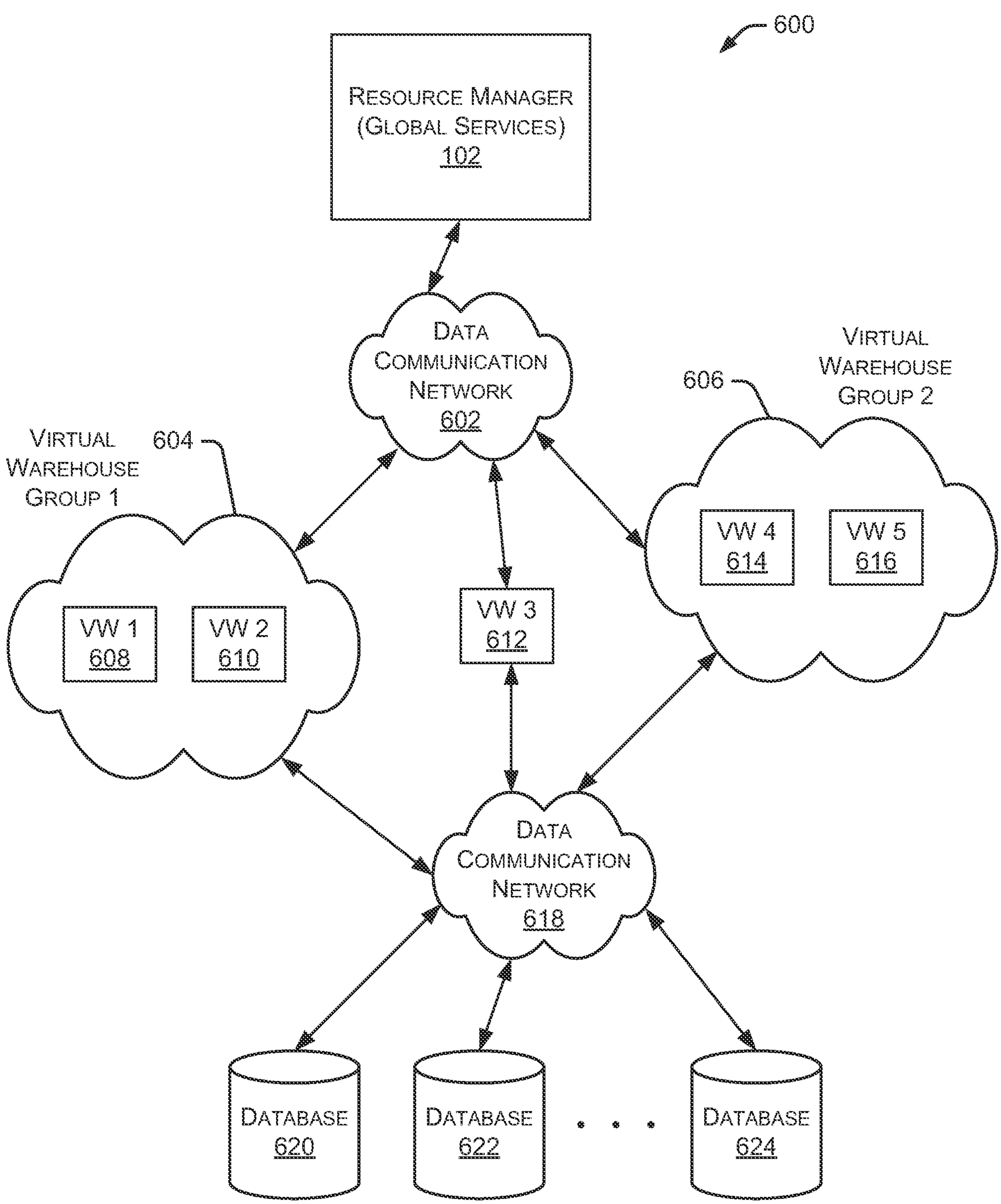

600
RESOURCE MANAGER
(GLOBAL SERVICES)
102
DATA
COMMUNICATION
NETWORK
602
VIRTUAL
WAREHOUSE
GROUP 1
604
VW 1
608
VW 2
610
VW 3
612
606
VIRTUAL
WAREHOUSE
GROUP 2
VW 4
614
VW 5
616
DATA
COMMUNICATION
NETWORK
618
DATABASE
620
DATABASE
622
DATABASE
624

FIG. 6

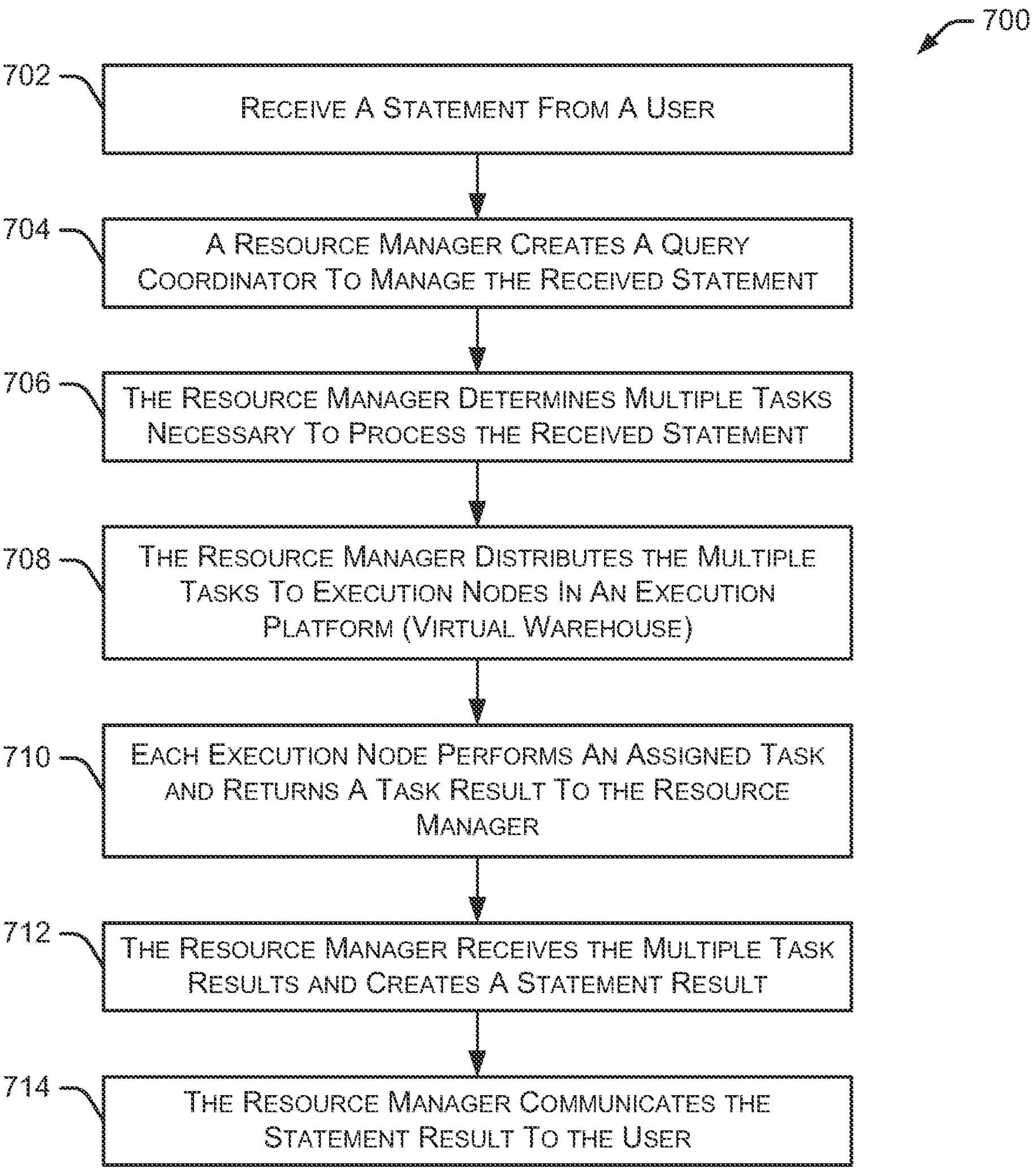

700
702
RECEIVE A STATEMENT FROM A USER
704
A RESOURCE MANAGER CREATES A QUERY COORDINATOR TO MANAGE THE RECEIVED STATEMENT
706
THE RESOURCE MANAGER DETERMINES MULTIPLE TASKS NECESSARY TO PROCESS THE RECEIVED STATEMENT
708
THE RESOURCE MANAGER DISTRIBUTES THE MULTIPLE TASKS TO EXECUTION NODES IN AN EXECUTION PLATFORM (VIRTUAL WAREHOUSE)
710
EACH EXECUTION NODE PERFORMS AN ASSIGNED TASK AND RETURNS A TASK RESULT TO THE RESOURCE MANAGER
712
THE RESOURCE MANAGER RECEIVES THE MULTIPLE TASK RESULTS AND CREATES A STATEMENT RESULT
714
THE RESOURCE MANAGER COMMUNICATES THE STATEMENT RESULT TO THE USER

FIG. 7

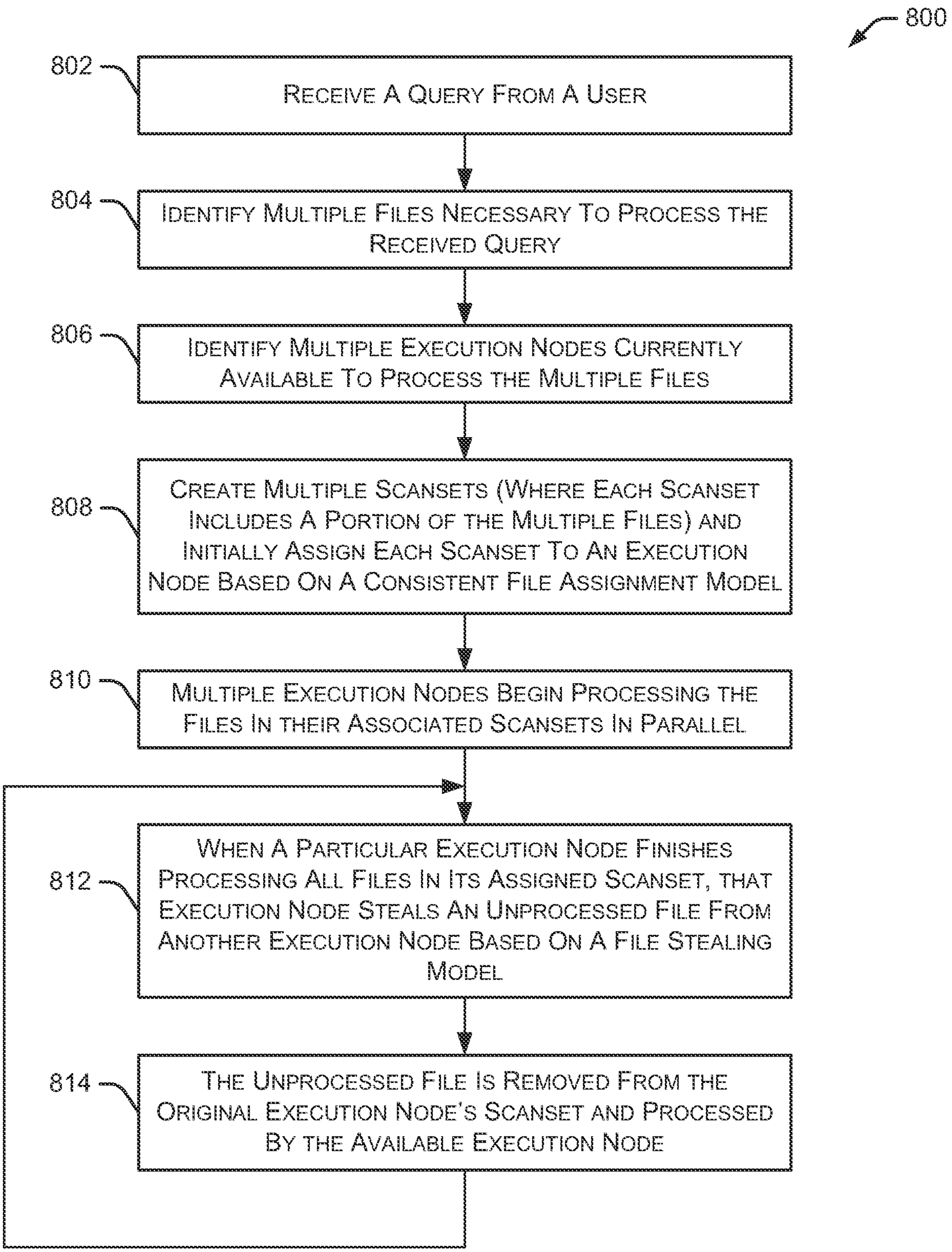

800
802
RECEIVE A QUERY FROM A USER
804
IDENTIFY MULTIPLE FILES NECESSARY TO PROCESS THE RECEIVED QUERY
806
IDENTIFY MULTIPLE EXECUTION NODES CURRENTLY AVAILABLE TO PROCESS THE MULTIPLE FILES
808
CREATE MULTIPLE SCANSETS (WHERE EACH SCANSET INCLUDES A PORTION OF THE MULTIPLE FILES) AND INITIALLY ASSIGN EACH SCANSET TO AN EXECUTION NODE BASED ON A CONSISTENT FILE ASSIGNMENT MODEL
810
MULTIPLE EXECUTION NODES BEGIN PROCESSING THE FILES IN THEIR ASSOCIATED SCANSETS IN PARALLEL
812
WHEN A PARTICULAR EXECUTION NODE FINISHES PROCESSING ALL FILES IN ITS ASSIGNED SCANSET, THAT EXECUTION NODE STEALS AN UNPROCESSED FILE FROM ANOTHER EXECUTION NODE BASED ON A FILE STEALING MODEL
814
THE UNPROCESSED FILE IS REMOVED FROM THE ORIGINAL EXECUTION NODE'S SCANSET AND PROCESSED BY THE AVAILABLE EXECUTION NODE

FIG. 8

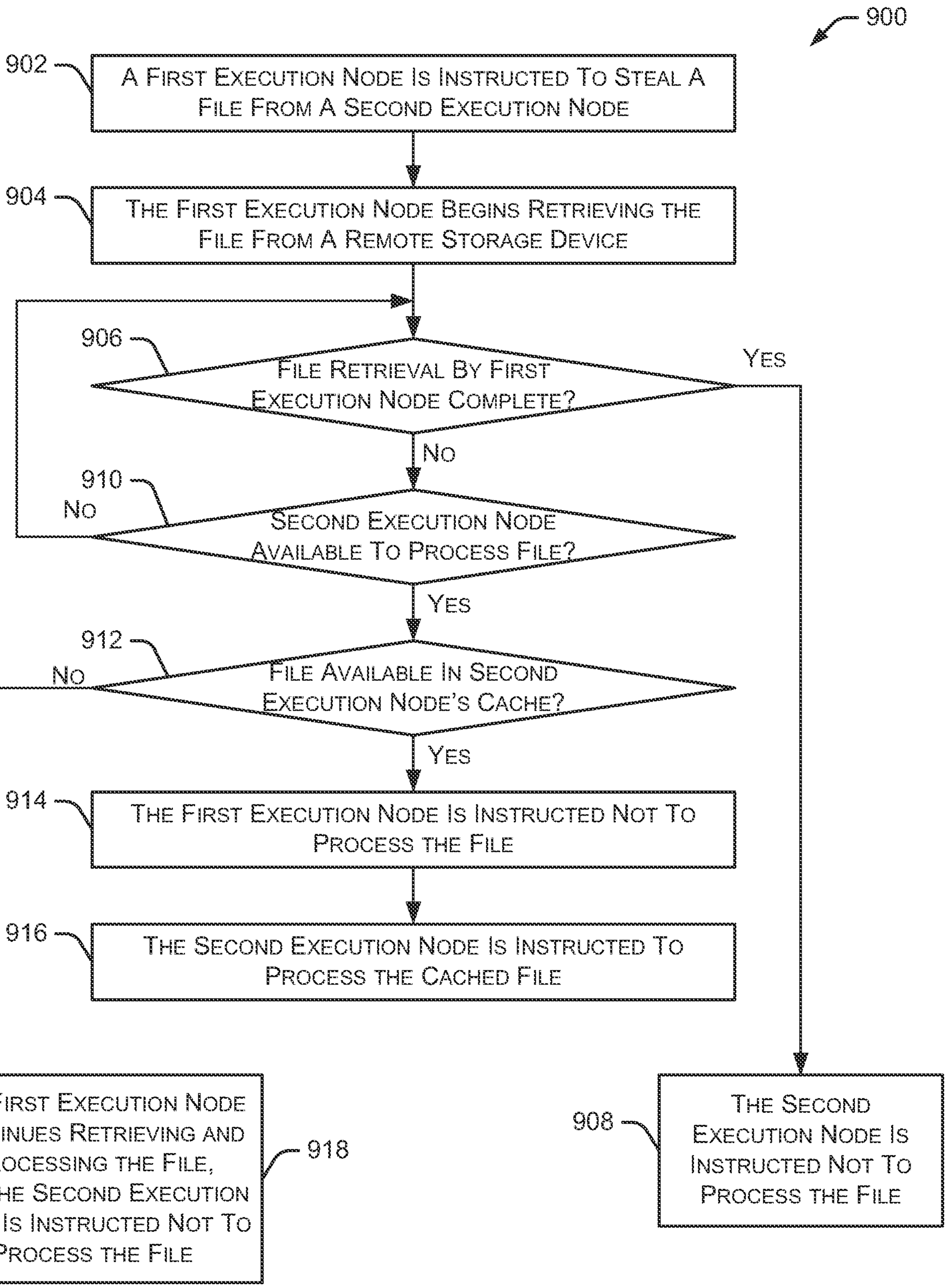

900
902
A First Execution Node Is Instructed To Steal A File From A Second Execution Node
904
The First Execution Node Begins Retrieving the File From A Remote Storage Device
906
File Retrieval By First Execution Node Complete?
Yes
No
910
No
Second Execution Node Available To Process File?
Yes
912
No
File Available In Second Execution Node's Cache?
Yes
914
The First Execution Node Is Instructed Not To Process the File
916
The Second Execution Node Is Instructed To Process the Cached File
The First Execution Node Continues Retrieving and Processing the File, and the Second Execution Node Is Instructed Not To Process the File
918
908
The Second Execution Node Is Instructed Not To Process the File

FIG. 9

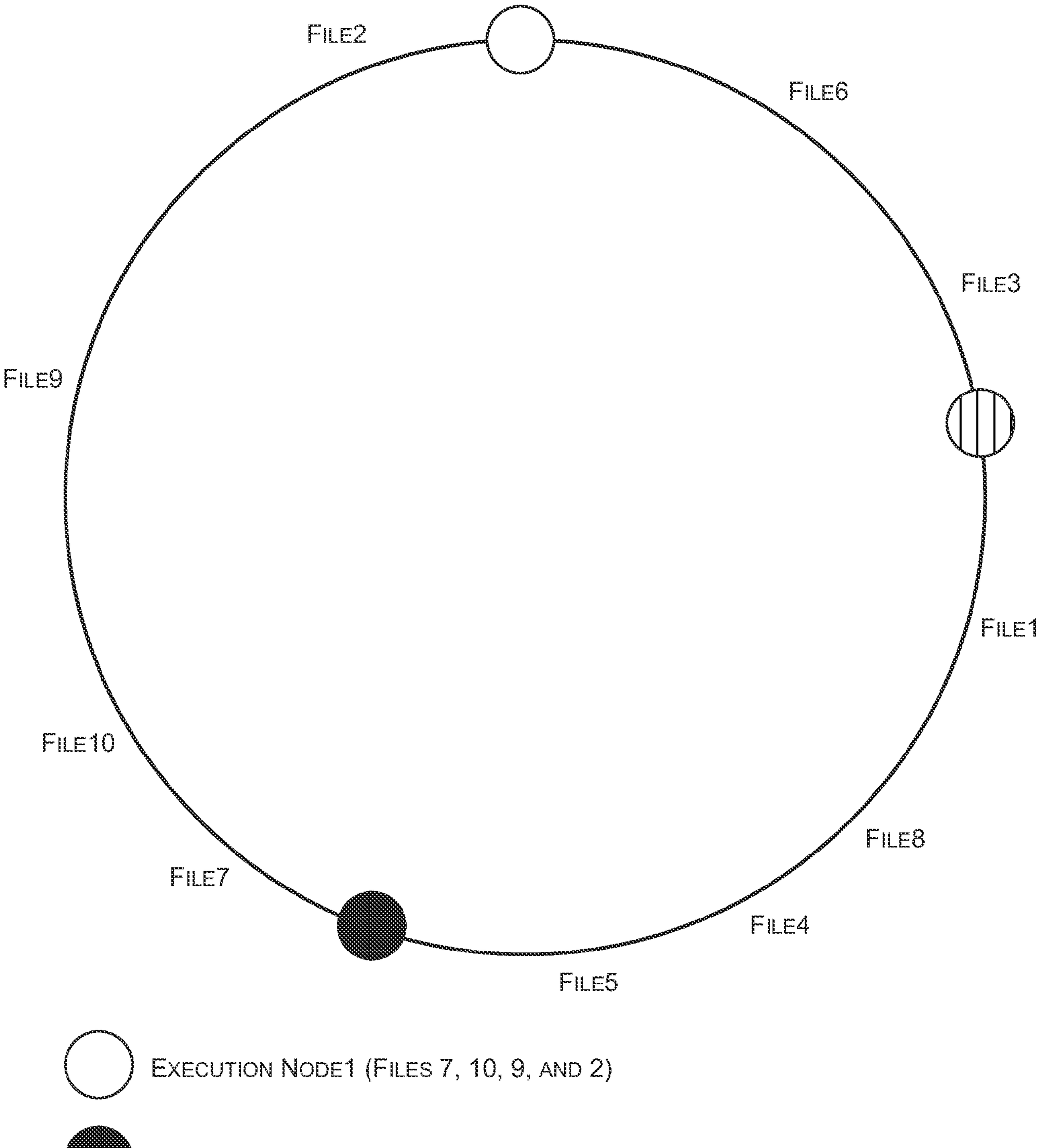

FILE2
FILE6
FILE3
FILE9
FILE1
FILE10
FILE8
FILE7
FILE4
FILE5
EXECUTION NODE1 (FILES 7, 10, 9, AND 2)
EXECUTION NODE2 (FILES 1, 8, 4, AND 5)
EXECUTION NODE3 (FILES 6 AND 3)

FIG. 10A

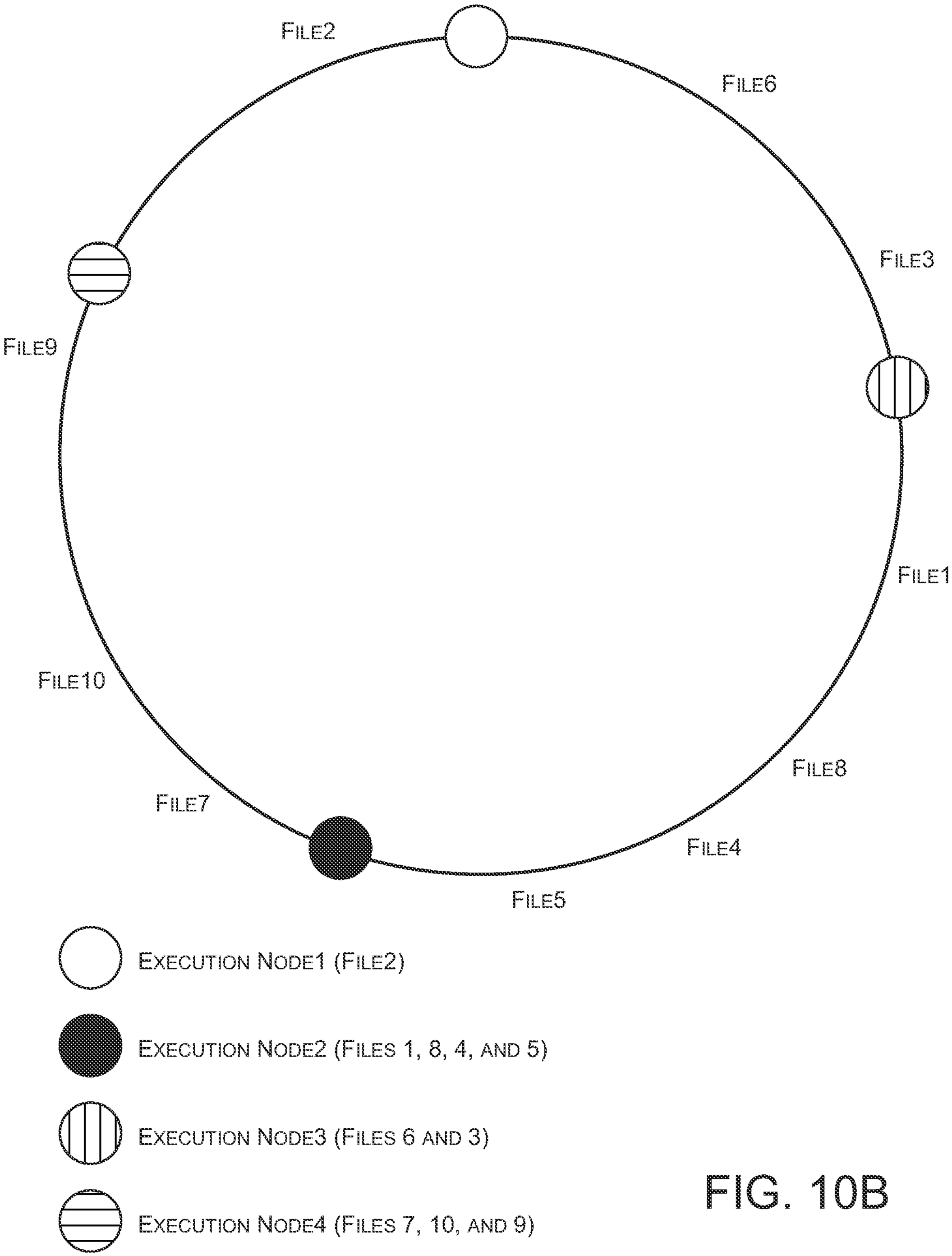

FILE2
FILE6
FILE3
FILE9
FILE1
FILE10
FILE8
FILE7
FILE4
FILE5
EXECUTION NODE1 (FILE2)
EXECUTION NODE2 (FILES 1, 8, 4, AND 5)
EXECUTION NODE3 (FILES 6 AND 3)
EXECUTION NODE4 (FILES 7, 10, AND 9)

FIG. 10B

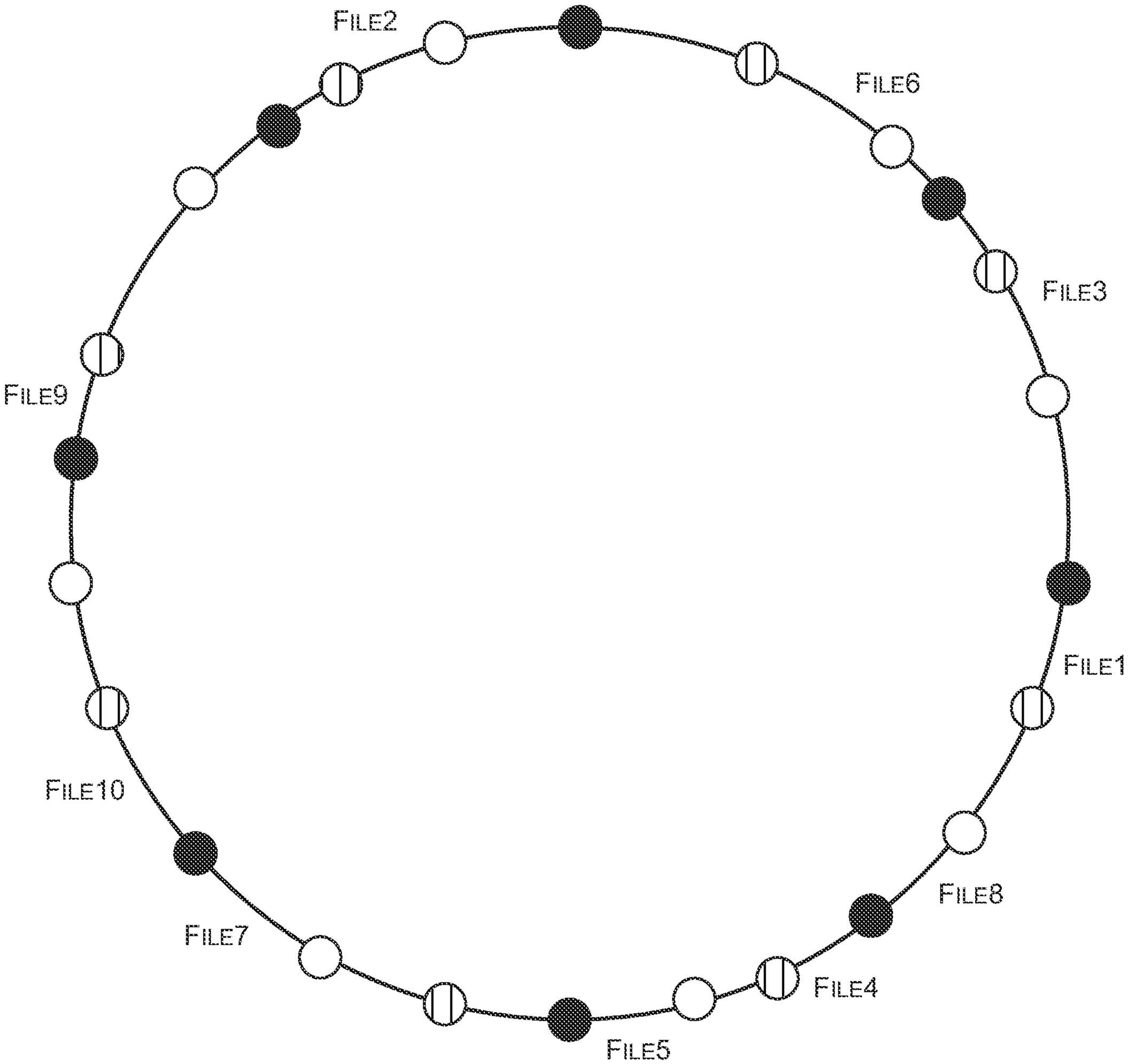

FILE2
FILE6
FILE9
FILE3
FILE1
FILE10
FILE7
FILE8
FILE4
FILE5
EXECUTION NODE1 (FILES 6, 3, AND 2)
EXECUTION NODE2 (FILES 8, 5, AND 7)
EXECUTION NODE3 (FILES 1, 4, 10, AND 9)

FIG. 10C

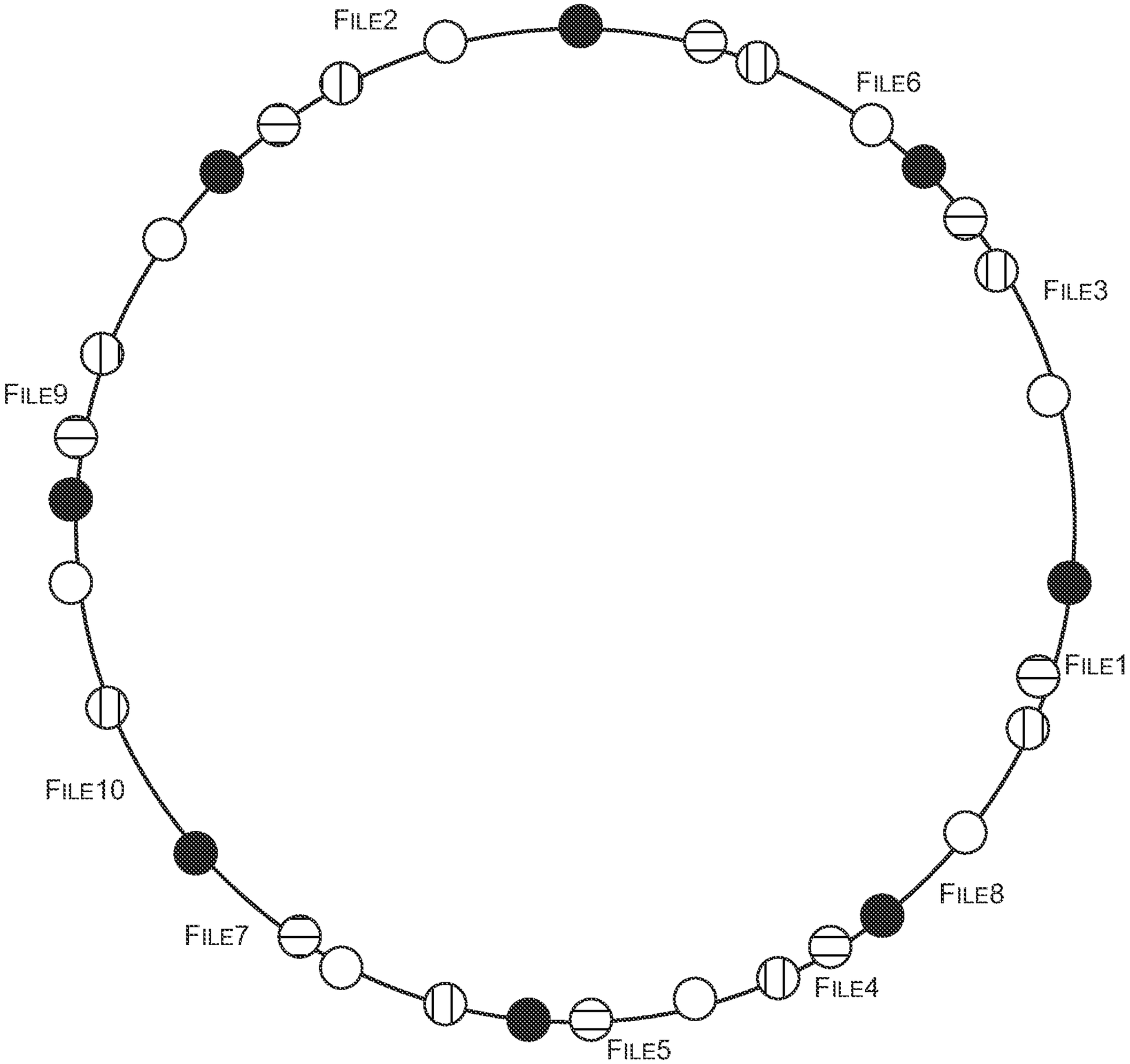

FILE2
FILE6
FILE9
FILE3
FILE1
FILE10
FILE7
FILE8
FILE4
FILE5
EXECUTION NODE1 (FILES 6, 3, AND 2)
EXECUTION NODE2 (FILES 8 AND 7)
EXECUTION NODE3 (FILES 4, 10, AND 9)
EXECUTION NODE4 (FILES 1 AND 5)

FIG. 10D

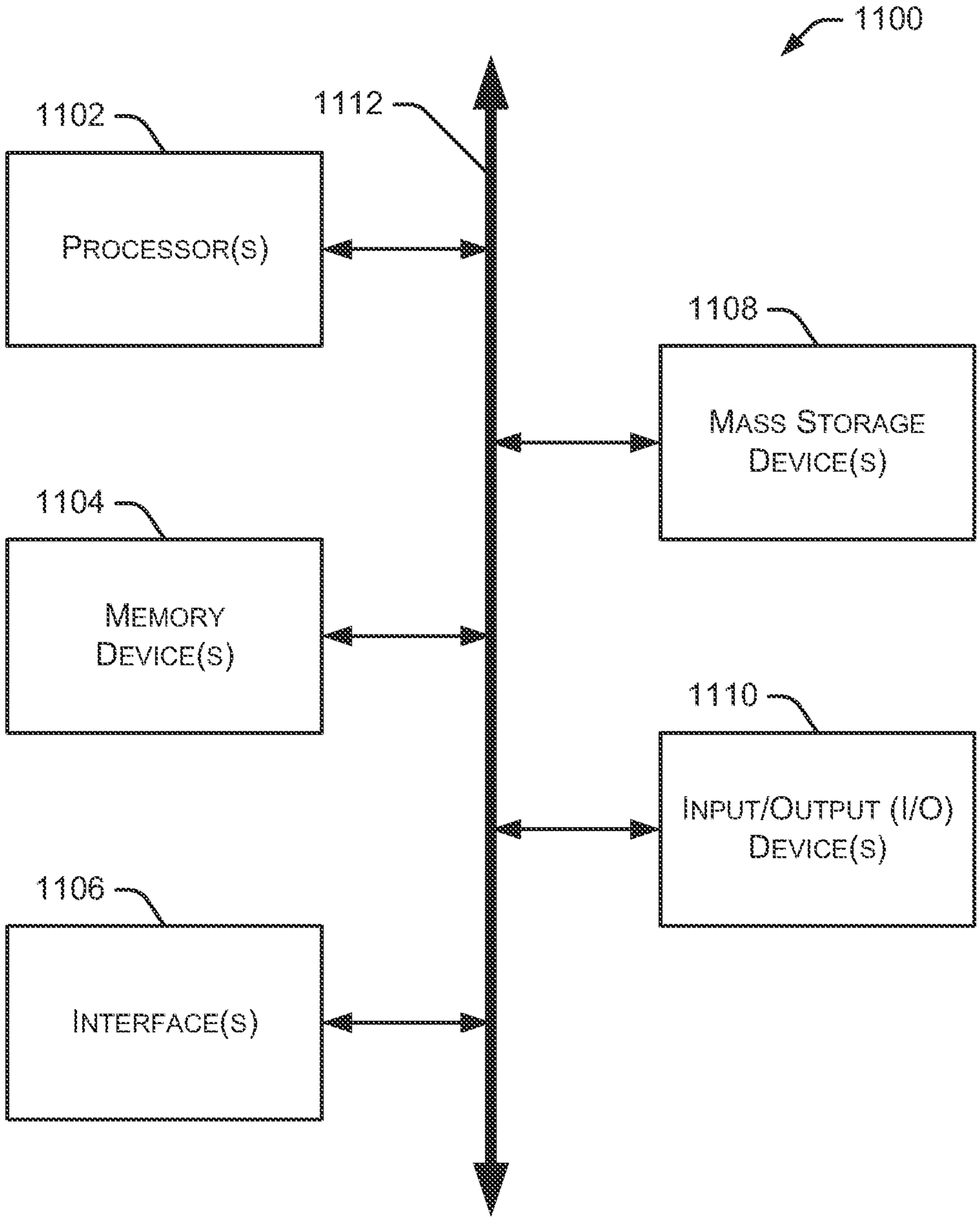

1100
1102
PROCESSOR(S)
1112
1108
MASS STORAGE DEVICE(S)
1104
MEMORY DEVICE(S)
1110
INPUT/OUTPUT (I/O) DEVICE(S)
1106
INTERFACE(S)

FIG. 11

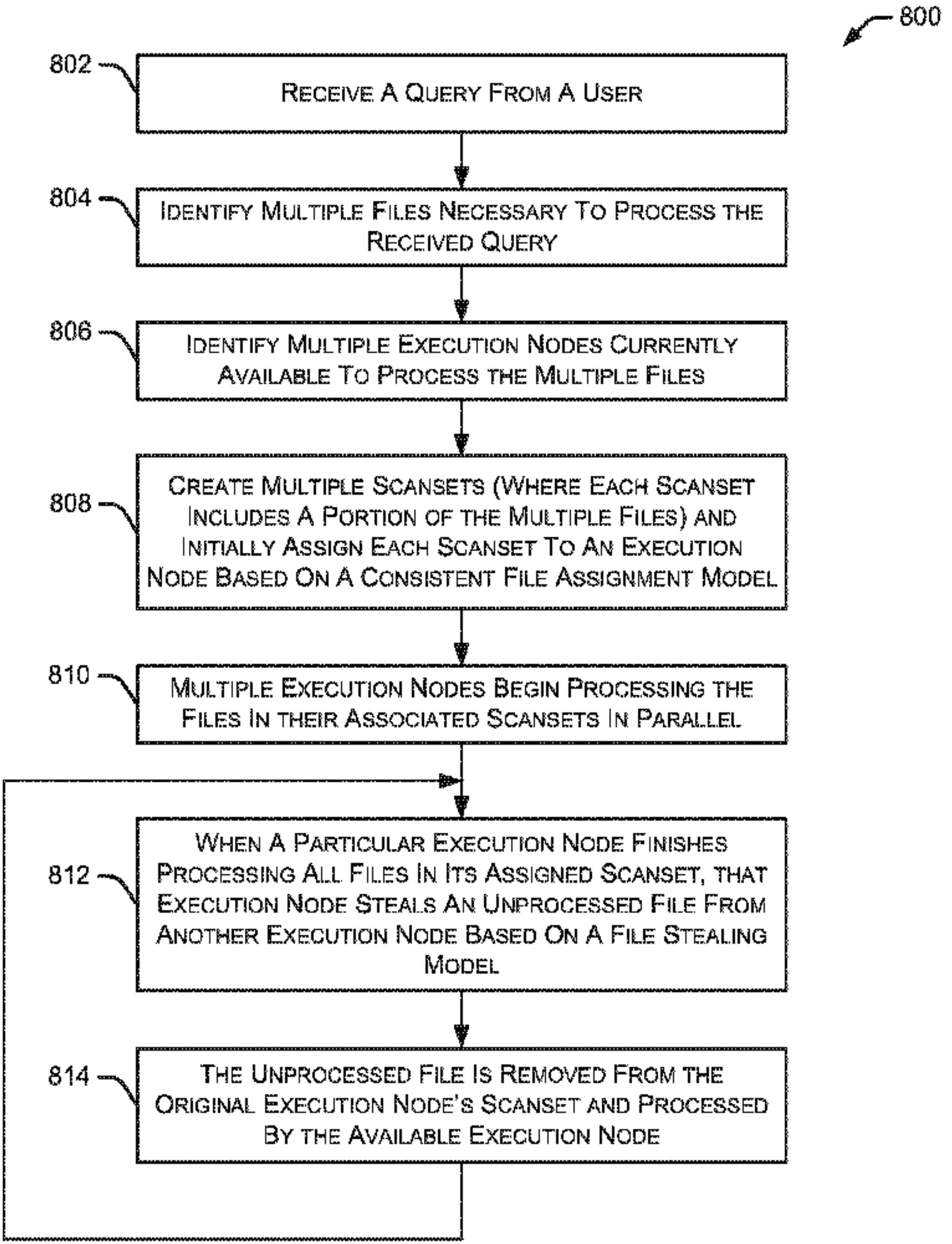

800
802
Receive A Query From A User
804
Identify Multiple Files Necessary To Process the Received Query
806
Identify Multiple Execution Nodes Currently Available To Process the Multiple Files
808
Create Multiple Scansets (Where Each Scanset Includes A Portion of the Multiple Files) and Initially Assign Each Scanset To An Execution Node Based On A Consistent File Assignment Model
810
Multiple Execution Nodes Begin Processing the Files In their Associated Scansets In Parallel
812
When A Particular Execution Node Finishes Processing All Files In Its Assigned Scanset, that Execution Node Steals An Unprocessed File From Another Execution Node Based On A File Stealing Model
814
The Unprocessed File Is Removed From the Original Execution Node's Scanset and Processed By the Available Execution Node

FIG. 8