(12) STANDARD PATENT (11) Application No. **AU 2002313826 B2**
(19) AUSTRALIAN PATENT OFFICE

(54) Title
Cluster caching with concurrency checking

(51) International Patent Classification(s)
G06F 12/00 (2006.01) ***G06Q 20/00*** (2006.01)
G06F 17/30 (2006.01) ***H04L 29/08*** (2006.01)

(21) Application No: **2002313826** (22) Date of Filing: **2002.08.28**

(87) WIPO No: **WO03/021484**

(30) Priority Data

(31) Number	(32) Date	(33) Country
60/316,187	**2001.08.30**	**US**
10/211,713	**2002.08.02**	**US**
10/211,712	**2002.08.02**	**US**
60/316,190	**2001.08.30**	**US**

(43) Publication Date: **2003.03.18**
(43) Publication Journal Date: **2003.06.05**
(44) Accepted Journal Date: **2008.01.24**

(71) Applicant(s)
Bea Systems, Inc.

(72) Inventor(s)
Messinger, Adam;Jacobs, Dean Bernard;White, Seth;Woolen, Rob

(74) Agent / Attorney
Davies Collison Cave, 1 Nicholson Street, Melbourne, VIC, 3000

(56) Related Art
US 5805798
US 5452445
US 5926816

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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
13 March 2003 (13.03.2003)

PCT

(10) International Publication Number
WO 03/021484 A3

(51) International Patent Classification[7]: G06F 17/30

(21) International Application Number: PCT/US02/27315

(22) International Filing Date: 28 August 2002 (28.08.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/316,187	30 August 2001 (30.08.2001)	US
60/316,190	30 August 2001 (30.08.2001)	US
10/211,713	2 August 2002 (02.08.2002)	US
10/211,712	2 August 2002 (02.08.2002)	US

(71) Applicant: **BEA SYSTEMS, INC** [US/US]; 2315 North First Street, San Jose, California 95131 (US).

(72) Inventors: **JACOBS, Dean Bernard**; 1747 Madera Street, Berkeley, California 94707 (US). **WOOLEN, Rob**; 2531 14th Avenue, San Francisco, CA 94127 (US). **MESSINGER, Adam**; 317 29th Street-Apartment 306, San Francisco, California 94131 (US). **WHITE, Seth**; 1045 Rivera Street-Apartment B, Sans Francisco, California 94116 (US).

(74) Agent: **MEYER, Sheldon, R.**; Fliesler Dubb Meyer and Lovejoy LLP, Four Embarcadero center-Fourth Floor, San Francisco, california 94111-4156 (US).

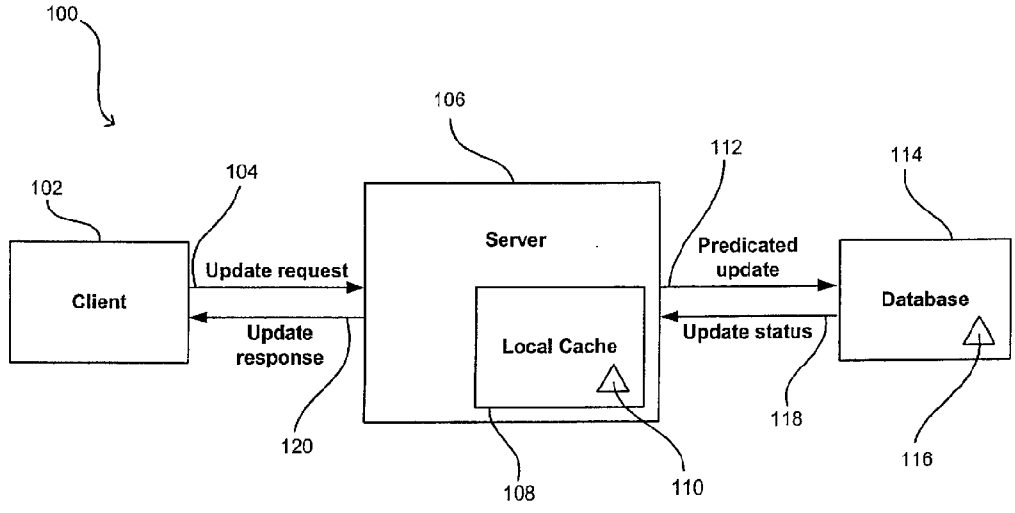
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
with international search report

[Continued on next page]

(54) Title: CLUSTER CACHING WITH CONCURRENCY CHECKING



(57) Abstract: Concurrency can be maintained in cluster caching when processing an update request on network server that is storing a local copy of data item. the request can be processed using the local copy of the data item. A predicated update request can be sent to a network database storing the data item, wherein the database can commit the update if the local copy is current with the data item. If the local copy is not current, the network server can request a new copy, process the request using the current copy, and try another predicated request. the process can continue until the update is committed to the database or aborted. Once committed, any other servers in the cluster can be notified that the data item has been updated. Those other servers can drop any local copy of the data item and can request an updated copy of the data item.

WO 03/021484 A3

(88) Date of publication of the international search report:
24 July 2003

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

COPYRIGHT NOTICE

[0001] A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document of the patent disclosure, as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

CLAIM OF PRIORITY

[0002] This application claims priority to the following applications which, are incorporated herein by reference: [0003] U. S. Provisional Patent Application entitled "CLUSTER CACHING WITH CONCURRENCY CHECKING", Application No. 60/316, 187, filed August 30, 2001.

[0004] U. S. Patent Application entitled "CLUSTER CACHING WITH CONCURRENCY CHECKING", Application No. 10/211,713, filed August 2, 2002.

[0005] U. S. Provisional Patent Application entitled "METHOD FOR MAINTAINING ACCOUNT CONSISTENCY," Application No. 60/316,190, filed August 30, 2001.

[0006] U. S. Patent Application entitled "METHOD FOR MAINTAINING ACCOUNT CONSISTENCY, "Application No. 10/211, 712, filed August 2, 2002.

CROSS-REFERENCED CASE:

[0007] The following application is cross-referenced and incorporated herein by reference:

[0008] U. S. Provisional Application No. 60/305,986 entitled "DATA REPLICATION PROTOCOL," by Dean Bernard Jacobs, Reto Kramer, and Ananthan Bala Srinvasan, filed July 16, 2001.

FIELD OF THE INVENTION

[0009] The invention relates generally to a computer implemented method for maintaining account balance consistency in networked account access systems; a computer implemented method for providing quick electronic transactions; a computer implemented system and method for maintaining concurrency for account information cached on a network; a computer implemented system for allowing a transaction over an account access system network; a method for updating a data item in a cluster; a method for maintaining concurrency for a copy of a data item cached on a cluster server; a computer program product for execution by a server computer for updating a data item on a network; and a system for updating a data item on a network. The invention relates generally to a system for storing data. The invention relates more specifically to a system and method for caching data and checking concurrency.

BACKGROUND OF THE INVENTION

[0010] When a data item is stored in a single database or data store that is accessible over a network, it is often the case that multiple servers or clients will require access to that data item. Traditionally, this requires data be read from the database each time the data item is accessed. Each read from the database is relatively resource intensive and may be relatively inefficient.

[0011] One way of overcoming some of the efficiency and scalability problems, associated with requiring a server or client to read from the database each time a data item is to be accessed, is to store the data item in cache memory. In this way, once a server or client has read a data item from the database it may simply store a copy of that item in a local cache. That local copy of the data item can then be used if future access is needed. This process may be appropriate and efficient for data items that never change, but problems arise when a data item is updated in the database.

[0012] If a data item stored in the database is updated, a copy of that data item stored in a local cache on the network may be different from the item in the

database, as it will not automatically receive the update. The problem intensifies when there are multiple local copies on different servers and/or clients on the network. Since each of these local copies is created at a different time, there can be multiple versions of the data item on the network. If a user tries to update or view the data item, the copy accessed by the user may not be current and/or correct.

[0013] These problems with concurrency can have drastic consequences, such as for example when a user accesses a data item showing a bank account balance. If the local copy of the bank account balance has not been updated to show a withdrawal, for example, the bank account balance shown to the user may in fact show an incorrectly large balance. This could lead the user to unknowingly overdraw the account. Further, a third party accessing the account balance, or a device such as an ATM, would have no way of knowing that the balance being shown is incorrect.

[0013A] It is generally desirable to overcome, or ameliorate, one or more of the above described difficulties, or to at least provide a useful alternative.

SUMMARY OF THE INVENTION

[0013B] In accordance with one aspect of the present invention, there is provided a computer implemented method for maintaining account balance consistency in networked account access systems, comprising:

receiving a request relating to the account balance of a customer, the request being received by an account access system storing a local copy of the account balance;

processing the request using the local copy;

verifying that the local copy reflects the current account balance for the user by sending a conditional update request to an account database; and

whereby, as a result of the conditional update request, the account database updates the account balance if the local copy is current with the account balance before the update, and whereby the account database does not update

2002313826 14 Dec 2007

the account balance if the local copy is not current with the account balance before the update.

[0013C] In accordance with a further aspect of the present invention, there is provided a computer implemented method for maintaining account balance consistency in networked account access systems, comprising:

processing an update request on an account access system, the account access system storing a local copy of an account balance to be used in processing the request; and

sending a conditional update request to an account database containing the account balance, whereby, as a result of the conditional update request, the account database updates the account balance if the local copy is current with the account balance before the update, and whereby the account database does not update the account balance if the local copy is not current with the account balance before the update.

[0013D] In accordance with a further aspect of the present invention, there is provided a computer implemented method for providing quick electronic transactions, comprising:

receiving a transaction request from a participant in a transaction to an electronic transaction system, the electronic transaction system storing a local copy of information related to a participant in the transaction;

processing the request with the local copy;

verifying that the local copy reflects the current information for the participant in a main database with a conditional update request; and

whereby, as a result of the conditional update request, the main database updates the account balance if the local copy is current with the account balance before the update, and whereby the main database does not update the account balance if the local copy is not current with the account balance before the update.

[0013E] In accordance with a further aspect of the present invention, there is provided a computer implemented method for maintaining concurrency for account information cached on a network, comprising:

receiving an update request to a network server, the network server storing a local copy of a account information in a local cache;

processing the request using the local copy of the account information;

sending a conditional update request to a network database storing an original copy of the account information whereby, as a result of the conditional update request, the database updates the account information if the local copy is current with the account information, and whereby the database does not update the account information if the local copy is not current with the account information;

receiving a current copy of the account information to the network server and sending another conditional update if the local copy was not the same version as the original copy; and

notifying from the network server any other servers on the network storing a local copy of the account information that the original copy has been updated.

[0013F] In accordance with a further aspect of the present invention, there is provided a computer implemented system for assuring concurrency among account access systems on a network, comprising:

an account access system adapted to receive a transaction request from a customer and process the request using a local copy of the account information for the customer, the account information being stored in an account database;

wherein the account access system is adapted to send a conditional update request to the account database whereby, as a result of the conditional update request, whereby the account database updates the account information if the local copy is current with the account information before the update, and

whereby the account database does not update the account information if the local copy is not current with the account information before the update.

[0013G] In accordance with a further aspect of the present invention, there is provided a computer implemented system for allowing a transaction over an account access system network, comprising:

an ATM terminal adapted to allow a customer to make a transaction request involving bank account information;

an account access system adapted to store a copy of the bank account information for the customer and process the transaction request; and

a bank account database adapted to store bank account information for the customer and provide access to that bank account information over the network;

wherein the account access system is adapted to send a conditional update request to the bank account database after processing the request whereby, as a result of the conditional update request, the bank account database is adapted to update the bank account information if the local copy is current with the bank account information before the update, and whereby the bank account database does not update the bank account information if the local copy is not current with the bank account information before the update.

[0013H] In accordance with a further aspect of the present invention, there is provided a method for updating a data item in a cluster, comprising:

processing an update request on a cluster server, the cluster server storing a local copy of a data item at a local cache to be used in processing the request;

2002313826 14 Dec 2007

sending a predicated update request to a cluster database containing the data item, whereby the database updates the data item if the local copy at the local cache is current with the data item before the update, and whereby the database does not update the data item if the local copy is not current with the data item before the update.

[0013I] In accordance with a further aspect of the present invention, there is provided a method for maintaining concurrency for a copy of a data item cached on a cluster server, comprising:

receiving an update request, the update request being received by a cluster server storing a local copy of a data item in local cache;

processing the request using the local copy of the data item;

sending a predicated update request to a cluster database storing the data item, whereby the cluster database updates the data item if the data item is current with the local copy, and whereby the database does not update the data item if the data item is not current with the local copy;

requesting a current copy of the data item for the cluster server and sending another predicated update if the local copy is not current with the data item in the cluster database, the step of requesting a current copy and sending another predicated update continuing until one of the cluster database updating the data item and the method being aborted; and

notifying other servers in the cluster that the data item in the database has been updated.

[0013J] In accordance with a further aspect of the present invention, there is provided a computer-readable medium, comprising:

means for processing an update request on a cluster server, the cluster server storing a local copy of a data item to be used in processing the request at a local cache;

2002313826 14 Dec 2007

means for sending a predicated update request to a cluster database containing the data item, whereby the database updates the data item if the local copy is current with the data item before the update, and whereby the database does not update the data item if the local copy is not current with the data item before the update.

[0013K] In accordance with one aspect of the present invention, there is provided a computer program product for execution by a server computer for updating a data item on a network, comprising:

computer code that can process an update request on a cluster server, the cluster server storing a local copy of a data item to be used in processing the request at a local cache;

computer code that can send a predicated update request to a cluster database containing the data item, whereby the database updates the data item if the local copy is current with the data item before the update, and whereby the database does not update the data item if the local copy is not current with the data item before the update.

[0013L] In accordance with a further aspect of the present invention, there is provided a system for updating a data item on a network, comprising:

means for processing an update request on a cluster server, the cluster server storing a local copy of a data item to be used in processing the request at a local cache;

means for sending a predicated update request to a cluster database containing the data item, whereby the database updates the data item if the local copy is current with the data item before the update, and whereby the database does not update the data item if the local copy is not current with the data item before the update.

[0013M] In accordance with a further aspect of the present invention, there is provided a computer system comprising:

a processor;

2002313826 14 Dec 2007

object code executed by said processor, said object code configured to:

process an update request on a cluster server, the cluster server storing a local copy of a data item to be used in processing the request at a local cache;

send a predicated update request to a cluster database containing the data item, whereby the database updates the data item if the local copy is current with the data item before the update, and whereby the database does not update the data item if the local copy is not current with the data item before the update.

[0014] It is therefore desirable to develop a system and method for caching data items and data objects that ensures the accuracy of the cached copy.

[0015] It is further desirable to develop a system and method to ensure that any change to a copy of a data item is not allowed unless that copy reflects the current state of the data item in the database.

[0016] Systems and methods in accordance with the present invention preferably provide a way to maintain concurrency in data item caching. A request to update an item is received by a network server, which can store a local copy of the data item, such as in local cache. The network server can process the request using the local copy of the data item. A "conditional" or "predicated" update request can be sent from the network server to a network database, whereby the database can update the data item if the data item contains the same version of the data as the local copy. The database may not update the data item if the data item is not the same version as the local copy.

[0017] If the copies do not contain the same version, the network server can request a current copy of the data item, and can process the update request using the new copy of the data item. The network server can send another predicated update to the database. This process continues until the data item in the database is updated. Once the data item is updated, the other network servers, such as

servers in a common cluster, can be notified that the data item has been updated. At this point, those network servers can drop any local copy of the data item and can request a new copy to store in local cache.

[0018] The notification to the network servers can be done by any of several appropriate methods, such as by multicasting an update message or version number to any other servers on the network. The network servers can also connect to each other directly, such as by a point- to-point protocol, or can heartbeat information to the other servers.

BRIEF DESCRIPTION OF THE DRAWINGS

[0019] Preferred embodiments of the present invention are hereafter described, by way of non-limiting example only, with reference to the accompanying drawings, in which:

[0020] Figure 1 is a diagram of the first part of an approach in accordance with one embodiment of the present invention.

[0021] Figure 2 is a diagram of the first and second parts of an approach in accordance with one embodiment of the present invention.

[0022] Figure 3 is a flowchart for an update process in accordance with one embodiment of the present invention.

[0023] Figure 4 is a flowchart for a process for updating a data item when the local copy and original copy are out-of-sync, in accordance with one embodiment of the present invention.

[0024] Figure 5 is a flowchart for a process for updating network servers on the network, in accordance with one embodiment of the present invention.

[0025] Figure 6 is a flowchart for a one phase process in accordance with one embodiment of the present invention. [0026] Figure 7 is a flowchart for a two phase process in accordance with one embodiment of the present invention.

2002313826 14 Dec 2007

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS OF THE INVENTION

[0027] Systems in accordance with a preferred embodiment of the present invention allow for the caching of data while maintaining concurrency across a network, such as a local area network (LAN), ethernet, or Internet. Such a system can utilize one or both of a two-part approach to updating data items while maintaining concurrency. Such systems can implement concurrent caching through any software or hardware means known or used in the computer arts, or hereinafter developed. These systems can also utilize any appropriate software applications, objects, languages, or executables, such as may be designed to utilize, for example, Java, HTML, and XML.

[0028] In the first part of one such approach in accordance with the present invention, a client or server on a network reads a data item from a database and stores a copy of the data item in a local cache. If the server or client wishes to update the data item in the database, the update can be "conditioned" or "predicated" on whether the data item stored in local cache corresponds to the current version of the data item stored in the database. This approach maintains concurrency between the client/server desiring to update the data item and the database. The caching of data in this manner can also improve performance and scalability.

[0029] One example of a first part of an approach 100 is shown in Figure 1. Here, a client 102 makes an update request 104 to a network server 106. The network server 106 in this example stores a copy of the data item 110 to be updated in a local cache 108. When the network server 106 receives the update request 104, the server 106 checks the local copy of the item 110 to see if the update may be processed. If the server 106 determines that the update may be processed using information in the local copy of the data item 110, the server 106 sends a predicated

update **112** to the database **114** storing the original copy of the data item **116**. If the information in the original copy of the data item **116** is the same as the information in the local copy **110**, the update may be committed to the database. If the information is different, the update is not committed. The server **106** receives an update status message **118** from the database **114**, indicating whether the update was committed.

[0030] If the update was committed, the server can also commit the update to the copy of the data item **110** in local cache **108**. If the update was not committed, because the data items **110**, **116** were out of sync, the server can drop its copy of the data item **110** from local cache **108** and request a new copy from the database **114**.

[0031] Once the server **106** has the new data item, it can again send a predicated update **112** to the database **114**. Alternatively, the server can send a message to the client **102** asking whether or not to attempt an update on the new data item. The server **106** can either abort the update, or continue the process of trying a predicated update and getting new copies of the data item as needed until the update is committed. Once the update is committed or aborted, the server **106** can send an update response **120** to the client, indicating the end result of the update attempt.

[0032] The second part of this approach occurs after a client/server has updated a data item in the database. Since other clients and/or servers on the network may also have a copy of the data item stored in local cache, the client/server making the update can contact the other servers on the network to let them know that the data item has been updated. The other clients and/or servers on the network can then update a copy of the data item stored in a local cache, request a current copy of the data item, or simply drop the local copy of the data item and request a copy from the database if and when it is needed. If a copy is later requested, the copy can be stored in local cache at that time.

[0033] **Figure 2** shows the second stage of the approach **100** described with respect to Figure 1. In **Figure 2**, once server **106** updates the data item **116** in the database **114** and the copy of the data item **110** in local cache **108**, server **106** sends update messages **128**, **130** to the other servers **122**, **124** in the cluster **126**. These messages can take the form of point-to-point messages or multicast heartbeats, such as is described above.

[0034] For example, in a banking system, each server on the banking system network can potentially store a copy of a user's bank account balance in local cache. Each local cache can include other information about a user account, such as account information and transaction history. This information can be cached, in whole or in part, on each server on the network.

[0035] In such a system, a transaction may occur such as an ATM transaction. A server in communication with the ATM can store a cached copy of the account balance of the user initiating the transaction. If a user of the ATM wishes to withdraw $100 from a user account, for example, the server could read the balance from memory, determine whether the account contains sufficient funds for the transaction, and subtract the $100 from the account balance either before or after disbursing the funds.

[0036] In order to prevent the user from overdrawing the account, the server can first verify that the local copy of the user account balance is current with the balance stored in the database. For example, if the previous balance stored locally was $500, the server could send an update message to the database such as "update balance = $400", which could also include the current value of the account in local cache, and make the update predicated on the fact that the current account balance in the database is the same as the current balance of the account in local cache.

[0037] If the account balance stored in the database is not the same as the balance in local cache, the server may roll back the update. Once an update is rolled back, the server can drop its copy in local cache, read

the account information from the database, then attempt the update again. For example, if the local copy of the account balance said that the balance was $500, and the database reflected a balance of $1000, the server would roll back the $400 update attempt and try a predicated $900 update, subtracting the $100 withdrawal from the current $1000 balance. This second update can again be predicated on the fact that the account balance has not changed since it was last read by the server. This process continues until either the account balance is updated appropriately, or the transaction is aborted due to insufficient funds, etc.

[0038] If a server succeeds in updating a data item in the database, it can also update the copy in local cache, such that the local copy is current with the version of the data item in the database. For a network in which multiple servers (or clients) can have a copy of the data item in a local cache, the server updating the data item can notify the other servers that the data item has been updated. This can include any or all other servers or clients on a network, in a domain, in a cluster, or in any other network grouping. This notification can be accomplished in any of a number of ways, such as by a point-to-point connection with each server/client, by multicasting, by a one-phase distribution method, by a two-phase distribution method, by heartbeating an update or a delta, or any other appropriate messaging technique.

[0039] It may be desirable that the sending of the notification is both reliable in the face of failures and scalable, such that the process makes efficient use of the network. One simple approach is to have the server updating the data item ("updating server") individually contact each server or client on the network ("network server") and transfer a message over a point-to-point link, such as a TCP/IP connection. The message can tell these network servers that the data item has been updated, and that the network servers should drop any copy of this data item in local cache. This approach may lead to inconsistent copies of the data if one or more of the

network servers are temporarily unreachable, or if the network servers encounter an error in processing the update.

[0040] Steps in a general process that can be used in accordance with the present invention are shown in **Figures 3-5**. In the process **200** of **Figure 3**, an update request is received, such as from a client, to a network server storing a local copy of the data item to be updated **202**. The update is processed by the network server using the local copy of the data item **204**. A predicated update request is sent from the network server to the network database containing the original copy of the data item **206**. If the original copy and local copy of the data item contain the same version of the data item, the update request is committed **208**. If not, the predicated update request is aborted **210**.

[0041] **Figure 4** shows a process **300** that can be used if the predicated update request is aborted. The network server can request a current copy of the data item **302**. The network server can then process the update request using the current copy of the data item **304**. A predicated update request is again sent from the network server to the network database containing the original copy of the data item **306**. If the original copy and current copy of the data item contain the same version of the data item, the update request is committed **308**. If not, the predicated update request is again aborted **310** and the process **300** may be repeated until the update is committed.

[0042] **Figure 5** shows a process **400** that may be used once the update is committed. Any other servers on the network, such as servers in the scope of an update or servers in a common cluster or domain, are notified that the data item is being updated **402**. Any server that is notified then drops any local copy of the data item being stored, such as in a local cache **404**. Those servers may choose to request an updated copy of the data item, either soon after dropping the local copy or upon receiving a subsequent request relating to that data item **406**.

[0043] In the case of a two-phase commit, any other servers on the network, such as in the scope of an update or in the same cluster, can be notified that an item is being updated during the commit. For example, an update can first go through a prepare stage in which it is determined whether or not the update can be successfully committed. During this phase, or at least before the update is committed, any server that is notified of the update can veto the commit. By vetoing the commit, any preparation is rolled back and the update does not get written to the database. If the update successfully goes through a prepare phase, and does not get vetoed by a server, the update can get committed to the data item in the database.

[0044] The sending of the notification can also be sent by multicasting the notification to the other servers/clients that might be caching a local copy of the data item. Multicasting in this instance may comprise the updating server sending the notification once to the network/cluster/domain, which is then passed to the network servers/clients. In simple multicasting, the message is only sent once, such that a server that does not receive the update may fail to drop the outdated copy of the item. This can result in that server having to go through two or more iterations of predicated update attempts for that data item when processing a subsequent request.

[0045] The sending of the notification can also be sent through a "heartbeat." A heartbeat in this approach is a periodic message, typically multicast although other messaging means may be utilized, that is sent to servers/clients that might be storing a local copy of the data item. An updating server can continue to heartbeat the latest update(s) for a given period of time, for a given number of heartbeats, until each server/client responds it has received the heartbeat, or any other appropriate measure.

[0046] Each update to a data item can be packaged as an incremental delta between versions. A protocol in accordance with the present invention may integrate two methods for the distribution of

updates, although other appropriate methods can be used accordingly. These distribution methods are referred to as a one-phase method and a two-phase method, and provide a tradeoff between consistency and scalability. In a one-phase method, which can favor scalability, each of the network servers obtains and processes updates at its own pace. The network servers get updates from an updating server at different times, but commit to each update as soon as the update is received. One of the network servers can encounter an error in processing an update, but in the one-phase method this does not prevent the network servers from processing the update.

[0047] In a two-phase method in accordance with the present invention, which can favor consistency, the distribution is "atomic," in that either all or none of the network servers successfully process the update. There are separate phases, such as prepare and commit phases, which can allow for a possibility of abort. In the prepare phase, the updating server determines whether each of the network servers can take the update. If all the network servers indicate that they can accept the update, the new data is sent to the network servers to be committed in the commit phase. If at least one of the network servers cannot take the update, the update can be aborted, resulting in no commit. In this case, an updating server is informed that it should roll back the prepare and nothing is changed. Such a protocol in accordance with the present invention is reliable, as one of the network servers that is unreachable when an update is committed, in either method, eventually gets the update.

[0048] A system in accordance with the present invention can also ensure that a temporarily unavailable server eventually receives all updates. For example, a server may be temporarily isolated from the network, then come back into the network without restarting. Since the server is not restarting, it normally would not check for updates. The server coming back into the network can be accounted for by having the server

check periodically for new updates, or by having an updating server check periodically to see whether the network servers have received the updates.

[0049] In one embodiment, an updating server regularly sends multicast "heartbeats" to the network servers, such as for a given period of time or a given number of heartbeats. Since a multicast approach can be unreliable, it is possible for one of the network servers to miss arbitrary sequences of heartbeats. For this reason, heartbeats can contain a window of information about recent updates. Such information about previous updates can be used to reduce the amount of network traffic, as explained below. In an example such as an account balance, historical information may not be necessary, such that a heartbeat may simply contain the current balance.

[0050] The updating server can continue to periodically send a multicast heartbeat containing the version number to the network servers. This allows any server that was unavailable, or unable to receive and process a delta, to determine that it is not on the current version of the data item and request a delta or update at a later time, such as when the slave comes back into the system. If the current value is contained in the heartbeat, the server may simply commit the new value.

[0051] For an update in a one-phase method, these heartbeats can cause each of the network servers to request a delta starting from that server's current version of the data item. Such a process is shown in the flowchart of **Figure 6**. In this basic process **500** a version number for the current data item on the updating server, or in the database, is sent from the updating server to one of the other network servers **502**. The network server determines whether it has been updated to the current version number **504**. If the network server is not on the current version, it requests that a delta be sent from the updating server containing the information needed to update the data item **506**. When the delta is sent, the network server processes the delta in order to update to the current version **508**.

The network server also updates its version number for the data item to the current version number **510**.

[0052] For an update in a two-phase method, the updating server can begin with a prepare phase in which it pro-actively sends each of the network servers a delta from the immediately-previous version. Such a process is shown in the flowchart of **Figure 7**. In this basic process **600**, a packet of information is sent from the updating server to at least one other network server **602**. Each of the network servers receiving the packet determines whether it can process that packet and update to the current version **604**. Each server receiving the packet responds to the updating server, indicating whether the network server can process the packet **606**. If all the network servers (to which the delta is sent) acknowledge successful processing of the delta within some timeout period, the updating server can decide to commit the update. Otherwise, the updating server can decide to abort the update. Once this decision is made, the updating server sends a message to the network server(s) indicating whether the update should be committed or aborted **608**. If the decision is to commit, each of the network servers processes the commit **610**. Heartbeats can further be used to signal whether a commit or abort occurred, in case the command was missed by one of the slaves.

[0053] In addition to the ability of a server to pull a delta, an updating server can have the ability to push a delta during two-phase distribution. In one embodiment, these deltas are always between successive versions of the data. This two-phase distribution method can minimize the likelihood of inconsistencies between participants. Servers can process a prepare as far as possible without exposing the update to clients or making the update impossible to roll back. This may include such tasks as checking the servers for conflicts. If any of the servers signals an error, such as by sending a "disk full" or "inconsistent configuration" message, the update can be uniformly rolled back.

[0054] It is still possible, however, that inconsistencies may arise. For instance, there may be errors in processing a commit, for reasons such as an inability to open a socket. Servers may also commit and expose the update at different times. Because the data cannot reach every managed server at exactly the same time, there can be some rippling effect. The use of multicasting provides for a small time window, in an attempt to minimize the rippling effect. In one embodiment, a prepared server will abort if it misses a commit, whether it missed the signal, the master crashed, etc.

[0055] A best-effort approach to multicasting can cause a server to miss a commit signal. If an updating server crashes part way through the commit phase, there may be no logging or means for recovery. There may be no way for the updating server to tell the remaining servers that they need to commit. Upon abort, some servers may end up committing the data if the version is not properly rolled back. In one embodiment, the remaining servers could get the update using one-phase distribution. This might happen, for example, when a server pulls a delta in response to a heartbeat received from an updating server. This approach may maintain system scalability, which might be lost if the system tied down distribution in order to avoid any commit or version errors.

[0056] If the information regarding the previous versions was not included in a delta, a server might have to abort and restart if that server was prepared but missed a commit. With the inclusion of older version information, the server can commit that portion of the update it was expecting upon the prepare, and ask for a new delta to handle more recent updates. Information about a given version can be included for at least some fixed, configurable number of heartbeats, although rapid-fire updates may cause the window to increase to an unacceptable size. In another embodiment, information about an older version is discarded once an updating server determines that all network servers have received the update.

[0057] Multicast heartbeats can have several properties that need to be taken into consideration. These heartbeats can be asynchronous or "one-way". As a result, by the time a server responds to a heartbeat, the updating server or database may have advanced to a new state. Further, not all servers respond at exactly the same time. As such, an updating server can assume that a server has no knowledge of its state, and can include that which the delta is intended to update.

[0058] These heartbeats can also be unreliable, as a slave may miss arbitrary sequences of heartbeats. This can again lead to the inclusion of older version information in the heartbeats. In one embodiment, heartbeats are received by a server in the order in which they were sent. For example, a server may not commit version seven until it has committed version six. The server can wait until it receives six, or it can simply throw out six and commit seven. This ordering eliminates the possibility for confusion that might be created by versions going backwards.

[0059] The foregoing description of the preferred embodiments of the present invention has been provided for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Many modifications and variations will be apparent to the practitioner skilled in the art. Embodiments were chosen and described in order to best describe the principles of the invention and its practical application, thereby enabling others skilled in the art to understand the invention, the various embodiments and with various modifications that are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following claims and their equivalents.

2002313826 14 Dec 2007

Throughout this specification and claims which follow, unless the context requires otherwise, the word "comprise", and variations such as "comprises" or "comprising", will be understood to imply the inclusion of a stated integer or group of integers or steps but not the exclusion of any other integer or group of integers.

The reference in this specification to any prior publication (or information derived from it), or to any matter which is known, is not, and should not be taken as an acknowledgment or admission or any form of suggestion that that prior publication (or information derived from it) or known matter forms part of the common general knowledge in the field of endeavour to which this specification relates.

2002313826 14 Dec 2007

THE CLAIMS DEFINING THE INVENTION ARE AS FOLLOWS:

1. A computer implemented method for maintaining account balance consistency in networked account access systems, comprising:

receiving a request relating to the account balance of a customer, the request being received by an account access system storing a local copy of the account balance;

processing the request using the local copy;

verifying that the local copy reflects the current account balance for the user by sending a conditional update request to an account database; and

whereby, as a result of the conditional update request, the account database updates the account balance if the local copy is current with the account balance before the update, and whereby the account database does not update the account balance if the local copy is not current with the account balance before the update.

2. A computer implemented method according to claim 1, further comprising:

notifying other account access systems on the network that the account balance has been updated.

3. A computer implemented method according to claim 1, further comprising:

requesting the account balance if the local copy is not current with the account balance.

4. A computer implemented method for maintaining account balance consistency in networked account access systems, comprising:

processing an update request on an account access system, the account access system storing a local copy of an account balance to be used in processing the request; and

sending a conditional update request to an account database containing the account balance, whereby, as a result of the conditional update request, the

2002313826 14 Dec 2007

account database updates the account balance if the local copy is current with the account balance before the update, and whereby the account database does not update the account balance if the local copy is not current with the account balance before the update.

5. A computer implemented method according to claim 4, further comprising: receiving a message from the account database to the account access system indicating whether the account balance has been updated.

6. A computer implemented method according to claim 4, further comprising: reading the account balance from the account database and storing a local copy of the account balance on the account access system.

7. A computer implemented method according to claim 4, further comprising: receiving an update request from a client to the account access system, the client specifying the account balance to be updated.

8. A computer implemented method according to claim 4, further comprising: checking the local copy to determine whether the update request can be processed.

9. A computer implemented method according to claim 4, further comprising: updating the local copy on the account access system if the account balance in the account database is updated.

10. A computer implemented method according to claim 4, further comprising: deleting the local copy and storing a new copy of the account balance on the account access system if the local copy is not current with the account balance.

11. A computer implemented method according to claim 10, further comprising:

sending an additional predicated update request to the account database containing the account balance, whereby the account database updates the account balance if the new copy is current with the account balance before the update, and whereby the account database does not update the account balance if the new copy is not current with the account balance before the update.

12. A computer implemented method according to claim 10, further comprising:

determining whether the client initiating the update request wishes to attempt the update with the account balance current with the new copy.

13. A computer implemented method according to claim 4, further comprising:

notifying another account access system on the network that the account balance in the account database has been updated.

14. A computer implemented method according to claim 4, further comprising:

multicasting an update message to other account access systems on the network.

15. A computer implemented method according to claim 4, further comprising:

multicasting a version number for the updated account balance to other account access systems on the network.

16. A computer implemented method according to claim 4, further comprising:

heartbeating the version number for the updated account balance to other account access systems on the network.

17. A computer implemented method according to claim 4, further comprising:

dropping a local copy of the account balance on any other account access system on the network after the account balance is updated.

18. A computer implemented method according to claim 4, further comprising:

2002313826 14 Dec 2007

requesting an updated copy of the account balance on any other account access system on the network.

19. A computer implemented method according to claim 13, further comprising:

deleting a local copy of the account balance on any other account access system on the network being notified the account balance has been updated.

20. A computer implemented method according to claim 4, further comprising:

notifying another account access system on the network that the account balance in the database is going to be updated.

21. A computer implemented method according to claim 20, further comprising:

allowing said another account access system to veto the update of the account balance in the database.

22. A computer implemented method according to claim 4, further comprising:

sending a packet of information to another account access system on the network, the packet of information containing changes to the account balance due to the update.

23. A computer implemented method according to claim 22, wherein:

the packet of information contains changes between the state of the account balance after the update and the prior state of the account before the update.

24. A computer implemented method according to claim 4, further comprising:

determining whether other account access systems on the network can accept the update to the account balance; and

committing the update to the other account access systems if the other account access systems can accept the update.

25. A computer implemented method according to claim 24, further comprising:

rolling back the update if the other account access systems cannot accept the update.

26. A computer implemented method for providing quick electronic transactions, comprising:

receiving a transaction request from a participant in a transaction to an electronic transaction system, the electronic transaction system storing a local copy of information related to a participant in the transaction;

processing the request with the local copy;

verifying that the local copy reflects the current information for the participant in a main database with a conditional update request; and

whereby, as a result of the conditional update request, the main database updates the account balance if the local copy is current with the account balance before the update, and whereby the main database does not update the account balance if the local copy is not current with the account balance before the update.

27. A computer implemented method for maintaining concurrency for account information cached on a network, comprising:

receiving an update request to a network server, the network server storing a local copy of a account information in a local cache;

processing the request using the local copy of the account information;

sending a conditional update request to a network database storing an original copy of the account information whereby, as a result of the conditional update request, the database updates the account information if the local copy is current with the account information, and whereby the database does not update the account information if the local copy is not current with the account information;

receiving a current copy of the account information to the network server and sending another conditional update if the local copy was not the same version as the original copy; and

2002313826 14 Dec 2007

notifying from the network server any other servers on the network storing a local copy of the account information that the original copy has been updated.

28. A computer implemented system for assuring concurrency among account access systems on a network, comprising:

an account access system adapted to receive a transaction request from a customer and process the request using a local copy of the account information for the customer, the account information being stored in an account database;

wherein the account access system is adapted to send a conditional update request to the account database whereby, as a result of the conditional update request, whereby the account database updates the account information if the local copy is current with the account information before the update, and whereby the account database does not update the account information if the local copy is not current with the account information before the update.

29. A computer implemented system according to claim 28, further comprising:

an account database adapted to contain the account information for the customer.

30. A computer implemented system according to claim 28, further comprising:

a client terminal adapted to allow a customer to initiate the transaction request.

31. A computer implemented system according to claim 28, wherein:

the account access system is adapted to receiving a message from the account database indicating whether the account information has been updated.

32. A computer implemented system according to claim 28, wherein:

the account access system is further adapted to check the local copy to determine whether the update request can be processed.

2002313826 14 Dec 2007

33. A computer implemented system according to claim 28, wherein:

the account access system is further adapted to update the local copy if the account information in the account database is updated.

34. A computer implemented system according to claim 28, wherein:

the account access system is further adapted to delete the local copy and store a new copy of the account information if the local copy is not current with the account information.

35. A computer implemented system according to claim 34, wherein:

the account access system is further adapted to send an additional predicated update request to the account database, whereby the account database updates the account information if the new copy is current with the account information before the update, and whereby the account database does not update the account information if the new copy is not current with the account information before the update.

36. A computer implemented system according to claim 28, further comprising:

additional account access systems on the network capable of storing a local copy of the data item.

37. A computer implemented system according to claim 36, wherein:

the account access system is further adapted to notify the additional account access systems that the account information in the account database has been updated.

38. A computer implemented system according to claim 37, wherein:

the account access system is further adapted to notify the additional account access systems by one of multicasting and point-to-point messaging.

39. A computer implemented system according to claim 37, wherein:

2002313826 14 Dec 2007

the account access system is further adapted to include a version number for the account information when notifying the additional account access systems.

40. A computer implemented system according to claim 39, wherein:

the account access system is further adapted to heartbeat the version number for the account information after the update to the additional account access systems.

41. A computer implemented system according to claim 36, wherein:

the additional account access systems are each adapted to do at least one of delete a local copy of the account information and request an updated copy of the account information.

42. A computer implemented system according to claim 36, wherein:

the account access system is further adapted to notify the additional account access systems that the account information in the account database is going to be updated.

43. A computer implemented system according to claim 36, wherein:

the additional account access systems are each capable of vetoing the update of the account information in the database.

44. A computer implemented system according to claim 36, wherein:

the account access system is further adapted to send a packet of information to the additional account access systems, the packet of information containing changes to the account information due to the update.

45. A computer implemented system according to claim 36, wherein:

the account access system is further adapted to determine whether the additional account access systems can accept the update to the account

2002313826 14 Dec 2007

information and commit the update to the additional account access systems if the additional account access systems can accept the update.

46. A computer implemented system according to claim 36, wherein:

the account access system is further adapted to roll back the update if the additional account access systems cannot accept the update.

47. A computer implemented system for allowing a transaction over an account access system network, comprising:

an ATM terminal adapted to allow a customer to make a transaction request involving bank account information;

an account access system adapted to store a copy of the bank account information for the customer and process the transaction request; and

a bank account database adapted to store bank account information for the customer and provide access to that bank account information over the network;

wherein the account access system is adapted to send a conditional update request to the bank account database after processing the request whereby, as a result of the conditional update request, the bank account database is adapted to update the bank account information if the local copy is current with the bank account information before the update, and whereby the bank account database does not update the bank account information if the local copy is not current with the bank account information before the update.

48. A method for updating a data item in a cluster, comprising:

processing an update request on a cluster server, the cluster server storing a local copy of a data item at a local cache to be used in processing the request;

sending a predicated update request to a cluster database containing the data item, whereby the database updates the data item if the local copy at the local cache is current with the data item before the update, and whereby the database does not update the data item if the local copy is not current with the data item before the update.

2002313826 14 Dec 2007

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

receiving a message from the database to the cluster server indicating whether the data item has been updated.

50. A method according to claim 48, further comprising:

reading a data item from the database and storing a local copy of the data item in local cache on the cluster server.

51. A method according to claim 48, further comprising:

receiving an update request from a client to the cluster server.

52. A method according to claim 48, further comprising:

checking the local copy to determine whether the update request can be processed.

53. A method according to claim 48, further comprising:

updating the local copy on the cluster server if the data item in the database is updated.

54. A method according to claim 48, further comprising:

deleting the local copy and storing a new copy of the data item on the cluster server if the local copy is not current with the data item.

55. A method according to claim 54, further comprising:

sending an additional predicated update request to a cluster database containing the data item, whereby the database updates the data item if the new copy is current with the data item before the update, and whereby the database does not update the data item if the new copy is not current with the data item before the update.

2002313826 14 Dec 2007

56. A method according to claim 54, further comprising:

determining whether the client initiating the update request wishes to attempt the update with the data item current with the new copy.

57. A method according to claim 48, further comprising:

notifying another server in the cluster that the data item in the database has been updated.

58. A method according to claim 48, further comprising:

multicasting an update message to other servers in the cluster.

59. A method according to claim 48, further comprising:

multicasting a version number for the data item to other servers in the cluster after updating the data item.

60. A method according to claim 48, further comprising:

contacting each server in the cluster directly to indicate that the data item has been updated.

61. A method according to claim 48, further comprising:

heartbeating the version number for the updated data item to any other servers on the network.

62. A method according to claim 57, further comprising:

dropping a local copy of the data item stored on any cluster server being notified the data item has been updated.

63. A method according to claim 57, further comprising:

requesting an updated copy of the data item on any cluster server being notified the data item has been updated.

2002313826 14 Dec 2007

64. A method according to claim 57, further comprising:

deleting a local copy of the data item on any cluster server being notified the data item has been updated.

65. A method according to claim 48, further comprising:

notifying another server in the cluster that the data item in the database is going to be updated.

66. A method according to claim 65, further comprising:

allowing said another server to veto the update of the data item in the database.

67. A method according to claim 48, further comprising:

sending a packet of information to another server in the cluster, the packet of information containing changes to the data item due to the update.

68. A method according to claim 67, wherein:

the packet of information contains changes between the state of the data item after the update and the prior state of the data item before the update.

69. A method according to claim 48, further comprising:

determining whether other servers in the cluster can accept the update to the data item; and

committing the update to the other servers if the other servers can accept the update.

70. A method according to claim 69, further comprising:

rolling back the update if the other server cannot accept the update.

71. A method for maintaining concurrency for a copy of a data item cached on a cluster server, comprising:

2002313826 14 Dec 2007

receiving an update request, the update request being received by a cluster server storing a local copy of a data item in local cache;

processing the request using the local copy of the data item;

sending a predicated update request to a cluster database storing the data item, whereby the cluster database updates the data item if the data item is current with the local copy, and whereby the database does not update the data item if the data item is not current with the local copy;

requesting a current copy of the data item for the cluster server and sending another predicated update if the local copy is not current with the data item in the cluster database, the step of requesting a current copy and sending another predicated update continuing until one of the cluster database updating the data item and the method being aborted; and

notifying other servers in the cluster that the data item in the database has been updated.

72. A method according to claim 71, wherein the step of notifying other servers in the cluster further comprises:

multicasting an update message to other servers in the cluster that might be storing a local copy of the data item.

73. A method according to claim 71, wherein the step of notifying other servers in the cluster further comprises:

multicasting a version number for the data item to other servers in the cluster after the data item is updated.

74. A method according to claim 71, wherein the step of notifying other servers in the cluster further comprises:

contacting another server in the cluster by a point-to-point connection to indicate that the data item has been updated.

2002313826 14 Dec 2007

75. A method according to claim 71, wherein the step of notifying other servers in the cluster further comprises:

heartbeating the version number for the data item to other servers in the cluster after the data item is updated.

76. A method according to claim 71, further comprising:

dropping a local copy of the data item on any other server in the cluster being notified that the data item has been updated.

77. A method according to claim 71, further comprising:

requesting an updated copy of the data item to any other server in the cluster being notified that the data item has been updated.

78. A computer-readable medium, comprising:

means for processing an update request on a cluster server, the cluster server storing a local copy of a data item to be used in processing the request at a local cache;

means for sending a predicated update request to a cluster database containing the data item, whereby the database updates the data item if the local copy is current with the data item before the update, and whereby the database does not update the data item if the local copy is not current with the data item before the update.

79. A computer program product for execution by a server computer for updating a data item on a network, comprising:

computer code that can process an update request on a cluster server, the cluster server storing a local copy of a data item to be used in processing the request at a local cache;

computer code that can send a predicated update request to a cluster database containing the data item, whereby the database updates the data item if the local copy is current with the data item before the update, and whereby the

2002313826 14 Dec 2007

database does not update the data item if the local copy is not current with the data item before the update.

80. A system for updating a data item on a network, comprising:

means for processing an update request on a cluster server, the cluster server storing a local copy of a data item to be used in processing the request at a local cache;

means for sending a predicated update request to a cluster database containing the data item, whereby the database updates the data item if the local copy is current with the data item before the update, and whereby the database does not update the data item if the local copy is not current with the data item before the update.

81. A computer system comprising:

a processor;

object code executed by said processor, said object code configured to:

process an update request on a cluster server, the cluster server storing a local copy of a data item to be used in processing the request at a local cache;

send a predicated update request to a cluster database containing the data item, whereby the database updates the data item if the local copy is current with the data item before the update, and whereby the database does not update the data item if the local copy is not current with the data item before the update.

82. A computer implemented method for maintaining account balance consistency in networked account access systems, substantially as hereinbefore described with reference to the accompanying drawings.

83. A computer implemented method for maintaining account balance consistency in networked account access systems, substantially as hereinbefore described with reference to the accompanying drawings.

2002313826 14 Dec 2007

84. A computer implemented method for providing quick electronic transactions, substantially as hereinbefore described with reference to the accompanying drawings.

85. A computer implemented method for maintaining concurrency for account information cached on a network, substantially as hereinbefore described with reference to the accompanying drawings.

86. A computer implemented system for assuring concurrency among account access systems on a network, substantially as hereinbefore described with reference to the accompanying drawings.

87. A computer implemented system for allowing a transaction over an account access system network, substantially as hereinbefore described with reference to the accompanying drawings.

88. A method for updating a data item in a cluster, substantially as hereinbefore described with reference to the accompanying drawings.

89. A method for maintaining concurrency for a copy of a data item cached on a cluster server, substantially as hereinbefore described with reference to the accompanying drawings.

90. A computer program product for execution by a server computer for updating a data item on a network, substantially as hereinbefore described with reference to the accompanying drawings.

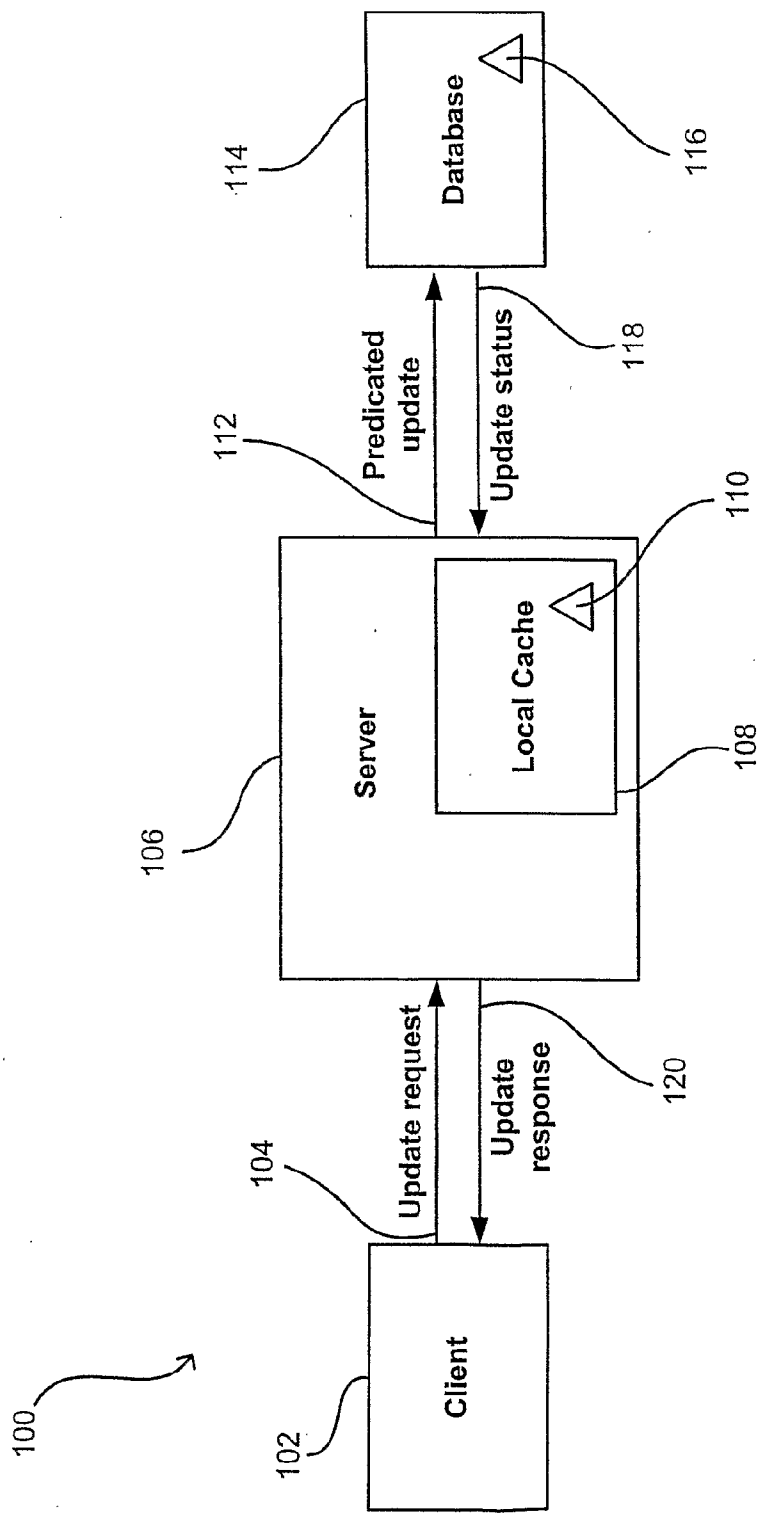

Database
114
116
Predicated update
Update status
118
112
110
Server
Local Cache
108
106
Update request
Update response
120
104
Client
102
100

Figure 1

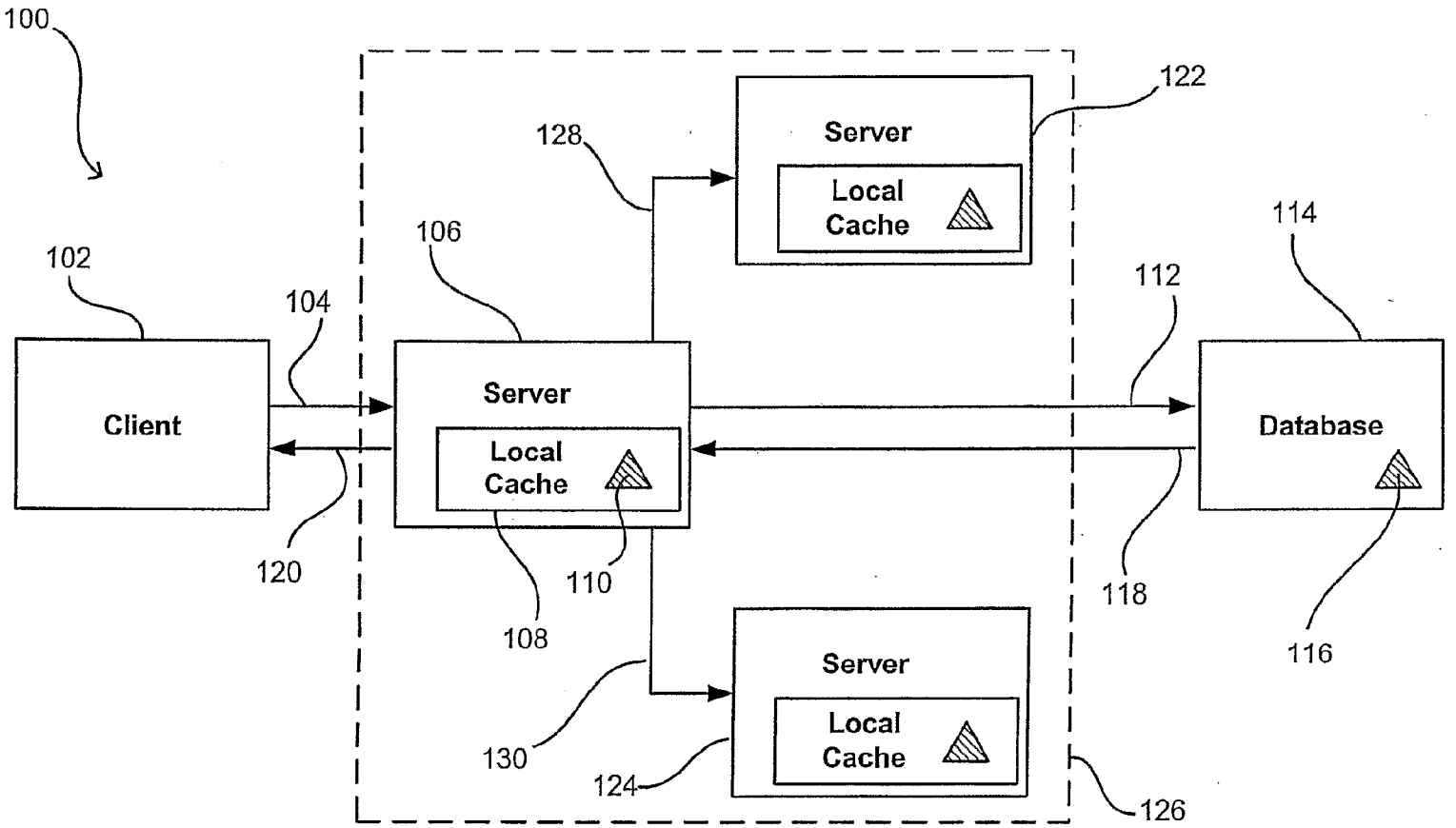

100
102
104
106
128
122
Server
Local Cache
114
112
Client
Server
Local Cache
Database
120
110
108
118
116
Server
Local Cache
130
124
126

Figure 2

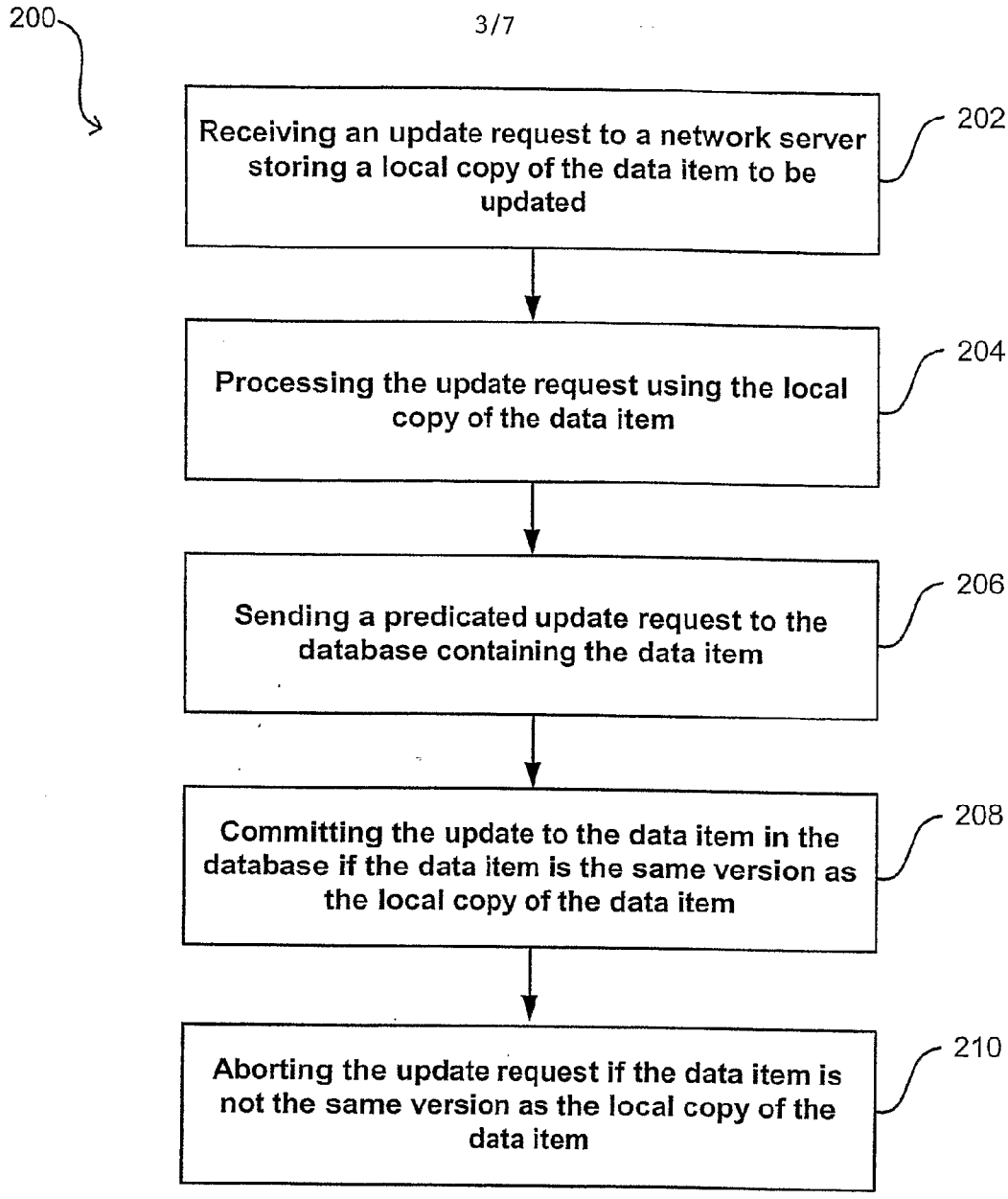

200
Receiving an update request to a network server storing a local copy of the data item to be updated
202
Processing the update request using the local copy of the data item
204
Sending a predicated update request to the database containing the data item
206
Committing the update to the data item in the database if the data item is the same version as the local copy of the data item
208
Aborting the update request if the data item is not the same version as the local copy of the data item
210

Figure 3

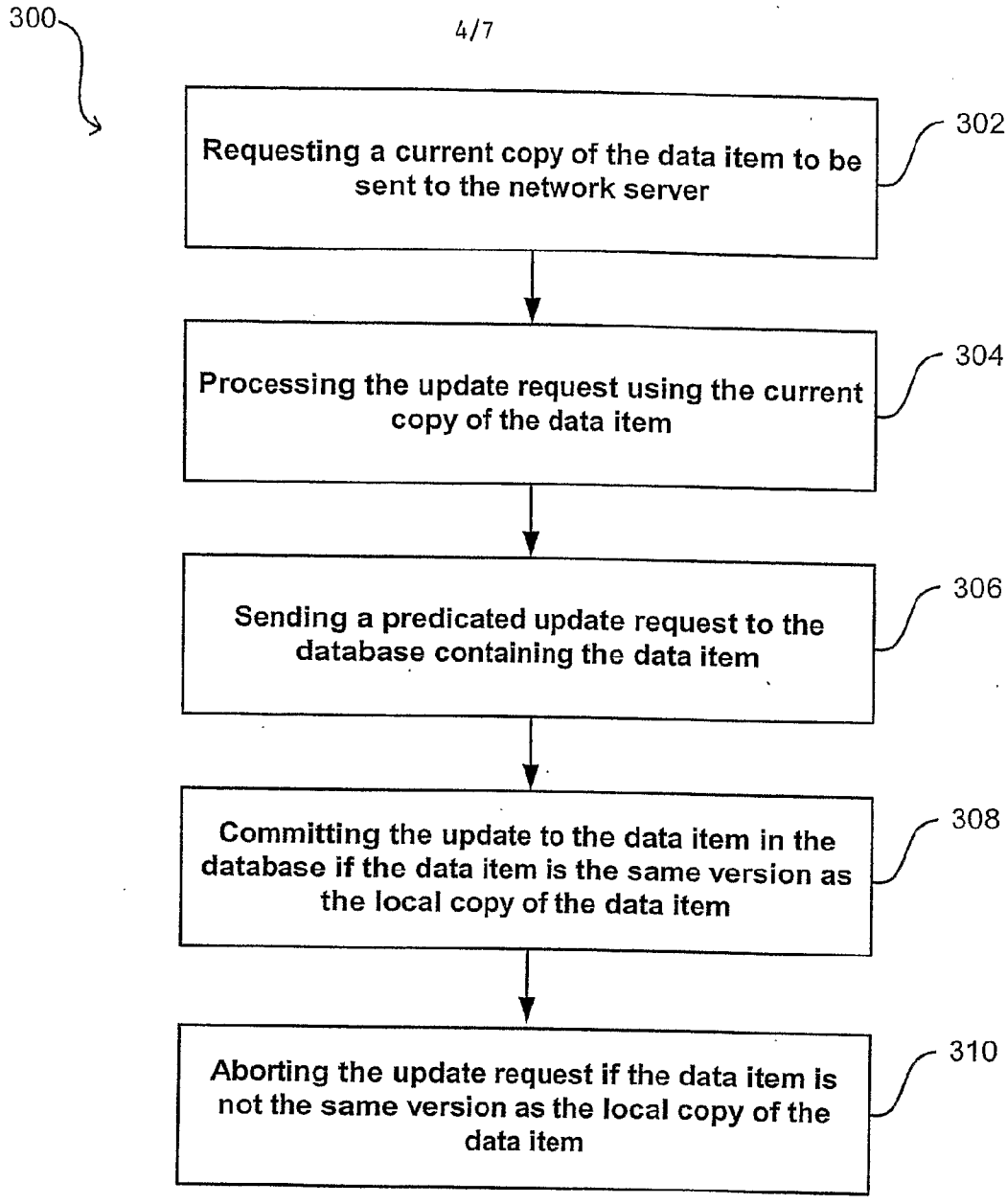

300
Requesting a current copy of the data item to be sent to the network server
302
Processing the update request using the current copy of the data item
304
Sending a predicated update request to the database containing the data item
306
Committing the update to the data item in the database if the data item is the same version as the local copy of the data item
308
Aborting the update request if the data item is not the same version as the local copy of the data item
310

Figure 4

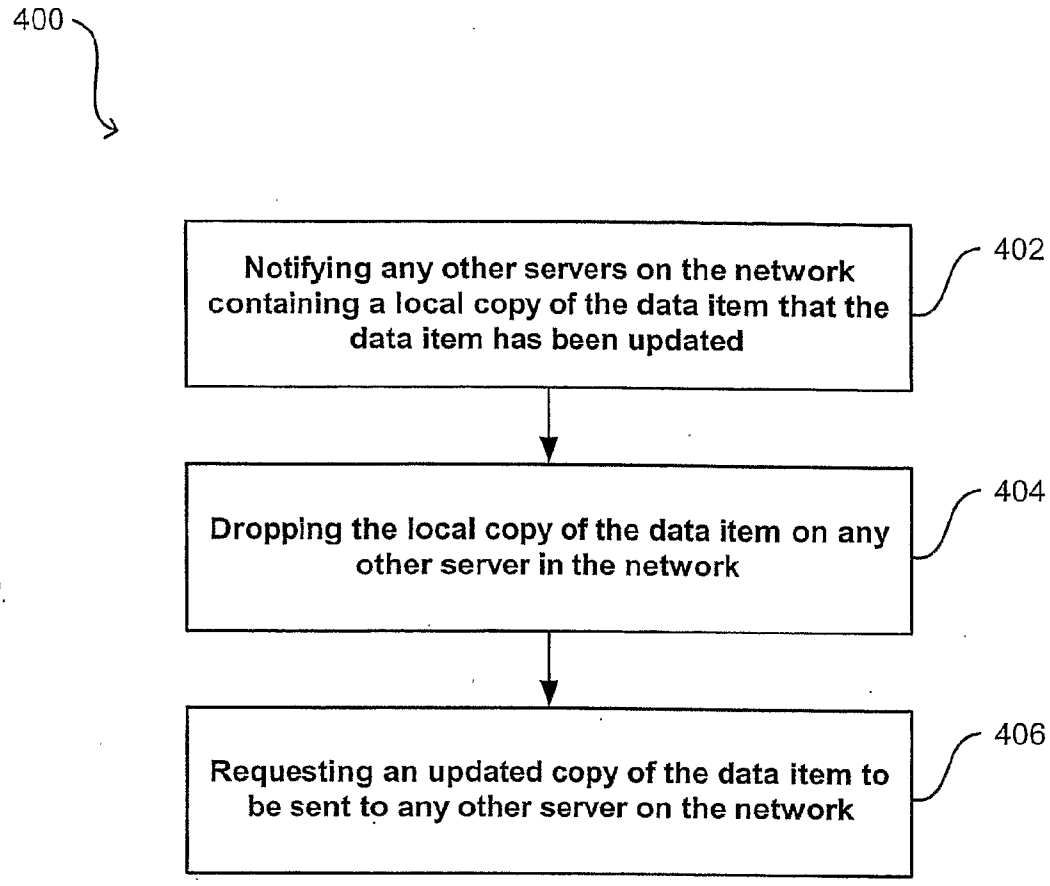

400
Notifying any other servers on the network containing a local copy of the data item that the data item has been updated
402
Dropping the local copy of the data item on any other server in the network
404
Requesting an updated copy of the data item to be sent to any other server on the network
406

Figure 5

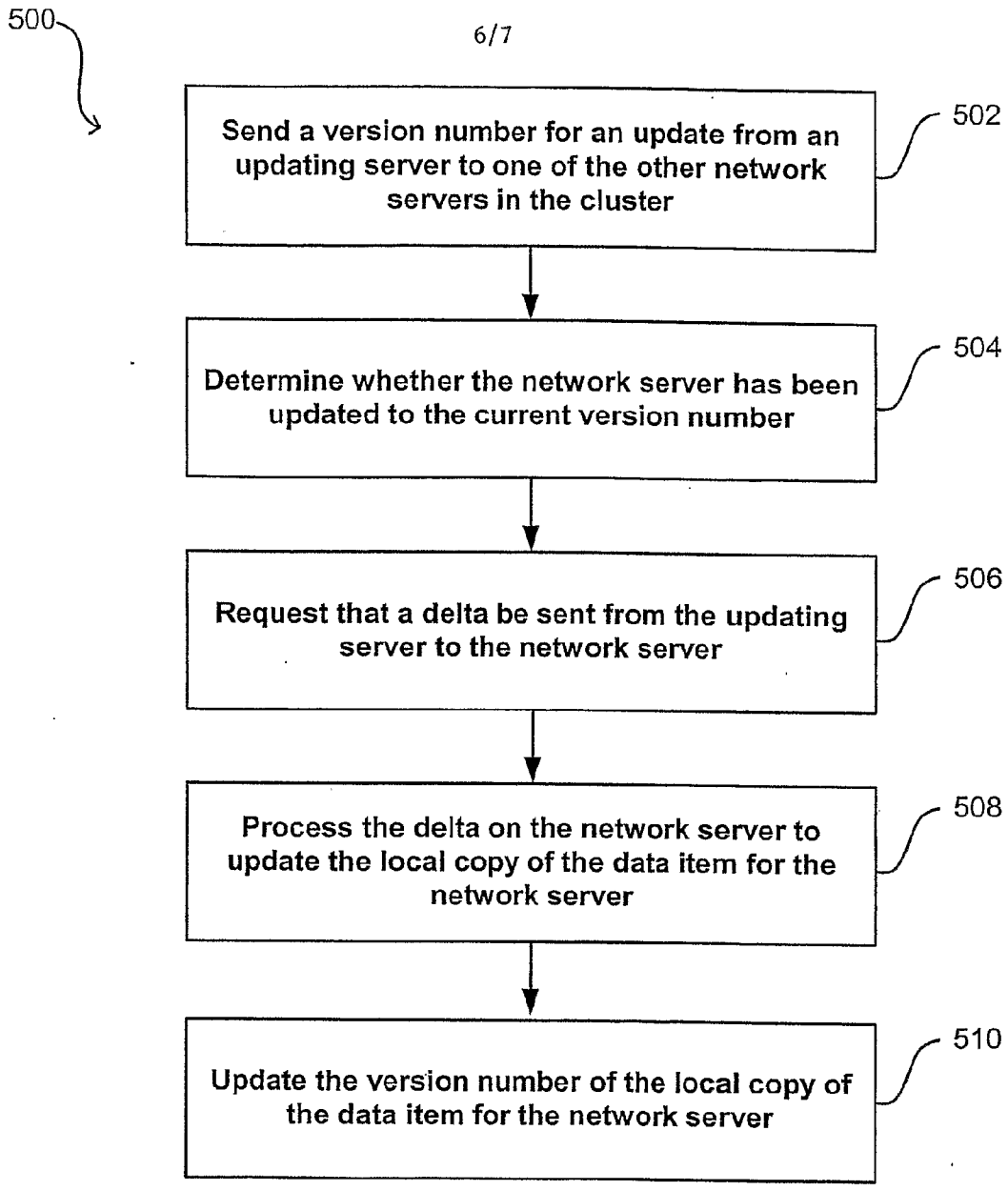

500
Send a version number for an update from an updating server to one of the other network servers in the cluster
502
Determine whether the network server has been updated to the current version number
504
Request that a delta be sent from the updating server to the network server
506
Process the delta on the network server to update the local copy of the data item for the network server
508
Update the version number of the local copy of the data item for the network server
510

Figure 6

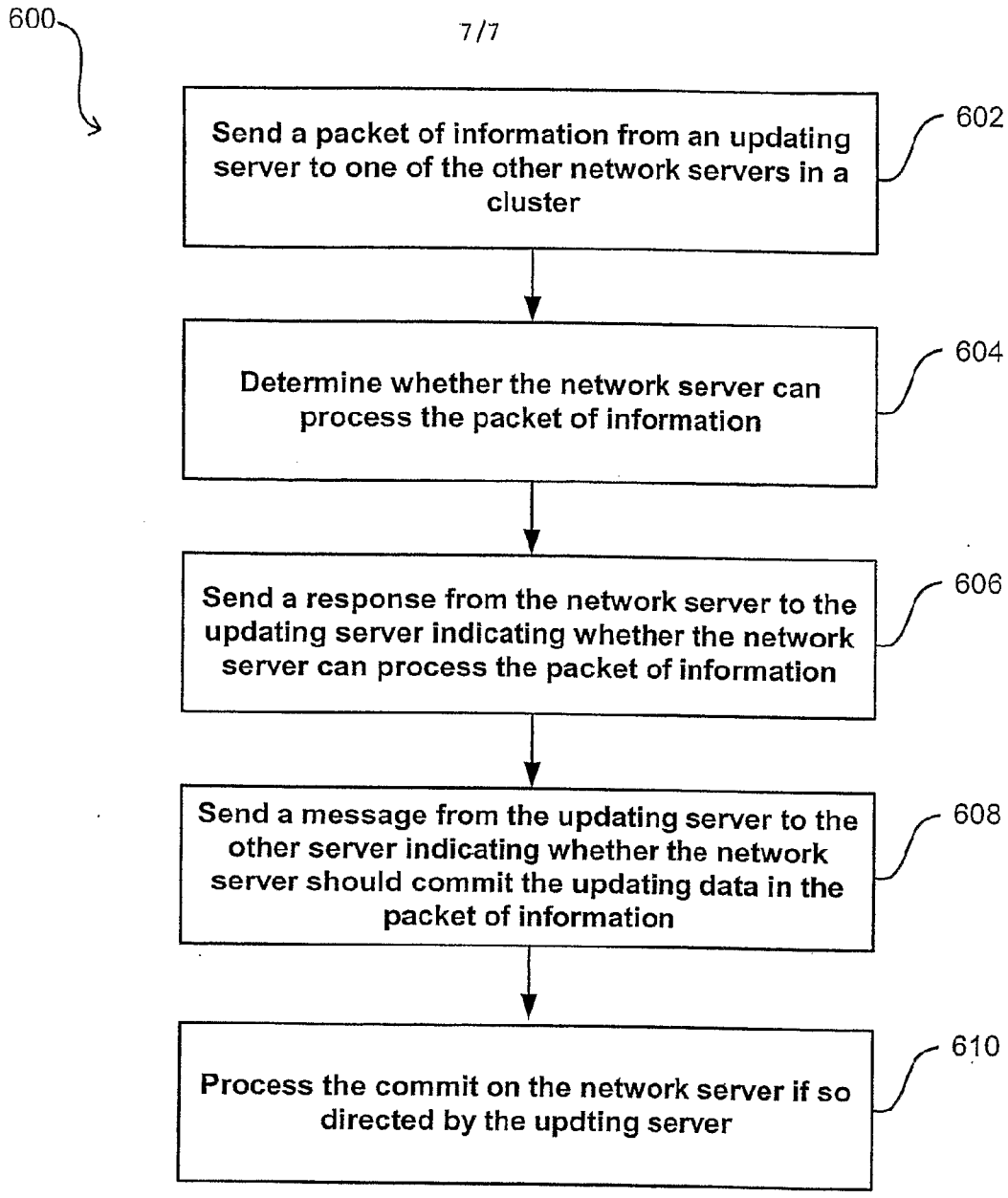

600
Send a packet of information from an updating server to one of the other network servers in a cluster
602
Determine whether the network server can process the packet of information
604
Send a response from the network server to the updating server indicating whether the network server can process the packet of information
606
Send a message from the updating server to the other server indicating whether the network server should commit the updating data in the packet of information
608
Process the commit on the network server if so directed by the updting server
610

Figure 7