(19) **United States**

(12) **Patent Application Publication** (10) Pub. No.: **US 2006/0149993 A1**

Srivastava et al. (43) **Pub. Date:** **Jul. 6, 2006**

(54) **METHOD FOR EVENT TRIGGERED MONITORING OF MANAGED SERVER HEALTH**

(75) Inventors: **Rahul Srivastava**, Randolph, NJ (US); **Ananthan Bala Srinivasan**, San Francisco, CA (US); **Eric M. Halpern**, San Francisco, CA (US); **Dean Bernard Jacobs**, Berkeley, CA (US)

Correspondence Address:
FLIESLER MEYER, LLP
FOUR EMBARCADERO CENTER
SUITE 400
SAN FRANCISCO, CA 94111 (US)

(73) Assignee: **BEA Systems, Inc.**, San Jose, CA (US)

(21) Appl. No.: **11/344,307**

(22) Filed: **Jan. 31, 2006**

Related U.S. Application Data

(63) Continuation of application No. 10/338,981, filed on Jan. 9, 2003.

(60) Provisional application No. 60/359,009, filed on Feb. 22, 2002.

Publication Classification

(51) **Int. Cl.**
G06F 11/00 (2006.01)
(52) **U.S. Cl.** **714/4**

(57) **ABSTRACT**

A Node Manager monitors the status of multiple servers. The Node Manager detects server failures, periodically monitors server health status, and performs server maintenance. When the Node Manager detects a server failure, it determines whether or not the server should be restarted. While periodically monitoring servers, the Node Manager may determine how often to trigger a health check, how long to wait for a response, and how to proceed if the server is deemed failed. The Node Manager may be controlled by an Administrative Server directly or by an external administrative agent. An administrative agent may control the Node Manager by interfacing with the Administrative Server. The Node Manager and AS may authenticate each other and encode their communications to each other for increased security.
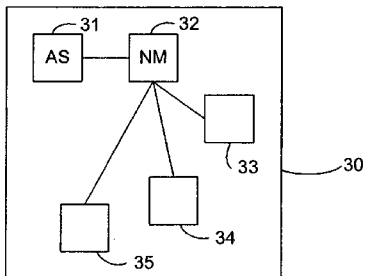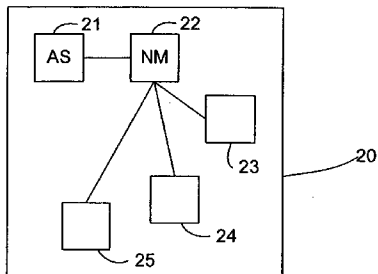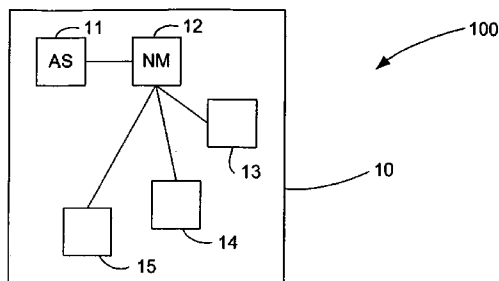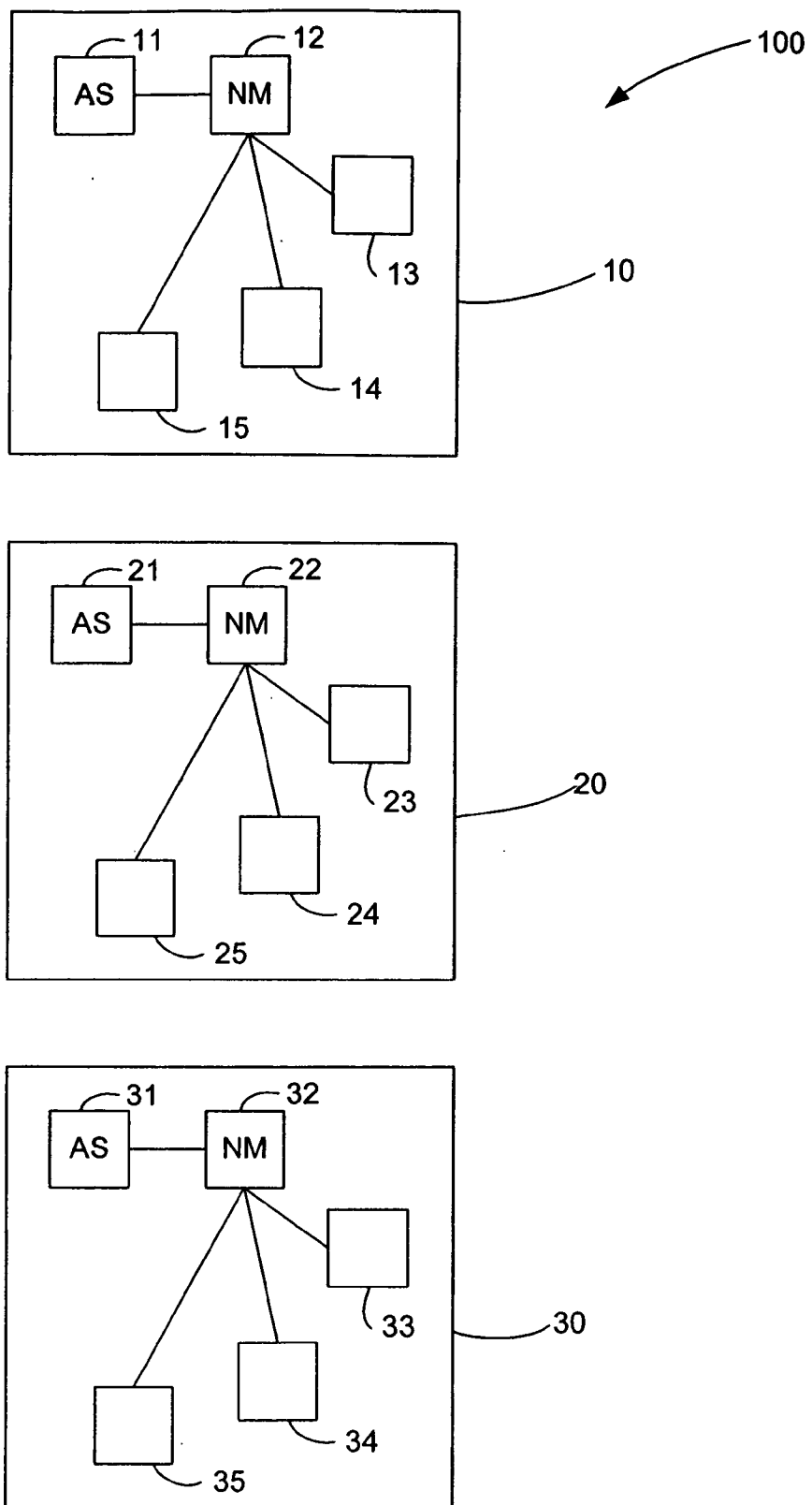
FIG. - 1

FIG. - 2

FIG. - 3

START — 410

400

INITIALIZE
SELF HEALTH
MONITOR — 420

430 — RECEIVE
HEALTH STATUS
INQUIRY

440 — PERFORM
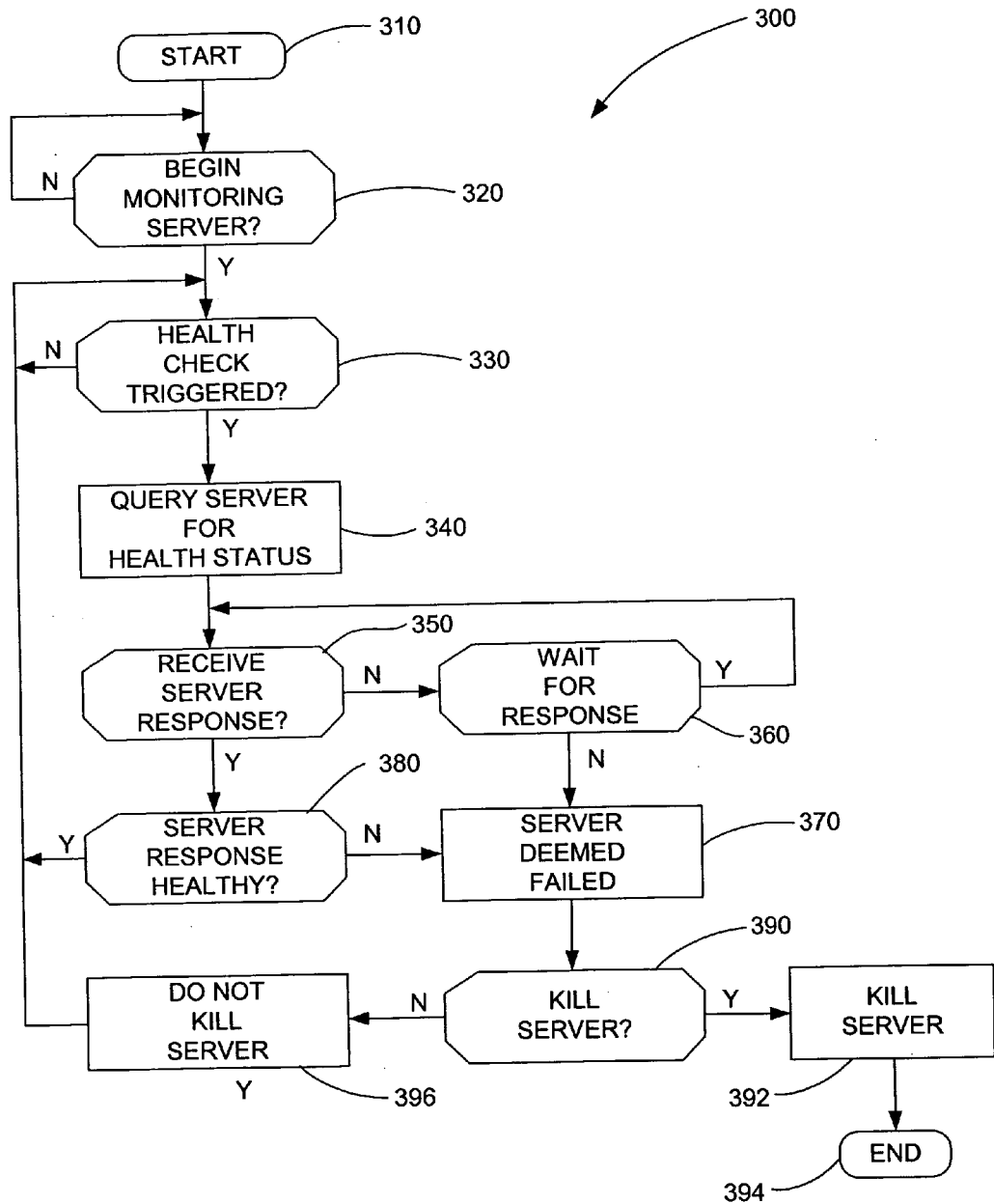SELF HEALTH
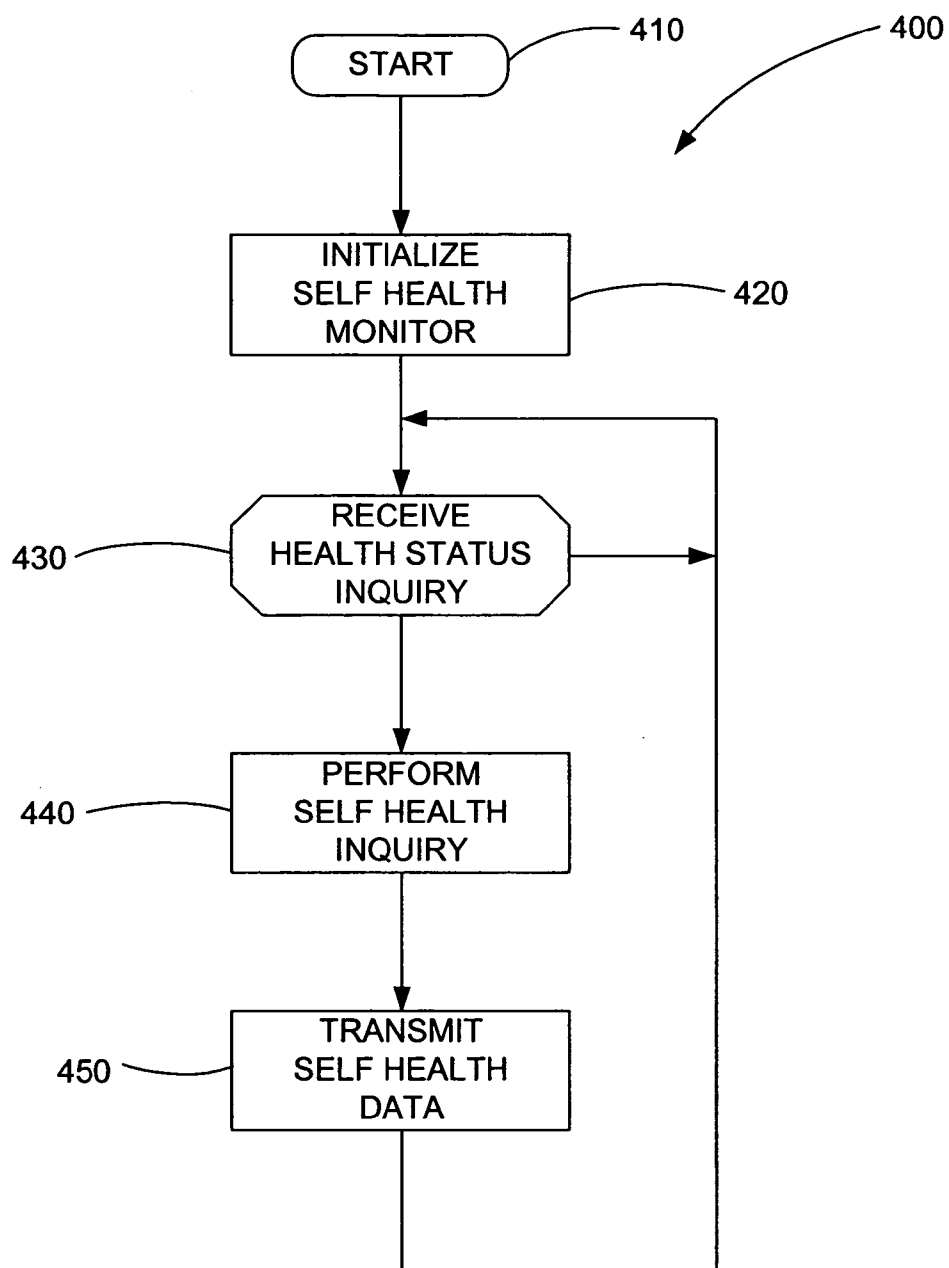INQUIRY
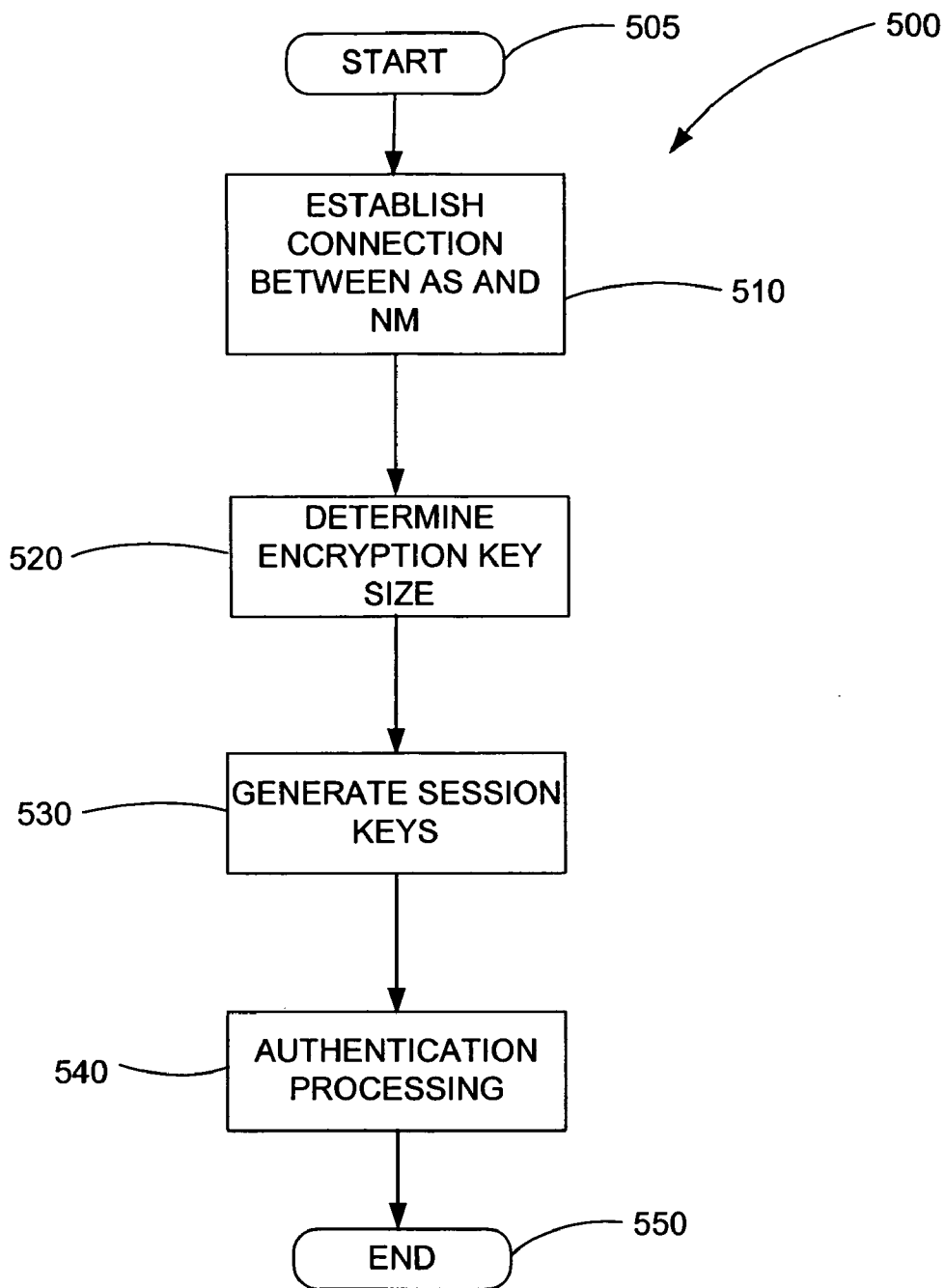
450 — TRANSMIT
SELF HEALTH
DATA

FIG. - 4

FIG. - 5

METHOD FOR EVENT TRIGGERED MONITORING OF MANAGED SERVER HEALTH

CLAIM TO PRIORITY

[0001] This application is a Continuation of U.S. patent application Ser. No. 10/338,981 entitled METHOD FOR EVENT TRIGGERED MONITORING OF MANAGED SERVER HEALTH, by Rahul Srivastava et al., filed Jan. 9, 2003 (Attorney Docket No: BEAS-01172US4), which claims priority under 35 U.S.C. §119(e) to:

[0002] U.S. Provisional Patent Application No. 60/359, 009 entitled ECS NODE MANAGER FOR ENSURING HIGH AVAILABILITY SERVER AND APPLICATION, by Rahul Srivastava et al., filed Feb. 22, 2002 (Attorney Docket Number BEAS-01172us0), which applications are incorporated herein by reference.

CROSS REFERENCE TO RELATED APPLICATIONS

[0003] The present application is related to the following United States Patents and Patent Applications, which patents/applications are assigned to the owner of the present invention, and which patents/applications are incorporated by reference herein in their entirety:

[0004] U.S. patent application Ser. No. 10/339,469 entitled METHOD FOR AUTOMATIC MONITORING OF MANAGED SERVER HEALTH, by Rahul Srivastava et al., filed on Jan. 9, 2003 (Attorney Docket Number BEAS-01172us3), which claims priority to U.S. Provisional Patent Application 60/359,009 entitled ECS NODE MANAGER FOR ENSURING HIGH AVAILABILITY SERVER AND APPLICATION, by Rahul Srivastava et al., filed Feb. 22, 2002 (Attorney Docket Number BEAS-01172us0);

[0005] U.S. patent application Ser. No. 10/339,144 entitled SYSTEM FOR MONITORING MANAGED SERVER HEALTH, by Rahul Srivastava et al., filed Jan. 9, 2003 (Attorney Docket Number BEAS-01172us5), which claims priority to U.S. Provisional Patent Application 60/359,009 entitled ECS NODE MANAGER FOR ENSURING HIGH AVAILABILITY SERVER AND APPLICATION, by Rahul Srivastava et al., filed Feb. 22, 2002 (Attorney Docket Number BEAS-01172us0);

[0006] U.S. patent application Ser. No. 10/340,496 entitled METHOD FOR INITIATING A SUB-SYSTEM HEALTH CHECK, by Rahul Srivastava et al., filed Jan. 10, 2003 (Attorney Docket Number BEAS-01183us1), which claims priority to U.S. Provisional Patent Application No. 60/359,010 entitled Server Self-Health Monitor, by Rahul Srivastava et al., filed Feb. 22, 2002 (Attorney Docket Number BEAS-01183us0);

[0007] U.S. patent application Ser. No. 10/340,227 entitled METHOD FOR MONITORING A SUB-SYSTEM HEALTH, by Rahul Srivastava et al., filed on Jan. 10, 2003 (Attorney Docket Number BEAS-01183us2), which claims priority to U.S. Provisional Patent Application No. 60/359, 010 entitled Server Self-Health Monitor, by Rahul Srivastava et al., filed Feb. 22, 2002 (Attorney Docket Number BEAS-01183us0); and

[0008] U.S. patent application Ser. No. 10/340,002 entitled SYSTEM FOR MONITORING A SUBSYSTEM HEALTH, by Rahul Srivastava et al., filed on Jan. 10, 2003 (Attorney Docket Number BEAS-01183us3), which claims priority to U.S. Provisional Patent Application No. 60/359, 010 entitled Server Self-Health Monitor, by Rahul Srivastava et al., filed Feb. 22, 2002 (Attorney Docket Number BEAS-01183us0).

Copyright Notice

[0009] A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

FIELD OF THE INVENTION

[0010] The present invention relates generally to managing a network of servers, and more particularly to monitoring the health of a network of servers.

BACKGROUND

[0011] As computer and computer systems have evolved over the years, the processes they implement have evolved in their complexity. One approach to implementing computer processes to solve more complex problems is to assign a number of computers to handle different parts of a process. Each part or task may be handled by different computers, computer objects, applications, or servers, hereafter referred to collectively as servers. These servers make up a distributed network. Within the network, different servers may handle functions such as management, data base maintenance, accessibility, server boot-up, shut-down, and so forth.

[0012] Servers within a distributed network perform transactions with other servers and use resources within the system. As the servers require the use of other servers and resources, the operability and reliability of the servers become more important. If a server fails while performing a task, it may affect other servers and resources that were tied up in transactions with the server at the time of its failure. Whether a server has failed completely or the server's condition has degraded is important information to a network. Thus, it is important to know the status of a server in order to maintain the health of the server and the network in which it operates. A maintenance system should be able to require a server to provide health information and be able to maintain or correct servers not operating properly.

[0013] What is needed is a system for monitoring and inquiring into the health of a server and for taking corrective action if deemed appropriate.

SUMMARY

[0014] In one embodiment of the present invention, a Node Manager (NM) monitors the status of multiple servers. The NM detects server failures, periodically monitors server health status, and performs server maintenance. When the NM detects a server failure, it determines whether or not the server is restartable. If the server is restartable, the NM checks to see if any other conditions exist that limit the server from being restarted. If no other conditions exist, the server is restarted. If the failed server is not restartable or

other conditions exist preventing the server from being restarted, the failed server is not restarted.

[0015] In another embodiment of the present invention, the NM periodically monitors the health of a server whether or not the NM detects a server failure. This process begins when the NM makes a health query to a server. Then, the NM waits for a server response containing the server's health information. If the server replies that it is healthy, the NM continues to monitor the server. If the server's reply indicates the server's health is degenerate or the server does not reply at all, the NM presumes the server has failed. The NM may wait a specified period of time before deciding the server has failed to respond to a query. Once a server is deemed failed, the NM then determines whether to terminate the server.

[0016] The NM may be controlled by parameters located within the NM or Administration Server (AS). The parameters may be burned into system software or modified at runtime. In another embodiment, the NM may be controlled by an external administrative agent. An administrative agent may control the NM by interfacing with the AS. For increased security, the NM and AS may authenticate each other and encode their communications between each other.

BRIEF DESCRIPTION OF THE DRAWINGS

[0017] FIG. 1 is a block diagram of several nodes having servers in a self health monitoring system in accordance with one embodiment of the present invention.

[0018] FIG. 2 is a diagram showing the operation of the automatic monitoring system of a Node Manager in accordance with one embodiment of the present invention.

[0019] FIG. 3 is a diagram showing the operation of a health monitoring and corrective action system of a Node Manager in accordance with one embodiment of the present invention.

[0020] FIG. 4 is a diagram showing the operation of a managed server in a health monitoring system in accordance with one embodiment of the present invention.

[0021] FIG. 5 is a diagram showing an encryption method for a self health monitoring system in accordance with one embodiment of the present invention.

DETAILED DESCRIPTION

[0022] A self health monitoring system may be composed of several nodes. A node may be a single physical machine or take some other form. In one embodiment of the present invention, each node has a Node Manager (NM), an Administration Server (AS), and several other managed servers or server instances. The AS and NM may send and transmit messages to each other. The NM may also send and transmit messages with the other servers located on the node.

[0023] In one embodiment, the NM performs two primary functions. First, the NM automatically detects and restarts failed servers. The NM continually monitors servers running on the local machine. Upon detecting a server has failed, the NM will automatically restart the failed server. The server restart may occur as soon as the NM detects the server failure. Secondly, the NM periodically monitors and restarts failed or degenerate servers. The NM will periodically monitor servers running on the local machine. When the NM

detects that a server is less than healthy, the NM may restart the server depending on server parameters and the condition of the server. In one embodiment, runtime java MBeans hosted on the AS are used in conjunction with the NM to achieve these functions. The runtime java MBeans offer the functionality of the NM and the health information acquired by the NM to clients in a programmatic manner.

[0024] FIG. 1 depicts a self health monitoring system 100 in accordance with one embodiment of the present invention. As shown, system 100 includes a first node 10, a second 20, and a third node 30. Each node may contain an AS 11, 21, and 31, and an NM 12, 22, and 32, respectively. In each node, the AS communicates with the NM. In one embodiment of the present invention, the AS and the NM communicate through a (SSL) secure socket layer connection. Each node also contains at least one managed server. In one embodiment, these managed servers may be composed of server instance processors or logic servers all located on one hardware machine. Hereinafter, the term "server" shall be understood to include server instance processors, server instance logic, and other managed servers. A node may be one physical machine with servers that communicate with other servers on the same machine. As shown in FIG. 1, node 10 includes servers 13-15, node 20 includes servers 23-25, and node 30 includes servers 33-35. An NM may communicate with the servers within the particular NM's node. For example, NM 12 can communicate with servers 13, 14, 15, all within node 10. In one embodiment, the NM communicates with the servers within its node through a secure socket layer connection.

[0025] In accordance with one embodiment of the present invention, the operation of an automatic monitoring system for detecting failed servers in the self health monitoring system of FIG. 1 is shown in flow chart 200 of FIG. 2 and described as follows. The operation of an NM starts at step 205. Next, the NM undergoes start-up and configuration operations in step 210. In one embodiment of the present invention, the NM receives instructions from an AS at start-up. The AS may instruct the NM to start an instance on a local machine. The AS may also instruct the NM to provide information to the AS regarding servers previously monitored during previous monitoring periods by the NM. The NM may assume that all of the monitored servers are alive upon NM startup and sets each server state to "unknown". In step 220, the NM begins monitoring a server. In one embodiment, the server is monitored over an SSL connection established with the server. In another embodiment, the server is monitored over a plain text protocol connection or some other type of connection.

[0026] The NM determines if a server has failed in step 230. In one embodiment, the server failure is detected by a breakdown of the connection between the NM and the server. In these embodiments, the NM monitors the server by confirming the integrity of the connection between the NM and the server. When the server being monitored dies, the NM is notified accordingly. In one embodiment, the NM receives an IOException error when the server dies. The integrity of the connection may also be verified in other ways, all considered within the scope of the invention. If the NM does not detect a failed server, operation of the system returns to step 220 to continue monitoring the server.

[0027] If the NM does detect a failed server in step 230, the NM will determine if the server is restartable in step 240.

In one embodiment, a restart parameter specifies whether the server should be restarted upon detecting a server failure. The restart parameter may reside on the server, the NM, or in some other memory location. The parameter may be defined per server instance or for a number of servers. The parameter may also be modifiable at runtime though commands issued through the AS. If the server is not restartable, operation continues to step **250** and the server is not restarted. In one embodiment, a message is written to a log file indicating that the server is not restartable and no further action is taken by the NM towards the failed server. Once the event is recorded, the process ends at step **260** and the NM ceases monitoring the failed server. Though no further action is performed on the server at step **260**, the server may be monitored again beginning at step **220** if the server is restarted or at step **205** if the NM is restarted.

[0028] If the server is deemed restartable in step **240**, operation of the system continues to step **270**. At step **270**, the system checks to confirm that no other conditions exist to prevent the server from being restarted. If at step **270** any conditions exist preventing a server restart, then system operation proceeds to step **275**. In step **275**, an action or inaction is taken to address the condition that is preventing the server from being restarted. The action or inaction may be taken by either the NM, AS, or some other server. After the condition is addressed in step **275**, the system determines whether the condition is satisfied in step **280**. If the condition is not satisfied in step **280**, operation returns to step **275**. If the condition is satisfied in step **280**, operation continues to step **270**. In one embodiment of the present invention, operation continues from step **275** to step **270** whether the condition is satisfied or not. In yet another embodiment of the present invention, the NM will check to see if the server may be restarted after each time a restart is considered in step **270**. In this embodiment (not shown), operation continues from step **275** or step **280** to step **240**. In any case, the results of step **280** may be recorded in a log or memory as either condition satisfied, condition not satisfied, or some other message. If at step **270** the conditions are satisfied, then operation continues to step **290**.

[0029] In one embodiment, system conditions may exist at step **270** that limit the server to a maximum number of restarts allowed during a period of time. In this case, parameters may control the maximum number of server restarts permitted within the period of time and the length of the time period. The number of actual restarts for a particular server may be indicated by a restart counter. If at step **270**, the value in the maximum restarts allowed parameter is larger than the restart counter, then the maximum number of restarts has not occurred during the current time period and the process continues to step **290**. If the restart counter value is larger than the maximum number of server restarts permitted within the particular time period, then operation continues to step **280**. Operation of the system may remain at step **280** until the current time period has elapsed. Once the time period has elapsed, the restart counter is reset to zero and the time period begins again. The system then continues to step **270**. At step **270**, the restart counter is again compared to the maximum number of restarts parameter and operation continues accordingly. In another embodiment, system operation will continue past step **280** even though the maximum start parameter has been exceeded. In this case, a message is logged regarding this event and operation continues. System operation in this

embodiment will consist of a loop between step **270** and step **280** until the time period has elapsed and the restart counter is reset to zero.

[0030] At step **290**, the server is restarted. Then, the NM, server, or AS may perform actions or process events. In one embodiment, the server restart counter is incremented. In another embodiment, certain server parameters may be configured to take effect upon the next server incarnation.

[0031] Parameters determine how a NM is to act upon detecting server failure. Examples of these parameters in one embodiment of the present invention are shown below. These parameters can be defined per server instance and modified at runtime via the Admin Console.

[0032] AutoRestartEnabled =<true|false>

[0033] This parameter specifies whether the servers are restartable or not. In one embodiment, the default is true.

[0034] RestartIntervalSeconds=<number of seconds>

[0035] If a Server is restartable, this parameter specifies that it can be restarted RestartMax times within the specified number of seconds. In one embodiment, the default is 3600 seconds (60 minutes).

[0036] RestartMax=<number>

[0037] If Servers are restartable, this parameter specifies the max # times a Server can be started within the period specified by RestartIntervalSeconds. In one embodiment, the default is 2.

[0038] Certain methods implemented in java may be used to access or modify the parameter examples listed above. Examples of these methods include boolean getAutoRestart-Enabled( ), void setAutoRestartEnabled(boolean), int getRe-startIntervalSeconds( ), void setRestartIntervalSeconds(int), int getRestartMax( ), and void setRestartMax(int).

[0039] In addition to detecting the failure of a server, the NM may monitor the health of a server or perform maintenance on a server. The NM may monitor server health or perform server maintenance without detecting a change or degradation in the health status of the server. Server maintenance and monitoring may be performed simultaneously on multiple servers at any time. The simultaneous monitoring and maintenance may be synchronous or asynchronous. The operation of a system for monitoring the health of a server with a NM in accordance with one embodiment of the present invention is shown in diagram **300** of **FIG. 3** and described as follows. Health monitoring system operation **300** starts off with a start step **310**. Next, the system determines whether the NM should begin monitoring a server in step **320**. If the system determines the NM should monitor the particular server, operation continues to step **330**. If the system determines the particular server should not be monitored at the current time, the NM will not monitor the current server. In one embodiment, a server will not be monitored until a period of time has passed since the server has been restarted. In this case, a monitor delay parameter will determine the period of time the NM shall wait before monitoring the restarted server. The delay parameter may be stored by the AS, NM, or the server itself. The delay parameter may correspond to a particular server or several servers. In one embodiment of the present invention, the value of the delay parameter may be modified at server runtime.

[0040] Next, the NM determines if a health check is triggered for a particular server in operation **330**. A health check may be triggered by an internal event in the NM. In another embodiment, the health check is triggered by an external event occurring outside the NM, the occurrence of which is communicated to the NM. In one embodiment, a health check is triggered for a server after a period of time has elapsed. In this case, a health check interval parameter may specify a period of time. The expiration of the time period specified by the interval parameter will trigger a health check for a server. In one embodiment, an interval parameter corresponds to a single server. In another embodiment, an interval parameter corresponds to several servers. In any case, the interval parameter may be modified at server runtime. If a health check is not triggered in step **330**, the system continues in a standby state waiting for a triggering event to occur regarding a particular server. If a health check triggering event does occur, system operation continues to step **340**.

[0041] In step **340**, the NM queries a server for it's health status. In one embodiment of the present invention, the NM invokes a java servlet located on the server to return the server's health status to the NM. This java servlet is an asynchronous servlet that will return the server's health information upon the occurrence of a particular event. In one embodiment, the event is the elapse of a period of time. The NM may inquire about the server's health status by communicating with the server itself or a server self health monitor application running on the server. The query may be transmitted over a TCP connection established between the NM and server or in some other manner. After querying the server for it's health status, the NM determines if a response is received from the server in step **350**. In one embodiment, there are at least three possible response scenarios between the NM and the server subject to the NM's inquiry. In the first scenario, the server may be unable to receive the NM's query. The server may be too busy to accept a connection from the NM. In another scenario, the server may have failed and be unable to accept an NM connection request. In either case, the NM may throw an IOException and consider the server as "failed". The NM would then set an internal value of the server state to "failed". In the final scenario, no response is received from the server although the NM and server have established an initial connection. In this case, the NM will wait for a response from the server for a specified period of time. In one embodiment, a timeout parameter may specify the period of time the NM will wait for a response from the server. Until the length of time specified in the timeout parameter has transpired, the NM will continue to wait for a response as indicated in the loop of steps **350** and **360**. If the NM has not received a response from the server in step **350** and the NM has determined not to wait any longer to receive a response in operation **360**, operation continues to step **370** where the server is deemed failed. In one embodiment, the NM may attempt to inquire about the delay of the response or resend a health inquiry to the server before proceeding from to **370**. In this embodiment, the NM may proceed to step **350**, **360**, or **380** depending on the result of the delay inquiry or the health inquiry.

[0042] If the NM does receive a response in step **350**, operation flows to step **380** where the NM interprets the server's response. The NM interprets the server's response to determine if the server is healthy. If the NM determines the server is healthy from the response received by the server, operation flows to step **330** where the NM waits for another health check to be triggered. If the NM determines that the server is not healthy in step **380**, operation continues to step **370**. In step **370**, the NM deems the server has failed. In one embodiment, the NM sets a parameter indicating the state of the particular server to "failed". The parameter may be stored internally within the NM, in the AS, or at some other memory location. Once deemed failed, operation continues to step **390** where the NM determines whether to terminate the server. In one embodiment, the NM contains an auto-terminate parameter. The auto-terminate parameter may relate to a single server or multiple servers at once. A user may set a value for the auto-terminate parameter or the parameter may be preset by the system. If the auto-terminate parameter indicates the server should not be terminated upon server failure, then operation continues to step **396**. In one embodiment of the present invention, the system enters a message in a log indicating the failed status of the server and that the server is not to be restarted. After step **396**, system operation proceeds to step **330**. If the auto-terminate parameter indicates the server should be terminated upon server failure in step **390**, then operation continues to step **392**. The failed server is terminated in step **392**. In one embodiment of the present invention, an entry is made to a log indicating the server is deemed failed and that the server was terminated. Monitoring of the terminated server ends in step **394**. Once the server is terminated, the automatic detection system of **FIG. 1** may detect the terminated server at step **230**. The NM may then proceed to determine whether to restart the server as shown in **FIG. 1**.

[0043] In one embodiment of the present invention, certain parameters will control how the server periodically checks the servers running on the local machine. Examples of parameters controlling the check are shown below.

[0044] HealthCheckIntervalSeconds =<number of seconds>

[0045] This parameter specifies the interval of time (in seconds) between which periodic scans are done by NM to check if Servers are Failed. In one embodiment, the default is 180 seconds.

[0046] HealthCheckTimeoutSeconds =<number of seconds>

[0047] This parameter specifies the length of time (in seconds) the Node Manager will wait for a response to the health check query after which it will deem the monitored server Failed. In one embodiment, the default is 60 seconds.

[0048] AutoKillIfFailedEnabled =<true|false>

[0049] This parameter specifies if a Server is deemed Failed, this parameter will control whether NM will kill the Server or not. In one embodiment, the default is false.

[0050] HealthCheckStartDelaySeconds =<number of seconds>

[0051] The time that a server takes to startup depends upon the applications being deployed on it. The NM will wait for a server to complete its startup before the NM starts monitoring the server. This parameter specifies the length of time (in seconds) the NM will wait before starting its monitoring of the server. In one embodiment, the default is 120 seconds.

[0052] The HealthCheckIntervalSeconds and Health-CheckTimeoutSeconds and parameters can be defined per NM and on the NM command line. The AutoKillIfFailedEnabled and HealthCheckStartDelaySeconds parameters can be defined per server instance and can be modified at runtime via the Admin Console. These new parameters for the Server will be modifiable at runtime via the Admin Console. In conjunction with the parameter examples above, methods implemented in java code can be added to the server MBean and may be used to access or modify the parameters. Examples of these java methods include boolean getAutoKillIfFailedEnabled( ), void setAutoKillIfFailedEnabled(boolean), int getHealthCheckStartDelaySeconds( ), and void setHealthCheckStartDelaySeconds (int secs).

[0053] In one embodiment, the NM may allow its functionality and access to server health information to become available to external administrative clients. External administrative clients such as third party application monitors and high availability frameworks may need to be able to start and kill servers using the functionality of the NM. In one embodiment of the present invention, this is done programmatically with runtime MBeans. Use of an admin console is not required. The MBeans provide a programmatic interface to the NM's functionality. Further, the MBeans allow the NM's record of a server's health to be shared. Internal or external administrative clients may use the MBeans to access server health information collected by the NM. In one embodiment, the AS hosts [one]a NodeManagerRuntime MBean that provides methods to accomplish different tasks. Each machine may have one such MBean. [One task may involve starting a specified server.] In one embodiment, the AS may have methods according to the examples shown below.

[0054] java.io.reader start(serverMBean server) throws NodeManagerException;

[0055] This method starts the specified server. It then returns the reader to local log file containing output of executed command. The method throws NodeManagerException if any error occurs.

[0056] java.io.reader starInStandby (serverMBean server) throws NodeManagerException;

[0057] This method starts the specified server in Standby Mode. It then returns the reader to local log file containing output of executed command. The method throws NodeManagerException if any error occurs.

[0058] java.io.Reader shutdown (ServerMBean server) throws NodeManager-Exception;

[0059] This method shuts down the specified server. It then returns the reader to local log file containing output of executed command. The method throws NodeManagerException if any error occurs.

[0060] java.io.reader kill(ServerMBean server) throws NodeManagerException;

[0061] This method kills specified server. It is used to kill the server process when the server does not respond to shutdown operation. It then returns the reader to local log file containing output of executed command. The method throws NodeManagerException if any error occurs.

[0062] java.io.reader startMonitoring (ServerMBean server) throws NodeManagerException;

[0063] This method instructs the NM to start monitoring the specified server. The NM will automatically restart the server if it crashes (if auto restartEnabled is set to true) or gets into failed state (if AutoKillIfFailedEnabled and AutoRestartEnabled are set to true). It then returns the reader to local log file containing output of executed command. The method throws NodeManagerException if any error occurs.

[0064] java.io.reader stopmonitoring (serverMBean server) throws NodeManagerException;

[0065] This method instructs the NM to stop monitoring the specified server. It then returns the reader to local log file containing output of executed command. The method throws NodeManagerException if any error occurs.

[0066] java.io.Reader getlogs(Server MBean server, String type) throws NodeManagerException;

[0067] This method get logs from the NM for the specified server. The type is either "WL_output" or "WL_Error". It then returns the reader to local log file containing output of executed command. The method throws NodeManagerException if any error occurs.

[0068] string get state (ServerMBean server) throws NodeManagerException;

[0069] This method queries the NM for its view of the specified server state. It is used when the server does not respond to queries to its ServerRuntimeMBean. The method will return "unknown" if NM is either not monitoring the server or does not have any knowledge of the server. It then returns the reader to local log file containing output of executed command. The method throws NodeManagerException if any error occurs.

[0070] In another embodiment, MBeans may provide an interface for JMX clients to access the functionality of the NM. In this case, the MBeans for JMX client interfacing may have a different interface than the Server configuration MBeans. Operations such as "start" and "shutdown" may return precise information on their success or failure. They will throw an exception if the operation fails. All operations on the Node Manager Runtime MBeans may be blocking. A TaskMBean interface may be provided around the Server Lifecycle MBeans to provide an asynchronous interface to JMX clients. JMX clients can make use of the NM functionality to perform a wide variety of Server lifecycle and health monitoring control operations. Detailed below are the interactions between these two entities during each of the Server Lifecycle state transitions. Admin console, weblogic.Admin command line utility and other Admin Clients will be effecting these state transitions by invoking methods on the ServerLifecycleRuntimeMBean.

[0071] start( ) [SHUTDOWN->RUNNING]

[0072] startInStandby( ) [SHUTDOWN->STANDBY]

[0073] ServerLifecycleRuntimeMBean hosted on the Admin Server will invoke the start( ) or startInStandby( ) methods on the corresponding NodeManagerRuntimeMBean

[0074] shutdown( ) [STANDBY->SHUTDOWN]

[0075] If a NM is configured, ServerLifecycleRuntimeMBean hosted on the Admin Server will invoke the shutdown( ) method on the corresponding NodeManagerRuntimeMBean. If not, it will invoke the shutdown( ) method on the

ServerLifecycleRuntimeMBean hosted on the Managed Server.

[0076] getState( )

[0077] ServerLifecycleRuntimeMBean hosted on the Managed Server will return the State attribute of the Server. ServerLifecycleRuntimeMBean hosted on the Admin Server will invoke the getState( ) method on the ServerLifecycleRuntimeMBean hosted on the Managed Server. If this operation times out, it will then invoke the getState( ) method on the NodeManagerRuntimeMBean.

[0078] The operation of a server that is monitored by a NM in accordance with one embodiment of the present invention is shown in diagram 400 of FIG. 4 and described as follows. The operation of flow chart 400 starts off with a start step 410. Next, the server is initialized in step 420. In one embodiment, the NM and the server establish a connection as part of the server initialization. While establishing the connection and initializing the server, the NM may pass the NM's listening address to the server. In one embodiment, server initialization includes the server initializing and running a program that monitors its health and interfaces with query attempts from the NM and other sources. Once a connection between the NM and the server is established and the server is initialized, the server may send a message to the NM indicating the server experienced a successful start up.

[0079] After initialization, the server listens for an inquiry regarding the server's health status in step 430. The health status inquiry may come from the NM or an external administrative agent. If no health status inquiry is received, the server continues to listen for an inquiry as shown in FIG. 4. If a health status inquiry is received, operation continues to step 440. In step 440, the server performs a self health check on itself. Next, the results of the health check are transmitted by the server in step 450. In one embodiment, the results are transmitted to the NM or the AS. In another embodiment, the results are transmitted according to the instructions of an external administrative agent. The administrative agent may have the results sent to the agent itself or some other entity. After transmitting the results of the self health inquiry, operation of the server returns to step 430 where the server listens for a health status inquiry. In one embodiment, if at any point the server fails, the server will inform the NM as soon as it enters a failed state.

[0080] The communication between the NM and the AS may be encoded to increase the integrity of the system. In one embodiment of the present invention, the communication may be encoded according to a Diffie-Helman based Authentication and Encryption scheme. The encryption parameters may be negotiated at link establishment time and depend upon configuration parameters and installed software. A flow chart showing the operation of a Diffie-Helman based Authentication and Encryption scheme 500 in accordance with one embodiment of the present invention is shown in FIG. 5. The operation starts at step 505. Next, the connection between the AS and NM is established in operation 510. In one embodiment, the connection between the AS and the NM is initialized by the AS. Once the connection has been initialized by the AS, the NM receives the initial connection. Next, the encryption key size is determined in step 520. In one embodiment, the encryption key size is determined by a negotiation between the AS and the NM.

The AS and NM each have a minimum key length parameter and maximum key length parameter. The pair for each of the NM and AS is denoted as (min, max). The minimum key length parameter is the minimum encryption level a process will support. The maximum key length parameter is the maximum encryption level a process is will support. In one embodiment, the possible key lengths are 0, 40, or 128 bits. The AS and NM will negotiate a connection that uses an encryption level as high as the lowest maximum key length between the NM and AS, but no lower than the highest minimum key length between the NM and AS. For example, if the AS has key length parameters of (0, 128), and the NM has key length parameters of (40, 128), the connection may have a key length of 40 or more preferably 128 bits. The key length may not be 0 bits because the NM has a minimum key length parameter of 40 bits. Once the key length for the connection has been established, the key length is in effect for the lifetime of the connection between the AS and NM. In one embodiment, the maximum key length parameter may be reduced by the capabilities of software installed on the NM or the AS. If the minimum key length parameter for either the AS or NM is higher than the maximum key length parameter for the other of the AS or NM, then no overlap exists in key length parameters. If there is no overlap in key length parameters, the established link will fail and an appropriate error message is logged. In one embodiment, command line arguments may be used to specify the minimum encryption level parameter and maximum encryption level parameter.

[0081] After the key size is determined, the session keys are generated in step 530. In one embodiment, a first session key is used for transmitted data from the AS to the NM and a second session key is used from for data transmitted from the NM to the AS. In one embodiment of the present invention, the session keys are 128 bits. The input parameters to the Diffie-Hellman key agreement algorithm may be fixed or generated. In one embodiment, the input parameters are fixed into software existing on the server network and accessible to the AS. The AS server may transmit the fixed input parameters to the NM. The Diffie-Hellman algorithm also requires the generation of a random number. In one embodiment, the random number is generated from a cryptographically secure pseudo-random number source. An RC4 bulk encryption method may be used as the encryption method for the link. The details of generating a session key using the Diffie-Hellman algorithm are generally known in the art and therefore not described here in detail.

[0082] Once the session keys are generated in step 530, the NM and AS may engage in an authentication process at step 540. In one embodiment, both the NM and AS are authenticated using a shared secret. One method of authenticating both the NM and AS in accordance with the present invention is as follows. First, both the NM and AS will generate a fingerprint. In one embodiment, the fingerprint may be a 128 bit MD5 message digest created using the send and receive session keys already negotiated. Next, the AS will generate a challenge, encrypt the challenge and fingerprint, and send the encrypted challenge and fingerprint to the NM. In one embodiment, the challenge may be a 64 bit random number. Next, the NM will receive and decrypt the challenge and fingerprint from the AS. The information may be encrypted and decrypted using the AS's password. If the information received by the NM is decrypted and does not match the NM's fingerprint, the NM will reject the authen-

tication request by the AS. Next, the NM will encrypt the challenge received from the server, the session key generated by the NM, and the fingerprint generated by the NM. The NM will then send the encrypted challenge, session key, and fingerprint to the AS. The AS will receive and decrypt the information received from the NM. Upon decrypting the received information, the AS will compare the received challenge and fingerprint to it's own challenge and fingerprint. If either the fingerprints or the challenges do not match, the AS will reject the authentication request from the NM. If the comparisons performed by the AS and NM reveal matching information, then the authentication requests will be accepted. The encryption and authentication process then ends at step **550**. The encryption by the AS and NM may be done using a DES encryption method or some other method suitable for the particular requirements of the system.

[0083] In one embodiment, new command line arguments are specified for the AS and NM regarding AS/NM communication encryption and authentication. These new arguments can be modified at runtime via the Admin Console. Arguments specified for the AS may include the examples listed below.

[0084] WebLogic.management.minEncryptionBits =<number>

[0085] This argument specifies the minimum encryption level the process will accept. Possible values are 0, 40, 128. In one embodiment, the default value is 0.

[0086] WebLogic.management.maxEncryptionBits =<number>

[0087] This argument specifies the maximum encryption level the process will accept. Possible values are 0, 40, 128. In one embodiment, the default value is 128.

[0088] WebLogic.management.enableChannelBinding= 0|1

[0089] This argument sets an Authentication Protocol. In one embodiment, the a value of 1 enables the Authentication Protocol and the default is 0.

[0090] WebLogic.management.passwordKey =<string>

[0091] This argument specifies the key to be used to decrypt the encrypted NM passwords stored in the configuration file. The key must be specified if WebLogic.management.enableChannelBinding is set to 1.

[0092] The utility WebLogic.wtc.gwt.genpasswd will be used to encrypt NM passwords to be stored in the configuration file.

[0093] Arguments specified for the NM may include the examples listed below.

[0094] WebLogic.nodemanager.minEncryptionBits =<number>

[0095] This argument specifies the minimum encryption level the process will accept. Possible values are 0, 40, 128. In one embodiment, the default value is 0.

[0096] WebLogic.nodemanager.maxEncryptionBits =<number>

[0097] This argument specifies the maximum encryption level the process will accept. Possible values are 0, 40, 128. In one embodiment, the default value is 128.

[0098] WebLogic.nodemanager.enableChannelBinding= 0|1

[0099] This argument sets an Authentication Protocol enable. In one embodiment, a value of 1 enables the Authentication Protocol and the default is 0.

[0100] WebLogic.nodemanager.password =<string>

[0101] This argument specifies the NM password. The NM password must be specified if WebLogic.nodemanager-.enableChannelBinding is set to 1.

[0102] WebLogic.nodemanager.adminServerPassword =<string>

[0103] This argument specifies the Admin Server password. Must be specified if WebLogic.nodemanager.enableChannelBinding is set to 1.

[0104] In one embodiment, an alternate NM may have features in addition to those described above. In particular, the alternate NM may function to aggregate administrative actions and information in the NM. The actions and information could then be accessed by third party application monitors and high availability frameworks using standard JMX interfaces. The NM may also achieve internal interactions with the admin console and cluster group leader using standard JMX interfaces.

[0105] In one embodiment, certain services are required by the administrator for the alternate NM to operate properly. One such service is a ProcessControl ("PC") service. The PC service operates to start, kill and restart managed servers on the local node. The service can be hosted by the AS and by the NM on the other nodes. Restart capability is provided to internal and external clients via Runtime MBeans. Another service is the HealthMonitoring ("HM") service. The HS service monitors state and other runtime attributes of managed servers on local or remote nodes. Either the AS or an individual NM can host the HM service. Server health information is provided to internal and external clients via Runtime MBeans.

[0106] The alternate NM has several advantageous characteristics. In one embodiment, some aspects of the configuration for the alternate NM are similar to the basic NM of the present invention. The alternate NM may be hosted on a stripped-down managed server. In this case, the alternate NM registers as a managed server with the AS. With this configuration, the NM performs startup independently of the AS. Further, no configuration MBeans are required for runtime configuration changes

[0107] The alternate NM may include a "watchdog" service. The watchdog service operates to monitor the NM on platforms where operating system monitoring is not available, such as non-NT and Solaris platforms. The watchdog service may be configured to spawn the NM when it performs startup, thereby allowing administrators to manually start just one process. In one embodiment, a system could use a watchdog service to bootstrap the NM service on a local node upon command of an AS. This watchdog service configuration would eliminate manual configuration of the NM on each remote node and allow runtime configuration through configuration MBeans implemented in java. For Solaris and NT systems, the NM may be used as the operating system in one embodiment of the present inven-

tion. In this case, the NM may be installed on a node in conjunction with other software and can be started manually.

[0108] The alternate NM can be configured to enable its operation to enhance aspects of an entire domain or local node. One NM could be used for an entire domain. This would simplify security in that multiple certificates would not need to be managed. In one embodiment, the NM is the highest release. The alternate NM may be used to aggregate cluster heartbeats for all cluster members on a local node. The cluster members may include managed servers on the local node. The alternate NM may also be configured to operate as a surrogate AS.

[0109] The present invention includes a Node Manager that monitors the status of multiple servers. The NM detects server failures, periodically monitors server health status, and performs server maintenance. When the NM detects a server failure, it determines whether or not the server should be restarted. While periodically monitoring servers, the NM may determine how often to trigger a health check, how long to wait for a response, and how to proceed if the server is deemed failed. The NM may be controlled by an AS directly or by an external administrative agent. An administrative agent may control the NM by interfacing with the AS. The NM and AS may authenticate each other and encode their communications to each other for increased security.

[0110] An Appendix is attached to this application containing examples within the scope and spirit of the present invention.

[0111] In addition to an embodiment consisting of specifically designed integrated circuits or other electronics, the present invention may be conveniently implemented using a conventional general purpose or a specialized digital computer or microprocessor programmed according to the teachings of the present disclosure, as will be apparent to those skilled in the computer art.

[0112] Appropriate software coding can readily be prepared by skilled programmers based on the teachings of the present disclosure, as will be apparent to those skilled in the software art. The invention may also be implemented by the preparation of application specific integrated circuits or by interconnecting an appropriate network of conventional component circuits, as will be readily apparent to those skilled in the art.

[0113] The present invention includes a computer program product which is a storage medium (media) having instructions stored thereoii/in which can be used to program a computer to perform any of the processes of the present invention. The storage medium can include, but is not limited to, any type of disk including floppy disks, optical discs, DVD, CD-ROMs, microdrive, and magneto-optical disks, ROMs, RAMs, EPROMs, EEPROMs, DRAMs, VRAMs, flash memory devices, magnetic or optical cards, nanosystems (including molecular memory ICs), or any type of media or device suitable for storing instructions and/or data.

[0114] Stored on any one of the computer readable medium (media), the present invention includes software for controlling both the hardware of the general purpose/specialized computer or microprocessor, and for enabling the computer or microprocessor to interact with a human user or other mechanism utilizing the results of the present inven-

tion. Such software may include, but is not limited to, device drivers, operating systems, and user applications. Ultimately, such computer readable media further includes software for implementing Node Managers.

[0115] Included in the programming (software) of the general/specialized computer or microprocessor are software modules for implementing the teachings of the present invention, including, but not limited to, separating planes of a source image, averaging at least one of foreground and background colors, replacing colors, and compensating for error introduced by color replacement in one plane by feeding error into a second plane, storage, communication of results, and reconstructing an image according to the processes of the present invention.

[0116] Other features, aspects and objects of the invention can be obtained from a review of the figures and the claims. It is to be understood that other embodiments of the invention can be developed and fall within the spirit and scope of the invention and claims.

[0117] The foregoing description of preferred embodiments of the present invention has been provided for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Obviously, many modifications and variations will be apparent to the practitioner skilled in the art. The embodiments were chosen and described in order to best explain the principles of the invention and its practical application, thereby enabling others skilled in the art to understand the invention for various embodiments and with various modifications that are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following claims and their equivalence.

Appendix

1 Product Perspective (O)

[0118] The existing NM implementation enables the administrator to start and kill Servers remotely from the Administration Console. However, there is no automatic monitoring or restart of these Servers after that.

[0119] The goal of this release is to improve the availability of these Servers by monitoring them and automatically restarting them if necessary.

[0120] In addition, NM's functionality and information will be exposed to JMX clients via new runtime MBeans.

1.1 Product Functions (O)

1.1.1 Automatic Detection and Restart of Crashed Servers

[0121] NM will continuously monitor Servers running on the local machine and will automatically detect and restart failed Servers. This detection and restart will occur as soon as NM detects the Server failure.

1.1.2 Monitoring and Restart of Failed Servers

[0122] NM will periodically monitor Servers running on the local machine and will automatically detect and restart Failed Servers. This detection and restart will occur as soon as possible after the Server is deemed to be Failed.

1.1.3 Node Manager Runtime MBeans

[0123] New Node Manager Runtime MBeans will be provided. They will be hosted on the Admin Server and will offer methods that wrap NM functionality and expose the health information collected by NM.

2 Automatic Detection and Restart of Failed Servers

2.1 Functional Description

[0124] NM will continuously monitor Servers running on the local machine and will automatically detect and restart failed Servers. This detection and restart will occur as soon as NM detects the Server failure.

2.2 Functional Requirements

[0125] NM will continuously monitor Servers running on the local machine and will automatically detect and restart failed Servers. This detection and restart will occur as soon as possible after the Server failure.

[0126] Once a Server failure is detected, NM's actions will be controlled by these parameters:

[0127] AutoRestartEnabled=<true|false>

[0128] Specifies whether Servers are restartable or not. Default is true.

[0129] RestartIntervalSeconds=<number of seconds>

[0130] If a Server is restartable, this parameter specifies that it can be restarted RestartMax times within the specified number of seconds. Default is 3600 seconds (60 minutes).

[0131] RestartMax =<number>

[0132] If Servers are restartable, this parameter specifies the max # times a Server can be started within the period specified by RestartIntervalSeconds. Default is 2.

[0133] These parameters will be defined per Server instance. They will also be modifiable at runtime via the Admin Console.

2.3 Software Interfaces (Javadoc, MBean, Objects, Classes)

[0134] This feature has only an administrative interface, and it is via the new parameters described in section 3.2.

[0135] These new parameters for the Server will be modifiable at runtime via the Admin Console.

[0136] These methods will be added to the weblogic.management.configuration.ServerMBean to access or modify these parameters:

boolean getAutoRestartEnabled( );

void setAutoRestartEnabled(boolean);

int getRestartIntervalSeconds( );

void setRestartIntervalSeconds(int);

int getRestartMax( );

void setRestartMax(int);

3. Monitoring and Restart of Failed Servers

3.1 Functional Description

[0137] NM will periodically monitor Servers running on the local machine and will automatically detect and restart Failed Servers. This detection and restart will occur as soon as possible after the Server is deemed to be Failed.

3.2 Functional Requirements

[0138] NM will periodically check Servers running on the local machine and will automatically detect and restart Failed Servers. This detection and restart will occur as soon as possible after the Server is deemed to be Failed.

[0139] This check will be controlled by these parameters:

[0140] HealthCheckIntervalSeconds=<number of seconds>

[0141] The interval of time (in seconds) between which periodic scans are done by NM to check if Servers are Failed. Default is 180 seconds.

[0142] HealthCheckTimeoutSeconds=<number of seconds>

[0143] The length of time (in seconds) the Node Manager will wait for a response to the health check query after which it will deem the monitored server Failed. Default is 60 seconds.

[0144] AutoKillIfFailedEnabled=<true|false>

[0145] If a Server is deemed Failed, this parameter will control whether NM will kill the Server or not. Default is false.

[0146] The time that a Server takes to startup depends upon the applications being deployed on it. NM will wait for a Server to complete its startup before it (the NM) starts monitoring the Server. This wait time can be specified using the following parameter:

[0147] HealthCheckStartDelaySeconds=<number of seconds>

[0148] The length of time (in seconds) the Node Manager will wait before starting its monitoring of the Server. Default is 120 seconds.

[0149] HealthCheckIntervalSeconds and HealthCheckTimeoutSeconds and parameters will be defined per NM, and can be specified on the NM command line.

[0150] AutoKillIfFailedEnabled and HealthCheckStartDelaySeconds parameters will be defined per Server instance and will be modifiable at runtime via the Admin Console. After NM has killed a Failed server, its restartability is controlled by the parameters defined in section 3.2 above.

3.3 External Interface Requirements

3.3.1 Software Interfaces (Javadoc, MBean, Objects, Classes)

[0151] This feature has only an administrative interface, and it is via the new command line arguments described in section 4.2.

[0152] These new parameters for the Server will be modifiable at runtime via the Admin Console.

[0153] These methods will be added to the ServerMBean to access or modify these parameters:

boolean getAutoKillIfFailedEnabled( );

void setAutoKillIfFailedEnabled(boolean);

int getHealthCheckStartDelaySeconds( );

void setHealthCheckStartDelaySeconds(int secs);

4. Node Manager Runtime MBeans

4.1 Functional Description

[0154] These MBeans will serve the following purposes:

1. Expose NM Functionality to External Administrative Clients

[0155] External administrative clients (3rd party application monitors, HA frameworks, etc.) need to be able to start and kill Servers using the NM. They should be able to do this programmatically without using the admin console.

[0156] These MBeans will provide a programmatic interface to NM's functionality.

2. Expose NM's View of Server Health

[0157] As described in section 4 above, NM periodically collects health information on Servers. Internal or external administrative clients (e.g., admin console) need to be able to access this information.

[0158] These MBeans will provide an interface to this information.

4.2 Functional Requirements

[0159] The Admin Server will host the new "NodeManagerRuntimeMBean". There will be one NodeManagerRuntimeMBean per machine in the domain.

[0160] This MBean will offer methods that wrap NM functionality and expose the health information collected by NM.

[0161] The ServerLifecycleRuntimeMBean hosted on the Admin Server will use these MBeans internally. They will NOT be exposed to external JMX clients.

4.3 External Interface Requirements

4.3.2 Software Interfaces (Javadoc, MBean, Objects, Classes)

[0162] public interface NodeManagerRuntimeMBean extends WebLogic.management.runtime.RuntimeMBean

Method Summary

java.io.Reader start(ServerMBean server) throws NodeManagerException;

[0163] starts the specified Server.

[0164] Returns Reader to local log file containing output of executed command. Throws NodeManagerException if any error occurs.

java.io.Reader startInStandby(ServerMBean server) throws NodeManagerException;

[0165] starts the specified Server in Standby state.

[0166] Returns Reader to local log file containing output of executed command. Throws NodeManagerException if any error occurs.

java.io.Reader shutdown(ServerMBean server) throws NodeManagerException;

[0167] shuts down the specified Server.

[0168] Returns Reader to local log file containing output of executed command. Throws NodeManagerException if any error occurs.

java.io.Reader kill(ServerMBean server) throws NodeManagerException;

[0169] kills the specified Server. Used to kill the server process when the server does not respond to shutdown operation.

[0170] Returns Reader to local log file containing output of executed command. Throws NodeManagerException if any error occurs.

java.io.Reader startMonitoring(ServerMBean server) throws NodeManagerException;

[0171] Instruct NM to start monitoring the specified server. NM will automatically restart the server if

[0172] it crashes (if AutoRestartEnabled is set to true) or gets into Failed state (if AutoKillIfFailedEnabled and AutoRestartEnabled are set to true).

[0173] Returns Reader to local log file containing output of executed command. Throws NodeManagerException if any error occurs.

java.io.Reader stopMonitoring(ServerMBean server) throws NodeManagerException;

[0174] Instruct NM to stop monitoring the specified server.

[0175] Returns Reader to local log file containing output of executed command. Throws NodeManagerException if any error occurs.

java.io.Reader getLogs(ServerMBean server, String type) throws NodeManagerException;

[0176] Get logs from the NM for the specified server. The type is either "WL_output" or "WL_error".

[0177] Returns Reader to log retrieved. Throws NodeManagerException if any error occurs. String getState(ServerMBean server) throws NodeManagerException;

[0178] Query the NM for its view of specified server's state. Used when server does not respond to queries to its ServerRuntimeMBean.

[0179] Will return "Unknown" if NM is either not monitoring or does not have any knowledge of the server. Throws NodeManagerException if any error occurs.

5.1 Node Manager—Managed Server Communication

5.1.1 Health Monitoring Communication

[0180] NM will periodically poll the Server to check its health. The algorithm used in this polling is as follows:

[0181] 1. Upon startup, NM reads an on-disk file to retrieve information on Servers it was monitoring during its previous incarnation.

[0182] It assumes that all of the monitored Servers are alive when it starts up. It assumes no knowledge of their current States (i.e., it sets its view of their State to "Unknown").

[0183] 2. NM invokes the NMCommandServlet deployed on the Server. This is an asynchronous Servlet that will return the Server's health information after HealthCheckIntervalSeconds have elapsed.

[0184] 3. One of the following happens when NM invokes the Servlet:

3.1 IOException is Thrown.

[0185] This could mean a number of different things:

[0186] Server has crashed or is not running

[0187] Server too busy to accept TCP connection

[0188] Server has Failed, unable to accept TCP connection

[0189] Transient IO exception

[0190] All cases are treated as if Server has Failed. NM sets its internal value of Server state to "Failed".

[0191] To Handle "False Negatives", NM Kills the Server.

[0192] If the Server's AutoKillIfFailedEnabled parameter is "true", NM sets its internal value of Server state to "Failed" and kills the Server.

[0193] If AutoKillIfFailedEnabled is false, NM sets it internal value of Server state to "Failed Not Restartable", logs a warning and continues.

3.3 Server returns its State value after HealthCheckIntervalSeconds.

3.3.1 Server State is Running

[0194] No action.

3.3.2 Server State is Failed

[0195] If the Server's AutoKillIfFailedEnabled parameter is "true", NM sets its internal value of Server state to "Failed" and kills the Server.

[0196] If AutoKillIfFailedEnabled is false, NM sets it internal value of Server state to "Failed Not Restartable", logs a warning and continues.

[0197] 4. In the next iteration, if NM sees that its interval value of Server state is "Failed", it will try to restart the Server.

[0198] NM checks the Server's AutoRestartEnabled parameter. If it is true and less than RestartMax restarts have been done in the current RestartIntervalSeconds window, NM will restart the Server.

[0199] If Server has already been restarted RestartMax times in the current RestartIntervalSeconds window, NM will wait till the next RestartIntervalSeconds window begins before doing another restart.

[0200] If AutoRestartEnabled is false, NM will not restart the Server.

5.2.1 Server State Transition Notifications

[0201] When certain transitions occur in the Server's State value, it will inform the NM of them. This will be particularly useful in the following cases:

[0202] When the NM starts a Server, there is currently no mechanism to determine if the Server started successfully or not. Now, the Server will inform the NM once it has entered a Standby State.

[0203] If a Server fails, NM will discover this only in the next iteration of its health-monitoring query. Now, the Server will inform the NM as soon as it has entered a Failed State.

[0204] Passing the NM's listening address to the Server when the latter is starting up will facilitate this communication.

[0205] Impact on JMX Clients

[0206] In WLS 6.1, JMX clients (like the Admin Console) performed Server lifecycle operations by invoking methods on the Server configuration MBeans. In Acadia, these clients will be accessing the new Server Lifecycle MBeans for this purpose.

[0207] These new MBeans have a different interface than the Server configuration MBeans.

[0208] Detailed below are the changed semantics:

[0209] Operations such as "start" and "shutdown" will now return precise information on their success or failure. They will throw an exception if the operation fails. Look at [SLC] for details.

[0210] All operations on the Node Manager Runtime MBeans are now blocking. A TaskMBean interface is being provided around the Server Lifecycle MBeans to provide an asynchronous interface to JMX clients.

[0211] The new Server Lifecycle and Node Manager Runtime MBeans provide a rich set of functionality. JMX clients can make use of these to perform a wide variety of Server lifecycle and health monitoring control operations. Look at [SLC] and section 5.2.3 above for details on this.

[0212] 5.3 NM—Server Lifecycle Interactions

[0213] Detailed below are the interactions between these two entities during each of the Server Lifecycle state transitions.

[0214] Admin console, weblogic.Admin command line utility and other Admin Clients will be effecting these state transitions by invoking methods on the ServerLifecycleRuntimeMBean.

[0215] 1. start( ) [SHUTDOWN->RUNNING]

[0216] startInStandby( ) [SHUTDOWN->STANDBY]

[0217] ServerLifecycleRuntimeMBean hosted on the Admin Server will invoke the start( ) or startInStandby( ) methods on the corresponding NodeManagerRuntimeMBean.

[0218] 2. shutdown( ) [STANDBY->SHUTDOWN]

[0219] If a NM is configured, ServerLifecycleRuntimeMBean hosted on the Admin Server will invoke the shutdown( ) method on the corresponding NodeManagerRuntimeMBean. If not, it will invoke the shutdown( ) method on the ServerLifecycleRuntimeMBean hosted on the Managed Server.

3. getState( )

[0220] ServerLifecycleRuntimeMBean hosted on the Managed Server will return the State attribute of the Server.

[0221] ServerLifecycleRuntimeMBean hosted on the Admin Server will invoke the getState( ) method on the ServerLifecycleRuntimeMBean hosted on the Managed Server. If this operation times out, it will then invoke the getState( ) method on the NodeManagerRuntimeMBean.

6. Diffie-Hellman Based Authentication/Encryption Scheme

[0222] A Diffie-Hellman based Authentication/Encryption scheme was proposed as an alternative to the current X.509 Certificates based scheme.

[0223] After much discussion, it was decided that customers do not require this. Detailed below is the proposal.

6.1 Functional Description

[0224] All data being sent over the network link between the Admin Server and NM will be encrypted using a new scheme based on the Diffie-Hellman algorithm. The encryption parameters will be negotiated at link establishment time and will depend upon configuration parameters and installed encryption software.

[0225] The Admin Server and NM will be authenticated with each other using a shared secret based mechanism.

6.2 Functional Requirements

6.2.1 Concepts and Definitions

[0226] Admin Server begins the communication session.

[0227] A NM receives the initial connection.

[0228] Both processes are aware of the encryption feature, and have two configuration parameters.

[0229] The first configuration parameter is the Minimum encryption level a process will accept. It is expressed as a key length: 0, 40, or 128 bits.

[0230] The second configuration parameter is the Maximum encryption level a process is willing to support. It also is expressed as a 0, 40, or 128 bit key size.

[0231] For convenience, this document will denote the two parameters as (min, max). So (40, 128) means a process will accept at least 40-bit encryption but desires 128-bit if possible.

[0232] Encryption parameters negotiated are for the lifetime of the communication session.

6.2.2 Encryption Key Size Negotiation

[0233] The first step is for the two processes to agree on the largest common key size supported by both. This negotiation itself need not be encrypted or hidden.

[0234] A pre-processing step temporarily reduces the maximum key size parameter configured to agree with the installed software's capabilities. This must be done at link negotiation time, because at configuration time it may not be possible to verify a particular machine's installed encryption package. For example, the administrator may configure (0, 128) encryption for an unbooted machine that only has a 40-bit encryption package installed. When the machine actually negotiates a key size, it should represent itself as (0, 40). In some cases this may cause a run-time error; for example (128, 128) is not possible with a 40-bit encryption package.

[0235] The following table shows how the configured parameters are modified based on which encryption package is installed. This is a local computation that each process performs itself. The result serves as input for the subsequent cross-machine negotiation.

configured parameters (min, max)	no encryption package installed	40-bit encryption package installed	128-bit encryption package installed
0, 0	0, 0	0, 0	0, 0
0, 40	0, 0	0, 40	0, 40
0, 128	0, 0	0, 40,	0, 128
40, 40	error	40, 40	40, 40
40, 128	error	40, 40	40, 128
128, 128	error	error	128, 128

[0236] Next the two processes jointly agree on the largest key size acceptable to both. It may be that there is no overlap, in which case network link establishment fails (with an appropriate log error message). This table shows the outcome for all possible combinations of min/max parameters:

	0, 0	0, 40	0, 128	40, 40	40, 128	128, 128
0, 0	0	0	0	error	error	error
0, 40	0	40	40	40	40	error
0, 128	0	40	128	40	128	128
40, 40	error	40	40	40	40	error
40, 128	error	40	128	40	128	128
128, 128	error	error	128	error	128	128

6.2.3 Session Key Agreement

[0237] Two session keys will be generated for the encrypted network link using the Diffie-Hellman algorithm. One session key will be used for data flowing from the

Admin Server to the NM, and the other key will be used for traffic flowing in the reverse direction. The generated session keys will always be 128-bit.

[0238] Input parameters to the Diffie-Hellman key agreement algorithm will be fixed (burned-in to WebLogic software). Admin Server will transmit the parameters it wants to use to the NM. This permits the burned-in parameters to be changed in future releases.

[0239] The actual Diffie-Hellman parameters to burn-in for the first release of this feature are shown in Appendix A.

[0240] Diffie-Hellman session key agreement also requires a cryptographically secure pseudo-random number source. The seed value for the pseudo-random number generator must contain a large amount of unpredictable data, so that a network-based attacker cannot iterate through likely seed values.

6.2.6 40-bit Protocol Requirements

[0241] If a 40-bit key size is negotiated, the 128-bit session key produced by Diffie-Hellman should be used for RC4 bulk encryption. However, 88 bits must be disclosed in the clear in one of the link establishment messages. This allows an attacker to conduct a 40-bit brute-force key search. Security is better than with a simple 40-bit key, because the 88-bits act as salt and prevent an attacker from using pre-computed lookup tables.

[0242] A network attacker may not be permitted to defeat the 88-bit disclosure requirement by tampering with packets or deleting packets:

[0243] The actual bits disclosed in the clear must be used by the Admin Server. If they do not agree with the locally computed Diffie-Hellman session key, or are not supplied as expected, Admin Server will generate a log error message and terminate the connection.

[0244] Both the processes implicitly agree to permute their Diffie-Hellman session key when 40-bit encryption is negotiated, as shown below. This prevents a network attacker from tampering with messages and tricking the Admin Server into a 128-bit session when 40-bit was the NM s negotiation result.

6.2.7 Authentication Protocol

[0245] To guard against a Man-in-the-Middle attack, Admin Server and NM will be authenticated using a shared secret, as follows:

[0246] Both processes will generate a 128-bit MD5 message digest (fingerprint) using the send and receive session keys negotiated (see Section 6.2.3 above)

[0247] Admin Server will generate a 64-bit random number (challenge). It will then DES-encrypt the challenge and the fingerprint using its password as the key and send this to the NM.

[0248] NM will decrypt the received message with the Admin Server s password. If the Fingerprints don t match, it will reject the authentication request.

[0249] NM will generate a 64-bit session key. NM will then DES-encrypt the previously-received challenge, the generated session key and the fingerprint using its password as the key and send this to the Admin Server.

[0250] Admin Server will decrypt the received message with the NM s password. It will check the received challenge and fingerprint with its local values. If either doesn't t match, it will reject the authentication request.

[0251] If the above sequence is completed successfully, Admin Server and NM will be considered authenticated with each other.

6.3 External Interface Requirements

6.3.1 Hardware Interfaces (O)

6.3.2 Software Interfaces (Javadoc, MBean, Objects, Classes)

[0252] These new command line arguments will be specified for the Admin Server.

[0253] WebLogic.management.minEncryptionBits= <number>

[0254] the Minimum encryption level the process will accept. Possible values are 0, 40, 128. Default value is 0.

[0255] WebLogic.management.maxEncryptionBits= <number>

[0256] the Maximum encryption level the process will accept. Possible values are 0, 40, 128. Default value is 128.

[0257] WebLogic.mianagement.ena- bleChannelBinding=0|1

[0258] a value of 1 enables the Authentication Protocol (Section 5.2.7). Default is 0.

[0259] WebLogic.management.passwordKey=<string>

[0260] key to be used to decrypt the encrypted NM passwords stored in the configuration file.

[0261] It must be specified if WebLogic.management.ena- bleChannelBinding is set to 1.

[0262] The utility WebLogic.wtc.gwt.genpasswd will be used to encrypt NM passwords to be stored in the configuration file.

[0263] These new command line arguments will be specified for the NM.

[0264] WebLogic.nodemanager.minEncryptionBits= <number>

[0265] the Minimum encryption level the process will accept. Possible values are 0, 40, 128. Default value is 0.

[0266] WebLogic nodemanager.maxEncryptionBits= <number>

[0267] the Maximum encryption level the process will accept. Possible values are 0, 40, 128. Default value is 128.

[0268] WebLogic.nodemanager.en- ableChannelBinding=0|1

[0269] a value of 1 enables the Authentication Protocol (Section 5.2.7). Default is 0.

[0270] WebLogic.nodemanager.password=<string>

[0271] the NM s password.

[0272] Must be specified if WebLogic.nodemanager.en-ableChannelBinding is set to 1.

[0273] WebLogic.nodemanager.adminServ-erPassword=<string>

[0274] the Admin Server s password.

[0275] Must be specified if WebLogic.nodemanager.en-ableChannelBinding is set to 1.

[0276] These new arguments will be modifiable at runtime via the Admin Console.

7.5 Alternate Node Manager

[0277] In one embodiment of the invention, an alternate type of node manager architecture is used. The alternate node manager may be operable to achieve at least the following functions:

[0278] Aggregation of administrative actions and information in NM for access by 3rd party application monitors and HA frameworks using standard JMX interfaces

[0279] Internal interactions with admin console, cluster group leader, etc. using standard JMX interfaces

[0280] Described below are some of the design points of the new NM architecture.

Summary of New NM Architecture

Services Required by Administrator/App Monitor

1. ProcessControl ("PC") service

[0281] start, kill and restart Managed Svrs ("MS") on local node

[0282] will be hosted by Admin Svr ("AS")

[0283] will be hosted by NM on other nodes

[0284] provide restart capability to internal and external clients via Runtime MBeans

2. HealthMonitoring ("HM") service

[0285] monitor State and other runtime attributes of MS on local or remote node

[0286] can be hosted either by AS or individual NMs

[0287] provide this info to internal and external clients via Runtime MBeans

New NM Charactertistics

0. hosted on stripped-down MS

[0288] registered as a MS with the AS (in config.xml)

1. Configuration

[0289] all config passed as command line args (like for today's NM)

[0290] startup independent of AS

[0291] no Config MBeans for runtime configuration changes (like for today's NM)

2. WatchDog ("WD")

[0292] reqd to mon NM on platforms where OS monitoring not avl (non NT and Solaris platforms)

[0293] 1 WD per NM

[0294] can spawn the NM when started so Admins will have to manually start just 1 process

3. NM may be the OS service on NT and Solaris

[0295] installed when installing WLS on node

[0296] can also be started manually

4. Interoperability

[0297] could have1 NM per Domain

[0298] (Security) won't have to manage multiple Certificates

[0299] NM must be of highest release wrt all MSs on node

5. If NM is not used:

[0300] won't get PC svc on remote nodes

[0301] HM svc will won't be able to restart remote MSs

[0302] better scalability if HM svc hosted on NMs

6. Additional Enhancements

[0303] make NM highly preferable (advantages listed in #5)

[0304] use WD to bootstrap NM on local node upon AS's command (will eliminate manual config of NM on each remote node and allow runtime config via Config MBeans)

[0305] use NM to aggregate Cluster heartbeats for all cluster members on local node (broader implications—Eric/Mesut/Dean)

[0306] make NM a surrogate AS? (broader implications)

What is claimed is:

1. A method for monitoring at least one server comprising:

determining, at the occurrence of a triggering event, a health status of the at least one server, the server configured to return the health status of the server in response to a query of the health status;

detecting at least one of a degenerate health status of the at least one server and a failure to respond to the query of the health status; and

performing treatment on the at least one server when at least one of a degenerate health status of the at least one server and a failure to respond to the query of the health status is detected, by:

determining a value of an auto-terminate parameter, the value specified in the auto-terminate parameter indicating whether the server can be automatically killed; and

determining whether to kill the at least one server based at least in part upon whether the auto-terminate parameter indicates that the server may be killed when degenerate health or a failure to respond has been detected.

2. The method of claim 1 wherein determining, at the occurrence of a triggering event includes determining a period of time has elapsed.

3. The method of claim 1 wherein determining, at the occurrence of a triggering event includes receiving a signal from an entity external to an object monitoring the server.

4. The method of claim 1 wherein detecting at least one of a degenerate health status of the at least one server and a failure to respond to the query of the health status comprises:

determining whether the server responds to the query signal.

5. The method of claim 4 wherein determining whether the server responds to the query signal includes waiting a specified period of time before determining the server has not responded to the query signal.

6. The method of claim 1 wherein detecting at least one of a degenerate health status of the at least one server and a failure to respond to the query of the health status comprises:

receiving the server's response to the query signal; and

determining the server's health status based upon the server's response.

7. The method of claim 1 wherein performing treatment includes terminating the server if the server is deemed failed.

8. The method of claim 1 further comprising controlling monitoring of the at least one server at an administration server.

9. The method of claim 8 further comprising encoding communication between an entity performing monitoring of the at least one server and the administration server.

10. The method of claim 1 wherein determining the health status of the server includes:

invoking a servlet on the server, the servlet configured to return a health status of the server.

11. The method of claim 10 wherein invoking a servlet on the server, the servlet configured to return a health status of the server includes:

invoking an asynchronous servlet configured to return the health status to the monitoring instance upon the occurrence of an event.

12. The method of claim 1 wherein determining, at the occurrence of a triggering event includes:

determining whether an interval period has elapsed, the interval period corresponding to a time between successive health checks performed on a server, wherein a value specified in a health check interval parameter corresponds to the interval period.

13. The method of claim 12, further comprising accessing and setting the health check interval parameter.

14. The method of claim 1 wherein determining, at the occurrence of a triggering event includes:

determining whether a timeout period has elapsed, the timeout period corresponding to a time the monitoring instance will wait for a response to a health check query performed by the monitoring instance to the server, wherein a value specified in a health check timeout parameter corresponds to the timeout period.

15. The method of claim 14 further comprising accessing and setting the health check interval parameter.

16. The method of claim 1, further comprising:

continuing operations with the server in a degenerated health status, if it is determined not to kill the server.

17. The method of claim 1, further comprising:

restarting the server if it is determined to do so.

18. A machine-readable medium carrying one or more sequences of instructions for implementing interoperable management of application servers, which instructions, when executed by one or more processors, cause the one or more processors to carry out the steps of:

determining, at the occurrence of a triggering event, a health status of the at least one server, the server configured to return the health status of the server in response to a query of the health status;

detecting at least one of a degenerate health status of the at least one server and a failure to respond to the query of the health status; and

performing treatment on the at least one server when at least one of a degenerate health status of the at least one server and a failure to respond to the query of the health status is detected, by:

determining a value of an auto-terminate parameter, the value specified in the auto-terminate parameter indicating whether the server can be automatically killed; and

determining whether to kill the at least one server based at least in part upon whether the auto-terminate parameter indicates that the server may be killed when degenerate health or a failure to respond has been detected.

19. A monitoring apparatus for remotable monitoring of an application server, the apparatus comprising:

a monitoring instance, triggered at the occurrence of a triggering event to query health status of the application server,

a monitoring servlet to return the health status of the application server in response to a query of the health status; wherein

when the monitoring instance detects at least one of a degenerate health status of the application server and a failure to respond to the query of the health status, the monitoring instance performs treatment on the application server when at least one of a degenerate health status of the application server and a failure to respond to the query of the health status is detected; and wherein performing treatment includes:

determining a value of an auto-terminate parameter, the value specified in the auto-terminate parameter indicating whether the application server can be automatically killed; and

determining whether to kill the application server based at least in part upon whether the auto-terminate parameter indicates that the application server may be killed when degenerate health or a failure to respond has been detected.

* * * * *