



US 20060106848A1

(19) **United States**(12) **Patent Application Publication****Harvey et al.**(10) **Pub. No.: US 2006/0106848 A1**(43) **Pub. Date: May 18, 2006**(54) **METHOD AND APPARATUS FOR
HANDLING DIRECTORY OPERATIONS****Related U.S. Application Data**

(60) Provisional application No. 60/573,512, filed on May 21, 2004.

(76) Inventors: **Richard H. Harvey**, Ringwood (AU);
Damon N. Groenveld, Pheasant Creek
(AU); **Ronald W. Ramsay**, Ringwood
(AU)**Publication Classification**(51) **Int. Cl.**
G06F 17/00 (2006.01)
G06F 7/00 (2006.01)(52) **U.S. Cl.** **707/101**Correspondence Address:
COOPER & DUNHAM, LLP
1185 AVENUE OF THE AMERICAS
NEW YORK, NY 10036(57) **ABSTRACT**

A method for handling a directory service operation includes determining whether an alternate evaluator can respond to the operation and evaluating the operation using the alternate evaluator when it is determined that the alternate evaluator can respond to the operation.

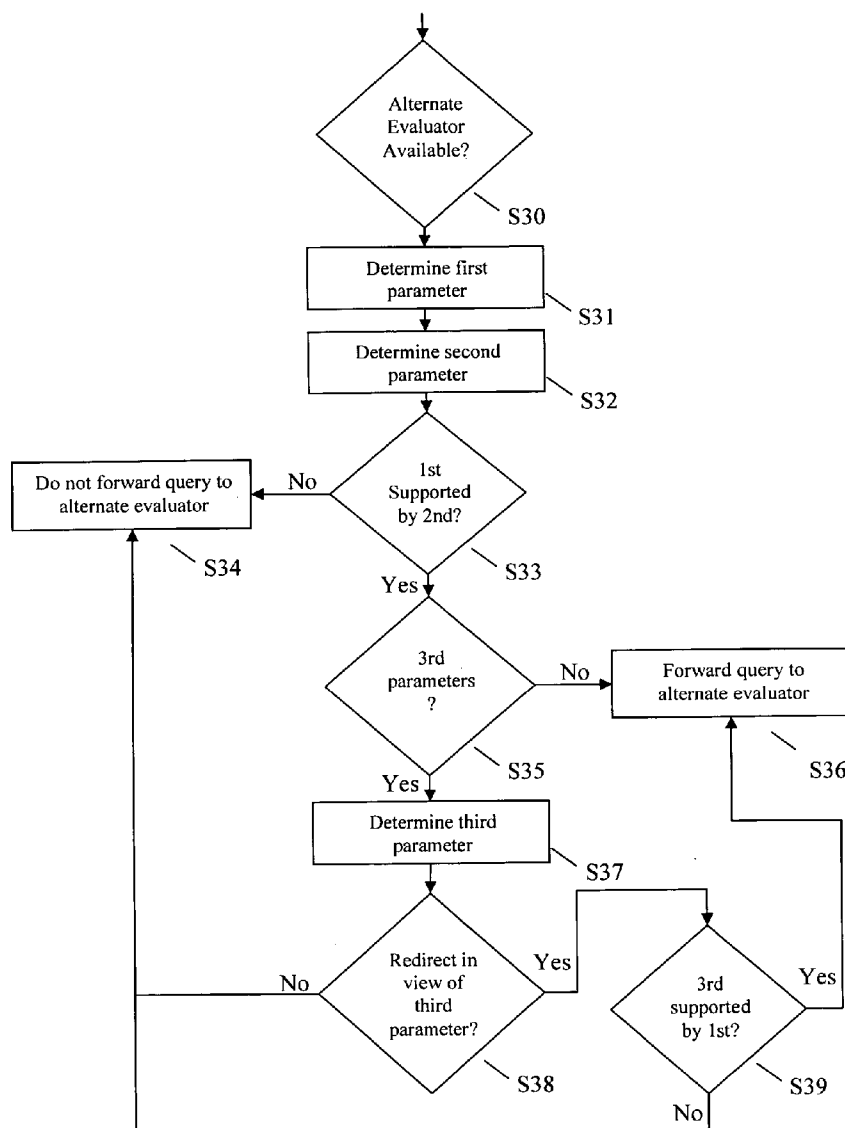
(21) Appl. No.: **11/134,251**(22) Filed: **May 20, 2005**

Fig. 1

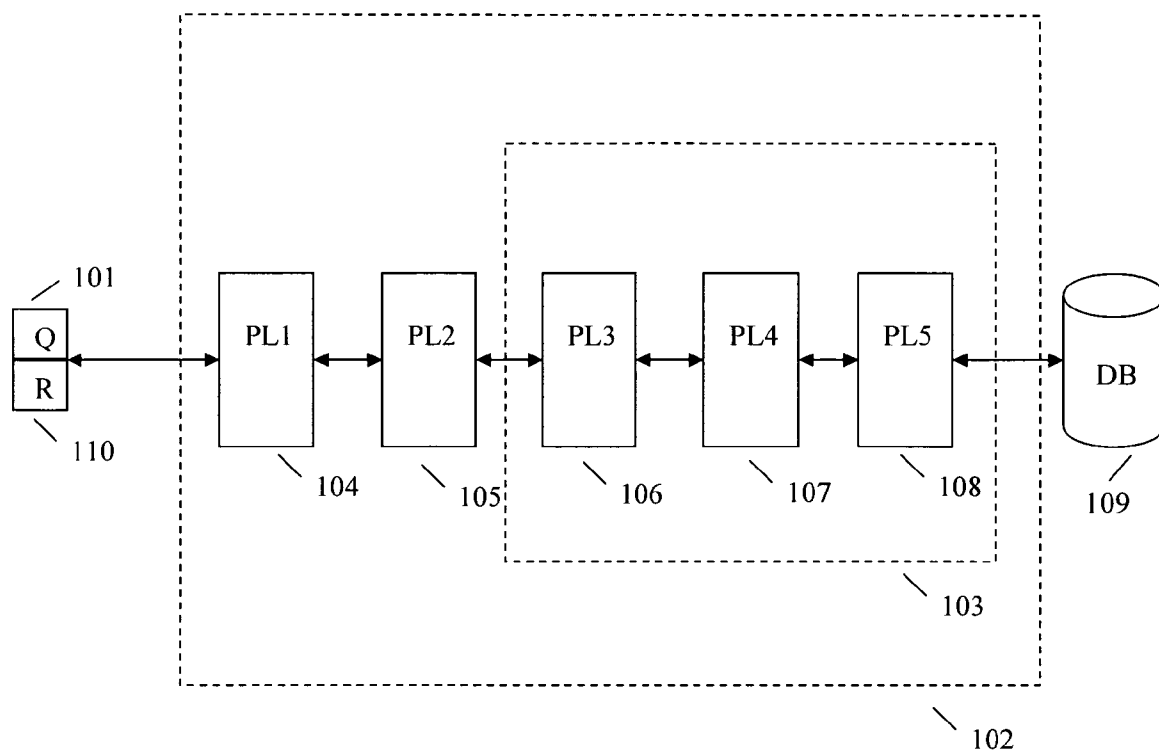


Fig. 2

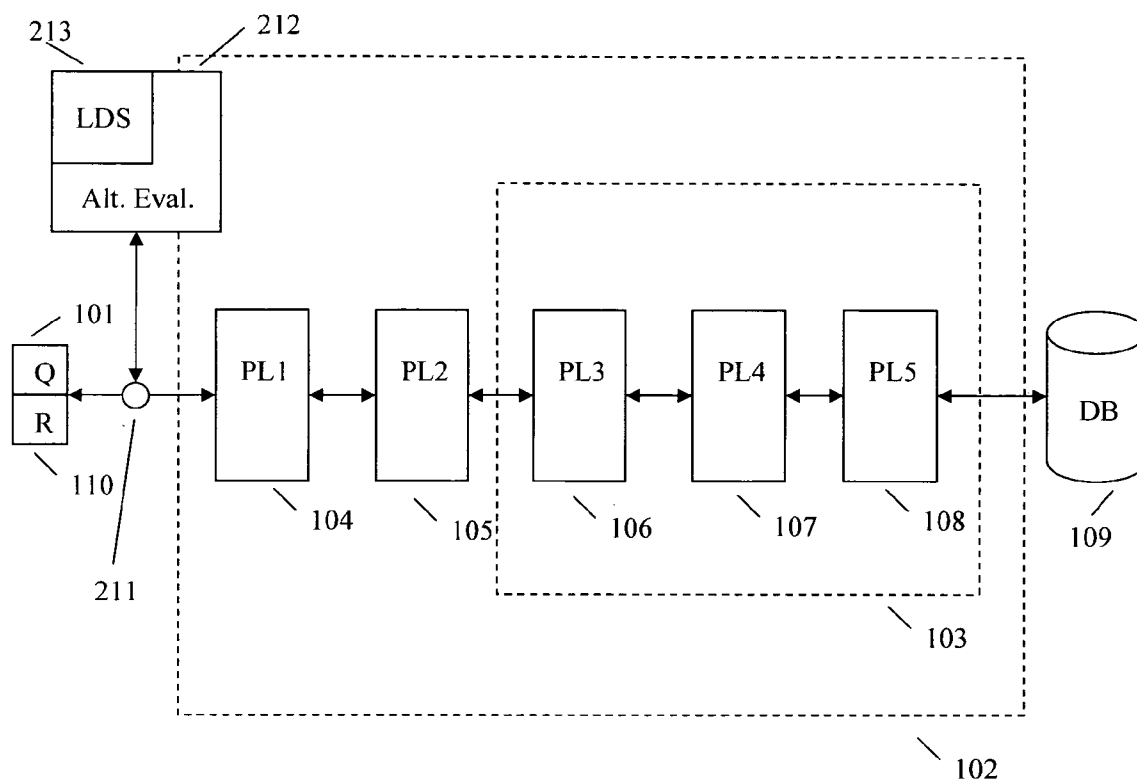


Fig. 3A

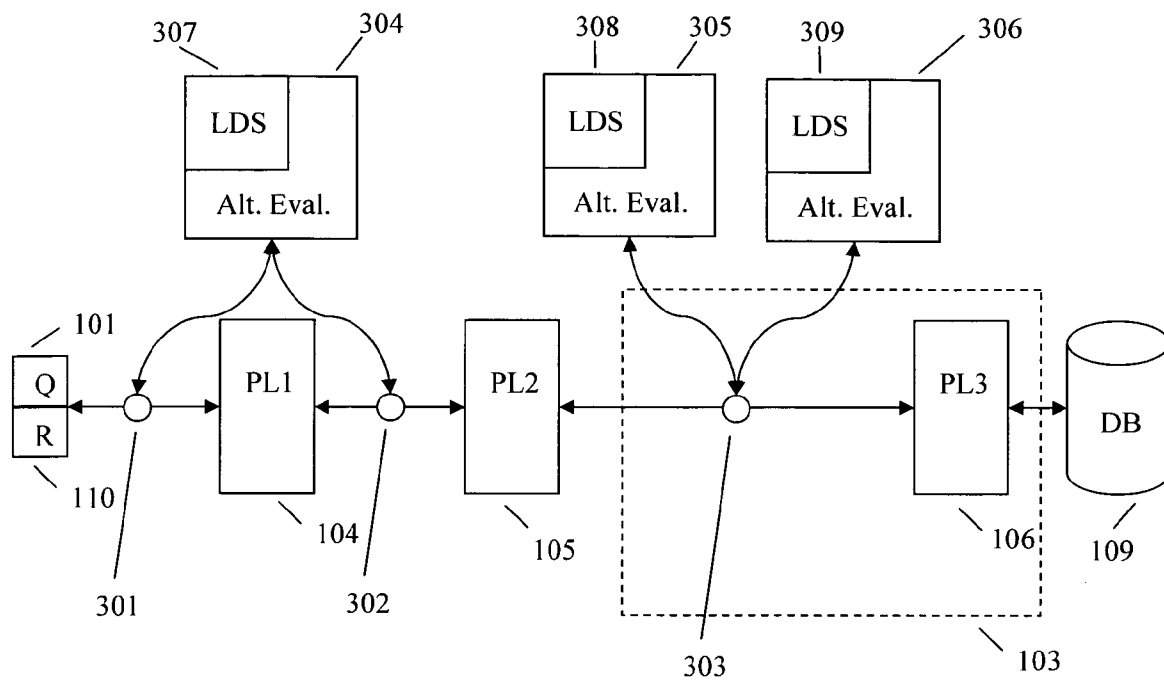


Fig. 3B

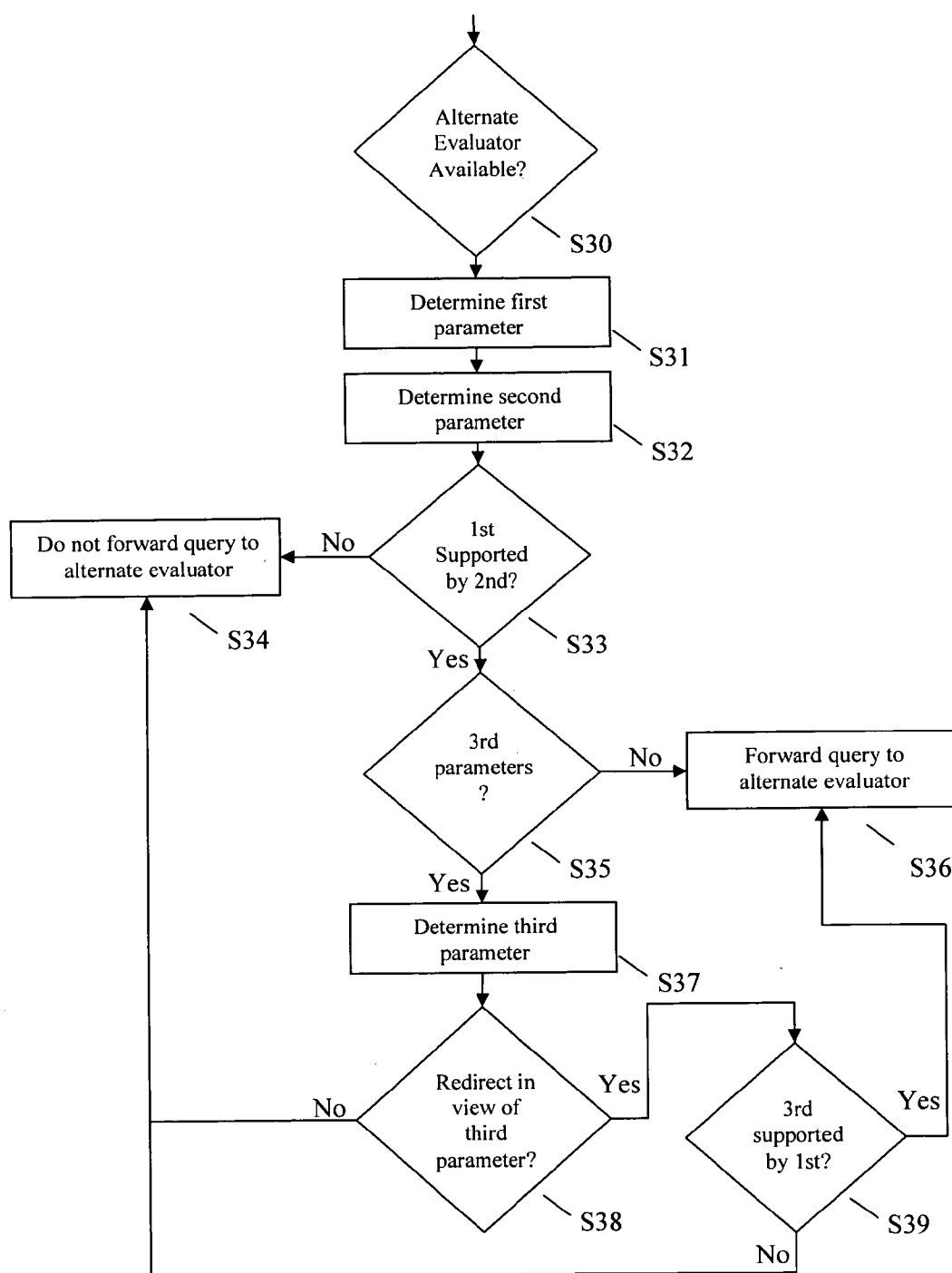


Fig. 4

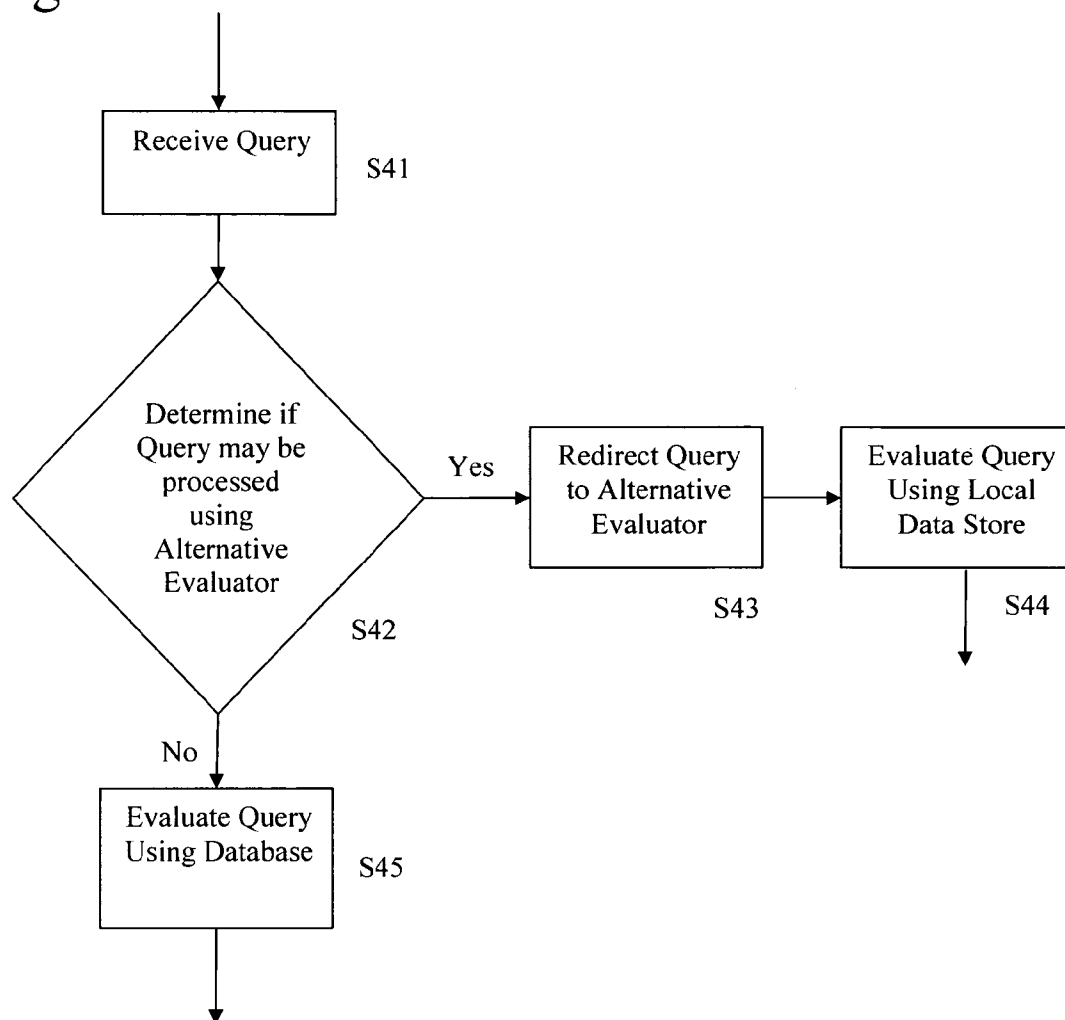


Fig. 5

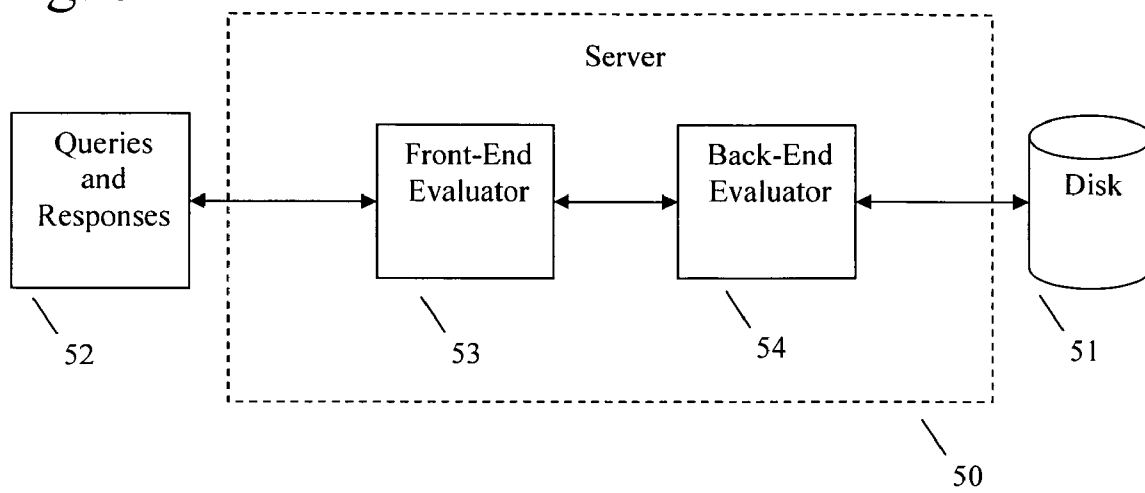


Fig. 6

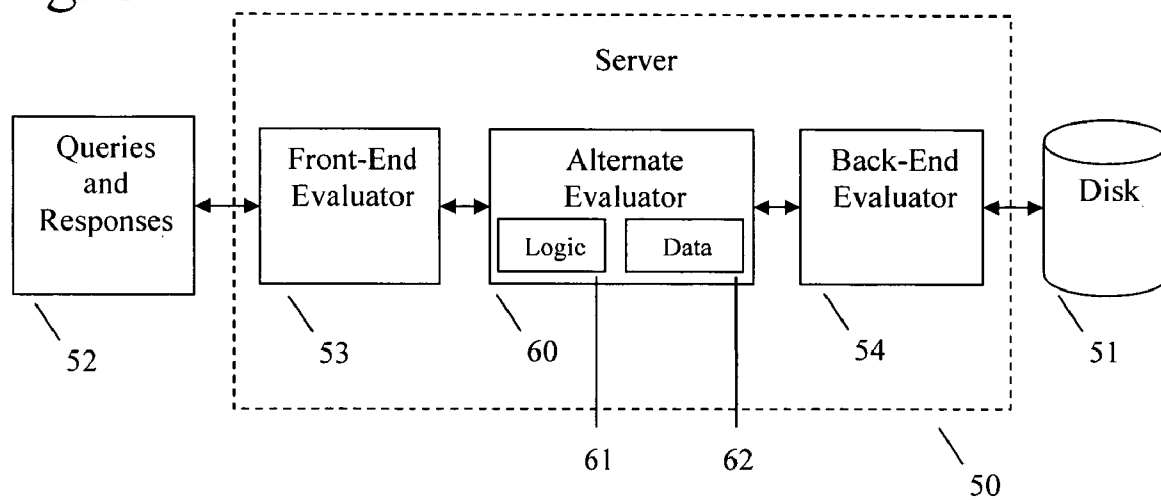


Fig. 7

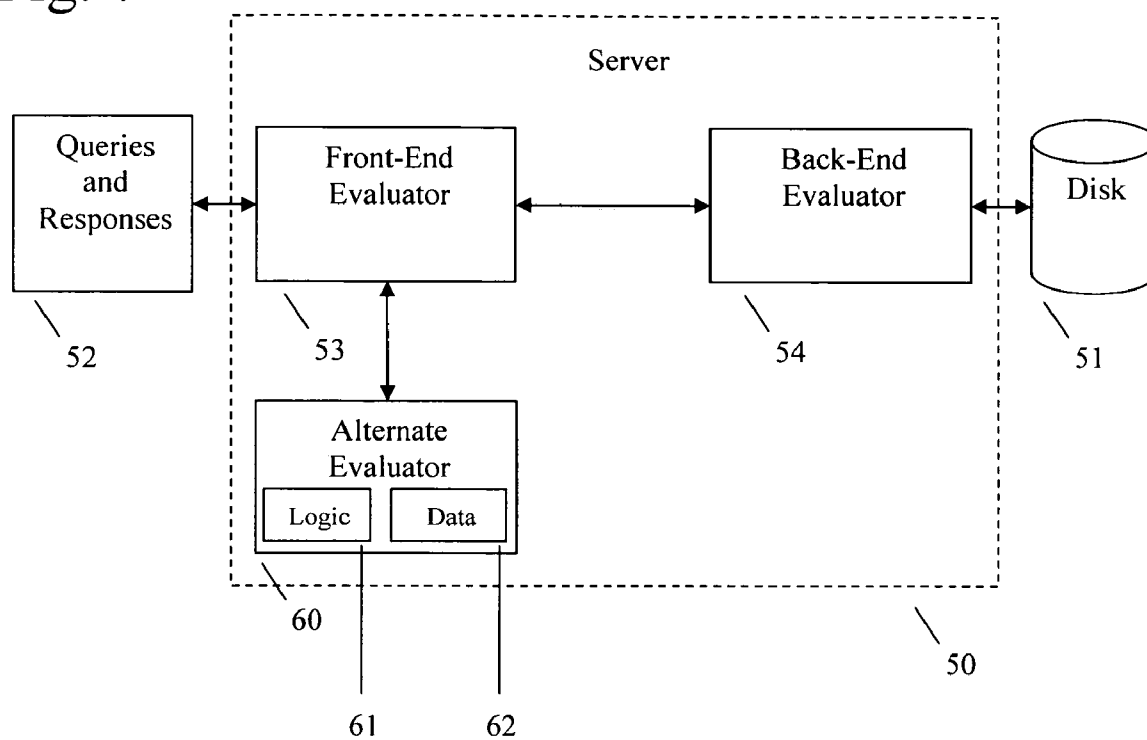


Fig. 8

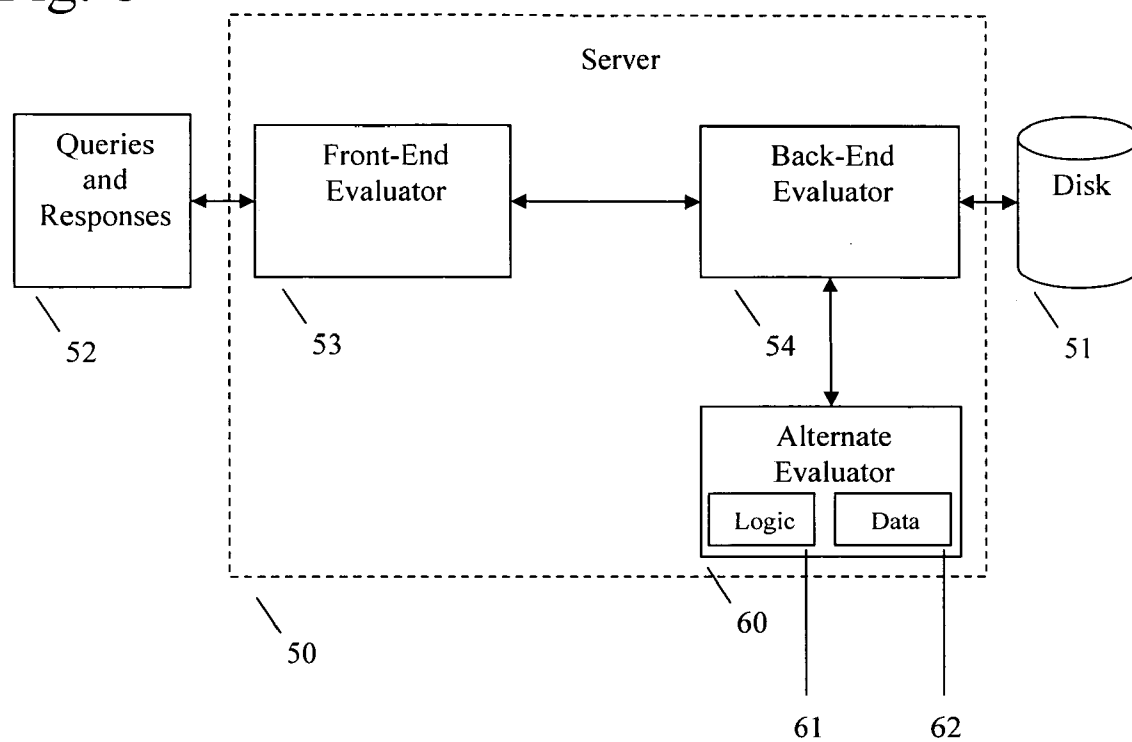


Fig. 9

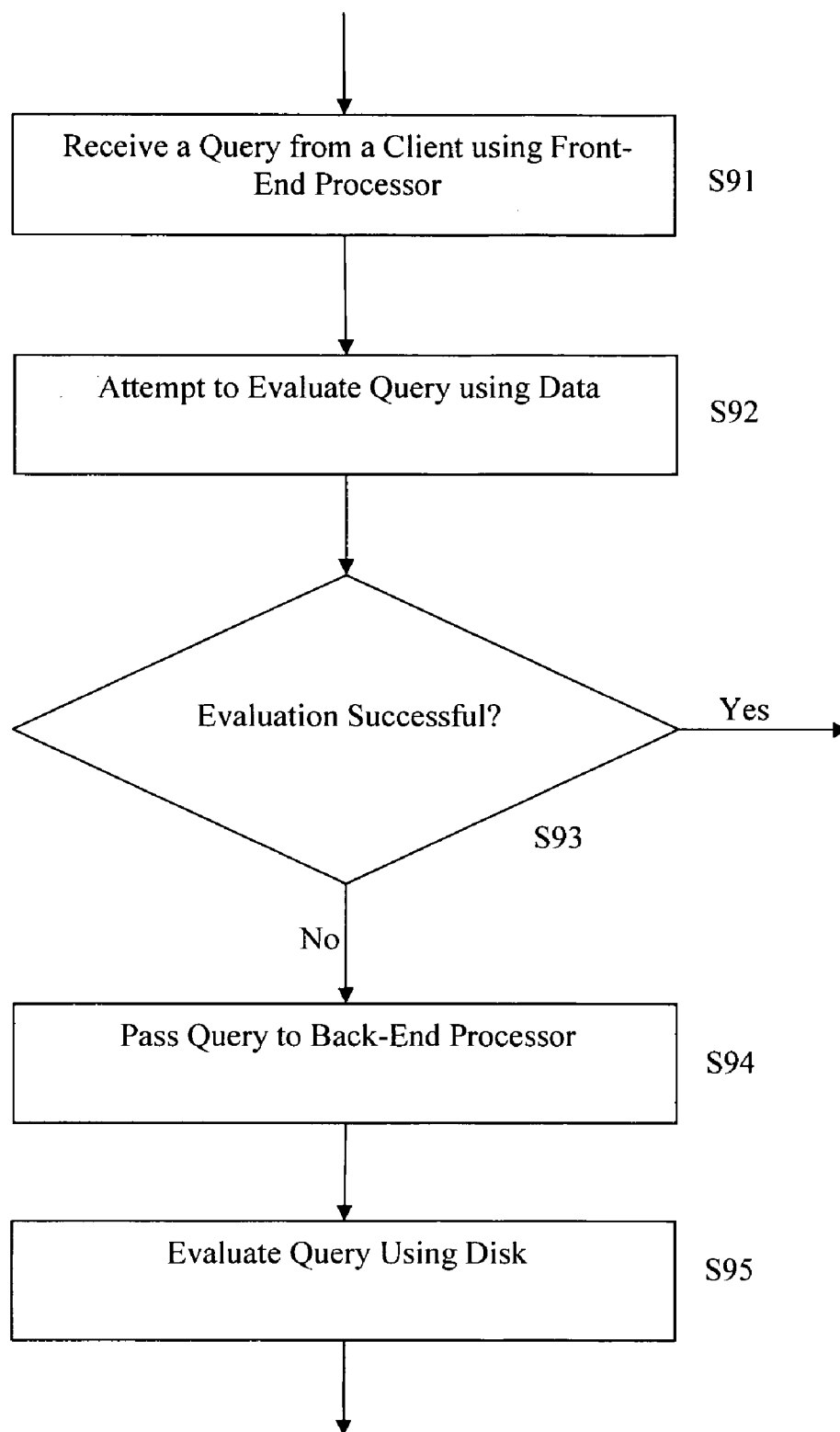


Fig. 10

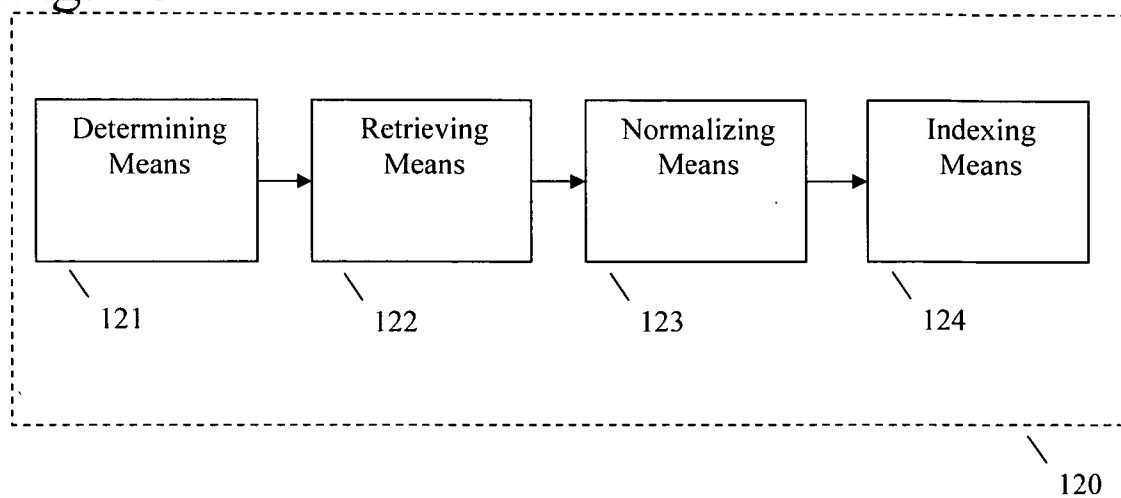


Fig. 11

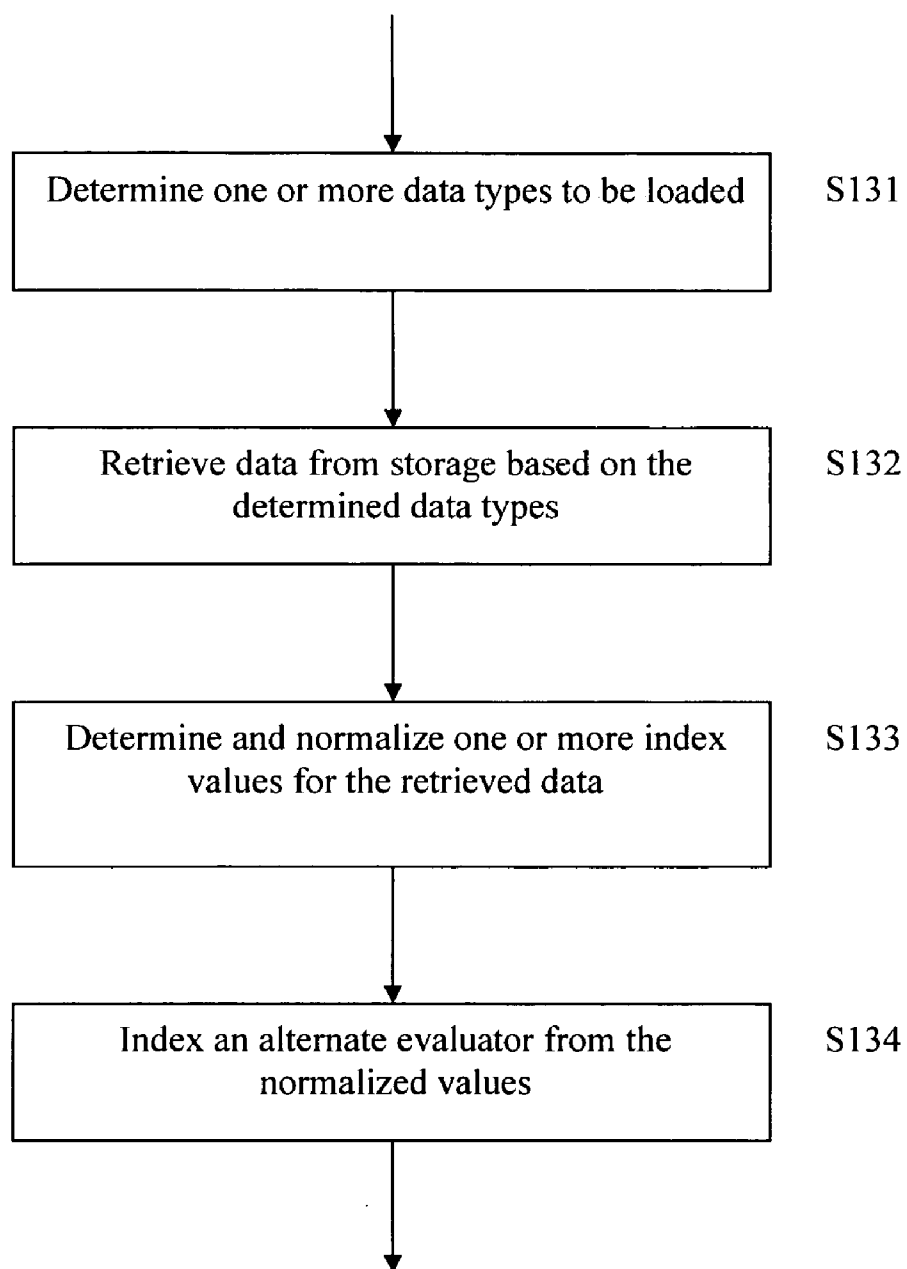


Fig. 12

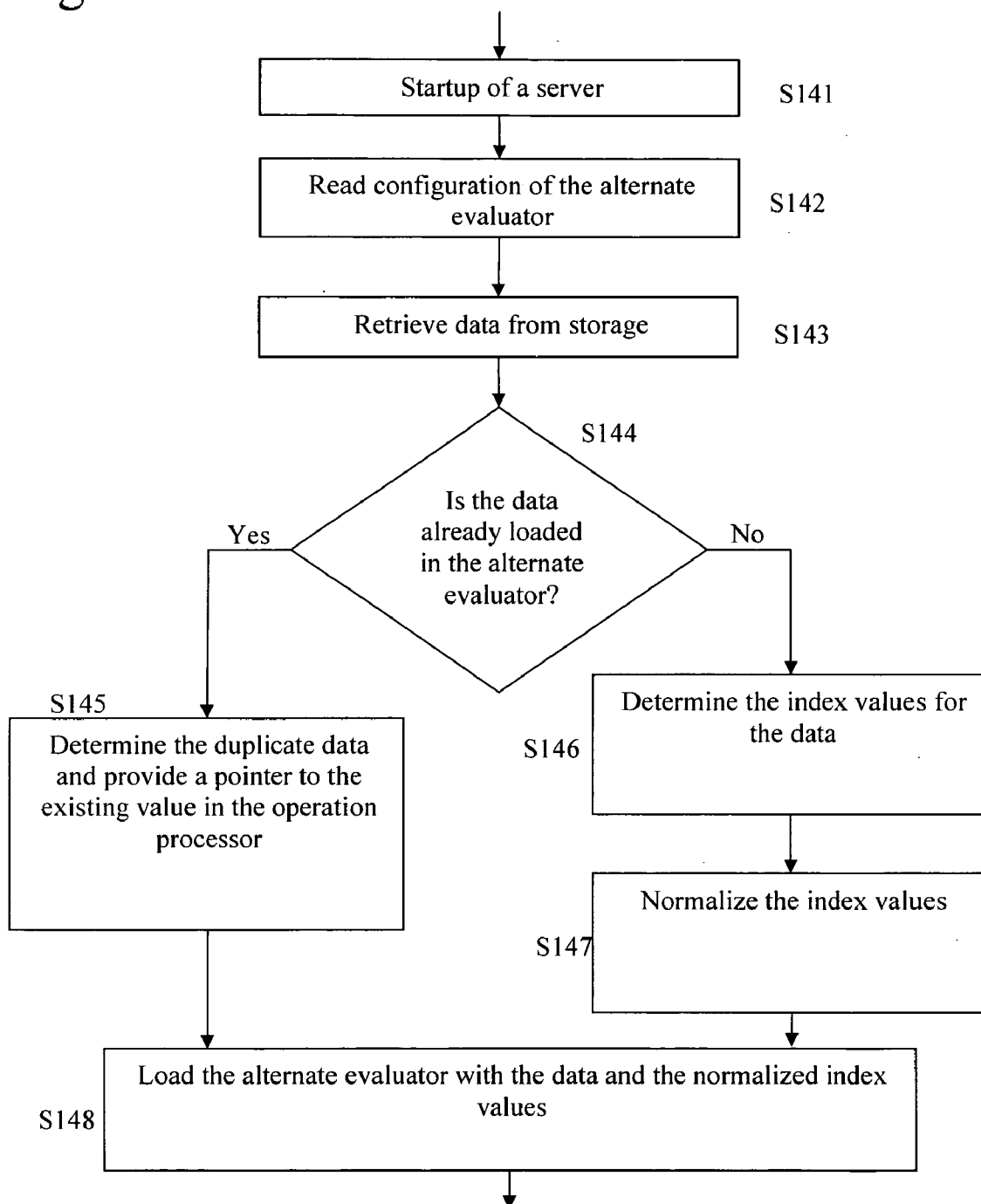


Fig. 13

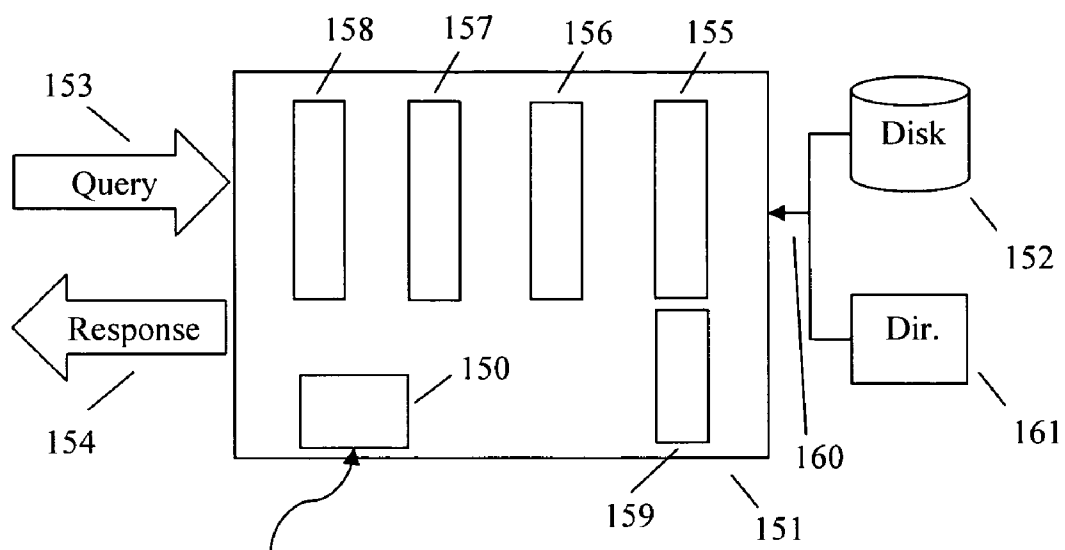


Fig. 14

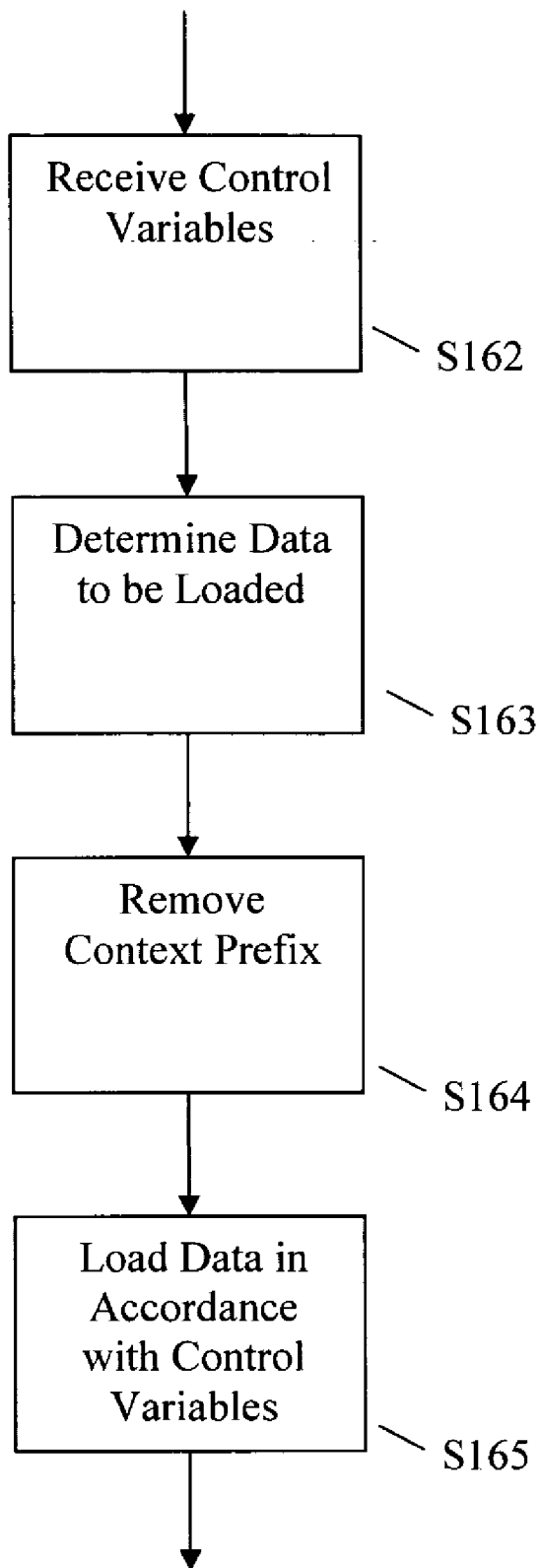


Fig. 15

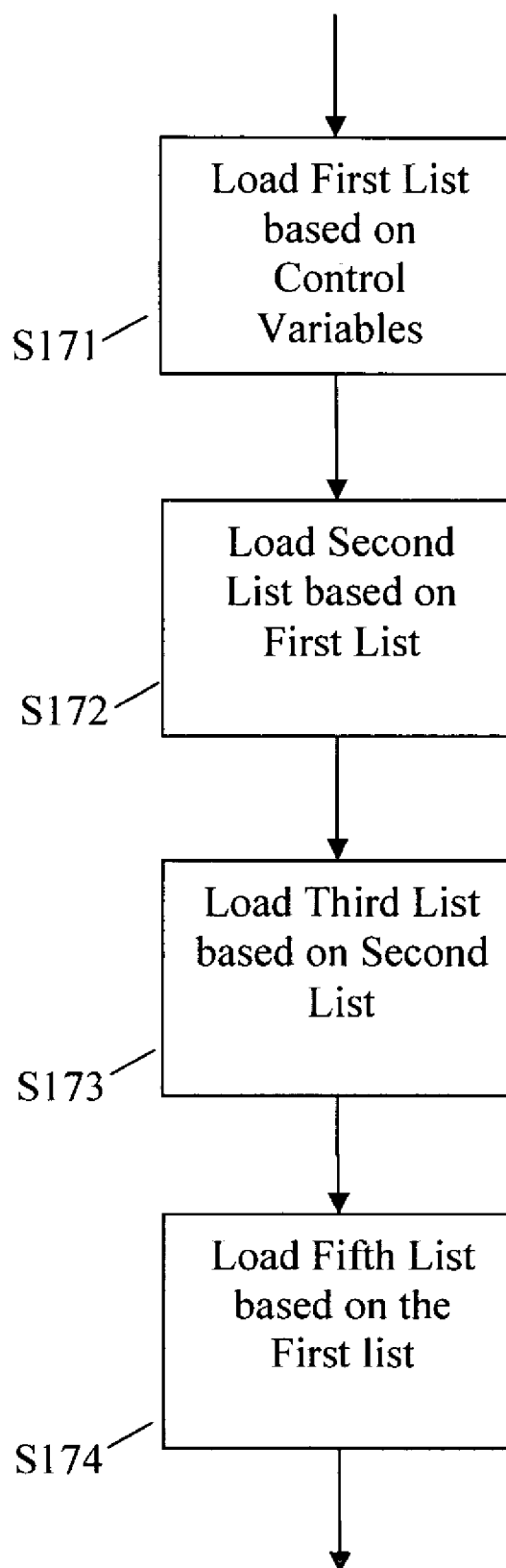


Fig. 16

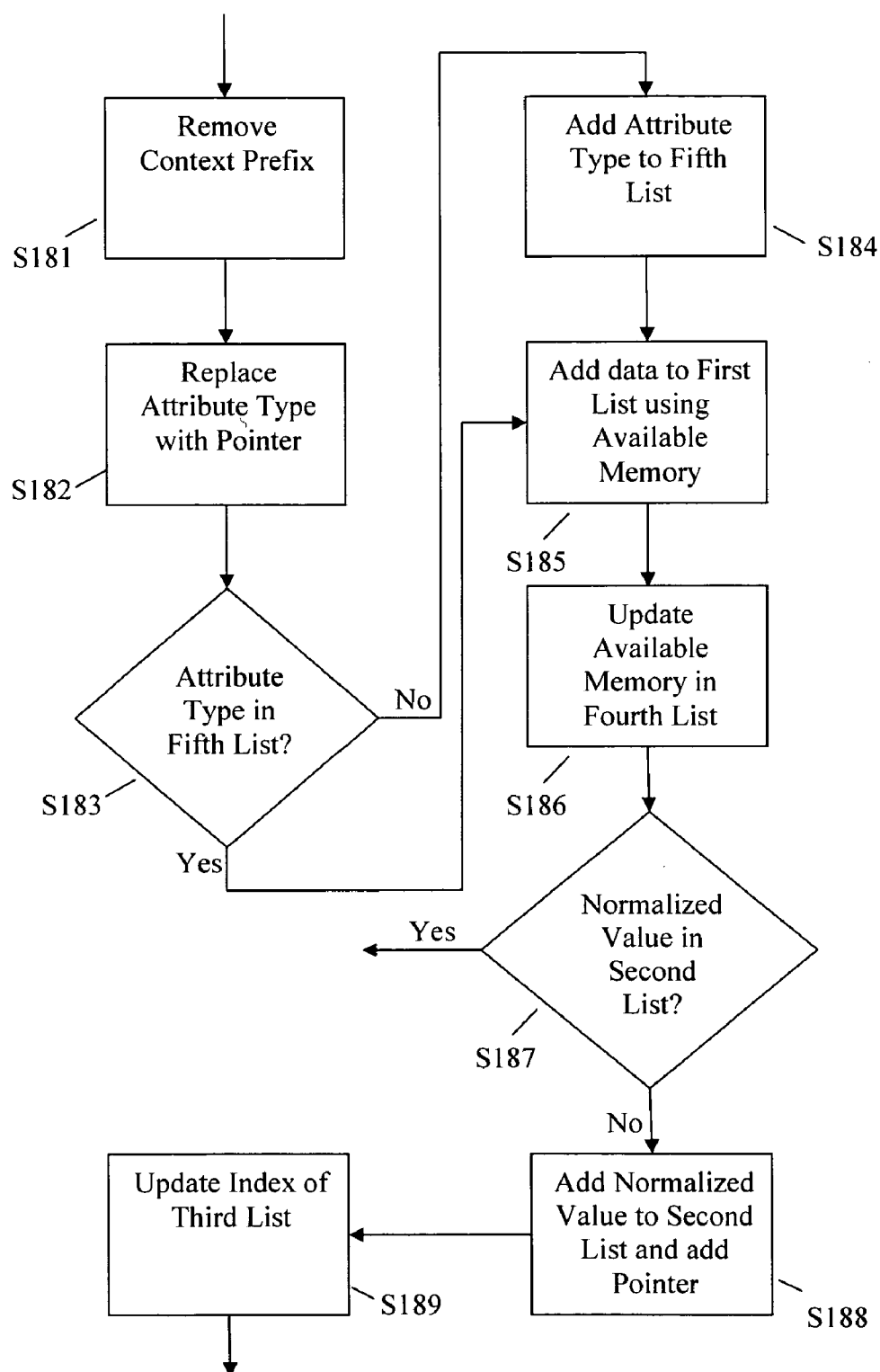


Fig. 17

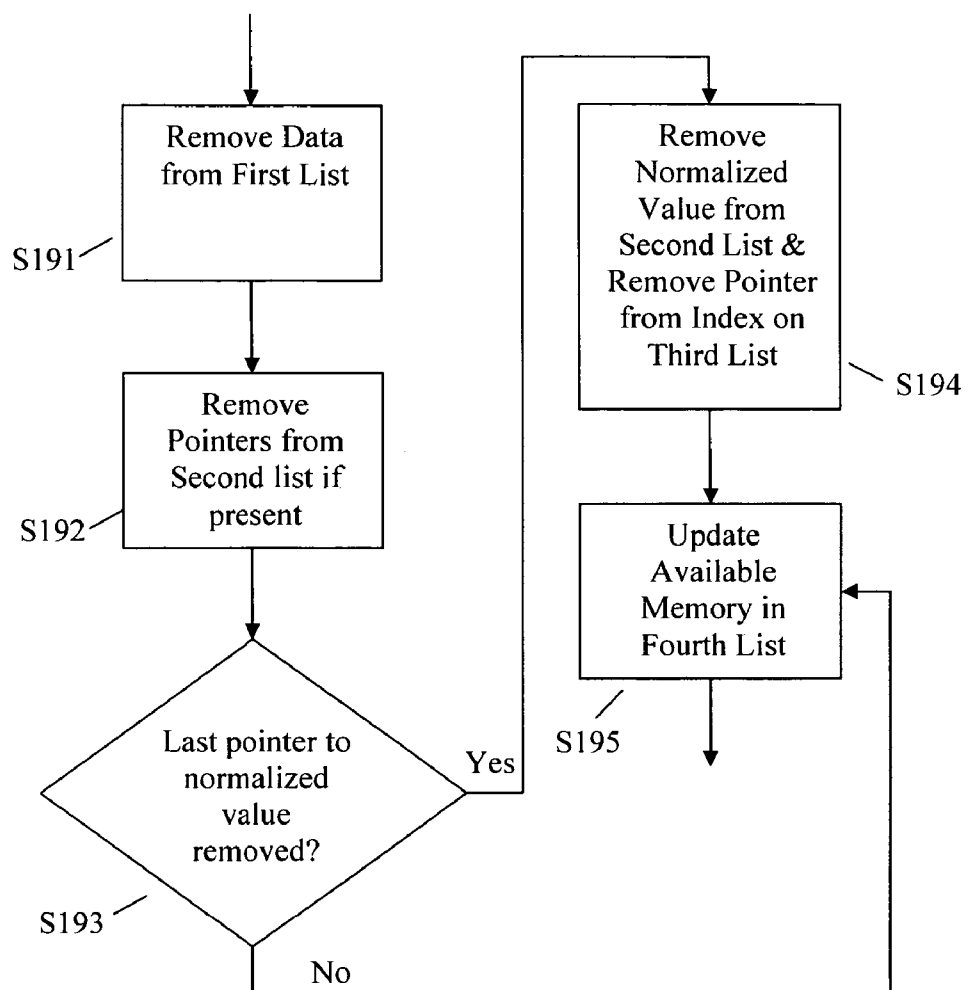
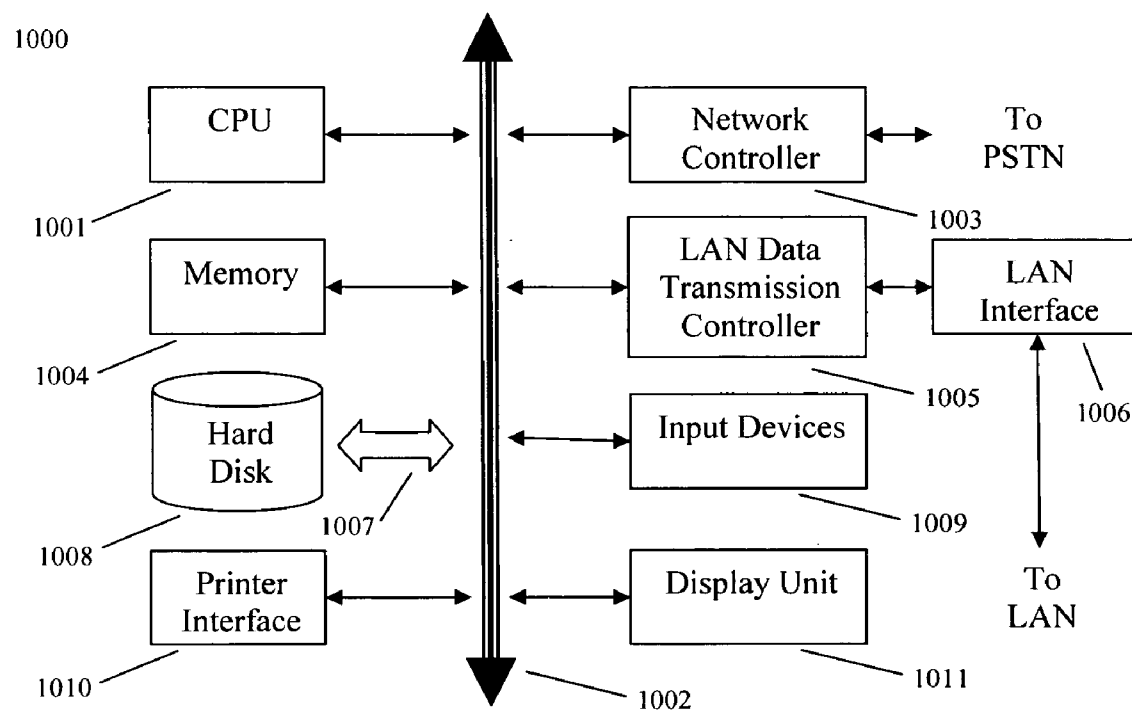


Fig. 18



METHOD AND APPARATUS FOR HANDLING DIRECTORY OPERATIONS

REFERENCE TO RELATED APPLICATION

[0001] The present application is based on provisional application Ser. No. 60/573,512, filed May 21, 2004, the entire contents of which are herein incorporated by reference.

BACKGROUND

[0002] 1. Technical Field

[0003] The present disclosure relates to directory operations and, more specifically, to a method and apparatus for handling directory operations.

[0004] 2. Description of the Related Art

[0005] Directories are information services that provide users and applications with quick access to information. Directories are optimized for information retrieval. A directory server is a computer application that provides directory service to users and applications. One or more directory servers may be used to provide directory service to one or more users and/or applications. By utilizing additional directory servers, directory service may be simultaneously provided to a large number of users and/or applications. These directory servers may be located together or they may be distributed. Directory servers may be globally distributed and/or replicated to provide efficient directory service world-wide over a global computer network such as the internet.

[0006] X.500 is an international set of standards that defines directories. X.500 includes protocols for allowing users and applications to access directory service. Other protocols may also be used for allowing users and applications to access directory service. For example, LDAP, or the Lightweight Directory Access Protocol, is a string based protocol. Another example is Directory Services Markup Language (DSML) which is a variant of Extensible Markup Language (XML).

[0007] Directory protocols usually utilize TCP/IP transfer services or similar transfer services. Directory protocols may also include internal security features that may be useful for securing directory service access. Directory protocols are therefore well suited for providing access to directory service over the internet or private company intranets.

[0008] Users and applications may utilize directory services to gain access to one or more data objects. The data objects may have one or more attributes and the attributes may contain one or more values.

[0009] The hierarchical arrangement of directory information may provide for efficient searching. Hierarchical objects are commonly referred to as parent objects and child objects depending on their relationship to one another. For example, where a computer and a printer are represented as data objects, an object representing a printer may be the child of an object representing a computer.

[0010] The speed and efficiency of directories are important aspects of overall directory performance. Conventional methods for increasing directory performance involve

increased use of hardware, software, or a combination of both hardware and software. For example, additional CPUs, disks, memory, directory server processes, etc. may be utilized in order to improve response times, latency in a replicated environment, and/or throughput or volume of operations, etc. However, while these conventional methods do improve performance, they do so at a high cost in terms of hardware, software, maintenance, and operations. For example, complex design and configuration, such as, routing, security, administrative controls, etc., may be involved in addition to complex management of the system, including network, host administration, directory monitoring, distribution, replication, backup and recovery, etc.

[0011] Protocols for accessing directories define multiple operations that a user and/or application may perform. These operations may be categorized into three groups: queries, updates and connections. Queries are operations for retrieving information from a directory. Query operations include "read" for reading data, "list" for listing data, "search" for searching for data and "compare" for comparing data. Update operations include "add" for adding data, "remove" for deleting data, "modify" for changing data and "ModifyDN" for changing the distinguished name associated with data, i.e. renaming data.

[0012] A query operation may include one or more sub-queries. For example, a query may include a Navigation sub-query.

[0013] Conventional methods of handling operations in a directory typically require access to data generally stored in a non-volatile form, such as on a disk. Because the speed at which data may be accessed from disks and other non-volatile forms may be limited, caches may be used to speed up the retrieval of information. Caches are areas of memory, generally volatile memory, that are used to store previously accessed information so that subsequent access attempts may be rapidly accessed from the cache without having to go back to the disk itself.

[0014] Caches are generally incremental caches, copying the most recently accessed information from the disk to the cache. Caches improve access speed because commonly accessed information is likely to have been placed in the cache from the last time that information was accessed. Unfortunately, incremental caches may take time to load commonly used information to the cache as information is only loaded into the cache as that information is accessed from the disk. Additionally, because it is not always clear when a cache is fully loaded or what types of information are stored in the cache, it might not be possible to effectively optimize operations such as queries for use with the cache. Additionally, if the sizes of the incremental caches are set too high, the stability of the caches may be reduced.

[0015] There are also time constraints associated with the incremental cache; for example, existing systems may handle at most only hundreds of operations per second. Customers of large organizations may demand greater throughput and speed of operations. For example, a directory operation that is evaluated may involve many time-consuming steps, such as, decoding the protocol, converting to SQL, interpreting the SQL, executing the SQL, handling by the disk cache buffers, and fetching the data.

[0016] Even though cache memory may provide improvements in the speed of directories, a cache typically operates

as a buffer and has little, if any, high level intelligence about the nature of directory operations. In addition, a cache may manage low-level data read/write operations without necessarily understanding what the data is.

[0017] Accordingly, it would be beneficial to provide a reliable and effective way to quickly and efficiently process directory operations.

SUMMARY

[0018] A method for handling a directory service operation includes determining whether an alternate evaluator can respond to the operation and evaluating the operation using the alternate evaluator when it is determined that the alternate evaluator can respond to the operation.

[0019] A system for handling a directory service operation includes a switch for determining whether an alternate evaluator can respond to the operation and the alternate evaluator for evaluating the operation when it is determined that the alternate evaluator can respond to the operation.

[0020] A computer system includes a processor and a computer recording medium including computer executable code readable by the computer system. The recording medium embodying a program of instructions executable by the processor for handling a directory service operation. The computer executable code includes determining whether an alternate evaluator can respond to the operation and evaluating the operation using the alternate evaluator when it is determined that the alternate evaluator can respond to the operation.

[0021] A computer recording medium including computer executable code for handling a directory service operation. The computer executable code includes determining whether an alternate evaluator can respond to the operation and evaluating the operation using the alternate evaluator when it is determined that the alternate evaluator can respond to the operation.

[0022] A method of processing a Directory operation in a directory system includes a disk based evaluator and an alternate evaluator. The method includes the step of examining the query to determine if it can be processed by the alternate evaluator.

[0023] A directory system adapted to process directory queries. The system includes a disk based evaluator adapted to process directory queries in association with a data store. An alternate evaluator adapted to process directory queries in association with a local data store. Logic means adapted to examine the query to determine if it can be processed by the alternate evaluator.

[0024] In a directory system adapted to process directory queries, the system includes a disk based evaluator adapted to process directory queries in association with a data store. An alternate evaluator adapted to process directory queries in association with a local data store. The improvement includes switch means adapted to examine the query to determine if it can be processed by the alternate evaluator.

BRIEF DESCRIPTION OF THE DRAWINGS

[0025] A more complete appreciation of the present disclosure and many of the attendant advantages thereof will be readily obtained as the same becomes better understood by

reference to the following detailed description when considered in connection with the accompanying drawings, wherein:

[0026] FIG. 1 is a block diagram showing a system for processing directory operations;

[0027] FIG. 2 is a block diagram showing a system for processing directory operations using an alternate evaluator according to an embodiment of the present disclosure;

[0028] FIG. 3A is a block diagram showing a configuration with multiple switches and multiple alternate evaluators according to an embodiment of the present disclosure;

[0029] FIG. 3B is a flow chart illustrating another embodiment of the present invention;

[0030] FIG. 4 is a flow chart showing a method for handling directory operations according to an embodiment of the present disclosure;

[0031] FIG. 5 is a block diagram showing an apparatus for a directory server;

[0032] FIG. 6 is a block diagram showing an apparatus for optimizing directory performance according to an embodiment of the present disclosure;

[0033] FIG. 7 is a block diagram showing another embodiment of the present disclosure;

[0034] FIG. 8 is a block diagram showing another embodiment of the present disclosure;

[0035] FIG. 9 is a flow chart showing an embodiment of the present disclosure;

[0036] FIG. 10 is a block diagram showing an apparatus for enhancing directory performance according to an embodiment of the present disclosure;

[0037] FIG. 11 is a flow chart showing a method for enhancing directory performance according to an embodiment of the present disclosure;

[0038] FIG. 12 is a flow chart illustrating a method for enhancing directory performance according to an embodiment of the present disclosure;

[0039] FIG. 13 shows an example of a computer system capable of implementing the method and apparatus according to embodiments of the present disclosure;

[0040] FIG. 14 is a flow chart showing the loading of an alternative evaluator according to an embodiment of the present disclosure;

[0041] FIG. 15 is a flow chart detailing the loading of an alternative evaluator according to an embodiment of the present disclosure;

[0042] FIG. 16 is a flow chart showing an example of an update operation according to an embodiment of the present disclosure;

[0043] FIG. 17 is a flow chart showing another example of an update operation according to an embodiment of the present disclosure; and

[0044] FIG. 18 shows an example of a computer system which may implement the method and system of the present disclosure.

DETAILED DESCRIPTION

[0045] The present disclosure provides tools (in the form of methodologies, apparatuses, and systems) for processing directory operations. The tools may be embodied in one or more computer programs stored on a computer readable medium or program storage device and/or transmitted via a computer network or other transmission medium.

[0046] In describing the preferred embodiments of the present disclosure illustrated in the drawings, specific terminology is employed for sake of clarity. However, the present disclosure is not intended to be limited to the specific terminology so selected, and it is to be understood that each specific element includes all technical equivalents which operate in a similar manner.

[0047] FIG. 1 is a block diagram showing a system for processing directory operations. A directory operation, for example a query 101 may be handled by a directory server infrastructure 102, where the directory server 102 may utilize X.500 standards and X.500 protocols, LDAP protocols and/or DSML protocols. The directory server infrastructure 102 may have multiple processing layers 104-108 to execute the query 101. The processing layers 104-108 may be functional components of a server. For example, processing layers 104-108 may be used to decode protocols used by the query, converting the query into standard protocols such as SQL, interpreting the SQL, executing the SQL, handling disk cache buffers, and/or utilizing an operating system to retrieve data from a database 109. A response 110 to the query 101 may be generated by accessing the database 109 that may be stored on one or more physical disks, for example a hard disk or a RAID array. One or more of the processing layers 106-108 may be located within a disk based evaluator 103 that may be responsible for invoking one or more interactions with the disk 109.

[0048] A query 101 may be passed from one processing layer 104-108 to another until a response 110 may be generated by accessing data on the disk 109. The response 110 may then be passed back along the processing layers 104-108 allowing the response 110 to be delivered to the same location and conforming to the same protocols as the query 101.

[0049] Each processing layer 104-108 may provide an important step in the processing of the query 101. For example, processing layers 104 and 105 may be software for decoding protocols, authentication and/or routing. For example, processing layers 106, 107, and 108 may be software for checking schema, controlling access and/or optimizing filters.

[0050] While the plurality of processing layers 104-108 may be useful for processing a query 101 to generate a response 110, having multiple layers may increase the time required to respond to a query 101. It may therefore be possible to lessen the time needed to process queries 101, in some instances, by reducing the number of processing layers 104-108 required to process queries 101.

[0051] According to one embodiment of the present disclosure, an alternate evaluator may be used to lessen the time needed to process a query 101 and generate a response 110. An alternate evaluator may be a directory operation processor that may be capable of processing directory operations. FIG. 2 is a block diagram showing a system for processing

directory operations using an alternate evaluator according to an embodiment of the present disclosure.

[0052] According to this embodiment of the present disclosure, a switch 211 may be provided at a point along the data path that connects the processing layers 104-108. For example, the switch 211 may be placed before the first processing layer 104. The switch may examine an incoming operation, for example a query 101 to determine whether an alternate evaluator 212 may be able to process the query 101 to generate a response 110. In making this determination, the switch 211 may rely on local configurations, for example, programmed logic that resides locally within the switch 211. For example, the switch 211 may be programmed to redirect some or all queries 101 that are search queries to the alternate evaluator 212 for processing.

[0053] The alternate evaluator 212 may then process the query 101 by accessing information stored on a local data store 213 that resides locally within the alternate evaluator 212. A response 110 may then be generated and returned along the same data path that the query 101 was sent over. In this way, one or more processing layers 104-108 may be avoided potentially reducing the time required to process the query 101.

[0054] The local data store 213 may be persistent or non-persistent information store. In a preferred form, the local data store 213 uses local memory.

[0055] According to another embodiment of the present disclosure, the determination as to whether the alternate evaluator 212 may be able to process the query 101 may be made by the switch 211 in conjunction with the alternate evaluator 212.

[0056] In determining whether the alternate evaluator 212 may be able to process the query 101, the switch 211 may be configured to categorize the query 101 according to one or more known query types and/or recognize the content of the query 101. If it is determined that the alternate evaluator 212 may not be able to process the query 101, the switch 211 may allow the query 101 to proceed through the processing layers 104-108 according to the data path rather than being diverted to the alternate evaluator 212.

[0057] According to another embodiment of the present disclosure, the switch 211 may allow a query 101 to proceed to the alternate evaluator 212 and to proceed through the processing layers 104-108 potentially allowing the query to be evaluated in both ways. The processing of the query 101 through the processing layers 104-108 may then later be discontinued if the alternate evaluator 212 is able to successfully process the query 101.

[0058] According to one embodiment of the present disclosure, the alternative evaluator 212 may reside within the directory server infrastructure 102. According to another embodiment of the present disclosure, the alternate evaluator 212 may reside outside of the directory server infrastructure 102.

[0059] According to other embodiments of the present disclosure, there may be any number of switches 211 and/or alternate evaluators 212. FIG. 3A is a block diagram showing a configuration with multiple switches 301-303 and multiple alternate evaluators 304-306 according to an embodiment of the present disclosure.

[0060] One or more switches 301-303 may be provided at multiple points along the data path that connects the processing layers 104-106. According to some embodiments of the present disclosure, one alternate evaluator 304-306 may be provided for each switch 301-303. Alternatively, multiple switches, for example switches 301 and 302, may each be capable of redirecting a directory operation, for example a query 101 to the same alternate evaluator, for example 304. This may be beneficial in situations where a query 101 may be able to be handled by an alternate evaluator 304 either before being processed by one or more processing layers, for example processing layer 104, or after being processed by the one or more processing layers. Additionally, a single switch, for example switch 303, may be able to redirect a query 101 to one of one or more alternate evaluators, for example alternate evaluators 305 and 306. This may be useful when a switch determines that a query 101 may be handled by one alternate evaluator but not another. Each alternate evaluator 304-306 may have its own respective local data store 307-309 or alternatively, one or more alternate evaluators 304-306 may share one or more local data stores 307-309.

[0061] As a query 101 progresses down the data path, it may be progressively evaluated and/or transformed into a number of sub-queries. The alternate evaluators 304-306 may be able to process one or more queries 101, one or more parts of a query 101, partly evaluated queries, and/or sub-queries.

[0062] FIG. 3B is a flow chart illustrating another embodiment of the present invention. A determination may be made as to whether to direct (forward) an operation, for example a query, to an alternate evaluator. A first test may be made to determine whether the alternate evaluator is available (Step S30). For example, if the alternate evaluator is enabled, busy or loading, the alternate evaluator would not be considered "available" (No, Step S30) and the query would not be forwarded to the alternate evaluator (Step S34).

[0063] If the alternate evaluator is available (Yes, Step S30), the first parameter(s) are determined (Step S31) and the second parameter(s) are determined (Step S32). Examples of first parameters of the query which are examined include its type, complexity, operands and properties, and the attribute types including those required to process a query and those needed to be returned. Examples of the second parameters include type, complexity, operands and properties, and the attributes available including those indexed and those carried (not indexed).

[0064] A test may be performed to determine if the first parameters are supported by the second parameters (Step S33). If they are not supported (No, Step S33) then the query is not forwarded to the alternate evaluator (Step S34). If the first parameters are supported by the second parameters (Yes, Step S33), then a further test may be performed to determine if there are third (optional) parameters (Step S35).

[0065] Examples of third parameters include pre-determined criteria provided by the user or system, arbitrary data provided by user, system policies, origin of query, user or user's role, operating conditions such as load, size limits and time limits. If there are no third parameters (No, Step S35), then the query may be forwarded to an alternate evaluator (Step S36). If there are third parameters (Yes, Step S35), then third parameters may be determined (Step S37).

[0066] A further test may be performed to determine if the third parameters require the query to be redirected in view of the third parameters (Step S38). If the query is not to be redirected (No, Step S38), then the query is not forwarded to the alternate evaluator (Step S34).

[0067] If the third parameters allow redirection of the query, (Yes, Step S38) then a further test may be made to determine if the first parameters are supported by the third parameters (Step S39). If they are not supported (No, Step S39), then the query is not directed to the alternate evaluator (Step S34). However, if the first parameters are supported by the third parameters (Yes, Step S39), then the query may be forwarded to the alternate evaluator (Step S36).

[0068] Example of the first parameters include: Operation type—For example operation types are Read, Search, Compare, List, Add, Modify, etc. Operation complexity—For example of complexity would be a multi-term filter. Operation operands—For example substring match, approximate match, greater than or equal to match etc. Operation attribute(s) required to process a query—For example the attributes in a Search filter or the attribute in a Compare. Operation attribute(s) that needs to be returned—For example the attributes to return on a Search or Read. Service controls—For example size limit, time limit, prefer chaining, originator, attribute size limit, priority, other X.500 service control options. LDAP controls—for example proxied user, sorting, paging, event management. Network information—for example source address.

[0069] Examples of the second parameters include: Supported operation types—For example operation types are Read, Search, Compare, List, Add, Modify, etc. Supported operation complexity—For example of complexity would be a multi-term filter. Supported operation operands—For example substring match, approximate match, greater than or equal to match etc. Attribute(s) that are indexed. Attribute(s) that are carried (not indexed).

[0070] Examples of third parameters include: Arbitrary data provided by user. System policies. Origin of query. User or user's role. Operating conditions such as load. Administrative thresholds—for example size limit, time limit, prefer chaining, security level, attribute size limit, priority or other physical limitations. Service options—for example sorting, paging, event management, or other functional abilities.

[0071] FIG. 4 is a flow chart showing a method for handling directory operations according to an embodiment of the present disclosure. An operation, for example a query may be received (Step S41). Next, it may be determined whether the query may be processed using an alternate evaluator (Step S42). If the query may be so processed (Yes, Step S42), then the query may be redirected to the alternate evaluator (Step S43). The alternate evaluator may then evaluate the query using a local data store (Step S44). If it is determined that the query cannot be processed by the alternate evaluator (No, Step S42), then the query may be evaluated using a disk-based database.

[0072] The combination of switches 211 and alternate evaluators 212 or a directory service incorporating one or more switches and/or one or more alternate evaluators may be thought of more generally as an apparatus for optimizing directory performance.

[0073] FIG. 5 is a block diagram showing an apparatus for optimizing directory performance. Directory operations, for

example queries may originate from a location, for example from one or more query clients **52**. Responses returned in response to the queries may be directed to the same query client **52**. Queries may be sent from the query client **52** to a server **50**. The server **50** may contain a first directory alternate evaluator, for example a front-end processor **53** that may manage communications, queries and/or responses. The server **50** may also contain a second directory alternate evaluator, for example a back-end processor **54** that may manage the evaluation of operations, for example queries, using information stored on a disk **51**.

[0074] The server **50** may provide a directory service. For example, the server **50** may be a directory server conforming to implement directory X.500 standards and one or more communications protocols, for example X.500 protocols, LDAP protocols and DSML protocols.

[0075] **FIG. 6** is a block diagram showing an apparatus for optimizing directory performance according to an embodiment of the present disclosure. In this embodiment, the server **50** may additionally contain an alternate evaluator **60**. The alternate evaluator **60**, may be provided between the front-end processor **53** and the back-end processor **54** as shown in **FIG. 6**.

[0076] The alternate evaluator **60** may act as an alternate evaluator as described above and may be able to generate a response for a received query without having to send the query to the back-end processor **54**. Queries that may be responded to by the alternate evaluator **60** may be responded to more quickly than queries that may be responded to by the back-end processor **54** that uses the disk **51**.

[0077] The alternate evaluator **60** may contain logic **61** and data **62**. The logic **61** may manage the evaluation of the queries. The logic **61** may use the data **62** to evaluate the queries. The data **62** may be a local data store as described above. The data **62** may be stored using a cache or a similar memory store that may be faster than the disk **51**.

[0078] The alternate evaluator **60** need not be located between the front-end processor **53** and the back-end processor **54**. **FIG. 7** is a block diagram showing another embodiment of the present disclosure. Here the alternate evaluator **60** may be connected to the front-end processor **53** alone. According to this embodiment, the alternate evaluator **60** may be able to receive and handle a query sent by the front-end processor **53**.

[0079] **FIG. 8** is a block diagram showing another embodiment of the present disclosure. Here the alternate evaluator **60** may be connected to the back-end processor **54**. According to this embodiment, the alternate evaluator **60** may be able to receive and handle a query sent by the back-end processor.

[0080] The front-end processor **53**, the alternate evaluator **60**, and the back-end processor **54** need not be located on the same server **50**. According to another embodiment of the present disclosure, each of the front-end processor **53**, the alternate evaluator **60**, and the back-end processor **54** may be provided as a separate server, wherein each of the servers may be interconnected. According to another embodiment of the present disclosure, any two of the front-end processor **53**, the alternate evaluator **60**, and the back-end processor **54** may be provided on a first server while the other may be provided on a second server, wherein each of the servers may be interconnected.

[0081] **FIG. 9** is a flow chart showing an embodiment of the present disclosure. A directory operation, for example a query may be received, for example, from a client, by a front-end processor (Step S91). An alternate evaluator may attempt to evaluate the query using data in the alternate evaluator, for example, data stored in a cache (Step S92). If the evaluation is successful (Yes, Step S93) then a response to the query may be returned to the query client. If the evaluation is not successful (No, Step S93) then the query may be passed to a back-end processor (Step S94). The back-end processor may evaluate the query using a disk (Step S95). A response to the query may thereby be returned to the query client.

[0082] The alternate evaluator's data (local data store) may be a cache of random access memory (RAM). According to embodiments of the present disclosure, the alternate evaluator may be preloaded with selected information from the disk or disk's conventional cache (a cache associated with the disk as opposed to the local data store that is associated with the alternate evaluator) prior to the use of the alternate evaluator. The preloading of the alternate evaluator may allow the alternate evaluator to evaluate the query without having to access the disk.

[0083] An apparatus for enhancing directory server performance, according to one embodiment of the present disclosure, will be discussed with reference to **FIG. 10**. The apparatus **120** includes a determining means **121**, retrieving means **122**, normalizing means **123** and indexing means **124**.

[0084] A method for enhancing directory server performance will be explained below with reference to **FIGS. 10 and 11**. The determining means **121** may determine one or more data types from the disk or disk cache (storage) to be loaded into the alternate evaluator (Step S131). The retrieving means **122** may retrieve data from the storage based on the determined data types (Step S132). The normalizing means **123** may determine and normalize one or more index values for the data retrieved by the retrieving means **122** (Step S133). The indexing means **124** may load an alternate evaluator with the retrieved data and the normalized index values (Step S134). In one embodiment, any of the determining means **121**, retrieving means **122**, normalizing means **123** or indexing means **124** may be based on control variables **150** (**FIG. 13**).

[0085] By preloading the alternate evaluator, the alternate evaluator or other logic may be able to determine if the alternate evaluator can process a given query without having to access the disk storage. Additionally, by preloading the alternate evaluator, the alternate evaluator may be able to achieve maximum performance on startup, unlike conventional caches that may require time to warm up.

[0086] According to an embodiment of the present disclosure, selected information may be preloaded into the alternate evaluator from storage before a query is handled by the system. Storage may include a physical disk, disk cache, memory, or any other means of storing data. The process of pre-loading may include, for example, extracting from storage, such as a disk, selected information which can be indexed by the alternate evaluator. The selected information may be determined by a configuration as defined in the alternate evaluator. For example, index "CommonName" and carry "Surname" and "TelephoneNumber" may be pro-

vided in the configuration. It should be noted that “carry” means that certain information may be stored in an alternate evaluator, but may not be indexed. The index values may be normalized, or transformed into a standard form according to matching rules for the data type. For example, a “case-ignore” matching rule would convert the data types into one case. The selected information may be in a raw format, or stored in a binary encoded form, such as Abstract Syntax Notation 1 (ASN.1).

[0087] According to an embodiment of the present disclosure, if the selected information is already loaded in the alternate evaluator, a pointer to the existing value may be added to the alternate evaluator rather than adding the selected information to the alternate evaluator multiple times.

[0088] FIG. 12 is a flow chart illustrating a method for enhancing directory performance according to an embodiment of the present disclosure. After a directory server is started up or re-initialized (Step S141), the configuration of the alternate evaluator may be read (Step S142). The configuration may specify the attribute types to be loaded, or may specify that all attribute types are to be loaded. The configuration may also specify whether each attribute type is indexed. Based on the configuration, the appropriate data may be retrieved from storage (Step S143). The retrieved data may include at least one directory object, where each object may include one or more attributes, and where each attribute may include one or more values. For each data value retrieved, it may be determined whether the data is already loaded in the alternate evaluator (Step S144). If the data is already loaded in the alternate evaluator (Yes, Step S144), then the duplicate data may be determined and a pointer may be provided to the existing value in the alternate evaluator (Step S145). If the data is not already loaded in the alternate evaluator (No, Step S144), then the index values for the data are determined (Step S146) provided that the data values are indexed (based on the configuration), and the index values are normalized (Step S147). For example, the normalized index values may be stored as a structured index, such as, a BTREE, BTREE+, and/or any other indexing structures. The alternate evaluator may then be loaded with the data values and the normalized index values (Step S148). In another embodiment, the indexes may be determined dynamically upon loading of the alternate evaluator. In a further embodiment, the indexes may be determined in accordance with control variables.

[0089] A BTREE is a tree data structure that may be used by databases and indexes to facilitate searching. BTREES allow for amortized logarithmic time insertions and deletions of tree elements.

[0090] According to an embodiment of the present disclosure, the alternate evaluator may be loaded with all the data values and the index values in a single pass. According to another embodiment of the present disclosure, the alternate evaluator may be loaded with the data values in one step and loaded with the index values in another step.

[0091] As described above, the alternate evaluator, for example a local data store within an alternate evaluator, may comprise a memory store. According to some embodiments of the present disclosure, the alternate evaluator may be pre-loaded and progressively updated to enhance the alternate evaluator's ability to quickly and accurately handle directory operations, for example queries.

[0092] After the alternate evaluator is preloaded, the alternate evaluator may receive progressive updates to maintain accuracy of stored data, efficiency of memory storage and speed of evaluation. For example, data structures may be retrieved from storage and used to progressively update data stored in the alternate evaluator. In so doing, embodiments of the present disclosure may be able to facilitate the storage of a high occupancy of indexes, minimize the number of comparisons required to execute a query and to provide multiple index types, such as, for example, reverse indexes.

[0093] In determining what data to pre-load and/or progressively update, control variables may be considered. Additionally, the type of attributes that are indexed may be considered. Control variables may describe, for example, whether an alternate evaluator is available, what indexes are in effect, what attributes are stored, and/or the maximum amount of alternate evaluator memory available. For example, entries and/or attributes relating to one or more attribute indexes may be loaded and/or progressively updated to the alternate evaluator. It may also be possible to load and/or progressively update all entries and/or attributes, for example, where a configuration of indexed attributes is set to “all attributes.”

[0094] In loading and/or progressively updating the alternate evaluator, the effectiveness of the alternate evaluator may be maximized by using the available alternate evaluator memory as efficiently as possible. Embodiments of the present disclosure may utilize one or more methods for efficiently utilizing alternate evaluator memory. For example, where data is repeated in multiple locations of the alternate evaluator memory, the instances of that data may be replaced with a pointer. For example, attribute type data may be replaced with a pointer that points to an attribute type store where attribute type data is stored. For example, where data utilizes names that are comprised of a context prefix (the distinguished name of a root entry) and a relative name, the context prefix may be removed from the name to reduce the amount of memory required to store the names. For example, an available memory list may be maintained to keep track of available memory so that memory that was previously used may be quickly identifiable as free memory. For example, a list of normalized values may be maintained for attribute values that are indexed so that multiple units of data do not need to individually store normalized values for the same sets of values. Normalized values may be created from attribute values that are configured as indexed attributes.

[0095] According to embodiments of the present disclosure, data stored in the alternate evaluator need not be updated every time the data in storage (external data source), upon which the data stored in the alternate evaluator was loaded from, is updated. For example, where stored and/or indexed attributes from a list of entries comprising attributes and attribute values is stored in the alternate evaluator, the stored and/or indexed attributes may only be updated when those stored and/or indexed attributes are modified on the list of entries and may not be updated when the list is modified but the stored and/or indexed attributes remain unchanged.

[0096] FIG. 13 is a block diagram showing an alternate evaluator according to an embodiment of the present disclosure. Directory operations, for example queries may be

directed towards the alternate evaluator **151** over a data path for receiving operations **153**. Each query directed towards the alternate evaluator **151** may result in a response. The response may be sent along a data path for providing results **154**. Data loaded into the alternate evaluator **151**, may be organized into one or more lists **155-159** within an arrangement of memory included within the alternate evaluator. The lists **155-159** may represent data structures for the storing of data entries, attributes, attribute values, normalized values, indexes, available memory, etc. Control variables may be entered into the alternate evaluator **151** and stored within a portion of memory provided to store control variables **150**. Control variables **150** may be used to define a set of controls, for example, to define what data is to be stored from an external data store **152** to the one or more lists **155-159**. For example, to determine what indexes are in effect and/or to determine the maximum amount of available memory

[0097] The control variables **150** may be used, for example, to define attributes to be indexed. For example, the control variables **150** may define that all attributes be indexed by setting an indexed attribute control variable to "all attributes." Similarly, the control variables **150** may define attributes to be stored. For example, the control variables **150** may define that all attributes be stored by setting a stored attribute control variable to "all attributes."

[0098] The control variables **150** may be used, for example, to define entries to be loaded into the alternate evaluator **151** from the external data store **152**. For example, the control variables **150** may define that all entries should be loaded by setting the indexed attribute control variable to "all entries." Alternatively, the indexed attribute control variable may be set to a list of attributes. In such a case only those attributes present in the list may be loaded.

[0099] The control variables **150** may be used, for example, to define attributes to be loaded with each entry. For example, where either the indexed attribute control variable or the stored attribute control variable is set to "all attributes" then all attributes within each entry may be loaded. For example, where both the indexed attribute control variable and the stored attribute control variable are set to a list of attributes, then the attributes loaded within each entry may be a combination of both lists.

[0100] A configuration function (not shown) may be provided to set the control variables.

[0101] An alternate evaluator **151** according to an embodiment of the present disclosure may have a first list **155**. Data may be first loaded into the first list **155** via a data path **160** from a disk **152** and/or directory **161**. The first list **155** may comprise one or more entries. Each entry may have a name and/or one or more attributes. Each attribute may have a type and/or one or more values. For example, the entries may be directory entries. The name may be a distinguished name for the directory entry. The attributes may be directory attributes. The types may be object identifiers or strings, either of which may represent the types of the attributes. The values may be the values of the attributes and may be stored in binary form.

[0102] To reduce memory requirements, entries that are loaded into the alternate evaluator **151** may have the context prefix removed from their distinguished name.

[0103] The alternate evaluator **151** may have a second list **156**. The second list **156** may be loaded with normalized

values of data that has been loaded into the first list **155**. The normalized values may be representations of the values in a predetermined format, for example in the internationally defined X.500 matching format. Normalizing values may allow for comparison regardless of the format of the values. For example, where a CommonName attribute for an entry is "Rick Harvey" a CaseIgnoreString matching rule may be used to create the normalized value "RICK HARVEY" where the CaseIgnoreString matching rule is defined as removing leading and trailing spaces, replacing multiple consecutive spaces with a single space and converting the string into upper case. The second list **156** may additionally be loaded with pointers that point to the corresponding data that has been loaded into the first list **155**. The indexed attribute control variable may determine what normalized data and pointers may be loaded into the second list **156**.

[0104] The alternate evaluator **151** may have a third list **157**. The third list **157** may be loaded with indexes and pointers to the normalized values that have been loaded into the second list **156**. For example, the third list **157** may have at least one index that is organized as a BTREE.

[0105] The alternate evaluator **151** may have a fourth list **158**. The fourth list **158** may be a list of available memory locations. For example, the fourth list may list one or more memory fragments not currently used by the alternate evaluator. The fourth list **158** may be initially empty. The fourth list may be updated during the operation of the alternate evaluator as memory fragments are used and become available (returned to an unused state).

[0106] The alternate evaluator **151** may have a context prefix store (not shown). The context prefix store may be used to store the context prefix of data names, for example, entry names. The context prefix store may store a normalized value of the content prefix. The context prefix store may be associated with the data path for receiving operations **153** and/or the data path for providing results **154**.

[0107] The alternate evaluator **151** may have a fifth list **159**. The fifth list **159** may be a list of types, for example, attribute types, that occur one or more times in the first list **155**. Types may take the form of an object identifier or a string. The first list **155** may replace instances of attribute types stored in the fifth list **159** with pointers that point to the same attribute type in the fifth list **159**. By utilizing these pointers to the fifth list **159**, available memory may be saved by reducing the number of times the same attribute type is stored.

[0108] FIG. 14 is a flow chart showing the loading of a list according to an embodiment of the present disclosure. Control variables may be retrieved (Step S162). The retrieved control variables may then be used to determine data to be loaded from the external data store (Step S163). The context prefixes from distinguished names may be removed from the determined data (Step S164). The determined data may then be loaded into the alternate evaluator and/or a list, for example a list within an alternate evaluator (Step S165).

[0109] FIG. 15 is a flow chart detailing the loading of lists, such as any of the lists in an alternate evaluator according to an embodiment of the present disclosure. First the first list may be loaded, for example, the first list may be loaded based on inputted control variables (Step S171). Next the second list

may be loaded, for example, based on the first list (Step S172). Next the third list may be loaded, for example, based on the second list (Step S173). The fourth list may be used to store items removed from any of the other lists during operation. Next the fifth list may be loaded, for example, based on the first list (Step S174). Loading of the lists may be executed, for example, in the fashion described above.

[0110] As described above, as data within the external data store is changed, it may be beneficial to perform an update operation to update the information within the alternate evaluator and/or lists. Because the lists may be interdependent, updating the first list may often be followed by updating subsequent lists as necessary to maintain the data structure of the alternate evaluator. The update operation may include, for example, one or more of the following: Adding an entry with a name and/or a set of attributes, each attribute containing one or more attribute values. Removing an entry including any of its associated names, attributes and/or attribute values. Modifying an entry to add and/or remove an attribute and/or attribute values. Rename an entry to change an entry's name by adding, removing or selecting existing attribute values.

[0111] FIG. 16 is a flow chart showing an example of an update operation according to an embodiment of the present disclosure. For example, the update operation may involve adding data, for example, entries, attributes and/or attribute values, to the first list. For example, added data may be indexed attributes or stored attributes. Prior to adding the data, context prefixes may be removed from names within the data (Step S181). Prior to adding the data, attribute types within the data may be replaced with pointers that point to the same attribute type in the fifth list (Step S182). Where the attribute type is not already present in the fifth list (No, Step S183), it may be added to the fifth list (Step S184). Next, the data may be added to the first list within available memory that may be determined using the fourth list (Step S185). Available memory may then be updated within the fourth list based on the memory used in storing the data (Step S186).

[0112] It may be advantageous to have normalized values for the data present within the second list. This may be especially true where the data is an indexed attribute. Where it may be so advantageous, the second list may be checked to see if it contains normalized values of the data (Step S187). Where it does not (No, Step S187), normalized values may be added to the second list and a pointer to the data is added to the normalized values in the second list (Step S188). In adding to the second list, available memory may be ascertained from the fourth list and memory used may be reflected in the fourth list, for example, by removing the used memory from the fourth list. Additions made to the second list may be reflected within the index and pointers of the third list (Step S189). For example, a pointer may be added to the index.

[0113] FIG. 17 is a flow chart showing another example of an update operation according to an embodiment of the present disclosure. For example, the update operation may involve removing data, for example, entries, attributes and/or attribute values, to the first list. First the data may be removed from the first list (Step S191). If the removed data is pointed to by a pointer in the second list corresponding to normalized values for the data, the corresponding pointers

may be removed from the second list (Step S192). When the last pointer of a normalized value in the second list is removed (Yes, Step S193), the normalized value may be removed and the corresponding pointer may be removed from the index on the third list (Step S194). The fourth list may then be updated to reflect that the memory used by deleted elements is available memory (Step S195).

[0114] According to embodiments of the present disclosure, the lists may be of any length. According to embodiments of the present disclosure, the lists may be dynamic in size and content, for example, during the operation of the alternate evaluator. According to embodiments of the present disclosure, the lists may be linked lists and/or arrays.

[0115] Numerous additional modifications and variations of the present disclosure are possible in view of the above teachings. It may therefore be understood that within the scope of the appended claims, the present disclosure may be practiced other than as specifically described herein.

[0116] FIG. 18 shows an example of a computer system which may implement the method and system of the present disclosure. The system and method of the present disclosure may be implemented in the form of a software application running on a computer system, for example, a mainframe, personal computer (PC), handheld computer, server, etc. The software application may be stored on a recording media locally accessible by the computer system and accessible via a hard wired or wireless connection to a network, for example, a local area network, or the Internet.

[0117] The computer system referred to generally as system 1000 may include, for example, a central processing unit (CPU) 1001, random access memory (RAM) 1004, a printer interface 1010, a display unit 1011, a local area network (LAN) data transmission controller 1005, a LAN interface 1006, a network controller 1003, an internal bus 1002, and one or more input devices 1009, for example, a keyboard, mouse etc. As shown, the system 1000 may be connected to a data storage device, for example, a hard disk, 1008 via a link 1007.

[0118] The above specific embodiments are illustrative, and many variations can be introduced on these embodiments without departing from the spirit of the disclosure or from the scope of the appended claims. For example, elements and/or features of different illustrative embodiments may be combined with each other and/or substituted for each other within the scope of this disclosure and appended claims.

1. A method for handling a directory service operation, comprising:

determining whether an alternate evaluator can respond to the operation; and

evaluating the operation using the alternate evaluator when it is determined that the alternate evaluator can respond to the operation.

2. The method of claim 1, wherein the directory service operation is handled by a directory service conforming to X.500 standards.

3. The method of claim 2, wherein the directory service utilizes DSML communications protocols.

4. The method of claim 2, wherein the directory service utilizes LDAP communications protocols.

5. The method of claim 1, wherein the directory service operation is a query.

6. The method of claim 5, wherein the query is any directory query.

7. The method of claim 5, wherein the query is any directory sub-query such as Navigation.

8. The method of claim 1, wherein determining whether an alternate evaluator can respond to the operation is based on content of the operation.

9. The method of claim 1, wherein evaluating the operation using the alternate evaluator comprises generating a response to the operation.

10. The method of claim 1, further comprising evaluating the operation using a primary evaluator when it is determined that the alternate evaluator cannot respond to the operation.

11. The method of claim 10, wherein evaluating the operation using the primary evaluator comprises generating a response to the operation.

12. The method of claim 10, wherein the primary evaluator accesses information stored on a disk in evaluating the operation.

13. The method of claim 1, wherein a switch is used in determining whether the alternate evaluator can respond to the operation.

14. The method of claim 13, wherein the switch uses local configurations to determine whether the alternate evaluator can respond to the operation.

15. The method of claim 13, wherein the switch uses the alternate evaluator to determine whether the alternate evaluator can respond to the operation.

16. The method of claim 1, wherein logic is used in determining whether the alternate evaluator can respond to the operation.

17. The method of claim 1, wherein evaluating the operation using the alternate evaluator is faster than evaluating the operation using a primary evaluator.

18. The method of claim 1, wherein evaluating the operation using the alternate evaluator involves fewer processing layers than evaluating the operation using a primary evaluator.

19. The method of claim 1, wherein the alternate evaluator evaluates the operation using data that is stored locally to the alternate evaluator.

20. The method of claim 19, wherein the data that is stored locally comprises a local data store.

21. The method of claim 1, wherein determining whether the alternate evaluator can respond to the operation comprises categorizing the operation according to one or more known operation types.

22. The method of claim 1, wherein determining whether the alternate evaluator can respond to the operation comprises recognizing content of the operation.

23. The method of claim 1, wherein two or more alternate evaluators are used to evaluate one or more operations at various stages of a data path of processing layers.

24. The method of claim 23, wherein one or more switches are used in determining whether the two or more alternate evaluator can respond to the operations.

25. A system for handling a directory service operation, comprising:

a switch for determining whether an alternate evaluator can respond to the operation; and

the alternate evaluator for evaluating the operation when it is determined that the alternate evaluator can respond to the operation.

26. The system of claim 25, wherein the directory service operation is handled by a directory service conforming to X.500 standards.

27. The system of claim 26, wherein the directory service utilizes DSML communications protocols.

28. The system of claim 26, wherein the directory service utilizes LDAP communications protocols.

29. The system of claim 25, wherein the directory service operation is a query.

30. The system of claim 29, wherein the query is any directory query.

31. The system of claim 29, wherein the query is any directory sub-query such as Navigation.

32. The system of claim 25, wherein determining whether an alternate evaluator can respond to the operation is based on content of the operation.

33. The system of claim 25, wherein evaluating the operation using the alternate evaluator comprises generating a response to the operation.

34. The system of claim 25, further comprising a primary evaluator for evaluating the operation when it is determined that the alternate evaluator cannot respond to the operation.

35. The system of claim 34, wherein the primary evaluator generates a response to the operation when it is determined that the alternate evaluator cannot respond to the operation.

36. The system of claim 34, wherein the primary evaluator accesses information stored on a disk in evaluating the operation.

37. The system of claim 25, wherein the switch uses local configurations to determine whether the alternate evaluator can respond to the operation.

38. The system of claim 25, wherein the switch uses the alternate evaluator to determine whether the alternate evaluator can respond to the operation.

39. The system of claim 25, wherein logic is used in determining whether the alternate evaluator can respond to the operation.

40. The system of claim 25, wherein evaluating the operation using the alternate evaluator is faster than evaluating the operation using a primary evaluator.

41. The system of claim 25, wherein evaluating the operation using the alternate evaluator involves fewer processing layers than evaluating the operation using a primary evaluator.

42. The system of claim 25, wherein the alternate evaluator evaluates the operation using data that is stored locally to the alternate evaluator.

43. The system of claim 42, wherein the data that is stored locally comprises a local data store.

44. The system of claim 25, wherein determining whether the alternate evaluator can respond to the operation comprises categorizing the operation according to one or more known operation types.

45. The system of claim 25, wherein determining whether the alternate evaluator can respond to the operation comprises recognizing content of the operation.

46. The system of claim 25, wherein two or more alternate evaluators are used to evaluate one or more operations at various stages of a data path of processing layers.

47. The system of claim 46, wherein one or more switches are used in determining whether the two or more alternate evaluator can respond to the operations.

48. The system of claim 25, wherein the switch is comprised of hardware.

49. The system of claim 25, wherein the switch is comprised of software.

50. A computer system comprising:

a processor; and

a computer recording medium including computer executable code readable by the computer system, embodying a program of instructions executable by the processor for handling a directory service operation, the computer executable code comprising:

determining whether an alternate evaluator can respond to the operation; and

evaluating the operation using the alternate evaluator when it is determined that the alternate evaluator can respond to the operation.

51. (canceled)

52. (canceled)

53. (canceled)

54. (canceled)

55. (canceled)

56. (canceled)

57. (canceled)

58. (canceled)

59. (canceled)

60. (canceled)

61. (canceled)

62. (canceled)

63. (canceled)

64. (canceled)

65. (canceled)

66. (canceled)

67. (canceled)

68. (canceled)

69. (canceled)

70. (canceled)

71. (canceled)

72. (canceled)

73. (canceled)

74. A computer recording medium including computer executable code for handling a directory service operation, the computer executable code comprising:

determining whether an alternate evaluator can respond to the operation; and

evaluating the operation using the alternate evaluator when it is determined that the alternate evaluator can respond to the operation.

75. (canceled)

76. (canceled)

77. (canceled)

78. (canceled)

79. (canceled)

80. (canceled)

81. (canceled)

82. (canceled)

83. (canceled)

84. (canceled)

85. (canceled)

86. (canceled)

87. (canceled)

88. (canceled)

89. (canceled)

90. (canceled)

91. (canceled)

92. (canceled)

93. (canceled)

94. (canceled)

95. (canceled)

96. (canceled)

97. (canceled)

98. A method of processing a Directory operation in a directory system comprising a disk based evaluator and an alternate evaluator, the method comprising the step of:

examining the query to determine if it can be processed by the alternate evaluator.

99. A method as claimed in claim 98, further comprising the step of:

redirecting the query to the alternate evaluator.

100. A method as claimed in claim 98, further comprising the step of:

repeatedly examining the query at a number of points along a mainline path prior to evaluation by the disk based evaluator.

101. A method as claimed in claim 98, wherein the query is any LDAP or DSML directory query.

102. A method as claimed in claim 98, wherein the query is any X.500 directory query.

103. A method as claimed in claim 98, wherein the query is any directory sub-query such as Navigation.

104. A method as claimed in claim 98, wherein the step of examining the query is external and based on the content of the query.

105. A method as claimed in claim 98, wherein the criterion for examination is configurable.

106. (canceled)

107. (canceled)

108. (canceled)

109. (canceled)

110. (canceled)

111. (canceled)

112. (canceled)

113. (canceled)

114. (canceled)

115. (canceled)

116. (canceled)

117. (canceled)

118. (canceled)

119. (canceled)

120. (canceled)

121. (canceled)

122. (canceled)

* * * * *