



(86) **Date de dépôt PCT/PCT Filing Date:** 2014/12/02
(87) **Date publication PCT/PCT Publication Date:** 2015/07/02
(85) **Entrée phase nationale/National Entry:** 2016/06/01
(86) **N° demande PCT/PCT Application No.:** US 2014/068002
(87) **N° publication PCT/PCT Publication No.:** 2015/099961
(30) **Priorités/Priorities:** 2013/12/02 (US61/910,864);
2013/12/02 (US61/910,873); 2013/12/02 (US61/910,850);
2013/12/02 (US61/910,860); 2013/12/02 (US61/910,867);
2013/12/02 (US61/910,834); 2013/12/02 (US61/910,856)

(51) **Cl.Int./Int.Cl.** *G06F 17/30* (2006.01)
(71) **Demandeur/Applicant:**
QBASE, LLC, US
(72) **Inventeurs/Inventors:**
LIGHTNER, SCOTT, US;
WECKESSER, FRANZ, US;
ZIMMERMAN, BRYAN, US
(74) **Agent:** DENTONS CANADA LLP

(54) **Titre : SYSTEMES ET PROCEDES PERMETTANT D'HEBERGER UNE BASE DE DONNEES EN MEMOIRE**
(54) **Title: SYSTEMS AND METHODS FOR HOSTING AN IN-MEMORY DATABASE**

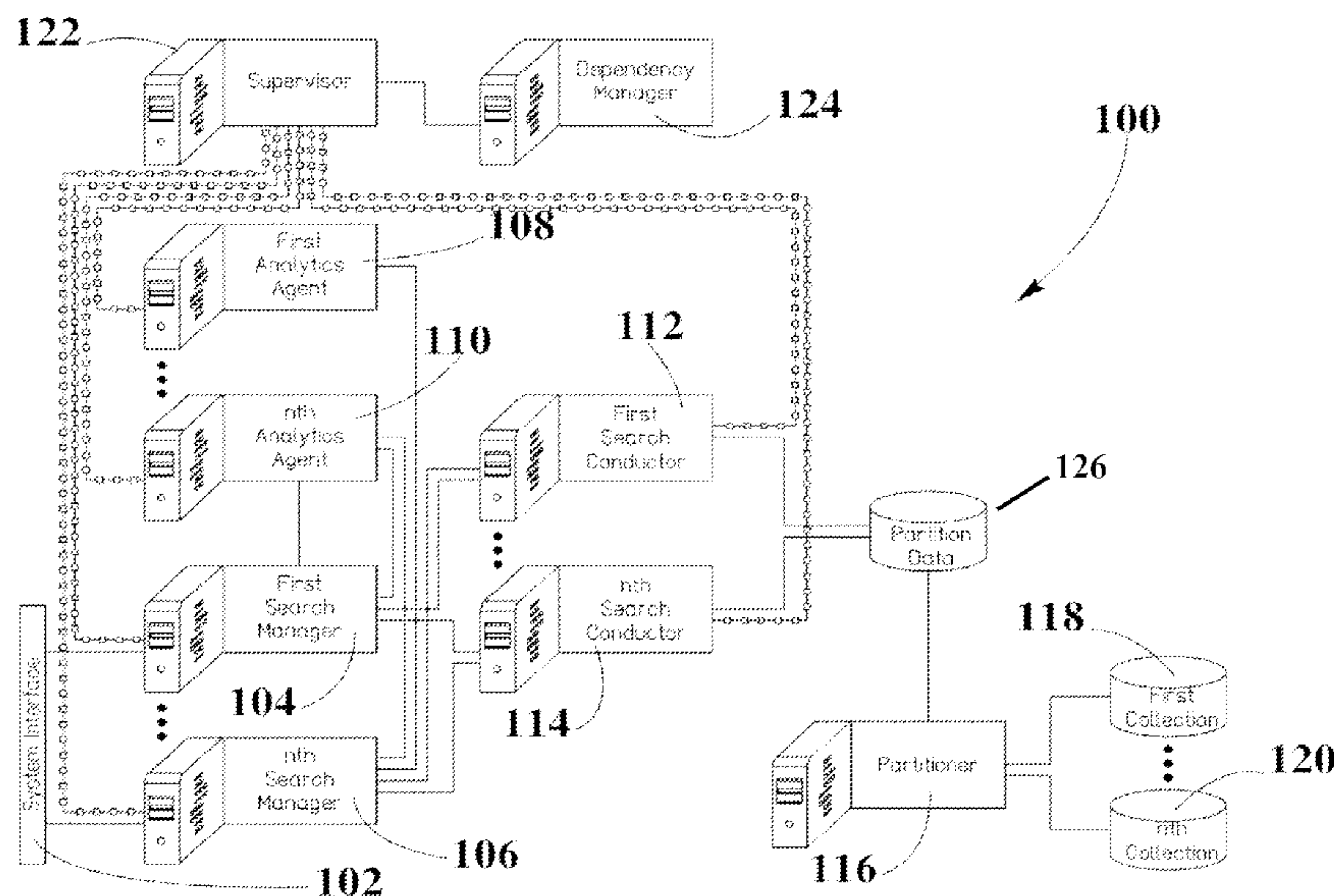


FIG. 1

(57) **Abrégé/Abstract:**

An in-memory database system and method for administrating a distributed in-memory database, comprising one or more nodes having modules configured to store and distribute database partitions of collections partitioned by a partitioner associated with a search conductor. Database collections are partitioned according to a schema. Partitions, collections, and records, are updated and removed when requested by a system interface, according to the schema. Supervisors determine a node status based on a heartbeat signal received from each node. Users can send queries through a system interface to search managers. Search managers apply a field processing technique, forward the search query to search conductors, and return a set of result records to the analytics agents. Analytics agents perform analytics processing on a candidate results records from a search manager. The search conductors comprising partitioners associated with a collection, search and score the records in a partition, then return candidate result records.



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

(19) World Intellectual Property

Organization

International Bureau

(43) International Publication Date

2 July 2015 (02.07.2015)



(10) International Publication Number

WO 2015/099961 A1

(51) International Patent Classification:

G06F 17/30 (2006.01)

(21) International Application Number:

PCT/US2014/068002

(22) International Filing Date:

2 December 2014 (02.12.2014)

(25) Filing Language:

English

(26) Publication Language:

English

(30) Priority Data:

61/910,850	2 December 2013 (02.12.2013)	US
61/910,856	2 December 2013 (02.12.2013)	US
61/910,834	2 December 2013 (02.12.2013)	US
61/910,867	2 December 2013 (02.12.2013)	US
61/910,860	2 December 2013 (02.12.2013)	US
61/910,873	2 December 2013 (02.12.2013)	US
61/910,864	2 December 2013 (02.12.2013)	US

(71) Applicant: QBASE, LLC [US/US]; 12018 Sunrise Valley Drive, Suite 300, Reston, VA 20191 (US).

(72) Inventors: LIGHTNER, Scott; 22596 Redhill Manor Court, Leesburg, VA 20175 (US). WECKESSER, Franz; 3942 E Centerville Road, Spring Valley, OH 45370 (US). ZIMMERMAN, Bryan; 3086 Catlett Road, Catlett, VA 20119 (US).

(74) Agent: SOPHIR, Eric; Dentons Us LLP, P.O. Box 061080, Wacker Drive Station, Willis Tower, Chicago, IL 60606 (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))

(54) Title: SYSTEMS AND METHODS FOR HOSTING AN IN-MEMORY DATABASE

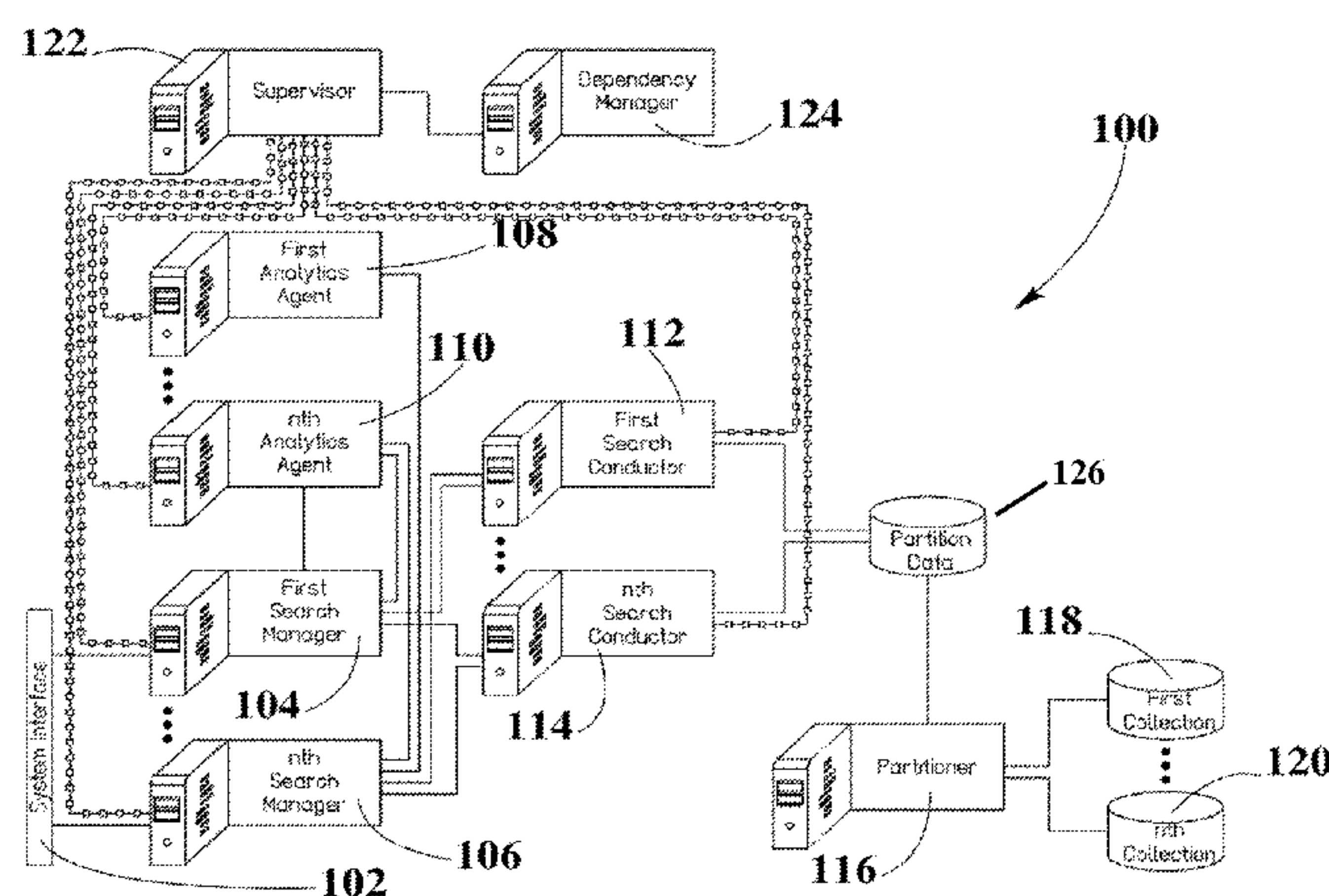


FIG. 1

(57) Abstract: An in-memory database system and method for administrating a distributed in-memory database, comprising one or more nodes having modules configured to store and distribute database partitions of collections partitioned by a partitioner associated with a search conductor. Database collections are partitioned according to a schema. Partitions, collections, and records, are updated and removed when requested by a system interface, according to the schema. Supervisors determine a node status based on a heartbeat signal received from each node. Users can send queries through a system interface to search managers. Search managers apply a field processing technique, forward the search query to search conductors, and return a set of result records to the analytics agents. Analytics agents perform analytics processing on a candidate results records from a search manager. The search conductors comprising partitioners associated with a collection, search and score the records in a partition, then return candidate result records.

SYSTEMS AND METHODS FOR HOSTING AN IN-MEMORY DATABASE**TECHNICAL FIELD**

[0001] The present disclosure relates in general to in-memory databases, and more specifically to faceted searching and search suggestions within in-memory databases. The present disclosure relates in general to databases, and more specifically to in-memory databases. The present disclosure relates in general to in-memory databases, and more specifically to hardware configurations of use in in-memory databases. The present disclosure relates in general to databases architectures, and more particularly to a fault tolerant system architectures. The present disclosure relates in general to databases, and more particularly, to a dependency manager that may be used for in-memory databases. The present disclosure relates in general to in-memory databases, and more specifically to pluggable in-memory analytic modules. The present disclosure relates in general to in-memory databases, and more specifically to non-exclusionary searching within in-memory databases. The present disclosure relates in general to data compression and databases, and more specifically to methods of compression for use in in-memory databases as well as document databases.

BACKGROUND

[0002] Faceted searching provides users with an incremental search and browse experience that lets them begin with a keyword search and go through the search results in an organized and simple way. Faceted searching, in many occasions, is used to serve up maps of the search results that may provide useful insights into the organization and content of these search results. Faceted navigation also allows users to systematically narrow down the search results in a fairly simple manner. Due to its many advantages, faceted search and navigation is being deployed rapidly across a wide variety of contexts and platforms. Unfortunately, the

way facet engines work is too slow and very memory intensive, which prevent these types of search engines from performing better and scaling.

[0003] Therefore, there is still a need of accurate, fast and scalable faceted search engines.

[0004] Computers are powerful tools of use in storing and providing access to vast amounts of information, while databases are a common mechanism for storing information on computer systems while providing easy access to users. Typically, a database is an organized collection of information stored as “records” having “fields” of information (e.g., a restaurant database may have a record for each restaurant in a region, where each record contains fields describing characteristics of the restaurant, such as name, address, type of cuisine, and the like).

[0005] In operation, a database management system frequently needs to retrieve data from or persist data to storage devices such as disks. Unfortunately, access to such storage devices can be somewhat slow. To speed up access to data, databases typically employ a “cache” or “buffer cache” which is a section of relatively faster memory (e.g., random access memory (RAM)) allocated to store recently used data objects. Memory is typically provided on semiconductor or other electrical storage media and is coupled to a CPU (central processing unit) via a fast data bus which enables data maintained in memory to be accessed more rapidly than data stored on disks.

[0006] One approach that may be taken when attempting to solve this problem is to store all the information in the database in memory, however as memory provided on computer systems has a limited size there are a number of obstacles that must be faced when attempting to handle databases of a larger scale.

[0007] As such, there is a continuing need for improved methods of storing and retrieving data at high speeds at a large scale.

[0008] Computers are powerful tools of use in storing and providing access to vast amounts of information, while databases are a common mechanism for storing information on computer systems while providing easy access to users. Typically, a database is an organized collection of information stored as “records” having “fields” of information. (e.g., a restaurant database may have a record for each restaurant in a region, where each record contains fields describing characteristics of the restaurant, such as name, address, type of cuisine, and the like).

[0009] In operation, a database management system frequently needs to retrieve data from or persist data to storage devices such as disks. Unfortunately, access to such storage devices can be somewhat slow. To speed up access to data, databases typically employ a “cache” or “buffer cache” which is a section of relatively faster memory (e.g., random access memory (RAM)) allocated to store recently used data objects. Memory is typically provided on semiconductor or other electrical storage media and is coupled to a CPU (central processing unit) via a fast data bus which enables data maintained in memory to be accessed more rapidly than data stored on disks.

[0010] One approach that may be taken when attempting to solve this problem is to store all the information in the database in memory, however as memory provided on computer systems has a limited size there are a number of obstacles that must be faced when attempting to handle databases of a larger scale. Some of these obstacles may include determining the technologies required to operate the database, including the networking needed, the hardware required for different nodes, and others.

[0011] As such, there is a continuing need for improved methods of storing and retrieving data at high speeds at a large scale.

[0012] Computers are powerful tools of use in storing and providing access to vast amounts of information. Databases are a common mechanism for storing information on computer systems while providing easy access to users. Typically, a database is an organized collection of information stored as “records” having “fields” of information. (e.g., a restaurant database may have a record for each restaurant in a region, where each record contains fields describing characteristics of the restaurant, such as name, address, type of cuisine, and the like).

[0013] Often, databases may use clusters of computers in order to be able to store and access large amounts of data. This may require that the state of the computer clusters be managed. One approach that may be taken when attempting to solve this problem is to employ a team of professionals that may have access to the tools necessary to maintain the system either on-site or remotely. However, there may be a number of obstacles that must be faced when attempting to use such a method with databases of a larger scale.

[0014] As such, there is a continuing need for improved methods of fault tolerance and high availability at a large scale.

[0015] Package management systems may be designed to save organizations time and money through remote administration and software distribution technology that may eliminate the need for manual installation and updates for any suitable component, such as, software, operating system component, application program, support library, application data, general documentation, and other data, from a system or process. One conventional approach in the art related to package management system may be the Red Hat package manager

(RPM). Package managers may present a uniform way to install and/or update software programs and associated components.

[0016] To install a set of software or data packages, a package manager may order the packages and its dependent packages in topological order onto a graph. Subsequently, the package manager may collect the packages at the bottom of the graph and install these packages first. Finally, the package manager may move up the graph and install the next set of packages.

[0017] However, the conventional approach in the art related to database management systems refers that some package managers may only keep the software configuration in the system, but may not support metadata or primary data collection dependencies. In a database, particularly an in-memory database or other distributed storage architectures, deployment focuses as much on data as software, and therefore maintaining dependency trees required for data deployment are essential.

[0018] Conventional technologies may automate deployment, installation, and configuration of software components and associated dependencies, across a cluster of one or more computers in a conventional distributed computing architectures. What is needed is a solution to automate the deployment, installation, and configuration of data, metadata and software of a primary datastore of a distributed database, in a distributed computing architecture, such as in-memory databases and other distributed data platforms. Moreover, because conventional solutions focus on deploying a static set of services and data, conventional systems lack the ability to detect service or data failures and then automatically recover from those failures by moving a package of data, metadata and software to other available nodes in the distributed system.

[0019] For the aforementioned reasons, there is a need for an improved package management application to guarantee/keep a successful execution of the system configuration and dependencies into a data management system.

[0020] Traditional databases allow users to find pieces of information that are relevant to an entity, and while millions or billions of records may describe that entity the records are generally not linked together without formal foreign key relationships. Large collections of records exist with informal, unreliable or non-existent foreign key relationships. This makes it extremely challenging to bring together all of the records relevant to a single entity.

[0021] The traditional approach is to pre-link all of the data within such a collection such that finding one record will lead you directly to the collection of records. This traditional approach has two distinct problems.

[0022] First, pre-linking a large collection of records is an intensive process taking considerable time. This imposes a significant lag on the time it can take to integrate new records into the linked collection adversely affecting the timeliness of the data in that collection.

[0023] Second, pre-linking the data by definition is restricted to the model used to perform that pre-linking, drastically reducing the ability of a user of the system to vary the parameters of how strongly or weakly records are linked. Pre-linking is also limited to the data available at the time of the pre-linking step.

[0024] Another approach is to avoid any pre-linking of the data, but rather to link in real time, or "link-on-the-fly," in response to a user query. This approach allows new records to immediately participate in the collection avoiding any issues of timeliness. It also allows a

wide variety of models to be applied to perform the linking using varying algorithms and parameters in the linking process. The traditional disadvantage to this approach has been the ability to run that data intensive query and achieve acceptable interactive response times. This can be overcome by placing the collection in an in-memory database with embedded analytics.

[0025] There is therefore a need in the art for flexible database architecture capable of supporting multiple customized analytics modules, designed to process data in real time without having to change how the data is managed, prepared and stored within the system.

[0026] Accessing data may be simpler, more accurate and much faster from structured and semi-structured data than non-structured data. When performing a search using structured and semi-structured data by indicating key data fields it is possible to get very accurate results in a very short time frame, but it is also possible that many records relevant to the query may be excluded from the results list. This may happen because the records may be stored in collections with different schemata or the records may have some missing or null fields that correspond to some of the fields specified in the query.

[0027] Therefore, there is a need for search methods with improved recall capabilities that allow mixing and matching records with different schemata.

[0028] Computers are powerful tools of use in storing and providing access to vast amounts of information, while databases are a common mechanism for storing information on computer systems while providing easy access to users. Typically, a database is an organized collection of information stored as “records” having “fields” of information. (e.g., a restaurant database may have a record for each restaurant in a region, where each record contains fields describing characteristics of the restaurant, such as name, address, type of cuisine, and the like).

[0029] Often, databases may use clusters of computers in order to be able to store and access large amounts of data. This may require a large amount of information storage space. Often, compression may be used to reduce the amount of storage space necessary to host the information, but it may increase the computational load significantly as many common compression methods require the entire record or many records to be decompressed every time they are accessed.

[0030] As such, there is a continuing need for improved methods of storing and retrieving data at high speeds at a large scale.

SUMMARY

[0031] Disclosed herein is a system architecture hosting an in-memory database, which may include any suitable combination of computing devices and software modules for storing, manipulating, and retrieving data records of the in-memory database that is hosted within the distributed computing architecture of the system. Software modules executed by computing hardware of the system may include a system interface, a search manager, an analytics agent, a search conductor, a partitioner, collections of data, a supervisor, a dependency manager; any suitable combination of these software modules may be found in the system architecture hosting the in-memory database.

[0032] Nodes executing software modules may compress data stored in the records to make in-memory storage, queries, and retrieval feasible for massive data sets. Compression and decompression may be performed at nearly any level of the database (e.g., database level, collection level, record level, field level). Nodes executing software modules may provide support for storing complex data structures, such as JavaScript Object Notation (JSON) in the distributed in-memory database. Embodiments of an in-memory database system may be fault-tolerant due to the distributed architecture of system components and the various

hardware and software modules of the system that are capable of monitoring and restoring faulty services. Fault-tolerance may include system component redundancy, and automated recovery procedures for system components, among other techniques. The in memory database may effectively and efficiently query data by scoring data using scoring methods. Search results may be ranked according to the scoring methods used to score the data, thereby allowing users and/or nodes executing queries to utilize data in ways that are more tailored and contextually relevant from one query to the next. Nodes executing analytics agents may perform various advanced analytics on records stored in the in-memory database image of data. In some cases, analytics may be performed on the records retrieved with a set of search query results by search conductors.

[0033] In one embodiment, a computing system hosting an in-memory database, the system comprising: a partitioner node comprising a processor configured to, in response to receiving a collection of one or more records of a database, determine whether to compress the collection based on a machine-readable schema file associated with the collection, logically partition the collection into one or more partitions according to the schema file, and distribute the one or more partitions to one or more storage nodes according to the schema file; a storage node comprising non-transitory machine-readable main memory storing a partition received from the partitioner associated with the storage node; a search manager node comprising a processor receiving a search query from a client device of the system, and transmitting the search queries as search conductor queries to one or more search conductors in response to receive the search query from the client device, wherein the search query is a machine-readable computer file containing parameters associated with one or more records satisfying the search query; a search conductor node associated with one or more partitioners and comprising a processor configured to, in response to receiving a search conductor query from the search manager node: query a set of one or more partitions indicated by the search

conductor query, identify one or more candidate records stored in the set of queried partitions, calculate a first score for each respective candidate record using a scoring algorithm, and transmit to the search manager a set of one or more query results containing one or more candidate records satisfying a threshold value; and an analytics agent node comprising a processor configured to automatically generate a machine-readable computer file containing a set of one or more data linkages for the set of query results, responsive to identifying in the set of query results received from the search manager node a data linkage correlating two or more records, wherein the data linkage correlates data contained in a first record associated with data contained in a second record.

[0034] In another embodiment, a computer implemented method comprises receiving, by a search manager computer of a system hosting an in-memory database, binary data representing a search query containing parameters querying the database, wherein the system comprises one or more storage nodes comprising main memory storing one or more collections of the database, wherein each collection contains one or more records, transmitting, by the computer, the search query to one or more search conductor nodes according to the search query, wherein the search query indicates a set of one or more collections to be queried; transmitting, by the computer, to one or more analytics agent nodes a set of search results based on the search query responsive to receiving from the one or more search conductors the set of search results containing one or more records satisfying the search query, wherein each respective record of the set of search results is associated with a score based on a scoring algorithm in the search query; and responsive to the computer receiving a computer file containing a set of one or more data linkages from the one or more analytics agent nodes: updating, by the computer, the one or more records of the set of search results according to the set of one or more data linkages received from the analytics agent nodes.

[0035] In another embodiment, a computer-implemented method comprises receiving, by a computer, one or more collections from a search conductor according to a schema file, wherein each of the collections comprises a set of one or more records having one or more fields; partitioning, by the computer, each collection according to the schema; compressing, by the computer, the records in the partition according to the schema; and distributing, by the computer, each of the partitions to one or more associated search conductors to include each of the partitions in each collection corresponding to the partitioner associated with the search conductor.

[0036] Methods for faceted searching within clustered in-memory databases are disclosed. In some embodiments, the disclosed faceted searching methods and search engine may be used to generate search suggestions. The faceted search engine may be able to use literal or non-literal key construction algorithms for partial prefix fuzzy matching. The disclosed search engine may be capable of processing large amounts of unstructured data in real time to generate search suggestions. The system architecture of an in-memory database that may support the disclosed faceted search engine may include any suitable combination of modules and clusters; including one or more of a system interface, a search manager, an analytics agent, a search conductor, a partitioner, a collection, a supervisor, a dependency manager, or any suitable combination. In some embodiments, when a user starts typing in a suitable search window, the system may automatically generate one or more suggestions which may be derived from the fuzzy matches of the words that are being typed by the user on the search box. The system may score a query against the one or more records, where the system may score the match of one or more fields of the records and may then determine a score for the overall match of the records to the query. The system may determine whether the score is above a predefined acceptance threshold, where the threshold may be defined in the search query or may be a default value.

[0037] During processing, facets with different levels of specificity may be extracted from documents, disambiguated, normalized, grouped by topic and indexed and added temporarily to a knowledge base. In some embodiments, the extracted facets may be used to map search results. In further embodiments, fuzzy matching algorithms may compare facets temporarily stored in collections with the one or more queries being generated by the system and drop down windows, which may include the most relevant level one facets, may serve search suggestions to users; the users may be allowed to select facets of different levels to narrow down search queries.

[0038] In one embodiment, a computer-implemented method comprises extracting, by a computer, one or more facets from a corpus comprising data representing text-based information; disambiguating, by the computer, each of the one or more facets extracted from the corpus; generating, by the computer, one or more indices associated with the one or more facets respectively; retrieving, by the computer, each of the facets based on the associated index from a record of partition comprising one or more records of a database, wherein a collection of the database comprises one or more partitions, and wherein each respective facet indicates a hierarchical relation of data stored in the database relative to the one or more records of data; and generating, by the computer, a suggested search query based on each of the facets.

[0039] Disclosed herein are connection configurations for nodes of a system hosting an in-memory database, the nodes having multiple connection bandwidth and latency tiers, where a first bandwidth tier may be associated with a bandwidth higher than a second bandwidth tier, the second bandwidth tier may be associated with a bandwidth higher than a third bandwidth tier, the third bandwidth tier may be associated with a bandwidth higher than

a fourth bandwidth tier, and the first latency tier may be associated with a latency lower than the second latency tier.

[0040] Disclosed herein is a distributed-computing system having multiple network segments, each with bandwidth and latency tiers applied to the distributed in-memory data platform. The system includes connection configurations having a suitable number of network segments, where network segments may be connected to a number of servers internal and external to the system, and to clusters of servers in the system. The servers of the system may include software modules such as search managers, analytics agents, search conductors, dependency managers, supervisors, and partitioners, amongst others. Servers and modules may be connected to the desired network segments to achieve desired bandwidth and latency needs. Servers and modules may be connected to the desired network segments to separate different classes of network traffic, to prevent one class of traffic from interfering with another.

[0041] In one embodiment, a system comprising one or more nodes hosting an in-memory database, the system comprises a plurality of storage nodes comprising non-transitory machine-readable storage medium storing one or more partitions of a collection, wherein the collection stored by each respective storage node contains one or more records of a database, and wherein the storage medium of each respective storage node comprises main memory; a search manager node comprising a processor generating one or more search conductor queries using a search query received from a user node, transmitting the one or more search conductor queries to one or more search conductor nodes according to the search query, and forward one or more sets of search results to one or more analytics agent nodes according to the search query responsive to receive the one or more sets of search results; an analytics agent node comprising a processor executing one or more analytics algorithms

responsive to receiving a set of search results from the search manager node; a search conductor node comprising a processor querying the collection of the database records of a storage node according to a search conductor query in response to receiving the search conductor query from the search manager, and transmitting the set of one or more search results to the search manager node in response to identifying the one or more search results of the set, wherein each respective search result corresponds to a database record containing data satisfying the search conductor query; and a plurality of network segments comprising one or more connections between one or more nodes communicating over each respective network segment, wherein a first network segment comprises the search manager, the search conductor, and the analytics agent.

[0042] Disclosed here is a fault tolerant architecture suitable for use with any distributed computing system. A fault tolerant architecture may include any suitable number of supervisors, dependency managers, node managers, and other modules distributed across any suitable number of nodes to maintain desired system functionality, redundancies and system reliability while sub-components of the system are experiencing failures. The present disclosure describes a fault tolerant architecture suitable for use with any distributed computing system. An example of a distributed computing system may be an in-memory database, but other distributed computing systems may implement features described herein. Systems and methods described herein provide fault-tolerance features for a distributed computing system, by automatically detecting failures and recovering from the detected failures by moving processing modules and each of the modules' associated dependencies (software, data, metadata, etc.) to other computer nodes in the distributed computing system capable of hosting the modules and/or the dependencies.

[0043] In one embodiment, a computer-implemented method comprises monitoring, by a computer comprising a processor executing a supervisor module, a heartbeat signal generated by a node manager monitoring one or more software modules stored on a node, wherein the heartbeat signal contains binary data indicating a status of each respective software module monitored by the node manager; detecting, by the computer, a failed software module in the one or more software modules of the node based on the heartbeat signal received from the node manager of the node; automatically transmitting, by the computer, to the node manager of the node a command instructing the node to restore the failed software module, in response to detecting the failed software module; and determining, by the computer, whether the node manager successfully restored the module based on the heartbeat signal received from the node manager.

[0044] In another embodiment, a computer-implemented method comprises continuously transmitting, by a computer, a heartbeat signal to a supervisor node; restoring, by the computer, the failed module when receiving a restore command; and transmitting, by the computer, a restored status signal to the supervisor node when the computer detects the module is restored.

[0045] In another embodiment, a fault-tolerant distributed computing system comprising: one or more nodes comprising a processor transmitting a heartbeat signal to a supervisor node and monitoring execution of one or more software modules installed on the node; and one or more supervisor nodes comprising a processor monitoring one or more heartbeat signals received from the one or more nodes, and determining a status of each respective node based on each respective heartbeat signal.

[0046] Disclosed herein are systems and methods for handling dependencies during the process of installing, upgrading, and configuring different software, data or metadata

packages for any suitable database or search engine. The systems and methods may automate processes for deploying, installing, and configuring various data, metadata, and software stored in a primary datastore of the distributed-computing system, such as a distributed system hosting an in-memory database, or other types of distributed data platforms. Exemplary embodiments may describe systems and methods in which a dependency manager (configuration management) may be linked directly to a supervisor (systems management), where supervisor may maintain the system in a fully functional manner, and may accept configuration requests to make changes in the system.

[0047] In one embodiment, a computer-implemented method comprises transmitting, by a computer of a distributed computing system, a request for a machine-readable deployable-package file associated with a target node of the system to a dependency manager node comprising a non-transitory machine-readable storage medium storing one or more deployable package files associated respectively with one or more nodes of the system according to dependency tree; transmitting, by the computer, the deployable package file to the target node in response to receiving the deployable package file from the dependency node, wherein the deployable package file associated with the target node contains a set of one or more dependency files based on the dependency tree; and instructing, by the computer, the target node to install the set of dependencies in the deployable package onto the target node.

[0048] In another embodiment, a computer-implemented method comprises determining, by a computer, a set of one or more dependency files to be installed onto a target node using a dependency tree associated with the target node responsive to receiving a request to configure the target node from a supervisor node; fetching, by the computer, each of the dependency files of the set of one or more dependency files from at least one dataframe

comprising non-transitory machine-readable storage medium storing one or more dependency files; generating, by the computer, a deployable package file comprising the set of one or more dependency files; and transmitting, by the computer, the deployable package file to the supervisor node.

[0049] In another embodiment, a database management system comprises one or more nodes comprising a non-transitory machine-readable storage memory storing one or more dependency files, and a processor monitoring a status of the one or more dependency files, wherein each respective dependency file is a component of the node having a comparative relationship with a corresponding component installed on a second node; one or more supervisor nodes comprising a processor monitoring a status for each of the one or more nodes and configured to transmit a deployable package comprising a set of dependencies files to each of the nodes based on the status of each respective node; and one or more dependency manager nodes comprising a non-transitory machine-readable storage medium storing one or more dependency tree files associated with the one or more nodes, and a processor configured to compile a deployable package file in accordance with a dependency tree associated with a node, wherein the deployable package file comprises a set of one or more dependencies files stored on at least one data frame, and wherein the dependency manager node determines a dependency to include in the deployable package based on a dependency tree associated with a node targeted to receive the deployable package.

[0050] Disclosed herein is a distributed-computing system architecture hosting an in-memory database, which may include any suitable combination of modules and clusters, including one or more of a system interface, a search manager, an analytics agent, a search conductor, a partitioner, a collection, a supervisor, a dependency manager, or any suitable combination. Embodiments of the system may have a pluggable architecture of nodes and

software modules, which may facilitate installing, embedding, or otherwise including additional components (e.g., nodes, modules, database instances) on-the-fly (i.e., without interrupting or otherwise disrupting status quo operation of the system). Embodiments of the system may accept later-developed or external, third-party custom analytics modules for inclusion to the in-memory database. Database queries may specify which analytics modules and parameters are to be applied on-the-fly to intermediate query results, without having to first retrieve data out of the database. Systems and methods described herein enable custom tailor analytics modules to be developed independently from the in-memory database, and can also be deployed within the system hosting the database to receive the performance benefits of executing analytics using the in-memory database. Exposed and accessible APIs may be used for communicating data with independently created analytics modules, which, because of the APIs, may be seamlessly plugged-into or otherwise integrated with the in-memory database. Validation of data may be available to determine whether the new modules conform to expectations of the API.

[0051] In one embodiment, an in-memory database system comprises one or more storage nodes comprising a non-transitory machine-readable storage media storing one or more records of a database, wherein the storage media of each respective storage node is a main memory of the respective storage node; an analytics agent nodes comprising a processor executing an analytics module using a set of query results as input parameters responsive to receiving a request for analytics indicating the analytics module, wherein the set of query results contains binary data representing one or more records retrieved from the one or more storage nodes storing the one or more records; and an analytics module datastore comprising non-transitory machine-readable storage media storing one or more analytics modules, and a processor configured to transmit a new analytics module to one or more analytics agent nodes.

[0052] Described herein are systems and methods providing a search paradigm that may be implemented for data storage systems, such as an in-memory database system, to provide users the ability to specify a query algorithm and a detailed scoring and ranking algorithm, such that different algorithms may be determined according to each of the separate aspects of a search query. Nodes conducting the search query may then find each of the possible candidate records using each of the specified query algorithms (even if some fields are empty or not defined in a particular schema), and then score and rank the candidate records using the specified scoring and ranking algorithms. Conventional systems do not offer the ability to provide separate query and scoring algorithms within a single search query, such that each scoring algorithm may operate on a completely separate set of fields. Systems and methods described herein provide such approaches to reduce the burden of data preparation and enables re-use of data for purposes not originally intended when the data was loaded.

[0053] Systems and methods described herein provide for non-exclusionary searching within clustered in-memory databases. The non- exclusionary search methods may allow the execution of searches where the results may include records where fields specified in the query are not populated or defined. The disclosed methods include the application of fuzzy indexing, fuzzy matching and scoring algorithms, which enables the system to search, score and compare records with different schemata. This significantly improves the recall of relevant records.

[0054] The system architecture of an in-memory database that may support the disclosed non-exclusionary search method may include any suitable combination of modules and clusters; including one or more of a system interface, a search manager, an analytics

agent, a search conductor, a partitioner, a collection, a supervisor, a dependency manager, or any suitable combination.

[0055] The system may score records against the one or more queries, where the system may score the match of one or more available fields of the records and may then determine a score for the overall match of the records. If some fields are missing, a penalty or lower score may be assigned to the records without excluding them. The system may determine whether the score is above a predefined acceptance threshold, where the threshold may be defined in the search query or may be a default value. In further embodiments, fuzzy matching algorithms may compare records temporarily stored in collections with the one or more queries being generated by the system.

[0056] Disclosed here are methods for compressing structured or semi-structured data, though it should be appreciated that a variety of suitable compression algorithms may be utilized (i.e., no particular compression algorithm is required). System and method embodiments described herein may implement a combination of suitable data compression processes to each field of database, such that a compressed database record achieves a compression ratio comparable to commercially-accepted ratios, while still allowing decompression of the fields to occur only for the records and fields of interest (i.e., only decompressing data records or fields satisfying a database search query). Implementing compression techniques that facilitate selective decompression of records or fields allows for horizontal record-based storage of the compressed data, but also columnar or vertical access to the fields of the data on decompression. This provides the reduced storage benefits of compression, while avoiding much of the compute power and latency associated with decompression when only specific fields are to be decompressed.

[0057] Systems and methods described herein may also implement N-gram compression techniques. Conventionally, N-grams are restricted to compressing only one of chains of letters (successive characters of a string), or to chains of words (successive strings in text). Conventional N-gram compression is unable to compress chains of letters, individual words, and/or chains of words, within a single implementation of such a compression technique. Described herein is the use of N-gram-related compression for columnar compression during record storage, thereby allowing good overall compression, while still providing low-latency access to a single record or a single field within a record, in response to search queries.

[0058] Systems and methods described herein describe embodiments of compression techniques as applying to in-memory databases and document databases. However, it should be appreciated that such techniques and other aspects of the systems and methods may be applied to more general data compression.

[0059] In one embodiment, a computer-implemented method comprises determining, by a computer, a compression technique to apply to one or more data elements received in a set of data elements, wherein the computer uses a schema to determine the compression technique to apply to each data element based on a data type of the data element; compressing, by a computer, a data element using the compression technique defined by the schema, wherein the compression technique compresses the data element such that the data element is individually decompressed when returned in response to a search query; storing, by the computer, each compressed data element in a field of a record that stores data of the data type of the data element; associating, by the computer, a field notation in a reference table for each field according to a schema, wherein the representative notation identifies the data type of the field; querying, by the computer, the database for a set of one or more data

elements satisfying a search query received from a search conductor; and decompressing, by the computer, each of the one or more data elements of the one or more data elements satisfying the search query using the compression technique to apply responsive to identifying the set one or more data elements satisfying the search query, wherein each data element not satisfying the search query remains compressed.

[0060] In another embodiment, a computing system comprises one or more nodes storing one or more collections, each collection comprising a set of one or more records, each record comprising a set of fields storing data; and a compression processor compressing one or more of the fields according to a schema that is associated with a collection.

[0061] Numerous other aspects, features of the present disclosure may be made apparent from the following detailed description. Additional features and advantages of an embodiment will be set forth in the description which follows, and in part will be apparent from the description. The objectives and other advantages of the invention will be realized and attained by the structure particularly pointed out in the exemplary embodiments in the written description and claims hereof as well as the appended drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[0062] The present disclosure can be better understood by referring to the following figures. The components in the figures are not necessarily to scale, emphasis instead being placed upon illustrating the principles of the disclosure. In the figures, reference numerals designate corresponding parts throughout the different views.

[0063] **FIG. 1** shows an in-memory database architecture, according to an embodiment.

[0064] **FIG. 2** is a flow chart describing a method for faceted searching, according to an embodiment.

[0065] **FIG. 3** is a flow chart of a method for generating search suggestions using faceted searching, according to an embodiment.

[0066] **FIG. 4** shows an in-memory database architecture according to an exemplary embodiment.

[0067] **FIG. 5** shows a node configuration according to an exemplary embodiment.

[0068] **FIG. 6** is a flow chart for setting up a node according to an exemplary embodiment.

[0069] **FIG. 7** is a flow chart depicting module set up in a node according to an exemplary embodiment.

[0070] **FIG. 8** is a flow chart describing the function of a search manager according to an exemplary embodiment.

[0071] **FIG. 9** is a flow chart describing the function of a search conductor according to an exemplary embodiment.

[0072] **FIG. 10** is a flow chart describing the function of a partitioner according to an exemplary embodiment.

[0073] **FIG. 11** is a flow chart describing a process of setting up a partition in a search conductor according to an exemplary embodiment.

[0074] **FIG. 12A** shows a collection, its updated version, and their associated partitions according to an exemplary embodiment.

[0075] **FIG. 12B** shows a first and second search node including a first collection connected to a search manager according to an exemplary embodiment.

[0076] **FIG. 12C** shows a first search node including a first collection disconnected from a search manager and a second search node including a first collection connected to a search manager according to an exemplary embodiment.

[0077] **FIG. 12D** shows a first search node loading an updated collection, and a second search node connected to a search manager according to an exemplary embodiment.

[0078] **FIG. 12E** shows a first search node including an updated collection connected to a search manager, and a second search node including a first collection disconnected from a search manager according to an exemplary embodiment.

[0079] **FIG. 12F** shows a second search node loading an updated collection, and a first search node connected to a search manager according to an exemplary embodiment.

[0080] **FIG. 12G** shows a first and second search node including an updated collection connected to a search manager according to an exemplary embodiment.

[0081] **FIG. 13** shows a cluster of search nodes including partitions for two collections according to an exemplary embodiment.

[0082] **FIG. 14** is a connection diagram for a computing system hosting an in-memory database system, in which the nodes are logically clustered.

[0083] **FIG. 15** shows components of a distributed system management architecture, according to an exemplary system embodiments.

[0084] **FIG. 16** shows an exemplary node configuration for a node in an exemplary system embodiment.

[0085] **FIG. 17** is a flowchart showing fault handling by a distribute computing system, according to an exemplary method embodiment.

[0086] **FIG. 18** illustrates a block diagram connection of supervisor and dependency manager, according to an embodiment.

[0087] **FIG. 19** is a flowchart diagram of a configuration process, according to an embodiment.

[0088] **FIG. 20** illustrates a block diagram of dependencies used for the configuration of a system, according to an embodiment.

[0089] **FIG. 21** shows an in-memory database architecture, according to an embodiment.

[0090] **FIG. 22** is a flowchart of a method for adding new modules to an in-memory database, according to an embodiment.

[0091] **FIG. 23** shows an in-memory database architecture, according to an embodiment.

[0092] **FIG. 24** is a flow chart describing a method for non-exclusionary searching, according to an embodiment.

[0093] **FIG. 25** illustrates a data compression apparatus according to an exemplary embodiment.

[0094] **FIG. 26** illustrates a structured data table according to an exemplary embodiment.

[0095] **FIG. 27** illustrates a token table according to an exemplary embodiment.

[0096] **FIG. 28** illustrates a n-gram table according to an exemplary embodiment.

[0097] **FIG. 29** illustrates a table describing compressed records according to an exemplary embodiment.

DEFINITIONS

[0098] As used here, the following terms may have the following definitions:

[0099] **"Entity Extraction"** refers to information processing methods for extracting information such as names, places, and organizations.

[0100] **"Corpus"** refers to a collection of one or more documents

[0101] **"Features"** is any information which is at least partially derived from a document.

[0102] **"Event Concept Store"** refers to a database of Event template models.

[0103] **"Event"** refers to one or more features characterized by at least the features' occurrence in real-time.

[0104] **"Event Model"** refers to a collection of data that may be used to compare against and identify a specific type of event.

[0105] **"Module"** refers to a computer or software components suitable for carrying out at least one or more tasks.

[0106] **"Database"** refers to any system including any combination of clusters and modules suitable for storing one or more collections and suitable to process one or more queries.

[0107] **"Query"** refers to a request to retrieve information from one or more suitable databases.

[0108] **"Memory"** refers to any hardware component suitable for storing information and retrieving said information at a sufficiently high speed.

[0109] **"Node"** refers to a computer hardware configuration suitable for running one or more modules.

[0110] **"Cluster"** refers to a set of one or more nodes.

[0111] **"Collection"** refers to a discrete set of records.

[0112] **"Record"** refers to one or more pieces of information that may be handled as a unit.

[0113] **"Partition"** refers to an arbitrarily delimited portion of records of a collection.

[0114] **"Search Manager"**, or **"S.M."**, refers to a module configured to at least receive one or more queries and return one or more search results.

[0115] **"Analytics Agent"**, **"Analytics Module"**, **"A.A."**, or **"A.M."**, refers to a module configured to at least receive one or more records, process said one or more records, and return the resulting one or more processed records.

[0116] **"Search Conductor"**, or **"S.C."**, refers to a module configured to at least run one or more search queries on a partition and return the search results to one or more search managers.

[0117] **"Node Manager"**, or **"N.M."**, refers to a module configured to at least perform one or more commands on a node and communicate with one or more supervisors.

[0118] **"Supervisor"** refers to a module configured to at least communicate with one or more components of a system and determine one or more statuses.

[0119] **"Heartbeat"**, or **"HB"**, refers to a signal communicating at least one or more statuses to one or more supervisors.

[0120] **"Partitioner"** refers to a module configured to at least divide one or more collections into one or more partitions.

[0121] **"Dependency Manager"**, or **"D.M."**, refers to a module configured to at least include one or more dependency trees associated with one or more modules, partitions, or suitable combinations, in a system; to at least receive a request for information relating to any one or more suitable portions of said one or more dependency trees; and to at least return one or more configurations derived from said portions.

[0122] **"Document"** refers to a discrete electronic representation of information having a start and end.

[0123] **"Live corpus"** refers to a corpus that is constantly fed as new documents are uploaded into a network.

[0124] **"Feature"** refers to any information which is at least partially derived from a document.

[0125] **"Feature attribute"** refers to metadata associated with a feature; for example, location of a feature in a document, confidence score, among others.

[0126] **"Fact"** refers to objective relationships between features.

[0127] **"Knowledge base"** refers to a base containing disambiguated features or facets.

[0128] **"Link on-the-fly module"** refers to any linking module that performs data linkage as data is requested from the system rather than as data is added to the system.

[0129] **"Sentiment"** refers to subjective assessments associated with a document, part of a document, or feature.

[0130] **"Topic"** refers to a set of thematic information which is at least partially derived from a corpus.

[0131] **"Prefix"** refers to a string of length p which comprises of the longest string of key characters shared by all sub-trees of the node and a data record field for storing a reference to a data record

[0132] **"Facet"** refers to clearly defined, mutually exclusive, and collectively exhaustive aspects, properties or characteristics of a class, specific subject, topic or feature.

[0133] **"Field"** refers to one data element within a record.

[0134] **"Schema"** refers to data describing one or more characteristics of one or more records.

[0135] **"Fragment"** refers to separating records into smaller records until a desired level of granularity is achieved.

[0136] **"Resources"** refers to hardware in a node configured to store or process data. In one or more embodiments, this may include RAM, Hard Disk Storage, and Computational Capacity, amongst others.

[0137] **"Dependency Tree"** refers to a type of data structure, which may show the relationship of partitions, modules, files, or data, among others.

[0138] **"Deployable Package"** refers to a set of information, which may be used in the configuration of modules, partitions, files, or data, among others.

[0139] **"Analytical Method"** refers to any operation that may be applied to an aggregate of search results.

[0140] **"Analytics Parameters"** refers to parameters that describe the operation that an analytic **module** may have to perform in order to get specific results.

[0141] **"Application Programming Interface,"** or **"API"** allows different user-developed **software** modules to be compatible with the components (e.g., software modules, nodes) of the distributed computing system.

[0142] **"Dictionary"** refers to a centralized repository of information, which includes details about the fields in a MEMDB such as meaning, relationships to other data, origin, usage, and format.

[0143] **"Not defined"** refers to a field that is not part of a particular schema.

[0144] **"Not populated"** refers to fields that are part of the schema, but have no assigned values.

[0145] **"Object"** refers to a logical collection of fields within a data record.

[0146] **"Array"** refers to an ordered list of data values within a record.

[0147] **"Compress"** may refer to reducing the amount of electronic data needed to represent a value.

[0148] **"Token Table"** refers to a table defining one or more simpler values for one or more other more complex values.

[0149] **"N-gram"** refers to N successive integral units of data which can be characters, words, or groups of words where N is greater than or equal to 1. I.e. in the sentence "The quick brown fox jumped over the lazy dog.", "the", "e", "he", and "brown fox" are all valid N-grams.

[0150] **"N-gram Table"** refers to a table defining one or more simpler values for one or more other more complex values.

[0151] **"JSON"** refers to the JavaScript Object Notation, a data-interchange format.

[0152] **"BSON"** refers to Binary JSON, a data-interchange format.

[0153] **"YAML"** refers to the coding language "YAML Ain't Markup Language," a data-interchange format.

[0154] **"Document Database"** refers to a document-oriented database, designed for storing, retrieving, and managing document-oriented information.

DETAILED DESCRIPTION

[0155] Reference will now be made in detail to the preferred embodiments, examples of which are illustrated in the accompanying drawings. The embodiments described above are intended to be exemplary. One skilled in the art recognizes that numerous alternative components and embodiments may be substituted for the particular examples described herein and still fall within the scope of the invention. Other embodiments may be used and/or other changes may be made without departing from the spirit or scope of the present disclosure. The illustrative embodiments described in the detailed description are not meant to be limiting of the subject matter presented here.

[0156] It will nevertheless be understood that no limitation of the scope of the invention is thereby intended. Alterations and further modifications of the inventive features illustrated here, and additional applications of the principles of the inventions as illustrated here, which would occur to one skilled in the relevant art and having possession of this disclosure, are to be considered within the scope of the invention.

[0157] The present disclosure describes a system and method for detecting, extracting and validating events from a plurality of sources. Sources may include news sources, social media websites and/or any sources that may include data pertaining to events.

[0158] Various embodiments of the systems and methods disclosed here collect data from different sources in order to identify independent events.

[0159] **FIG. 1** shows in-memory database **100** system architecture, according to an embodiment. MEMDB **100** system architecture may include system Interface **102**, first search manager **104**, nth search manager **106**, first analytics agent **108**, nth analytics agent **110**, first search conductor **112**, nth search conductor **114**, partitioner **116**, first collection **118**, nth collection **120**, supervisor **122**, and dependency manager **124**.

[0160] In one or more embodiments, system interface **102** may be configured to feed one or more queries generated outside of the system architecture of MEMDB **100** to one or more search managers in a first cluster including at least a first search manager **104** and up to nth search manager **106**. Said one or more search managers in said first cluster may be linked to one or more analytics agents in a second cluster including at least a first analytics agent **108** and up to nth analytics agent **110**.

[0161] Search managers in said first cluster may be linked to one or more search conductors in a third cluster including at least a first search conductor **112** and up to nth

search conductor **114**. Search conductors in said third cluster may be linked to one or more partitions **126**, where partitions corresponding to at least a First Collection **118** and up to nth Collection **120** may be stored at one or more moments in time.

[0162] One or more nodes, modules, or suitable combination thereof included in the clusters included in MEMDB **100** may be linked to one or more supervisors **122**, where said one or more nodes, modules, or suitable combinations in said clusters may be configured to send at least one heartbeat to one or more supervisors **122**. Supervisor **122** may be linked to one or more dependency managers **124**, where said one or more dependency managers **124** may include one or more dependency trees for one or more modules, partitions, or suitable combinations thereof. Supervisor **122** may additionally be linked to one or more other supervisors **122**, where additional supervisors **122** may be linked to said clusters included in the system architecture of MEMDB **100**.

[0163] **FIG. 2** is a flow chart describing a method for faceted searching **200**, according to an embodiment. Separating or grouping documents using facets may effectively narrow down search results. When performing a faceted search, each facet may be considered a dimension of a document in a multidimensional space and by selecting specific document facets the possibilities of finding relevant search results may be significantly improved while the time required to perform a search may be substantially shortened.

[0164] The process may start with query received by search manager **202**, in which one or more queries generated by an external source may be received by one or more search managers. In some embodiments, these queries may be automatically generated by a system interface **102** as a response to an interaction with a user. In one or more embodiments, the queries may be represented in a markup language, including XML and HTML. In one or more other embodiments, the queries may be represented in a structure, including

embodiments where the queries are represented in JSON. In some embodiments, a query may be represented in compact or binary format.

[0165] Afterwards, the received queries may be parsed by search managers **204**. This process may allow the system to determine if field processing is desired **206**. In one or more embodiments, the system may be capable of determining if the process is required using information included in the query. In one or more other embodiments, the one or more search managers may automatically determine which one or more fields may undergo a desired processing.

[0166] If the system determined that field processing for the one or more fields is desired, the one or more search managers may apply one or more suitable processing techniques to the one or more desired fields, during search manager processes fields **208**. In one or more embodiments, suitable processing techniques may include address standardization, proximity boundaries, and nickname interpretation, amongst others. In some embodiments, suitable processing techniques may include the extraction of prefixes from strings and the generation of non-literal keys that may later be employed to perform fuzzy matching techniques.

[0167] Then, when S.M. constructs search query **210**, one or more search managers may construct one or more search queries associated with the one or more queries. In one or more embodiments, the search queries may be constructed so as to be processed as a stack-based search.

[0168] Subsequently, S.M. may send search query to S.C. **212**. In some embodiments, one or more search managers may send the one or more search queries to one or more search conductors, where said one or more search conductors may be associated with collections specified in the one or more search queries.

[0169] The one or more search conductors may score records against the one or more queries, where the search conductors may score the match of one or more fields of the records and may then determine a score for the overall match of the records. The system may determine whether the score is above a predefined acceptance threshold, where the threshold may be defined in the search query or may be a default value. In one or more embodiments, the default score thresholds may vary according to the one or more fields being scored. If the search conductor determines in that the scores are above the desired threshold, the records may be added to a results list. The search conductor may continue to score records until it determines that a record is the last in the partition. If the search conductor determines that the last record in a partition has been processed, the search conductor may then sort the resulting results list. The search conductor may then return the results list to a search manager.

[0170] When S.M. receives and collates results from S.C.'s **214**, the one or more search conductors return the one or more search results to the one or more search managers; where, in one or more embodiments, said one or more search results may be returned asynchronously. The one or more search managers may then compile results from the one or more search conductors into one or more results list.

[0171] The system may determine whether analytics processing **216** of the search results compiled by the one or more search managers is desired. In one or more embodiments, the system determines if the processing is desired using information included in the query. In one or more other embodiments, the one or more search managers may automatically determine which one or more fields may undergo a desired processing.

[0172] If the system determines that aggregate analytics processing is desired, one or more analytics agents may process results **218**, through the application of one or more suitable processing techniques to the one or more results list. In one or more embodiments,

suitable techniques may include rolling up several records into a more complete record, performing one or more analytics on the results, and determining information about neighboring records, amongst others. In some embodiments, analytics agents may include disambiguation modules, linking modules, link on-the-fly modules, or any other suitable modules and algorithms. During processing, facets with different levels of specificity may be extracted from documents, disambiguated, normalized, group by topics and indexed. The facets may be indexed according to a hierarchy, where the hierarchy may be predefined or defined by the system on the fly. In one or more embodiments, level 1 facets may be broadest facets and subsequent levels may be derived with descending relevance or higher degree of specificity.

[0173] The facets from the results list may be stored in collections. In some embodiments, each facet type may be stored in a different collection or group of collections.

[0174] After processing, according to some embodiments, the one or more analytics agents may return one or more processed results lists to the one or more search managers.

[0175] A search manager may return search results **220**. In some embodiments, the one or more search managers may decompress the one or more results list and return them to the system that initiated the query.

[0176] Optionally, according to some embodiments, the search results may be temporarily stored in a knowledge base **222** and returned to a user interface **224**. The knowledge base may be used to temporarily store clusters of relevant disambiguated facets and their related features. When new documents may be loaded into the MEMDB, the new disambiguated set of facets may be compared with the existing knowledge base in order to determine the relationship between facets and determine if there is a match between the new facets and previously extracted facets. If the facets compared match, the knowledge base may

be updated and the ID of the matching facets may be returned. If the facets compared do not match with any of the already extracted facets, a unique ID is assigned to the disambiguated entity or facet, and the ID is associated with the cluster of defining features and stored in within the knowledge base of the MEMDB.

[0177] **FIG. 3** is a flow chart of method for generating search suggestions **300** using faceted searching, according to an embodiment. Method for generating search suggestions **300** may begin with query generation **302**. In some embodiments, when a user starts typing on a suitable search window, the system may automatically generate queries which may be derived from the prefixes of the words that are being typed by the user on the search box. These queries may be generated even with a minimum number of characters typed in the search window (3 or 4) and before the user has finished typing a string on the search window.

[0178] Once a suitable query is generated, method for faceted searching **200** may be applied. The application of this method may include the use of literal or non-literal key algorithms that may allow matching of partial prefixes.

[0179] To generate search suggestions **304**, fuzzy matching algorithms may compare facets temporarily stored in collections with the one or more queries being generated by the system. In this manner, counts of hits with respect to the current one or more queries may be assigned to the facets of different hierarchy levels that may be in different partitions of the collections. Then, level 1 facets, may be assigned a cumulative count of hits and sorted according to the total number of hits.

[0180] Afterwards, search suggestions may be presented to user **306**. In some embodiments, this may be done in the form of a drop down window, which may include the most relevant level one facets, each one with its associated number of hits and the user may be allowed to select facets of different levels to narrow down search queries or to start new

queries. According to some embodiments, this process may be able to generate and serve faceted search suggestions before a user has finished typing a string on a search window, as characters in a search window start to develop words this process may happen several times.

[0181] FIG. 4 shows system architecture 400 having system interface 402, first search manager 410, nth search manager 412, first analytics agent 420, nth analytics agent 422, first search conductor 430, nth search conductor 432, partition data 440, partitioner 450, first collection 460, nth collection 462, supervisor 470, and dependency manager 480.

[0182] In one or more embodiments, system interface 402 may feed one or more queries generated outside system architecture 400 to one or more search managers 410, 412 in a first cluster including at least one node including a first search manager 410 and up to n nodes including an nth search manager 412. The one or more search managers 410, 412 in said first cluster may be linked to one or more analytics agents 420, 422 in a second cluster including at least a first analytics agent 420 and up to nth analytics agent 422.

[0183] Search managers 410, 412 in the first cluster may be linked to one or more search conductors 430, 432 in a third cluster. The third cluster may include at least a first search conductor 430 and up to an nth search conductor 432. Each search node (i.e., node executing search manager 410, 412) may include any suitable number of search conductors 430, 432.

[0184] Search conductors 430, 432 in the third cluster may be linked to one or more database nodes storing partition data 440. Partition data 440 may include one or more partitions (i.e., arbitrarily delimited portions of records partitioned from a discrete set of records) generated by a node executing one or more partitioners 450, which may be a module configured to at least divide one or more collections into one or more partitions. Each of the partitions may correspond to at least a first collection 460 and up to nth collection 462. The

collections **460**, **462** may additionally be described by one or more schemata files, which may define the data in the collections **460**, **462**. The one or more schemata may include information about the name of the fields in records of the partitions, whether said fields are indexed, what compression method was used, and what scoring algorithm is the default for the fields, amongst others. The schemata may be used by partitioners **450** when partitioning the first collection **460** and up to nth collection **462**, and may be additionally be used by the first search manager **410** and up nth search manager **412** when executing one or more queries on the collections.

[0185] One or more nodes may execute a supervisor **470** software module that receives a heartbeat signal transmitted from other nodes of the system **400**. A supervisor **470** may be configured to receive data from nodes of the system **400** that execute one or more dependency manager **480** software modules. A dependency manager **480** node may store, update, and reference dependency trees associated with one or more modules, partitions, or suitable combinations thereof, which may indicate configuration dependencies for nodes, modules, and partitions, based on relative relationships. A supervisor **470** may additionally be linked to other nodes in the system **400** executing one or more other supervisors **470**. In some cases, links to additional supervisors **470** may cross between clusters of the system architecture **400**.

[0186] Nodes executing an analytics agent **420**, **422** may execute one or more suitable analytics modules, which conform to a specified application programming interface (API) that facilitates interoperability and data transfer between the components of the system (e.g., software modules, nodes). Analytics agents **420**, **422** may be configured to process aggregated query results returned from search conductors **430**, **432**. For example, a search manager **410** may receive a search query and then generate search conductor queries, which the search manager **410** issues to one or more search conductors **430**, **432**. After the search

conductors **430**, **432** execute their respectively assigned search conductor queries, the search manager **410** will receive a set of aggregated query results from the one or more search conductors **430**, **432**. The search manager **410** may forward these search query results to an analytics agent **420** for further processing, if further processing is required by the parameters of the search query.

[0187] In some implementations, after a search manager **410** determines the search query has requested for an analytics agent **420** to process one or more sets of aggregated results received from the search conductors **430**, **432**, the search manager **410** may transmit a database schema file and/or one or more analytical parameters to the analytics agents **420**, **422**. In some cases, the search query may request particular analytics algorithms to be performed, which the search manager **410** may use to identify which analytics agent **420** should receive aggregated search results. In some cases, one or more of the sets of aggregated results may be transmitted to the analytics agents **420**, **422** in the form of compressed records, which contain data compressed according to a compression algorithm. In some cases, data of the records may be compressed at the fields of the records; and in some cases, full records may be compressed.

[0188] Nodes executing analytics agents **420**, **422** having various analytics modules. Non-limiting examples may include: disambiguation modules, linking modules, and link on-the-fly modules, among other suitable modules and algorithms. As detailed later, linking modules and link-on-the-fly modules may identify, generate, and/or store metadata that links data previously stored in records of the database. Suitable modules may include any software implementation of analytical methods for processing any kind of data. In some embodiments, particular analytics modules or analytics agents **420**, **422** may be accessible only to predetermined instances, clusters, partitions, or/or instantiated objects of an in-memory database.

[0189] **FIG. 5** is a diagram showing a configuration of a node **500**, according to an exemplary embodiment. The node **500** in **FIG. 5** may comprise a processor executing a node manager **502** software module and any number of additional software modules **510**, **512**, which may include a first software module **510** and up to nth module **512**.

[0190] According to the exemplary configuration of **FIG. 5**, the node **500** may be communicatively coupled over a data network to a second node executing a supervisor module, or supervisor node. A node manager **502** be installed and executed by the node **500** may also configured to communicate with the supervisor node, and may also be configured to monitor a software modules **510**, **512** installed on the node, including a first module **510**, up to nth module **512**. Node manager **502** may execute any suitable commands received from the supervisor, and may additionally report on the status of one or more of the node **500**, node manager **502**, and from the first module **510** to the nth module **512**. The first module **510** may be linked to the one or more supervisors and may be linked to one or more other modules in the node, where other modules in the node may be of a type differing from that of first module **510** or may share a type with first module **510**. Additionally, first module **510** may be linked with one or more other modules, nodes, or clusters in the system.

[0191] **FIG. 6** is a flowchart depicting node set-up **600** having steps **602**, **604**, and **606**.

[0192] In step **602**, an operating system (OS) suitable for use on a node is loaded to the node. In one or more embodiments, the OS may be loaded automatically by the node's manufacturer. In one or more other embodiments, the OS may be loaded on the node by one or more operators.

[0193] In step **604**, a node manager suitable for use with the OS loaded on the node is installed manually by one or more operators, where the installation may determine which one or more desired modules additional to node manager will be installed on the node.

[0194] In step **606**, the node manager sends a heartbeat to a supervisor, where said heartbeat may include information sufficient for the supervisor to determine that the node is ready to receive instructions to install one or more modules.

[0195] **FIG. 7** is a flow chart depicting module set-up **700** having steps **702**, **704**, **706**, **708**, **710**, **712**, and **714**.

[0196] In step **702**, the supervisor determines one or more modules are to be installed on one or more nodes, based on the needs of the data collections defined for the system. A supervisor then sends the installation preparation instruction to one or more node managers on said one or more nodes. In some embodiments, the supervisor may track the data collections (including data shards, or portions of data) and the configuration settings associated with the respective collections. The supervisor may also be aware of all available nodes and their resources (as reported by Node Managers). The supervisor may map (i.e., correlate) the system needs to available node resources to determine which data shards or portions, and which system services or resources, should be running on each respective node. The supervisor may then sends deploy/install requests, including any dependencies defined, to the appropriate Node Managers to instruct the node managers to execute the installation on the client-side.

[0197] In step **704**, the node manager allocates the node's resources, such as computer memory, disk storage and/or a portion of CPU capacity, for running the one or more desired modules. In one or more embodiments, the allocation of resources may expire after a period of time should the supervisor discontinue the process. Non-limiting examples of resources

can include computer memory, disk storage and/or a portion of CPU capacity. The resources required may be determined using the data and/or the services that the supervisor is assigning to a given node. Details of required resources may be specified in the package that defines the software and data dependencies, which is stored in the dependency manager.

[0198] In step **706**, the supervisor sends a request to a dependency manager for one or more configuration packages associated with the one or more modules to be installed on the node.

[0199] In step **708**, the supervisor may then send the configuration package to the node manager to be deployed, installed and started. The configuration package, which includes all data, software and metadata dependencies, is defined by a system administrator and stored in the dependency manager.

[0200] In step **710**, the node manager reads any software and data required to run the one or more modules from a suitable server. Suitable software and data may include software, data and metadata suitable for indexing, compressing, decompressing, scoring, slicing, joining, or otherwise processing one or more records, as well as software and data suitable for communicating, coordinating, monitoring, or otherwise interacting with one or more other components in a system.

[0201] In step **712**, the node manager installs the required software fetched in step **710**.

[0202] In step **714**, the node manager executes the software installed in step **712**.

[0203] **FIG. 8** is a flow chart depicting Query Processing **800**, having steps **802**, **804**, **808**, **810**, **812**, **814**, **818**, and **820**, and having checks **806** and **816**.

[0204] In step **802**, database queries generated by an external source, such as a browser-based graphical user interface (GUI) hosted by the system or a native GUI of the client computer, are received by one or more search managers. The queries may comprise binary data representing any suitable software source code, which may contain a user's submitted or a program's automatically-generated search parameters. The source code language used for search queries may be a data serialization language capable of handling complex data structures, such as objects or classes. Data serialization languages may be used for converting complex data objects or structures to a sequence of digital bits, and may provide a data of complex objects in a format that may be managed by most any devices. In some embodiments, the queries may be represented in a markup language, such as XML and HTML, which may be validated or otherwise understood according to a schema file (e.g., XSD). In some embodiments, queries may be represented as, or otherwise communicate, a complex data structure, such as JSON, which may be validated or otherwise understood according to a schema file. Queries may contain instructions suitable to search the database for desired records satisfying parameters of the query; and in some embodiments the suitable instructions may include a list of one or more collections to search.

[0205] In step **804**, the queries received from the external source may be parsed using according to the associated query language (e.g., SQL) by the one or more search managers, thereby generating a machine-readable query to be executed by the appropriate nodes (e.g., search conductor, analytics agent). In some cases, schema files associated with the software language of the queries may be provided with the query, generated by code generating the query, an accepted standard, or native to the search managers. The schema files may instruct the search managers on parsing the search queries appropriately. For example, if the search queries are prepared using one or more markup languages (e.g., XML) or include a data structure (e.g., JSON), then a schema file, such as an XSD-based schema file, may be

associated with the search query code or the data structure to identify and/or validate data within each of the markup tags of the XML code or the JSON code.

[0206] In check **806**, a search manager may determine, based on the user-provided or application-generated query, whether processing one or more fields of database and/or the queries should be performed. Non-limiting examples of field processing may include: address standardization, determining proximity boundaries, and synonym interpretation, among others. In some embodiments, automated or manual processes of the system may determine and identify whether any other processes associated with the search process **800** will require the use of the information included in the fields of the queries. In some embodiments, the one or more search managers may automatically determine and identify which of the one or more fields of a query may undergo a desired processing.

[0207] In step **808**, after the system determines that field processing for the one or more fields is desired in check **806**, the search managers may apply one or more suitable field processing techniques to the desired fields accordingly.

[0208] In step **810**, search managers may construct search conductor queries that are associated with the search queries. In some embodiments, the search conductor queries may be constructed so as to be processed by the various nodes of the system (e.g., search managers, search conductors, storage nodes) according to any suitable search query execution plan, such as a stack-based search. It should be appreciated that the search queries may be encoded using any suitable binary format or other machine-readable compact format.

[0209] In step **812**, the one or more search managers send the one or more search conductor queries to one or more search conductors. In some embodiments, the search managers may automatically determine which search conductors should receive search conductor queries and then transmit the search conductor queries to an identified subset of

search conductors. In such embodiments, search conductors may be pre-associated with certain collections of data; and search queries received from the system interface may specify collections to be queried. As such, the search managers transmit search conductor queries to the search conductors associated with the collections specified in the one or more search queries.

[0210] In step **814**, search conductors return search results to the corresponding search managers. In some embodiments, the search results may be returned synchronously; and in some embodiments, the search results may be returned asynchronously. Synchronously may refer to embodiments in which the search manager may block results or halt operations, while waiting for search conductor results from a particular search conductor. Asynchronously may refer to embodiments in which the search manager can receive results from many search conductors at the same time, *i.e.*, in a parallel manor, without blocking other results or halting other operations. After receiving the search results from search conductors, the search managers may collate the results received from the respective search conductors, based on record scores returned from the search conductors, into one or more results lists.

[0211] In check **816**, a search manager may determine whether additional analytics processing of the search results compiled by the search managers should be performed, based on an indication in the search query. In some cases, the indication may be included in the search query by the user. In some embodiments, the system determines if the analytics processing is desired using information included in the search query. In some embodiments, the one or more search managers may automatically determine fields should undergo a desired analytics processing. Search queries may be constructed in a software programming language capable of conveying instructions along with other data related to the search query (e.g., strings, objects). Some programming languages, such as markup languages, may use

metadata tags embedded into the code to identify various types of data, such as a field indicating a Boolean value whether analytics should be performed or a more complex user-defined field indicating a specific analytics module to be executed and/or the analytics agent node hosting the specific analytics module. Some programming languages, such as javascript or PHP, may reference stored computer files containing code that identifies whether analytics should be performed, which may be a more complex user-defined field indicating the specific analytics module to be executed and/or the analytics agent node hosting the specific analytics module.

[0212] In step **818**, if the system determines in check **816** that processing is desired, one or more analytics agents apply one or more suitable processing techniques to the one or more results lists. In one or more embodiments, suitable techniques may include rolling up several records into a more complete record, performing one or more analytics on the results, and/or determining information about relationships between records, amongst others. The analytics agent may then return one or more processed results lists to the one or more search managers.

[0213] In step **820**, the one or more search managers may decompress the one or more results lists and return them to the system that initiated the query.

[0214] **FIG. 9** is a flow diagram depicting search conductor function **900**, having steps **902**, **904**, **908**, **910**, and **912** as well as check **906**.

[0215] In step **902**, a search manager sends a query to one or more search conductors.

[0216] In step **904**, a search conductor executes the query against its loaded partition, generating a candidate result set. In one or more embodiments, step **904** may include one or

more index searches. In one or more embodiments, the search conductor may use information in one or more schemata to execute the query.

[0217] In check **906**, the search conductor determines, based on the specified query, whether scoring has been requested in the search conductor query. Scoring may be indicated in the search query received by the search manager.

[0218] If scoring is requested, the search conductor scores the candidate result set in step **908**. A default score threshold may be defined in the schema, or may be included in the search conductor query sent by the search manager in step **902**. In one or more embodiments, an initial scoring may be done by the search conductor at the field level with field specific scoring algorithms, of which there may be defaults which may be overridden by one or more other scoring algorithms. Scoring algorithms may be defined or otherwise identified in the search query and/or the search conductor query, and may be performed by the search conductor accordingly. The search conductor may give the record a composite score based on those individual field scores. In some embodiments, one or more aggregate scoring methods may be applied by the search conductor, which can compute scores by aggregating one or more field scores or other aggregated scores.

[0219] In step **910**, the search conductor then uses the scores to sort any remaining records in the candidate result set.

[0220] In check **912**, the search conductor returns the candidate result set to the search manager, where the number of results returned may be limited to a size requested in the query sent by the search manager in step **902**.

[0221] **FIG. 10** is a flow diagram depicting collection partitioning **1000**, having steps **1002**, **1004**, **1006**, **1010**, and **1012**, as well as perform check **1008**.

[0222] In step **1002**, one or more collections are fed into one or more partitioners. The collections are fed in conjunction with one or more schemas so that the one or more partitioners can understand how to manipulate the records in the one or more collections.

[0223] In step **1004**, the records in the one or more collections are fragmented.

[0224] In check **1008**, the system checks the schema for the given data collection and determines whether any fields in the partitions are to be indexed by the partitioner. An index may be any suitable example of a field-index, used in any known database, such as a date index or a fuzzy index (e.g., phonetic).

[0225] In step **1010**, if the system determined in check **1008** that the partitioner is to index any fields in the partitions, the partitioner indexes the partitions based on the index definition in the schema.

[0226] In check **1012**, the system checks the schema for the given data collection and determines whether the partitions are to be compressed by the partitioner.

[0227] In step **1014**, if the system determined in check **1012** that the partitioner is to compress the partitions, the partitioner compressed the fields and records using the compression methods specified in the schema, which can be any technique suitable for compressing the partitions sufficiently while additionally allowing decompression at the field level.

[0228] In step **1016**, the system stores the partitions suitable for distributing the partitions to one or more search conductors.

[0229] Collection partitioning **1000** may create an initial load, reload or replacement of a large data collection. The partitioner may assign unique record IDs to each record in a collection and may assign a version number to the partitioned collection, and may

additionally associate the required collection schema with that partition set version for use by one or more SMs and one or more SCs.

[0230] In a second embodiment, new records may be added to a collection through one or more suitable interfaces, including a suitable query interface. The query interface may support returning result sets via queries, but may also support returning the collection schema associated with a collection version. Additionally, the search interface may allow one or more users to use that collection schema to add new records to the collection by submitting them through the search interface into the search manager. The search manager may then distribute the new record to an appropriate search conductor for addition to the collection. In some embodiments, the search manager may ensure eventual-consistency across multiple copies of a given partition and may guarantee data durability to non-volatile storage to ensure data is available after a system failure.

[0231] In one or more embodiments, records may be deleted in a similar manner. The result set from a query may include an opaque, unique ID for each record. This unique ID may encode the necessary information to uniquely identify a specific record in a given version of a collection and may include one or more of the collection name, the partition set version, and the unique record ID, amongst others. With appropriate permissions, the query interface may accept requests to delete a record corresponding to the unique record ID. This record may not be physically deleted immediately, and may be marked for deletion and may no longer be included in future answer sets.

[0232] In one or more other embodiments, a new collection schema or a delete request may be submitted to the query interface to create a new collection or remove an existing collection, respectively. A new collection created this way may start out empty,

where records can be added using any suitable mechanism, including the mechanism described above.

[0233] **FIG. 11** is a flow chart depicting partition loading **1100**, having steps **1102**, **1104**, **1106**, **1108**, **1112**, **1114**, **1116**, **1118** and **1120**, as well as perform check **1110**.

[0234] In step **1102**, a supervisor determines one or more partitions are to be loaded into one or more search conductors.

[0235] In step **1104**, the supervisor sends a configuration request to a dependency manager, and the dependency manager returns one or more configuration packages associated with the one or more partitions to be loaded on the one or more search conductors.

[0236] In step **1106**, the supervisor determines which search conductors the partitions are to be loaded on. In one or more embodiments, the supervisor determines which one or more search conductors will be used so as to provide a desired failover ability. In one or more other embodiments, the supervisor determines which one or more search conductors will be used so as to better level out the work load perceived by one or more clusters.

[0237] In step **1108**, the supervisor sends a command to one or more node managers associated with the nodes including the one or more search conductors. In one or more embodiments, the command informs the one or more node managers to await further instructions from the supervisor for loading the partition onto the one or more search conductors. In another embodiment, the command may include the one or more configuration packages associated with the one or more partitions to be loaded into the one or more search conductors. In one or more other embodiments, the command may include instructions to prepare said one or more search conductors for loading a new partition into memory.

[0238] In step 1110, the one or more node managers allocate any node resources required for loading the partition.

[0239] In check 1112, the one or more node managers determine if one or more software or data updates are required to load the one or more partitions.

[0240] In step 1114, if the one or more node managers determined one or more software or data updates are required, the one or more node managers then retrieve said one or more software or data updates from one or more nodes suitable for storing and distributing said one or more software updates. The one or more node managers then proceed to install the one or more retrieved software or data updates.

[0241] In step 1116, the one or more node managers retrieve the one or more partitions from one or more nodes suitable for storing and distributing one or more partitions. In one or more embodiments, the retrieved partitions have previously been indexed and stored and once retrieved are loaded into memory associated with the one or more search conductors. In another embodiment, the retrieved partitions have not been indexed or compressed previous to being retrieved, and are indexed or compressed by the one or more search conductors prior to being loaded into memory associated with the one or more search conductors.

[0242] In step 1118, the one or more search conductors send heartbeats to the supervisor and the supervisor determines the one or more search conductors are ready for use in the system.

[0243] In step 1120, the supervisor informs one or more search managers the one or more search conductors are ready to receive search requests.

[0244] FIG. 12A shows collection 1202 and an update of collection 1202 denoted collection' 1210. Collection 1202 may be divided into at least a first partition 1204 and up to nth partition 1206, and collection' 1210 may be divided into at least a first partition' 1212 and up to nth partition' 1214.

[0245] FIG. 12B shows first search node 1220 having a first set of first partition 1204 and up to nth partition 1206 and second search node 1230 having a second set of first partition 1204 and up to nth partition 1206, where both first search node 1220 and second search node 1230 may be connected to at least one search manager 1240. Additionally, first search node 1220, second search node 1230 and search manager 1240 may be connected to one or more supervisors 1250.

[0246] FIG. 12C shows first search node 1220 having been disconnected from search manager 1240 as a result of a command from supervisor 1250, while second search node 1230 still maintains a connection. In one or more embodiments, this may allow search manager 1240 to run searches for records in collection 1202 as first search node 1220 is being upgraded.

[0247] FIG. 12D shows first search node 1220 being updated to include collection' 1210.

[0248] FIG. 12E shows first search node 1220 having first partition' 1212 and up to nth partition' 1214 connected to search manager 1240 as a result of a command from supervisor 1250. supervisor 1250 then sends a command to disconnect second search node 1230 from search manager 1240. In one or more embodiments, this may allow search manager 1240 to run searches for records in collection' 1210.

[0249] **FIG. 12F** shows second search node **1230** being updated to include collection' **1210**.

[0250] **FIG. 12G** shows first search node **1220** having a first set of first partition' **1212** and up to nth partition' **1214** and second search node **1230** having a second set of first partition' **1212** and up to nth partition' **1214** connected to search manager **1240**, where the connection between second search node **1230** and search manager **1240** may have been re-established as a result of a command from supervisor **1250**. This may allow search manager **1240** to run searches for records in collection' **1210** in either first search node **1220** or second search node **1230**.

[0251] **FIG. 13** shows search node cluster **1300**, having first search node **1302**, second search node **1304**, third search node **1306**, fourth search node **1308**, first partition **1310**, second partition **1312**, third partition **1314**, and fourth partition **1316** for a first collection, and a first partition **1320**, second partition **1322**, third partition **1324**, and fourth partition **1326** for a second collection.

[0252] Search node cluster **1300** may be arranged to as to provide a desired level of partition redundancy, where one or more search nodes may be added or removed from the system accordingly. Additionally, the partitions included in the one or more search nodes may vary with time, and may be loaded or unloaded by the search node's node manager following a process similar to partition loading **1100**. When updating or otherwise changing the partitions in search node cluster **1300**, a method similar to that described in **FIGs. 12A, 12B, 12C, 12D, 12E, 12F, and 12G** may be used.

[0253] **FIG. 14** show Connection Diagram **1400** having Line Type A **1402**, Line Type B **1404**, Line Type C **1406**, Line type D **1408**, First Network Segment **1410**, Second Network Segment **1412**, Third Network Segment **1414**, First Search Manager **1420**, nth

Search Manager **1422**, First Analytics Agent **1430**, nth Analytics Agent **1432**, First Search Conductor **1440**, nth Search Conductor **1442**, Partitioner **1450**, First Dependency Manager **1460**, nth Dependency Manager **1462**, First Supervisor **1470**, and nth Supervisor **1472** .

[0254] In one or more embodiments, Line Type A **1402** may represent a connection having a first bandwidth tier and a first latency tier, Line Type B **1404** may represent a connection having a second bandwidth tier and the first latency tier, Line Type C **1406** may represent a connection having a third bandwidth tier and a second latency tier, and Line Type D **1408** may represent a connection having the fourth bandwidth tier and the second latency tier. In one or more embodiments the first bandwidth tier may be associated with a bandwidth higher than the second bandwidth tier, the second bandwidth tier may be associated with a bandwidth higher than the third bandwidth tier, the third bandwidth tier may be associated with a bandwidth higher than the fourth bandwidth tier, and the first latency tier may be associated with a latency lower than the second latency tier.

[0255] In one or more embodiments, a First Network Segment **1410** may be connected to external servers using any suitable connection, including Line Type A **1402**, Line Type B **1404**, and Line Type C **1406**. First Network Segment **1410** may also be connected to a first cluster including a First Search Manager **1420** and up to an nth Search Manager **1422** using a Line Type A **1402** connection.

[0256] In one or more embodiments, a Second Network Segment **1412** may be connected to the first cluster including First Search Manager **1420** and up to nth Search Manager **1422** using a Line Type A **1402** connection. Second Network Segment **1412** may also be connected to a second cluster including a First Analytics Agent **1430** and up to an nth Analytics Agent **1432** a Line Type A **1402** connection, a third cluster including a First Search Conductor **1440** up to an nth Search Conductor **1442** using a Line Type B **1404** connection, a

fourth cluster including a First Dependency Manager **1460** up to nth Dependency Manager **1462** using a Line Type D **1408** connection, and a fifth cluster including a First Supervisor **1470** up to nth Supervisor **1472** using a Line Type D **1408** connection.

[0257] In one or more embodiments, the bandwidth tier of Line Type A **1402** may be sufficient for ensuring the first cluster including First Search Manager **1420** and up to nth Search Manager **1422** is able to at least receive an appropriate amount of information from a suitable number of search conductors in the third cluster including First Search Conductor **1440** up to an nth Search Conductor **1442**. The latency tier of Line Type A **1402** may be sufficiently low so as to at least allow the system to be responsive enough to carry out a desired number of queries.

[0258] In one or more embodiments, the bandwidth tier of Line Type B **1404** may be sufficient for ensuring search conductors in the third cluster including First Search Conductor **1440** up to an nth Search Conductor **1442** are able to at least return a desired size of results. The latency tier of Line Type B **1404** may be sufficiently low so as to at least allow the system to be responsive enough to carry out a desired number of queries.

[0259] In one or more embodiments, the bandwidth tier of Line Type D **1408** may be sufficient for ensuring dependency managers in the fourth cluster including First Dependency Manager **1460** up to nth Dependency Manager **1462** are able to at least receive a desired number of package requests and return a desired number of packagers. Additionally, the bandwidth tier of Line Type D **1408** may be sufficient for ensuring supervisors in the fifth cluster including First Supervisor **1470** up to nth Supervisor **1472** are able to at least monitor and manage a desired number of nodes and modules. The latency tier of Line Type D **1408** may be sufficiently low so as to at least allow the system to be managed in a desired period of time and to provide a desired monitoring frequency.

[0260] In one or more embodiments, a Third Network Segment **1414** may be connected to the third cluster including a First Search Conductor **1440** up to an nth Search Conductor **1442** using a Line Type C **1406** connection, the fourth cluster including a First Dependency Manager **1460** up to nth Dependency Manager **1462** using a Line Type D **1408** connection, the fifth cluster including a First Supervisor **1470** up to nth Supervisor **1472** using a Line Type D **1408** connection, and a sixth cluster including one or more Partitioners **1450** using a Line Type C **1406** connection.

[0261] In one or more embodiments, the bandwidth tier of Line Type B **1404** may be sufficient for ensuring one or more Partitioners **1450** are able to at least access a desired collection and output a desired number of partitions within a desired period of time. Additionally, the bandwidth tier of Line Type B **1404** may be sufficient for ensuring the first cluster including First Search Manager **1420** and up to nth Search Manager **1422** is able to at least load a desired number of partitions within a desired period of time. The latency tier of Line Type B **1404** may be sufficiently low so as to at least allow nodes using the connection to react to system commands within a desired period of time, and to allow the system to provide a desired monitoring frequency.

[0262] In one or more embodiments, the bandwidth tier of Line Type D **1408** may be sufficient for ensuring dependency managers in the fourth cluster including First Dependency Manager **1460** up to nth Dependency Manager **1462** are able to at least receive a desired number of package requests and return a desired number of packagers. Additionally, the bandwidth tier of Line Type D **1408** may be sufficient for ensuring supervisors in the fifth cluster including First Supervisor **1470** up to nth Supervisor **1472** are able to at least monitor and manage a desired number of nodes and modules. The latency tier of Line Type D **1408** may be sufficiently low so as to allow the system to be managed in a desired period of time and to provide a desired monitoring frequency.

[0263] In one or more embodiments, the fifth cluster including First Supervisor **1470** up to nth Supervisor **1472** may have a Line Type D **1408** connection to one or more node managers in any suitable number of nodes.

[0264] In one or more other embodiments, additional clusters including one or more other types of modules may be connected to First Network Segment **1410**, Second Network Segment **1412**, and/or Third Network Segment **1414**, where the connections may include Line Type A **1402**, Line Type B **1404**, Line Type C **1406**, and/or Line type D **1408** connections.

[0265] **FIG. 15** shows fault tolerant architecture **1500**, including supervisor **1502**, nth supervisor **1504**, first dependency node **1510**, dependency node manager **1512**, dependency manager **1514**, nth dependency node **1520**, nth dependency node manager **1522**, nth dependency manager **1524**, first node **1530**, node manager **1532**, modules **1534**, nth node **1540**, nth node manager **1542**, and nth modules **1544**. Some embodiments, such as the exemplary system **1500** of **FIG. 15**, may logically organize nodes into a plurality of clusters. However, some embodiments may have one logical cluster, if any.

[0266] In one or more embodiments, a first cluster may include a supervisor **1502** and up to nth supervisor **1504**. Each supervisor **1502** may comprise network interface components, such as a network interface card (NIC), suitable for facilitating communications between the supervisor **1502** and one or more nodes in a second cluster. The second cluster may include first dependency node **1510** and up to nth dependency node **1520**, where first dependency node **1510** may include node manager **1512** and dependency manager **1514** and nth dependency node **1520** may include nth node manager **1522** and nth dependency manager **1524**. Supervisors in said first cluster may additionally have any suitable number of connections suitable for communicating with one or more nodes in a third cluster including

first node **1530** and up to nth node **1540**, where first node **1530** may include node manager **1532** and any suitable number of modules **1534**, and nth node **1540** may include nth node manager **1542** and any suitable number of nth modules **1544**.

[0267] One or more supervisors **1502** may receive heartbeats from one or more node managers **1512**, one or more dependency managers **1514**, and any suitable number of node managers **1532** and modules **1534**. In one or more embodiments, this may allow the one or more supervisors **1502** to monitor the status of one or more nodes and/or modules in a distributed computing system. Additionally, supervisors **1502** may transmit one or more suitable commands to any suitable number of node managers **1512** and any suitable number of node managers **1532**.

[0268] In one or more embodiments, supervisors **1502** may request a configuration package from one or more dependency nodes **1510** when installing one or more modules **1534** on one or more nodes **1530**.

[0269] **FIG. 16** is a diagram showing a configuration of a node **1600**, according to an exemplary embodiment. The node **1600** in **FIG. 16** may comprise a processor executing a node manager **1602** software module and any number of additional software modules **1610**, **1612**, which may include a first software module **1610** and up to nth module **1612**. The software modules may include any of the system modules, including search managers, search conductors, analytics agents, supervisors and dependency managers.

[0270] According to the exemplary configuration of **FIG. 16**, the node **1600** may be communicatively coupled over a data network to a second node executing a supervisor module, or supervisor node. A node manager **1602** be installed and executed by the node **1600** may also configured to communicate with the supervisor node, and may also be configured to monitor a software modules **1610**, **1612** installed on the node, including a first

module **1610**, up to nth module **1612**. Node manager **1602** may execute any suitable commands received from the supervisor, and may additionally report on the status of one or more of the node **1600**, node manager **1602**, and from the first module **1610** to the nth module **1612**. The first module **1610** may be linked to the one or more supervisors and may be linked to one or more other modules in the node, where other modules in the node may be of a type differing from that of first module **1610** or may share a type with first module **1610**. Additionally, first module **1610** may be linked with one or more other modules, nodes, or clusters in the system.

[0271] **FIG. 17** is a flowchart for fault handling **1700**.

[0272] The supervisor maintains the definition and configuration of all data collections in the system, which may include settings per collection that indicate how many redundant copies of each partition are desired, how many times to try to restart failed components before moving them to another node, etc. The supervisor also maintains a list of available nodes and their resources, as provided by the node managers. From that information, the supervisor computes a desired system state by mapping the needed system modules to available nodes, while still complying with configuration settings. Fault handling **1700** begins with supervisor detecting a module failure **1702**, where one or more supervisors may detect failures of one or more modules by comparing the actual system state to a desired system state. In one or more embodiments, supervisors may detect failure when one or more heartbeats from node managers or system modules are no longer detected. In one or more other embodiments, heartbeats from one or more modules may include status information about one or more other modules that may be interpreted by the one or more supervisors.

[0273] A supervisor may store definitions of data collections and the configurations settings associated with the data collections. The supervisor may also store information about

available system resources, as reported by node managers. The configuration information may include settings per collection that indicate how many redundant copies of each respective partition are desired, how many times to try to restart failed components before moving them to another node, among other. From all this information, the supervisor derives a ‘desired’ system state that maps the needed system modules to available nodes, while still complying with configuration settings. All this information is represented as JSON objects which may be stored as JSON files on disk, or in a predefined data collection within the IMDB.

[0274] The supervisor may then detect if the associated node manager is functioning **1704**.

[0275] If the node manager associated with the one or more failed modules is functioning as desired or according to a status quo configuration, then supervisor may send one or more commands to the node manager instructing the node manager to attempt to restart the one or more failed modules, in a step **1706**.

[0276] The supervisor may then check if module is restored **1708**, and if so the process may proceed to end **1710**. In some implementations, the first action of any module is to report a status via heartbeats to one or more available supervisors. If it is determined that module function is not restored, as indicated by heartbeats, the supervisor may determine if the restart threshold has been reached **1712**. The threshold number of attempts is a configuration setting per collection, which may be set by the system administrator and stored with the supervisor. The supervisor determines that a module has failed and should be restarted or moved to another node. The supervisor sends commands to If the number of attempts has not been reached, the node manager attempts to restart module **1706**.

[0277] If the threshold has been reached, the supervisor determines the next suitable node to place the module 1714 and the supervisor requests the node manager on the new node to stage all module dependencies and start the current module 1716.

[0278] The supervisor may then check if module is restored 1718, and if so the process may proceed to end 1710. If the module is not restored, the system may check if the restart threshold for the new node has been reached 1720. If the threshold has not been reached, the supervisor requests the node manager on the new node to stage and start the current module 1716.

[0279] Otherwise, the supervisor may check if the global node retry threshold has been reached 1722. This value is also defined by a system administrator and may be stored with the supervisor in a script, or as JSON or similar data structure object. If the threshold has not been reached, the supervisor determines the next suitable node to place the module 1714 and attempts to restart the node on the new node. If the global threshold has been reached, the system may then raise an alarm indicating module failure 1724.

[0280] If the supervisor detects that the associated node manager is not functioning based on the corresponding heartbeats, as indicated by a lack of heartbeats or heartbeats from the node manager indicating a failed state, the supervisor selects a module associated with the node with a failed node manager 1726. Then, the supervisor determines the next suitable node to place the module 1728. Afterwards, the supervisor requests the node manager on the new node to stage and start the current module 1730.

[0281] The supervisor may then check if module is restored 1732. If the module is not restored, supervisor checks if the restart threshold for the new node has been reached 1734. If the threshold has not been reached, the supervisor requests the node manager on the new node to stage and start the current module 1730.

[0282] If the threshold has been reached, the supervisor then checks if the global node retry threshold has been reached **1736**. If the threshold has not been reached, the supervisor determines the next suitable node to place the module **1728** and attempts to restart the node on the new node. If the global threshold has been reached, the system may then raise an alarm indicating module failure **1738**.

[0283] Otherwise, if the module is restored, the supervisor then checks if there are more modules to be migrated off the failed node **1740**. If a node has failed, the supervisor is configured to migrate all of the services that had been running on the failed node **1740**, as defined in the desired state. The supervisor will calculate a new desired state without the failed node **1740** and will need to migrate services accordingly. In some implementations, the supervisor may select a module associated with the node having a failed node manager **1726** and the node manager attempts to stage and start the module.

[0284] If the supervisor determines no more modules are to be migrated, the process may end **1710**.

[0285] **FIG. 18** illustrates a block diagram connection **1800** of supervisor **1802** and dependency manager **1804**. Generally, supervisor **1802** may monitor the system and/or execute processes and tasks that maintain an operating state for the system. Supervisor **1802** may accept any suitable configuration requests to make changes in the system. Software or data configurations may be handled by nodes executing a dependency manager **1804** software module or a supervisor **1802** software module; however, the deployable package may be provided from a separate data frame. The separate data frame is a non-transitory machine-readable storage medium storing one or more releasable files used in preparing a deployable package according to a configuration.

[0286] According to one embodiment, the dependency manager **1804** may be used as a non-transitory machine-readable storage medium containing the maintenance or configuration of any suitable software or data component in the system. Those configurations may be driven by new data, metadata or software updates in a release process.

[0287] The dependency manager **1804** may play a role in configurations required by some processes in the system. That is, dependency manager **1804** may be directly connected with supervisor **1802** in order to provide the suitable dependencies, otherwise referred to as "packages," "configurations," "components," and/or "files," for the partitions, which may be used to update any suitable collection. Furthermore, supervisor **1802** may be linked to one or more dependency managers **1804** and may additionally be linked to one or more other supervisors **1802**, where additional supervisors **1802** may be linked to other components in the system.

[0288] **FIG. 19** is a flowchart diagram **1900** of a configuration process in the system.

[0289] According to another embodiment, the configuration process or maintenance process may include the information regarding what dependencies a module may have and needs to be deployed along with the module. The required files may be fetched from a separate non-transitory machine-readable storage, or "data frame." In some embodiments, this data frame may be external from the system architecture; for example, in the case of third-party vendor providing software updates. The dependencies in a suitable deployable package may include different types of files, data, or software that are directly linked or wrapped around the module or the partition that is being configured. The configuration process may include different steps step **1902**, **1904**, **1906**, **1908**, **1910**, and **1912**. The configuration process **1900** may begin in response to requests requiring the system to install or update, data or software components.

[0290] In a first step **1902**, processors of the system may automatically detect a situation that may trigger the configuration process **1900** sequence/steps.

[0291] In some embodiments, in step **1902**, a node of the system executing a supervisor module may poll components of the system, such as node manager software modules, responsible for reporting a health update, or "status," to the supervisor. In such embodiments, the supervisor may automatically detect failures throughout the system based on a lack of a heartbeat (HB) signal the supervisor expects to receive from any system module, as defined by the system configuration. The supervisor may then trigger configuration process **1900**, among other remedial processes, in response to detecting the missing HB signal.

[0292] In some embodiments, in step **1902**, a node of the system executing a supervisor module may trigger configuration process **1900** when the supervisor receives an external request for one or more changes in the system configuration, such as updates to a component or migration to new node hardware.

[0293] In step **1904**, the supervisor may send a request to the dependency manager to retrieve one or more deployment packages associated with one or more modules that are to be installed on the node. A deployment package defines each of the files and/or other materials required to satisfy the node configuration according to the dependency manager. The deployable package may contain all required dependencies, including source and destination information necessary for proper deployment and may contain module properties needed to configure or start the module. A particular dependency may have its own dependencies, also defined in the dependency manager, and therefore may be referred to as a dependency tree.

[0294] In step **1906**, the supervisor may transmit instructions to the dependency manager to fetch the required deployment packages from a data frame storing the deployment

package. The data frame may be any non-transitory machine-readable storage media, which may be located on any suitable computing device communicatively coupled to a node executing the dependency manager. In some cases, when a deployment package is generated, the deployment package contains all dependencies for the module being transmitted, as well as the source and destination information needed to properly deploy the deployment package. The deployment package may also include one or more module properties needed to configure or start the deployment package. Deployment packages may be generated through automated or manual processes. In manual example, a system administrator may identify and/or create a deployment package with the requisite files and data. In an automated example, the supervisor or dependency manager may automatically identify and/or generate the deployment package using the automatically identified files, which is usually accomplished through a test script generated by the dependency manager, thereby yielding installation speeds and distribution rates higher than could be done by a human.

[0295] In step **1908**, after the dependency manager receives the deployment packages from the data frame, the dependency manager may transmit the deployable package to the node executing the supervisor that requested the deployment packages.

[0296] In step **1910**, the supervisor may send the deployable package to the node manager of the node requiring the configuration.

[0297] In step **1912**, the node manager may copy files, install, and/or execute the deployable package received from the supervisor, thereby implementing the requisite maintenance, update, or configuration for the system.

[0298] **FIG. 20** illustrates block diagram of dependencies **2000** used for the configuration of a system. According to a further embodiment, the process for the maintenance or configuration of a system may include different components, such as,

dependency manager **2002**, supervisor **2004**, search node **2006**, node manager **2008**, and dependency tree **2010**, among others.

[0299] A dependency tree **2010** may include different types of files that may be directly linked or wrapped around a module or partition, such that, a dependency may be the degree to which each member of a partition relies on each one of the other members in the partition. For instance, dependency tree **2010** may include partition 1, which may depend on phonetic 1.0 and compression 1.0; subsequently, phonetic 1.0 may depend on software libraries (such as, processing DLL 1.0 and Input DLL 1.0), and compression 1.0 may depend on data-table 1.0 and so on.

[0300] The dependency manager **2002** may store a dependency tree **2010** associated with any releasable file of the system. In a further embodiment, if any suitable software or data component is released to components indicated within the dependency tree **2010**, the dependency manager **2002** may create a deployable package from one or more files stored on a data frame.

[0301] Supervisor **2004** may be linked to one or more dependency managers **2002** including one or more dependency trees **2010** for one or more modules, partitions, or suitable combinations thereof. Supervisor **2004** may additionally be linked to one or more supervisor **2004**, where additional supervisors **2004** may be linked to other components in the system.

[0302] FIG. 21 shows system architecture **2100** having system interface **2102**, first search manager **2110**, nth search manager **2112**, first analytics agent **2120**, nth analytics agent **2122**, first search conductor **2130**, nth search conductor **2132**, partition data **2140**, partitioner **2150**, first collection **2160**, nth collection **2162**, supervisor **2170**, and dependency manager **2180**.

[0303] In one or more embodiments, system interface **2102** may feed one or more queries generated outside system architecture **2100** to one or more search managers **2110**, **2112** in a first cluster including at least one node including a first search manager **2110** and up to n nodes including an n th search manager **2112**. The one or more search managers **2110**, **2112** in said first cluster may be linked to one or more analytics agents **2120**, **2122** in a second cluster including at least a first analytics agent **2120** and up to n th analytics agent **2122**.

[0304] Search managers **2110**, **2112** in the first cluster may be linked to one or more search conductors **2130**, **2132** in a third cluster. The third cluster may include at least a first search conductor **2130** and up to an n th search conductor **2132**. Each search node (i.e., node executing search manager **2110**, **2112**) may include any suitable number of search conductors **2130**, **2132**.

[0305] Search conductors **2130**, **2132** in the third cluster may be linked to one or more database nodes storing partition data **2140**. Partition data **2140** may include one or more partitions (i.e., arbitrarily delimited portions of records partitioned from a discrete set of records) generated by a node executing one or more partitioners **2150**, which may be a module configured to at least divide one or more collections into one or more partitions. Each of the partitions may correspond to at least a first collection **2160** and up to n th collection **2162**. The collections **2160**, **2162** may additionally be described by one or more schemata, which may define the data in the collections **2160**, **2162**. The one or more schemata may include information about the name of the fields in records of the partitions, whether said fields are indexed, what compression method was used, and what scoring algorithm is the default for the fields, amongst others. The schemata may be used by partitioners **2150** when partitioning the first collection **2160** and up to n th collection **2162**,

and may be additionally be used by the first search manager **2110** and up nth search manager **2112** when executing one or more queries on the collections.

[0306] One or more nodes may execute a supervisor **2170** software module that receives a heartbeat signal transmitted from other nodes of the system **2100**. A supervisor **2170** may be configured to receive data from nodes of the system **2100** that execute one or more dependency manager **2180** software modules. A dependency manager **2180** node may store, update, and reference dependency trees associated with one or more modules, partitions, or suitable combinations thereof, which may indicate configuration dependencies for nodes, modules, and partitions, based on relative relationships. A supervisor **2170** may additionally be linked to other nodes in the system **2100** executing one or more other supervisors **2170**. In some cases, links to additional supervisors **2170** may cross between clusters of the system architecture **2100**.

[0307] Nodes executing an analytics agent **2120**, **2122** may execute one or more suitable analytics modules, which conform to a specified application programming interface (API) that facilitates interoperability and data transfer between the components of the system (e.g., software modules, nodes). Analytics agents **2120**, **2122** may be configured to process aggregated query results returned from search conductors **2130**, **2132**. For example, a search manager **2110** may receive a search query and then generate search conductor queries, which the search manager **2110** issues to one or more search conductors **2130**, **2132**. After the search conductors **2130**, **2132** execute their respectively assigned search conductor queries, the search manager **2110** will receive a set of aggregated query results from the one or more search conductors **2130**, **2132**. The search manager **2110** may forward these search query results to an analytics agent **2120** for further processing, if further processing is required by the parameters of the search query.

[0308] In some implementations, after a search manager **2110** determines the search query has requested for an analytics agent **2120** to process one or more sets of aggregated results received from the search conductors **2130**, **2132**, the search manager **2110** may transmit a database schema file and/or one or more analytical parameters to the analytics agents **2120**, **2122**. In some cases, the search query may request particular analytics algorithms to be performed, which the search manager **2110** may use to identify which analytics agent **2120** should receive aggregated search results. In some cases, one or more of the sets of aggregated results may be transmitted to the analytics agents **2120**, **2122** in the form of compressed records, which contain data compressed according to a compression algorithm. In some cases, data of the records may be compressed at the fields of the records; and in some cases, full records may be compressed.

[0309] Nodes executing analytics agents **2120**, **2122** having various analytics modules. Non-limiting examples may include: disambiguation modules, linking modules, and link on-the-fly modules, among other suitable modules and algorithms. Suitable modules may include any software implementation of analytical methods for processing any kind of data. In some embodiments, particular analytics modules or analytics agents **2120**, **2122** may be accessible only to predetermined instances, clusters, partitions, or/or instantiated objects of an in-memory database.

[0310] **FIG. 22** is a flowchart of a method for adding analytics modules **2200** to a system hosting an in-memory database having steps **2202**, **2204**, **2206**, **2208**, **2210**, **2212**, **2214**, **2216** and **2218**, according to an embodiment.

[0311] In step **2202**, one or more suitable analytics modules may be created that conform to a suitable API for pluggable analytics in an in-memory database. The API may

have required methods that the analytics module must implement to provide system interoperability. Analytics modules may be created to satisfy user specific needs.

[0312] One or more analytics modules may be stored in a suitable module store. The module store is a non-transitory machine-readable storage medium that may be managed by a supervisor. Depending upon the embodiment, an entity, developer, user, component, module, external source, and/or other source responsible for building and/or managing analytics modules, may develop the analytics module using one or more suitable programming languages.

[0313] According to some embodiments, an API may serve as a software-to-software interface that may include sets of source code programming instructions and standards for a computer to compile and/or implement, such as parameters or arguments for routines, data structures, object classes, and variables. The APIs may allow the system to accept data inputs from, and output results to, later-developed software modules, while remaining agnostic to ownership, capabilities, or other characteristics of the later-developed modules, as long as the data inputs conform to the data formats (i.e., expected arguments). Some software routines of the system APIs responsible for data input and output may be "exposed" to such newly-developed or later-developed, and often external, software modules. Exposed APIs may validate data acceptability when the exposed APIs receive, fetch, or otherwise "consume" data from the software modules. Authoring software source satisfying the expected arguments of the system APIs may allow developers and other users to develop a variety of software modules, such as analytics modules, to communicate (i.e., transmit, receive) data with the nodes and modules of the system, such as the analytics agents.

[0314] Analytics agents may include one or more nodes within the system hosting the in-memory database, where each analytics agents' node may be able to store and execute one

or more analytics modules. APIs may allow different user-developed analytics modules to be compatible with the various nodes and modules of the system and the in-memory database.

[0315] In some embodiments, one or more modules may be external modules developed by third parties using any suitable programming language compatible with the APIs available. In such embodiments, these newly developed modules may be stored in the analytics module store.

[0316] In step 2204, the created module may be loaded into the in-memory database by adding the corresponding definition and any dependencies into the dependency manager, which may be accomplished using any suitable automated or manual processes capable of deploying, uploading, or otherwise storing, the appropriate files and instructions onto the dependency manager.

[0317] In step 2206, the supervisor determines, based on module settings in the dependency manager, if one or more modules are to be installed on one or more nodes. In some circumstances, module settings stored in the dependency manager will include whether a loaded analytic module is “enabled” or “disabled.” For example, if the settings indicate an analytics module is enabled, then the analytics module may be deployed to each respective node running the analytics agents performing that analytics module. A supervisor then sends installation preparation instructions to one or more node managers on said one or more nodes.

[0318] In step 2208, the node manager allocates the node's resources, based on module settings in the dependency manager, for running the one or more desired modules. In one or more embodiments, the allocation of resources may expire after a period of time should the supervisor discontinue the process. The module settings in the dependency manager will indicate how much memory, CPU and/or disk are required by the module.

[0319] In step **2210**, the supervisor sends a request to a dependency manager for one or more configuration packages associated with the one or more modules to be installed on the node. As an example, automated or manual processes (e.g., system administrator) may have added a new module to the dependency manager, which may also define any dependencies corresponding to configuration packages, such as other software or metadata.

[0320] In step **2212**, the supervisor may then send the configuration package to the node manager.

[0321] In step **2214**, the node manager reads any software and data required to run the one or more modules, as defined in the dependency manager.

[0322] In step **2216**, the node manager installs the required software and data fetched in step **2214**. Analytics Agents may dynamically load and unload modules once they are installed, so there may not be a need to restart any equipment or software and the installed one or more modules may be ready to be used.

[0323] In step **2218**, the node manager executes the software installed in step **2216**. After the analytics agent starts the analytics module, each analytics agent running the new module may transmit a heartbeat signal to a supervisor. The heartbeat signals may indicate the new module was properly started and is ready to use.

[0324] **FIG. 23** shows in-memory database **2300** system architecture, according to an embodiment. MEMDB **2300** system architecture may include system Interface **2302**, first search manager **2304**, nth search manager **2306**, first analytics agent **2308**, nth analytics agent **2310**, first search conductor **2312**, nth search conductor **2314**, partitioner **2316**, first collection **2318**, nth collection **2320**, supervisor **2322**, and dependency manager **2324**.

[0325] In one or more embodiments, system interface **2302** may be configured to feed one or more queries generated outside of the system architecture of MEMDB **2300** to one or more search managers in a first cluster including at least a first search manager **2304** and up to nth search manager **2306**. Said one or more search managers in said first cluster may be linked to one or more analytics agents in a second cluster including at least a first analytics agent **2308** and up to nth analytics agent **2310**.

[0326] Search managers in said first cluster may be linked to one or more search conductors in a third cluster including at least a first search conductor **2312** and up to nth search conductor **2314**. Search conductors in said third cluster may be linked to one or more partitioners **2316**, where partitions corresponding to at least a First Collection **2318** and up to nth Collection **2320** may be stored at one or more moments in time.

[0327] One or more nodes, modules, or suitable combination thereof included in the clusters included in MEMDB **2300** may be linked to one or more supervisors **2322**, where said one or more nodes, modules, or suitable combinations in said clusters may be configured to send at least one heartbeat to one or more supervisors **2322**. Supervisor **2322** may be linked to one or more dependency managers **2324**, where said one or more dependency managers **2324** may include one or more dependency trees for one or more modules, partitions, or suitable combinations thereof. Supervisor **2322** may additionally be linked to one or more other supervisors **2322**, where additional supervisors **2322** may be linked to said clusters included in the system architecture of MEMDB **2300**.

[0328] **FIG. 24** is a flow chart describing a method **2400** for non-exclusionary searching, according to an embodiment. Method **2400** for non-exclusionary searching may allow the system to execute searches and bring back results from records where fields specified in the query are not populated or defined in the records being searched.

[0329] The process may start with query received by search manager **2402**, in which one or more queries generated by an external source may be received by one or more search managers. In some embodiments, these queries may be automatically generated by a system interface **2302** as a response to an interaction with a user. In one or more embodiments, the queries may be represented in a markup language, or other suitable language, including XML, JavaScript, HTML, other suitable language for representing parameters of search queries. In one or more other embodiments, the queries may be represented in a structure, including embodiments where the queries are represented in YAML or JSON. In some embodiments, a query may be represented in compact or binary format.

[0330] Afterwards, the received queries may be parsed by search managers **2404**. This process may allow the system to determine if field processing is desired **2406**. In one or more embodiments, the system may be capable of determining if the process is required using information included in the query. In one or more other embodiments, the one or more search managers may automatically determine which one or more fields may undergo a desired processing.

[0331] If the system determined that field processing for the one or more fields is desired, the one or more search managers may apply one or more suitable processing techniques to the one or more desired fields, during search manager processes fields **2408**. In one or more embodiments, suitable processing techniques may include address standardization, geographic proximity or boundaries, and nickname interpretation, amongst others. In some embodiments, suitable processing techniques may include the extraction of prefixes from strings and the generation of non-literal keys that may later be used to apply fuzzy matching techniques.

[0332] Then, when S.M. constructs search query **2410**, one or more search managers may construct one or more search conductor queries associated with the one or more queries. In one or more embodiments, the search conductor queries may be constructed so as to be processed as a stack-based search.

[0333] Subsequently, S.M. may send search conductor queries to S.C. **2412**. In some embodiments, one or more search managers may send the one or more search queries to one or more search conductors, where said one or more search conductors may be associated with collections specified in the one or more search queries.

[0334] Then, the one or more Search Conductors may apply any suitable Boolean search operators **2414** (e.g., AND, OR, XOR) and index look-ups without excluding records based on not having specific fields present. The Search Conductor may execute the user-provided or application-provided Boolean operators and index look-ups. Thus, embodiments may execute user queries implementing fuzzy-indexes and 'OR' operators, instead of 'AND' operators, to get a candidate set of records that do not “exclude” potentially good results. Scoring features allow the best results (i.e., most relevant) to score highest, and the less-relevant records to score lower. In some cases, there are two stages to executing search queries. A search stage, in which Boolean operators, fuzzy indexes and filters may return a candidate set of results of potential results satisfying the search query. A next scoring stage may apply one or more user-specified or application-specified scoring methods to score the records in the candidate set, so the best results score high; poorer or less-relevant results below a given threshold can be excluded, so as to return only a reasonable result size. This may lead to having a very large candidate set of records that need to be scored, however in-memory database systems may be fast enough to handle sets of search results having sizes that may be too large in some cases for conventional systems. And the result is we don't

miss good results just because some fields were empty or there was some noisy or erroneous data.

[0335] As mentioned, the Search Conductors may apply any suitable search filters **2416** while not excluding records based on missing fields. The one or more search conductors may score **2418** the resulting answer set records against the one or more queries, where the search conductors may score the match of one or more fields of the records and may then determine a score for the overall match of the records. The search conductors may be capable of scoring records against one or more queries, where the queries include fields that are omitted or not included in the records. In some embodiments, a search manager may send a query to a search conductor to be performed on a collection with a schema including less or different fields than those defined in the query. In this case the query may be reformed to modify those fields which do conform to the schema of the collection being searched to indicate they are there for scoring purpose only. In some cases, search manager can generate and/or modify the search query. That is, the Search Manger may builds a query plan that may be tailored or adjusted to account for missing fields, or fields that may not have an index defined in one or more collections.

[0336] According to some embodiments, collections with a schema different than that of the query may not be excluded, the available fields may be scored against the queries and a penalty or lower score may be assigned to records with missing fields. The fields in collections across MEMDB **2300** may be normalized and each search conductor may have access to a dictionary of normalized fields to facilitate the score assignment process. Normalization may be performed through any suitable manual or automated process. If the user or application providing the search query defines fields that are normalized across multiple collections, the system may build queries that can be applied across multiple

collections, even if each respective collection does not conform to the exact same schema or storage rules.

[0337] In some embodiments, fuzzy matching techniques may be applied to further broaden the lists of possible relevant results.

[0338] The system may determine whether the assigned score is above a specified acceptance threshold, where the threshold may be defined in the search query or may be a default value. In one or more embodiments, the default score thresholds may vary according to the one or more fields being scored. If the search conductor determines in that the scores are above the desired threshold, the records may be added to a results list. The search conductor may continue to score records until it determines that a record is the last in the current result set. If the search conductor determines that the last record in a partition has been processed, the search conductor may then sort the resulting results list. The search conductor may then return the results list to a search manager.

[0339] When S.M. receives and collates results from S.C.'s **2420**, the one or more search conductors return the one or more search results to the one or more search managers; where, in one or more embodiments, said one or more search results may be returned asynchronously. The one or more search managers may then compile results from the one or more search conductors into one or more results list.

[0340] The system may determine whether analytics processing **2422** of the search results compiled by the one or more search managers is desired. In one or more embodiments, the system determines if the processing is desired using information included in the query. In one or more other embodiments, the one or more search managers may automatically determine which one or more fields may undergo a desired processing.

[0341] If the system determines that analytics processing **2422** is desired, one or more analytics agents may process results **2424**, through the application of one or more suitable processing techniques to the one or more results list. In one or more embodiments, suitable techniques may include rolling up several records into a more complete record, performing one or more analytics on the results, and determining information about neighboring records, amongst others. In some embodiments, analytics agents may include disambiguation modules, linking modules, link on-the-fly modules, or any other suitable modules and algorithms.

[0342] After processing, according to some embodiments, the one or more analytics agents may return one or more processed results lists to the one or more search managers.

[0343] A search manager may return search results **2426**. In some embodiments, the one or more search managers may decompress the one or more results list and return them to the system that initiated the query. The returned results may be formatted in one of several formats, including XML, JSON, RDF or any other format.

[0344] **FIG. 25** shows Compression Apparatus **2500** including Storage Unit **2502**, RAM **2504**, and CPU **2506**.

[0345] In one or more embodiments, one or more of a collection of data records, one or more schema, one or more dictionaries, one or more n-gram tables, and one or more token tables may be stored in a hardware Storage Unit **2502** in Compression Apparatus **2500**. RAM **2504** in Compression Apparatus **2500** may have loaded into it any data stored in Storage Unit **2502**, as well as any suitable modules, including Fragmentation Modules, Compression Modules, and Indexing Modules, amongst others. In one or more embodiments, Compression Apparatus **2500** may include one or more suitable CPUs **2506**,

[0346] **FIG. 26** shows Collection Data Table **2600**.

[0347] In one or more embodiments, one or more collections may include structured or semi-structured data as shown in Collection Data Table **2600**. In one or more embodiments, the structured data may contain any number of fields, and the semi-structured data, such as data represented using JSON, BSON, YAML or any other suitable format, may contain that may include any suitable number of fields, arrays, or objects. Collections may be described using any suitable schema, where suitable schema may define the data structure and the compression method used for one or more fields in the schema.

[0348] In one or more embodiments, one or more fields may include data values that may have a semantic similarity. In one or more embodiments, semantically similar data may include first names, last names, date of birth, and citizenship, amongst others. In one or more embodiments, a compression apparatus may compress one or more fields using one or more methods suitable for compressing the type of data stored in the field, where the compression apparatus may use custom token tables. In one or more embodiments, a compression apparatus may use n-gram compression as a default compression method for any number of fields with data not associated with a desired method of compression.

[0349] In one or more embodiments, one or more data in one or more fields of a collection may include data that may be better compressed after fragmentation. This type of data is typically where fields have multiple values per record, and a compression apparatus may better achieve matching and scoring by de-normalizing those records into multiple record fragments. Examples of data suitable for fragmentation may include full names, addresses, phone numbers and emails, amongst others. In one or more embodiments, a compression apparatus may fragment one or more data prior to compression. A compression apparatus may store fragmented data contiguously in the same partition. In one or more

embodiments, a compression apparatus may use fragmented record identifiers to identify which record they were fragmented from to ensure the system remains aware that the records originate from the same original record in the collection.

[0350] In one or more embodiments, a record may contain an array of data values. Arrays may contain zero or more values and array values may have a null value to represent a missing value while preserving the proper order of values.

[0351] In one or more embodiments, a compression apparatus may group one or more data fields as an object. Objects may contain other objects and may be elements in an array. A compression apparatus may further compress objects within a record by including a value that refers the system to another object in the partition with identical values. When a module may output data to other modules in the system, the module may replace the referring object with the actual object values.

[0352] In one or more embodiments, a compression apparatus may compress one or more data in fields representing numbers using known binary compression methods.

[0353] In one or more embodiments, a compression apparatus may compress one or more data in fields representing dates using known Serial Day Number compressions algorithms.

[0354] In one or more embodiments, a compression apparatus may normalize one or more data prior to compression. Data suitable for normalization prior to compression may include street suffixes and prefixes, name suffixes and prefixes, and post/pre directional information (i.e. east, north, west, amongst others), amongst others.

[0355] **FIG. 27** shows Token Table **2700**.

[0356] In one or more embodiments, a compression apparatus may compress fields including data with a suitably semantic similarity using any suitable token table, where suitable token tables may be similar to Token Table 2700.

[0357] In one or more embodiments, when one or more records may be added to a field with an associated token table the system determines whether the data may match previously encountered data in the token table. In one or more embodiments, if the data does not match, the system may use an alternate compression method instead of token tables. In one or more other embodiments, if the data does not match, the system may update its token table so as to include the data.

[0358] In one or more embodiments, the token table may be updated periodically and stored data may be re-evaluated to determine if compressibility has improved. If the compressibility of one or more data has improved, the system may decompress and re-compress any suitable data.

[0359] In one or more embodiments, the most frequently occurring values may be stored in the lower numbered indices, which may allow for the most frequently used values to be represented with fewer bytes.

[0360] In one or more embodiments, a longer value may be preferred over a shorter value for inclusion in the token table, which may allow for greater compression by eliminating longer values with the same index size as a smaller value.

[0361] In one or more embodiments a special index value may be reserved to indicate that no token data exists for the data value.

[0362] FIG. 28 shows N-gram Table 2800.

[0363] In one or more embodiments, a compression apparatus may compress fields including data with a suitably semantic similarity using any suitable n-gram table, where suitable n-gram tables may be similar to N-gram Table **2800**.

[0364] In one or more embodiments, when one or more records may be added to a field with an associated n-gram table the system determines whether the data may match previously encountered data in the n-gram table. In one or more embodiments, if the data does not match, the system may use an alternate compression method instead of n-gram tables. In one or more other embodiments, if the data does not match, the system may update its n-gram table so as to include the data.

[0365] In one or more embodiments, the n-gram table may be updated periodically and stored data may be re-evaluated to determine if compressibility has improved. If the compressibility of one or more data has improved, the system may decompress and re-compress any suitable data.

[0366] In one or more embodiments, the most frequently occurring values may be stored in the lower numbered indices, which may allow for the most frequently used values to be represented with fewer bytes.

[0367] In one or more embodiments a special index value may be reserved to indicate that no n-gram data exists for the data value.

[0368] **FIG. 29** shows Record Representation **2900**, which may represent compressed data in one or more embodiments.

[0369] In Record Representation **2900**, each row value in the record index column may include zero or more record descriptor bytes with information about the record, including the length, offset, or the record's location in memory amongst others. In one or

more embodiments, each data node (array, field, or object) present in the record may include zero or more descriptor bytes, where suitable information about the node may be included, including a node identifier, the length of the stored data, and number of elements of the array if applicable. Following the zero or more node descriptor bytes, any suitable number of bytes may represent the data associated with the record. In one or more embodiments, the data may include one or more bits describing the contents of the data including array separation marker bits.

[0370] In one or more embodiments, data in a field associated with a token table may use one or more bits to state whether the information stored in the record is represented in a suitable Token Table, or whether another suitable compression method, such as N-gram compression, was used.

[0371] In one or more embodiments, a system may use length or offset data included in the one or more record descriptor bytes and/or the one or more node (array, object, or field) descriptor bytes to navigate through the compressed data without decompressing the records or nodes (arrays, objects, or fields).

[0372] In one or more embodiments, any suitable module in a system may index or compress data, including one or more search conductors or one or more partitioners in a MEMDB system.

[0373] In one or more embodiments, a compression apparatus employing one or more compression methods here disclosed allow data to be compressed at rates similar to other prominent compression methods while allowing data to be decompressed and/or accessed at the node (array, object, or field) level.

[0374] In one or more embodiments, a compression apparatus employing one or more compression methods here disclosed allow the system to skip individual records and nodes (arrays, objects, or fields) when accessing information in the records.

[0375] In one or more embodiments, a compression apparatus employing one or more compression methods here disclosed allow the system to exit decompression of a record early when the target fields are found.

[0376] In **Example #1**, the disclosed method for faceted searching is applied. The MEMDB analyses documents from a large corpus, extracts facets, disambiguates and indexes the extracted facets and the stores them in different partitions of more than two collections according to the facet type and hierarchy. In **example # 1**, a user types the word "united" in a search box and the system returns the search results by facets. Level one facets include "Class", "Location", "Product", "Technology" and "Company", amongst others. The number of hits for each level 2 facet is shown and the user is able to narrow down the search at least 3 more levels.

[0377] In **example # 2**, the disclosed method for faceted searching is applied. The MEMDB analyses documents from a large corpus, extracts facets, disambiguates and indexes the extracted facets and the stores them in different partitions of more than two collections according to the facet type and hierarchy. In **example # 2**, a user types the characters "ply" in a search box and the system automatically generates search suggestions by facets. Level one facets include "Class", "Location", "Product", "Technology" and "Company", amongst others. The number of hits for each level 2 facet is shown and the user is able to narrow down the search at least 3 more levels.

[0378] **Example #1** is an in-memory database system including a search manager, an analytics agent, node managers on each node, eight search nodes each having two search

conductors, a supervisor, a backup supervisor, a dependency manager, a backup dependency manager, and a partitioner on a node able to store and distribute partitions (where the node includes information for two collections split into four partitions each, collection 1 and collection 2). When a search query for records in collection 1 is received by the database, the search manager sends a query to all the search conductors having the partitioner associated with collection 1. The search conductors work asynchronously to search and score each compressed record, make a list of compressed results having a score above the threshold defined in the query, sort the list of results and return the list of compressed records to the search manager. In this example, the search conductors decompress only the fields that are to be scored. The search manager receives and aggregates the list of results from each search conductor, compiles the query result, and sends it to analytics agent for further processing. The analytics agent combines records it determines are sufficiently related, and returns the processed list of results to the search manager. The search manager then returns the final results through the system interface.

[0379] **Example #2** is an in-memory database that can perform semantic queries and return linked data results on data that is not explicitly linked in the database. Data or record linking is just one example of an aggregate analytical function that may be implemented in an Analytics Agent. This example is an in-memory database with an analytics agent capable of discovering data linkages in unlinked data and performing semantic queries and returning semantic results. Unlinked data is data from disparate data sources that has no explicit key or other explicit link to data from other data sources. In this example, a pluggable analytics module could be developed and deployed in an Analytics Agent to discover/find data linkages across disparate data sources, based on the data content itself. When a semantic search query is executed, all relevant records are retrieved via search conductors, using non-exclusionary searches, and sent to an analytics agent where record linkages are discovered,

based on the specific implementation of the analytics agent module, and confidence scores assigned. These dynamically linked records can be represented using semantic markup such as RDF/XML or other semantic data representation and returned to the user. This approach to semantic search allows unlinked data to be linked in different ways for different queries using the same unlinked data.

[0380] **Example #3** is an in-memory database that can perform graph queries and return linked data results on data that is not explicitly linked or represented in graph form in the database. This example is an in-memory database with an analytics agent capable of discovering data linkages in unlinked data and performing graph queries and returning graph query results. When a graph search query is executed, all relevant records are retrieved via search conductors, using non-exclusionary searches, and sent to an analytics agent where record linkages are discovered and confidence scores assigned. These dynamically linked records can be represented in graph form such as an RDF Graph, Property Graph or other graph data representation and returned to the user. This approach to graph search allows unlinked data to be linked in different ways for different queries using the same unlinked data.

[0381] **Example #4** is a system hosting an in-memory database with connections set up in a manner similar to that described in **FIG. 14**. Search Managers, Search Conductors and Analytics Agents are all directly participating in the flow of an interactive user query. To minimize the latency of the user query, these modules are connected with the lowest latency connections. Search Managers and Analytics Agents work with the larger aggregated answer sets and benefit from the greatest bandwidth, where as the Search Conductors deal with the hundreds of partition based answer set components which require less bandwidth. Partitioners deal with large data volumes but at non-interactive speeds so they have both moderate latency and moderate bandwidth connections. Supervisors and Dependency managers are non-

interactive and low data volume and so require lowest bandwidth and the highest latency connections. This configuration attempts to minimize cost based on actual need.

[0382] Therefore, in this example, Line Type A is an InfiniBand connection with a 40 Gb bandwidth and a latency of 1 microsecond or less; Line Type B is an InfiniBand connection with a 20 Gb bandwidth and a latency of 1 microsecond or less; Line Type C is a 10 Gb Ethernet connection; and Line Type D is a 100 Mb Ethernet connection. In this example, nodes including a search manager include CPUs able to operate at 2 Teraflops; nodes including a search conductor include CPUs able to operate at 4 Teraflops; nodes including an analytics agent include CPUs able to operate at 4 Teraflops; and nodes including a partitioner include CPUs able to operate at 6 Teraflops. In this example, nodes including a search conductor include 32 to 64 GB of RAM, nodes including an analytics agent include 32 to 64 GB of RAM, and 6 nodes including a partitioner each include 64 GB of RAM and a 10,000RPM hard disk.

[0383] **Example #5** is a system hosting in-memory database with connections set up in a manner similar to that described in **FIG. 14**. Search Managers, Search Conductors and Analytics Agents are all directly participating in the flow of interactive user queries and data inserts. To separate high-volume, backend data insert network traffic from interactive, low-latency user queries, modules are connected using different network tiers. This configuration allows for responsive, interactive user queries by utilizing a low-latency network tier, such as InfiniBand, while also allowing high-volume data inserts utilizing a separate high-bandwidth network tier. Both types of operations run optimally without interfering with each other.

[0384] **Example #6** illustrates what happens if a single module fails due to some resource no longer available on the node but the node itself is not otherwise adversely affected.

[0385] In this case, when the module fails the heartbeat connections to the supervisor are dropped, thereby alerting the supervisor to the module failure. The supervisor will attempt to reconnect to the module to check if the failure was just a connection issue or a module failure. In some embodiments, failure to reconnect is assumed to be a module failure.

[0386] The supervisor will first request the associated node manager to restart the module in place. Starting the module in place does not incur the cost of re-staging the module and any corresponding software or data, so can be accomplished more quickly than staging and starting on another node. However, in this example the problem is due to some resource unavailability on the specified node, thus the restart will also fail.

[0387] After making a predetermined number of attempts to restart the module in place, the supervisor will look for another suitable node to start the module on. The supervisor will contact a dependency manager to acquire the correct package required to deploy the failed module. The supervisor will then pass that package on to the node manager for the newly selected node to stage and run the module. The module finds the required resources on the new node and creates a heartbeat connection to the supervisor indicating it is running properly. The supervisor marks the functionality as restored and the event is over.

[0388] **Example #7** illustrates a total node fail such as a failed power supply. In this case the node manager and all modules on the server drop their heartbeat connections to the supervisor. The supervisor recognizes this as a complete node failure and marks that node as failed and unavailable. The supervisor then walks through the list of modules that were allocated to that node. For each module in that list the supervisor will look for another suitable node to start the module on. The supervisor will contact a dependency manager to acquire the correct package required to deploy the current module. The supervisor will then pass that package on to the node manager for the newly selected node to stage and run the

module. The module executes and creates a heartbeat connection to the supervisor indicating it is running properly. The supervisor marks the functionality as restored for that module. This continues until all modules have been reallocated to new nodes and the event is over.

[0389] In **Example #8**, a system hosts an in-memory database, similar to the one described in **FIG. 21**. In this example, the in-memory database and system includes a plurality of analytics modules. One analytics module may implement record linking utilizing a weighted model while another uses decision trees. Some modules may be optimized to operate on any available data, while others are tuned to produce desired results from a restricted set of fields or data collections. Some modules were developed and uploaded by different user groups. Each user query can specify different analytics modules be applied and use different parameters for said modules. It is possible for different users to use the in-memory database to extract information at the same time and even process the same data in several different ways at the same time. It is also possible for some users to plug-in new analytics modules at any time, without affecting the performance of the in-memory database or the experience of other users.

[0390] In **Example #9**, the disclosed method for non-exclusionary searching is applied. A user defines a query with the following fields: FN (first name): John, LN (last name): Smith, DOB (date of birth): 05/15/1965 and PH (phone number): 555-1234-7890. The system performs the search and among the relevant results there are two records with missing fields, from two different collections with different schemata. The first one is from collection 1001, in this collection the following fields are defined FN: John, LN: Smith, PH: --- and DOB: 05/15/1965. The second one is from collection '8021,' in this collection the following fields are defined FN: John, LN: Smith, PH: 555-1234-7890 and DOB: ---. Since there is a good match in most fields both of the records, neither is excluded, and they get a similar final score and are positioned in the top 10 results for the query.

[0391] In **Example #10**, the disclosed method for non-exclusionary searching is applied. A user defines a query with the following fields: FN (first name): John, LN (last name): Smith, DOB (date of birth): 05/15/1965 and PH (phone number): 555-1234-7890. The system performs the search and among the relevant results there are two records with similar but not exactly matched fields, from two different collections with different schemata. The first one is from collection **1001**, in this collection the following fields are defined FN: Jonathan, LN: Smith, PH: 1234-7890. The second one is from collection **8021**, in this collection the following fields are defined FN: John, LN: Smyth, PH: 555-1234-7890 and DOB: 1965. Since there is a good match in most fields both of the records get a final score that exceeds the score threshold and are positioned in the top 10 results for the query.

[0392] **Example #11** illustrates a method for compressing names using a compression apparatus. In this example, a data set includes a collection including one million full name records with 350 unique first names and 300 unique last names represented. The records were fragmented into a first name field and a last name field.

[0393] The individual tokens were then weighted via the product of their frequency and length from highest to lowest. Tokens with a weight less than a certain threshold were discarded to reduce the token table size.

[0394] A token table was then generated for each field by maximizing the aggregate space savings in assigning indices whereby space savings for an individual token is the product of frequency and the sum of its length minus stored index length.

[0395] The number of entries associated with single byte indices was varied from 1 to 255 to inclusive during the maximization procedure.

[0396] The algorithm guarantees that the generated token table is optimal, and the highest savings will go to the single byte stored index entries while subsequent values compress to two or more bytes. Short or infrequent entries may realize no savings and are not included in the token table. These values revert to another compression method such as n-gram compression.

[0397] **Example #12** illustrates a method for compressing text using a compression apparatus.

[0398] In this example, a large body of text was analyzed for frequency of n-grams where n-grams can represent successive sequences of characters, words, or groups of words. The text is usually acquired via analyzing a large column of field data in order to achieve columnar compression results in a field by field horizontal compression.

[0399] The individual n-grams were then weighted via the product of their frequency and length from highest to lowest. N-grams with a weight less than a certain threshold were discarded to reduce the n-gram table size.

[0400] A n-gram table was then generated for the field by maximizing the aggregate space savings in assigning indices whereby space savings for an individual n-gram is the product of frequency and the sum of its length minus stored index length.

[0401] The number of entries associated with single byte indices was varied from 1 to 255 to inclusive during the maximization procedure.

[0402] The algorithm guarantees that the generated n-gram table is optimal, and the highest savings will go to the single byte stored index entries while subsequent values compress to two or more bytes. Infrequent entries may realize no savings and are not included in the n-gram table. These values revert to some other method of basic storage.

[0403] An example of some of the n-grams generated in the table via this method is as follows:

Index	N-gram
1	e
11	th
47	and
800	out of

[0404] During compression the field data is compressed from beginning to end using a greedy algorithm to replace the largest amount of data as possible with an indexed value until the entire data is consumed.

[0405] **Example #13** is a method for compressing semi-structured data in JSON documents using a compression apparatus.

[0406] In this example, JSON input documents are compressed using the following schema, with token table compression for Title, FirstName, LastName, NameSuffix and PhoneType fields, Serial Day Number compression for DateOfBirth field and number n-gram compression for PhoneNumber field:

```
[0407] { "collection": "Example2",
"record": [
{"id": "1", "type": "FIELD", "name": "Title", "compressionMethod":
"TitleNameTokenTable"},
{"id": "2", "type": "FIELD", "name": "FirstName", "compressionMethod":
"FirstNameTokenTable"},
{"id": "3", "type": "FIELD", "name": "LastName", "compressionMethod":
"LastNameTokenTable"},
{"id": "4", "type": "FIELD", "name": "NameSuffix", "compressionMethod":
"NameSuffixNameTokenTable"},
```

```

{"id": "5", "type": "FIELD", "name": "DateOfBirth", "compressionMethod":
"SerialDayNumber"},
{"id": "6", "type": "ARRAY", "name": "Phones", "array": [
{"id": "7", "type": "STRUCT", "struct": [
{"id": "8", "type": "FIELD", "name": "PhoneType", "compressionMethod":
"PhoneTypeTokenTable"},
{"id": "9", "type": "FIELD", "name": "PhoneNumber", "compressionMethod":
"NumberNgramTable"},
]}
]}
]}

```

[0408] The input record below requires 266 bytes to be represented in JSON (after removing unnecessary whitespace). After compressing, using the compression methods described in the schema above, the resulting compressed record requires only 44 bytes.

```

[0409]      { "Title": "Dr.",
"FirstName": "Johnathan",
"LastName": "Patterson",
"NameSuffix": "M.D.",
"DateOfBirth": "19600101",
"Phones": [
{"PhoneType": "Office", "PhoneNumber": "1234567890"},
{"PhoneType": "Mobile", "PhoneNumber": "2345678901"},
{"PhoneType": "Home", "PhoneNumber": "3456789012"}
]}

```

[0410] The input record below requires 108 bytes to be represented in JSON (after removing unnecessary whitespace). After compressing, using the compression methods described in the schema above, the resulting compressed record requires only 13 bytes.

```
[0411]    { "Title": "Dr.",
"FirstName": "Johnathan",
"LastName": "Patterson",
"NameSuffix": "M.D.",
"DateOfBirth": "19600101"
}
```

[0412] **Example #14** is an example of fragmenting a record. In this example, the 53rd record of a collection includes data for a couple, Bob and Carol Wilson, having a first and second address. In this example, the record is fragmented as shown in the following table.

	Record Index	Name	Address
Original	53	Bob and Carol Wilson	Address 1, Address 2
Fragmented	53.1	Bob	Address 1
	53.2	Bob	Address 2
	53.3	Carol	Address 1
	53.4	Carol	Address 2

[0413] The record index is maintained to ensure the system remains aware that the records originate from the same original record in the collection. In this example, the fragmented records further compress the data by including a value that refers the system to the previous record in the partition, i.e. when the system accesses record the name of record 53.2, the value refers the system back to the value for the name in record 53.1. When the system in **Example #14** outputs data to other modules in the system, even in compressed format, the module replaces the referring values for the actual values.

[0414] **Example #15** is an example of compression for archiving semi-structured data. In this example, JSON documents from a document oriented database such as MongoDB, Cassandra, or CouchDB are compressed using a schema that defines all the desired fields, including the unique identifier of each JSON document. An index is then created that maps the unique identifier to the compressed record. The resulting compressed records and index consume less than 15% of the storage required for the original document-oriented database and each JSON document or select fields of a document can be immediately accessed without decompressing unwanted data.

[0415] The various illustrative logical blocks, modules, circuits, and algorithm steps described in connection with the embodiments disclosed herein may be implemented as electronic hardware, computer software, or combinations of both. To clearly illustrate this interchangeability of hardware and software, various illustrative components, blocks, modules, circuits, and steps have been described above generally in terms of their functionality. Whether such functionality is implemented as hardware or software depends upon the particular application and design constraints imposed on the overall system. Skilled artisans may implement the described functionality in varying ways for each particular application, but such implementation decisions should not be interpreted as causing a departure from the scope of the present invention.

[0416] Embodiments implemented in computer software may be implemented in software, firmware, middleware, microcode, GPUs, hardware description languages, or any combination thereof. A code segment or machine-executable instructions may represent a procedure, a function, a subprogram, a program, a routine, a subroutine, a module, a software package, a class, or any combination of instructions, data structures, or program statements. A code segment may be coupled to another code segment or a hardware circuit by passing and/or receiving information, data, arguments, parameters, or memory contents. Information,

arguments, parameters, data, etc. may be passed, forwarded, or transmitted via any suitable means including memory sharing, message passing, token passing, network transmission, etc.

[0417] The actual software code or specialized control hardware used to implement these systems and methods is not limiting of the invention. Thus, the operation and behavior of the systems and methods were described without reference to the specific software code being understood that software and control hardware can be designed to implement the systems and methods based on the description herein.

[0418] When implemented in software, the functions may be stored as one or more instructions or code on a non-transitory computer-readable or processor-readable storage medium. The steps of a method or algorithm disclosed herein may be embodied in a processor-executable software module which may reside on a computer-readable or processor-readable storage medium. A non-transitory computer-readable or processor-readable media includes both computer storage media and tangible storage media that facilitate transfer of a computer program from one place to another. A non-transitory processor-readable storage media may be any available media that may be accessed by a computer. By way of example, and not limitation, such non-transitory processor-readable media may comprise RAM, ROM, EEPROM, CD-ROM or other optical disk storage, magnetic disk storage or other magnetic storage devices, or any other tangible storage medium that may be used to store desired program code in the form of instructions or data structures and that may be accessed by a computer or processor. Disk and disc, as used herein, include compact disc (CD), laser disc, optical disc, digital versatile disc (DVD), floppy disk, and blu-ray disc where disks usually reproduce data magnetically, while discs reproduce data optically with lasers. Combinations of the above should also be included within the scope of computer-readable media. Additionally, the operations of a method or algorithm may reside as one or any combination or set of codes and/or instructions on a non-

transitory processor-readable medium and/or computer-readable medium, which may be incorporated into a computer program product.

[0419] The preceding description of the disclosed embodiments is provided to enable any person skilled in the art to make or use the present invention. Various modifications to these embodiments will be readily apparent to those skilled in the art, and the generic principles defined herein may be applied to other embodiments without departing from the spirit or scope of the invention. Thus, the present invention is not intended to be limited to the embodiments shown herein but is to be accorded the widest scope consistent with the following claims and the principles and novel features disclosed herein.

[0420] While various aspects and embodiments have been disclosed, other aspects and embodiments are contemplated. The various aspects and embodiments disclosed are for purposes of illustration and are not intended to be limiting, with the true scope and spirit being indicated by the following claims.

[0421] The foregoing method descriptions and the process flow diagrams are provided merely as illustrative examples and are not intended to require or imply that the steps of the various embodiments must be performed in the order presented. As will be appreciated by one of skill in the art the steps in the foregoing embodiments may be performed in any order. Words such as “then,” “next,” etc. are not intended to limit the order of the steps; these words are simply used to guide the reader through the description of the methods. Although process flow diagrams may describe the operations as a sequential process, many of the operations can be performed in parallel or concurrently. In addition, the order of the operations may be re-arranged. A process may correspond to a method, a function, a procedure, a subroutine, a subprogram, etc. When a process corresponds to a

function, its termination may correspond to a return of the function to the calling function or the main function.

[0422] The various illustrative logical blocks, modules, circuits, and algorithm steps described in connection with the embodiments disclosed herein may be implemented as electronic hardware, computer software, or combinations of both. To clearly illustrate this interchangeability of hardware and software, various illustrative components, blocks, modules, circuits, and steps have been described above generally in terms of their functionality. Whether such functionality is implemented as hardware or software depends upon the particular application and design constraints imposed on the overall system. Skilled artisans may implement the described functionality in varying ways for each particular application, but such implementation decisions should not be interpreted as causing a departure from the scope of the present invention.

[0423] Embodiments implemented in computer software may be implemented in software, firmware, middleware, microcode, hardware description languages, or any combination thereof. A code segment or machine-executable instructions may represent a procedure, a function, a subprogram, a program, a routine, a subroutine, a module, a software package, a class, or any combination of instructions, data structures, or program statements. A code segment may be coupled to another code segment or a hardware circuit by passing and/or receiving information, data, arguments, parameters, or memory contents. Information, arguments, parameters, data, etc. may be passed, forwarded, or transmitted via any suitable means including memory sharing, message passing, token passing, network transmission, etc.

[0424] The actual software code or specialized control hardware used to implement these systems and methods is not limiting of the invention. Thus, the operation and behavior of the systems and methods were described without reference to the specific software code

being understood that software and control hardware can be designed to implement the systems and methods based on the description herein.

[0425] When implemented in software, the functions may be stored as one or more instructions or code on a non-transitory computer-readable or processor-readable storage medium. The steps of a method or algorithm disclosed herein may be embodied in a processor-executable software module which may reside on a computer-readable or processor-readable storage medium. A non-transitory computer-readable or processor-readable media includes both computer storage media and tangible storage media that facilitate transfer of a computer program from one place to another. A non-transitory processor-readable storage media may be any available media that may be accessed by a computer. By way of example, and not limitation, such non-transitory processor-readable media may comprise RAM, ROM, EEPROM, CD-ROM or other optical disk storage, magnetic disk storage or other magnetic storage devices, or any other tangible storage medium that may be used to store desired program code in the form of instructions or data structures and that may be accessed by a computer or processor. Disk and disc, as used herein, include compact disc (CD), laser disc, optical disc, digital versatile disc (DVD), floppy disk, and blu-ray disc where disks usually reproduce data magnetically, while discs reproduce data optically with lasers. Combinations of the above should also be included within the scope of computer-readable media. Additionally, the operations of a method or algorithm may reside as one or any combination or set of codes and/or instructions on a non-transitory processor-readable medium and/or computer-readable medium, which may be incorporated into a computer program product.

[0426] It is to be appreciated that the various components of the technology can be located at distant portions of a distributed network and/or the Internet, or within a dedicated secure, unsecured and/or encrypted system. Thus, it should be appreciated that the

components of the system can be combined into one or more devices or co-located on a particular node of a distributed network, such as a telecommunications network. As will be appreciated from the description, and for reasons of computational efficiency, the components of the system can be arranged at any location within a distributed network without affecting the operation of the system. Moreover, the components could be embedded in a dedicated machine.

[0427] Furthermore, it should be appreciated that the various links connecting the elements can be wired or wireless links, or any combination thereof, or any other known or later developed element(s) that is capable of supplying and/or communicating data to and from the connected elements. The term module as used herein can refer to any known or later developed hardware, software, firmware, or combination thereof that is capable of performing the functionality associated with that element. The terms determine, calculate and compute, and variations thereof, as used herein are used interchangeably and include any type of methodology, process, mathematical operation or technique.

[0428] The preceding description of the disclosed embodiments is provided to enable any person skilled in the art to make or use the present invention. Various modifications to these embodiments will be readily apparent to those skilled in the art, and the generic principles defined herein may be applied to other embodiments without departing from the spirit or scope of the invention. Thus, the present invention is not intended to be limited to the embodiments shown herein but is to be accorded the widest scope consistent with the following claims and the principles and novel features disclosed herein.

[0429] The embodiments described above are intended to be exemplary. One skilled in the art recognizes that numerous alternative components and embodiments that may be

substituted for the particular examples described herein and still fall within the scope of the invention.

CLAIMS

What is claimed is:

1. A computer-implemented method comprising:
 - extracting, by a computer, one or more facets from a corpus comprising data of one or more types of data and containing information;
 - disambiguating, by the computer, each of the one or more facets extracted from the corpus;
 - generating, by the computer, one or more indices associated with the one or more facets respectively;
 - retrieving, by the computer, each of the facets based on the associated index from a record of partition comprising one or more records of a database, wherein a collection of the database comprises one or more partitions, and wherein each respective facet indicates a hierarchical relation of data stored in the database relative to the one or more records of data; and
 - generating, by the computer, a suggested search query based on each of the facets.
2. A computing system hosting an in-memory database, the system comprising:
 - a partitioner node comprising a processor configured to, in response to receiving a collection of one or more records of a database, determine whether to compress the collection based on a machine-readable schema file associated with the collection, logically partition the collection into one or more partitions according to the schema file, and distribute the one or more partitions to one or more storage nodes according to the schema file;
 - a storage node comprising non-transitory machine-readable main memory storing a partition received from the partitioner associated with the storage node;
 - a search manager node comprising a processor receiving a search query from a client device of the system, and transmitting the search queries as search conductor queries to one or more search conductors in response to receive the search query from the client device, wherein the search query is a machine-readable computer file containing parameters associated with one or more records satisfying the search query;

a search conductor node associated with one or more partitioners and comprising a processor configured to, in response to receiving a search conductor query from the search manager node: query a set of one or more partitions indicated by the search conductor query, identify one or more candidate records stored in the set of queried partitions, calculate a first score for each respective candidate record using a scoring algorithm, and transmit to the search manager a set of one or more query results containing one or more candidate records satisfying a threshold value; and

an analytics agent node comprising a processor configured to automatically generate a machine-readable computer file containing a set of one or more results derived from the set of query results, responsive to identifying in the set of query results received from the search manager node.

3. The system according to claim 2, wherein the processor of the analytics agent node is further configured to transmit the set of one or more data linkages to the search manager.
4. The system according to claim 2, wherein the processor of the search manager node is further configured to execute one or more field processing algorithms in accordance with the search query.
5. The system according to claim 2, further comprising a supervisor node comprising a processor receiving one or more heartbeat signals from one or more nodes of the system and determining a status for each of the one or more nodes based on a heartbeat signal received from each respective node, wherein each of the respective heartbeat signals indicates the status of the respective node.
6. The system according to claim 5, wherein each respective node comprises a processor configured to monitor the status of the node.
7. The system according to claim 5, further comprising a dependency manager node associated with the supervisor node and comprising a processor monitoring a node configuration status of a node monitored by the supervisor using a machine-readable dependency tree file stored in a non-transitory machine-readable storage medium.
8. The system according to claim 7, wherein the status of the heartbeat signal indicates the node configuration status, and wherein the supervisor node transmits a machine-readable

configuration package file responsive to the dependency manager determining the node configuration status indicates the node is misconfigured.

9. The system according to claim 2, wherein the search conductor calculates a field score for each respective candidate record of a set of one or more updated result records, wherein the first score of each respective candidate in the set of updated result records satisfies the threshold value indicated by the search query, and transmits the updated result records to the search manager node.

10. The method according to claim 9, wherein the search conductor decompresses data stored in a candidate record in the set of updated result records using a data compression algorithm, in response to determining the data of the candidate result record is compressed according to the data compression algorithm.

11. The system according to claim 2, further comprising a node comprising a processor executing a query interface module receiving a new collection schema file associated with one or more collections, wherein at least search conductor node is configured to automatically reconfigure one or more collections associated with the search conductor according to the new schema file.

12. The system according to claim 2, wherein the partitioner assigns a unique record identifier to each of the respective records stored in the collection according to the schema file, and generates a machine-readable index file associated with each of the partitions of the collection using the unique record identifier assigned to each respective record in the collection.

13. The system according to claim 12, wherein the search manager node distributes to the search conductor node a set of one or more new records; and wherein the search conductor automatically adds each of the new records to a partition of a collection according to the schema file, responsive to receiving the set of one or more new records.

14. The system according to claim 12, wherein the one or more search managers are further configured to receive and distribute a request to delete one or more records that correspond to a set of unique record identifiers and distribute the request to at least one search conductor; and wherein the search conductor is further configured to mark for deletion each record associated with the set of unique record identifiers.

15. The system according to claim 14, wherein marking a record for deletion precludes the record from a future search results record.
16. The system according to claim 12, wherein the unique record identifier associated with each of the records comprises one or more of a unique identifier number, a collection version number, a collection name, and a partition version number.
17. The system according to claim 12, wherein the search manager node receives a set of one or more new collections comprising one or more new records, and transmits a set of new collections to the one or more search conductor node according to the schema file, and wherein each respective search conductor node, responsive to receiving the one or more new collections, automatically populates one or more collections associated with the respective search conductor node with the set of new one or more records in accordance with the schema file.
18. The system according to claim 2, wherein a search manager receives a request to remove a collection, the search manager processor is configured to forward the collection deletion request to a search conductor, and the search conductor is further configured to remove the collection from the database.
19. The system according to claim 2, wherein the search manager asynchronously receives each of the search result records from each of the search conductors.
20. The system according to claim 2, wherein the schema describes a collection according to one or more of names of the fields, whether the fields are indexed, a compression used, and a default scoring algorithm for the fields.
21. The system according to claim 2, wherein the analytics agent is further configured to concatenate several records into a more complete record and determine information about neighboring records to the search result records.
22. The system according to claim 2, wherein the search conductor limits the size of the search result records based on the search query received from the search manager.
23. The system according to claim 2, wherein a supervisor instructs a partitioner to compress one or more records in a collection.

24. The system according to claim 2, wherein a supervisor is further configured to determine one or more new partitions to be loaded, requests a node configuration for a node from a dependency manager, wherein the supervisor instructs a node manager of the node to retrieve the node configuration from the dependency manager;

wherein the node manager is configured to allocate memory resources of the node and loads a new partition; and

wherein the search conductor associated with the new partition in accordance with the schema informs the supervisor that the partition is loaded.

25. The system according to claim 2, wherein the analytics agent node identifies in the set of query results received from the search manager node a data linkage correlating two or more records, and wherein the data linkage correlates data contained in a first record associated with data contained in a second record

26. A computer implemented method comprising:

receiving, by a search manager computer of a system hosting an in-memory database, binary data representing a search query containing parameters querying the database, wherein the system comprises one or more storage nodes comprising main memory storing one or more collections of the database, wherein each collection contains one or more records;

transmitting, by the computer, the search query to one or more search conductor nodes according to the search query, wherein the search query indicates a set of one or more collections to be queried;

transmitting, by the computer, to one or more analytics agent nodes a set of search results based on the search query responsive to receiving from the one or more search conductors the set of search results containing one or more records satisfying the search query, wherein each respective record of the set of search results is associated with a score based on a scoring algorithm in the search query; and

responsive to the computer receiving a computer file containing a set of one or more data linkages from the one or more analytics agent nodes:

updating, by the computer, the one or more records of the set of search results according to the set of one or more data linkages received from the analytics agent nodes.

27. The method according to claim 26, wherein the computer asynchronously receives a subset of search results from each respective search conductor.

28. The method according to claim 27, wherein each subset of search records received from each respective search conductor node is ranked according to the score calculated for the respective record.

29. The method according to claim 26, wherein each respective search conductor associated with the set of collections to be queried determines a set of one or more search results containing the one or more records of the search results according to the parameters of the search query.

30. A computer-implemented method comprising:

receiving, by a computer, one or more collections from a search conductor according to a machine-readable schema file, wherein each of the collections comprises a set of one or more records having one or more fields;

partitioning, by the computer, each collection according to the schema;

compressing, by the computer, the records in the partition according to the schema;

and

distributing, by the computer, each of the partitions to one or more associated search conductors to include each of the partitions in each collection corresponding to the partitioner associated with the search conductor.

31. The method according to claim 30, further comprising fragmenting, by the computer, the records in each set of records according to the schema.

32. The method according to claim 30, further comprising decompressing, by the computer, the records at a level selected from the group consisting of: a field level, a record level, a partition level, a collection level, and a database level.

33. A system comprising one or more nodes hosting an in-memory database, the system comprising:

a plurality of storage nodes comprising non-transitory machine-readable storage medium storing one or more partitions of a collection, wherein the collection stored by each respective storage node contains one or more records of a database, and wherein the storage medium of each respective storage node comprises main memory;

a search manager node comprising a processor generating one or more search conductor queries using a search query received from a user node, transmitting the one or more search conductor queries to one or more search conductor nodes according to the search

query, and forward one or more sets of search results to one or more analytics agent nodes according to the search query responsive to receive the one or more sets of search results;

an analytics agent node comprising a processor executing one or more analytics algorithms responsive to receiving a set of search results from the search manager node;

a search conductor node comprising a processor querying the collection of the database records of a storage node according to a search conductor query in response to receiving the search conductor query from the search manager, and transmitting the set of one or more search results to the search manager node in response to identifying the one or more search results of the set, wherein each respective search result corresponds to a database record containing data satisfying the search conductor query; and

a plurality of network segments comprising one or more connections between one or more nodes communicating over each respective network segment, wherein a first network segment comprises the search manager, the search conductor, and the analytics agent.

34. The system according to claim 33,

wherein each respective network segment comprises the one or more connections based upon a resource demand associated with a communication type, and

wherein the communications type is selected from the group consisting of search query data, one or more collections of database records, status data, and configuration package files.

35. The system according to claim 34, wherein the resource demand of each respective communication type is determined according to a latency demand and a bandwidth demand, wherein the latency demand is based on a permissible amount of time for the communication, and wherein the bandwidth demand is based on an amount of data for the communication.

36. The system according to claim 34, wherein the first network segment comprises the one or more connections communicating search query data.

37. The system according to claim 36, wherein the first network segment comprises a physical connection using InfiniBand.

38. The system according to claim 36, further comprising one or more partitioner nodes partitioning one or more collections of database records and distributing the one or more collections to the one or more storage nodes according to a schema file responsive to receiving the one or more collections,

wherein a second network segment comprises the one or more connections communicating the one or more collections from each respective partitioner to the one or more storage nodes.

39. The system according to claim 38, wherein the second network segment comprises a second physical connection using InfiniBand.

40. The system according to claim 39, wherein the second network segment comprises a second physical connection using Ethernet.

41. The system according to claim 38, further comprising one or more supervisor nodes comprising a processor continuously monitoring one or more statuses associated with the one or more nodes of the system, and configured to transmit a configuration package file to a failed node of the one or more nodes in response to receiving a status indicating a failed resource of the failed node,

wherein a third network segment comprises the one or more connections communicating each respective status of each respective node to the one or more supervisor nodes and the configuration package file transmitted from the supervisor node to the failed node.

42. The system according to claim 41, wherein the third network segment comprises a third physical connection using Ethernet.

43. The system according to claim 41, further comprising a dependency manager comprising non-transitory machine-readable storage medium storing one or more configuration package files and a dependency tree, and a processor configured to determine the configuration package file for the failed node and transmit the configuration package file to a supervisor node based on the dependency tree in response to a request for configuration package file from the supervisor node;

wherein the third network segment further comprises the one or more connections communicating the request for the configuration package file.

44. A computer-implemented method comprising:

monitoring, by a computer comprising a processor executing a supervisor module, a heartbeat signal generated by a node manager monitoring one or more software modules

stored on a node, wherein the heartbeat signal contains binary data indicating a status of each respective software module monitored by the node manager;

detecting, by the computer, a failed software module in the one or more software modules of the node based on the heartbeat signal received from the node manager of the node;

automatically transmitting, by the computer, to the node manager of the node a command instructing the node to restore the failed software module, in response to detecting the failed software module; and

determining, by the computer, whether the node manager successfully restored the module based on the heartbeat signal received from the node manager.

45. The method according to claim 44, the method further comprising:

determining, by the computer, a failover node to execute the module when the node manager does not restore the module within a threshold number of attempts;

retrieving, by the computer, a configuration package associated with the module from a dependency manager node;

transmitting, by the computer, the configuration package to a failover node manager associated with the failover node, wherein the failover node manager attempts to install the module on the failover node, and wherein the failover node manager attempts to restore the module;

determining, by the computer, if the failover node manager successfully installed the module on the failover node; and

determining, by the computer, if the failover node manager successfully restored the module.

46. The method according to claim 45, further comprising:

determining, by the computer, a next failover node to execute the module when the failover node manager fails to install the module on the failover node or when the failover node manager fails to restore the module within a threshold number of attempts;

transmitting, by the computer, the configuration package to a next failover node manager associated with the next failover node;

determining, by the computer, if the next failover node manager successfully installs the module on the next failover node; and

determining, by the computer, if the next failover node manager successfully restores the module.

47. The method according to claim 46, further comprising generating, by the computer, a module failure alert after one or more next failover node managers exceed a global threshold number of attempts to restore the module, wherein the computer sequentially determines a next failover node until the global threshold of attempts to restore the module is met.

48. The method according to claim 44, further comprising:

detecting, by the computer, a failure of the node manager monitoring the module;

determining, by the computer, a failover node to execute the module, wherein the failover node is associated with a failover node manager;

retrieving, by the computer, a configuration package associated with the module from a dependency manager node;

transmitting, by the computer, the configuration package to the failover node manager, wherein the failover node manager attempts to install the module on the failover node, and wherein the failover node manager attempts to restore the module;

determining, by the computer, if the failover node manager successfully installs the module on the failover node; and

determining, by the computer, if the failover node manager successfully restores the module.

49. The method according to claim 48, further comprising:

determining, by the computer, that the node is a failed node when the node is not functioning according to a status quo;

determining, by the computer, one or more modules executed by the failed node to be migrated off the failed node and restored at one or more new nodes;

retrieving, by the computer, a configuration package for each of the one or more modules from the dependency manager node; and

transmitting, by the computer, each configuration package to the one or more new nodes.

50. The method according to claim 49, further comprising:

determining, by the computer, a next new node having a set of available resources capable of installing and executing a module in the one or more modules migrated off the failed node;

instructing, by the computer, a new node manager of a new node storing the module to unload the module; and

transmitting, by the computer, the configuration package to the next new node.

51. A computer-implemented method comprising:

continuously transmitting, by a computer, a heartbeat signal to a supervisor node;

restoring, by the computer, the failed module when receiving a restore command; and

transmitting, by the computer, a restored status signal to the supervisor node when the computer detects the module is restored.

52. The method according to claim 51, further comprising transmitting, by the computer, a restore failure status signal to the supervisor node when a threshold number of restore attempts is reached.

53. A fault-tolerant distributed computing system comprising:

one or more nodes comprising a processor transmitting a heartbeat signal to a supervisor node and monitoring execution of one or more software modules installed on the node; and

one or more supervisor nodes comprising a processor monitoring one or more heartbeat signals received from the one or more nodes, and determining a status of each respective node based on each respective heartbeat signal.

54. The system according to claim 53, wherein the node processor is configured to attempt to restore a software module executed by the node to a status quo configuration responsive to receiving a command to restore the software module from the supervisor node.

55. The system according to claim 54, further comprising a failover node comprising a processor transmitting a heartbeat signal to the supervisor node, configured to execute a software module installed on a node of the one or more nodes.

56. The system according to claim 55, further comprising a dependency manager node comprising non-transitory machine-readable storage media storing one or more machine-readable configuration package files.

57. The method according to claim 56, wherein the processor of the supervisor node determines a number of attempts to restore the software module by the node, and wherein the processor of the supervisor node automatically retrieves from the dependency manager a configuration package file associated with the software module responsive to determining the number of attempts exceeds a threshold number of attempts to restore the software module.

58. The method according to claim 57, wherein the processor of the supervisor node transmits the configuration package file to the failover node in response to receiving the configuration package file from the dependency manager, and instructs the processor of the failover node to attempt to restore the software module.

59. The system according to claim 56, wherein the processor of the dependency manager node transmits a configuration package file to the supervisor node in response to receiving from the supervisor node a request identifying the configuration package file.

60. The method according to claim 59, wherein the configure package file is associated with a software module detected as a failure according to the node status of the heartbeat signal of the node executing the software module.

61. The system according to claim 60, wherein the supervisor node transmits a resource-shifting command to the failover node responsive to determining the node status of the failover nodes indicates that the failover node has insufficient resources to restore the software module.

62. The system according to claim 61, wherein the processor of the failover node automatically uninstalls an installed software module from the failover node in response to the resource-shifting command, and wherein the failover node attempts install and restore the software module received from the supervisor node.

63. The system according to claim 53, further comprising a redundant node comprising a non-transitory machine-readable storage medium storing a redundant copy of a software module of the one or more software modules, and a processor configured to automatically

attempt to execute the redundant copy of the software module responsive to receiving a command to restore the module from the supervisor node instructing the redundant node to attempt to execute the redundant copy of the software module.

64. A computer-implemented method comprising:

transmitting, by a computer of a distributed computing system, to a dependency manager node a request for a machine-readable deployable package file associated with a target node, wherein the dependency manager node comprises a non-transitory machine-readable storage medium storing one or more deployable package files associated with one or more nodes of the system according to dependency tree;

transmitting, by the computer, the deployable package file to the target node in response to receiving the deployable package file from the dependency node, wherein the deployable package file associated with the target node contains a set of one or more dependency files based on the dependency tree; and

instructing, by the computer, the target node to install the set of dependencies in the deployable package onto the target node.

65. The method according to claim 64, wherein a dependency is selected from the group consisting of: a partition, a module, a file, and data.

66. The method according to claim 64, wherein the dependency manager node fetches a dependency from one or more one dataframes and compiles the deployable package.

67. The method according to claim 64, wherein each respective dependency file in the deployable package file is determined by the dependency manager node using at least one dependency tree associated with the target node, and

68. The method according to claim 67, wherein each respective dependency file in the deployable package file is based on a comparative relationship of the dependency file with a corresponding dependency file installed on a subset of one or more nodes in a system.

69. The method according to claim 64, further comprising:

receiving, by the computer, from each of the one or more nodes of the distributed computing system a heartbeat signal indicating a status of the respective node, at a predetermined interval; and

determining, by the computer, the status of each respective node based on each respective heartbeat signal.

70. The method according to claim 69, further comprising detecting, by the computer, a failure of one or more dependencies installed on the target node according to the heartbeat signal received from the target node, wherein the computer automatically requests the deployment package responsive to detecting the failure.

71. The method according to claim 69, further comprising receiving, by the computer, from a vendor server associated with a dependency file installed on a node of the system, a prompt to update the node having the dependency file associated with the vendor server, wherein the one or more dependency files of the deployment package file contains the dependency file associated with the vendor server.

72. A computer-implemented method comprising:

determining, by a computer, a set of one or more dependency files to be installed onto a target node using a dependency tree associated with the target node responsive to receiving a request to configure the target node from a supervisor node;

fetching, by the computer, each of the dependency files of the set of one or more dependency files from at least one dataframe comprising non-transitory machine-readable storage medium storing one or more dependency files;

generating, by the computer, a deployable package file comprising the set of one or more dependency files; and

transmitting, by the computer, the deployable package file to the supervisor node.

73. The method according to claim 72, further comprising updating, by the computer, the dependency tree associated with the target node upon receiving a request to update the dependency from a third-party dependency source.

74. The method according to claim 73, further comprising:

detecting, by the computer, an update to the dependency associated with an updated dependency tree; and

automatically transmitting, by the computer, an updating deployment package comprising the updated dependency to each node having the dependency that was updated, wherein the updating deployment package for each node is based on the updated dependency tree.

75. A database management system comprising:

one or more nodes comprising a non-transitory machine-readable storage memory storing one or more dependency files, and a processor monitoring a status of the one or more dependency files, wherein each respective dependency file is a component of the node having a comparative relationship with a corresponding component installed on a second node;

one or more supervisor nodes comprising a processor monitoring a status for each of the one or more nodes and configured to transmit a deployable package comprising a set of dependencies files to each of the nodes based on the status of each respective node; and

one or more dependency manager nodes comprising a non-transitory machine-readable storage medium storing one or more dependency tree files associated with the one or more nodes, and a processor configured to compile a deployable package file in accordance with a dependency tree associated with a node, wherein the deployable package file comprises a set of one or more dependencies files stored on at least one data frame, and wherein the dependency manager node determines a dependency to include in the deployable package based on a dependency tree associated with a node targeted to receive the deployable package.

76. The system according to claim 75, further comprising an external dataframe comprising a non-transitory machine-readable storage medium storing one or more machine-readable update files corresponding to one or more dependency files install updates for a dependency generated by the third-party; and

a processor configured to transmit a request to a supervisor node to update a set of one or more nodes having the dependency of the third-party dependency source,

wherein a dependency manager node compiles an update deployment package comprising the one or more dependency updates received from the data frame of the third-party dependency source.

77. The system according to claim 75, further comprising a new node comprising a processor executing a node manager module that transmits a new node status to a supervisor node and receives one or more deployment packages according to a dependency tree associated with the new node.

78. The system according to claim 75, wherein the dependency manager node updates a dependency tree associated with a node when a request to update the dependency tree is received from a supervisor node,

79. The system according to claim 75, wherein a dependency is selected from the group consisting of: a partition, a module, a file, and data.

80. An in-memory database system comprising:

one or more storage nodes comprising a non-transitory machine-readable storage media storing one or more records of a database, wherein the storage media of each respective storage node is a main memory of the respective storage node;

an analytics agent nodes comprising a processor executing an analytics module using a set of query results as input parameters responsive to receiving a request for analytics indicating the analytics module, wherein the set of query results contains binary data representing one or more records retrieved from the one or more storage nodes storing the one or more records; and

an analytics module datastore comprising non-transitory machine-readable storage media storing one or more analytics modules, and a processor configured to transmit a new analytics module to one or more analytics agent nodes.

81. The system according to claim 80, further comprising a search manager node comprising a processor transmitting the set of query results to the analytics agent node responsive to determining the search query contains the request for analytics.

82. The system according to claim 81, wherein the search manager node identifies the analytics agent nodes prior to transmitting the search query to a search conductor node.

83. The system according to claim 82, wherein the search manager node determines one or more analytics parameters to be applied prior to transmitting the search query to the search conductor node.

84. The system according to claim 81, wherein, responsive to the search manager determining the search query contains the request for analytics: identifying one or more analytics agent nodes executing the analytics module indicated by the request for analytics.

85. The system according to claim 84, further comprising a search conductor node comprising a processor executing a search conductor query and transmitting a subset of query results containing one or more records of a storage node satisfying the search conductor query, in response to receiving the search conductor query from the search manager node,

wherein the search manager node generates one or more search conductor queries according to the search query, and wherein the search manager transmits each respective search conductor query to each respective search conductor node.

86. The system according to claim 85, wherein the search manager node aggregates each of the respective subsets of query results received from the respective search conductors, thereby generating the set of query results.

87. The system according to claim 80, wherein the search manager node transmits to the analytics agent at least one record in the set of query results is compressed according to a binary data compression algorithm.

88. The system according to claim 80, wherein the analytics module is selected from the group consisting of: a disambiguation module, a linking module, a scoring module, and a link on-the-fly module.

89. The system according to claim 80, further comprising a system interface node comprising a processor transmitting a new analytics module to the analytics module store responsive to receiving the new analytics module from a client computer.

90. The system according to claim 80, further comprising a system interface node comprising a processor validating data of the search query in accordance with one or more application programming interfaces.

91. The system according to claim 80, further comprising a system interface node storing a new analytics module in one or more analytics agents and configured to generate one or more new analytics modules based on an application programming interface.

92. The system according to claim 80, further comprising a client computer comprising a processor transmitting one or more search queries to the search manager.

93. The system according to claim 80, wherein the analytics agent node automatically transmits a heartbeat signal to a supervisor node indicating the analytics agent node received a new analytics module responsive to receiving the new analytics module from a client computer.

94. The system according to claim 93, further comprising one or more supervisor nodes comprising a processor transmitting a new analytics module from the analytics module store to a new analytics agent according to a dependency tree associated with the new analytics agent stored in a dependency manager node.

95. A system comprising:

one or more storage nodes comprising a non-transitory machine-readable storage medium storing one or more collections of database records, wherein each respective collection stores one or more records according to a schema, and wherein the schema defines one or more fields storing a type of data in each respective record;

a search manager comprising a processor receiving a search query containing search parameters associated with data of one or more data types, determining whether field processing is requested in the search query, generating one or more search conductor queries based on a search query and transmitting the one or more search conductor queries to one or more search conductors nodes;

a search conductor node comprising a processor querying a collection associated with the search conductor and transmitting to the search manager a set of one or more results records containing data satisfying one or more parameters of the search conductor query in response to receiving the search conductor query from the search manager node.

96. The system according to claim 95,

wherein the search manager receives a first set of search result records containing data matching at least one parameter of the search query, wherein the fields of the records are based on a first schema, and

wherein the search manager receives a second set of search result records containing data matching at least one parameter of the search query, wherein the fields of the records are based on a second schema.

97. The system according to claim 95, wherein the search conductor matches each respective field of the one or more records satisfying the search parameters based on a match of the type of data in the respective field of the one or more records and the type of data of the search query.

98. The system according to claim 97, wherein the search conductor assigns a score for each respective records based on a number of matches between data in the fields of each record and the one or more search parameters.

99. The system according to claim 98, wherein the search conductor scores each respective field of the records of the collection based on the number of matches, and wherein the score assigned to each record is based on a search algorithm specified by the search conductor query.

100. The system according to claim 98, wherein the search conductor query indicates a threshold score value, and wherein the score assigned to each respective record of the set of results records satisfies the threshold score.

101. The system according to claim 98, wherein the search conductor sorts the set of results records based on the score assigned to each respective record of the collection associated with the search conductor.

102. The system according to claim 95, where the processor of the search manager executes field processing on one or more fields in accordance with the search query in response to determining that field processing is requested in the search query.

103. A computer-implemented method comprising:

determining, by a computer, a compression technique to apply to one or more data elements received in a set of data elements, wherein the computer uses a schema to determine the compression technique to apply to each data element based on a data type of the data element;

compressing, by a computer, a data element using the compression technique defined by the schema, wherein the compression technique compresses the data element such that the data element is individually decompressed when returned in response to a search query;

storing, by the computer, each compressed data element in a field of a record that stores data of the data type of the data element;

associating, by the computer, a field notation in a reference table for each field according to a schema, wherein the representative notation identifies the data type of the field;

querying, by the computer, the database for a set of one or more data elements satisfying a search query received from a search conductor; and

decompressing, by the computer, each of the one or more data elements of the one or more data elements satisfying the search query using the compression technique to apply responsive to identifying the set one or more data elements satisfying the search query, wherein each data element not satisfying the search query remains compressed.

104. The method according to claim 103, wherein the computer compresses the data using a compression technique selected from the group consisting of: token tables, n-gram compression, serial day number compression, and binary number compression.

105. The method according to claim 103, further comprising:

fragmenting, by the computer, the data elements in one or more fields of a record into fragmented data, wherein a fragmented datum is a data element in a field of a record that is a more granular than a parent data element, and wherein each fragmented datum is associated with a database field that is more granular; and

storing, by the computer, the fragmented data contiguously in the same partition.

106. The method according to claim 105, further comprising:

associating, by the computer, a fragmented record identifier with each of the fragmented datum, wherein the fragmented record identifier identifies the parent data element for a fragmented datum.

107. The method according to claim 103, further comprising:

replacing, by the computer, a first data element object comprising the data elements in one or more fields of a first record with a data element notation object that references to a second data element object identical to the first data element object, wherein the second data element object comprises the data elements in one or more fields of a second record.

108. The method according to claim 107, further comprising:

querying, by the computer, the database for the first object in response to a query received from a software module; and

outputting, by the computer, to the software module the second object referenced by the referring object that replaced the first object.

109. The method according to claim 103, wherein the computer compresses the one or more fields using a token table technique, further comprising:

adding, by the computer, one or more new fields storing new data to a record, wherein each field is associated with a token table;

compressing, by the computer, the new field when the new data does not match existing data in the associated token table; and

updating, by the computer, the token table using the new data of the new field.

110. The method according to claim 109, further comprising:

decompressing, by the computer, data that is compressed according to an obsolete technique;

compressing, by the computer, the data according to an improved compression technique.

111. The method according to claim 109, further comprising:

storing, by the computer, a higher weighted datum comprising frequency times length in a lower numbered index of a token table.

112. The method according to claim 104, wherein the computer compresses the one or more fields using a n-gram table technique, further comprising:

adding, by the computer, one or more new fields storing new data to a record, wherein each field is associated with a n-gram table;

compressing, by the computer, the new field when the new data does not match existing data in the associated n-gram table; and

updating, by the computer, the n-gram table using the new data of the new field.

113. The method according to claim 112, further comprising:

decompressing, by the computer, data that is compressed according to an obsolete technique; and

compressing, by the computer, the data according to an improved compression technique.

114. The method according to claim 112, further comprising:

storing, by the computer, a higher weighted datum comprising frequency times length in a lower numbered index of a n-gram table.

115. The method according to claim 104, further comprising:

searching, by the computer, a set of fields using an offset of a record descriptor byte;

decompressing, by the computer, accessed data at a field-level; and
returning, by the computer, the accessed data.

116. The method according to claim 115, further comprising:

halting, by the computer, decompression when a set of fields satisfying a query is found

117. A computing system comprising:

one or more nodes storing one or more collections, each collection comprising a set of one or more records, each record comprising a set of fields storing data; and

a compression processor compressing one or more of the fields according to a schema that is associated with a collection.

118. The system according to claim 117, the compression processor compresses the one or more fields based on a data type of the data stored in a field.

119. The system according to claim 117, wherein the compression processor compresses a field using a compression technique selected from the group consisting of: token table, Ngram compression, serial day number compression, and binary number compression.

120. The system according to claim 119, further comprising:

one or more partitioners partitioning an associated a collection into one or more partitions according to the schema.

121. The system according to claim 120, wherein the compression processor stores fragmented data in one or more fields of a record, and

wherein the compression processor contiguously stores the fragmented data in a partition of the collection.

122. The system according to claim 121, wherein each of the fragmented datum is associated with one or more fragmented record identifiers that identify an origin record of the fragmented data.

123. The system according to claim 122, wherein a plurality of fields store semantically similar data.

124. The system according to claim 117, wherein one or more fields store an array of data values comprising a set of zero or more values.

125. The system according to claim 124, wherein a record is an object comprising a plurality of fields that are grouped together according to the schema.

126. The system according to claim 125, wherein a partition stores a value reference table having a reference value associated with an object,

wherein the object is stored in a first record and comprises a plurality of grouped fields, and

wherein a second record stores the reference value associated with the object according to the reference table.

127. The system according to claim 117, further comprising:

one or more output processors outputting one or more object values of the object to one or more module processors.

128. The system according to claim 117, wherein the compression processor is configured to update the reference value stored in the second record according to an updated object stored in the first record.

129. The system according to claim 117, further comprising a normalizing processor normalizing a datum stored in a field of a record prior to compression.

130. The system according to claim 117, further comprising:

a token table store storing one or more token tables, each token table associated with a field, wherein the compression processor adds one or more new fields having new data to a record according to each of the token tables associated with each of the new fields.

131. The system according to claim 117, wherein the compression processor applies a second compression technique when new data does not match existing data in a token table.

132. The system according to claim 117, wherein the compression processor updates the token table according to the new data when the new data does not match existing data in a token table.

- 133.** The system according to claim 117, wherein the compression processor periodically updates the token table.
- 134.** The system according to claim 117, wherein one or more descriptor bytes comprising offset data address data in a field for retrieval.
- 135.** The system according to claim 117, wherein data in the one or more fields is indexed and/or compressed according to the schema of the collection by a search conductor processor.
- 136.** The system according to claim 117, wherein the data in the one or more fields is indexed and/or compressed according to the schema of the collection by a partitioner.
- 137.** The system according to claim 117, wherein a search conductor processor fetches and decompresses data satisfying a search query at a field-level.
- 138.** The system according to claim 137, wherein the search conductor processor is configured to halt decompression of a record when a set of data satisfying the search query are returned.
- 139.** The system according to claim 117, further comprising an n-gram table store storing one or more n-gram tables, each n-gram table associated with a field, wherein the compression processor adds one or more new fields having new data to a record according to each of the n-gram tables associated with each of the new fields.
- 140.** The system according to claim 117, wherein the compression processor updates the n-gram table according to the new data when the new data does not match existing data in a n-gram table.
- 141.** The system according to claim 117, wherein the compression processor periodically updates the n-gram table.

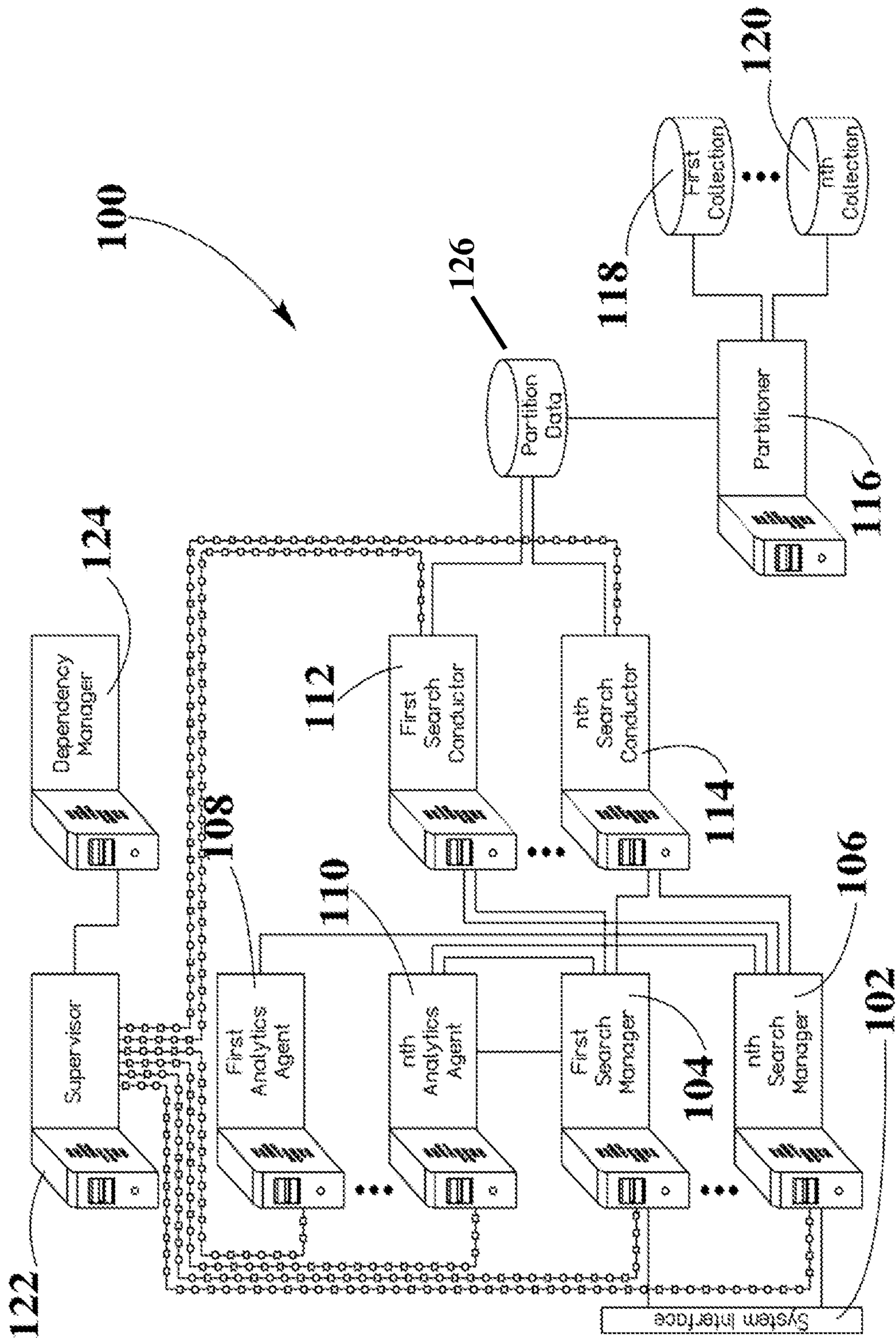


FIG. 1

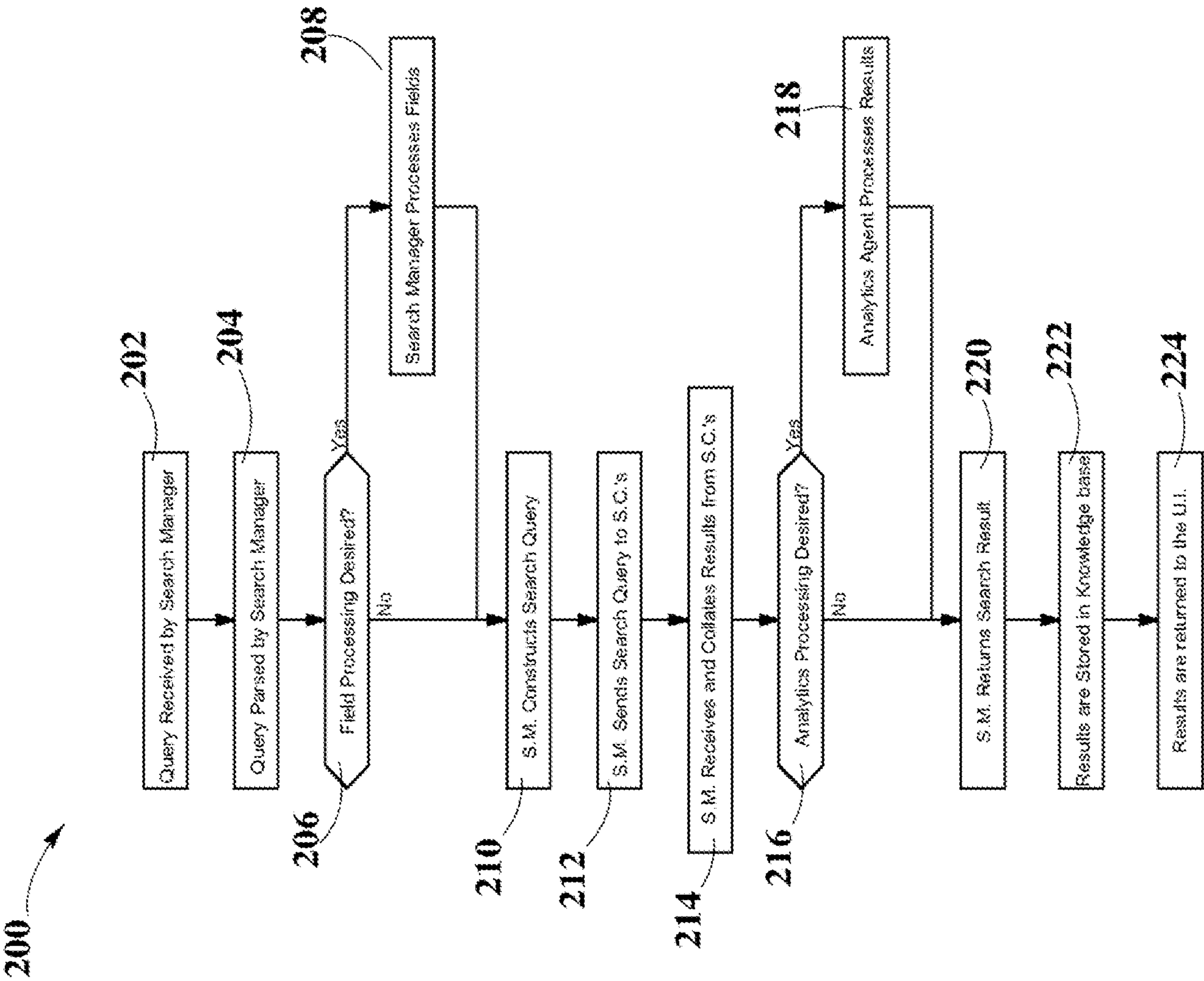


FIG. 2

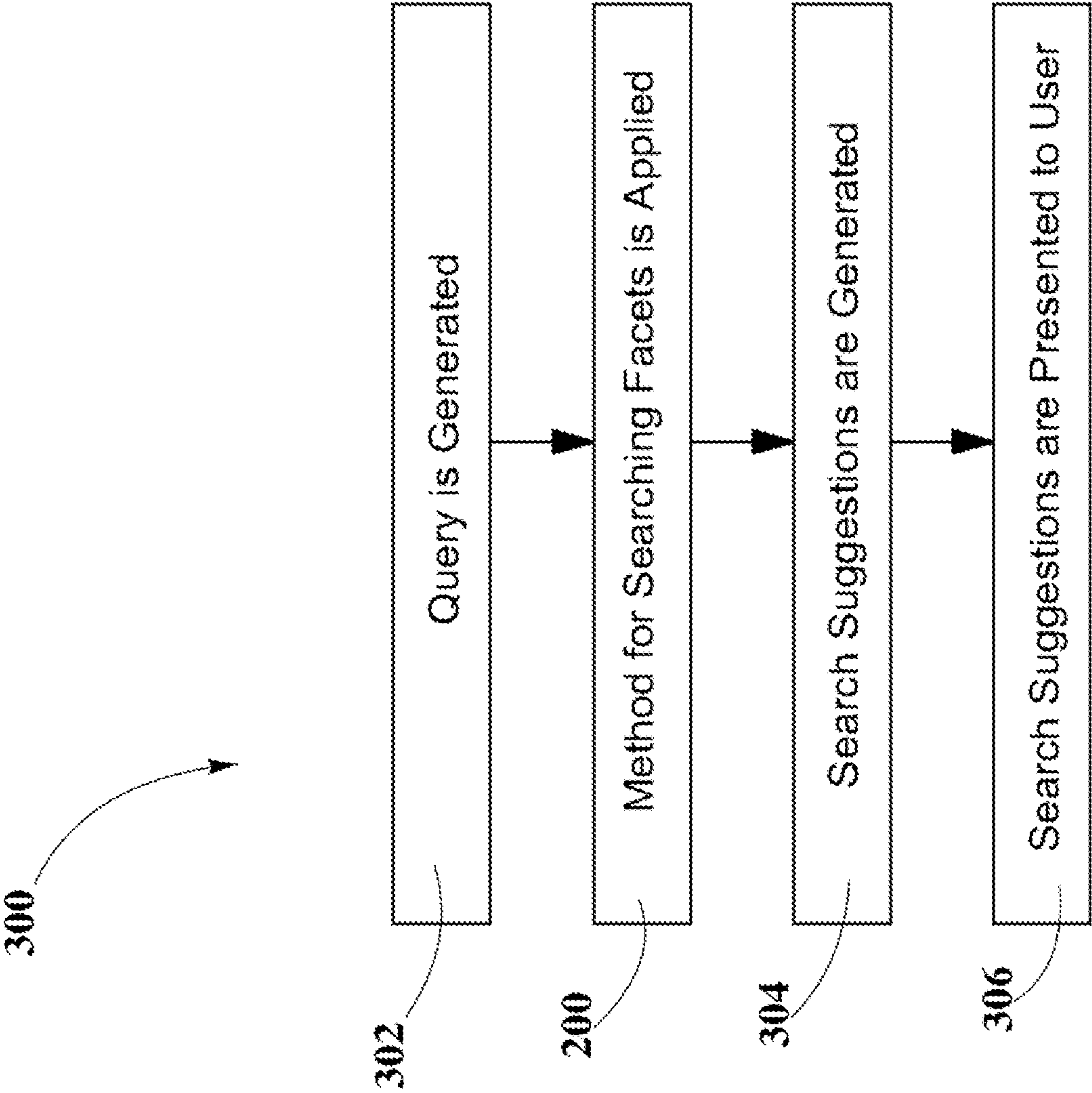


FIG. 3

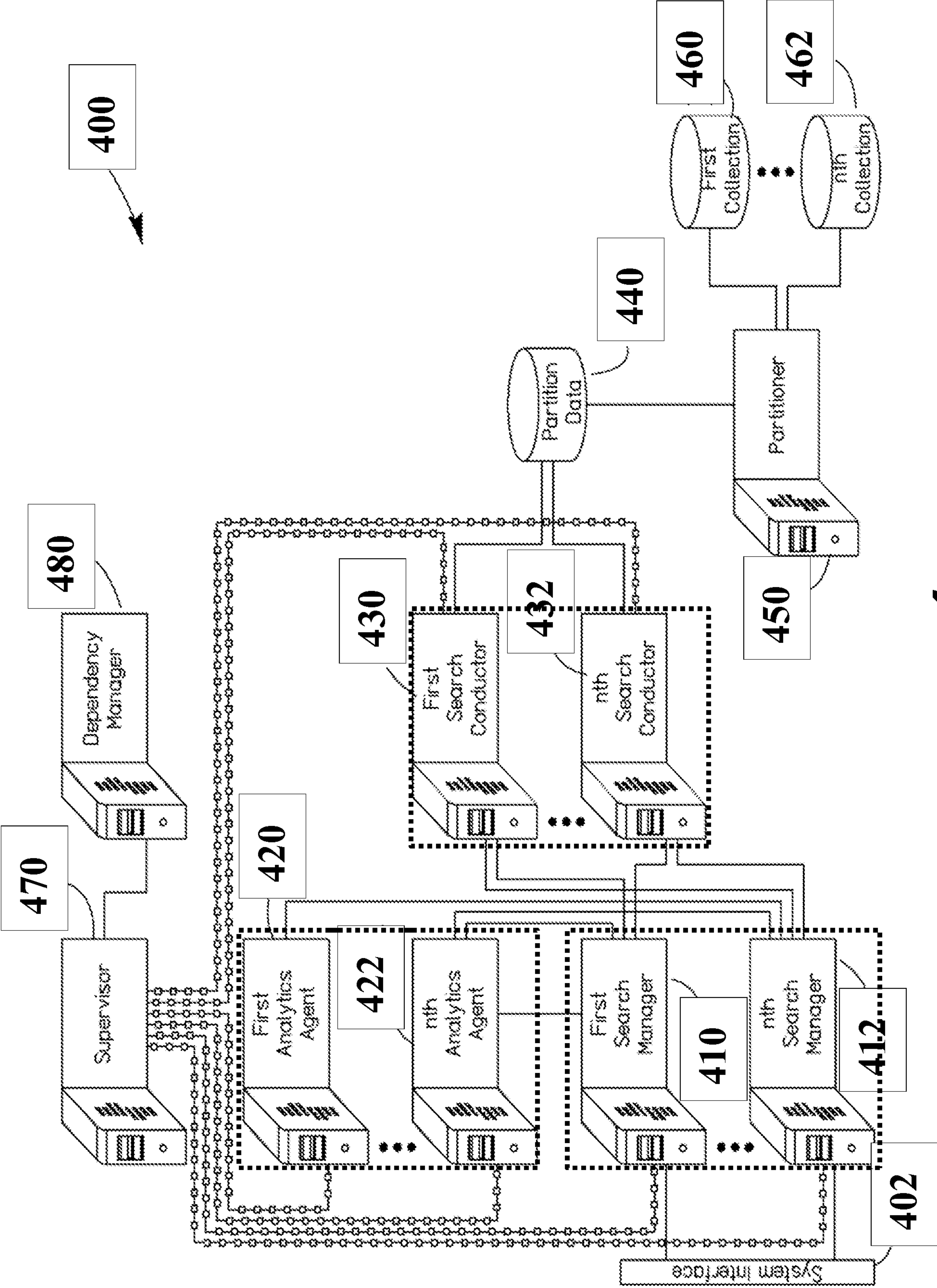


FIG. 4

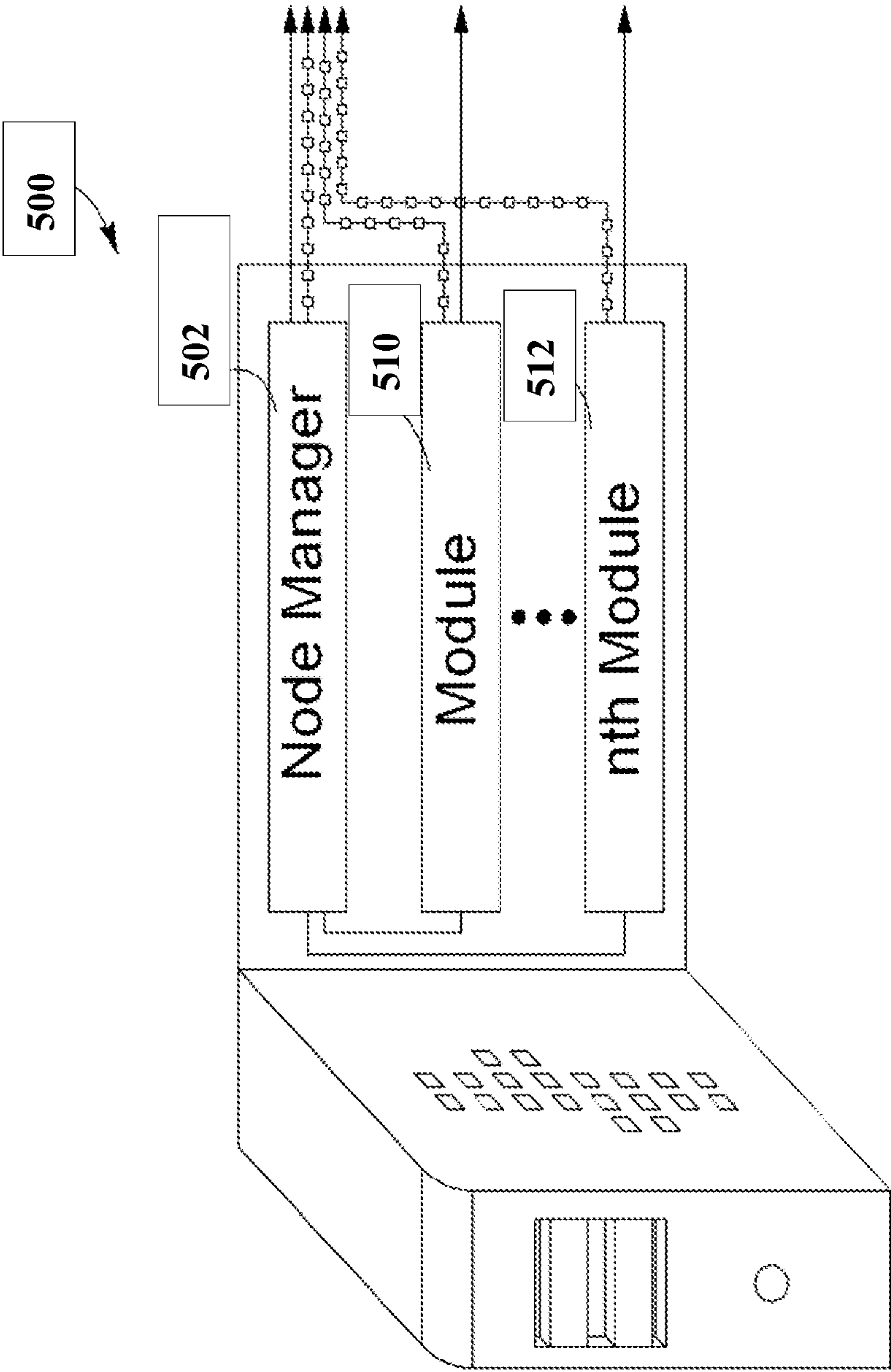


FIG. 5

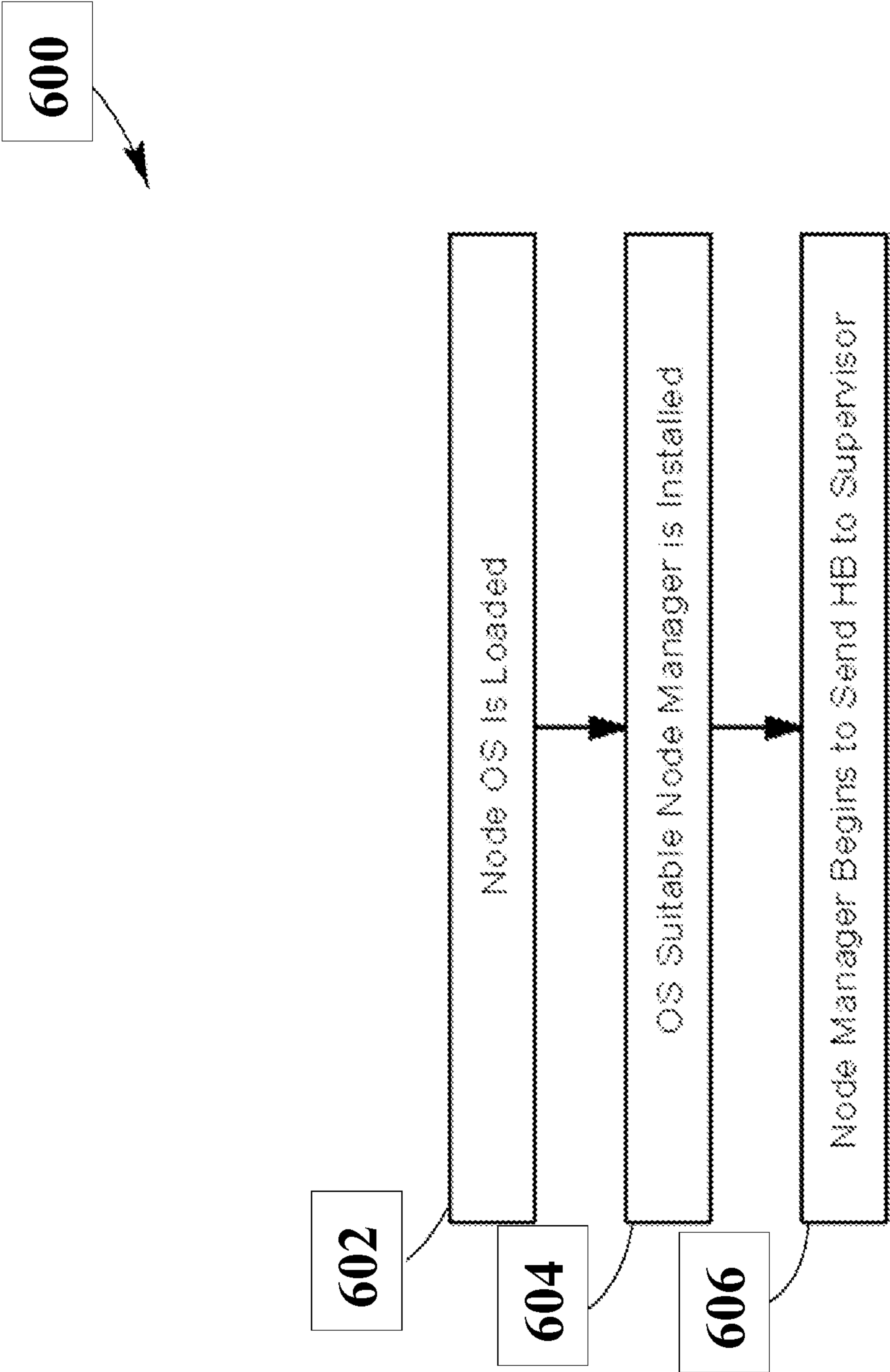


FIG. 6



FIG. 7

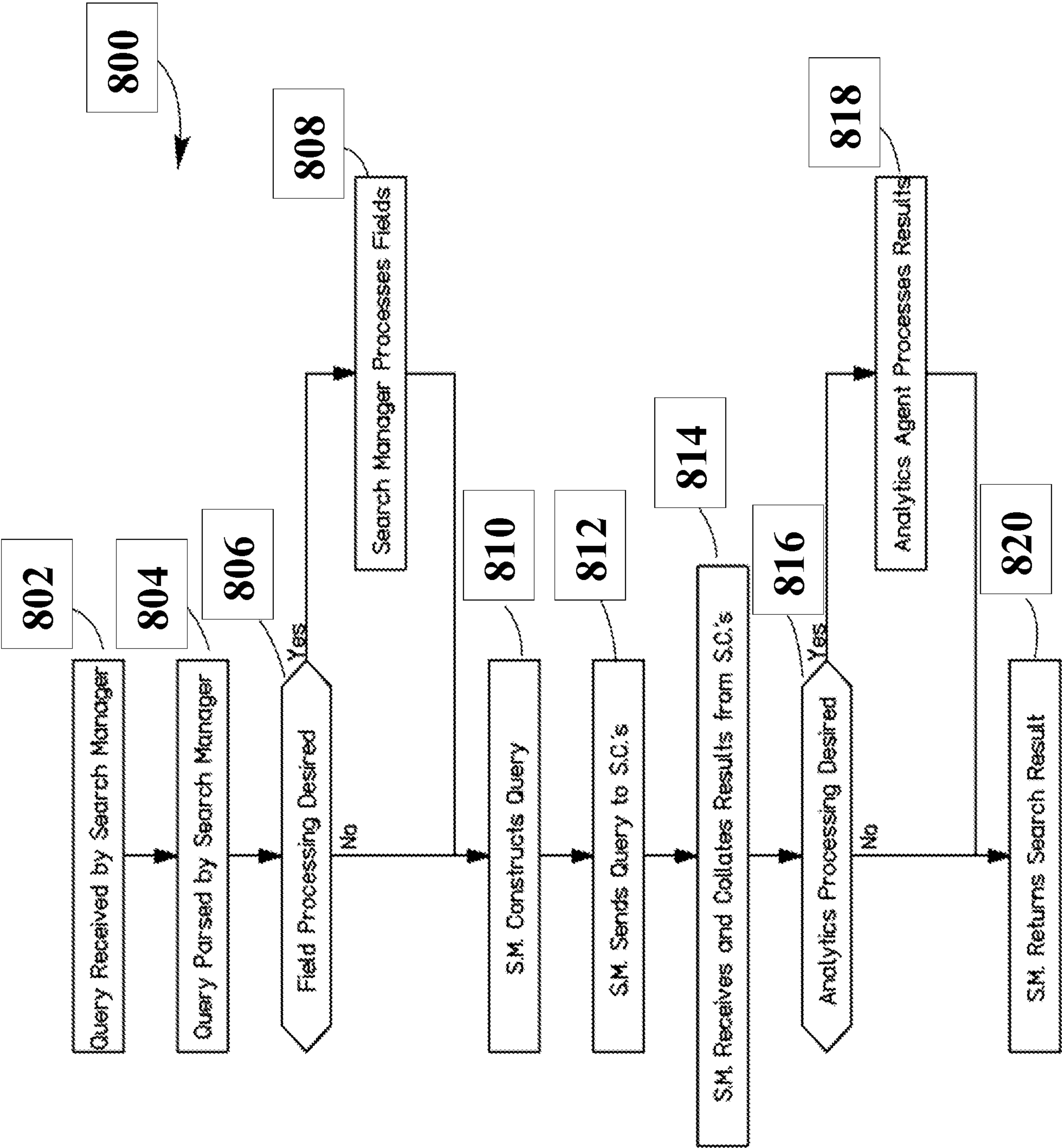


FIG. 8

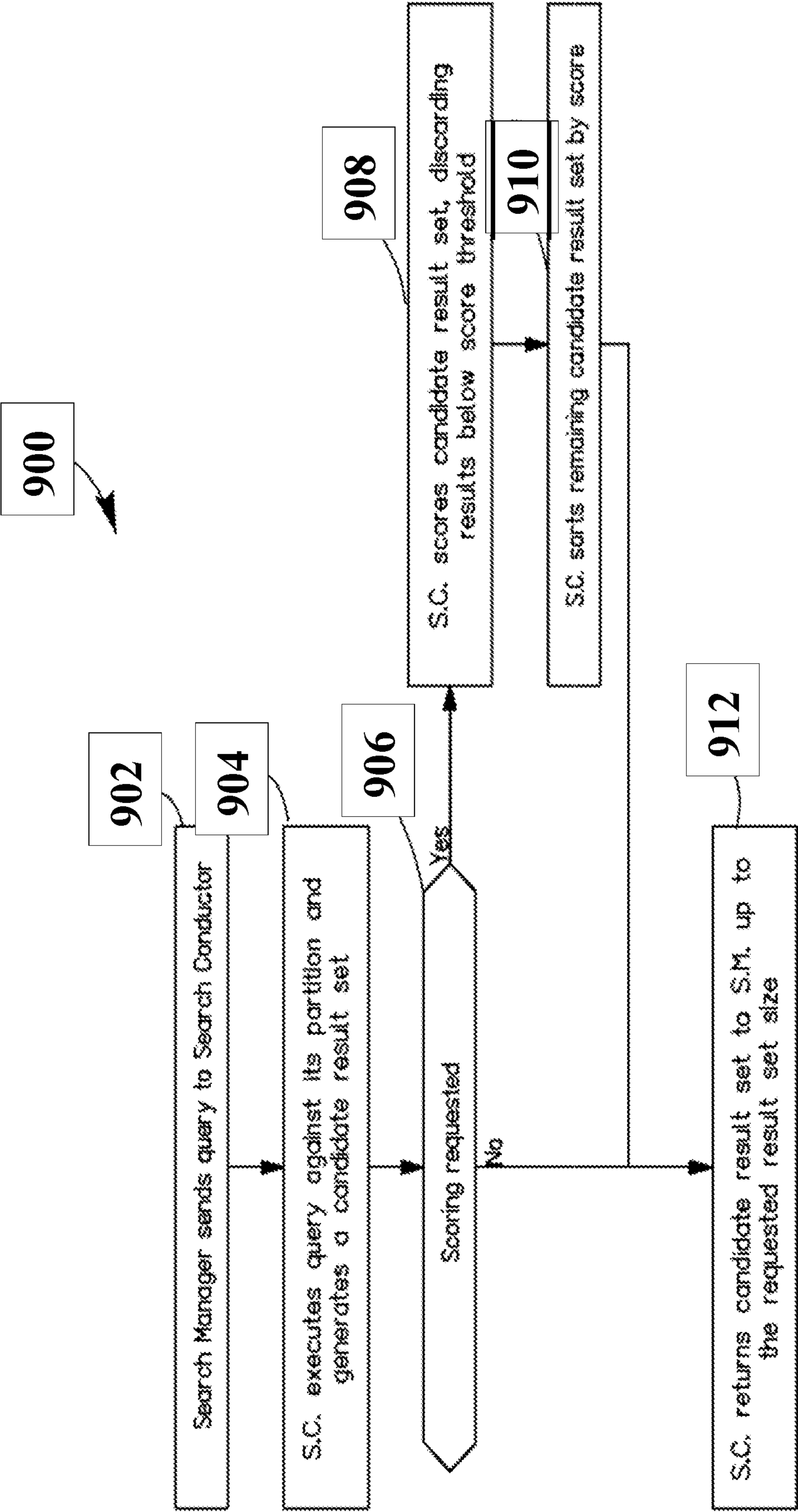


FIG. 9

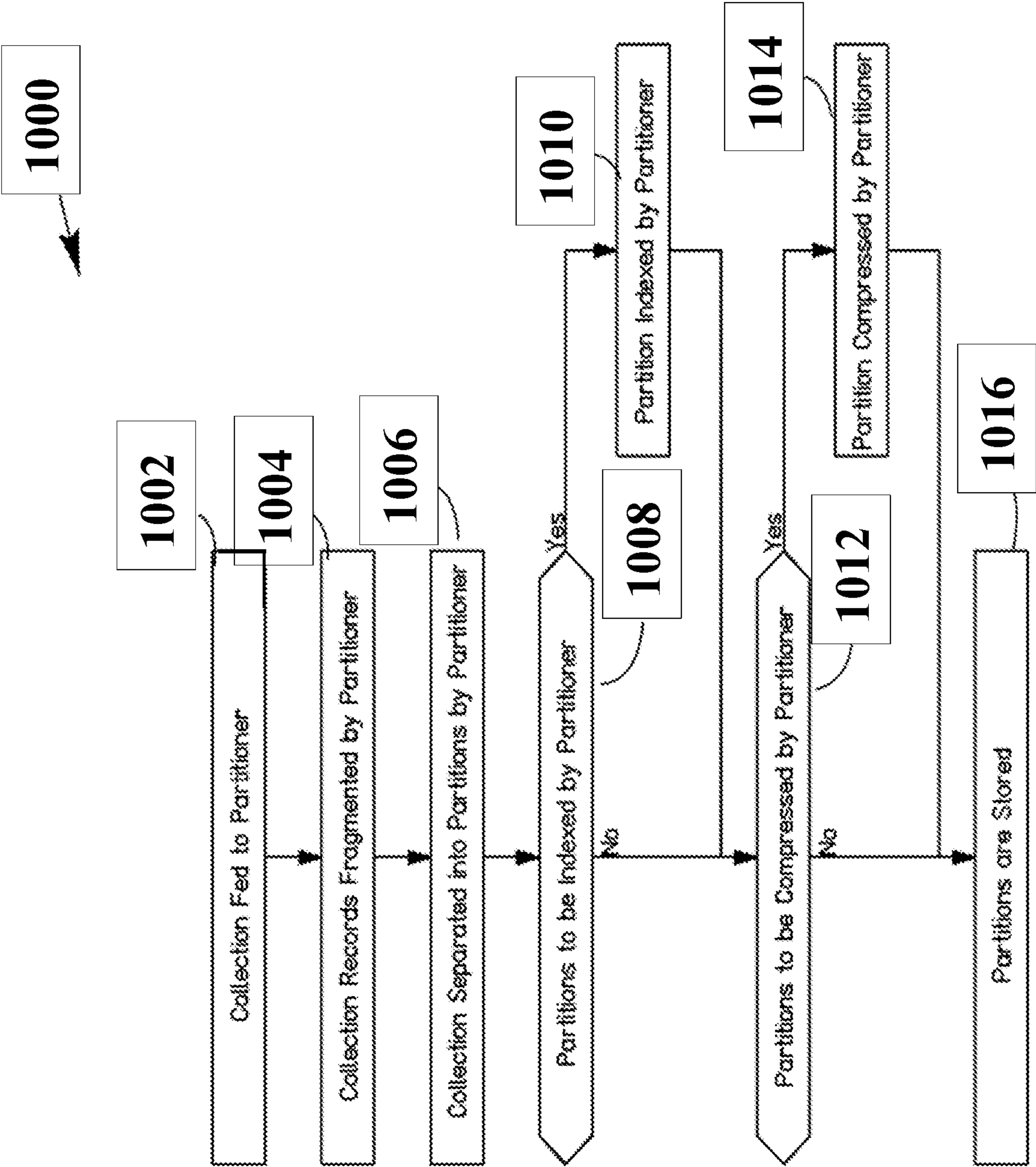


FIG. 10

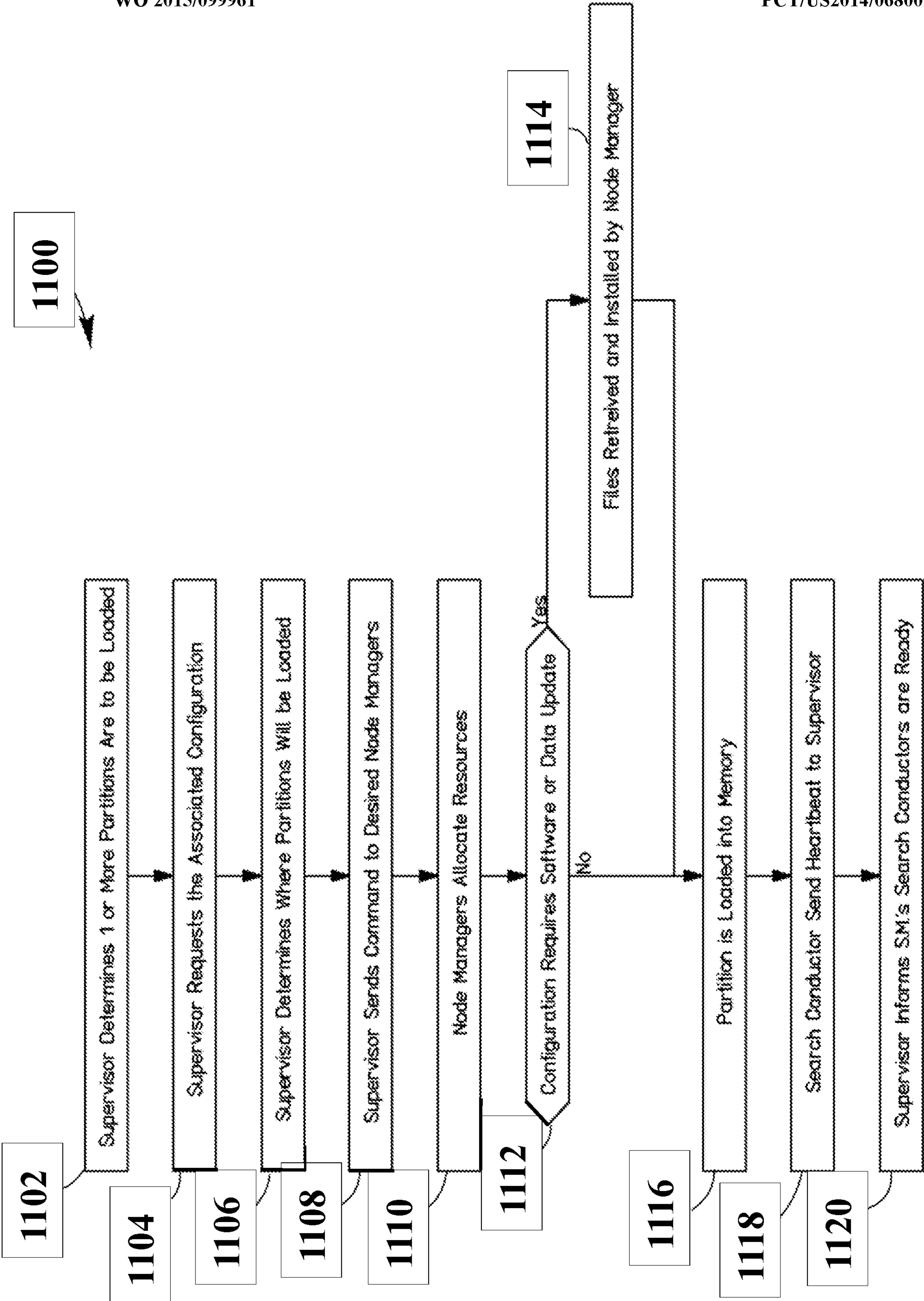


FIG. 11

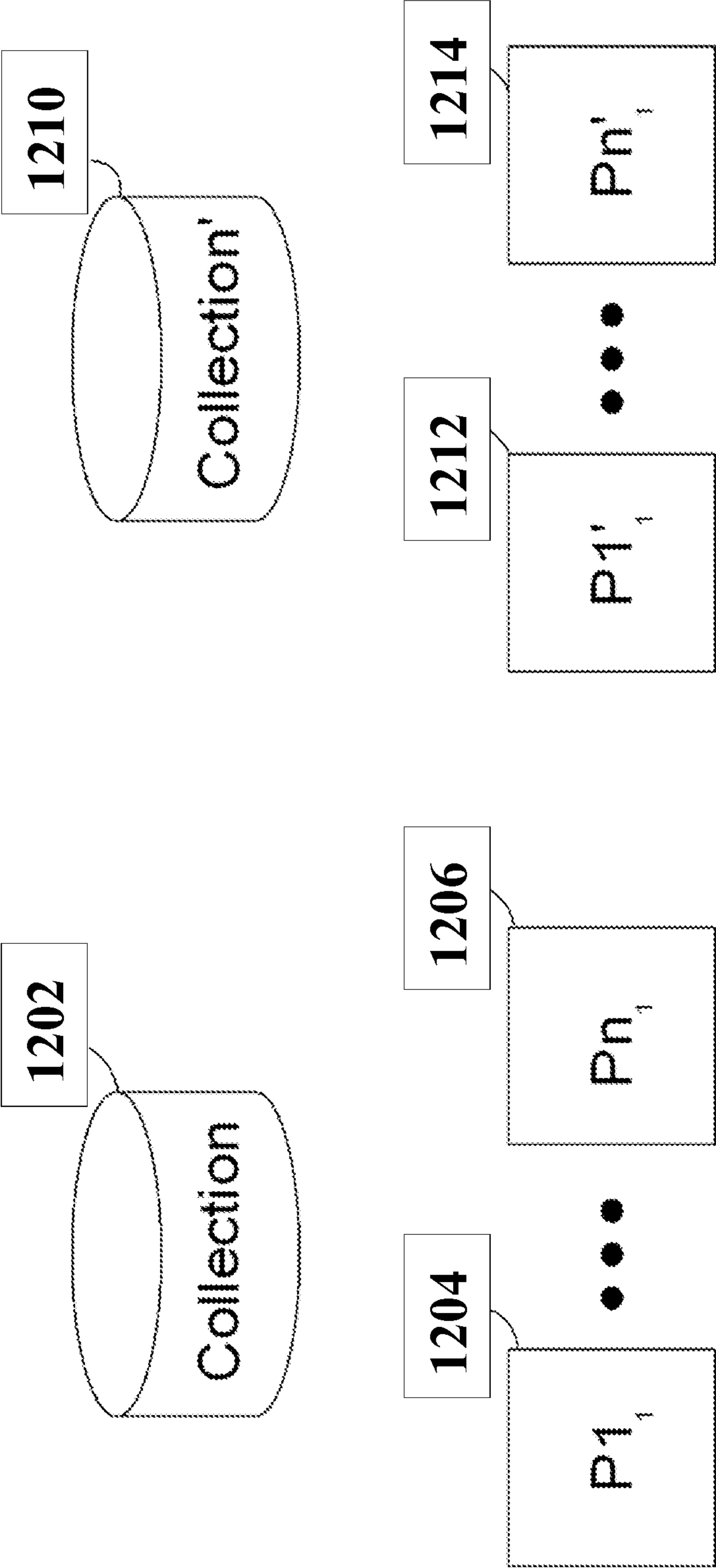


FIG. 12A

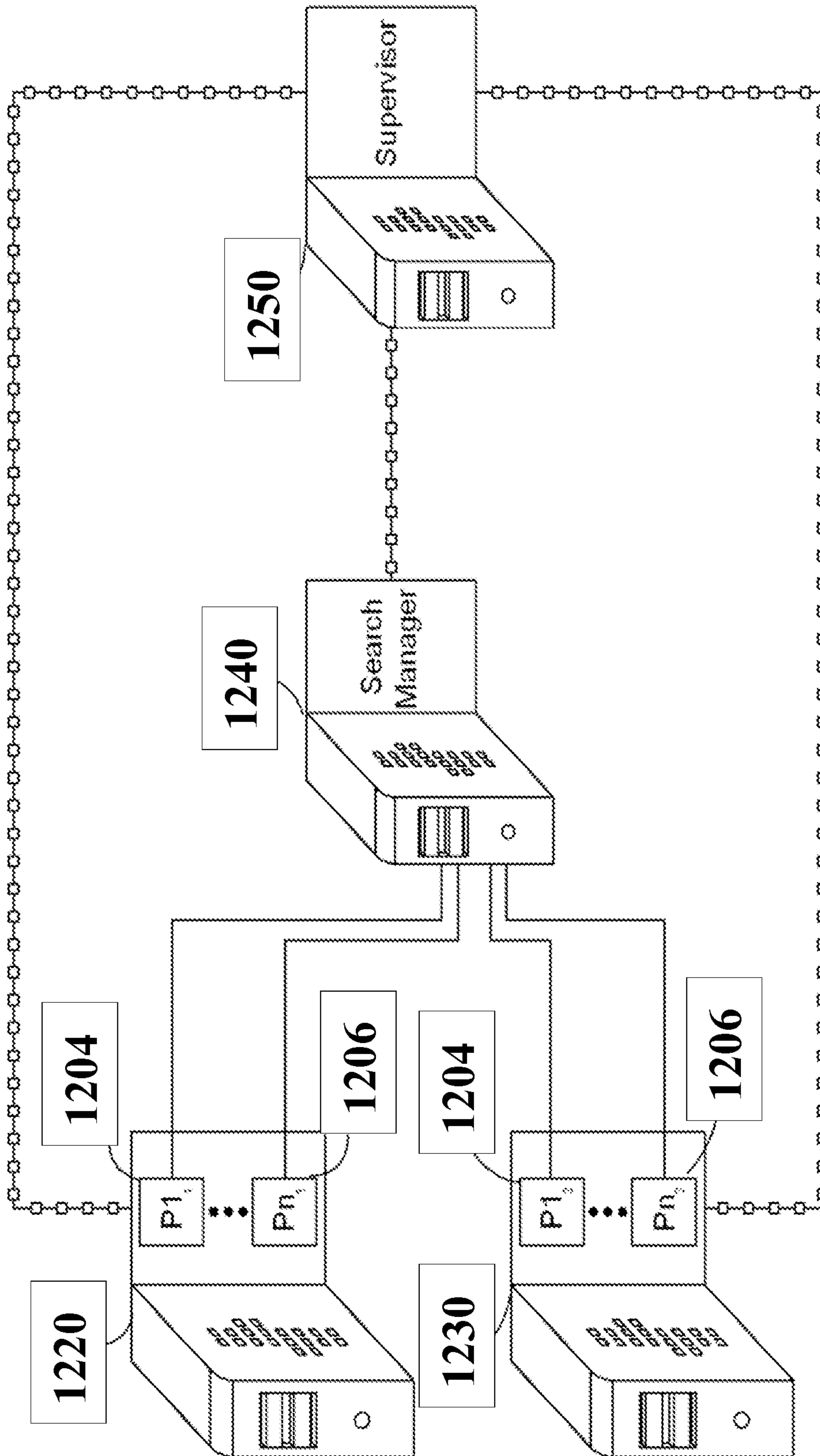


FIG. 12B

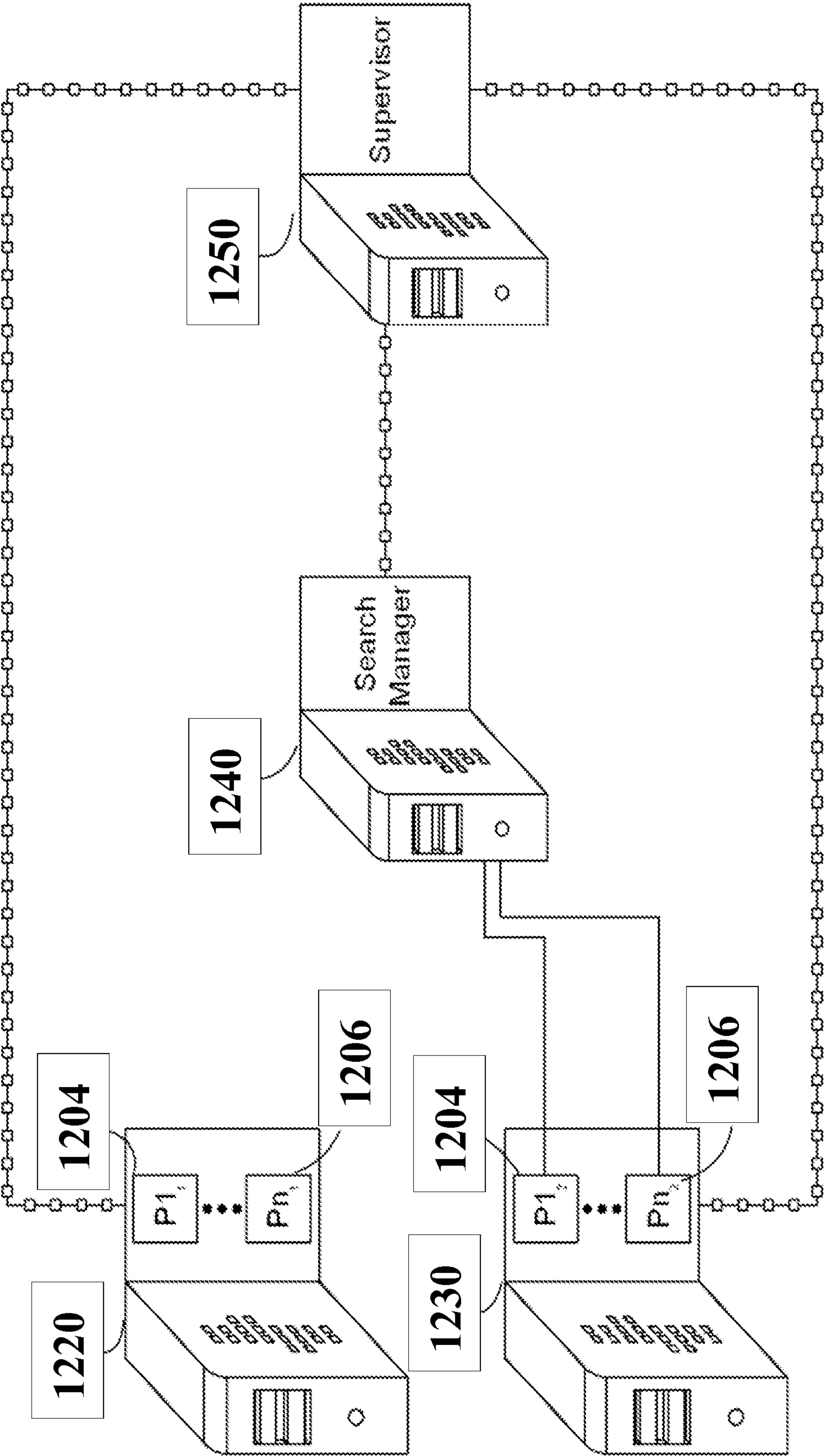


FIG. 12C

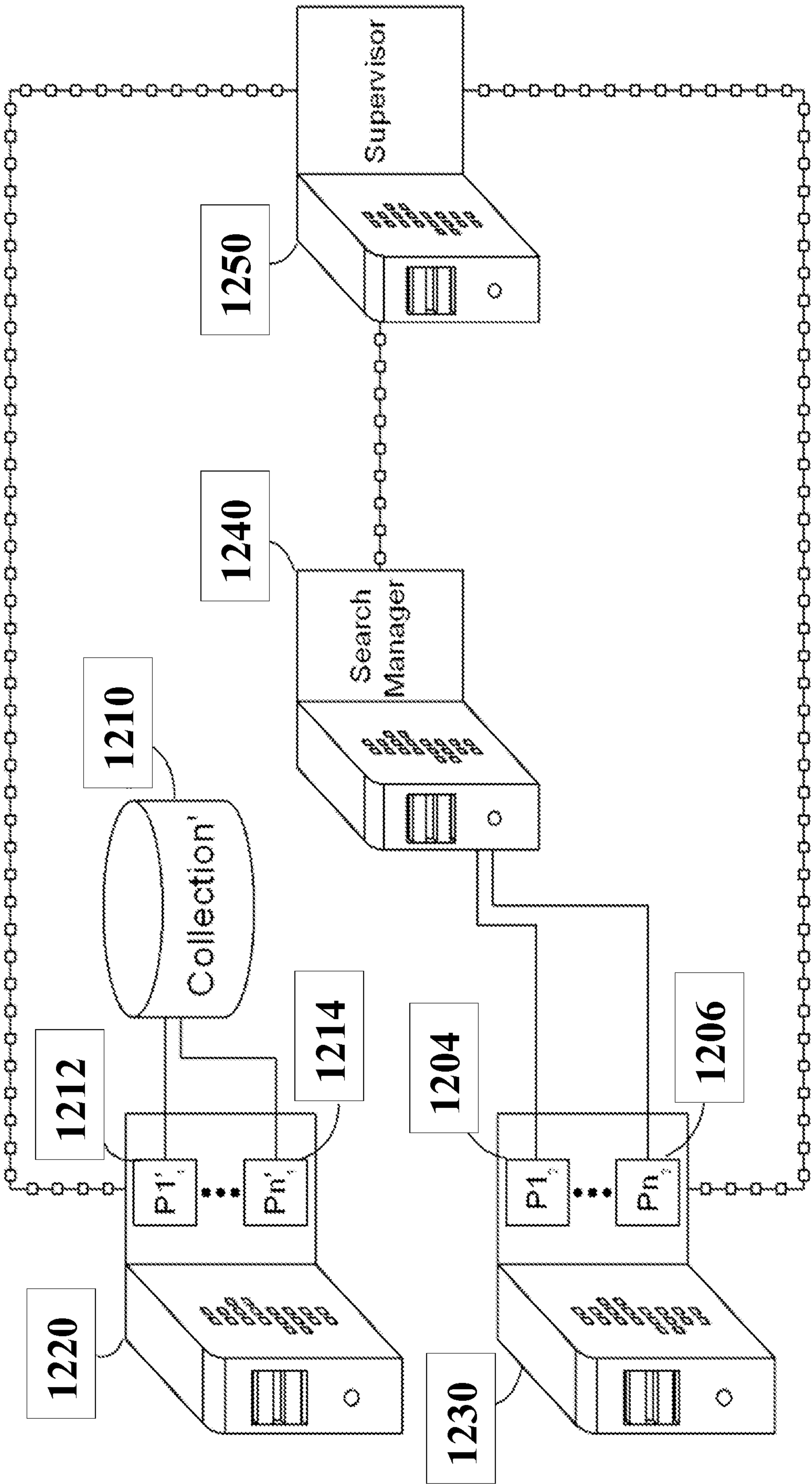


FIG. 12D

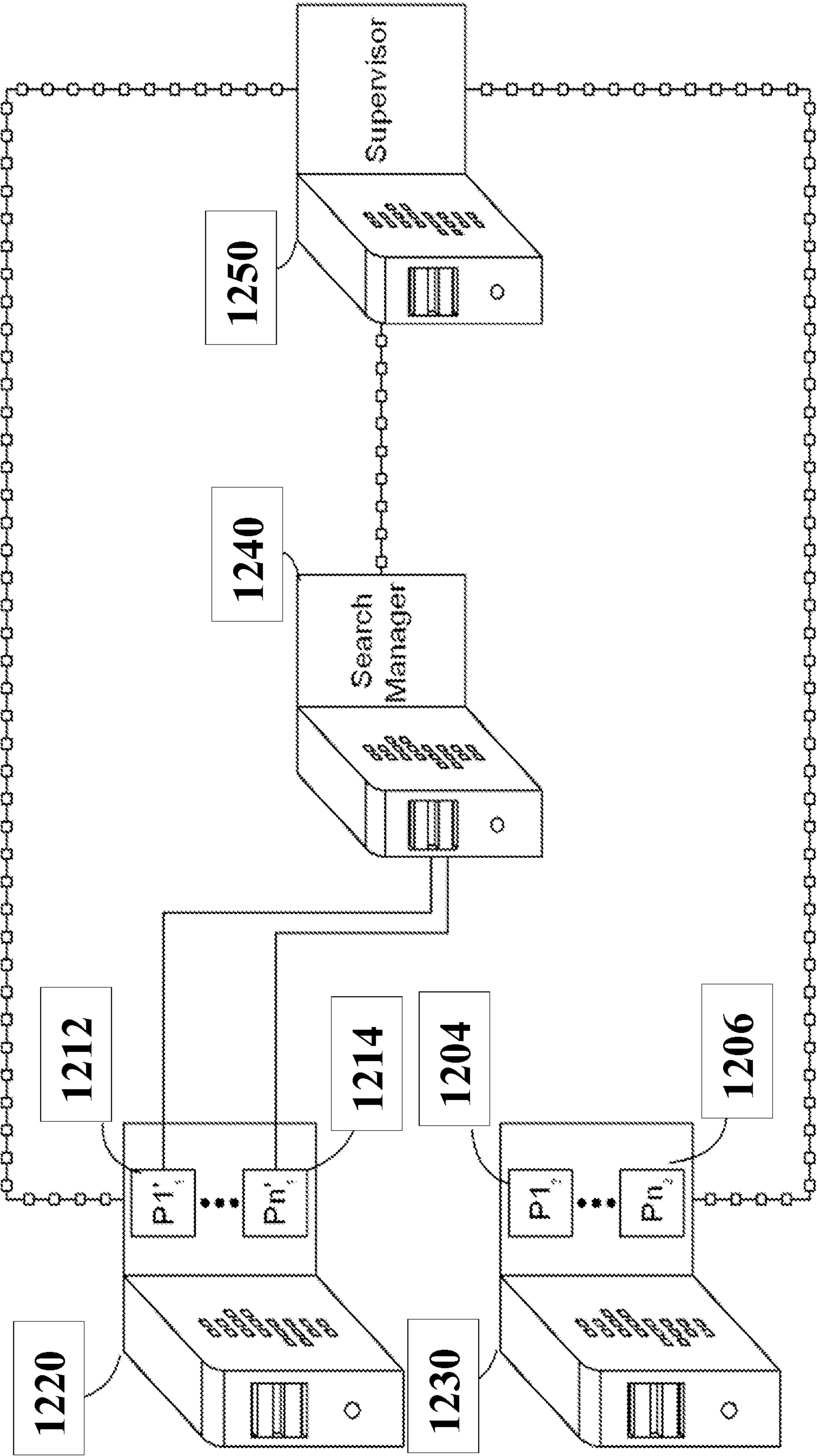


FIG. 12E

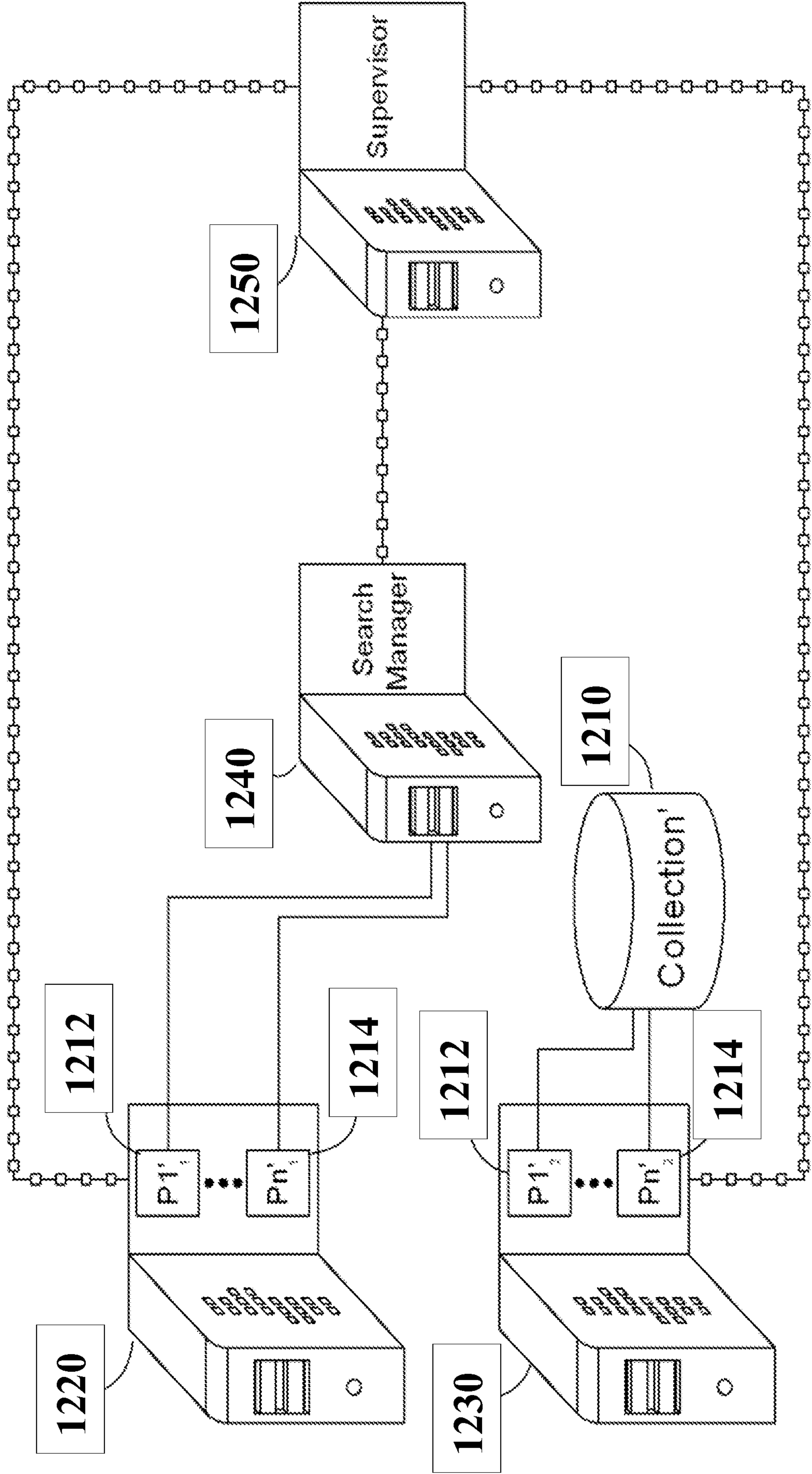


FIG. 12F

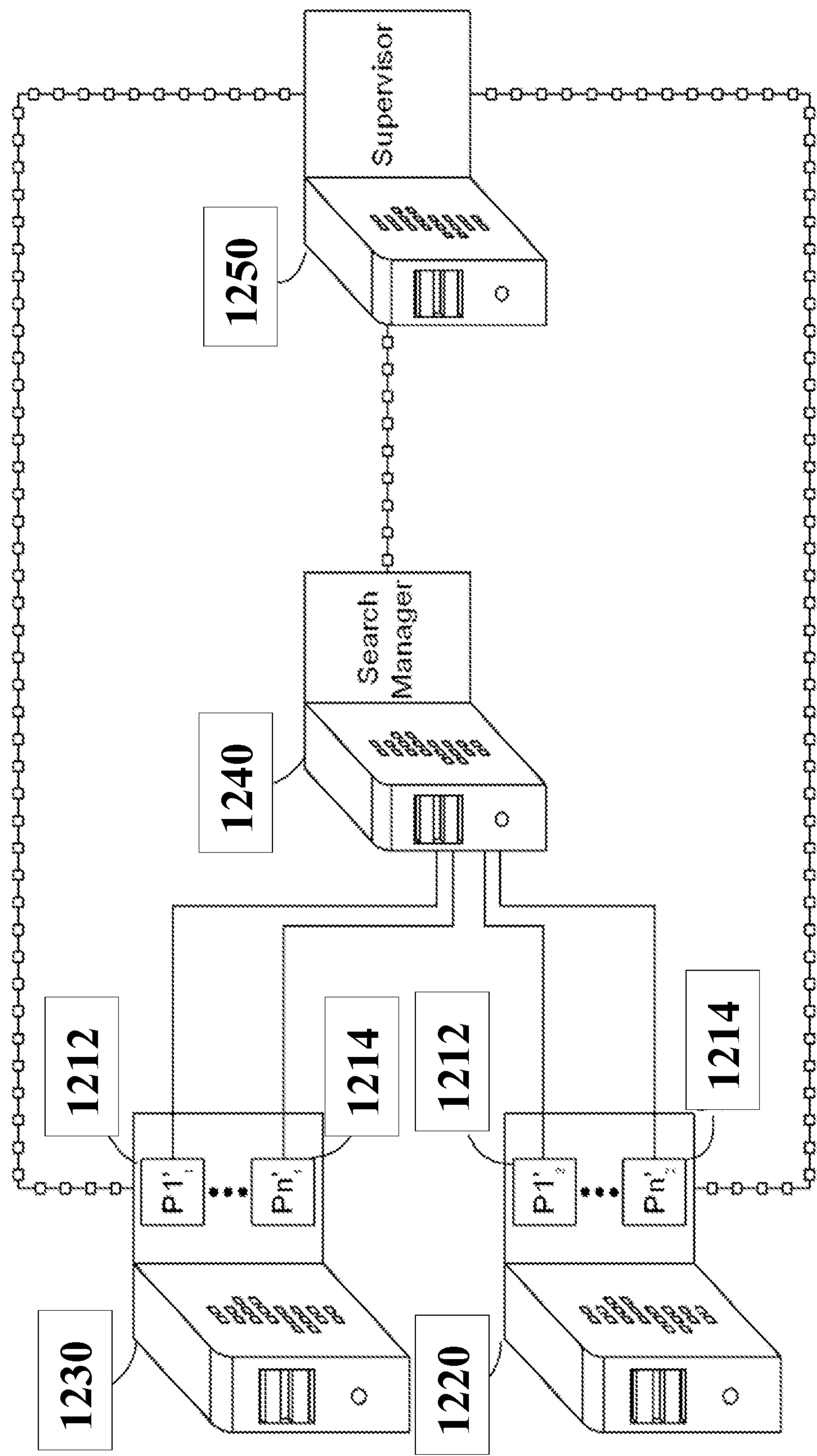


FIG. 12G

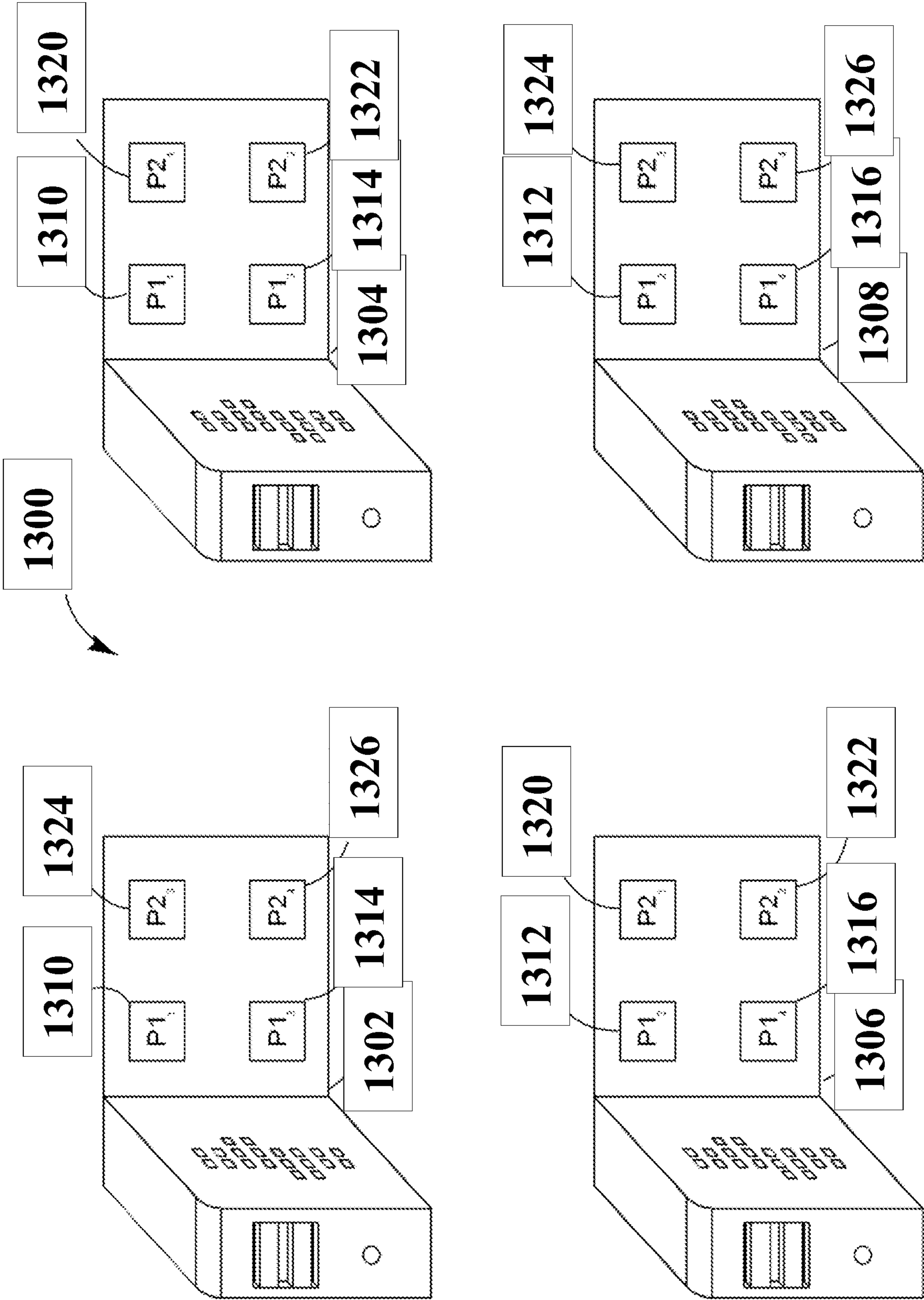


FIG. 13

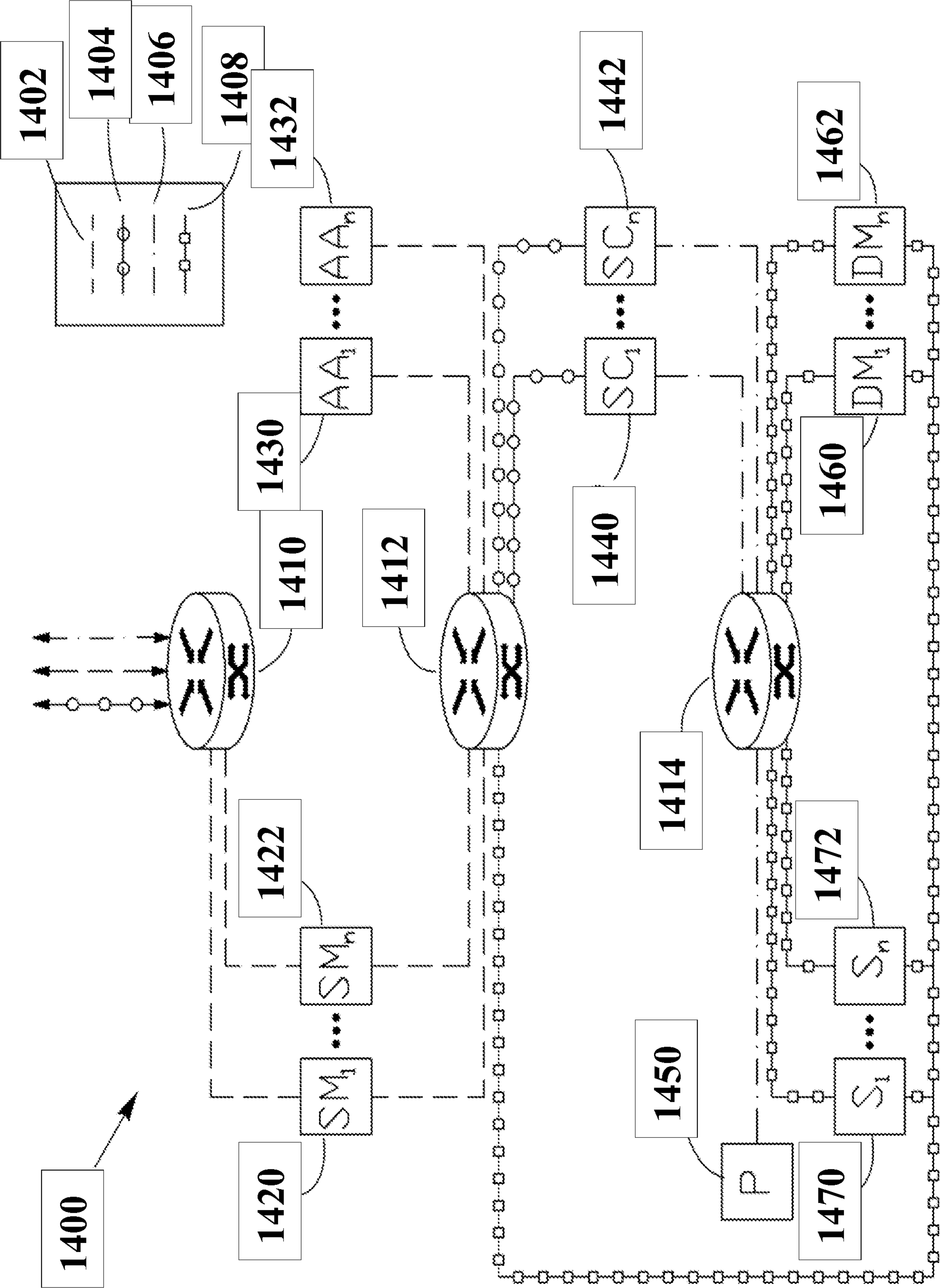


FIG. 14

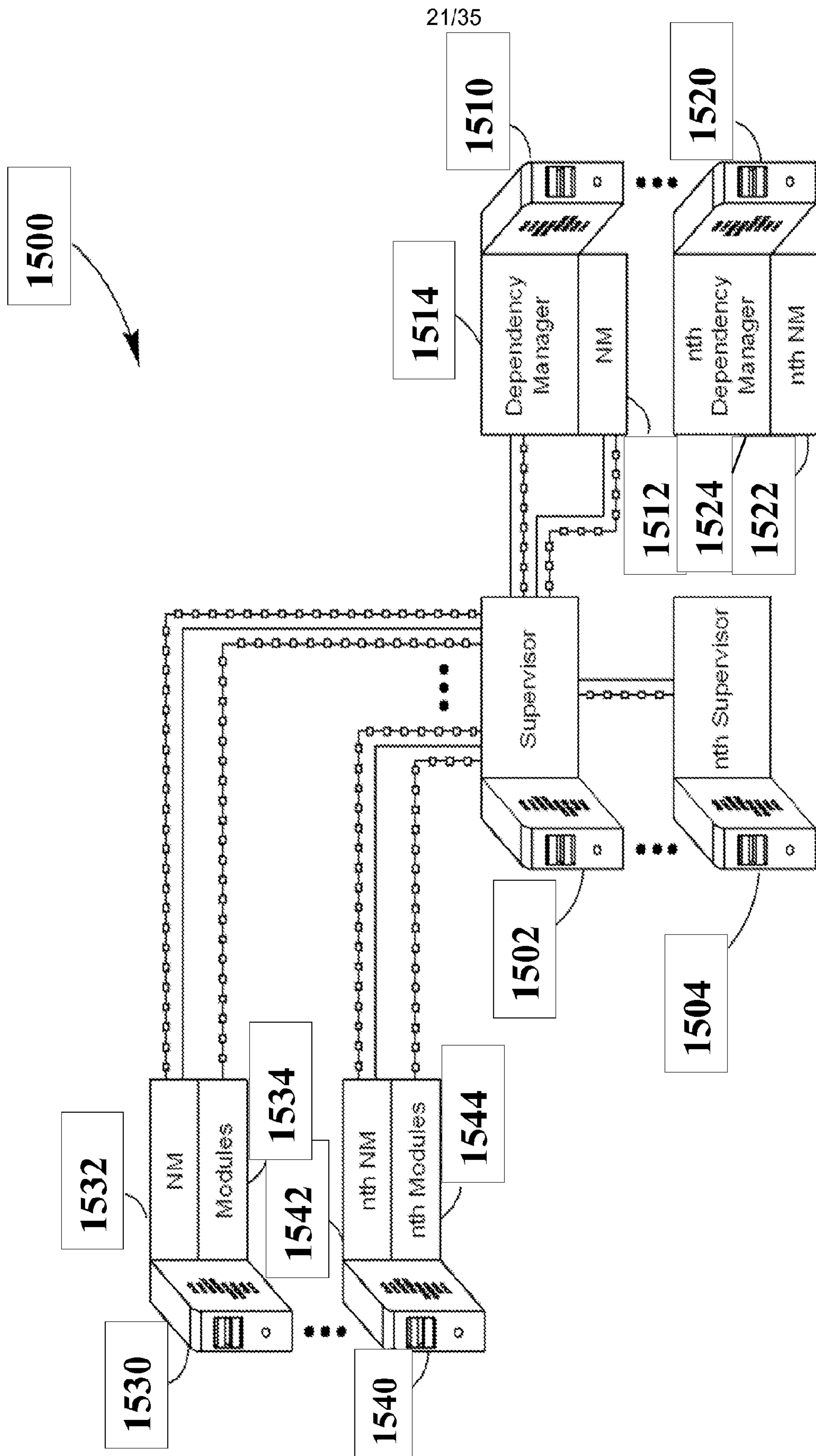


FIG. 15

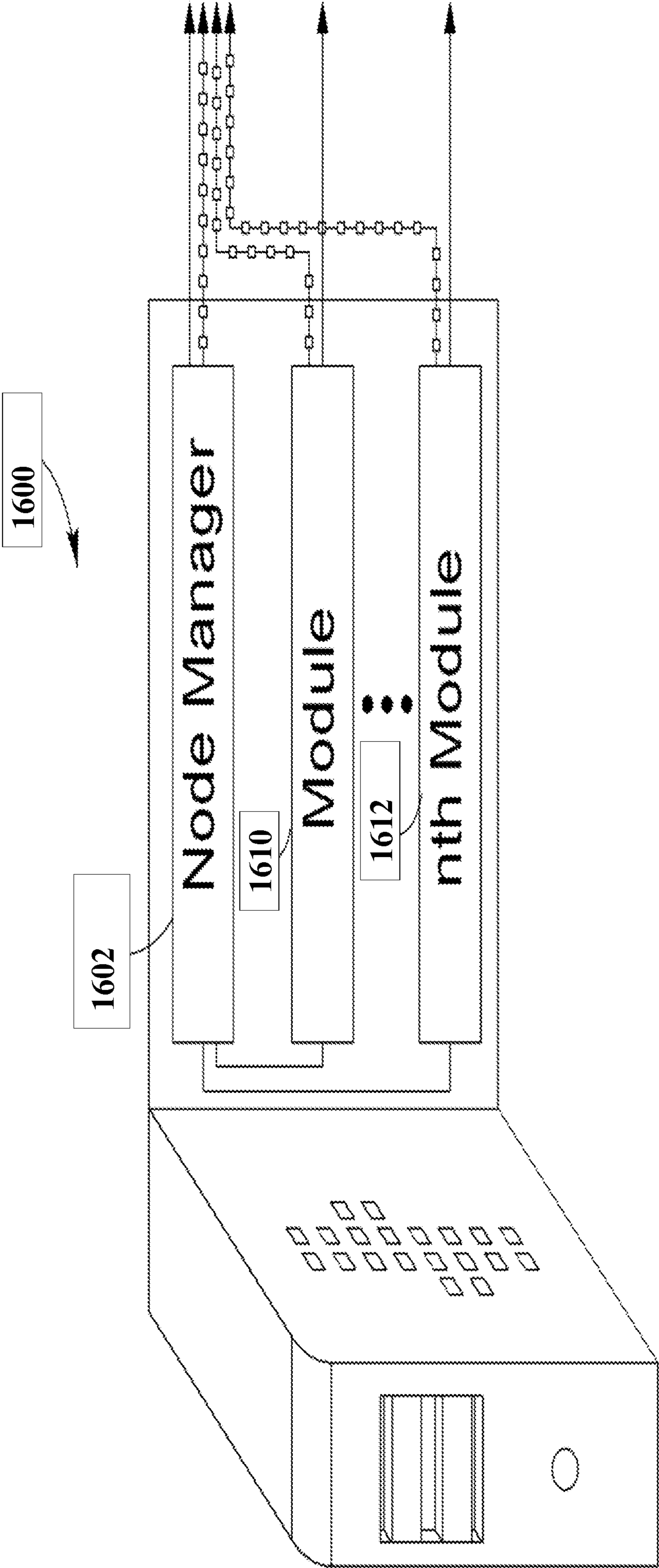
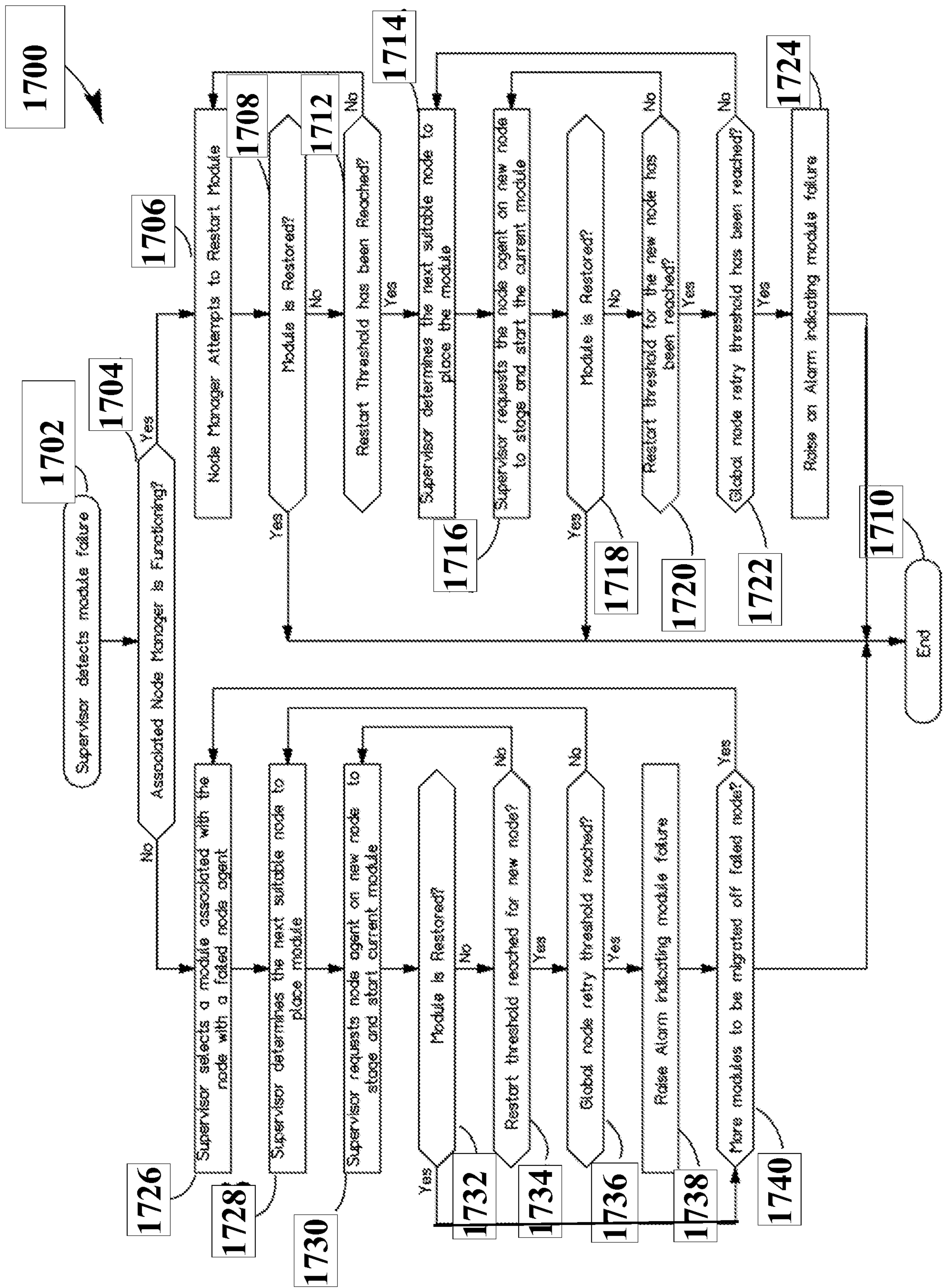


FIG. 16



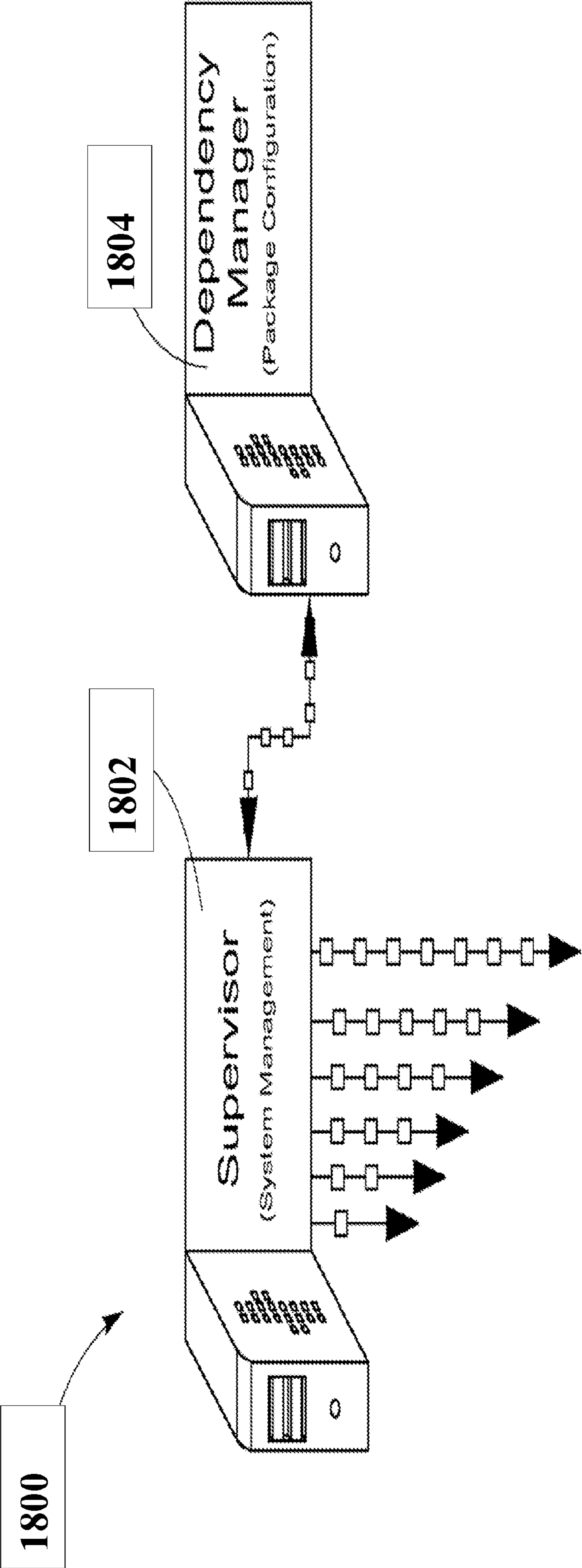


FIG. 18

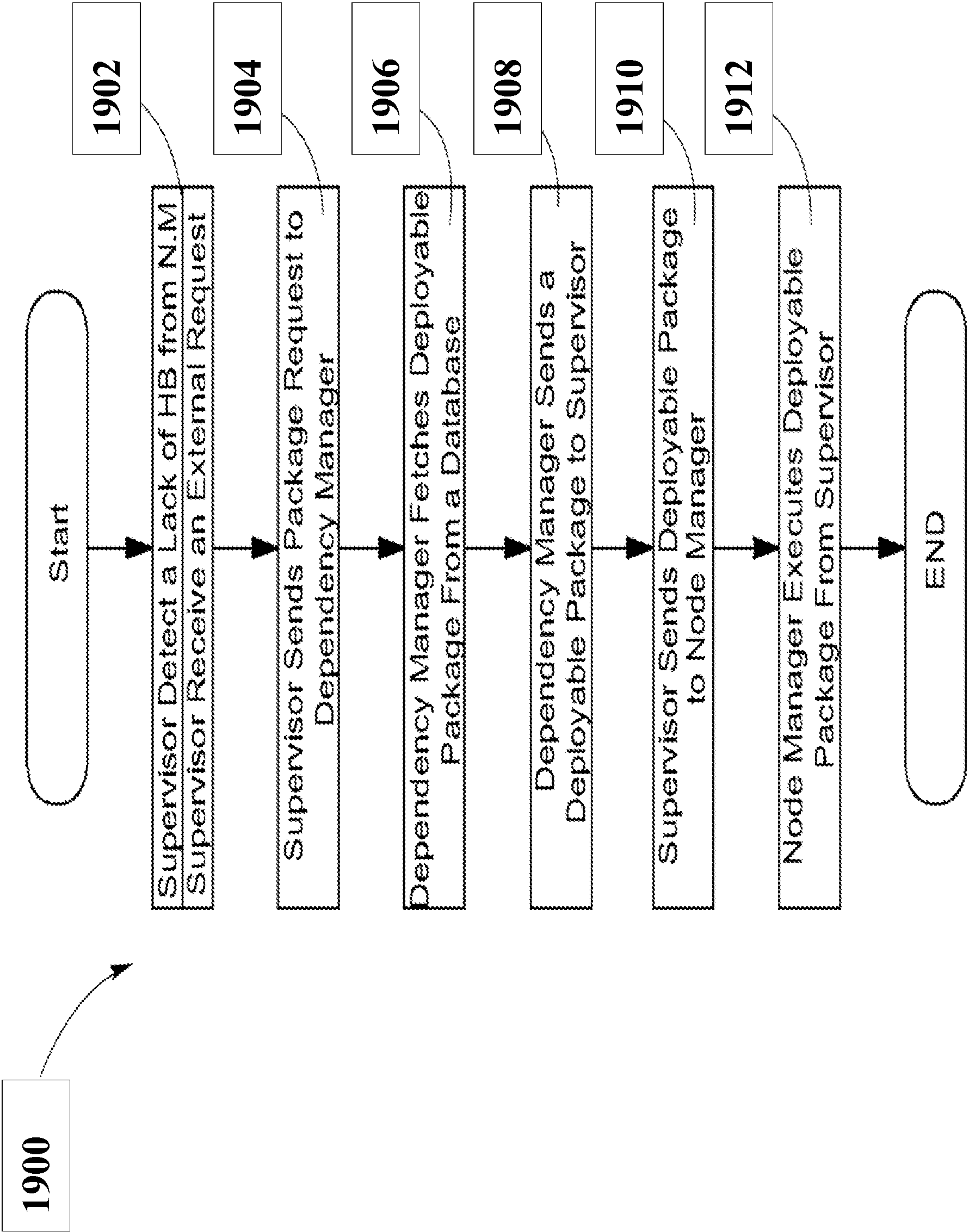


FIG. 19

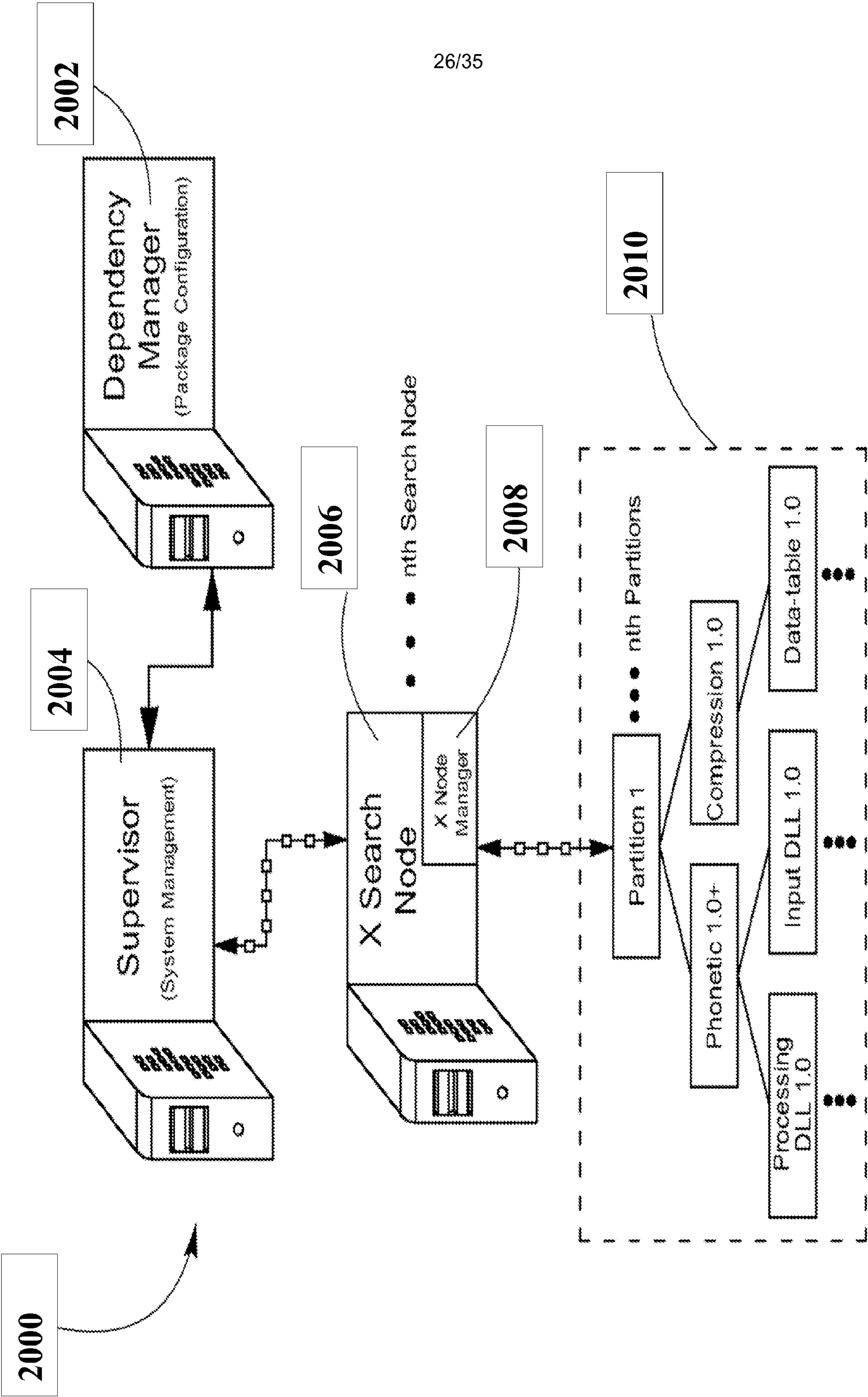


FIG. 20

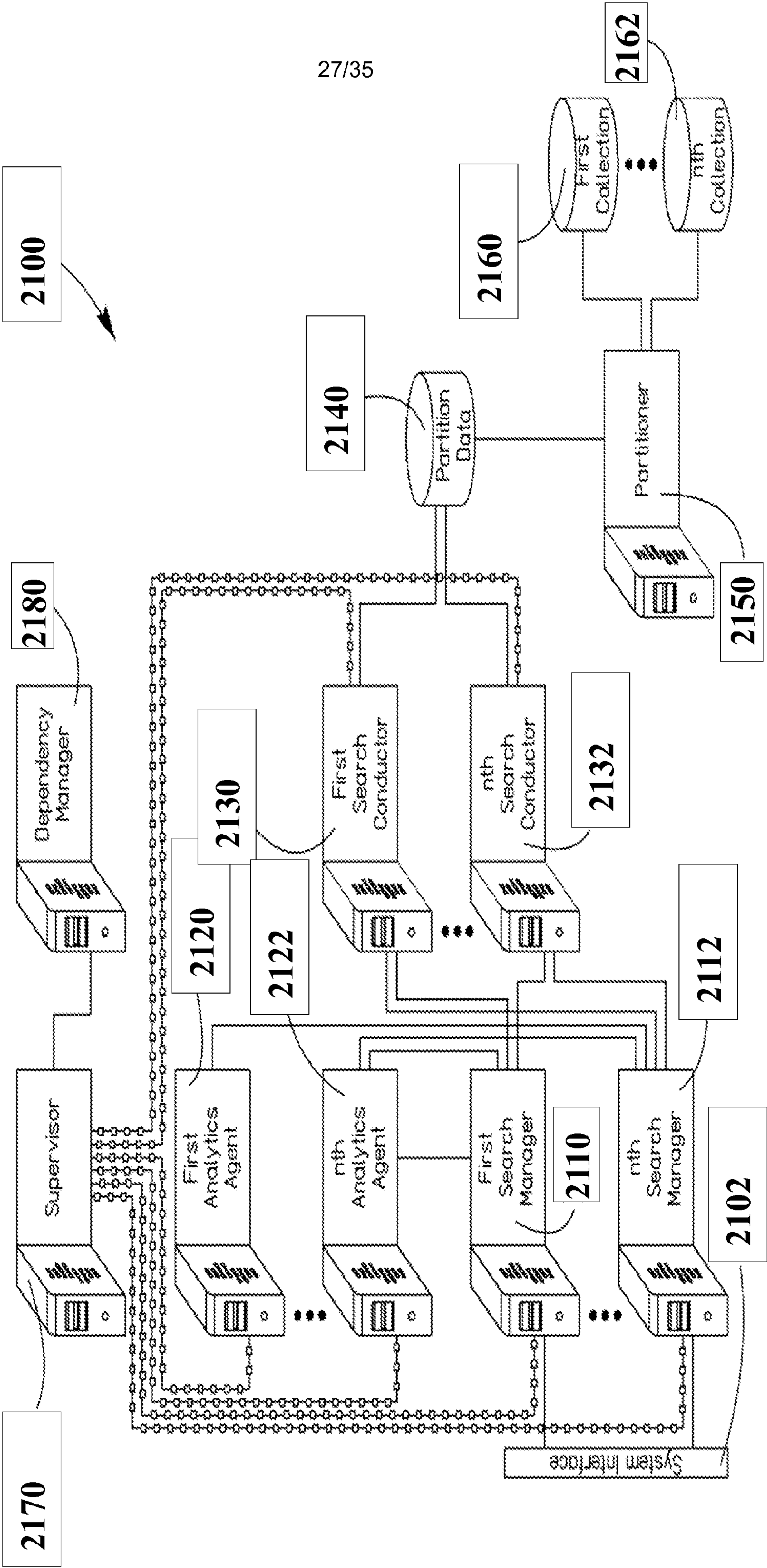


FIG. 21

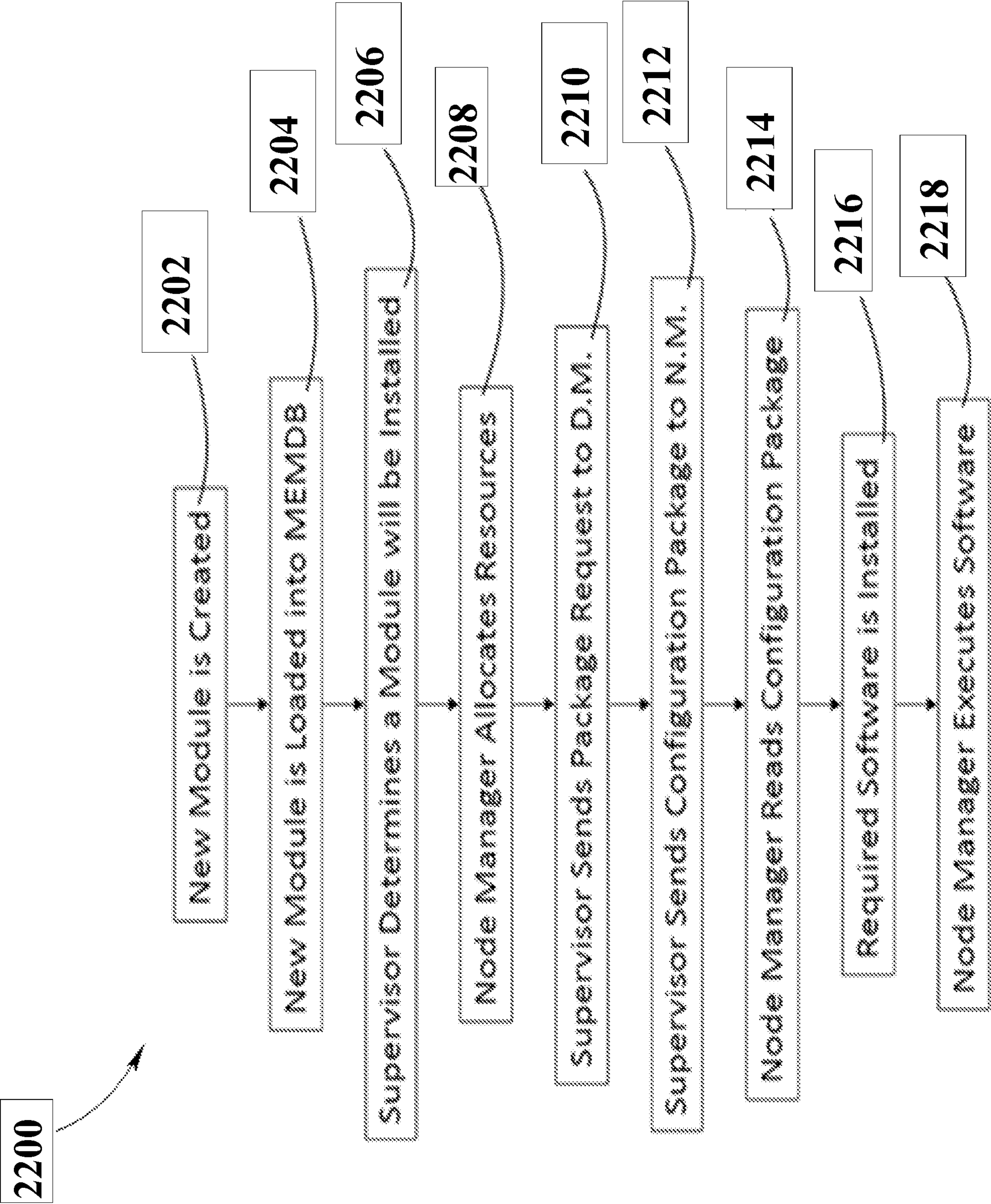


FIG. 22

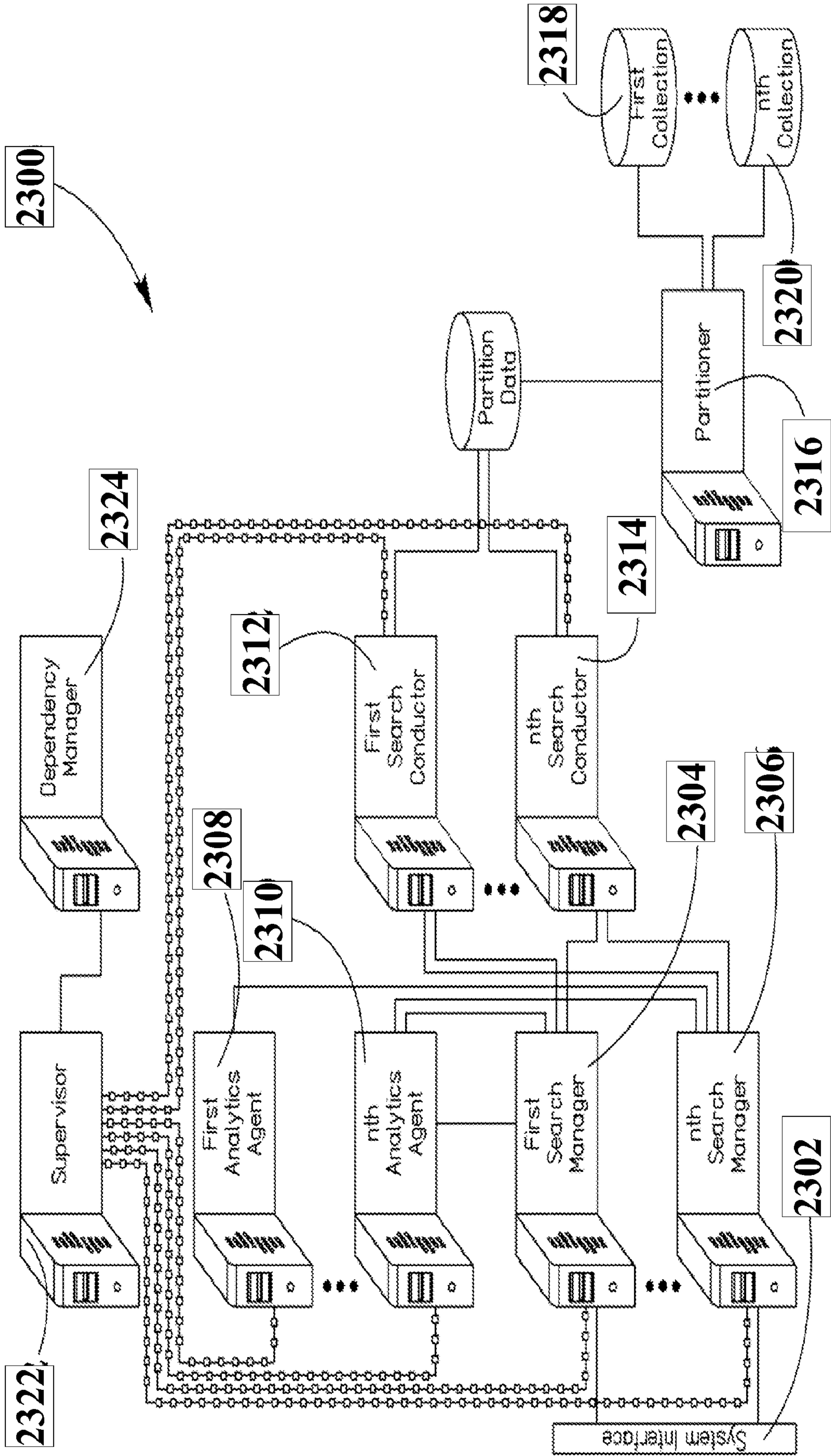


FIG. 23

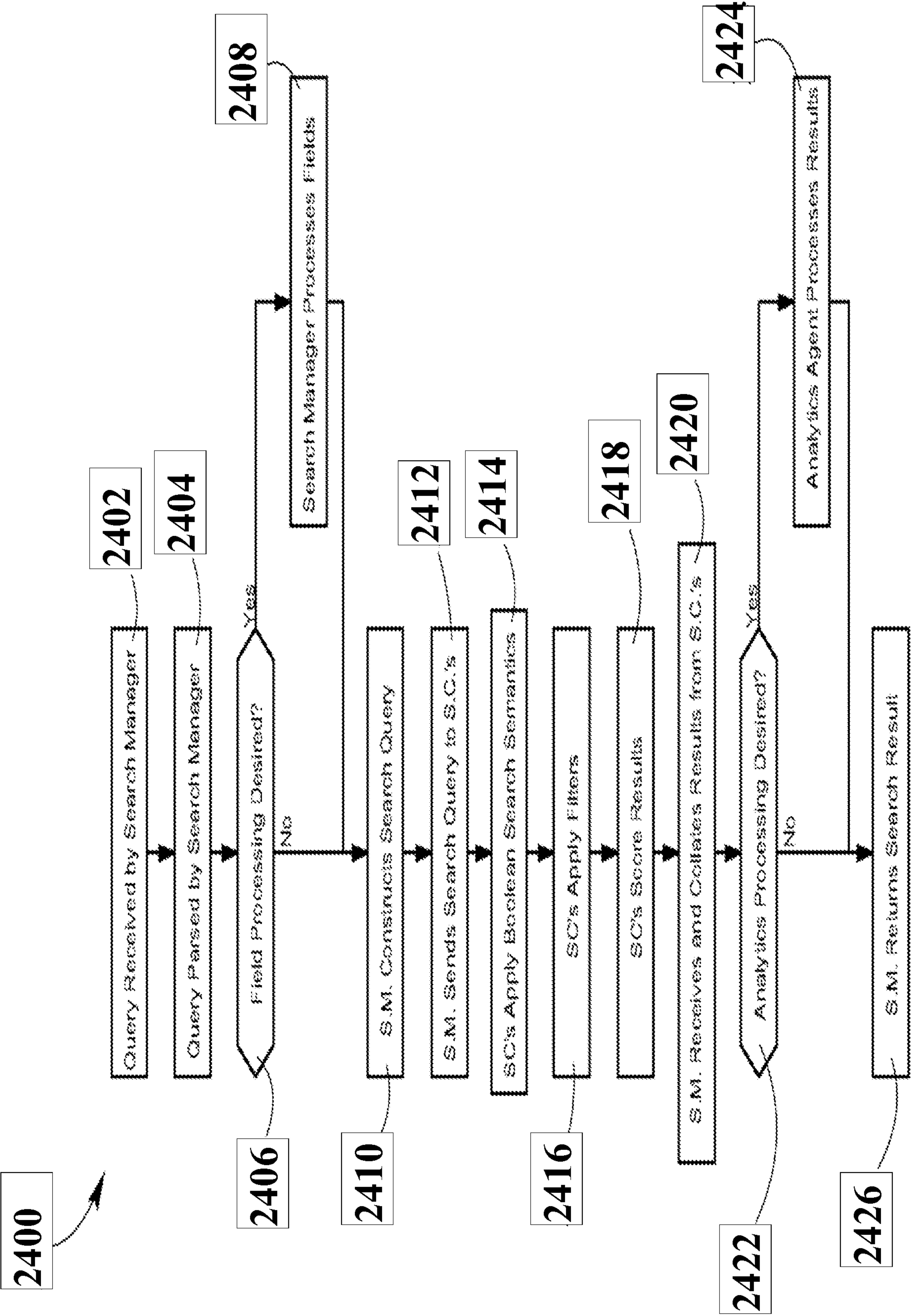


FIG. 24

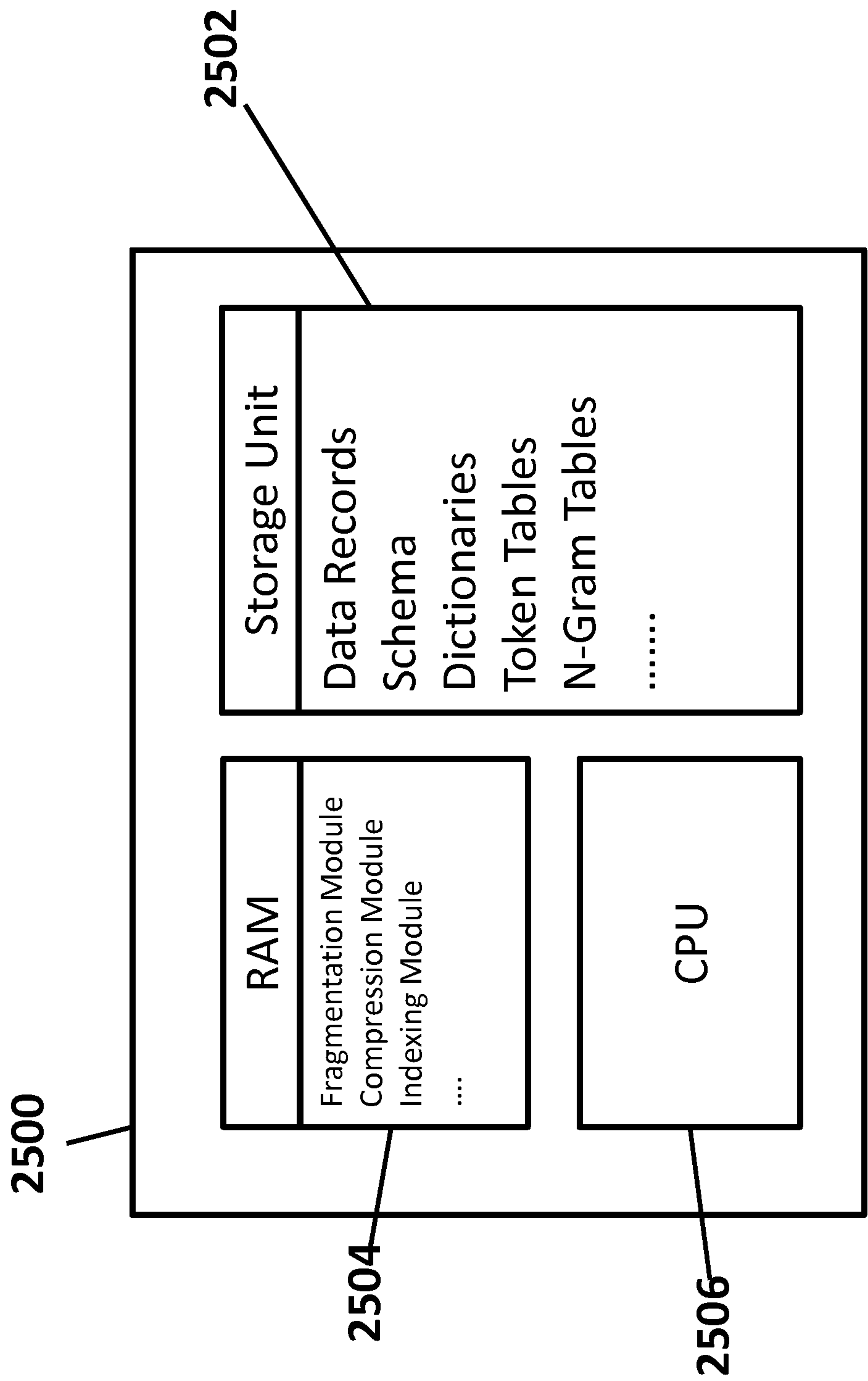


FIG. 25

<u>Index</u>	<u>Node 1</u>	<u>Node 2</u>	<u>...</u>	<u>Node N</u>
<u>0</u>	<u>Value 0.1</u>	<u>Value 0.2</u>	<u>...</u>	<u>Value 0.N</u>
<u>1</u>	<u>Value 1.1</u>	<u>Value 1.2</u>	<u>...</u>	<u>Value 2.N</u>
<u>...</u>	<u>...</u>	<u>...</u>	<u>...</u>	<u>...</u>
<u>x</u>	<u>Value x.1</u>	<u>Value x.2</u>	<u>...</u>	<u>Value x.N</u>

FIG. 26

<u>Index</u>	<u>Token Data</u>
<u>0</u>	<u>Value 0</u>
<u>1</u>	<u>Value 1</u>
<u>...</u>	<u>...</u>
<u>X</u>	<u>Value 2</u>
<u>zzz</u>	<u>No Token Value</u>

FIG. 27

<u>Index</u>	<u>N-gram Data</u>
<u>0</u>	<u>Value 0</u>
<u>1</u>	<u>Value 1</u>
<u>...</u>	<u>...</u>
<u>X</u>	<u>Value 2</u>
<u>zzz</u>	<u>No N-gram Value</u>

FIG. 28

<u>Record Index</u>	<u>Record Attributes</u>	<u>Node 1</u>		<u>Node 2</u>		...	<u>Node N</u>	
		<u>ID and Attributes</u>	<u>Data</u>	<u>ID and Attributes</u>	<u>Data</u>		<u>ID and Attributes</u>	<u>Data</u>
<u>0</u>	<u>Byte(s)</u>	<u>Byte(s)</u>	<u>Value 0.1</u>	<u>Byte(s)</u>	<u>Value 0.2</u>	<u>...</u>	<u>Byte(s)</u>	<u>Value 0.N</u>
<u>1</u>	<u>Byte(s)</u>	<u>Byte(s)</u>	<u>Value 1.1</u>	<u>Byte(s)</u>	<u>Value 1.2</u>	<u>...</u>	<u>Byte(s)</u>	<u>Value 2.N</u>
<u>...</u>	<u>...</u>	<u>...</u>	<u>...</u>	<u>...</u>	<u>...</u>	<u>...</u>	<u>...</u>	<u>...</u>
<u>x</u>	<u>Byte(s)</u>	<u>Byte(s)</u>	<u>Value x.1</u>	<u>Byte(s)</u>	<u>Value x.2</u>	<u>...</u>	<u>Byte(s)</u>	<u>Value x.N</u>

FIG. 29

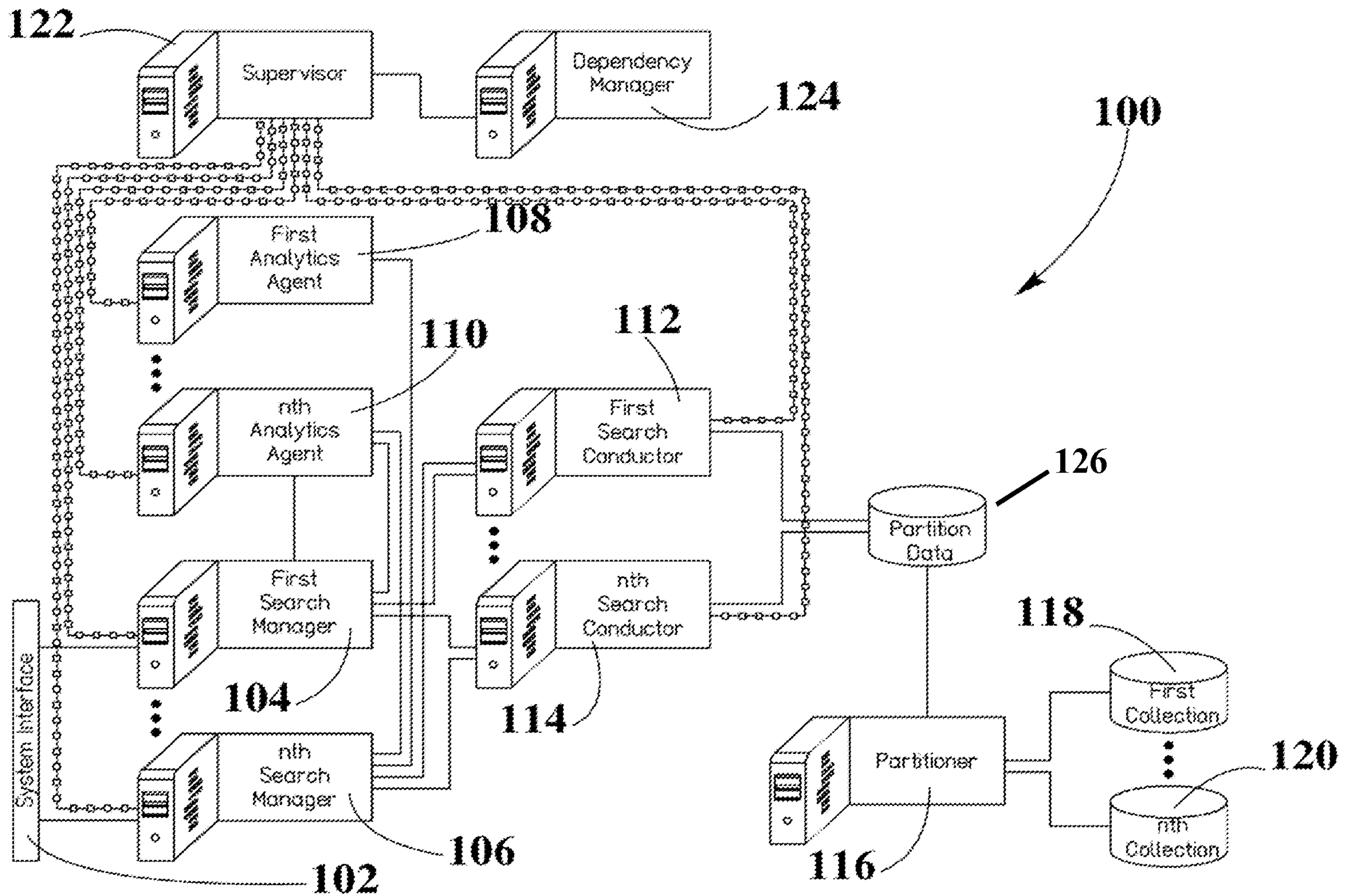


FIG. 1