PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G01R 31/28, G06F 11/00	A1	(11) International Publication Number: WO 97/16744
		(43) International Publication Date: 9 May 1997 (09.05.97)

(21) International Application Number: PCT/US96/17603

(22) International Filing Date: 4 November 1996 (04.11.96)

(30) Priority Data:
08/552,316 2 November 1995 (02.11.95) US

(71) Applicant: SUN MICROSYSTEMS, INC. [US/US]; 2550 Garcia Avenue, Mountain View, CA 94043 (US).

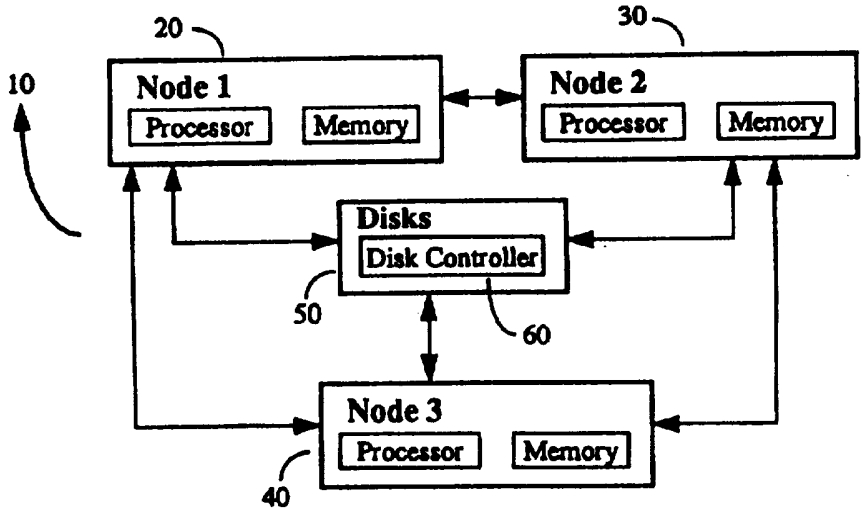
(72) Inventor: MATENA, Vladimir; 1322 Kentfield Avenue, Redwood City, CA 94061 (US).

(74) Agents: HYMAN, Eric, S. et al.; Blakely, Sokoloff, Taylor & Zafman, 7th floor, 12400 Wilshire Boulevard, Los Angeles, CA 90025-1026 (US).

(81) Designated States: European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.

(54) Title: METHOD AND APPARATUS FOR RELIABLE DISK FENCING IN A MULTICOMPUTER SYSTEM



(57) Abstract

An apparatus for fast/reliable fencing of resources such as share disks (50) on a networked system (10). For each new configuration of nodes/resources, a membership program module generates a list and based upon that, a new epoch number uniquely identifying the membership correlated with the time that it exists. A control key based upon the epoch number is generated and stored at each resource controller and node (20, 30, 40). If a node is identified as failed, it is removed from the membership list and a new epoch number/control key are generated. When a node sends an access request to a resource (50), the controller compares its locally stored key with the key stored at the node. The access request is executed only if the keys match. The membership list is revisited based upon a node's determination of the failure of a resource and is carried out independently of the failed resource.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AM	Armenia	**GB**	United Kingdom	**MW**	Malawi
AT	Austria	**GE**	Georgia	**MX**	Mexico
AU	Australia	**GN**	Guinea	**NE**	Niger
BB	Barbados	**GR**	Greece	**NL**	Netherlands
BE	Belgium	**HU**	Hungary	**NO**	Norway
BF	Burkina Faso	**IE**	Ireland	**NZ**	New Zealand
BG	Bulgaria	**IT**	Italy	**PL**	Poland
BJ	Benin	**JP**	Japan	**PT**	Portugal
BR	Brazil	**KE**	Kenya	**RO**	Romania
BY	Belarus	**KG**	Kyrgystan	**RU**	Russian Federation
CA	Canada	**KP**	Democratic People's Republic of Korea	**SD**	Sudan
CF	Central African Republic			**SE**	Sweden
CG	Congo	**KR**	Republic of Korea	**SG**	Singapore
CH	Switzerland	**KZ**	Kazakhstan	**SI**	Slovenia
CI	Côte d'Ivoire	**LI**	Liechtenstein	**SK**	Slovakia
CM	Cameroon	**LK**	Sri Lanka	**SN**	Senegal
CN	China	**LR**	Liberia	**SZ**	Swaziland
CS	Czechoslovakia	**LT**	Lithuania	**TD**	Chad
CZ	Czech Republic	**LU**	Luxembourg	**TG**	Togo
DE	Germany	**LV**	Latvia	**TJ**	Tajikistan
DK	Denmark	**MC**	Monaco	**TT**	Trinidad and Tobago
EE	Estonia	**MD**	Republic of Moldova	**UA**	Ukraine
ES	Spain	**MG**	Madagascar	**UG**	Uganda
FI	Finland	**ML**	Mali	**US**	United States of America
FR	France	**MN**	Mongolia	**UZ**	Uzbekistan
GA	Gabon	**MR**	Mauritania	**VN**	Viet Nam

1

Method and Apparatus for Reliable Disk Fencing in a Multicomputer System

5 The present invention relates to a system for reliable disk fencing of shared disks in a multicomputer system, e.g. a cluster, wherein multiple computers (nodes) have concurrent access to the shared disks. In particular, the system is directed to a high availability system with shared access disks.

10 ## Background of the Invention

In clustered computer systems, a given node may "fail", i.e. be unavailable according to some predefined criteria which are followed by the other nodes. Typically, for instance, the given node may have failed to respond to a request in less than some predetermined amount of time. Thus, a node that is executing unusually slowly may be considered to have failed, and the other
15 nodes will respond accordingly.

When a node (or more than one node) fails, the remaining nodes must perform a system reconfiguration to remove the failed node(s) from the system, and the remaining nodes preferably then provide the services that the failed node(s) had been providing.

It is important to isolate the failed node from any shared disks as quickly as possible. Oth-
20 erwise, if the failed (or slowly executing) node is not isolated by the time system reconfiguration is complete, then it could, e.g., continue to make read and write requests to the shared disks, thereby corrupting data on the shared disks.

Disk fencing protocols have been developed to address this type of problem. For instance, in the VAXcluster system, a "deadman brake" mechanism" is used. See Davis, R.J., VAXcluster
25 Principles (Digital Press 1993), incorporated herein by reference. In the VAXcluster system, a failed node is isolated from the new configuration, and the nodes in the new configuration are required to wait a certain predetermined timeout period before they are allowed to access the disks. The deadman brake mechanism on the isolated node guarantees that the isolated node becomes "idle" by the end of the timeout period.

30 The deadman brake mechanism on the isolated node in the VAXcluster system involves both hardware and software. The software on the isolated node is required to periodically tell the

cluster interconnect adaptor (CI), which is coupled between the shared disks and the cluster interconnect, that the node is "sane". The software can detect in a bounded time that the node is not a part of the new configuration. If this condition is detected, the software will block any disk I/O, thus setting up a software "fence" preventing any access of the shared disks by the failed node. A disadvantage presented by the software fence is that the software must be reliable; failure of (or a bug in) the "fence" software results in failure to block access of the shared disks by the ostensibly isolated node.

If the software executes too slowly and thus does not set up the software fence in a timely fashion, the CI hardware shuts off the node from the interconnect, thereby setting up a hardware fence, i.e. a hardware obstacle disallowing the failed node from accessing the shared disks. This hardware fence is implemented through a sanity timer on the CI host adaptor. The software must periodically tell the CI hardware that the software is "sane". A failure to do so within a certain time-out period will trigger the sanity timer in CI. This is the "deadman brake" mechanism.

Other disadvantages of this node isolation system are that:

- it requires an interconnect adaptor utilizing an internal timer to implement the hardware fence.
- the solution does not work if the interconnect between the nodes and disks includes switches or any other buffering devices. A disk request from an isolated node could otherwise be delayed by such a switch or buffer, and sent to the disk after the new configuration is already accessing the disks. Such a delayed request would corrupt files or databases.
- depending on the various time-out values, the time that the members of the new configuration have to wait before they can access the disk may be too long, resulting in decreased performance of the entire system and contrary to high-availability principles.

From an architectural level perspective, a serious disadvantage of the foregoing node isolation methodology is that it does not have end-to-end properties; the fence is set up on the *node* rather than on the disk controller.

It would be advantageous to have a system that presented high availability while rapidly setting up isolation of failed disks at the *disk controller*.

Other UNIX-based clustered systems use SCSI (small computer systems interface) "disk reservation" to prevent undesired subsets of clustered nodes from accessing shared disks. See, e.g., the ANSI SCSI-2 Proposed Standard for information systems (March 9, 1990, distributed by Global Engineering Documents), which is incorporated herein by reference. Disk reservation has a number of disadvantages; for instance, the disk reservation protocol is applicable only to systems having two nodes, since only one node can reserve a disk at a time (i.e. no other nodes can access that disk at the same time). Another is that in a SCSI system, the SCSI bus reset operation removes any disk reservations, and it is possible for the software disk drivers to issue a SCSI bus reset at any time. Therefore, SCSI disk reservation is not a reliable disk fencing technique.

Another node isolation methodology involves a "poison pill"; when a node is removed from the system during reconfiguration, one of the remaining nodes sends a "poison pill", i.e. a request to shut down, to the failed node. If the failed node is in an active state (e.g. executing slowly), it takes the pill and becomes idle within some predetermined time.

The poison pill is processed either by the host adaptor card of the failed node, or by an interrupt handler on the failed node. If it is processed by the host adaptor card, the disadvantage is presented that the system requires a specially designed host adaptor card to implement the methodology. If it is processed by an interrupt handler on the failed node, there is the disadvantage that the node isolation is not reliable; for instance, as with the VAXcluster discussed above, the software at the node may itself by unreliable, time-out delays are presented, and again the isolation is at the node rather than at the shared disks.

A system is therefore needed that prevents shared disk access at the disk sites, using a mechanism that both rapidly and reliably blocks an isolated node from accessing the shared disks, and does not rely upon the isolated node itself to support the disk access prevention.

Summary of the Invention

The present invention utilizes a method and apparatus for quickly and reliably isolating failed resources, including I/O devices such as shared disks, and is applicable to a virtually any shared resource on a computer system or network. The system of the invention maintains a membership list of all the active shared resources, and with each new configuration, such as when a resource is added or fails (and thus should be functionally removed), the system generates a new epoch number or other value that uniquely identifies that configuration at that time. Thus, identi-

cal memberships occurring at different times will have different epoch numbers, particularly if a different membership set has occurred in between.

Each time a new epoch number is generated, a control key value is derived from it and is sent to the nodes in the system, each of which stores the control key locally as its own node key. The controllers for the resources (such as disk controllers) also store the control key locally. Thereafter, whenever a shared resource access request is sent to a resource controller, the node key is sent with it. The controller then checks whether the node key matches the controller's stored version of the control key, and allows the resource access request only if the two keys match.

When a resource fails, e.g. does not respond to a request within some predetermined period of time (indicating a possible hardware or software defect), the membership of the system is determined a new, eliminating the failed resource. A new epoch number is generated, and therefrom a new control key is generated and is transmitted to the all the resource controllers and nodes on the system. If an access request arrives at a resource controller after the new control key is generated, the access request will bear a node key that is different from the current control key, and thus the request will not be executed. This, coupled with preventing nodes from issuing access requests to resources that are not in the current membership set, ensures that failed resources are quickly eliminated from access, by requiring that all node requests, in order to be processed, have current control key (and hence membership) information.

The nodes each store program modules to carry out the functions of the invention -- e.g., a disk (or resource) manager module, a distributed lock manager module, and a membership module. The distribution of these modules allows any node to identify a resource as failed and to communicate that to the other nodes, and to generate new membership lists, epoch numbers and control keys.

The foregoing system therefore does not rely upon the functioning of a failed resource's hardware or software, and provides fast end-to-end (i.e. at the resource) resource fencing.

Brief Description of the Drawings

Figure 1 is a top-level block diagram showing several nodes provided with access to a set of shared discs.

Figure 2 is a more detailed block diagram of a system similar to that of Figure 1, but show-

ing elements of the system of the invention that interact to achieve disk fencing.

Figure 3 is a diagram illustrating elements of the structure of each node of Figure 2 or Figure 3 before and after reconfiguration upon the unavailability of node D.

Figure 4 is a block diagram of a system of the invention wherein the nodes access more than one set of shared disks.

Figure 5 is a flow chart illustrating the method of the invention.

Description of the Preferred Embodiments

The system of the invention is applicable generally to clustered systems, such as system 10 shown in Figure 1, including multiple nodes 20-40 (Nodes 1-3 in this example) and one or more sets of shared disks 50. Each of nodes 20-40 may be a conventional processor-based system having one or more processors and including memory, mass storage, and user I/O devices (such as monitors, keyboards, mouse, etc.), and other conventional computer system elements (not all shown in Figure 1), and configured for operation in a clustered environment.

Disks 50 will be accessed and controlled via a disk controller 60, which may include conventional disk controller hardware and software, and includes a processor and memory (not separately shown) for carrying out disk control functions, in addition to the features described below.

The system of the invention may in general be implemented by software modules stored in the memories of the nodes 20-40 and of the disk controller. The software modules may be constructed by conventional software engineering, given the following teaching of suitable elements for implementing the disk fencing system of the invention. Thus, in general in the course of the following description, each described function may be implemented by a separate program module stored at a node and/or at a resource (e.g. disk) controller as appropriate, or several such functions may be implemented effectively by a single multipurpose module.

Figure 2 illustrates in greater detail a clustered system 70 implementing the invention. The system 70 includes four nodes 80-110 (Nodes A-D) and at least one shared disk system 120. The nodes 80-110 may be any conventional cluster nodes (such as workstations, personal computers or other processor-based systems like nodes 20-40 or any other appropriate cluster nodes), and the disk system may be any appropriate shared disk assembly, including a disk system 50 as discussed in connection with Figure 1.

Each node 80-110 includes at least the following software modules: disk manager (DM),

an optional distributed lock manager (DLM), and membership monitor (MM). These modules may be for the most part conventional as in the art of clustered computing, with modifications as desired to implement the features of the present invention. The four MM modules MMA-MMD are connected in communication with one another as illustrated in Figure 2, and each of the disk manager modules DMA-DMD is coupled to the disk controller (not separately shown) of the disk system 120.

Nodes in a conventional clustered system participate in a "membership protocol", such as that described in the VAXcluster Principles cited above. The membership protocol is used to establish an agreement on the set of nodes that form a new configuration when a given node is dropped due to a perceived failure. Use of the membership protocol results in an output including (a) a subset of nodes that are considered to be the current members of the system, and (b) an "epoch number" (EN) reflecting the current status of the system. Alternatives to the EN include any time or status value uniquely reflecting the status of the system for a given time. Such a membership protocol may be used in the present system.

According to membership protocol, whenever the membership set changes a new unique epoch number is generated and is associated with the new membership set. For example, if a system begins with a membership of four nodes A-D (as in Figure 2), and an epoch number 100 has been assigned to the current configuration, this may be represented as <A, B, C, D; #100> or <MEM=A, B, C, D; EN=100>, where MEM stands for "membership". This is the configuration represented in Figure 3(a), where all four nodes are active, participating nodes in the cluster.

If node D crashes or is detected as malfunctioning, the new membership becomes <MEM=A, B, C; EN=101>; that is, node D is eliminated from the membership list and the epoch number is incremented to 101, indicating that the epoch wherein D was most recently a member is over. While all the nodes that participate in the new membership store the new membership list and new epoch number, failed node D (and another other failed node) maintains the old membership list and the old epoch number. This is as illustrated in Figure 3(b), wherein the memories of nodes A-C all store <MEM=A, B, C; EN=101>, while failed and isolated node D stores <MEM=A, B, C, D; EN=100>.

The present invention takes utilizes this fact -- i.e. that the current information is stored by active nodes while outdated information is stored by the isolated node(s) -- to achieve disk fencing. This is done by utilizing the value of a "control key" (CK) variable stored by the nodes and

the shared disk system's controller (e.g. in volatile memory of the disk controller).

Figure 4 is a block diagram of a four-node clustered system 400 including nodes 410-440 and two shared disk systems 450-460 including disks 452-456 (system 450) and 462-466 (system 460). Disk systems 450 and 460 are controlled, respectively, by disk controllers 470 and 480 coupled between the respective disk controllers and a cluster interconnect 490.

The nodes 410-440 may be processor-based systems as described above, and the disk controllers are also as described above, and thus the nodes, shared disk systems (with controllers) and cluster interconnect may be conventional in the art, with the addition of the features described herein.

Each node stores both a "node key" (NK) variable and the membership information. The NK value is calculated from the current membership by one of several alternative functions, described below as Methods 1-3. Figure 4 shows the generalized situation, taking into account the possibility that any of the nodes may have a different CK number than the rest, if that node has failed and been excluded from the membership set.

As a rule, however, when all nodes are active, their respective stored values of NK and the value of CK stored at the disk controllers will all be equal.

Node/Disk Controller Operations Using Node Key and Control Key Values

Each read and write request by a node for accessing a disk controller includes the NK value; that is, whenever a node requests read or write access to a shared disk, the NK value is passed as part of the request. This inclusion of the NK value in read and write requests thus constitutes part of the protocol between the nodes and the controller(s).

The protocol between the nodes and disk controller also includes two operations to manipulate the CK value on the controller: GetKey to read the current CK value, and SetKey to set the value of CK to a new value. GetKey does not need to provide an NK value, a CK value, or an EN value, while the SetKey protocol uses the NK value as an input and additionally provides a new CK value "new.CK" to be adopted by the controller.

The four foregoing requests and their input/output arguments may be represented and summarized as follows:

Read(NK, ...)

Write(NK, ...)

GetKey(...)

SetKey(NK, new.CK)

The GetKey(...) operation returns the current value of CK. This operation is never rejected by the controller.

The SetKey(NK, new.CK) operation first checks if the NK field in the request matches the current CK value in the controller. In the case of a match, the CK value in the controller is set equal to the value in the "new.CK" field (in the SetKey request). If NK from the requesting node doesn't match the current CK value stored at the controller, the operation is rejected and the requesting node is sent an error indication.

The Read(NK, ...) and Write(NK, ...) operations are allowed to access the disk only if the NK field in the packet matches the current value of CK. Otherwise, the operation is rejected by the controller and the requesting node is sent an error indication.

When a controller is started, the CK value is preferably initialized to 0.

Procedure Upon Failure of a Node

When the membership changes because one or more failed nodes are being removed from the system, the remaining nodes calculate a new value of CK from the new membership information (in a manner to be described below). One of the nodes communicates the new CK value to the disk controller using the SetKey(NK, new.CK) operation. After the new CK value is set, all member (active) nodes of the new configuration set their NK value to this new CK value.

If a node is not a part of the new configuration (e.g. a failed node), it is not allowed to change its NK. If such a node attempts to read or write to a disk, the controller finds a mismatch between the new CK value and the old NK value.

When a node is started, its NK is initialized to a 0 value.

Procedures for Calculating Values of the Control Key (CK)

The control key CK may be set in a number of different ways. The selected calculation will be reflected in a software or firmware module stored and/or mounted at least at the controller. In general, the calculation of the CK value should take into account the membership information:

$$CK = func(MEM, EN)$$

where: MEM includes information about the active membership list;

and EN is the epoch number.

Method 1. Ideally, the CK value would explicitly include both a list of the new membership set (an encoded set of nodes) and the epoch number. This may not be desired if the number of nodes is high, however, because the value of CK would have to include at least a bit of information for each node. That is, in a four-node configuration at least a four-bit sequence BBBB (where B = 0 or 1) would need to be used, each bit B indicating whether a given associated node is active or inactive (failed). In addition, several bits are necessary for the epoch number EN, so the total length of the variable CK may be quite long.

Method 2 and 3 below are designed to compress the membership information when calculating the CK value.

Method 2 uses only the epoch number EN and ignores the membership list MEM. For example, the CK value is set to equal the epoch number EN.

Method 2 is most practical if the membership protocol prevents network partitioning (e.g. by majority quorum voting). If membership partitioning is allowed, e.g. in the case of a hardware failure, the use of the CK value without reflecting the actual membership of the cluster could lead to conflicts between the nodes on either side of the partition.

Method 3 solves the challenge of Method 2 with respect to partitions. In this method, the CK value is encoded with an identification of the highest node in the new configuration. For example, the CK value may be a concatenation of a node identifier (a number assigned to the highest node) and the epoch number. This method provides safe disk fencing even if the membership monitor itself does not prevent network partitioning, since the number of the highest node in a given partition will be different from that of another partition; hence, there cannot be a conflict between requests from nodes in different partitions, even if the EN's for the different subclusters happen to be the same.

Of the foregoing, with a small number of nodes Method 1 is preferred, since it contains the most explicit information on the state of the clustered system. However, with numerous nodes Method 3 becomes preferable. If the system prevents network partitioning, then Method 2 is suitable.

The Method of the Invention

Given the foregoing structures and functions, and appropriate modules to implement them, the disk fencing system of the invention is achieved by following the method 510 illustrated in the flow chart of Figure 5. At box (step) 520, the membership of the clustered system is determined in a conventional manner, and the value of the membership set (or list) is stored as the value of MEM. An epoch number EN (or other unique state identifier) is generated at box 530. These two functions are carried out by the membership monitor (MM) module, which is implemented among the member nodes to determine which nodes are present in the system and then to assign a value of EN to that configuration. An example of a system that uses an MM module in this way is applicant Sun Microsystems, Inc.'s SparcCluster PDB (parallel database).

In current systems, the epoch numbers are used so that a node can determine whether a given message or data packet is stale; if the epoch number is out of date then the message is known to be have been created during an older, different configuration of the cluster. (See, for instance, T. Mann et al., "An Algorithm for Data Replication", DEC SRC Research Report, June 1989, incorporated herein by reference, wherein epoch numbers are described as being used in stamping file replicas in a distributed system.)

The present system uses the epoch number in an entirely new way, which is unrelated to prior systems' usage of the epoch number. For an example of a preferred manner of using a cluster membership monitor in Sun Microsystems, Inc.'s systems, see Appendix A attached hereto, in which the reconfiguration sequence numbers are analogous to epoch numbers. Thus, the distinct advantage is presented that the current invention solves a long-standing problem, that of quickly and reliably eliminating failed nodes from a cluster membership and preventing them from continuing to access shared disks, without requiring new procedures to generate new outputs to control the process; rather, the types of information that is already generated may be used in conjunction with modules according to the invention to accomplish the desired functions, resulting in a reliable high-availability system.

Proceeding to box 540, the node key NK (for active nodes) and control key CK are generated by one of the Methods 1-3 described above or by another suitable method.

At box 550, it is determined whether a node has become unavailable. This step is carried out virtually continuously (or at least with relatively high frequency, e.g. higher than the frequency of I/O requests); for instance, at almost any time a given node may determine that another

node has exceeded the allowable time to respond to a request, and decide that the latter node has failed and should be removed from the cluster's membership set. Thus, the step in box 550 may take place almost anywhere during the execution of the method.

Box 560 represents an event where one of the nodes connected to the cluster generates an I/O request (such as a disk access request). If so, then at box 570 the current value of NK from the requesting node is sent with the I/O access request, and at box 580 it is determined whether this matches the value of CK stored by the controller. If not, the method proceeds to step 600, where the request is rejected (which may mean merely dropped by the controller with no action), and proceeds then back to box 520.

If the node's NK value matches the controller's CK value, then the request is carried out at box 590.

If a node has failed, then the method proceeds from box 550 back to box 520, where the failed node is eliminated in a conventional fashion from the membership set, and thus the value of MEM changes to reflect this. At this time, a new epoch number EN is generated (at box 530) and stored, to reflect the newly revised membership list. In addition, at box 540 a new control key value CK is generated, the active nodes' NK values take on the value of the new CK value, and the method proceeds again to boxes 550-560 for further disk accesses.

It will be seen from the foregoing that the failure of a given node in a clustered system results both in the removal of that node from the cluster membership and, importantly, the reliable prevention of any further disk accesses to shared disks by the failed node. The invalidating of the failed node from shared disk accesses does not rely upon either hardware or software of the failed node to operate properly, but rather is entirely independent of the failed node.

Since the CK values are stored at the disk controllers and are used by an access control module to prevent failed nodes from gaining shared disk access, the disk fencing system of the invention is as reliable as the disk management software itself. Thus, the clustered system can rapidly and reliably eliminate the failed node with minimal risk of compromising the integrity of data stored on its shared disks.

The described invention has the important advantage over prior systems that its end-to-end properties make it independent of disk interconnect network or bus configuration; thus, the node configuration alone is taken into account in determining the epoch number or other unique status value, i.e. independent of any low-level mechanisms (such as transport mechanisms).

Note that the system of the invention may be applied to other peripheral devices accessed by multiple nodes in a multiprocessor system. For instance, other I/O or memory devices may be substituted in place of the shared disks discussed above; a controller corresponding to the disk controllers 470 and 480 would be used, and equipped with software modules to carry out the fencing operation.

In addition, the nodes, i.e. processor-based systems, that are members of the cluster can be any of a variety of processor-based devices, and in particular need not specifically be personal computers or workstations, but may be other processor-driven devices capable of issuing access requests to peripheral devices such as shared disks.

NAME

cmm - cluster membership monitor

AVAILABILITY

SUNWcmm

INTERFACE CLASSIFICATION

Sun Private

DESCRIPTION

This manual page describes the cluster membership monitor (CMM). From the perspective of CMM, a cluster is a collection of nodes participating in the cluster membership protocol. The membership protocol is used to establish an agreement on cluster membership (i.e. which nodes are currently in the cluster).

If cluster membership changes (when nodes leave or join the cluster), CMM coordinates automatic reconfiguration of various cluster services running on the nodes. Automatic reconfiguration of cluster services is the key mechanism for achieving high availability on clustered systems.

CMM is responsible for maintaining cluster integrity and must assure that nodes never reach inconsistent agreement on membership, even in the presence of various system failures. CMM tolerates nodes joining or leaving the cluster during cluster reconfiguration.

Configuring a Cluster

A cluster is described by a single configuration file (see **cmm.conf**(4)) replicated on all cluster nodes. Generation of the configuration file is not considered a part of CMM and is typically done through an editor or a cluster specific GUI program. Once the configuration file is created, a node can join the cluster by starting up the clustd daemon.

Each node in a cluster is assigned a unique node identifier (nodeid). Nodeids are integers between 0 and 31.

Heart-beat Messages

CMM assumes that in the absence of failures, any node can communicate with any other node by sending messages. Each node sends periodic heart-beat messages to all other nodes. CMM assumes that the underlying protocol supports unreliable datagram delivery and CMM itself handles duplicate, out of order, lost, or delayed messages.

For applications in high availability clustered systems, CMM can be configured to send each heart-beat message concurrently over multiple physical networks. The message is received if it arrives at least on one network.

Agreement on Membership

Each node maintains its view of current cluster membership. clustd includes this information in the heart-beat messages sent to other nodes. Since each node receives messages from all other nodes, a distributed agreement on membership can be reached eventually. The algorithm for distributed agreement is designed to work in the presence of various failures, such as communication failures, slowly executing nodes, errors in cluster reconfiguration programs, or nodes joining and leaving the cluster during reconfiguration. If CMM is configured to vote by majority quorum, CMM guarantees that the cluster nodes never reach an inconsistent agreement on membership.

Quorum Voting

CMM uses a quorum voting mechanism to prevent cluster partitioning (split brain). Each node is assigned a number of votes in the configuration file. clustd dynamically computes the current cluster quorum as the total number of votes of the nodes that the node can communicate with. The value of the current cluster quorum is compared with the minimum cluster quorum specified in the configuration file. CMM begins cluster reconfiguration only if the current cluster quorum equals or exceeds the minimum cluster quorum.

By setting the minimum cluster quorum to a value greater than half the sum of all votes (i.e. requiring majority quorum), CMM automatically prevents cluster partitioning. Nodes with insufficient quorum wait in the s-begin state until more nodes join the cluster.

States and Transitions

CMM coordinates cluster reconfiguration (reconfiguration of distributed cluster services). The cluster reconfiguration protocol is defined by an I/O automaton running on each node. The "input" to the automaton are messages received from other nodes, timeouts, exit status from local reconfiguration programs, and user requests. The automaton "output" controls execution of local reconfiguration programs. A complete specification of the I/O automaton behavior is beyond the scope of this manual page and the following description is intended mainly for developers of cluster services interfacing with CMM.

The node automaton has the following states: s-start, s-begin, s-step(1), s-step(2), ..., s-step(N), s-end, s-return, s-stop, s-abort, and s-down. We let s-step(N+1) be the same as s-end. s-start is the initial state and s-down is the final state.

The following table defines the transitions between cluster states (a diagram with states and transitions here would be worth a thousand words).

From State	To State	Transition	Condition
s-start	s-begin	t-start	executed once at startup
s-begin	s-step1	-	reached agreement on membership
s-step(i)	s-step(i+1)	t-step(i)	all nodes reached s-step(i)
s-step(i)	s-return	-	detected membership change
s-step(i)	s-return	-	received an *reconfigure* request
s-step(i)	s-abort	-	errors in t-step(i) programs
s-return	s-begin	t-return	return to begin state
s-begin	s-stop	-	received a *stop* request
s-stop	s-down	t-stop	
any	s-abort	-	detected local errors
any	s-abort	-	received an *abort* request
s-abort	s-down	t-abort	

(the "-" transitions are internal to the automaton and do not generate output actions)

Reconfiguration Programs

Each state transition (with the exception of internal transition) generates an output action resulting in execution of cluster reconfiguration programs. Cluster services interested in notification of membership changes associate reconfiguration programs

with the CMM transitions. By running these programs, CMM orchestrates reconfiguration of the cluster services.

If multiple programs are associated with a transition, the programs are executed in parallel and no assumptions should be made on the order in which they are started or completed.

A node joins the cluster by starting the clustd daemon. clustd starts up in the s-start state and executes the t-start transition to enter s-begin. After clustd has synchronized with clustd's on other nodes, it begins stepping through the t-step(i) transitions.

The transitions t-step(1) through t-step(N) are executed in lock-step on all cluster nodes. Before executing t-step(i+1), each node waits until all other nodes have successfully completed t-step(i). Lock-step execution is important to reconfiguration of cluster services that reconfigure in several steps, each separated by a cluster-wide synchronization barrier. CMM automatically provides this synchronization barrier to the cluster services.

Cluster reconfiguration is complete when all nodes enter the s-end state. The nodes will remain in s-end until cluster membership becomes unstable (i.e. a node has left or wants to join the cluster), or a user request is received.

Should cluster membership become unstable during a reconfiguration step, clustd internally transitions to s-return and executes the t-return transition to enter s-begin. After a new agreement on cluster membership has been negotiated, clustd re-runs the reconfiguration steps starting from s-step(1). Reconfiguration of cluster services should be designed to be idempotent with respect to s-begin. The t-return transition is provided to clean up after previously executed t-step(i) transitions.

The reconfiguration programs executed during cluster transitions communicate back to clustd through their exit status. A zero exit status indicates that the program has successfully completed. A non-zero status indicates that the program could not complete its actions because of an error on the local system. Upon receiving a non-zero status, clustd immediately transitions to s-abort and sends the SIGTERM signal to all remaining programs of the failed transition. After all the programs terminate, clustd runs the t-abort transition to remove the failed node from the cluster. Other nodes will return to s-begin (via t-return) and rerun the reconfiguration steps.

To prevent the situation that a stuck reconfiguration program on one node blocks the whole cluster, a timeout is specified for each state transition. clustd starts a timer before forking the reconfiguration programs. If the timer expires before all programs have completed, clustd transitions to s-abort, sends SIGTERM to the remaining programs, and initiates the t-abort transition, as if the program returned a non-zero status code.

If a reconfiguration program cannot complete its work because of non-local error conditions (e.g. because of a timeout when communicating with other nodes), it should issue the "reconfigure" request (see clustm(1m) or cm_reconfigure(3n)) and return a zero status to clustd. After all other programs for the transition complete, clustd will return to s-begin (via t-return), resynchronize with other nodes, and retry the reconfiguration steps. It is the responsibility of the programs to return before clustd's

timeout for the transition expires. A failure to do so would result in clustd's removing the local node from the cluster through the t-abort transition, as described above.

User Requests

Cluster programs can communicate with clustd via a command level or C library level interface (see clustm(1m) and cmm(3n)). Any program can access clustd's state information and a program with super-user privileges can affect clustd's otherwise automatic behavior.

A privileged program can send the "reconfigure", "stop", and "abort" requests to clustd. The "stop" request forces clustd to transition to s-begin via the t-return transition and then execute the t-stop transition.

The "abort" request instructs clustd to immediately perform the t-abort transition. If reconfiguration programs are in progress when the "abort" request is received, clustd sends SIGTERM to these programs before performing t-abort.

If clustd receives the "reconfigure" request when in the s-step(i) or s-end state, it returns back to s-begin via the s-return and t-return. The "reconfigure" command is ignored if clustd is in other states. The "reconfigure" command is typically used in two situations: when a t-step(i) reconfiguration program cannot complete its action because of remote errors, or when reconfiguration of cluster services is controlled by some external parameter. A (somewhat academic) example of the latter case is a transaction processing (TP) system distributed across all the cluster nodes during peak processing hours, but limited to a smaller subset of nodes during off-peak hours. By sending the "reconfigure" request at the beginning of each peak/off-peak period, the TP system requests clustd to coordinate its reconfiguration. The t-step(i) programs used for reconfiguration of the TP system read perform different actions based on the value of the current time.

Reconfiguration Sequence Numbers

CMM assigns a unique sequence number to each cluster reconfiguration. The assignment is done during clustd's internal transition from s-begin to s-step1. All current cluster members have agreed on the value of the sequence number before transitioning to s-step1. The sequence numbers are essential to detecting stale isolated nodes (see below).

Each node stores the most recently used sequence number in its local stable storage. CMM guarantees that sequence numbers are monotonic and never reused by the same cluster.

Isolated Nodes

The discussion of isolated nodes assumes that the cluster is configured to vote with majority quorum. A node becomes "isolated" when the current quorum of the node is lower than the required majority quorum. The cluster monitor provides two important guarantees regarding isolated nodes:

First, if one or more nodes become isolated from the nodes holding majority quorum, the nodes with majority quorum will not begin cluster reconfiguration before the isolated nodes are in a "safe" state. The safe states are s-begin and s-down. Cluster services should be designed to assure that nodes are idle (with respect to shared

resources and communication with the outside world) in these states by associating appropriate programs with the t-return, t-stop, and t-abort transitions. CMM supervises timely transitions of isolated nodes to the idle states and delays the beginning of cluster reconfiguration until the isolated nodes can be assumed in safe states.

Second, an isolated node automatically aborts if it learns that other nodes have performed cluster reconfiguration while the node was in isolation. Reconfiguration sequence numbers sent in the heart-beat messages are used to detect such a situation. Aborting such a node is necessary because the state of the isolated node might have become stale.

CMM provides strong guarantees concerning isolated nodes because isolated nodes could compromise cluster integrity. Assuming that the clustd program is free of bugs, CMM's guarantees are as dependable as the node failfast mechanism (described below). Cluster systems that require stronger guarantees have to employ additional mechanisms to fence off isolated nodes from shared resources and the outside environment.

Node Failfast

One major difficulty in the design of distributed systems is the fact that it is impossible to distinguish between a node executing very slowly from one that has crashed. If a node doesn't respond to messages within a timeout period, other nodes declare that the node is down (in the s-down state) and reconfigure the cluster to remove the node. If the removed node were executing slowly rather than being down, cluster integrity would be violated.

CMM uses the failfast device driver (see ff(7)) to prevent this undesirable situation. If the clustd process becomes untimely when executing sections in which timing is critical, the failfast driver aborts the node in a bounded time. Cluster nodes make a mutual agreement on the timing via the cluster configuration file.

NOTES

CMM interfaces are designed to support at least 4096 nodes. The current implementation limits the number of nodes to 32.

The content of the cmm.conf files on all cluster nodes must be identical. Operating a cluster with inconsistent cmm.conf files could yield unpredictable results. It is recommended that cluster nodes compare the contents of the file during the t-step1 transition and abort if the files are not identical.

The current implementation of distributed agreement on cluster membership blocks cluster reconfiguration if the communication graph is not a clique (fully connected subgraph). This limitation does not reduce availability of clusters that use multiple physical links for the CMM messages.

XXX - Explain the relationship of getclustbyname() to cmm

The current implementation sends the heart-beat messages via the UDP/IP protocol.

The current implementation of the cluster membership protocol does not allow the configuration file to modified while the cluster is online. clustd reads the file only once at startup and any changes to the file will take effect when clustd is restarted.

What is claimed is:

1. A method for preventing access to a shared peripheral device by a processor-based node in a multinode system, including the steps of:

(1) storing at the peripheral device a first unique value representing a first configuration of the multinode system;

(2) sending an access request from the node to the device, the request including a second unique value representing a second configuration of the multi-node system;

(3) determining whether said first and second values are identical; and

(4) if the first and second values are identical, then executing the access request at the peripheral device.

2. The method of claim 1, wherein:

said first value is generated utilizing at least in part information relating to a first time when the multinode system was in said first configuration; and

said second value is generated utilizing at least in part information relating to a second time when the multinode system was in said second configuration.

3. The method of claim 2, wherein:

step 3 includes the step of determining whether said first and second times are identical.

4. The method of claim 1, wherein said first and second values are generated based at least in part on epoch numbers generated by a membership protocol executing on said multinode system.

5. The method of claim 4, wherein each of said first and second values is generated based at least in part on respective membership sets of said multinode system generated by said membership protocol.

6. The method of claim 1, wherein each of said first and second values is generated based at least in part on respective membership sets of said multinode system generated by said membership protocol.

7. An apparatus for preventing access to at least one shared peripheral resource by a processor-based node in a multinode system, the resource being coupled to the system by a resource controller including a controller memory, each of a plurality of nodes on the system including a processor coupled to a node memory storing program modules configured to executing functions of the invention, the apparatus including:

a membership monitor module configured to determine a membership list of the nodes, including said resource, on the system at predetermined times, including at least at a time when the membership of the system changes;

a resource manager module configured to determine when the resource is in a failed state and for communicating the failure of the resource to said membership monitor to indicate to the membership monitor to generate a new membership list;

a configuration value module configured to generate a unique value based upon said new membership list and to store said unique value locally at each node on the system; and

an access control module stored at said controller memory configured to block access requests by at least one said requesting node to said resource when the locally stored unique value at said requesting node does not equal the unique value stored at said resource controller.

8. The apparatus of claim 7, wherein said configuration value monitor module is configured to determine said unique value based at least in part upon a time stamp indicating the time at which the corresponding membership list was generated.

9. The apparatus of claim 7, wherein said unique value is based at least in part upon an epoch number generated by a membership protocol module.

10. The apparatus of claim 7, wherein said membership monitor module is configured to execute independently of any action by said shared resource when said shared resource is in a failed state.

11. The apparatus of claim 7, wherein said resource manager module is configured to execute independently of any action by said shared resource when said shared resource is in a failed state.

12. The apparatus of claim 7, wherein said configuration module is configured to execute independently of any action by said shared resource when said shared resource is in a failed state.

13. The apparatus of claim 7, wherein said access control module is configured to execute independently of any action by said shared resource when said shared resource is in a failed state.

Figure 1

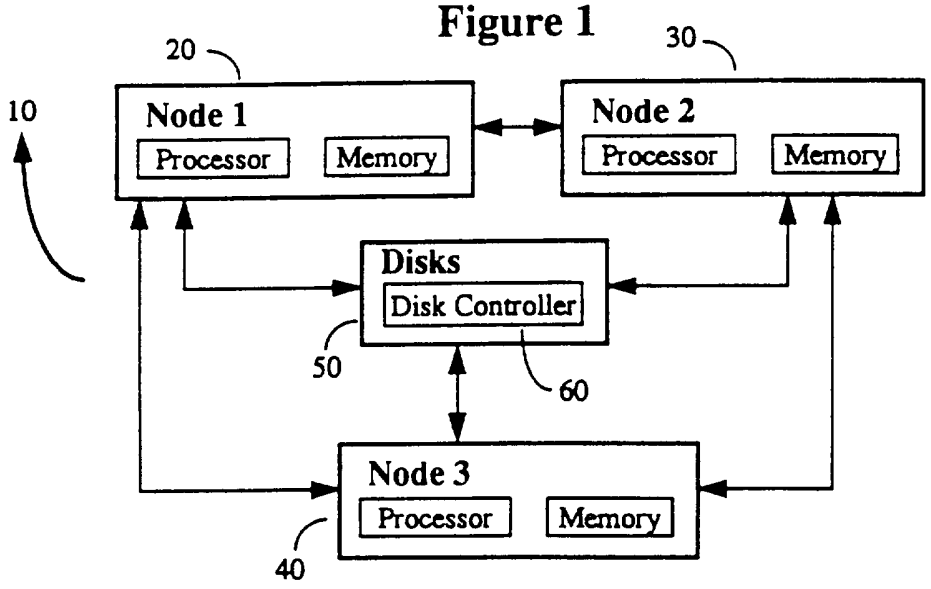
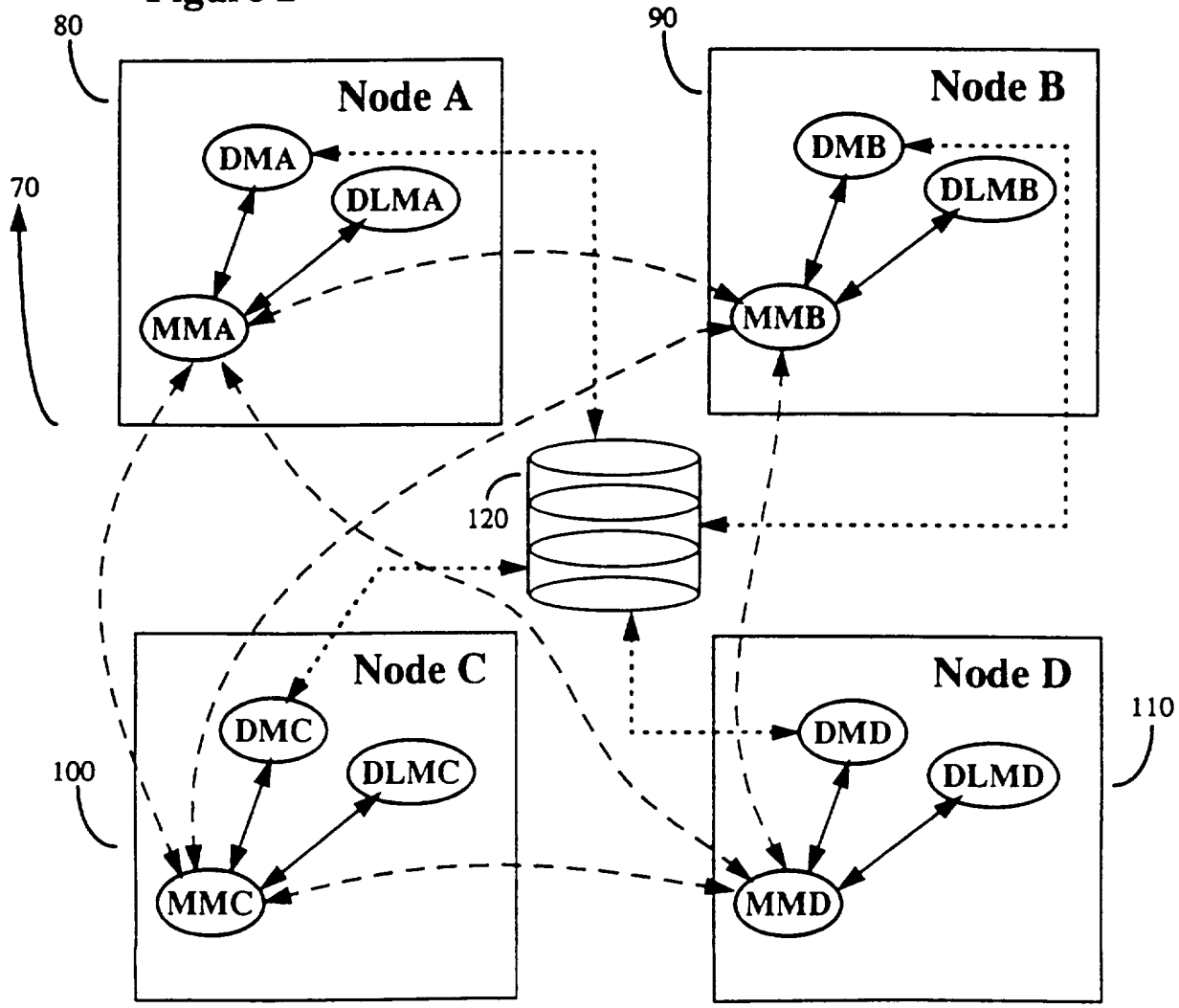
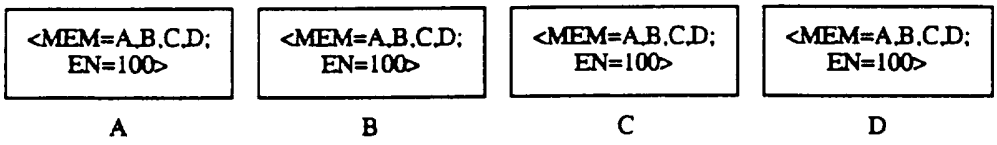


Figure 2



3(a) Before Reconfiguration:



3(b) After Reconfiguration:

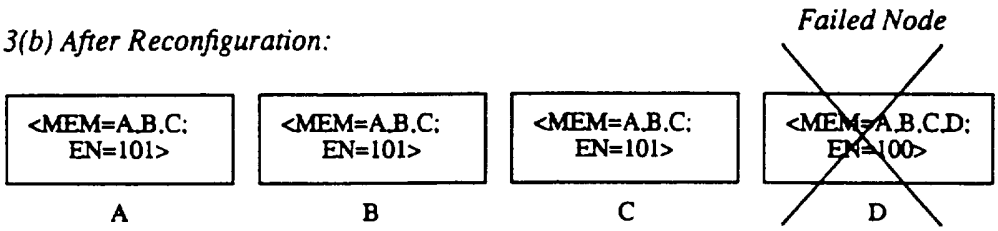


Figure 3

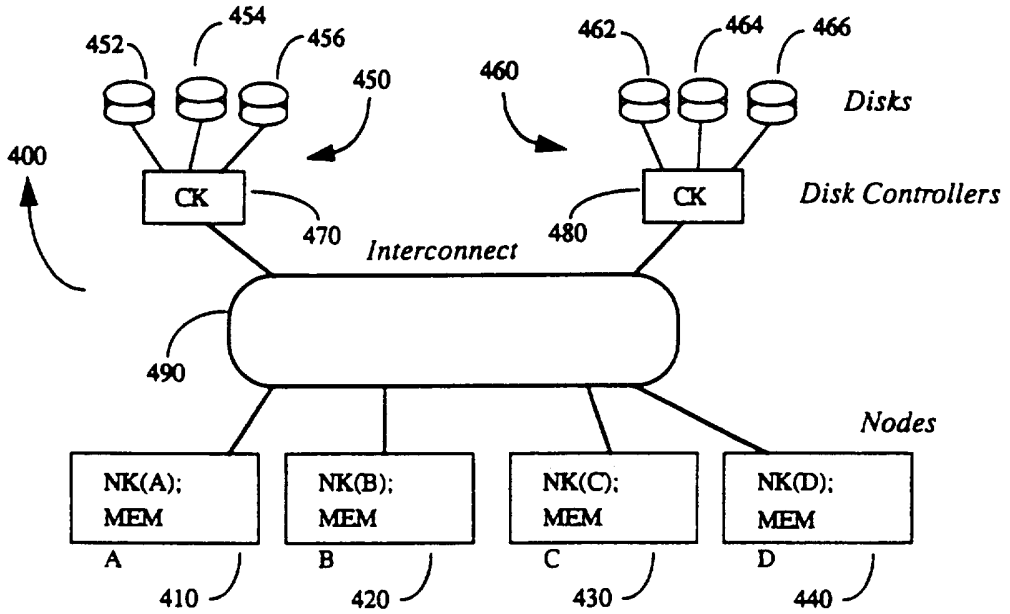


Figure 4

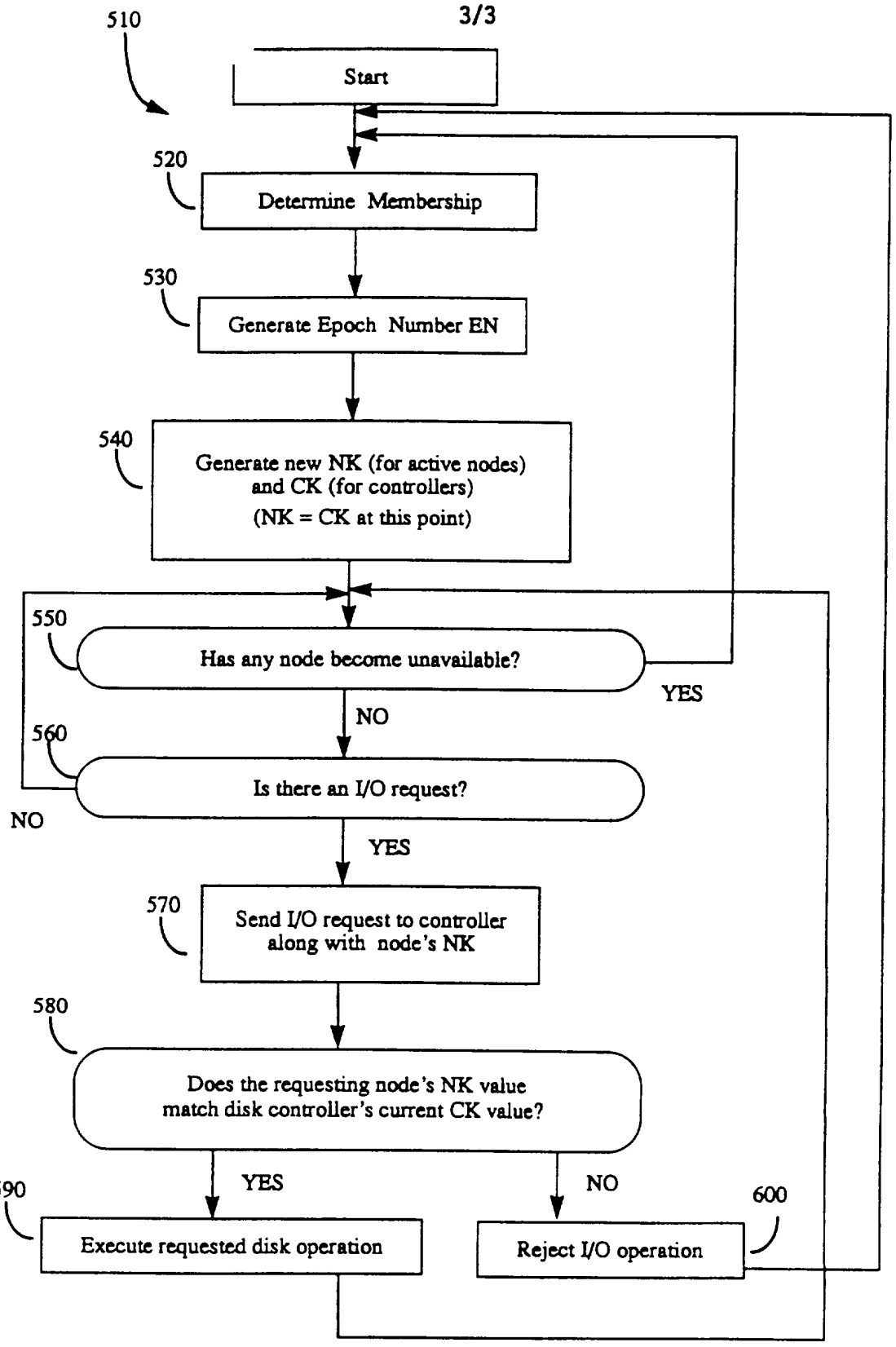

510
Start
520
Determine Membership
530
Generate Epoch Number EN
540
Generate new NK (for active nodes) and CK (for controllers)
(NK = CK at this point)
550
Has any node become unavailable?
YES
NO
560
Is there an I/O request?
NO
YES
570
Send I/O request to controller along with node's NK
580
Does the requesting node's NK value match disk controller's current CK value?
YES
NO
590
Execute requested disk operation
600
Reject I/O operation

Figure 5

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US96/17603

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G01R 31/28; G06F 11/00
US CL :395/186

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 395/186, 187.01, 188.01

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

APS, IEEE ProQuest

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US, A, 5,416,921 (FREY ET AL.) 16 May 1995, col. 1, lines 9-14, col. 2, lines 2-12 and col. 5, lines 29-35	1-3
A	US, A, 4,480,304 (CARR ET AL.) 30 October 1984, see entire document.	1-13
A	US, A, 4,805,134 (CALO ET AL.) 14 February 1989, see entire document.	1-13
A	US, A, 4,961,224 (YUNG) 02 October 1990, see entire document.	1-13
A	US, A, 5,204,961 (BARLOW) 20 April 1993, see entire document.	1-13
A	US, A, 5,321,841 (EAST ET AL.) 14 June 1994, see entire document.	1-13

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 09 DECEMBER 1996	Date of mailing of the international search report 31 JAN 1997
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer GLENN SNYDER Telephone No. (703) 305-9688

Form PCT/ISA/210 (second sheet)(July 1992)*

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US96/17603

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US, A, 5,339,443 (LOCKWOOD) 16 August 1994, see entire document.	1-13
A	US, A, 5,361,347 (GLIDER ET AL.) 01 November 1994, see entire document.	1-13
A	US, A, 5,408,653 (JOSTEN ET AL.) 18 April 1995, see entire document.	1-13
A	US, A, 5,463,733 (FORMAN ET AL.) 31 October 1995, see entire document.	1-13
A, P	US, A, 5,469,556 (CLIFTON) 21 November 1995, see entire document.	1-13
A, P	US, A, 5,568,491 (BEAL ET AL.) 22 October 1996, see entire document.	1-13