US006988219B2

(12) **United States Patent**
Hitz et al.

(10) **Patent No.: US 6,988,219 B2**
(45) **Date of Patent: *Jan. 17, 2006**

(54) **PROVIDING PARITY IN A RAID SUB-SYSTEM USING NON-VOLATILE MEMORY**

(75) Inventors: **David Hitz**, Los Altos, CA (US); **Michael Malcolm**, Los Alto, CA (US); **James Lau**, Los Altos Hills, CA (US); **Byron Rakitzis**, Burlingame, CA (US)

(73) Assignee: **Network Appliance, Inc.**, Sunnyvale, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 509 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **10/233,311**

(22) Filed: **Aug. 28, 2002**

(65) **Prior Publication Data**

US 2003/0037281 A1 Feb. 20, 2003

Related U.S. Application Data

(63) Continuation of application No. 09/345,246, filed on Jun. 30, 1999, now Pat. No. 6,480,969, which is a continuation of application No. 08/471,218, filed on Jun. 5, 1995, now Pat. No. 5,948,110, which is a continuation of application No. 08/071,798, filed on Jun. 4, 1993, now abandoned, which is a continuation of application No. PCT/US94/06321, filed on Jun. 2, 1994.

(51) **Int. Cl.**
G06F 11/00 (2006.01)

(52) **U.S. Cl.** **714/6**; 714/711

(58) **Field of Classification Search** 714/6, 714/5, 7, 8, 710, 711; 711/114, 113
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,761,785 A 8/1988 Clark et al.

(Continued)

FOREIGN PATENT DOCUMENTS

EP 0 492 808 A2 7/1992

(Continued)

OTHER PUBLICATIONS

Menon et al. "The Architecture of a Fault-Tolerant Cached RAID Controller," Proceedings of the 20th Annual International Symposium on Computer Architecture, May 16-19, 1993, pp. 76-86, IEEE Computer Society, Los Alamitos, CA.

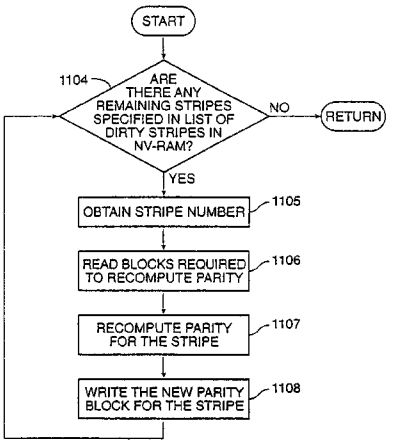
(Continued)

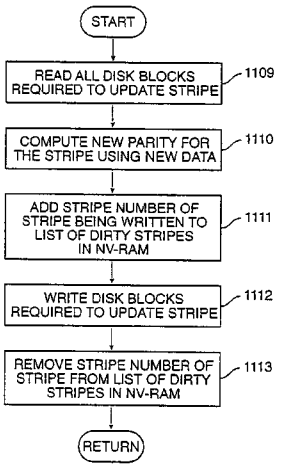
Primary Examiner—Dieu-Minh Le
(74) *Attorney, Agent, or Firm*—Swernofsky Law Group PC

(57) **ABSTRACT**

The present invention is a method for providing error correction for an array of disks using non-volatile random access memory (NV-RAM). Non-volatile RAM is used to increase the speed of RAID recovery from a disk error(s). This is accomplished by keeping a list of all disk blocks for which the parity is possibly inconsistent. Such a list of disk blocks is much smaller than the total number of parity blocks in the RAID subsystem. The total number of parity blocks in the RAID subsystem is typically in the range of hundreds of thousands of parity blocks. Knowledge of the number of parity blocks that are possibly inconsistent makes it possible to fix only those few blocks, identified in the list, in a significantly smaller amount of time than is possible in the prior art. The technique for safely writing to a RAID array with a broken disk is complicated. In this technique, data that can become corrupted is copied into NV-RAM before the potentially corrupting operation is performed.

25 Claims, 8 Drawing Sheets





U.S. PATENT DOCUMENTS

5,088,081	A		2/1992	Farr
5,134,619	A		7/1992	Henson et al.
5,146,588	A		9/1992	Crater et al.
5,195,100	A		3/1993	Katz et al.
5,208,813	A		5/1993	Stallmo
5,235,601	A		8/1993	Stallmo et al.
5,239,640	A		8/1993	Froemke et al.
5,255,270	A		10/1993	Yanai et al.
5,274,799	A		12/1993	Brant et al.
5,305,326	A		4/1994	Solomon et al.
5,315,602	A		5/1994	Noya et al.
5,335,235	A		8/1994	Arnott
5,390,327	A	*	2/1995	Lubbers et al. 714/7
5,452,444	A		9/1995	Solomon et al.
5,488,731	A	*	1/1996	Mendelsohn 711/114
5,550,975	A		8/1996	Ichinomiya et al.
5,948,110	A		9/1999	Hitz et al.

FOREIGN PATENT DOCUMENTS

EP	0 497 067	A1	8/1992
EP	0 559 488	A2	9/1993
EP	0 569 313	A2	11/1993
EP	0 829 956	A2	12/1994
EP	0 747 829	A1	12/1996
EP	0 756 235	A1	1/1997
EP	0 462 917	B1	9/1999
EP	1 031 928	A2	8/2000
EP	1 031 928	A3	8/2000
JP	4-278641	A	10/1992
WO	WO 91/13405	A1	9/1991
WO	WO 94/29795	A1	12/1994
WO	WO 98/21658	A1	5/1998

OTHER PUBLICATIONS

Gray et al. "Parity Striping of Disc Arrays: Low-Cost Reliable Storage with Acceptable Throughput." Proceedings of the International Conference on Very Large Data Bases, 16th International Conference, Aug. 13-16, 1990, pp. 148-161, Brisbane, Australia.

IBM Corporation, "Mapping the VM Text Files to the Aix Text Files."IBM Technical Discosure Bulletin, Jul. 1990, p. 341, vol. 33, No. 2.

Nass, Richard."Connect Disk Arrays to EISA or PCI Syses. "Electronic Design, Nov. 11, 1993, pp. 152-154, vol. 41, No. 23.

* cited by examiner

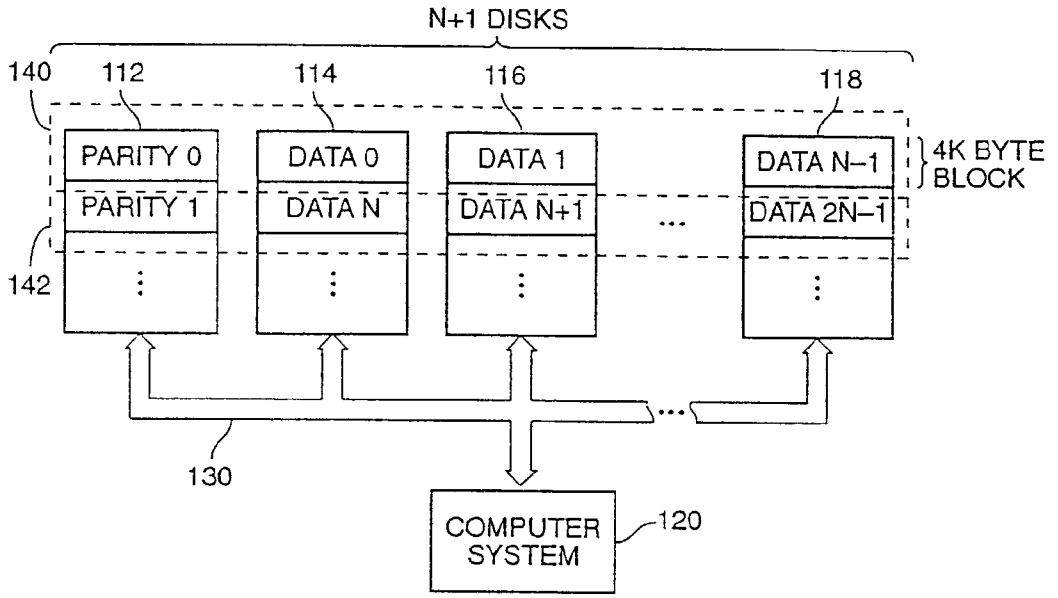


FIG. 1
(PRIOR ART)

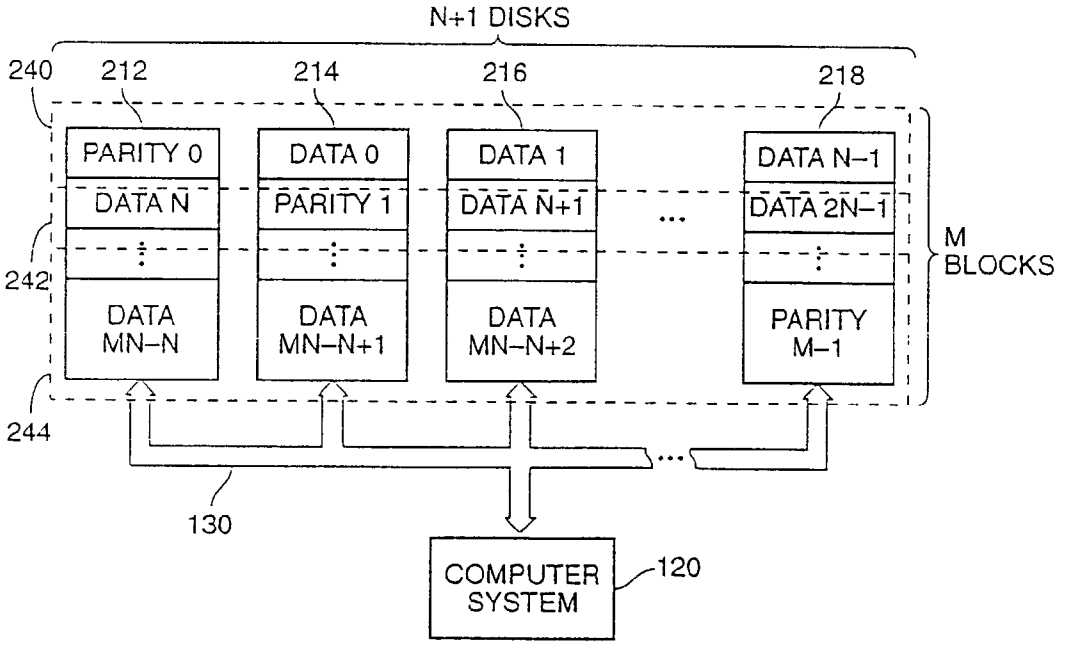


FIG. 2
(PRIOR ART)

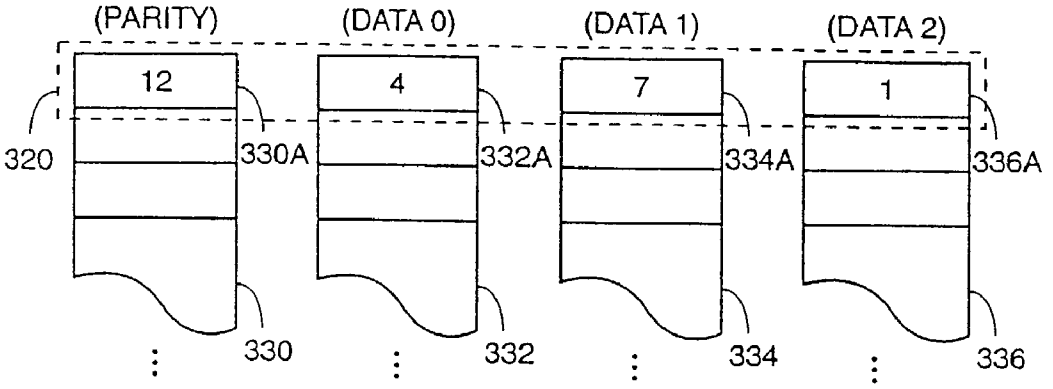


FIG. 3A
(PRIOR ART)

	PARITY	DATA 0	DATA 1	DATA2
0				
t_A	12	4	7	1
t_B	12	4	?	1
t_C	12	4	(7)	1

TIME

FIG. 3B
(PRIOR ART)

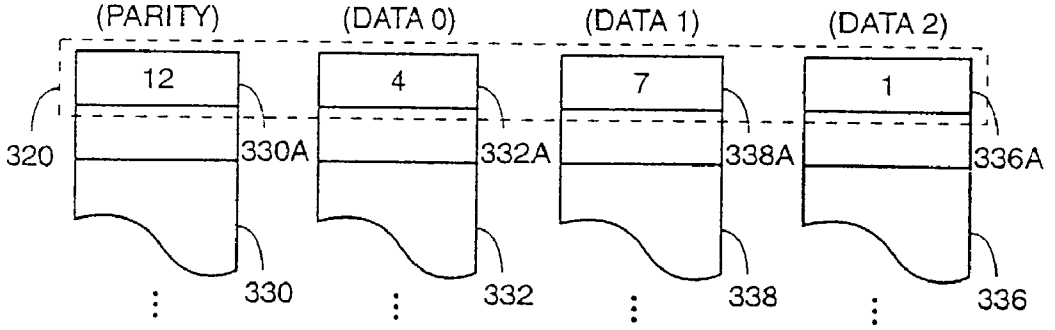


FIG. 3C
(PRIOR ART)

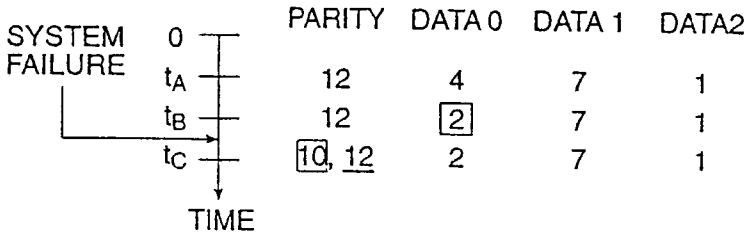


FIG. 4A
(PRIOR ART)

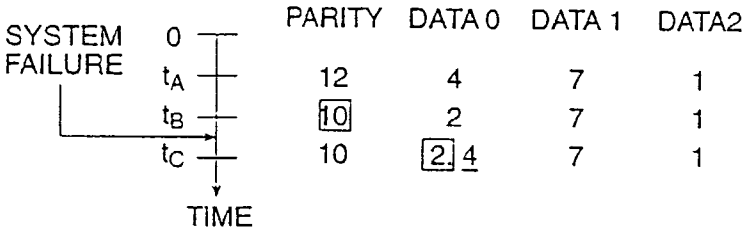


FIG. 4B
(PRIOR ART)

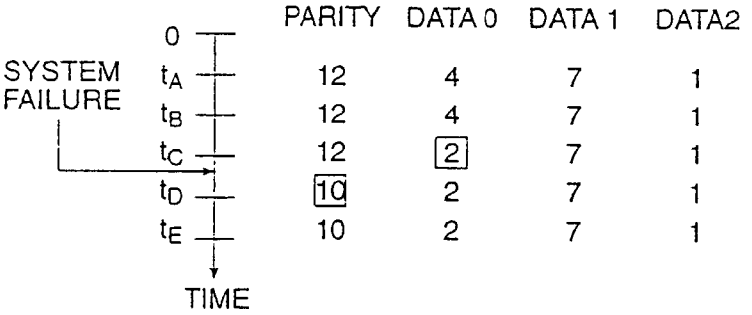


FIG. 5

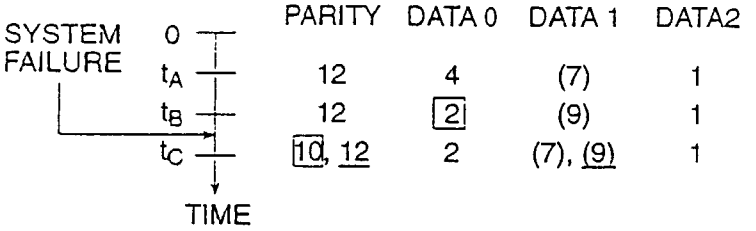


FIG. 6
(PRIOR ART)

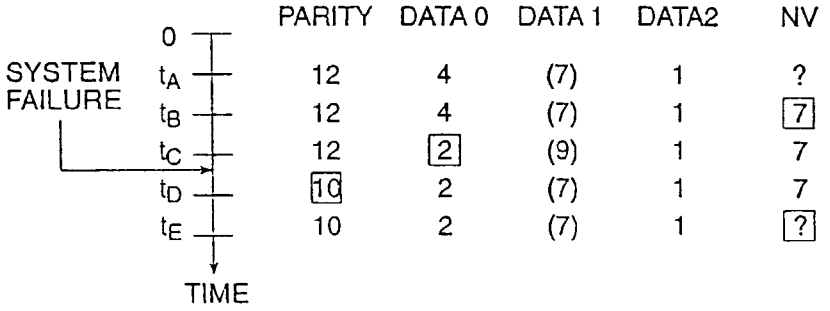


FIG. 7A

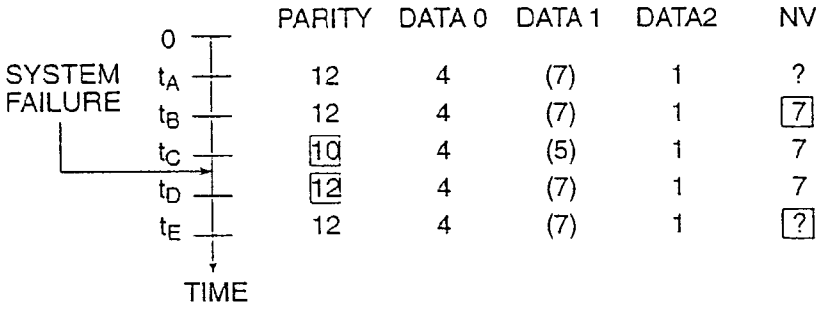


FIG. 7B

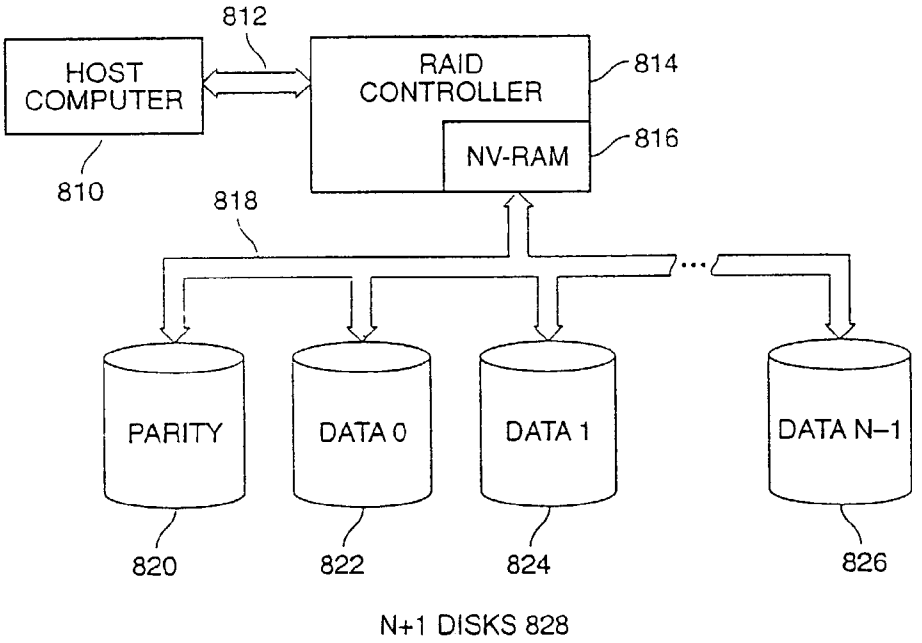
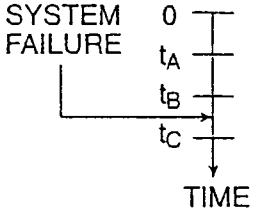
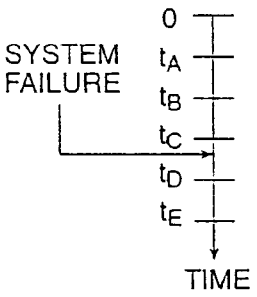


FIG. 8



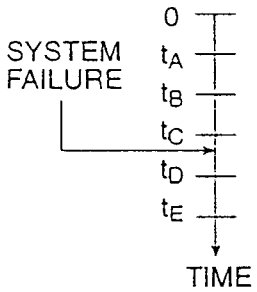
PARITY	DATA 0	DATA 1	DATA2
12	4	7	1
12	$\boxed{2}$	7	1
12	2	(9)	1

FIG. 9



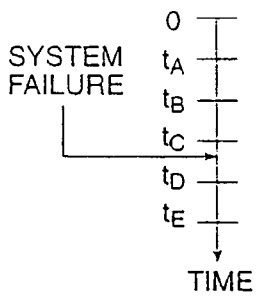
PARITY	DATA 0	DATA 1	DATA2	NV-RAM PARITY	NV-RAM DATA 0
12	4	7	1	?	?
12	4	7	1	12	4
12	$\boxed{2}$	7	1	12	4
($\boxed{10}$)	2	7	1	12	4
(10)	2	7	1	?	?

FIG. 10A



PARITY	DATA 0	DATA 1	DATA2	NV-RAM PARITY	NV-RAM DATA 0
12	4	7	1	?	?
12	4	7	1	12	4
12	$\boxed{2}$	7	1	12	4
$\boxed{12}$	($\boxed{4}$)	7	1	12	4
12	4	7	1	?	?

FIG. 10B



PARITY	DATA 0	DATA 1	DATA2	NV-RAM PARITY	NV-RAM DATA 0
12	4	7	1	?	?
12	4	7	1	12	4
12	$\boxed{2}$	7	1	12	4
$\boxed{12}$	$\boxed{4}$	($\boxed{7}$)	1	12	4
12	4	(7)	1	?	?

FIG. 10C

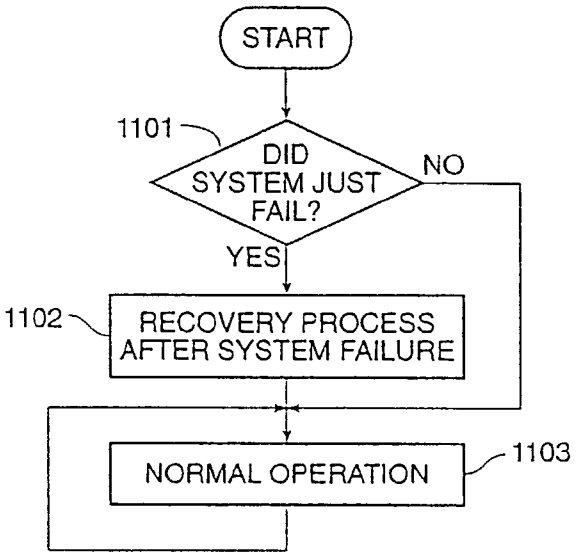


FIG. 11A

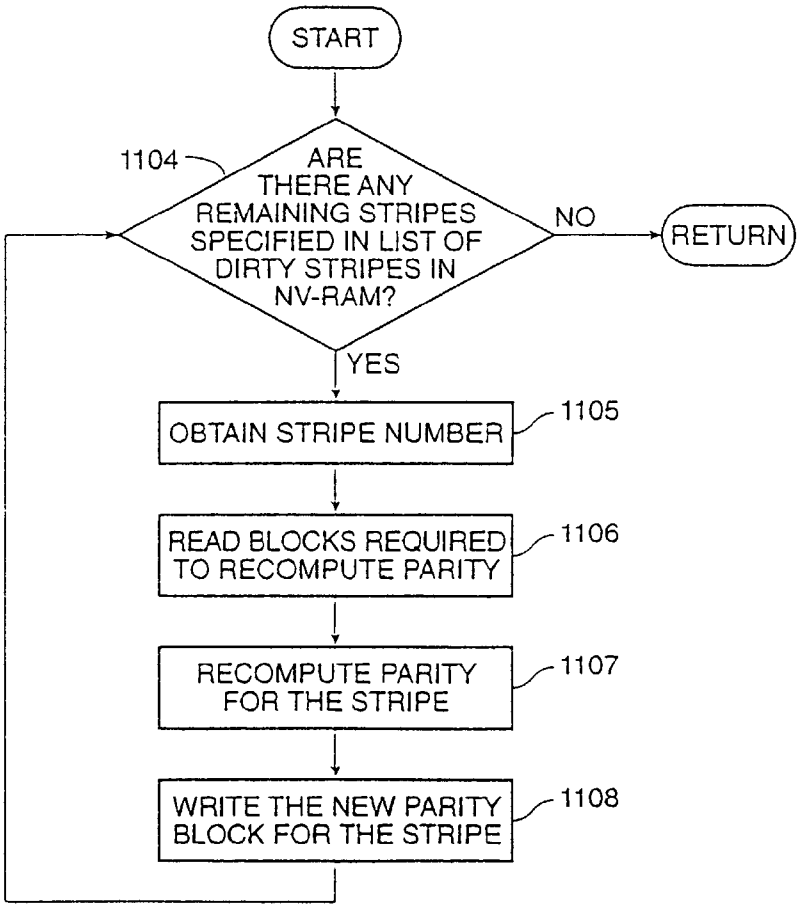


FIG. 11B

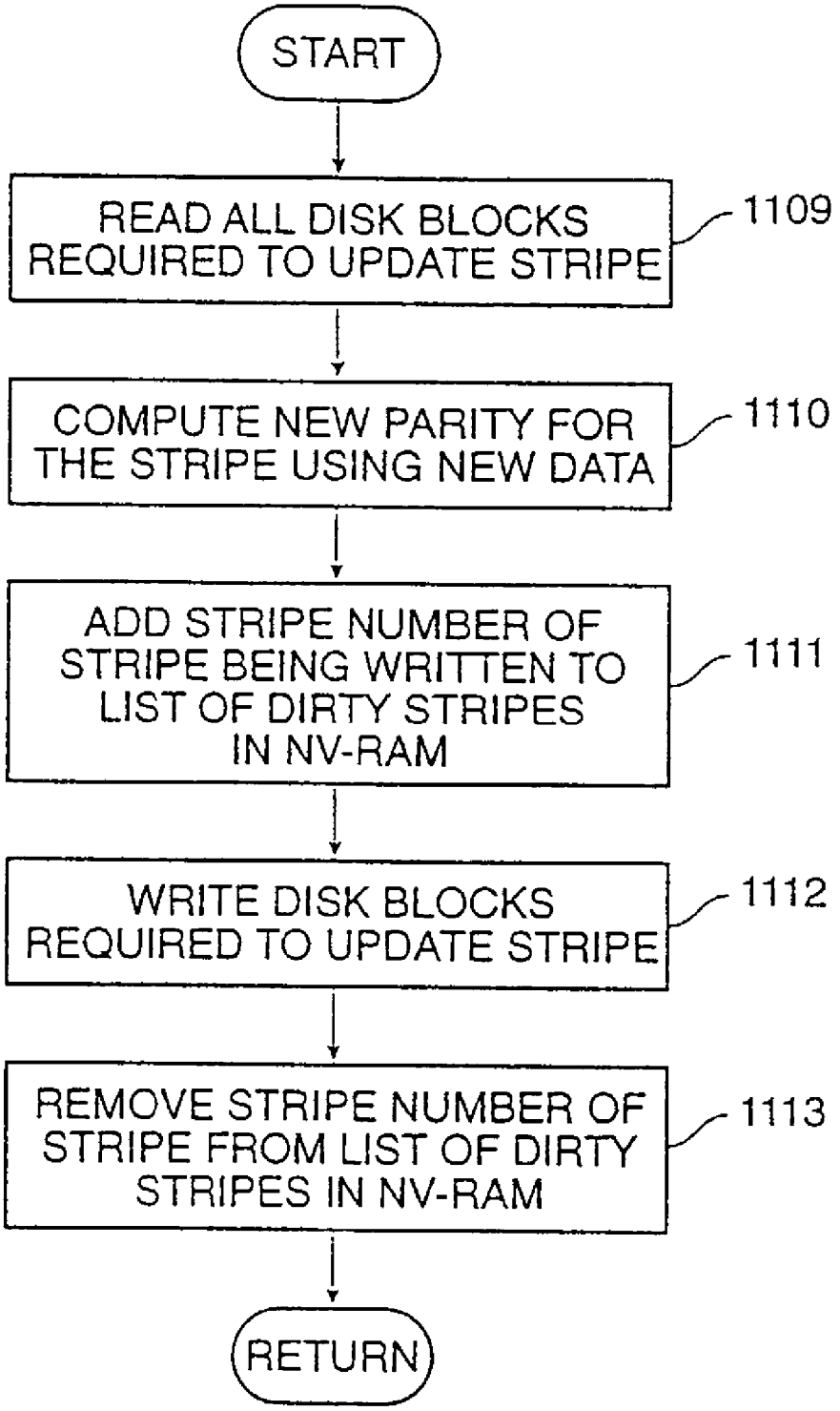

START
READ ALL DISK BLOCKS REQUIRED TO UPDATE STRIPE
1109
COMPUTE NEW PARITY FOR THE STRIPE USING NEW DATA
1110
ADD STRIPE NUMBER OF STRIPE BEING WRITTEN TO LIST OF DIRTY STRIPES IN NV-RAM
1111
WRITE DISK BLOCKS REQUIRED TO UPDATE STRIPE
1112
REMOVE STRIPE NUMBER OF STRIPE FROM LIST OF DIRTY STRIPES IN NV-RAM
1113
RETURN

FIG. 11C

	PARITY	DATA 0	DATA 1	DATA2	NV-RAM DATA 1	NV-RAM DATA 2
0						
t_A	12	4	7	1	?	?
t_B	12	4	7	1	$\boxed{7}$	$\boxed{1}$
t_C	12	$\boxed{2}$	7	1	7	1
***						
t_D	(10)	2	7	1	7	1
t_E	(10)	2	7	1	$\boxed{?}$	$\boxed{?}$

TIME

FIG. 12A

	PARITY	DATA 0	DATA 1	DATA2	NV-RAM DATA 1	NV-RAM DATA 2
0						
t_A	12	4	7	1	?	?
t_B	12	4	7	1	$\boxed{7}$	$\boxed{1}$
t_C	12	$\boxed{2}$	7	1	7	1
***						
t_D	12	(4)	7	1	7	1
t_E	$\boxed{8}$	(0)	7	1	7	1
t_F	8	(0)	7	1	$\boxed{?}$	$\boxed{?}$

TIME

FIG. 12B

	PARITY	DATA 0	DATA 1	DATA2	NV-RAM DATA 1	NV-RAM DATA 2
0						
t_A	12	4	7	1	?	?
t_B	12	4	7	1	$\boxed{7}$	$\boxed{1}$
t_C	12	$\boxed{2}$	7	1	7	1
***						
t_D	12	2	(9)	1	7	1
t_E	$\boxed{10}$	2	(7)	1	7	1
t_F	10	2	(7)	1	$\boxed{?}$	$\boxed{?}$

TIME

FIG. 12C

PROVIDING PARITY IN A RAID SUB-SYSTEM USING NON-VOLATILE MEMORY

CROSS REFERENCE TO RELATED APPLICATIONS

This is a preliminary amendment for a continuation of application Ser. No. 09/345,246 filed Jun. 8, 1999 (now allowed, projected to issue as U.S. Pat. No. 6,480,969 B1 on Nov. 12, 2002), which is a continuation of application Ser. No. 08/471,218, filed Jun. 5, 1995 (now U.S. Pat. No. 5,948,110), which is a continuation of application Ser. No. 08/071,798, filed Jun. 4, 1993 (now abandoned). This application also is a continuation of PCT application Ser. No. PCT/US94/06321 filed Jun. 2, 1994.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention is related to the field of error correction techniques for an array of disks.

2. Background Art

A computer system typically requires large amounts of secondary memory, such as a disk drive, to store information (e.g. data and/or application programs). Prior art computer systems often use a single "Winchester" style hard disk drive to provide permanent storage of large amounts of data. As the performance of computers and associated processors has increased, the need for disk drives of larger capacity, and capable of high speed data transfer rates, has increased. To keep pace, changes and improvements in disk drive performance have been made. For example, data and track density increases, media improvements, and a greater number of heads and disks in a single disk drive have resulted in higher data transfer rates.

A disadvantage of using a single disk drive to provide secondary storage is the expense of replacing the drive when greater capacity or performance is required. Another disadvantage is the lack of redundancy or back up to a single disk drive. When a single disk drive is damaged, inoperable, or replaced, the system is shut down.

One prior art attempt to reduce or eliminate the above disadvantages of single disk drive systems is to use a plurality of drives coupled together in parallel. Data is broken into chunks that may be accessed simultaneously from multiple drives in parallel, or sequentially from a single drive of the plurality of drives. One such system of combining disk drives in parallel is known as "redundant array of inexpensive disks" (RAID). A RAID system provides the same storage capacity as a larger single disk drive system, but at a lower cost. Similarly, high data transfer rates can be achieved due to the parallelism of the array.

RAID systems allow incremental increases in storage capacity through the addition of additional disk drives to the array. When a disk crashes in the RAID system, it may be replaced without shutting down the entire system. Data on a crashed disk may be recovered using error correction techniques.

RAID Arrays

RAID has six disk array configurations referred to as RAID level 0 through RAID level 5. Each RAID level has advantages and disadvantages. In the present discussion, only RAID levels 4 and 5 are described. However, a detailed description of the different RAID levels is disclosed by Patterson, et al. in *A Case for Redundant Arrays of Inexpensive Disks* (*RAID*), ACM SIGMOD Conference, June 1988. This article is incorporated by reference herein.

RAID systems provide techniques for protecting against disk failure. Although RAID encompasses a number of different formats (as indicated above), a common feature is that a disk (or several disks) stores parity information for data stored in the array of disks. A RAID level 4 system stores all the parity information on a single parity disk, whereas a RAID level 5 system stores parity blocks throughout the RAID array according to a known pattern. In the case of a disk failure, the parity information stored in the RAID subsystem allows the lost data from a failed disk to be recalculated.

FIG. **1** is a block diagram illustrating a prior art system implementing RAID level 4. The system comprises N+1 disks **112**–**118** coupled to a computer system, or host computer, by communication channel **130**. In the example, data is stored on each hard disk in 4 KByte (KB) blocks or segments. Disk **112** is the Parity disk for the system, while disks **114**–**118** are Data disks **0** through N−1. RAID level 4 uses disk "striping" that distributes blocks of data across all the disks in an array as shown in FIG. **1**. A stripe is a group of data blocks where each block is stored on a separate disk of the N disks along with an associated parity block on a single parity disk. In FIG. **1**, first and second stripes **140** and **142** are indicated by dotted lines. The first stripe **140** comprises Parity **0** block and data blocks **0** to N−I. In the example shown, a first data block **0** is stored on disk **114** of the N+1 disk array. The second data block **1** is stored on disk **116**, and so on. Finally, data block N−i is stored on disk **118**. Parity is computed for stripe **140** using well-known techniques and is stored as Parity block **0** on disk **112**. Similarly, stripe **142** comprising N data blocks is stored as data block N on disk **114**, data block N+1 on disk **116**, and data block 2N−1 on disk **118**. Parity is computed for the 4 stripe **142** and stored as parity block **1** on disk **112**.

As shown in FIG. **1**; RAID level 4 adds an extra parity disk drive containing error-correcting information for each stripe in the system. If an error occurs in the system, the RAID array must use all of the drives in the array to correct the error in the system. RAID level 4 performs adequately when reading small pieces of data. However, a RAID level 4 array always uses the dedicated parity drive when it writes data into the array.

RAID level 5 array systems also record parity information. However, it does not keep all of the parity sectors on a single drive. RAID level 5 rotates the position of the parity blocks through the available disks in the disk array of N+1 disk. Thus, RAID level 5 systems improve on RAID 4 performance by spreading parity data across the N+1 disk drives in rotation, one block at a time. For the first set of blocks, the parity block might be stored on the first drive. For the second set of blocks, it would be stored on the second disk drive. This is repeated so that each set has a parity block, but not all of the parity information is stored on a single disk drive. In RAID level 5 systems, because no single disk holds all of the parity information for a group of blocks, it is often possible to write to several different drives in the array at one instant. Thus, both reads and writes are performed more quickly on RAID level 5 systems than RAID 4 array.

FIG. **2** is a block diagram illustrating a prior art system implementing RAID level 5. The system comprises N+1 disks **212**–**218** coupled to a computer system or host computer **120** by communication channel **130**. In stripe **240**, parity block **0** is stored on the first disk **212**. Data block **0** is stored on the second disk **214**, data block **1** is stored on the

third disk **216**, and so on. Finally, data block N−1 is stored on disk **218**. In stripe **212**, data block N is stored on the first disk **212**. The second parity block **1** is stored on the second disk **214**. Data block N+1 is stored on disk **216**, and so on. Finally, data block 2N−1 is stored on disk **218**. In M−1 stripe **244**, data block MN−N is stored on the first disk **212**. Data block MN−N+1 is stored on the second disk **214**. Data block MN−N+2 is stored on the third disk **216**, and so on. Finally, parity block M−1 is stored on the nth disk **218**. Thus, FIG. **2** illustrates that RAID level 5 systems store the same parity information as RAID level 4 systems, however, RAID level 5 systems rotate the positions of the parity blocks through the available disks **212**–**218**.

In RAID level 5, parity is distributed across the array of disks. This leads to multiple seeks across the disk. It also inhibits simple increases to the size of the RAID array since a fixed number of disks must be added to the system due to parity requirements.

The prior art systems for implementing RAID levels 4 and 5 have several disadvantages. The first disadvantage is that, after a system failure, the parity information for each stripe is inconsistent with the data blocks stored on the other disks in the stripe. This requires the parity for the entire RAID array to be recalculated. The parity is recomputed entirely because there is no method for knowing which parity blocks are incorrect. Thus, all the parity blocks in the RAID array must be recalculated. Recalculating parity for the entire RAID array is highly time consuming since all of the data stored in the RAID array must be read. For example, reading an entire 2 GB disk at maximum speed takes 15 to 20 minutes to complete. However, since few computer systems are able to read very many disks in parallel at maximum speed, recalculating parity for a RAID array takes even longer.

One technique for hiding the time required to recompute parity for the RAID array is to allow access to the RAID array immediately, and recalculate parity for the system while it is on-line. However, this technique suffers two problems. The first problem is that, while recomputing parity, blocks having inconsistent parity are not protected from further corruption. During this time, a disk failure in the RAID array results in permanently lost data in the system. The second problem with this prior art technique is that RAID subsystems perform poorly while calculating parity. This occurs due to the time delays created by a plurality of input/output (I/O) operations imposed to recompute parity.

The second disadvantage of the prior art systems involves writes to the RAID array during a period when a disk is not functioning. Because a RAID subsystem can recalculate data on a malfunctioning disk using parity information, the RAID subsystem allows data to continue being read even though the disk is malfunctioning. Further, many RAID systems allow writes to continue although a disk is malfunctioning. This is disadvantageous since writing to a broken RAID array can corrupt data in the case of a system failure. For example, a system failure occurs when an operating system using the RAID array crashes or when a power for the system fails or is interrupted otherwise. Prior art RAID subsystems do not provide protection for this sequence of events.

SUMMARY OF THE INVENTION

The present invention is a method for providing error correction for an array of disks using non-volatile random access memory (NV-RAM).

Non-volatile RAM is used to increase the speed of RAID recovery from disk error(s). This is accomplished by keeping a list of all disk blocks for which the parity is possibly inconsistent. Such a list of disk blocks is smaller than the total number of parity blocks in the RAID subsystem. The total number of parity blocks in the RAID subsystem is typically in the range of hundreds of thousands of parity blocks. Knowledge of the number of parity blocks that are possibly inconsistent makes it possible to fix only those few blocks, identified in the list, in a significantly smaller amount of time than is possible in the prior art. The present invention also provides a technique of protecting against simultaneous system failure and a broken disk and of safely writing to a RAID subsystem with one broken disk.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram of a prior art RAID level 4 subsystem;

FIG. **2** is a block diagram of a prior art RAID level 5 subsystem;

FIGS. **3**A–**3**C are prior art diagrams illustrating recomputation of data stored in a "stripe";

FIGS. **4**A–**4**B are prior art timing diagrams for parity corruption on system failure;

FIG. **5** is a timing diagram of the present invention for preventing data corruption on occurrence of a write to a malfunctioning disk;

FIG. **6** is a prior art timing diagram illustrating data corruption on a write with a broken disk;

FIGS. **7**A–**7**B are timing diagrams of the present invention for preventing data corruption on a write with a broken disk;

FIG. **8** is a diagram illustrating the present invention.

FIG. **9** is a prior art timing diagram illustrating data corruption for simultaneous system and disk failures;

FIGS. **10**A–**10**C are timing diagrams of the present invention preventing data corruption for simultaneous system and disk failures; and,

FIGS. **11**A–**11**C are flow diagrams of the present invention illustrating the process of recovery.

FIGS. **12**A–**12**C are timing diagrams for parity by recalculation.

DETAILED DESCRIPTION OF THE PRESENT INVENTION

A method and apparatus for providing error correction for an array of disks using non-volatile random access memory (NV-RAM) is described. In the following description, numerous specific details, such as number and nature of disks, disk block sizes, etc., are described in detail in order to provide a more thorough description of the present invention. It will be apparent, however, to one skilled in the art, that the present invention may be practiced without these specific details. In other instances, well-known features have not been described in detail so as not to unnecessarily obscure the present invention.

In particular, many examples consider the case where only one block in a stripe is being updated, but the techniques described apply equally well to multi-block updates.

The present invention provides a technique for: reducing the time required for recalculating parity after a system failure; and, preventing corruption of data in a RAID array when data is written to a malfunctioning disk and the system crashes. The present invention uses non-volatile RAM to reduce these problems. A description of the prior art and its

corresponding disadvantages follows. The disadvantages of the prior art are described for: parity corruption on a system failure; data corruption on write with broken disk; and, data corruption with simultaneous system and disk failures.

Recomputing Lost Data with RAID

Parity is computed by Exclusive-ORing the data blocks stored in a stripe. The parity value computed from the N data blocks is recorded in the parity block of the stripe. When data from any single block is lost (i.e., due to a disk failure), the lost data for the disk is recalculated by Exclusive-ORing the remaining blocks in the stripe. In general, whenever a data block in a stripe is modified, parity must be recomputed for the stripe. When updating a stripe by writing all N data blocks, parity can be computed without reading any data from disk and parity and data can be written together, in just one I/O cycle. Thus, writing to all N data blocks in a stripe requires a minimum amount of time. When writing a single data block to disk, parity-by-subtraction is used (described below). One I/O cycle is required to read the old data and parity, and a second I/O cycle is required to write the new date and parity. Because the spindles of the disks in the RAID array are not synchronized, the writes do not generally occur at exactly the same time. In some cases, the parity block will reach the disk first, and in other cases, one of the data blocks will reach the disk first. The techniques described here do not depend on the order in which blocks reach the disk.

Another alternative for disks having non-synchronized spindles is for parity to be computed first and the parity block written to disk before a data block(s) is written to disk. Each data block on a disk in the RAID array stores 4 KB of data. In the following discussion, the data in each 4 KB block is viewed as a single, large integer (64 K-bits long). Thus, the drawings depict integer values for information stored in the parity and data disk blocks. This convention is used for illustration only in order to simplify the drawings.

FIG. **3**A is a diagram illustrating a prior art RAID level 4 subsystem, where N=3, comprising four disks, **330**–**336**. In the diagram, disk **330** is the parity disk. Disks **332**–**336** are data disks. The diagram illustrates a stripe **320** contained on the disks **330**–**336** in the RAID array. Disk block **330**A is a parity block containing the integer value 12. Disk blocks **332**A–**336**A are data blocks of the stripe **320**, respectively. Data blocks **332**A–**336**A contain data values of 4, 7 and 1, respectively. Data for each block **332**A–**336**A in a single stripe **320** is represented as an integer. Parity for stripe **320** is represented as the sum of data values stored in data blocks **332**A–**336**A. Parity block **330**A contains the value 12 (i.e., 4+7+1). FIG. **3**A is a drawing that is merely one example of an error correction technique using parity. The parity value is the Exclusive-OR of the data blocks **332**A–**336**A, but the mathematical properties of addition match those of the Exclusive-OR function. Therefore, addition is used in FIG. **3**A.

FIG. **3**B is a timing diagram of activity on the stripe **320** illustrated in FIG. **3**A. The table has headings of Parity, Data **0**, Data **1** and Data **2**. The values 12, 4, 7 and 1 are illustrated under the corresponding table headings.

FIG. **3**B is a table illustrating a stripe having a lost data block at time T_B. As illustrated in FIG. **3**B, stripe **320** contains lost data in data block **1** from data disk **334** of FIG. **3**A. This is illustrated in the table by a question mark enclosed in a box under the data **1** heading. At time T_A, parity, data **0** and data **2** have values of 12, 4 and 1, respectively. The data on disk **334** for data block **1** can be recalculated in real time as follows:

$$\text{Data } \mathbf{1} = \text{Parity} - \text{Data } \mathbf{0} - \text{Data } \mathbf{2} = 12 - 4 - 1 = 7, \quad (1)$$

where data block **1** is computed using the parity block, data block **0** and data block **2**. Thus, the data value 7 stored in data block **1** of disk **334** shown in FIG. **3**A can be recomputed at time T_C. In FIG. **3**B, at time T_C, the value 7 that has been recomputed for data block **1**, is indicated by being enclosed within parentheses. In subsequent figures, recomputed values are represented using parentheses. That is, the parentheses indicate data for a broken disk as computed by the parity and data on the other disks.

As shown in FIG. **3**B, data on a broken disk can be recomputed using the parity disk and the remaining disks in the disk array. The broken disk **334** of FIG. **3**A can eventually be replaced and the old contents of the disk can be recalculated and written to a new disk. FIG. **3**C is a block diagram of the RAID subsystem containing a new data **1** disk **338**. As shown in FIG. **3**E, stripe **320** has values of 12, 4, 7 and 1 for parity, data **0**, new data **1** and data **2**. These values are stored in parity block **330**A and data blocks **332**A, **338**A and **336**A. Thus, a new disk **338** replaces broken disk **334** of the RAID system, and the data value stored previously in data block **334**A of disk **334** can be computed as shown above and stored in data block **338**A of replacement disk **338**.

When new data is written to a data block, the parity block is also updated. Parity is easily computed, as described above, when all data blocks in a stripe are being updated at once. When this occurs, the new value for parity is recalculated from the information being written to the disks. The new parity and data blocks are then written to disk. When only some of the data blocks in a stripe are modified, updating the parity block is more difficult since more I/O operations are required. There are two methods for updating parity in this case: parity update by subtraction; and, parity update by recalculation.

For example, when a single data block is written, the RAID system can update parity by subtraction. The RAID system reads the parity block and the block to be overwritten. It first subtracts the old data value from the parity value, adds the new data value of the data block to the intermediate parity value, and then writes both the new parity and data blocks to disk.

For recalculation of parity, the RAID system first reads the other N−1 data blocks in the stripe. After reading the N−1 data blocks, the RAID system recalculates parity from scratch using the modified data block and the N−1 data blocks from disk. Once parity is recalculated, the new parity and data blocks are written to disk.

Both the subtraction and recalculation technique for updating parity can be generalized to situations where more than one data block is being written to the same stripe. For subtraction, the parity blocks and the current contents of all data blocks that are about to be overwritten are first read from disk. For recalculation, the current contents of all data blocks that are not about to be overwritten are first read from disk. The instance where all N data blocks in the stripe are written simultaneously is a degenerate case of parity by recalculation. All data blocks that are not being written are first read from disk, but in this instance, there are no such blocks.

How Stripes Become Inconsistent During System Failure

An inconsistent stripe comprises a parity block that does not contain the Exclusive-OR of all other blocks in the

stripe. A stripe becomes inconsistent when a system failure occurs while some of the writes for an update have been completed but others have not. For example, when a first data block is being overwritten. As previously described, the parity block for the stripe is recomputed and overwritten as well as the data block. When the system fails after one of the data blocks has been written to disk, but not the other, then the stripe becomes inconsistent.

A stripe can only become inconsistent when it is being updated. Thus, the number of potentially inconsistent stripes at any instant is limited to the number of stripes that are being updated. For this reason, the present invention maintains a list in NV-RAM comprising all the stripes that are currently being updated. Since only these stripes can potentially be corrupted, parity is recalculated after a system failure for only the stripes stored in the list in NV-RAM. This greatly reduces the total amount of time required for recalculating parity after a system failure in comparison to the prior art methods, described previously, that take much longer.

Parity Corruption on a System Failure in the Prior Art

In the following diagrams, the value indicated within parentheses for a malfunctioning data disk is not an actual value stored on disk. Instead, it is a calculated value retained in memory for the broken disk in the RAID array.

FIG. **4**A is a prior art diagram illustrating a system crash while changing values are written to disks **330**–**336** of FIG. **3**A. The diagram is for the case where the data block reaches the disk before the parity block. As indicated in FIG. **4**A, time is increasing in a downward direction. At time T_A, the parity block has a value of 12 and data blocks **0** to **2** have values of 4, 7, and 1, respectively. At time T_B, a new value of 2 is written (indicated by a box around the value 2) to data block **0**, thereby replacing the value of 4 that is stored in data block **0** at time T_A. The other values stored in data blocks **1** and **2** do not change. When operating normally, the prior art writes a new parity value of 10 (indicated by a box) at time T_C to the parity disk as indicated under the parity heading. This updates the parity block for the write to data block **0** at time T_B. The new value of 10 for parity at time T_C is computed from the values of 2, 7, and 1 of data blocks **0** to **2**, respectively. Thus, the timing diagram in FIG. **4**A illustrates a prior art RAID subsystem in the case where the data block reaches disk before the parity block.

When a system failure occurs between time T_B and T_C in FIG. **4**A, parity is corrupted for the stripe. The timing diagram shows that a new data value of 2 is written to data disk **0** at time T_B before recomputed parity for the stripe is updated. Thus, when the RAID subsystem subsequently restarts, the parity disk has the old value of 12 (indicated by an underline) instead of the correct value of 10. This occurs since the stripe was not updated before the system failure occurred. The parity for the stripe is now corrupted since:

$$\text{Parity}=\text{Data } \mathbf{0}+\text{Data } \mathbf{1}+\text{Data } \mathbf{2}=2+7+1=10\neq 12. \quad (2)$$

Similarly, FIG. **4**B is another prior art diagram illustrating a system crash while changing values are written to disks **330**–**336** of FIG. **3**A. The diagram is for the case where the parity block reaches disk before the data block. At time T_A, the parity block has a value of 12 and data blocks **0** to **2** have values of 4, 7, and 1, respectively. At time T_B, a new value of 10 is written (indicated by a box around the value 2) to the parity block, thereby replacing the value of 12 that is stored in the parity block at time T_A. The data values stored in data blocks **0**–**2** do not change. The new value of 10 for parity at time T_B is computed from the values of 7 and 1 for data blocks **1** and **2**, respectively, and the new value of 2 for data block **0**. When operating normally, the prior art writes the new data value of 2 (indicated by a box) it time T_C to the data disk **0** as indicated under the Data **0** heading. This updates the data block **0** in accordance with the write to the parity block at time T_B. Thus, the timing diagram in FIG. **4**A illustrates a prior art RAID subsystem in the case where the parity block reaches disk before the data block.

When a system failure occurs between time T_B and T_C in FIG. **4**B, parity is corrupted for the stripe. The timing diagram shows that the new parity value of 10 is written to the parity disk at time T_B before data block **0** of the stripe is updated. Thus, when the RAID subsystem subsequently restarts, data disk **0** has the old value of 4 (indicated by an underline) instead of the correct value of 2. This occurs because the stripe was not updated before the system failure occurred. The parity for the stripe is now corrupted since:

$$\text{Parity}=\text{Data } \mathbf{0}+\text{Data } \mathbf{1}+\text{Data } \mathbf{2}=4+7+1=12\neq 10. \quad (3)$$

FIGS. **4**A–**4**B illustrate two cases of writing new data to a data block and updating the parity disk where the spindles of the disks in the RAID array are not synchronized. The first case shown in FIG. **4**A illustrates a new data value reaching the data disk first, and then subsequently updating the parity value on the parity disk. The second case illustrated in FIG. **4**B illustrates parity reaching disk first followed by the data update. For FIGS. **4**A and **4**B, when the system fails between times T_B and T_C, corruption of the file system occurs. If the system fails after time T_B in FIGS. **4**A and **4**B, then the parity values illustrated are not correct for the system. In the case of the system illustrated in FIG. **4**A, the new data values have a sum of 10, which is equal to the values of 2, 7 and 1. However, the parity value at time T_B indicates a value of 12. Thus, the parity value stored on the parity disk does not equal the new parity value for the data values stored on data disk **0**–**2**. Similarly, if a failure occurs after time T_B for the second system illustrated in FIG. **4**B, the data disks **0**–**1** have values of 4, 7 and 1, respectively. The parity value for these data blocks is equal to 12. However, parity in this system is first updated before writing the new data value to disk, therefore, the parity stored on the parity disk at time T_B is equal to 10. Thus, subsequent to time T_B, the parity stored on the parity disk does not equal the parity value for the data blocks since the new data was not updated before the system failed.

In the prior art, after a system fails, parity is recalculated for all of the stripes on occurrence of a system restart. This method of recalculating parity after a failure for all stripes requires intensive calculations, and therefore, is very slow. The present invention is a method for recalculating parity after a system failure. The system maintains a list of stripes having writes in progress in non-volatile RAM. Upon restarting after a system failure, just the list of stripes with writes in progress that are stored in non-volatile RAM are recalculated.

Data Corruption on Write with Broken Disk in the Prior Art

When writing to a RAID array that has a malfunctioning or broken disk, data corruption occurs during system failure. FIG. **6** is a prior art diagram illustrating data corruption for a malfunctioning disk when a system failure occurs where the data disk is updated for the new data value before parity is written to disk. In FIG. **6**, data disk **1** is shown to be malfunctioning by indicating values within parentheses under the Data **1** heading. At time T_A, the parity disk has a value of 12. Prior to time T_A when data disk **1** malfunctions, the parity disk value is equal to the sum of data disks **0** to

2 having values of 4, 7 and 1, respectively. The value of 7 for data block **1** at time T_A is enclosed within parentheses. This value does not represent a value stored on data disk **1**, but instead is computed from the parity block and data blocks **0** and **2** of the stripe as follows:

$$\text{Data } \mathbf{0}=\text{Parity}-\text{Data } \mathbf{1}-\text{Data } \mathbf{2}=12-4-1=7. \quad (4)$$

At time T_B, a new value of 2 is written to data disk **0** (indicated by enclosing 2 within a box). At time T_B, parity has not been updated for the new value of 2 written to data disk **0** and has a value of 12. Thus, the computed value for data block **1** is 9 instead of 7. This is indicated in FIG. **6** by enclosing the value 9 within parentheses for data disk **1** at time T_B.

When operating normally at time T_C, the parity block is updated to 10 due to the value of 2 written to data block **0** at time T_B. The new value of 10 for parity at time T_C is indicated within a rectangle. For a parity value of 10, the correct value of 7 for data block **1** is indicated within parentheses. As indicated in the FIG. **6**, because data disk **1** is broken, the data stored in data block **1** is calculated based on the other blocks in the disk array. After the first write at time T_B for data block **0**, the computed value of data block **1** is incorrect. The value of 9 for data block **1** is incorrect until the second write for parity at time T_C is completed.

When a system failure occurs between times T_B and T_C, writing to a RAID array that has a malfunctioning or broken disk corrupts data in the stripe. As shown in FIG. **6** for the prior art, parity is not updated and therefore has a value of 12 (indicated by an underline). Thus, the computed value for data block **1** of the stripe is incorrect and the stripe is corrupted as follows:

$$\text{Data } \mathbf{1}=\text{Parity}-\text{Data } \mathbf{0}-\text{Data } \mathbf{2}=12-2-1=9\neq 7. \quad (5)$$

Similar corruption of data occurs for the case where parity reaches disk before data does.

Data Corruption with Simultaneous System and Disk Failures

RAID systems are most likely to experience a disk failure when a system failure occurs due to power interruption. Commonly, a large, transient voltage spike occurring after power interruption damages a disk. Thus, it is possible for a stripe to be corrupted by simultaneous system and disk failures.

FIG. **9** is a prior art diagram illustrating simultaneous system and disk failures where the data disk is updated for a new data value before parity is written to disk. At time T_A, the parity disk has a value of 12 and data disks **0–2** have values of 4, 7, and 1, respectively. At time T_B, a new value of 2 is written to data disk **0** (indicated by a box). At time T_B, parity is not updated for the new value of 2 written to data disk **0** and has a value of 12. When a system failure occurs between times T_B and T_C, the value of disk **1** is corrupted. This occurs due to simultaneous system and disk failures between times T_B and T_C.

At time T_C, parity is not updated due to the system failure and therefore has a value of 12 instead of 10. Further, data disk **1** is corrupted due to the disk failure. The computed value of 9 for data block **1** is incorrect. It is computed incorrectly for data disk **1** using the corrupt parity value as follows:

$$\text{Data } \mathbf{1}=\text{Parity}-\text{Data } \mathbf{0}-\text{Data } \mathbf{2}=12-2-1=9\neq 7. \quad (7)$$

Data is similarly corrupted for the case where parity reaches disk before data.

Overview of the Present Invention

FIG. **8** is a diagram illustrating the present invention for providing error correction using NV-RAM for a RAID system comprising host computer **810**, RAID controller **814** including NV-RAM **816**, and N+1 disks **820–826**. Host computer **810** is coupled to RAID controller **814** by a first communications channel **812**. RAID controller **814** comprises NV-RAM **816** for storing stripes of the RAID array **828** that are possibly in an inconsistent state. RAID controller **814** is coupled to the N+1 disks **820–826** of the RAID array **828** by a second communications channel **818**. The RAID array **828** comprises parity disk **820** and N data disks **822–826**, respectively.

NV-RAM **816** is used to increase the speed of RAID recovery after a system failure by maintaining a list of all parity blocks stored on parity disk **820** that are potentially inconsistent. Typically, this list of blocks is small. It may be several orders of magnitude smaller than the total number of parity blocks in the RAID array **828**. For example, a RAID array **828** may comprise hundreds of thousands of parity blocks while the potentially inconsistent blocks may number only several hundred or less. Knowledge of the few parity blocks that are potentially inconsistent facilitates rapid recalculation of parity, since only those parity blocks have to be restored.

The present invention also uses NV-RAM **816** to safely write data to a RAID array **828** having a broken disk without corrupting data due to a system failure. Data that can be corrupted is copied into NV-RAM **816** before a potentially corrupting operation is performed. After a system failure, the data stored in NV-RAM **816** is used to recover the RAID array **828** into a consistent state.

FIGS. **11**A–C are flow diagrams illustrating the steps performed by the present invention. Referring first to FIG. **11**A, a boot operation is executed. At decision block **1101** a check is made to determine if the system has just failed. If decision block returns true, the system proceeds to step **1102** (FIG. **11**B) and executes a recovery process. If decision block returns false, the system proceeds to step **1103** (FIG. **11**C) for normal operation.

Referring now to FIG. **11**B, a flow diagram of recovery after system failure is illustrated. At decision block **1104** a check is made to determine if there are any remaining stripes specified in the list of dirty stripes in NV-RAM. If decision block **1104** returns false, the system proceeds to a return step. If decision block **1104** returns true, the system proceeds to step **1105**.

At step **1105**, the stripe number is obtained. At step **1106**, the data blocks of the identified stripe required to recompute parity are read. Parity is recomputed for the stripe at step **1107**. At step **1108**, the new parity block for the stripe is written. The system then returns to decision block **1104**.

Normal operation is illustrated in FIG. **11**C. At step **1109**, all blocks required to update a stripe are read. At step **1110** a new parity for the stripe is computed using the new data. The stripe number of that stripe is written to the list of dirty stripes in NV-RAM at step **1111**. At step **1112**, the disk blocks required to update the stripe are written to disk. At step **1113** the number of the stripe is removed from the list of dirty stripes in NV-RAM.

Parity Corruption for a System Failure Using NV-RAM

FIG. **5** is a diagram illustrating the present invention for preventing corruption of parity using NV-RAM. The timing diagram is described with reference to FIGS. **11**A–C. FIG. **5** is a timing diagram for the present invention illustrating a system crash while changing values are written to disks

330–336 of FIG. 3A. The diagram is for the case where parity is computed for a new data value and the data is written to disk prior to updating the parity block. At time T_A, the parity block has a value of 12 and data blocks **0** to **2** have values of 4, 7, and 1, respectively. At time T_A, step **1109** of FIG. **11**C for the present invention is performed where blocks necessary to update the stripe are read into memory. The system performs step **1110** where a new parity value is computed dependent upon the new data value(s). At time T_B, step **1111** of FIG. **11**C is performed where the stripe number is written to NV-RAM.

At time T_C, in step **1112**, the new data value of 2 is written (indicated by a box around the value 2) to data block **0**, thereby replacing the value of 4 that is stored in data block **0** at time T_B. The other values stored in data blocks **1** and **2** do not change. First, consider the normal case where the system does not fail. The present invention writes a new parity value of 10 (indicated by a box under the parity heading) at time T_D in step **1112**. This updates the parity block for the write to data block **0** at time T_C. At time T_E, in step **1113**, the stripe number in NV-RAM is cleared. Thus, the stripe comprising the blocks for the parity disk and data disks **0**–**2** have values of 10, 2, 7, and 1, respectively.

Next, consider the ruse when the system does fail between time t_i and t_D (between steps **1111** and **1113**). The system reboots, and begins execution at START in FIG. **11**A.

In decision block **1101**, at time T_D, when a system fault occurs, decision block **1101** returns true (Yes). The stripe has a value of 12 (indicated by an underline) for parity and values for data disks **0**–**2** of 2, 7, and 1, respectively. As illustrated in FIG. **5** for time T_C, parity is corrupted after a system failure since:

$$\text{Parity}=\text{Data } \mathbf{0}+\text{Data } \mathbf{1}+\text{Data } \mathbf{2}=2+7+1=10\neq 12. \tag{9}$$

However, the stripe can be recovered to a consistent state. NV-RAM includes an indication of the stripes that are candidates for recovery, i.e. a list of stripes that are being updated. Everything but the parity value is available on disk (the "2" having been written to disk at time T_C). The data values for the stripe are read from disk and a new parity value of 10 is calculated.

$$\text{Parity}=\text{Data } \mathbf{0}+\text{Data } \mathbf{1}+\text{Data } \mathbf{2}=2+7+1=10. \tag{10}$$

Thus, the newly calculated parity value of 10 is written to the parity disk in step **1108** at time T_D, and the stripe is no longer corrupt.

The following is an example of pseudo code that describes the operation of FIG. **11**C:

(1) Read all disk blocks required to update stripe.
(2) Calculate new parity contents.
(3) Add stripe # for stripe being written to NV-RAM dirty stripe list.
(4) Write all disk blocks required to update stripe.
(5) Remove stripe # for stripe just written from NV-RAM dirty stripe list.

After a system failure, a part of the start-up procedure of FIG. **11**B can be described by the following pseudo code:

```
for (all stripes specified in the NV-RAM dirty stripe list)
{
    (1) Read all data blocks in the stripe.
    (2) Recompute the parity block for the stripe.
    (3) Write the new parity block for the stripe.
}
```

Thus, the present invention prevents parity corruption after a system failure by using NV-RAM.

Parity Corruption Detection with a Bitmap Instead of a List

The previous section describes a technique in which a list of potentially corrupted stripes is kept in NV-RAM so that on reboot after a system failure, only the stripes in the list need to have their parity blocks recalculated. An alternate embodiment of the present invention uses a bitmap in NV-RAM to indicate the potentially corrupted stripes whose parity blocks must be recalculated after a system failure.

This technique uses a bitmap in which each bit represents a group of one or more stripes. A typical disk array might have 250,000 stripes. If each entry in the bitmap represents a single stripe, the bitmap will be about 32 KB. Letting each bit represent a group of 32 adjacent stripes reduces the size to 1 KB.

After a system failure, this technique is essentially identical to the "list of stripes" technique, except that the bitmap is used to determine which stripes need parity recalculation instead of the list. All stripe in groups whose bit is set in the bitmap have their parity recalculated.

Managing the bitmap during normal operation is slightly different than managing the list. It is no longer possible to clear a stripe's entry as soon as the update is complete, because a single bit can indicate activity in more than one stripe. One stripe's update may be done, but another stripe sharing the same bit may still be active.

Instead, the appropriate bit for a stripe is set just before the stripe is updated, but it is not cleared after the update is complete. Periodically, when the bitmap has accumulated too many entries, all blocks are flushed to disk, ensuring that there can be no inconsistent stripes, and the entire bitmap is cleared. The following pseudo-code implements this:

(1) Read all blocks required to update stripe.
(2) Calculate new parity contents.
(3) Set bitmap entry for stripe being updated.
(4) Write all disk blocks required to update stripe.
(5) If bitmap is too full, wait for all blocks to reach disk and clear the entire bitmap.

In case of system failure, the bitmap results in more blocks to clean than the list, but the savings are still considerable compared with recomputing parity for all stripes in the system. A typical RAID system has 250,000 stripes, so even if 2,500 potentially-corrupted stripes are referenced in the bitmap, that is just 1% of the stripes in the system.

The bitmap technique is especially useful with write-caching disks which don't guarantee that data will reach disk in the case of power failure. Such disks may hold data in RAM for some period before actually writing it. This means that parity corruption is still a possibility even after the stripe update phase has completed. The list technique would not work, because the stripe's parity is still potentially corrupted even though the stripe has been removed from the list.

Thus, using the bitmap technique and instructing each disk to flush its internal cache at the same time that the bitmap is cleared, allows the invention to work in combination with write-caching disk drives.

Data Corruption on Write with Broken Disk Using NV-RAM

The present invention solves this problem for data corruption on occurrence of a write with a malfunctioning disk by saving data from the broken disk in non-volatile RAM. FIG. **7**A is a timing diagram of the present invention for preventing data corruption by storing data from a malfunctioning disk in NV-RAM. The drawing is discussed with reference to FIGS. **11**A–C. In FIG. **7**A, data is written to disk before parity is updated. At time T_A, broken data disk **1** is

illustrated having a value of 7 indicated within parentheses. The value of 7 within parentheses indicates that data disk **1** is malfunctioning and that it is the computed value for the disk. This value is computed by subtracting the values of 4 and 1 of data disks **0** and **2** from the value of 12 stored in the parity disk. In step **1109**, the stripe is read from the RAID array at time T_A. The NV-RAM is erased. This is indicated in FIG. **7**A by a question mark under the heading for NV-RAM.

At time T_B, a value of 7 for the malfunctioning data disk **1** is written into NV-RAM according to step **1109**. The value of 7 for data disk **1** that is written into NV-RAM is indicated by a rectangular box in FIG. **7**A. The system then computes a new value of parity for the stripe in step **1110** of FIG. **11**C.

At time T_C, a new value of 2 (indicated by a box) for data disk **0** is written to the disk before parity for the stripe is updated according to step **1112**. Therefore, at time T_C, the value for data disk **1** is 9 and is indicated within parentheses accordingly. In the normal case, where the system does not fail, a new parity value of 10 is written to disk at time T_D, and the computed value of disk **1** becomes 7 again, which is correct. When a system failure occurs between times T_C and T_D, a new value of parity is updated correctly using NV-RAM with respect to the value of 2 written to data disk **0** at time T_C.

The parity is correctly updated at time T_D by first reading the value for all functioning data disks, according to step **1106**, stored in NV-RAM, and recalculating its value as follows:

$$\text{Parity}=\text{Data } \mathbf{0}+\text{NV-RAM}+\text{Data } \mathbf{2}=2+7+1=10. \quad (12)$$

Thus, a correct value of 10 is computed for parity when the present invention restarts after a system crash. In step **1108**, the value of 10 is written to the parity disk at time T_D, thus returning the computed value of D1 to 1, which is correct. At time T_E, NV-RAM is cleared in step **1113**. Thus, the present invention prevents data from being corrupted by a system fault when a disk is malfunctioning by using NV-RAM.

FIG. **7**B is a timing diagram of the present invention for preventing data corruption by storing data from a malfunctioning disk in NV-RAM for the case where parity is written to disk before data is updated. At time T_A, broken data disk **1** is illustrated having a value of 7 indicated within parentheses. This value is computed as described above with reference to FIG. **7**A. In step **1109**, the stripe is read from the RAID array at time T_A. The NV-RAM is cleared which is indicated by a question mark under the heading for NV-RAM.

At time T_B, a value of 7 for the malfunctioning data disk **1** is written into NV-RAM according to step **1109**. The value of 7 for data disk **1** that is written into NV-RAM is indicated by a rectangular box in FIG. **7**B. The system then computes a new value of parity for the stripe in step **1110** of FIG. **11**.

At time T_C, a new value of 10 (indicated by a box) for parity is written to the parity disk in step **1108** before data block **0** is updated. Therefore, at time T_C, the value for data disk **1** is 5 and is indicated within parentheses accordingly. When a system failure occurs between times T_C and T_D, a new parity value is updated correctly for the parity disk using NV-RAM. At decision block **1101** after the system reboots, a check is made if a system failure occurred. The decision block accordingly returns true (Yes) in the present example, and continues at step **1104**.

Parity is correctly updated at time T_D by recalculating its value as follows:

$$\text{Parity}=\text{NV-data for broken disk (7)}+\text{on-disk data for all non broken disks}=4+7+1=12. \quad (13)$$

Thus, as shown in FIGS. **7**A–**7**B, when the system is about to write to a stripe, it saves the value for malfunctioning data disk **1** in non-volatile RAM. It then writes the new value for data disk **0** (parity) to disk. If a system crash occurs after the new value is written to disk **0** (the parity disk) at time T_C, the value for data disk **1** is corrupt. After the system failure, the new value of parity (data disk **0**) is calculated using the value of 7 stored in NV-RAM instead of the computed value of 5 for data disk **1**. The value of parity (data disk **0**) is then written to disk. Once this is completed, NV-RAM is erased.

Simultaneous System and Disk Failure Using NV-RAM

The present invention solves the problem of parity and data corruption when simultaneous system and disk failures occur by saving blocks of stripes in NV-RAM. Using NV-RAM allows the system to be recovered to a consistent state when a system crash occurs while updating multiple blocks (in the following example, data blocks **0** and **1**) in the system. Changing these data blocks further requires that the parity of the stripe be updated. The present invention always saves into NV-RAM any block that is read from disk (e.g., before updating data block **0**, read it into NV-RAM) for this purpose. Thus, stripe information can be recomputed from the data stored in NV-RAM. The present invention provides two solutions for this using parity by subtraction and parity by recalculation.

In parity by subtraction, data including parity and data blocks is read from disk before it is updated. FIG. **10**A is a timing diagram of the present invention for preventing parity and data corruption by storing blocks of a stripe in NV-RAM. The drawing is discussed with reference to FIGS. **11**A–C. In FIG. **10**A data is written to disk before parity is updated. At time T_A, the parity block and data block **0** are read from the RAID array. The NV-RAM is erased. This is indicated in FIG. **10**A by a question mark under the heading for NV-RAM.

At time T_B, the parity block and data block **0** are written into NV-RAM as they are read from disk. The parity block and data block **0** that are written into NV-RAM are indicated by a rectangular box in FIG. **10**A. The system then computes a new value of parity for a value of 2 for data block **0**.

At time T_C, the new value of 2 (indicated by a box) for data disk **0** is written to the disk before parity for the stripe is updated. When a system failure occurs between times T_C and T_D, a disk in the RAID array malfunctions, and thus the present invention provides solutions for the three cases of a broken disk: the parity disk; data disk **0**; and, data disk **2** (or **3**). At decision block **1101**, a check is made if a system failure occurred. The decision block accordingly returns true (Yes) in the present example, and continues at step **1104**. The three cases of a broken disk due to system failure where parity is calculated by subtraction are shown in FIGS. **10**A–**10**C, respectively.

At time T_D in FIG. **10**A, the parity disk malfunctions due to the system failure between times T_C and T_D. In this case, there is nothing to be done. No data is lost, and no information can be written to the parity disk.

Referring to FIG. **10**B, at time T_D, data disk **0** malfunctions due to the system failure between times T_C and T_D. The general equation for recalculating parity in this case is:

$$\text{parity}=\text{“NV-value for broken disk”}+\text{“on-disk values for all non-broken disks”}$$

In the present example that becomes:

parity=NV(Data **0**)+Data **1**+Data **2**=4+7+1=12

In effect, the parity is being updated so as to restore the broken disk to the value stored for it in the NV-RAM. In this particular example, the new value for parity happens to match the old value. If other data blocks besides data **0** were also being updated, and if one of them reached disk before the system failure, then the new parity value would not match the old.

Referring to FIG. **10**C, at time T_D, data disk **1** malfunctions due to the system failure between times T_C and T_D. This case is handled by reading the parity and data disk **0** values from NV-RAM in step **1114** and writing them to disk (indicated by a box). Thus, the change to data disk **0** is overwritten, but the stripe is returned to a consistent state. Data disk **1** is indicated as being broken at times T_D and T_E by enclosing its value in parentheses. Thus, the value of broken data disk **1** is correctly computed to be 7 (indicated by a box) at time T_D as follows:

Data **1**=NV(Parity)−NV(Data **0**)−Data **2**=12−4−1=7, (14)

where NV(Parity) and NV(Data **0**) are the values for parity and data block **0** stored in NV-RAM. At time T_E, NV-RAM is cleared. Thus, in FIG. **10**C, the stripe is maintained in a consistent state by the present invention although data disk **1** (a broken disk **2** is handled similarly) malfunctions after the system failure between times T_C and T_D.

This case can also be addressed by first calculating the old contents of the broken disk as follows:

*D***1**−calc=NV-parity−"*NV* values for disks being updated".−"on-disk values of data disks not being updated".

A new parity value is calculated based on:

parity="*D***1**−calc from stepabove"+on-disk values for all no-busted data disks".

Simultaneous System and Disk Failure with Parity by Recalculation

In parity by recalculation, the data blocks that are not being updated are first read from disk, and then parity is recalculated based on these values combined with the new data about to be written. This is typically used in cases where multiple data blocks are being updated at once, because it is more efficient than parity by subtraction in those cases. For simplicity, in the present example, only one block is updated. The techniques shown apply for updates of any number of blocks.

FIGS. **12**A–**12**C are timing diagrams for parity by recalculation. For times T_A, T_B and T_C, they are all identical.

At time T_A in step **1109**, blocks D**1** and D**2** are read from disk. In step **1110**, the system computes the new parity based on the new data for disk **0** along with the data just read from disks **1** and **2**.

At time T_B in step **1111**, blocks D**1** and D**2** are written into NV-RAM, along with an indication of the stripe to which they belong.

At time T_C, during step **1112**, the new value "2" is written to disk **0**. In the normal case, the parity block would also have been written during step **1112**, and there would be no corruption.

In the present example, there is a system failure in combination with a disk failure. When the system reboots after a system failure, execution begins at step **1101**. Because there is a failure, the decision block returns true (Yes) and continues at step **1102** and performs the necessary steps to recover the RAID sub-system based on the contents of NV-RAM.

FIG. **12**A shows the case where the parity disk fails. In this case, nothing needs to be done. There is no possibility of data loss, because no data disks have failed.

FIG. **12**B shows the case where the disk being updated fails. Note that at time T_D, the calculated value for disk **0** is incorrect. In general, if multiple blocks are being updated, there is not enough information in NV-RAM to reconstruct the lost data block. This is because with parity by recalculation, it is the data that is not being updated that is loaded into NV-RAM. The data on the failed disk is not saved anywhere.

In this case, the present invention computes a new parity value that sets the contents of the failed disk to zero. The general equation for this is:

parity=sum of non-broken disks

And in this example that is:

parity=*D***1**+*D***2**=7+1=8

At time T_E, the new parity value is written, and at time T_F, the NV-RAM values for D**1** and D**2** are cleared.

With a prior-art file system that writes new data in the same location as old data, zeroing out a data block would be unacceptable. But with WAFL, which always writes new data to unused locations on disk, zeroing a block that was being written has no harmful effect, because the contents of the block were not part of the file system.

FIG. **12**C shows the case where the disk not being updated fails. Note that at time T_D, the calculated value for disk **1** is incorrect. The equation to recalculate parity is:

parity="NV-RAM value for failed disk"+"on-disk values for non-failed disks:"

In the present example, that is:

parity=NV(*D***1**)+*D***0**+*D***2**=7+2+1=10

At time T_E, the new parity value is written, and at time T_F, the NV-RAM values for D**1** and D**2** are cleared.

In this manner, a method and apparatus are disclosed for providing error correction for an array of disks using non-volatile random access memory (NV-RAM).

What is claimed is:

1. A method for providing parity correction for a RAID array in a computer system after a system failure, including steps of:

maintaining information identifying dirty stripes, said dirty stripes having possibly inconsistent parity;

storing into non-volatile memory, from stripes to which data is going to be written, blocks of those stripes;

after said system failure, identifying stripes in response to said information; and

for each said stripe so identified, correcting said possibly inconsistent parity using said blocks stored into said non-volatile memory;

wherein said step of correcting includes steps of reading data blocks of said each said stripe necessary to recomputed a correct parity for said stripe, recomputing said correct parity, and writing said correct parity to said stripe; and

wherein said information includes a bitmap and including the further step of, when said bitmap is sufficiently full and after all pending writes are successfully completed, clearing said bitmap.

2. The method of claim 1, wherein said step of identifying includes the step of rebooting.

3. The method of claim 1, wherein said information includes stripe number information.

4. The method of claim 1, wherein said information is stored on said non-volatile memory.

5. The method of claim 1, wherein said information includes a list of dirty stripes.

6. The method of claim 1, wherein each bit in said bitmap is associated with a group of one or more stripes.

7. The method of claim 1, wherein said RAID array includes one or more write caching disks, each said one or more write caching disks having an internal cache and including the further step of, prior to said step of clearing said bitmap, flushing each said internal cache.

8. A method as in claim 1, wherein said blocks that are stored from said stripes into said non-volatile memory are those blocks to which said data is going to be written or those blocks to which said data is not going to be written.

9. An apparatus for providing parity correction for a RAID array in a computer system after a system failure, said apparatus including:

a maintenance mechanism configured to maintain information identifying dirty stripes, said dirty stripes having possibly inconsistent parity;

an identification mechanism configured to identify stripes in response to said information, said identification apparatus activated in response to check on reboot after a system failure;

a storage mechanism configured to store into non-volatile memory, from stripes to which data is going to be written, blocks of those stripes; and

a correction mechanism configured to correct said possibly inconsistent parity for each of said stripes identified by said identification mechanism, said correction mechanism using said blocks stored into said non-volatile memory to correct said possibly inconsistent parity;

wherein said correction mechanism includes a read mechanism configured to read data blocks of said each stripe necessary to recompute a correct parity for said each stripe, a recompute mechanism configured to recompute said correct parity responsive to said reading, and a write mechanism configured to write said correct parity to said stripe responsive to said recomputing; and

wherein said information includes a bitmap and further including a clear mechanism configured to clear said bitmap when said bitmap is sufficient full and after all pending writes are successfully completed.

10. The apparatus of claim 9, wherein said information includes stripe number information.

11. The apparatus of claim 9, wherein said information is stored on said non-volatile memory.

12. The apparatus of claim 9, wherein said information includes a list of dirty stripes.

13. The apparatus of claim 9, wherein each bit in said bitmap is associated with a group of one or more stripes.

14. The apparatus of claim 9, wherein said RAID array includes one or more write caching disks, each one or more write caching disks having an internal cache and further including a flush mechanism configured to flush each said internal cache.

15. An apparatus as in claim 9, wherein said blocks that are stored from said stripes into said non-volatile memory are those blocks to which said data is going to be written or those blocks to which said data is not going to be written.

16. A memory storing information including instructions, the instructions executable by a processor to provide parity correction for a RAID array in a computer system after a system failure, the instruction comprising the steps of:

maintaining information identifying dirty stripes, said dirty stripes having possibly inconsistent parity;

storing into non-volatile memory, from stripes to which data is going to be written, blocks of those stripes;

after said system failure, identifying stripes in response to said information; and

for each said stripe so identified, correcting said possibly inconsistent parity using said blocks stored into said non-volatile memory;

wherein said step of correcting includes steps of reading data blocks of said each said stripe necessary to recompute a correct parity for said each stripe, recomputing said correct parity, and writing said correct parity to said stripe; and

wherein said information includes a bitmap and the instructions includes the further step of, when said bitmap is sufficiently full and after all pending writes are successfully completed, clearing said bitmap.

17. The memory of claim 16, wherein said step of identifying includes the step of rebooting.

18. The memory of claim 16, wherein said information includes stripe number information.

19. The memory of claim 16, wherein said information is stored on said non-volatile memory.

20. The memory of claim 16, wherein said information includes a list of dirty stripes.

21. The memory of claim 16, wherein each bit in said bitmap is associated with a group of one or more stripes.

22. The memory of claim 16, wherein said RAID array includes one or more write caching disks, each said one or more write caching disks having an internal cache and including the further step of, prior to said step of clearing said bitmap, flushing each said internal cache.

23. A memory as in claim 16, wherein said blocks that are stored from said stripes into said non-volatile memory are those blocks to which said data is going to be written or those blocks to which said data is not going to be written.

24. A computer program product, including:

a computer usable storage medium having computer readable code embodied therein for causing a computer to provide parity correction for a RAID array in a computer system after a system failure, said computer readable code including:

computer readable program code configured to cause said computer to maintain information identifying dirty stripes, said dirty stripes having possibly inconsistent parity;

computer readable program code configured to cause said computer to store into non-volatile memory, from stripes to which data is going to be written, blocks of those stripes;

computer readable program code configured to cause said computer to identify stripes in response to said information, activated in response to check on reboot after a system failure;

computer readable program code configure to cause said computer to correct said possibly inconsistent parity using said blocks stored into said non-volatile memory for each of said stripes identified by said computer readable program code configured to cause said computer to identify stripes;

wherein said computer readable program code to correct said possibly inconsistent parity includes computer readable program code configured to read data blocks of said each said stripe necessary to recompute a correct parity for said each stripe, to recompute said correct parity, and to write said correct parity to said stripe; and

wherein said information includes a bitmap and the computer usable storage medium includes the further computer readable program code configured to clear said bitmap when said bitmap is sufficiently full and after all pending writes are successfull completed.

25. The computer program product of claim **24**, wherein said blocks that are stored from said stripes into said non-volatile memory are those blocks to which said data is going to be written or those blocks to which said data in not going to be written.

* * * * *