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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
4 August 2016 (04.08.2016)

WIPO | PCT

(10) International Publication Number
WO 2016/122603 A1

(51) International Patent Classification:
H04L 12/26 (2006.01) *H04L 12/24* (2006.01)

(21) International Application Number:
PCT/US2015/013797

(22) International Filing Date:
30 January 2015 (30.01.2015)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant: **HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP** [US/US]; 11445 Compaq Center Drive West, Houston, TX 77070 (US).

(72) Inventors: **ZUCKERMAN, Boris**; 165 Dascomb Road, Andover, Massachusetts 01810-5897 (US). **BATUNER, Oskar Y.**; 165 Dascomb Road, Andover, Massachusetts 01810-5897 (US). **YE, Manny**; 165 Dascomb Road, Andover, Massachusetts 01810-5897 (US).

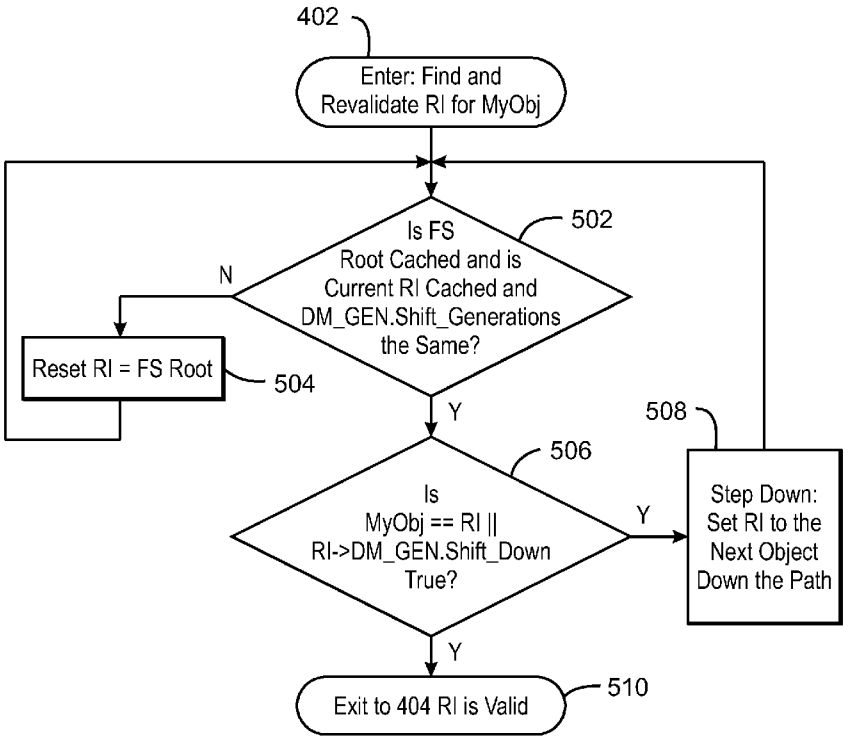
(74) Agents: **ORTEGA, Arthur S.** et al.; Hewlett Packard Enterprise, 3404 E. Harmony Road, Mail Stop 79, Fort Collins, CO 80528 (US).

(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,

[Continued on next page]

(54) Title: DYNAMICALLY INHERITABLE ATTRIBUTE



500

FIG. 5

(57) Abstract: In one example, disclosed is a method for controlling contention in hierarchical storage The method includes monitoring loading on a destination server (DS) controlling a directory that is a root of inheritance (RI) for a dynamically inheritable attribute. If the loading on the DS is higher than a predetermined level, move the RI for the dynamically inheritable attribute to a lower level in a name space tree.

WO 2016/122603 A1

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).

Declarations under Rule 4.17:

— *as to the identity of the inventor (Rule 4.17(i))*

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

Published:

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

DYNAMICALLY INHERITABLE ATTRIBUTE

BACKGROUND

[0001] Data stored in a storage system can be organized into files and directories of a file system. A large storage system typically has a large number of computer nodes. As a result, information associated with the file system can be a distributed across the computer nodes. Performing certain operations in a distributed file system can be complex and can result in inefficiency if not performed properly.

DESCRIPTION OF THE DRAWINGS

[0002] Certain exemplary embodiments are described in the following detailed description and in reference to the drawings, in which:

[0003] Fig. 1 is a schematic of an example of a highly distributed segmented parallel file system;

[0004] Fig. 2 is a schematic of an example of propagating an inheritable snapshot time mark (STM);

[0005] Fig. 3 is an example of a method for handling dynamically inheritable attributes;

[0006] Fig. 4 is an example of a method for propagating dynamically inheritable attributes;

[0007] Fig. 5 is an example of a method for finding and validating a root of inheritance for dynamically inheritable attributes in a name space tree;

[0008] Fig. 6 is a schematic of an example of propagation of roots of inheritance to different levels of a name space tree; and

[0009] Fig. 7 is an example of server that may allow roots of inheritance to be moved up or down a hierarchical name space tree.

DETAILED DESCRIPTION

[0010] Dynamic inheritance allows for efficient setting of various properties in the environment of a highly scalable distributed name space. Dynamic inheritance is widely used for variety of reasons ranging from propagating snapshot marks,

snapshot restore information, anti-virus checking policies, placement and tiering rules, among others. However, the mechanism generally assumes a low rate of such modifications, and often uses a single defined root of inheritance, such as the root of a file system, as a coordinating node. As the number of changes in properties, file location, and the like, increases, the number of updates to the root of inheritance also increase. This can create undesirable contention, e.g., loading on the server controlling the root of inheritance, and reduce productivity due to the number of systems trying to access the root directory substantially simultaneously. The overlapping access attempts may be especially problematic for very large multi-tenant name spaces with a large number of independent modifications of the dynamically inheritable attributes.

[0011] In examples described herein, techniques are employed to measure contention and control it by selecting multiple inheritance roots dynamically. The use of different roots of inheritance (RI) may help lower the number of access attempts on the main RI, which may increase the efficiency of the file system.

[0012] Fig. 1 is a schematic 100 of an example of a highly distributed segmented parallel file system. The core component of the distributed segmented parallel file system (FS) 102 is a storage segment 104. The FS 102 may include a plurality, such as thousands, of storage segments 104, many of which may be in the terabyte or petabyte size.

[0013] The individual storage segments 104 are controlled by corresponding storage servers 106. However, for load balancing purposes or due to component failures or maintenance reasons, the control over storage segments 104 can migrate from one server 106 to another. Storage servers 106 can be connected to the storage segments 104 directly, for example, through a direct attached storage (DAS) model or through various interconnect technologies 108 such as Fibre Channel (FC), serial attached SCSI (SAS), internet protocol SCSI (iSCSI), and the like. The FS 102 may also include client servers 110 that may not control storage segments 104. The client servers 110 can be used to run applications or provide access to the FS 102 through other protocols 112 such as network file system (NFS), server message block (SMB), HTTP, FTP, and the like.

[0014] To provide file system services, participating nodes, such as storage servers 106 and client servers 110, exchange messages over Ethernet or other networks. In comparison to keeping all of the elements of the hierarchical name space in a single storage segment 104, a higher degree of parallelism may achieved by widely distributing individual elements throughout the storage segments 104. In this arrangement, individual elements are controlled by different storage servers 106. As an example, Fig. 1 illustrates how individual elements 114 of a file path 116, e.g., /Dir1/Dir2/Dir3/My_file, may be placed on five different storage segments 104 and controlled by three different storage servers 106.

[0015] For purposes of this example, servers 106 or 110, which perform an FS service for applications or for SMB, NFS, FTP, HTTP and other servers, may be termed an entry server (ES). In one example, the storage servers 106 which control segments can play both roles. They could be ES for FS level requests originated locally and could be DS for requests coming from other computers.

[0016] To execute an operation, the server 106 or 110 may have to request services of storage servers 106 that control segments associated with objects involved in operation. The storage servers 106 may be termed destination servers (DS). An ES that creates a new file, e.g., My_file, may place it on disk segment 3 and may have to link it into a directory Dir3 on segment 5. Therefore, the ES may have to request services of storage servers 1 or 3 106 to create a file and services of storage server 3 to link the file into directory Dir3.

[0017] The system described with respect to Fig. 1 is configured designed to support a large number of objects and to be able to scale up linearly by adding storage and processing capacities as needed. In hierarchically organized FS, such as shown in Fig. 1, many properties may be passed from a parent node to a child node, e.g., from "/" down the intermediate directories to "My_file". This process may be termed "inheritance". Such inheriting can be done statically or dynamically. Statically inherited attributes are calculated usually during creation of an object and are recorded as the part of the meta-data of a newly recorded object. In one example, the FS does not have to change them automatically.

[0018] Dynamically inheritable attributes may be calculated or revalidated when they are needed. When an object, such as a directory of file, is moved from one

parent to another and if the parents have different sets of dynamically inheritable attributes, the moved object and all its descendants have to inherit a new set of such attributes from the new parent. In contrast, statically inheritable attributes may be persistently recorded as part of on-disk inodes, e.g., data structures that hold metadata about files. When such inodes are renamed from one parent to another, the statically inherited attributes are not changed. Most of the time the static attributes are set during a create or a mkdir FS operation. Thus, dynamically and statically inheritable attributes are semantically different. Dynamic inheritance is processed 'run-time' by ES at appropriate points. Dynamically inherited attributes are associated only with in-core inode objects. There are many different examples or cases when a FS uses dynamic inheritance mechanism. These include setting various policies that describe a number of replicas, placement rules for new objects, tracking changes for various notification purposes, defining security rules, auditing and write-once, read many (WORM) policies, and the like.

[0019] Further, dynamic inheritance operations may engage more objects and correspondently depend on correct actions and coordination of a number of destination servers 106. For example, properties, such as policies, for the newly created My_file can be dynamically inherited from the file system root directory “/”, which is under the control of storage server S3 106. Dynamic inheritance may provide an efficient mechanism to set various policies at any position in the name space. Such policies should affect all the nodes below. In other words, policy-like attributes should be dynamically inherited by all descendants of the nodes in the name space.

[0020] In one example, RI may coincide with the root directory “/” of the FS, FS Root. However, as the number of instances of dynamic inheritance increase, the number of update requests to the root of the file system may increase or become large. The efficiency of the system may be impacted if all intermediate nodes are to be delegated and revalidated when delegations are revoked. Further, the traffic to the DS hosting FS Root associated with revocation of delegations and revalidation of the root grows and system may become unbalanced. Therefore, methods described herein can be used to track contention and take actions in form of redistributing RI activities to a next level of the name space tree. The methods assume that if

delegations are revoked, only the node of the interest and the node that is used as the RI have to be delegated for a given check and have to be revalidated. Avoiding the validation of all intermediate path nodes for validating of dynamically inheritable attributes may substantially improve the operation of the system.

[0021] Fig. 2 is a schematic 200 of an example of propagating an inheritable snapshot time mark (STM). There are many different techniques to identify snapshots. For example, the time when the snapshot was requested may be a characteristic. Accordingly, a direct form of identification may be made by time and termed a Snapshot Time Mark (STM). Such STM can be recorded on any object at any place of the FS name space and is treated and propagated as a dynamically inheritable attribute.

[0022] In this example, it may be assumed to illustrate operation, there are three files, File1 202, File2 204, and File3 206 and three directories, Dir1 208, Dir2 210, and Dir3 212. The file system (FS) also has a file system root "/", termed FS Root 214. The various file system entities are managed by respective destination servers, S1 216, S2 218, and S3 220. A dashed line 222 between a destination server 216, 218, and 220 and a respective file system entity 202-212 indicates that the file system entity is being managed or controlled by the destination server. Thus, for example, the destination server S2 218 manages file system entities File3 206 and Dir2 210.

[0023] Two entry point servers, ES1 224 and ES2 226 are also shown in this example. File system operations, including snapshot operations, can be initiated at the entry point servers 224 and 226.

[0024] For purposes of this example, it may be assumed that the entry point server ES2 226 has worked with entities under /Dir2 210 for some amount of time and thus entities under /Dir2 210 are stored in the cache of ES2 226. It may also be assumed that a previous snapshot request was applied to the FS Root 214, and is associated with STM value stm_1. A snapshot of the root (/) is basically a snapshot of the entire file system under the root. All file system entities under the root (/), such as those shown in Fig. 5, inherit the STM value STM_1 from the root. If a file system entity was created after the time indicated by STM_1 and is subsequently deleted,

prior to another snapshot being taken, then such entity would not be preserved by the file system.

[0025] At a later point in time, the entry point server ES1 may issue a snapshot request 228 to take a snapshot of Dir1 208. The snapshot of Dir1 is a request to take a snapshot of Dir1 and all of the file system entities that are under Dir1. In one example, the root (/) can have other sub-directories, and thus, the snapshot of Dir1 would be a snapshot of a subset of the data of the entire file system. The snapshot of Dir1 is associated with STM value stm_2, which is larger than stm_1. Because stm_2 is larger than stm_1, the new value of stm_2 should be inherited by all file system objects under Dir1 208. As a result, file system entities that were created before stm_1, as well as file system entities created after stm_1 but before stm_2, should be preserved in the snapshot at stm_2.

[0026] In addition to changing the STM value on Dir1 208, ES1 224 may also request a change of the value for dm_gen.stm on the FS Root 214. When "snap /Dir1" is executed and a new value of STM stm_2 was recorded on Dir1 208 by S3 220 and when dm_gen.stm was incremented on the FS Root 214 by S1 216 invalidation requests may be sent by S3 220 and S1 216 to ES2 226 to indicate that ES2 226 cannot trust a locally stored copy of the attributes of Dir1 208 or the FS Root 214. The rest of the cached objects, Dir2 210, Dir3 212, File1 202, File2 204, and File3 206 are not affected by these invalidations. However, the inherited STM should then be stm_2 and not the value of stm_1 that was recorded previously.

[0027] If a user on ES2 226 performs a command 230, "rm –rf /Dir1/Dir2/*", that is supposed to delete all objects below Dir2 210, because of the snap that was requested at stm_2 on /Dir1 208, all the files should be preserved. The first object that may be removed is File3 206. The system performs an inheritance checking process, for example, as described with respect to Fig. 3. It detects that a root "is not cached", i.e., does not have caching delegation associated with it, and re-reads it from S1 216. It may be determined that File3 206 is cached and can be trusted. However, the value for dm_gen.stm for File3 206 is different from the value of dm_gen.stm for the FS Root 214. Consequently, a new list of the nodes hierarchically linking File3 206 to the FS Root 214 is rebuilt.

[0028] Since all of the nodes have values for dm_gen.stm that are different from the current value for dm_gen.stm for the FS Root 214, all of them are placed on the list. While building this list, it may be detected that Dir1 208 is not cached as well the directory is reread. The check inheritance process then propagates the new inherited STM by setting the value of dm_gen.stm for all of the elements of the list to the new value. Further, it detects that the STM value of stm_2 for Dir1 208 is larger than the values for FS Root 214 (0) and Dir2 210 (stm_1) and proceeds to use the value stm_2 for all nodes below Dir1 208. At this point, ES2 226 has performed two network requests to refresh to "non-cached" nodes and updated four in-core objects.

[0029] The next file to be deleted is File2 204. The system performs the same checking inheritance process. At this point, both File2 204 and the FS Root 214 are cached, but the values for dm_gen.stm do not match. Again, the list of nodes hierarchically linking File2 204 to the FS Root 214 are rebuilt. However, the building of the list is stopped after placing a single node, Dir3 212, on the list because the value for dm_gen.stm for Dir3 212 matches the value of dm_gen.stm for the FS Root 214. As a result, no network requested were made, and only one node has been updated. The same result is seen for every other node in the /Dir1/Dir2 sub-tree, minimizing requests made to S1 216.

[0030] Fig. 3 is an example of a method 300 for handling dynamically inheritable attributes. The method may be implemented on the systems discussed with respect to Figs. 1 and 2. At block 302, entities of a hierarchically arranged file system are stored in the distributed storage system. At run-time of the file system, an operation is performed at block 304 that sets a value of a dynamically inheritable attribute of a particular one of the file system entities. For example, the dynamically inheritable attribute can be an STM, as discussed above. In other examples, other types of dynamically inheritable attributes include a replication policy, a placement rule, information relating to tracked changes, a security rule, an audit policy, and so forth.

[0031] At block 306, a determination is made as to whether the dynamically inheritable attribute of at least a second one of the file system entities related to the particular file system entity is to be refreshed. If so, process flow proceeds to block 308, where the value of the dynamically inheritable attribute is propagated to at least the second file system entity.

[0032] The determination that a dynamically inheritable attribute of a file system entity is to be refreshed can be part of a validation procedure, in which the value of the dynamically inheritable attribute for a given file system entity is validated. For example, a validation procedure can be performed of all file system entities along a particular path from a particular file system entity. For performance reasons, techniques or mechanisms according to some implementations are provided to move a root of inheritance down a chain of nodes when contention is detected to be a problem, or return the root of inheritance back to a root directory if all directories below are invalidated. Further, the method may intelligently determine that certain file system entities along the path do not have to be re-validated provided certain conditions are satisfied, as discussed further below. In one example, the techniques of the present application according to some implementations may help avoid traversing the entire chain of nodes, corresponding to a sub-tree of file system entities, during a validation procedure.

[0033] In some examples, a dynamically inherited generation field, such as dm_gen.stm, among others, in an in-core or in memory inode representing a file system entity is used during a validation procedure to determine when traversal of a chain of nodes can be stopped. The dm_gen.stm field is maintained by entry point servers in in-core inodes and is copied from the parent of the inode during the process of propagation of a dynamically inheritable attribute, such as an STM. The dm_gen.stm field is updated at the root of the file system whenever a dynamically inheritable attribute is updated, such as in response to taking of a new snapshot.

[0034] The dm_gen.stm field is changed, for example, monotonically incremented, at the root of the file system with respective changes of the corresponding dynamically inheritable attribute. The dm_gen.stm field is propagated from the root to other nodes during lookups or during a validation procedure to validate the dynamically inheritable attribute, such as the STM, as discussed further with respect to Fig. 4.

[0035] Fig. 4 is an example of a method 400 for propagating dynamically inheritable attributes. The validation procedure of Fig. 4 is performed by an entry point server and is used to validate a dynamically inheritable attribute (e.g. STM) at a given file system entity, referred to as “my_object”. The processing of dynamically

inheritable attribute is based on dynamically inherited generation (dm_gen) field in in-core inode structure representing a file object. Though the description above suggests that dm_gen is a single value, in one example, it may be a vector describing various distinct inheritable properties such as snapshot IDs, generation of antivirus checking tools, generation of placement rules, etc. Below reference is made to a specific dm_gen slot as dm_gen.xxx, for example, dm_gen.stm. This field is maintained by an ES and is copied from the parent of the inode during process of propagation of inheritable attribute.

[0036] The method can be further explained by using the example of snapshot time marks (STM). Rules of STM propagation are based on the fact that time moves in one direction and STM for snapshots grow monotonically. Request to preserve affected object later simply overwrites previous request. Therefore, effective update STM is always the largest requested STM on the chain of all predecessors of the object.

[0037] The method starts at block 402, with the location and revalidation of RI. This may be performed by the method discussed with respect to Fig. 5.

[0038] At block 404, the dm_gen field is checked to confirm that it is the same for all nodes on the name path up to the root of inheritance. Ultimately dm_gen field is monotonically incremented only on the root of inheritance (RI) and then propagated to other nodes by the process of validation of dynamically inheritable attributes. When the value of an inheritable attribute is changed, then the corresponding dm_gen for the root inode must be updated. In one example,the above processdoes not check values of inheritable attributes for all the nodes of the hierarchy. The processing stops as soon as dm_gen attribute of any node in the path matches dm_gen of RI.

[0039] If the dm_gens checked for at 404 do not match, then the process proceeds to task 406. If they do match, the process ends at block 412.

[0040] In one example,if the root is not cached or if my_object is not cached, then the corresponding dm_gen field is not locally accessible at the entry point server. At block 406, the entry point server builds a list (L) of all nodes in the hierarchy from my_object to the root. As part of the process of building the list (L), the entry point server retrieves the root from the corresponding destination server, unless such

information is already cached at the entry point server. Further, information pertaining to my_object is retrieved from the corresponding destination server, unless such information is already cached at the entry point server. Moreover, the entry point server further retrieves information pertaining to any intermediate file system entities between my_object and the root, unless any such information associated with a given intermediate object is already cached at the entry point server.

[0041] As indicated by loop 408, nodes associated with file system entities in the hierarchy are iteratively added to the list, so long as the dm_gen field of the corresponding file system entity does not match the dm_gen field of the root. The process of adding of nodes to the list stops when the dm_gen field of a corresponding file system entity matches the root's dm_gen field.

[0042] After the list has been built, at block 410, the value of the dynamically inheritable attribute, such as STM, is propagated from the first node in the list, where the first node is typically the root of inheritance, to other nodes in the list. In the process according to Fig. 4, propagation of a dynamically inheritable attribute is made to the file system entities associated with nodes in the list. These are the file system entities having dm_gen values that do not match that of the root of inheritance. This may help to reduce traffic and resource consumption associated with propagation of dynamically inheritable attributes, which can grow rapidly in a large distributed storage system.

[0043] After propagation of the value of the dynamically inheritable attribute to the file system entities associated with nodes in the list, the process of Fig. 4 exits at block 412.

[0044] Fig. 5 is an example of a method 500 for finding and validating a root of inheritance for dynamically inheritable attributes in a name space tree. The method can also track contention and set the root of inheritance to a lower level to decrease the loading on a server, or reset the RI back to the primary root if needed.

[0045] The system keeps tracks of the relative cost of updates and queries associated with the RI. This is done using the formulas in Eqn. 1 and 2.

$$N/M > R \qquad \text{Eqn. 1}$$

$$L/M > T \qquad \text{Eqn. 2}$$

In Eqns. 1 and 2, M represents the maximum productivity of the host in the number of input-output transactions per second (IO/s). Generally, M depends on the speed of the network adapter, productivity of the host's CPU, and other factors. To start the tracking, it may be pre-set with a reasonable conservative default, but is continuously tracked, and increased until it matches the actual maximum for the measured total load value. L is the total load value for the server in IO/s for the last time interval, e.g., a second. N is the root load actual number of invalidation, revalidation, and dm_gen update IO requests per second associated with RI.

[0046] In one example, R is the target ratio of root related operations (N) to the maximum productivity (M). Typically it is set between 0.2 and 0.3. T is the threshold ratio of the actual (L) to maximum productivity (M). In this calculation, the inheritance root overloading event triggers when T exceeds a preset level, such as 0.5. An RI overloading event, as determined by the values for R and T, sets the value of dm_gen.shift_down to TRUE and starts reassigning of inheritance root responsibilities to the nodes on the next level of name hierarchy.

[0047] The process of offloading RI responsibilities to the next level is performed during the revalidation of an RI, wherein initial RI responsibilities belong to the FS Root. The RI responsibilities are propagated down the tree until one of the resetting events return the RI back to the FS Root. Accordingly, it may be viewed as an oscillating process that is controlled by two fields, the dm_gen.shift_down flag and the dm_gen.shift_generation. The value of dm_gen.shift_generation indicates that RI responsibilities were assigned to this node during specific shift_generation and the dm_gen.shift_down flag set to FALSE indicates that the current RI is operational. TRUE indicates that an overloading event was detected and the RI must be shifted down.

[0048] The process of selecting and appointing RI to the next level is driven by ES, as block 402 of the method of Fig. 4 for propagating inheritable attributes. At this point RI itself is an inheritable property. For example, RI is calculated and recorded in in-core representations of all ES objects as part of method 400 in Fig. 4.

[0049] The method 500 begins at block 502 by determining if FS Root, and current RI cached and if dm_gen.shift_generations are the same. If not, the RI is

reset to the FS Root at block 504, and process flow returns to block 502 to restart the process. If so, process flow proceeds to block 506.

[0050] At block 506, the ES determines if dm_gen.shift_down set to TRUE and dm_gen.shift_down_gen matching of FS Root. If so, process flow proceeds to block 508. At block 508, the RI is set to the next object down the path to the object. If at block 508 either dm_gen.shift_down is FALSE or dm_gen.shift_down_gen does not match FS Root, process flow proceeds to block 510 at which the method 500 exits back to block 404 of method 400.

[0051] Fig. 6 is a schematic of an example of propagation of roots of inheritance to different levels of a name space tree 600. In one example, in the method 500 described with respect to Fig. 5 newly chosen RIs may be on different levels of the name space tree 600. For example, some name space sub-trees with relatively low activities, e.g., sub-tree 602, may set their RIs closer to the root 603 of FS. Some dormant sub-trees may not appoint RIs at all, e.g., sub-tree 604. Further, some active sub-trees may move their RIs deeper into the name space tree 600 and closer to the areas of activity.

[0052] Fig. 6 illustrates how RIs were propagated to different levels of the name space tree 600. RI nodes are represented as shaded areas. More active portions, such as sub-trees 606 and 608, of the name space had their RIs propagated deeper, for example, to Dir_11 610 and Dir_12 612 levels. Less active portions, e.g., sub-tree 602, of the name space had their RIs propagated shallower to Dir_2 614 and dormant areas of the name space below Dir_3 616 do not have their RI established.

[0053] There are several events in the system when shifting of RI responsibilities down the name space tree 600 is stopped and the RI is returned to the FS Root 603. For example, if the dm_gen.shift_generation value of the FS root 603 is changed then all RIs are reset to the FS Root 603. As described with respect to the method 500 of Fig. 5, this event is analyzed by every ES side revalidation of dynamically inheritable attributes, and if dm_gen.shift_generations of the current RI and FS Root do not match, the RI is reset to the FS Root 603.

[0054] A number of events may trigger an RI Reset. For example, a reboot of the DS controlling the FS Root node or a transfer of control of FS Root segment. In this

case, FS Root generation is initialized with a content randomized by the reboot time and ID of the handling host.

[0055] Another event that may trigger an RI reset to the FS Root 603, is a renaming of a portion of the name space tree 600, with concomitant changes of inheritance. For example, some parts of the name space may be moved out of their scope of inheritance. In such case, the elements that are moved may either retain some inheritable properties from a previous path or have to re-inherit them from a new path. To assure that this process affects all objects RI is reset, either to the most common ancestor or, as a simplification, to the FS Root 603.

[0056] Changing inheritable properties between FS Root 603 and RI may also trigger a reset of the RI. For example, to the place of change or, as a simplification, to the FS Root 603.

[0057] Fig. 7 is an example of a server 700 that may allow roots of inheritance to be moved up or down a hierarchical name space tree. The server may be an ES as described herein. The server 700 has one or more processors 702, which may be single core processors, multi-core processors, virtual processors, cloud processors, or any combinations thereof. The processors 702 may be coupled to a network interface controller (NIC) 704 through a bus 706. The bus 706 may include any number of technologies, such as PCIe, Fibre Channel, or any number of other commercial or proprietary bus technologies.

[0058] The processors 702 may be coupled to a storage system 708 through the bus 706. The storage system 708 may include a RAM, a ROM, a flash drive, a hard drive, a SAN, an optical drive or any combinations thereof. The storage system 708 forms a tangible machine readable medium that includes instructions code to direct the processors 702 to perform the methods described herein. For example, a root validation module 710 could include code to direct the processor to validate a root of inheritance, for example, as discussed with respect to Fig. 5. A contention monitor 712 may direct the processor to monitor the amount of propagation accesses to a root directory controller, or other directory controllers, to determine when the RI should be moved down a name space tree. A node list 714 may be used to keep track of the attributes for the nodes, including, for example, STMs and RIs. An

attribute propagator 716 may be used to direct the processors 702 to confirm the RI and propagate attributes in a name space tree.

[0059] While the present techniques may be susceptible to various modifications and alternative forms, the exemplary examples discussed above have been shown only by way of example. It is to be understood that the techniques are not intended to be limited to the particular examples disclosed herein. Indeed, the present techniques include all alternatives, modifications, and equivalents falling within the scope of the present techniques.

CLAIMS

What is claimed is:

1. A method for controlling contention in hierarchical storage, the method comprising:

monitoring loading on a destination server (DS) controlling a directory that is a root of inheritance (RI) for a dynamically inheritable attribute; and

if the loading on the DS is higher than a predetermined level, then moving the RI for the dynamically inheritable attribute to a lower level in a name space tree.

2. The method of claim 1, further comprising:

finding the RI for the dynamically inheritable attribute;

revalidating the RI for the dynamically inheritable attribute;

building a list of entities from the RI to the dynamically inheritable attribute; and

propagating the dynamically inheritable attribute to each of the entities in the list.

3. The method of claim 1, further comprising resetting the root of inheritance to a file system root directory (FS Root) if the FS Root is not cached, the RI is not cached or the generations between the FS root and the RI do not match.

4. The method of claim 1, further comprising determining the loading by determining a ratio (T) between the number of input-output transactions per time unit to the maximum loading for a DS over the same time unit.

5. The method of claim 4, further comprising using T to determine loading, and wherein the predetermined level is set to about 0.5.

6. The method of claim 1, further comprising determining the loading by determining a ratio (R) between the number of transactions for a hosted root of inheritance per time unit to the maximum loading for a DS over the same time unit.

7. The method of claim 6, further comprising using R to determine loading, and wherein the predetermined level is set to between about 0.2 and 0.3.

8. The method of claim 1, further comprising:

at run-time of a file system, performing an operation that sets a value of the dynamically inheritable attribute of a particular entity, wherein the dynamically inheritable attribute relates to a snapshot;

determining whether the dynamically inheritable attribute of at least a second entity is to be refreshed; and in response to determining that the dynamically inheritable attribute of at least the second entity is to be refreshed, propagating the value of the dynamically inheritable attribute to at least the second entity.

9. The method of claim 8, wherein propagating the value to the second entity comprises propagating the value to the second entity that is a descendant of the particular entity in a hierarchy of the hierarchically storage system.

10. The method of claim 9, wherein propagating the value of the dynamically inheritable attribute comprises propagating a time property of the snapshot.

11. An entry server (ES) for a hierarchical storage system comprising:

a processor; and

a storage system, where the storage system comprises:

a list of file system entities in a name space tree, wherein each entry in the list comprises a root of inheritance and a dynamically inheritable attribute;

instructions to direct the processor to:

monitor loading on a destination server (DS) controlling a segment that is the root of inheritance (RI) for the dynamically inheritable attribute; and

if the loading on the DS is higher than a predetermined level, move the RI for the dynamically inheritable attribute to a lower level in the name space tree.

12. The entry server of claim 11, comprising:

a network interface; and

the storage system, wherein the storage system comprises instructions to direct the processor to access the DS over a network.

13. The entry server of claim 11, comprising instructions to direct the processor to:

find the RI for the dynamically inheritable attribute;

revalidate the RI for the dynamically inheritable attribute;

build a list of entities from the RI to the dynamically inheritable attribute; and

propagate the dynamically inheritable attribute to each of the entities in the list.

14. The entry server of claim 11, comprising instructions to determine loading on the DS by determining a ratio (T) between the number of input-output transactions per time unit to the maximum loading for a DS over the same time unit.

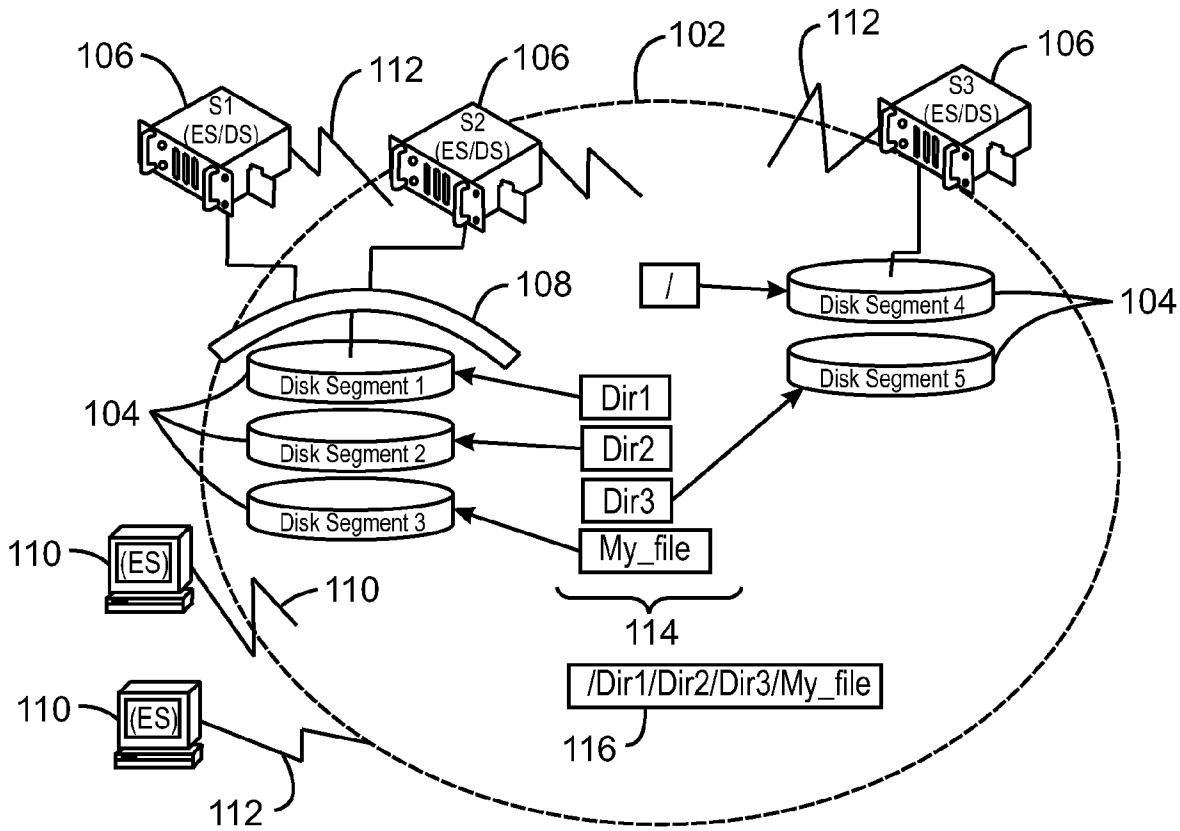
15. A non-transitory machine readable medium comprising instructions to instruct a processor to:

monitor contention on a destination server and move a root of inheritance for an entity to a lower level of a name space tree if the contention is beyond preset limits;

validate the root of inheritance (RI) for the entity;

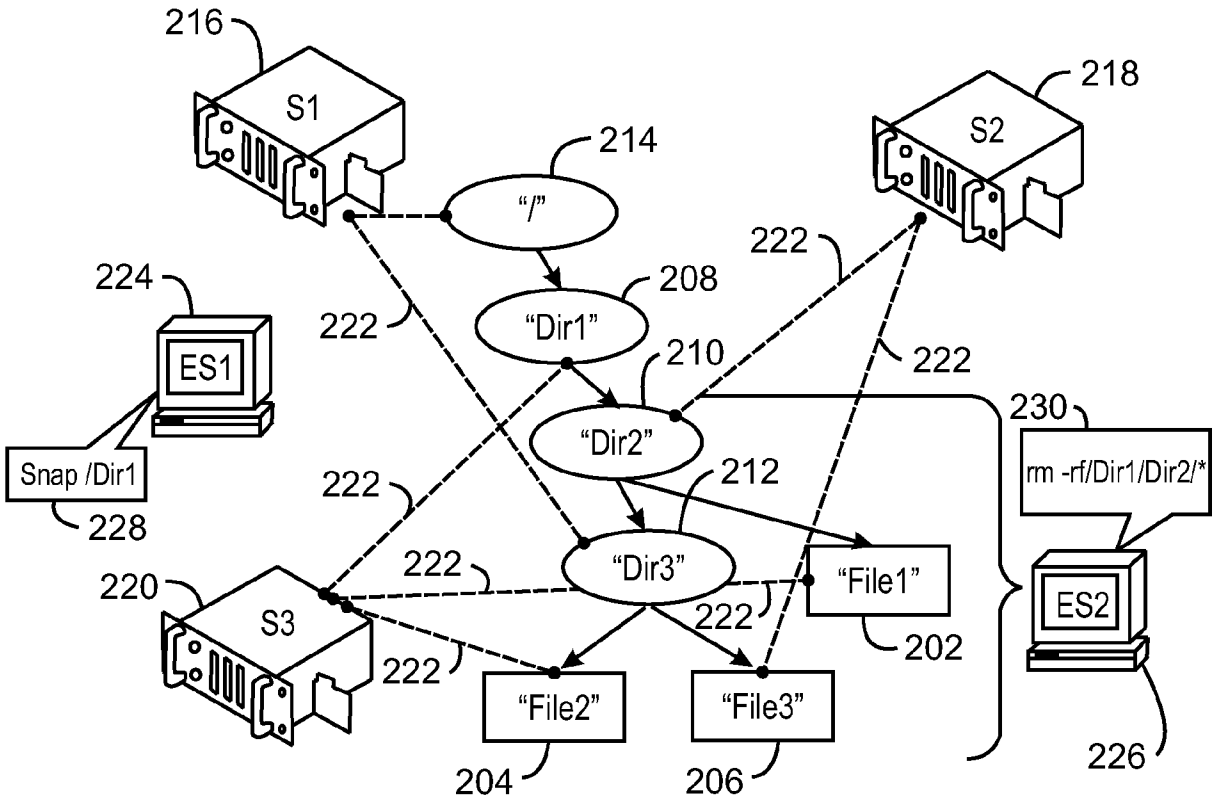
build a node list of entities in the name space tree from the RI to the entity; and

propagate a dynamically inheritable attribute to each of the entities in the node list.


106
S1
(ES/DS)
112
106
S2
(ES/DS)
102
112
106
S3
(ES/DS)
108
/
Disk Segment 4
104
Disk Segment 1
Disk Segment 5
104
Dir1
Disk Segment 2
Dir2
Dir3
Disk Segment 3
My_file
110
(ES)
110
114
/Dir1/Dir2/Dir3/My_file
110
(ES)
116
112

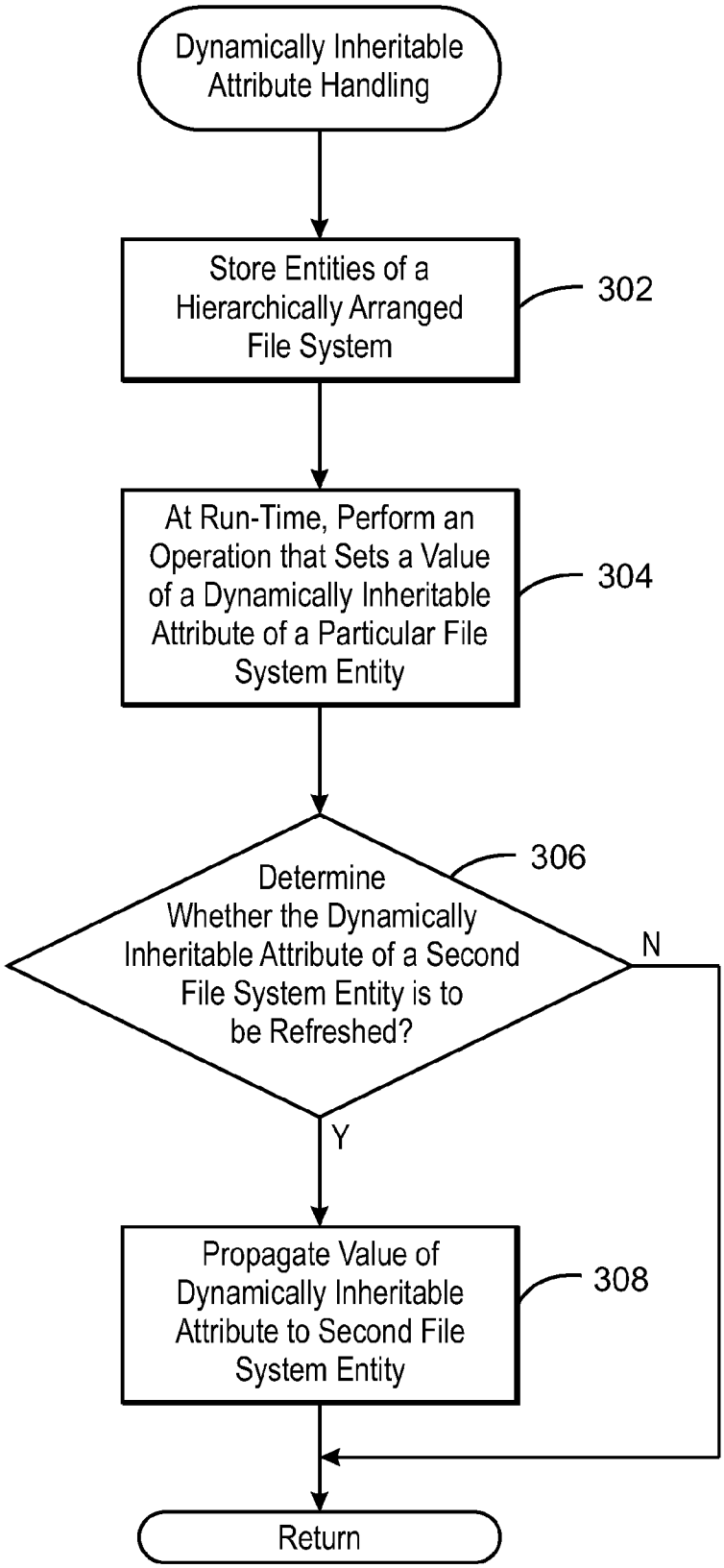
100

FIG. 1


216
S1
214
"/"
218
S2
222
224
208
"Dir1"
222
222
ES1
210
"Dir2"
230
rm -rf/Dir1/Dir2/*
Snap /Dir1
228
222
212
"Dir3"
"File1"
222
220
S3
222
222
202
"File2"
"File3"
ES2
204
206
226

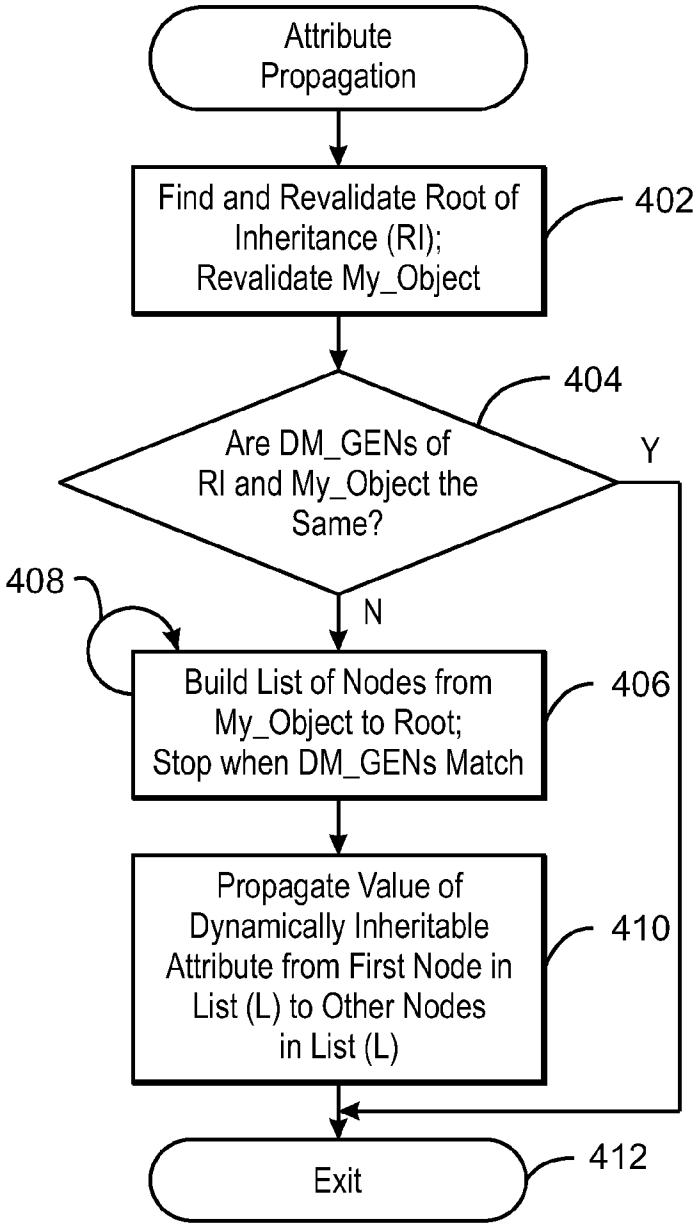
200

FIG. 2


Dynamically Inheritable Attribute Handling
Store Entities of a Hierarchically Arranged File System
302
At Run-Time, Perform an Operation that Sets a Value of a Dynamically Inheritable Attribute of a Particular File System Entity
304
Determine Whether the Dynamically Inheritable Attribute of a Second File System Entity is to be Refreshed?
306
N
Y
Propagate Value of Dynamically Inheritable Attribute to Second File System Entity
308
Return

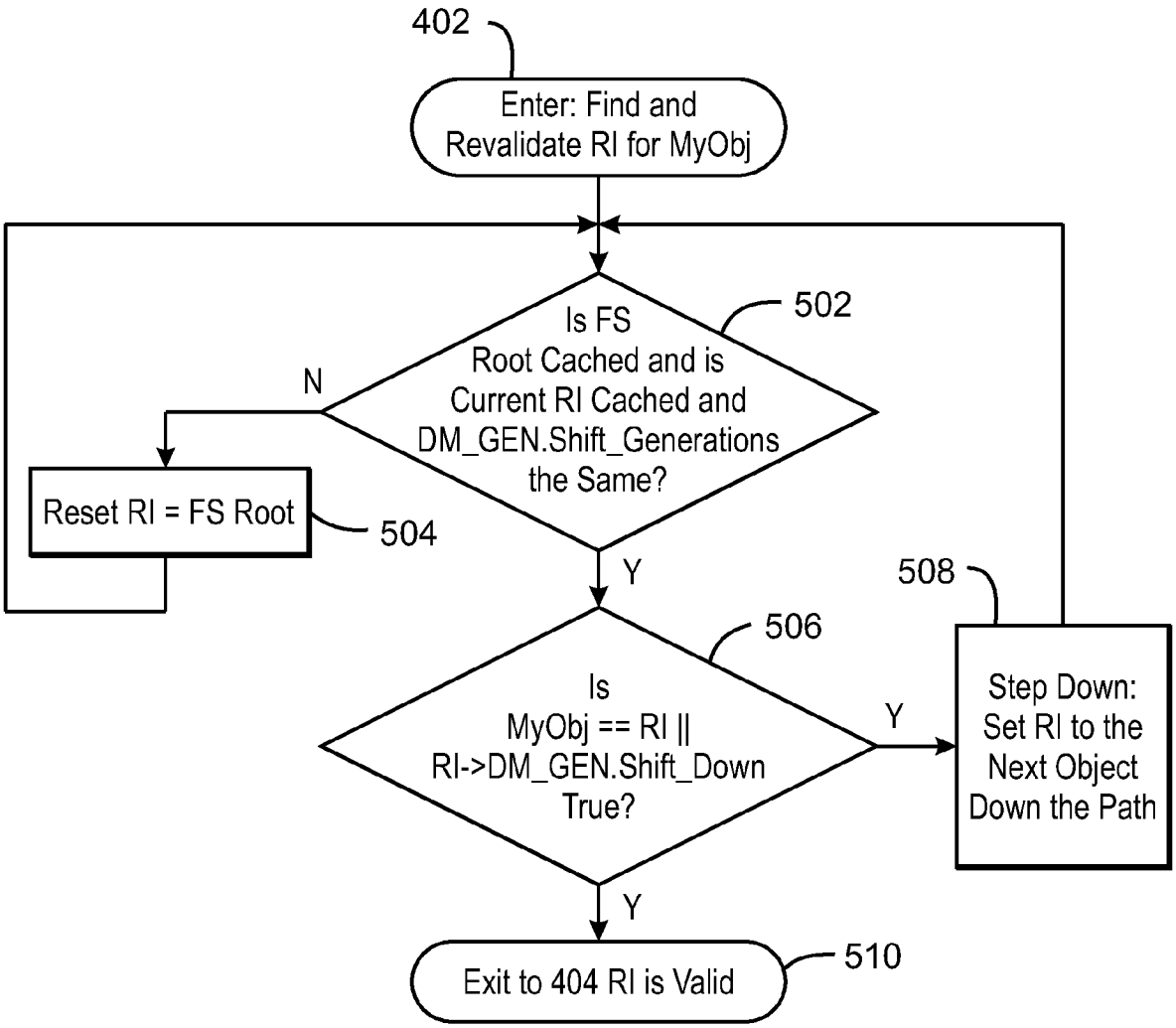
300

FIG. 3


Attribute Propagation
Find and Revalidate Root of Inheritance (RI); Revalidate My_Object
402
404
Are DM_GENs of RI and My_Object the Same?
Y
N
408
Build List of Nodes from My_Object to Root; Stop when DM_GENs Match
406
Propagate Value of Dynamically Inheritable Attribute from First Node in List (L) to Other Nodes in List (L)
410
Exit
412

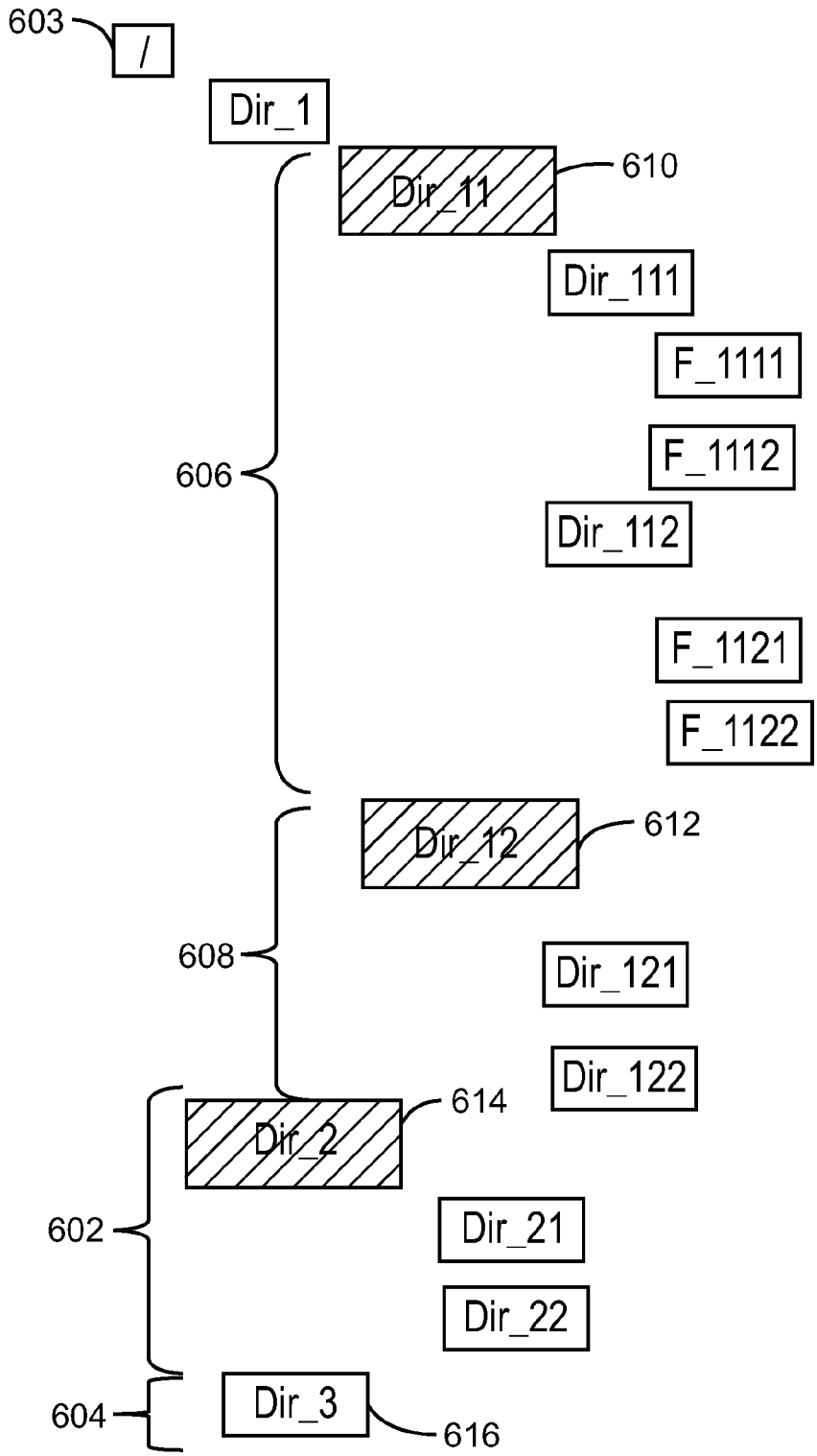
400

FIG. 4


402
Enter: Find and Revalidate RI for MyObj
Is FS Root Cached and is Current RI Cached and DM_GEN.Shift_Generations the Same?
502
N
Reset RI = FS Root
504
Y
Is MyObj == RI || RI->DM_GEN.Shift_Down True?
506
Y
Step Down: Set RI to the Next Object Down the Path
508
Y
Exit to 404 RI is Valid
510

500

FIG. 5


603
/
Dir_1
Dir_11
610
Dir_111
F_1111
F_1112
606
Dir_112
F_1121
F_1122
Dir_12
612
608
Dir_121
Dir_122
614
Dir_2
602
Dir_21
Dir_22
604
Dir_3
616

600

FIG. 6

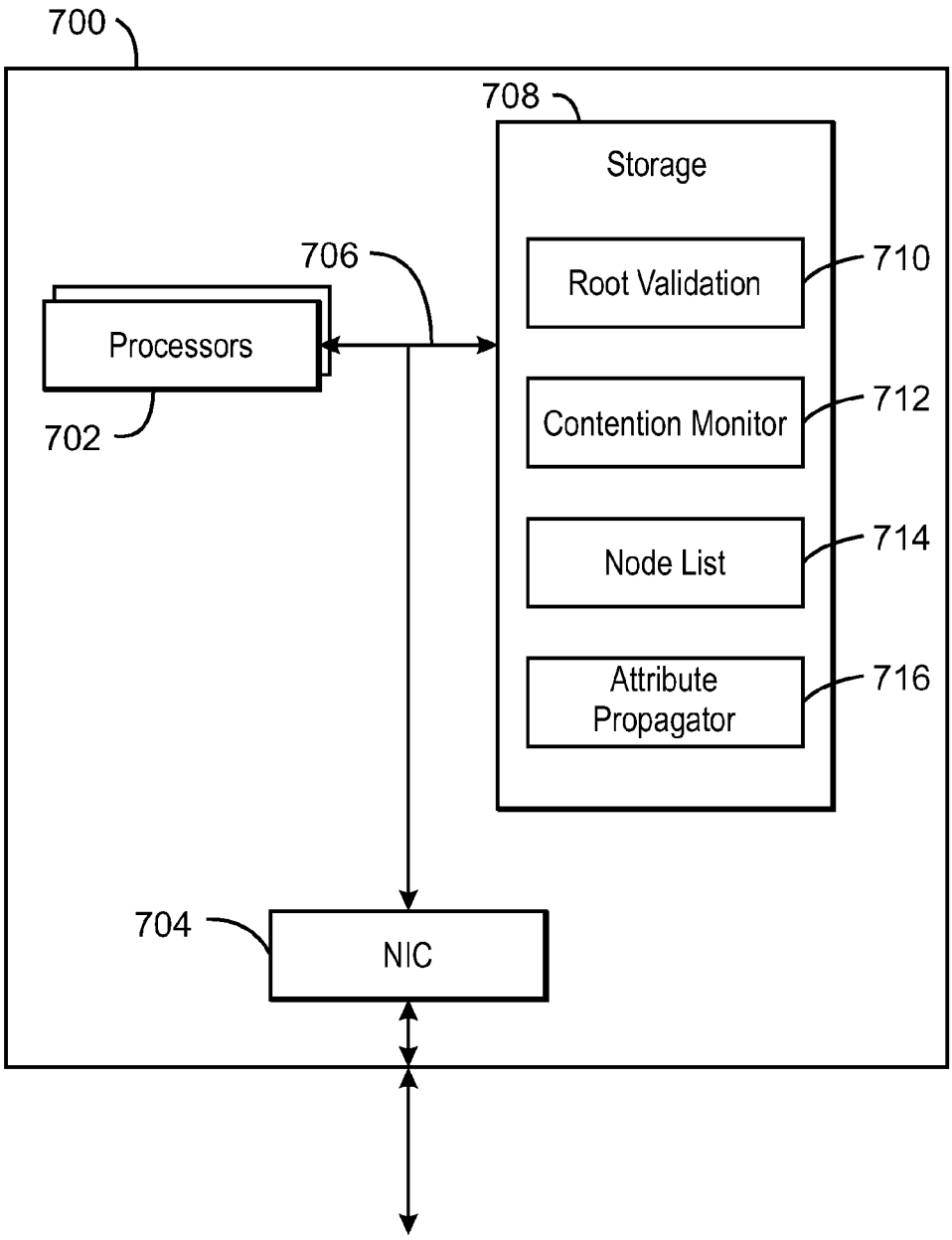

700
708
Storage
706
Root Validation
710
Processors
712
Contention Monitor
702
714
Node List
716
Attribute Propagator
704
NIC

FIG. 7

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2015/013797

A. CLASSIFICATION OF SUBJECT MATTER

H04L 12/26(2006.01)i, H04L 12/24(2006.01)i

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
H04L 12/26; G06F 17/00; G06F 7/00; G06F 17/30; G06F 15173; H04L 12/24

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
Korean utility models and applications for utility models
Japanese utility models and applications for utility models

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
eKOMPASS(KIPO internal) & Keywords: root, inheritance, dynamic, high, low, level, hierarchical, file system, snapshot, load balancing, directory, ratio, propagate, revalidate.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 6834301B1 (PAUL F. HANCHETT) 21 December 2004 See column 1, line 66 - column 2, line 59; column 3, line 43 - column 8, line 32; column 10, line 21 - column 11, line 34; claims 1-3; and figures 1-2, 5.	1-3,8-13,15
A		4-7,14
Y	US 2012-0303585 A1 (BORIS ZUCKERMAN et al.) 29 November 2012 See paragraphs [0007]-[0054], [0061], [0067]; claims 1-3, 13; and figures 1-4, 7.	1-3,8-13,15
A	US 7761432 B2 (MATTHEW A. AHRENS et al.) 20 July 2010 See column 3, line 24 - column 4, line 44; and figures 3-4.	1-15
A	US 2014-0214773 A1 (HEWLETT-PACKARD DEVELOPMENT COMPANY, L.P.) 31 July 2014 See paragraphs [0035]-[0054]; and figures 4-5.	1-15
A	US 7383286 B2 (SEISHIRO HAMANAKA et al.) 03 June 2008 See column 7, lines 41-63; and figure 5.	1-15

☐ Further documents are listed in the continuation of Box C.

☒ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents,such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
12 October 2015 (12.10.2015)	**28 October 2015 (28.10.2015)**

Name and mailing address of the ISA/KR
International Application Division
Korean Intellectual Property Office
189 Cheongsa-ro, Seo-gu, Daejeon Metropolitan City, 35208, Republic of Korea
Facsimile No. +82-42-472-7140

Authorized officer
KIM, Seong Woo
Telephone No. +82-42-481-3348

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.
PCT/US2015/013797

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 6834301 B1	21/12/2004	None	
US 2012-0303585 A1	29/11/2012	US 09069780 B2	30/06/2015
US 7761432 B2	20/07/2010	US 2007-0106678 A1	10/05/2007
US 2014-0214773 A1	31/07/2014	None	
US 7383286 B2	03/06/2008	JP 2003-022206 A US 2003-0009484 A1	24/01/2003 09/01/2003