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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
10 September 2004 (10.09.2004)

PCT

(10) International Publication Number
WO 2004/077221 A2

(51) International Patent Classification[7]: G06F

(21) International Application Number:
PCT/IN2004/000032

(22) International Filing Date: 29 January 2004 (29.01.2004)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
127/MUM/2003 30 January 2003 (30.01.2003) IN

(71) Applicant *(for all designated States except US)*: **VAMAN TECHNOLOGIES (R & D) LIMITED** [IN/IN]; Pawani Plot, Near Vipul Apartment, Bhakti Marg, Mulund (West), Mumbai 400 080 (IN).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: **RAO, Vinayak, K.** [IN/IN]; A/4 Vishwakarma Jyoti, Subhash Lane, Malad (East), Mumbai 400 097, Maharashtra (IN).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, 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, NA, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

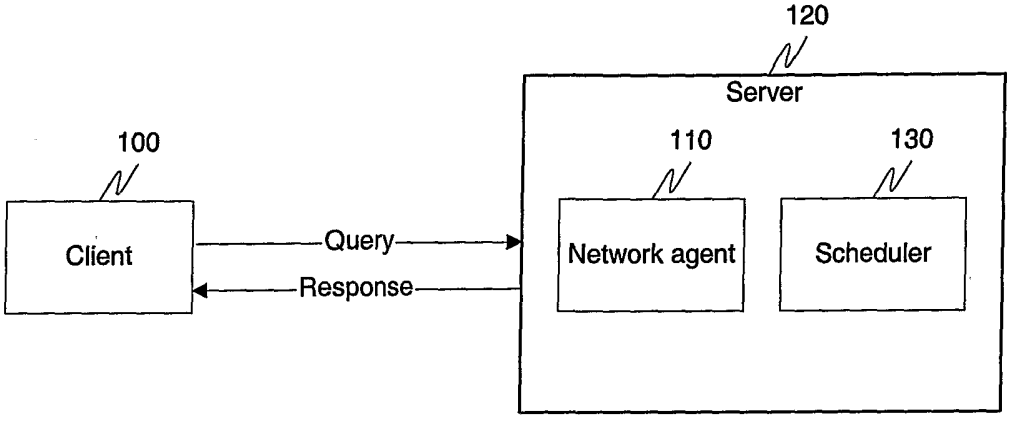
(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC,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

Declarations under Rule 4.17:

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii)) for the following designations AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, 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, NA, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, UZ, VC, VN, YU, ZA, ZM, ZW, ARIPO patent (BW, 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, HU, IE, IT, LU, MC, NL, PT, RO, SE, SI, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG)*

[Continued on next page]

(54) Title: SYSTEM AND METHOD OF CONCURRENT COMMUNICATION BETWEEN A CLIENT AND SERVER IRRESPECTIVE OF INDIVIDUAL FUNCTIONALITIES



(57) Abstract: The present invention relates generally to a system and method for receiving and transmitting a plurality of concurrent requests and responses, irrespective of their functionality, protocol, with optimal utilization of available resources. More particularly the present invention relates to a system and method of reading incoming data from one or a plurality of Clients, such as a web-based client or a mail client or an ODBC-compliant client, irrespective of their protocol and protocol functionality, reading and writing data and establishing a connection with a Server irrespective of its functionality.

WO 2004/077221 A2

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

Published:

— *without international search report and to be republished upon receipt of that report*

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.

TITLE OF INVENTION

System and Method of concurrent communication between a client and server irrespective of individual functionalities

BACKGROUND OF THE INVENTION

A 'Client-Server' computer architecture consists of a class of software, where processing is distributed among one or more information requesters (Clients) and one or more information providers (Servers), as well as in the interfaces (network, protocols, middleware) between them. This approach was developed with a view to overcome the limitations of the mainframe architecture and the file sharing architecture, which were exposed due to the drastic increase in traffic and connectivity. Almost all popular applications on the Internet follow the Client-Server design like the e-mail clients, File Transfer Protocol (FTP) clients and Web servers.

In order for Clients to interact with Servers, they have to communicate in the same protocol. A protocol defines how computers identify one another on a network, the form that the data should take in transit, and how this information is processed once it reaches its final destination. However, there exist problems with regards to communication between Clients and Servers largely due to the existence of numerous vendor specific protocols and types of functionality expected from them. These protocols include TCP/IP (for UNIX, Windows NT, Windows 95 and other platforms), IPX (for Novell NetWare), DECnet (for networking Digital Equipment Corp. computers), AppleTalk (for Macintosh computers), and NetBIOS/NetBEUI (for LAN Manager and Windows NT networks). In addition, FTP, HTTP, HTTPS, FINGER, SMTP, SOCKS and TELNET are some of the prominent protocols included in the popular TCP/IP Suite.

An attempted solution to this problem can be seen in the HTTP Tunneling method, implemented by Microsoft. In this technique, if a Remote Method Invocation (RMI) fails to make a normal (or SOCKS) connection to the intended server, and it notices that an HTTP proxy server is configured, it will attempt to tunnel RMI requests through that proxy server, one at a time. However, the drawbacks of this technique are that it can only be implemented on an HTTP configured proxy server and hence, does not provide complete compatibility across all protocols used in Client-Server communication. Additionally, using this technique can expose a fairly large security loophole on the Server machine and reduces the bandwidth utilization because of tunneling overheads. Another major disadvantage of HTTP tunneling is that it does not permit multiplexed connections.

The two main types of Client-Server architecture are the two-tier architecture model and the three-tier architecture model. In a 'two-tier architecture", the Client and the Server communicate with each other

directly, while in the case of a "three-tier architecture", the business logic is separated from the user interface. This is the core concept used in developing a multi-tier application.

In order to accomplish this separation, the Server is placed in the middle tier, and the added third tier typically represents data repositories of information that may be accessed by the Server as part of the task of processing the Client's request. A data server normally stores and manages the data that is used by an application, and includes the software products that are used in storing, accessing, and retrieving the data.

In the case of any Client-Server environment, response to requests, which are received from a Client must be returned very quickly. A particular Server may receive thousands of Client requests at any given instant. Receipt of such a number of requests creates a significant amount of overhead to the processing of a Client's request, which may greatly outweigh the time actually spent in completing the transaction that answers the Client's request. This gives rise to the need for a method to manage requests and responses in a more efficient manner because apart from standard database Client requirements one is required to efficiently handle requests and responses between Web Servers, Mail Servers, FTP Servers, Database Server and their Clients.

Generally for any database Server the nature of network Clients is concurrent Open Database Connectivity (ODBC) sessions. As per ODBC standards and default network protocols adapted for database connectivity each session has a default timeout and retry count and each session has a continuous connectivity with the database Server (though they may be pooled by the Server).

In some of the existing database management systems (DBMS) it is possible to provide concurrent connections to a database by using demultiplexed connection environments (See US Patent Application Number 20020042850). In such DBMS, worker agents carry out processes in response to requests made by applications (or Clients). Often agents are associated with applications by an application scheduler or manager for the life of a transaction. To handle the potentially large number of inbound connections being distributed across a smaller number of worker agents, DBMS use schedulers and wait queues. However, this system does not solve the task of processing the concurrent requests in an efficient manner, without overheads and delays and within the protocol timeout.

In another technique, the networking task is carried out with the help of a different listener process for each type of Server, which then communicates to the main Server of any database requests. However, this resulted in a lot of overheads of inter process communication (IPC) and is largely dependent on the operating system (OS) kernel to schedule the IPC mechanism. When there is a burst of requests, which

is often the case in connections between Clients and Web Servers or in the case of unexpected database activity, the listener is unable to handle these high frequency calls and results in a timeout.

In the case of Web Servers, there also currently exist mechanisms to maintain connections, even after completion of processing of the Client request. (See USPTO Patent No. 6,105,067) Existing connections are maintained in a connection pool. When a Client request is received, a connection from the pool is used if a suitable connection is available--thus avoiding the overhead of establishing the connection. (When no connection from the pool can be used, a new connection may be established.) This pool may contain many connections to a single data server, many connections to multiple data servers of a single type (such as multiple DB2 databases), and/or many connections to data servers of different types (for example, DB2 databases, MQSeries message services, etc.). A common interface is used for requesting a connection from the pool, reducing the number of APIs (Application Programming Interfaces) that programmers and support personnel are required to understand and use. Although pool management for backend servers considerably improves performance in a Web environment it does not improve the overall communication efficiency between the Client and the Server while receiving requests and transmitting responses. Pooling might be a disadvantage also because of leaks (that is programming bugs in drivers/bindings/custom code) that are getting eliminated currently because of connections getting recycled constantly.

Accordingly, a need exists for a system and method by which a number of concurrent requests can be received, irrespective of their protocol, from the Client and transmitted to the Server and any number of concurrent responses is transmitted back from the Server to the Client, without the use of significant resources and without compromising in efficiency. In addition, there exists the need for a technique whereby concurrent connections can be established with the server without any limitations and wherein a multitude of requests can all be managed in an efficient manner, with optimum utilization of current hardware

SUMMARY OF THE INVENTION

To meet the foregoing needs, the present invention provides a software-implemented process, system and method for use in a computing environment having a connection to a network. The system typically includes a Server, a Client configured to connect to the Server via a computer network and a Network Agent. The Network Agent is part of the Server and it acts as an interface between the Client and the Server. As a result of being part of the Server, the Network Agent saves up on Inter process communication (IPC), which is an expensive overhead, without compromising on the functionality of a networking device.

The Network Agent is configured to receive requests from the Client and transfer the said requests to the Server. It is further configured to receive responses from the Server and transmit the same to the Client. The Network Agent functions in a manner, which gives flexibility to the user to control and balance the computer resources and achieves efficient response to requests. The Network agent consists of a Socket, which received requests from the Client. Any activity on the Socket triggers an event to the network agent, thus avoiding starvation and supporting concurrency. The Network agent consists of a Master Thread which schedules the tasks performed by the Network Agent. In addition it contains a Sub-Thread and two queues viz., the "Transmit Queue" and the "Receive Queue". The Master Thread uses the Sub Thread to carry out the tasks of receiving requests and transmitting responses between the Client and the Server. The Master Thread polls the Transmit Queue, the receive Queue and the Socket and depending on the largest count, it dynamically allocates resources to manage the requests or responses.

The entire design is based on state machines and modules comprising of various events communication via messages that is it is event driven using Finite State Machine (FSM) concept, the functionality is broken down into a series of events scheduled by kernel.

BRIEF DESCRIPTION OF THE DRAWINGS

The various objects and advantages of the present invention will become apparent to those of ordinary skill in the relevant art after reviewing the following detailed description and accompanying drawings, wherein:

Fig. 1 is a block diagram of a computer system including a Client, a Network Agent and a Server.

Fig 2 is a diagram illustrating typical servers like a Database Server or Web Server or Mail Server having their own network agent capable of making communication between a functional specific server and functional server specific client.

Fig. 3 is a block diagram depicting the internal components of the Network Agent.

Fig. 4 illustrates the manner in which the Network Agent processes requests and responses on the occurrence of any Socket Activity.

Fig 5 is a flow diagram illustrating the process that results from the Master Thread receiving a response from the Server

DETAILED DESCRIPTION OF THE INVENTION

While the present invention is susceptible to embodiment in various forms, there is shown in the drawings and will hereinafter be described a presently preferred embodiment with the understanding that the present disclosure is to be considered an exemplification of the invention and is not intended to limit the invention to the specific embodiment illustrated.

In the present disclosure, the words "a" or "an" are to be taken to include both the singular and the plural. Conversely, any reference to plural items shall, where appropriate, include the singular.

Referring now to the drawings, more particularly Fig. 1, there is shown a system consisting of a Client 100 and a Server 120. The Server 120 is further comprised of a Network Agent 110 and a Scheduler 130.

In the preferred embodiment of the present invention the Client 100 sends requests to the Network Agent 110. The Network Agent 110 facilitates communication between the Client 100 and the Server 120. The Network Agent 110 is configured to receive requests from the Client 100 and transmit the said requests to the Server 120. It is further configured to receive responses from the Server 120 and transmit the same to the Client 100. In the preferred embodiment of the present invention, the Network Agent 110 is a part of the Server 120 and acts as an interface between the Client 100 and the Server 120. The Scheduler 130 is responsible for the scheduling function of the Server 120. In an alternate embodiment of the present invention, the Network Agent 110 is a separate component, and does not form a part of the Server 120. The Server 120 is configured to perform multiple functions, which include, receiving, retrieving, transferring and storing requests, comparing and matching information and processing requests received by the Network Agent 110.

As illustrated in Fig 1., in the preferred embodiment, one Client 100 sends requests and receives responses from the Network Agent 110, which acts as an interface between the Client 100 and the Server 120. In an alternate embodiment, a plurality of Clients send requests and receive responses from the Network Agent 110, which acts as an interface between the Client 100 and the Server 120.

Fig. 2 is a block diagram illustrating a plurality of typical servers, such as a Database Server 210, a Web Server 220 and a Mail Server 230 having their own Network Agent 110 capable of making communication between a specific functional server and server specific client. In the preferred embodiment of the present invention the Network Agent 110 is capable of managing data interchange across any network irrespective of hardware bounded with the protocol.

Technology evolution gave rise to various standards of networking, which were vendor specific or application specific based on mode of communication or networking medium used. For example a

coaxial cable or UTP / STP cable the most popular transport layers used are TCPIP / IPX / SPX / Netbieu etc. In course of usage of these technologies various application which used these protocols as the basic communication mechanism to transport data needed to exchange data. For example, a Database Server 210 can work on any of these protocols but web servers / mail servers require specific protocols like http / SMTP to propagate data. But database servers could not use these protocols as they were stateless and non-connection oriented and managing transactional data integrity across these transport mechanism was unreliable.

The current trend of convergence of communication protocols over gave rise to piggy packing protocols so that data could be exchanged. For example, SOAP uses http tunneling to manage data exchange. But this reduced bandwidth efficiency and stripping and wrapping overheads while maintaining client server connectivity. Also this technology did not give a seamless integration to legacy clients.

This needs some consideration for making these functionalities work. They are:

1) Each protocol has its own limitations in managing data delivery model. Mostly the evolution of these protocols like http / ftp / smtp / pop /nntp was more of functional evolution rather than technology evolution. Hence some guarantee data delivery and connection consistency whereas some have stateless connectivity.

2) The data delivered by these network modules is specific as per server functionality and none of these have a standard persistent format, which can be interchanged w/o any intermediate conversion steps.

3) Server specific functional objects (like application / session / request / response for web servers or tables / indexes for databases) are capable of having direct communication mechanism or data interchange.

4) The connection polling mechanism and information managed at server side for each client varies like – ODBC / OLEDB clients, web clients, mail clients.

5) For managing protocol interchange for server specific functionality, short comings of the protocol have to be taken care at the server end and relevant information management overheads and other housekeeping and conversion tasks need to be handled.

6) Sub objects of a server, which have similar functionality but different pattern of data handling and manipulation, can be merged.

The basic purpose is to create a network interface capable of transporting any client request to any server irrespective of server functionality or protocol related limitations. The data exchange expected between various functional servers can be managed at the server end if a common pattern of data mapping is used for achieving and retrieving data. Typically a Database Server 210 has a lot of sub objects like tables, indexes etc. Similarly in a Database Server 210 environment the ODBC/OLEDB clients 240 have to explicitly specify the server details, the database to be connected to, user name and password as part of the connect parameters in an ODBC specified standard DSN (Data Source Name) parameters. The connection process between any client and server then exchanges data as per the server functionality and server maintains information for every client connection. The Clients could be ODBC/OLEDB clients 240 on a network using different protocols such as IPX, SPX, TCP IP, NetBeui, Named pipes etc could be using RAS device. The Database Server 210 has its data stored on to the Database as persistent data also the global cache is used for virtual data storage that can be further transferred to the data store and stored as persistent data.

Similarly a Web Server 220 also has sub-objects like application, sessions, request, response, object context and error object. There are various web-clients such as Cuteftp / Getright / IE/ Netscape/ Mosaic 250. The web-client for example the browser communicated using http protocol to the web-server and it receives data mostly as html from the Web Server 220. This Web Server 220 has data stored either on the global cache or in the persistent form from the web server. Functionality may demand persistence of some of these object but in case of web servers most of the data is instance specific i.e. virtual or never persisted. To maintain relations across sessional data currently cookies are used at the client end. The common functionality for any client and server before any data interchange begins is first to establish a communication link, which usually is CONNECT command for a specific server followed by authentication details. As per the functionality of these servers the approach followed for connection establishment varies that is for a web server the server name undergoes a translation for Internet Protocol (IP) through a Domain Name Server (DNS) to establish a connection but does not require any username or password unlike the ftp command on the server which requires authentication.

The Mail clients like Microsoft Outlook Express / Eudora / Pine etc 260 connect to the Mail Server 230. The communication between the mail clients and mail server is only through the protocols like SMTP or POP etc and the data could be stored in the Global Cache 200, that is the virtual memory or in the persistent format in the Mail Server 230 storage area.

The interconnection between these different server functionalities is made possible by the Network Agent 110. The current network implementation merges this connection information for clients

irrespective of protocol or nature of clients and helps maintain this information, which is instance specific like records in a virtual table.

Also the implementation of the Network Agent 110 is purely based on an event driven kernel hence any object functional implementation can be programmed as series of state transitions and inter object data exchange can be managed through messaging. This unique approach of finite state machine based implementation of network related objects or sub objects irrespective of server specific functionality considering all limitations and scope of features expected makes data and functionality exchange seamless. Hence after successful connection parameters are verified the process of authentication, rights, grants, roles and privileges can be centralized as except different standards of authentication (encryption mechanism and algorithms used) the concept of the process of user validation still remains the same. In other words the talk among these different functionality server storing data in proprietary format and using proprietary communication protocols is made possible using the Network Agent 110.

In Fig. 3, there is shown a system consisting of the internal components of the Network Agent 110, consisting of a Master Thread 300, a Sub-Thread 310, a Transmit Queue 320, Socket 330 and a Receive Queue 340.

The Master Thread 300 schedules the tasks carried out by the Network Agent 110 and controls the Sub-Thread 310, to receive requests from the Client 100 and transmit them to the Server 120 and transmit responses from the Server 120 to the Client 100. The Master Thread 300 is activated on the occurrence of any socket activity and polls the Transmit Queue 220, the Socket 230 and the Receive Queue 340 to check which of the said entities has the highest count of requests or responses and accordingly carries out one of the functions of receiving requests from the Client 100, transmitting requests to the Server 120 or transmitting responses from the Server 120, back to the Client 100. In the preferred embodiment of the present invention, there exists one Sub-Thread 310, which is programmed to receive requests based on one or a plurality of protocols. In an alternate embodiment of the present invention, a plurality of Sub-Threads are generated to handle multiple requests, with each Sub-Thread 310 based on a separate protocol. The Network Agent 110 is capable of receiving requests based on different protocols from the Client 100 and transmitting the same to the Server 120. The Socket 330 connects the entire system to a network and thereby enables the Network Agent 110 to receive requests from one or a plurality of Clients. The Receive Queue 340 functions as a storage device and stores the requests, which are transferred by the Sub-Thread 310 from the Socket 320, until such time that the requests are transferred by the Sub-Thread 310 to the Master Thread 300. The Transmit Queue 340 also functions as a storage device and stores the responses received from the Master Thread 300, until such time that the responses are transferred by the Sub-Thread 310 to the Socket 320.

Fig 4 illustrates the manner in which the Network Agent 110 processes requests and responses on the occurrence of any Socket Activity. In the preferred embodiment of the present invention, a Client 100 sends an outgoing CONNECT or a Read or Write request to the Network Agent 110. In an alternate embodiment of the present invention, a plurality of Clients send a plurality of outgoing CONNECT or a Read or Write requests to the Network Agent 110. The requesting client such as browser which uses http or ftp protocol or Microsoft Outlook Express which uses SMTP or POP protocol can connect using either a modem or any Remote Access Device (RAS) and sends outgoing CONNECT or READ or WRITE requests 400 to the Network Agent 110 The polling is done continuously by the Master Thread to create READ or WRITE or CONNECT or DISCONNECT and adding the said requests 401 to the Receive queue 340 or Transmit Queue 320. The Transmit Queue 320 is checked continuously for pending requests 402. In the event of a request existing in the transmit queue 320 the request is taken and processed according to priority logic and pushed in the active state 403. The priority logic is present in the Network Agent 110 and default priority is for the Transmit Queue 320 so that balancing between receiving and transmitting queues is possible. The priority logic in the Network Agent 110 works as per the Network protocol and pending messages in queue and as per timeout of each protocol. In the event of no pending request in the Transmit Queue 320, the Master Thread 300 checks 408 the Receive Queue 340 and in the event of network activity with respect to the Receive Queue 340 the checking for errors is done 409 and an error is reported 410. In the event of no errors in the Receive Queue 340, the request is set to active state 411 and sent to the Scheduler 130. The Scheduler 130 manages the division of operating system time by process and the division of the process time by the agents.

On checking for network activity with respect to the Receive Queue 340, if there is no network activity, it polls for network activity 412.

On setting the request set to active state 411, the Master Thread 300 proceeds to check for incoming connect requests 413. In the event of no incoming connect request, the data is read from the network 414. Next, the reply is sent to the requesting client with data 415. It then proceeds to check for repeat requests 416. In the event of a repeat request existing, the process from checking for requests 402 in the Transmit Queue 320 onwards is repeated. In the case of no repeat request the request is destroyed 417 and next, it proceeds to check for requests 402 in the Transmit Queue 320.

After checking for incoming connect requests 413, in the event of an incoming connect request, the connection is accepted 418 and next a response is given to the requesting client in the form of a handle 419. Next the polling is done to check the request 402 in the Transmit Queue 320. If there exists a request in the Transmit Queue 320, it gets the request according to its priority and pushes it to an active state 403. It then proceeds to check the type of activity such as READ or WRITE or CONNECT 404. In the event of a Write activity the data is written to network 405 and the reply is send to the requesting

client 406 and the further process is repeated. In the event of an outgoing connect the connection to end point is sought 407 and then the reply is given to the requesting client 406. It then proceeds to check 402 for requests in the Transmit Queue 320.

Fig. 5 illustrates the result of the Master Thread 300 receiving 500 a response from the Server 120. On receiving the said response, the Master Thread 300 is activated 510 and the Master Thread 300 then proceeds to transfer 520 the said response to the Transmit Queue 320, after which the Master Thread 300 goes into a state of wait 530, thereby completing the process initiated as a result of the response received 500.

1. A system for managing network resources irrespective of functional servers comprising

a pattern translator to associate a pattern of data and a set of functionalities of said data under a plurality of conditions

a management means to analyze and optimize the type of requests received on said patterns of data

a memory map to aid the process of caching whereby said network can be mapped and utilized as memory queues of requests received and response transmitted.

2. The system as recited in claim 1 wherein said network resource functionality is dictated using a set of generic patterns including data pattern and functional pattern.

3. The system as recited in claim 2 wherein said functional patterns are used to manage optimum load balancing of said data pattern requests.

4. The system as recited in claim 1 wherein data pattern from any functional server is communicated in the form of one or a plurality of network packets and communication protocols.

5. The system as recited in claim 1 enforces industry standard compliant format for communication of said data exchange between functionally different server objects without any layer of conversion.

6. The system as recited in claim 1 exposes any functionality expected by any functional server in a multi-user environment from said network resource for final or intermediate communication stages to achieve simplicity, standardization, operating system portability, said data pattern and functional event portability across vendors and versions of said functional servers.

7. The method of allocating network resources irrespective of functional servers, comprising of:

associating a pattern of data and a set of functionalities of said data under a plurality of conditions

analyzing and optimizing the type of requests received on said patterns of data

caching said patterns of data whereby said network can be mapped and utilized as memory

8. The method as recited in claim 7 wherein said network resource functionality is dictated using a set of generic patterns including data pattern and functional pattern.

9. The method as recited in claim 8 wherein the process of allocating network resource using functional pattern is done by using said functional patterns for managing optimum bandwidth utilization of said data pattern to recover and rebuild said data pattern from any abnormal termination.

10. The method as recited in claim 7 wherein data pattern from any functional server is finally communicated in the form of one or a plurality of network packets.

11. The method as recited in claim 10 wherein said system is capable of performing a network burst operations efficiently by opening multiple instances of network threads with associated protocols

12. The method as recited in claim 7 enforces industry standard compliant format for communication of said data exchange between functionally different server objects without any layer of conversion.

13. The method as recited in claim 7 imparts atomicity, consistency, isolation and durability properties to any sever objects irrespective of server functionality.

14. The method as recited in claim 11 exposes any functionality expected by any functional server in a multi-user environment from said network resource for communication to achieve simplicity, standardization, operating system portability, said data pattern and functional event portability across vendors and versions of said functio

Fig. 1

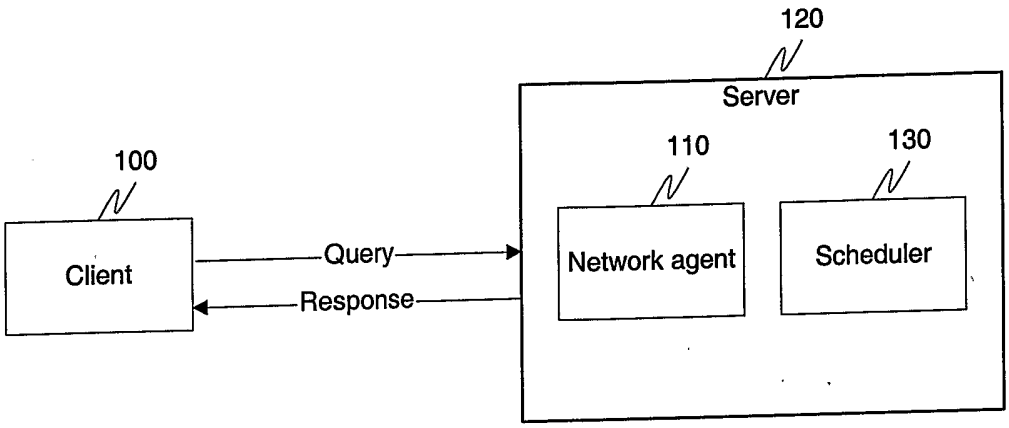

120
Server
110
130
100
Client
Query
Response
Network agent
Scheduler

Fig. 2

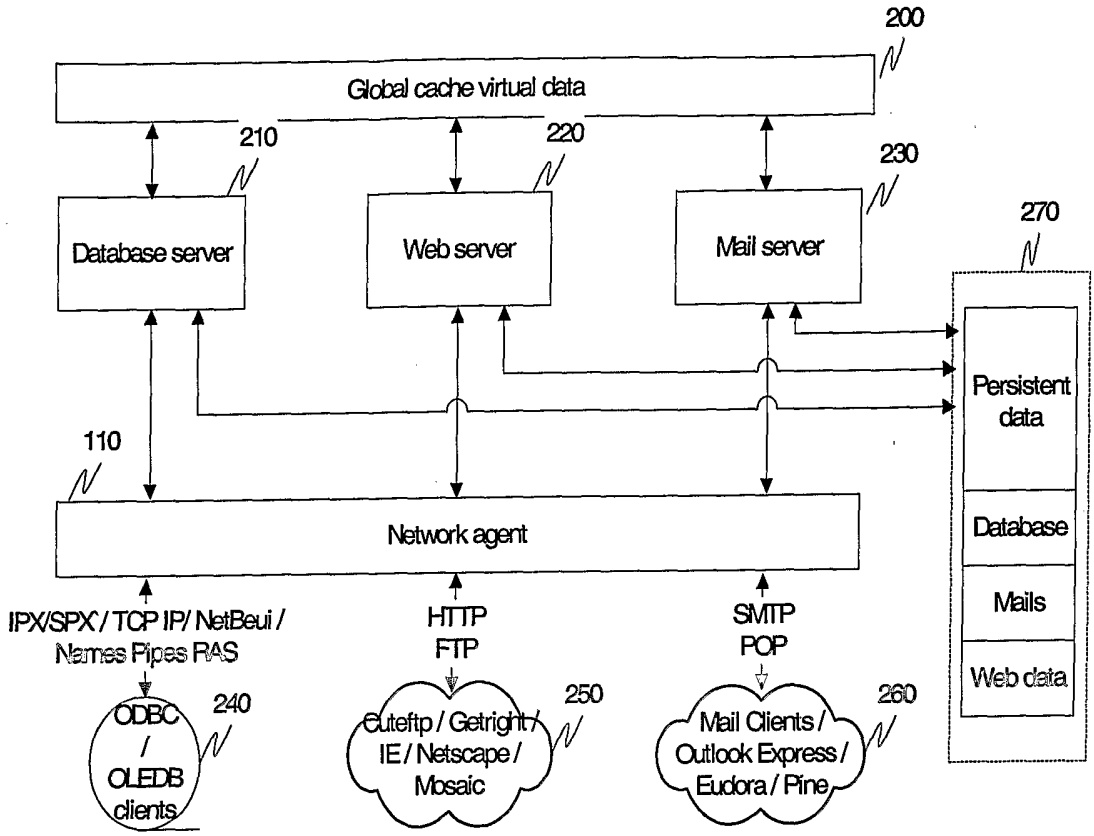

200
Global cache virtual data
210
220
230
270
Database server
Web server
Mail server
Persistent data
110
Network agent
Database
Mails
Web data
IPX/SPX / TCP IP/ NetBeui / Names Pipes RAS
HTTP FTP
SMTP POP
240
250
260
ODBC / OLEDB clients
Cuteftp / Getright / IE / Netscape / Mosaic
Mail Clients / Outlook Express / Eudora / Pine

Fig. 3

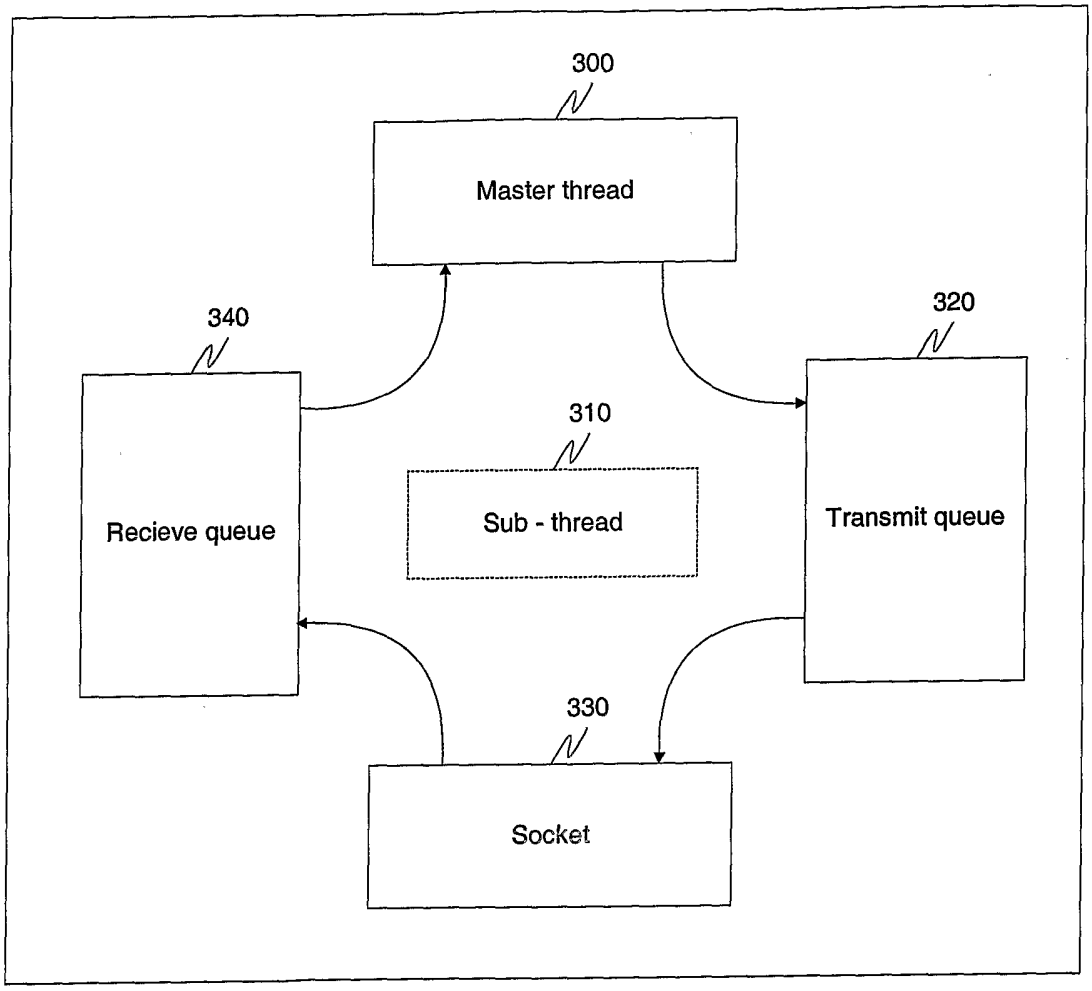

300
Master thread
340
320
310
Recieve queue
Sub - thread
Transmit queue
330
Socket

Fig. 4a

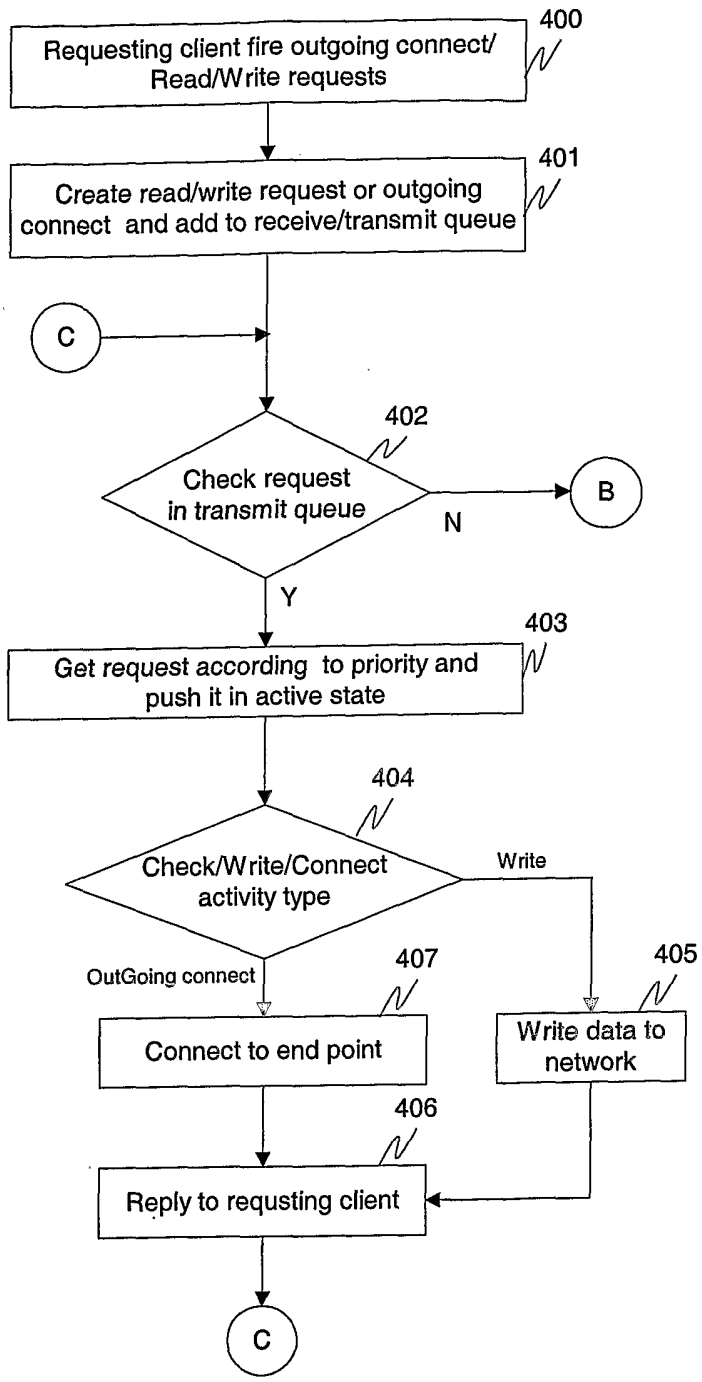

Requesting client fire outgoing connect/ Read/Write requests
400
Create read/write request or outgoing connect and add to receive/transmit queue
401
C
Check request in transmit queue
402
N
B
Y
Get request according to priority and push it in active state
403
Check/Write/Connect activity type
404
Write
OutGoing connect
407
405
Connect to end point
Write data to network
406
Reply to requsting client
C

Fig. 4b

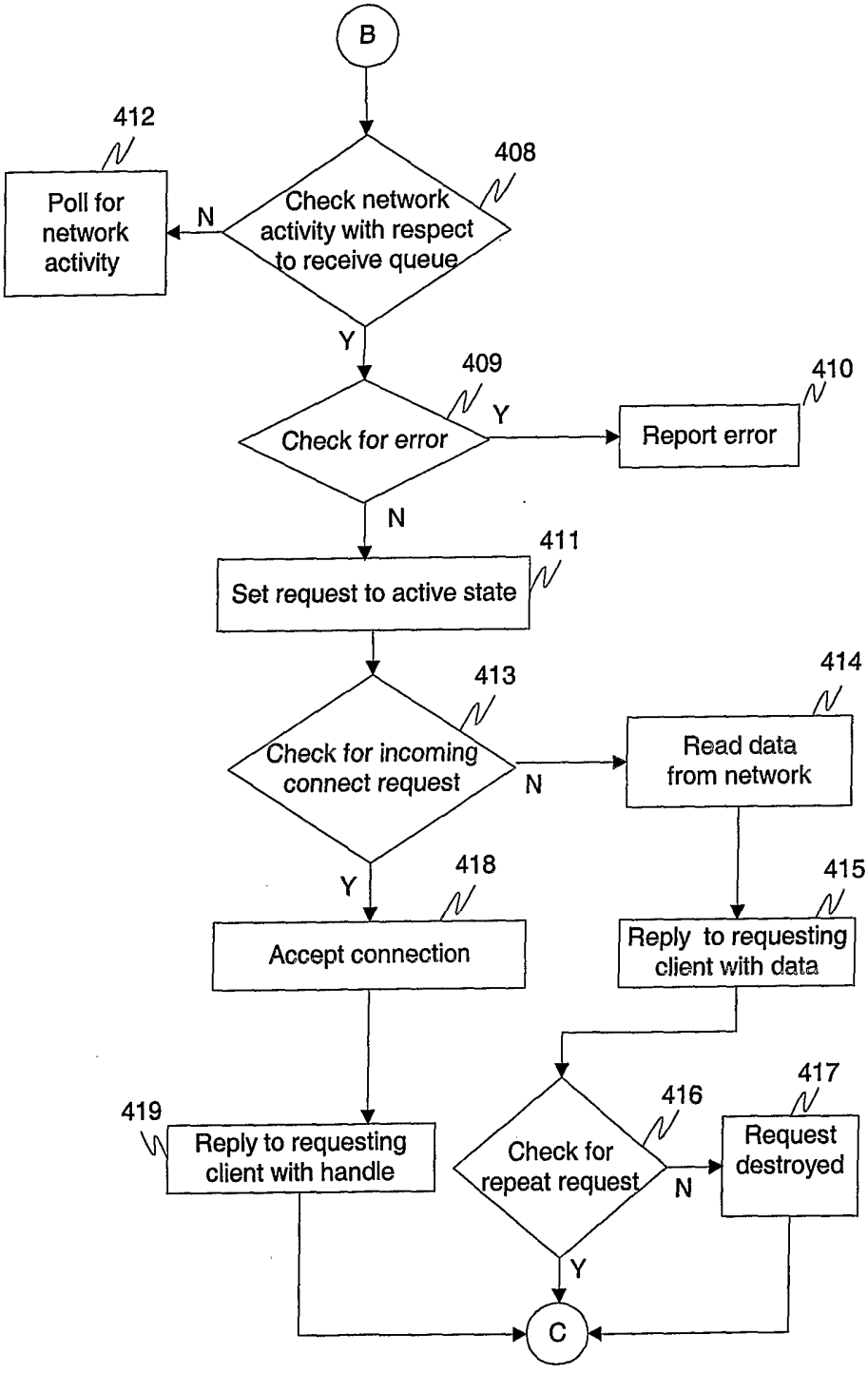
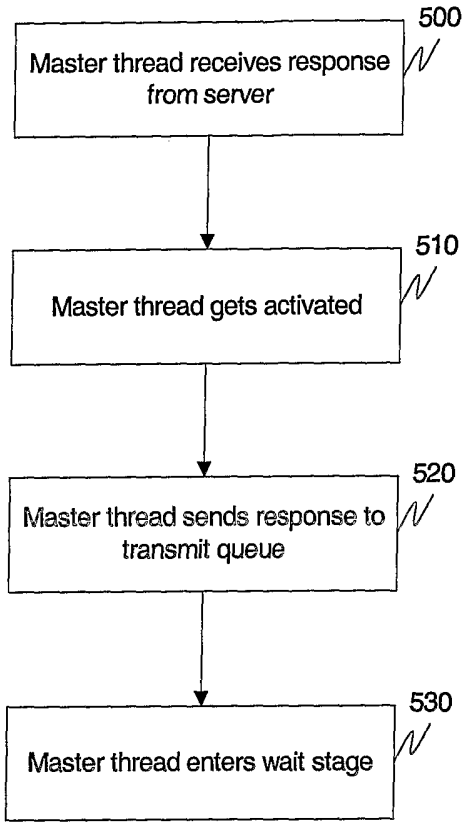

B
412
Poll for network activity
N
408
Check network activity with respect to receive queue
Y
409
Check for error
Y
410
Report error
N
411
Set request to active state
413
Check for incoming connect request
N
414
Read data from network
Y
418
Accept connection
415
Reply to requesting client with data
419
Reply to requesting client with handle
416
Check for repeat request
N
417
Request destroyed
Y
C

Fig. 5


500
Master thread receives response from server
510
Master thread gets activated
520
Master thread sends response to transmit queue
530
Master thread enters wait stage