US 20030191983A1

(19) **United States**

(12) **Patent Application Publication**

(10) **Pub. No.: US 2003/0191983 A1**

(43) **Pub. Date: Oct. 9, 2003**

Mann et al.

(54) **LOOSELY COUPLED MASS STORAGE COMPUTER CLUSTER**

(75) Inventors: **Bruce E. Mann**, Mason, NH (US); **Philip J. Trasatti**, Brookline, NH (US); **Michael D. Carlozzi**, Canton, MA (US); **John A. Ywoskus**, Merrimack, NH (US); **Edward J. McGrath**, Wayland, MA (US)

Correspondence Address:
FISH & RICHARDSON PC
225 FRANKLIN ST
BOSTON, MA 02110 (US)

(73) Assignee: **Seachange International, Inc., a Delaware corporation**

(21) Appl. No.: **10/414,059**

(22) Filed: **Apr. 15, 2003**

Related U.S. Application Data

(60) Continuation of application No. 10/137,085, filed on Apr. 30, 2002, now Pat. No. 6,574,745, which is a continuation of application No. 09/412,232, filed on Oct. 5, 1999, now Pat. No. 6,571,349, which is a continuation of application No. 08/991,278, filed on Dec. 16, 1997, now Pat. No. 5,996,089, which is a division of application No. 08/547,565, filed on Oct. 24, 1995, now Pat. No. 5,862,312.

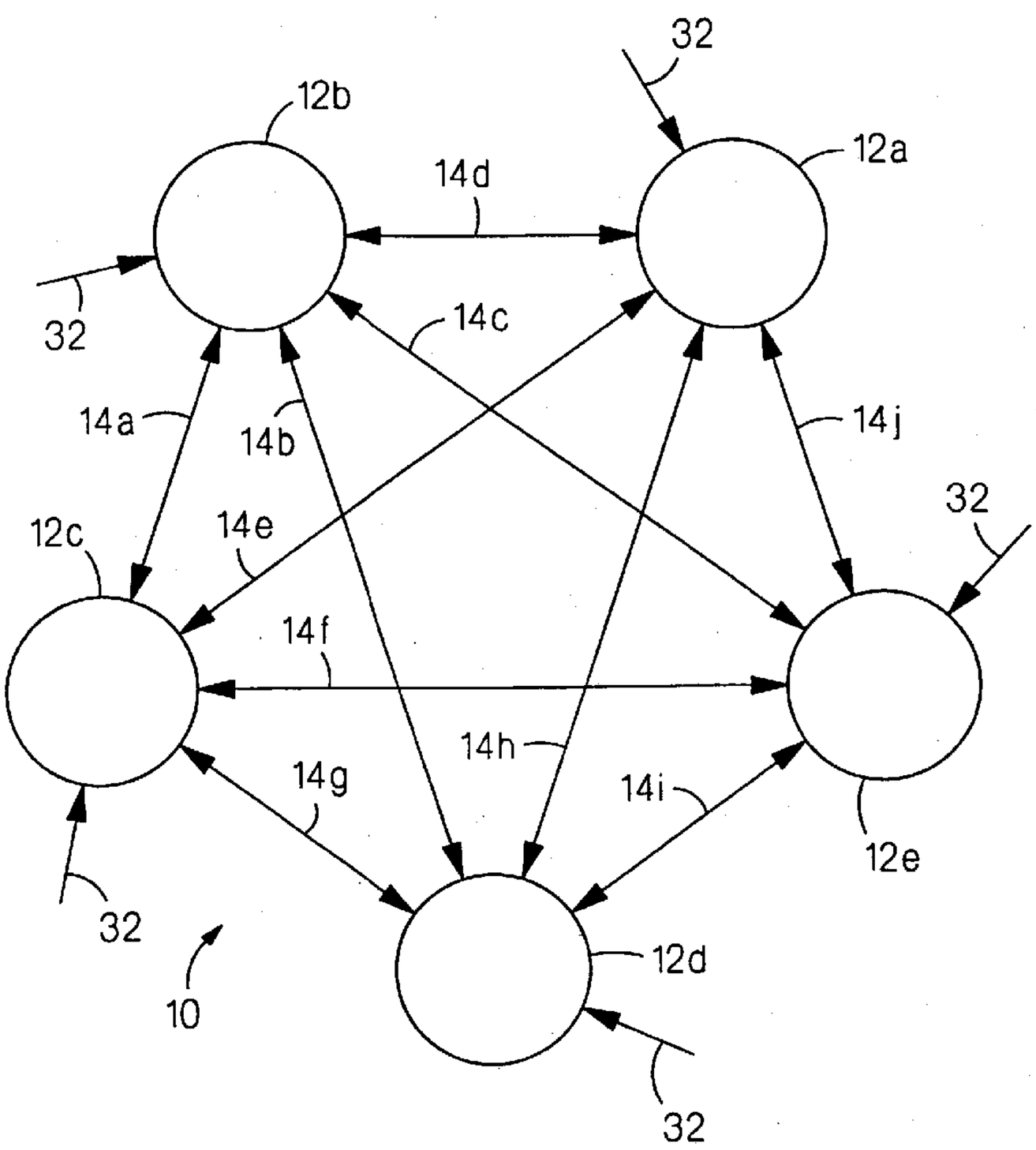
Publication Classification

(51) **Int. Cl.**[7] **H02H 3/05**

(52) **U.S. Cl.** **714/11**

(57) **ABSTRACT**

A method and apparatus redundantly store data, in particular video data objects, in a distributed computer system having at least three processor systems, each processor system being connected in point to point two way channel interconnection with each other processor system. The data is stored in a redundant fashion both at the computer system level as well as the processor system level. Accordingly, the failure of a single processor does not adversely affect the integrity of the data. The computer system can also overlay a switching system connected in a ring fashion for providing a fault tolerance to the failure of a single connected processor system at the switch level. Accordingly, there results a fault tolerant data distribution system.



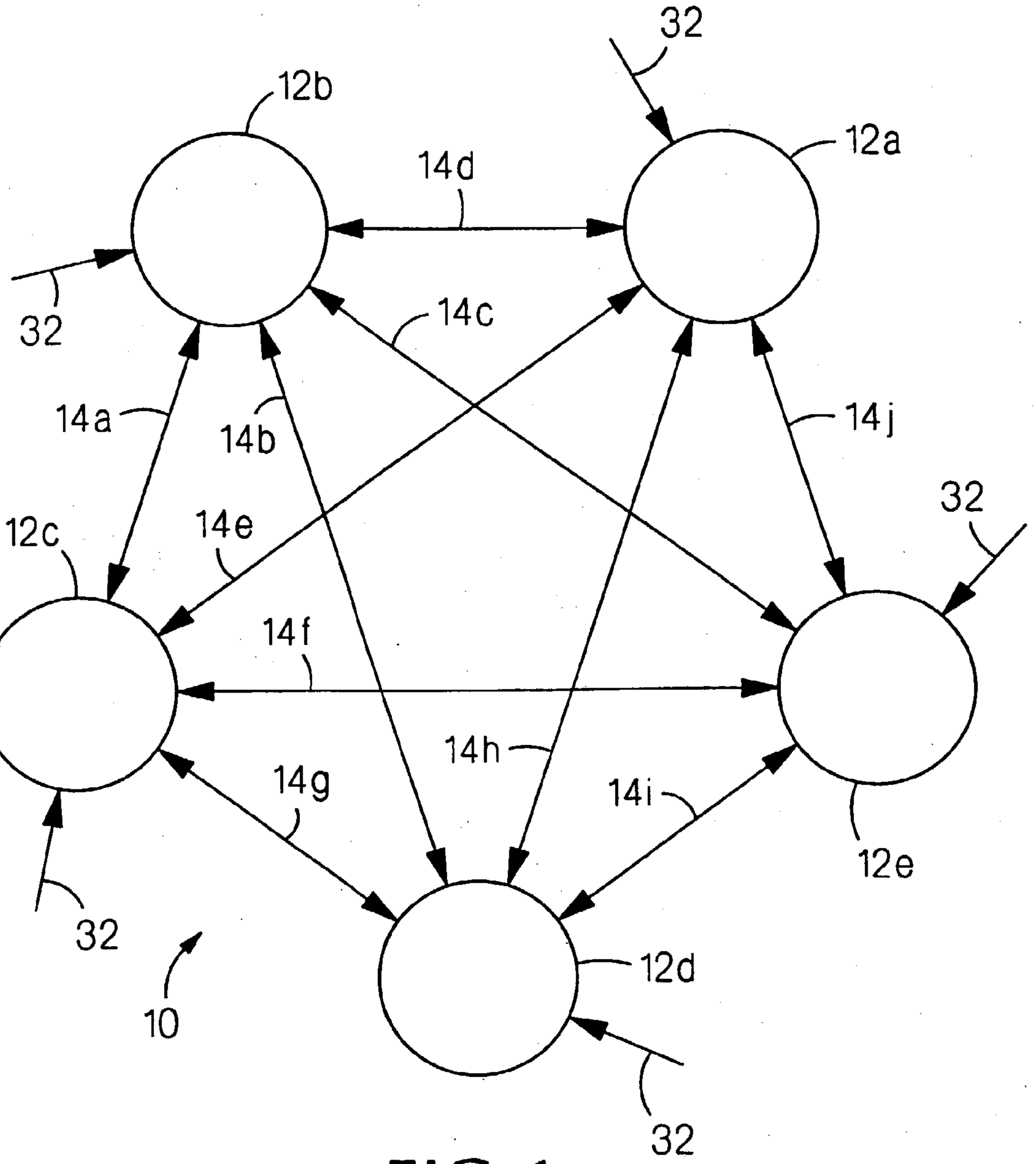

32
12b
14d
12a
32
14c
14a
14b
14j
12c
14e
32
14f
14h
14g
14i
32
12e
12d
10
32

FIG. 1

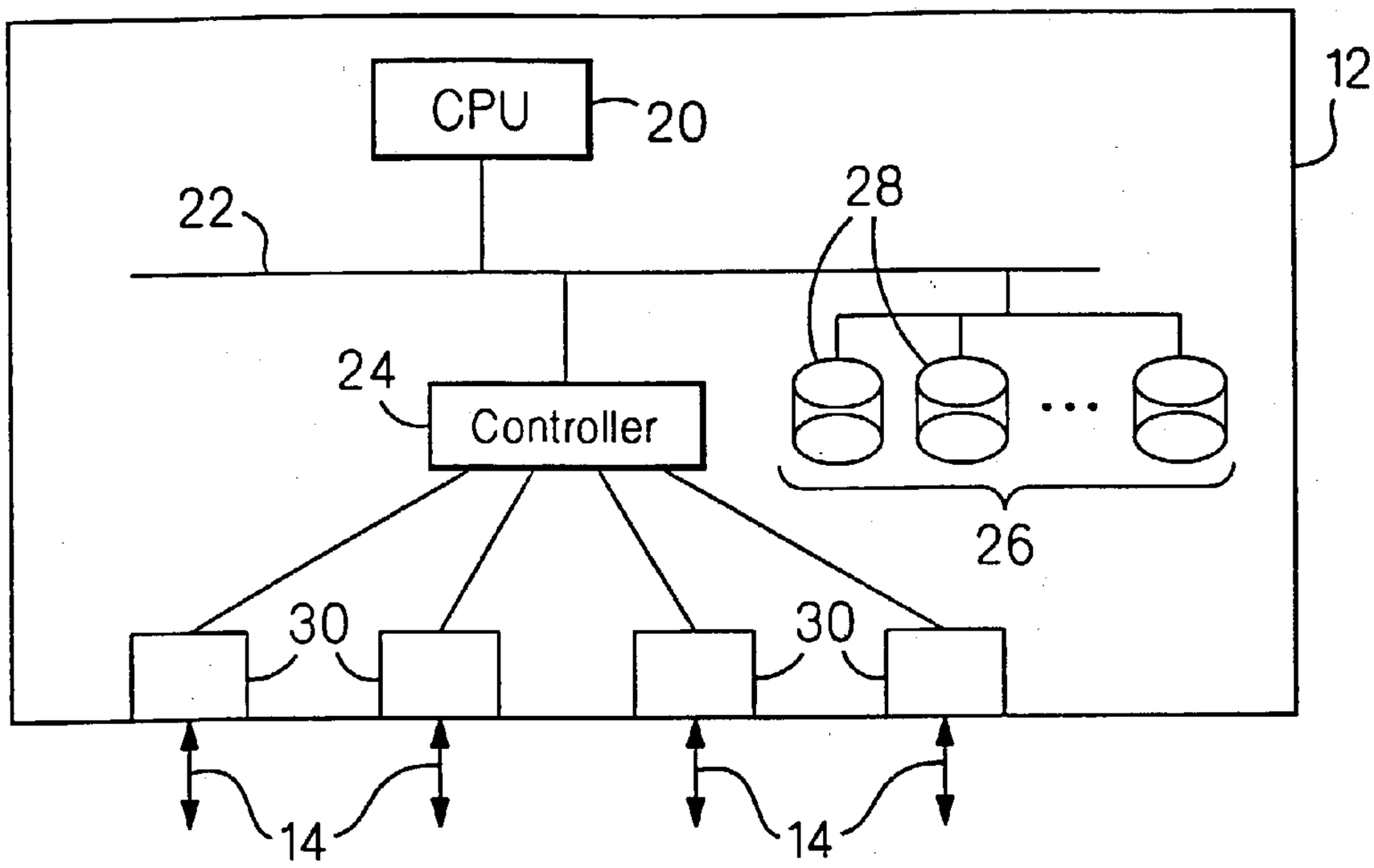


FIG. 2

Proc 12a	Proc 12b	Proc 12c	Proc 12d	Proc 12e
Block 1	Block 2	Block 3	Block 4	Parity(1-4)
Parity(5-8)	Block 5	Block 6	Block 7	Block 8
Block 9	Parity(9-12)	Block 10	Block 11	Block 12
Block 13	Block 14	Parity(13-16)	Block 15	Block 16
•	•	•	•	•
•	•	•	•	•
•	•	•	•	•

FIG. 3

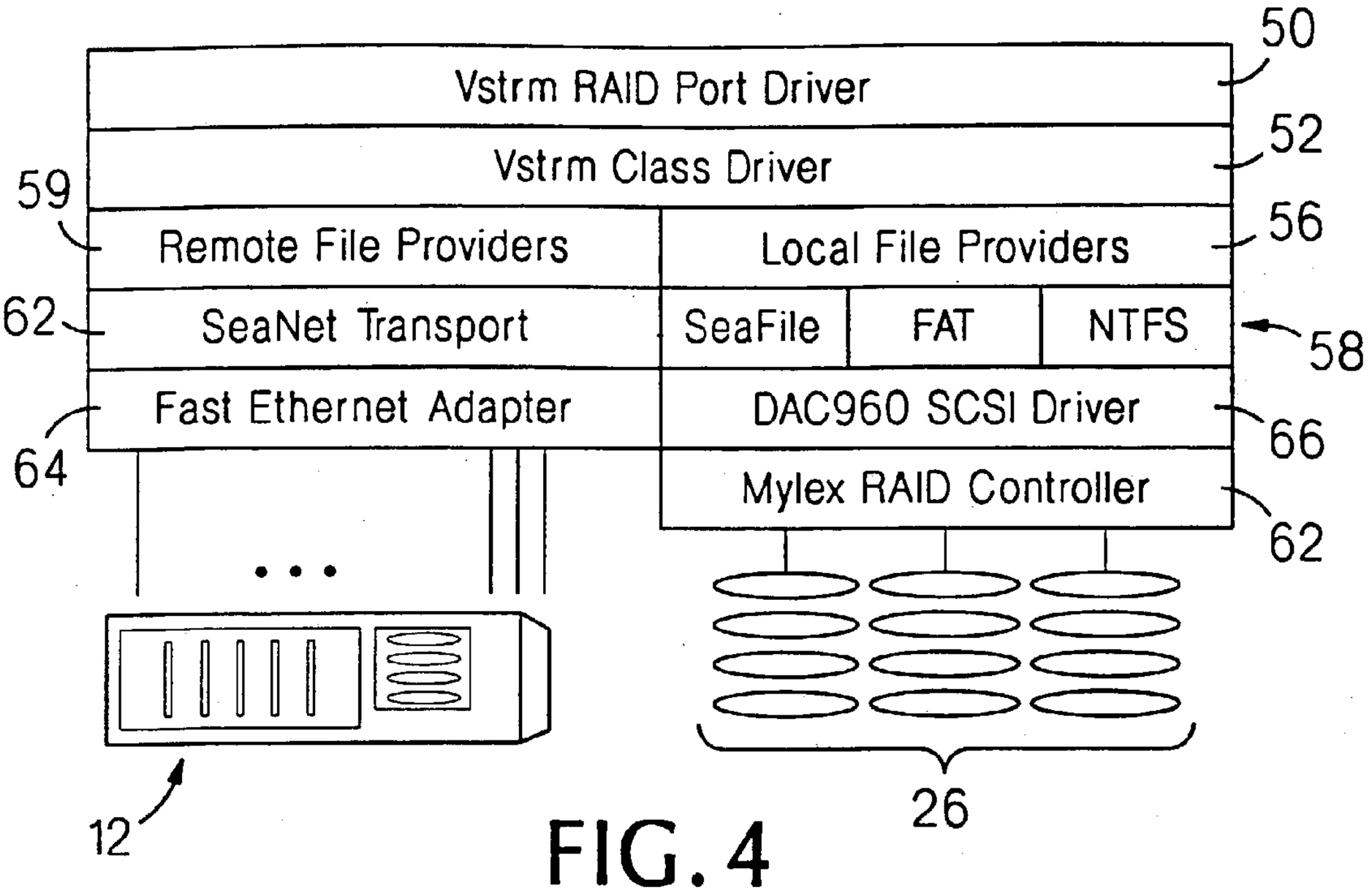

50
Vstrm RAID Port Driver
52
Vstrm Class Driver
59
Remote File Providers
Local File Providers
56
62
SeaNet Transport
SeaFile
FAT
NTFS
58
Fast Ethernet Adapter
DAC960 SCSI Driver
66
64
Mylex RAID Controller
62
12
26

FIG. 4

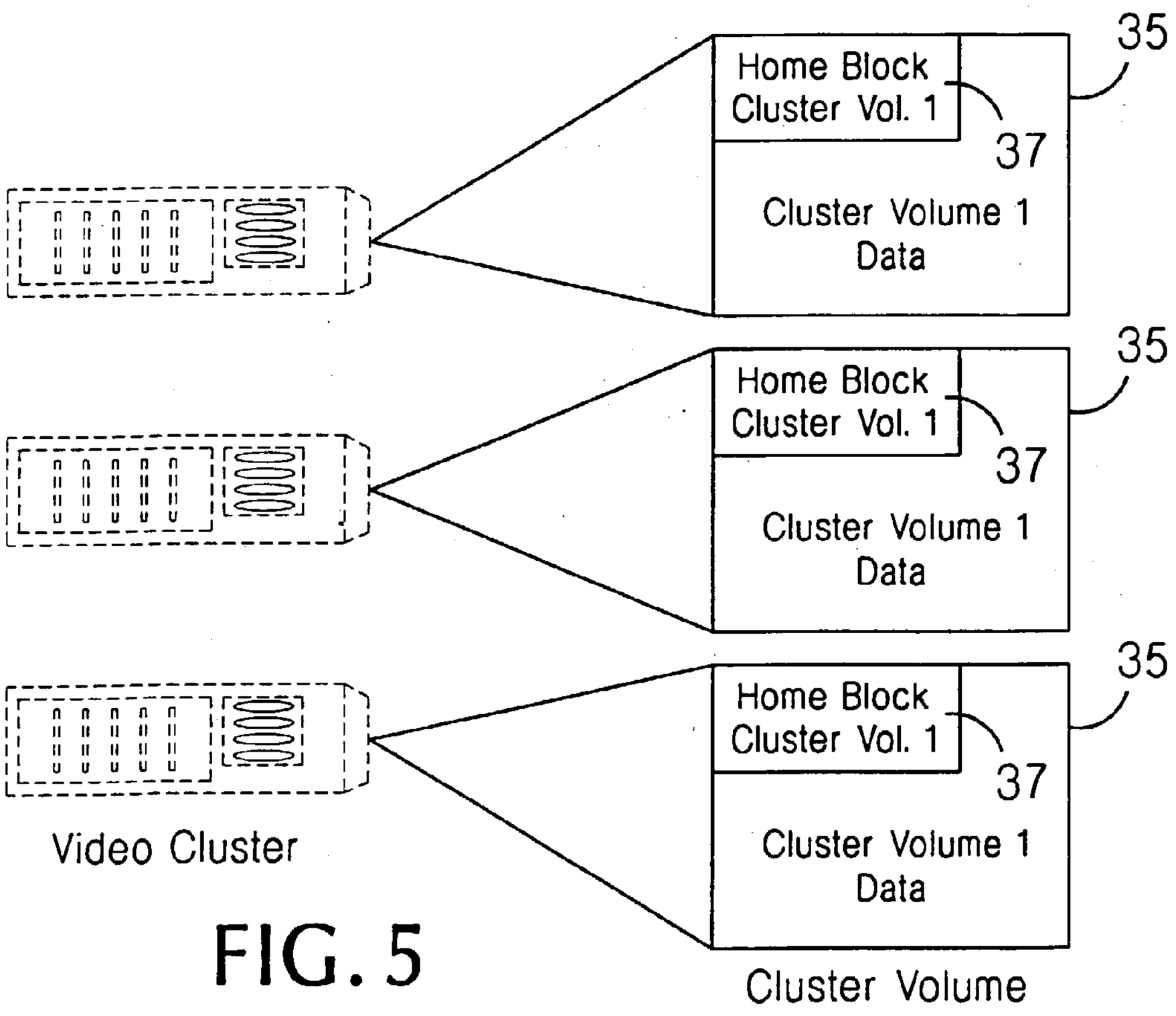

35
Home Block
Cluster Vol. 1
37
Cluster Volume 1
Data
35
Home Block
Cluster Vol. 1
37
Cluster Volume 1
Data
35
Home Block
Cluster Vol. 1
37
Cluster Volume 1
Data
Video Cluster
Cluster Volume

FIG. 5

Offset	Byte 3	Byte 2	Byte 1	Byte 0
0	Logical Id	Member Count	Minor Version	Major Version
4	Cluster Volume Id			
8	Cluster Volume Incarnation			
12	Cluster Volume Creation Time(Low)			
16	Cluster Volume Creation Time(High)			
20	Cluster Volume State			
24	Cluster Volume Name			
28	Cluster Volume Name			
32	Cluster Volume Name			
36	Cluster Volume Name			
40	Cluster Volume Name			
44	Cluster Volume Name			
48	Cluster Volume Name			
52	Cluster Volume Name			
56	MBZ			
60	MBZ			
64	MBZ			
68	MBZ			
72	MBZ			
76	MBZ			
80	Cluster Home Block Checksum			

FIG. 6

FIG. 6A

Field Name	Length	Default Value	Description
Major Version Number	1 Byte	1	The major version defines the format of the cluster volume.
Minor Version Number	1 Byte	0	The minor version of the cluster volume format.
Member Count	1, Byte	NA	Defines the number of cluster members participating in this cluster volume.
Logical Id	1 Byte	NA	Defines the unit id of the local cluster volume within the cluster volume.
Cluster Volume Id	4 Bytes	NA	This is the identifier for the cluster volume. It is assigned when the volume is created and is never modified.
Cluster Volume Incarnation	4 Bytes	NA	This incarnation is used to detect cluster members with stale cluster volume data.
Cluster Volume Creation Time	8 Bytes	NA	The time and date the cluster volume was created.
Cluster Volume State	4 Bytes	NA	the current state of the cluster volume, TBS.
Cluster Volume Name	32 Bytes	NA	The name of the cluster volume.
MBZ	24 Bytes	NA	Reserved. Must be zero.
Cluster Home Block Checksum	4 Bytes	NA	Checksum of the Cluster Volume Home Block.

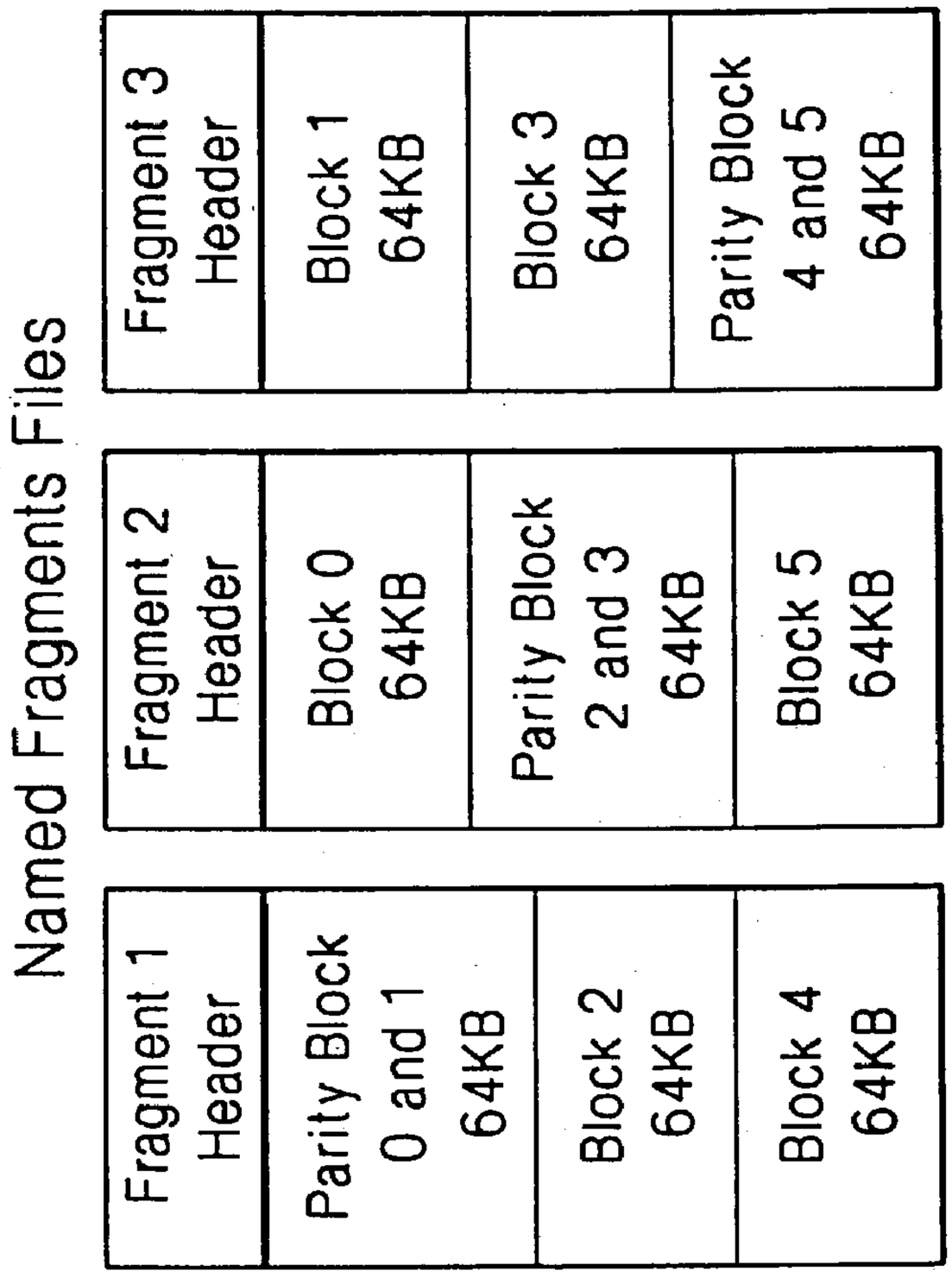

Named Fragments Files
Fragment 1 Header
Parity Block 0 and 1 64KB
Block 2 64KB
Block 4 64KB
Fragment 2 Header
Block 0 64KB
Parity Block 2 and 3 64KB
Block 5 64KB
Fragment 3 Header
Block 1 64KB
Block 3 64KB
Parity Block 4 and 5 64KB

FIG. 6C

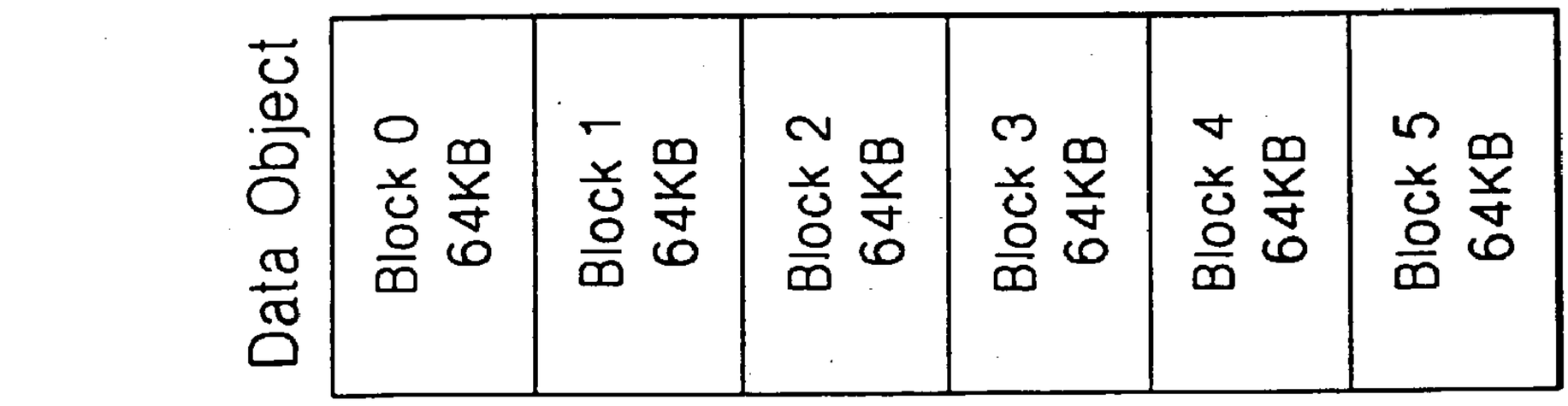

Data Object
Block 0 64KB
Block 1 64KB
Block 2 64KB
Block 3 64KB
Block 4 64KB
Block 5 64KB

FIG. 6B

Offset	Byte 3	Byte 2	Byte 1	Byte 0
0	RAID Level	Fragment Count	Minor Version	Major Version
4	Data Object Stripe Size			
8	Data Object Incarnation			
12	Named Fragment File Creation Time(Low)			
16	Named Fragment File Creation Time(High)			
20	Data Object Name			
24	Data Object Name			
28	Data Object Name			
32	Data Object Name			
36	Data Object Name			
40	Data Object Name			
44	Data Object Name			
48	Data Object Name			
...	...			
504	MBZ			
508	Header Block Checksum			

FIG. 6D

Field Name	Length	Default Value	Description
Major Version Number	1 Byte	1	The major version defines the format of the cluster volume.
Minor Version Number	1 Byte	0	The minor version of the cluster volume format.
Fragment Count	1. Byte	NA	Defines the number of fragments composing the data object.
Raid Level	1 Byte	5	RAID 0(0) or RAID 5(5)
Data Object Stripe Size	4 Bytes	64KB	Stores the stripe size of the data object in bytes.
Data Object Incarnation	4 Bytes	NA	This incarnation is used to detect cluster members with stale cluster volume data.
Data Object Creation Time	8 Bytes	NA	The time and date the cluster volume was created.
Data Object Name	32 Bytes	NA	The name of the cluster volume.
MBZ	24 Bytes	NA	Reserved. Must be zero.
Header Block Checksum	4 Bytes	NA	Checksum of the Cluster Volume Home Block.

FIG. 6E

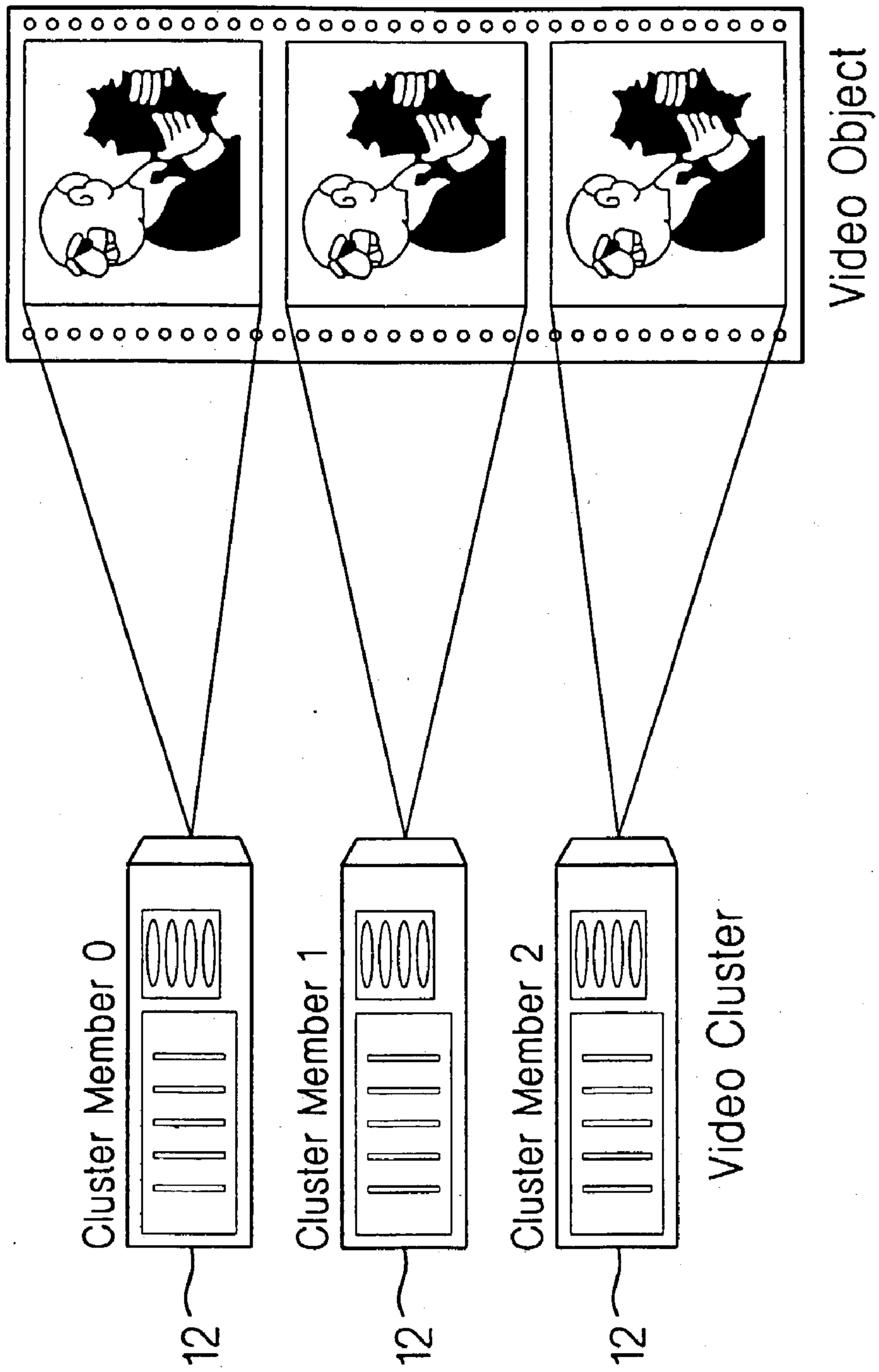

Video Object
Cluster Member 0
Cluster Member 1
Cluster Member 2
Video Cluster
12
12
12

FIG. 6F

Local Block Number	Cluster Member 0	Cluster Member 1	Cluster Member 2	Cluster Member 3	Cluster Member 4/5
0	Parity Block 0/1	Block 0	Block 1	Block 3	Block 4/5
1	Block 2	Parity Block 2/3	Block 3	Block 5	Block 6
2	Block 4	Block 5	Parity Block 4/5	Block 6	Block 7
3	Parity Block 6/7	Block 6	Block 7	Parity Block 6/7	Block 8

FIG. 7

Variable	Acronym	Description
Block Number	BN	Block Number to be located
Parity Block Count	PBC	Parity Block Count up to the last row
Adjusted Block Number	ABN	Block Number plus Parity Block Count
Optional Parity Count	OPC	Optional Parity Block in last row
Final Block Number	FBN	Adjusted Block Number plus Optional Parity Count
Cluster Member	CM	Cluster member storing final block number
Local Block Number	LBN	Cluster Member Local Block Number
Parity Block Number	PBN	Parity Block Number for Block Number
Cluster Size	CS	Number od Cluster Members

FIG. 8

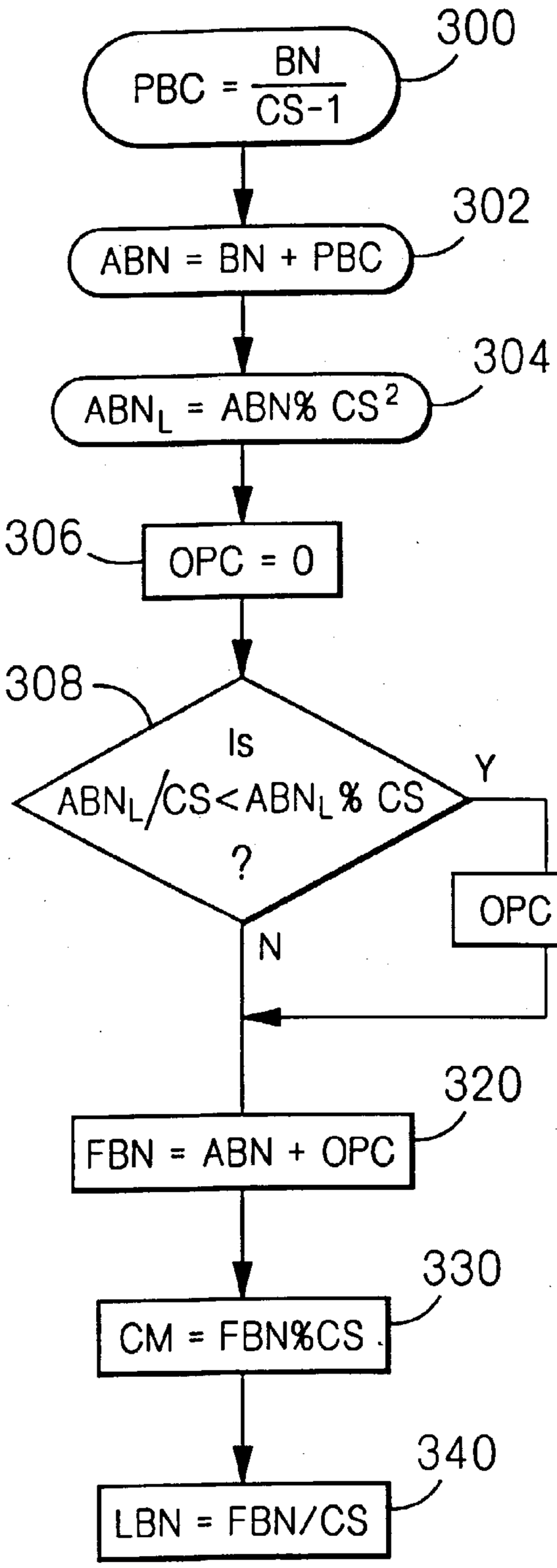


FIG. 9A

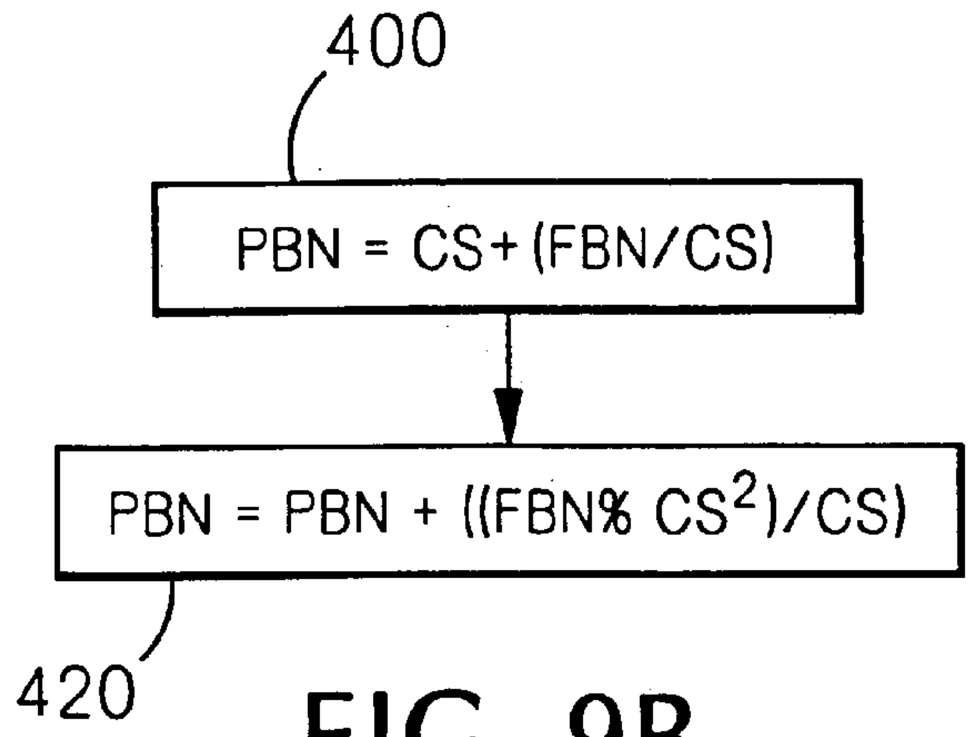


FIG. 9B

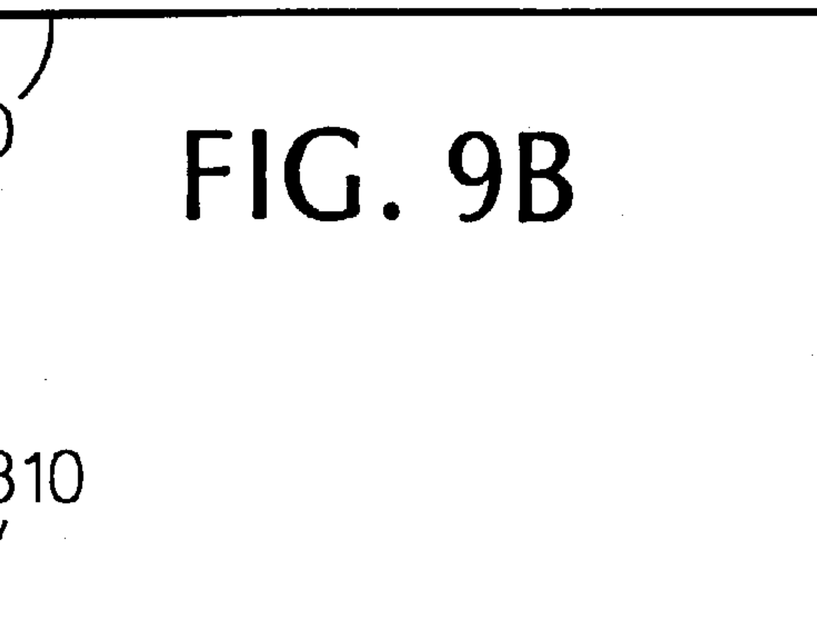


FIG. 9C

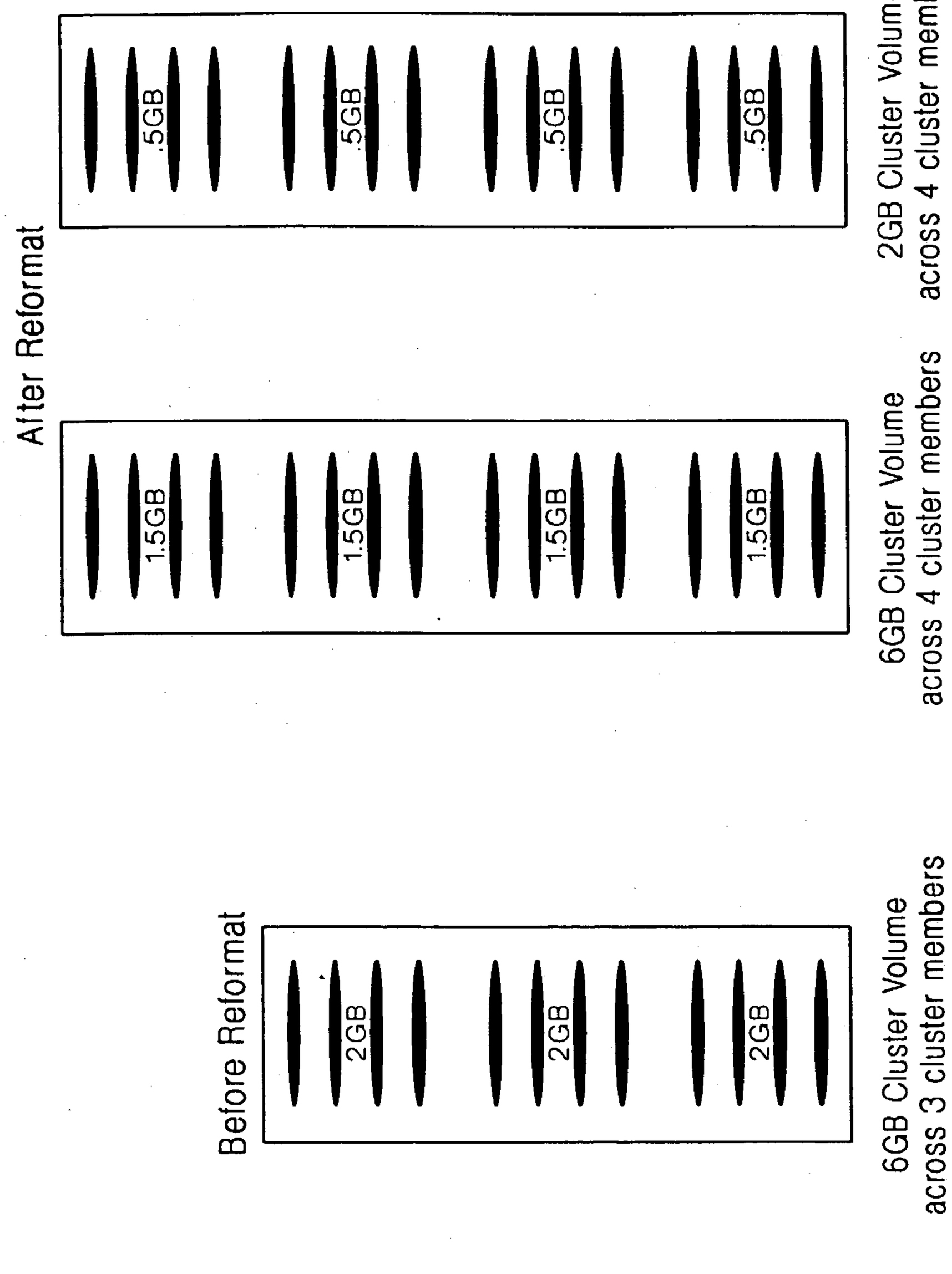

Before Reformat
2GB
2GB
2GB
6GB Cluster Volume
across 3 cluster members
After Reformat
1.5GB
1.5GB
1.5GB
1.5GB
6GB Cluster Volume
across 4 cluster members
.5GB
.5GB
.5GB
.5GB
2GB Cluster Volume
across 4 cluster members

FIG. 10

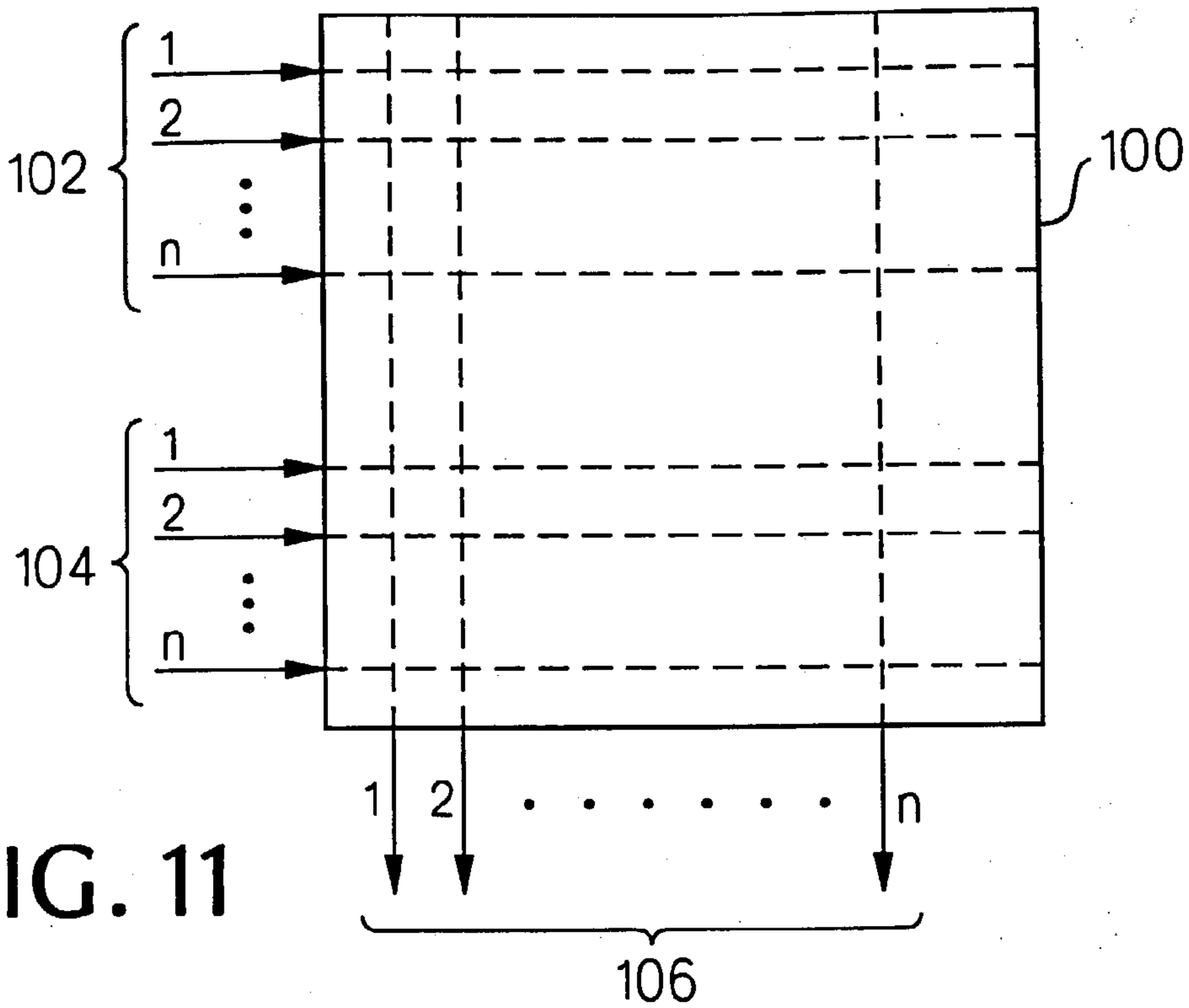

102
1
2
n
100
104
1
2
n
1
2
n
106

FIG. 11

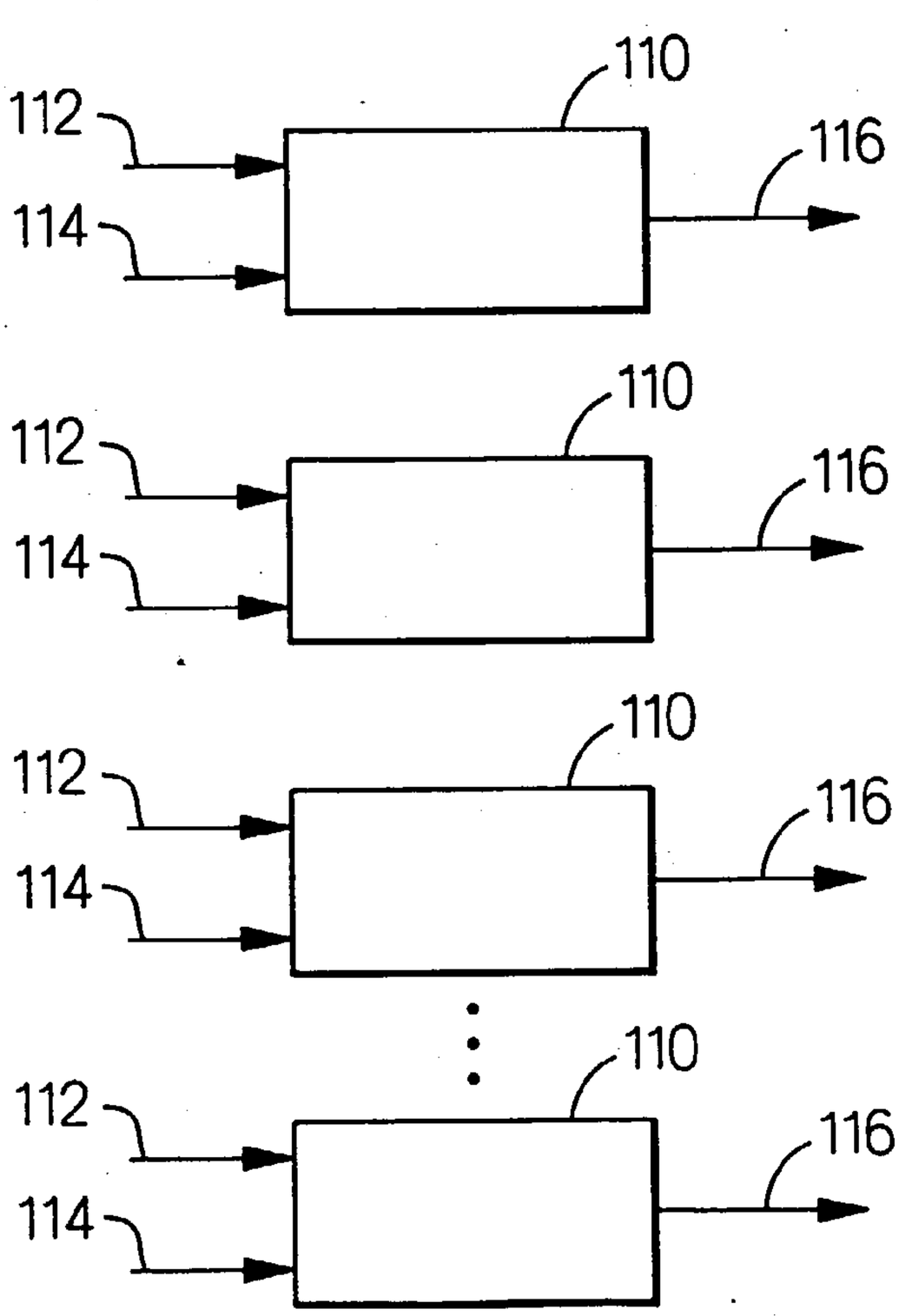

110
112
114
116
110
112
114
116
110
112
114
116
110
112
114
116

FIG. 12

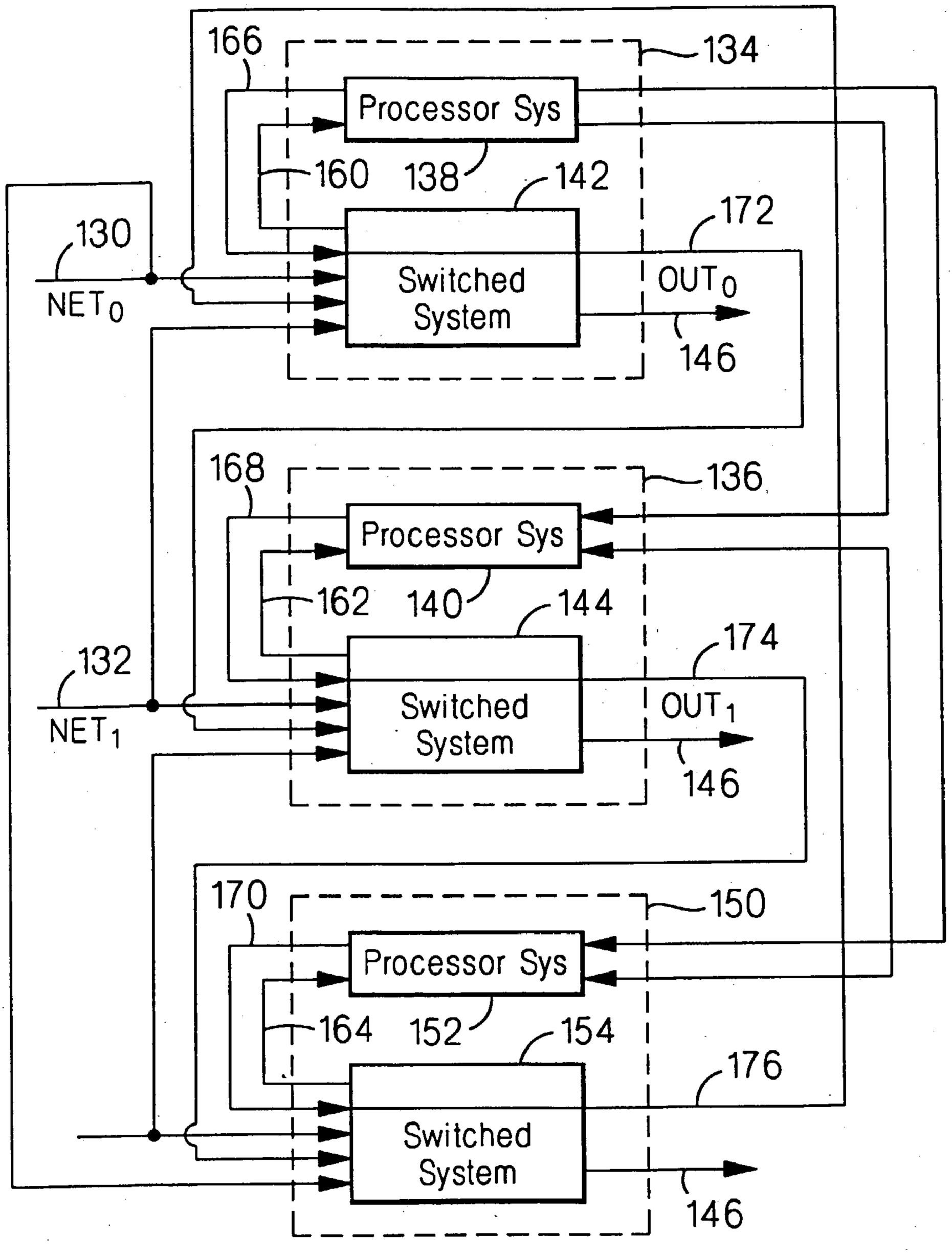

166
134
Processor Sys
160
138
142
172
130
Switched System
OUT0
NET0
146
168
136
Processor Sys
162
140
144
174
132
Switched System
OUT1
NET1
146
170
150
Processor Sys
164
152
154
176
Switched System
146

FIG. 13

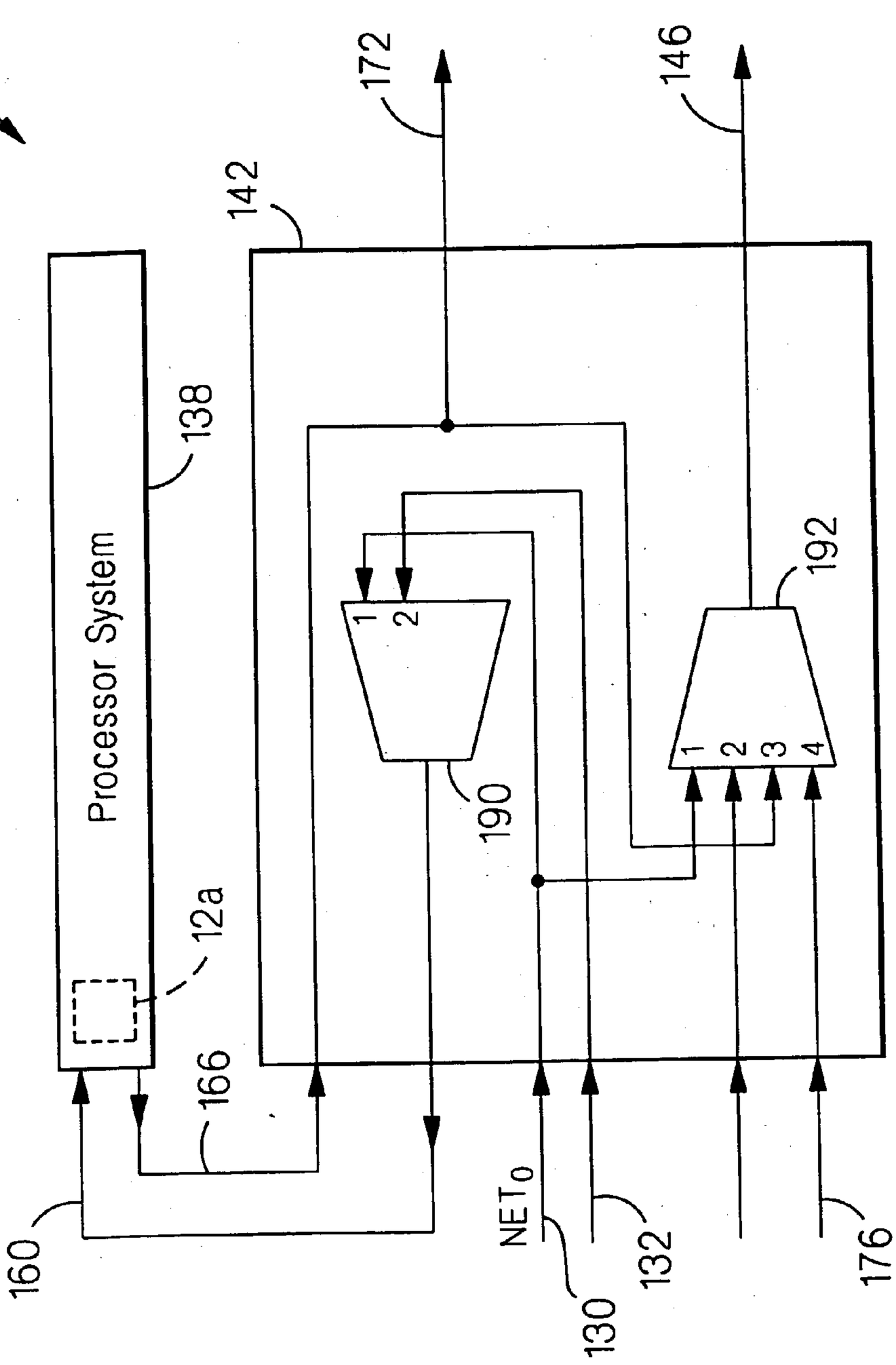

134
142
172
146
Processor System
138
12a
166
160
190
192
1
2
1
2
3
4
NET0
130
132
176

FIG. 14

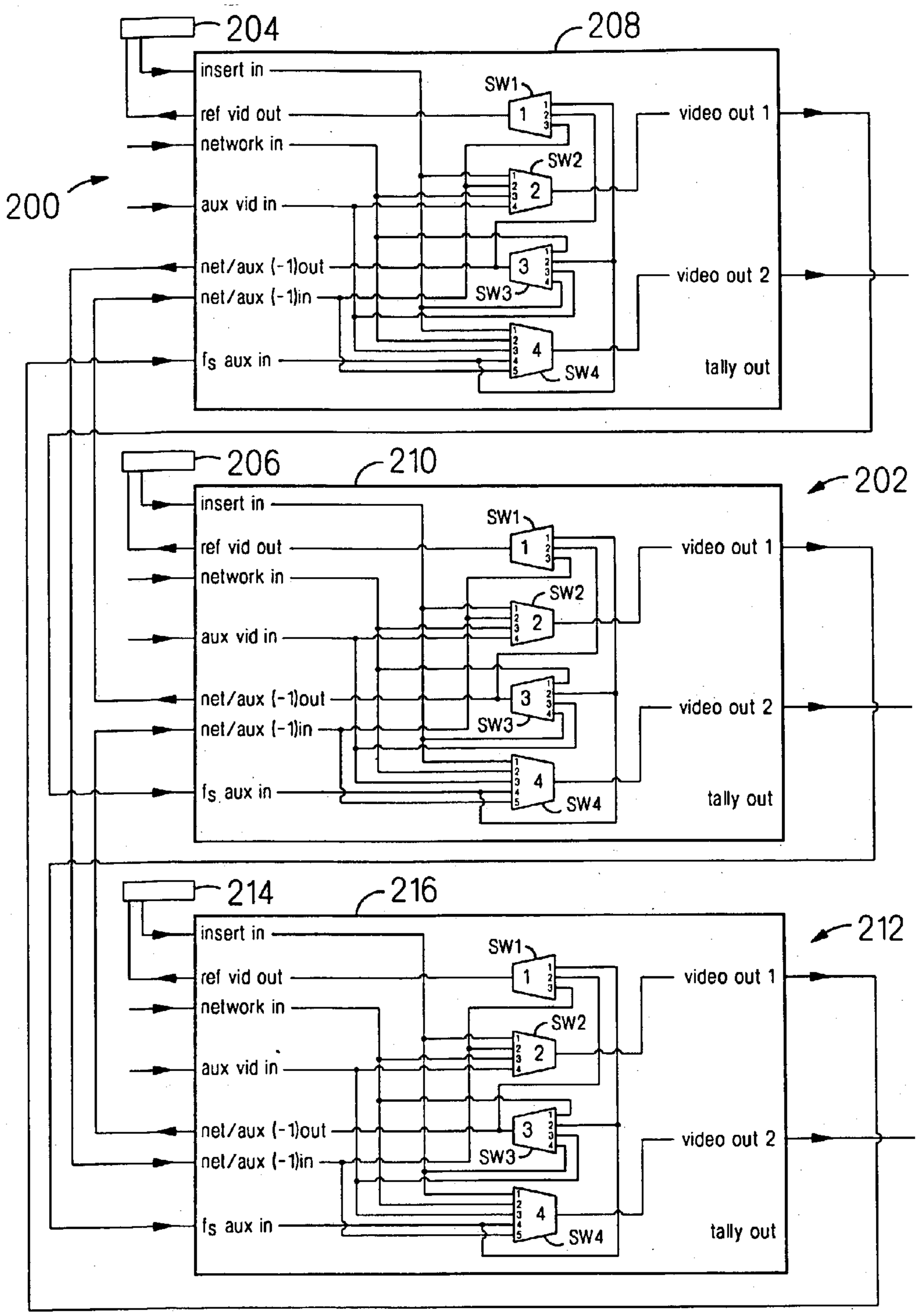

200
204
208
insert in
ref vid out
network in
aux vid in
net/aux (-1)out
net/aux (-1)in
fs aux in
SW1
SW2
SW3
SW4
video out 1
video out 2
tally out
206
210
202
214
216
212

FIG. 15

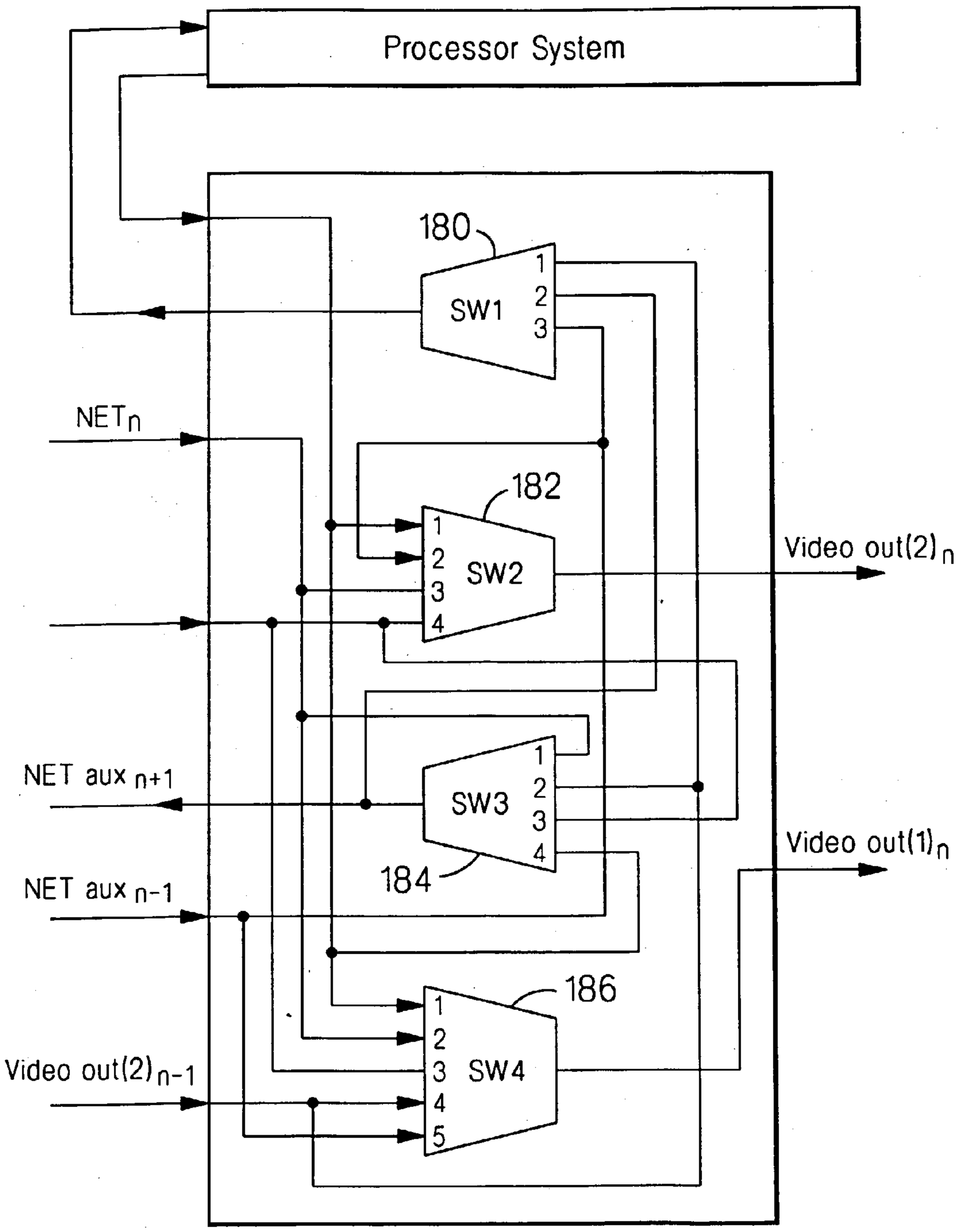

Processor System
180
SW1
1
2
3
NET n
182
SW2
1
2
3
4
Video out(2) n
NET aux n+1
SW3
1
2
3
4
184
NET aux n-1
Video out(1) n
186
SW4
1
2
3
4
5
Video out(2) n-1

FIG. 16

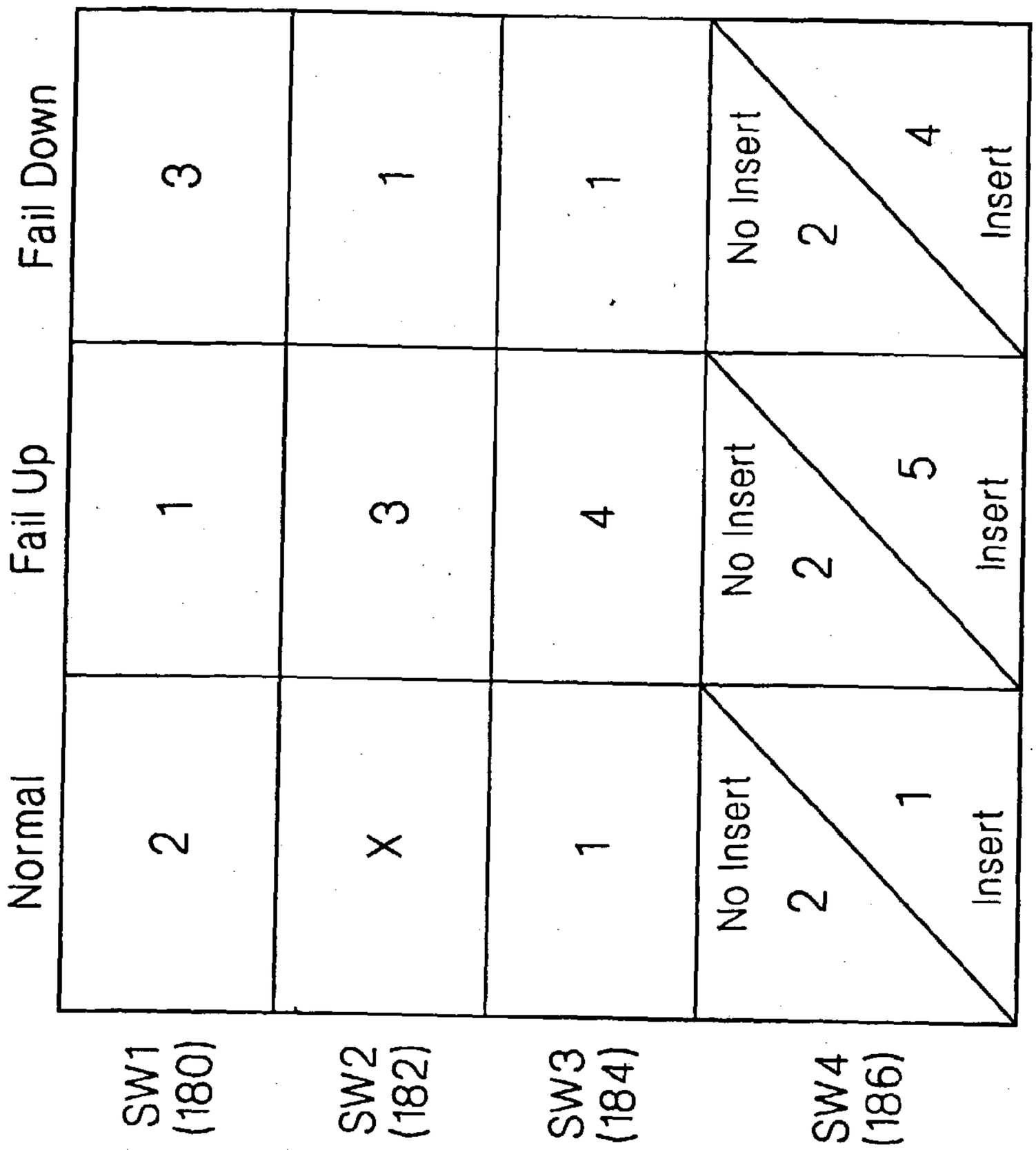

SW1 (180)
SW2 (182)
SW3 (184)
SW4 (186)
Normal
Fail Up
Fail Down
2
X
1
No Insert
2
1
Insert
1
3
4
No Insert
2
5
Insert
1
1
No Insert
2
4
Insert

FIG. 17

LOOSELY COUPLED MASS STORAGE COMPUTER CLUSTER

[0001] The invention relates generally to mass storage device interconnections and in particular, to a method and apparatus for increasing delivery bandwidth, providing fault tolerance, and input/output load balancing in a multiprocessor computer cluster.

BACKGROUND OF THE INVENTION

[0002] Modern reliable computer systems require a large capacity mass storage, and large bandwidth access to that mass storage. While disk sizes have increased substantially, for example a typical personal computer today can be configured with over a gigabyte of storage, the bandwidth available to access the storage has improved, but not significantly. Thus, while large volumes of information can be stored, the rate at which the storage can be accessed has not generally changed in the past few years. In particular, considering the requirements of a digital video system for the delivery of constant bit rate video streams such as MPEG-2 digital video streams, it is important to provide a high bandwidth to accommodate the data requirements of the digital video network, for example an ATM OC-3 network interface.

[0003] While various methods have been employed to provide sufficient quantities of data in a reliable configuration, perhaps one of the more popular, and least expensive, approaches is the use of RAID-5 striping and parity techniques to organize data in a fault tolerant and efficient manner. The RAID (Redundant Array of Inexpensive Disks) approach is well described in the literature and has various levels of operation, including RAID-5, and the data organization can achieve data storage in a fault tolerant and load balanced manner.

[0004] In particular, RAID-5 provides that the stored data is spread among three or more disk drives, in a redundant manner, so that even if one of the disk drives fails, the data stored on the drive can be recovered in an efficient and error free manner from the other storage locations. This method also advantageously, using RAID-5 striping, makes use of each of the disk drives in relatively equal and substantially parallel operations. Accordingly, if one has a six gigabyte cluster volume which spans three disk drives, each disk drive would be responsible for servicing two gigabytes of the cluster volume. Each two gigabyte drive would be comprised of one-third redundant information, to provide the redundant, and thus fault tolerant, operation required for the RAID-5 approach.

[0005] Consider a processor reading a video data object from a local RAID-5 array. In normal operation, using a RAID-5 approach, when the processor needs to access a video data object which is spread across all of the disk drives connected to the processor, the processor reads a portion of the video data object in a round robin fashion from each of the disk drives. For example, a first 64 kilobyte block of the video data object can be stored and read on a first drive, the next 64 kilobyte block being stored on the second drive, and so on. In addition, however, the parity check (actually an EXCLUSIVE-OR function), also a 64 kilobyte block, is stored so that if there were n disk drives there would be one parity block written for each n−1 blocks of data.

[0006] The processor reading the disk drives, however, is still "stuck" with a relatively narrow bandwidth. Accordingly, the amount of data which can be read is limited by the bus to which the drives are connected. For example, a SCSI bus which, while providing substantial improvements over buses from years ago, is still relatively slow compared to the needs of video applications. Also, the use of a local RAID-5 controller can combine the outputs of multiple local SCSI buses, but is subject to the failure of the local processor. Such a failure eliminates access to all the data.

[0007] Accordingly, objects of the invention are a method and apparatus having improved and increased mass storage read and write bandwidth (delivery bandwidth), operating using a reliable and fault tolerant protocol in a novel topology and enabling large quantities of data to be read and written in accordance with well known and accepted techniques. Other objects of the invention include a method and apparatus which is relatively inexpensive, reliable, simple to build, and easy to maintain.

SUMMARY OF THE INVENTION

[0008] The invention relates to a method and apparatus for redundantly storing data in a distributed computer system having at least three processor systems, each processor system having at least one central processing unit and at least one mass storage sub-system. The method features the steps of interconnecting each one of the processor systems in a point to point two way channel interconnection with each other one of the processor systems and storing input data across the processor systems according to a distributed, redundant storage process. Thereby, data is stored at each mass sub-storage system and some of a redundant representation of the data is stored also at each processor mass storage sub-system.

[0009] In particular aspects of the invention, the method features storing data across the processor systems according to a RAID-5 process and, further, storing data within each processor system according to a RAID-5 process.

[0010] The method further features the step of reading data from the computer system, in the absence of a failure of any of the processor systems, over respective ones of the data channel interconnections, whereby the reading step establishes a load balance across the processor systems. In the presence of a failure of one of the processor systems, the reading of data from the computer system features the steps of reading data from each non-failed processor system storing the data, and reading redundant data from the non-failed processor systems in place of the data stored at the failed processor system. Thereafter, the needed data stored at the failed processor system can be recreated using the redundant data and the data read from the non-failed processor systems. In some embodiments of the invention, during the time when a failure has occurred at any processor system, the method may prevent the writing of any data at any processor system until the failed processor system is brought back on line.

[0011] In another aspect, the invention further features the limiting case wherein there are only two processor systems initially. In accordance with this aspect of the invention, the system continues, in the absence of a failure, to provide increased bandwidth by reading succeeding blocks of data from alternate processors in sequence; and, in this manner,

effects a load balancing and provides increased read bandwidth compared to a typical so-called "mirrored" system. In a typical mirrored system, data is read from one processor only, the other processor acting as a backup. Thus, in accordance with the invention, data is read from all of the processors thereby providing an increased read bandwidth and load balancing. As noted hereinafter, therefore, the two processor version of the invention, while not providing all of the advantages of the system with a larger number of processors, does allow easy scalability to a processor system having greater capacity, less overhead, and greater bandwidth.

[0012] In various aspects of the storage step, the method, in some embodiments of the invention wherein data storage is modelled at a disk abstraction level, feature either designating one processor system to effect all write functions for the computer system, designating one processor for allocating files for each data input and enabling all processor systems to write input data to each of its associated and allocated files, or arbitrating write operations among the processor systems using a distributed lock manager.

[0013] In another aspect of the storage step, however, the data input is stored as named fragment files, or named files, in each processor system. When stored as named fragment files, or named files, they can be accessed and reconstructed, for example even when a process or system is added to the distributed computer system. The system continues to deliver stored data as an output stream even as the number of processor systems, network interfaces, and amount of storage is changed. This is possible because the method uses file names to distinguish, modulus "N", data fragments from modulus "N+1" data fragments, even as these modulus "N+1" data fragments are created from the modulus "N" fragments. Further, the method features the step of reconstructing a failed processor system by reconstructing only the data objects which were written while the processor system was in a failed state.

[0014] The distributed computer system in accordance with the invention has at least three processor systems for redundantly storing data, each processor system having at least one central processing unit and at least one mass storage system. The distributed computer system features interconnecting channels providing a point to point two way channel interconnection from each one of the processor systems to each other one of the processor systems, and a data storage controller at each processor system. The controllers act to store data input at any one of the processor systems according to a distributed redundant storage process whereby data is stored at each of the computer processors and some of a redundant representation of the data is stored also at each of the processors. In a preferred embodiment, the storage controllers store data across the processing systems according to a RAID-5 process, and further, can store data at each processor system in its associated mass storage sub-system according to a RAID-5 process.

[0015] The apparatus of the invention further features a system in which the controllers read data from the computer system, in the absence of a failure of any processor system, so as to maintain and establish a load balance across the computer system. In the presence of a failure of one of the processor systems, the controllers read data from each non-failed processor system storing the data (over the appropriate dedicated network connection) and read redundant data from each non-failed processor system in place of the data stored at the failed processor system. The requesting processor system can then recreate the data stored at the failed processor using the read data and the redundant data. In a preferred embodiment, an "external" processor can be employed to recreate the data stored at the failed processor, thereby preserving the delivery bandwidth of the system even in the face of a failed processor.

[0016] In another aspect, the apparatus features a storage controller which stores the input data as named fragment files, or named files, in the distributed computer system.

[0017] The apparatus of the invention also relates to a redundant switch having at least n interruptible inputs, n interrupting inputs, and n outputs. The redundant switch features n+1 switched systems, each switched system having at least two control inputs, a first input, a second input, a third input, and a fourth input, and a first and a second output. Each switched system is connected at its second output to n interrupting signal generator, n interrupting output of the associated signal generator being connected to the second input of the connected switched system. The switched systems are interconnected in a ring structure so that each switched system further has n interruptible input signal connected to the first input, the second input of a first neighbor switched system in the ring being connected to the third input, the interruptible input from the other neighbor switched system on the ring being connected to the fourth input, and each switched system having switching circuitry responsive to the control input for switching any of its inputs to at least its first output and for connecting either of its first and fourth inputs to its second output. The controller provides the control signals to the control inputs of the switched system to enable the cross-bar operation.

[0018] In another aspect, the redundant switch has n+1 switched systems each switched system having at least two control inputs, four signal inputs, and two signal outputs. Each switched system is connected at one of its outputs to an associated interrupting signal generator and n interrupting output of the associated signal generator is connected to an input of the connected switched system. The switched systems are interconnected in a ring structure so that each switched system is connected to a first and a second nearest neighbor. Each switched system has switching circuitry responsive to the control input for the switched system for selectively switching its inputs to its outputs. A controller provides the control inputs to the switched systems to enable the switched systems to effectively rotate signal switching functions one position in one or both directions around the ring. Thereby, a failed signal generator can be bypassed and the signals on the n first outputs continue uninterrupted.

[0019] In another aspect, the invention relates to a distributed data delivery system having at least three processor systems for redundantly storing data, each processor system having at least one central processing unit and one mass storage system. Interconnecting data channels provide a point to point, two way, channel interconnection from each one of the processor systems to each other one of the processor systems. A data storage controller at each processor system stores data input from any one of the processor systems according to a distributed, redundant storage process whereby data is stored at each of the processor systems

and some of a redundant representation of the data is stored at each of the processors as well. A switching circuit having n interruptible input ports, at least n+1 interrupting input ports, and at least n+1 output ports, has n interruptible input signals connected to respective primary processor systems, each primary processor system having an output connected to a respective interrupting input. A switching controller, connected to the switching circuit for selectively interrupting each of the n interruptible input ports with the signal information available from a respective one of the processor systems, in the event of a failure at a processor, and using a previously unused processor system in that process, causes the switching circuit to connect the interruptible input port of the failed processor system to a different processor system, and to replace the output of the failed processor system with the output of another processor system.

BRIEF DESCRIPTION OF THE DRAWINGS

[**0020**] Other objects, features, and advantages of the invention will be apparent from the following drawings taken together with the description of a particular embodiments in which:

[**0021**] **FIG. 1** is a schematic block diagram of an interconnected computer system according to the invention;

[**0022**] **FIG. 2** is a more detailed block diagram of a processor system according to the invention;

[**0023**] **FIG. 3** is a table illustrating an index file in accordance with the invention;

[**0024**] **FIG. 4** is a diagram illustrating the software architecture in accordance with the invention;

[**0025**] **FIG. 5** illustrates the cluster volume arrangement according to the invention;

[**0026**] **FIG. 6** describes the cluster volume HomeBlock format in accordance with the invention;

[**0027**] **FIG. 6A** is a table defining the HomeBlock format of **FIG. 6**;

[**0028**] **FIG. 6B** shows a data object broken into a plurality of blocks;

[**0029**] **FIG. 6C** shows the relationship of a data object to its stored named fragment files;

[**0030**] **FIG. 6D** is a table defining the header block of a named fragment file according to one embodiment of the invention;

[**0031**] **FIG. 6E** is a table defining the data object format of **FIG. 6D**;

[**0032**] **FIG. 6F** illustrates reading a video object from the cluster members;

[**0033**] **FIG. 7** illustrates the data and parity organization for a data object according to a RAID-5 striping protocol;

[**0034**] **FIG. 8** is a table describing the terminology used in connection with the flow chart of **FIG. 9**;

[**0035**] FIGS. **9**A-**9**C are flow charts for determining data and ParityBlock locations according to the invention;

[**0036**] **FIG. 10** illustrates a cluster volume both before and after reformatting in accordance with the invention;

[**0037**] **FIG. 11** illustrates a 2n×n switch;

[**0038**] **FIG. 12** illustrates n 2×1 switches in a circuit arrangement;

[**0039**] **FIG. 13** illustrates a 2n×n system in accordance with the invention;

[**0040**] **FIG. 14** is a more detailed block diagram of the switched system of **FIG. 13**;

[**0041**] **FIG. 15** illustrates a video system in a ring configuration accordance with an alternate embodiment of the invention;

[**0042**] **FIG. 16** illustrates, in more detail, the switched system of **FIG. 15**; and

[**0043**] **FIG. 17** illustrates a truth table for the switched system of **FIG. 16** in accordance with the invention.

DESCRIPTION OF THE PREFERRED PARTICULAR EMBODIMENTS

[**0044**] Referring to **FIG. 1, a** redundant distributed computer system **10** has a plurality of processor systems **12***a*, **12***b*, **12***c*, **12***d*, **12***e*, in the illustrated embodiment, which are interconnected by interconnecting channels **14***a*, **14***b*, **14***c*, . . . **14***j* in a plurality of point to point channel interconnections. Thus, each processor system **12** is directly connected in a point to point connection to each other processor system. In other embodiments of the invention, more or less processor systems can be used, although a practical upper limit may be between nine and thirteen and the lower limit is three. (As noted earlier, a two processor system, can be used, to effect some of the advantages of the invention; however, for purposes of description hereinafter, the three or more processor embodiment will be detailed.)

[**0045**] Referring in more detail to each processor system **12**, and referring to **FIG. 2**, each processor system **12** has a CPU **20** connecting, in the illustrated embodiment, to an internal data communications bus **22**, to which are connected at least a memory and communications controller **24** and a mass memory storage unit **26**. The mass storage unit typically has a plurality of disk drive units **28**. Controller **24** is also connected to a plurality of channel interface units **30**, each of which is connected to a different interconnecting channel **14** for establishing the point to point communications with other processor systems through their respective channel interface units **30**.

[**0046**] In the illustrated embodiment of the invention, the interconnecting channels **14** use a protocol running on Fast Ethernet datalink devices. This protocol provides a relatively high efficiency and enables communications between processors, in the illustrated embodiment, at a bandwidth on the order of 100 megabits/sec. Accordingly, referring to **FIG. 1**, each processor **12**, being connected to four interconnecting channels, has a bandwidth to the external processor memories of approximately 400 megabits/sec. (less overhead), in addition to its own capabilities with regard to its local mass storage **26**.

[**0047**] In one particular application of the computer system illustrated in **FIG. 1**, video input information and data is provided to one or more processor systems **12** over external feedlines, for example, network feeds **32** which require high bandwidth storage of the substantial data

needed to represent and store even small durations of video data (including audio). In particular, for example, one minute of video data can require sixty-six megabytes of storage. Fortunately, the particular embodiment described herein and displayed in **FIG. 1** has substantial bandwidth to enable the video information to be distributed among the various processor systems so that in a preferred embodiment of the invention the video data input to one of the processor systems **12** is actually stored along many, and preferably all of the video processor systems **12**.

[**0048**] In accordance with a particular embodiment of the invention, the controllers **24** of the processor systems **12** individually and collectively act to store data across the entire computer system **10** network in a redundant fashion so that if any one processor system **12** fails the remaining processor systems can nevertheless reconstruct all the data available in the entire system. In addition, this approach, as will be described in more detail below, provides, in the illustrated embodiment, load balancing across the various processing systems as well as enabling any one processor system requiring either to read or write data the capability of a very large bandwidth memory communication channel.

[**0049**] In the preferred embodiment of the invention, a RAID-5 architecture is implemented, for the first time, at the system level to provide the redundancy, load balancing, and bandwidth necessary to meet the objectives of the distributive computer system. In a particular application, assuming that video data is to be received by processor system **12***e* on input line **30**, the computer system **10** will have, before receipt of the data, allocated storage in each of the processor systems in one of a plurality of different ways. In one method, but not the preferred method which will be described hereinafter, a storage cluster volume having a specified capacity will be deployed among the processor systems **12**. For example, if the storage cluster volume was **10** gigabytes, in the embodiment of **FIG. 1**, each processor system would be responsible for servicing two gigabytes of the cluster volume for use not only for itself, but for the other processor systems or members of the cluster.

[**0050**] Under normal operating conditions therefore the systems will have preestablished a protocol for both writing and reading data. According to one protocol, the systems will have selected one system for writing of all data for the cluster volume. In another aspect of the invention, a distributed lock may be used to arbitrate write operations among the processor systems. In a third embodiment, one processor system can allocate files for each data input and thereafter enable each of the processor systems to write input data to its associated allocated files.

[**0051**] The cluster volume described above is a collection of logical disk blocks (some local and some remote) that are shared between cluster members (the different processor systems of the cluster). In this design, each cluster member has an instance of a file system running, and each node would have access to the entire set of logical blocks. While this solution works, it has several problems. First, only one system can write to the cluster volume; second, the integrity of the cluster volume has to be strictly managed; and third, changes to the file system structure (the creation or deletion of files) has to be replicated in each instance of the file system running on each cluster member.

[**0052**] Rather than use the cluster volume structure identified above, in a preferred embodiment of the invention, the cluster architecture provides access to data objects and named fragment files, much in the way a file server provides "files" to network clients. Since the cluster members keep their file system private and only export access to the data objects, each cluster member can read, write, or delete files from its local file system without disrupting the other members of the cluster. There is no significant penalty for this method and it reduces the complexity of the cluster software. Data objects are then fragmented and written to the members of a cluster using the RAID-5 striping and parity techniques, allowing each individual data object to be read, recovered, or written independently of all other data objects. In addition, because all data objects are stored independently, only the data objects written while a cluster member is down need be recovered. In the cluster volume method, as will be described below, the entire local cluster volume has to be reconstructed. The reconstruction of the entire cluster volume can take anywhere from several hours to several days depending upon the size of a volume. When only the data objects are stored, only a small subset of the data will need to be reconstructed if written during the time when a cluster member is down.

[**0053**] In order to achieve a redundant, and hence fault tolerant, storage system, each of the processors **12** is viewed, even though they exist at the system level, in a RAID-5 context, with striping, as follows. Assuming that each processor writes in **64** kilobytes blocks, a first block will be sent to and written by processor **12***a*, a second block by processor **12***b*, a third block by processor **12***c*, a fourth block by processor **12***d*, and a fifth block, a redundancy block or ParityBlock by processor **12***e*. In accordance with the RAID-5 approach, the redundancy block or ParityBlock to be written by processor **12***e* in its mass storage will be the EXCLUSIVE-OR of the blocks sent to processors **12***a*, **12***b*, **12***c*, and **12***d*. Clearly, other redundancy methods can be used including various forms of, for example, Huffman coding and other redundant coding methods so that not only may one failure of a processor be taken into account but multiple processor failures can be taken into account. The cost of course, is increased processing in both the writing and perhaps reading of data. Further, and importantly, because each processor is connected in a point to point two way connection to each other processor, it is possible to write all five blocks of data substantially in parallel, thus making full use of the bandwidth available to the writing controller and, at the same time, distributing substantially equally, the writing load across the entire computer system.

[**0054**] After the first four data blocks (and one redundancy block) have been written, the next block of data (a DataBlock) can be written to, for example, processor system **12***b*, the sixth block to processor system **12***c*, the seventh block to processor system **12***d*, and the eighth block to processor system **12***e*. Then, the parity or redundancy block would be written in processor system **12***a*. In accordance with this practice, each of the redundant blocks would be written in a determined and round robin, rotating manner, in accordance with the RAID-5 processing protocol. The location of the blocks is illustrated in **FIG. 3**. A short algorithm can be employed to determine the location of a particular block, as described in more detail below. Further,

[**0055**] Further, within each processor system itself, the processors can use a RAID-5 protocol, in its ordinary and well known sense, to store data among its plurality of disk

drive devices **26** associated with that processor. Thus, there is provided the novel circumstance of employing the RAID-5 technology twice, both at the storage level as is well known, but also at the system level, which is new, to achieve a high reliability, lower cost, computer system.

[**0056**] The structure of **FIG. 1** as described herein has a number of constraints in order to maintain its proper functionality. First, each cluster volume must have at least three members. Second, a cluster cannot continue to operate, if it uses the RAID-5 protocol, if more than one cluster member should fail. (The use of other, more complex protocols, can relax, somewhat this constraint.) Third, it appears that a practical limit, under current operating parameters, is nine to thirteen cluster members. When more cluster members are employed, the point to point wiring becomes progressively more difficult and expensive. Indeed, nine cluster members would require thirty-six interconnecting channels while thirteen cluster volumes would have **78** interconnecting channels.

[**0057**] Referring now to **FIG. 4**, the major software components of a single processor system **12** (also called a "cluster member") include a port driver **50**, a class driver **52**, a remote file provider **54** a local file provider **56**, a file system **58** (SeaFile, FAT, NTFS), a SCSI driver **60**, a RAID controller **62**, a fast Ethernet adapter **64**, and a SeaNet transport **66**.

[**0058**] The cluster architecture utilizes RAID-5 technology to build a fault tolerant distributed system. The data objects are stored as named fragment files across the members of the cluster. Data objects are striped (in accordance with RAID-5 protocol) and stored with parity information to allow a missing named fragment file to be reconstructed if a cluster member fails. This is described in more detail below.

[**0059**] In order to provide transparent data object access, a RAID port driver masks the fact that the data object exists as a set of named fragment files. It provides the multiplexing and demultiplexing services to merge the named fragment files into a data object byte stream. The RAID port driver registers as both a provider and a consumer. When the class driver attempts to open a data object, it calls all the provider port drivers in the system. Upon being called, the RAID port driver becomes a consumer and uses the class driver to open each of the data fragments that comprise the data object. Once a session has been established to each of the named fragment files, that is, once access has been provided to each named fragment file, the RAID port driver performs an open call back to notify the class driver that the data object is available.

[**0060**] In the particular application of accessing the data (video) objects, the port driver accesses the data (video) objects stored on the cluster. The video named fragment files are read or written using RAID-5 methods from the local and remote providers. It masks other port drivers (for example, video decoders or ATM links) from any failures, since it will reconstruct missing data fragments in real time. The remote file provider represents any third party application or device driver that might use the cluster technology. Examples include Lotus Notes, medical applications, or database systems. The on-disk structure of the cluster volume (file system **54**) can be either NTFS, FAT, SeaFile or raw disk access, in the illustrated embodiment. The file system component is responsible for storing and retrieving the named fragment files.

[**0061**] The Transport component **66** provides an efficient network service to other cluster members. It detects and reports failures in real-time to the RAID driver. The fast Ethernet Adapter provides a **100** Mb/second full duplex link between each cluster member through interface units **30**. The SCSI driver, for example a DAC960, provides access to local disk storage and allows the cluster RAID driver to read or write data for local file system managers.

[**0062**] A RAID controller provides efficient access to the named fragment files. The local portion of a data object is read or written by the RAID controller. The RAID controlled volume can be configured, to either a RAID-0, RAID-1, RAID-5, RAID-6 or RAID-7 level in the illustrated embodiment, and as noted above, configuring a volume in a RAID-5 protocol allows a cluster member to continue even when a single disk fails to perform properly.

[**0063**] As described above, when cluster volumes are used, a cluster volume is a logical disk volume that spans multiple cluster members or processor systems. Considering a specific application, such as the storage and retrieval of video data objects, the design principal for a cluster volume is to utilize a fraction of each processor system memory for each video stream thus creating a balanced and scalable system. Since the video data stored in the cluster volumes is mission critical, the integrity of the cluster volume must be carefully maintained. Accordingly, a set of checks are used to ensure that the cluster members agree on a consistent view of the cluster volume and that only one cluster member writes to the cluster volume.

[**0064**] In the illustrated embodiment of the invention, and referring to **FIG. 5**, each cluster member of a cluster volume that is, each processor system **12**, maintains and verifies its cluster volume "HomeBlock" to ensure the integrity of the cluster volume. When a cluster member processor system boots, it checks the cluster volume HomeBlock incarnation and volume identifier against the other cluster member processor systems HomeBlocks to ensure that the cluster volume was not modified while it was down. If the cluster volume was modified, a rebuild process can repair the out of date cluster member.

[**0065**] Accordingly, therefore, each cluster member **35**, in the illustrated embodiment of the invention, has one cluster volume HomeBlock **37** for each cluster volume. Referring to **FIGS. 6 and 6A**, the various offsets and byte identifications are detailed for the preferred embodiment of the invention. The cluster volumes are organized using the RAID-5 protocol across the processor systems which are the members of the cluster. (Note that a cluster volume need not extend across all processors **12** of the system **10**, but must extend across at least three processors, or two if mirroring is permitted.) The controllers organize the data and the writing controller writes blocks of data in a round robin fashion, as described above, across the cluster members participating in the cluster volume.

[**0066**] As noted above, in the preferred embodiment, data objects are employed. Each data object in this system is stored as a set of named fragment files. Each fragment file contains a header block that allows the named fragment file

to be self describing. Data objects are fragmented when they are written to the cluster. **FIGS. 6B and 6C** illustrate the relationship of a data object to its named fragment files. As illustrated, a fragment file written on any individual cluster member includes the named fragment header and the plurality of blocks comprising the fragment file. In the example, the data object is fragmented into three files.

[**0067**] The amount of data in a block is called the volume stripe size. In the illustrated embodiment of the invention the default stripe size is **64** kilobytes. In addition to striping, the RAID-5 protocol uses a ParityBlock to recover from a single fault. The ParityBlock is written for every n minus 1 blocks where n is the number of cluster processor system members. This technique, as noted above, thus allows for the reconstruction of the cluster volume data when any one cluster member fails. In the preferred embodiment of the invention, parity blocks are created by EXCLUSIVE-OR'ing the n−1 DataBlocks forming a stripe level. For the preferred embodiment of the invention, wherein data is stored as data objects, **FIG. 6D** describes the named fragment file header format. As noted above, the header block describes the content of the fragment. **FIG. 6E** describes, in table form, the particular components of the header for one particular embodiment of the invention.

[**0068**] **FIG. 6F** illustrates reading the DataBlocks of a single video object spread across the members of a cluster. To play this video object, a cluster member opens each named fragment file and reads the first block from cluster member **0**, for example, the second block from cluster member **1**, and the third block, assuming no parity block, from cluster member **2**. At this point the read process would cycle back to cluster member **0**. The complexity of this process is hidden, as noted above, from the consumer by the RAID port driver. Since, in the above description, the parity blocks which are stored with the named fragment files were ignored, in fact when the video data object is read the parity blocks are skipped and only the actual data is read. The organization of the parity blocks thus introduces an additional complexity which must be attended to. The parity DataBlocks are also written, as noted above, in a round robin fashion to avoid a single set of disk heads from being left idle during the read process.

[**0069**] Accordingly, therefore, referring to **FIG. 7**, there is illustrated the organization wherein parity blocks are stored in each named fragment file on each cluster member. During a failure, the blocks associated with a missing block are read to reconstruct the missing block. For example, if cluster member **2** were unavailable, and block **7** was requested, parity block **6|7** and block **6** would be read and EXCLUSIVE OR'd to generate the missing block. While it would certainly be possible to determine the location of a DataBlock using tables, in a preferred embodiment of the invention a algorithmic process is preferred and operates more quickly. In this illustrated embodiment, integer arithmetic is employed to locate data and parity blocks stored in named fragment files.

[**0070**] Referring to **FIG. 8**, there is illustrated a list of the variables used in a computer software program used to determine the location of a block. A flow chart of the program will now be described in connection with **FIG. 9A**. Initially, at **300**, the number of parity blocks is first determined by dividing the block number to be located, by the cluster size minus 1. This, in essence, provides the number of parity blocks up to and but not including the row in which the block to be found resides. The division is integer division and the remainder is discarded. Thus, PBC provides the number of parity blocks up to and including the last complete row of blocks. Next, the so-called adjusted block number is set equal to the total number of blocks which have been stored, including the parity blocks within complete rows, up to the block to be found. This is indicated at **302**.

[**0071**] The optional parity block count is determined next. To determine whether the parity block, in the row of the block to be found, is located before or after the block to be located, a quantity ABN_L, equal to the total number of blocks modulo the number of cluster members squared, is generated at **364**. The optional parity count is set to "zero" at **306**, and, if ABN_L divided by the number of clusters is less than or equal to ABN_L modulo the number of clusters at **308**, the optional parity count is set to "one" at **310**. A "zero" optional parity count indicates that the block is before the parity block in the row while a "one" optional parity count indicates that the block is after the parity block of the row. This assumes that the parity is configured as described in **FIG. 6C**.

[**0072**] The final block number is then the adjusted block number plus the value of OPC. This is indicated at **320**. Then, the cluster member can be calculated by taking the final block number modulo the number of clusters. This is indicated at **330**. The local block number within that cluster is then the final block number divided by the number of clusters. This is indicated at **340**. (Recall that the first block number is zero.)

[**0073**] If a failure has occurred, the parity block number for the binary block number must be determined. This is performed as illustrated below, referring to the flow chart of **FIG. 9B**. First, a parity block number is determined by multiplying the number of clusters by the integer portion of a division, the numerator of which is the final block number and the denominator of which is the number of clusters. This is indicated at **400**. Next, the parity block offset within the repeating pattern is determined and added to the parity block number (PBN) previously determined. This is accomplished by taking the final block number modulo the number of cluster members squared and dividing that number by the number of clusters. That value is then added to the parity block number to obtain the final value of the parity block number. This is indicated at **420**.

[**0074**] It may also be necessary to locate the DataBlocks associated with a parity block. Referring to **FIG. 9**C, this is determined by finding the row in which the parity block number (PBN) is to be found. The row is determined by taking the integer value of the division of the parity block number and the number of clusters and multiplying that value times the number of clusters. This is indicated at **430**. The parity block location is equal to the parity block number modulo the number of cluster members squared, that quantity divided by the number of cluster members. This is indicated at **440**. The following subroutine (written in C) is then employed to determine the blocks associated with the parity blocks:

[**0075**] for (i=0; i<CS; i++)

[**0076**] if (PBN_{row}+i!=PBN)

[**0077**] FBN=PBN_{row}+i

[**0078**] If a cluster processor system member fails, the reading controller for the data object implements the pre-

ceding steps to recover a missing DataBlock. The output of this operation yields the missing (failed) DataBlock.

[0079] When cluster volumes are employed, as noted above, the cluster volume master controls access to each block of the cluster volume. The computer system **10** provides a protocol to determine the cluster volume master for each volume. Each cluster volume, in the illustrated and preferred embodiment, is controlled by a different processor system member (to the extent possible) to distribute the write load across all of the cluster members. When a cluster volume master detects that a cluster processor system member has failed, it instructs the remaining cluster members to advance the cluster volume incarnation (in the HomeBlock) and to clear the cluster volume "dirty bit" in the cluster volume state. The next write operation to the cluster volume will cause the dirty bit to be set, indicating that the failed cluster member must execute a rebuilding process before it can rejoin the cluster volume. If the implemented cluster protocol prevents a write to the cluster volume while a cluster member is in a failed state, the failed member may not need to rebuild.

[0080] The cluster volume does not have to be write locked during the rebuilding operation. Write operations to already rebuilt DataBlocks simply update the DataBlock. Write operations to DataBlocks not yet rebuilt can simply be ignored and they will be reconstructed at a later time. Special care however is required when handling a write operation for the current DataBlock being reconstructed. In this case, reconstruction should be completed and the write operation should be executed after, and only after, the reconstructed data is written.

[0081] On the other hand, when the system operates upon data objects, the system can continue to operate, modifying data objects as necessary. When the failed member has been restored, it executes a rebuilding process only for the data objects which have been changed. The remaining objects are not affected. In this way, even during a failure, writing can continue to be implemented and the failure becomes transparent to the user or consumer. Importantly, also, since the rebuilding occurs object by object, it can be done at a more leisurely pace since not all of a cluster volume will be adversely affected by writing any of file within the volume.

[0082] Once a cluster processor system member has failed, the failure of any other cluster processor system member will cause the cluster volume to become unreadable in this illustrated embodiment. Only after the failed cluster member has been reconstructed, therefore, can another failure be handled for this embodiment of the invention. However, as noted above, in other embodiments of the invention, two or even more failures could be handled, however, more complex, and hence more lengthy reconstruction and encryption processes would need to be employed.

[0083] When expansion of a cluster is required, for example when cluster volumes become full or when the capacity of the cluster needs to be increased, the method and apparatus of the invention provide a process for increasing the storage capacity "on line". For example, to add storage to a cluster, disk drives are added to each cluster member (that is, each processor system **12**) and a new cluster volume or new data objects can be created. This is a simple operation and the existing cluster volumes or data objects remain fault tolerant during the upgrade. Cluster members may have to be rebooted however in order to "see" (that is, recognize) the new disk drives.

[0084] However, the method and apparatus of the invention can further provide for the addition of a new cluster processor system (and its associated storage) during operation. This is a much more complex operation and can proceed as follows.

[0085] The new cluster processor system member is inserted into the cluster by networking the new cluster member to each of the original cluster processor system members **12** as illustrated in **FIG. 1**. Each cluster volume is then "write locked" so that writing to any portion of the volume is not allowed. Each cluster volume is reformatted by initiating a change in the current cluster volume format to a new format. This operation is in essence a translation from an n member cluster volume to an n+1 member cluster volume. Each DataBlock is read and rewritten, new ParityBlocks are generated, and the progress is check-pointed in case a system failure occurs during the reformatting operation. The size of any one cluster volume is not increased by this operation; rather, each local cluster volume size is decreased and the residual disk space from the operation can be configured into yet additional cluster volume(s). Finally, as cluster volume reformatting is completed, the "write lock" is removed from the cluster volume. Referring to **FIG. 10** in a typical system, the various sizes of the cluster volume within each of the cluster processor system members is illustrated both before and after reformatting.

[0086] When a new cluster member is added to a system wherein the format of the stored data is in data objects, the existing data objects need to be refragmented from n named fragment files to n+1 named fragment files. This operation can occur as a background activity allowing access to the original data object until the new named fragment files have been created. Once the new fragment files exist the old fragment files can be deleted. This process takes place, therefore, at "leisure" and at no time is the named fragment unavailable for use.

[0087] Referring now to FIGS. **11-13**, there is illustrated a switching system useful in conjunction with the structure of **FIG. 1**, and in which a fault tolerant operation provides resiliency and robustness, as well as relatively low cost for the described system.

[0088] In a most general sense, in a video insertion system, to which the invention is particularly applicable, it is desirable to have a 2n×n cross-bar system wherein any one of 2n inputs can be placed on any of n outputs. Such a cross-bar system **100**, referring to **FIG. 11**, might have, for example, n (interruptible) network feeds **102** and n advertising or other interrupting feeds **104** which are to be used to selectively replace the network feeds. Each of the n output selections **106** represents one of the network feeds which is switchably replaced, from time to time by one of the interrupting feeds **104**. The n outputs connect to output channels for transmission over, for example, a cable network.

[0089] A simpler approach, though less flexible and hence to some extent less desirable, is illustrated in **FIG. 12** wherein the 2n×n switch **108** is replaced by n two by one switches **110**. In this configuration, a network input **112** is

provided to each 2×1 switch **110** and an interrupting or other input **114** is also provided to each 2×1 switch **110**. This system works well provided that none of the interrupting feeds is lost. (The interrupting feeds over lines **114** are typically generated by a processor system **12** and it is implicitly assumed that the network feeds are reliable and always present.) If an interrupting feed is lost, by failure of a processor system **12**, then the output over lines **116** will be incorrect since there is no provision for an alternate source of the interrupting feed. In summary then, in a typical system, the network feeds over lines **112** are considered reliable and available at all times. Thus, it is only the interrupting feed over lines **114** which may fail. The interrupting feeds are typically provided by, for example, a processor system **12**, and thus if the processor system **12** fails, there is no flexibility or robustness in the switches **110**, as configured in **FIG. 12**, (or even the crossbar switch provided in **FIG. 11**, since the failed source cannot be replaced) to recover.

[**0090**] In accordance with the invention, however, a "spare" switched system is provided which can be used to replace a failed interrupting feed in the **FIG. 12** embodiment, in a robust and flexible manner. Thus, referring to **FIG. 13**, the invention provides a method and apparatus for compensating for a failed insertion feed from a system processor by providing a complete spare system. The spare system interconnects with the active systems in a ring structure, to create a robust and fault-tolerant signal, for example video, delivery system. **FIG. 13** illustrates a limited video delivery system in which there are two network feeds and two interrupting feeds generated by local processor systems. In other embodiments more network interruptible and interrupting feeds can be used and the system can be accordingly scaled. The network feeds over lines **130** and **132**, designated NET0 and NET1 are input to respective video insertion and feed systems **134** and **136**, respectively. Each video system has a processor system **138**, **140** (these can be the same as processor systems **12** described in connection with **FIG. 1**) and a switched system **142**, **144** which is able to place, in a controlled manner, as described below, ones of its four inputs, on its output lines.

[**0091**] The switched systems provide a network feed output signal on lines **146**, **148**, designated as OUT0 and OUT1, respectively. In the illustrated embodiment, a spare video insertion and feed system **150**, which has a processor system **152** and a switched system **154** mirroring the interconnection of the processor system and switched system of video systems **134** and **136**, provides a fault tolerance under the conditions described below.

[**0092**] The processor systems **138**, **140**, and **152** are interconnected by point-to-point communications and operate in a manner, with respect to each other, which is identical to that described for the larger computer system illustrated in connection with **FIG. 1**. Thus, any processor system **138**, **140**, **152** has access to the video stored by the other processor systems, and the video is stored at the processor level preferably according to a RAID-5 protocol. In the illustrated embodiment, the video objects are stored in a processor's local disk array also in accordance with a RAID-5 protocol. Each processor system receives an output feed (typically a network feed) from its associated switched system over lines **160**, **162**, and **164** and provides n interrupting output in response to the input feed, for example an advertisement, to its associated switched system over lines **166**, **168**, and **170**. The signals carried over lines **160**, **162**, **164**, **166**, **168** and **170** are video signals as will be described in more detail below.

[**0093**] The video systems **134**, **136** and **150** are interconnected in a ring structure through their switched systems. Thus, each switched system provides an output, labelled **172**, **174**, **176**, which becomes an input to a neighboring video system in the ring configuration; and the network input line of video systems **134** and **136** is connected as another input to that other neighboring switched system on the ring which, as described above, received an input signal from its other neighboring system. Thus, the NET1 input is provided to both switched system **144** and switched system **142** while the NET0 input is provided to switched system **142** and switched system **154** outputs **173**, **174**, **176** are connected respectively as inputs to systems **144**, **154**, and **142**. Each processor system also provides, either from the processor itself or through an on-board controller **12***a*, typically controlled by the processor CPU, two switch control signals as described in more detail below.

[**0094**] Referring now to **FIG. 14**, there is illustrated a preferred embodiment of the video system which allows, referring to the orientation of **FIG. 13**, the video systems to switch in a downward round robin direction (meaning that spare video system **150** can replace the operation of video system **134** should video system **134** or video system **136** fail). Thus, as noted above, should video system **134** fail due to a processor failure (it is assumed that the switched system will not fail), the spare video system **150** replaces it and provides the correct interrupting feed to the switched system **142**. Should the video system **136** fail due to a processor system failure, in a chain reaction, video system **134** will act to provide the correct interrupting feed to switched system **144** and video system **150** will thereafter operate to provide the correct interrupting feed to the switched system **142** of video system **134**. This "downward" movement can be extended to a larger operating system having, for example, nine network feeds in which case there would be the nine network video systems like circuitry **134** plus a spare video system corresponding to system **150**. In other embodiments, the ring can be reconfigured to move upward or as will be hereinafter described, the switched systems can be structured so that the motion of the ring can be directed in either the upward or downward direction as viewed in **FIG. 13**. In this latter configuration, the interconnecting structure is more complex as will be described hereinafter.

[**0095**] Referring now to **FIG. 14**, in the illustrated embodiment, each switched system of **FIG. 13** has two multi-input single output controlled switch elements **190**, **192**. In **FIG. 14**, the reference numbers correspond to the reference numbers of **FIG. 13** and, in particular, the illustrated video system **134**.

[**0096**] In normal operation, the network feed over line **130** is directed to the default or number 1 position of each of switches **190** and **192**. Thus, the signal on line **130** is fed through switch **192** and is output over line **146**. Simultaneously that signal is also fed through switch **190** and is output over line **160** to the processor system **138**. The processor system, analyzes the input signal and provides, over line **166**, at the correct time, an insert which interrupts the signal over line **130** and replaces it. This is effected by

switch element **192** which changes the signal over line **146** from that connected at its number 1 position to that connected at its number 3 position. (The controller (and control lines) for switching the outputs of switches **190** and **192** have been omitted for clarity of presentation. However, each processor has two (or three) control lines for controlling the switching functions of its associated switch and at least one neighboring switch, depending upon whether the ring will fail up, fail down, or optionally fail in either direction. The processors are further, preferably, interconnected through a further communications line, such as an Ethernet bus, so that each processor is aware of the status, and actions, taken by each other processor. This allows an orderly compensation for a failed processor as described herein.) When the insert (interrupting) video over line **166** ends, the switch element **192** returns to its "default" number 1 position. Thus, in normal operation, switch **192**, under the control of a controller, switches back and forth as processor system **138** generates the correct interrupting video in accordance with the processor interconnections illustrated in **FIG. 13** (as an example of a simpler version of the more complex system illustrated in **FIG. 1**).

[**0097**] If the processor **140**, referring now to **FIG. 13**, of video system **136** were to fail, the first step of the fault-tolerant system would provide that the video system of **134** would be modified so that it generates the necessary insert video signals for switched system **144** and provides those insert signals to switched system **144** over line **172**. Accordingly, in this failure mode of operation, referring now also to **FIG. 14**, the network feed for video system **136**, that is NET1 over line **132**, connects to video system **134**, is switched by switch element **190** of video system **134** and is placed on the output of switch element **190**, that is on line **160**. The processor system **138** in response to this new video generates an insert video over line **166** which is also available over line **172**. That insert video signal is thus provided to the switched system **144** and its switch, which corresponds to switch **192** of switched system **142**, then connects the input on its corresponding line input **4** to line **146**, and outputs it over line **146** as "OUT1".

[**0098**] In this replacement process, however, video system **134** has lost use of its processor system **138**. It thus turns to the spare video system **150** and its processor system **152** which, in combination with switched system **154** provides the appropriate insert video over line **176** (along with the correct insert control signals). Switch **192**, thus, at the correct time, switches to place the insert video available over line **176** at its output, over line **146**, to become OUT0. After the insert ends, switch **192** returns to its default value to place NET0, its network input, at switch position **1**, as its output. In this manner, the video systems **134**, **136**, and **150**"cover" redundantly for each other so that a failure of one processor system will not adversely affect operation of the delivery system.

[**0099**] This ringed redundancy operates not only because the switched systems **142**, **144**, **154** are interconnected in a ring structure, but also because the processor systems themselves are interconnected and operate in a redundant fashion in accordance with the embodiment of **FIG. 1**. Thus, each processor system is capable of fully acquiring the necessary insert video objects stored redundantly in the collective system memory, for any of the network feeds.

[**0100**] The embodiment illustrated in **FIGS. 13 and 14** fails in an "up direction". By simply modifying the "neighboring connections" so that, for example, video system **136** provides its network input not to video system **134** but to spare video system **150** and accordingly provides its output over line **174** not to spare video system **150** but to video system **134**, the system would fail in the reverse direction.

[**0101**] In a second particular embodiment of the video distribution system, there is provided a more complex video switched system which is capable of shifting "in either direction" in response to a failed system. By this is meant that the ring connection, depending upon the proximity of the failure to the spare, will fail either up or down to reduce the time delay for correcting the failed situation. It will be seen, however, referring to **FIGS. 13 and 14**, that it is not necessary to use this system, but that single direction failure system will work substantially as well in most failure situations.

[**0102**] Accordingly, referring to **FIGS. 15 and 16**, there is illustrated a **3** element ringed system which can fail in either direction. This system operates in a manner corresponding to that of the video delivery system illustrated in connection with **FIGS. 13 and 14**, and accordingly, its detailed operation need not be described. The ringed system has two video systems **200** and **202**, each of which has a video processor **204**, **206**, and a switched system **208**, **210** respectively. In addition there is a spare video system **212** having a video processor **214** and a switched system **216**. (The point to point, two way, interconnecting channels between the processors **204**, **206**, **214**, and the processor controllers, have not been shown for purposes of more clearly illustrating the other connections in the figure.) In operation, if a failure occurs, the failed system can be replaced by a shift upward by one closest "lower" neighbor or a shift downward by its other closest neighbor. The direction of the shift will depend primarily upon where the failure occurred in the ring. Accordingly, the fewer shifts needed to achieve full operation of the video distribution system will determine the direction of shift along the ring.

[**0103**] Independent of the direction of the shift, and referring to **FIG. 15**, the processor system **214** of the spare video processor **212** will, in this illustrated embodiment, replace the processor system which has failed. The switch elements of the switched systems for the spare system and the failed system will reconfigure the switches to provide the network input for-the failed system to the spare system, which will provide that network input to its processor system. The video insert output of the processor system will then be routed through the spare switched system to the switch system corresponding to the failed processor for delivery, at the correct time, to its output. When more than two input network feeds are used, a larger element ringed system can be employed, and, as with the embodiment of FIGS. **13**, and **14**, the video systems will chain with a first nearest neighbor replacing the failed processor and a next second nearest neighbor then acting to replace the processor of the first nearest neighbor, etc., until the spare video system is used.

[**0104**] Referring to **FIG. 16, a** typical switched system has four multi-input switches **180**, **182**, **184**, and **186** connected as illustrated in **FIG. 16**, in one particular embodiment of the invention, in order to enable the switching described hereinabove to take place. Referring to **FIG. 17**,

there is provided a truth table indicating, for each switch of **FIG. 16**, its status depending upon whether the switch is operating in a "normal" mode, in a failed-up mode, or a failed-down mode. The number within the matrix identifies, for a switch of **FIG. 16**, which input to direct to the output. (An "x" indicates a "don't care" condition.) For switch **4** (reference number **186**) the selected input depends upon whether the switch is expected to place the interruptible input ("NO INSERT") or the interrupting input ("INSERT") at its output port. As with the circuitry of **FIGS. 13 and 14**, the processor controller and the control lines to each of the switches **180**, **182**, **184**, and **186** are not detailed in order to be able to understand better the operation of the system.

[**0105**] Additions, subtractions, and other modifications of the preferred embodiments of the invention will be apparent to those practiced in the art and are within the scope of following claims.

What is claimed is:

1. A method for redundantly storing data in a distributed computer system having at least three processor systems, each processor system comprising at least one central processing unit and at least one mass storage sub-system, comprising the steps of:

interconnecting each one of said processor systems in a point-to-point two way channel interconnection with each other one of said processor systems; and

storing data input at any one of said processor systems according to a distributed, redundant storage process whereby data is stored at each of said processor systems and some of a redundant representation of the data is stored at each of said processors.

2. The method of claim 1 wherein said storing step comprises the step of storing data across said processor systems according to a RAID-5 process.

3. The method of claim 2 further comprising the step of storing data at each processor system according to a RAID-5 process.

4. The method of claim 1 further comprising the step of

reading data from said computer system, in the absence of a failure of any of said processor systems, from each of said processor systems over respective ones of said data channel interconnections, whereby said reading step establishes a load balance across said processor systems.

5. The method of claim 4 further comprising the steps of

reading data from said computer system, in the presence of a failure of one of said processor systems, said reading step, in the presence of failure, comprising the steps of

reading data from each non-failed processor system storing said data, and

reading redundant data from said non-failed processor systems in place of said data stored at said failed processor system, and

recreating said data stored at said failed processor system using said redundant data and data read from said non-failed processor systems.

6. The method of claim 5 wherein said data input storing step comprises the step of storing said input data according to a RAID-5 process.

7. The method of claim 6 wherein each said processor system stores data on its associated mass storage sub-system according to a RAID-5 process.

8. The method of claim 5 further comprising the step of

preventing, during the presence of the failure of any of said processing systems, the writing of any data in the mass storage sub-system of any of said processor systems.

9. The method of claim 1 wherein said storing step comprises the step of

designating one processor system to effect all write functions for said computer system.

10. The method of claim 1 wherein said storing step comprises the steps of

one processor system allocating files for each data input, and

enabling all processor systems to write input data to its associated allocated files.

11. The method of claim 1 wherein said storing step comprises the step of

arbitrating write operations among said processor systems using a distributed lock manager.

12. The method of claim 1 wherein said data input storing step comprises the step of

storing said input data as named fragment files in each said processor system.

13. The method of claim 12 wherein said data input storing step further comprises the step of

storing said input data according to a RAID-5 process.

14. The method of claim 13 further comprising the steps of

adding a processor system to said distributed computer system, and

reconstructing said named fragment files to extend over all the processor systems while said distributed computer system continues to actively deliver stored data as an output stream.

15. The method of claim 14 further comprising the step of

reconstructing a failed processor system by reconstructing only data objects which were written while the processor system was in a failed state.

16. A distributed computer system having at least three processor systems for redundantly storing data, each processor system comprising at least one central processing unit and at least one mass storage sub-system, comprising

interconnecting channels providing a point-to-point two way channel interconnection from each one of said processor systems to each other one of said processor systems; and

a data storage controller at each processor system, said controller storing data input at any one of said processor systems according to a distributed, redundant storage process whereby data is stored at each of said processor systems and some of a redundant representation of the data is stored at each of said processors.

17. The apparatus of claim 16 wherein said storage controllers store data across said processor systems according to a RAID-5 process.

18. The apparatus of claim 17 further wherein each said storage controller stores data at each processor system according to a RAID-5 process.

19. The apparatus of claim 16 further comprising

said controllers reading data from said computer system, in the absence of a failure of any of said processor systems, from each of said processor systems over respective ones of said interconnecting channels, whereby said controllers establish a load balance across said processor systems.

20. The apparatus of claim 19 further comprising

said controllers reading data from said computer system, in the presence of a failure of one of said processor systems, said controllers reading data from each non-failed processor system storing said data, and reading redundant data from said non-failed processor systems in place of said data stored at said failed processor system, and

the requesting processor system recreating said data stored at said failed processor system using said read data and said redundant data.

21. The apparatus of claim 18 wherein said data storage controllers store said input data among the processors according to a RAID-5 process.

22. The apparatus of claim 21 wherein each said processor system controller stores data on its associated mass storage according to a RAID-5 process.

23. The apparatus of claim 20 further wherein said storage controllers prevent, during the presence of a failure of any of said processing systems, writing of any data to the mass storage sub-system of any of said processor systems.

24. The apparatus of claim 16 wherein

one processor system is designated to effect all write functions for said computer system.

25. The apparatus of claim 16 wherein

one processor system allocates files for each data input, and

each processor system is enabled to write input data to its associated allocated files.

26. The apparatus of claim 16 wherein said controllers arbitrate write operations among said processor systems using a distributed lock.

27. The apparatus of claim 16 further wherein

said storage controllers store said input data as named fragment files in said distributed computer system.

28. The apparatus of claim 27 wherein said storage controllers store said data objects as named fragment files across said processor systems according to a RAID-5 process.

29. A redundant switch having n interruptible inputs, n interrupting inputs and n outputs comprising

(n+1) switched systems, each switched system having at least two control inputs, a first input, a second input, a third input, and a fourth input, and a first output and a second output, each switched system being connected at its second output to an interrupting signal generator, an interrupting output of said associated signal generator being connected to the second input of said connected switched system,

said switched systems being interconnected in a ring structure so that each switched system further has an interruptible input signal connected to the first input, the second input of a first neighbor switched system connected to the third input, and the interruptible input from the other neighbor switched system on the ring connected to the fourth input,

each said switched system having switching circuitry responsive to said control inputs for switching any of its inputs to at least its first output, and for connecting either of its first and fourth inputs to its second output, and

a control system providing said control inputs of said switched system.

30. A redundant switch having n interruptible inputs, n interrupting inputs and n outputs comprising

(n+1) switched systems, each switched system having at least two control inputs, four signal inputs, and two signal outputs, each switched system being connected at one of its outputs to an associated interrupting signal generator, and an interrupting output of said associated signal generator being connected to an input of said connected switched system,

said switched systems being interconnected in a ring structure so that each switched system is connected to a first and a second neighbor switched system,

each said switched system having switching circuitry responsive to said control inputs for selectively switching its inputs to its outputs, and

a control system for providing said control inputs to said switched systems to enable said switched systems to effectively rotate signal switching functions one position in either direction around the ring whereby a failed signal generator can be bypassed and the signals on said n first outputs continue uninterrupted.

31. The redundant switch of claim 30 wherein said control system can effectively rotate switching signal functions in either direction around the ring.

32. A distributed data delivery system comprising at least three processor systems for redundantly storing data, each processor system comprising at least one central processing unit and at least one mass storage sub-system,

interconnecting data channels providing a point-to-point two way channel interconnection from each one of said processor systems to each other one of said processor systems,

a data storage controller at each processor system, said controller storing data input at any one of said processor systems according to a distributed, redundant storage process whereby data is stored at each of said processor systems and some of a redundant representation of the data is stored at each of said processors,

a switching circuit having n interruptible input ports, at least n+1 interrupting input ports, and at least n+1 output ports, said n interruptible input ports being connected to a respective primary processor system, each said primary processor system having an output connected to a respective interrupting input port, and

each said processor system connected to two of said switching circuits and able to selectively interrupt one of the n interruptible input ports with the signal information available from a respective one of said processor systems, and

said processor systems, in the event of a failure at one processor system, using a previously unused one of said processor systems, and causing said switching circuit to connect the interruptible input port of the failed processor system to a different processor system, and to replace the output of said failed processor system with the output of said different processor system.

33. A distributed data delivery system comprising

at least three processor systems for redundantly storing data, each processor system comprising at least one central processing unit and at least one mass storage sub-system,

interconnecting data channels providing a point-to-point two way channel interconnection from each one of said processor systems to each other one of said processor systems,

a data storage controller at each processor system, said controller storing data input at any one of said processor systems according to a distributed, redundant storage process whereby data is stored at each of said processor systems and some of a redundant representation of the data is stored at each of said processors,

(n+1) switched systems, each switched system having at least two control inputs, a first input, a second input, a third input, and a fourth input, and a first output and a second output, each switched system being connected at its second output to a processor system, an interrupting output of said associated processor system being connected to the second input of said connected switched system,

said switched systems being interconnected in a ring structure so that each switched system further has an interruptible input signal connected to the first input, the second input of a first neighbor switched system connected to the third input, and the interruptible input from the other neighbor switched system on the ring connected to the fourth input,

each said switched system having switching circuitry responsive to switch control signals at said control inputs for switching any of its inputs to at least its first output, and for connecting either of its first and fourth inputs to its second output, and

said processor systems for providing said switch control input signals to said switched systems.

34. A distributed data delivery system comprising

at least three processor systems for redundantly storing data, each processor system comprising at least one central processing unit and at least one mass storage sub-system,

interconnecting data channels providing a point-to-point two way channel interconnection from each one of said processor systems to each other one of said processor systems,

a data storage controller at each processor system, said controller storing data input at any one of said processor systems according to a distributed, redundant storage process whereby data is stored at each of said processor systems and some of a redundant representation of the data is stored at each of said processors,

(n+1) switched systems, each switched system having at least two control inputs, four signal inputs, and two signal outputs, each switched system being connected at one of its outputs to an associated processor system, and an interrupting output of said associated processor system being connected to an input of said connected switched system,

said switched systems being interconnected in a ring structure so that each switched system is connected to a first and a second neighbor switched system,

each said switched system having switching circuitry responsive to switch control signals at said control inputs for selectively switching its inputs to its outputs, and

said processors providing said switch control input signals to said switched systems to enable said switched systems to effectively rotate signal switching functions one position around the ring whereby a failed signal processor system can be bypassed and the signals on said n first outputs continue unimpaired.

35. The distributed data delivery system of claim 24 further wherein said processors effectively rotate signal switching functions in either direction around the ring.

36. A method for redundantly storing data in a distributed computer system having at least two processor systems, each processor system comprising at least one central processing unit and at least one mass storage sub-system, comprising the steps of:

interconnecting each one of said processor systems in a point-to-point two way channel internconnection with each other one of said processor systems;

storing data input at any one of said processor systems according to a distributed, redundant storage process whereby data is stored at each of said processor systems and some of a redundant representation of the data is stored at each of said processors; and

reading data from said computer system, in the absence of a failure of any of said processor systems, from each of said processor systems over repsective ones of said data channel interconnections, whereby said reading step establishes a load balance across said processor systems.

37. The method of claim 36 further comprising the step of storing data at each processor system according to a RAID-5 process.

38. A distributed computer system having at least two processor systems for redundantly storing data, each processor system comprising at least one central processing unit and at least one mass storage sub-system, comprising

interconnecting channels providing a point-to-point two way channel interconnection from each one of said processor systems to each other one of said processor systems;

a data storage controller at each processor system, said controller storing data input at any one of said processor systems according to a distributed, redundant storage process whereby data is stored at each of said processor systems and some of a redundant representation of the data is stored at each of said processors; and

said controllers reading data from said computer system, in the absence of a failure of any of said processor systems, from each of said processor systems over respective ones of said interconnecting channels, whereby said controllers establish a load balance across said processor systems.

39. The apparatus of claim 38 further wherein each said storage controller stores data at each processor system according to a RAID-5 process.

* * * * *