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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
1 November 2001 (01.11.2001)

PCT

(10) International Publication Number
WO 01/82075 A2

(51) International Patent Classification[7]: G06F 9/46

(21) International Application Number: PCT/US01/13138

(22) International Filing Date: 25 April 2001 (25.04.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/199,401 25 April 2000 (25.04.2000) US

(71) Applicant: **ICPLANET ACQUISITION CORPORATION** [US/US]; 2570 North First Street, San Jose, CA 95131 (US).

(72) Inventors: **FOULGER, Michael, G.**; 637 Olive Avenue, Novato, CA 94945 (US). **CHIPPERFIELD, Thomas, R.**; 7729 Isabel Drive, Cotati, CA 94931 (US). **COOPER, Jeremy, S.**; 59 Burlington Drive, Petaluma, CA 94952 (US).

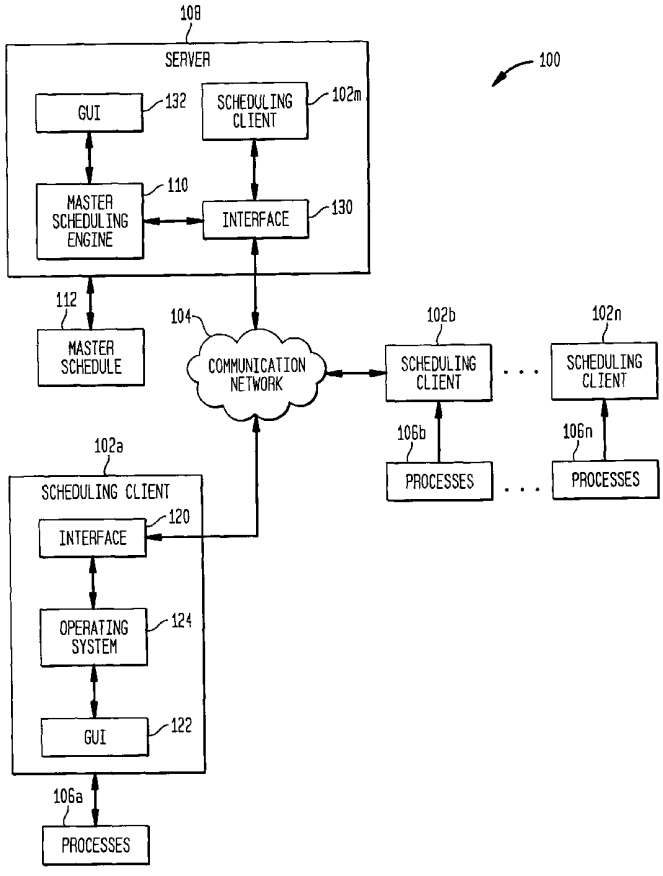
(74) Agents: **RAY, Michael, B.** et al.; Sterne, Kessler, Goldstein & Fox P.L.L.C., Suite 600, 1100 New York Avenue, N.W., Washington, DC 20005-3934 (US).

(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, 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, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

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

[Continued on next page]

(54) Title: SYSTEM AND METHOD FOR SCHEDULING EXECUTION OF CROSS-PLATFORM COMPUTER PROCESSES



(57) Abstract: A distributed computing system includes at least first and second distinct computers each having a different operating system. First processes are compatible with the first operating system and second processes are compatible with the second operating system. A third scheduling computer, coupled to the first and second computer via a communication network, includes a scheduler for scheduling the first processes and the second processes to execute respectively on the first and second computers. The scheduler accesses a master schedule that defines an executing sequence of the first processes and the second processes. The master schedule can define conditional inter-relationships between the first processes and the second processes.

WO 01/82075 A2

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.

SYSTEM AND METHOD FOR SCHEDULING EXECUTION OF CROSS- PLATFORM COMPUTER PROCESSES

Background of the Invention

Field of the Invention

The present invention relates to a system, method, and computer program product for scheduling the execution of computer processes in a network environment.

Related Art

Known distributed computing systems are useful for performing a variety of different computing tasks. "Distributed" refers to physically separated computers that are capable of communicating with one another and/or with a central computer. One such system includes a plurality of distributed computers, wherein each of the computers has its own operating system, which is different from at least one of the other computers. For example, numerous Unix-based computers execute computer programs compatible with Unix, while numerous Microsoft Windows NT based computers execute computer programs compatible with Windows NT. The Unix compatible computer programs can be incompatible with the Windows NT based computers, and vice versa. The Unix and Windows NT compatible programs are collectively referred to as cross-platform processes because the processes collectively execute on plural computer platforms, wherein each computer platform respectively hosts an operating system different from the operating systems hosted by at least one of the other computer platforms.

It is desirable in such a distributed system to coordinate the execution of the computer programs on the distributed computers, so as to achieve one or more useful results. Coordinating the execution of the computer programs requires scheduling the computer programs to execute on the different, incompatible, and

distributed Unix and Windows NT based computers. It is desirable to schedule the computer programs to execute in a preferably user defined executing sequence. It is further desirable to schedule the computer programs to execute in a sequence that depends on the execution results produced by executing or executed computer programs. It is even further desirable to define the scheduling sequence and then control the scheduling sequence (that is, the sequence in which the distributed computer programs are executed) from a centralized location and computer.

Therefore, what is needed is a system, method and computer program product for coordinating the execution of the computer programs on distributed computers and having the above-mentioned desirable features.

Summary of the Invention

The present invention meets the above-mentioned needs and advantageously provides the above-mentioned desired features. The present invention provides a system, method and computer program product for scheduling cross-platform computer programs or processes to execute on distributed computers having operating systems compatible with the processes associated with the computers, each of the computers having an operating system that is different from the operating system of at least one of the other distributed computers.

The present invention advantageously schedules the computer programs to execute on the computers in a user defined executing sequence.

The present invention advantageously schedules the computer programs to execute on the computers in a sequence that depends on the execution results produced by executing or executed computer programs.

In the present invention, the scheduling sequence is defined at a centralized computer that can communicate with each of the distributed computers. Then, the centralized computer controls the scheduling sequence (that

is, the sequence in which the distributed computer programs are executed). In the present invention, the centralized computer and the distributed computers are advantageously networked together.

In one embodiment, the present invention provides a system for scheduling the execution of cross-platform computer processes on client computers. The client computers include first and second distinct computers having respective first and second different operating systems. The system includes a process scheduling computer coupled to the first and second computers. The process scheduling computer includes a scheduler that schedules a first process compatible with the first operating system and a second process compatible with the second operating system to respectively execute on the first and second client.

The system also includes a master schedule that is accessible to the scheduler. The master schedule includes a first process identifier identifying the first process and a second process identifier identifying the second process, the first and second process identifiers being linked together to define an executing sequence of the first and second processes, wherein the scheduler schedules the first and second processes to execute on the respective first and second computers according to the defined executing sequence.

The master schedule also includes one or more conditional inter-relationships between the first and second processes, wherein the scheduler schedules the first and second processes to execute based on the one or more conditional inter-relationships. The one or more conditional inter-relationships include a success criteria associated with the first process. The scheduler includes means for executing the first process, means for comparing the success criteria to execution results produced by the first process, and means for determining whether the first process executed successfully based on a comparison result produced by the comparing means.

The present invention further provides a method and a computer program product for scheduling computer processes to execute in accordance with the above mentioned system for performing same.

Additional features and advantages of the present invention, as well as the structure and operation of various embodiments of the present invention, are described in detail below with reference to the accompanying drawings.

Brief Description of the Figures

The accompanying drawings, which are incorporated herein and form part of the specification, illustrate the present invention and, together with the description, further serve to explain the principles of the invention and to enable a person skilled in the pertinent art make and use the invention.

FIG. 1 is an illustration of a system according to an embodiment of the present invention.

FIG. 2 is an illustration of an exemplary master schedule used in the present invention to schedule processes.

FIG. 3 is a flow chart of a high-level method according to an embodiment of the present invention.

FIG. 4 is an exemplary series of detailed method steps corresponding to a scheduling method step of FIG. 3.

FIG. 5A is a diagram of an example internetwork environment according to the present invention.

FIG.5B is an illustration of a simplified four-layered communication model supporting Web commerce including an application layer, a transport layer, an Internet layer, and a physical layer.

FIG. 5C is an exemplary computer architecture on which the present invention can be implemented.

Detailed Description of the Preferred Embodiments

Fig. 1 is an illustration of a system 100 according to an embodiment of the present invention. System 100 includes a plurality of client computers 102a-102n (also referred to as client or clients 102) coupled to a computer communication network 104. A plurality of processes or tasks 106a-106n are respectively assigned to client computers 102a-102n. Each client (for example, client 102a) can execute one or more processes (for example, processes 106a) assigned to the client.

A server computer 108 (also referred to as server 108) is coupled to computer communication network 104. Server computer 108 includes a Master Scheduling Engine (MSE) 110 for scheduling processes 106a-106n to execute on associated clients 102a-102n, according to the present invention. Note that clients 102 can be thought of as "scheduling clients" because the clients are schedule clients of the MSE 110. However, clients 102 can operate as both servers or clients in a server-client environment, such as the Internet. A master schedule 112, residing in or external to server 108, is accessible to server 108. According to the present invention, server 108 generates master schedule 112 and accesses the contents of master schedule 112 to facilitate the scheduling of processes 106 for execution. Server 108 communicates with clients 102 over communication network 104, which can be any known computer communication network, including the Internet, a company intranet, a local area network (LAN), a wide area network (WAN), the Public Switch Telephone Network (PSTN), and so on. An exemplary network and computer environment in which the present invention can be implemented is described in further detail below, in connection with FIGs. 5A, 5B and 5C.

A logical configuration of client 102a, in an embodiment of the present invention that is typical of the other clients in system 100, is depicted in FIG. 1. Client 102a includes a message interface 120 for sending information (such as messages) to and receiving information from server 108 and the other clients via

communication network 104. In an embodiment of the present invention, client 102a may also include a Graphical User Interface (GUI) 122 for permitting a user to enter information and commands into client 102a and for displaying information to the user. Client 102a operates (for example, executes processes) under an operating system 124. Operating system 124 responds to user commands entered via GUI 102a, and also responds to commands and/or messages received from server 108 via message interface 120.

An exemplary logical configuration of server 108 is also depicted in FIG. 1. Server 108 includes a message interface 130 for sending information (such as messages) to and receiving information from clients 102 via communication network 104. Server 108 also includes a GUI 132 for permitting a user to enter information and commands into server 108 and for displaying information to the user. Server GUI 132 and client GUI 122 can be, for example, web-based or browser GUIs. The MSE 110 is responsive to information including messages received from clients 102 over communication network 104 (and via interface 130), and from commands and information input to server 108 via GUI 132. The MSE 110 generates master schedule 112 in response to such user input and the information received from clients 102.

Each client executes one or more processes 106, as mentioned above. A process is an executable program, such as a compiled program or "executable", a program script, and any other type of executable program that can be executed on a computer, as would be apparent to one skilled in the relevant art. Such processes are often also referred to as "tasks", as is known in the art. A client (for example, client 102a) can execute a process after the process has been installed on the client computer. Installing a process in a client computer typically includes loading the process, for example, a compiled program, into client computer memory such that the client computer can execute the process to produce a useful result. Typically, the installed process is executed under the supervision of the client OS. Each of processes 106 (for example, process 106a).

can be several processes installed on an associated client 102 (for example, client 102a).

As will be appreciated by those skilled in the relevant art(s), the configuration of system 100 may include at least one scheduling client 102m (and associated processes) which is physically located on the same computer as server 108 as shown in FIG. 1.

In an embodiment of the present invention, at least two of client computers 102a-102n are distinct from one another. This means each of the at least two client computers includes, for example, its own processing unit for executing program instructions, memory, user interface hardware (such as, a keyboard), display, logical configuration (for example, its own operating system), etc. For example, client computer 102a comprises a first computer workstation or platform, while client computer 102b comprises a second computer platform. Also in accordance with the present invention, the two distinct clients have different operating systems. For example, client 102a runs under a Microsoft Windows NT operating system (OS), while client 102b runs under any Unix based OS. Thus, system 100 has a "cross-platform" configuration, meaning that a plurality of processes can execute on a plurality of associated clients, each having different operating systems. It is envisioned that system 100 includes many distinct computer platforms that respectively operate under many different operating systems. The different operating systems can be any presently known or future developed operating systems.

It is important to note that while the present invention is described in terms of the above example, this is for convenience only and is not intended to limit the application of the present invention. In fact, after reading the following description, it will be apparent to one skilled in the relevant art(s) how to implement the following invention in alternative embodiments. For example, client computers 102a-n may each operate under a first operating system (e.g., Unix), while server 108 may operate under a second, distinct operating system (e.g., Windows NT).

Consider such an exemplary "cross-platform" configuration of system 100, wherein client 102a is a Microsoft Windows NT platform and client 102b is a Unix based platform. In this exemplary configuration, processes 106a can execute on client 102a only under the Windows NT OS, and processes 106b can execute on client 102b only under the Unix based OS. Accordingly, only Windows NT compatible processes (processes 106a) and only Unix compatible processes (processes 106b) can run on clients 102a and 102b, respectively. In the present invention, the MSE 110 advantageously performs cross-platform scheduling, whereby Windows NT compatible processes 106a and Unix compatible processes 106b are scheduled to execute on respective compatible clients 102a and 102b in accordance with a user defined executing sequence captured in master schedule 112, as will be described in further detail below. In the present invention, the cross-platform processes 106 are advantageously scheduled to execute on associated clients 102 from one central controller (that is, the MSE 110).

FIG. 2 is an illustration of an exemplary master schedule 112. Master schedule 112 includes a process identifier column 202 for listing process identifiers identifying all of the processes 106 installed in and to be executed on associated clients 102. Master schedule 112 includes a client address column 204 for listing client network addresses corresponding to clients 102, to enable server 108 to communicate with each client. Similarly, a network address of server 108 is known to each client 102. Master schedule 112 includes a result criteria column 206 for listing criteria associated with the execution of corresponding processes on clients 102. Such criteria can include, for example, expected outcomes or results produced by processes when the processes execute to successful completions.

Master schedule 112 also includes an action column 210 for listing actions, for example branching commands, that define the sequence in which processes 106a-106n are to be executed (this is also referred to as the "executing sequence" of processes 106). The actions can be conditional, that is, dependent

upon the outcome or results reported by an executed process. Alternatively, the actions can be unconditional, that is, not dependent on such an outcome. Master schedule 112 can optionally include an operating system column 212 for listing the operating systems under which the processes listed in column 202 will run. In an embodiment, master schedule may also include a column (not shown in FIG. 2) for listing process priorities corresponding to each of the identified process. For example, processes can be identified as having high, medium or low execution priorities.

Master schedule 112 includes a plurality of records or rows 220a-220n, each corresponding to a process that is scheduled to be executed in system 100. Processes that are to be installed and executed are also referred to as "jobs". Each row includes a plurality of fields for respectively storing information associated with columns 202-212, described above. For example, row 220a includes a field 230 for storing a process identifier "p1" identifying one of the process 106a in FIG. 1. Row 220a includes a field 231 for storing an exemplary client address corresponding to client 102a. Row 220a also includes a field 232 for storing the following exemplary actions:

1. if p1 fails, goto p1
2. if p1 successful, goto p2

The above listed actions, along with the result criteria information of column 206, define conditional inter-relationships between the processes p1, p2 and p3. The format of the above actions as listed in FIG. 2 is exemplary, and therefore any format can be used that would be apparent to one skilled in the relevant art based on the descriptions provided above and below. The actions (for example, actions (1) and (2)) direct the MSE 110 to cause the processes identified in column 202 of master schedule 112 to execute in a defined executing sequence. Assume, for example, process p1 is currently executing on Windows NT client 102a, and process p2 is installed and waiting to be executed on Unix based client 102b. Action (1) above directs MSE 110 to re-execute process p1 on client 102b when it is determined that process p1 failed because, for example, p1 did not

successfully execute to completion or returned erroneous data to a monitoring function of the MSE 110. On the other hand, action (2) directs MSE 110 to execute process p2 on client 102b when it is determined, for example, that process p1 executed to successful completion. In an another example scenario, the actions could be redefined to direct the MSE 110 to execute process p2 when it is determined p1 has executed successfully, or to execute p3 instead of p2, when it is determined p1 did not execute successfully. Many other conditional, process executing sequence permutations and combinations are possible by simply redefining the above described master schedule, as would be apparent to one skilled in the relevant art based on the above description. For example, as will be appreciated by those skilled in the relevant art(s), master schedule 112 may include timing information which would allow MSE 110 to execute processes on a pre-determined schedule (e.g., hourly, daily, weekly, etc.).

FIG. 3 is a flow chart of a high-level method 300 according to an embodiment of the present invention. Method 300 is described with reference to system 100 and under the assumption that at least two of clients 102 are distinct and are operating under different operating systems, as described above.

Again, the present invention is described under this assumption for convenience only and is not intended to limit the application of the present invention. As will be appreciated by one skilled in the relevant art(s), all client computers 102a-n may each operate under a first operating system (e.g., Unix), while server 108 may operate under a second, distinct operating system (e.g., Windows NT). In fact, as will also be appreciated by one skilled in the relevant art(s), the configuration of system 100 may include scheduling clients102a and 102b (and associated their processes) which are physically located (and executing) on the same computer.

Returning to FIG. 3, method 300 begins at a step 305 when processes 106a-106n are installed to execute on respective clients 102a-102n. In an example scenario, one or more Windows NT compatible processes 106a are installed on Windows NT based client 102a, and one or more Unix compatible

processes are installed on Unix based client 102b. Whenever a process is installed on a client 102 in the present invention, the client sends a notification message to server 108 indicating the installed process needs to be scheduled for execution. The notification message includes a process identifier (for example "p1"). In an alternative embodiment, server 108 will assign the process identifier. In yet another embodiment, an OS type identifier (for example, "Unix") identifying the type of OS residing on the client is also included in the notification message from client 102 to server 108.

At a next step 310, server 108 receives the notification message or messages corresponding to each installed process. Such notification messages are displayed to a user at server 108. The user enters information and commands into server 108 as necessary to construct master schedule 112. Such information includes the information required to populate the fields of each of rows 220a-220n, where each row corresponds to an installed process that needs to be scheduled for execution. For example, the fields of results criteria column 206 are populated with result criteria. Such criteria includes success criteria by which the successful execution of the corresponding processes can be judged or determined. The fields of action column 210 are populated with actions, such as branch commands similar to the action (1) and action (2) mentioned above, defining executing sequences of the installed processes (identified in column 202 of master schedule 112). The user can also enter process executing priorities for the installed processes into master schedule 112. The end result of step 310 is the generation of master schedule 112. Master schedule 112 links together the installed processes, associated with process identifiers in column 202, in such a way as to define executing sequences for the cross-platform processes 106. It is to be understood that the particular form of the construct used to link processes 106 together in master schedule 112 (such as goto statements) is not limited to those depicted in FIG. 2. Any construct, such as linked links, and the like, that would be apparent to one skilled in the relevant art, can be used.

At a next, run-time step 315, the MSE 110 schedules cross-platform processes 106 to execute on clients 102. To do this, the MSE 110 accesses master schedule 112 to thereby schedule processes 106 to execute according to the executing sequence defined by the master schedule. Run-time scheduling step 315 is now described in further detail with reference to FIG. 4, wherein exemplary detailed method steps 400 corresponding to method step 315, are depicted.

At an initial step 405, the MSE 110 accesses master schedule 112 to determine which one of the processes 106 (for example process p1) is to be executed "next". Depending on the processing requirements associated with system 100, master schedule 112 can indicate that several processes 106 are to be executed concurrently, "next". Initially, the "next" process is the first process (or processes) that is to be executed in master schedule 112 (for example, process p1).

At a next step 410, the MSE 110 sends an "initiate execution command" to each of the clients 102 (for example, client 102a) hosting an installed process that is to be executed "next". This command prompts the operating system of the client that receives the command to initiate execution of the process identified in the command.

At a next step 415, the MSE 110 monitors interface 130 for an incoming status message from any of clients 102. Each status message includes a process identifier and a client identifier respectively identifying the sending client and associated process. The status message also includes process execution results produced by the associated process during execution of the process, or when the process completes execution. The process execution results can indicate, for example, that the process successfully executed to completion, or that the process did not successfully execute to completion.

At a next step 420, after receiving such a status message, the MSE 110 uses the process identifier included in the received status message to access and retrieve the appropriate result criteria in the appropriate row of the master

schedule 112. The MSE 110 compares the retrieved result criteria to the process execution results included in the status message.

At a next step 425, the MSE 110 determines which process scheduling action is appropriate based on comparison step 420 and the action information stored in action column 210 of master schedule 112, as described above in connection with the FIG. 2. In other words, the MSE 110 determines a "next" process to be executed, and flow control proceeds back to step 410. In this manner, master schedule 112 and the MSE 110 together form a dynamic and flexible, centralized, cross-platform process scheduling mechanism, whereby a process executing sequence can be based on execution results produced by the processes identified in the master schedule. In other words, the executing sequence can be adjusted based on the execution results, the user defined result criteria, and the actions defined in the master schedule 112.

In another embodiment, the MSE 110 determines which process scheduling action is appropriate based on the above mentioned factors, and in addition, on a process priority stored in master schedule 112. For example, the MSE 110 may cause a high priority "next" process to preempt a low priority "next" process on one of clients 102.

In yet another embodiment, the MSE 110 monitors a processing loading or "busy-state" of each of clients 102. In this embodiment, the MSE 110 determines which process scheduling action is appropriate based on the factors described above in connection with method steps 420 and 425, and in addition, on the "busy-state" of each client. This embodiment gives the MSE 110 the flexibility to transfer "next" processes from busy clients to available clients, and to initiate execution of the transferred "next" processes on the available clients. To do this, the MSE 110 determines which "next" processes are scheduled to be executed on busy clients. The MSE 110 also determines which clients are available to execute processes. Assuming the MSE 110 determines that one or more clients are available, the MSE sends "process transfer" commands to the busy clients associated with the "next" processes. Each process transfer

command received by a busy client directs the busy client to transfer its "next" installed process to an available client identified by a client destination address in the transfer command. Each available destination client has an operating system compatible with the busy client from where the process is being transferred. MSE 110 is able to determine such compatible client transfer pairs based on the OS type information in column 212 of master schedule 112. When the "next" process has been successfully transferred from the busy client to the available client, then the MSE 110 initiates execution of the transferred "next" process on the available client.

With reference to FIG. 1 an example cross-platform scheduling scenario is now described. The example scenario includes the following client/process configuration. Client 102a is an Internet Web-server having a Sun/Solaria Unix based operating system. An Internet information gathering process installed on client 102a (e.g., a Web search engine or crawler) is used to automatically search Internet-web sites for predetermined information, collect "found" information, and send the found information to another computer within the same network. All of this is referred to as "job1."

In the example scenario, the particular web-information collected by the Internet information gathering process of client 102a is passed to client 102b. Client 102b is a work station operating under the Windows NT OS. A chart generating process installed on client 102b is used to generate bit mapped charts based on the web-information passed to client 102b. The charts are displayed to a user. This is referred to as "job2". The chart generating process of client 102b submits queries to a database application (this is referred to as "job3") residing on client 102c, which is also a Windows NT based client.

In this example scenario, a master schedule 112 is generated to schedule the following conditional executing sequence of processes (i.e., job1, job2 and job3):

1. job1 (Execute the information gathering process on client 102a to collect web information and then pass the gathered web information to client 102b);

2. If job1 successful, goto job2 (concurrently execute the chart generating process on client 102b and the database application on client 102c, to generate and display charts),

If job1 unsuccessful, goto job1 (re-execute job1 one time only); and

3. If job2 successful, then DONE,

If job2 unsuccessful, then ERROR.

In yet another, example cross-platform scenario, a Unix based client 102a executes the web-based information gatherer (job1) described above, a Unix based client 102b executes a parser to parse the gathered information (job2), and a Windows NT based client 102c executes an email generator to generate email messages available to a user (job3 and job4). The jobs are scheduled to execute according to the following master schedule:

1. job1 (execute the information gatherer);

2. If job1 successful, goto job2 (parse the gathered data); and

3. If job2 successful, goto job3 (generate an email Success message),

If job1 unsuccessful, goto job4 (generate an email Error message).

In the above example scenarios, only a single job is scheduled to execute on each distinct client 102. However, in an alternate embodiment of the present invention, master schedule 112 can be constructed such that several jobs are queued for scheduled execution on one or more of clients 102.

Example Network Environment

The present invention can be implemented in any communication network, such as, the Internet, which supports interactive services and applications. In particular, the present invention can be implemented in any Web service, preferably a Web service supporting secure transactions, such as, the Secure Socket Layer (SSL) protocol and/or using a Secure HyperText Transport Protocol (S-HTTP). In one example, the present invention is implemented in a multi-platform (platform independent) programming language such as Java. Java-enabled browsers are used, such as, Netscape, HotJava, and Microsoft Explorer browsers. Active content Web pages can be used. Such active content Web pages can include Java applets or ActiveX controls, or any other active content technology developed now or in the future. The present invention, however, is not intended to be limited to Java or Java-enabled browsers, and can be implemented in any programming language and browser, developed now or in the future, as would be apparent to a person skilled in the art given this description. Further, the present invention is not intended to be limited to a Web-based implementation or environment and can be implemented in any communication network now or in the future, as would be apparent to a person skilled in the art given this description. Even further, the present invention can operate in the absence of a network, for example, on a computer not connected with a network.

FIG. 5A is a diagram of an example internetwork environment according to the present invention. FIG. 5A shows a communication network or combination of networks (Internet) 500 (corresponding to communication network 104 of FIG. 1) which can support the invention. Internet 500 consists of interconnected computers which supports communication between many different types of users including businesses, universities, individuals, government, and financial institutions. Internet 500 supports many different types of communication links implemented in a variety of architectures. For example,

voice and data links can be used including phone, paging, cellular, and cable TV (CATV) links. Terminal equipment can include local area networks, personal computers with modems, content servers of multi-media, audio, video, and other information, pocket organizers, personal digital assistants (PDAs), and set-top boxes.

Communication over a communication network such as, Internet 500, is carried out through different layers of communication. FIG. 5B shows a simplified four-layered communication model supporting Web commerce including an application layer 508, transport layer 510, Internet layer 520, physical layer 530. As would be apparent to a person skilled in the art, in practice, a number of different layers can be used depending upon a particular network design and communication application. Application layer 508 represents the different tools and information services which are used to access the information over the Internet. Such tools include, but are not limited to, telenet log-in service 501, IRC chat 502, Web service 503, and SMTP (Simple Mail Transfer Protocol) electronic mail service 506. Web service 503 allows access to HTTP documents 504, and FTP and Gopher files 505. A Secure Socket Layer (SSL) is an optional protocol used to encrypt communications between a Web browser and Web server.

Description of the example environment in these terms is provided for convenience only. It is not intended that the invention be limited to application in this example environment. In fact, after reading the following description, it will become apparent to a person skilled in the relevant art how to implement the invention in alternative environments.

Example Computer System

An example of a computer system 540 is shown in FIG. 5C. The computer system 540 represents any single or multi-processor computer. Single

or multi-tasking computers can be used. Unified or distributed memory systems can be used.

Computer system 540 includes one or more processors, such as processor 544. In one embodiment, computer system 540 corresponds to server 108 of FIG. 1, wherein scheduling engine 110 (also referred to as scheduler 110) comprises one or more processors 544 for executing software implemented methods 300 and 400 as described above, and as appropriate. Each processor 544 is connected to a communication infrastructure 542 (e.g., a communications bus, cross-bar, or network). Various software embodiments are described in terms of this exemplary computer system. After reading this description, it will become apparent to a person skilled in the relevant art how to implement the invention using other computer systems and/or computer architectures.

Computer system 540 also includes a main memory 546, preferably random access memory (RAM), and can also include a secondary memory 548. The secondary memory 548 can include, for example, a hard disk drive 552 and/or a removable storage drive 552, representing a floppy disk drive, a magnetic tape drive, an optical disk drive, etc. The removable storage drive 552 reads from and/or writes to a removable storage unit 554 in a well known manner. Removable storage unit 554 represents a floppy disk, magnetic tape, optical disk, etc., which is read by and written to by removable storage drive 552. As will be appreciated, the removable storage unit 554 includes a computer usable storage medium having stored therein computer software and/or data.

In alternative embodiments, secondary memory 560 may include other similar means for allowing computer programs or other instructions to be loaded into computer system 540. Such means can include, for example, a removable storage unit 562 and an interface 560. Examples can include a program cartridge and cartridge interface (such as that found in video game devices), a removable memory chip (such as an EPROM, or PROM) and associated socket, and other removable storage units 562 and interfaces 560 which allow software and data to be transferred from the removable storage unit 562 to computer system 540.

Computer system 540 can also include a communications interface 564. Communications interface 564 allows software and data to be transferred between computer system 540 and external devices via communications path 566. Examples of communications interface 564 can include a modem, a network interface (such as Ethernet card), a communications port, etc. Software and data transferred via communications interface 564 are in the form of signals 568 which can be electronic, electromagnetic, optical or other signals capable of being received by communications interface 564, via communications path 566. Note that communications interface 564 provides a means by which computer system 540 can interface to a network such as the Internet.

The present invention can be implemented using software running (that is, executing) in an environment similar to that described above with respect to FIG. 5A. In this document, the term "computer program product" is used to generally refer to removable storage unit 554, a hard disk installed in hard disk drive 552, or a carrier wave carrying software over a communication path 566 (wireless link or cable) to communication interface 564. A computer useable medium can include magnetic media, optical media, or other recordable media, or media that transmits a carrier wave or other signal. These computer program products are means for providing software to computer system 540.

Computer programs (also called computer control logic) are stored in main memory 546 and/or secondary memory 548. Computer programs can also be received via communications interface 564. Such computer programs, when executed, enable the computer system 540 to perform the features of the present invention as discussed herein. In particular, the computer programs, when executed, enable the processor 544 to perform the features of the present invention, as related to proximity searching. Accordingly, such computer programs represent controllers of the computer system 540.

The present invention can be implemented as control logic in software, firmware, hardware or any combination thereof. In an embodiment where the invention is implemented using software, the software may be stored in a

computer program product and loaded into computer system 540 using removable storage drive 552, hard drive 550, or interface 560. Alternatively, the computer program product may be downloaded to computer system 540 over communications path 566. The control logic (software), when executed by the one or more processors 544, causes the processor(s) 544 to perform the functions of the invention as described herein.

In another embodiment, the invention is implemented primarily in firmware and/or hardware using, for example, hardware components such as application specific integrated circuits (ASICs). Implementation of a hardware state machine so as to perform the functions described herein will be apparent to persons skilled in the relevant art(s).

Conclusion

While various embodiments of the present invention have been described above, it should be understood that they have been presented by way of example only, and not limitation. It will be understood by those skilled in the art that various changes in form and details may be made therein without departing from the spirit and scope of the invention as defined in the appended claims. Thus, the breadth and scope of the present invention should not be limited by any of the above-described exemplary embodiments, but should be defined only in accordance with the following claims and their equivalents.

What Is Claimed Is:

1. In a system including first and second distinct computers respectively having first and second different operating systems, a cross-platform process scheduling method comprising the step of:

scheduling a first process compatible with the first operating system and a second process compatible with the second operating system to respectively execute on the first and second computers.

2. The method of claim 1, wherein a master schedule includes a first process identifier identifying the first process and a second process identifier identifying the second process, the first and second process identifiers being linked together to define an executing sequence of the first and second processes, the method further comprising the step of scheduling the first and second processes to execute on the respective first and second computers according to the defined executing sequence.

3. The method of claim 2, wherein the master schedule includes one or more conditional inter-relationships between the first and second processes, the method further comprising the step of scheduling the first and second processes to execute based on the one or more conditional inter-relationships.

4. The method of claim 3, wherein the one or more conditional inter-relationships include a success criteria associated with the first process, the method further comprising the steps of:

executing the first process;

comparing the success criteria to execution results produced by the first process; and

determining whether the first process executed successfully based on the comparison step.

5. The method of claim 4, wherein the master table includes a third process identifier identifying a third process, the method further comprising the alternative steps of:

executing the second process when the first process executed successfully according to the determining step, and

executing the third process but not the second process when the first process did not execute successfully according to the determining step.

6. The method of claim 5, further comprising the steps of:

sending a command to the first computer to initiate execution of the first process on the first computer;

receiving a result message from the first computer, the result message including the execution results produced by the first process; and

sending a command to the second computer when the first process executed successfully according to the determining step to initiate execution of the second process on the second computer.

7. The method of claim 2, further comprising the step of monitoring processor loading associated with the first and second computers and adjusting the executing sequence based on the processor loading.

8. The method of claim 2, wherein the master table includes a priority associated with each process identifier, the method further comprising the step of adjusting the executing sequence based on the respective priorities associated with the first and second processes.

9. The method of claim 1, further comprising prior to the scheduling step, the steps of:

receiving a first message from the first computer indicating that the first process needs to be scheduled for execution on the first computer;

receiving a second message from the second computer indicating that the second process needs to be scheduled for execution on the second computer; and

generating the master schedule based on the first and second messages.

10. The method of claim 9, wherein the generating step includes the steps of:

receiving one or more commands indicating an executing sequence of the first and second processes; and

linking the first and second processes together according to the commands.

11. A system for scheduling the execution of cross-platform computer processes on a plurality of client computers, the plurality of client computers including first and second distinct computers having respective first and second different operating systems, comprising:

a process scheduling computer coupled to the first and second computers, the scheduling computer including a scheduler that schedules a first process compatible with the first operating system and a second process compatible with the second operating system to respectively execute on the first and second client computers.

12. The system of claim 11, further comprising a master schedule that is accessible to the scheduler, the master schedule including a first process identifier identifying the first process and a second process identifier identifying the second process, the first and second process identifiers being linked together to define an executing sequence of the first and second processes, wherein the scheduler schedules the first and second processes to execute on the respective first and second computers according to the defined executing sequence.

13. The system of claim 12, wherein the master schedule includes one or more conditional inter-relationships between the first and second processes, and wherein the scheduler schedules the first and second processes to execute based on the one or more conditional inter-relationships.

14. The system of claim 13, wherein the one or more conditional inter-relationships include a success criteria associated with the first process, and wherein the scheduler includes:

means for executing the first process;

means for comparing the success criteria to execution results produced by the first process; and

means for determining whether the first process executed successfully based on a comparison result produced by the comparing means.

15. The system of claim 14, wherein the master table includes a third process identifier identifying a third process, and wherein the scheduler includes:

means for executing the second process when the first process executed successfully according to the determining step; and

means for executing the third process but not the second process when the first process did not execute successfully according to the determining step.

16. The system of claim 15, wherein the scheduler further comprises:

means for sending a command to the first computer to initiate execution of the first process on the first computer;

means for receiving a result message from the first computer, the result message including the execution results produced by the first process; and

means for sending a command to the second computer to initiate execution of the second process on the second computer when the determining means determines the first process executed successfully.

17. The system of claim 12, wherein the scheduler comprises means for monitoring a processor loading associated with the first and second computers and adjusting the executing sequence based on the processor loading.

18. The system of claim 12, wherein the master table includes a priority associated with each process identifier, and wherein the scheduler includes means for adjusting the executing sequence based on the respective priorities associated with the first and second processes.

19. The system of claim 11, wherein the scheduler comprises:

means for receiving a first message from the first computer indicating that the first process needs to be scheduled for execution on the first computers;

means for receiving a second message from the second computer indicating that the second process needs to be scheduled for execution on the second computer; and

means for generating the master schedule based on the first and second messages.

20. The system of claim 19, wherein the generating means includes:

means for receiving one or more commands indicating an executing sequence of the first and second processes; and

means for linking the first and second processes together according to the commands.

21. In a system including first and second distinct computers respectively having first and second different operating systems, a computer program product comprising computer usable media having computer readable program code means embodied in the media for causing application programs to execute on a computer processor to perform cross-platform computer process scheduling, the computer readable program code means comprising:

first computer readable program code means for causing the processor to schedule a first process compatible with the first operating system and a second process compatible with the second operating system to respectively execute on the first and second computers.

22. The computer program product of claim 21, wherein a master schedule includes a first process identifier identifying the first process and a second process identifier identifying the second process, the first and second process identifiers being linked together to define an executing sequence of the first and second processes, the computer program product further comprising a second computer readable program code means for causing the processor to schedule the first and second processes to execute on the respective first and second computers according to the defined executing sequence.

23. The computer program product of claim 22, wherein the master schedule includes one or more conditional inter-relationships between the first and second processes, the computer program product further comprising a third computer readable program code means for causing the processor to schedule the first and second processes to execute based on the one or more conditional inter-relationships.

24. The computer program product of claim 23, wherein the one or more conditional inter-relationships include a success criteria associated with the first process, the computer program product further comprising:

fourth computer readable program code means for causing the processor to execute the first process;

fifth computer readable program code means for causing the processor to compare the success criteria to execution results produced by the first process; and

sixth computer readable program code means for causing the processor to determine whether the first process executed successfully based on a result of the compare.

25. The computer program product of claim 24, wherein the master table includes a third process identifier identifying a third process, the computer program product further comprising:

seventh computer readable program code means for causing the processor to execute the second process when the first process executed successfully; and

eighth computer readable program code means for causing the processor to execute the third process but not the second process when the first process did not execute successfully.

26. The computer program product of claim 25, further comprising:

ninth computer readable program code means for causing the processor to send a command to the first computer to initiate execution of the first process on the first computer;

tenth computer readable program code means for causing the processor to receive a result message from the first computer, the result message including the execution results produced by the first process; and

eleventh computer readable program code means for causing the processor to send a command to the second computer to initiate execution of the second process on the second computer when it is determined that the first process executed successfully.

FIG. 1

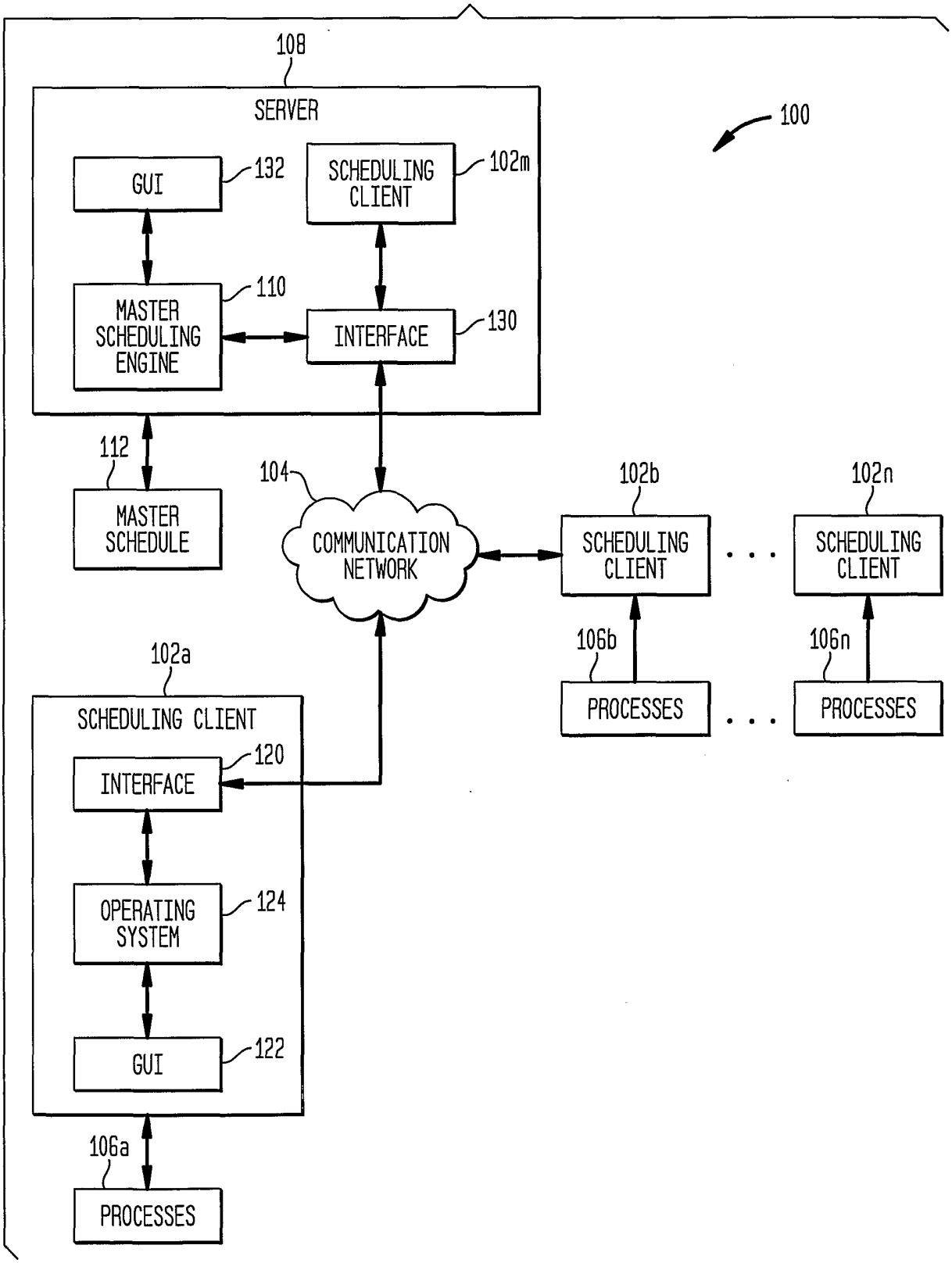

108
SERVER
100
GUI
132
SCHEDULING CLIENT
102m
MASTER SCHEDULING ENGINE
110
INTERFACE
130
112
MASTER SCHEDULE
104
COMMUNICATION NETWORK
102b
SCHEDULING CLIENT
102n
SCHEDULING CLIENT
106b
PROCESSES
106n
PROCESSES
102a
SCHEDULING CLIENT
INTERFACE
120
OPERATING SYSTEM
124
GUI
122
106a
PROCESSES

FIG. 2

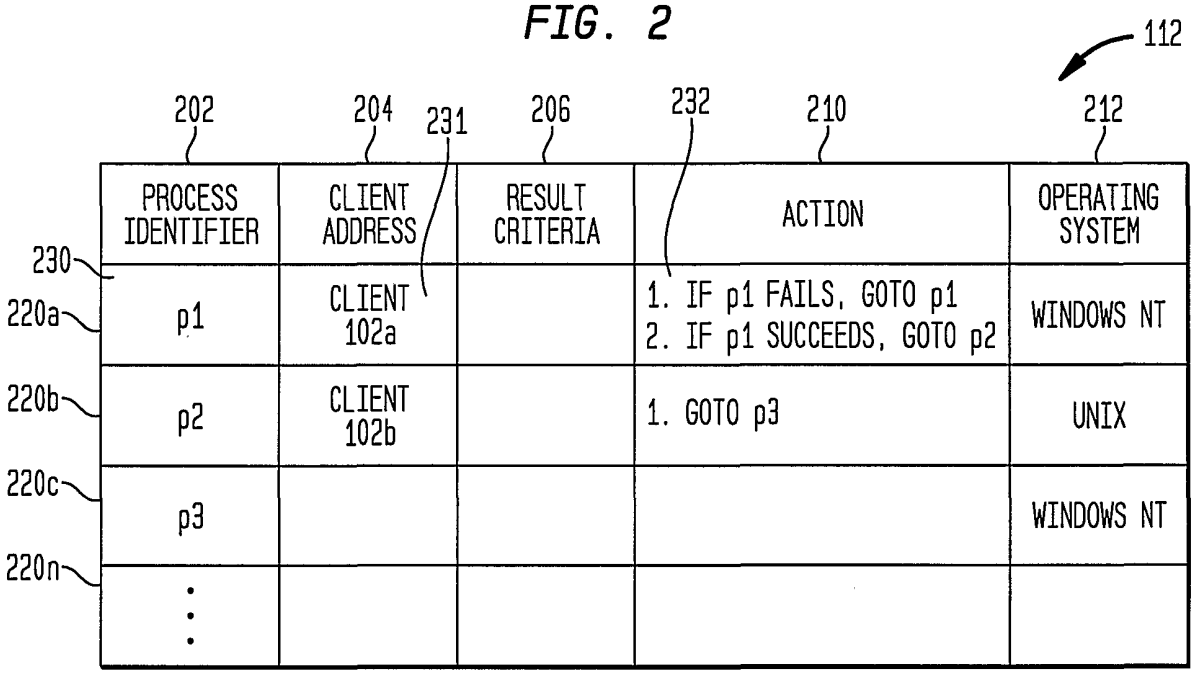

112
202
204
231
206
232
210
212
PROCESS IDENTIFIER
CLIENT ADDRESS
RESULT CRITERIA
ACTION
OPERATING SYSTEM
230
220a
p1
CLIENT 102a
1. IF p1 FAILS, GOTO p1
2. IF p1 SUCCEEDS, GOTO p2
WINDOWS NT
220b
p2
CLIENT 102b
1. GOTO p3
UNIX
220c
p3
WINDOWS NT
220n

FIG. 3

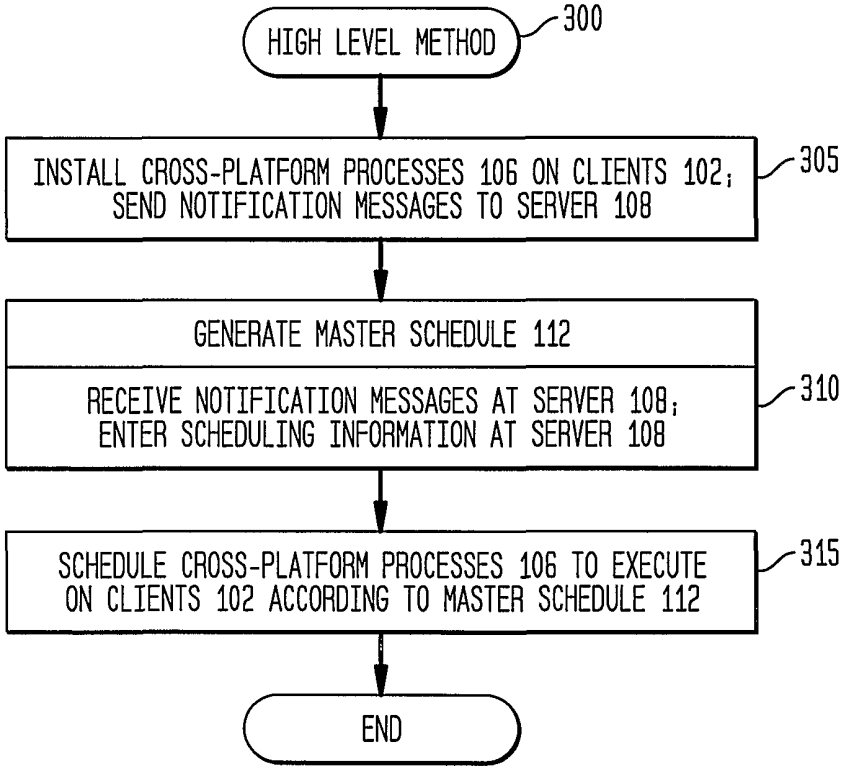

HIGH LEVEL METHOD
300
INSTALL CROSS-PLATFORM PROCESSES 106 ON CLIENTS 102; SEND NOTIFICATION MESSAGES TO SERVER 108
305
GENERATE MASTER SCHEDULE 112
RECEIVE NOTIFICATION MESSAGES AT SERVER 108; ENTER SCHEDULING INFORMATION AT SERVER 108
310
SCHEDULE CROSS-PLATFORM PROCESSES 106 TO EXECUTE ON CLIENTS 102 ACCORDING TO MASTER SCHEDULE 112
315
END

FIG. 4

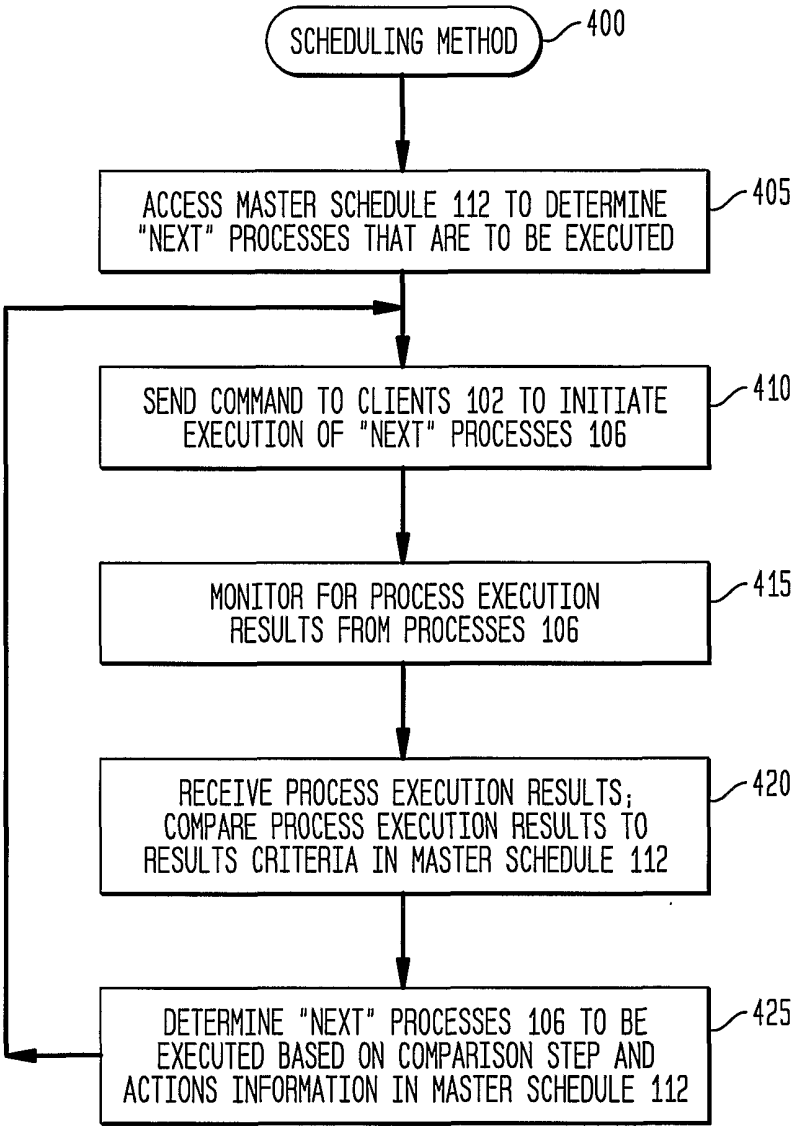

SCHEDULING METHOD
400
ACCESS MASTER SCHEDULE 112 TO DETERMINE "NEXT" PROCESSES THAT ARE TO BE EXECUTED
405
SEND COMMAND TO CLIENTS 102 TO INITIATE EXECUTION OF "NEXT" PROCESSES 106
410
MONITOR FOR PROCESS EXECUTION RESULTS FROM PROCESSES 106
415
RECEIVE PROCESS EXECUTION RESULTS; COMPARE PROCESS EXECUTION RESULTS TO RESULTS CRITERIA IN MASTER SCHEDULE 112
420
DETERMINE "NEXT" PROCESSES 106 TO BE EXECUTED BASED ON COMPARISON STEP AND ACTIONS INFORMATION IN MASTER SCHEDULE 112
425

FIG. 5A

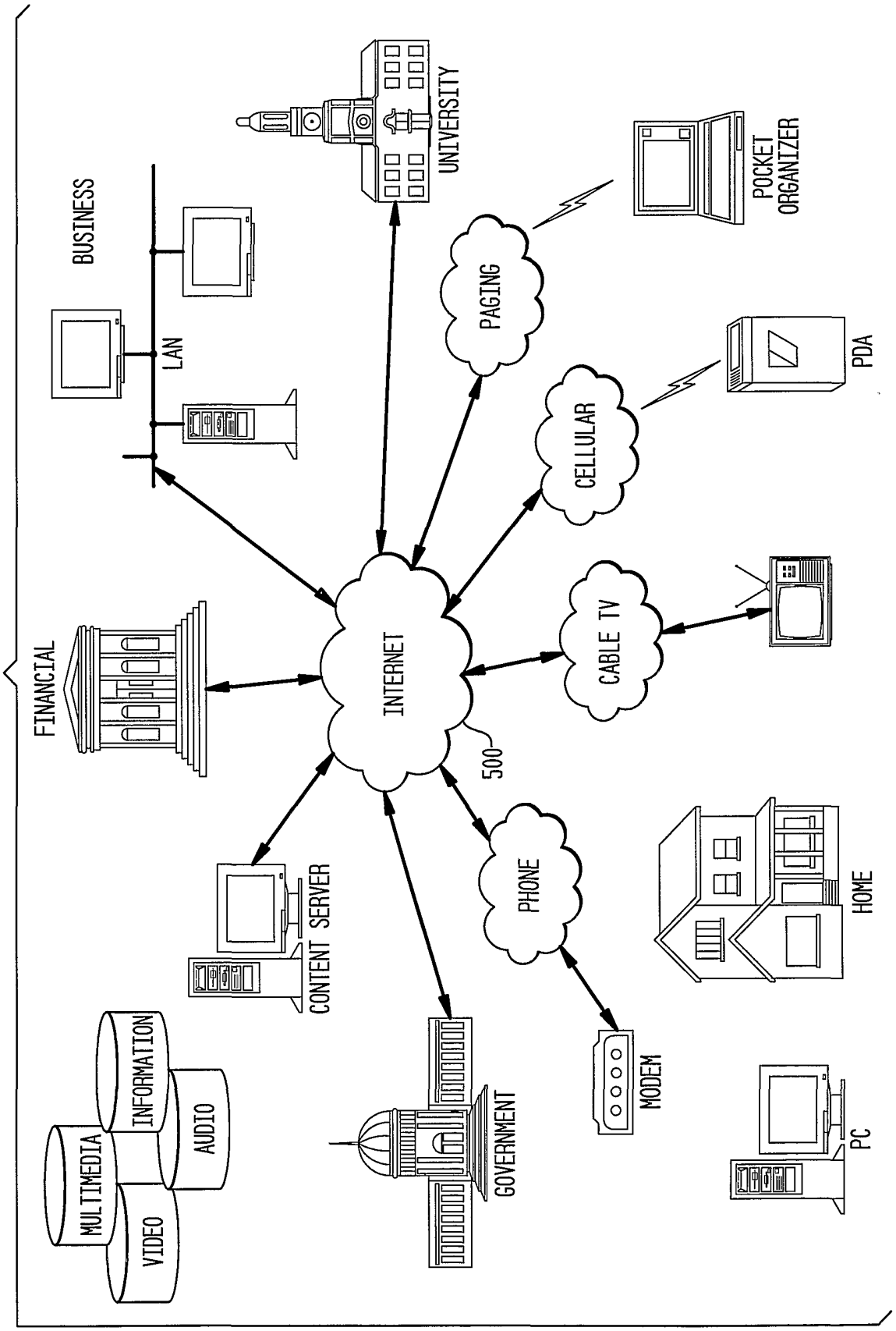

BUSINESS
LAN
UNIVERSITY
POCKET ORGANIZER
PAGING
PDA
CELLULAR
FINANCIAL
INTERNET
CABLE TV
500
CONTENT SERVER
PHONE
HOME
MULTIMEDIA
INFORMATION
AUDIO
VIDEO
GOVERNMENT
MODEM
PC

FIG. 5B

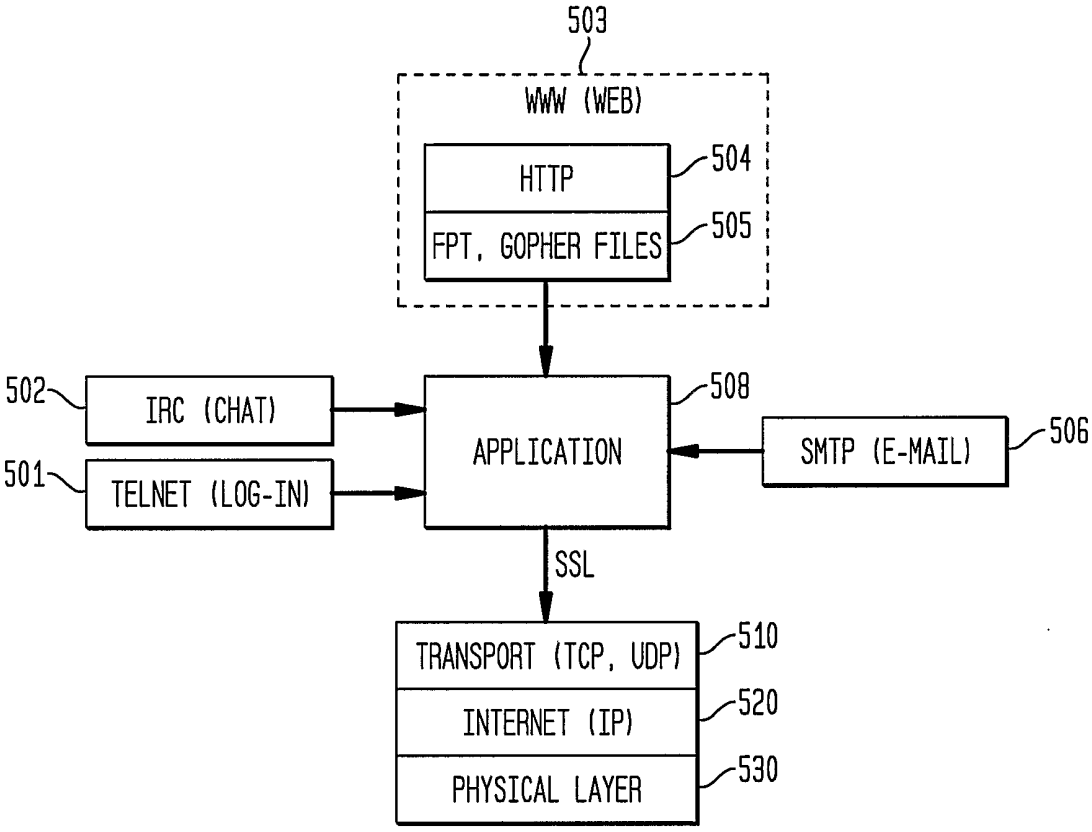
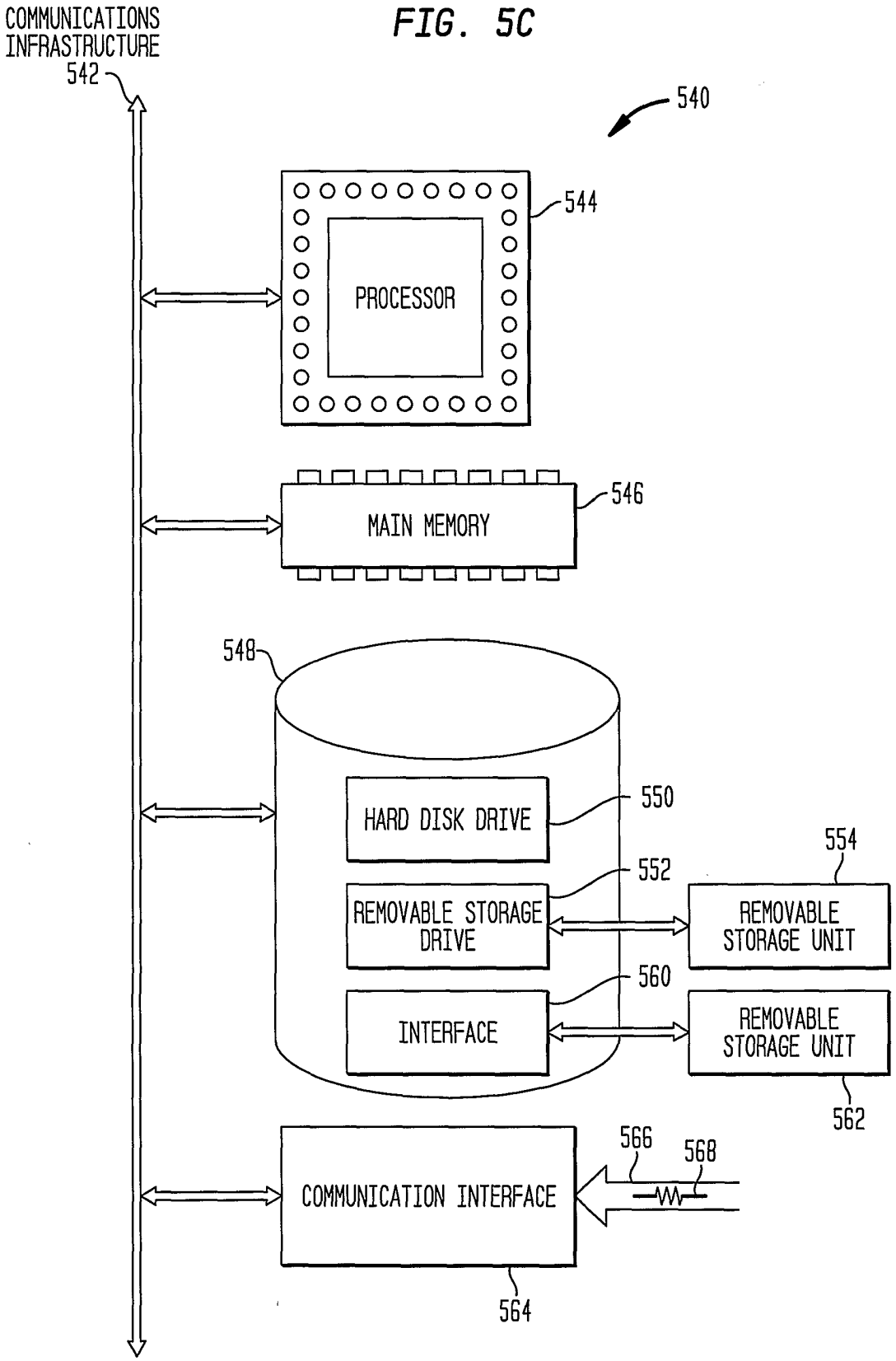

503
WWW (WEB)
HTTP
504
FPT, GOPHER FILES
505
502
IRC (CHAT)
501
TELNET (LOG-IN)
APPLICATION
508
SMTP (E-MAIL)
506
SSL
TRANSPORT (TCP, UDP)
510
INTERNET (IP)
520
PHYSICAL LAYER
530

FIG. 5C


COMMUNICATIONS INFRASTRUCTURE
542
540
544
PROCESSOR
546
MAIN MEMORY
548
HARD DISK DRIVE
550
552
554
REMOVABLE STORAGE DRIVE
REMOVABLE STORAGE UNIT
560
INTERFACE
REMOVABLE STORAGE UNIT
562
566
568
COMMUNICATION INTERFACE
564