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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
14 March 2013 (14.03.2013)

WIPO | PCT

(10) International Publication Number
WO 2013/036697 A2

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

(21) International Application Number:
PCT/US2012/054038

(22) International Filing Date:
7 September 2012 (07.09.2012)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
13/228,732 9 September 2011 (09.09.2011) US

(71) Applicant *(for all designated States except US)*: **MICROSOFT CORPORATION** [US/US]; One Microsoft Way, Redmond, Washington 98052-6399 (US).

(72) Inventors: **KRUSE, David, M.**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, Washington 98052-6399 (US). **FATHALLA, Diaa**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, Washington 98052-6399 (US). **PINKERTON, James, T.**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, Washington 98052-6399 (US). **GEORGE, Mathew**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, Washington 98052-6399 (US). **PRAHALAD, Prashanth**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, Washington 98052-6399 (US). **JOLLY, Thomas, E.**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, Washington 98052-6399 (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, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, 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, 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, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

[Continued on next page]

(54) **Title**: CLUSTERED CLIENT FAILOVER

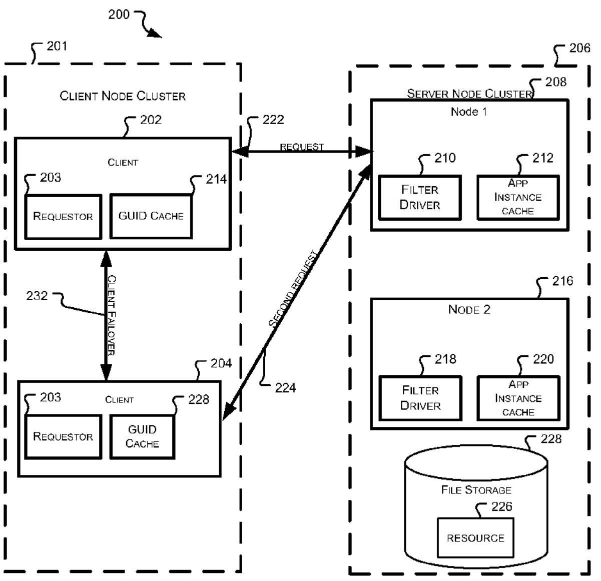

FIG. 2

(57) **Abstract**: An application instance identifier is employed with various systems and methods in order to provide a requestor with continuous access to a resource when operating in a client clustered environment. A requestor residing on a first client in may attempt to access a resource. The first client sends a request to access the resource. The request may be associated with an application instance identifier that identifiers the requestor. At some point, the first client fails and the requestor is associated with a second client via a failover mechanism. The second client sends a second request to access the resource on behalf of the requestor. The second request is associated with the requestor's application instance identifier. The application instance identifier is used to identify the second request as belonging to the same requestor as the first request, thereby granting the second request to access the resource while avoiding a conflict situation.

WO 2013/036697 A2

Declarations under Rule 4.17:

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

— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

Published:

— *without international search report and to be republished upon receipt of that report (Rule 48.2(g))*

CLUSTERED CLIENT FAILOVER

Background

[0001] Clustered environments, e.g., environments where workload is distributed across multiple machines, are commonly used to provide failover and high availability of information to clients. Clustered environments allow clients to access resources via the one or more nodes that are a part of the environment. A clustered environment can act as a client, a server, or both. In a client cluster server, an application may reside on any of the nodes that make up the cluster. The application may issue requests for resources that are stored locally within the client cluster or stored remotely. If an error occurs on the node, the client failover, or migrate, to a different node in the cluster. However, when the client again requests to access a resource that it was working with at the time of the error, the resource may be fenced or locked by the server for the previous client node that the application resided on.

[0002] It is with respect to these and other considerations that embodiments have been made. Also, although relatively specific problems have been discussed, it should be understood that the embodiments should not be limited to solving the specific problems identified in the background.

Summary

[0003] This summary is provided to introduce a selection of concepts in a simplified form that are further described below in the Detail Description section. This summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used as an aid in determining the scope of the claimed subject matter.

[0004] Systems and methods are disclosed herein that provide an application or a process with continuous access to a resource after the application migrates to a new node in a clustered client environment. An application or process residing on a node in a client cluster sends a request to a server to access a resource. In embodiments, a unique application instance identifier is used to identify an application requesting a resource. The unique application identifier may be provided with the request. When the client accesses a resource, the application instance identifier is associated with the requested resource.

[0005] Before the application or process completes its operations on the resource, the node upon which the client resides in the clustered environment may experience an error that causes it to fail or otherwise lose access to the resource prior to the application properly releasing the resource. In such circumstances, the resource may remain in a fenced or locked state on the server per the previous client's request. Upon failing over to a different node in the client cluster, the application on the new client node may reestablish a connection with the server managing the resource and make a second request for the resource that the application previously had access to at the time of the error. The second request may include the application instance identifier was sent with the first request. Although the second request for the resource may be received from a different node in the clustered environment, the application instance identifier permits the server managing the request to determine that the second request belongs to the same application or process that had previously locked the resource. Doing so allows the server to invalidate the resource and grant the client's second request to access the resource while insuring a conflict situation does not arise.

[0006] Embodiments may be implemented as a computer process, a computing system or as an article of manufacture such as a computer program product or computer readable media. The computer program product may be a computer storage media readable by a computer system and encoding a computer program of instructions for executing a computer process. The computer program product may also be a propagated signal on a carrier readable by a computing system and encoding a computer program of instructions for executing a computer process.

Brief Description of the Drawings

[0007] Non-limiting and non-exhaustive embodiments are described with reference to the following figures.

[0008] FIG. 1 illustrates a system that may be used to implement embodiments described herein.

[0009] FIG. 2 is a block diagram illustrating a software environment that may be used to implement the embodiments disclosed herein.

[0010] FIG. 3 is an embodiment of a method that a client may perform to gain continuous access to a resource in a clustered environment.

[0011] FIG. 4 is an embodiment of a method performed by a node in a clustered environment to provide continuous access to a resource.

[0012] FIG. 5 illustrates a block diagram of a computing environment suitable for implementing embodiments.

Detailed Description

[0013] Various embodiments are described more fully below with reference to the accompanying drawings, which form a part hereof, and which show specific exemplary embodiments. However, embodiments may be implemented in many different forms and should not be construed as limited to the embodiments set forth herein; rather, these embodiments are provided so that this disclosure will be thorough and complete, and will fully convey the scope of the embodiments to those skilled in the art. Embodiments may be practiced as methods, systems or devices. Accordingly, embodiments may take the form of a hardware implementation, an entirely software implementation or an implementation combining software and hardware aspects. The following detailed description is, therefore, not to be taken in a limiting sense.

[0014] Embodiments of the present disclosure are related to providing clustered client failover mechanisms that allow a requestor to regain access to a resource after a failover event. In embodiments, a requestor may be a process, an application, or one or more child processes of an application. A resource may be a file, an object, data, or any other type of resource in a computing environment. In embodiments, a resource may reside on a standalone server or it may reside in a clustered environment. In the embodiments disclosed herein, a clustered environment may include one or more nodes (e.g., client and/or server devices).

[0015] In an example embodiment, an application residing on a node in a clustered environment may request access to a particular resource. In embodiments, the resource may be stored locally (e.g., on the client node), in a remote device (e.g., a remote server or a different node in the client clustered environment), or in a clustered environment (e.g., an environment containing multiple nodes) that is different from the client clustered environment. For example, in embodiments the clustered environment may be a client or server cluster; however, one of skill in the art will appreciate that the systems and methods disclosed herein may be employed in any other type of environment, such as, but not limited to, a virtual network.

[0016] In such environments, resources may be shared among clients and applications. When an application accesses a resource, the resource may be fenced or locked, thereby prohibiting other applications from accessing the resource until the accessing application releases the resource. Fencing or locking the resource may be employed to protect against a conflict, that is, protect against modification of the resource by another application before the accessing application has performed its operations on the resource. However, if the node in a clustered client environment fails, the application accessing the resource may not properly release the resource from a fenced or locked state. For example, the client node accessing the resource on behalf of the application may lose a network connection, may crash, or may otherwise lose access to the resource prior to the application completing its operations and properly releasing the resource. Thus, the resource may remain in a state in which it is unavailable to other clients or applications. Mechanisms may be employed that automatically release a resource from a fenced or locked state, thereby preventing the resource from being permanently locked out. However, such mechanisms often wait a period of time before releasing a fenced of locked resource.

[0017] In some instances, when the application performs a failover to migrate from the failed client node to a different client node in the client cluster, the application may attempt to reestablish its previous connection with the server and resume its operation(s) on the resource via the different client node. However, because the resource was not properly released by the failed client node, which previously accessed the resource on the application's behalf, due to the error, the application that had previous access to the resource may not be able to resume its access of the resource until the server releases the resource from its fenced or locked state. However, because a different node is now attempting to access the resource on the application's behalf, the server may not be able to identify the application as the same application that previously established the lock on the resource. However, because the same application is trying to access the resource, a conflict situation does not exist. In such situations, waiting for the server to release the previous lock on the resource may cause an unacceptable delay for the application.

[0018] As described, because the application is operating in a clustered client environment, when the application requests to access the resource a second time, the request to access to the resource may be made from a different location, such as, a different node in the client clustered environment. Thus, the second request may come from a location or different IP address. Because the request may be made from a different location, a server may have difficultly ensuring that the client or application attempting to again access the resource is actually the same client that previously accessed the resource. The systems and methods disclosed herein provide mechanisms to identify situations where the same application is attempting to access a resource, thereby avoiding such delay and providing an application continuous access to the resource.

[0019] FIG. 1 illustrates a system 100 that may be used to implement some of the embodiments disclosed herein. System 100 includes client cluster 102 and a server cluster 106. Client cluster includes multiple nodes, such as clients 102A and 102B. Clients 102A and 102B may be a device or application residing in client cluster 102. Client cluster 102 may communicate with server cluster 106 through network 108. In embodiments, network

108 may be the Internet, a WAN, a LAN, or any other type of network known to the art. Server cluster 106 stores resources that are accessed by applications on client cluster 102 (e.g., applications residing on Client 102A or Client 102B). In embodiments, a client (e.g., Client 102A) may establish a session with cluster 106 to access the resources on cluster 106 on behalf of an application residing on the client. Although in FIG. 1 client cluster 102 only includes two clients (e.g., Client 102A and Client 102B), one of skill in the art will appreciate that any number of clients may be included in client cluster 102.

[0020] As shown in FIG. 1 server cluster 106 includes servers 106A, 106B, and 106C, which provide both high availability and redundancy for the information stored on cluster 106. In embodiments, the cluster 106 may have a file system, a database, or other information that is accessed by clients 102 and 104. Although three servers are shown in FIG. 1, in other embodiments cluster 106 may include more than three servers, or fewer than three servers. Furthermore, while the embodiments herein described relate to a client communicating with a server that is part of a server cluster, one of skill in the art will appreciate that the embodiments disclosed herein may also be performed using a standalone server.

[0021] In embodiments, client cluster 102 provides failover mechanisms that allow a client to migrate from a first client node to a second client node in case of an error or failure occurring on the first client node. One of skill in the art will appreciate that any type of failover mechanism may be employed with the systems and methods disclosed herein. The methods and systems disclosed herein may be employed to avoid undue delay when an application attempts to regain access to a resource migrating from one client to another (e.g., from Client 102A to Client 102B) in the case of a failover. In embodiments, an application instance identifier identifying the application accessing the resource may be associated with the resource. The application instance identifier may be a globally unique identifier (GUID) that is associated with an application, an action performed by an application, or a child process of an application. For example, in one embodiment an application may be associated with an application instance identifier that is a GUID. In another embodiment, an application instance identifier may be associated with a specific operation or action performed by an application. For example, if the application issues two different open requests for two different files, each open request may have its own application instance identifier. In yet another embodiment, an application instance identifier may be associated with one or more child processes of the application. As will be clear to one of skill in the art from the embodiments described herein, associating the application instance identifier of an application with its one or more child processes will allow the child processes to access the resource if the resource is placed in a locked or fenced state that belongs to the application. In embodiments, the application instance identifier may be sent by the client when, or after, sending a request for a resource.

[0022] In accordance with another embodiment, in addition to storing information accessed by clients that are a part of client cluster 102, server cluster 106 also provide a failover mechanism that allows continuous access of a resource in case of a server node failure. Again, one of skill in the art will appreciate that any type of failover mechanism may be employed with the systems and methods disclosed herein.

[0023] In embodiments, when a client requests access to a resource on behalf of an application, the application instance identifier of the application is sent with the request. The server receiving the request may associate the application instance identifier with the resource. For example, the server cluster may store the application instance identifier in a table or cache located on one or more nodes (e.g., servers such as servers 106A, 106B, and/or 106C) located in the server cluster 106 in such a manner that the application instance identifier is associated with the resource. Before the client is done with the resource, the client may experience an error that will force it to lose connection with the resource. For example, the client hosting the application or performing requests or operations on the application's behalf may lose its network connection to the server cluster, the client may crash, or any other type of error may occur that interferes with the applications use of the resource. Upon experiencing the error, the application may failover to a new client node in the client cluster 102. The new client node may reconnect to the server cluster and send a second request to access the resource on the application's behalf. In embodiments, the client may reconnect to the same node in the sever cluster 106 or a different node. The second request to access the resource may include the application instance identifier of the application. Upon receiving the second request, the sever (e.g., a Server 106A of server cluster 106) compares the application instance identifier of the second request with the application instance identifier associated with the resource. If the two application instance identifiers match, the server cluster invalidates the resource. In embodiments, invalidating the resource may comprise closing a file, removing a lock on the resource, or otherwise taking any action that frees the resource for use. The server node may then grant the application's second request to access the resource. If the application instance identifier of the second node does not match application identifier associated with the resource, the server will not allow access to the resource until the resource becomes free.

[0024] To illustrate one embodiment, a requestor (e.g., a process, application, etc.) on Client 102A in client cluster 106 may request that Client 102A establishes a session with a server of server cluster 106. For example, Client 102A may establish a session with server 106A to access a database stored that is on server 106A or that is a part of server cluster 106, in which server 106A may access the database. Client 102A then sends a request for a resource on behalf of the requestor. An application instance identifier that identifies the requestor is associated with the request. In embodiments, the request may include the application instance identifier or the application instance identifier may be sent separately in a manner such that the server 106A can determine that the application instance identifier is associated with the request. In yet another embodiment, the server 106A or the server cluster 106A may already have information needed to associate the application instance identifier with the request without having to receive the application instance identifier along with the request. Server 106A then grants the requestor access to the resource, thereby allowing the requestor to perform operations on or otherwise access the resource. When granting the requestor access to the resource, server 106A associates an application instance identifier with the resource in a manner that indicates the requestor is currently accessing the resource. The resource may then be fenced or locked so other clients or applications cannot access or modify the resource until client 102 has completed its operation.

[0025] Before the requestor completes its operations on the resource, an error occurs that causes Client 102A to fail or to otherwise lose its connection to the resource. Because client requestor completed its operation, it has not released control of the resource. Thus, the resource may remain in a fenced or locked state. The requestor or client cluster 102 may employ a failover mechanism to migrate the requestor from client 102A to client 102B. Once the failover operation is complete, client 102B may reconnect to server cluster 106 on the requestor's behalf. Client 102B may reconnect to server 106A or establish a new connection with any other server in server cluster 106 (e.g., server 106B or 106C). In an example situation, Client 102B reconnects to server 106A. Upon reconnecting, the client 102B may send a second request to access the resource on behalf of the requestor. As previously noted, because the requestor did not release control the resource, the resource may still be in a locked or fenced state. In order to access the resource, without waiting for the server to automatically change the state of the resource, for example, through a time out operation, the requestor may again provide its application instance identifier with the second request. Server 106A compares the application instance identifier provided with the second request to the application instance identifier associated with the resource. For example, by comparing the application instance identifier received or otherwise associated with the second request to an application instance identifier that Server 106A associated with the resource. The associated application instance identifier may be stored in a local cache or table of server 106A, or it may be stored elsewhere in server cluster 106. If the application instance identifier stored in the cache matches the application instance identifier that is associated with the resource, server 106A invalidates or otherwise frees the resource and allows client 102B to again access the resource on the requestor's behalf without waiting for the resource to be released by some other mechanism (e.g., by the fenced or locked state timing out). If the application instance identifiers do not match, Client 102B will have to wait for the resource to become free before accessing it.

[0026] While in the above example Client 102B reconnected to the same server 106A, it is also possible, in other embodiments, for the client to connect to another node in server cluster 106. For example, client 102B may reconnect to server 106B and submit a second request to regain access to the resource on behalf of the requestor. The second request may again be associated with the requestor's application instance identifier, for example, by being included in the second request or otherwise associated with the second request. In this example, Server 106B may not have the application instance identifier associated with the resource stored in its local cache because the original access of the resource was on server 106A. In such a situation, server 106B may contact the other servers in server cluster 106 to determine if they have an application identifier associated with the resource. If the application identifier associated with the resource is stored on a different node in the server cluster (e.g., server 106A), the application instance identifier on the other node in the server cluster is compared with the application instance identifier provided with the second request. If they match, server 106B may send a request to server 106A to invalidate the resource, and then server 106B may allow the requestor (now on client 102B) to access the resource. If the application instance identifiers do not match, Client 102B will have to wait for the resource to free.

[0027] Based on the above examples, one of skill in the art will appreciate that any client node in the client cluster 102 may request to access for, and then provide access to, a requestor in the client cluster 102. Furthermore, any server node in a server cluster (e.g., any server in server cluster 106) is capable of determining whether the requestor previously had access to the resource even if the access occurred on a different server node in the server cluster. One of skill in the art will appreciate that the following description is merely one example of how the embodiment shown in FIG. 1 may operate and other embodiments exist. For example, rather than accessing resources on a remote server or server cluster, a client nodes may perform the embodiments described herein to provide requestors (e.g., applications or processes) continuous access to resources residing in the clustered environment (e.g., on the same or different client cluster nodes that make up the client cluster). As described in greater detail below, embodiments described herein may involve various different steps or operations. Furthermore, the embodiments described herein may be implemented using any appropriate software or hardware component or module.

[0028] Turning now to FIG. 2, the figure illustrates a block diagram of a software environment 200 showing client node cluster 201 with multiple client nodes (e.g., clients 202 and 204) and a server node cluster 206 with multiple server nodes (e.g., Node 1 208 and Node 2 216). In embodiments, client 202 requests to access a resource, such as resource 226, in a server cluster environment 206 on behalf of a requestor. Client node cluster 201 may be a client cluster such as client cluster 102 (FIG. 1). Although not illustrated, client cluster may contain more than two clients. Server node cluster 206, may be a server cluster, such as server cluster 106 (FIG. 1) or it may be any other type of clustered environment such as, but not limited to, a virtual network. Resource 226 may be stored in a datastore 228 that is part of the clustered environment. Although not shown, in alternate embodiments, the datastore 228 may not be part of the clustered environment, but may be connected to the clustered environment over a network. Examples of such a network include, but are not limited to, the Internet, a WAN, a LAN, or any other type of network known to the art. In still further embodiments, datastore may be part of a node (e.g., a device) that is a part of cluster 206.

[0029] Server node cluster 206 may include one or more nodes, such as Node 1 208 and Node 2 216. Although only two node clusters are illustrated in FIG. 2, any number of node clusters may be included in clustered environment 206. In embodiments, node clusters 208 and 216 are capable of receiving a request for, performing an operation on, and/or granting access to resource 226. In embodiments, resource 226 may be a file, an object, an application, data, or any other type of resource stored on or accessible by node in node cluster 206 or by a standalone server.

[0030] In embodiments, a client sends an initial request 222 to clustered environment 206. As illustrated in FIG. 2, the initial request 222 may be sent by client 202 and received by Node 1 208. However, in alternate embodiments, the initial request 222 may be sent by client or any other client node in client cluster 201 and received by Node 2 216 or any other node in server cluster 206. An example requests include, but are not limited to, requests to create, open, or otherwise access a file. Request 222 may be transmitted from the client to the Node cluster via a network such as, but not limited to, the Internet, a WAN, a LAN, or any other type of network known to the art. Initial request 222 may include a request to access a resource, such as resource 226. In embodiments, request 222 may also include an application instance identifier that identifies the requestor that client 202 is making the request on behalf of. In embodiments, initial request 222 may consist of one or more messages. For example, request 222 may be a single message containing both the request and an application instance identifier. In another embodiment, request 222 may be multiple messages that include one or more requests as well as one or more application instance identifiers. In embodiments, client 202 may include an App Instance Cache 214 that is used to store and/or generate one or more application instance identifiers that may be transmitted with request 222.

[0031] As shown in FIG. 2, Node 1 208 may receive request 222 and an application instance identifier from client 202. If the requested resource 226 is available, e.g., not fenced or locked by another client or application, Node 1 may grant the client's (e.g., client 202) request to access resource 226 on behalf of a requestor that is executing on the client. Upon granting access to resource 226, filter driver 210 may allocate or otherwise create an association between client 202 and the resource 226 by storing the application instance identifier it received from client 202. In embodiments, the association may be stored in as an object in app instance cache 212 that is a part of Node 1. Although the illustrated embodiment shows app instance cache 212 as a part of Node 1 208, in embodiments, app instance cache 212 may be stored elsewhere as a part of node cluster 206. One of skill in the art will appreciate Node cluster 206 may include one or more app instance caches, such as app instance cache 220 on Node 2 216. In embodiments when more than one app instance cache is present, the data stored in the multiple app instance caches may be replicated across all app instance caches, or each app instance cache may store separate data.

[0032] In one embodiment, the application instance identifier received from client an application instance identifier that identifies a requestor (e.g., an application or process) may be stored in a _NETWORK_APP_INSTANCE_ECP_CONTEXT structure. The _NETWORK_APP_INSTANCE_ECP_CONTEXT structure may be defined as follows:

```
            typdef struct _NETWORK_APP_INSTANCE_ECP_CONTEXT {
            USHORT Size;
            USHORT Reserved;
            GUID AppInstanceID;
            } _NETWORK_APP_INSTANCE_ECP_CONTEXT,
*PNETWORK_APP_INSTANCE_ECP_CONTEXT;
```

[0033] In such embodiments, the variable size may store information related to the size of the structure and the variable AppInstanceID may be a unique application instance identifier for a failover cluster client application, such as a requestor executing on client 202. In embodiments, the _NETWORK_APP_INSTANCE_ECP_CONTEXT, or another object or variable containing the requestor's application instance identifier may be stored in the globally unique identifier (GUID) cache 214. In embodiments the _NETWORK_APP_INSTANCE_ECP_CONTEXT structure may be sent from a client to a server in association with a request to access a resource (e.g., a create or open request)In one embodiment, the requestor's application instance identifier may be stored in the GUID cache of the client node that the requestor is executing on in the clustered client environment 201. In another embodiment, although not shown in FIG. 2, the client node cluster 201 may have a central repository that stores application instance identifiers. In such an embodiment, multiple client nodes in the client node cluster 201 may access the centralized repository. In yet another embodiment, application instance identifiers may be stored across multiple GUID caches (e.g., GUID cache 214 and GUID cache 216). In such embodiments, the client node cluster 201 may employ a replication algorithm to ensure that the multiple GUID caches contain the same application instance identifiers.

[0034] As previously described, the application instance identifier may be associated with resource 226 while client 202 accesses resource 226 on behalf of a requestor. A server node 206 may store such an association in one or more app instance caches that are part of server node cluster 206, such as app instance caches 212 and 220. In one embodiment, the application instance identifier may be associated with the resource by adding it to an Extra Create Parameter (ECP) list for the resource 226. The ECP list may be stored in an app instance cache that is part of the server node cluster 206, such as app instance caches 212 and 220. In embodiments, when an ECP is received by a server, the server extracts an application instance identifier from the ECP and adds it to a cache to be associated with a resource, resource handle, etc. As described with respect to storing application instance identifiers in client cluster 201, the application instance identifiers associated with a node may be stored in an individual app instance cache on node in server node cluster 206, in a central repository in server cluster 206, or replicated across multiple app instance caches on multiple nodes in server node cluster 206.

[0035] In embodiments, resource 226 is fenced or locked while a requestor executing on client 202 has access to resource 222, thereby preventing other client or applications from accessing resource 226 and avoiding any potential conflicts. In embodiments, before the requestor completes its operation on resource 226, client 202 experiences an error that causes it to lose connection with the resource. For example, the client may crash, be taken offline, or lose its network connection to server node 208. In such instances, resource 226 may still be in a fenced or locked state because the requestor did not release a lock on the resource, thereby preventing other clients from accessing resource 226.

[0036] When the error occurs to client 202, the requestor may utilize a client failover mechanism 232 to migrate to a new client node (e.g., client 204) in the client cluster 201. One of skill in the art will appreciate that any type of failover mechanism may be employed at client failover 232. In embodiments, the failover mechanism 232 may also include the migration of the requestor's application instance identifier which may have been stored in GUID cache 214 on the now failed client 202. Upon completing the migration, the requestor may attempt to regain access to the resource 202. In embodiments, client 216 may send a second request 224 to Node 1 to request access to resource 226 on behalf of the requestor. However without the continuous access embodiments disclosed herein, when Node 1 208 receives a request to access the resource 226 on behalf of client 204 (the sender of second request 224), it may deny the request because resource 226 is still in a fenced or locked state from the previous access that client 202 made on behalf of the resource. Without the embodiments disclosed herein, Node 1 208 would recognize that the second request to access resource 226 was from a different location (e.g., client 204). Node 1 208 would not be able to determine that the request is for the same requestor that holds the lock on resource 226, and would therefore determine that granting the request would result in a conflict. However, if the same requestor is attempting to access resource 224, there is no issue of conflict and forcing the client to wait for the resource to be freed by the system may result in undue delays.

[0037] The application instance identifier may be used to solve this problem. In embodiments, the second request 224 may also include the application instance identifier from identifying the requestor that migrated to client 204 during the failover shown at 232. In embodiments, the requestor's application instance identifier may be present in the GUID cache 228 of client 204 prior to the migration of the requestor during the client failover 232. For example, a replication mechanism may have been employed to replicate the requestor's application instance identifier across the nodes in client cluster 201. In another embodiment, the requestor 203 may store its application instance identifier. In yet another embodiment, the requestor's 203 application instance identifier may be migrated during client failover 232.

[0038] As described with respect to request 222, the application instance identifier may be transmitted in the same message as the second request 224 or the second request 224 may be composed of a number of different messages. When the second request is received at the node cluster 206, or an individual node in the cluster, such as Node 1 208, and the receiving server determines that the resource is fenced or locked, a determination is made as to whether the application instance identifier in the second request 224 is the same as the application instance identifier associated with resource 226. In embodiments, Node 2 216 will compare the application instance identifier received with the second request 222 with the application instance identifier that is associated with resource 226. The application identifier associated with resource 220 may be stored in the app instance cache 212 of Node 1 212. In embodiments where multiple app instance caches exist in node cluster 206, the determination may check more than one application instance caches in the node cluster 206. In such embodiments, if a matching application instance identifier is not located in app instance cache 212, Node 1 216 may send a request to Node 2 212 to determine if a matching application instance identifier is located in app instance cache 220.

[0039] In one embodiment, if the application instance identifier received in second request 224 does not match the application instance identifier associated with resource 226 (which may be stored in application instance cache 212 and/or 220), the second request 224 may not be granted until resource 226 is free. However, if a match is found, the receiving server (e.g., Node 1 208) and/or the server node cluster 206 perform actions to grant access to resource 226 without causing undue delay to client 204 and requestor 203. In such instances, node cluster 206 may invalidate the resource 226, thereby removing resource 226 from a fenced or locked state. In embodiments, invaliding a previous access may comprise any action that brings a resource out of a fenced or locked state. One non-limiting example is closing an opened file (e.g., if resource 226 is a file). Once the previous access is invalidated, the second request 224 to access resource 226 may be granted, thereby providing continuous access to the requestor 203.

[0040] In one embodiment, the node receiving the second request 224, such as Node 1 208 in FIG. 2, may perform the required actions to invalidate the previous access of resource 226 if a different node (e.g., Node 2 216) has access and/or permission to invalidate the previous access. However, in some instances, the node receiving the request may not have access or permission to invalidate the previous access. For example, such an instance may occur if the original request 222 was made to Node 2 216, in which case Node 2 216 may have control over the resource. In such instances, the node receiving the second request 224 may send a request to the controlling node to invalidate the previous access. Once the controlling node has invalidated the previous access, the node receiving the second request 224 may grant the second request 224. In other embodiments, the node receiving the second request 224 may send a request to a different node to grant client 204 and/or requestor 203 (now residing on client 204) access to resource 226.

[0041] The described process avoids undue delay in granting a second request 224 to access a resource 226 from a resource 203 that previously accessed and still holds a lock on resource 226 through the use of application instance identifiers. Furthermore, the application instance identifiers provide the benefit of ensuring that any request granted does not create a conflict on resource 226. For example, if the request was received from a different application, the request will include an application instance identifier that is different from the application instance identifier associated with the resource which would result in the request being denied. Because application instance identifiers are globally unique identifiers, the application instance identifier for different applications will not be the same.

[0042] FIG. 3 is an embodiment of a method 300 that a requestor may employ to gain continuous access to a resource in a client clustered environment. For example, a requestor may be a client, such as client 202 (FIG. 2), that employs the method 300 to access a resource (e.g., resource 226). In embodiments, the resource may reside on a remote machine, such as a server. The server may be a standalone server or part of a clustered environment, such as sever cluster 206 (FIG. 2). Flow begins at operation 302 where a request for a resource is sent to a server. In embodiments, the request may be to access a resource. In embodiments, accessing a resource may comprise opening a file, creating a file, or otherwise accessing or performing an operation on a resource that may be remote to a client. In embodiments, a requestor may operate in a client clustered environment. In such embodiments, the request sent at operation 302 may be sent from a first client in the client clustered environment.

[0043] Flow continues to operation 304 where an application instance identifier is sent, for example, to a server (e.g., a standalone server or a node in a clustered environment). In one embodiment, the first client that sent the request may also send the application instance identifier on behalf of the requestor. As earlier described, an application instance identifier is a GUID identifying the requestor (e.g., an application, client, or a child process of an application requesting access to a resource). In one embodiment, the application instance identifier may be sent in a message transmitted via a network. The application instance identifier may be transmitted in the same message containing the request in operation 302 or it may be transmitted in a different message. In such embodiments, an object containing the application instance identifier, such as but not limited to the _NETWORK_APP_INSTANCE_ECP_CONTEXT described with respect to FIG. 2, may be sent at operation 302.

[0044] In one embodiment, an interface may be used to send the application instance identifier at operation 304. The interface may be a kernel level interface located on a client or available to a client operating in a client clustered environment. In embodiments, the kernel level interface may be used by the requestor and/or client to send an application instance identifier to a server. The following is a non-limiting example of a kernel level interface that may be employed at operation 304 to send an application instance identifier:

```
#if (NTDDI_VERSION >= NTDDI_WIN8)
//
// ECP context for an application to provide its instance ID.
//

typedef struct _NETWORK_APP_INSTANCE_ECP_CONTEXT {
    //
    // This must be set to the size of this structure.
    //

    USHORT Size;

    //
    // This must be set to zero.
    //

    USHORT Reserved;

      //
    // The caller places a GUID that should always be unique for a single instance of
    // the application.
  //

    GUID AppInstanceID;

} NETWORK_APP_INSTANCE_ECP_CONTEXT,
*PNETWORK_APP_INSTANCE_ECP_CONTEXT;

//
// The GUID used for the APP_INSTANCE_ECP_CONTEXT structure.
//
// {6AA6BC45-A7EF-4af7-9008-FA462E144D74}
//
DEFINE_GUID(GUID_ECP_NETWORK_APP_INSTANCE, 0x6aa6bc45, 0xa7ef,
0x4af7, 0x90, 0x8, 0xfa, 0x46, 0x2e, 0x14, 0x4d, 0x74);

#endif // NTDDI_Version >= NTDDI_WIN8
```

[0045] Although a specific kernel level interface is provided, one of skill in the art will appreciate that other kernel level interfaces may be employed at operation 304 to send the application instance identifier.

[0046] In another embodiment, an application user interface (API) may be employed at operation 304 to send an application instance identifier. In such embodiment, the requestor and/or client may send an application instance identifier by making a call to the API. The API may be hosted on the client performing the operation 304 (e.g., the first client in a server cluster) or the API may be hosted on another device and accessed by the requestor or another application or process. The following is a non-limiting example of an API that may be employed at operation 304 to send an application instance identifier:

```
NTSTATUS RegisterAppInstance (
_in PGUID   AppInstance
);
```

[0047] Although a specific API is provided, one of skill in the art will appreciate that other API's may be employed at operation 304. Furthermore, although operation 304 is illustrated as a discrete operation, one of skill in the art will appreciate that sending the application instance identifier may be performed simultaneously with sending the request at operation 302.

[0048] When the requested resource is not locked, the request sent at operation 302 is granted and flow continues to operation 306 where the resource is accessed. As previously described, the server or device controlling the resource may place the resource in a fenced or locked state while the requestor accesses the resource at operation 306. At some point while accessing the resource, an error occurs, such as the errors described with reference to FIG. 2 which causes the client to fail or otherwise lose connection to the resource. The error may cause the client (e.g., the first client in the server cluster) to lose access to the resource before the requestor completes its use of the resource. Under such circumstances, the resource may not be released from its fenced or locked state.

[0049] Flow continues to operation 308, where a failover operation is performed. In embodiments, the failover operation may comprise cloning the requestor and its state to a different client in the client node cluster (e.g., a second client). In embodiments, the requestor's state may be cloned on the second on the second and the requestor may be executed on the second client in a manner such that it can resume execution from the point where the first client failed. In another embodiment, the requestor may be in communication with the first client (rather than executing on) at the time of the first clients failover. In such embodiments, the failover operation may comprise the requestor establishing communications with a second client in the client cluster.

[0050] In embodiments, state information, including but not limited to the requestors application instance identifier, may be transferred from the first client to the second client. In one embodiment, the first client may send a message including the requestor's application instance identifier and/or the requestor's state information. The application instance identifier and/or state may be sent during the failover process or, in embodiments, may be sent before the first client fails, such as, during a replication process that clone's information across the clients in a client clustered environment. In another embodiment, the requestor's application instance identifier and/or state information may be stored in a central location or repository in the client clustered network. In such embodiments, the failover process may provide the second client with the location of the requestor's application instance identifier and/or state information. In yet another embodiment, the requestor may maintain its application instance identifier. In such embodiments, the client failover operation may comprise relocating to or otherwise establishing a connection between the requestor and a second client.

[0051] In embodiments, after the client failover operation flow continues to operation 310. At operation 310 a second request for the same resource is sent to the clustered environment. In embodiments, the second request is sent by the second client in the client cluster on behalf of the requestor. The second request may be sent using the same manner as described with respect to the first resource at operation 302. In order to maintain continuous access to the resource and avoid undue delay, flow continues to operation 312 where the application instance identifier is again sent to the clustered environment. The application instance identifier may be sent at operation 308 according to one of the embodiments described with respect to operation 304. In embodiments, because the a different client (e.g., the second client) is sending the second request, the server receiving the request may not be able to identifier the second request as belonging to the same requestor that holds a lock on the resource (e.g., because the request is made from a different machine, a different address etc.) However, by sending the application instance identifiers at operations 304 and 308, the server will be able to identify the requests as belonging to the same requestor, and will grant continuous access to the resource as previously described with respect to FIGs. 1 and 2. Flow continues to operation 314 and the requestor resumes access to the resource. In embodiments, the second client may receive a response to the second request from the server indicating that the server granted the second request. In embodiments, upon receiving the indication, the second client may access the resource on behalf of the requestor.

[0052] FIG. 4 is an embodiment of a method 400 performed by a node in a server clustered environment to provide continuous access to a resource. Embodiments of the method 400 may be performed by a node such as Node 1 208 (FIG. 2) in a clustered environment, such as node cluster 206 (FIG. 2). In embodiments, method 400 may be performed by a node that has access to a resource. Flow begins at operation 402 where the node receives a request for a resource. In embodiments, a resource may be a file, an object, a method, data, or any other type of resource that is under the control of and/or may be accessed by the node performing operation 400. An application instance identifier may be received with the request at operation 402.

[0053] Flow continues to decision operation 404 where a determination is made as to whether the resource is in a fenced or locked state. One of skill in the art will appreciate that any manner of determining whether a resource is fenced or locked may be employed at operation 404. If the resource is not in a fenced or locked state, flow branches NO to operation 412 where the request for the resource is granted. In embodiments, granting the request may comprise allowing the requestor access to the resource, performing an operation on the resource on behalf of the requestor, or permitting any kind of access or modification to the resource. For example, granting the request in operation 412 may include opening a file or creating a file.

[0054] If the resource is in a fenced or locked state, flow branches YES from operation 404 to decision operation 406. At decision operation 406, the application instance identifier received with the request at operation 402 is compared to an application instance identifier that is associated with the resource. For example, as describe with respect to FIG. 2, a node may associate an application instance identifier with a resource when a client or application accesses a resource. As described earlier, the association of the application instance identifier of a requestor accessing a resource may be stored on a node, for example, in an app instance cache, as described in various embodiments discussed in FIG. 2. In embodiments, the application instance identifier that is provided in an ECP sent with a request for a resource, for example, in a _NETWORK_APP_INSTANCE_ECP_ CONTEXT structure, may be added to an ECP list associated with the resource.

[0055] In one embodiment, the association of the application instance resource may reside locally on the node performing method 400. In such instances, the comparison may be made at a local app instance cache resident on the server. However, as discussed with respect to FIG. 2, a clustered environment may contain a number of app instance caches distributed across different nodes. Furthermore, the different application instance caches may each store separate and/or different data. The application identifier associated with the fenced or locked resource may be stored on a different node in the clustered environment. In such instances, operation 406 may include sending a request to a different node to perform the comparison at operation 406. The request may include the application instance identifier received at operation 402.

[0056] If the received application instance identifier is not the same as the application instance identifier associated with the resource, flow branches NO to operation 410. At operation 410 the request to access the resource received at operation 402 is denied. In embodiments, the request may be denied in order to avoid a resource conflict. Because the received application identifier is not the same as the associated application instance identifier, the request to access the resource received at operation 402 is from a different requestor or application. Granting a request to the different client or application, as may be in this case, may cause a conflict situation that will interfere with the application currently accessing the resource. For example, the different application may modify the resource in a manner that modifies or otherwise interferes with the operations performed on the resource by the requestor that currently holds a lock on the resource.

[0057] However, receiving an application identifier with the request 402 that is the same as the application identifier associated with the fenced or locked resources indicates that an error may have occurred that caused the requestor that was accessing the resource to lose its access to the resource without properly releasing the resource. For example, the requestor may operate in a client node cluster. The particular client the requestor was operating on may have lost connection to the server or otherwise failed before the requestor completed its operations upon the resource. In order to provide continuous access the resource, that is, to allow the requestor to regain access to the resource without experiencing undue or unacceptable delay, flow branches YES to operation 408.

[0058] At operation 408, the resource is invalidated. As earlier described herein, invalidating the resource may include changing the fenced state of the resource or otherwise removing a lock on the resource. For example, if the resource is a file, invalidating the resource may include closing the file. One of skill in the art will appreciate that any method of releasing a fenced or locked resource may be employed at operation 408.

[0059] Referring back to FIG. 2, in embodiments, access to a resource may be under control of a node in the clustered environment different than the node that receives the request for access to the resource at operation 402. For example, a handle to the resource may reside on a different node in the clustered environment. In such embodiments, invalidating the resource may include sending a request to the node controlling access to the resource to invalidate the resource. In response to sending the request, the remote node may invalidate the resource.

[0060] After the resource is invalidated, flow continues to operation 412 where the request to access the resource is granted. Granting the request may comprise allowing the requestor access to the resource, performing an operation on the resource on behalf of the requestor, or permitting any kind of access or modification to the resource. For example, granting the request in operation 412 may include opening a file or creating a file. Granting such access may be performed by the node receiving the request at operation 402, or by another node in the clustered environment.

[0061] Methods 300 and 400 are merely some examples of operational flows that may be performed in accordance with embodiments. Embodiments are not limited to the specific description provided above with respect to FIGS. 3-6 and may include additional operations. Further, operational steps depicted may be combined into other steps and/or rearranged. Further, fewer or additional steps may be used, employed with the methods described in FIGs. 3-4.

[0062] FIG. 5 illustrates a general computer system 500, which can be used to implement the embodiments described herein. The computer system 500 is only one example of a computing environment and is not intended to suggest any limitation as to the scope of use or functionality of the computer and network architectures. Neither should the computer system 500 be interpreted as having any dependency or requirement relating to any one or combination of components illustrated in the example computer system 500. In embodiments, system 500 may be used as the clients and/or servers described above with respect to FIGs. 1 and 2.

[0063] In its most basic configuration, system 500 typically includes at least one processing unit 502 and memory 504. Depending on the exact configuration and type of computing device, memory 504 may be volatile (such as RAM), non-volatile (such as ROM, flash memory, etc.) or some combination. This most basic configuration is illustrated in FIG. 5 by dashed line 506. System memory 504 stores instructions 520 such as the instructions to perform the continuous availability methods disclosed herein and data 522 such as application instance identifiers that may be stored in a file storage system with storage such as storage 508.

[0064] The term computer readable media as used herein may include computer storage media. Computer storage media may include volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information, such as computer readable instructions, data structures, program modules, or other data. System memory 504, removable storage, and non-removable storage 508 are all computer storage media examples (e.g. memory storage). Computer storage media may include, but is not limited to, RAM, ROM, electrically erasable read-only memory (EEPROM), flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store information and which can be accessed by computing device 500. Any such computer storage media may be part of device 500. Computing device 500 may also have input device(s) 514 such as a keyboard, a mouse, a pen, a sound input device, a touch input device, etc. Output device(s) 516 such as a display, speakers, a printer, etc. may also be included. The aforementioned devices are examples and others may be used.

[0065] The term computer readable media as used herein may also include communication media. Communication media may be embodied by computer readable instructions, data structures, program modules, or other data in a modulated data signal, such as a carrier wave or other transport mechanism, and includes any information delivery media. The term "modulated data signal" may describe a signal that has one or more characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media may include wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, radio frequency (RF), infrared, and other wireless media.

[0066] Embodiments of the invention may be practiced via a system-on-a-chip (SOC) where each or many of the components illustrated in Figure 5 may be integrated onto a single integrated circuit. Such an SOC device may include one or more processing units, graphics units, communications units, system virtualization units and various application functionality all of which are integrated (or "burned") onto the chip substrate as a single integrated circuit. When operating via an SOC, the functionality, described herein, with respect to providing continuous access to a resource may operate via application-specific logic integrated with other components of the computing device/system 500 on the single integrated circuit (chip).

[0067] Reference has been made throughout this specification to "one embodiment" or "an embodiment," meaning that a particular described feature, structure, or characteristic is included in at least one embodiment. Thus, usage of such phrases may refer to more than just one embodiment. Furthermore, the described features, structures, or characteristics may be combined in any suitable manner in one or more embodiments.

[0068] One skilled in the relevant art may recognize, however, that the embodiments may be practiced without one or more of the specific details, or with other methods, resources, materials, etc. In other instances, well known structures, resources, or operations have not been shown or described in detail merely to avoid obscuring aspects of the embodiments.

[0069] While example embodiments and applications have been illustrated and described, it is to be understood that the embodiments are not limited to the precise configuration and resources described above. Various modifications, changes, and variations apparent to those skilled in the art may be made in the arrangement, operation, and details of the methods and systems disclosed herein without departing from the scope of the claimed embodiments.

Claims

1. A method of providing continuous access to a resource, the method comprising:

receiving a first request to access a resource from a requestor, wherein the request is received from a first client;

associating a first application instance identifier with the first resource;

allowing the first request access to the resource;

receiving a second request for the resource from the requestor after recovering from a failure, wherein the second request is received from a second client different from the first client;

receiving a second application identifier associated with the second request;

determining if the first application identifier and the second application identifier are the same; and

when the first and second application identifiers are the same, performing the steps comprising:

invalidating the first request; and

granting the second request to access the resource.

2. The method of claim 1, wherein the first application identifier is associated with an application instance of an open request.

3. The method of claim 1, wherein the first application identifier is associated with a process.

4. The method of claim 1, wherein the first application identifier is associated with at least one child process of an application.

5. The method of claim 1, wherein associating the first application instance identifier comprises receiving the first application instance identifier in a NETWORK_APP_INSTANCE_ECP_CONTEXT structure.

6. A method for providing clustered client failover, the method comprising:

receiving, at a second client, an application instance identifier for a requestor, wherein the requestor previously accessed a resource using a first client;

sending, from the second client, a second request to access the resource on behalf of the requestor;

sending, from the second client, the application instance identifier for the requestor;

receiving an indication that a server granted the second request; and

accessing, by the second client, the resource on behalf of the client.

7. The method of claim 6, wherein the server granting the second request previously granted a first request to access the resource from the first client on behalf of the requestor.

8. The method of claim 6, wherein the second client sends the second request in response to a client failover.

9. A system for facilitating client failover in a clustered environment, the system comprising:

at least one server comprising:

at least one processor configured to execute computer executable instructions;

at least one computer readable storage media storing the computer executable instructions that when executed by the at least one processor provide:

receiving a first request to access a resource from a first client on behalf of a requestor;

associating a first application instance identifier with the first resource;

allowing the requestor access to the resource;

receiving a second request for the resource from a second client, wherein the second client is different from the first client;

receiving a second application identifier associated with the second request;

determining if the first application identifier and the second application identifier are the same;

when the first and second application identifiers are the same, performing the steps comprising:

invalidating the first request; and

granting the second request to access the resource.

10. The system of claim 18, wherein the system further comprises:

the first client, comprising:

at least one processor configured to execute computer executable instructions;

at least one computer readable storage media storing the computer executable instructions that when executed by the at least one processor:

sending the first request;

sending the application instance identifier to a second client.

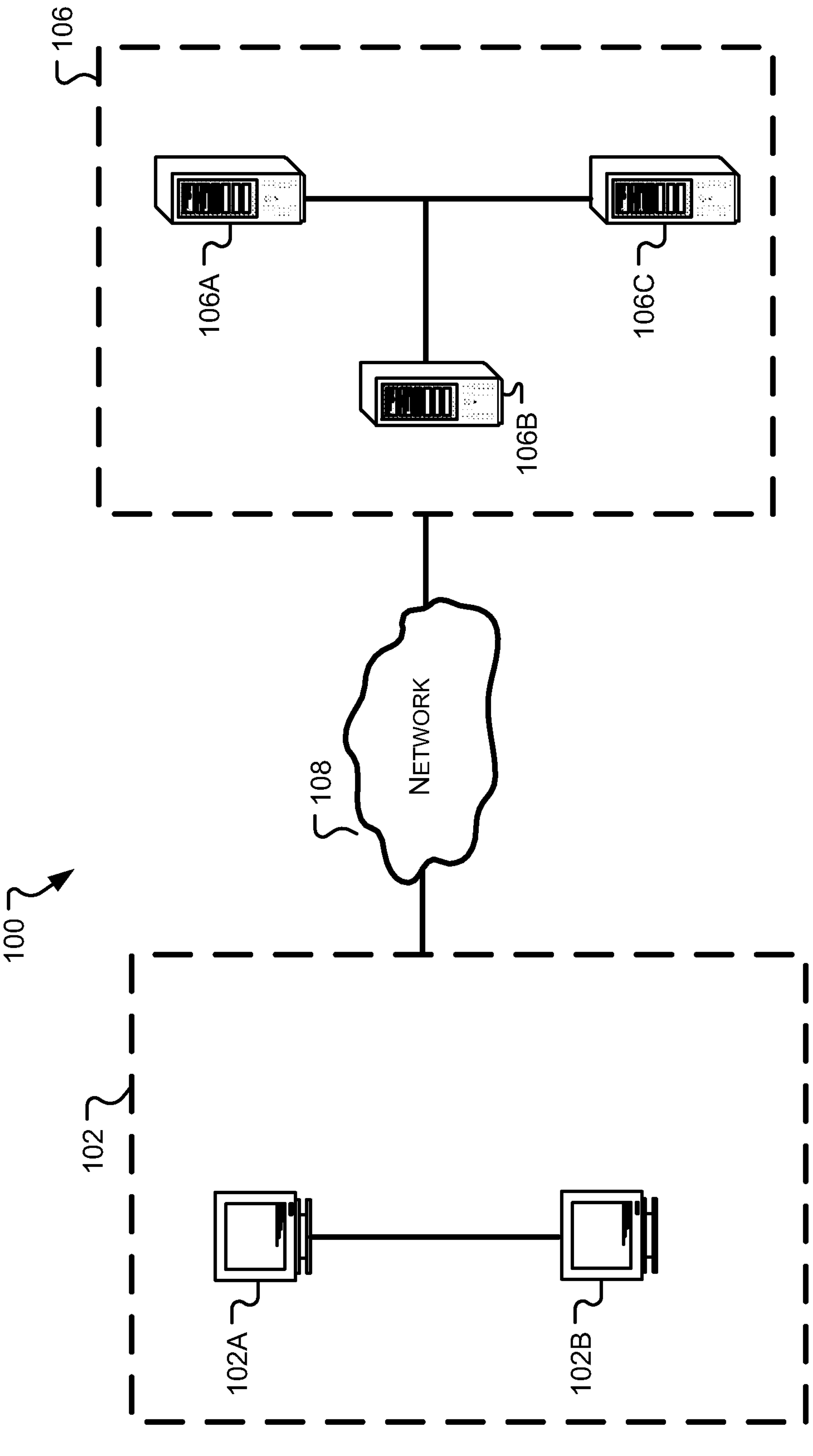

FIG. 1

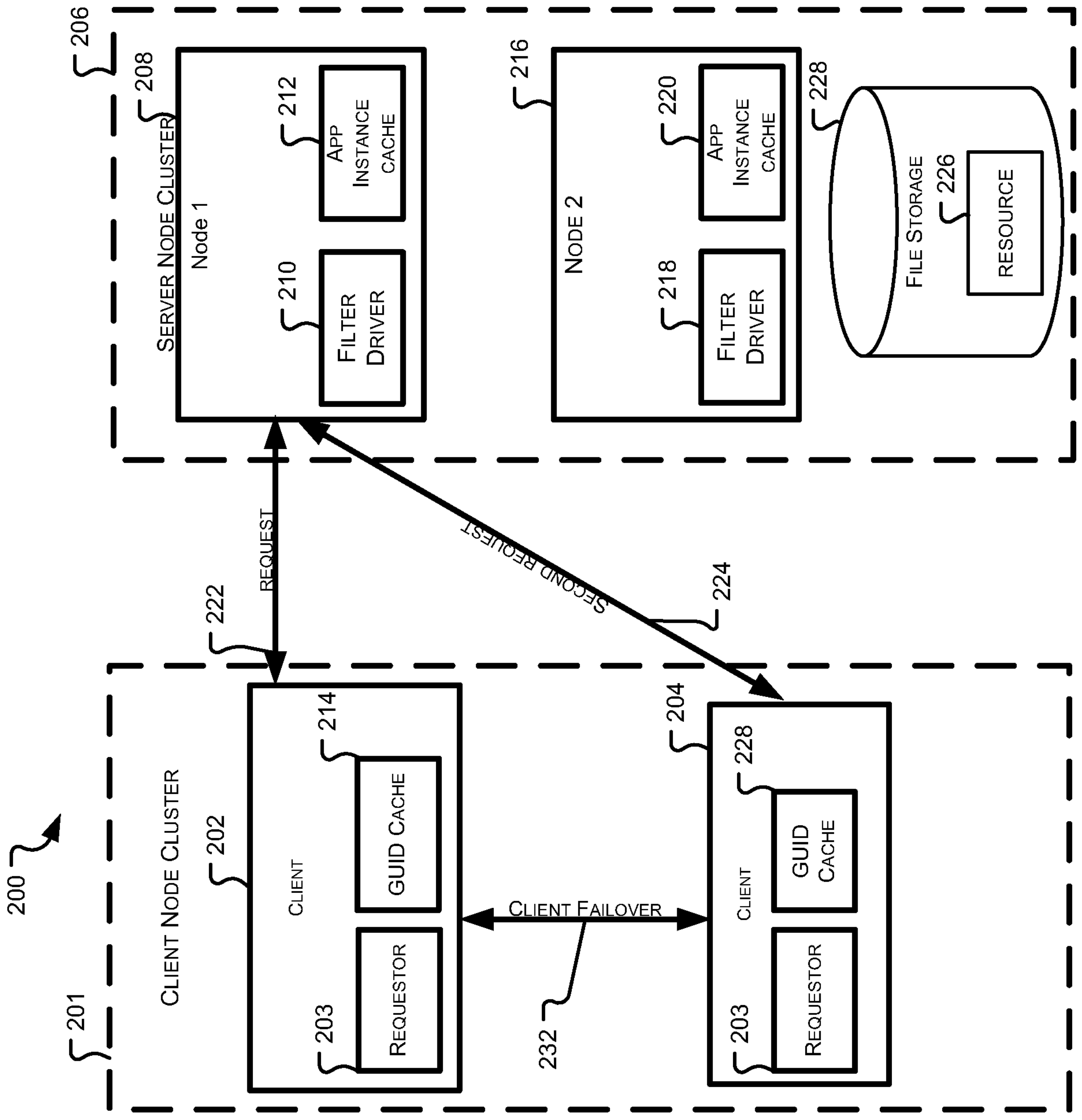
200
201
Client Node Cluster
202
Client
GUID Cache
203
Requestor
214
204
Client
228
GUID Cache
203
Requestor
Client Failover
232
222
Request
224
Second Request
206
Server Node Cluster
208
Node 1
210
Filter Driver
212
App Instance cache
216
Node 2
218
Filter Driver
220
App Instance cache
228
File Storage
226
Resource

FIG. 2

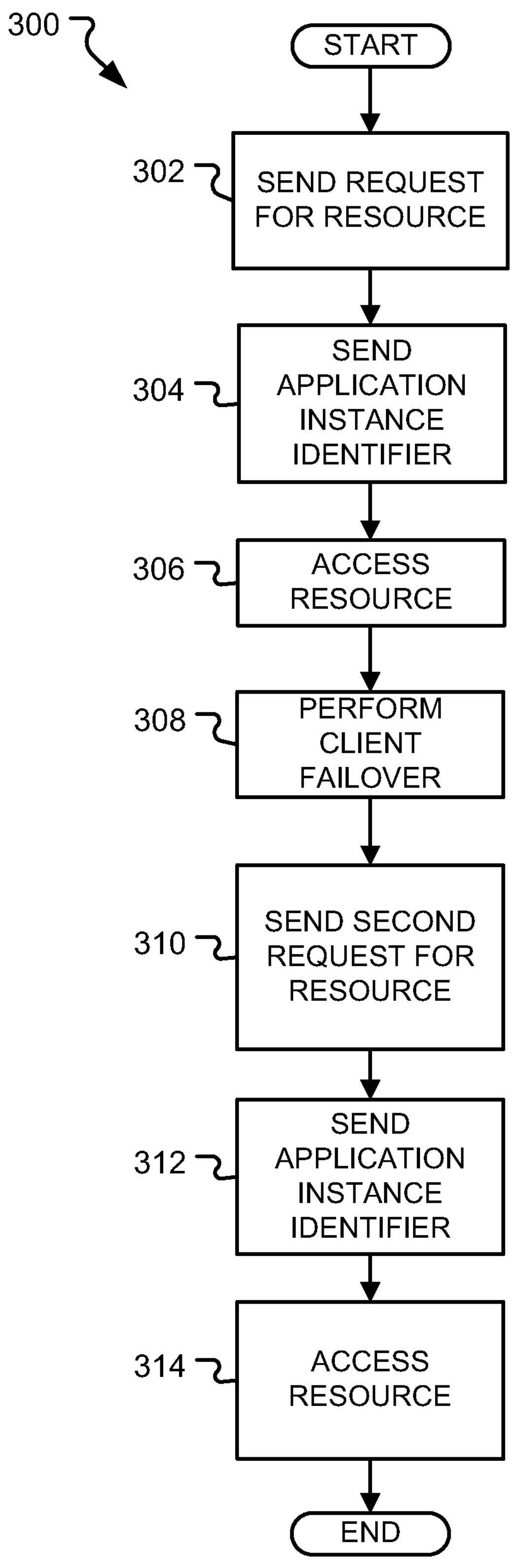
300
START
302
SEND REQUEST FOR RESOURCE
304
SEND APPLICATION INSTANCE IDENTIFIER
306
ACCESS RESOURCE
308
PERFORM CLIENT FAILOVER
310
SEND SECOND REQUEST FOR RESOURCE
312
SEND APPLICATION INSTANCE IDENTIFIER
314
ACCESS RESOURCE
END

FIG. 3

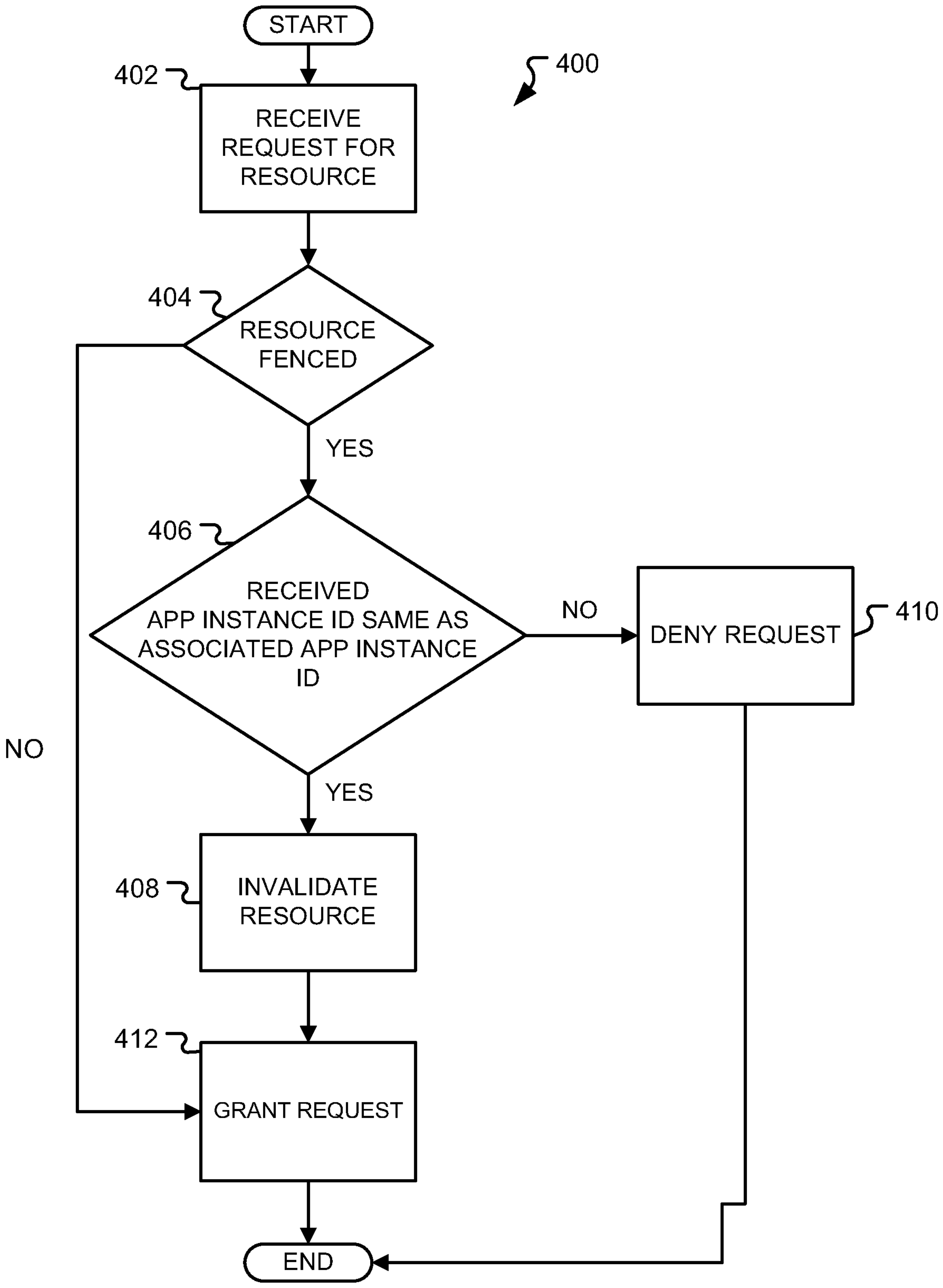
START
402
RECEIVE REQUEST FOR RESOURCE
400
404
RESOURCE FENCED
YES
406
RECEIVED APP INSTANCE ID SAME AS ASSOCIATED APP INSTANCE ID
NO
DENY REQUEST
410
NO
YES
408
INVALIDATE RESOURCE
412
GRANT REQUEST
END

FIG. 4

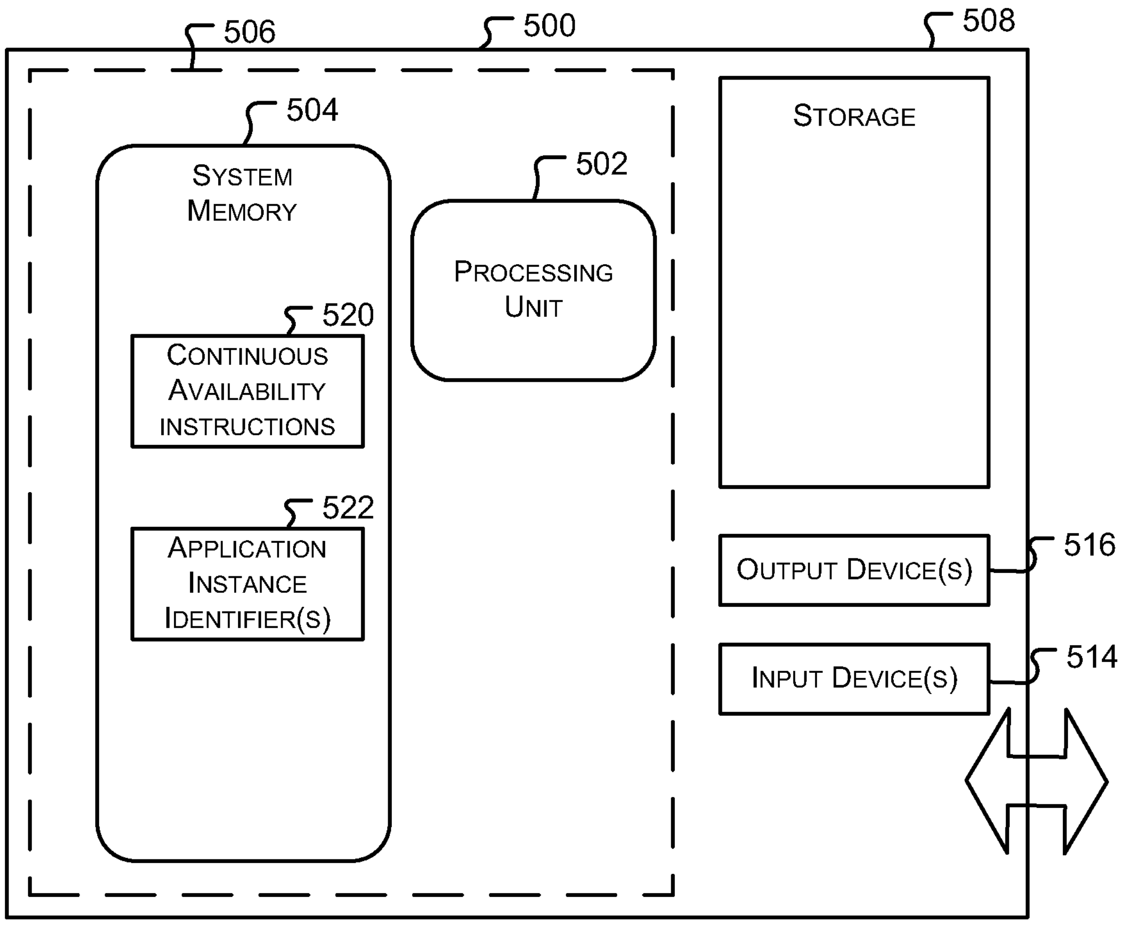
506
500
508
504
System Memory
502
Processing Unit
520
Continuous Availability instructions
522
Application Instance Identifier(s)
Storage
Output Device(s)
516
Input Device(s)
514

FIG. 5