US 20020129146A1

(19) United States
(12) Patent Application Publication (10) Pub. No.: US 2002/0129146 A1
Aronoff et al. (43) Pub. Date: Sep. 12, 2002

(54) HIGHLY AVAILABLE DATABASE CLUSTERS THAT MOVE CLIENT CONNECTIONS BETWEEN HOSTS

(76) Inventors: **Eyal Aronoff**, Irvine, CA (US); **Eyal Kalderon**, Tustin, CA (US); **Bill Romine**, Tustin, CA (US)

Correspondence Address:
KNOBBE MARTENS OLSON & BEAR LLP
620 NEWPORT CENTER DRIVE
SIXTEENTH FLOOR
NEWPORT BEACH, CA 92660 (US)

(21) Appl. No.: **10/072,317**

(22) Filed: **Feb. 6, 2002**

Related U.S. Application Data

(60) Provisional application No. 60/266,908, filed on Feb. 6, 2001.

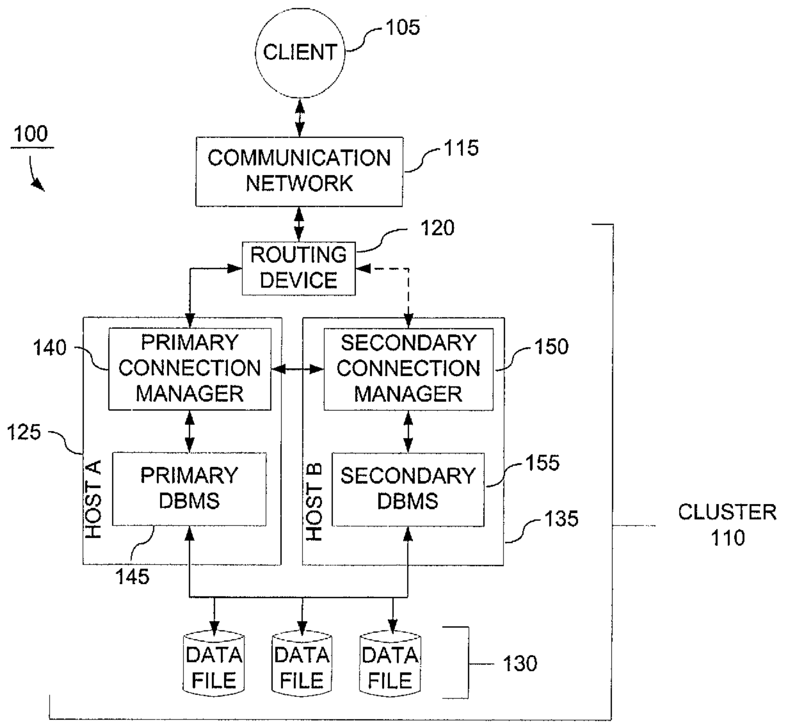
Publication Classification

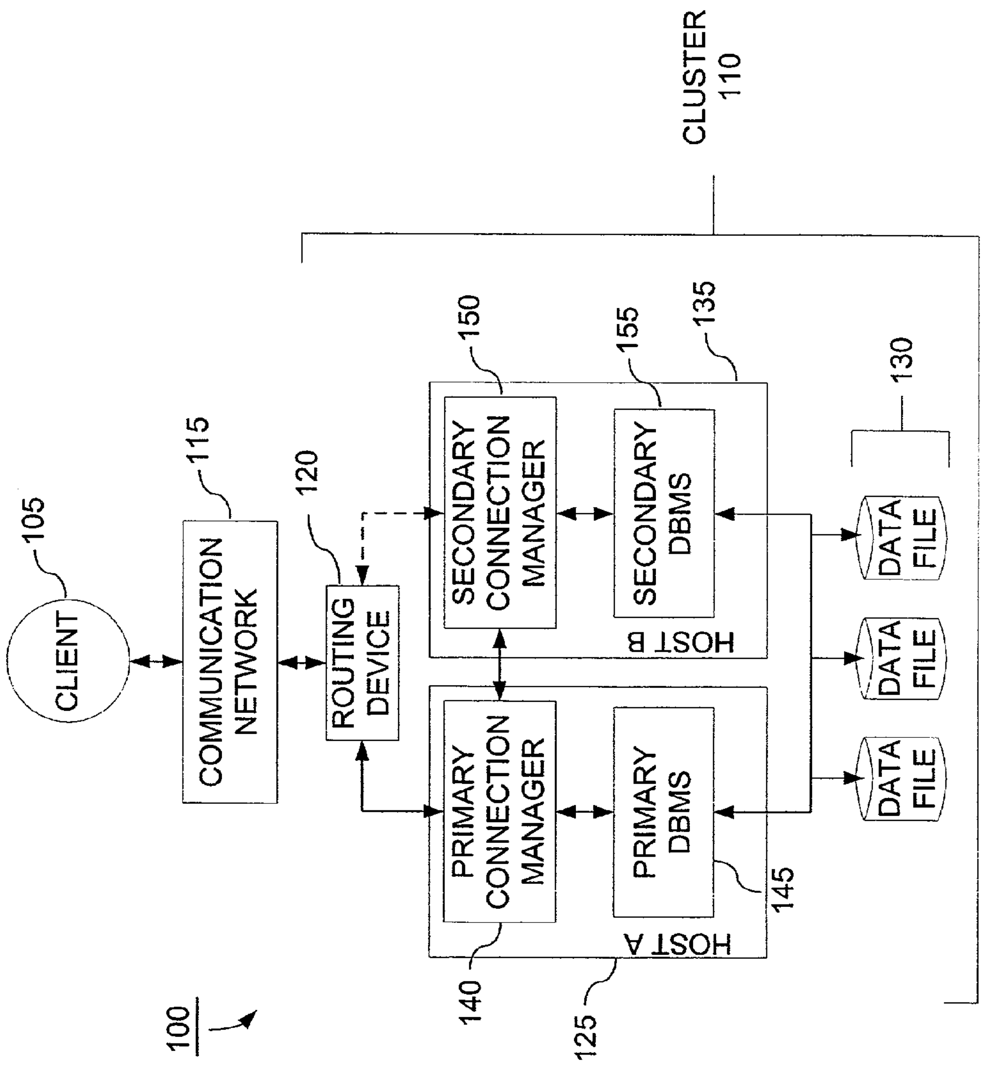
(51) **Int. Cl.**[7] **G06F 15/173**; G06F 17/30; G06F 7/00

(52) **U.S. Cl.** **709/225**; 707/10

(57) **ABSTRACT**

Aspects of embodiments of the present disclosure include a highly available database cluster that can maintain a connection with potentially geographically remote client application programs, including non-fault tolerant application programs, even in the event of one of the database management systems (DBMS) of the cluster becoming unavailable. For example, the database cluster can advantageously move a client connection between a failing, unbalanced, or overloaded DBMS, to another DBMS within the cluster. The database cluster can include connection managers that monitor a connection between a client application program and a primary DBMS. When one connection manager determines that the primary DBMS is unavailable, the connection manager of a secondary DBMS can assume the connection to the client application as if it were the primary DBMS. The connection manager can finish all open transactions, thereby avoiding the need to roll back the same. Moreover, the connection managers can monitor the connection at the DBMS communication level, such as, for example, the SQL*Net level.





100
CLIENT
105
COMMUNICATION NETWORK
115
ROUTING DEVICE
120
CLUSTER 110
PRIMARY CONNECTION MANAGER
140
PRIMARY DBMS
145
HOST A
125
SECONDARY CONNECTION MANAGER
150
SECONDARY DBMS
155
HOST B
135
DATA FILE
DATA FILE
DATA FILE
130

FIG. 1

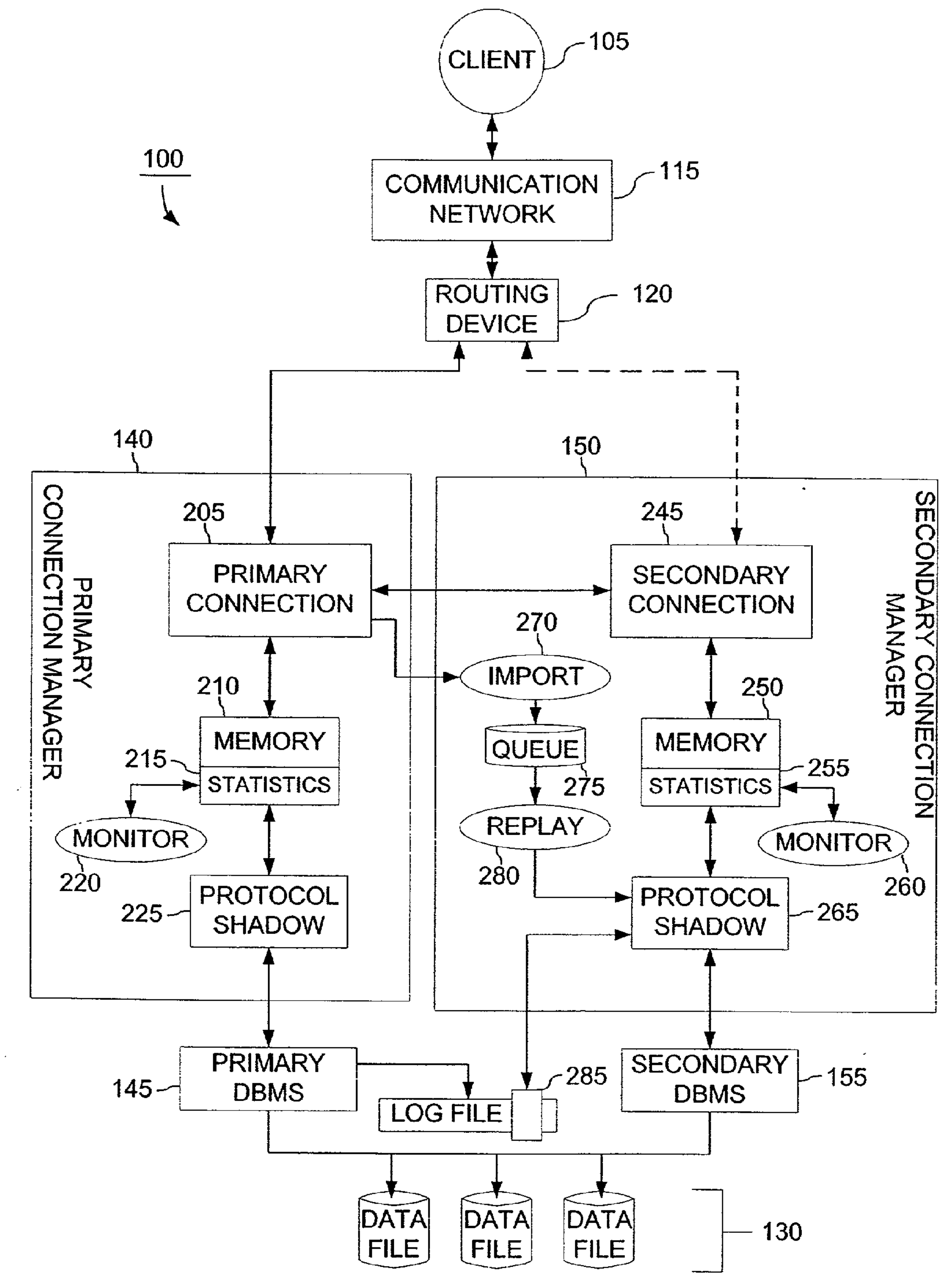

CLIENT
105
100
COMMUNICATION NETWORK
115
ROUTING DEVICE
120
140
150
PRIMARY CONNECTION MANAGER
SECONDARY CONNECTION MANAGER
205
PRIMARY CONNECTION
245
SECONDARY CONNECTION
270
IMPORT
210
MEMORY
250
215
STATISTICS
255
QUEUE
275
MONITOR
220
REPLAY
280
MONITOR
260
PROTOCOL SHADOW
225
PROTOCOL SHADOW
265
PRIMARY DBMS
145
LOG FILE
285
SECONDARY DBMS
155
DATA FILE
DATA FILE
DATA FILE
130

FIG. 2

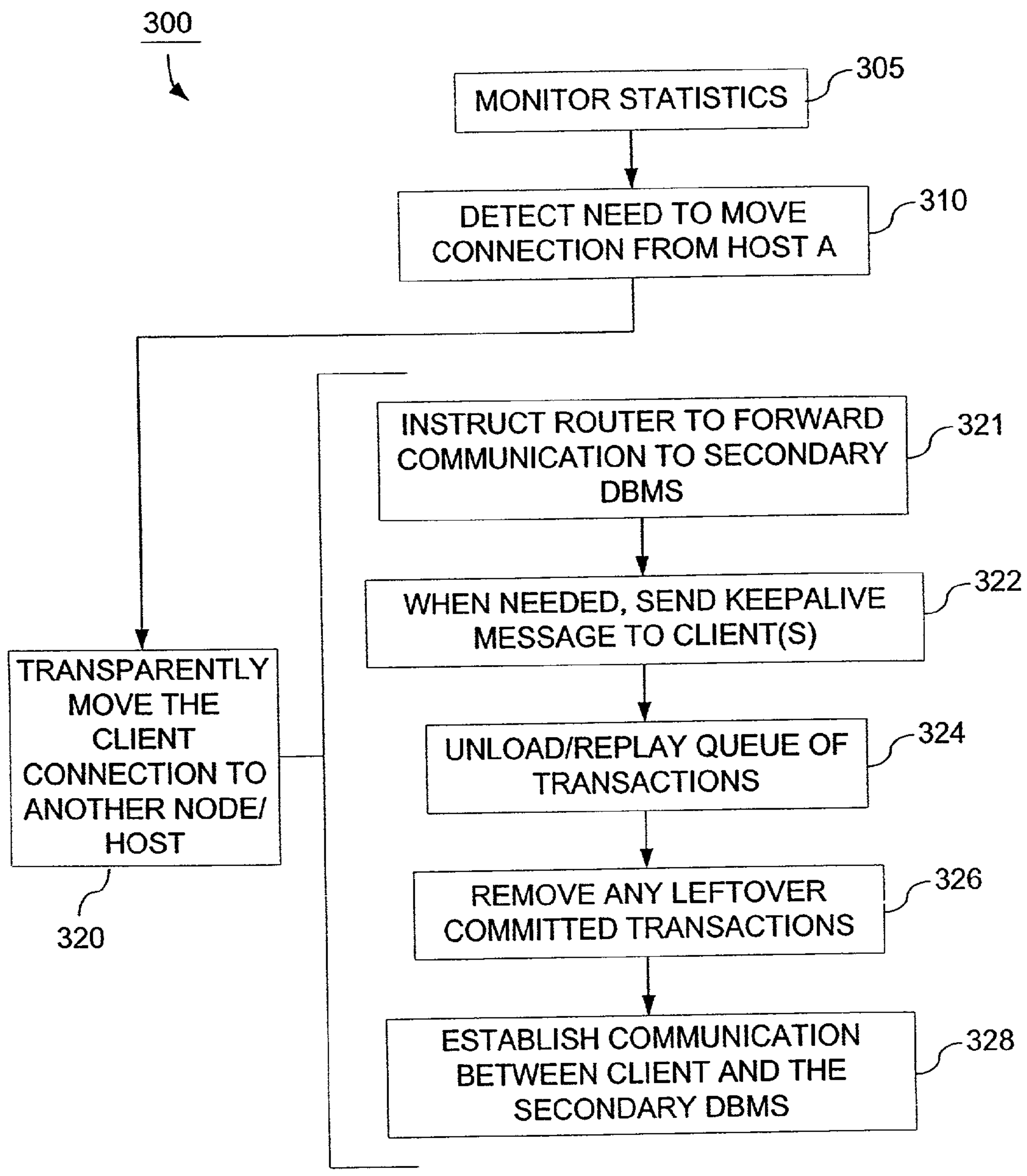

300
MONITOR STATISTICS
305
DETECT NEED TO MOVE CONNECTION FROM HOST A
310
TRANSPARENTLY MOVE THE CLIENT CONNECTION TO ANOTHER NODE/ HOST
320
INSTRUCT ROUTER TO FORWARD COMMUNICATION TO SECONDARY DBMS
321
WHEN NEEDED, SEND KEEPALIVE MESSAGE TO CLIENT(S)
322
UNLOAD/REPLAY QUEUE OF TRANSACTIONS
324
REMOVE ANY LEFTOVER COMMITTED TRANSACTIONS
326
ESTABLISH COMMUNICATION BETWEEN CLIENT AND THE SECONDARY DBMS
328

FIG. 3

HIGHLY AVAILABLE DATABASE CLUSTERS THAT MOVE CLIENT CONNECTIONS BETWEEN HOSTS

REFERENCE TO RELATED APPLICATION

[0001] The present application claims priority benefit under 35 U.S.C. §119(*e*) from U.S. Provisional Application No. 60/266,908, filed Feb. 6, 2001, entitled "HIGHLY AVAILABLE DATABASE CLUSTERS," which is incorporated herein by reference.

FIELD OF THE INVENTION

[0002] The present invention relates to the field of highly available database clusters. More specifically, the invention relates to database clusters that transparently move client connections between hosts.

BACKGROUND OF THE INVENTION

[0003] A database is generally considered to be a collection of information or data organized in a way that computer programs can quickly access or select desired portions of the collection. A database management system (DBMS) includes the collection of computer programs that enable the quick storage, selection, modification, and extraction of desired portions of data from the database. Exemplary DBMSs include those commercially available from Oracle Corporation, IBM, or the like. Application programs, on the other hand, typically include client programs that connect to a DBMS to provide users the ability to interact with the data of the database, such as, for example, to select, modify, organize, delete, or the like, some or all of the foregoing data. Exemplary application programs include payroll or inventory programs, online stores, or the like.

[0004] Often, the application programs are designed to be continually connected to a DBMS, thereby having substantially continuous access to data stored within the same. Unless specifically coded to recover, these application programs typically fail when their connection to the DBMS fails or is otherwise unavailable, such as during a system failure. For many application program environments, this failure is undesirable.

[0005] System designers have created various solutions to reduce the effects of an application program losing a connection to a DBMS. For example, system designers often employ database clusters to offer backup solutions to failed systems. Database clusters can include two or more DBMSs accessing shared data files. For example, the shared data files can include data files having the same set of data from the replication of changes from one DBMS to another. Also, the shared data files can include multiple DBMSs that access the same physical storage. Through the shared data files, system designers allow one DBMS to replace another in the event of a failure.

[0006] There are several drawbacks associated with the foregoing database clustering solution, especially when employed in environments allowing for little or no down time, such as, for example, high availability solutions. For example, when a DBMS fails, the connection from the application program to the DBMS can be lost, thereby potentially losing all open transactions from the same. Additionally, data not replicated from a failing DBMS can be lost. Moreover, during load balancing, simultaneous updates of the same data on different DBMSs can occur in some replication solutions. Also, a large amount of communication traffic among a cluster, and/or hardware limitations of the same, can reduce the cost effectiveness of geographically diverse systems. Moreover, as discussed, the failure of an individual DBMS results in a failure of non-fault tolerant program applications.

[0007] On the other hand, system designers may also employ application servers in order to reduce the effects of losing a connection to a DBMS. For example, system designers often have application programs connect to an application server, where the application server includes the functionality to recover lost client connections to one or more secondary DBMSs within a database cluster. However, the application server generally includes a proprietary protocol used in communications from the application program to the application server. The proprietary protocol is generally not native to the DBMS and therefore, each connecting application program will first be routed through the application server. Thus, the application server solution is not well suited for geographically diverse storage systems.

[0008] Embodiments of the present invention seek to overcome some or all of these and other problems.

SUMMARY OF THE INVENTION

[0009] Therefore, a need exists for a database cluster that can maintain a connection with potentially geographically remote client application programs, including non-fault tolerant application programs, even in the event of a failure or other unavailability of the primary DBMS. Accordingly, aspects of embodiments of the present disclosure include a highly available database cluster that can maintain a connection with potentially geographically remote client application programs, including non-fault tolerant application programs. For example, the database cluster can advantageously move a client connection between a failing, unbalanced ,or overloaded DBMS, to another DBMS within the database cluster.

[0010] According to one embodiment, the database cluster includes connection managers which monitor a connection between a client application program and a primary DBMS. When one connection manager determines that the primary DBMS is unavailable, has an unbalanced share of the workload of the cluster, or the like, the connection manager of a secondary DBMS can assume the connection to the client application as if it were the primary DBMS. For example, the connection manager can finish all open transactions, thus avoiding the need to roll back the same. Embodiments of the connection managers can also monitor the connection at the DBMS communication level, such as, for example, the SQL*Net level. According to one embodiment, the connection managers capture enough information about the connection to restore the connection to its current state on another DBMS in the cluster.

[0011] Based on the foregoing, an aspect of an embodiment of the invention includes a data processing system comprising a database cluster which can move a connection between a remote client and a first DBMS within the cluster to a second DBMS within the cluster when the database cluster determines that the first DBMS has failed, wherein the movement of the connection is transparent to the remote client and the connection includes communication in a protocol native to the first and second DBMSs, such as, for example, SQL*Net.

[0012] Another aspect of an embodiment of the invention includes a data processing system comprising a database cluster which can move a connection between a remote client and a first DBMS to a second DBMS when the database cluster determines that the first DBMS is executing an unbalanced portion of the cluster workload, wherein the movement of the connection is transparent to the remote client.

[0013] Another aspect of an embodiment of the invention includes a method of moving a client connection from a first DBMS to a second DBMS. The method comprises monitoring a state of a client connection to a first DBMS, wherein the client connection includes communication in a protocol native to the first DBMS. The method also comprises detecting a condition of the connection which indicates the connection should be moved, and moving the client connection to a second DBMS without the client dropping the client connection.

[0014] For purposes of summarizing the invention, certain aspects, advantages and novel features of the invention have been described herein. Of course, it is to be understood that not necessarily all such aspects, advantages or features will be embodied in any particular embodiment of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

[0015] A general architecture that implements the various features of the invention will now be described with reference to the drawings. The drawings and the associated descriptions are provided to illustrate embodiments of the invention and not to limit the scope of the invention. Throughout the drawings, reference numbers are re-used to indicate correspondence between referenced elements. In addition, the first digit of each reference number indicates the figure in which the element first appears.

[0016] **FIG. 1** illustrates a block diagram of an exemplary data processing system including a database cluster according to embodiments of the invention.

[0017] **FIG. 2** illustrates a block diagram of exemplary connection managers of the database cluster of **FIG. 1**, according to embodiments of the invention.

[0018] **FIG. 3** illustrates a flow chart of a fail-over process, according to embodiments of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

[0019] Aspects of embodiments of the present disclosure include a highly available database cluster that can move connections with one or more client program applications from a first host to a second in the event of, for example, a failure of the first host, an unbalanced or overloaded workload present on the first host, or the like. Additionally, the cluster can provide communication in the native protocol of underlying database management systems (DBMSs), thereby providing fault tolerant connections for potentially geographically remote and potentially non-fault tolerant client application programs.

[0020] According to one embodiment, the database cluster includes connection managers that monitor a connection between a client application program and a primary DBMS. When one connection manager determines that the primary DBMS is unavailable, has an unbalanced share of the workload of the cluster, or the like, the connection manager of a secondary DBMS can assume the connection to the client application as if it were the primary DBMS. In one embodiment, the assumption of the connection by the secondary connection manager is transparent to the client. Moreover, the secondary connection manager can replay or finish all open transactions, thus picking up the connection to the client in a state exactly where the primary DBMS failed. Embodiments of the connection managers can also monitor a connection at the DBMS communication level, such as, for example, the SQL*Net level. According to one embodiment, the connection managers capture enough information about the connection to restore the connection to its current state on another DBMS in the cluster.

[0021] According to one embodiment, the connection manager monitors a current state of TCP and IP protocols of a TCP/IP connection. When one connection manager determines that it should assume the TCP/IP connection, the connection manager continues the TCP conversation that the client originally started with the other connection manager.

[0022] The foregoing solution advantageously provides a database cluster offering high availability to its connecting clients, including non-fault tolerant clients, by moving connections between DBMSs within a database cluster.

[0023] To facilitate a complete understanding of the invention, the remainder of the detailed description describes the invention with reference to the drawings, wherein like reference numbers are referenced with like numerals throughout.

[0024] **FIG. 1** illustrates a block diagram of an exemplary data processing system **100**, according to an embodiment of the invention. As shown in **FIG. 1**, the data processing system **100** includes a client application program **105** (client **105**) communicating with a highly available database cluster **110** (cluster **110**) through a communication network **115**. According to one embodiment, the client **105** comprises computer programs such as payroll or inventory programs, online stores, human resource applications, or the like, executing on one or more remote computer devices or systems.

[0025] An artisan will recognize from the disclosure herein that the client **105** can comprise virtually any client program designed to connect with a DBMS to interact with data stored therein, such as, for example, to select, modify, organize, delete, index, or the like, some or all of the foregoing data. Moreover, the artisan will recognize from the disclosure herein that the client **105** can execute on a wide variety of computer devices, such as, for example, personal digital assistants, mobile telephones, handheld computer devices, laptop computers, workstations, mainframe computers, combinations of the same, or the like.

[0026] The cluster **110** can comprise two or more DBMSs, able to access portions of shared, replicated, or otherwise mirrored data. Exemplary DBMSs include those commercially available from Oracle Corporation, IBM, or the like. According to one embodiment, the DBMSs of the cluster **110** execute on one or more hosts or other computing devices.

[0027] The communication network **115** comprises some or all of the Internet. However, an artisan will recognize from the disclosure herein that the communications network **115** can include a wide range of interactive communications mediums. For example, the communications network **115** can include interactive television networks, telephone networks, wireless data transmission systems, two-way cable systems, customized computer networks, interactive kiosk networks, automatic teller machine networks, direct links, private local or wide area networks, and the like.

[0028] In one embodiment, the client **105** connects to the cluster **110** through the communication network **115**. The client **105** issues instructions or transactions including one or more operational statements to be carried out against data stored in data files accessible by the cluster **110**. When the cluster **110** has executed the instructions or transactions, the cluster **110** returns an indication of the same to the client **105**. Moreover, the cluster **110** can move the foregoing connection with the client **105** from a first host to a second in the event of, for example, a failure of the first host, an unbalanced or overloaded workload present on the first host, or the like. Additionally, the cluster **110** can provide communication in the native protocol of the underlying two or more DBMSs, thereby providing fault tolerant connections for the potentially geographically remote and potentially non-fault tolerant client **105**. For example, the cluster **110** can monitor a connection at the DBMS communication level, such as, for example, a SQL*Net level. The cluster **110** can capture enough information about the connection to restore the connection to its current state on another DBMS in the cluster.

[0029] **FIG. 1** also shows the cluster **110** including a routing device **120** communicating with a primary host **125** (Host A **125**) to execute transactions against one or more shared data files **130**. Additionally, **FIG. 1** shows the routing device **120** having the ability to communicate with a secondary host **135** (Host B **135**), which in turn also includes the ability to execute transactions against the one or more shared data files **130**. According to one embodiment, Host A **125** includes a primary connection manager **140** and a primary DBMS **145**, while Host B **135** includes a secondary connection manager **150** and a secondary DBMS **155**. **FIG. 1** also shows the primary connection manager **140** communicating with the secondary connection manager **150**.

[0030] Routing device **120** comprises a device, such as, for example, a router, hub, or the like, that connects any number of computing systems or networks. Generally, routing device **120** uses information in data packets, along with a forwarding table to determine where the data packets go. According to one embodiment, the routing device **120** is configured in such as fashion as to forward all packets destined for the database cluster **110** to both the primary connection manager **140** and the secondary connection manager **150**. An artisan will recognize that the function of such routing will be to enable a virtual IP address (VIP) that may be shared between hosts.

[0031] In one embodiment, the routing device **120** sends all data packets from the client **120** to both the primary connection manager **140** and the secondary connection manager **150**. The secondary connection manager monitors statistics related to, for example, the number of clients connected to the primary connection manger. The primary connection manager assumes responsibility for the data packets send from the client **105** to the primary DBMS **145**. Thus, when the client **105** sends transactions, in the form of data packets, through the communication network **115** to the primary DBMS **145**, the data packets are routed to the primary connection manager **140**, forming a connection between the primary connection manager **140** and the client **105**. The primary connection manager **140** then forwards the data packets to the primary DBMS **145**, forwards a copy of the data packets to the secondary connection manager **150**, and monitors statistics related to, for example, the number of connected clients and the status of the secondary connection manager **150**. Meanwhile, the secondary connection manager **150** receives the copied data packets, holds them in memory, and monitors statistics related to, for example, the number of connected clients and the status of the primary connection manager **140**.

[0032] The primary DBMS **145** receives the data packets from the primary connection manager **140**, assembles them into operational statements of transactions, and executes the same against the data files **130**. The primary DBMS **145** then returns the requested data and/or acknowledgment of the received data packets back to the primary connection manager **140**, which in turns forwards a copy to the secondary connection manager **150** and a copy to the respective client **105** through the communication network **115**.

[0033] In an embodiment, the secondary connection manager **150** can detect a condition of the connection between the primary connection manager **140** and the client **105** from the statistics being monitored. For example, the secondary connection manager **150** can detect a failure of the connection, an unbalanced or overloaded workload on the primary connection, or the like. In such circumstances, the secondary connection manager assumes control of the connection and replays any rolled back transactions against the data files **130** through the secondary DBMS **155** as follows.

[0034] The secondary connection manager **150** communicates with the routing device **120** to acknowledge TCP requests from the client **105** to the primary connection manager **140**. These acknowledgements advantageously keep the client TCP connection from timing out and failing. Additionally, the secondary connection manager **150** replays any operational statements of transactions rolled back due to, for example, the failure of the primary connection. As is generally known in the art, upon failure of a DBMS, all operational statements of open transactions (for, example, non-committed transactions) executed against the data files **130** are rolled back as if they never occurred. However, because the operational statements of open transactions are stored in the foregoing memory of the secondary connection manager **150**, these operational statements from open transactions can be reexecuted against the data files **130** through the secondary DBMS **155**. After replaying the foregoing operational statements, the secondary connection manager **150** begins forwarding data packets from the client **105** to the secondary DBMS **155** to be executed against the data files **130**.

[0035] Based on the foregoing disclosure, the database cluster **110** advantageously moves a connection between the primary DBMS **145** and the client **105** to the secondary DBMS **155** in the cluster **110**, when the primary DBMS **145** fails, becomes unbalanced, overloaded, or the like. Additionally, the database cluster **110** advantageously replays any rolled back statements of open transactions during fail-over to the secondary DBMS **155**, thereby providing an assumption of the connection that is transparent to the client **105**. Accordingly, the cluster **110** avoids failure of non-fault tolerant clients by moving the connection rather than allowing it to fail. Additionally, the cluster **110** advantageously provides communication in the native protocol of the underlying two or more DBMSs, thereby providing fault tolerant connections for the potentially geographically remote and potentially non-fault tolerant client **105**.

[**0036**] **FIG. 2** illustrates a block diagram of embodiments of the primary and secondary connection managers, **140** and **150**, of the cluster **110**, according to embodiments of the invention. **FIG. 2** shows the primary connection manager **140** including a primary connection **205** communicating with a memory **210** including statistics **215**, a monitor process **220** also communicating with the memory **210**, and a protocol shadow **225** communicating with the memory **210** and the primary DBMS **145**. Moreover, **FIG. 2** shows the secondary connection manager **150** including a secondary connection **245** communicating with a memory **250** including statistics **255**, a monitor process **260** also communicating with the memory **250**, and a protocol shadow **265** communicating with the memory **250** and the secondary DBMS **155**. In addition, the secondary connection manager **150** includes an import process **270** communicating with the primary connection **205** and a queue **275**. The secondary connection manager **150** also includes a replay process **280** communicating with the queue **275** and the protocol shadow **265**. Moreover, while not shown, an additional redo monitor can access one or more log files **285** associated with the primary DBMS **145**. The redo monitor also can communicate with the memory **210** and review the statistics **215**. **FIG. 2** also shows the protocol shadow **265** accessing the one or more log files **285**.

[**0037**] The following simplified exemplary transactions are disclosed to provide an understanding of the operation of the primary and secondary connection managers, **140** and **150** respectively, however, they are not intended to limit the scope of the disclosure. Rather, an artisan will recognize from the disclosure herein, alternative arrangements to simplify or expand one or more of the features or aspects disclosed herein.

Normal Operation

[**0038**] When the client **105** begins a transaction by issuing an operational statement to be applied against the data files **130**, the client **105** distributes the statement across one or more data packets. The data packets are forwarded through the communication network **115** to the routing device **120**, where, as disclosed, the routing device **120** forwards the packets to the primary connection **205** and to the secondary connection **245**. The primary connection **205** examines statistics in the statistics **215** generated by the redo monitor. These statistics include, for example, the current location of transaction being stored in the log files **285**. The primary connection transmits a copy of each data packet along with the current log file location, such as a sequence number, to the import process **270** of the secondary connection manager **150**, and places a copy in the memory **210**. The import process **270** stores the data packets in the queue **275**. The protocol shadow **225** accesses the memory **210** and retrieves the data packets. The protocol shadow **225** forwards the packets to the primary DBMS **145**, where the packets are assembled and the operational statement executed against the data files **130**. Moreover, as is generally known in the art, the DBMS can also keep a record or log of the executed statement, generally in the log file **285**.

[**0039**] The DBMS **145** forwards a result of the statement and/or and acknowledgement of receipt of the same, back to the protocol shadow **225**, preferably in one or more acknowledgement data packets. The protocol shadow **225** transfers the data packets back to the memory **210**, where they are picked up by the primary connection **205**. The primary connection **205** forwards a copy of the data packets to the import process **270** and to the client **105**. Thus, the client **105** receives the results and/or acknowledgement of the transmitted statement of an open transaction.

[**0040**] The client **105** may then desire to finalize, or commit the transaction against the data files **130**. In such case, the client **105** issues a commit statement, which is forwarded to the primary DBMS **145** and the import process **270**, along with the subsequent result and/or acknowledgement, in a manner similar to that disclosed. In one embodiment, the protocol shadow **225** stores sufficient data from the data packets that it can assemble the statements of a given transaction. When the protocol shadow **225** determines the data packets for a commit statement have been sent to the primary DBMS **145**, the protocol shadow attaches a marker to the result/acknowledgement data packets associated with the primary DBMS **145** acknowledging execution of the commit statement. According to one embodiment, the marker comprises a location marker, such as, for example, a sequence number from the primary DBMS **145**. Then, as disclosed, the result/acknowledgement data packets are transmitted with their marker to the import process **270**. According to one embodiment, the import process **270** recognizes the marker placed on the data packets associated with the commit statement, and recognizes that the entire transaction has been executed by the primary DBMS **145** against the data files **130**. Therefore, the import process **270** deletes the data packets associated with the now finalized transaction from the queue **275**.

[**0041**] Based on the foregoing, the protocol shadow **225** and the import process **270** advantageously work together to ensure that only the data packets associated with open transactions remain in the queue **275**.

[**0042**] The primary connection **205** also stores the statistics **215** related to the connection with the client **105** in the memory **210**. In one embodiment, the statistics include sufficient information for the monitor process **220** to determine whether the primary connection **205** has failed, is processing an unbalanced or overloaded workload, or the like, and whether the secondary connection **245** has failed, is processing an unbalanced or overloaded workload, or the like. For example, the statistics **215** can include the number of clients seen by the primary connection **205**, the number of clients seen by the secondary connection **245**, the status of communication with secondary communication manager **150**, or the like. The primary connection **205** acquires the statistics **215** corresponding to information from the secondary connection manager **150** through the connection between the primary connection **205** and the secondary connection **245**. Moreover, according to one embodiment, the foregoing status of the secondary communication manger **150** can be ascertained through straightforward ping or ping-like commands.

Fail-Over

[**0043**] **FIG. 3** illustrates a flow chart of a fail-over process **300**, according to embodiments of the invention. As shown in **FIG. 3**, the fail-over process **300** begins with BLOCK **305** where the cluster **110** monitors the statistics of one or more connections with one or more clients. In the foregoing example, the monitoring corresponds to the monitor processes **220** and **260**. In BLOCK **310**, the cluster **110** detects the need to move the connection from one DBMS to another. For example, the monitor **260** may determine that the primary DBMS **145** has failed, become unbalanced, overloaded, or the like, and determine that the secondary connection manager **150** should assume the connection with the client **105**. When the determination that a connection move is desired, the fail-over process **300** proceeds to BLOCK **320**, where the cluster **110** moves the connection from one DBMS to another without losing the connection or causing even a non-fault tolerant client to fail. For example, the secondary connection **245** can communicate with the routing device **120** to assume the IP address (or VIP) of the primary DBMS **145**. Additionally, the secondary connection manager **150** can replay all statements of open transactions which were rolled back in the data files **130**. Accordingly, the move is transparent to the client **105** who does not lose the connection and does not know that a change has been made.

[**0044**] According to one embodiment, BLOCK **320** can include SUBBLOCK **321**, where the cluster **110** instructs the routing device **120** to forward communication from the client to another DBMS. For example, as disclosed, the secondary connection **245** can assume the IP address of the primary DBMS **145**. BLOCK **320** can also include SUBBLOCK **322**, where the cluster **110** can send a keepalive message to one or more clients to ensure against failure of the connection to the same. According to one embodiment, the client **105** resends data packets which are not responded to or otherwise acknowledged by the cluster **110**. When the client **105** resends the same data packets a predetermined amount of times, the client **105** may register a failure of the connection, thereby causing non-fault tolerant clients (such as those clients not programmed to recover) to also fail. Thus, during the fail-over process **300**, the cluster **110** can respond to the client **105** with a message or acknowledgement that keeps the client **105** from resending the same data packets, therefore keeping the client from determining that the connection has failed. According to one embodiment, the secondary connection **245** sends the foregoing keepalive messages.

[**0045**] BLOCK **320** of the fail-over process **300** can also include SUBBLOCK **324** where the cluster **110** replays any statements from open transactions that were rolled back during the failure of the primary DBMS **145**. For example, the replay process **280** can access the queue **275** to retrieve data packets associated with rolled back transactions and to forward them to the protocol shadow **265**. For example, as disclosed in the foregoing, the import process **270** removes the statements associated with all finalized or committed transactions, thereby leaving only rolled back transactions in the queue **275**.

[**0046**] BLOCK **320** of the fail-over process **300** can also include SUBBLOCK **326** where the cluster **110** removes any leftover committed transactions that may have slipped through. For example, it is possible that Host A **125** can fail after the primary DBMS **145** executes a commit statement for a particular transaction, but before the result/acknowledgement of the same can be transmitted to the import process **270**. Thus, the secondary connection manager **150** believes the statements associated with the foregoing transaction were rolled back, e.g., because they were left in the queue **275**, and therefore, the replay process **280** will forward the already committed statements to the protocol shadow **265**. In one embodiment, the protocol shadow **265** parses the log file **285** of the primary DBMS **145** to ensure a commit statement associated with the open transaction was not received. When the protocol shadow **265** determines that a commit statement was received, the protocol shadow **265** deletes the statements associated therewith before their associated data packets are forwarded to the secondary DBMS **155** to be executed against the data files **130**.

[**0047**] BLOCK **320** of the fail-over process **300** can also include SUBBLOCK **328** where the cluster **110** establishes communication between the client and the secondary DBMS. For example, after all rolled back statements are either executed against the data files **130** through the secondary DBMS **155** or deleted from the queue **275** by the protocol shadow **265**, the protocol shadow **265** begins accessing new data packets stored in the memory **250** by the secondary connection **245** after it assumed the connection to the client **105** from the primary connection manager **140**. Thus, after bringing the secondary DBMS **155** back up to the point of failure of the primary DBMS **145**, the secondary connection manager **150** performs operations similar to the normal operations of the primary connection manager **140** as disclosed above.

[**0048**] According to one embodiment, the system administrator of the database cluster **110** can designate whether the secondary connection manager **150** through the monitor process **260** fails-back to the primary connection manager **140** after the cause of failure of the same is repaired, or whether the secondary connection manager **150** simply becomes the primary and vice versa.

[**0049**] Although the foregoing invention has been described in terms of certain preferred embodiments, other embodiments will be apparent to those of ordinary skill in the art from the disclosure herein. For example, the data packets captured from the primary connection manager **140** can be replicated to other DBMSs by replaying the same on the other DBMSs. This replication has several advantages over other replication techniques including a potential reduction in the traffic keeping the database cluster synchronized, thereby advantageously providing economical replication of geographically diverse data files.

[**0050**] The captured data packets can also be used to assist a transaction log based replication system. For example, the data packets can be directed to the other databases in the cluster prior to committing the transactions. Accordingly, committed transactions on a particular DBMS are not lost when the DBMS fails, as these transactions may advantageously be replayed on the other DBMSs in the cluster.

[**0051**] The captured data packets can also be used to assist a transaction log based replication system when posting replicated modifications. Some modifications (such as a vertical table update or DDL operation) may be difficult to replicate via a log-based replication. When the original data packets are available, posting the original SQL rather than the data from the transaction log may be more efficient and straightforward.

[**0052**] According to another embodiment, software may be added just below the client **105**, thereby providing a mechanism to replay incomplete transactions. For example, a typical client application does not access the database directly, but instead uses some type of intermediate layer such as ODBC or JDBC, OCI, or the like. The foregoing added software can advantageously replace this intermediate layer.

[**0053**] Additionally, other combinations, omissions, substitutions and modifications will be apparent to the skilled artisan in view of the disclosure herein. Accordingly, the present invention is not intended to be limited by the reaction of the preferred embodiments, but is to be defined by reference to the appended claims.

[**0054**] Additionally, all publications, patents, and patent applications mentioned in this specification are herein incorporated by reference to the same extent as if each individual publication, patent, or patent application was specifically and individually indicated to be incorporated by reference.

What is claimed is:

1. A database cluster which avoids client failure by connecting to multiple nodes of the cluster, the database cluster comprising:

a first computing system including:

a primary connection manager which forms a client connection with and receives transactions from at least one client, and

a primary database management system (DBMS) which communicates with the primary connection manager to receive the transactions and executes the transactions on data stored in one or more data files; and

a second computing system including:

a secondary connection manager, and

a secondary DBMS which communicates with the secondary connection manager and can access data stored in the one or more data files,

wherein when the second connection manager determines that a predetermined condition is met, the second connection manager receives data from the client connection, replays incomplete portions of open transactions on the data through the secondary DBMS, and begins to receive additional transactions from the at least one client to be executed against the one or more data files.

2. The highly available database cluster of claim 1, wherein the predetermined condition comprises a failure of the first computing system.

3. The highly available database cluster of claim 1, wherein the predetermined condition comprises a failure of the primary DBMS.

4. The highly available database cluster of claim 1, wherein the predetermined condition comprises an unbalanced workload between the first and second computing systems.

5. The highly available database cluster of claim 1, wherein the primary connection manager and the secondary connection manager communicate with one another.

6. The highly available database cluster of claim 5, wherein the primary connection manager transmits copies to the secondary connection manager of data packets which include the transactions and responses or acknowledgements to the transactions.

7. The highly available database cluster of claim 5, wherein the primary connection manager and the secondary connection manager exchange statistics in order to monitor the client connection.

8. The highly available database cluster of claim 7, wherein the statistics include the number of clients connected to the primary connection manager.

9. The highly available database cluster of claim 7, wherein the statistics include the number of clients the secondary connection manager can see connected to the primary connection manager.

10. The highly available database cluster of claim 7, wherein the statistics include whether the secondary connection manager can communicate with the primary connection manager.

11. A primary and at least one secondary connection manager of a database cluster, which manage a connection between at least one client and two or more database management systems (DBMSs), wherein the primary and at least one secondary connection manager can move the connection from the primary connection manager to the at least one secondary connection manager while providing protocols for the connection native to the two or more DBMSs, the primary and secondary connection manager comprising:

a first memory;

a primary connection configured to form a connection with a client and to place statements from transactions from the client into the first memory;

a primary protocol shadow configured to retrieve the statements and forward the statements to a primary DBMS;

a secondary memory;

a secondary connection configured to receive transactions from the connection with the client when one or more predetermined conditions are met and to place new statements from the transactions from the client into the second memory;

at least one process configured to replay any incomplete statements of open transactions; and

a secondary protocol shadow configured to connect to the at least one process until the incomplete statements are forwarded to a secondary DBMS and then to connect to the secondary memory to retrieve the new statements and forward the new statements to the secondary DBMS.

12. The primary and at least one secondary connection manager of claim 11, wherein the protocol native to the two or more DBMSs comprises SQL*Net.

13. The primary and at least one secondary connection manager of claim 11, wherein the at least one process further comprises:

an import process configured to retrieve the statements from the primary connection and store those statements associated with open transactions; and

a replay process configured to access the stored statements and to forward the stored statements to the secondary protocol shadow.

14. The primary and at least one secondary connection manager of claim 11, wherein the secondary protocol shadow is configured to access a log file of the primary DBMS to ensure against replaying of statements of closed transactions.

15. The primary and at least one secondary connection manager of claim 11, wherein the primary and secondary connections communicate with one another.

16. The primary and at least one secondary connection manager of claim 15, wherein the primary connection and the secondary connection exchange statistics in order to monitor the connection.

17. A method of providing native protocol access and transparent fail-over to a client connection thereby avoiding a client failure when a primary host fails, the method comprising:

rerouting a client connection between a first host and a client to a second host;

replaying at least one statement from open transactions, wherein the at least one statement includes a statement received but not committed by the first host when the client connection was moved from the first host; and

establishing communication between the second host and the client over the client connection.

18. The method of claim 17, further comprising sending keepalive messages to the client in order to keep the client from dropping the client connection.

19. The method of claim 17, wherein the replaying at least one statement further comprising removing leftover statements of closed transactions.

20. A method of providing transparent fail-over to a client connection thereby avoiding a client failure when a primary database management system DBMS fails, the method comprising:

monitoring statistics of a client connection between a first DBMS and a client;

determining from the statistics a need to move the client connection to a second DBMS while keeping the client connection alive from a perspective of the client;

rerouting the client connection to the second DBMS;

replaying any statements from open transactions rolled back when the client connection was moved from the first DBMS; and

establishing communication between the second DBMS and the client over the client connection.

21. A data processing system which provides transparent fail-over to a client connection, thereby avoiding a client failure when a primary host fails, the data processing system comprising:

a first host configured to accept a client connection from a client;

a connection manager which reroutes the client connection to a second host without recognition by the client; and

a replay process which forwards to the second host at least one incomplete statement from open transactions when the client connection was moved from the first host, wherein the connection manager establishes communication between the second host and the client over the client connection.

22. The data processing system of claim 21, wherein the client communication comprises a protocol native to the primary host.

23. The data processing system of claim 22, wherein the protocol comprises SQL*Net.

* * * * *