(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 101501623 B
(45)授权公告日 2013.03.06

(21)申请号 200780025208.7
(22)申请日 2007.05.03
(30)优先权数据
60/797,127 2006.05.03 US
(85)PCT申请进入国家阶段日
2009.01.04
(86)PCT申请的申请数据
PCT/US2007/068139 2007.05.03
(87)PCT申请的公布数据
WO2007/128005 EN 2007.11.08
(73)专利权人 数据机器人技术公司
地址 美国加利福尼亚州
(72)发明人 朱丽安·M·特里
尼尔·A·克拉克森
杰弗里·S·巴拉尔
(74)专利代理机构 中原信达知识产权代理有限责任公司 11219
代理人 张焕生 安翔
(51)Int.Cl.
*G06F 3/06*(2006.01)
*G06F 17/30*(2006.01)
(56)对比文件
US 6606651 B1,2003.08.12, 全文.
GB 2411746 A,2005.09.07, 全文.
US 2004/0078641 A1,2004.04.22, 全文.
审查员 欧阳琦

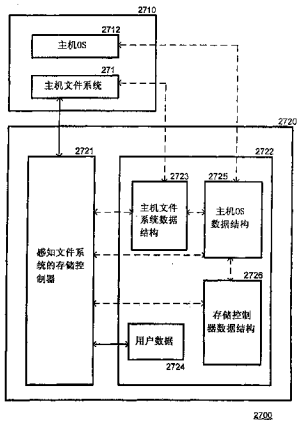
权利要求书 3 页 说明书 50 页 附图 31 页

(54)发明名称

感知文件系统的块存储系统、装置和方法

(57)摘要

一种感知文件系统的存储系统，其定位和分析主机文件系统数据结构，以便确定所述主机文件系统的存储使用。为此，所述存储系统可能对操作系统分区进行定位，对所述操作系统分区进行解析以定位其数据结构，并且对操作系统数据结构进行解析以定位主机文件系统数据结构。所述存储系统基于所述主机文件系统的存储使用对数据存储进行管理。所述存储系统能够使用存储使用信息来识别不再被主机文件系统所使用的存储区域并且回收那些区域用作其它数据存储容量。而且，所述存储系统能够识别主机文件系统所存储的数据的类型，并且基于数据类型对数据存储进行管理，诸如基于数据类型选择用于数据的存储布局和/或编码方案。




CN 101501623 B

1. 一种在与主机计算机进行通信的块级存储系统中存储数据块的方法，所述主机计算机具有主机文件系统，所述块级存储系统具有被配置来处理来自所述主机文件系统的存储请求的存储控制器，所述方法包括：

定位在所述块级存储系统中所存储的用于所述主机文件系统的主机文件系统数据结构；

对所述主机文件系统数据结构进行分析以确定所述主机文件系统有关所述数据块的存储使用；和

基于所确定的所述主机文件系统的存储使用管理所述数据块的物理存储，

其中所述定位、分析以及管理步骤由所述存储控制器在所述块级存储系统中执行。

2. 如权利要求 1 所述的方法，其中，分析所述主机文件系统数据结构包括识别与所要存储的数据相关联的数据类型，并且其中，管理物理存储包括使用基于所述数据类型所选择的存储方案存储所述数据，由此根据不同的存储方案存储具有不同数据类型的数据。

3. 如权利要求 2 所述的方法，其中使用基于所述数据类型所选择的存储方案存储所述数据包括：

使用基于所述数据类型所选择的存储布局存储所述数据。

4. 如权利要求 3 所述的方法，其中使用基于所述数据类型所选择的存储布局存储所述数据包括：

存储频繁访问的数据以便提供增强的访问性能。

5. 如权利要求 4 所述的方法，其中存储频繁访问的数据以便提供增强的访问性能包括：

将所述频繁访问的数据以未压缩的形式存储在连续存储中。

6. 如权利要求 3 所述的方法，其中使用基于所述数据类型所选择的存储布局存储所述数据包括：

存储不频繁访问的数据以便提供增强的存储效率。

7. 如权利要求 6 所述的方法，其中存储不频繁访问的数据以便提供增强的存储效率包括：

使用数据压缩和非连续存储中的至少一个存储所述不频繁访问的数据。

8. 如权利要求 2 所述的方法，其中使用基于所述数据类型所选择的存储方案存储所述数据包括：

使用基于所述数据类型所选择的编码方案存储所述数据。

9. 如权利要求 8 所述的方法，其中所述编码方案包括以下的至少一个：

数据压缩；和

加密。

10. 如权利要求 1 所述的方法，其中定位在所述块级存储系统中所存储的用于所述主机文件系统的主机文件系统数据结构包括：

维护分区表；

对所述分区表进行解析以定位操作系统分区；

对所述操作系统分区进行解析以识别所述操作系统并定位操作系统数据结构；和

对所述操作系统数据结构进行解析以识别所述主机文件系统并定位所述主机文件系

统数据结构。

11. 如权利要求 10 所述的方法，其中所述操作系统数据结构包括超级块，并且其中对所述操作系统数据结构进行解析包括对所述超级块进行解析。

12. 如权利要求 1 所述的方法，其中对所述主机文件系统数据结构进行分析包括：

制作主机文件系统数据结构的工作副本；和

对所述工作副本进行解析。

13. 如权利要求 1 所述的方法，其中，分析所述主机文件系统数据结构包括识别已被所述主机文件系统释放的数据块，并且其中，管理存储包括再使用所释放的块以便有效扩展所述块级存储系统的数据存储容量。

14. 一种在与主机计算机进行通信的块级存储系统中存储数据块的系统，所述主机计算机具有主机文件系统，所述块级存储系统具有被配置来处理来自所述主机文件系统的存储请求的存储控制器，所述系统包括：

用于定位在所述块级存储系统中所存储的用于所述主机文件系统的主机文件系统数据结构的装置；

用于对所述主机文件系统数据结构进行分析以确定所述主机文件系统有关所述数据块的存储使用的装置；和

用于基于所确定的所述主机文件系统的存储使用管理所述数据块的物理存储的装置，

其中所述定位、分析以及管理步骤由所述存储控制器在所述块级存储系统中执行。

15. 如权利要求 14 所述的系统，其中，用于分析所述主机文件系统数据结构的装置包括用于识别与所要存储的数据相关联的数据类型的装置，并且其中，用于管理物理存储的装置包括用于使用基于所述数据类型所选择的存储方案存储所述数据，由此根据不同的存储方案存储具有不同数据类型的数据的装置。

16. 如权利要求 15 所述的系统，其中用于使用基于所述数据类型所选择的存储方案存储所述数据的装置包括：

用于使用基于所述数据类型所选择的存储布局存储所述数据的装置。

17. 如权利要求 16 所述的系统，其中用于使用基于所述数据类型所选择的存储布局存储所述数据的装置包括：

用于存储频繁访问的数据以便提供增强的访问性能的装置。

18. 如权利要求 17 所述的系统，其中用于存储频繁访问的数据以便提供增强的访问性能的装置包括：

用于将所述频繁访问的数据以未压缩的形式存储在连续存储中的装置。

19. 如权利要求 16 所述的系统，其中用于使用基于所述数据类型所选择的存储布局存储所述数据的装置包括：

用于存储不频繁访问的数据以便提供增强的存储效率的装置。

20. 如权利要求 19 所述的系统，其中用于存储不频繁访问的数据以便提供增强的存储效率的装置包括：

用于使用数据压缩和非连续存储中的至少一个存储所述不频繁访问的数据的装置。

21. 如权利要求 15 所述的系统，其中用于使用基于所述数据类型所选择的存储方案存储所述数据的装置包括：

用于使用基于所述数据类型所选择的编码方案存储所述数据的装置。

22. 如权利要求 21 所述的系统，其中所述编码方案包括以下的至少一个：

数据压缩；和

加密。

23. 如权利要求 14 所述的系统，其中用于定位在所述块级存储系统中所存储的用于所述主机文件系统的主机文件系统数据结构的装置包括：

用于维护分区表的装置；

用于对所述分区表进行解析以定位操作系统分区的装置；

用于对所述操作系统分区进行解析以识别所述操作系统并定位操作系统数据结构的装置；和

用于对所述操作系统数据结构进行解析以识别所述主机文件系统并定位所述主机文件系统数据结构的装置。

24. 如权利要求 23 所述的系统，其中所述操作系统数据结构包括超级块，并且其中用于对所述操作系统数据结构进行解析的装置包括用于对所述超级块进行解析的装置。

25. 如权利要求 14 所述的系统，其中用于对所述主机文件系统数据结构进行分析的装置包括：

用于制作主机文件系统数据结构的工作副本的装置；和

用于对所述工作副本进行解析的装置。

26. 如权利要求 14 所述的系统，其中，用于分析所述主机文件系统数据结构的装置包括用于识别已被所述主机文件系统释放的数据块的装置，并且其中，用于管理存储的装置包括用于再使用所释放的块以便有效扩展所述块级存储系统的数据存储容量的装置。

# 感知文件系统的块存储系统、装置和方法

[0001] 优先权

[0002] 该PCT申请要求于2006年5月3日以Julian M.Terry、Neil A.Clarkson和Geoffrey S.Barrall的名义提交的题为Filesystem-Aware BlockStorage System, Apparatus,and Method美国临时专利申请No.60/797127的优先权。

[0003] 该申请还涉及于2005年11月4日以Geoffrey S.Barrall的名义提交的题为Dynamically Expandable and Contractible Fault-Tolerant StorageSystem Permitting Variously Sized Storage Devices and Method的美国专利申请No.11/267938,其要求于2004年11月5日提交的美国临时专利申请No.60/625495和于2005年9月20日提交的美国临时专利申请No.60/718768的优先权。

[0004] 以上所有专利申请均在此全文引用以供参考。

## 技术领域

[0005] 本发明涉及数字数据存储系统和方法,更具体地涉及提供容错存储的系统和方法。

## 背景技术

[0006] 已知现有技术依照根据各种RAID(独立盘片冗余阵列)协议的任何一种的模式来提供冗余盘片存储器。典型地,使用RAID模式的盘片阵列是需要由经验丰富的信息技术人员来管理的复杂结构。而且在许多使用RAID模式的阵列设计中,如果所述阵列中的各盘片驱动器是非统一容量的,那么该设计可能不能够使用超过该阵列中最小驱动器容量的驱动器上的任何容量。

[0007] 标准RAID系统的一个问题是盘面损坏可能发生在盘片阵列中不经常使用的区域。在另一个驱动器出现故障的情况下,并不总是能确定发生了损坏。在这种情况下,当所述RAID阵列重构所述故障驱动器时,损坏的数据可能被传播和保存。

[0008] 在许多存储系统中,将以就绪状态(ready state)维持备用存储设备,使得其在另一个存储设备出现故障时可以被使用。这种备用存储设备通常被称作“热后备”。所述热后备在存储设备的正常操作期间不用于存储数据。当运行的存储设备出现故障时,该故障存储设备被该热后备逻辑上替换,而且要移动数据或者通过其他方式在所述热后备上再产生数据。当修复或者更换所述故障存储设备时,典型地要移动数据或者通过其他方式在(重新)运行的存储设备上重新产生这些数据,而且使所述热后备脱机,以准备好在另一个故障事件中使用。热后备盘片的维护通常是复杂的,并因此通常由经验丰富的管理者进行处理。热后备盘片同样代表着附加费用。

[0009] 一般而言,当主机文件系统向存储系统写数据块时,所述存储系统为数据分配存储块并且更新其数据结构来指示处于使用中的存储块。从这一点来看,即使主机文件系统随后停止使用它的块,存储系统也认为存储块处于使用中。

[0010] 主机文件系统通常使用位图来跟踪其使用的盘片块(disk block)。在卷创建之

后，位图通常将立刻典型地通过使得所有位清空来指示大多数块空闲。当文件系统被使用时，主机文件系统将通过使用其空闲块的位图来独立分配块。

[0011] 当主机文件系统将一些块释放回其空闲池中时，其简单地清空在其空闲块的位图中的对应位。在存储系统上，这表示为对恰好包含主机的空闲块位图的某部分的聚簇的写入，并且可能是对日志文件的写入；几乎肯定没有到实际聚簇的输入/输出(I/O)是本身空闲的。如果主机文件系统在增强的安全模式下运行，则由于主机对当前盘片上的数据进行覆盖从而减少了陈旧聚簇内容可被攻击者读取的机会，所以可能有对游离块的I/O，但是无法将这样的写入识别为删除过程的一部分。因此，存储设备无法将主机文件系统正在使用的块与先前所使用且随后被标记为空闲的块区分开来。

[0012] 存储系统无法识别游离块会导致多种消极后果。例如，存储系统会明显过度报告正在使用的存储量并且会过早用尽存储空间。

## 发明内容

[0013] 根据本发明的一个方面，提供了一种通过在主机文件系统的控制下存储数据的块级存储系统来存储数据的方法。该方法涉及定位在块级存储系统中所存储的用于主机文件系统的主机文件系统数据结构；通过对主机文件系统数据结构进行分析以识别与所要存储的数据相关联的数据类型；以及使用基于所述数据类型所选择的存储方案存储所述数据，由此使用基于所述数据类型所选择的不同存储方案能够存储具有不同数据类型的数据。

[0014] 根据本发明的另一方面，提供了一种在主机文件系统的控制下存储数据的块级存储系统。该系统包括：块级存储，其中存储用于主机文件系统的主机文件系统数据结构；以及可操作地耦合到块级存储的存储控制器，其用于定位块级存储中所存储的主机文件系统数据结构，对主机文件系统数据结构进行分析以识别与所要存储的数据相关联的数据类型，并使用基于所述数据类型所选择的存储方案存储所述数据，由此使用基于所述数据类型所选择的不同存储方案能够存储具有不同数据类型的数据。

[0015] 在各个可选实施例中，可以使用基于数据类型所选择的存储布局和/或编码方案来存储数据。例如，可以存储频繁访问的数据以便提供增强的访问性能(例如，以未压缩的形式和以连续存储的形式)，而存储不频繁访问的数据以便提供增强的存储效率(例如，使用数据压缩和/或非连续存储)。额外地或替选地，根据数据类型数据可以是压缩的和/或加密的。

[0016] 在各个可选实施例中，可通过以下步骤来定位所述主机文件系统数据结构：对分区表进行维护；对分区表进行解析以定位操作系统分区；对操作系统分区进行解析以识别所述操作系统并定位操作系统数据结构；并且对所述操作系统数据结构进行解析来识别主机文件系统并定位主机文件系统数据结构。所述操作系统数据结构可包括超级块，在这种情况下，对操作系统数据结构进行解析可包括解析所述超级块。可通过制作主机文件系统数据结构的工作副本并对所述工作副本进行解析来解析所述主机文件系统数据结构。

## 附图说明

[0017] 通过参考下列附图参照下列详细说明，本发明上述特征将变得更加容易理解，其中：

[0018] 图 1 示出本发明的实施例，其中将对象解析为一系列存储块。

[0019] 图 2 是说明在相同实施例中大块 (chunk) 的容错存储模式如何根据附加更多存储器而动态改变。

[0020] 图 3 说明本发明另一实施例，在使用不同大小存储设备构造的存储系统上按不同容错模式的大块的存储。

[0021] 图 4 说明本发明的另一个实施例，其中指示器状态用于警告无效存储使用和低等级容错。

[0022] 图 5 是根据本发明实施例的所述数据存储、检索和再布局中使用的功能模块的框图。

[0023] 图 6 表示在包含两个以上驱动器的阵列中使用镜像的示例。

[0024] 图 7 表示使用不同布局方案以存储其数据的一些示例性存储区。

[0025] 图 8 表示用于实施后备卷 (sparse volume) 的查找表。

[0026] 图 9 表示根据本发明示例性实施例的状态指示器，其用于具有可用存储空间并按容错方式操作的示例性阵列。

[0027] 图 10 表示根据本发明示例性实施例的状态指示器，其用于不具有足够空间以维护冗余数据存储并必须增加更大空间的示例性阵列。

[0028] 图 11 表示根据本发明示例性实施例的状态指示器，其用于在故障情况下不能够维护冗余数据的示例性阵列。

[0029] 图 12 表示根据本发明示例性实施例的示例性阵列的状态指示器，其中存储设备已经出现故障。用存储设备填充插槽 B、C、和 D。

[0030] 图 13 示出的模块层次表示示例性实施例的不同软件层以及它们彼此如何相关。

[0031] 图 14 表示根据本发明的示例性实施例的聚簇存取表如何用于访问存储区中的数据聚簇。

[0032] 图 15 表示根据本发明的示例性实施例的日志表更新。

[0033] 图 16 表示根据本发明的示例性实施例的驱动器布局。

[0034] 图 17 示出了根据本发明的示例性实施例的存储区 0 的布局和其他存储区如何被参照。

[0035] 图 18 说明了根据本发明的示例性实施例的读差错处理。

[0036] 图 19 说明了根据本发明的示例性实施例的写差错处理。

[0037] 图 20 是根据本发明的示例性实施例的逻辑流程图，其说明了通过差错管理器备份坏区域。

[0038] 图 21 是根据本发明的示例性实施例的示意框图，其表示存储阵列的相关组件。

[0039] 图 22 是根据本发明的示例性实施例的逻辑流程图，其表示管理虚拟热后备的示例性逻辑。

[0040] 图 23 是根据本发明的示例性实施例的逻辑流程图，其说明确定每个可能盘片故障的再布局情况的示例性逻辑，如图 22 的框 2102。

[0041] 图 24 是根据本发明的示例性实施例的逻辑流程图，其表示调用虚拟热后备功能的示例性逻辑。

[0042] 图 25 是根据本发明的示例性实施例的逻辑流程图，其表示自动再配置一个或多

个剩余驱动器以恢复数据容错的示例性逻辑，如图 24 的框 2306。

[0043] 图 26 是根据本发明的示例性实施例的逻辑流程图，其表示用于升级存储设备的示例性逻辑。

[0044] 图 27 是根据本发明的示例性实施例的计算机系统的概念框图。

[0045] 图 28 是根据本发明的示例性实施例的用于感知文件系统的存储控制器的高级别逻辑流程图。

[0046] 图 29 是根据本发明的示例性实施例的用于定位主机文件系统数据结构的逻辑流程图。

[0047] 图 30 是根据本发明的示例性实施例的用于回收未使用的存储空间的逻辑流程图。

[0048] 图 31 是根据本发明的示例性实施例的用于基于数据类型对用户数据的存储进行管理的逻辑流程图。

[0049] 图 32 是表示根据本发明的示例性实施例的回收器 (scavenger) 的相关部件的示意性框图。

[0050] 图 33 是根据本发明的示例性实施例的用于定位主机文件系统的位图的伪代码。

[0051] 图 34 是根据本发明的示例性实施例的用于 BBUM 的高级别伪代码。

[0052] 图 35 是根据本发明的示例性实施例的用于创建新分区的 LBA 0 更新的同步处理的高级别伪代码。

[0053] 图 36 是根据本发明的示例性实施例的用于（重新）格式化分区的 LBA 0 更新的同步处理的高级别伪代码。

[0054] 图 37 是根据本发明的示例性实施例的用于删除分区的 LBA 0 更新的同步处理的高级别伪代码。

[0055] 图 38 是根据本发明的示例性实施例的用于异步任务的高级别伪代码。

## 具体实施方式

[0056] 定义。如在本说明书和权利要求书中使用的，除非上下文另有要求，否则下面术语具有如下所解释的意思。

[0057] 对象的“大块 (chunk)”是对象的提取片，由所使用的任何物理存储独立地形成，且典型地是对象的固定数量的连续字节。

[0058] 数据存储的容错“模式 (pattern)”是在一个或者多个存储设备上冗余地分布数据的特定方式，并且除了其他的，可以是：镜像 (mirroring，例如按类似 RAID1 的方式）、条带化 (striping，例如按类似 RAID5 的方式）、RAID6、双奇偶校验、对角线奇偶校验、低密度奇偶校验码、涡轮码、或者其他冗余方案或者这些冗余方案的组合。

[0059] 当给定大块产生通常与其他任何大块的散列号都不同的散列号时，除非当其他大块具有与该给定大块相同的数据内容时，该给定大块的散列号是“唯一的 (unique)”。即，当两个大块的内容不相同时，该两个大块通常将具有不同散列号。如下面所要进一步详细描述的，在本上下文中，术语“唯一的”用于覆盖由那些偶尔对不同的大块产生相同散列号的散列函数所产生的散列号，因为散列函数通常不能完美地在对不同的大块产生不同的号。

[0060] “区域 (region)”是存储介质（例如硬盘驱动器）上的一组连续物理块。

[0061] “存储区 (zone)”是由两个或以上的区域组成的。组成存储区的各区域通常不需要是连续的。如在下面描述的示例性实施例中，存储区存储相当于 1GB 的数据或者控制信息。

[0062] “聚簇 (cluster)”是存储区中的单元尺寸，并表示压缩单位（下面论述）。在如下面所述的示例性实施例中，聚簇是 4KB（即八个 512 字节的扇区）并且实质上等同于大块。

[0063] “冗余集 (redundant set)”是为一组数据提供冗余的一组扇区 / 聚簇。

[0064] “备份区域 (backing up a region)”涉及将一个区域的内容复制到另一个区域。

[0065] 存储设备的“第一对”和“第二对”可以包括公共存储设备。

[0066] 存储设备的“第一组多个”和“第二组多个”可以包括一个或者多个公共存储设备。

[0067] 存储设备的“第一布置”和“第二布置”或者“不同布置”可以包括一个或多个公共存储设备。

[0068] 在本发明的实施例中，感知文件系统的存储系统对主机文件系统的数据结构进行分析以便确定主机文件系统的存储使用。例如，块存储设备可对主机文件系统的数据结构进行解析以确定诸如已用块、未用块和数据类型等内容。所述块存储设备基于所述主机文件系统的存储使用对物理存储进行管理。

[0069] 这样的感知文件系统的块存储系统能够进行与数据的物理存储相关的智能决策。例如，所述感知文件系统的块存储设备能够识别已经被主机文件系统释放的块并且再使用被释放的块，以便有效扩展系统的数据存储容量。被释放的块的这种再使用，其在此后被称作“回收 (scavenging)”或“垃圾收集”，在实现虚拟存储时可以是特别有用的，其中主机文件系统被配置以比实际的物理存储容量更大的存储。所述感知文件系统的块存储设备还能够识别文件系统所存储的对象的数据类型，并且基于数据类型使用不同的存储方案来存储所述对象（例如，频繁访问的数据可以未压缩的方式并在连续块中存储，而不频繁访问的数据可被压缩和 / 或在非连续块中存储；基于数据类型可对不同对象应用诸如数据压缩和加密之类的不同编码方案）。

[0070] 所述感知文件系统的块存储设备通常将支持预先确定的文件系统集合，其充分“理解”所述预先确定的文件系统集合的内部工作以定位和利用潜在的数据结构（例如，空闲块的位图）。为了确定文件系统类型（例如，NTFS、FAT、ReiserFS、ext3），所述感知文件系统的块存储设备典型地对分区表进行解析以定位操作系统 (OS) 分区，并接着对所述 OS 分区进行解析以定位主机文件系统的超级块，并由此识别文件系统类型。一旦知道了文件系统类型，所述感知文件系统的块存储能够对所述超级块进行解析以找到用于所述主机文件系统的空闲块的位图，并且由此能够对所述空闲块的位图进行解析来识别已用和未用的块。

[0071] 为了检测数据结构（例如，空闲块的位图）随时间的变化，所述感知文件系统的块存储设备可周期性地制作数据结构的副本（例如，在私有的、非冗余存储区中），并随后将数据结构的当前活动版本与较早制作的副本进行比较来检测变化。例如，能够识别从已分配变换到空闲的任意位图项，允许将垃圾收集操作准确引导至作为良好回收候选的聚簇。随着每个位图聚簇得到处理，可利用当前副本替代历史副本，以保持位图操作的滚动历史。空闲块的位图的副本可随时间变为暂时不连续的聚簇的拼凑 (patchwork)，但是由于当前副本被用来分配空闲项，所以这不会导致任何问题。

[0072] 此后参考存储阵列系统描述示例性实施例。

[0073] 图 1 是本发明实施例的说明，其中，将对象 (object) 解析成一系列用于存储的大块，在本例中对象是文件 (file)。开始，文件 11 被传递到存储软件，在其中被指定为对象 12 并被分配一个唯一对象标识号，在此是 #007。在对象表 13 中形成新项 131，用来表示这个新对象的分配。现在该对象被解析成数据“大块”121、122、和 123，它们是固定长度的对象段。每个大块都经过散列算法，该算法返回大块的唯一散列号。以后这个算法可以应用于检索块，并且结果与原始散列相比较以确保重试块与存储的相同。每个大块的所述散列号按对象 132 的项行存储在对象表 13 中，以便于以后所述完成对象可以通过所述各大块的集合进行检索。

[0074] 同样在图 1 中，所述大块散列现在与大块表 14 中现有项进行比较。匹配现有项 141 的任何散列都已经被存储并且因此不采取任何行动（即数据不会被再次存储，导致对象的自动压缩）。新散列（在大块表 14 中没有对应项的散列）被输入大块表 141。然后大块中的数据以最有效的容错存储方式存储在可用的存储设备 151、152 和 153 上。这种方法可导致所述大块数据例如按镜像方式存储在包括一个或者两个设备的存储系统上，或者按奇偶校验条带化存储在具有两个以上存储设备的系统。该数据将存储在存储设备上的物理位置 1511、1521 和 1531，而 且这些位置和位置编号将存储在大块表列 143 和 142 中，使得以后可以定位和检索大块的所有物理部分。

[0075] 图 2 说明在相同实施例中大块的容错存储模式可以如何根据增加更多添加更多存储而动态改变。特别地，图 2 示出了一旦附加存储被添加到整个系统，大块的物理存储如何在所述存储设备上按新模式来布局。在图 2(a) 中，所述存储系统包括两个存储设备 221 和 222，而大块数据在位置 2211 和 2221 被物理镜像到所述两个存储设备上以提供容错。在图 2(b) 中，添加第三存储设备 223，以奇偶校验条带化方式存储所述大块成为可能，这种模式较之所述镜像模式是更有效的存储。所述大块按该新模式布局在三个物理位置 2311、2321、和 2331，占用更少比例的可用存储。更新所述大块表 21 以表现在三个位置 212 的新布局，并且还在 213 记录新的大块物理位置 2311、2321、和 2331。

[0076] 图 3 示出根据本发明实施例的成熟存储系统，其已经运行一段时间。该图说明了各大块可以如何随时间物理地存储在存储容量变化的存储设备上。该图示出包括 40GB 的存储设备 31、80GB 的存储设备 32 和 120GB 的存储设备 33 的存储系统。最初，各大块按容错条带化模式 34 来存储，直到 40GB 存储设备 31 变满。然后，由于缺乏存储空间，新数据就按镜像模式 36 存储到 80GB 的存储设备 32 和 120GB 的存储设备 33 的可用空间上。一旦 80GB 的存储设备 32 满了，则新数据按单个盘片容错模式 37 来布局。尽管存储设备包括用于存储数据的单一池，如按块存储的数据本身已经按多种不同模式来存储。

[0077] 图 4 说明本发明的另一实施例，其中使用指示器状态来警告低效的存储使用和低级的容错。在图 4A 中，全部三个存储设备 41、42 和 43 具有空闲空间，而指示灯 44 是绿色以表示数据以有效和容错方式存储。在图 4B 中，40GB 的存储设备 41 已经变满了，因此新数据可按镜像模式 46 只能存储在具有剩余空闲空间的两个存储设备 42 和 43 上。为了表示数据仍然十分冗余但不能有效存储，指示灯 44 变成黄色。在图 4C 中，仅 120GB 的存储设备 43 具有空闲空间剩余，因此全部新数据可以按镜像模式仅存储在这一台设备 43 上。因为容错性不强并且系统严重地缺乏空间，因此指示灯 44 变红以指示需要增加更多存储。

[0078] 在一个替换实施例中，为阵列中每个驱动器 / 插槽提供指示器，例如，以三色灯形式（例如绿、黄、红）。在一个特定实施例中，所述灯用于照亮具有发光效果的盘片机柜的整个前面。控制这些灯，以不仅用于指示该系统的整体状态，还用于指示哪个驱动器 / 插槽需要加以注意（若有的话）。每个三色灯都可以处于至少四种状态：分别是关闭、绿色、黄色、红色。如果特定插槽为空且系统按充足存储和冗余在运行因此不需要在插槽中安装驱动器，则相应插槽的灯可以处于关闭状态。如果相应驱动器充足并且不需要替换，则特定插槽的灯可以置于绿色状态。如果系统运行在降级，则可以将特定插槽的灯置于黄色状态，以建议用较大驱动器替换相应驱动器。如果相应驱动器必须被安装或者替换，则特定插槽的灯可以置于红色状态。如果需要或者期望的话，可以指示附加状态，例如，通过使得灯在开启状态和关闭状态之间闪烁或者在两种不同颜色闪烁（例如在替换驱动器之后并且在进行数据重新布局时在红色和绿色之间闪烁）。下面说明示例性实施例的附加细节。

[0079] 当然，可以使用其他指示技术来指示系统状态和驱动器 / 插槽状态。例如，可以使用单个 LCD 显示器来指示系统状态，并且如果需要的话，可以指示需要注意的插槽号。同样，可以使用其他类型指示器（例如，系统的单个状态指示器（例如绿色 / 黄色 / 红色），还有每个插槽的插槽指示器或者灯）。

[0080] 图 5 是根据本发明实施例的数据存储、检索和重新布局中使用的功能模块框图，如上与图 1 至 3 相关的讨论。通信的入口和出口点是：对象接口 511，用于将对象传递给用于存储或者检索对象的系统；块接口 512，它使存储系统看来是一个大的存储设备以及 CIFS 接口 513，它使存储系统看来是 Windows 文件系统。当这些接口需要数据存储时，数据被传递到大块解析器 52，所述解析器 52 将数据分解为大块，并在对象表 512 中创建初始项（如上与图 1 相关的讨论）。这些大块然后被传递到散列码生成器 53，散列码生成器 53 产生每个大块的相关联散列码，并将其输入到对象表中，这样与每个对象相关联的各大块被列出 512（如上与图 1 相关的讨论）。大块散列号与大块表 531 中的项进行比较。在发现匹配时，该新的大块被废弃，因为它与已经存储在该存储系统中的某个大块相同。如果大块是新的，则在大块表 531 中为其建立新项，而且将散列的大块传递到物理存储管理器 54。该物理存储管理器以可能的最有效模式在可用存储设备 571、572、和 573 上存储该大块，并在大块表 531 中制作相应项以表示哪里已经发生该大块的物理存储，使得以后可以在 512 检索该大块的内容（如上与图 1 的相关讨论）。

[0081] 图 5 中由对象接口 511、块接口 512 或者 CIFS 接口 513 的数据检索通过向检索管理器 56 请求来执行，检索管理器查询对象表 521 以确定哪些大块包括该对象，然后从物理存储管理器 54 请求这些大块。所述物理存储管理器 54 查询大块表格 531 以确定所请求的大块存储在哪里，然后检索它们并将完成数据（对象）传递回检索管理器 56，检索管理器 56 向请求接口返回该数据。图 5 还包括容错管理器 (FTL)55，其不断扫描块表以确定大块是否以可能的最有效方式存储。（这可能由于添加和去除存储设备 571、572 和 573 而改变。）如果大块不是以可能的最有效方式存储，则 FTL 将请求物理存储管理器 54 创建该大块的新布局模式并更新大块表 531。这样全部数据对包括该阵列的若干存储设备按可能的最有效方式来存储（如上与图 2 和 3 相关的讨论）。

[0082] 下面提供本发明的示例性实施例的其他细节。

[0083] 数据布局方案——存储区 (zone)

[0084] 除了其他，存储区影响隐式冗余和存储在盘片上的实际数据的盘片再布局。存储区允许增加和改变附加布局方法而不影响存储区的用户。

[0085] 存储阵列在盘片上按称作存储区的虚拟分段来布局数据。存储区存储给定的和固定数量的数据（例如 1G 字节）。存储区可以驻留在单个盘片或者跨越一个或者多个驱动器。存储区的物理布局以特定于该存储区的形式提供冗余。

[0086] 图 6 示出在包含两个以上驱动器的阵列中使用镜像的示例。图 7 示出使用不同布局方案来存储它们的数据的一些示例存储区。该图假定存储区存储 1GB 数据。注意以下几点：

[0087] i) 跨多个驱动器的存储区不需要在整个集合中使用到驱动器的相同偏移。

[0088] ii) 单个驱动器镜像需要 2G 的存储量来存储 1G 的数据。

[0089] iii) 双驱动器镜像需要 2G 的存储量来存储 1G 的数据。

[0090] iv)3 驱动器条带化需要 1.5GB 的存储量来存储 1G 的数据。

[0091] v)4 驱动器条带化需要 1.33GB 的存储量来存储 1G 的数据。

[0092] vi) 存储区 A、存储区 B 等是任意的存储区名。在实际实施中每个存储区都用唯一编号来标识。

[0093] vii) 虽然通过该图隐含，但是存储区不是必须在盘片上连续的（见后面所述的区域）。

[0094] viii) 镜像被限于两个驱动器没有技术原因。例如，在三个驱动器的系统中，数据的 1 个副本可存储在 1 个驱动器上，且一半镜像数据可以存储在另外两个驱动器的每个上。同样，数据可跨三个驱动器来镜像，一半数据在两个驱动器的每个上以及一半镜像在其他两个驱动器上。

[0095] 数据布局方案——区域 (region)

[0096] 每个盘片都分割为一组相等尺寸的区域。区域的尺寸比存储区小得多，存储区由来自一个或者多个盘片的一个或者多个区域构成。为了有效使用盘片空间，区域的尺寸典型地是不同存储区尺寸和阵列所支持的不同盘片数量的公因数。在示例性实施例中，区域是存储区数据尺寸的 1/12。下面的表列出了根据本发明示例性实施例的各种布局的区域 / 存储区的数量和区域 / 盘片的数量。

[0097]

| 布局方法 | 区域 / 存储区的数量 | 区域 / 盘片的数量 |
|---|---|---|
| 1 驱动器镜像 | 24 | 24 |
| 2 驱动器镜像 | 24 | 12 |
| 3 驱动器条带化 | 18 | 6 |
| 4 驱动器条带化 | 16 | 4 |

[0098] 各区域可以标记为：使用、空闲或者损坏。当创建存储区时，选择来自适当盘的一组空闲区域并登记到表中。这些区域可以是任何任意顺序而且不需要在盘片上是连续的。当从存储区读取或者向其写入数据时，存取被重定向到适当区域。除了其他，这允许以灵活

和有效的方式发生数据再布局。随着时间的过去，不同尺寸的存储区将可能导致发生碎片(fragmentation)，使得许多盘片区太小而不能保持完整存储区。通过使用适当区域尺寸，碎片留下的所有间隙都将是至少一个区域的尺寸，这些小间隙易于重用，而不必重新分段整个盘片。

[0099] 数据布局方案——再布局

[0100] 为了便于实施，可以强制扩展和收缩的固定顺序。例如，如果突然增加两个驱动器，则存储区的扩展可以经过中间扩展，就像增加一个驱动器一样，然后执行第二扩展以结合第二个驱动器。或者，包含多个驱动器的扩展和收缩可以被原子化地处理，不需要中间步骤。

[0101] 在发生任何再布局之前，所需空间都必须是可用的。这应当在开始再布局之前计算，以确保不会发生不必要的再布局。

[0102] 数据布局方案——驱动器扩展

[0103] 下面描述根据本发明示例性实施例的从单驱动器镜像扩展为双驱动器镜像的一般过程：

[0104] i) 假定单驱动器镜像具有数据‘A’和镜像‘B’

[0105] ii) 在驱动器上分配 12 个区域以扩展存储区到‘C’

[0106] iii) 复制镜像‘B’到区域集‘C’

[0107] iv) 任何对已经复制的数据的写都必须镜像到‘C’中的适当位置

[0108] v) 当完成复制时，使用新布局类型更新存储区表并利用指向‘C’的指针来替换指向‘B’的指针

[0109] vi) 将构成‘B’的区域标记为空闲的。

[0110] 下面描述根据本发明示例性实施例的从双驱动器镜像扩展为具有奇偶校验的三驱动器条带化的一般过程：

[0111] i) 假定一个驱动器具有数据‘A’而第二驱动器具有镜像‘B’

[0112] ii) 为奇偶校验信息‘C’在第三驱动器上分配 6 个区域

[0113] iii) 使用‘A’的第一组 6 区域和‘B’的第二组 6 区域计算奇偶校验信息

[0114] iv) 在‘C’中放置奇偶校验信息

[0115] v) 任何对已经处理的数据的写都必须奇偶校验到‘C’中适当位置

[0116] vi) 当复制完成时，用新布局类型点表将存储区表更新为‘A’的前一半、‘B’和‘C’的后一半

[0117] vii) 标记‘A’的前一半和‘B’的前一半为空闲

[0118] 下面描述根据本发明示例性实施例的从三驱动器条带化扩展为具有奇偶校验的四驱动器条带化的一般过程：

[0119] i) 假定一个驱动器具有数据‘A’，第二驱动器具有数据‘B’而第三个具有奇偶校验‘P’

[0120] ii) 对条带数据‘C’在第四驱动器上分配四个区域

[0121] iii) 将‘A’的最后两个区域复制到‘C’的最初两个区域

[0122] iv) 将‘B’的最初两个区域复制到‘C’的最后两个区域

[0123] v) 在奇偶校验驱动器‘D’上分配四个区域

[0124] vi) 使用 A、C 的最初四个区域和 B 的最后四个区域计算奇偶校验信息

[0125] vii) 在‘D’中放置奇偶校验信息

[0126] viii) 任何对已经处理的数据的写都必须奇偶校验到‘D’中适当位置

[0127] ix) 用新布局类型和点表更新存储区表为‘A’、‘C’的最初四区域、‘B’和‘D’的其次四区域

[0128] x) 标记‘A’的最后两区域和‘B’的最初两区域为空闲。

[0129] 数据布局方案——驱动器收缩

[0130] 驱动器收缩在盘片去除或者故障时发生。在这种情况下，如果可能的话，阵列收缩数据使全部存储区返回到冗余状态。驱动器收缩比扩展稍微复杂，因为要做更多选择。但是，在布局方法之间的转换按照与扩展相类似的方式发生，但要反过来。保持要再生的数据量为最小以使得尽快实现冗余。在空间可用时，驱动器收缩通常一次处理一个存储区，直到全部存储区被再布局。一般而言，将仅重构位于已去除或发生故障的盘片上的数据。

[0131] 选择如何收缩

[0132] 下表描述根据本发明示例性实施例的用于需要再布局的每个存储区的决策树：

[0133]

| 存储区类型缺失数据 | 条件 | 操作 |
|---|---|---|
| 任何 | 对存储区再布局没有可用空间 | 让存储区处于降级状态直到增加新盘片或者替换去除的盘片 |
| 单驱动器镜像 | 数据不一致 | 锁定系统并等待复位或者替换缺失的驱动器 |
| 双驱动器镜像 | 在系统剩下 1 个盘片 | 转换为单驱动器镜像 |
| | 仅在包含剩余数据的驱动器上有可用空间 | |
| | 在系统剩留下具有可用空间的 2 或 3 个盘片 | 在其他驱动器上重建镜像 |
| 3 驱动器条带化 | 系统剩下具有可用空间的 2 个盘片 | 转换为 2 驱动器镜像 |
| | 系统剩下具有可用空间的 3 个盘片 | 第三驱动器上重构缺失的条带化分片 |
| 4 驱动器条带化 | 系统剩下具有可用空间的 3 个盘片 | 转换为 3 驱动器条带化 |

[0134] 下面描述根据本发明示例性实施例的从双驱动器镜像收缩为单驱动器镜像的一般过程：

[0135] i) 假定单驱动器镜像具有数据‘A’和缺失的镜像‘B’或者反过来

[0136] ii) 在包含‘A’的驱动器上分配 12 个区域作为‘C’

[0137] iii) 将数据‘A’复制到区域集‘C’

[0138] iv) 任何对已经复制的数据的写都必须镜像到‘C’中适当位置

[0139] v) 当复制完成时，用新布局类型更新存储区表并用指向‘C’的指针替换指向‘B’

的指针

[0140] 下面说明根据本发明示例性实施例的从三驱动器条带化收缩为双驱动器镜像（缺失奇偶校验）的一般过程：

[0141] i）假定所述条带由不同驱动器上的数据块‘A’、‘B’、‘C’组成。缺失奇偶校验‘C’。

[0142] ii）定义‘A’为包括该存储区的前一半，而‘B’为该存储区的后一半。

[0143] iii）在‘A’驱动器上分配 6 个区域的‘D’并在‘B’驱动器上分配 6 个区域的‘E’。

[0144] iv）将‘A’复制到‘E’。

[0145] v）将‘B’复制到‘D’。

[0146] vi）任何对已经复制的数据的写都必须镜像到‘D’和‘E’中的适当位置

[0147] vii）当复制完成时，用新布局类型更新存储区表并将指针设置到指向‘A’/‘D’和‘E’/‘B’

[0148] 下面说明根据本发明示例性实施例的从三驱动器条带化收缩为双驱动器镜像（缺失数据）的一般过程：

[0149] i）假定所述条带由不同驱动器上的数据块‘A’、‘B’和‘C’组成。缺失数据‘C’。

[0150] ii）定义‘A’为包括该存储区的前一半，而‘C’为该存储区的后一半。

[0151] iii）在‘A’驱动器上分配 6 个区域的‘D’并在‘B’驱动器上分配 12 个区域的‘E’。

[0152] iv）将‘A’复制到‘E’的前一半。

[0153] v）重构从‘A’和‘B’缺失的数据。将数据写入‘D’。

[0154] vi）将‘D’复制到‘E’的后一半。

[0155] vii）任何对已经复制的数据的写都必须镜像到‘D’和‘E’中的适当位置

[0156] viii）当复制完成时，用新布局类型更新存储区表并将指针设置为指向‘A’/‘D’和‘E’

[0157] ix）将‘B’区域标记为空闲。

[0158] 下面说明根据本发明示例性实施例的从四驱动器条带化收缩为三驱动器条带化（缺失奇偶校验）的一般过程：

[0159] i）假定所述条带由不同驱动器上的数据块‘A’、‘B’、‘C’和‘D’组成。缺失奇偶校验‘D’。

[0160] ii）定义‘A’为包括存储区的前三分之一，‘B’为第二个三分之一，而‘C’为第三个三分之一。

[0161] iii）在‘A’驱动器上分配 2 个区域的‘G’，在‘C’驱动器上分配 2 个区域的‘E’并在‘B’驱动器上分配 6 个区域的‘F’。

[0162] iv）将‘B’的前一半复制到‘G’。

[0163] v）将‘B’的后一半复制到‘E’。

[0164] vi）从‘A’/‘G’和‘E’/‘C’构造奇偶校验并将其写入‘F’。

[0165] vii）任何对已经复制的数据的写都必须镜像到‘G’、‘E’和‘F’中的适当位置

[0166] viii）当复制完成时，用新布局类型更新存储区表并将指针设置为指向‘A’/‘G’、‘E’/‘C’和‘F’

[0167] ix）将‘B’区域标记为空闲。

[0168] 下面说明根据本发明示例性实施例的从四驱动器条带化收缩为三驱动器条带化(缺失前 1/3)的一般过程：

[0169] i)假定所述条带由不同驱动器上的数据块‘A’、‘B’、‘C’和‘D’组成。缺失数据‘A’。

[0170] ii)定义‘A’为包括存储区的前三分之一，‘B’为第二个三分之一而‘C’为第三个三分之一以及‘D’为奇偶校验。

[0171] iii)在‘B’驱动器上分配 4 个区域的‘E’，在‘C’驱动器上分配 2 个区域的‘F’并在‘D’驱动器上分配 6 个区域的‘G’。

[0172] iv)将‘B’的后一半复制到‘F’。

[0173] v)根据‘B’、‘C’和‘D’构造缺失数据并写入‘E’

[0174] vi)根据‘E’/‘B’的前一半和‘F’/‘C’构造新奇偶校验并写入‘G’

[0175] vii)任何对已经复制的数据的写都必须镜像到‘B’、‘E’、‘F’和‘G’中的适当位置

[0176] viii)当复制完成时，用新布局类型更新存储区表并将指针设置为指向‘E’/‘B’的前一半和‘F’/‘C’以及‘G’

[0177] ix)将‘B’的后一半和‘D’区域标记为空闲。

[0178] 下面说明根据本发明示例性实施例的从四驱动器条带化收缩为三驱动器条带化(缺失第二个 1/3)的一般过程：

[0179] i)假定所述条带由不同驱动器上的数据块‘A’、‘B’、‘C’和‘D’组成。缺失数据‘B’。

[0180] ii)定义‘A’为包括存储区的前三分之一，‘B’为第二个三分之一而‘C’为第三个三分之一以及‘D’为奇偶校验。

[0181] iii)在‘A’驱动器上分配 2 个区域的‘E’，在‘C’驱动器上分配 2 个区域的‘F’并在‘D’驱动器上分配 6 个区域的‘G’。

[0182] iv)根据‘A’的前一半、‘C’的前一半和‘D’的前一半构造缺失数据并写入‘E’

[0183] v)从‘A’的后一半、‘C’的后一半和‘D’的后一半构造缺失数据并写入‘F’

[0184] vi)从‘A’/‘E’和‘F’/‘C’构造新奇偶校验并写入‘G’

[0185] vii)任何对已经复制的数据的写都必须镜像到‘E’、‘F’和‘G’中的适当位置

[0186] viii)当复制完成时，用新布局类型更新存储区表并将指针设置为指向‘E’、‘F’以及‘G’

[0187] ix)将‘D’区域标记为空闲。

[0188] 下面说明根据本发明示例性实施例的从四驱动器条带化收缩为三驱动器条带化(缺失第三个 1/3)的一般过程：

[0189] i)假定所述条带由不同驱动器上的数据块‘A’、‘B’、‘C’和‘D’组成。缺失数据‘C’。

[0190] ii)定义‘A’为包括该存储区的前三分之一，‘B’为第二个三分之一而‘C’为第三个三分之一以及‘D’为奇偶校验。

[0191] iii)在‘A’驱动器上分配 2 个区域的‘E’，在‘B’驱动器上分配 4 个区域的‘F’并在‘D’驱动器上分配 6 个区域的‘G’。

[0192] iv) 将‘B’的前一半复制到‘E’

[0193] v) 根据‘A’、‘B’和‘D’构造缺失数据并写入‘F’

[0194] vi) 从‘A’/‘E’和‘B’/‘F’的后一半构造新奇偶校验并写入‘G’

[0195] vii) 任何对已经复制的数据的写都必须镜像到‘E’、‘F’和‘G’中的适当位置

[0196] viii) 当复制完成时，用新布局类型更新存储区表并将指针设置为指向‘A’/‘E’和‘B’/‘F’的后一半以及‘G’

[0197] ix) 将‘B’的前一半和‘D’区域标记为空闲。

[0198] 例如，再次参考图 3，如果驱动器 0 或者驱动器 1 丢失，只要在驱动器 2 上有足够的可用空间，就可以在驱动器 2 上重构双驱动器镜像（存储区 B）。类似地，如果损失任何驱动器 0 至 2，只要驱动器 3 上有足够可用空间，三驱动器镜像（存储区 C）就可以利用驱动器 3 重构。

[0199] 数据布局方案——存储区重构

[0200] 当已经去除驱动器并且剩余驱动器上有足够空间用于理想的存储区再布局或者驱动器已经用新的更大尺寸驱动器替换，会发生存储区重构。

[0201] 下面说明根据本发明示例性实施例的双驱动器镜像重构的一般过程：

[0202] i) 假定单驱动器镜像具有数据‘A’和缺失镜像‘B’

[0203] ii) 在不同于包含‘A’的驱动器的驱动器上分配 12 个区域的‘C’

[0204] iii) 将数据‘A’复制到‘C’

[0205] iv) 任何对已经复制的数据的写都必须镜像到‘C’中适当位置

[0206] v) 当复制完成时，用指向‘C’指针更新存储区表指向‘B’的指针

[0207] 下面说明根据本发明示例性实施例的三驱动器条带化重构的一般过程：

[0208] i) 假定一个驱动器具有数据‘A’，第二驱动器具有数据‘B’而第三个具有奇偶校验‘P’。缺失‘B’。注意缺失哪片无关紧要，在所有情况下需要的操作都是相同的。

[0209] ii) 在不同于包含‘A’和‘P’的驱动器的驱动器上分配 6 个区域的‘D’

[0210] iii) 从‘A’和‘P’构造缺失数据。向‘D’写数据

[0211] iv) 任何对已经复制的数据的写都必须奇偶校验到‘D’中的适当位置

[0212] v) 通过用指向‘D’的指针替换指向‘B’的指针来更新存储区表

[0213] 在这个示例性实施例中，四驱动器重构仅当如果去除的驱动器被其他驱动器替换时发生。所述重构包括在新驱动器上分配六个区域并根据其他三个区域集合重构缺失的数据。

[0214] 数据布局方案——暂时缺失驱动器问题

[0215] 当去除驱动器而没有用于再布局的空间时，阵列将继续按降级模式来运行，直到旧驱动器插回或者用新的替换该驱动器。如果插入新的驱动器，那么将再建驱动器组。在这种情况下，数据将再布局。如果旧的盘片放回该阵列，那么其将不再是当前盘片组的一部分并且将被视其为新盘片。但是，如果该阵列中没有放入新盘片而是放回旧的那个，那么旧的那个将仍然被视为该盘片组的一员，尽管是被废弃的成员。在这种情况下，任何已经再布局的存储区都将保持其新配置而该旧盘片上的区域将空闲。任何没有被再布局的区域仍然指向该旧盘片存储区的适当区域。但是，由于已经对降级存储区执行了某些写，因此这些存储区需要刷新。可以标记已经改变的降级区域，而不是记录每个已经发生的写。这样，当替

换盘片时，仅已经改变的区域需要刷新。

[0216] 而且，任何已经被写入的存储区可被置于更高的优先表用于再布局。这应当减少了应该替换盘片需要刷新的区域数量。还可以使用超时，在这一点之后，即使该盘片被替换也将被擦去。但是，这种超时可能相当大，可能是数小时而非数分钟。

[0217] 数据布局方案——数据完整性

[0218] 如上所述，标准 RAID 系统的一个问题是盘面损坏可能发生在盘片阵列的不常使用的区域。在另一个驱动器出现故障的情况下，经常不能确定损坏已经发生。在这种情况下，当 RAID 阵列重建该故障驱动器时，会传播并保存该损坏的数据。

[0219] 上述散列机制提供对于在 RAID 下可用的数据破坏检测的附加机制。如在其他地方提及的，当存储大块时，为该大块计算并存储散列值。每次读取该大块的时候，都可以计算被检索的大块的散列值并与存储的散列值进行比较。如果该散列值不匹配（指示大块损坏），那么大块数据可以从冗余数据恢复。

[0220] 为了最小化在其中可能发生盘片上数据损坏的时间窗口，将执行盘片数据常规扫描以尽快发现并校正损坏的数据。可选地，也允许执行对阵列读取的检查。

[0221] 数据布局方案——卷 (volume)

[0222] 在后备卷中，不管阵列中盘片上可用存储空间量如何，总是要求阵列是固定尺寸——例如 M 千兆字节。假定该阵列包含实际存储空间的 S 字节，其中 S ＜＝ M，并且可以将请求的数据存储在 M 千兆字节空间的位置 L1、L2、L3 等。如果所请求的位置 Ln ＞ S，那么用于 Ln 的数据必须存储在位置 Pn ＜ S。这通过基于 Ln 的包括查找表的索引 Pn 来进行管理，如图 8 所示。这种特征使得阵列与不支持卷扩展的操作系统共同工作，所述操作系统例如 Windows、Linux、和苹果操作系统。另外，该阵列可以提供多个都共享相同物理存储器的后备卷。每个后备卷都将具有专用查找表，但是将共享数据存储器的相同物理空间。

[0223] 驱动器插槽指示器

[0224] 如上所述，存储阵列包括一个或者多个驱动器插槽。每个驱动器插槽都可以是空的或者容纳硬盘驱动器。每个驱动器插槽具有能够指示四种状态的专用指示器，所述四种状态是：关闭、正常、降级和故障。该状态通常解释如下：

[0225]

| 指示器状态 | 对阵列用户的含义 |
| --- | --- |
| 关闭 | 驱动器插槽为空并且对待插入的附加驱动器是可用的 |
| 正常 | 插槽中的驱动器正确运行 |
| 降级 | 建议用户的操作：如果插槽为空，则向这个插槽增加驱动器；如果插槽已包含驱动器，则应该用另一个更高容量的驱动器替换该驱动器。 |
| 故障 | 请求用户的 ASAP 操作：如果插槽为空，则向这个插槽增加驱动器；如果插槽已包含驱动器，则用另一个更高容量的驱动器替换该驱动器。 |

[0226] 在这个示例性实施例中，红 / 黄 / 绿发光二极管 (LED) 用作指示器。所述 LED 通常解释如下：

[0227]

| LED 状态 | 指示器状态 | 可发生状态的示例情况 | 附图 |
|---|---|---|---|
| 关闭 | 关闭 | 插槽为空。阵列有可用空间。 | 9,10,12 |
| 绿色 | 正常 | 驱动器正确运行，阵列数据冗余并且阵列有可用盘空间。 | 9,10,11,12 |
| 黄色 | 降级 | 阵列接近故障条件；盘片故障情况下，没有足够空间维护冗余数据。 | 11 |
| 红色 | 故障 | 该插槽中的盘片已经故障并且必须替换；阵列没有足够空间维护冗余数据存储而且必须增加更多空间。 | 10,12 |

[0228] 图 9 示出了根据本发明示例性实施例的具有可用存储空间并按容错方式运行的示例性阵列。用存储设备填充插槽 B、C 和 D，并且有充足存储空间可用于冗余地存储附加数据。插槽 B、C 和 D 的指示器是绿色（指示这些存储设备操作正确，阵列数据冗余，且该阵列具有可用盘片空间），而插槽 A 的指示器关闭（指示插槽 A 中不需要填充存储设备）。

[0229] 图 10 示出了根据本发明示例性实施例的示例性阵列，其不具有足够空间用于维持冗余数据存储，并且必须要增加更大空间。用存储设备填充插槽 B、C 和 D。插槽 C 和 D 中的存储设备是满的。插槽 B、C 和 D 的指示器是绿色（指示这些存储设备正确运行），而插槽 A 的指示器是红色（指示该阵列没有足够空间用来维持冗余数据存储以及插槽 A 中应当填充存储设备）。

[0230] 图 11 示出了根据本发明示例性实施例的在故障情况下不能维持冗余数据的示例性阵列。插槽 A、B、C 和 D 用存储设备填充。插槽 C 和 D 中的存储设备是满的。插槽 A、B、和 C 的指示器是绿色（指示它们正确运行），而插槽 D 的指示器是黄色（指示插槽 D 中的存储设备应当用具有更大存储容量的存储设备来填充）。

[0231] 图 12 示出了根据本发明示例性实施例的示例性阵列，其中的存储设备已经故障。用存储设备填充插槽 B、C 和 D。插槽 C 中的存储设备故障。插槽 B 和 D 的指示器是绿色（指示它们正确运行），而插槽 C 的指示器是红色（指示应当替换插槽 C 中的存储设备），而插槽 A 的指示器关闭（指示插槽 A 中不需要填充存储设备）。

[0232] 下面说明本发明示例性实施例的软件设计。该软件设计基于六个软件层，其跨越从物理访问该盘片到与主机计算系统通信的逻辑体系结构。

[0233] 在这个示例性实施例中，文件系统驻留在主机服务器上，例如 Windows、Linux、或者苹果服务器，并访问如 USB 或者 iSCSI 设备的存储阵列。由主机请求管理器 (HRM) 处理经过主接口到达的物理盘片请求。主 I/O 接口将主 USB 或者 iSCSI 接口的表示协调至该主机 (host)，以及与 HRM 有接口。HRM 协调来自主 I/O 接口的数据读 / 写请求，调度读和写请求，并在它们完成时协调这些请求的结果返回该主机。

[0234] 本存储阵列的主要目的是保证一旦系统接受数据，其按可靠方法来存储，使用系统当前存储的最大量冗余。随着该阵列改变物理配置，而重新组织数据以维持（以及可能最大化）冗余。另外，基于压缩的简单散列用于减少使用的存储量。

[0235] 最基本的层包括盘片驱动器，用来在不同盘片上存储数据。可以经由各种接口连接盘，例如经 USB 接口的 ATA 隧道。

[0236] 所述盘片上的扇区被组织成区域、存储区和聚簇，其中每个都具有不同逻辑角色。

[0237] 区域表示盘片上的一组连续物理块。在四驱动器系统中，每个区域是 1/12GB 大小，并表示冗余的最小单位。如果发现区域中的扇区是物理损坏的，那么将放弃整个区域。

[0238] 存储区表示冗余单位。存储区包括许多区域，可能在不同盘片上用来提供适当的冗余量。存储区将提供 1GB 的数据容量，但是可能需要更多的区域以便于提供冗余。不具有冗余的 1GB 需要 12 个区域的集合（1GB）；1GB 镜像存储区需要两组 1GB 的区域（24 个区域）；1GB 三盘片条带化存储区将需要三组 0.5GB 的区域（18 个区域）。不同存储区将具有不同冗余特征。

[0239] 聚簇表示压缩的基本单位，并且是存储区之内的单元尺寸。它们当前是 4KB：8x512 字节扇区大小。盘片上的许多聚簇会包含相同数据。聚簇存取表（CAT）用于经由散列函数跟踪聚簇的使用。CAT 在逻辑主地址和存储区中适当聚簇位置之间转换。

[0240] 当向盘片写入时，使用散列函数来发现数据是否已经存在于该盘片上。如果是，CAT 表中的适当项会被设置为指向现有聚簇。

[0241] CAT 表驻留在其自己的存储区。如果超过该存储区大小，会使用附加存储区，并且使用表把逻辑地址映射到该存储区用于 CAT 的部分。替代地，预分配存储区用于包含该 CAT 表。

[0242] 为了减少主写入等待时间并保证数据可靠性，日志管理器将记录全部写请求（或者写入盘片或者写入 NVRAM）。如果系统重启，在重启时要提交日志项。

[0243] 盘片可以添加或者去除，或者如果发现区域已经损坏则可以让该区域退出。在任何这些情况下，布局管理器都可以在存储区中重组区域，以改变其冗余类型，或者改变存储区的区域构成（如果某个区域被损坏）。

[0244] 由于存储阵列提供虚拟盘阵列因此通过改变物理盘空间的级别来返回，并且由于提供块级别的接口，当聚簇不再被文件系统使用时就不是显然的。结果，所使用的聚簇空间将继续扩展。垃圾收集器（位于主机或者按固件形式）将分析该文件系统以确定哪个聚簇已经空闲，并且从散列表中删除它们。下表示出了根据本发明该示例性实施例的六个软件层：

[0245]

| |
|---|
| 层 5：垃圾收集器，主接口（USB/iSCSI） |
| 层 4：主请求管理器 |
| 层 3：CAT，HASH，日志管理器 |
| 层 2：存储区管理器。分配 / 释放称作存储区的扇区块。知道 SDM、DDM、SD3 等，以处理差错和差错恢复。布局管理器 |

| |
|---|
| 层 1 :读 / 写物理聚簇 / 扇区。分配每张盘的区域。 |
| 层 0 :盘存取驱动器 |

[0246] 图 13 示出了模块层次，其表示不同软件层和它们如何彼此相关。软件层优选地固定以便于提供清晰的 API 和描述。

[0247] 垃圾收集器释放不再由主机文件系统使用的聚簇。例如，当删除文件时，优选地释放用于包含该文件的聚簇。

[0248] 日志管理器提供某种形式的写日志，从而在电源故障或者其他差错条件的情况下不丢失挂起的写操作。

[0249] 布局管理器提供存储区相对于其区域的运行时再布局。这可能根据盘插入 / 删除或者故障而发生。

[0250] 聚簇管理器在一组数据存储区中分配聚簇。盘利用守护程序 (diskutilization daemon) 周期性检查空闲的盘片空间。

[0251] 加锁表 (Lock Table) 处理写操作冲突问题后的读操作。

[0252] 主请求管理器处理来自主机和垃圾收集器的读 / 写请求。写操作被传递到日志管理器，而读操作经由聚簇存取表 (CAT) 管理层来处理。

[0253] 如上所述，在典型文件系统中，一定量的数据通常实质上重复。为了减少盘空间利用，这种数据的多个复制不会写入该盘片。而是写入一个实例，相同数据的其他全部实例引用 (reference) 这一个实例。

[0254] 在本示例性实施例中，任何时间系统在数据聚簇上操作 ( 例如 8 个物理扇区 )，而这是散列的单位。使用 SHA1 算法来产生 160 位的散列。这样具有很多好处，包括好的唯一性，并且在许多处理器中被片上支持。全部 160 位将存储在散列记录中，但是仅最低有效 16 位被用作散列表中的索引。其他匹配该最低 16 位的实例将经由链接表链接。

[0255] 在这个示例性实施例中，仅一个读 / 写操作可以同时发生。为了性能的目的，当向盘片写聚簇时不允许发生散列分析。而散列分析将由散列管理器作为后台活动而发生。

[0256] 从日志的写队列读取写请求，并处理以完成写操作。为了保证数据一致性，如果已经有写操作在该聚簇上活动，必须延迟该写操作。在其他聚簇上的操作可以不受阻碍的进行。

[0257] 除非写整个聚簇，否则写入的数据将需要与现有存储在该聚簇中的数据归并。根据逻辑扇区地址 (LSA)，定位聚簇的 CAT 项。从这个记录获得散列关键字、存储区和聚簇偏移信息，然后它们可用于搜索散列表以发现匹配。这就是聚簇。

[0258] 双重散列该散列表可能是必需的 ；一旦经由 SHA1 摘要 (digest)，然后就通过存储区 / 聚簇偏移用于改进正确散列项的查找速度。如果已经使用了散列记录，引用计数被递减。如果引用计数现在为零，并且没有由散列项快照引用，该散列项和聚簇可被释放回它们各自的空闲表。

[0259] 现在归并原来的聚簇数据和聚簇的更新扇区，而且数据要被再散列。新的聚簇从空闲表中被剔除，把归并数据写入该聚簇，向散列表中增加新项，CAT 表中的项更新为指向该新聚簇。

[0260] 作为更新散列表的结果，该项同样添加到用于由后台任务处理的内部队列。这个

任务将把新添加的聚簇和散列项与匹配该散列表行地址的其他散列项进行比较，如果它们重复，将结合记录，在适当时释放散列项和 CAT 表项。这保证了写等待时间不由这个活动负担。如果在这个处理期间发生故障（例如掉电），则可能删除各种表，导致丢失数据。各个表应当以这种方式管理，最终的提交是原子的 (atomic)，或者日志项可以再运行，如果它没有全部完成的话。

[0261] 下面是写逻辑的伪代码：

```
[0262] While(stuff to do)
[0263] WriteRecord = journalMgr.read();
[0264] lsa = writeRecord.RetLsa();
[0265] catEntry = catMgr.GetCATEntry(lsa);
[0266] if(catMgr.writeInProgress(catEntry))delay();
[0267] originalCluster = catMgr.readCluster(catEntry);
[0268] originalHash = hashMgr.calcHash(originalCluster);
[0269] hashRecord = hashMgr.Lookup(originalHash,zone,offset);
[0270] if((hashRecord.RefCount == 1)&&(hashRecord.snapshot = 0))
[0271]         hashRecord.free();
[0272]         originalCluster.free();
[0273] //注意这里有某种优化，可重用该聚簇而无须释放或重新分配它。
[0274] //otherwise,still users of this cluster,so update & leave it alone
[0275] hashRecord.RefCount--;
[0276] hashRecord.Update(hashRecord);
[0277] //现在添加新记录
[0278] mergedCluster = mergeCluster(originalCluster,newCluster);
[0279] newHash = hashMgr.calcHash(mergedCluster);
[0280] newCluster = clusterMgr.AllocateCluster(zone,offset);
[0281] clusterMgr.write(cluster,mergedCluster);
[0282] zoneMgr.write(cluster,mergedCluster);
[0283] ...\
[0284] hashMgr.addHash(newHash,newCluster,zone,offset)
[0285]       (internal:queue new hash for background processing)
[0286] catMgr.Update(lba,zone,offset,newHash);
[0287] //我们已成功完成该日志项。移至下一个。
[0288] JournalMgr.next();
```

[0289] 读请求同样按每次一个聚簇（相对于“扇区”）来处理。读请求不通过上述的与散列相关的处理。而是，使用主逻辑扇区地址来引用 (reference)CAT 并获得存储区编号和聚簇到该存储区的偏移。读请求应当在 CAT 缓存中查找 CAT 表项，并且必须在设置了写进行位 (write-in-progress bit) 时延迟。其他读 / 写可以不受阻止的进行。为了改进数据完整性检查，当读聚簇时，其将被散列，并且该散列值与存储在散列记录中的 SHA1 散列值进行比较。这将需要使用该散列、存储区和聚簇偏移作为进入散列表的搜索关键字。

[0290] 分配聚簇以使用尽可能少的存储区。这是因为存储区直接对应于盘驱动器利用率。对每个存储区而言，硬驱动器阵列上有两个或者更多区域。通过最小化存储区数量，物理区域的数量最小化，并由此减少硬盘驱动器阵列上的空间消耗。

[0291] 聚簇管理器分配来自一组数据存储区的聚簇。使用链接列表跟踪存储区中的空闲聚簇。但是，空闲聚簇信息在盘片上存储为位图（32KB 每存储区）。该链接列表从位图（bit map）动态地构造。最初，在存储器中创建特定量空闲聚簇的链接列表。当分配聚簇时，该列表收缩。在预定最低点，从盘片上的位图提取表示空闲聚簇的新链接列表节点。这样，为了发现用于分配的空闲聚簇，不需要解析位图。

[0292] 在这个示例性实施例中，散列表是 64K 记录表（由散列的低 16 位索引）并且具有下列格式：

[0293]

| 偏移 | 位尺寸 | 名称 | 值 / 有效范围 | 说明 |
|---|---|---|---|---|
| 0 | 160 | Sha 1 散列 | | 完整 SHA1 散列摘要 |
| | 16 | RefCount | | 该散列的实例数；如果超出 16 位我们该怎么办 |
| | 18 | 聚簇偏移 | | 存储区中的聚簇偏移 |
| | 14 | 存储区 # | | 包含该聚簇的存储区 # |
| | 8 | 快照 | | 每快照实例一位用于指示该聚簇项由该快照使用。该模型支持 8 个快照（可能仅 7 个） |

[0294] 全零的聚簇可以相当常见，因此全零的情况可以视作特殊情况，例如，使得它永远不被删除（因此覆盖计数将不是问题）。

[0295] 当多个散列具有相同最低有效散列时，或者当两个散列项指向不同数据聚簇时，使用空闲散列记录的链接列表。在两种情况下，空闲散列记录都将从该列表中取出，并经由 pNextHash 指针链接。

[0296] 散列管理器将整理添加到散列表中的项并将合并该盘片上的相同聚簇。随着新的散列记录增加到该散列表中，消息将被传递到散列管理器。这可由散列管理器自动执行。作为后台活动，散列管理器即将处理其队列上的项。将比较全部散列值以发现其是否匹配任何现有散列记录。如果是，将同样比较完整聚簇数据。如果聚簇匹配，则新散列记录可以被废除回空闲队列，散列记录计数递增，并且重复的聚簇将被返回聚簇空闲队列。当合并记录时，散列管理器必须注意向前传播快照位。

[0297] 聚簇存取表（CAT）包含间接指针。该指针指向存储区中的数据聚簇（0 是第一数据聚簇）。一个 CAT 项引用单个数据聚簇（暂定 4KB 大小）。使用 CAT（连同散列）使得当存在大量重复的数据时减少盘片使用需求。单个 CAT 通常表示连续存储块。CAT 包含在非数据存储区中。每个 CAT 项是 48 位。下表示出每个项如何布局（假定每个数据存储区包

含 1GB 数据）：

[0298]

| 位 0-17 | 位 18-31 | 位 32-47 | 位 48-63[...] |
|---|---|---|---|
| 存储区中数据聚簇的偏移 | 包含数据的存储区 # | 散列关键字 | 保留。候选包括垃圾收集器写位；快照位；快照表格散列关键字 |

[0299] 希望 CAT 适合 64 位，但这不是必需的。2TB 阵列的 CAT 表当前是约 4GB 大小。每个 CAT 项指向包含该数据的存储区和存储区编号。

[0300] 图 14 显示 CAT 如何用于存取存储区中的数据聚簇。冗余数据通过 CAT 中的一个以上的项来引用。两个逻辑聚簇包含相同数据，因此它们的 CAT 项指向相同物理聚簇。

[0301] 散列关键字项包含完整聚簇的 160 位 SHA1 散列值的 16 位摘取。这个项用于在写操作期间更新该散列表。

[0302] CAT 中的每个项有足够的位用于引用 16TB 的数据。但是，如果每个数据聚簇都彼此不同（根据内容），那么只需要 3 个存储区的 CAT 项来引用 2TB 的数据（每个存储区都是 1GB 大小，并由此可以存储 1GB/ 大小的 CAT 项。假定 6 字节 CAT 项，则 178956970CAT 项 / 存储区，即表引用大约 682GB/ 存储区，如果每个聚簇是 4K 的话）。

[0303] 主逻辑扇区转换表用于将主逻辑扇区地址转换成存储区编号。相应于主逻辑扇区地址的 CAT 的一部分将驻留在这个存储区中。注意每个 CAT 项表示 4096 字节的聚簇大小。这是八个 512 字节扇区。下面显示主逻辑扇区转换表的表示：

[0304]

| 启动主逻辑扇区地址 | 结束主逻辑扇区地址 | CAT 的存储区 # |
|---|---|---|
| 0（聚簇 #0） | 1431655759（聚簇 #178956969） | |
| 1431655760（聚簇 #178956970） | ... | |

[0305] 可以预分配存储区以保持整个 CAT。替代地，存储区可以分配给 CAT，如需要更多 CAT 项。由于 CAT 把 2TB 虚拟盘片映射至主扇区地址空间，因此由主机做硬盘分区或者格式化期间将引用 CAT 的很大部分。为此，要预分配存储区。

[0306] CAT 是大的 1GB/ 存储区表。使用的工作聚簇集将是来自这个大表的后备集。为了性能的原因，活动的项（可能暂时）可以在处理器存储器中缓存而不总是从盘片读取。至少有两个选项用于填充该缓存——来自 CAT 的个别项，或者来自 CAT 的整个聚簇。

[0307] 因为写进行 (write-in-progress) 与 CAT 缓存表相组合，所以需要确保该缓存中保持全部未完成的写。因此，需要该缓存至少与未完成写请求的最大数量一样大。

[0308] 缓存中的项将是聚簇大小（即 4K）。需要知道聚簇上的操作中是否还有写进行。这个指示可以作为标志存储在该聚簇的缓存项中。下表显示 CAT 缓存项的格式：

[0309]

| 位 0-17 | 位 18-31 | 位 32-47 | 位 48-63 |
|---|---|---|---|

| 存储区中数据聚簇的偏移 | 包含数据的存储区 # | 散列关键字 | 位 48 :写进行<br>位 49 :脏 |
|---|---|---|---|

[0310] 缓存项中的写进行标志有两种含义。首先，它指出写操作正在进行，并且在这个聚簇上的任何读（或者附加写）必须延迟，直到完成该写操作。其次，当设置该位时，绝不能刷新缓存中的这个项。这部分地保护了该位的状态，同时反映了这个聚簇当前被使用的事实。另外，这意味着缓存的尺寸必须至少与未完成的写操作数量一样大。

[0311] 在聚簇的缓存项中存储写进行指示符的一个优点是它反映了操作正在进行的事实，省去了使用其他表格，并且省去了另外的基于散列的查找，或用于检查该位的表查找。该缓存可以是写延迟缓存。只需要当写操作完成时，将缓存项写回盘片，虽然将其更早写回可能更好。散列函数或者其他机制可用于增加可散列的未完成的写项。

[0312] 一种替换方法是缓存整个 CAT 的聚簇（即各项的 4K 项）。这通常有助于性能，如果有好的访问空间定位的话。需要注意，因为 CAT 项是 48 位宽，因此缓存中没有全部的项。下表显示聚簇 CAT 缓存项的示例 ：

[0313]

| 两个字 | 两个字 | 两个字 | 两个字 |
|---|---|---|---|
| CAT 项 1（最后两个字的部分项） | CAT 项 2 | | |
| CAT 项 3 | | | CAT 项 4 |
| CAT 项 4 | CAT 项 5 | | |
| CAT 项 5 | CAT 项 6 | | |
| … | | | |
| CAT 项 682 | | | CAT 项 683（前两个字的部分项） |
| 写进行位阵列[682 位]：位 0-255 | | | |
| 写进行位阵列位 256-511 | | | |
| 写进行位阵列位 512-682+备用位 | | 脏数 | 保留 |

[0314] 该表尺寸可以是 4096+96(4192 字节）。假定需要具有 250 项的缓存大小，该缓冲可以占据大约 1MB。

[0315] 通过逻辑 CAT 项地址的适当屏蔽可以计算首项和末项是否未完成。缓存查找例程应当在加载项之前执行这个过程并且应当加载需要的 CAT 聚簇。

[0316] 当主机发送扇区（或者聚簇）读请求时，其通过逻辑扇区地址发送。该逻辑扇区地址用作到 CAT 的偏移以获得包含主机所请求的实际数据的存储区中的聚簇的偏移。结果是存储区编号和到该存储区的偏移。该信息传递给层 2 软件，然后其从（多个）驱动器提取原始（多个）聚簇。

[0317] 为了处理主机从未写过的聚簇，所有 CAT 项被初始化为指向包含全零的“默认”聚

簇。

[0318] 日志管理器是两级写(bi-level write)日志系统。该系统的一个目标是保证可以从主机接收写请求并快速向该主机返回指示，数据已经在保证其完整性的同时被接收。另外，该系统需要保证在任何盘写入期间的系统复位的情况下，不会有块级数据或者系统元数据（例如 CAT 和散列表项）的损坏或丢失。

[0319] J1 日志管理器尽快缓存所有从主机向盘片的写请求。一旦写入成功完成（即数据已经被阵列接收），主机就可以发信号指示操作已经完成。日志项允许当从故障恢复时，恢复写请求。日志记录包括要写入盘片的数据，以及与写事务相关的元数据。

[0320] 为了减少盘片读 / 写，与写入相关的数据将被写入空闲聚簇。这样将自动镜像该数据。将从空闲聚簇列表去除空闲聚簇。一旦写数据，空闲聚簇就必须写回盘片。

[0321] 日志记录将被写回非镜像存储区上的日志队列。每个记录都将是扇区大小，并且对齐到扇区边界，以使得减少日志写期间的故障会破坏以前日志项的风险。日志项在记录的末尾包含唯一的、递增的顺序计数，因此可以轻易识别队列的结尾。

[0322] 日志写操作在主机队列处理线程中同步发生。日志写必须按照它们写入盘片的次序来排序，因此在任何时候只有一个线程可以写入该日志。J1 表中日志项的地址可以用作唯一标识符，因此 J1 日志项可以与 J2 日志中的项相关联。一旦写入日志项，将向主完成队列传递事务完成通知。现在可以执行写操作。要保证在完成日志写之前延迟任何后续的对该聚簇的读，这一点很重要。

[0323] 下表示出了 J2 日志记录的格式：

[0324]

| 大小（位） | 名称 | 细节 |
|---|---|---|
| 32 | LBA | 逻辑块地址 |
| 14 | 存储区 | 相关聚簇的存储区 # |
| 18 | 偏移 | 相关聚簇的聚簇偏移 |
| 16 | 大小 | 数据大小 |
| 16 | 顺序号 | 增量顺序号以易于发现队列结束 |

[0325] 每个日志记录都对齐到扇区边界。日志记录可以包含存储区 / 偏移 / 大小的元组的阵列。

[0326] 图 15 示出了根据本发明示例性实施例的日志表更新。尤其是当接收到主机写请求时，更新该日志表，分配一个或多个聚簇，并向（多个）聚簇写入数据。

[0327] 处理主日志请求。这引起聚簇被写入，并同样引起更新元数据结构，所述结构必须投影回盘片（例如 CAT 表）。重要的是保证这些元数据结构正确地写回盘片，即使当系统发生复位。为此将使用低级盘片 I/O 写 (J2) 日志。

[0328] 为了处理主接口日志项，应当确定元数据结构的适当的操作。改变应当发生在存储器并且要产生对各盘片块改变的记录。这种记录包含在盘片上应该进行的实际改变。更

新的每种数据结构都用 J2 日志管理器来登记。这种记录应当记录到基于盘片的日志，并用标识符来加印戳。当记录与 J1 日志项相连接，标识符就应当被链接。一旦存储该记录，就可以进行盘片的改变（或者可以经由后台任务执行）。

[0329] 逻辑上 J2 日志存在于层 3。它用于把那些涉及经存储区管理器的写的元数据更新登记到日志。当发生日志项的再现时，将使用存储区管理器方法。日志本身可以存储在专门区域。由于日志项的短生命期，不对其做镜像。

[0330] 不是所有的元数据更新都需要经过 J2 日志，尤其是，如果对结构的更新是原子的。区域管理器结构可不使用 J2 日志。可检测区域管理器位图中的不一致，例如，使用完整性检测后台线程。

[0331] 用于 J2 日志的一种简单方法是包含单个记录。一旦该记录提交给盘片，就被重放，更新盘片上的结构。可具有多个 J2 记录，并使后台任务提交盘片上的更新记录。这种情况下，需要密切注意日志和与各种数据结构相关的任何缓存算法之间的交互作用。

[0332] 一旦提交给盘片，初始方法就将运行日志项。原则上，会有 J2 的多个并发的用户，但是 J2 日志会在一个时候锁定到一个用户。即使在这种情况下，一旦提交，也会提交日志项。

[0333] 重要的是保证在任何更高级的日志活动发生前修复元数据结构。在系统重新引导时，分析 J2 日志，并将重现任何记录。如果日志项与 J1 日志项相关，则将 J1 日志项标记为已完成，并可以被删除。一旦完成全部 J2 日志项，元数据就处于可靠状态，并且可以处理任何剩余 J1 日志项。

[0334] J2 日志记录包括下列信息：

[0335] • 操作号

[0336] • 每个操作包含：

[0337] ° J1 记录指示符

[0338] ° 待写入存储区 / 数据偏移

[0339] ° 待写入数据

[0340] ° 数据大小

[0341] ° 到数据聚簇的偏移

[0342] • 日志记录标识符

[0343] • 结束标记

[0344] 这种方案可以按类似于 J1 日志的方案来操作，例如，使用顺序号用于识别 J2 日志项的结尾并将 J2 日志项置于扇区边界处。

[0345] 如果设置 J1 数据指针指示符，那么这个特殊操作会指向 J1 日志记录。主机提供的写数据不必复制到日志项。操作阵列将可以定义为固定大小，因日志记录中操作的最大数量是已知的。

[0346] 为了允许从低级写操作期间的扇区损坏（例如由于掉电）恢复，J2 日志可以存储被写入的整个扇区，使得如果需要该扇区可以根据这个信息来重写。作为替换或附加，为每个改变的扇区计算的 CRC 可以存储在 J2 记录中，并与从盘片扇区（例如由存储区管理器）计算的 CRC 进行比较，以确定是否需要写操作的重放。

[0347] 不同日志可以存储在不同位置，因此提供接口层用于写日志记录到备份存储。该

位置应当是非易失的。两种候选是硬盘和 NVRAM。如果 J1 日志存储到硬盘，它将存储在 J1 日志非镜像存储区中。J1 日志是存储在 NVRAM 的候选。J2 日志应当存储在盘片上，尽管它可以存储在专门区域中（即，不冗余，因为它具有短生命期）。将 J2 日志存储在盘片的优点是，如果在内部数据结构更新期间存在系统复位，那么该数据结构可以返回到一致状态（即使该单元长时期掉电）。

[0348] 存储区管理器 (ZM) 分配更高级软件需要的存储区。向 ZM 的请求包括：

[0349] a. 分配存储区

[0350] b. 解除分配 / 释放存储区

[0351] c. 控制数据读 / 写传递到 L1（？）

[0352] d. 读 / 写存储区中的聚簇（给出聚簇偏移和存储区号）

[0353] ZM 管理冗余机制（随驱动器的数量和它们的相关大小而改变）并处理镜像、条带化、以及用于数据读 / 写的其他冗余方案。

[0354] 当 ZM 需要分配存储区时，它将请求两个或者更多区域集合的分配。例如，可以为 1GB 的数据分配存储区，组成这个存储区的区域将可以包含 1GB 数据，包括冗余数据。对镜像机制，存储区将由各为 1GB 的两个区域集合构成。另一个示例，3 盘片条带化机制使用各为 1/2GB 的 3 组区域。

[0355] ZM 使用 ZR 转换表 (6) 以发现组成该存储区的每组区域的位置（驱动器号和起始区域号）。假定是 1/12GB 区域大小，将需要最多 24 个区域。24 个区域组成 2x1GB 的存储区。因此 ZR 转换表包含 24 列，用于提供驱动器 / 区域数据。

[0356] ZM 通常工作如下：

[0357] a. 在 SDM 的情况下（单个驱动器镜像），使用 24 列。驱动器号在所有列中都相同。每个项对应于组成该存储区的物理驱动器上的一个物理区域。前 12 个项指向包含该数据的一个副本的区域。后 12 个项指向包含该数据的第二个副本的区域。

[0358] b.DDM（双驱动器镜像）的情况与 SDM 的情况相同，只是前 12 个项的驱动器号与后 12 个项中的驱动器号不同。

[0359] c. 在条带化的情况下，可以使用三个或者更多列。例如，如果跨三个驱动器使用条带化，则需要来自三个不同驱动器的六个区域（即使用 18 个项），前 6 项包含相同驱动器号，接下来的 6 项包含另一个驱动器号，以及随后的 6 项包含第三驱动器号，未使用的项置为 0。

[0360] 下表显示存储区区域转换表的表示法：

[0361]

| 存储区 # | 存储区大小 | 每个区域大小 | 使用 | 驱动器 / 区域 (1) | 驱动器 / 区域 (2) | ... | 驱动器 / 区域 (23) | 驱动器 / 区域 (24) |
|---|---|---|---|---|---|---|---|---|
| 0 | 1GB | 1/12 | SDM | 0,2000 | 0,1000 | ... | 0,10 | 0,2000 |
| 1 | 1GB | 1/12 | DDM | 0,8000 | 0,3000 | ... | 1,2000 | 1,10 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 2 | 1GB | 1/12 | SD3 | 3,4000 | 3,3000 | | 4,2000 | 4,1000 |
| ... | | | | | | | | |
| N | | | 空闲 | | | | | |

[0362] 当读/写请求到达时，对ZM提供存储区号和到该存储区的偏移。ZM查看ZR转换表中用于解决该存储区的冗余机制，并使用该偏移用于计算哪个驱动器/区域包含必须读/写的扇区。然后该驱动器/区域信息提供给L1层以进行实际的读/写。在“使用(usage)”列中的另外可能项是“空闲”。“空闲”指存储区被定义但当前没有使用。

[0363] 聚簇管理器分配并解分配数据存储区集合中的聚簇。

[0364] 布局管理器提供存储区关于其区域的的运行时再布局。这可根据盘片插入/除去或者故障而发生。

[0365] 层1(L1)软件知道物理驱动器和物理扇区。除了其他，L1软件分配物理驱动器的区域用于存储区管理器来使用。在这个示例性实施例中，每个区域具有用于四驱动器阵列系统的1/12GB大小(即174762扇区)。具有更大最大数量驱动器(8,12或16)的系统将具有不同的区域大小。

[0366] 为了创建包含具有SD3(在三驱动器上条带化；两个数据加奇偶校验)的1GB数据存储区，我们应当使用各在三驱动器中的六个区域(每个驱动器为6x1/12 = 1/2GB)。

[0367] 当存储区被移动或者重配置时，例如根据镜像到条带化，使用这种区域方案允许我们提供更好的盘片空间利用。L1软件利用区域位图(bitmap)跟踪物理驱动器上的可用空间。每个驱动器都有一个位图。每个区域都用位图中的两位表示，用于跟踪该区域是否空闲、使用、或者损坏。当L2软件(ZM)需要创建存储区时，它从L1层获得一组区域。构成存储区的区域不必在盘片中连续。

[0368] 向L1的请求包括：

[0369] a. 数据读/写(到一组区域中的聚簇)

[0370] b. 控制数据读/写(表格、数据结构、DIC等)

[0371] c. 分配区域的物理空间(1驱动器内的实际物理扇区)

[0372] d. 解除分配区域

[0373] e. 向物理驱动器的物理聚簇的原始读/写(raw read/write)

[0374] f. 从一个区域向另一个复制数据

[0375] g. 将区域标记为损坏。

[0376] 空闲区域位图可以是大型的，因此查找空闲项(最糟的情况是没有空闲的项)的搜索可能缓慢。为了改进性能，部分位图可以预加载到内存中，而且空闲区域的链接列表可以存储在内存中。每个活动存储区都有列表。如果到达列表上的低水位线，可以从盘片读取更多的空闲项作为后台活动。

[0377] 盘片管理器运行在层0。如下表所示，有两个子层，分别是抽象层和与物理存储阵列通信的设备驱动器。

[0378]

| |
|---|
| 层 0a :抽象 |
| 层 0b :到设备驱动器的 OS 接口和设备驱动器 |
| 物理存储阵列硬件 |

[0379] 设备驱动器层同样可以包含多个层。例如,对使用 USB 驱动器的存储阵列,在 USB 传输层的上面有 ATA 或者 SCSI 堆栈。抽象层提供独立于存储阵列中使用的驱动器种类的基本读 / 写功能。

[0380] 可使用一个或者多个盘片存取队列来对盘存取请求进行排队。在我们的系统中,盘存取速度将是一个关键的性能瓶颈。我们想要保证盘片接口尽可能在所有时间保持忙碌,从而减少一般的系统延迟并改进性能。请求盘的接口应当具有异步接口,使用回调 (callback) 处理器用于当已经结束盘操作时来完成操作。一个盘请求的完成将自动启动队列中的下一个请求。每个驱动器有一个队列,或者全部驱动器用一个队列。

[0381] 层 1 将按逻辑驱动器号来引用驱动器。层 0 将逻辑驱动器编号转换为物理驱动器参考 ( 例如 /dev/sda 或作为 open() 调用的结果的文件设备号 )。为灵活起见 ( 经 USB 扩展 ),每个逻辑驱动器应当一个队列。

[0382] 下面是对象定义和数据流的一些示例。

[0383] MSG 对象 :从主机引入

[0384] Lba

[0385] Length

[0386] LUN

[0387] Data

[0388] REPLY 对象 :引出到主机

[0389] Status

[0390] Host

[0391] Length

[0392] Data

[0393] 数据读

[0394] 数据读流程 :

[0395] rc = lockm.islocked(MSG)

[0396] rc = catm.read(MSG,REPLY)

[0397] status = zonem.read(zone,offset,length,buffer)

[0398] regionm.read(logical_disk,region_number,region_offset,

[0399] length,buffer)

[0400] diskm.read((logical_disk,offset,length,buffer)

[0401] 数据写

[0402] 数据写流程 :

[0403] diskutildaemon.spaceavailable()

[0404] journalm.write(MSG)

[0405] lockm.lock(msg)

[0406] zonem.write(journal_zone,offset,length,buffer)

[0407] regionm.write -日志项

[0408] diskm.write

[0409] regionm.write -结束标记

[0410] diskm.write

[0411] catm.write(MSG)

[0412] catm.readcluster(lba,offset,length,buffer)

[0413] -如果需要向聚簇归并扇区

[0414] - 归并

[0415] “if(lba 已经分配)”

[0416] catm.readhashkey(lba)

[0417] hashm.lookup(hashkey,zone,offset)

[0418] “if(refcount = 1)”

[0419] hashentry.getrefcount()

[0420] hashm.remove(hashentry)

[0421] hashm.add(sha1,zone,offset)

[0422] zonem.write(zone,offset,length,buffer)-写数据

[0423] “else”

[0424] hashentry.removeref()

[0425] clusterm.allocate(zone.offset) -分配新聚簇

[0426] zonem.createzone(zone)

[0427] regionm.unusedregions(logical_disk)

[0428] regionm.allocate(logical_disk,number_regions,region_list)

[0429] zonem.write(...) -写数据

[0430] hashm.add(...) -向散列表增加新项

[0431] “endif”

[0432] hashdaemon.add(lba,sha1) -增加到散列新进程 Q

[0433] catm.writehashkey(lba,hashkey) -向 CAT 复制新散列关键字

[0434] “else”

[0435] catm.update(lba,zone,offset,hashkey)-用新项更新 CAT

[0436] “endif”

[0437] journalm.complete(MSG)

[0438] lockm.unlock(MSG)

[0439] -更新 r/w 指针

[0440] 下面是物理盘布局的说明。如上所述,每个盘片都划分为固定尺寸的区域。在这个示例性实施例中,每个区域具有用于四驱动器阵列系统的 1/12GB 大小(即 174763 扇区)。具有更大最大数量驱动器 (8,12 或 16) 的系统将具有不同的区域大小。开始,保留区域号 0 和 1 用于区域管理器并且不用于分配。区域号 1 是区域号 0 的镜像。对给定的硬盘,区

域管理器所使用的所有内部数据都存储在这个硬盘的区域号 0 和 1 中。这个信息并不重复（或镜像）到其他驱动器。如果区域 0 或 1 中有错误，可以分配其他区域来保留该数据。盘片信息结构指向这些区域。

[0441] 每个盘片将包含识别该盘片的DIS、它所属于的盘片组和该盘片的布局信息。该硬盘上的第一扇区被保留。DIS 存储在第一扇区后的第一非损坏聚簇中。DIS 包含相当 1KB 的数据。有 DIS 的两个副本。DIS 的副本将存储在其所属于的盘片上。另外，该系统中的每个盘片都将包含该系统中盘片的全部 DIS 副本。下表显示 DIS 格式：

[0442]

| 偏移 | 尺寸 | 名称 | 值 / 有效范围 | 描述 |
|---|---|---|---|---|
| 0 | 32 字节 | DisStartSigniture | “_DISC INFORMATION CLUSTER START_” | 识别聚簇为可能盘片信息聚簇。聚簇必须经 CRC 以检查其有效。 |
| | 字 16 | DisVersion | 二进制非零号 | 识别结构版本。仅当结构布局或内容意思发生实质改变使得其与固件的先前版本不相容时，才改变这个值。 |
| | 字 16 | disClusterSize | 二进制非零号 | 使聚簇在该盘的 512 字节扇区数 |
| | 字 32 | DisCRC | CRC-32 | DIS 结构的 CRC |
| | 字 32 | disSize !!! | | DIS 聚簇的尺寸（字节） |
| | 字 32 | DisDiskSet | | 该盘所属的盘组 |
| | 字 32 | disDriveNumber | 0 至 15 | 盘组中的驱动器号 |
| | 字 32 | disSystemUUID | | 该盘属于的机箱的 UUIN |
| | 字 64 | DisDiskSize | | 按扇区数的盘尺寸 |
| | 字 32 | DisRegionSize | | 按扇区数的区域尺寸 |
| | 字 64 | disRegionsStart | | 扇区到盘片上的第一区域的起始的偏移 |
| | 字 64 | disCopyOffset | | 到存储该 DIS 副本处的扇区偏移。每个 DIS 的 disCopyOffset 互相参考 |
| | 字 64 | disDISBackup | | 到包含全部盘片的 DIS 副本的表的扇区偏移 |

[0443]

| | | | | |
|---|---|---|---|---|
| | 字 32 | disDISBackupSize | | DIS 备份部分的 DIS 编号 |
| | 字 32 | disRIS0Region | | 存储 RIS 的第一副本的区域编号 |
| | 字 32 | disRIS0Offset | | 在区域中到该区域信息结构位于的扇区的扇区偏移数 |
| | 字 32 | disRIS1Region | | 用于 RIS 的副本 |
| | 字 32 | disRIS1Offset | | 用于 RIS 的副本 |
| | 字 32 | disZIS0Region | | 存储区信息结构所位于区域的区域号。仅当 ZTR 位于该盘时使用。否则,其为零。 |
| | 字 32 | disZIS0Offset | | 到区域中 ZIS 的偏移 |
| | 字 32 | disZIS1Region | | ZIS 的副本所位于区域的区域号。仅在单驱动器系统中使用。其它情况下,这个项为 0。 |
| | 字 32 | disZIS1Offset | | 到该区域中 ZIS 的偏移 |

[0444] 区域管理器在区域信息结构中存储其内部数据。下表显示该区域信息结构格式：

[0445]

| 偏移 | 尺寸 | 名称 | 值 / 有效范围 | 描述 |
|---|---|---|---|---|
| 0 | 字 64 | risSignature | | 指示这个是 RIS |
| | 字 32 | risSize | | 该结构（字节）的尺寸 |
| | 字 32 | risChecksum | | 校验和 |
| | 字 32 | risVersion | | 该表的版本（和位图） |
| | 字 32 | risDrive | | 逻辑驱动器号 |
| | 字 64 | risStartSector | | 区域利用位图的绝对起始扇区（盘片中） |
| | 字 32 | risSectorOffset | | 当前区域中区域利用位图的扇区偏移 |
| | 字 32 | risSizeBitmap | | 位图尺寸（位？） |
| | 字 64 | RisNumberRegions | | 该盘片上区域号（同样隐含位图尺寸） |

[0446] 存储区信息结构提供可以发现存储区表的存储区管理器上的信息。下面显示该存储区信息结构格式：

[0447]

| 偏移 | 尺寸 | 名称 | 值 / 有效范围 | 描述 |
|---|---|---|---|---|
| 0 | 字 64 | zisSignature | | 指示这是 ZIS |
| 8 | 字 32 | ZisSize | | 该结构的尺寸（字节） |
| 12 | 字 32 | zisChecksum | | 校验和 |
| 16 | 字 32 | zisVersion | | 该表的版本（和位图） |
| 20 | 字 16 | zisFlags | | 如果该盘片用于包含存储区信息位 14-15 则位 0 = 1 :冗余类型 (SDM 或者 DDM) |
| 22 | 字 16 | zisOtherDrive | | 包含存储区表的其他副本的驱动器的逻辑驱动器号 |
| 24 | 字 32 | zisNumberRegions | | 用于包含存储区表的每个副本的区域号。等于存储区表节点的编号。 |
| 28 | 字 32 | zisStartOffset | | 指向用于包含存储区表的区域链接表开始的字节偏移。该链接列表中的每个项称作‘存储区表节点’ |
| | 字 32 | zisNumberofZones | | 系统中的存储区编号（存储区表中的项） |
| | 字 32 | zisZoneSize | | 按字节的存储区尺寸 |

[0448] 高级信息存储区包含存储区表和其他由高级管理者使用的表。这将使用镜像进行保护。

[0449] 下表显示该存储区表节点格式：

[0450]

| 尺寸 | 名称 | 描述 |
|---|---|---|
| 字 32 | ZtNextEntry | 到链接列表中的下一个项的指针 |
| 字 32 | ZtCount | 这个项的计数 |
| 字 64 | ZtRegion | 区域数 |

[0451] 下面描述存储区信息布局。存储区表节点的链接列表以如下方式放置在 ZIS 后：

[0452]

| 存储区信息结构 |
| --- |
| 首存储区表节点（16 字节） |
| ...<br>末存储区表节点（16 字节） |

[0453] 这种信息存储在存储区表区域中。

[0454] 图 16 示出了根据本发明示例性实施例的驱动器布局。前两个区域是互为副本。第三（可选）存储区表区域包含该存储区表。在具有一个以上驱动器的系统中，只有两个驱动器包含 ZTR。在仅具有一个驱动器的系统中，两个区域用于保留该 ZTR 的两个（镜像）副本。DIS 包含有关 RIS 和 ZIS 位置的信息。注意 RIS 的第一副本不必在区域 0 中（例如，如果区域 0 包含坏扇区，则可以位于不同区域）。

[0455] 存储区管理器需要在系统启动时加载该存储区表。为此，其从 DIS 提取区域号和偏移。这将指向 ZIS 的开头。

[0456] 特定模块（例如 CAT 管理器）在存储区中存储它们的控制结构和数据表。层 3 和更高层中的模块的所有控制结构由存储在存储区 0 中的结构来引用。这意味着，例如，实际 CAT（聚簇分配表）位置由存储区 0 中存储的数据结构来引用。

[0457] 下表显示存储区 0 信息表格式：

[0458]

| 偏移 | 尺寸 | 名称 | 值 / 有效范围 | 描述 |
| --- | --- | --- | --- | --- |
| 0 | 字 64 | zitSignature | | 指示这是 ZIT |
| | 字 32 | zitSize | | 该结构的尺寸（字节） |
| | 字 32 | zitChecksum | | 该结构的校验和 |
| | 字 32 | zitVersion | | 该结构的版本 |
| | 字 32 | zitCATLStartOffset | | CAT 链接列表的起始字节偏移（在该存储区中） |
| | 字 32 | zitCATSize | | CAT 链接列表中的节点数。等于包含该 CAT 的存储区数 |
| | 字节 64 | zitCATAddressable | | CAT 所支持的最大 LBA。有效 CAT 尺寸 |
| | 字 32 | ZitHTStartOffset | | 散列表链接列表的起始字节（这个存储区中） |
| | 字 32 | zitHTNumberNodes | | 散列表链接列表中的节点数 |
| | 字 64 | ZitHTSize | | 按字节的散列表数据尺寸 |

[0459] CAT 链接列表是描述包含 CAT 的存储区的节点链接列表。下表显示 CAT 连接列表节点格式：

[0460]

| 尺寸 | 名称 | 描述 |
|---|---|---|
| 字 32 | catll NextEntry | 到链接列表中的下一个项的指针 |
| 字 16 | catll Count | 这个项的计数 |
| 字 16 | catll Zone | 包含 CAT 的该部分的存储区数 |

[0461] 散列表链接列表是描述保持散列表的存储区的节点的链接列表。下表显示该散列表链接列表节点格式：

[0462]

| 尺寸 | 名称 | 描述 |
|---|---|---|
| 字 32 | htll NextEntry | 到链接列表中的下一个项的指针 |
| 字 16 | htll Count | 这个项的计数 |
| 字 16 | htll Zone | 包含这个散列表部分的存储区数 |

[0463] 图 17 示例根据本发明示例性实施例的存储区 0 的布局以及其他存储区如何被引用。

[0464] 如上所述，冗余集是为数据集提供冗余的一组扇区 / 聚簇。备份某个区域包括将一个区域的内容复制到另一个区域。

[0465] 在数据读取出错的情况下，较低级软件（盘管理器或者设备驱动器）在最初失败尝试之后再做两次读请求的尝试。故障状态传递回存储区管理器。存储区管理器接着尝试根据盘片阵列中冗余聚簇来重构所请求（通过读）数据。该冗余数据可以是镜像的聚簇（用于 SDM、DDM）或者一组包括奇偶校验的聚簇（条带化实施）。接着重构数据传递回主机。如果 ZM 不能重构该数据，则将读报错传递回主机。存储区管理器发送差错通知分组给差错管理器。图 18 示例根据本发明示例性实施例的读报错处理。

[0466] 在数据写出错的情况下，较低级软件（盘管理器或者设备驱动器）在最初失败尝试之后再两次尝试写请求。故障状态传递回存储区管理器。存储区管理器发送报错通知分组给差错管理器。

[0467] 当数据写在这个级别执行时，冗余信息同样写入盘片。这样，只要仅一个聚簇具有写差错，后续的读可以重构该数据。如果有多个盘差错并且不能读或者写冗余信息，则有至少两种可能途径：

[0468] a. 向主机返回写差错状态。把与该冗余集相关联的所有区域备份至新分配的不包含坏扇区的区域。

[0469] b. 延迟写。把与该冗余集相关联的所有区域备份至新分配的不包含坏扇区的区域。随后，在新分配区域中的适当聚簇上写（连同全部冗余部分，例如奇偶校验等）。单独

的写队列将用于包含已经被延迟的写。

[0470] 因为写状态可能已经发送到主机作为日志成功写入的结果，所以方法 (a) 是有问题的，因此主机可能不知道已经有差错。一种替换是报告读的故障，但是允许写。CAT 中的某个位用于跟踪应当返回坏的读的特别 LBA。

[0471] 图 19 示例根据本发明示例性实施例的写差错处理。

[0472] 差错管理器 (EM) 检查聚簇以发现其是否真的损坏。如果是，则认为整个区域损坏。该区域中的内容复制到相同盘的新分配区域上。然后标记当前区域损坏。当在区域上复制时，当遇到坏扇区时，差错管理器将在必要时重构数据。如 20 是示例根据本发明示例性实施例的由差错管理器备份坏区域的逻辑流程图。

[0473] 如果有数据读差错且差错管理器不能重构给定聚簇的数据（例如，由于在整个冗余集的读差错），那么接着使用零代替不能重构的数据。在这种情况下，将同样必须备份其他包含坏扇区的区域（来自相同冗余集）。再次将使用零代替不能重构的数据。

[0474] 一旦执行了冗余集复制，EM 禁用对应于存储区这部分的聚簇的存取。然后更新存储区表以指向新分配区域。随后再启用对聚簇的存取。

[0475] 这个示例性实施例被设计为支持八个快照（其允许使用一个字节指示特定快照实例是否使用散列 / 聚簇项）。有两个表涉及快照：

[0476] 1. 需要存在每个快照的 CAT 表，以捕获逻辑扇区地址与包含用于 LSA 的数据的盘片上的聚簇之间的关系。最终，每快照 CAT 必须是在快照发生时 CAT 的副本。

[0477] 2. 系统散列表，其在散列值和数据聚簇之间映射。散列函数返回相同结果，无论使用哪个快照实例，并且对全部快照结果都是一样的。这样，这个表必须理解唯一聚簇是否由任何快照使用。散列聚簇项不能被释放，或者被新数据替换，除非没有使用该散列项的快照。

[0478] 总是有当前和被添加的快照。当散列项创建或者更新时，我们将需要把当前快照号应用到散列项。当制作快照时，将递增当前快照号。

[0479] 通过查找散列表并且发现任何具有退出快照位设置的散列项并清空该位，由此来释放不再被任何快照需要的聚簇 / 散列项。如果该快照字节现在是零，则散列项可以从该表中删除，并且可以释放该聚簇。

[0480] 为了预防与添加到散列树的任何新项的冲突（因为新快照号与退出快照号相同），仅允许批准 7 个快照，最后的（第八个）快照是正退出的快照。可以以后台活动来查找散列表。

[0481] 为了创建快照，无论主 CAT 何时更新，都可以写入第二 CAT 存储区。这些更新可以被排队，并且影子 CAT 可以按其他任务来更新。为了快照，影子 CAT 成为快照 CAT。

[0482] 一旦进行快照，可以离开后台处理以将这个快照表复制到新存储区成为新快照 CAT。可以使用队列，使得不处理影子 CAT 队列，直到 CAT 复制完成。如果在更新影子 CAT 之前发生故障（队列中的项可能丢失的情况），则在阵列联机之前可以根据最初 CAT 表执行再投影。

[0483] 替代地，当需要快照时，“增量”的集合加上最初 CAT 副本可以组成快照。然后后台任务可根据这些信息重建完整快照 CAT。这会需要一点或者不需要停机时间来做此快照。其间，可能要为后续的快照收集另一组“增量”。

[0484] 感知文件系统的存储系统

[0485] 如上所述，本发明的实施例对主机文件系统的数据结构（即，元数据）进行分析，以便确定所述主机文件系统的存储使用并基于所述主机文件系统的存储使用对物理存储进行管理。为了方便，该功能此后可被称作“回收器”。可包括此后被称作“监视器”的类似功能来监视存储使用，但是并非必需对物理存储进行管理。下面讨论回收器和监视器这两者。

[0486] 图 27 是根据本发明示例性实施例的计算机系统 2700 的概念框图。除其它部件之外，计算机系统 2700 包括主机计算机 2710 和存储系统 2720。除其它部件之外，主机计算机 2710 包括主机操作系统 (OS)2712 和主机文件系统 2711。除其它部件之外，存储系统 2720 包括感知文件系统的存储控制器 2721 和存储 2722（例如，包括一个或多个组装盘片驱动器的阵列）。除其它内容之外，存储 2722 被用来存储存储控制器数据结构 2726、主机 OS 数据结构 2725、主机文件系统数据结构 2723 和用户数据 2724。感知文件系统的存储控制器 2721 在存储控制器数据结构 2726 中存储各种类型的信息（由感知文件系统的存储控制器 2721 和存储控制器数据结构 2726 之间的虚线表示），诸如包括到 OS 分区的引用的分区表（由存储控制器数据结构 2726 和主机 OS 数据结构 2725 之间的虚线表示）。主机 OS 2712 在主机 OS 数据结构 2725 中存储各种类型的信息（由主机 OS 2712 和主机 OS 数据结构 2725 之间的虚线表示），典型地包括到主机文件系统数据结构 2723 的指针 / 引用（由主机 OS 数据结构 2725 和主机文件系统数据结构 2723 之间的虚线表示）。主机文件系统 2711 在主机文件系统数据结构 2723 中存储与用户数据 2724 相关的信息（被称作元数据，并且由主机文件系统 2711 和主机文件系统数据结构 2723 之间的虚线表示）。感知文件系统的存储控制器 2721 对来自主机文件系统 2711 的存储请求（由主机文件系统 2711 和感知文件系统的存储控制器 2721 之间的实线表示）进行处理，利用主机 OS 数据结构 2725 和主机文件系统数据结构 2723 来确定主机文件系统 2711 的存储使用（由感知文件系统的存储控制器 2721 和主机 OS 数据结构 2725 之间的虚线以及感知文件系统的存储控制器 2721 和主机文件系统数据结构 2723 之间的虚线表示），并且基于主机文件系统 2711 的存储使用对用户数据 2724 的存储进行管理（由感知文件系统的存储控制器 2721 和用户数据 2724 之间的虚线表示）。特别地，感知文件系统的存储控制器 2721 可实现回收器和 / 或监视器，如下所述。

[0487] 感知文件系统的存储控制器 2721 通常需要具有对于（多个）主机文件系统的内部工作的充分理解，以便对主机文件系统数据结构进行定位和分析。当然，不同的文件系统具有不同的数据结构并且以不同的方式进行操作，并且这些差异会影响设计 / 实施选择。一般来说，感知文件系统的存储控制器 2721 定位存储 2722 中的主机文件系统数据结构 2723，并且对主机文件系统数据结构 2723 进行分析来确定主机文件系统 2711 的存储使用。感知文件系统的存储控制器 2721 接着能够基于这样的存储使用对用户数据存储 2724 进行管理。

[0488] 图 28 是根据本发明示例性实施例的用于感知文件系统的存储控制器 2721 的高级别逻辑流程图。在框 2802，感知文件系统的存储控制器 2721 定位存储 2722 中的主机文件系统数据结构 2723。在框 2804，感知文件系统的存储控制器 2721 对主机文件系统数据结构进行分析来确定主机文件系统的存储使用。在框 2806，感知文件系统的存储控制器 2721

基于所述主机文件系统存储使用对用户数据存储进行管理。

[0489] 图 29 是根据本发明示例性实施例的用于定位主机文件系统数据结构 2723 的逻辑流程图。在框 2902，感知文件系统的存储控制器 2721 在存储控制器数据结构 2726 中定位其分区表。在框 2904，感知文件系统的存储控制器 2721 对所述分区表进行解析来定位包含主机 OS 数据结构 2725 的 OS 分区。在框 2906，感知文件系统的存储控制器 2721 对所述 OS 分区进行解析来识别主机 OS 2712 并定位主机 OS 数据结构 2725。在框 2908，感知文件系统的存储控制器 2721 对所述主机 OS 数据结构 2725 进行解析来识别主机文件系统 2711 并定位主机文件系统数据结构 2723。

[0490] 一旦感知文件系统的存储控制器 2721 定位了主机文件系统数据结构 2723，其就对所述数据结构进行分析来确定主机文件系统 2711 的存储使用。例如，感知文件系统的存储控制器 2721 可使用主机文件系统数据结构 2723 以便进行诸如识别不再由主机文件系统 2711 所使用的存储块和识别主机文件系统 2711 所存储的数据类型之类的事情。感知文件系统的存储控制器 2721 接着能够基于数据类型动态地回收主机文件系统 2711 不再使用的存储空间和 / 或管理用户数据 2724 的存储（举例来说，例如将频繁访问的数据以未压缩的形式存储在连续块中以促进访问，将非频繁访问的数据进行压缩存储和 / 或存储在非连续块中，并且基于数据类型应用不同的编码模式）。

[0491] 图 30 是根据本发明示例性实施例的用于回收未使用的存储空间的逻辑流程图。在框 3002，感知文件系统的存储控制器 2721 识别被标记为未被主机文件系统 2711 使用的块。在框 3004，感知文件系统的存储控制器 2721 识别被标记为未被主机文件系统 2711 使用但是被标记为被感知文件系统的存储控制器 2721 使用的块。在框 3006，感知文件系统的存储控制器 2721 回收被标记为被感知文件系统的存储控制器 2721 使用但是不再被主机文件系统 2711 使用的块，并且使得所回收的存储空间可用于其它存储。

[0492] 图 31 是根据本发明示例性实施例的用于基于数据类型对用户数据 2724 的存储进行管理的逻辑流程图。在框 3102，感知文件系统的存储控制器 2721 识别与特定用户数据 2724 相关联的数据类型。在框 3104，感知文件系统的存储控制器 2721 可选地使用基于所述数据类型所选择的存储布局来存储特定用户数据 2724。在框 3106，感知文件系统的存储控制器 2721 可选地使用基于所述数据类型所选择的编码模式（例如，数据压缩和 / 或加密）对特定用户数据 2724 进行编码。这样，感知文件系统的存储控制器 2721 能够使用针对数据类型定制的不同布局和 / 或编码模式来存储不同类型的数据。

[0493] 回收器的一个示例是所谓的“垃圾收集器”。如上所述，垃圾收集器可用于释放不再被主机文件系统使用的聚簇（例如当删除文件时）。一般而言，通过发现空闲块、计算它们的主 LSA、并根据该 LSA 分配它们的 CAT 项进行垃圾收集。如果没有 CAT 项用于特定 LSA，则该聚簇已经空闲。但是如果 CAT 项被定位，则递减引用计数，而且如果该计数命中零，该聚簇空闲。

[0494] 对于垃圾收集器而言，一个问题是难于将正在由主机文件系统使用的块与先前已经使用并且在某个点标记为空闲的块区分开来。当主机文件系统写入块时，存储设备分配聚簇用于数据，以及 CAT 项来描述它。从这点上，聚簇一般将显示为在用 (in use)，即使主机文件系统随后停止使用它的块（即，聚簇将仍然通过有效 CAT 的项处于在用状态）。

[0495] 例如，特定主机文件系统使用位图 (bitmap) 来跟踪它使用的盘片块。一开始，位

图将指示全部的块为空闲，例如，通过将全部位清空。由于使用文件系统，因此主机文件系统将通过使用它的空闲块位图来分配块。存储系统将通过上述分配聚簇和 CAT 项来把这些文件系统分配与物理存储相关联。当主机文件系统把一些块释放回到它的空闲池时，其只需要在其空闲块位图中清空相应的位。在存储系统上，这可以设想成写入恰好包含该主机的空闲块位图的一部分的聚簇，就像没有 I/O 到空闲的实际聚簇本身（虽然可能有到空闲聚簇的 I/O，例如，如果主机文件系统以某种增强安全模式运行，其中可能把零或者随机数据的强保密散列写入聚簇以使得减少旧聚簇内容可能被攻击者读取的机会）。另外，当满意新分配请求时，不保证主机文件系统会重用以前被释放的块。因此，如果主机文件系统继续分配那些从存储系统的视点来看是新的、即以前未使用的块，则该存储系统将快速耗尽空闲聚簇，限于无论什么空间都可以经由压缩回收。例如，假定文件系统块是 4K，如果主机分配文件系统块 100 至 500，随后释放块 300 至 500，然后接着分配块 1000 至 1100，整个文件系统使用的将是 300 块，而阵列将有 500 个聚簇处于在用状态。

[0496] 在本发明示例性实施例中，存储系统可以通过访问主机文件系统布局来检测主机文件系统盘片资源的释放，解析它的空闲块位图，并使用该信息来识别不再被该文件系统使用的聚簇。为了存储系统能够以这种方式识别未使用的聚簇，存储系统必须能够定位并理解该文件系统的空闲块位图。因而，该存储系统将通常支持文件系统的预定设置，可充分“理解”内部工作以定位并利用这些空闲块位图。对不支持的文件系统，该存储系统可能不能够执行垃圾收集并将由此仅告之该阵列的实际物理尺寸以便于避免被过量使用。

[0497] 为了确定该文件系统类型（例如 NTFS、FAT、ReiserFS、ext3），需要定位该文件系统的超块（superblock，或者等同结构）。为了发现该超块，要解析分区表（partition table）以定位 OS 分区。假定 OS 分区被定位，则解析 OS 分区，以试图定位该超块并由此识别该文件系统类型。一旦该文件系统类型已知，则可以解析布局以查找空闲块位图。

[0498] 为了便于搜索空闲块，可以保持主机文件系统位图的历史数据，例如，通过制作可以存储在私有、非冗余存储区的空闲块位图的副本，并使用该副本执行搜索。给定位图的尺寸，可以每次为较少数量的聚簇保持信息而不是为整个位图保持信息。当执行垃圾收集时，可以把当前空闲块位图与历史副本逐个聚簇地进行比较。可以识别任何从分配转换成空闲的位图项，使得回收操作可准确地定向到作为用于回收的良好候选的聚簇。随着处理每个位图聚簇，可以用当前副本替换历史副本以维持位图操作的滚动式历史。空闲块位图的副本将随着时间变成时间上不连贯的聚簇的拼凑，但是由于当前副本总是用于定位空闲项，因此这并不产生任何问题。

[0499] 在特定条件下，会有关于空闲块位图的竞争条件，例如，如果主机文件系统使用其空闲块位图来分配盘片块，接着写入它的数据块，然后将修变的位图刷新回到盘片。在这种情况下，垃圾收集器会释放该聚簇，即使该文件系统在使用该聚簇。这会导致文件系统被破坏。可实现存储系统以避免或者处理这样的条件。

[0500] 由于垃圾收集可能是相当昂贵的操作，因即使低强度的回收也将占用后端的 I/O 带宽，因此不应滥用垃圾收集。垃圾收集器应能够按多种方式运行，从低强度的后台惰性回收到非常高强度或者甚高优先级的回收。例如，当使用了百分之三十的空间时，可以按低强度方式运行垃圾收集器，或者至少每星期一次，当使用了 50％的空间时，按稍微高强度的方式来运行，而当使用了百分之九十或者更多的盘片空间时，运行全高优先级的回收。在每次

进行收集时，可以限制要回收的目标聚簇数量和最大的可容许 I/O 计数，从而控制垃圾收集器的回收强度。例如，可配置垃圾收集器，通过使用不超过 10000 次的 I/O 来回收 1GB。获得回收请求的失败可用作对收集器的反馈，从而在下一次运行时按更高强度的方式。还可以是“回收一切”模式，允许垃圾收集器解析整个主机文件系统空闲块位图并回收可能的全部块。在阵列（几乎）完全装满时，这可以作为最后的办法来回收聚簇。可以周期性地运行垃圾收集器，对其施加规则，并可决定执行、或者可决定不执行回收操作。还能够明确地从其他模块来请求回收操作，例如区域管理器，当正在寻找用于构建区域的聚簇时，可请求回收操作。

[0501] 垃圾收集功能可以与状态指示符机制相结合。例如，在某些点，存储系统可以处于“红色”条件，尽管正在运行的垃圾收集操作可以释放足够空间以消除“红色”条件。可采用附加指示器状态以显示相关状态信息（例如可以使红色指示器灯闪烁来指示垃圾收集操作正在进行）。

[0502] 图 21 是根据本发明示例性实施例的存储阵列的相关部件的示意性框图。除了其他，存储阵列包括底盘 2502，在其上存储管理器 2504 与多个存储设备 $2508_1$-$2508_N$ 通信，这些存储设备分别通过多个插槽 $2506_1$-$2506_N$ 耦合到底盘。每个插槽 $2506_1$-$2506_N$ 都可以与一个或者多个指示器 $2507_1$-$2507_N$ 相关联。除了其他，存储管理器 2504 典型包括用于实施上述功能的各种硬件和软件组件。硬件组件典型包括存储器用于存储诸如程序代码、数据结构、以及数据的内存，和用于执行该程序代码的微处理器系统。

[0503] 实现感知文件系统的存储控制器 2721 的一个问题在于许多主机文件系统不对数据结构（即，元数据）进行实时更新。例如，日志文件系统通常不保证保存用于已经发生的事务的所有用户数据，或者不保证恢复用户这些事务的所有元数据，而是通常仅保证恢复到一致状态的能力。对于性能和效率而言，日志文件系统通常在用户数据写入和元数据写入之间采取一定程度的异步。特别地，通常惰性地执行元数据到盘片的写入，以使得在用户数据更新和相应的元数据更新之间存在延迟。在一些文件系统中（诸如依据第四版 Microsoft Windows 内核的 NTFS）还惰性地执行日志写入。此外，惰性的元数据写入可通过以逐个事务的方式结束日志来执行，并且这具有相当的潜在可能将元数据临时推入与已经在盘片上的用户数据不一致的状态。这样的示例可以是在主机已经重新分配了聚簇并且发送了与重新分配的聚簇相对应的用户数据之后显示出解除分配的位图更新。因此，存储系统通常需要对不可靠地指示用户数据的当前状态的元数据更新进行处理。在之前的示例中，这通常意味着存储系统无法解释解除分配来表示所述聚簇是可回收的且用户数据是可丢弃的。

[0504] 如果元数据和日志更新与和它们相对应的用户数据更新完全异步，以使得它们能够在任意时间发生，则存储系统可能需要具有对于文件系统的内部工作的相对详细的理解，并且由此可能需要存储大量状态信息以便进行适当的决策。然而，以下所描述的本发明的特定实施例被设计成在元数据更新将在写入与它们相关的用户数据之后的相对确定的时间窗内发生（例如，一分钟内）的假设下进行操作。要理解的是，这样的实施例本质上在复杂度和功能性之间进行折衷，因为虽然需要进行特殊的考虑以便对操作期间不附着到这样的窗口的主机文件系统（一个示例可以是 VxFS）或者导致用户数据更新和相应元数据更新之间的长期延迟（例如，丢失了来自主机的功能，这通常超出了存储系统的控制并总之

可能会导致数据丢失，或连接丢失，存储系统能够检测到它并由此抑制假设时间性主机活动的行为）的边界条件进行处理，但是它们通常不需要具有对于文件系统内部工作的详细理解和存储大量状态信息。

[0505] 在一个示例性实施例中，所述回收器能够以完全异步的方式进行操作。在该实施例中，所述回收器可以是完全的异步任务，其周期性地对位图进行全部或部分扫描，并且将所述位图与 CAT 中所包含的信息进行比较以确定是否能够回收任意的存储阵列。在检查位图之前，所述系统还可以检查包含所述位图的位置的那些块以便确定所述位图是否已经移动。

[0506] 完全异步的回收器的一个优点在于，虽然可能包括实质上异步的盘片 I/O（例如，对于逻辑上被划分为 4k 聚簇并具有 64MB 位图的 2TB 卷而言，读取整个位图将包括在回收器每次运行时要读取 64+MB 的盘片数据）并且由此可能会根据回收器多么频繁的运行而影响到整体的系统性能，但是其本质上对于主要数据路径内的处理器开销没有直接影响。因此，回收器频率可根据可用存储空间和 /. 或系统负载的量进行变化。例如，当可用存储空间充足或系统负载高时，可较不频繁地运行回收器功能。降低回收器频率通常会降低回收存储空间的速度，在存储空间充足时这通常是可接受的。另一方面，当可用空间不足且系统负载低时，可更为频繁地运行回收器功能以便提高回收存储空间的速度（以增加的处理开销为代价）。

[0507] 在另一个示例性实施例中，回收器可以以部分同步、部分异步的方式运行。在该实施例中，例如，所述回收器能够通过向主要写处理路径添加一些附加的检查来在位图变化发生时对其进行监视。所述回收器可在引导时建立包括（多个）感兴趣的 LBA 范围的表（此后被称作位图定位器表或 BLT）。对于未初始化的盘片或已初始化但未分区的盘片而言，BLT 通常仅包括 LBA 0。对于完全初始化和格式化的盘片而言，BLT 通常将包括 LBA 0、每个分区的引导扇区的（多个）LBA、包含位图元数据的（多个）LBA 以及包含位图数据自身的（多个）LBA 范围。

[0508] 主要写处理路径（例如，HRT）通常利用被处理的写入的细节调用所述回收器，在这种情况下，考虑到识别与感兴趣的（多个）LBA 范围重叠的那些写入，所述调用一般利用 BLT 在内部交叉引用写请求的（多个）LBA。所述回收器接着将需要对那些写入进行解析，这主要利用异步任务来进行（在这种情况下，通常需要存储异步任务的关键细节，如下所述），但是具有重要的写入解析内嵌（例如，如果更新潜在指示重新定位的位图，则该写入可以是经解析的内嵌从而可在任意的其它写入被交叉引用之前对 BLT 进行更新）。如以上参考完全异步的回收器所讨论的，异步任务的频率可根据可用存储空间量和 / 或系统负载进行变化。

[0509] 用于异步任务的存储可以为队列的形式。然而，简单队列将允许对针对相同块的多个请求进行排队，由于写入缓存的语义使得多个请求可能指向缓存中的相同数据块（即，最近数据）所以会发生这种现象，并且因为通常没有理由保存表示相同 LBA 的多个请求所以是低效的。可通过对队列进行检查并且移除针对相同块的较早请求来缓解这一问题。此外，假设异步任务的频率根据可用存储空间量和 / 或系统负载进行变化，则应当利用其会在密集活动期间（可能会持续数日）达到其最大尺寸的抑制异步任务的预期来规定所述队列。假设系统不允许针对相同 LBA 的多个项，所述队列的最大理论大小是 LBA 的大小

和位图内 LBA 数目的乘积，这会产生非常大的队列尺寸（例如，具有 64MB 位图的 2TB 卷，即 128K 个块），并由此可能需要 128K*4 = 512K 级别的队列大小；16TB 的卷可能需要 4MB 级别的队列大小。

[0510] 用于异步任务的可选存储可以是位图的位图的形式（此后被称作“位图块更新位图”或“BBUB”），其中每一位表示实际位图的一个块。BBUB 固有地避免了针对相同块的多个请求，原因在于针对这些请求中的每一个设置相同的位，从而多个请求在 BBUB 中仅显示一次。此外，BBUB 的大小基本上是固定的，与异步任务的频率无关，并且通常比队列占据更小的空间（例如，BBUB 对于 2TB 卷将占据 16KB 存储器，或对于 16TB 卷占据 128KB）。在实际位图移动的情况下，所述存储系统能够容易地对 BBUB 中位的映射进行调节，但是通常需要注意不要在主机已经交叉复制数据之前将未决的请求映射到新的位置（实际上，在假设主机文件系统无论如何都将重写每个 LBA 的情况下，可能在位图之外达到零）。BBUB 可被置于非易失性存储器 (NVRAM) 以防止丢失当前的 BBUB，或者被置于易失性存储器中，其中理解为当前的 BBUB 可被丢弃并且需要在重新引导之后的某个时刻运行位图的完全扫描以恢复丢失的信息。由于位图并非固有地提供具有多个请求的就绪测量的异步任务，所以所述存储系统能够保存与位图中所设置的位数相关的统计以使得所述异步任务不必仅为了确定没有内容需要更新而扫描整个位图。例如，所述存储系统可能保存在位图中设置有多少位的计数并且可以基于所述计数对异步任务的频率进行调节（例如，不允许所述异步任务除非且直至所述计数达到预先确定的阈值，所述阈值可以是用户可配置的）。这样的策略可以进一步改进，例如通过为多个映射部分（例如，1K 个块）中的每一个保存所设置的位的单独计数并且还对具有最高计数的映射部分进行跟踪，从而所述异步任务能够仅对可能返回最高回报的（多个）映射部分进行解析。

[0511] 对更新进行解析通常包括用于不同 LBA 的不同逻辑。例如，LBA0 的改变通常意味着已经添加了分区表，或者已经向表中添加了分区，或者已经删除了分区。对于分区引导扇区的更新可能意味着已经重新定位的位图元数据。对于位图元数据的更新可能意味着已经移动了所述位图，或者其已经被扩展。对于位图自身的更新可能指示聚簇的分配或解除分配。如果异步对更新进行解析，则所述系统通常无法轻易地将旧数据与新数据进行比较，原因在于到异步任务的运行时间时，新的数据可能已经覆盖了旧数据。为了避免这一问题，所述系统可能保留旧数据的单独副本以便进行比较或可能通过映射进行反复并将未设置的位与 CAT 进行比较（这可能需要略微更多的处理器开销但是较少的盘片 I/O）。将位图与 CAT 进行简单比较一般要求附加的逻辑和状态信息，原因在于位图状态可能与用户数据不同步，如上所述。此外，保留位图数据的副本将允许存储系统将新的数据与旧数据进行比较并且由此准确确定什么发生了变化，但是如上所述，与其可能依赖于状态本身相比，所述存储系统通常无法再依赖状态变换作为当前用户数据的精确考量。

[0512] 在又另一个示例性实施例中，所述回收器能够以完全同步的方式进行操作。在该实施例中，回收器将在写入发生时对它们进行处理。完全同步的实施例的一个优点在于其避免了与异步任务及其相关联的存储相关联的复杂度，虽然其在对来自主机的写入进行处理的临界时间期间在处理器上突然插入了开销，并且可能需要附加的逻辑和状态信息来对异步元数据更新进行补偿。

[0513] 在异步位图更新的上下文中回收聚簇的一个问题在于基于没有精确反映用户数

据的状态的位图值，回收器可能不适当地释放聚簇。为了克服这样的问题，存储系统可保留其执行的聚簇访问的一些历史（例如，最近是否访问过聚簇中的用户数据）并且仅在聚簇在先前某个时间间隔中是静止的情况下回收所述聚簇以确保没有元数据更新对于该聚簇是未决的。例如，所述存储系统可能在执行任意聚簇的回收之前要求聚簇静止至少一分钟（一般而言，增加静止时间降低不适当回收的风险但是增加了对数据删除进行反应的等待时间，从而在此要进行折衷）。虽然以附加的盘片 I/O 为代价，所述存储系统能够在评估聚簇活动中为了完整而附加地对聚簇读取进行跟踪，但是所述存储系统可以仅对聚簇写入进行跟踪。静止时间可以是固定值或者对于不同的文件系统可有所不同。

[0514] 例如，可通过向 CAT 写入回收器周期号作为访问时间相对于回收器运行的指示器而对聚簇访问进行跟踪。

[0515] 作为选择，可以通过在写数据之前向文件系统的位图写入位而对聚簇访问进行跟踪。虽然，为了避免与文件系统操作的任何不利交互，但是文件系统的元数据的任意这种修改都必须仔细进行调整。

[0516] 作为选择，可使用每个聚簇、块或大块（或任何大小）的位对聚簇访问进行跟踪。所述位通常在该实体被访问时进行设置并且可能在回收器完成其下一次运行或回收器下一次试图回收聚簇时进行重置。所述回收器通常仅在该位已经在试图执行回收时被重置的情况下回收所述聚簇，所述回收本身要由被清除的实际主机文件系统位图中的对应位进行驱动。这些位可保存在一起作为简单的位图或者可以被添加到 CAT 作为分布式位图（每个 CAT 记录需要一个附加位）。所述简单位图的方法可要求在大多数数据写入操作上进行附加的读 - 改 - 写 (read-modify-write)，潜在地导致主要数据路径的性能下降，除非所述位图在存储器中进行缓存（所述位图可缓存在易失性存储器中，如果所述位图由于不可预期的损耗量而丢失这可能是有问题的，或者缓存在非易失性存储器中，由于存储器约束以及由此产生的较小粒度，较小的位图可能成为必要）。所述 CAT 方法通常会从 J2 及其空闲的 NVRAM 缓存获益。

[0517] 作为选择，可通过保存在接收位图更新和执行聚簇修改时的时间戳对聚簇访问进行跟踪。接着，如果聚簇修改具有比位图更新更晚的时间戳，则所述系统通常不会释放所述聚簇。该方法优于位方法的一个优势在于回收器能够确定最后的访问发生了多久，并且如果足够长，则立即回收聚簇。还可以向 CAT 记录添加时间标签。作为选择，由于该字段真正地仅需要指示相对于回收器操作的时限，所以可对每次回收器运行指定全局标识符。所述系统接着可使用 CAT 内的类似字段来显示全局标识符的值。所述全局标识符可识别最近完成了哪个回收器运行，或者接下来应进行哪个回收器运行，或者聚簇何时被最后访问。该信息接着可被回收器用作时限的测量。为了节约 CAT 记录中的空间消耗，所述标识符可以仅为一个字节的计数器。由于计数器覆盖 (wrapping) 所产生的任意不正确的时限确定会使得旧的聚簇看起来比它们实际年轻很多。这些聚簇将在下一次运行被回收。所述字段可存储在 NVRAM 中以防止所述字段在每次重新引导时被重置为零，这将导致一些聚簇过早地访问时限。

[0518] 由此，例如，每个回收器运行可以与一个字节的标识符值相关联，其可被实现为 NVRAM 中的全局计数器，在回收器每次醒来时增加以使得回收器运行的标识符是所述计数器的后增量 (post-increment) 值。CAT 管理器可在其对聚簇的更新进行服务的任何时刻使

用所述全局计数器的当前值并且能够在对应的CAT记录中存储该数值的副本。这样的实施方式需要对CAT管理器的逻辑进行修改。

[0519] 作为选择，可通过在覆盖列表中保存聚簇更新的短暂历史来跟踪聚簇访问。所述回收器然后能够搜索所述列表来验证即将空闲的任意聚簇最近没有被主机所访问。所述列表的大小通常是特定于实施方式的。无论其多长，存储系统一般必须确保其在所述列表变满之前能够运行异步任务，并且会危及对任务进行延迟直至安静周期的能力。

[0520] 在支持回收的存储系统中，可能希望对过早的回收进行识别和跟踪，特别是由于过早的回收导致失败的读取（即，从已经被回收器释放的聚簇进行读取的尝试），而且还有对未分配聚簇的写入（这一般仅会导致分配且由此应当是无害的）。在一些情况下，可能要基于文件系统的位图来识别错误（例如，从用户数据写入对于位图进行交叉引用并且检查适当的位被设置），但是仅在保证位图更新在用户数据之前完成的情况下才会如此，并非一直是这样。作为选择，当对位图进行解析时，所述回收器可能检查所分配的位是否实际对应于所分配的聚簇；如果它们不是，则分配聚簇或至少分配CAT记录；在每个这样的CAT记录中设置指示分配是从回收器强制进行的位；位由对聚簇的数据写入进行重置；并且在下一次回收器运行时再次检查所述位，并且如果其仍然设置则发出叫声(scream)。可包括其它的自我诊断来为我们给出文件系统这样做和做多少的测量，诸如聚簇回收由于该防止性测量而中止的次数的计数。

[0521] 应当注意的是，以上所述的三种类型的回收器仅是示例性的，并非将本发明限制到特定的设计或实施方式。每个回收器类型具有某些相对优点和缺点，这可能使得特别适于或不适于特定的实施方式。此外，应当注意的是，特定的实施方式能够支持多于一个的回收器类型并且按照需要在它们之间进行动态切换，例如基于诸如主机文件系统、可用存储空间量和系统负载之类的内容。部分同步、部分异步回收器、使用BBUB存储异步任务的信息以及在CAT内使用字节大小的回收器运行计数器（作为类型的时间戳）来跟踪聚簇访问被预期用于特定的实施方式。

[0522] 除了或代替回收器，可使用单独的监视器来跟踪多少聚簇被主机文件系统所使用（例如，如果已知主机文件系统可靠地重新使用解除分配的块优先于使用新的块，则可以省略回收器，从而不需要回收且监视就足够了；在实现回收器的系统中，监视器由于重复可省略）。一般而言，所述监视器仅需要确定在位图中设置了多少位，并且无需准确知道哪些位被设置和哪些位被清除。此外，所述监视器可能不需要精确的位计数，而是可以仅需要确定所设置位的数目是多于还是少于特定阈值或者所述数目多于还是少于相同区的先前值。因此，所述监视器无需对整个位图进行解析。对于上述回收器实施例而言，可以使用异步任务全部或部分实现监视器功能，这能够周期性地将新的数据与CAT进行比较，或者保存位图的副本并在以新数据覆盖所述副本之前将当前位图（新数据）与所述副本（旧数据）进行比较。

[0523] 为了便利，以下主要在NTFS的背景下参考回收器对各种设计的操作考虑进行讨论。然而，应当意识到的是，许多设计和操作考虑同样地适用于监视器。

[0524] 图32是示出根据本发明示例性实施例的回收器3210的相关部件的示意性框图。除其它部分之外，回收器3210包括位图块更新监视器(BBUM)3211、包括用于每个LUN的BLT的位图定位器表(BLT)集合3212、包括用于每个分区的一个BBUB的BBUB集合3213、异

步任务 3214 和解除分配的空间表 (DST)3215。以下对这些部件中的每个进行更为详细的讨论。而且如以下更为详细的讨论，通过从 HRM3220 接收的调用对 BBUM 3211 通知写操作。

[0525] 回收器 3210 包括用于每个 LUN 的 BLT 3212。每个 BLT 3212 包含一系列记录，其包括分区标识符、LBA 范围、LBA 范围的角色指示以及指示该 LBA 范围是应当被同步还是异步解析的标志。每个 BLT 具有用于 LBA 0 的项，其是独立于分区的。所述 BLT 通常被要求在该 LUN 上提供对于到来写入的基于 LBA 的快速查找（不首先检查它们属于哪个分区）并且提供针对 LBA 0 写入的基于分区的相对快速的查找（例如，这可以使用用于存储和用于查找的分类矢量，开始 LBA 的较低边界二进制搜索加上先前元素是否具有比被查找的 LBA 更高的最后 LBA 的检查来实现）。例如，所述 BLT 通常需要在 LoadDiskPack 调用期间在任意主机写入通过之前被编程。BLT 利用 LBA 0 被编程并且分区创建由此包括向该 LBA 进行写入，所述 LBA 0 在此是分区表的位置。LBA 0 将在该表内被标记为其更新需要立即解析的位置。

[0526] 回收器 3210 包括所述存储系统所支持的每个分区的 BBUB 3213。每个 BBUB 3213 的大小适合于其属于的文件系统的大小。每个 BBUB3213 与反映位图中设置了多少位的计数器相关联。所述 BBUB 3213 也有一些映射信息，所述映射信息示出每个位图如何属于与其相对应的文件系统位图。

[0527] 回收器 3210 包括用于每个 LUN 的 DST 3215。每个 DST 3215 包括每个记录的一个 LBA 范围。表中存在的每个 LBA 范围是需要从 CAT 回收的所删除或截短的分区的一部分。例如，BBUM 3211 可在其识别用于在同步处理期间回收的未使用存储区域时更新 DST 3215（在这种情况下，BBUM 3211 向 DST 3215 添加 LBA 范围）。类似地，异步任务 3214 例如可在其识别用于在异步处理期间回收的未使用的存储区域时更新 DST 3215（在这种情况下，其向 DST 3215 添加 LBA 范围）。所述异步任务 3214 使用 DST 3215 异步地回收未使用的存储空间。DST3215 可以以对于未清除停止操作具有弹性的方式持久存储，或者可提供附加逻辑来从 DST 3215 的任意丢失进行恢复，例如，通过在引导之后执行完全扫描以找到不属于任意卷的分配聚簇。

[0528] BBUB 3213 和 DST 3215 的存储是特定于实施方式的决策。在示例性实施例中，BBUB 3213 对于存储在 NVRAM 中过大，由此可在易失性存储器中或盘片上进行存储，而 DST 3215 可存储在非易失性存储器、易失性存储器中或盘片上。如果 DST 3215 和 BBUB 3213 完全是易失性的，则回收器 3210 通常必须能够从 DST 3215 和 BBUB 3213 的丢失（例如，由于不可预期的停止操作）进行恢复。例如，可通过对整个 CAT 进行扫描并且将其与当前分区和聚簇位图信息进行比较来看是否每个聚簇都映射到已知分区以及其是否分配在对应文件系统的聚簇位图中来实现恢复。另一种可能性是将 DST 3215 存储在 NVRAM 中而将 BBUB 3213 留在易失性存储器中以使得卷外的盘片空间的状态信息跨重新引导而保留（潜在地防止需要向 CATM 查询关于分区之外的聚簇），虽然聚簇位图的当前状态会丢失，但是有必要对每个聚簇位图进行完全扫描。例如，通过将所有需要的状态信息存储在盘片上并且将仅在引导上重新加载它，可减少或消除这样的位图扫描。因为回收器 3210 无法预期停止操作的通知，所以需要与实时状态密切同步地保存记录，或者通过同步地更新它们或仅在数毫秒或数秒内将它们写回来保存它们。似乎有理由假设有意图的停止操作之前主机会数秒钟不活动，即使主机实际上没有停止操作，从而在几秒内进行更新对于大多数情况是足够

的；虽然在 I/O 中间发生的停止操作似乎仍需要完全扫描。如果记录被同步更新（即，在向盘片写入位图更新之前），则所述系统可能能够完全消除状态丢失以及在引导时进行完全扫描的需要（虽然以在稳定状态操作期间需要更多盘片 I/O 以获得更好的引导效率为代价）。另一个选择是在系统停止操作过程期间向盘片写入 BBUB3213 和 DST 3215，以使得在重新引导时可获取所述信息（除非在不可预期的失败 / 停止操作的情况下，在这种情况下，可能需要在重新引导时对聚簇进行完全扫描）。

[0529] 虽然在示例性实施例中，预期到在对诸如 CAT 管理器或缓存管理器（用于从盘片组进行读取）以及 NVRAM 管理器（用于增加计数器）之类的回收器所依赖的模块进行初始化之后系统管理员要对回收器 3210 进行初始化，但是一般而言，回收器 3210 在加载盘片组之前没有很多事情可做。作为选择，所述回收器可以惰性初始化，例如在加载盘片组之后。由于所述回收器可几乎立即开始从盘片组进行读取，所以所述回收器不应被指导来加载所述盘片组（即，LoadDiskPack）直至其它部件就绪并且已经加载了相同的盘片组自身。

[0530] 在回收器 3210 的初始化期间（或者在一些其它适当的时刻），BBUM 3211 在 LBA 0 查找 NTFS 分区表。所述 NTFS 分区表是位于与主引导记录相同的 LBA（即 LBA 0）中的 64 字节的数据结构，并且包含与 NTFS 主分区相关的信息。每个分区表项为 16 字节长，使得最多有 4 项可用。每项在从扇区和预定结构的开始的预定偏移量处开始。所述分区记录包括系统标识符，其使得所述存储系统能够确定分区类型是否是 NTFS。已经发现分区表位置和布局通常有些独立于写入其的操作系统，其中相同的分区表服务于某个范围的文件系统格式，不仅仅是 NTFS，并且不仅仅是 Microsoft 格式 (HFS+ 和可使用不同结构定位其分区的其它文件系统）。

[0531] 假设在 LBA 0 找到 NTFS 分区表，则 BBUM 3211 从 LBA 0 读取所述分区表，并且接着对于分区表中所识别的每个 NTFS 分区，读取所述分区的引导扇区（该分区的第一扇区），具体地是扩展的 BIOS 分区块，其是属于 NTFS 的将提供主文件表 (MFT) 的位置的结构。BBUM3211 接着读取 MFT 的常驻 $ 位图记录来得到文件属性，具体地是实际位图数据的（多个）位置和（多个）长度，BBUM 3211 还利用每个分区的引导扇区 LBA、（多个）位图记录的（多个）LBA 以及实际位图的 LBA 对 BLT 3212 进行编程。引导扇区 LBA 和位图记录 LBA 还将被标记为其更新永远需要立即解析的位置。实际位图通常不需要立即解析并且将因此被标记。如果在 LBA 0 没有找到分区表，则没有附加位置被添加到 BLT 3212。

[0532] 图 33 是根据本发明示例性实施例的用于定位主机文件系统位图的伪代码。感知文件系统的存储控制器 2721 首先在 LBA 0 查找分区表。假设找到所述分区表，则感知文件系统的存储控制器 2721 读取所述分区表以识别分区。接着，对于每个分区，感知文件系统的存储控制器 2721 读取所述分区的引导扇区以找到 MFT，并且读取 MFT 的常驻 $ 位图记录以得到文件属性，诸如实际位图的（多个）位置和（多个）长度。感知文件系统的存储控制器 2721 利用每个分区的引导扇区 LBA、（多个）位图记录的（多个）LBA 以及（多个）实际位图的（多个）LBA 对 BLT 进行编程，并且将（多个）引导扇区 LBA 和（多个）位图记录 LBA 标记为需要立即解析而将（多个）实际位图标记为不需要立即解析。如果感知文件系统的存储控制器 2721 无法在 LBA 0 找到分区表，则感知文件系统的存储控制器 2721 结束而不向 BLT 添加附加位置。

[0533] 在稳定状态操作期间，将通过从 HRM 3220 到 BBUM 3211 的调用针对 BLT 3212 交

叉引用所有写入。按照指示该动作的标记，被发现寻址到 LBA 0 的任意写入将被立即解析（同步地）。后续动作取决于更新的性质。

[0534] 如果添加分区，并且所述分区是认可的类型，则新分区的第一 LBA 将被添加到 BLT 3212 并且被标记为其更新永远需要立即解析的位置。预见到马上存在具有一系列更新、要驱动聚簇回收的位图，将清除 DST3215 中落入新分区内的任何 LBA 范围。一个问题在于，如果所述分区要在分区表更新之前写完，则该信息潜在地被写到 DST 3215 中的块中，并且可被回收器线程不正确地回收。例如，这可以通过检查每个所接收的写入以便与 DST 3215 中的范围相一致并且从 DST 3215 中移除任意的写入的块 (written-to-block) 而被缓解。

[0535] 如果所述分区被更新为将分区标识符从 Windows 缺省改变为 NTFS，则 BBUM 3211 将立即重新检查处于所述分区引导扇区位置的 LUN，因为标识符变化趋于在分区引导扇区已被写入之后发生。这实际上仅是分区添加的一部分。

[0536] 如果现有分区被删除，则 BLT 将被冲洗属于删除分区的记录，用于该分区的 BBUB 3213 将被删除，并且 LBA 范围将被添加到 DST 3215 以便聚簇的异步回收。

[0537] 如果现有分区被重新分配，则利用新的引导扇区 LBA 将 BLT 3212 中的现有引导扇区记录更新到监视器。在其已经被写入的情况下，可能要在新的位置对 LUN 立即进行重新检查，但是通常不这样做。

[0538] 如果现有分区被截短，则将切除的 LBA 范围添加到 DST 3215。在新的引导扇区已经被写入的情况下，可能在分区引导扇区的位置对 LUN 立即进行重新检查，但是通常不这样做。

[0539] 如果现有分区被放大，则将清除 DST 3215 中落入新分区内的任何 LBA 范围。在新的引导扇区已经被写入的情况下，可能在分区引导扇区的位置对 LUN 立即进行重新检查，但是通常不这样做。

[0540] 按照指导该动作的标记，所发现的要被寻址到分区的第一 LBA 的任何写入都要被立即（同步）解析。位图记录的开始 LBA 将被确定并添加到 BLT 3212，并且被标记为其更新永远需要立即解析的位置。

[0541] 图 34 是根据本发明示例性实施例的用于 BBUM 3211 的高级别伪代码。当 BBUM 3211 接收到客户端请求时，其从 ClientRequest 得到 LUN 并且基于所述 LUN 找到正确的 BLT。BBUM 3211 从 ClientRequest 得到 LBA，在 BLT 中查找该 LBA，并且检查“立即动作”字段来看是否需要对该 LBA 进行立即动作。如果需要进行立即动作，则 BBUM3211 同步处理所述客户端请求。然而，如果不需要进行立即动作，则 BBUM 3211 设置与所述 LBA 相对应的 BBUB 位以便进行异步处理。

[0542] 图 35 是根据本发明示例性实施例的用于对创建新分区的 LBA 0 更新进行同步处理的高级别伪代码。特别地，如果需要进行立即动作并且块是分区表，则 BBUM 3211 将新数据中的分区与 BLT 中的分区进行比较。如果新分区被添加，则 BBUM 3211 从新数据得到分区的开始和结束，检查 DST 3215 中任何重叠的 LBA 范围并移除它们，将所述分区的开始添加的 BLT，并且将该项进行标记以便进行立即动作。

[0543] 图 36 是根据本发明示例性实施例的用于对（重新）格式化分区的 LBA 0 更新进行同步处理的高级别伪代码。特别地，如果需要进行立即动作并且块是分区引导扇区，则 BBUM 3211 从新数据得到 MFT 的开始并且计算位图记录的位置。如果在用于该分区的 BLT

中已经有相同的位图记录项，则不需要进行任何动作。然而，如果位图记录位于与 BLT 版本不同的位置，则 BBUM 3211 更新所述 BLT 并且从盘片读取新的位置。如果该位置看上去不像位图记录（即，其不具有 $ 位图串），则不需要进行任何动作。然而，如果所述位置看上去像位图记录，则 BBUM 3211 得到（多个）新的位图位置并且将它们与 BLT 进行比较。如果（多个）新的位图位置相同，则不需要进行任何动作。如果新的位图处于不同位置，则 BBUM 3211 设置所有 BBUB 位，更新 BBUB 映射并且在 BLT 中移动 LBA 范围。如果新的位图小于现有位图，则 BBUM 3211 收缩 BBUB，将未映射的 LBA 范围添加到 DST 中并且收缩 BLT 中的 LBA 范围。如果新的位图大于现有位图，则 BBUM 3211 设置所有的附加 BBUB 位，放大 BBUB 并放大 BLT 中的 LBA 范围。

[0544] 图 37 是根据本发明示例性实施例的用于对删除分区的 LBA 0 更新进行同步处理的高级别伪代码。特别地，如果需要进行立即动作并且块是分区表，则 BBUM 3211 将新数据中的分区与 BLT 中的分区进行比较。如果分区被删除，则 BBUM 3211 删除 BBUB，从 BLT 删除引导扇区，从 BLT 删除位图记录，从 BLT 删除位图范围并且将分区范围添加到 DST。

[0545] 图 38 是根据本发明示例性实施例的用于异步任务 3214 的高级别伪代码。异步任务 3214 对 BBUB 进行解析，并接着对于 BBUB 中设置的每一位，异步任务 3214 检查对应的聚簇是否被标记为未被主机文件系统所使用。如果所述聚簇被标记为未被主机文件系统使用，则异步任务 3214 检查所述聚簇是否被标记为被存储控制器使用。如果所述聚簇被标记为被存储控制器所使用，则异步任务 3214 将 LBA 范围添加到 DST。异步任务 3214 还回收用于 DST 中的每个 LBA 范围的存储空间。

[0546] 在接收到引导扇区更新之后，等待位图记录的写入通常是不够的（通常不知道 NTFS 格式化出现的顺序，并且其无论怎样都可在较小补丁中变化），原因在于位图记录可能已经被写入盘片。如果位图记录在扩展的 BPB 之前写入，则 BBUM 3211 将不获取它，原因在于该位置没有出现在 BLT 3212 中；对此的例外是当位图记录的位置还没有变化时。尽管存在该例外，但是 BBUM 3211 通常必须在这一点立即从盘片读取位图记录位置，以察看是否存在位图记录，并且其通常需要能够将随机噪声与初始化的位图记录区分开来（可能对 $ 位图统一编码串进行检查）。如果还没有写入，其可以等待写入。如果已经在盘片上，则其通常必须立即被解析。解析通常需要为了（多个）位图位置对记录进行解码，并且那些位置被添加到 BLT 3212 并且被标记为不需要立即解析。如果位图大小已经变化，则解析通常还需要基于位图新的大小和（多个）位置例示新的 BBUB 3213；否则利用新的位置对现有 BBUB3213 进行更新一般就足够了。由于其通常不知道是在写入位图记录之前还是之后写完了新的位图，所以似乎还适于设置所有位（可能在之后，但是如果发生，则所述位将仅在位图写入到来时的数秒内被无害地设置第二次）。危险情况是如果写入在之前发生，在这种情况下由于处于不同的位置，所以写入会被错过；设置所有的位以确保所述位图得到解析。

[0547] 在引导扇区更新的情况下（例如，由于分区的重新格式化），位图可能为同样的大小并占据相同的空间，从而 BLT 3212 或 BBUB 3213 无需发生变化。可假定新的位图被重写，其中大多数块为全零，从而异步任务 3214 应当能够继续处理它们以便从 CAT 回收未分配的聚簇。可对引导扇区的卷序列号进行检查来确定所述更新是否是重新格式化的结果。

[0548] 出于独立于引导扇区的原因，位图记录还可以在任意时刻被更新。回收器 3210 可

能必须能够不断地对位图移动或大小变化进行处理；不清楚没有创建不同大小的分区的情况下位图大小是否能够不断变化，但是出于任何原因 NTFS 未来的版本都可对此进行支持。在这种情况下，位图的（多个）新位置通常必须被编程到 BLT 3212 中，其中移除旧项并添加新项。BBUB 3213 必须相应放大或收缩。通过收缩所释放的任何 LBA 范围可被添加到 DST 3215，虽然严格来说它们仍然映射到该分区。

[0549] 另一个问题在于，如果位图记录的最后更新字段的时间被频繁修改以反映正在进行的位图修改，则结果会是大量的内嵌解析。

[0550] 对于位图本身的所有后续写入都被经由 BBUB 3212 推到异步任务 3214。

[0551] 这里的基本策略是所有的被分配聚簇将表示在 BBUB 3213 或 DST3215 中，并且采用任何一种方式都将回收未分配的那些。替代方案将具有用于每个卷的卷标识符，所述卷标识符对于 BBUM 3211 和 CAT 都是已知的，以使得每个写入都必须被 BBUM 3211 映射到卷并且在到达 CAT 管理器之前以该标识符进行标记，其中所述卷标识符将存储在 CAT 记录中。新的卷通常从其所覆盖的旧卷那里得到不同的标识符，从而异步任务 3214 能够回收具有旧的卷标识符的记录，而没有回收已经被来自新卷的数据所覆盖的聚簇的危险。显然，这会消耗 CAT 记录中的空间。其还依赖于重新格式化中的写入顺序。由于系统在引导分区中看到新的卷序列号之前通常不了解新卷，所以在此之前对于卷的任何其它写入将被旧的卷标识符所标记。

[0552] 主要的工作将由专门的回收器任务 3214 来完成，正常情况下其一分钟醒来一次，从 BBUB 3213 收集某个工作，并且通过将位图块通过缓存进行页进入 (paging-in) 并将所述位与 CAT 进行比较来执行它。在示例性实施例中，BBUB 3213 将被逻辑地分段 (1k 的段大小)，具有显示对于该段的更新数目的用于每个段的计数器，以及反映任意计数器所保存的最高值的全局计数器；这些计数器将由工作产生器 (BBUM 3211) 进行增加并且由工作消耗器（回收器任务 3214) 减少。醒来的回收器任务 3214 将检查所述全局计数器并且决定其中的值是否足够高以证明位图中的页进入。如果是这样，则任务 3214 将确定值对应于哪个段（例如，通过在计数器阵列中进行反复）并接着开始通过适当 BBUB 段的位进行反复。当其找到所设置的位时，其将对所述位图的该块进行页进入并且将其与 CAT 进行比较。

[0553] 如上所述，回收器任务 3214 的操作可进行动态调节，例如，通过改变其运行的频率或其决定进行一些工作的阈值。然而，在本发明的示例性实施例中，通常不进行这样的动态调节，原因在于体系结构在某种程度上耦合到回收器所运行的频率。特别地，由于所提出的体系结构已经以主机文件系统 2711 将在最大时间窗口（例如，一分钟）内对其元数据进行更新的假设所设计，所以所提出的体系结构实际上无法比该最大时间窗口更为频繁地运行回收器任务 3214。所提出的体系结构可以进行较不频繁的运行，这实际上没有破坏规则，但是其通过看上去比实际更早地进行一些聚簇更新而可能使得聚簇回收效率低（例如，如果以每分钟发生运行的假设来设计时限计算，但是它们实际上每三分钟发生，则时限计算可因为因数 3 而空闲）。此外，任务优先级在编译时通常是固定的，并且由此通常在系统操作期间不发生变化。

[0554] 应当注意的是，在本发明的示例性实施例中，所述存储系统实现了大小为 4K 的聚簇。因此，如果文件系统以不同于 4K 的聚簇大小进行格式化，则所述文件系统位图中的位将不会与存储系统中的聚簇完美地关联。例如，如果文件系统聚簇的大小小于 4K，则位图中

的多个位通常必须被清除而干扰了与 CAT 的交叉引用。然而，如果文件系统的聚簇大小大于 4K，则位图的一个清除位通常将需要在 CAT 中进行多次查找，每 4K 一次。

[0555] 另一个问题是如何对回收器遇到聚簇过于年轻而无法回收的情况进行处理。在这样的情况下，所述回收器会可能留下设置在 BBUB 中的位，由此需要一个或多个后续扫描以再次解析整个 512 位（例如，通过 512 位的下一次扫描仅找到仍然过于年轻而无法回收的聚簇）。作为选择，所述回收器能够清除该位并且将所述聚簇添加到年轻块 / 需要再次检查的位偏移的列表。从实施的观点来看，如果可保持列表相当小，则只有后者的方法是切合实际的。

[0556] 从实施的观点来看，所述回收器和 BBUB 都将通过 CAT 管理器从盘片被读取。将通过 CAT 管理器所提供的特殊 API 来执行聚簇的回收。

[0557] 虚拟热后备 (virtual hot spare)

[0558] 如上所述，在许多存储系统中，热后备存储设备维持就绪状态，使得在其他存储设备出现故障的情况下，热后备可以快速被联机。在本发明特定实施例中，不是维持物理上分离的热后备，而是跨多个存储设备由未使用的存储容量来创建虚拟热后备。不同于物理热后备，如果以及当对于根据（多个）其余存储设备恢复的数据的存储发生存储设备故障时，这种未使用存储容量是可用的。

[0559] 该虚拟热后备特征需要阵列上的足够可用空间，以保证在盘片故障情况下，数据可以按冗余方式再布局。因而，根据运行，存储系统典型地确定实现虚拟热后备可能需要的未使用存储容量的量（例如，根据存储设备的数量、各存储设备的容量、数据存储的数量、以及存储数据的方式），并且，如果需要附加存储容量用于虚拟热后备的话则产生信号（例如，使用绿色 / 黄色 / 红色光指示状态和插槽，如上所述）。随着存储区的分配，按每个盘片保持记录，多少个区域需要再布局该存储区。下表示例使用四个驱动器的虚拟热后备：

[0560]

| 存储区 | 类型 | 存储在盘片上 | 注释 | 如果盘片故障需要的区域 | | | |
|---|---|---|---|---|---|---|---|
| | | | | 盘 0 | 盘 1 | 盘 2 | 盘 3 |
| 2 | 双驱动器镜像 | 0,1 | 如果 0 或 1 故障就在盘片 2 或 3 上重构 | 12 | 12 | 0 | 0 |
| 3 | 双驱动器镜像 | 0,3 | 如果 1 或 2 故障就在盘片 1 或 2 上重构 | 12 | 0 | 0 | 12 |
| 5 | 三驱动器条带 | 1,2,3 | 如果 1 或 2 或 3 故障就在盘片 0 上重构 | 0 | 6 | 6 | 6 |
| 10 | 四驱动器条带 | 0,1,2,3 | 转换为跨其他三个盘的三驱动器条带 | 2,2,2 | 2,2,2 | 2,2,2 | 2,2,2 |

[0561] 下表示例具有使用三个驱动器的虚拟热后备：

[0562]

| 存储区 | 类型 | 存储在盘片上 | 注释 | 如果盘故障需要的区域 | | |
|---|---|---|---|---|---|---|
| | | | | 盘 0 | 盘 1 | 盘 2 |
| 2 | 双驱动器镜像 | 0，1 | 在盘片 3 上重构 | 12 | 12 | 0 |
| 3 | 双驱动器镜像 | 0，3 | 在盘片 1 上重构 | 12 | 0 | 12 |
| 5 | 三驱动器条带 | 1，2，3 | 转换为双驱动器镜像 | 6,6 | 6,6 | 6,6 |

[0563] 在这个示例性实施例中，虚拟热后备并非在仅有 1 个或者 2 个驱动器的阵列上可用。根据每个存储区的信息和阵列中盘片的数量，该阵列确定每个可能盘片故障的再布局情况并保证对每种情况每个驱动器上都有足够的可用空间。产生的信息可以反馈回再布局引擎和存储区管理器，使得数据可以在数据存储和热后备特征之间正确平衡。注意，根据这些由存储区布局数据的计算，热后备特征需要足够备用工作空间区域，使得可发生再布局。

[0564] 图 22 是示出根据本发明示例性实施例的管理虚拟热后备的示例性逻辑的逻辑流程图。框 2102 中，该逻辑确定每个可能盘片故障的再布局情形。在框 2104 中，该逻辑确定最坏情况下冗余再布局数据的每个驱动器需要的空间量。框 2106 中，该逻辑确定在最坏情况下数据冗余再布局需要的后备工作空间区域的数量。框 2108 中，该逻辑确定每个驱动器上需要的空间总量，以允许在最坏情况下冗余再布局数据（实际上是再布局所需要的空间量和备用工作空间区域量的和）。框 2110 中，该逻辑确定存储系统是否包含足够的可用的存储量。如果有足够数量的可用存储（框 2112 中的是），则该逻辑迭代在框 2199 终止。但是，如果没有足够量的可用存储（框 2112 中的否），则该逻辑在框 2114 中确定哪个驱动器 / 插槽需要更新。接着在框 2116，该逻辑发出信号，指出需要附加存储空间并指示哪个驱动器 / 插槽需要更新。该逻辑迭代在框 2199 终止。

[0565] 图 23 是示出根据本发明示例性实施例的用于确定每个可能盘片故障的布局情形的示例性逻辑的逻辑流程图，如图 22 的框 2102 中。框 2202 中，该逻辑分配存储区。然后，在框 2204 中，该逻辑确定按每个盘需要多少个存储区用于再布局存储区。该逻辑迭代在框 2299 终止。

[0566] 图 24 是示出根据本发明示例性实施例的包括虚拟热后备功能的示例性逻辑的逻辑流程图。框 2302 中，该逻辑维持在最坏情况下的充足数量的可用存储以使得可冗余地再布局数据。当在框 2304 确定了驱动器丧失（例如移除或者故障），该逻辑在框 2306 自动重构一个或者多个其余驱动器，以恢复数据的容错。该逻辑迭代在框 2399 终止。

[0567] 图 25 是示出根据本发明示例性实施例的自动重构一个或者多个其余驱动器用于恢复数据容错的示例逻辑的逻辑流程图，如图 24 的框 2306。框 2402 中，该逻辑将跨四个或者更多存储设备的第一条带化模式转换为跨三个或者更多剩余存储设备的第二条带化模式。框 2404 中，该逻辑可以将跨三个存储设备的条带化模式转换为跨两个剩余存储设备的镜像模式。当然，该逻辑可以按其他方式转换模式以使得随着驱动器的丧失而冗余地再布局数据。该逻辑迭代在框 2499 终止。

[0568] 再参考图 21，存储管理器 2504 典型地包括适当部件和逻辑用于实施如上所述的虚拟热后备功能。

[0569] 动态升级

[0570] 上述用于处理存储的动态扩展和收缩的逻辑可以被扩充用于提供可动态升级的存储设备，其中，存储设备可根据需要用更大的存储设备来替换，并且现有数据跨各存储设备自动重新配置，使得冗余得到维持或者增强，并且该更大的存储设备所提供的附加存储空间将包括在跨多个存储设备的可用存储空间的池 (pool) 中。因而，当用较大存储设备替换较小存储设备时，附加存储空间可用于改进已存储数据和存储附加数据的冗余。无论何时需要更多存储空间，向用户提供适当信号（例如使用如上所述的绿色 / 黄色 / 红色灯光），而用户可以简单移除存储设备并用更大存储设备替换它。

[0571] 图 26 是示出根据本发明示例性实施例的用于升级存储设备的逻辑流程图。框 2602 中，该逻辑在第一存储设备上按照存储在其中的数据在其他存储设备上冗余出现的方式来存储数据。框 2604 中，该逻辑检测用比第一存储设备具有更大存储容量的替换设备来替换第一存储设备的替换。框 2606 中，该逻辑使用冗余存储在其他设备上的数据将存储在第一设备的数据自动再生到该替换设备上。框 2608 中，该逻辑使替换设备上的附加存储空间可用于冗余地存储新数据。框 2610 中，该逻辑可以在替换设备上的附加存储空间中冗余地存储新数据，如果没有其他设备具有充足的可用存储容量用于提供新数据的冗余的话。框 2612 中，如果至少一个其他设备具有充足的可用存储空间用于提供新数据冗余，则该逻辑可以跨多个存储设备冗余地存储新数据。

[0572] 再参考图 21，存储管理器 2504 典型包括适当部件和逻辑用于实施如上所述的动态升级功能。

[0573] 其他

[0574] 本发明的实施例可以用于向主机计算机提供存储容量，例如按美国临时申请第 60/625,495 所述的方式使用外围连接协议，所述申请以 Geoffrey S.Barrall 的名义于 2004 年 11 月 5 日提交，通过引用全文结合于此以供参考。

[0575] 应当注意，散列算法可能不产生严格唯一的散列值。因而，可以想到散列算法对具有不相同内容的两个数据块产生相同散列值。散列函数（其通常结合散列算法）典型包括确认唯一性的机制。例如，在上述本发明的示例性实施例中，如果一个块的散列值与另外块的散列值不同，则认为这些块的内容不相同。但是如果一个块的散列值与另一个块的散列值相同，那么散列函数可以比较这两个块的内容或者利用一些其他机制（例如不同散列函数）以确定内容是否相同。

[0576] 应当注意，这里使用的该逻辑流程图用于示例本发明的各个方面，而不应解释为将本发明限制在任何特殊逻辑流程或者逻辑工具。在不改变总体结果的前提下，或者不以另外的方式背离本发明范围的情况下，所述逻辑可划分为不同的逻辑块（例如程序、模块、函数、或者子程序）。通常，可以增加、改变、省略逻辑部件，可以按不同顺序执行，或者使用不同逻辑结构来实施（例如逻辑门、循环元语、条件逻辑、和其他逻辑机构），而不改变整体结果或者以其他方式脱离本发明的真实范围。

[0577] 本发明可以按多种不同形式来实施，包括但不限于：使用处理器（例如微处理器、微控制器、数字信号处理器、或者通用计算机）的计算机程序逻辑、使用可编程逻辑设备（例如现场可编程门阵列 (FPGA) 或者其他 PLD) 的可编程逻辑、离散组件、集成电路（例如特定用途集成电路 (ASIC))、或者包括其任意组合的其他装置。

[0578] 实施全部或者部分这里前述功能的计算机编程逻辑可以按多种形式实施，包括但

不限于：源代码形式、计算机可执行形式、各种中间形式（例如由汇编程序、编译程序、连接程序、或者定位程序等产生的形式）。源代码可以包括按任何各种编程语言实现的计算机程序指令序列（例如对象代码、汇编语言、或者例如 Fortran、C、C++、JAVA、或 HTML 等高级语言），在各种操作系统或者操作环境下使用。源代码可以定义并使用各种数据结构和通信消息。源代码可以是计算机可执行形式（例如经由解释程序）、或者源代码可以转换（例如经由转换器、汇编器、或者编译器）为计算机可执行形式。

[0579] 所述计算机程序可以持久地或者暂时地按任何形式（例如源代码形式、计算机可执行形式、或者中间形式）固定在有形存储介质中，例如半导体存储装置（例如 RAM、ROM、PROM、EEPROM、或者可编程闪存 RAM）、磁存储设备（例如软盘或者硬盘）、光存储设备（例如 CD-ROM）、PC 卡（例如 PCMCIA 卡）、或者其他存储器设备。该计算机程序可以按任何形式固定在信号中，该信号传输给使用各种通信技术的计算机，包括但不限于：模拟技术、数字技术、光技术、无线技术（例如蓝牙）、网络技术、以及互联网技术。该计算机程序可以按任何形式来发布，如使用附带有打印的文档或者电子文档的可移动存储介质（例如压缩包软件）、预装入计算机系统（例如在系统 ROM 或者硬盘上）、或者通过通信系统（例如因特网或者万维网）从服务器或者电子公告板来发布。

[0580] 实施全部或者部分这里的上述功能的硬件逻辑（包括用于可编程逻辑设备的可编程逻辑）可以使用传统人工方法设计，或者可以使用各种工具来设计、捕获、模拟、或者建立电子文档，例如计算机辅助设计 (CAD)、硬件描述语言（例如 VHEL 或者 AHDL）、或者 PLD 编程语言（例如 PALASM、ABEL、或者 CUPL）。

[0581] 所述可编程逻辑可以永久或者暂时固定在有形存储介质中，例如半导体存储设备（例如 RAM、ROM、PROM、EEPROM、或者可编程闪存 RAM）、磁存储设备（例如软盘或者硬盘）、光存储设备（例如 CD-ROM）、或者其他存储器设备。该可编程逻辑可以按任何形式固定在信号中，该信号传输给使用各种通信技术的计算机，包括但不限于：模拟技术、数字技术、光技术、无线技术（例如蓝牙）、网络技术、以及互联网技术）。该可编程逻辑可以按任何形式来发布，如使用带有打印文档或者电子文档的可移动存储介质（例如压缩包软件）、预装入计算机系统（例如在系统 ROM 或者硬盘上）、或者通过通信系统（例如因特网或者万维网）从服务器或者电子公告板发布。

[0582] 本发明涉及下列美国专利申请，通过引用全文并入此处：

[0583] 代理卷号 2950104、名称为 Dynamically Upgradeable Fault-TolerantStorage System Permitting Variously Sized Storage Devices and Method；

[0584] 代理卷号 2950105、名称为 Dynamically Expandable andContractible Fault-Tolerant Storage System With Virtual Hot Spare；和

[0585] 代理卷号 2950107、名称为 Storage System Condition Indicator andMethod；

[0586] 在不脱离本发明真实范围下，本发明可以实施为其他特殊形式。所述实施例在所有方面都应当认为作为说明而不是限制。

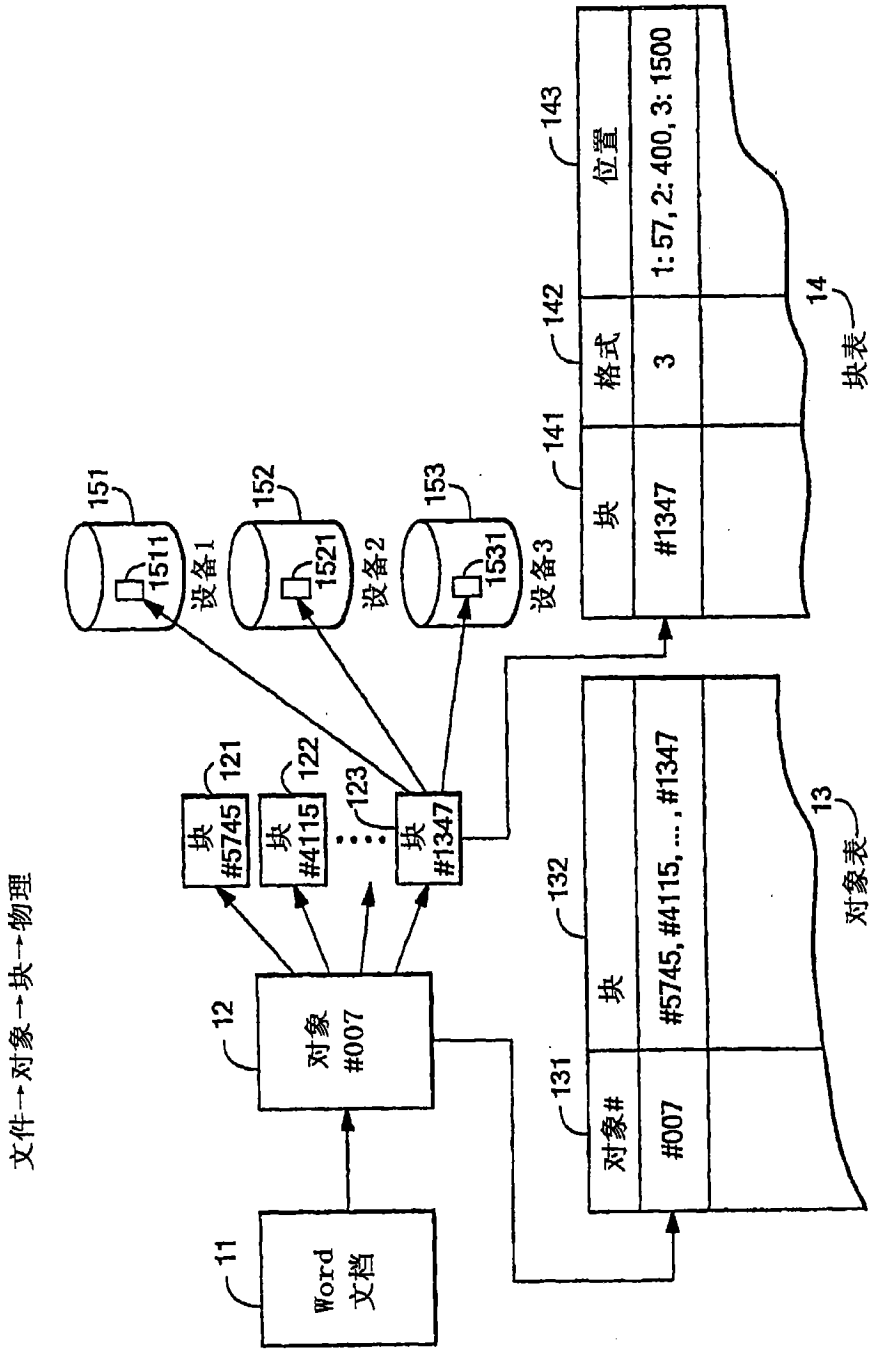

文件→对象→块→物理
Word 文档
11
对象 #007
12
块 #5745
121
块 #4115
122
123
块 #1347
151
1511
设备1
152
1521
设备2
153
1531
设备3
对象表
13
对象#
#007
块
#5745, #4115, ..., #1347
131
132
块表
14
块
#1347
格式
3
位置
1: 57, 2: 400, 3: 1500
141
142
143

图1

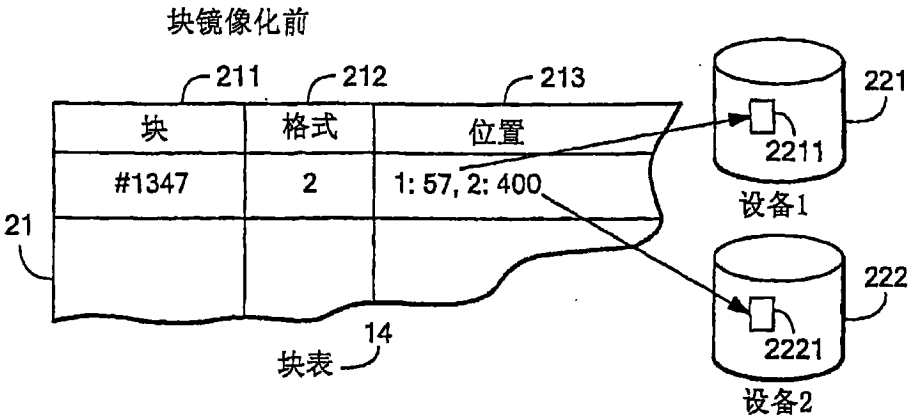

块镜像化前
211
212
213
块
格式
位置
#1347
2
1: 57, 2: 400
21
14
块表
221
2211
设备1
222
2221
设备2

图2A

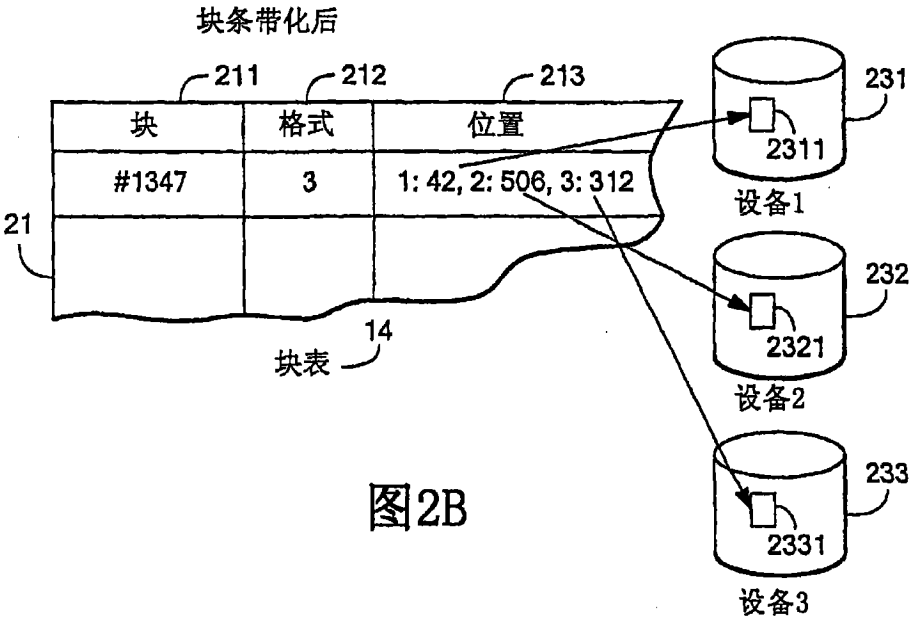

块条带化后
211
212
213
块
格式
位置
#1347
3
1: 42, 2: 506, 3: 312
21
14
块表
231
2311
设备1
232
2321
设备2
233
2331
设备3

图2B

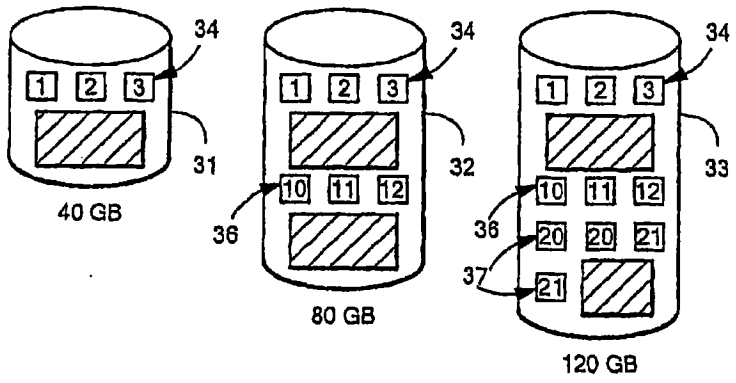



图 3 块 #1、#2、#3 使用模式 3 存储（使用 3 个设备的条带化）
块 #10、#11、#12 使用模式 2 存储（使用 2 个设备的镜像化）
块 #20、#21、使用模式 1 存储（使用 1 个设备的镜像化）

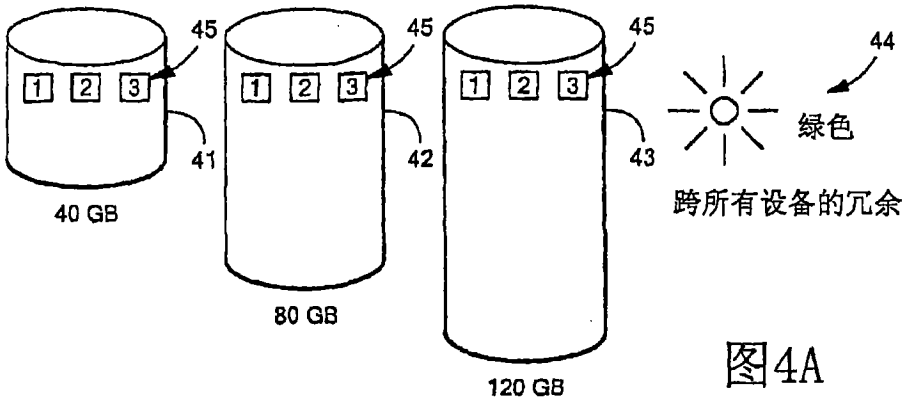

45
45
45
44
1 2 3
1 2 3
1 2 3
绿色
41
42
43
跨所有设备的冗余
40 GB
80 GB
120 GB

图4A

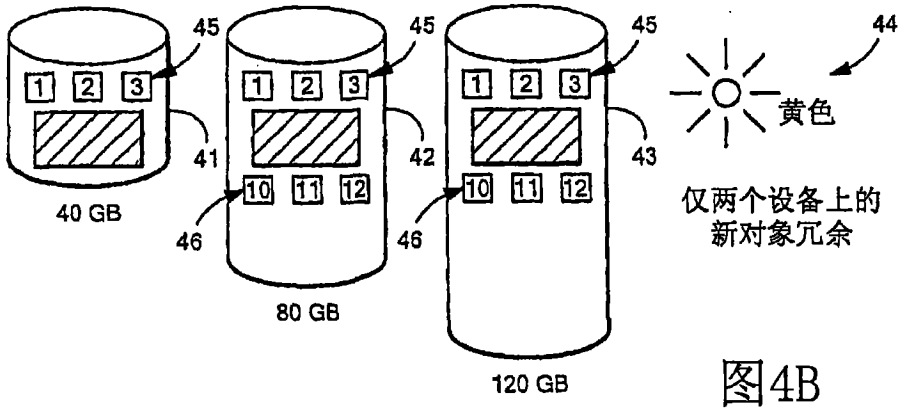

45
45
45
44
1 2 3
1 2 3
1 2 3
黄色
41
42
43
10 11 12
10 11 12
仅两个设备上的
新对象冗余
46
46
40 GB
80 GB
120 GB

图4B

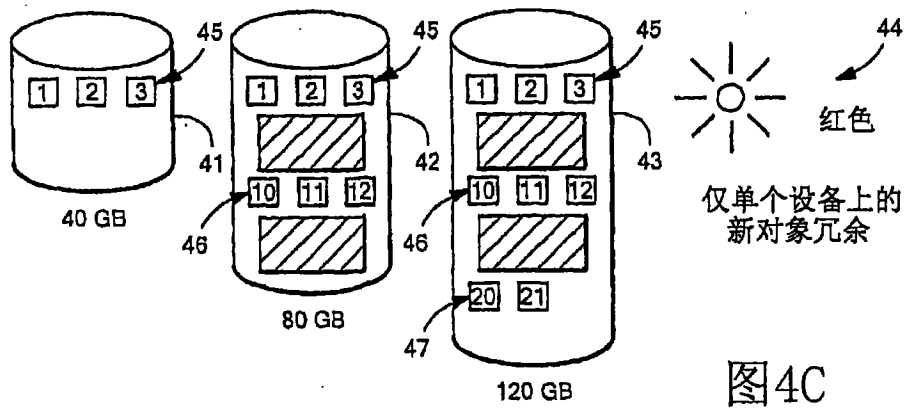

45
45
45
44
1 2 3
1 2 3
1 2 3
红色
41
42
43
10 11 12
10 11 12
仅单个设备上的
新对象冗余
46
46
20 21
47
40 GB
80 GB
120 GB

图4C

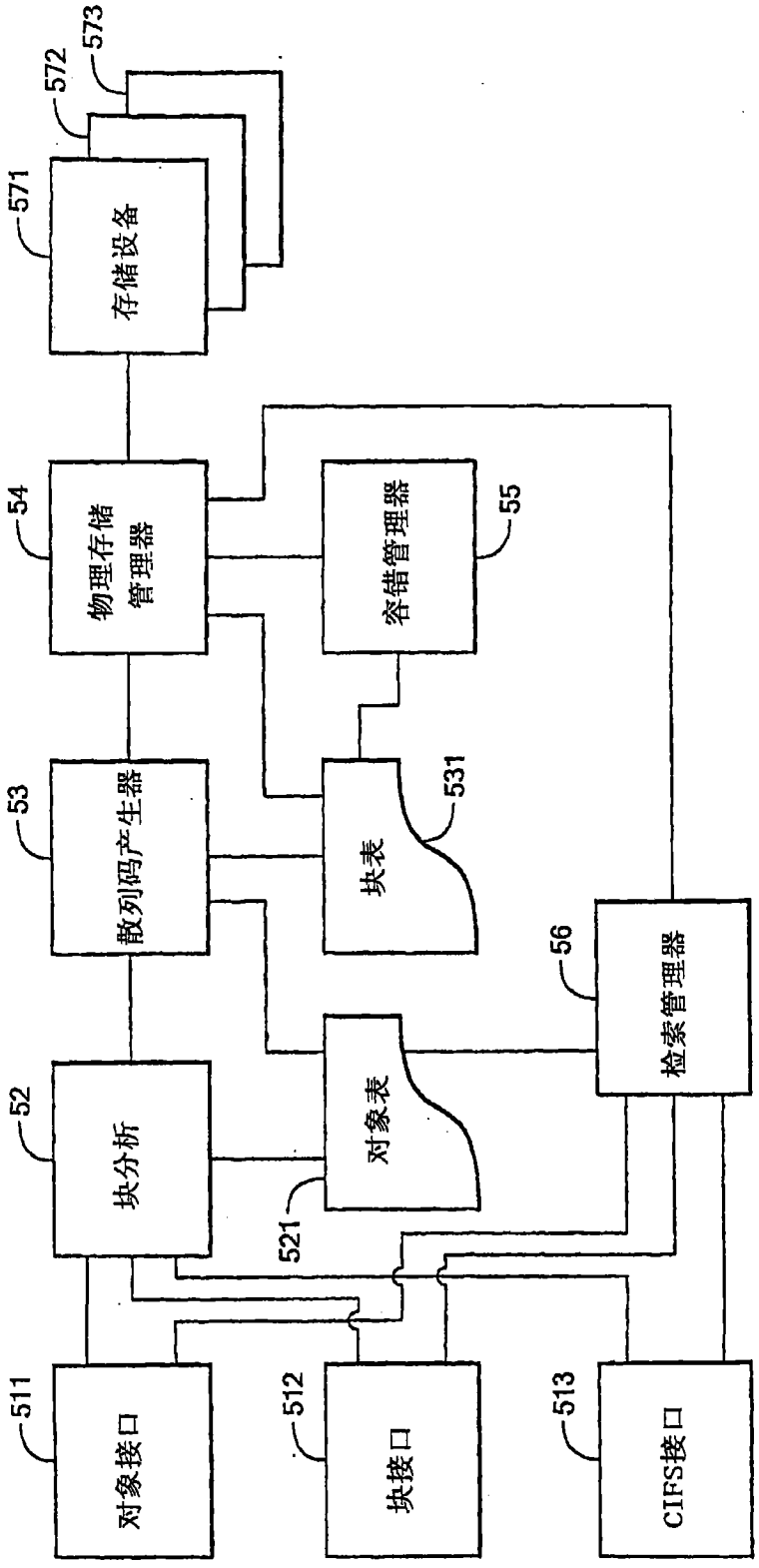

573
572
571
存储设备
54
物理存储管理器
55
容错管理器
53
散列码产生器
531
块表
56
检索管理器
52
块分析
对象表
521
511
对象接口
512
块接口
513
CIFS接口

图5

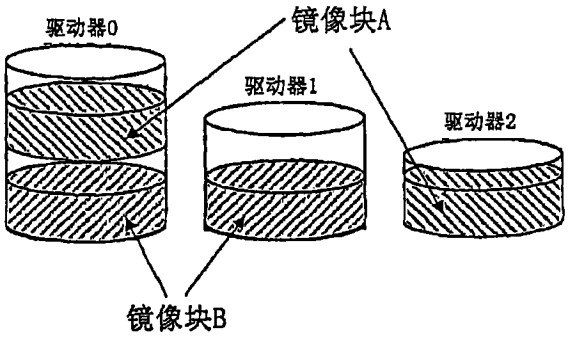

驱动器0
镜像块A
驱动器1
驱动器2
镜像块B

图 6

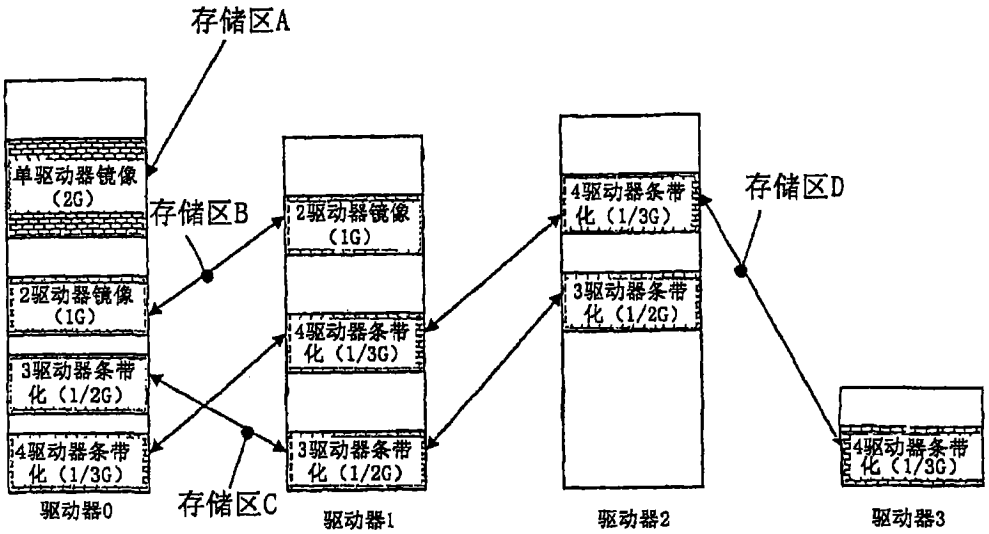

存储区A
单驱动器镜像（2G）
2驱动器镜像（1G）
3驱动器条带化（1/2G）
4驱动器条带化（1/3G）
驱动器0
存储区B
存储区C
2驱动器镜像（1G）
4驱动器条带化（1/3G）
3驱动器条带化（1/2G）
驱动器1
4驱动器条带化（1/3G）
3驱动器条带化（1/2G）
驱动器2
存储区D
4驱动器条带化（1/3G）
驱动器3

图 7

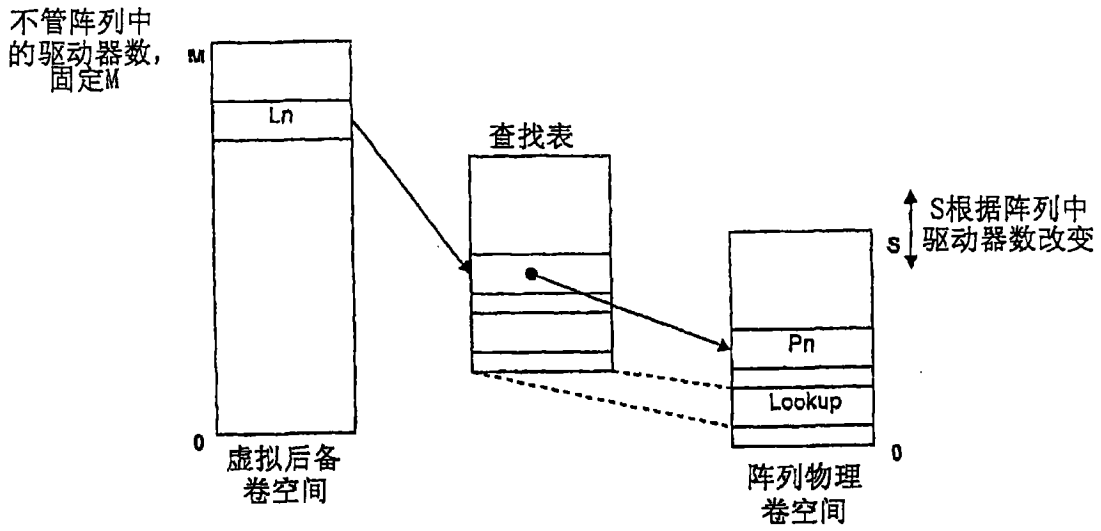



图 8

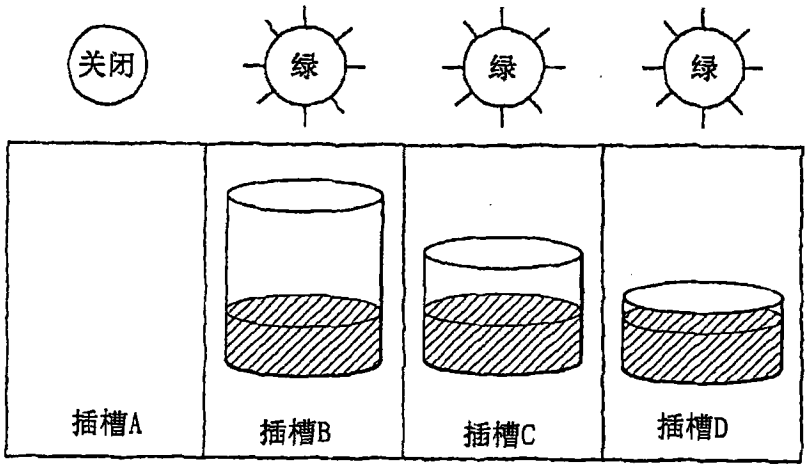



图 9 a) 阵列空间可用，全部驱动器 OK

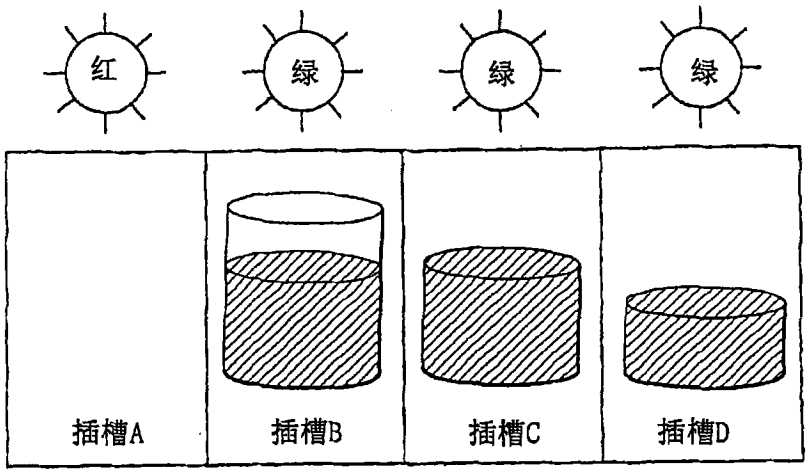



图 10 b) 插槽 C& 插槽 D 满 - 数据不冗余 :添加驱动器至空插槽 A

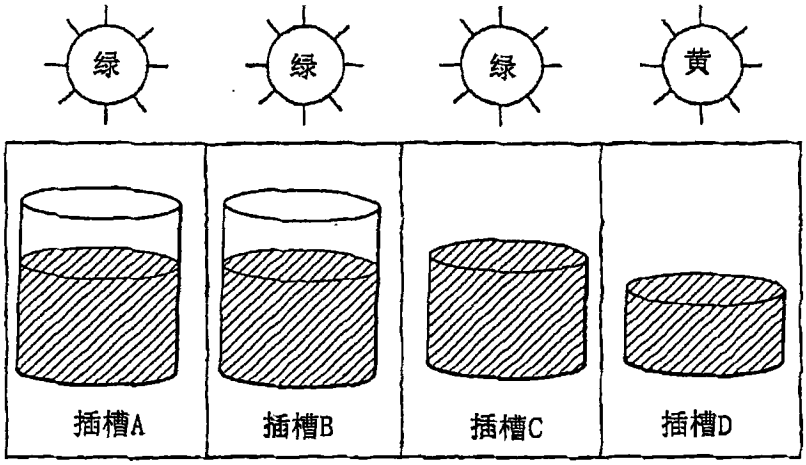



图 11 c) 在电源故障的情况下阵列不能保持冗余 :替换插槽 D 中的驱动器

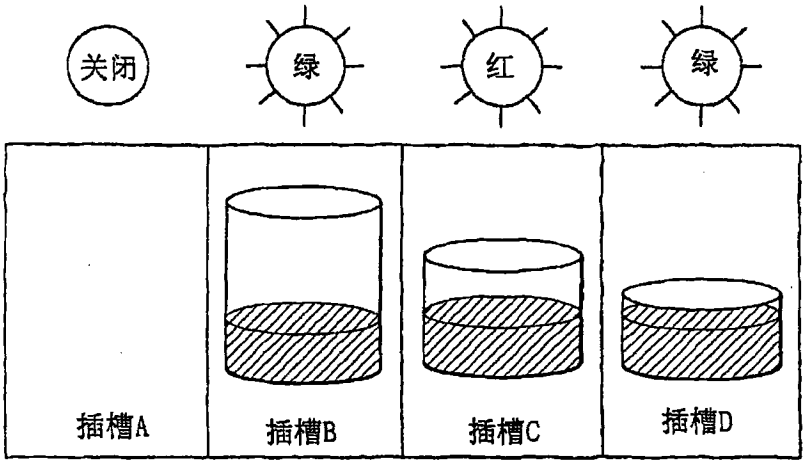



图 12 d) 插槽 C 中的驱动器已经失效

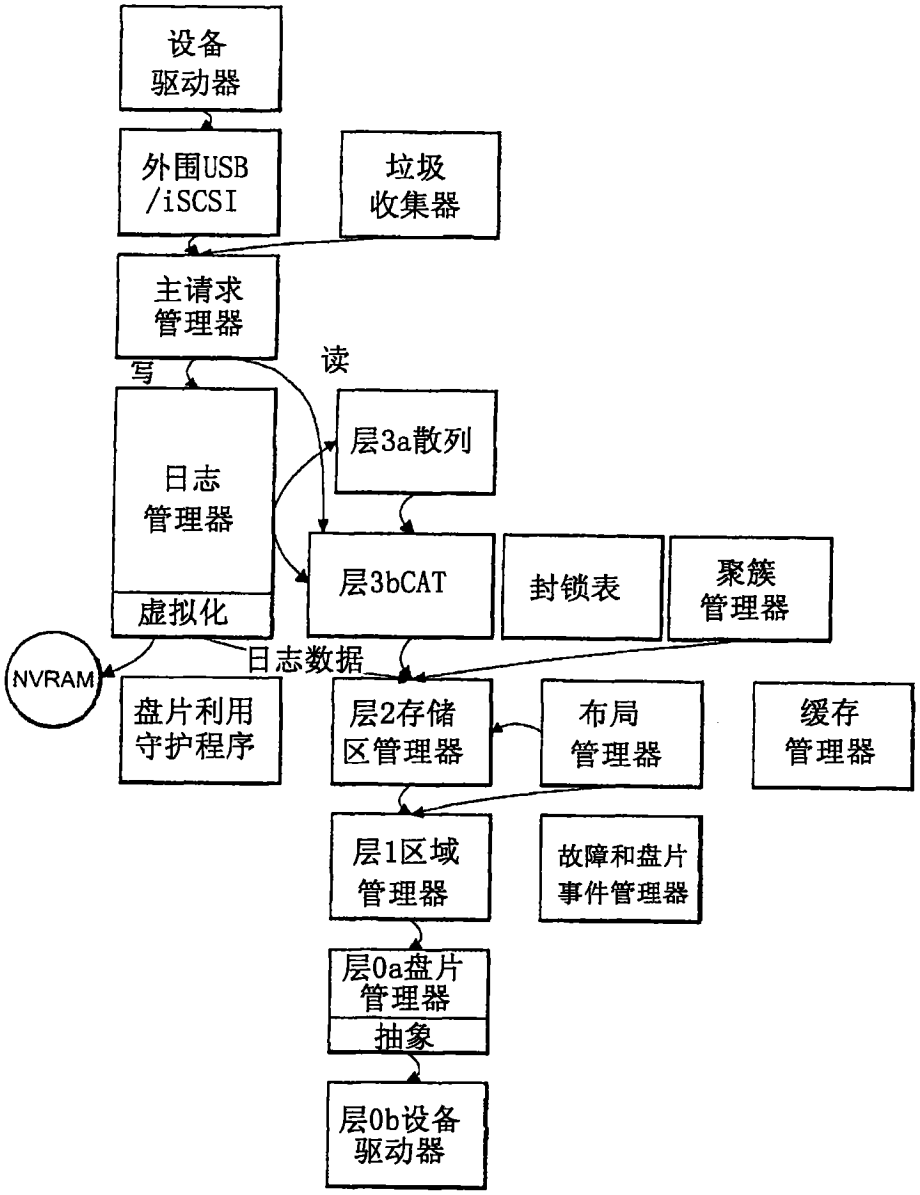

设备
驱动器
外围USB
/iSCSI
垃圾
收集器
主请求
管理器
写
读
日志
管理器
虚拟化
层3a散列
层3bCAT
封锁表
聚簇
管理器
NVRAM
日志数据
盘片利用
守护程序
层2存储
区管理器
布局
管理器
缓存
管理器
层1区域
管理器
故障和盘片
事件管理器
层0a盘片
管理器
抽象
层0b设备
驱动器

图 13

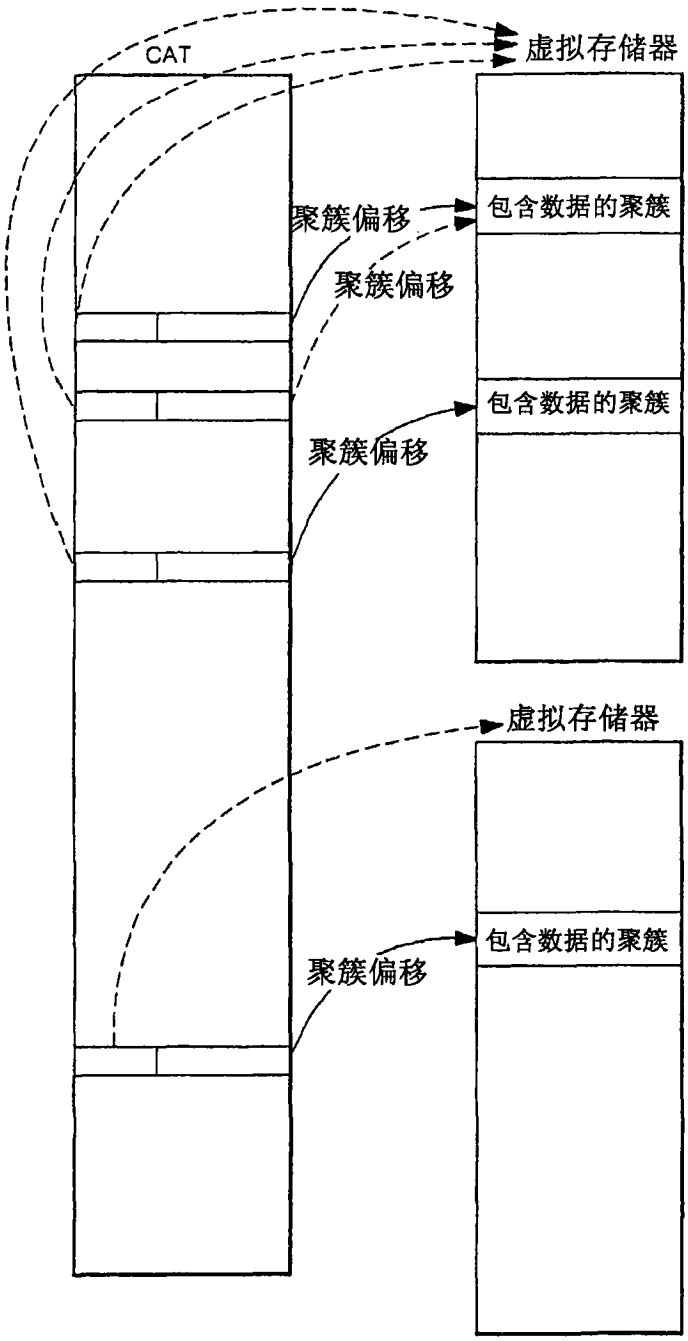

CAT
虚拟存储器
聚簇偏移
聚簇偏移
聚簇偏移
包含数据的聚簇
包含数据的聚簇
虚拟存储器
聚簇偏移
包含数据的聚簇

图 14

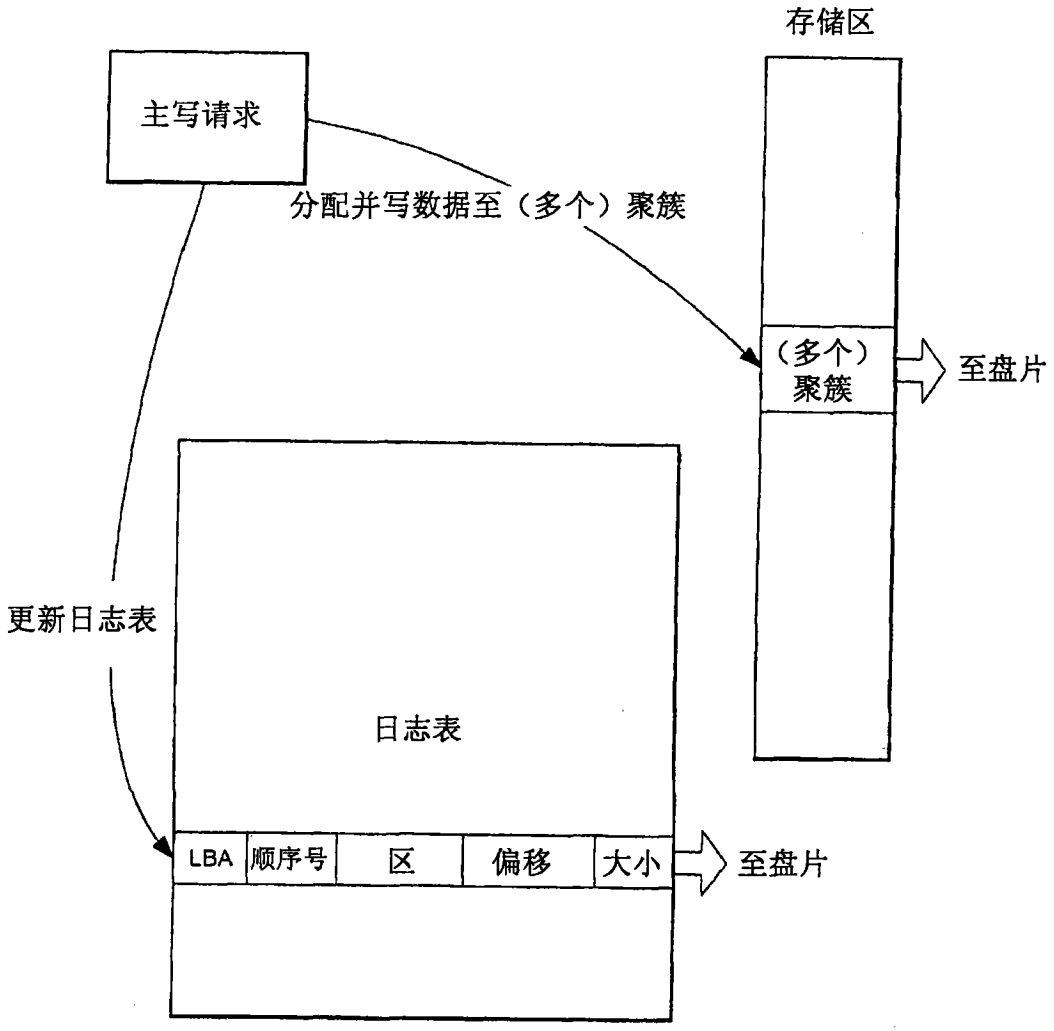



图 15

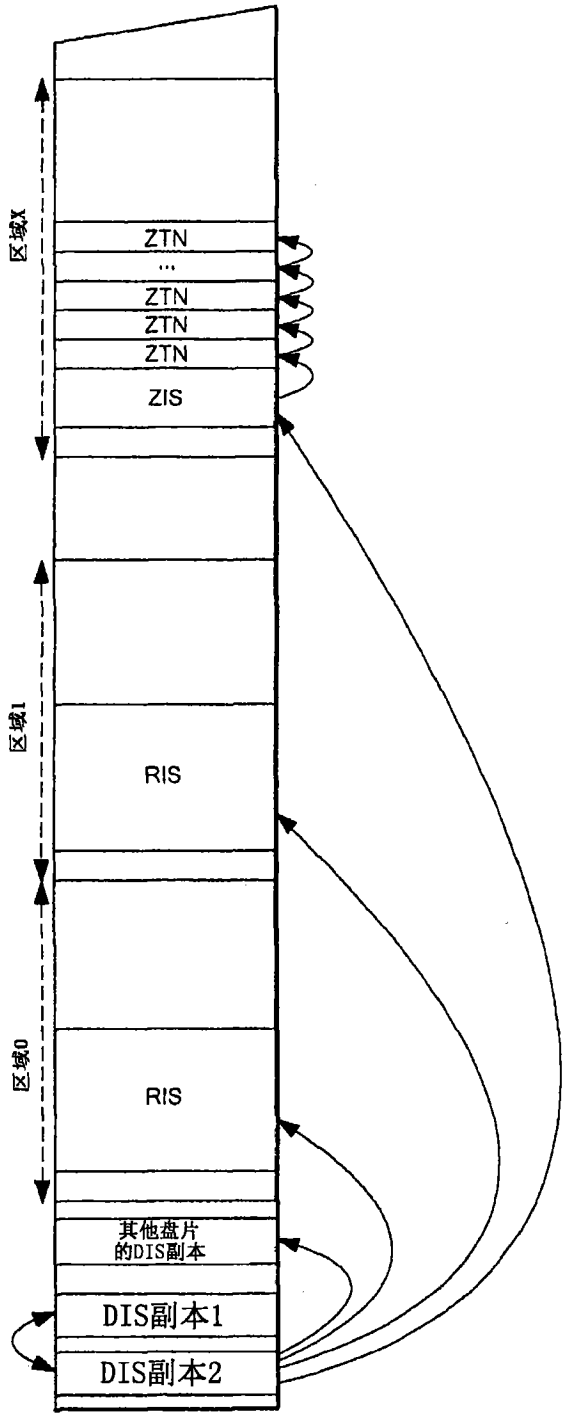

区域X
ZTN
...
ZTN
ZTN
ZTN
ZIS
区域1
RIS
区域0
RIS
其他盘片
的DIS副本
DIS副本1
DIS副本2

图 16

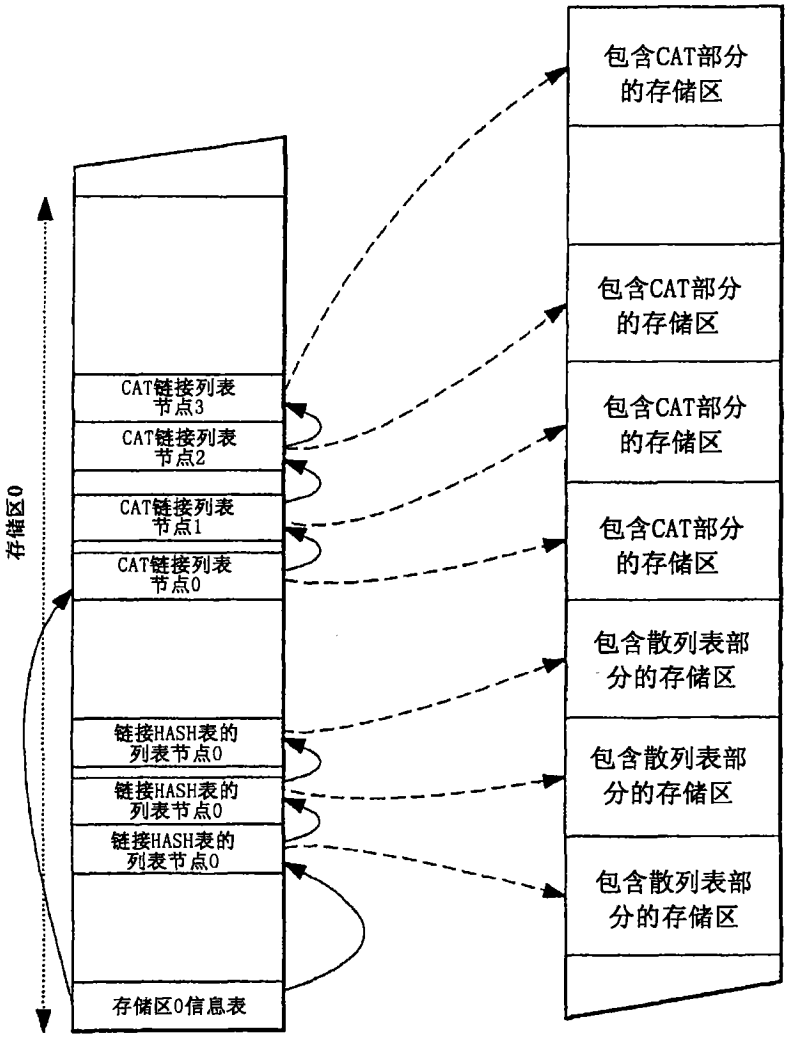

包含CAT部分的存储区
包含CAT部分的存储区
包含CAT部分的存储区
包含CAT部分的存储区
包含散列表部分的存储区
包含散列表部分的存储区
包含散列表部分的存储区
CAT链接列表节点3
CAT链接列表节点2
CAT链接列表节点1
CAT链接列表节点0
链接HASH表的列表节点0
链接HASH表的列表节点0
链接HASH表的列表节点0
存储区0信息表
存储区0

图 17

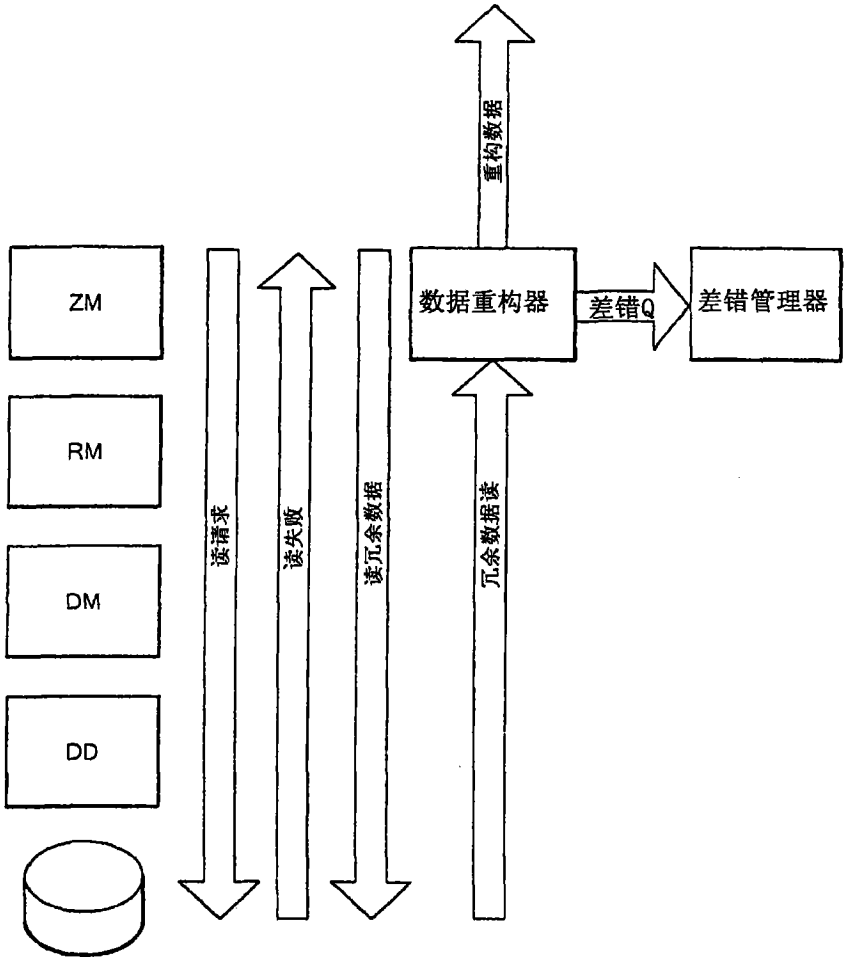



图 18

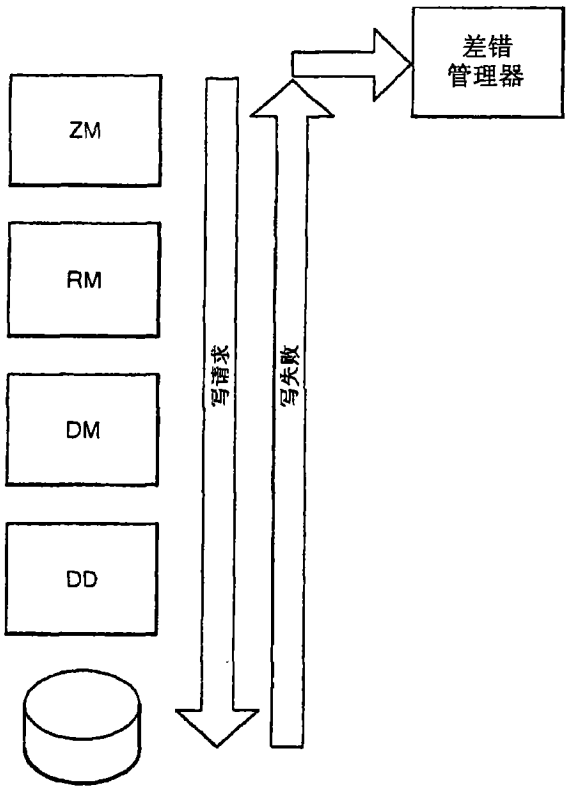



图 19

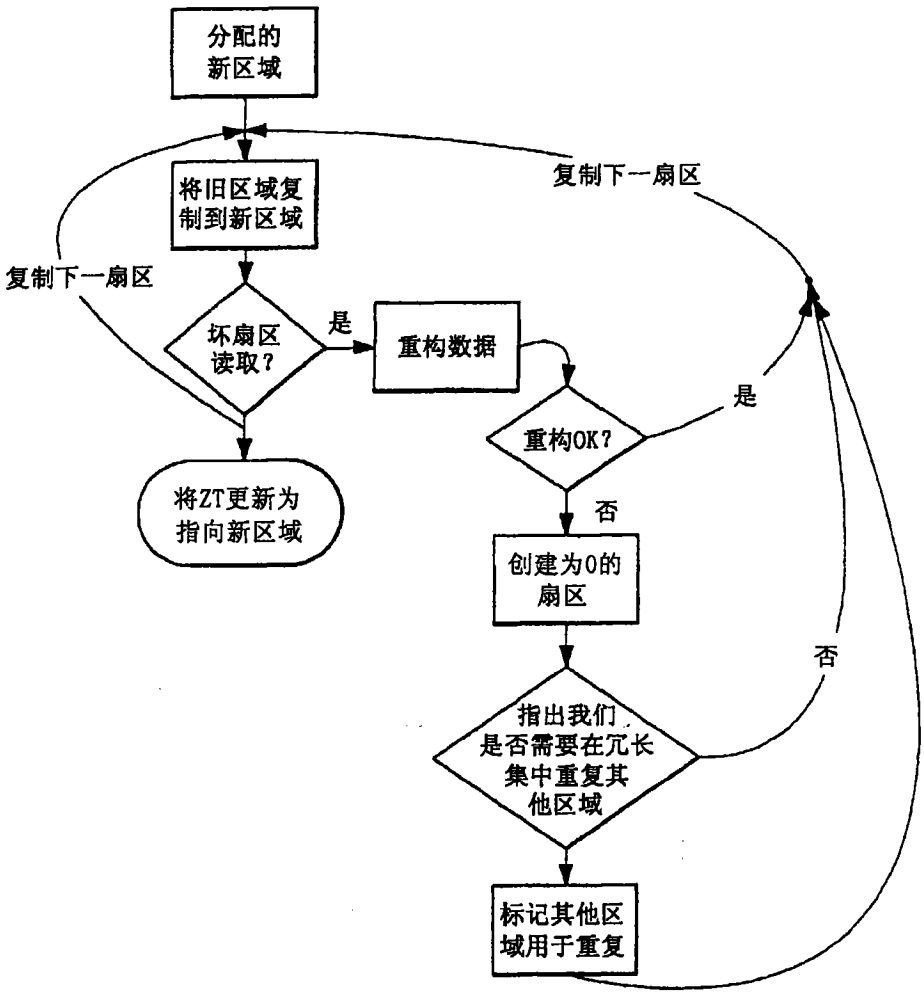

分配的新区域
将旧区域复制到新区域
复制下一扇区
复制下一扇区
坏扇区读取?
是
重构数据
重构OK?
是
否
将ZT更新为指向新区域
创建为0的扇区
否
指出我们是否需要在冗长集中重复其他区域
标记其他区域用于重复

图 20

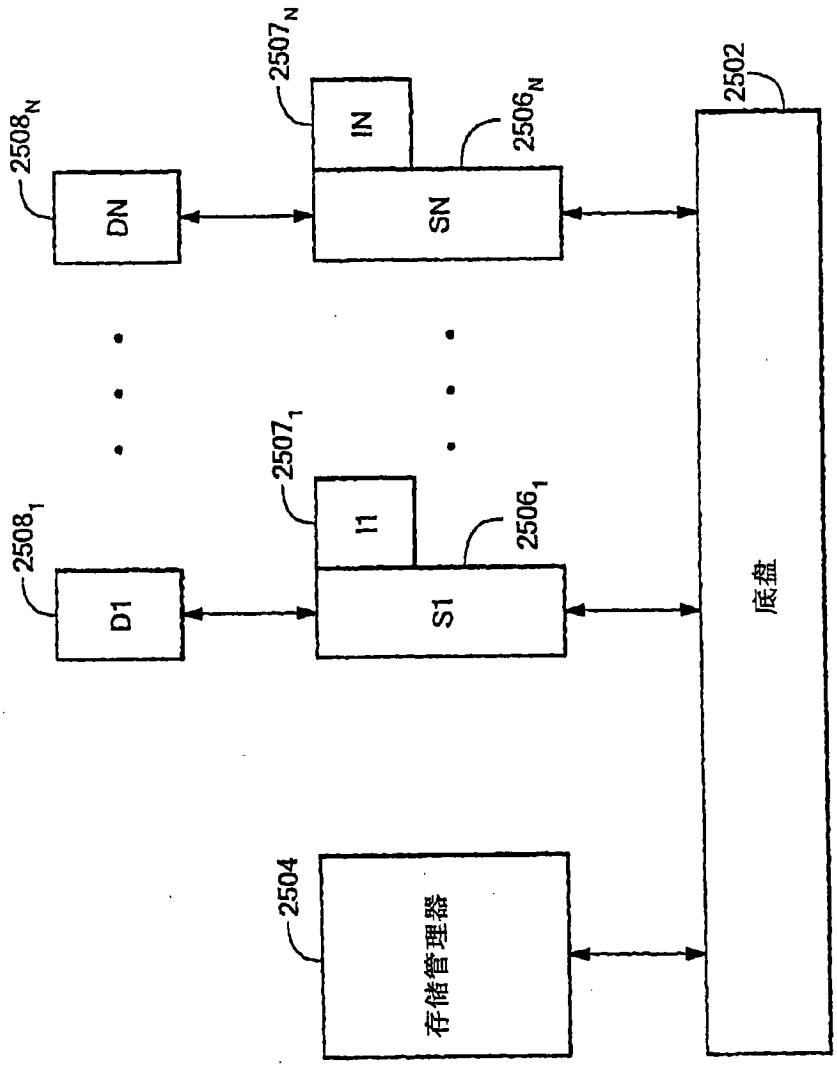

2508N
DN
2507N
IN
2506N
SN
2502
2508₁
D1
2507₁
I1
2506₁
S1
底盘
2504
存储管理器

图21

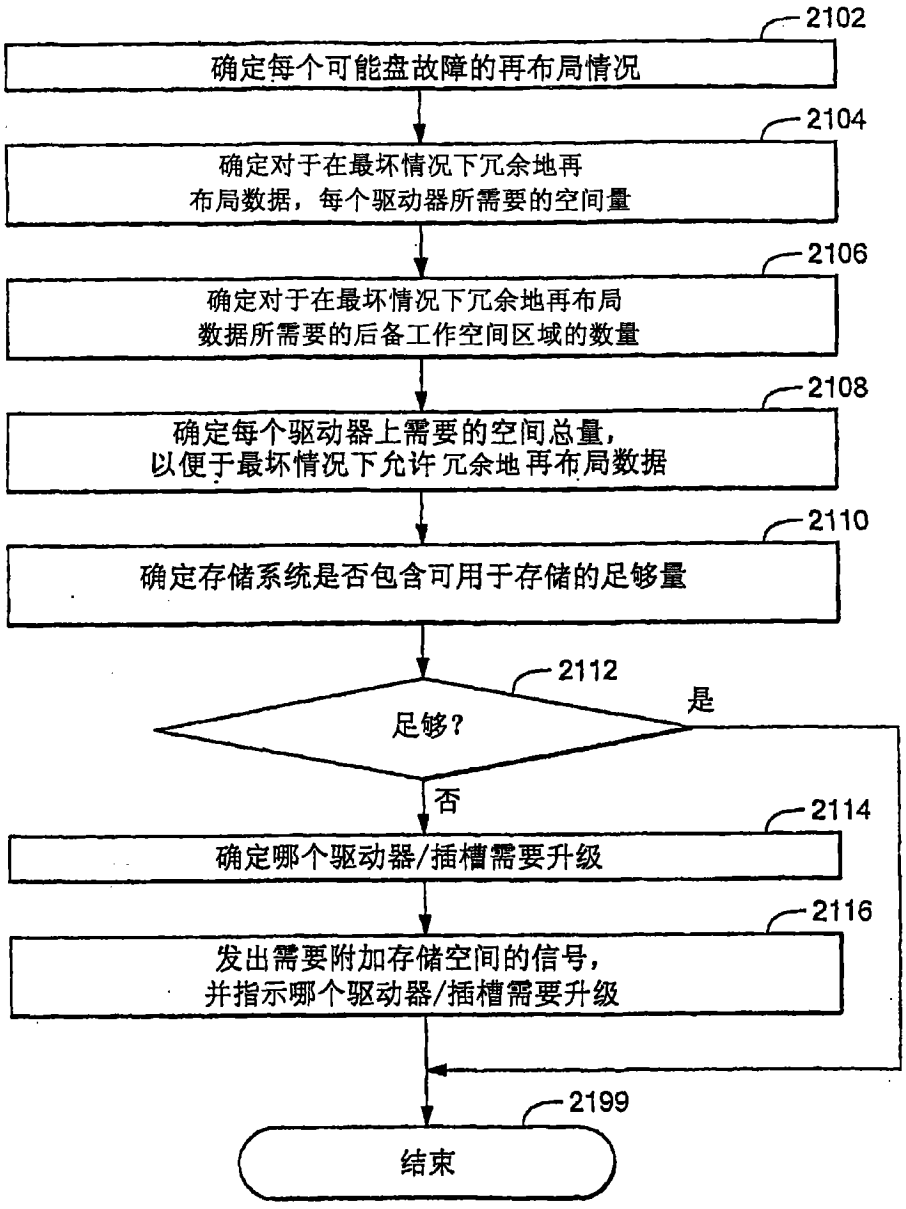

2102
确定每个可能盘故障的再布局情况
2104
确定对于在最坏情况下冗余地再布局数据，每个驱动器所需要的空间量
2106
确定对于在最坏情况下冗余地再布局数据所需要的后备工作空间区域的数量
2108
确定每个驱动器上需要的空间总量，以便于最坏情况下允许冗余地再布局数据
2110
确定存储系统是否包含可用于存储的足够量
2112
足够？
是
否
2114
确定哪个驱动器/插槽需要升级
2116
发出需要附加存储空间的信号，并指示哪个驱动器/插槽需要升级
2199
结束

图 22

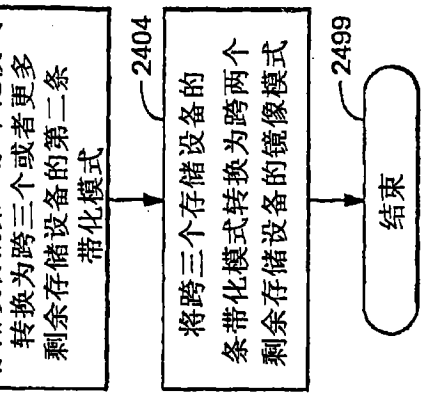

2306
2402
将跨四个或者更多存储设备的第一条带化模式转换为跨三个或者更多剩余存储设备的第二条带化模式
2404
将跨三个存储设备的条带化模式转换为跨两个剩余存储设备的镜像模式
2499
结束

图25

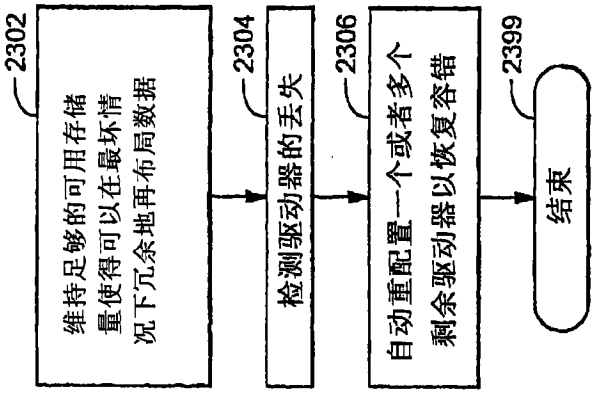

2302
维持足够的可用存储量使得可以在最坏情况下冗余地再布局数据
2304
检测驱动器的丢失
2306
自动重配置一个或者多个剩余驱动器以恢复容错
2399
结束

图24

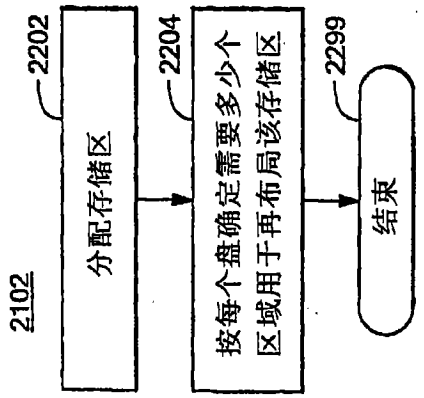

2102
2202
分配存储区
2204
按每个盘确定需要多少个区域用于再布局该存储区
2299
结束

图23

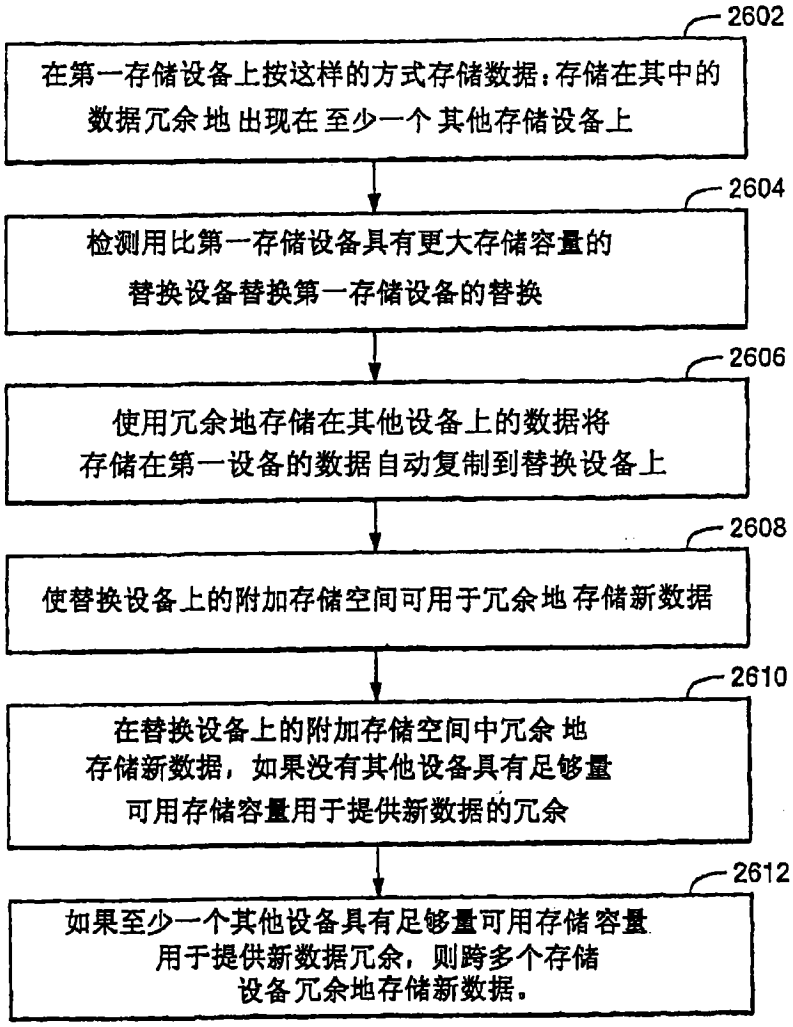

2602
在第一存储设备上按这样的方式存储数据：存储在其中的数据冗余地出现在至少一个其他存储设备上
2604
检测用比第一存储设备具有更大存储容量的替换设备替换第一存储设备的替换
2606
使用冗余地存储在其他设备上的数据将存储在第一设备的数据自动复制到替换设备上
2608
使替换设备上的附加存储空间可用于冗余地存储新数据
2610
在替换设备上的附加存储空间中冗余地存储新数据，如果没有其他设备具有足够量可用存储容量用于提供新数据的冗余
2612
如果至少一个其他设备具有足够量可用存储容量用于提供新数据冗余，则跨多个存储设备冗余地存储新数据。

图 26

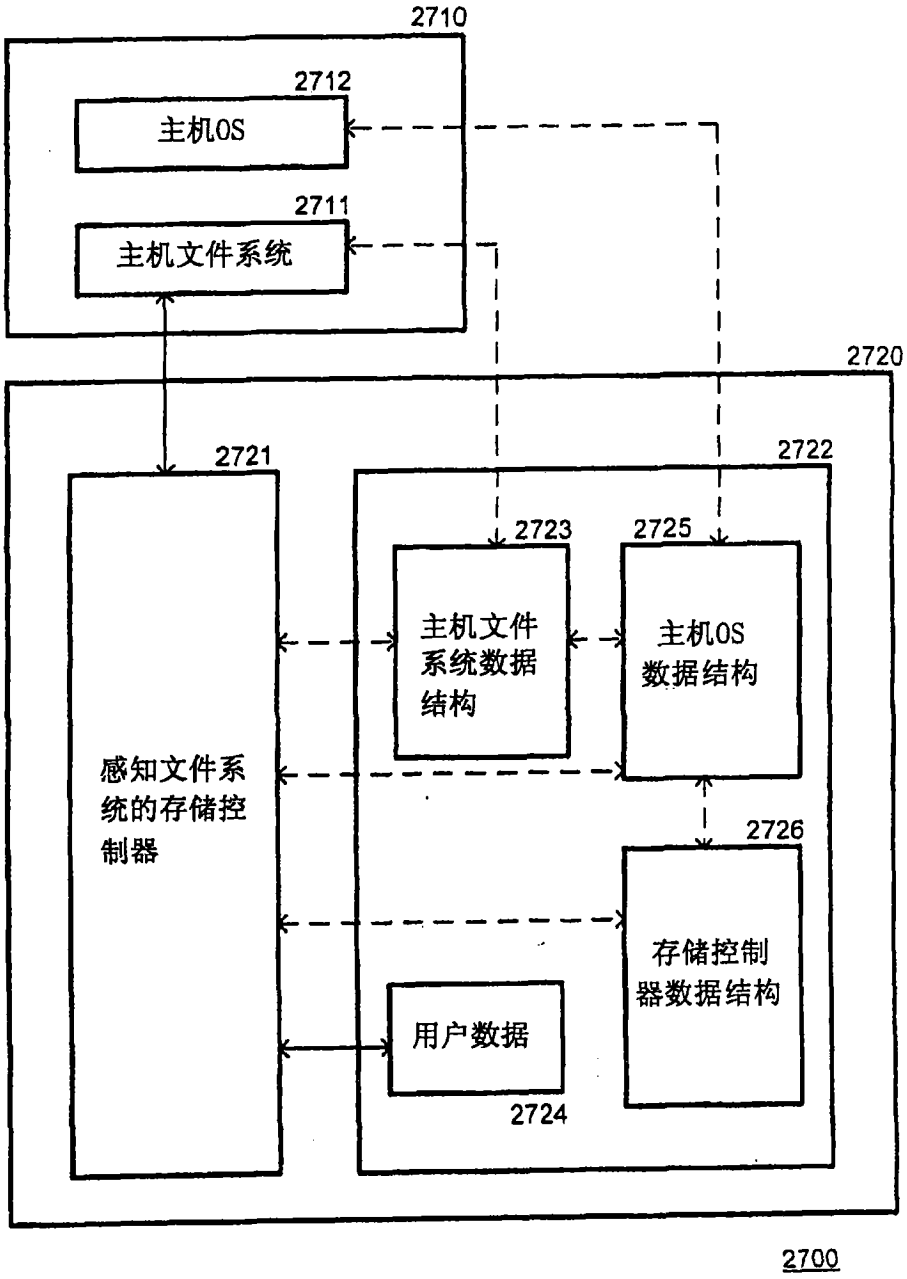

2710
2712
主机OS
2711
主机文件系统
2720
2721
2722
2723
2725
主机文件系统数据结构
主机OS数据结构
感知文件系统的存储控制器
2726
存储控制器数据结构
用户数据
2724
2700

图 27

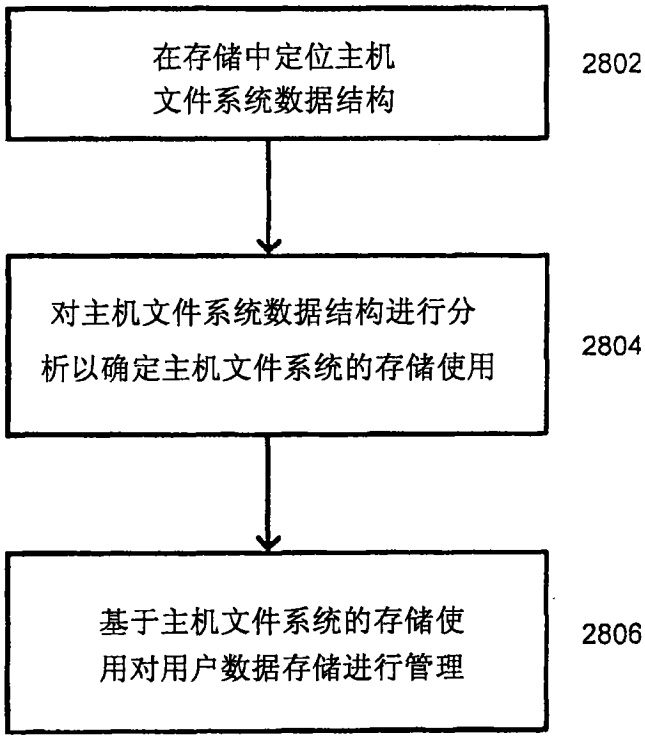

在存储中定位主机
文件系统数据结构
2802
对主机文件系统数据结构进行分
析以确定主机文件系统的存储使用
2804
基于主机文件系统的存储使
用对用户数据存储进行管理
2806

图 28

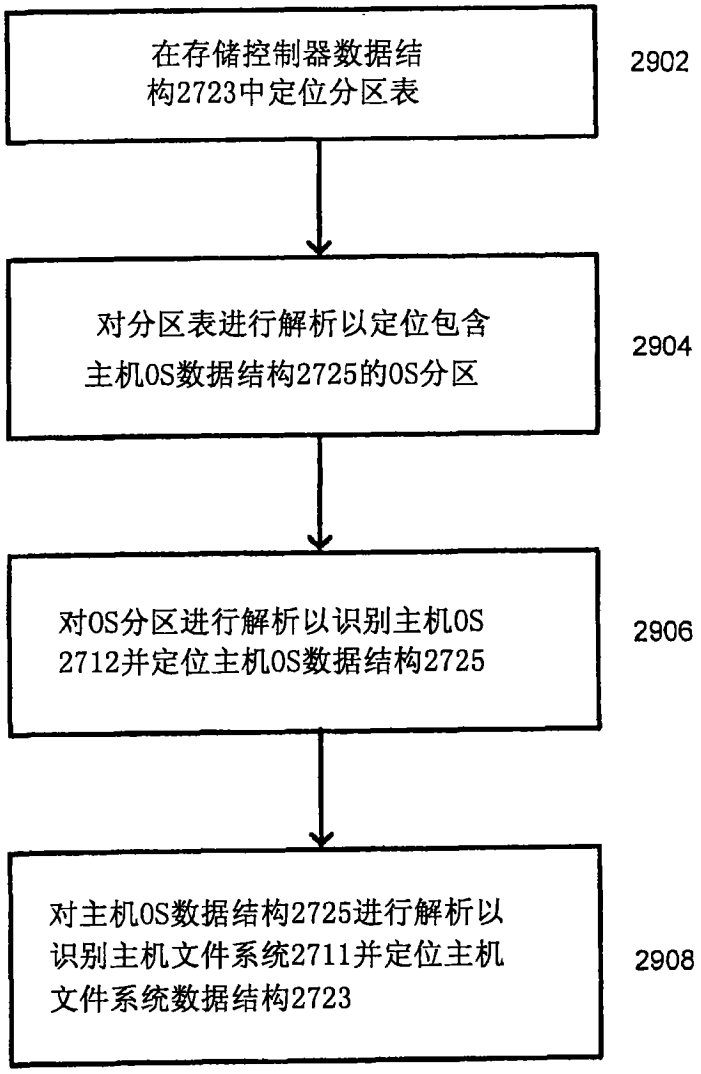

在存储控制器数据结构2723中定位分区表
2902
对分区表进行解析以定位包含主机OS数据结构2725的OS分区
2904
对OS分区进行解析以识别主机OS 2712并定位主机OS数据结构2725
2906
对主机OS数据结构2725进行解析以识别主机文件系统2711并定位主机文件系统数据结构2723
2908

图 29

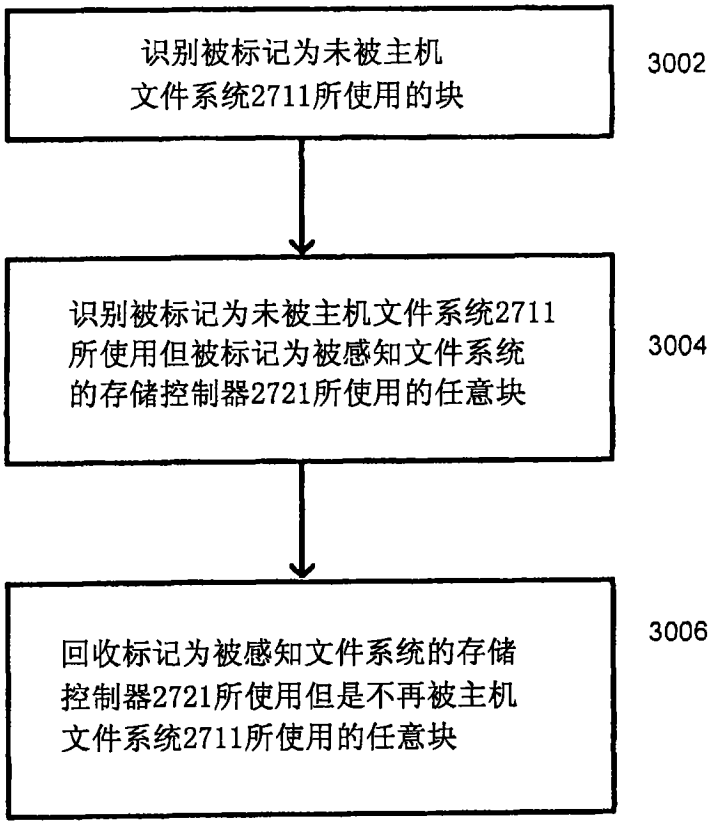

识别被标记为未被主机
文件系统2711所使用的块
3002
识别被标记为未被主机文件系统2711
所使用但被标记为被感知文件系统
的存储控制器2721所使用的任意块
3004
回收标记为被感知文件系统的存储
控制器2721所使用但是不再被主机
文件系统2711所使用的任意块
3006

图 30

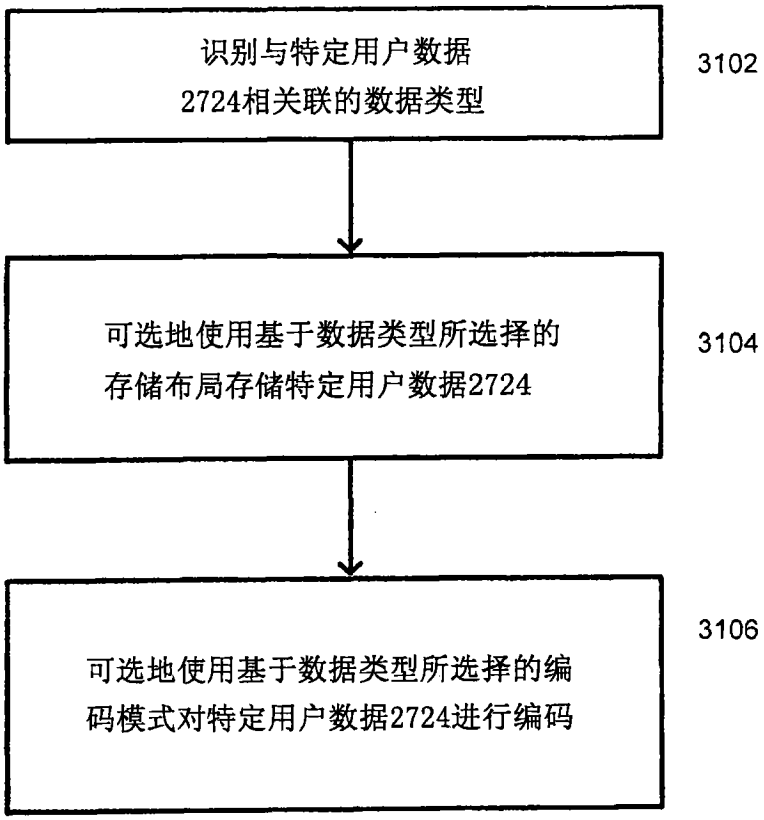

识别与特定用户数据
2724相关联的数据类型
3102
可选地使用基于数据类型所选择的
存储布局存储特定用户数据2724
3104
可选地使用基于数据类型所选择的编
码模式对特定用户数据2724进行编码
3106

图 31

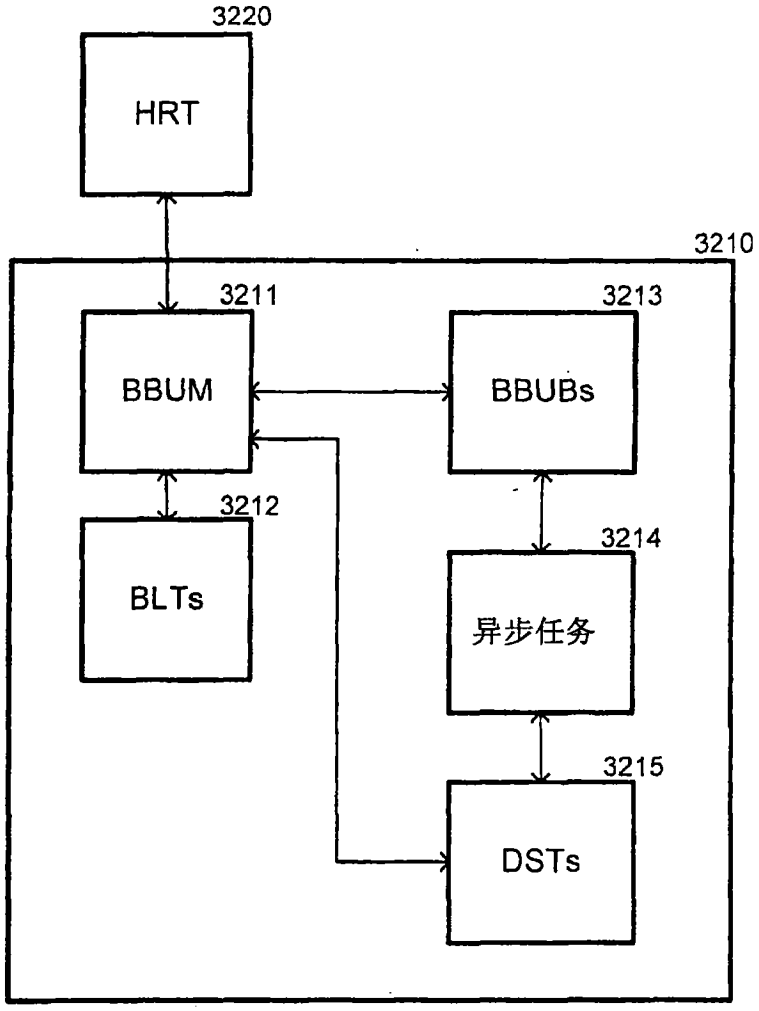

3220
HRT
3210
3211
BBUM
3213
BBUBs
3212
BLTs
3214
异步任务
3215
DSTs

图 32

```
查找在 LBA 0 的分区表
If（发现分区表）
    读取该分区表以识别分区
    For 每个分区
        读取分区的引导扇区以发现 MFT
        读取 MFT 的所驻留的$bitmap 记录以获得文件属性
        利用每个分区的引导扇区 LBA 编程 BLT
        利用位图记录的 LBA 编程 BLT
        利用实际位图的 LBA 编程 BLT
        标记引导扇区 LBA 以要求中间解析
        标记位图记录 LBA 以要求中间解析
        标记实际位图以不要求中间解析
    Endfor
Else
    结束而不向 BLT 添加附加位置
Endif
```

图 33

```
接收 ClientRequest
从 ClientRequest 获得 LUN
基于 LUN 发现正确的 BLT
从 ClientRequest 获得 LBA
在 BLT 中查找该 LBA
检查“immediate action”字段
If（需要 immediate action）
    同步处理 ClientRequest
Else
    为异步处理设置与 LBA 相应的 BBUB 位
Endif
```

图 34

```
If（块是分区表）
    将新数据中的分区与 BLT 中的分区进行比较
    If（在添加新分区）
        从新数据获得分区的开始和结束
        对 DST 检查任何重叠的 LBA 范围
        去除重叠的 LBA 范围
        向 BLT 添加分区的开始
        标记 "immediate action"
```

图 35

```
If（块是分区引导扇区）
    从新数据获得 MFT 的开始
    计算位图记录的位置
    If（对于该分区在 BLT 中已存在相同的位图记录项）
        不需要任何动作
    Endif
    If（位图记录与 BLT 版本是不同的位置）
        更新 BLT
        从盘片读取新位置
        If（看起来像位图记录即具有位图串）
            获得新的位图位置
            与 BLT 进行比较
            If（相同）
                不需要任何动作
            If（不相同的位置）
                设置所有的 BBUB 位
                更新 BBUB 映射
                在 BLT 中移动 LBA 范围
            If（小）
                收缩 BBUB
                将未映射的 LBA 范围添加到 DST
                在 BLT 中收缩 LBA 范围
             If（大）
                设置所有附加的 BBUB 位
                扩大 BBUB
                扩大在 BLT 中的 LBA 范围
        Else
            不需要任何动作
        Endif
    Endif
Endif
```

图 36

```
If（块是分区表）
    将新数据中的分区与 BLT 中的分区进行比较
    If（分区正在被删除）
        删除 BBUB
        从 BLT 删除引导扇区
        从 BLT 删除位图记录
        从 BLT 删除位图范围
        向 DST 添加分区范围
    Endif
Endif
```

图 37

```
解析 BBUB
For （BBUB 中设置的每个位）
    检查相应的聚簇是否被标记为未被主机文件系统使用
    If（聚簇被标记为未被主机文件系统使用）
        检查聚簇是否被标记为被存储控制器使用
        If（聚簇被标记为被存储控制器使用）
            向 DST 添加 LBA 范围
        Endif
    Endif
Endfor
For（DST 中的每个 LBA 范围）
    回收存储器空间
Endfor
```

图 38