(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 106462510 A
(43)申请公布日 2017.02.22

(21)申请号 201580023124.4

(22)申请日 2015.03.06

(30)优先权数据
61/949,190 2014.03.06 US

(85)PCT国际申请进入国家阶段日
2016.11.04

(86)PCT国际申请的申请数据
PCT/US2015/019304 2015.03.06

(87)PCT国际申请的公布数据
WO2015/134941 EN 2015.09.11

(71)申请人 伊姆西公司
地址 美国马萨诸塞州

(72)发明人 M·海默斯坦 J·亚伯勒 R·卡尔森 V·杜瑞乐 V·文卡塔拉加万 B·威尔福德 G·鸿 B·卡茨 R·凡加森贝克 D·亚力 D·R·爱伯森

(74)专利代理机构 北京润平知识产权代理有限公司 11283
代理人 金旭鹏 肖冰滨

(51)Int.Cl.
G06F 13/00(2006.01)
G06F 13/28(2006.01)
G06F 15/16(2006.01)

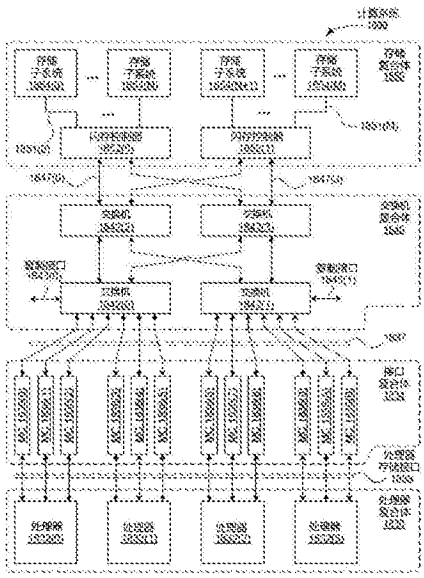
权利要求书2页 说明书31页 附图35页

(54)发明名称

具有独立直接接入大量固态存储资源的多处理器系统

(57)摘要

系统具有中央处理单元的集合。每一个中央处理单元连接至至少一个其他中央处理单元并具有至闪存存储资源的路径。中央处理单元支持从数据地址空间至闪存存储虚拟地址空间、至闪存存储虚拟页面码、至闪存存储物理地址空间的映射。



CN 106462510 A

1.一种系统，包括：

中央处理单元的集合，其中每一个中央处理单元连接至至少一个其他中央处理单元和至闪存存储资源的路径，其中中央处理单元支持从数据地址空间至闪存存储虚拟地址空间、至闪存存储虚拟页面码、至闪存存储物理地址空间的映射。

2.根据权利要求1所述的系统，其中中央处理单元的核具有至一组页面虚拟化表格项的专用接入，其中所述页面虚拟化表格项包括操作为针对闪存存储位置的索引的基本量和指定从所述闪存存储位置的步长值的偏置量。

3.根据权利要求1所述的系统，被配置为支持用于在中央处理单元之间传送命令和命令完成应答的无锁队列。

4.根据权利要求3所述的系统，包括：

执行队列；

完成队列；

第一中央处理单元，被配置为将尾值写入所述执行队列并消耗来自所述完成队列的头值；以及

第二中央处理单元，被配置为将尾值写入所述完成队列并消耗来自所述执行队列的头值。

5.根据权利要求4所述的系统，进一步包括能够接入所述第一中央处理单元和所述第二中央处理单元的比特表，所述比特表包括指定所完成的任务的项。

6.根据权利要求1所述的系统，其中中央处理单元的核具有由操作系统支持的独立输入/输出数据结构、在所述操作系统内的独立中断路径和专用硬件资源，以便于并行处理。

7.根据权利要求1所述的系统，进一步包括闪存存储控制器，用于在不利用所述中央处理单元的集合的情况下协调闪存存储资源之间的数据传递。

8.根据权利要求1所述的系统，被配置为在垃圾收集期间周期性地推迟读取。

9.根据权利要求1所述的系统，被配置为储存修剪信息的范围以减少记录需求，其中修剪信息的特征为失效的数据页面。

10.根据权利要求1所述的系统，被配置为将数据块和所关联的虚拟化表格储存在一起以便于从单个位置的数据恢复操作。

11.根据权利要求10所述的系统，其中所关联的虚拟化表格被储存在跨一组页面的条带中。

12.根据权利要求1所述的系统，被配置为响应于忙的存储资源的标识来重构数据。

13.根据权利要求1所述的系统，被配置为根据所述系统的操作时间自适应地实施更保守的数据保护协议。

14.根据权利要求1所述的系统，被配置为随机化闪存页面内容以最小化读取和写入干扰。

15.根据权利要求1所述的系统，其中中央处理单元被配置为将写入数据储存在DRAM中直至所述写入数据提交至闪存存储器。

16.根据权利要求1所述的系统，被配置为针对每一个数据保护条带利用单个序列号。

17.根据权利要求1所述的系统，被配置用于自适应垃圾收集，该自适应垃圾收集利用读取队列、写入队列和垃圾收集队列来选择性地从所述读取队列和所述写入队列装载作

业。

具有独立直接接入大量固态存储资源的多处理器系统

[0001] 相关申请的交叉引用

[0002] 本申请要求2014年03月06日递交的美国临时专利申请序列号61/949,190的优先权，该申请的内容通过引用的方式结合于此。本申请还是2014年04月09日递交的美国序列号14/249,289的部分继续申请，该申请要求2013年04月09日递交的美国临时专利申请序列号61/810,197的优先权。

技术领域

[0003] 本发明主要涉及信息处理。更具体地，本发明涉及具有独立直接接入大量(bulk)固态存储资源的多处理器系统。

背景技术

[0004] 随着越来越多的数据变得可用于分析，企业和政府需要能够开发这些数据以用于更快、更精确的进行决策和更有效的操作。

发明内容

[0005] 系统具有许多中央处理单元。每一个中央处理单元连接至至少一个其他中央处理单元并具有至闪存存储资源的路径。中央处理单元支持从数据地址空间至闪存存储虚拟地址空间、至闪存存储虚拟页码、至闪存存储物理地址空间的映射。

附图说明

[0006] 本发明更加充分地体现在以下结合附图的详细说明，其中：

[0007] 图1示出了根据本发明的实施方式配置的系统；

[0008] 图2示出了根据本发明的实施方式配置的根模块；

[0009] 图3示出了根据本发明的实施方式利用的分支和树架构；

[0010] 图4示出了根据本发明的实施方式利用的分支架构；

[0011] 图5示出了根据本发明的实施方式利用的流编码和解码技术；

[0012] 图6示出了根据本发明的实施方式可以利用的现成的组件；

[0013] 图7示出了根据本发明的实施方式利用的存储控制器；

[0014] 图8示出了根据本发明的实施方式可以利用的闪存接口；

[0015] 图9示出了根据本发明的实施方式利用的优先级FIFO方案；

[0016] 图10示出了根据本发明的实施方式可以利用的存储架构；

[0017] 图11示出了根据本发明的实施方式利用的逻辑单元FIFO控制器；

[0018] 图12示出了根据本发明的实施方式配置的逻辑单元控制器；

[0019] 图13示出了根据本发明的实施方式利用的闪存存储接口；

[0020] 图14示出了根据本发明的实施方式利用的数据保护系统；

[0021] 图15示出了根据本发明的实施方式利用的存储缓冲区；

[0022] 图16示出了根据本发明的实施方式利用的根复合体；

[0023] 图17示出了根据本发明的实施方式利用的地址映射方案；

[0024] 图18示出了根据本发明的实施方式配置的计算系统；

[0025] 图19示出了根据本发明的实施方式配置的存储控制器；

[0026] 图20示出了根据本发明的实施方式配置的闪存控制器；

[0027] 图21A示出了第一次写入之后的页面虚拟化表格；

[0028] 图21B示出了第二次写入之后的页面虚拟化表格；

[0029] 图21C示出了具有压缩和共享的项的页面虚拟化表格；

[0030] 图21D示出了具有伴随步长值的压缩和共享的项的页面虚拟化表格；

[0031] 图22示出了根据本发明的实施方式利用的无锁队列系统；

[0032] 图23示出了用于操作的系统与支持的硬件之间交互的现有技术方式；

[0033] 图24示出了根据本发明的实施方式的操作的系统核心组件之间的并行处理配置、专用输入/输出数据结构和专用硬件资源；

[0034] 图25示出了根据本发明的实施方式的事件计数；

[0035] 图26示出了根据本发明的实施方式执行的迁移枢轴(migrate pivot)；

[0036] 图27示出了依赖于中央处理单元的现有技术的垃圾收集操作；

[0037] 图28示出了根据本发明的实施方式的由闪存控制器执行的垃圾收集操作；

[0038] 图29示出了根据本发明的实施方式执行的垃圾手机操作；

[0039] 图30示出了根据本发明的实施方式利用的推迟读取垃圾收集技术；

[0040] 图31示出了根据本发明的实施方式的在一组LUN上实施的保护条带；

[0041] 图32示出了根据本发明的实施方式利用的初始页面虚拟化表格和存储关系；

[0042] 图33示出了根据本发明的实施方式的随后的页面虚拟化表格和存储关系；

[0043] 图34示出了根据本发明的实施方式的页面虚拟化、块虚拟化和存储关系；

[0044] 图35示出了根据本发明的实施方式的页面虚拟化、块虚拟化和存储关系；

[0045] 图36示出了根据本发明的实施方式的到LUN的并行接入；

[0046] 图37示出了根据本发明的实施方式利用的40Gb架构；

[0047] 图38示出了根据本发明的实施方式的通过每个核专用I/O结构和中断模块进行的并行处理；

[0048] 图39示出了根据本发明的实施方式的通过闪存控制器执行的关闭CPU(off-CPU)条带处理；

[0049] 图40示出了根据本发明的实施方式利用的DRAM快速写入缓冲区；

[0050] 图41示出了根据本发明的实施方式的异步I/O处理；

[0051] 图42示出了根据本发明的实施方式的同步I/O处理；

[0052] 图43示出了根据本发明的实施方式利用的数据保护技术；

[0053] 图44示出了根据本发明的实施方式利用的负载均衡架构；

[0054] 图45示出了根据本发明的实施方式的序列号处理；

[0055] 图46示出了根据本发明的实施方式利用的RAS条带处理；

[0056] 图47示出了根据本发明的实施方式执行的校验(parity)页面处理；

[0057] 图48示出了根据本发明的实施方式利用的集成的垃圾收集和数据保护；

[0058] 图49示出了根据本发明的实施方式利用的自适应数据保护；

[0059] 图50示出了根据本发明的实施方式利用的垃圾收集架构；

[0060] 图51示出了根据本发明的实施方式利用的基于序列的垃圾收集技术；

[0061] 图52示出了根据本发明的实施方式利用的页面随机化技术；

[0062] 图53示出了根据本发明的实施方式利用的LUN架构；

[0063] 图54示出了根据本发明的实施方式处理的命令；

[0064] 图55示出了根据本发明的实施方式处理的命令；

[0065] 图56示出了根据本发明的实施方式配置的微码引擎；

[0066] 图57示出了根据本发明的实施方式利用的冗余路径架构。

[0067] 同样的参考数字贯穿附图的几个图示指代相应的部分。

具体实施方式

[0068] 计算装置是具有集成的软件的专用硬件设备，所述软件被设计以提供特定计算功能。计算装置与通用计算机在任何基本方式上都没有不同，但是通常不被配置为允许顾客改变软件或重配置硬件。所公开的系统能够运行非常广泛范围的应用并且在这个意义上可以考虑作为通用计算机器。其实施节约成本的架构，该架构有效地创建了非常大的、共享的存储器。

[0069] 所公开的系统开发了借以提供“Flash As Memory™”的低成本的固态设备。这意味着固态设备(如，闪存存储芯片)在处理器的存储空间中具有地址。因此，处理器可以直接访问“闪存存储”中的数据，而不必首先将其交换至主存储器。在特定的实施方式中，该存储器空间存在非常大的范围，如几十太字节至几千兆字节。相应地，如以上所描述的特定设计和技术被使用。这些特定设计和技术支持跨系统的并行操作。

[0070] 设计和技术包括由硬件和软件二者组成的多个互联组件。每一个组件具有对于完整系统的操作所需要的唯一的特征和功能。在互联时，这些组件创建了所期望的计算能力。

[0071] 图1中展示了本发明的实施方式的框图。其包括多个CPU 100，每一个CPU 100具有许多通过存储速度接口102互联的计算核，有时称为集群连接。每一个CPU具有缓存104和本地存储器105(在这种情况下是DRAM)或另一类似类型的存储器。每一个CPU 100还拥有作为延伸的固态存储器操作的本地存储根108。计算核执行常驻在本地存储器105中的或在通过存储控制器110连接的分支上的软件堆栈106。在一个实施方式中，软件堆栈包括如以下讨论的应用程序、数据库、块驱动器及磨损(wear level)可靠性可用性可服务性(RAS)模块。该模块图示描述了本发明的多个可能配置中的一者。

[0072] 能够由计算机服务的同步数据接入的数量限制了许多数据驱动应用的性能。计算核的数量的增加使这个问题更糟糕。所公开的系统提供了大量的存储芯片，该存储芯片具有至包装计算核的CPU的多根互连。其提供对数据的有效并行应用接入。特定系统软件管理计算和数据接入的有效调度。

[0073] 所公开的系统可以以适于特定使用模式的各种配置来建立。本发明可以被优化以进行大量的特定用途，诸如这些大的存储消耗应用：商业智能、商业分析、地球-地震、医学成像、社交网络和患者管理。

[0074] 在一个实施方式中，根存储控制器110连接至互连的CPU 100的复合体，并且驱动

分支112和叶子114的层级，每一个CPU 100由多个核构成。观察到每一个分支附着至另一分支或多个叶子或二者的混合。叶子114由闪存存储器或其它固态或数字存储器组成。特别地，可以具有通过分支附着至单个根(如一个或多个FPGA或一个或多个ASIC)的1024个或更多的存储芯片。每一个CPU复合体可以被连接至八个或更多个根。因此，如果图1是准确的并且是按比例的，那么大量固态存储器叶子将淹没(overwhelm)该附图。在这个架构中，CPU具有可以并行接入的数千个存储目标。CPU复合体中的多个CPU及存储树的大小和数量还可以增加至非常大的量。平衡反映了特定用途(应用)的需求。该架构中的组件由硬件和软件二者组成。他们可以包括以下：

[0075] 1.数据管理系统

[0076] 数据库或数据管理系统，可以是、且常常是

[0077] 1)多线程的；

[0078] 2)利用单个共享存储模块，或分布式存储模块，或二者的组合，从而获得高度并行性。在一些实施方式中，这可以是缓存相干存储模块，其中每一个CPU线程缓存其在存储器中的状态。

[0079] 2.存储管理系统

[0080] 存储管理系统，可以是、且常常是

[0081] 1)多线程的以开发大的多核系统；

[0082] 2)高度并行的；

[0083] 3)非常大的容量

[0084] 4)作为比喻：向下移动存储管理系统导致提高并行性。在存储接入从根移动至分支至叶子时，在每一层次有效倍增并发操作。

[0085] 3.缓存管理系统

[0086] 在一些实施方式中，缓存管理系统维持跨计算机系统中的单独节点(或核)的数据相干性。

[0087] 4.存储系统

[0088] 如以上所述，每一个存储系统由根、分支和叶子组成。在一个实施方式中，具有四个根，在这里有时称为存储模块(MM)。从概念上讲，根取代了计算机底盘中的四个存储扩充卡。他们连接至分布网络，该分布网络提供了至多个分支的接口，每一个分支连接至多个叶子。

[0089] 图2显示了单个根108，该根108包括两个存储接口200、两个分支接口202、十六个分支和十六个叶子，每个叶子包含至少一个固态设备。图3是分支接口202及其至具有相应叶子114的一组分支112的连接的更具体的描述。在一个实施方式中，每一个根具有总共8TB的闪存以用于总共32TB的系统——一半可用于用户应用及一半被分配至冗余和系统使用。在其他实施方式中，分配至应用或冗余的存储资源可能是不同的或可以直接或间接地通过应用来控制。

[0090] 每一个分支接口具有内部存储器204以执行软件堆栈206。软件可以包含双数据速率(DDR)控制器、目标路由软件、RAS模块和非阻塞并行固态接口(NBSI)驱动器。分支接口可以具有FPGA或ASIC形式的计算资源。

[0091] 存储接口连接至处理器间数据分布网络，其中所有CPU具有对所有存储器的接入。

我们将存储器描述为由下文详细描述的分支和叶子组成的多根的树。我们将计算复合体描述为存储器共享的多处理器，该多处理器可以是相同或不同类型的。根可以是许多实例中的一者，这些实例是本地或远程连接的。互连技术可能影响一部分系统运作，但其不必改变基础架构或其操作。

[0092] 在一个实施方式中，存储器主控制器(MMC)和存储器从控制器(MSC)被实施具有专用硬件。根是MMC，而分支是MSC及叶子是固态存储设备。例如，Altera Stratix V FPGA可以被用于MMC和MSC二者。在这种情况下，每一个FPGA具有以12.8Gb/s操作的48个串行链路，并且来自每一个MMC的三个链路转到十六个MSC设备中的每一者。每一个MSC转而连接至16个叶子，每一个叶子是固态存储设备，例如32GB单个层单元(SLC)NAND闪存设备。许多其他实施也是可能的，包括组件及其互连是实时改变的实施。

[0093] 在一个实施方式中，存储分支是具有多个附着的叶子的组件，其中每一个叶子是闪存存储芯片，如图3所示。图4示出了具有叶子接口控制器400的分支112，该叶子接口控制器400执行软件堆栈402。软件堆栈402可以包括网络驱动器、RAS、纠错码(ECC)、数据库引擎、数据压缩引擎、加密引擎和固态驱动器。这些组件提供了在不需要将计算任务移动至另一处理单元的情况下在存储在叶子中的数据上执行计算任务的手段。观察到堆栈靠近媒介，因为在根和分支中具有计算资源。

[0094] 在可替换的实施方式中，存储分支是具有多个分支和附着的叶子的组件，其中每一个分支执行相同或不同的软件堆栈。在异构系统中，每一个分支可以知晓其邻居并协作实现网络驱动器、RAS、纠错码、数据库引擎、数据压缩引擎和固态驱动器。

[0095] 在进一步的实施方式中，如具有叶子的情况，分支包含重大意义的计算资源，该计算资源在数据被读取/写入至叶子时实施快速搜索或压缩/解压缩。本地处理器可以加密、压缩、擦洗、验证、编码和解码数据分组以及路由、验证、编码和解码报头和存在于CPU、分支和根组件之间的通信信道中的命令信息。

[0096] 分支最终以叶子结束。每一个叶子是用作读取和写入数据页面至非易失性存储中的设备。叶子可以以许多形式实施。存储管理系统控制叶子页面的使用。叶子可以以各种技术来实现，但是他们有已经写入的数据页面还可以被读取的性质。至于设备类型或操作参数叶子不需要是同构或异构的。

[0097] 在进一步的实施方式中，叶子包含意义重大的计算资源，该计算资源在数据被读取/写入叶子时实施快速搜索或压缩/解压缩。本地处理器可以加密、压缩、擦洗、验证、编码和解码数据分组以及路由、验证、编码和解码报头和存在于CPU、分支和根组件之间的通信信道中的命令信息。

[0098] 在一些实施方式中，一个或多个多层架构覆盖根-分支-叶子结构并包括如以下所描述的用于编码和解码的各种技术。

[0099] 在一些实施方式中，存储叶子具有至冗余系统中的镜像(mirrored)存储空间的端口接入。在一个实施方式中，大约一半的存储系统储存数据及另一半的存储空间支持快速数据接入。在另一个中，部分物理存储器被保留，以提供足够的性能。在进一步的实施方式中，存储器使用特别设计的最小化低效率的方法使数据在单独叶子之间分布。在另一实施方式中，存储组件本身可以由共同产生所需要的行为的单独的组件组成。在进一步的实施方式中，分段的系统已经隔离了在单独的域失败的情况下保持操作的域。在一个实施方式

中，这些组件是相互依赖的。为了使整个系统有效运行，互连组件相互依赖以正确运作并及时完成彼此的工作。

[0100] 本发明是由几个内部关联的部件组成的计算系统，几个部件可以具有有着不用用途的不同的实现产生机制。这些组件的有效相互作用以另外的达不到的水平创建了系统性能。在以下的阐述中，我们列举了几个组件及其运作。

[0101] 本发明的一个实施方式提供了在损耗和写入存储器方面的性能改进。资源(计算的、存储器、连接带宽等)的合并(pooling)创造了最优化的机会。当多个服务器试图为试图接入相同数据的多个客户端服务时，数据一致性变为重要的需求。多级缓存的使用进一步使架构复杂化，该架构可以被用于解决这些问题。

[0102] 本发明的实施方式是通过在一组设备中的所有设备之间分配工作来改进该组设备的性能的机制。传统的固态盘(SSD)在设备内进行“损耗均衡”并且必须进行这个操作而无论设备何时需要擦除块。这是因为他们仅能够在他们的设备内分配写入，即使更高级实体试图在其他设备上分配写入。在单个设备上执行单独动作的成本比当写入被合并到一起时的执行成本高很多。所公开的系统优化了大量信道上的擦除和写入，从而增强了整体性能。其包括在实质上没有增加客户端所关注的操作延迟的情况下明智地分散工作的“选择”机制和实时优化机制。

[0103] 例如，该技术在图1和/或图2的架构中使用。示例性的事务是使用日志结构分配数据页面，以提高收回(reclaim)擦除的块(如，“垃圾收集”)的效率。存储器的页面被映射至核的虚拟存储地址空间。页面以顺序的方式被写入连续固态存储地址。这些写入操作可以被分组在一起以更有效地利用存储设备的带宽。例如，四个页面可以被同时写入特定闪存设备的四个平面上。该方法在闪存设备的一个块移动至下一个之前填满闪存设备的这个块。因此，对于许多在再次执行写入操作之前以类似序列执行删除操作的应用来说；许多删除的页面可以从相同的块中同时收回。在其他情况下，写入操作可以被分派至不同的块，以最大化可以被收回的页面的数量。

[0104] 本发明的另一方面是有效移动存储层级中存储组件之间的块的机制。在以上示例中，我们具有包含从固态设备接入数据的处理器层级(CPU、根、分支)。任何时间数据从层级的一个等级移动至另一个等级，可能存在性能损失。相反，如果不跨等级情况下执行操作，通常会实现性能的改进。

[0105] 实现性能改善的机会的一个示例出现在擦除用于固态媒介的操作的期间。在准备擦除中，必须移动包含仍在使用(“活的”)的数据的所有页面。CPU不需要在这期间检查该数据；因此我们通过使用该数据不交叉CPU的层级等级来实现性能改善。一些实施方式可以将该数据从分支移动至分支及一些实施方式将该数据从根移动到根。在根之间移动的情况中，我们将副本移动至CPU并返回至原根或另一根。这些副本需要CPU中的存储空间，这转而影响了可用于根缓冲区管理的存储器、CPU上的缓存利用和CPU缓冲区。可以使用CPU中的负载/储存模块或使用根中的直接存储接入(DMA)模块来实现数据从根内或从根移动到根。使用一个路径或其他路径的决定可以在操作系统层中进行或可能在根本身中进行，或甚至使用二者结合的分布式机制进行。

[0106] 本发明的另一实施方式针对分配读取以最大化性能的机制。在具有不同延迟、吞吐量和接入方法的设备演化数据时，CPU不能接入数据并预期其在DRAM中进行预期的相同

性能。这个机制并行化读取以从非DRAM设备获取非常高的吞吐量。这些性能等级的数量级比其他方式更接近DRAM速度。

[0107] 在一个实施方式中，我们检查了包含具有8TB数据的固态存储模块的系统，该系统具有长的延迟和复杂的接入机制。特别地，根(存储模块)上的固态媒介与使用具有256GB数据的DRAM模块的单个64字节读取相比，允许2048与4096同时的字节读取。在本实施方式中，系统上每一个4096页面耗费100微秒及DRAM上每一个4096字节读取耗费1微秒。在本实施方式中，在同时读取之后，每个页面具有额外的5微秒用于系统。

[0108] 虽然使用DRAM比读取单个页面快100倍，但是当读取2048个页面时，DRAM仅比所公开的系统快5倍。使用更多存储芯片组成的较大实施方式大大缩小了这个差异。

[0109] 我们的发明包括软件、固件和硬件设计以实现以上所描述的并行性。本发明体现了其中读取请求和/或数据已经被分配从而随后的读取可以利用存储系统中的并行性的机制。基本的技术通过放置可能同时在不同的(独立的)设备(信道)上被读取的数据页面来实现。在一个实施方式中，每一个页面被写入与之前所写入的页面有关的不同信道(或分支)上的设备上。

[0110] 本发明的另一方面是用于分配写入以最大化性能的机制，并同时对缓冲区和信道带宽进行有效使用。之前注意到读取请求和/或数据必须是已经被分配的，以使得随后的读取可以利用并行性。本发明的本实施方式分配数据写入以便于随后读取的并行性。

[0111] 在一个实施方式中，使用了Linux操作系统。与大多数现代的操作系统类似，Linux使用存储管理系统，该存储管理系统将DRAM中的数据缓存至数据设备或从数据设备缓存DRAM中的数据，该数据设备类似于旋转盘或固态介质。应用可以通过直接使用Linux的DRAM、使用Linux的DRAM作为阶段区域或通过向Linux提供对应用的DRAM的参考来写入数据。在所有这些情况下，在时间和空间上分配写入。

[0112] 所公开的系统具有巨大量的独立操作单元。我们在那些单元之间分配页面写入。这是暂时的分配，因为我们基于在不同的单元准备好写入的时候向不同的单元写入页面。

[0113] 分配有助于两种方式下的性能。第一，正如读取，系统可以同时写入独立的单元(叶子)，实现并行性。第二，随后的读取还将被分配并因此能够利用并行性。因为具有如此多的叶子(其不会有相互间的读取冲突)，给定了暂时的分配和大量的单元。

[0114] 本发明的另一方面是在信道或逻辑单元之间实现损耗分组的机制。所公开的技术是有效再使用存储器的“擦除块”和在页面之间均匀地划分用途的方法。该技术可以在软件堆栈206中或在根108处(如图2的RAS)执行，或者通过二者协作来最优化地执行该技术。

[0115] 块擦除机制的工作与存储空间再利用(如，垃圾收集)非常类似。然而，该机制处理普通的垃圾收集技术不能处理的三个增加的并发症。第一，页面必须在其可以被再次写入之前在大的连续分块中被擦除。第二，页面最终耗尽，所以必须实现损耗均衡。第三，在尽可能多的信道之间扩展业务量以得到最佳读取和写入性能是令人满意的。通过实现了增强型一代垃圾收集器，可以获取良好的性能(解决点1和3)，并且还提供良好的损耗均衡(解决点2)。

[0116] 在一代垃圾收集器中，所有的新对象被放置在年轻一代的合并中。在垃圾收集在年轻一代的合并上执行之后，幸存对象的收集被放置在较老一代的合并中。幸存对象的收集再次被放置在更老的合并中，等等。这种如存储介质一样使用闪存的简单实现会导致极

坏的损耗均衡。用于年轻的存储一代的块会比用于较老一代的块擦除得更加频繁。我们通过周期性地改变物理闪存块属于哪一代来避免这个问题。通过采用高的擦除计数来将页面移动至较老的一代，我们减少了将来将看到的那些页面的擦除次数。用于不同的一代的块在不同的闪存信道之间平衡。这确保了能够在许多不同的信道之间扩展读取和写入业务量。

[0117] 本发明的另一实施方式涉及累积的冗余数据保护条带。本发明的本实施方式累积了将简单的(如，XOR)借/贷计算用在故障独立的存储设备上的数据保护条带。该技术包括快速重构损坏的数据块的方法。该机制可以体现在系统的CPU、根或分支中。依赖于特定的实施方式，将与不同方式下的RAS实施方式相互作用。

[0118] 应用于具有大量不对称读-写次数的设备的擦除码需要限制开销并提供大大减少未检出错误的方式。此外，非常需要低的计算开销和限制的空间。我们呈现了用于擦除译码数据的机制，该机制在空间和时间上获取低的、固定的具有可调节的错误检测和错误等级的开销。

[0119] 以下的数据保护方案建立在所公开的写入机制之上。图5示出了17个数据堆栈的流。在该示例中，计算校验值以每次用于四个数据块，但是该技术可以在任意数量的数据块上使用。校验值利用数据块的滑动窗口。为了改善效率，采用与新的数据块(如，D5)结合的结果，之前计算的校验值(如，P1)经历异或(XOR)操作。这个方法可以被级联用于所有新到达的数据。图5提供了建立方法有效性的数学基础。这个方法将低的固定的计算开销与固定的空间需求相结合。实际上，这个机制利用了“移动窗口”，该“移动窗口”包含了XOR的数据块以一起形成校验块。随着窗口移动，新的块与校验块XOR，而之前与校验块XOR的老的块再次与校验块XOR。这有效地增加了新的块并移动了老的块，从而“移动窗口”。

[0120] 本发明的另一实施方式针对在直接读取等待之前请求的擦除或比读取慢的其他操作时通过校验重构数据，以完成并释放信道。这个功能可以在软件堆栈的软件中实现，该软件堆栈在图1的CPU上运行。特别地，在本实施方式中，功能在软件堆栈的底部实现。通常，功能针对在初级设备忙的时候通过从储存在不同设备上的冗余数据来重构所接入的数据以用于最小化读取操作的延迟的方法。

[0121] 存储页面被分派不同的信道(如，从CPU通过根至叶子的链路)。每一个叶子储存多个数据页面。仅单个数据页面可以从叶子一次读取或一次写入叶子。叶子内的页面有时被擦除。执行擦除或其他慢操作比读取或写入需要更长时间，并且多个读取和写入是在叶子之间同时进行的。所以，期望避免在擦除操作正在进行时从叶子进行读取。

[0122] 因此，除了数据页面，我们还储存数据保护页面。也就是，对于给定的数据页面A，我们针对A储存一组数据保护页面，这些数据保护页面储存在不同的叶子上。在数据页面A丢失的情况下，数据页面A的内容可以通过读取一些针对A的数据保护页面和一些其他数据页面(不包括A)来进行重构。需要注意的重要的事情是数据页面A的内容可以在不必接入A所在的叶子的情况下被重构。

[0123] 该技术通常的应用是在包含A的页面(或甚至整个叶子或分支)丢失的情况下重构A的内容。然而，我们还可以使用该机制来改善性能，如下：如果包含A的叶子忙，并且将会忙很长时间，代替直接读取A，我们可以通过必要的数据保护信息重构A。在擦除的情况下，常常这将容许满足对A的请求比等待完成擦除更快。该技术可以与移动窗口技术相结合，以快

速重构数据。

[0124] 实现该技术的另一方法是利用类似擦除码的RAID 5(或其他RAID)。这意味着计算多个数据页面的异或以产生校验页面或计算更复杂的码，例如低密度校验码或所谓的Raptor码，这允许恢复所需要的数据。

[0125] 本发明的另一实施方式针对分级存储器中的有效读取缓冲区利用的机制。问题是通过更有效的缓冲区利用来改善系统性能。不是在软件发布读取命令的时候分配缓冲区，而是仅在数据变为可用之前分配缓冲区(缓式分配)。这允许软件读取在较少量的缓冲区上进行统计学复用。在限制缓冲区的数量的系统中，这导致更好的整体系统性能。

[0126] 当CPU将数据从较慢资源读入较快资源(诸如从闪存存储器读入高速存储器)时，CPU分配较快资源中的缓冲区并等待较慢设备填充特定缓冲区。缓冲区可以从操作开始通过操作系统被看作“忙”直至数据最终被返回。通过这个想法，系统发布读取，但是不会为数据预分配缓冲区。较慢的系统(从该系统读取)将仅在数据被传递至CPU之前分配缓冲区并且然后用数据填充缓冲区。这使缓冲区在较短的时间周期是“忙”的。这个机制改善了缓冲区在较快资源中的利用，这转而引起提高的系统性能并减少要求特定性能等级的缓冲区的数量。

[0127] 本发明的另一实施方式针对在最小化叶子资源时优化性能的数据保护方案。数据保护(DP)校验的发生一般是简单的且是非常快的。花费了很长的时间来校正大量的比特错误。实际上，所遭遇的大部分错误具有少量的错误比特。

[0128] 本发明的一个实施方式产生了大量的校验比特并具有双重校正算法。少量的错误比特可以在分支或叶子的硬件中进行校正，保持硬件实现很少。如果出现大量的比特错误，其可以通过使用与数据一起储存的大量的校验比特中的所有(或较多)的校验比特在软件或固件中进行校正。

[0129] 该层级机制改善了空间和计算效率。该机制将数据保护分离为两个部分：一个较小的部分，该部分具有有限的校正能力，及较慢但更有能力的部分，该部分在软件中具有校正能力(或一些其他“更高级别”的处理)。对于所有需要的校正中的99.999％的校正，叶子内的校正逻辑将是足够的。然而，当叶子不能校正数据时，软件将使用其额外的能力来校正数据。这将小的、频繁使用的叶子校正块加上非常不频繁使用的软件校正块与单个、较大的叶子块与不是非常频繁使用的广泛功能进行交换。进行这种交换导致了小得多的、更好利用的叶子块并改善了叶子的空间效率。该机制可以被级联至多个等级，每一个等级根据之前等级的数据校正能力进行改善。最终，其他系统等级数据保护可以被应用于从其他叶子上的冗余副本来恢复丢失的数据。基于块的码(如，BCH或Reed-Solomon)可以执行这个功能。此外，特别设计的低密度校验(LDPC)码可以被使用。这个“软错误”技术允许从设备重新读取数据并结合多个读取意图以产生较好的错误率，并且还可以与以上用于将错误检测和错误与需要更复杂逻辑的稀有情况分开的技术相结合。

[0130] 以下讨论针对本发明组件的各个特定实施方式。如以上所讨论的，系统组件包括与点对点分布系统连接的多处理器/多核复合体和多根的、冗余的并行可接入的(分支的)存储器。系统可以被配置有商业上可用的组件，如图6所示。

[0131] 多核、多处理器、服务器类、硬件平台被用于实现本发明的实施方式。在一个实例中，IBM 3850/3950X5系统(基于英特尔的Boxbor-EX平台)作为主机。工作模式包含4个DRAM

存储板和具有64太字节的闪存存储器的4个存储模块(根)。部分闪存存储器可用于用户应用。该部分可以依赖于使用历史和当前的、所测量的性能等级而改变。该系统代表在非均匀存储架构(NUMA)配置中所连接的多个多处理器系统，该配置具有高速点对点、缓存相干存储互连。

[0132] 硬件组件可以包括：

[0133] ○系统基板(母板)

[0134] ○可扩展存储接口(SMI)板(根)

[0135] ○闪存模块基板(分支)

[0136] ○存储模块子卡(叶子)

[0137] ○支持机械的、热的及电力的系统。

[0138] 图6显示了可以用于实现本发明的多核、服务器类平台。特别地，该图是IBM X5 3850机架的框图。基于英特尔的Boxboro-EX平台，包括经由快速路径互连(QPI)链路互连的四个Xeon 8870(Westmere)CPU。每一个Xeon芯片具有两个存储控制器。每一个存储控制器在闭锁步骤中运行两个英特尔SMI(可扩展存储互连)链路，以实现至存储器(具有检测比特的144比特)的128比特宽的数据路径。每一个SMI链路与英特尔7510扩展存储缓冲芯片通信，该芯片转而将SMI请求转变为双倍数据速率类型三(DDR 3)同步DRAM事务。在每一个7510上具有两个DDR 3链路。Xeon处理器执行128比特操作并且不在DDR 3链路上使用数据掩码比特。

[0139] 存储控制器和交叉棒是被设计为在串行链路接口与MSC内闪存控制器之间的传输数据页面的分布式存储结构。存储控制器是系统中的分支。闪存页面是所利用的特定芯片的属性，但是一般是4KB存储块。擦除块是闪存页面大小的倍数(如，256KB或512KB)。

[0140] 存储控制器被用于缓冲闪存页面和控制Interlaken接口与闪存控制器之间的消息。Interlaken是用于结合线(单独数据连接)的互连协议。也就是，其是与多信道的单个逻辑连接。MSC已经结合了数据传输信道(如2个Interlaken接口)，每个MMC一个。软件可以在两个MMC之间均匀地将页面调度至分支。在一个实施方式中，每个MSC具有32个分支。当在每一个MMC上具有入站和出站页面时，数据路径可以被划分为4个主要部分。

[0141] 复用到MMC0的来自32个分支的业务量

[0142] 复用到MMC1的来自32个分支的业务量

[0143] 解复用出至分支的来自MMC0的业务量

[0144] 解复用出至分支的来自MMC1的业务量

[0145] 这显示在图7中。项700表示一个数据路径；项702表示另一个数据路径，等等。还在图示中显示了统计块704，该统计块704仅发送STATS(统计)消息至MMC0。

[0146] 该分布式存储设计很好地有助于Altera芯片架构，该架构在整个设备扩展其m20k存储器。来自分支的128比特路径将通过Interlaken FIFO 706进行携带并进入分支FIFO的“顶部”接口。在Interlaken接口与分支之间具有时钟域改变。这在分支FIFO的“底部”接口进行。在图8中显示了示例性接口。

[0147] 该整个存储控制器被建立为一组分布式FIFO 800。本实施方式具有简单的FIFO，但是可以将优先权给予仅由报头构成的消息。在从Interlaken 706至闪存控制器802的“向南”方向，这允许将请求读取至“追上(overtake)”写入页面。在向北方向，允许将完成写入

至追上读取页面。仅FIFO进入闪存控制器将需要改变为优先FIFO。

[0148] 优先FIFO使报头和页面准备好同时传输至Interlaken接口706。如果Interlaken接口706是用于页面但不用于报头的流量控制器，那么报头可以围绕读取页面流动，允许在没有什么事情可以继续时完成写入。这种情况会使存储连贯性难于或不可能维持。

[0149] 为了确保存储内容的一致性，可以使用基于优先级的机制。如图9所示的优先FIFO由两个FIFO构成，一个大型的由存储器900建立且一个小型的由逻辑902建立。输入处的解复用块904会检查消息类型并将消息转发至合适的(短的或长的)FIFO。在FIFO的输出侧，采用简单优先方案的调度器906选择哪一个FIFO下一个发送其消息(具有所附页面)。

[0150] 在一个实施方式中，数据路径接口是通常的4线(开始(START)、结束(END)、有效(VALID)、准备好(READY))类型接口，该接口具有128个数据比特。该接口将被用于至Interlaken块的接口以及FIFO的两侧。这还可以被用于对至复用器的业务量进行流量控制。该接口可以被用于存储控制器中的所有块。

[0151] 图9的两个FIFO类型可以使用4线(开始、结束、有效、准备好)接口，虽然两个准备好信号可以被用于根据根来选择保持哪个队列，如图2所示。

[0152] 闪存控制器连接至存储控制器的底部并控制一个闪存信道及控制闪存设备的操作。图10示出了示例性存储器和闪存存储器架构。在一个实施方式中，每一个MSC具有32个闪存控制器。每一个闪存控制器与一个8比特闪存信道通信。每一个闪存设备具有两个信道，每一个信道具有4个逻辑单元(LUN)。因此每一个闪存控制器控制在与其进行通信的信道上的4个LUN。

[0153] 叶子存储控制器连接至存储控制器FIFO 1000、1002、1004和1006。在这个工作模式中，叶子由“闪存”存储芯片构成。写入页面当其到达闪存控制器时被存储在写入页面缓冲存储器中并且当数据需要被发送至闪存信道时通过信道调度和控制时钟读取。来自闪存的读取页面被存储在读取页面缓冲存储器并然后适当地向MMC0或MMC1发出。

[0154] 在一个实施方式中，闪存控制器被划分为5个主要部分：命令和页面解析器1008、LUN控制器FIFO 1010、LUN控制器1012、信道调度和控制1014及数据保护。在该设计中，至闪存控制器的命令可以在多个位置进行解析。这些可以包括：基于信用的流量控制、命令解析器、LUN控制器、信道调度器和在闪存控制器顶部的解复用块。基于信用的流量控制块可以从MMC接收信用(CREDIT)命令。这些命令被用作无操作(NOP)命令，仅为了将缓冲区充满度信息从MMC携带至MSC。信用控制器从数据流移除这些命令。该块显示在MSC设计页面上的主MSC框图中。解析命令的所有其他块在图10所示的闪存控制器框图中显示。

[0155] 参考图11所显示的，特别地，命令&页面解析器108寻找页面和写入命令。页面命令使多件事情发生。第一，缓冲区从写入页面缓冲存储器1102进行分配，及缓冲区的地址被放置进用于寻址的LUN的页面队列1104。然后该页面的剩余部分从输入FIFO(如，1000)传递至所分配的存储缓冲区。页面(PAGE)命令然后通过命令解析器块1008从数据流移除。

[0156] 命令解析器块1008用状态机检查写入命令，如果两个页面跟随写入1-页面命令到达，那么就存在错误。在那种情况下，将FREE_BUFFER(释放_缓冲区)命令插入命令队列并且LUN控制器释放缓冲区而不是留其分配不用。一旦正确的写入命令被检测到，就将其放置进写入队列1106。所有其他命令进入“其他”队列1108。FREE_BUFFER命令进入写入队列1106。

[0157] LUN控制器必须知道哪个命令去往闪存且它能处理哪个。LUN控制器可以直接处理

write_error(写入_错误)、ping(脉冲)和free_buffer。所有其他命令具有一些闪存交互并将通过信道调度器进行处理。LUN控制器执行从读取缓冲存储器的缓冲区分配和写入缓冲存储器中的缓冲区释放。

[0158] 信道调度器解析用于读取的操作码、写入和执行命令。这些是主命令。RESET_LUN(重置_LUN)和RESET_MSC(重置_MSC)还被理解为其重置闪存设备上的LUN操作。信道调度器发布合适的读取和写入命令至闪存并在缓冲区之间移动数据。完整的命令通过LUN控制器以及读取命令的缓冲地址传送至已完成队列1110。

[0159] 解复用1112使完整的命令传送至合适的MMC FIFO。必须还理解操作码命令。读取完成在LUN FIFO中具有并行页面地址信息。解复用1112生成正确形式的PAGE(页面)命令以用于将闪存页面传输至合适MMC。

[0160] 命令被解析为两个分组并放置到两个队列,写入队列1106和用于所有其他命令的队列1108。所有数据页面被放置进写入页面缓冲区1102并且缓冲区的地址被传送至页面队列1104。持有从其他命令分离的写入允许读取的优先级高于写入命令。

[0161] 在返回方向,即从闪存设备至MMC,所有具有响应的命令按执行的顺序放置在已完成队列1112。任意页面数据,即读取页面,具有放置在读取页面缓冲区1114中的数据,及该缓冲区的地址在页面队列1104中进行传送。

[0162] 图12显示了LUN控制器的实施方式。LUN控制器直接连接至LUN FIFO。LUN控制器具有两个主要分组的命令:转到闪存设备的命令和不转到闪存设备的命令。例如,PING是不转到闪存设备的命令的示例。ping命令完全在LUN控制器中执行。PING命令通过命令调度器进入并且直接发出至LUN FIFO中的已完成队列1112。WRITE_ERROR(写入_错误)和FREE_BUFFER也完全在LUN控制器中进行处理。

[0163] 所有其他命令与闪存设备具有一些交互。

[0164] RESET_MSC(重置_MSC)命令重置读取和写入缓冲存储器中的所有缓冲区。还会在进行中异常中断任何命令并发布RESET_LUN命令至闪存设备。RESET_MSC命令应仅在没有其他命令在进行的时候被调度。

[0165] 通常,LUN控制器将向信道调度控制器“呈现”命令。这意味着有效的命令,该命令由操作码、0、1或2个闪存地址及读取和写入缓冲地址构成。在写入的情况下,写入地址指定的存储位置被用于写入闪存地址的闪存。在读取的情况下,闪存地址被用于读取页面并被写入读取地址指定的缓冲区。

[0166] LUN控制器将保持读取、写入和闪存地址及操作码(在以上图示的底部显示)直到信道调度器已经在闪存信道上传送命令并具有结果。在命令的最后,闪存设备的状态被传送至LUN控制器并且LUN控制器将状态比特写入返回消息并终止命令。一旦命令完成,信道调度器就指示多少缓冲区应当被释放(在写入情况下)或多少缓冲区应当现在被分配(在读取情况下)。在此之后,命令调度器选择哪个命令队列应当被读取,并且下一个命令被呈现给信道调度器以用于在闪存设备中执行。

[0167] 图13示出了根据本发明的实施方式利用的信道调度器和控制块。信道调度器被配置用于闪存设备初始化并复用来自LUN控制器的命令请求和数据传递。在一个实施方式中,所有命令和数据在8比特闪存信道上传递至闪存设备。初始化包括至闪存设备的初始RESET_LUN指令和接下来的命令,以启动同时传递模式(闪存以非同步模式开始)。在需要设

置驱动器强度和其他初始化参数的情况下，初始化ROM用于以命令模式在接口上引导命令。一旦所有四个LUN被初始化，主命令轮询环被启动。通常，READ_STATUS_ENHANCED(读取_状态_增强)命令用于查看LUN是否空闲及查看之前的命令是否已经完成。该命令还有当寻址到特定LUN时所有其他LUN不能驱动来自8比特数据总线的数据或不能读取来自8比特数据总线的数据的副作用。

[0168] 如果闪存设备中的LUN空闲且命令可用，那么命令被发送至LUN。在写入的情况下，数据还在信道上传递至LUN。一旦命令在进行中，内部闪存BUSY(忙)状态就用READ_STATUS_ENHANCED命令进行轮询。一旦命令完成，从命令返回的状态就返回至LUN控制器。在读取命令的情况下，数据在信道上从闪存设备读取并发送至读取页面缓冲存储器。在擦除的情况下，没有页面数据需要在闪存信道上传递。

[0169] 数据保护可以用三个步骤来执行：增加保护校验比特至原始数据、处理数据比特和校验比特以查看是否存在错误(产生校验子(syndrome)以指示哪个比特是错误的)及校正发现的错误(如果有的话)并恢复原始数据比特。常常后面二者围绕FIFO进行结合。一般来说，数据分成块。一个问题是数据校正需要校正比特所在的数据块和信息，并且常常块在“调整比特”信息可用的时候已经“过去”。图14显示了这些块。特别地，附图示出了检查比特的块1400、调整比特的块1402和增加比特的块1404。

[0170] 增加DP比特块1404在其通过页面解析器被写入时接受4kB页面并且每一个512B的数据插入额外104B。在我们写入页面时，每一个512B变为包括校验字节的616B。在增加DP比特块1404需要写入存储器时，可以暂停页面流量并使输入FIFO后退一点，所以校验比特的插入是相当直接的。写入页面缓冲区中的闪存页面现在被保护并可以被写出至闪存。

[0171] 一段时间之后，当我们从闪存读取页面时，所保护的页面从闪存读取并通过检查DP比特块。该块将字节直接传送至读取页面缓冲区并使用数据字节和校验字节来产生指出每一个块中有什么错误的校验子。校验子信息通过LUN控制器以及缓冲地址中的之前非特定的队列。

[0172] 在从读取页面缓冲区读取闪存页面数据时，校验子中指定的校正可以被应用于616字节的块并且它们可以被写入解复用块1406作为512B的校正块。如果校验子比预期的大很多，那么可以使用FIFO。

[0173] 在一个实施方式中，至闪存控制器的输入包括4线握手及16比特的数据。4线是在其他方向流动的start_bit(开始_比特)、end_bit(结束_比特)、valid(有效)和ready(准备好)。闪存控制器的底部连接到实际的闪存设备并因此通过设备的数据表指定端口分配。

[0174] 系统基板(SBB)经由八个连接件附着至计算机机架中四个根接口板(RIB)的一侧的底部。四个存储模块基板(MMBB)卡插入SBB顶面上的连接件。除了功率分配，SBB单单是出于简化至计算机机架的连接的目的的互连机制。

[0175] 承载卡充当用于十六个分支的基板。每个根具有一个承载卡并每个系统高达八个。该板通过合适的连接件向承载卡提供功率分配。冗余的热插拔功率还向该板供应对接。

[0176] 分支卡被设计为现场可替换单元(FRU)。该FRU由单个叶子控制器和十六个固态设备构成。6U附件可以安装在计算机上的19”架子中。还包括足够冷却单元的电源和风扇，也是EMI屏蔽。

[0177] 现成的12V电源能够以220V AC或110V AC功率运行单元。12V供给电极合适地进行

分配并向下调节至必要的较低电压。本地稳压器(VR)和低压差稳压器(LDO)提供调整。

[0178] 为单元供电的12V的使用最终支持电池备份早期写入终止(EWT)单体。不间断电源(UPS)以AC输入水平进行操作。

[0179] 存储缓冲区(如,英特尔7510可扩展存储缓冲区)在概念上基于如JESD82-20A所描述的JEDEC完全缓冲的双重内联存储模块(FBDIMN)高级存储缓冲区(AMB)。图15显示了如两个独立的AMB 1500、1502的存储缓冲区的逻辑示图,每一个AMB操作一对DDR3RDIMM。与JEDEC AMB不同,英特尔7510不支持菊花链。因此,需要一个7510芯片以用于SMIB卡上的两个SMI总线中的每一者。

[0180] 与分支和叶子结合的根复合体创建了用于数据页面的分布式网络,该数据页面将被写入叶子中的固态存储器(如闪存存储器)页面。图16是显示…的根复合体的表示。

[0181] 创建以用于工作模式的软件包括Linux块设备驱动器和存储模块仿真器。仿真和单元测试框架可用于运行Linux块设备驱动器和存储模块仿真器。

[0182] 软件组件通常包括商业智能应用、欺诈检测应用、程序化交易应用或其他需要大型数据集及因此的大型存储器的应用。此外,应用常常需要对数据的随机接入和高读取/写入比率。这些应用是拥有者/使用者可以直接将时间结果转化为利益的类型。需要快速的(通常为实时的)响应的其他应用(诸如社交网络、大量玩家的在线游戏及实时数据挖掘)对大型存储器具有类似需求以保存正在处理的数据的。

[0183] 理想地,应用在系统中运行需要很少修改或不需要修改。否则,应用的优势可能一直通过调整/更改应用来获得,以利用系统的硬件和软件组件。

[0184] 在一个或多个实施方式中,数据库是数据集接近100TB或更多的相关的或对象数据库。这些数据集不能使用基于DRAM的存储系统进行成本有效的处理并且它们不能使用基于磁盘的系统在合理时间处理。设计用于在多核/多处理器环境下执行的数据库维持支持这些环境的算法和接口。本发明可以有效利用这些接口。一些数据库可能为了提供跨多个存储块分配数据接入的并行执行而进行修改。

[0185] 闪存存储硬件需要将由操作系统请求的逻辑数据地址与闪存存储硬件的物理地址解耦合。最低限度上,呈现给操作系统的数据地址空间与闪存设备的物理地址空间解耦合。该重新映射以单个储存数据页面的间隔来执行,对于该系统,该页面是4096字节的页面。其还有助于能够重新映射物理级闪存,以处理不良块替换和执行块重新排序以用于损耗均衡。这种重新映射可以在硬件中处理,但是还可以在软件中通过在闪存块上执行额外级别的地址转化来执行该重新映射。这种块重新排序在擦除块级别执行。

[0186] 我们把操作系统查看到的地址空间称为数据地址空间(DAS)。在该空间中的地址称为数据地址(DA)。数据地址的页码部分是数据页码(DPN)。

[0187] 我们把闪存存储的虚拟地址空间称为闪存虚拟地址空间(FVAS)。在该空间中的地址称为闪存虚拟地址(FVA)。闪存虚拟地址的页码部分是闪存虚拟页码(FVPN)。

[0188] 最后,我们把闪存存储的物理地址空间称为闪存物理地址空间(FPAS)。在该空间中的地址称为闪存物理地址(FPA)。闪存地址的页码部分是闪存物理页码(FPPN)。应当注意的是在x86中存在已知术语虚拟地址(VA)、虚拟页码(VPN)、线性地址(LA)、线性页码(LPN)、物理地址(PA)和物理页码(PPN)。我们不想使用术语逻辑地址或物理地址,以避免与x86的具有相同缩写的术语混淆。因而我们通过引用闪存虚拟或闪存物理地址替代仅虚拟或物理

地址来明确消除混淆。数据地址空间、闪存虚拟地址空间和闪存物理地址空间之间的映射如图17所示。

[0189] 负责提供数据的固件在可以改变大小的数据块上进行并行操作。较小的块比较大的块提供得更快。固件可以在ASIC或其他硬件中实现。

[0190] 图18示出了根据本发明的一个或多个实施方式的计算系统1800。如所示的，计算系统1800包括处理器复合体1830、接口复合体1834、交换机复合体1840和存储器复合体1850。处理器复合体1830可以包括一个或多个处理器1832。每一个处理器1832可以包括一个或多个通用中央处理单元(CPU)核、一个或多个多线程图形处理单元(GPU)核或其任何技术上可行的组合。在一个实施方式中，处理器复合体1830包括四个处理器1832(0)至1832(3)，其中每一个处理器1832包括至少十五个CPU核。至少十五个CPU核中的每一者可以包括本地缓存(如，L0缓存)或包括本地缓存的缓存层级。在另一实施方式中，至少一个处理器1832包括一个或多个GPU核。处理器1832通过处理器存储接口1833耦合至接口复合体1834。在一个实施方式中，至少一个处理器1832对应图1中的至少一个CPU 100。

[0191] 如所示，每一个处理器1832可以耦合至接口复合体1834内的多个存储控制器(MC)1836。在一个实施方式中，存储控制器1836对应图1的存储控制器110。在一个实施方式中，每一个处理器1832耦合至三个或更多存储控制器1836。每一个存储控制器1836通过交换机接口1837耦合至交换机复合体1840，该交换机接口1837包括独立链路。如所示的，每一个存储控制器1836耦合至在交换机复合体1840内的所关联的交换机1842上的独立端口。在一个实施方式中，交换机1842耦合在一起以形成非阻塞交换机集群(诸如交叉交换机)，该交换机集群配置为将接入请求从存储控制器1836转发至存储复合体1850并将由接入请求产生的数据传回相应的存储控制器1836。每一个交换机1842可以通过相应的闪存控制器接口链路1847耦合至存储复合体1850内的一个或多个闪存控制器1852。在一个实施方式中，交换机1842(2)和1842(3)每一者耦合至闪存控制器1852(0)和1852(1)二者。在此类实施方式中，交换机1842(0)和1842(1)可以包括复制接口1843，该复制接口1843配置为将存储请求复制到另外的存储复合体(未示出)、另外的交换机复合体(未示出)或其结合。所复制的存储请求可以反映出通过交换机1842(2)和1842(3)传送至存储复合体1850的存储接入请求(如，读取、写入请求)。

[0192] 存储复合体1850包括闪存控制器1852和存储子系统1854。每一个闪存控制器1852可以通过存储接口1851耦合至相应的存储子系统1854。在一个实施方式中，每一个闪存控制器1852耦合至多个存储子系统1854。每一个存储子系统1854可以包括一个或多个集成电路存储设备。在一个实施方式中，每一个存储子系统1854包括一个或多个闪存存储设备。在另一实施方式中，一个或多个存储子系统1854包括DRAM存储设备。在特定实施方式中，每一个存储子系统1854对应图1的叶子114。

[0193] 在正常操作期间，存储接入请求由处理器1832产生。存储接入请求通过存储控制器1836传送，并由交换机1842转发至合适的闪存控制器1852。每一个闪存控制器1852将给定的存储接入请求引导至合适的存储子系统1854，其中与存储接入请求相关联的数据驻留在该存储子系统。存储接入请求应答从闪存控制器1852返回。给定的应答可以包括由存储读取请求所请求的数据块或响应于存储写入请求的写入应答。缓存可以在存储控制器1836、交换机1842、闪存控制器1852或其任何组合中实现。在特定实施方式中，写入缓存可

以向由处理器1832产生的写入请求提供更低延迟应答。例如，在一个实施方式中，存储控制器1836实现写入缓存，从而在所关联的数据写入相应的目标存储子系统1854之前，写入完成应答从存储控制器1836传送回至处理器1832。

[0194] 在一个实施方式中，每一个存储控制器1836被配置为提供请求队列以容纳由多个处理器核和/或与每一个处理器1832关联的多个进程线程产生的多个未决存储请求。可以对读写请求给定比关联于存储复合体1850的写入请求高的队列优先级，从而可以对写入应答给定比关联于处理器复合体1830的应答的读取应答高的优先级。请求队列操作至少在图7-10中更加具体地描述。容纳多个未决的存储接入请求(如，并发的、未完成的读取请求和写入请求)为存储控制器1836创建多个设计结果和需求。

[0195] 在一个实施方式中，通过在处理器1832上执行的处理产生的存储地址从数据地址空间(DAS)重新映射至闪存虚拟地址(FVA)空间，并且进一步映射至闪存虚拟页码(FVPN)，并最后映射至闪存物理地址空间(FPAS)。该映射之前在图17中进行了描述。部分重新映射可以由处理器1832、存储控制器1836、交换机1842、闪存控制器1852或其任意结合来执行。

[0196] 计算系统1800的实施方式需要以在规模上显著大于(如，数量级大于)传统计算机系统的配置来操作，同时对每一个处理器1832保留对大规模数据的共同接入，该大规模数据可以驻留在存储复合体1850中。由于大的处理规模、应用数据的异常地大的存储规模及向处理器1832内执行的所有处理提供对驻留在存储复合体1850内的潜在的所有应用数据的共享接入的需求，特定的额外设计特征可以在计算系统1800内有利地实现。此类设计特征传统上在一般计算系统中不需要。与大规模有效性能相关联的特定设计特征包括：(1)可扩展分页操作、(2)存储容量的可扩展分配和利用、(3)为了容纳大型、低延迟读取和写入操作进行的可扩展存储页面和块操作、(4)对大量数据集的数据保护及(5)与传统固态存储设备(如，闪存存储器)所关联的操作限制有关的固态存储器的性能优化。

[0197] 更特别地，具有多个执行线程的多个处理器上的大型计算一般将产生大量并发的、独立的存储接入请求。此类存储接入请求可能被引导至存储复合体1850中的应用数据的共享数据图像，导致与传统计算系统相关的极其集中的接入利用。

[0198] 此外，因为给定的应用数据覆盖区(footprint)的数量级(如，数百太字节至数百拍字节)大于包括少于太字节的传统应用数据覆盖区，计算系统1800有利地实现了应用数据的数据保护。在此类设置中的数据保护是极其有帮助的，因为主要的固态存储技术中的数百太字节的数据的物理存储软错误率可能在目标应用的运行时间期间产生许多错误。特定存储设备技术甚至可能引起相对频繁比率的多个、并发的错误，导致需要两层或更多层的数据保护。在特定设置中，数据的整个块可能被破坏或“擦除”，这需要实现擦除码或技术等同以提供数据保护和恢复。例如，如果包括存储子系统1854的固态闪存存储设备失效或如果其中数据块被破坏，那么数据块就会丢失(如，擦除)。可以实施擦除码来恢复数据块，该数据块诸如在以上示例中丢失的数据块。导致数据显著损失的失效事件是非常罕见的，其没有激励传统系统设计者来开发特征以解决此类失效事件。然而，采用大小在数百太字节至许多拍字节的有效的、固态存储器主机应用数据，如果计算系统1800的适当操作不是必要的，数据保护就会变成非常有利。

[0199] 在特定实施方式中，计算系统1800可以实现可扩展分页操作以容纳与处理器1832接入存储复合体1850相关联的非常高吞吐量的、低延迟的存储操作。特别地，计算系统1800

可以实现用于提供压缩虚拟页面表格的机制，该表格实现为执行不同的地址空间映射(如，以上所讨论的DAS到FVA、到FVPN到FPAS映射)。因为存储复合体1850内的目标块范围非常大，所以传统虚拟页面映射技术会导致大的、无效的映射表格。于此公开的虚拟页面映射技术针对更多的存储接入和更高的执行效率的减少了整体表格大小。该技术在图21A-21D中描述。

[0200] 此外，计算系统1800可以实现无锁队列以用于在两个处理器之间传送命令和命令完成应答，而无需处理器阻止执行进行，前提是在队列中存在空间。在一个实施方式中，无锁队列被实现为循环缓冲区，如结合图22所描述的。额外的用于无锁缓冲区操作的技术结合图24和25描述。

[0201] 在特定实施方式中，计算系统1800可以实现存储容量的可扩展分配和利用以容纳极大量的应用数据覆盖区。特别地，计算系统1800可以实现在没有处理器干预的情况下移动存储复合体1850内的活动(配置为激活应用处理)块的机制。此类移动操作(于此称为迁移枢轴)结合图26和28进行描述。在一个实施方式中，迁移枢轴被实现为容纳数据保护。此外，计算系统1800可以实现所分配的读取拷贝操作，以准备用于擦除操作，从而实现高性能读取操作。该技术结合图29和30进行讨论。该操作可以使包括闪存存储设备的存储复合体1850有效服务来自处理器复合体1830的读取/写入请求。同时还收回和准备最近将要写入的页面。计算系统1800还可以实现修剪(trim)范围功能以结合驻留在存储复合体1850内的一个或多个文件系统进行操作。

[0202] 在一个实施方式中，存储复合体1850被配置为储存与应用数据相关联的数据块，以及储存所关联的虚拟映射表格/虚拟化表格、不良块信息、修剪信息和在技术上与存储复合体1850内的数据操作和重构有关的其他数据。通过将相同保护的数据集内的虚拟化映射、不良块映射等储存为目标数据，数据的全部恢复和与存储复合体1850相关联的映射可以仅使用可用于存储复合体1850上的数据来有利执行。作为该技术的一个示例，图31示出了处置保护条带内的元数据。相反，许多传统存储系统储存独立于目标数据的虚拟化表格，在恢复期间创建效率低下。图32-34示出了储存驻留在存储复合体1850内的块内的虚拟化信息的特定示例。图35示出了储存驻留在存储复合体1850内的块内的不良块信息。在每一情况下，擦除码保护可以跨块实现。

[0203] 在特定实施方式中，计算系统1800可以实现可扩展存储页面和块操作以容纳至存储复合体1850的大型、低延迟的读取和写入接入。这些操作被实现为在存储复合体1850内每一个可用存储子系统1854上获取总横截面的带宽的高度利用，从而向处理器复合体1830提供极高的存储带宽。在一个实施方式中，大型并行存储架构实现可扩展存储页面和块操作。大型并行存储架构的一个实施方式在图18中示出，并且概念上的细节在图36-38中进一步示出。在特定实施方式中，DRAM缓存提供了预读取缓存和重构相关的计算资源，例如可以在存储控制器1836的每一者中实现。用于预读取的DRAM缓存进一步在图39中示出。跟随在写入数据写入指定的目标之前的应答，一个实施方式通过DRAM缓冲区中的写入缓冲提供了写入操作的快速应答。DRAM缓冲区可以在存储控制器1836内实现。该技术在图40中示出。可替换地，DRAM缓冲区可以在系统存储器(未示出)内实现，该系统存储器与处理器1832相关联或直接耦合至处理器1832。

[0204] 在传统系统中，存储接入性能通常在处理较大(如，8MB)块接入请求时由系统效率

限制。操作系统可以提供异步操作，但是与管理大块接入请求相关联的开销可以消除通过执行异步输入/输出操作增加的效率。在一个实施方式中，用于多个、并发的输入/输出操作的技术改善了与执行大量输入/输出操作相关联的性能，该大量输入/输出操作诸如通过在包括一个或多个处理器1832的多个核上分配相关的工作量进行的大块读取/写入操作。图41和42更具体地示出了该技术。

[0205] 读取性能可以通过这里称为“环读”(read-around)的技术改善，从而与包括存储子系统1854的忙的存储资源相关联的数据块被重构，而不是读取。存储资源因为包括存储资源的块被写入而可能是忙的。存储资源可以包括这里称为LUN的子电路。在特定情况下，写入数据块比重构数据块耗费时间长很多(如，长20倍)。因此，环读技术与等待和执行目标数据的直接读取相比可以提供性能优势。该环读技术在图43中进一步示出。在特定实施方式中，写入操作被调度以有利地便于环读时机从而改善平均读取性能。在一个实施方式中，为不同的存储控制器1836根据每一者的可用带宽提供存储带宽。可用带宽可以表示固定系统特征或正在进行的工作量。图44更具体地示出了这个概念。

[0206] 在特定实施方式中，计算系统1800可以实现大数据集的数据保护。在一个此类实施方式中，如图45和46所示，计算系统1800可以结合序号实现数据保护以在系统崩溃的情况下实现页面虚拟化表格(PVT)的重新创建。在一个实施方式中，如图47-49所示，计算系统1800可以根据系统生命期实现针对不同的失效特征的自适应数据保护。例如，系统最初可以采用较少的保守保护方案和在操作时间的某一时刻过渡至更保守的保护方案而进行操作。在特定设置中，垃圾收集与数据保护方案相互作用。这种相互作用由于与需要在写入闪存存储器之前擦除闪存存储器所关联的实际需求而被进一步复杂化。在一个实施方式中，计算系统1800实现在垃圾收集期间写入块的循环分配系统，及实现可以根据自由空间阈值触发的收回机制。在一个实施方式中，连续的数据保护和连续的垃圾收集通过自适应地平衡应用接入请求和垃圾收集活动来提供。本实施方式在图50-51中示出。以上技术有利地使计算系统1800能够有效地在提供高度数据保护的同时操作在高性能级别。如之前所讨论的，数据保护非常有利地给定了计算系统1800支持的应用存储覆盖区的规模。

[0207] 在特定实施方式中，计算系统1800可以实现固态存储器的关于与传统固态存储设备(如，闪存存储器)相关联的操作约束的性能优化。一种约束涉及与闪存设备内的接入邻近数据块相关联的读取和/或写入“干扰”。图52中描述的随机化技术用作减少此类干扰效应的影响；该技术因此减少了净错误率和关联的性能减少机制。在一个实施方式中，有效命令处理电路模块实现了接入抽象，该模块包括用于实现具体的接入控制的一组状态机，状态机发信号至包括存储子系统1854的闪存设备。命令处理电路模块在概念上在图53-56中示出。计算系统1800的特定实施方式需要高度的故障容差以用于高可用性计算并因此实现组件级冗余。此类冗余的一个示例在图57中示出。

[0208] 以上技术和实施方式可以在各种系统架构中独立实现，然而其可以在计算系统1800内一起有利地实现以提供高性能、高可用性的用于执行需要异常地大的应用存储覆盖区的应用的计算平台。

[0209] 图19示出了根据本发明的一个或多个实施方式的存储控制器1836。如所示，存储控制器1836包括处理器接口模块1960、邮箱获取引擎1962、一组命令引擎1964、一组接口缓冲区1966、多个DRAM引擎1968、交换机接口模块1970、命令解析器1972和命令队列(CQ)引擎

1974。

[0210] 在一个实施方式中，处理器接口模块1960通过处理器存储接口1833耦合至处理器1832。在一个实施方式中，处理器存储接口1833实现PCI表示(TM)接口。处理器存储接口1833可以被配置为接收与存储接入请求有关的命令，该存储接入请求包括读取请求和写入请求。每一个存储接入请求可以包括用于待从存储复合体1850读取或待写入至存储复合体1850的任意大小的数据的请求。在特定实施方式中，处理器接口模块1960被配置为实现与图22有关的讨论的无锁通信和命令队列技术。

[0211] 邮箱获取引擎1962通过处理器接口模块1960检索命令(如，接入请求)并将请求邮递至接口缓冲区1966内的合适的执行队列邮箱存储器。命令解析器1972解码命令并引导合适的执行引擎，诸如一组命令引擎1964内的命令引擎。例如，读取数据块的命令可以通过邮箱获取引擎1962获取、向接口缓冲区1966内的执行队列邮箱存储器邮递执行、通过命令解析器1972解析并向驻留在一组命令引擎1964内的RBD命令引擎传送执行。

[0212] 在一个实施方式中，DRAM引擎1968包括至少读取存储数据缓冲区(RMD)、写入数据存储缓冲区(WDM)和写入存储闪存缓冲区(WMF)、复用器和DRAM特定接口，诸如DDR3存储控制器。此外，每一个DRAM引擎1968可以包括至少一个DRAM存储设备。

[0213] 包括写入存储接入请求的命令可以至少包括目标地址、请求范围(如，大小)和根据请求范围即将写入目标地址的写入数据块。在特定实施方式中，写入数据块由处理器接口模块1960接收并直接写入一个或多个DRAM引擎1968。可替换地，写入数据可以首先写入包括接口缓冲区1966的SRAM缓冲区。一旦写入数据块写入SRAM缓冲区或一个至少一个DRAM引擎1968中，应答就可以传送回至相应的请求方(如，处理器1932)。写入数据块通过WDM缓冲区写入至一个或多个DRAM存储设备，并且随后通过即将通过交换机接口模块1970传送至存储复合体1850的WMF缓冲区从相同的DRAM存储设备进行检索。在存储复合体1850中，写入数据块写入包括存储子系统1854的存储设备。

[0214] 包括读取存储接入请求的命令可以包括至少目标地址和请求大小。在特定实施方式中，所请求的数据块通过交换机接口模块1970从存储复合体1850接收并写入包括接口缓冲区1966的SRAM缓冲区。可替换地，所请求的数据块可以写入一个或多个DRAM引擎1968。命令队列引擎1974完成每一个命令并引起即将通过处理器接口模块1960传送回至命令发起方(如，处理器1832)的应答。

[0215] 如所示，存储控制器1836包括使用DDR3DRAM设备实现的三个DRAM引擎1968及使用双向40GE链路每一者实现的包括交换机接口1837的两个链路。在其他实施方式中，可以实现不同数量的DRAM引擎1968，可以使用不同类型的存储设备而不是DDR3DRAM设备，或者可以实现其任意结合。此外，不同数量的链路可以被实现用于交换机接口1837、可以使用不同的物理接口技术而不是40GE，或者可以实现其任意结合。

[0216] 图20示出了根据本发明的一个或多个实施方式的闪存控制器1852。如所示，闪存控制器1852包括耦合至闪存控制器接口链路1847的接口模块2080。在一个实施方式中，闪存控制器接口链路1847包括耦合至与交换机1842关联的相应的端口的一个双向40GE链路。在其他实施方式中，闪存控制器1852包括两个或更多闪存控制器接口链路1847。闪存控制器1852进一步包括多个闪存信道子系统2082，每一个闪存信道子系统耦合至存储子系统，诸如存储子系统1854。包括每一个闪存新的子系统2082的闪存I/O控制器通过存储接口

1851耦合至相应的存储子系统1854。在一个实施方式中，每一个闪存信道子系统2082被配置为独立地在相关联的存储子系统1854上执行读取、写入和清除操作。这里参考闪存存储设备讨论了示例性的实施方式；然而，任何技术上可行类型的存储设备可以实现一个或多个存储子系统1854。虽然闪存存储器的特征为非易失性，但是在特定实施方式中，易失性存储器可以被用于实现存储子系统1854。

[0217] 本发明的实施方式在存储层级中实现了多层独立操作元件，该存储层级配置为提供极大的而可靠且可复原存储器，该存储器可以在多个处理器上共享，每一个处理器执行多个线程。每一层可以调度、列队、转发、完成和缓存命令和所关联的数据，因此大量待决的请求可以在整个系统内同时共存和进行，从而实现存储复合体1850内每一个可用存储子系统1854上的总横截面的带宽的高度利用。

[0218] 在所公开的系统中，数据独立于文件系统进行移动。本发明的实施方式在没有文件系统元数据改变的情况下使用页面虚拟化表格(PVT)使得页面移动。在大型媒介系统中，该表格常常是巨大的(如，对于用于3PB的媒介的64比特页面虚拟化表格，为6TB)。

[0219] 为了减少表格大小，表格项被共享。在一个实施方式中，基本量(如，64比特)被储存用于第一个项且偏移量(如，16比特偏移)被储存用于共享项。在该示例中，如果共享了16个页面的项，那么表格可以是三分之一大小。需要确保页面足够接近来使用偏置。应当避免数据结构的争用。最后，数据可以暂时被分配从而可以并行读取。

[0220] 前述可以通过使中央处理单元的一个核具有至一组页面表格项的排他接入来获得。该组页面表格项是非连续的从而随后的读取将仍然跨核分配。

[0221] 图21A示出了具有1太字节物理存储或存储器的设备的PVT，其中任何时候70％的设备(或700GB的数据)被映射。剩余的30％可以被用于可靠性和/或性能目的。利用该设备的软件理解其具有700GB的数据。

[0222] 对于第一次写入，数据页面最终在具有相同或类似索引的页面上。但是随着时间流逝，设备上的页面可能由于重写数据或设备管理考虑(如，闪存上的垃圾收集和擦除)而移动至该设备上的不同位置。图21B示出了在随后写入之后PVT看起来像什么。

[0223] 在本实施方式中，PVT是相当大的。特别地，PVT是设备大小的0.15％并储存在类似DRAM的更昂贵的存储器中。对于100TB来说这个大小是～150GB及对于1PB来说PVT大小是1.5TB。这对系统来说是一种昂贵的负担。因此，多个项被压缩在一起以共享PVT项。在一个实施方式中，基本量(如，完整设备索引)被储存以用于第一个页面并且接着偏置量被储存以用于共享PVT项中的页面。这将地址中的每一者从8字节压缩到2字节。该示例在16个页面之间共享项但可以推广用于更多页面。

[0224] 图21C描述了共享项。PVT表格具有基本量(4K页面0指定页面52)和偏置量(1:34328、2:27、3:1429、4:52890等)。图中示出了指向存储器中的页面52和在位置52942(这是页面52的偏置加上页面4的偏置52890)的页面4的基本量。

[0225] 该机制限制了共享PVT项中的页面位置，该位置离索引指标至多64K，因此偏移是16比特。这种限制转而将限制分配OS视作连续的页面的能力。页面的那种分配便于与其他公开技术相关联的优化。

[0226] 可以将相互远离的固定偏置的页面储存在每一个PVT项中。相应地，OS可以使用简单的、固定的算法计算哪个项表示OS索引方案中的特定页面。所修改的表格在图21D中显

示。特别地，图21D用偏置值代替了图21C显示的PVT中的页面参考。

[0227] 本发明的实施方式提供了无锁机制以用于在处理器之间进行通信。图22示出了具有两个处理器2200和2202的系统，这两个处理器被编程以使得一个传送命令而另一个接收和完成命令。系统还具有执行队列2204、完成队列2206和存储缓冲区完成比特2208。执行队列2204和完成队列2206二者是环形的。发送命令的处理器拥有执行队列2204的尾(tail)及接收命令的处理器拥有头(head)。在完成队列2206的情况下，反转所有权。存储比特用于不要求状态的完成。

[0228] 处理器1 2200将新命令施加在执行队列2204上，及处理器2 2202移除并执行命令。因为仅有一个生产商和一个消费者，所以锁定是不必要的。相反，在多核系统中，如果每一对处理器分配具有单个生产商和单个消费者专用队列是可以避免锁定的。应当注意处理器仍需要从其他处理器读取尾或头。

[0229] 存在两种涉及一个处理器等待另一方的情况：(1)处理器1等待直至在队列上存在空间或(2)处理器2发现没有项并必须在将来试着获得。处理器可以通过利用轮询系统或中断系统继续发信号。在另一情况下，在完成时，处理器2将项施加在队列上或者设置位掩码2208中的比特以注意事情已完成。处理器1将检查队列和位掩码。位掩码是潜在比队列上的项需要更少状态的优化并允许处理器1通过同时测试完成比特的整个字来同时检查多个完成比特。

[0230] 图23示出了现有技术系统，该系统具有支持操作系统2302的多个处理器核2300_1至2300_N。每一个核2300需要接入IO数据结构2304，该IO数据结构2304驻留在操作系统(OS)2302中。该单个IO数据结构2304可能使系统慢下来。同步机制导致扩展至硬件的串行化操作。硬件常常具有进入该硬件的单个管道，其也可能使操作慢下来。

[0231] 最后，当结果被返回时，常常通过硬件中断OS来完成，如块2306所显示的。软件可以选择以使中断转到一个或多个核2300。常常选择核的一个子集来最小化系统上IO操作的广泛影响。这可能使核接入同步所需的数据结构，最后返回至原始请求参与的核以返回结果至应用。

[0232] 本发明的实施方式利用能够编程和以多个区间分区的IO子系统。仅同步发生的位置来自/来到用户应用，如图24所显示的。如图所示，单独的处理器核2400具有单独的I/O数据结构2402、单独的中断路径2404和专用硬件资源2406。因此，每一个核在操作系统和硬件二者中具有专用数据结构。此外，向每一个核分配IO区间的行为具有多生产商单生产商数据结构(即，多个IO可以同时出现，因而多生产商、而仅核消耗IO)。该结构减少了同步影响。

[0233] 本发明包括可以在堆栈中的各个点共享一些数据结构的变化，但对其余部分对每一个核保持唯一数据结构。还包括轮询且不使用中断的系统。在使用轮询的情况下，每一个核具有其自己的(唯一的)数据结构。

[0234] 因此，该机制跨执行程序代码的CPU核和IO处理器对数据结构进行分区，其从硬件接口跨至共享并行储存设备进行操作从而通过复制所需要的数据结构和专用设备接口使每一个核支持储存设备的单个部分。这移除了同步单独的CPU核的需求，单独的CPU核通常采用操作系统软件锁来实现。其延伸至一般地向软件提供单个接口的硬件，从而降低效率和并行性。

[0235] 在多核、多线程系统中，便于对具有小误差容限的事件进行计数。这种计数可以被

用于决策性且动态的策略管理。图25示出了支持事件计数的系统。累加器的环形队列具有存储区，其中每一个存储区表示时间周期。在时间周期期间，生产商增加了相关联的累加器(可以忽视冲突，因而计数是近似的)。在时间周期+2期间，消费者将原始时间周期的累加器添加至消费者的总累加器(选择时间周期+2来避免冲突)。在新的时间周期，新的时间周期的累加器从总累加器减去并且新的时间周期被设置为0。消费者维持当前时间周期指针并进行自动更新。

[0236] 在本实施方式中，生厂商也是消费者。使用该累加器的一个示例是逼近IO系统中最后10MS上的读取数量，其中线程(或处理器或核)独立进行读取。该机制使得线程有助于总累加器而不使用锁。时间存储区累加器是近似的，但是因为单个线程用其更新总累加器，因此总累加器是一致的并且无锁。

[0237] 本发明的实施方式包括一机制，该机制允许活页面移动以准备在设备附近进行闪存块擦除，而不需要拷贝以暂时储存数据并然后在较慢总线上再拷贝至闪存另外的地方。在需要擦除存储的系统中，在将其第二次写入之前，仍有数据在使用或有数据是活的的页面必须移动至最近擦除的页面，从而系统可以再使用页面。相同逻辑页面的第二次写入一直出现在最近擦除的页面，使数据将不再再次被使用(死的页面)。除非系统进行“垃圾收集”，设备将用死的页面填满并不能接受任何新数据。

[0238] 包含闪存(FLASH)(闪存存储器)或其他媒介的需要垃圾收集的存储系统变得更加复杂，需要更精细的机制用于垃圾收集。在一个实施方式中，数据从用逻辑单元号(LUN)标识的多个闪存页面移动。LUN是闪存存储器的可单独编程部分。移动以允许读取活的页面的方式进行，其中活的页面正在并行地进行垃圾收集。在本实施方式中，还需要合并活的页面以将可以一次被写入的最大量的数据写入闪存。在现代系统中其可以是64KB或更大，然而之前提到的读取是4KB。因此我们从不同的LUN并行读取16个4KB页面并并行将64KB写入最近擦除的LUN。在本实施方式中，在并行性的复杂性之上，我们具有累加了64k校验的数据保护系统，同时来自收回的(垃圾收集的)页面的数据被储存在最近擦除的页面。

[0239] 图26示出了用于获得十六个不同LUN和相关联的校验值的三个不同的页面。现场可编程门阵列可以使用执行内核来对来自相同页面的LUN值执行逻辑XOR。对块中的所有页面重复操作，以迁移完成块群组。在该示例中，256个16KB页面被迁移。

[0240] 本发明的几个实施方式的区别在于所合并的缓冲区的位置和所计算的校验。在传统系统中，该操作常常在主CPU上执行，该主CPU还用于运行应用。这个方法存在两个问题。第一，其需要在多个总线上移动数据。第二，其需要可以用于其他目的(诸如用户应用)的CPU时间。

[0241] 图27描述了在现有技术系统中执行垃圾收集的的数据流。图中示出了CPU协调操作。本发明的实施方式将CPU工作卸载至另一资源，诸如存储控制器或闪存控制器。图28示出了基于闪存控制的系统，该系统消除了对中央处理单元的利用。也就是，诸如闪存控制器1852的闪存控制器用于提供用于RAID系统条带和校验管理硬件并行性。可以在所有条带成员都已经被写入之后写入校验。

[0242] 在需要擦除存储的系统中，在将其第二次之前，系统需要对仍具有正在使用数据的页面进行“垃圾收集”并将数据移动至最近擦除的页面。系统然后可以擦除和再使用垃圾收集的页面。相同逻辑页面的第二次写入一直存在于使原始数据不会再次被使用(死的页

面)的最近擦除的页面。除了系统收回之前已经用于数据存储但现在空闲的页面,设备将用死的页面填满并不能接受任何新数据。该页面收回的整个过程常常被称为垃圾收集。

[0243] 图29描述了垃圾收集的一个实施方式。当活的页面被垃圾收集(或收回)时,系统可能考虑一个接一个地恢复在以上图29的块0看到的页面,但是如果应用需要系统从块1读取数据,是不可能的,因为(除了在很少的情况下)仅一个页面(或页面群组)可以一次被读取,因而限制了在其他地方读取。这种同时从多个块读取页面的无能对于类似闪存的设备来说是常见的并被称为接入冲突。

[0244] 图30展示了冲突和解决方案。该机制在时间上对操作进行了重新排序。这种重新排序通过对垃圾收集推迟读取一定量时间(在本实施方式中列为1)来实现,以使得对闪存LUN的其他要求就可以满足,从而避免周期性的互斥等待。

[0245] 修剪是大多数操作系统支持的功能。其指的是告诉类似闪存的设备其使用数据页面已完成的文件系统或应用的行为。例如,这可能在页面已经被删除(“死的”页面)时出现。类似闪存的设备必须保持死的页面的尾,以使得在系统进行垃圾收集时,其不会迁移死的页面。对于系统来说,保持已经被重写的死的页面的尾是容易的,因为系统看到了写入,但是删除的页面需要修剪功能来通知设备,因为对正在删除的一些东西的知识本身不会引起设备看到的操作。

[0246] 修剪信息通常保持在一些运行时间数据结构中,该数据结构通过CPU或设备保持。还通常在系统的永久存储上进行记录和写入。因此,如果系统崩溃或在重新启动,那么系统可能具有连续的修剪尾以作为将来擦除的准备。

[0247] 本发明的实施方式为了减少所要求的记录修改了包括范围的修剪记录。在大型系统中,如果你删除了太字节文件,那么文件中每一个页面的单独记录可能共计为260百万项,每一项花费一个字节或更多。修剪范围可能将其减少为低数量的项——可能是个位数。

[0248] 在包括类似闪存的存储器的系统中,需要在写入之间进行块擦除,能够储存可以用于重构运行时间数据结构的元数据是非常重要的。许多系统在各种位置记录该数据。它们运作以使数据共同位于记录中。这存在许多问题。第一,在数据与元数据写入之间的长时间减少了关于重构的记录的准确性。另一个问题是记录空间的管理,这需要另一个设备或其自己的垃圾收集算法。

[0249] 本发明的实施方式需要储存元数据以及数据。本发明依赖于具有大的并行系统,以使得元数据可以在重构时进行并行读取。还利用写入元数据以及大量数据,以利用可以在单个设备上在同一时间写入高达64KB和跨系统的数百万字节的设备及跨数据页面(例如序列号)共享元数据。进一步地,元数据在支持数据保护的系统中可以容易地跨条带中的页面或跨条带共享。

[0250] 图31示出了类似于7+1的RAID5的擦除码。图中示出的保护条带可以表示多个数据页面(如,112)和多个校验页面(如,16)。这些页面中的一者或多者用于表示其他数据页面的元数据。元数据可以包括将块设备页面映射至闪存页面的页面虚拟化表格项。元数据可以是序列号以区分哪个元数据最后写入。因为闪存要求重写操作把最近擦除的页面当作目标并不在原始页面上进行写入,所以可以具有多个版本的相同页面。元数据可以是用于详述最近发现的不良块的不良块信息。元数据可以是循环冗余校验(CRC)信息以提供端对端数据保护。元数据还可以是修剪信息以表示哪个闪存页面已经被擦除。在各个实施方式中,

元数据的数量和种类可能不同、准确的存储位置可以改变等等。

[0251] 当页面重新写入类似闪存的媒介中时，相同的位置可能在没有第一次执行非常昂贵的擦除操作的情况下不会两次写入。在重写期间，数据页面的位置被虚拟化，以使得操作系统可能将设备视为一组连续的页面，即使这些页面没有连续的次序或没有在闪存设备中相应的位置。第一次写入可以采用图32所示的形式。因为这是第一次写入，虚拟化的数据页面可能在具有相同或类似的索引的真实的(物理的)页面。随着时间流逝，出于重写或更新数据或其他设备管理考虑(如，闪存上的垃圾收集和擦除)，一些页面被移动至该设备上的不同位置。图33显示了在典型集合的随后写入之后的PVT。

[0252] 如以上所讨论的，闪存块包含页面。在大多数闪存设备中，整个块必须擦除。页面不能被独立擦除。块极其构成页面不能被写入直至块被擦除。随着时间的过去，块必须被擦除以使其可再次写入。因此，一旦第一个页面被写入，将块写入闪存的第一个页面是不可能的，即使操作系统或用户应用可以继续将其标识为第一个页面。PVT实现这个处理。

[0253] 如果块转为不良并变得不可用或采用精细的损耗均衡算法，处理会变得更复杂。随着时间的逝去，损耗均衡移动块，以在一些周期期间创建均匀损耗并在其他周期期间(如，在闪存寿命周期的最后，所以我们不能一次耗尽所有闪存)创建不均匀损耗。

[0254] 本发明的实施方式使用两个等级的虚拟化算法。一个等级用于页面以及一个等级用于块，如图34所示。也就是，图34显示了PVT 3400、块虚拟化表格(BVT)3402和存储器3404。在经由PVT将虚拟页面地址映射至闪存的物理页面地址之后，标识了块的地址的一个或多个部分映射至使用块虚拟化表格(BVT)的地址。使用该技术，可以在系统的其余部分不必理解处理的情况下确定哪个块被使用，如可以在图35看到的。

[0255] 在这种情况下，块0是不良的并且其中的数据移动至块73。在一些实施方式中，替代块的选择被优化，从而通过挑选具有相同衰减设置的替代块来实现类似RAID的功能。

[0256] 大多数IO设备包括少量的子设备。这种模型存在问题。第一，这些设备常常通过条带化被分组在一起。每一个设备可并行编程但是它们常常具有少量发放连续的或小数量的IO操作的单个控制器。此外，这些系统有太少的设备有效。

[0257] 本发明的实施方式将大量独立可编程设备并入系统以加强基础设备从而使其能够进行并行可编程。图36显示了具有称为LUN的独立可编程设备的设备。所有的LUN可以利用于软件。在闪存控制器中具有复制的资源来实现并行数据接入。

[0258] 图37显示了40GbE通信协议，该协议规定了处理来自独立闪存控制器资源的容量。如图38所示，该分配一直持续在堆栈顶端。图38显示了通过请求的发起方向CPU一直进行复制的资源。

[0259] 实施方式包括使软件和终端用户应用能够开发并行性的可编程设备和复制的资源。初级结果是通过进行并行IO的大型并行操作和性能改善。二级结果是本发明使系统能够在连续操作和随机操作上执行相同动作。因为进行了如此多的并行操作，所以执行同样好的处理的连续或随机请求。

[0260] 特定计算(诸如之前所讨论的环读)在无锁的数据页面上需要逻辑XOR操作及来自条带的校验。对此通常的方法是使用CPU，但是这种方法增加了用于CPU的DRAM需求、存储总线带宽使用、缓存利用和CPU利用。本发明的实施方式将非CPU资源用于缓冲数据和校验页面及非CPU资源用于执行逻辑XOR。一旦页面被重构，CPU就可以在没有进一步工作的情况下

进行直接读取。

[0261] 图39示出了闪存控制器是断开CPU资源的实施方式。条带0是从RAID集合读取的一个页面。在读取值之后，逻辑XOR操作通过闪存控制器执行。得到的页面然后可用于CPU。

[0262] 在需要写入缓慢设备时在计算机系统中会出现问题。需要完成写入事物以使得应用可以继续并且不用等待缓慢设备。解决这个问题的方法是将数据拷贝至另一资源上的CPU特定的缓冲区，诸如FPGA、主存储器缓冲区或一些其他外部缓冲区。通过将其与之前进行写入分配的发明耦合，可以在进行存储拷贝的时候完成写入并且可以通过使用IO处理器避免CPU时间。

[0263] 图40示出了系统采用具有DRAM快速写入缓冲区的写入分配的本发明的实施方式。一般地，系统会在原始8MB IOP中保持拷贝数据直至数据稳定地在非易失性存储上进行写入。进行原始IOP的应用期望数据安全地将其进行存储并且仅副本置于IOP中。如果IO失败，那么数据就会丢失，除非原始缓冲区被保持。在本发明中，我们快速地(快超过2个数量级)在DRAM缓冲区中储存数据并信号发送IOP完成的应用。如果IOP后来失败了，那么系统使用DRAM缓冲区中的数据进行重试。

[0264] 为了防止电源故障，本系统必须包括不间断的电源。如果电源故障发生，不间断电源可用于允许数据从暂时的DRAM缓冲区拷贝放至非易失性存储的时间。该机制不能免受其他类型的失败，例如，操作系统故障或“崩溃”，除非DRAM也是非易失性存储器。即使有非易失性DRAM缓冲区，免受所有是故障类型也是不可能的。

[0265] 第一次写入涉及将应用的写入缓冲区中的数据在其正在被储存时移动至安全的地方。因此，应用可能考虑到写入完成并因而更快地继续。可能在存储硬件中没有地方储存该数据。该机制将其储存在主机的DRAM中直至数据确认在非易失性存储上。

[0266] 本实施方式在运行原始应用的CPU上使用DRAM并将原始IOP用于在DRAM缓冲区中储存数据。将CPU命令或使用能够直接存储接入(DMA)至原始缓冲区或拷贝缓冲区的IO设备来将来自原始缓冲区的数据拷贝至DRAM缓冲区。

[0267] 传统的操作系统通常有多种方式进行IO或从文件进行读取。最常见的机制是阻塞读取。也就是，应用调用操作系统和来自设备或文件(单个大小的单个数据集合)的希望储存在单个目的地的请求，并且应用一直等待直至操作系统完成操作。这常常称为同步IO。

[0268] 第二个机制常常称为异步IO。为了使用这个机制，应用发送请求列表至操作系统。每一个请求具有唯一的源、目的地和大小。应用没有等待结果，因此名字异步。操作系统为应用提供各种机制以检查请求的状态。应用报告完成的请求直至对所有请求作出解释。图41描述了Linux操作系统的各种实现中的该机制。

[0269] 在大多数操作系统中，与管理该列表相关联的开销非常大并且可能引起8MB阻塞请求的速度，与2048个4KB异步请求相比快很多。这个速度因为两个因素。第一个因素是管理异步请求的花费。第二个因素是操作系统不能利用能够进行大型并行处理的设备。此外，如果应用试着提交2048个4KB的阻塞请求，那么对于操作系统来说进行那些单个请求的开销会进一步放慢累积的IO操作。

[0270] 本发明的实施方式是用于处理多个同时的同步IO请求的机制，类似异步机制，但具有单独的阻塞读取的效率。这需要能够进行大量并行IO的设备和具有每个核数据结构的操作系统以避免锁定——二者之前已经公开。该特征称为多IO或MIO。图42描述了一个架

构。每一个核4200具有专用IO结构4202和中断路径4204。

[0271] 在N+1数据保护层中连续写入的读取写入比为3:1或更好以及在数据保护条带成员中的一者上存在写入的情况下，与通过等待写入相比，通过读取条带成员可以更快地重构读取。本发明尤其适用于写入比读取慢很多及读取可以并行进行的系统。本发明还包括对按比例下降至0:1的读取写入比的使用，从而在没有足够的读取来使用“环读”时增加将数据保护条带成员数据输出的写入并行性。

[0272] 图43示出了用于闪存的数据保护系统的一个实施方式。如果试着从LUN 0的块1中的闪存上的页面进行读取并且系统也正在写入至LUN 0，那么读取必须等待直至写入完成。本公开中早先时候具有对重建RAS系统的页面的公开，而不是等待写入完成。这是可信的，因为在许多设备中你可以a)比写入页面更快地多次读取页面及b)你可以从多个LUN并行读取。参照图43，可以非常快速地从LUN 1-3进行读取，并且XOR结果以得到所要求的页面。该操作比等待写入快很多。该技术称为“环读”。

[0273] 然而，对于环读是有效的，LUN 1-3必须是可用的。特别地，因为它们是具有LUN 0的数据保护条带的一部分，所以系统常常准备好同时写入LUN 0-3(小排序需要用于校验而写入耗费很长时间，很有可能LUN 3甚至可以重叠在其他LUN中进行写入，如果它们尽可能快地发出的话)。

[0274] 一个解决方案是对条带内的写入进行排序。换句话说，如果你准备针对LUN 0-3写入数据，那么你仅可以允许其中的一者在一段时间很忙。这确保了你可以进行环读而不会被另一写入堵塞。然而，这是效率低的并且大大限制了写入带宽(在这种情况下可能是其可能性的1/4)。

[0275] 考虑多个因素以确定在哪里对数据保护条带中的写入进行排序或不排序的自适应方案被使用。在一个实施方式中，该因素包括最近历史上(如，10ms)读取写入比和跨系统的IO利用。如果读取写入比是3比1或更大，那么更可能存在写入所堵塞的读取并且具有该特征是重要的。如果IO利用上升(假设大于80％)，可能希望限制环读，因为其由于其读取了用于在环读期间执行的每一个读取的3个页面而增加了整个IO带宽利用。本实施方式还包括在读取写入比小于3比1并大于0比1及IO利用小于80％时对序列化写入百分比进行按比例分配以支持环读。

[0276] 关于N+1数据保护，如果保持数据的单元为忙，可以通过环读忙的单元(读取N-1数据页面和一个校验页面)来重建数据以满足读取请求。如果1/(N+1)个单元或更少的单元当前忙，那么可以确保所有数据可以通过均匀地分配写入进行环读。为了满足需要1/(N+1)以上的单元是忙的写入负载，将一个写入分派至每一个PG，然后分配总共N+1个写入至足够的单元以满足写入负载，从而留下尽可能多的PG具有仅一个写入。这最大化了环读可能的量。

[0277] 在外围组件互联表达(PCIe)线路的数量在所有套接口上不同的系统中，不同的套接口中的PCIe卡可能不处理相同大小的负载。否则，总系统带宽被可用于具有最少数量的PCIe线路的套接口中的PCIe带宽制约。该机制基于可用PCIe带宽调度闪存控制器上的IO，并使用QPI将到达PCIe总线的数据传递至用户缓冲区驻留的套接口。图44中的系统展示了PCIe带宽失衡。

[0278] 在该系统中具有4个套接口，每一个套接口附着至变化数量的PCIe线路，所以套接口1和套接口3具有10GB/s的PCIe带宽，而套接口0具有15GB/s，及套接口2具有5GB/s。每一

个PCIe总线具有所附着的多个闪存控制器，该多个闪存控制器能够在PCIe总线上发起DMA操作。假定存在待决操作，该操作由CPU密集的部分和在PCIe总线上传递数据的IO密集部分构成。出于多个原因需要在所有4个套接口上调度相同数量的操作。第一，可能存在一般的系统宽度调度算法，该算法对核进行负载均衡。第二，每一个套接口具有相同数量的核；所以为了在核之间均衡CPU密集部分的工作，将相同数量的操作分派给每一个核是有意义的。否则可能产生CPU瓶颈，并同时可能仍可能留下CPU资源。然而，平衡CPU密集部分的操作与IO密集部分的操作冲突，因为PCIe带宽在跨套接口上没有平衡。

[0279] 如果IO在4个套接口上平均调度并且如果每一个核被分配相同数量的操作去完成，那么整个系统可能仅完成5×4＝20GB/s的IO而全部系统能够达到(15+10+10+5)＝40GB/s。一种绕过这个问题的方式是使每一个核处理同等量的CPU密集部分的操作，但是在远程套接口上的闪存控制器上对核发起IO密集部分的操作，以使得可以使用每一个PCIe总线的全部带宽。远程套接口上的IO需要移动至经由QPI总线发起IO操作的CPU本地的套接口，但是这平衡了CPU密集部分的操作以及IO密集部分的工作。在本特定的示例中，在套接口1和3中处理操作的核不需要从远程套接口发起任何IO，因为其具有IO带宽，该IO带宽确切地是每一个套接口需要的平均带宽。当操作到达套接口2时，反而需要在套接口0的闪存模块上发起50％的IO请求，因为套接口2的PCIe总线相对每一个套接口所需要的平均在预分配以下的50％(5GB/s相对10GB/s)，而套接口0相对平均在预分配以上的50％(15GB/s相对10GB/s)。最后结果是5GB/s在QPI总线4400上从套接口0传递至套接口2，但是可以使用系统的40GB/s的PCIe带宽，并且所有CPU核将进行负载均衡并处理相同量的CPU密集部分的操作。

[0280] 在允许多版本的页面出现在储存(如，闪存)的系统中，序列号记录了版本创建的次序。因而，最近的拷贝将具有最大的序列号。在大型系统中，这些序列号的管理可能是过于繁重的。例如，划分为4KB的页面的70TB设备需要150GB的存储器来保持所有其页面的序列号。如果设备利用擦除码形成7个数据符号(块)和单个校验符号(块)的条带，称为7+1编码，那么用于4平面16KB闪存页面的序列号所需要的存储器可以共享整个条带的序列号，并将所需要的存储器的大小减小至～1.3GB。

[0281] 图45示出了一般如何使用序列号。在系统崩溃时使用序列号来重新创建页面虚拟化表格(PVT)。PVT将OS页面索引映射至设备上的页面索引。因为在由类似闪存的固态存储器构成的设备上的页面可能由于更新或随后的垃圾收集而进行移动，储存在PVT中的位置将随时间而改变。在崩溃的时候，页面的多个拷贝可能存在于设备上。OS将最后写入的版本选为实际数据。OS通过比较序列号进行这个操作。在本实施方式中，序列号在系统上是唯一的。序列号在接入时以原子的方式增加。因而，每一个使用是唯一的，从而保证所写入的页面准确地用标识了哪个页面先写入的序列号标记，即使多个拷贝在时间上相互非常接近地写入。

[0282] 本发明仅使用用于每一个数据保护条带的一个序列号。因为整个条带同时写入，所以仅需要一个号表示序列号。图46显示了RAS条带。在本实施方式中，条带的成员为可以一次写入闪存LUN的最大项的大小：64KB，为16个4k OS页面。因而，整个条带表示112个OS页面，对该OS页面使用一个序列号。所以，总节省超过2个数量级。本发明要求对在时间上非常接近地出现的相同设备页面的写入必须用不同的序列号写入条带。因为这不会非常经常发

生，所以该机制的一个实施方式可以对在时间上非常接近地出现在相同页面的写入进行串行化，从而确保其具有增加的序列号。

[0283] 包括闪存设备的设备的可恢复性需求随时间而改变。设备在其使用的开始需要较少的数据保护，因为其失败常常较少。因而，类似7+1RAID的数据保护方案在开始是足够的，而10+2或3+1数据保护方案将例如在寿命周期的75％需要。

[0284] 在数据保护系统的一个实施方式中，系统支持具有7+1(每一个条带7个数据和一个校验码)的类似RAID5的数据保护。该方案使用简单的在所有数据页面上使用XOR计算的擦除码以创建单个校验页面。图47显示了此类方案。

[0285] 图48示出了集成的垃圾收集和数据保护。垃圾从多个LUN同时收集。活的页面写入新的数据保护条带。每一个LUN当作数据的环形列表。写入针对最近的擦除块4800。数据从写入块4802进行检索及垃圾从最老的写入块4804进行收集和擦除。

[0286] 在需要类似3+1的更保守的数据保护方案时，将块标记为参与3+1方案。图49描述了方案与通过方案进行的块标识之间的转换。本实施方式通过使更保守的方案(3+1)成为较不保守的方案(7+1)的因素而使事情变得容易。本发明还支持更复杂的结合。

[0287] 在合并了闪存控制器(或需要垃圾收集的其他媒介)上的数据保护的传统系统中，数据保护常常在闪存系统的顶部建立，但是使独立的。当系统不得不进行垃圾收集时，垃圾收集的活的数据(准备闪存擦除而必须移动的数据)必须一直转到数据保护驱动器以维持校正校验或你必须维持死的页面(通过用户重写的页面)以维持比需要的更长的条带。

[0288] 这个问题在系统包含最有效写入多平面闪存页面的闪存时更加复杂，该闪存页面比系统的自然页面大小(如CPU指定的)更大。当数据保护系统写入条带成员时，其不得不写入例如16CPU大小的页面以用于每一个成员。在类似RAID的5+1中，每一个条带可能是484KB的数据+64K的校验(参见图29)。

[0289] 在闪存页面可以再次写入之前，闪存的工作是擦除整个闪存页面块。如果页面写入块并且有必要再次写入该页面，那么必须在闪存的其他地方写入。已经重写的页面称为“死的”及还没有重写的页面称为“活的”。在某一时刻，闪存设备将充满包含活的和死的数据的页面的块，并且没有或有很少未写的块。死的页面是浪费的空间并且必须收集。通过将活的页面从块写入新的位置并擦除封闭的块来执行收集，从而使其可再次写入。

[0290] 一个方法是以循环的方式使用块。在未写的块的数量很小时，回收所使用的块从而其准备再次写入。用所采用的数据保护，系统还必须重建如图48所示的校验，其中块4804正在回收(垃圾收集)。

[0291] 实施方式承担集成重建数据保护校验和对垃圾收集处理加条纹及对硬件使用进行优化。图26显示了如何同时移动页面以在垃圾收集期间建立新的raid条带。因为我们知道如何同时从不同的LUN读取活的页面，所以我们可以实际上从多个块同时进行垃圾收集并然后具有同时写入如何工作的知识，我们建立足够的活的页面来进行对新位置的多平面16k页面写入。

[0292] 我们还确保我们通过规定接近闪存的硬件而不会过度地使用CPU来进行这个工作，该硬件可以执行之前提到的命令并缓冲如图28所示的数据。命令和硬件的这种结合比在其他系统中更紧密地集成了垃圾收集和数据保护，并提供了唯一的性能改善等级。

[0293] 如以上所描述的，具有类似闪存的存储器的系统需要进行垃圾收集以重新使用存

储器。本发明是利用充足的供应及自适应地平衡垃圾收集操作和用户发起的操作的结合进行连续垃圾收集的机制。其他系统常常具有垃圾收集循环并在那些循环期间显示性能问题。本发明有利于统一的最大化性能。

[0294] 图50示出了规定用于垃圾收集以使其能够在标准操作期间出现的系统。特别地，在本实施方式中，两组资源相结合以确保适当处理明显的窥视(peek)。本系统可以处理70GB/s的写入。写入性能由从服务器框5000到闪存框5002的网络带宽限制在双向120GB/s。如果我们假定最糟糕的情况，垃圾收集器进行80GB/s的写入(每个用户写入是2个垃圾收集写入)，那么用户写入会限制在40GB/s。本实施方式在控制器卡上还具有DRAM缓冲以缓冲窥视写入。缓冲带来瞬间高达70GB/s的限制。本实施方式使用系统IO容量和溢出处理的结合来处理仅写入工作量。

[0295] 相同的实施方式还处理读取和写入的结合，其中CPU上的软件堆栈适于支持读取。为了从系统得到最大化的带宽，CPU上的软件将调度读取和写入以最大化带宽。

[0296] 图51示出了一个实施方式。在本实施方式中，用户应用5100作出读取和写入请求，该请求通过分开的队列进行处理；即，读取队列5102和写入队列5104。系统产生垃圾收集必要的读取、写入和擦除并将其放置在队列5106中。队列5104中的项越多，在队列5106中需要的项越多。系统可以平衡和调度来自所有三个队列的请求并确保在没有渴望写入(和随后的垃圾收集操作)的情况下尽快完成读取。写入和垃圾收集队列变得越满，它们接收的优先级越高。这显示了以连续的基础自适应处理垃圾收集的一个实施方式。

[0297] 在多个读取或写入发生在接近特定单元的单元中时闪存存储器遭受比特错误率(BER)的显著上升。这些减少错误的“读取干扰”或“写入干扰”可以通过改变原始数据来减少，以这种方式使将写入的字符串一或字符串零的出现最小化。完成这种转变的一种方式是通过将合适的伪随机生成器用于转变数据。

[0298] 本发明使用每一个闪存页面(每一者处于不同的偏置)的多个伪随机序列。这允许单独的数据扇区在不需要读取整个闪存页面的情况下被读取。现有的实现通过解码器读取整个闪存页面。该技术允许我们仅读取我们需要的部分数据并因而允许较少的传输及由此带来的改善的读取时间。

[0299] 实施方式对每一个储存的闪存页面(或其他可接入数据单元)利用多个(并行)随机化流。种子值可以用于从整个16k闪存页面产生伪随机流字节。在一个实施方式中，已知的密钥撒布伪随机数产生器，该产生器在子页面数据单元上生成随机流字节，在一个实施方式中是闪存页面的4k部分。图52示出了具有不同部分5202、5204、5206和5208的闪存页面5200。每一个部分具有唯一的页面随机化。

[0300] 另一实施方式是至闪存存储设备的接口，该接口向存储控制器及其互连协议提供所需要的接口。其具有两个主要接口。在输入，具有读取(READ)、写入(WRITE)和擦除(ERASE)命令。在输出，具有实际的闪存设备信号。

[0301] 将输入与输出连接的机制是处理子命令的引擎，该子命令是输入命令的构成部分：开始-RD、开始-WR、得到-状态(Get-STATUS)等。闪存控制器具有两层(1)将输入命令转换为子命令的每一个LUN的状态机及(2)具有将子命令转换为闪存设备信号的嵌入的微码机的状态机。

[0302] 对LUN控制器的输入命令来自经由存储根或存储控制器的计算应用CPU，例如

Intel®Xeon®处理器。在一个实施方式中，输入命令为：

[0303] 1.擦除-2：立即擦除在2个平面上的块

[0304] 2.写入-1：写入单独8k(或16k)页面

[0305] 3.写入-2：写入2个页面，2个平面的每一者上一个

[0306] 4.读取：读取4k部分的闪存页面

[0307] 每一个输入命令划分为如图54所示的子命令。在该机制中，等待准备好(READY)是围绕得到状态(Get STATUS)的循环，用伪码表示为：

[0308] Repeat

[0309] Get STATUS

[0310] Until READY bit is set

[0311] 在本发明的一个实施方式中，具有7个子命令。最后二者仅用于系统初始化。

[0312] 1.开始擦除

[0313] 2.得到状态

[0314] 3.开始写入

[0315] 4.开始读取

[0316] 5.完成读取

[0317] 6.重置

[0318] 7.得到特征

[0319] 每一个子命令转而由闪存设备命令组成，该闪存设备特别针对所选择的闪存设备。图55提供了示例。

[0320] 在一个特定闪存设备的情况下，存在6个闪存命令。(最后二者很少使用)。

[0321] 1.CMD

[0322] 2.ADDR

[0323] 3.Din

[0324] 4.Dout

[0325] 5.Idle

[0326] 6.Standby

[0327] 在一个实施方式中，处理这些命令的微码引擎具有如图56所示的结构。该微码引擎以所接收的次序一次一个处理闪存命令。每一个子命令为基于存储器的程序提供“开始”地址，其中该程序驱动至闪存设备的闪存命令接口。闪存命令由指令序列产生，并且记录存储器的输出直至完成。引擎具有四个基本的控制功能：

[0328] 1.控制闪存设备命令的序列

[0329] 2.对闪存设备的数据输入控制输入字节的选择，即在何时选择哪个寻址字节

[0330] 3.基于微控制器主时钟(200MHz)逐步地控制闪存定时

[0331] 4.控制重复命令，即4k Din循环

[0332] 常常IO控制器使用专有的或层级的IO定向通信机制来在CPU与较慢的设备之间通信。这存在的问题是a)常常是不灵活的或b)可扩展。可替换地，本发明使用标准通信网络将CPU连接至设备。图50示出了本发明的一个实施方式。在本实施方式中，我们从PCIe转换至40GbE。一旦我们处于40GbE网络，任何组件都可以与任何组件通话。存在与该方法相关的多

个益处。在具有大量设备的系统中，可以充分地分配工作和数据，从而保证并行操作。如果增加更多的设备，可以扩展网络。具有故障切换(fail-over)通信路径是容易的。

[0333] 为了提供高等级的容错，常常利用冗余组件和互连路径。本发明提供了改善的冗余数据传递路径。在一些实施方式中，闪存卡上的主要业务路径使用两个以太网，每一个以太网连接至FPGA。不是仅提供了从FPGA至交换机的辅助以太网链接，这可能需要总共4个以太网链路，而是我们将FPGA连接在一起并在至其他FPGA的链接上提供了辅助的、备用的路径，如图57所示。利用该辅助链路将使系统用退化的业务容量进行操作，但是将在不需要辅助以太网接口的情况下提供冗余路径。不是将闪存控制器FPGA连接至两个以太网端口，如块5700所示，而是“其他”FPGA以太网端口连接作为冗余路径，如块5702所示。这节约了系统和板资源并提供了相同等级的冗余。

[0334] 本发明的实施方式涉及具有永久性计算机可读存储介质的计算机存储产品，在永久性计算机可读存储介质上具有用于执行各种计算机实现的操作计算机码。媒介和计算机码可以是特别设计的并出于本发明的目的进行构造，或者可以是已知的类型并可用于具有计算机软件领域的技术的码。计算机可读介质的示例包括但不限于：磁介质、光介质、磁光介质和尤其配置为储存和执行编程码的硬件设备，诸如应用特定的集成电路(“ASIC”)、可编程逻辑设备(“PLD”)和ROM与RAM设备。计算机码的示例包括诸如由编译器产生的机器码及包含较高等级码的文件，该文件由计算机使用解译器执行。例如，本发明的实施方式可以使用JAVA®、C++、或其他面向对象编程语言和开发工具实现。本发明的另一实施方式可以在硬线电路中实现，该硬线电路代替机器可执行软件指令或与机器可执行软件指令结合。

[0335] 以上描述出于解释的目的使用特定术语提供了对本发明的彻底理解。然而，对本领域技术人员显而易见的是实行本发明不需要具体细节。因而，本发明特定实施方式的以上描述出于示出和描述的目的显示。并不意图详尽或将限制本发明为所公开的精确形式；显然，许多修改和改变考虑到以上教示是可能的。实施方式被选择和描述是为了最好地解释本发明的原理及其实际应用，其因而使本领域技术人员能够最好地利用具有各种修改的本发明和各个实施方式以适于预期的特定使用。应当意识到的是所附权利要求及其等效限定了本发明的范围。

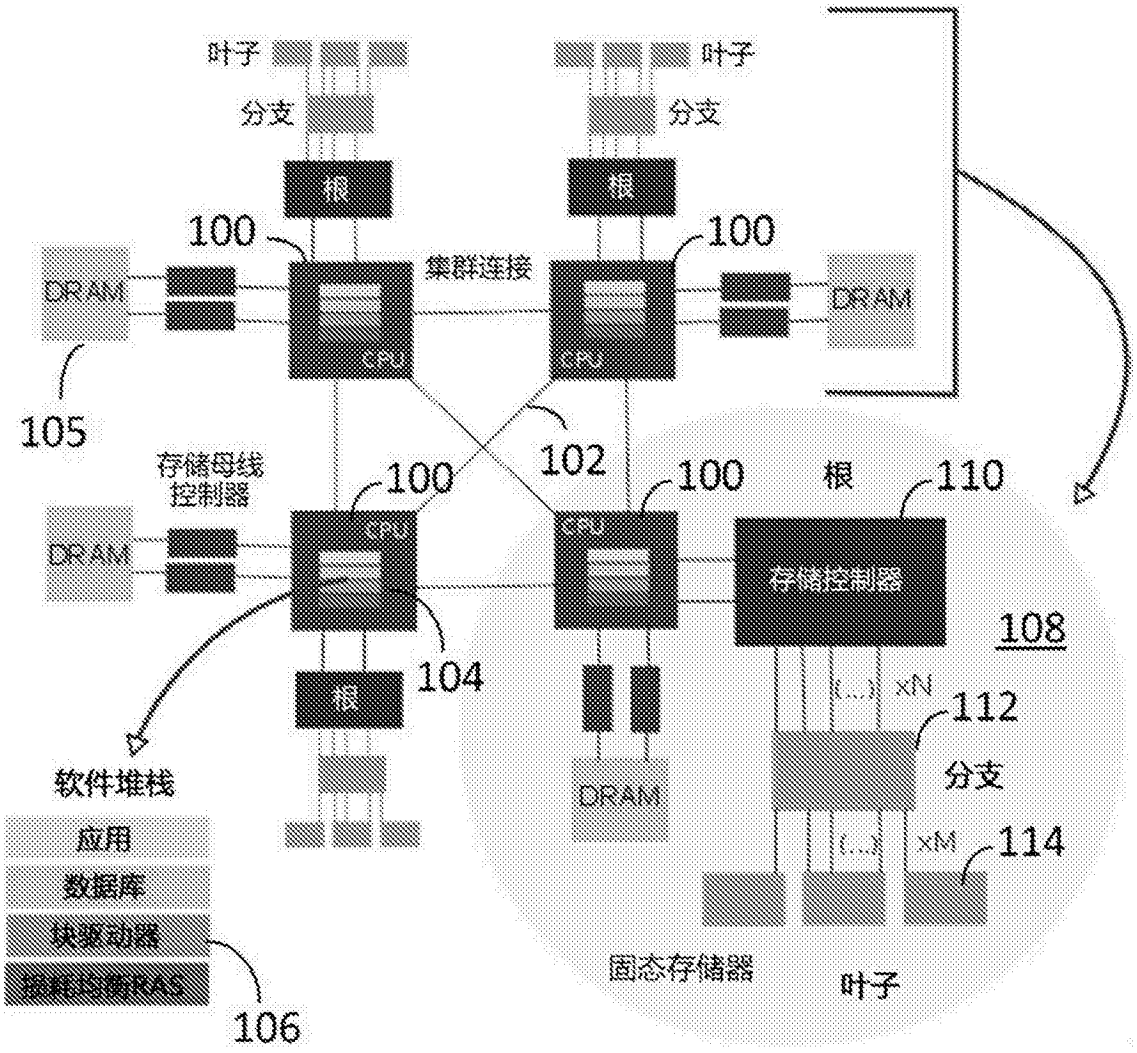

叶子
分支
根
100
DRAM
105
CPU
集群连接
102
存储母线
控制器
104
软件堆栈
应用
数据库
块驱动器
106
xN
112
xM
114
存储控制器
110
108
固态存储器
叶子

图1

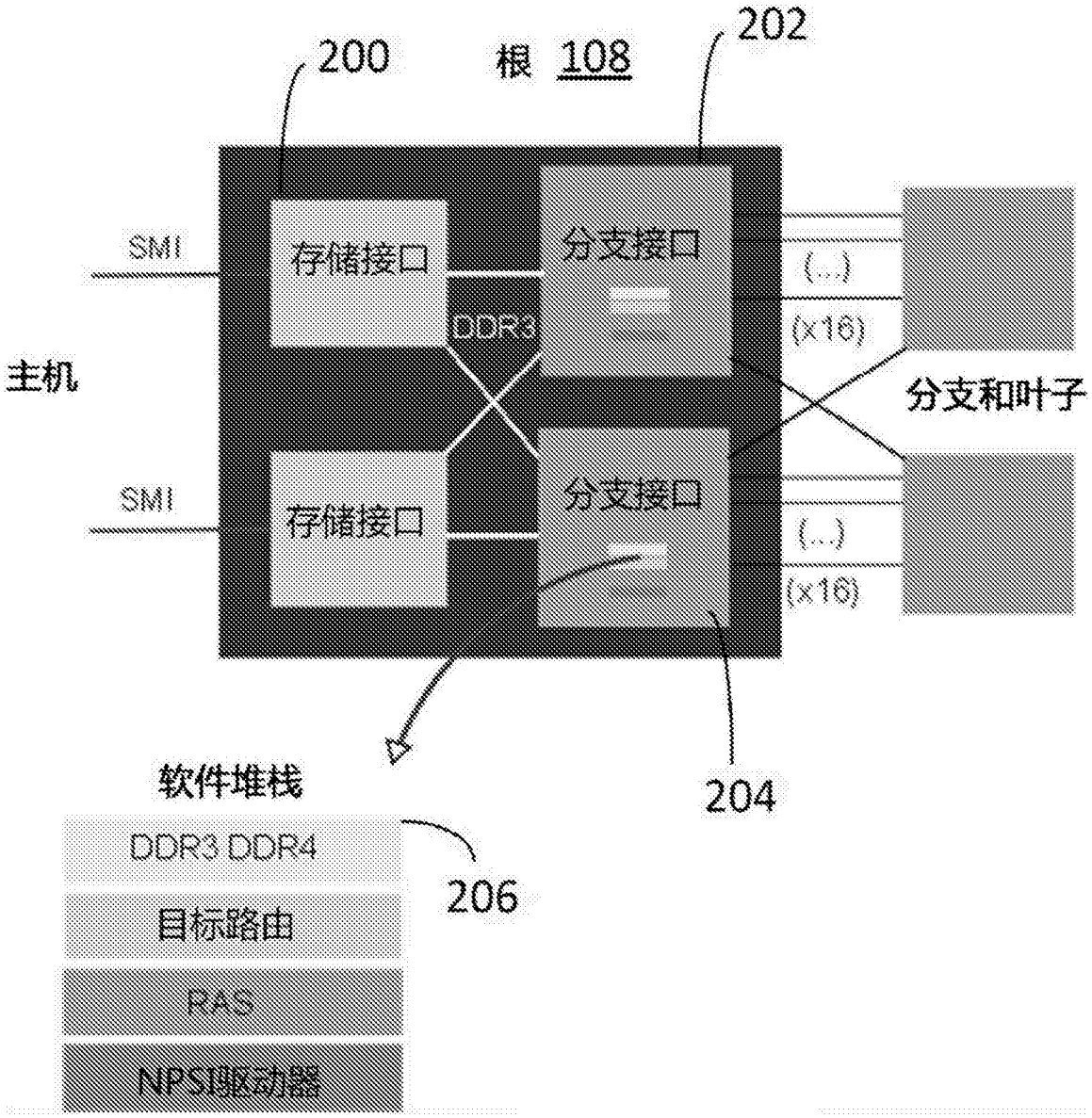

200
根 108
202
SMI
存储接口
分支接口
(...)
(x16)
DDR4
主机
分支和叶子
SMI
存储接口
分支接口
(...)
(x16)
软件堆栈
204
DDR3 DDR4
206
目标路由
RAS
NPSI驱动器

图2

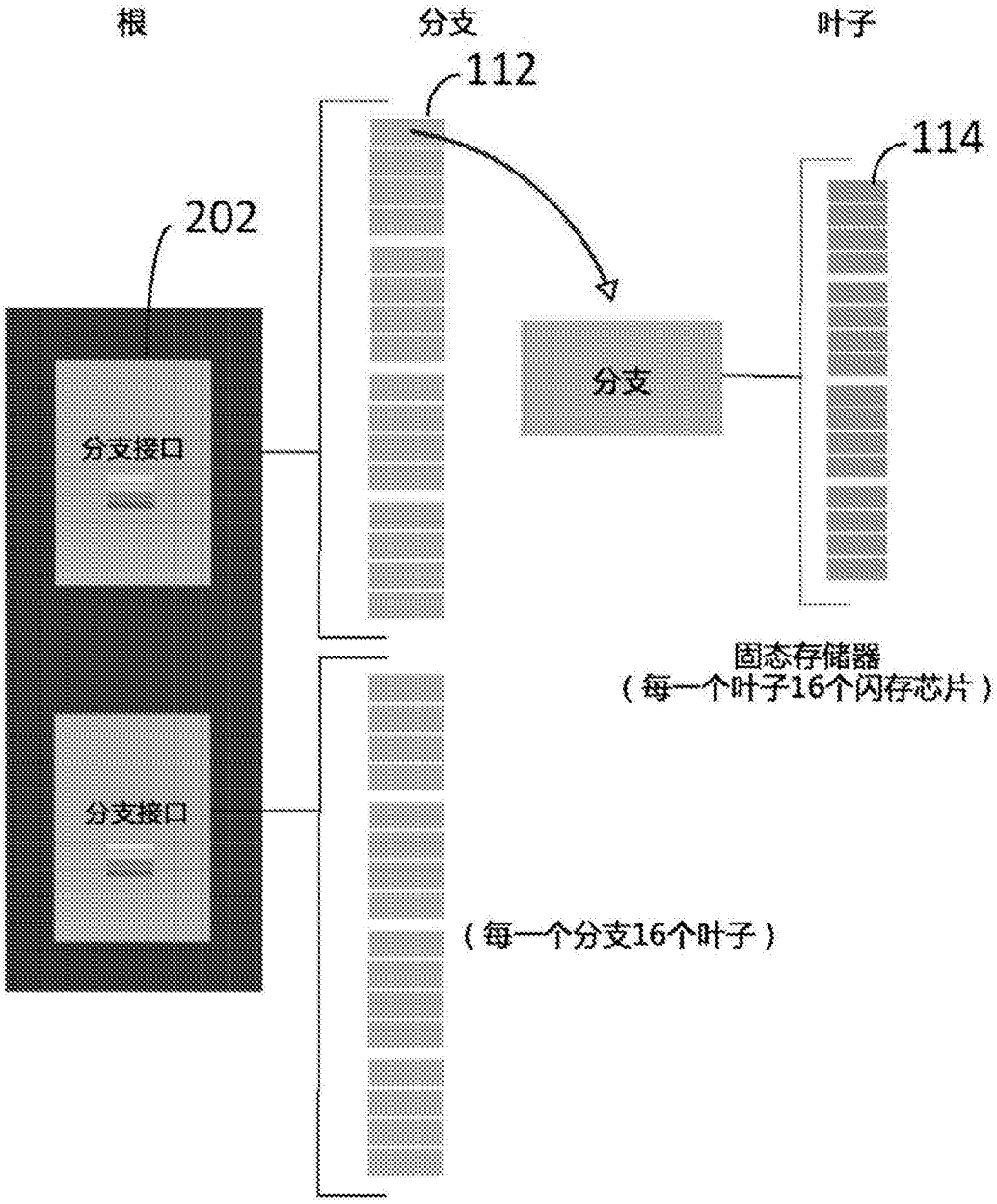

根
分支
叶子
112
114
202
分支
分支接口
分支接口
固态存储器
（每一个叶子16个闪存芯片）
（每一个分支16个叶子）

图3

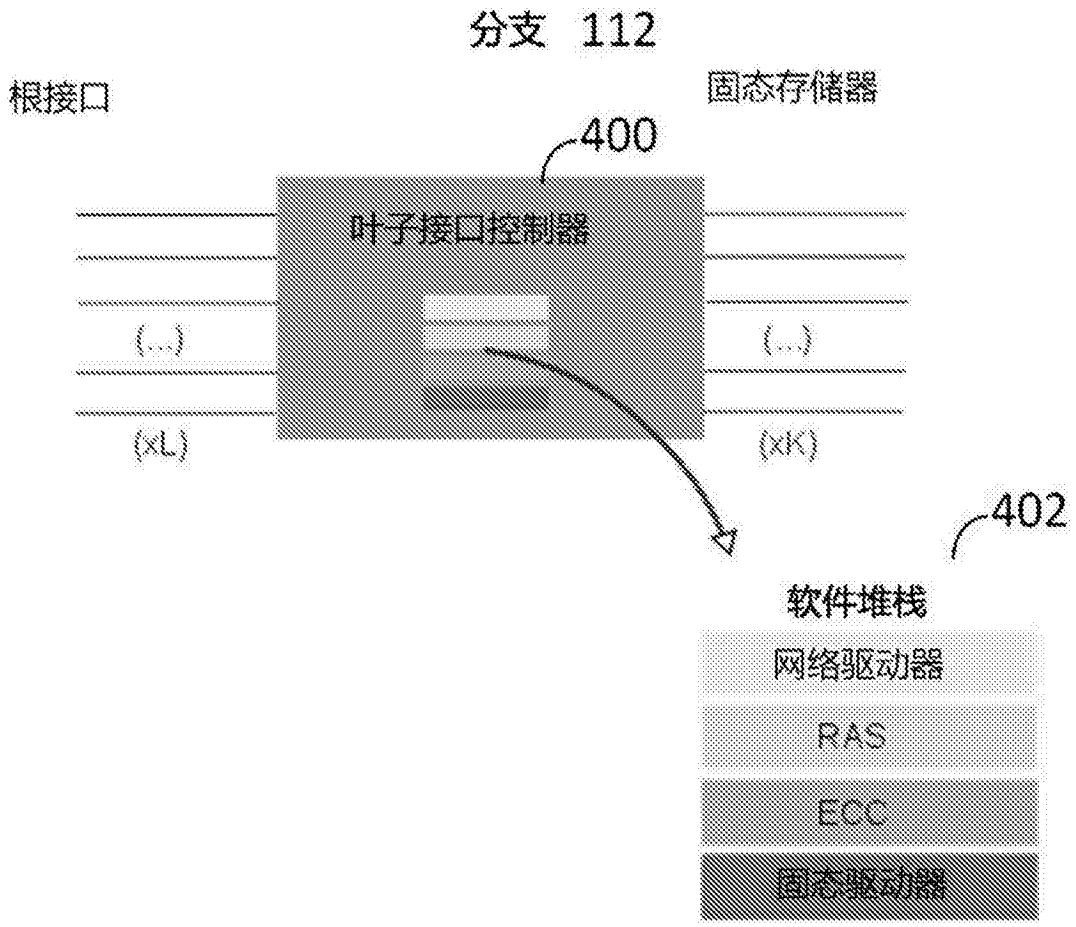


图4

$$P_i = \prod_{n=i}^{i+3} D_n \qquad D_i = P_i \otimes \prod_{n=i+1}^{i+3} D_n$$

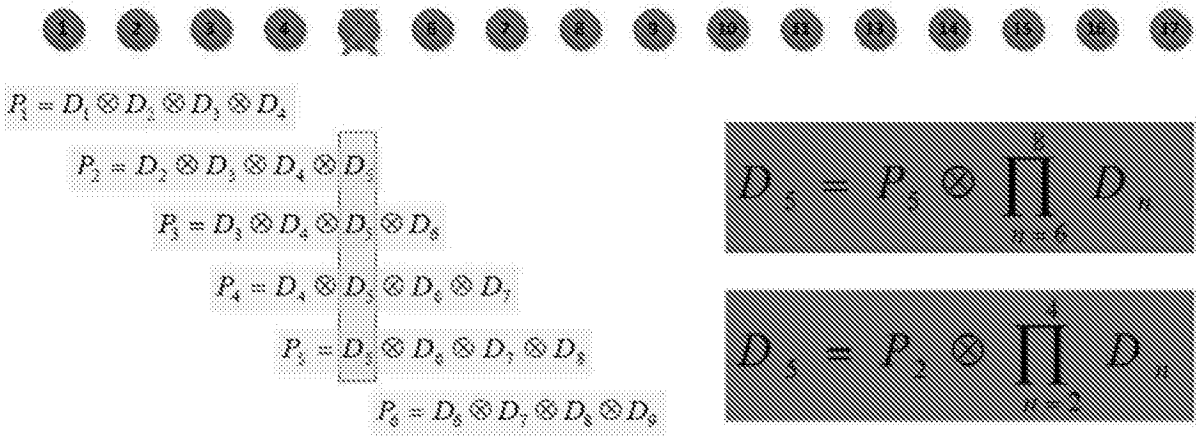


图5

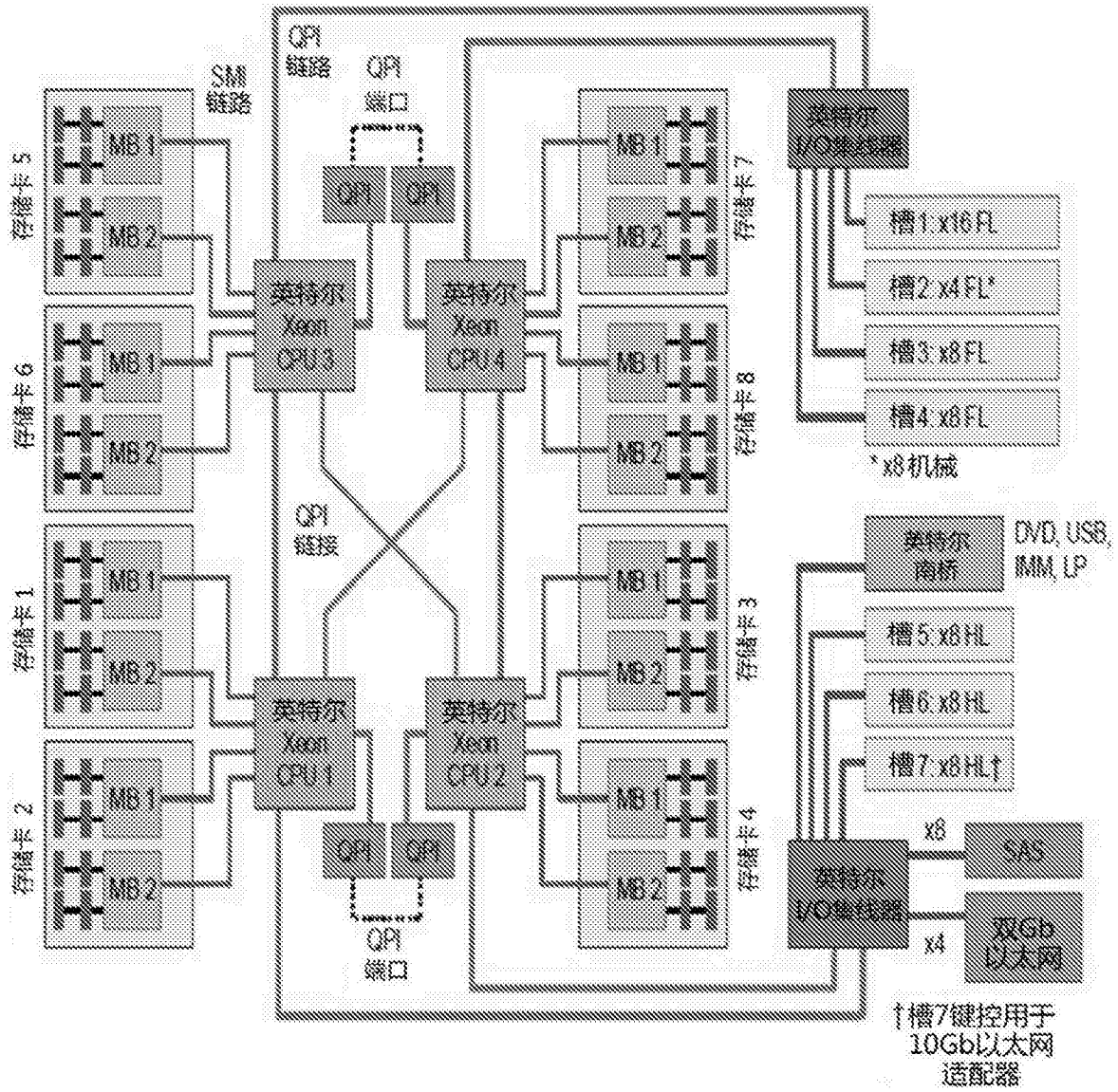

QPI
链路
SM
链路
QPI
端口
存储卡 5
存储卡 6
存储卡 1
存储卡 2
存储卡 7
存储卡 8
存储卡 3
存储卡 4
MB 1
MB 2
英特尔
Xeon
CPU 3
英特尔
Xeon
CPU 4
英特尔
Xeon
CPU 1
英特尔
Xeon
CPU 2
QPI
链接
英特尔
I/O集线器
槽1 x16 FL
槽2 x4 FL*
槽3 x8 FL
槽4 x8 FL
*x8机械
英特尔
南桥
DVD, USB,
IMM, LP
槽5 x8 HL
槽6 x8 HL
槽7 x8 HL†
英特尔
I/O集线器
x8
SAS
x4
双Gb
以太网
†槽7键控用于
10Gb以太网
适配器

图6

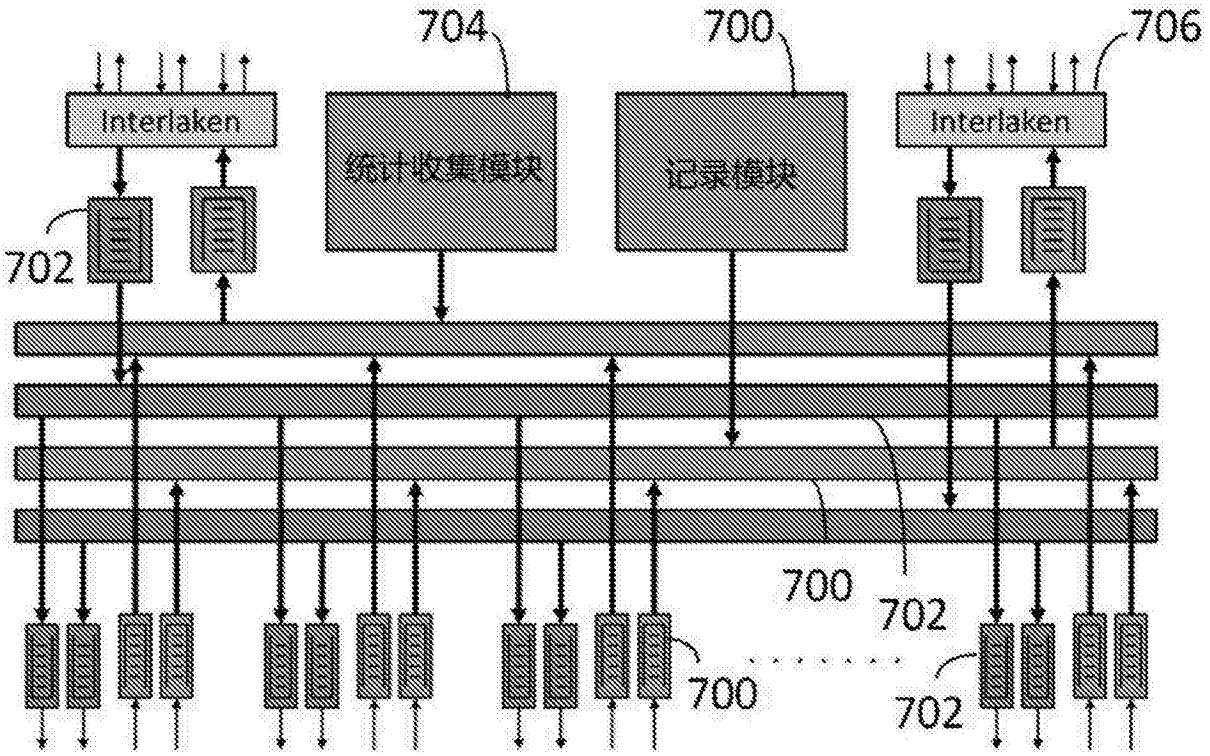

704
700
706
Interlaken
统计收集模块
记录模块
Interlaken
702
700
702
700
702

图7

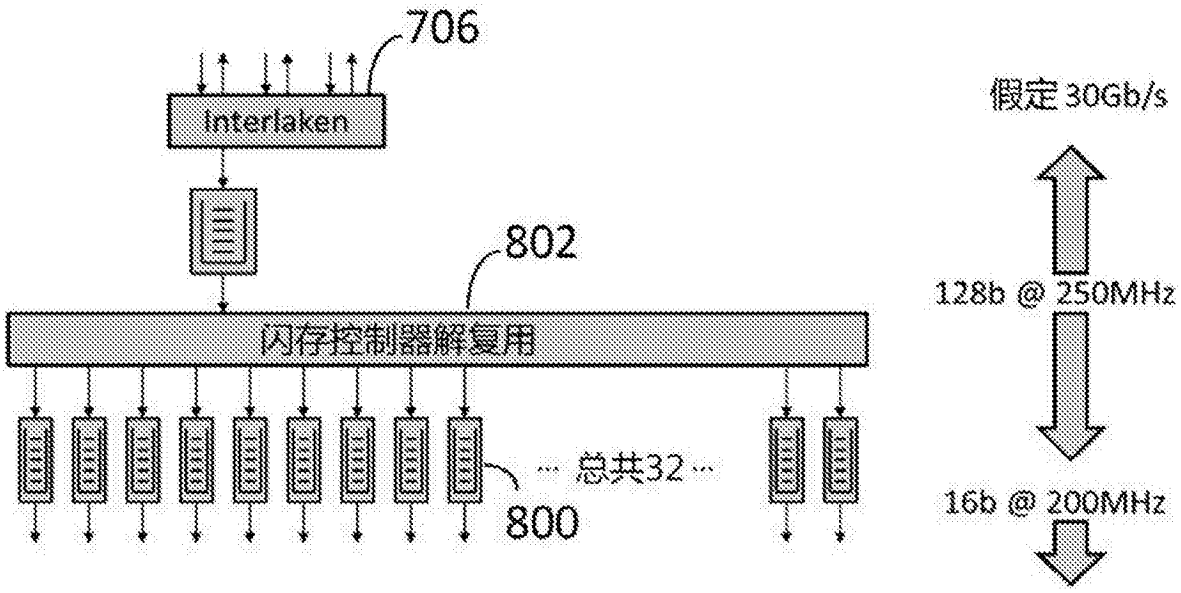

706
Interlaken
802
闪存控制器解复用
… 总共32 …
800
假定30Gb/s
128b @ 250MHz
16b @ 200MHz

图8

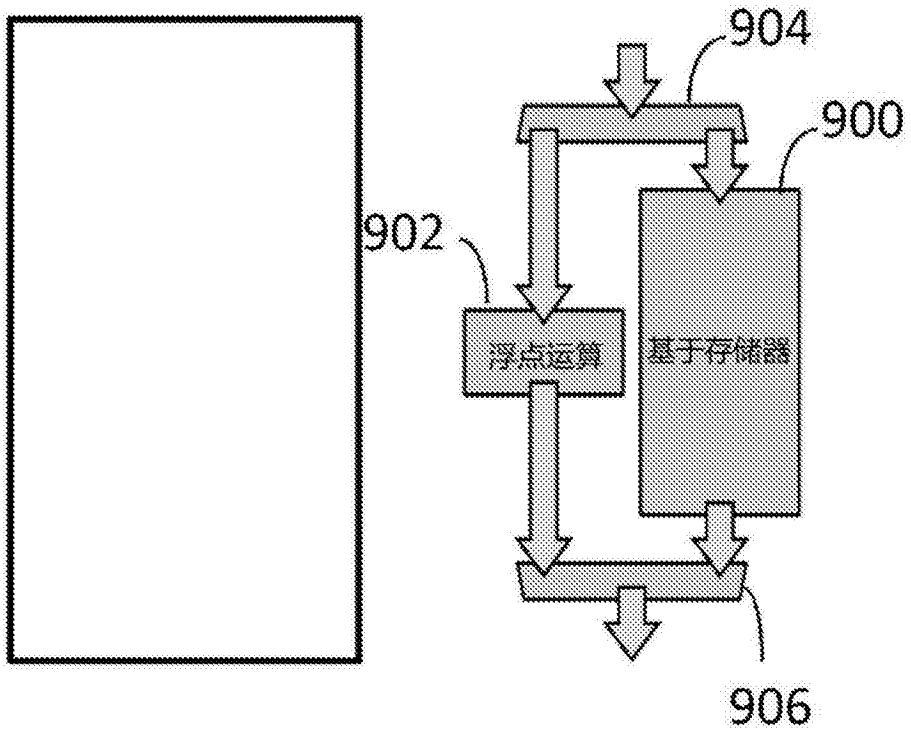

904
900
902
浮点运算
基于存储器
906

图9

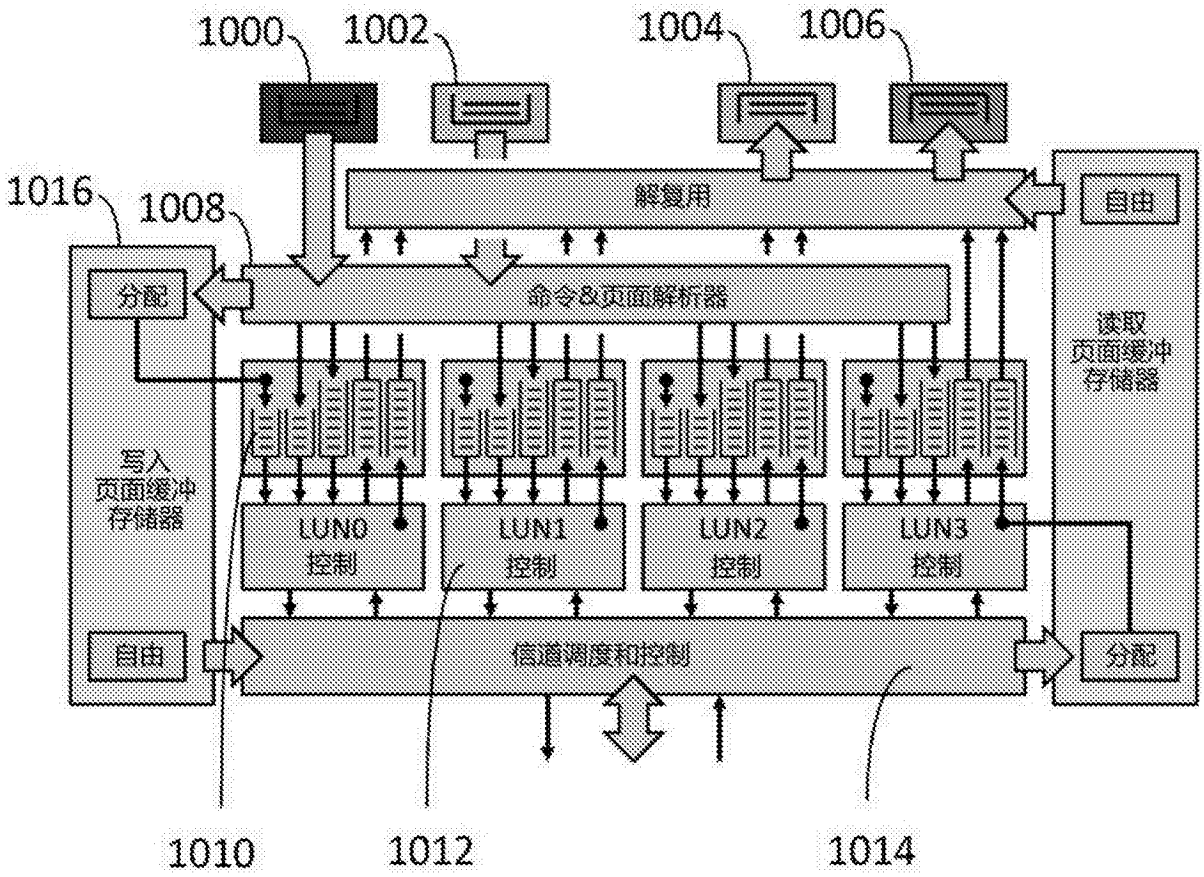

1000
1002
1004
1006
1016
1008
解复用
自由
分配
命令&页面解析器
读取
页面缓冲
存储器
写入
页面缓冲
存储器
LUN0
控制
LUN1
控制
LUN2
控制
LUN3
控制
自由
信道调度和控制
分配
1010
1012
1014

图10

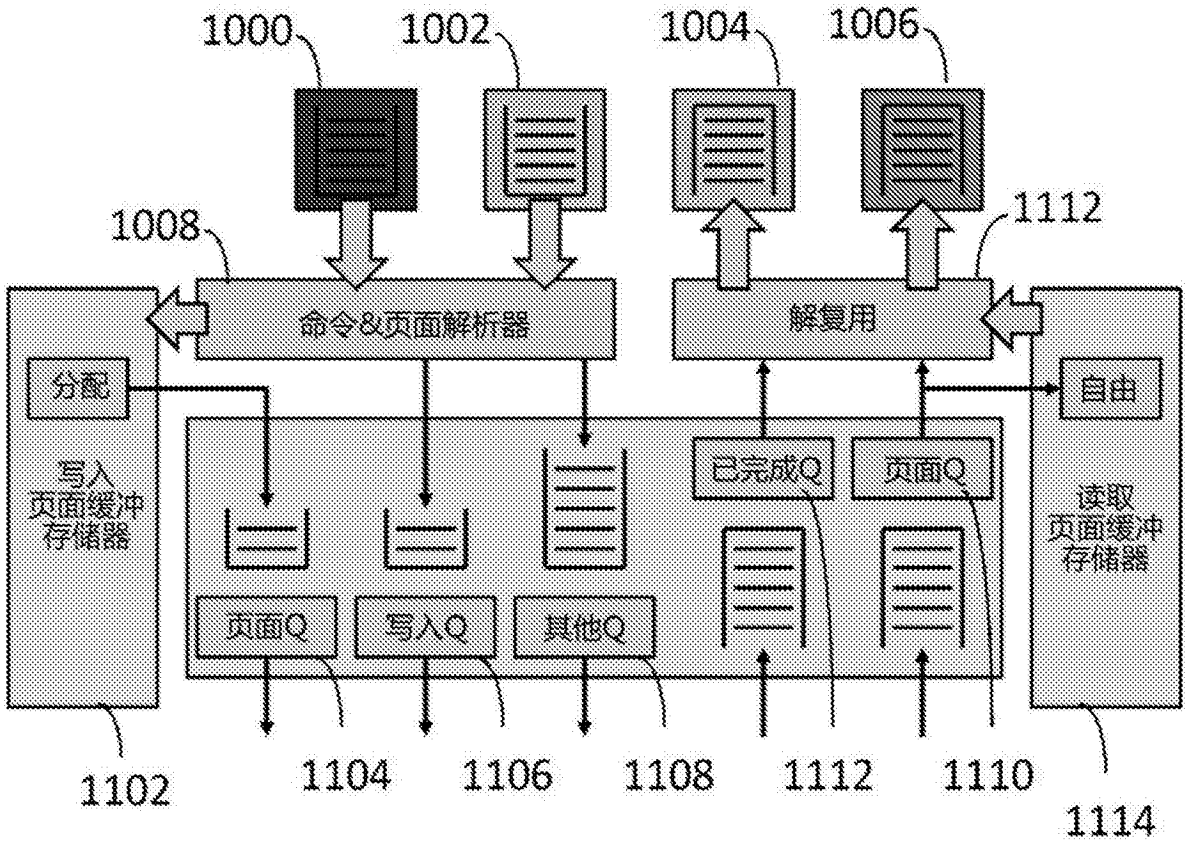

1000
1002
1004
1006
1008
1112
命令&页面解析器
解复用
分配
自由
写入
页面缓冲
存储器
已完成Q
页面Q
读取
页面缓冲
存储器
页面Q
写入Q
其他Q
1102
1104
1106
1108
1112
1110
1114

图11

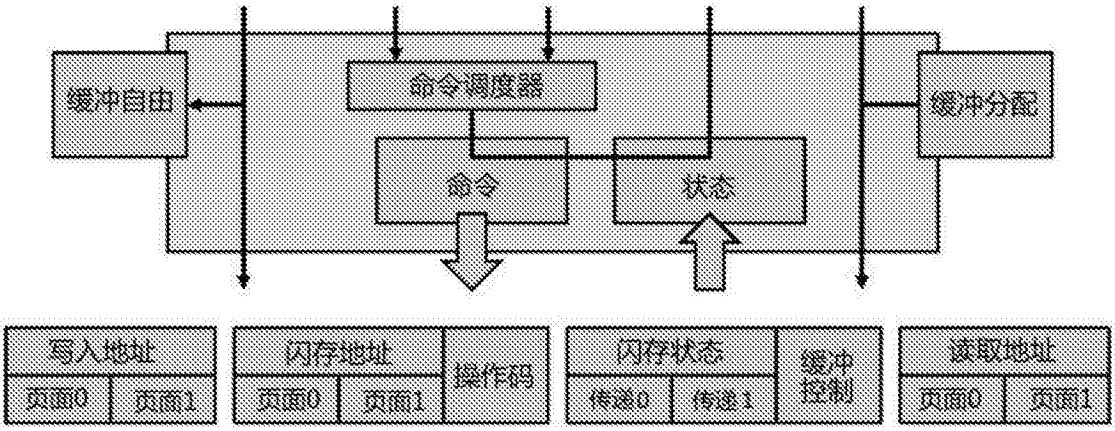

缓冲自由
命令调度器
缓冲分配
命令
状态
写入地址
页面0
页面1
闪存地址
页面0
页面1
操作码
闪存状态
传递0
传递1
缓冲
控制
读取地址
页面0
页面1

图12

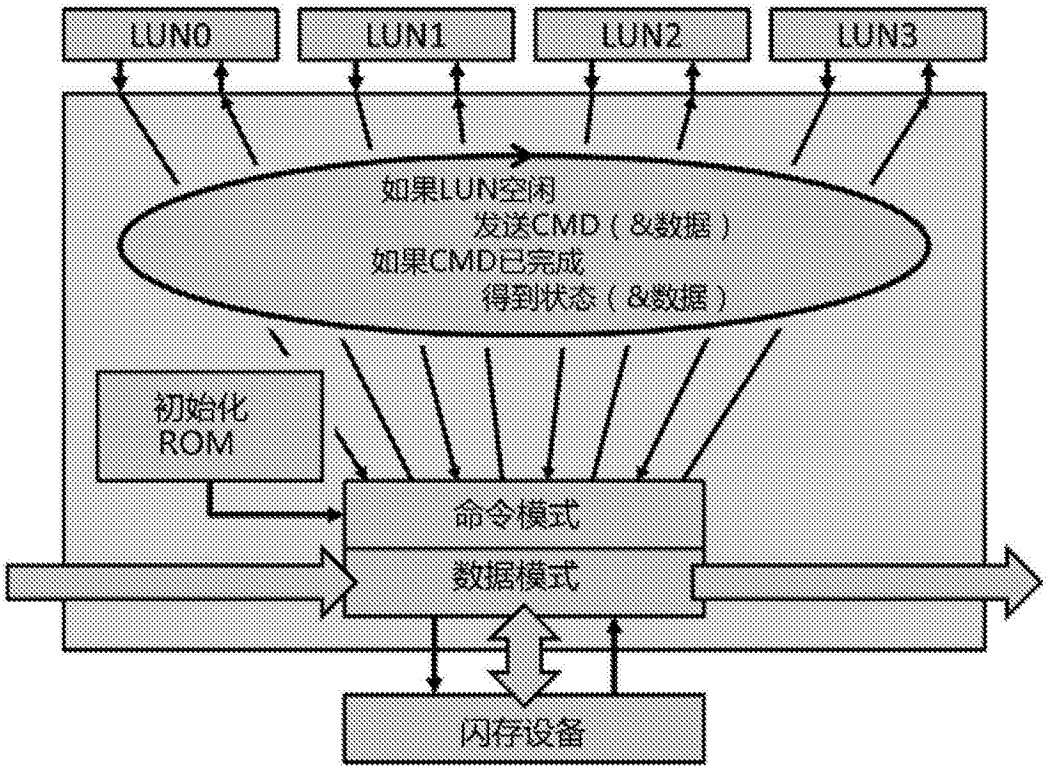

LUN0
LUN1
LUN2
LUN3
如果LUN空闲
发送CMD（&数据）
如果CMD已完成
得到状态（&数据）
初始化
ROM
命令模式
数据模式
闪存设备

图13

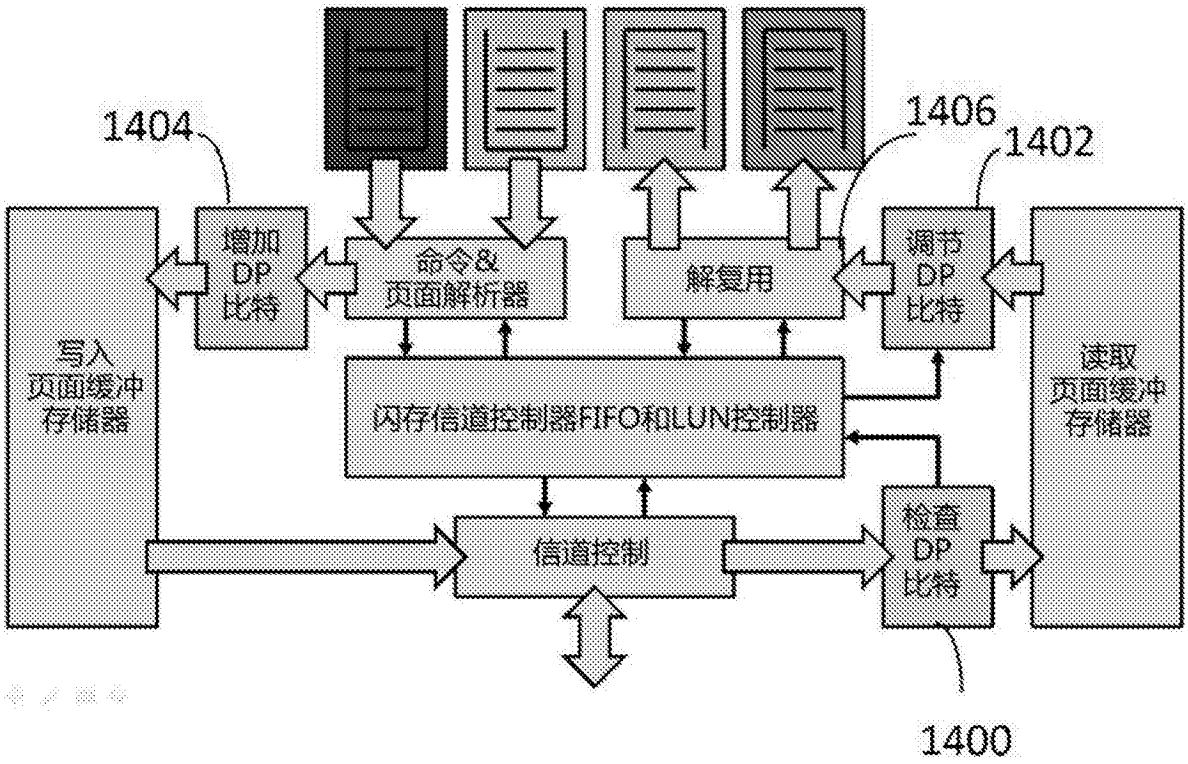

1404
1406
1402
增加
DP
比特
命令&
页面解析器
解复用
调节
DP
比特
写入
页面缓冲
存储器
闪存信道控制器FIFO和LUN控制器
读取
页面缓冲
存储器
信道控制
检查
DP
比特
1400

图14

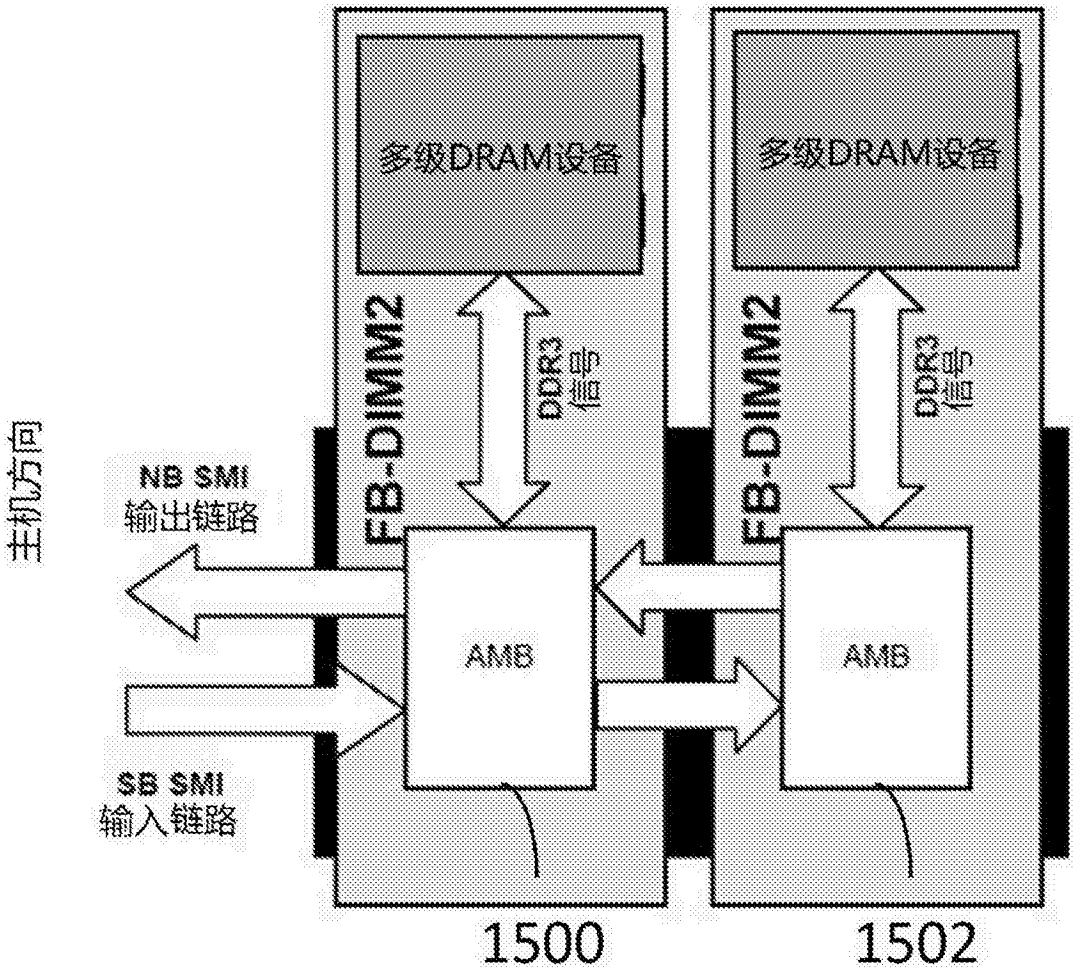

多级DRAM设备
多级DRAM设备
FB-DIMM2
FB-DIMM2
DDR3 命令
DDR3 命令
主机方向
NB SMI
输出链路
SB SMI
输入链路
AMB
AMB
1500
1502

图15

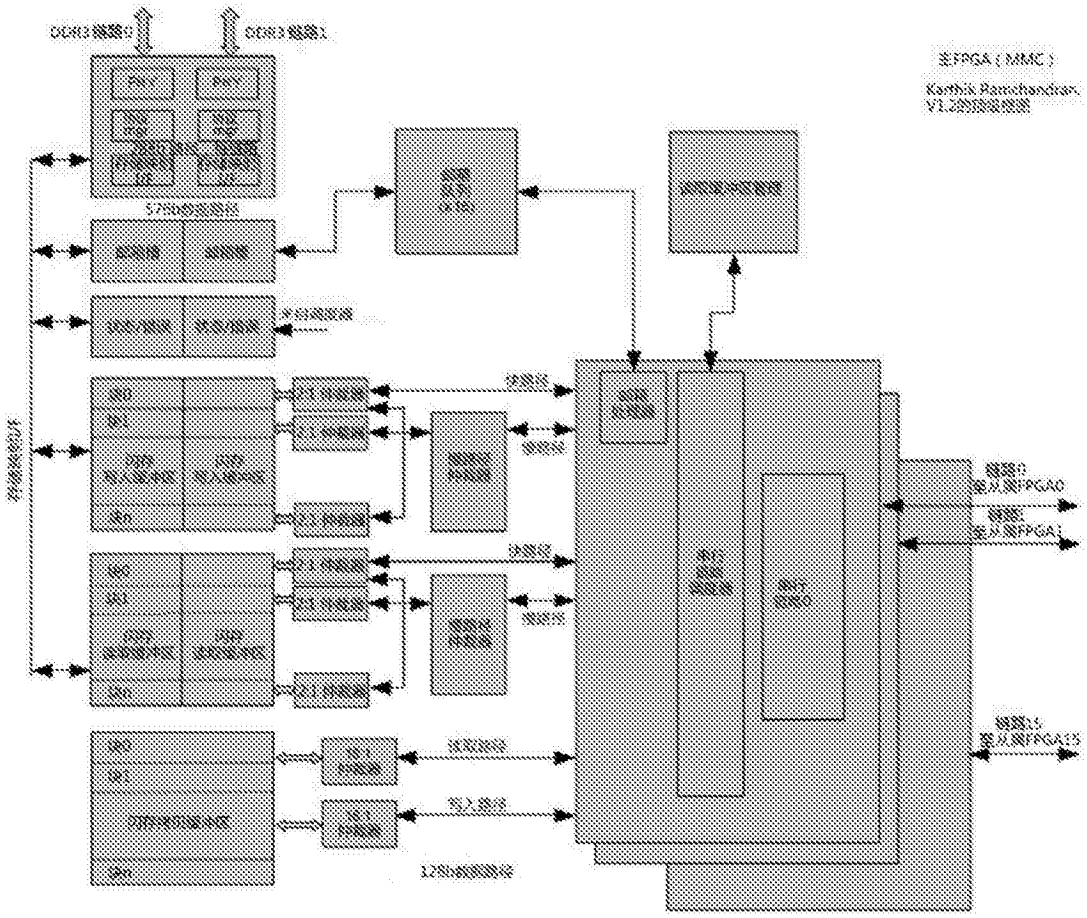


图16

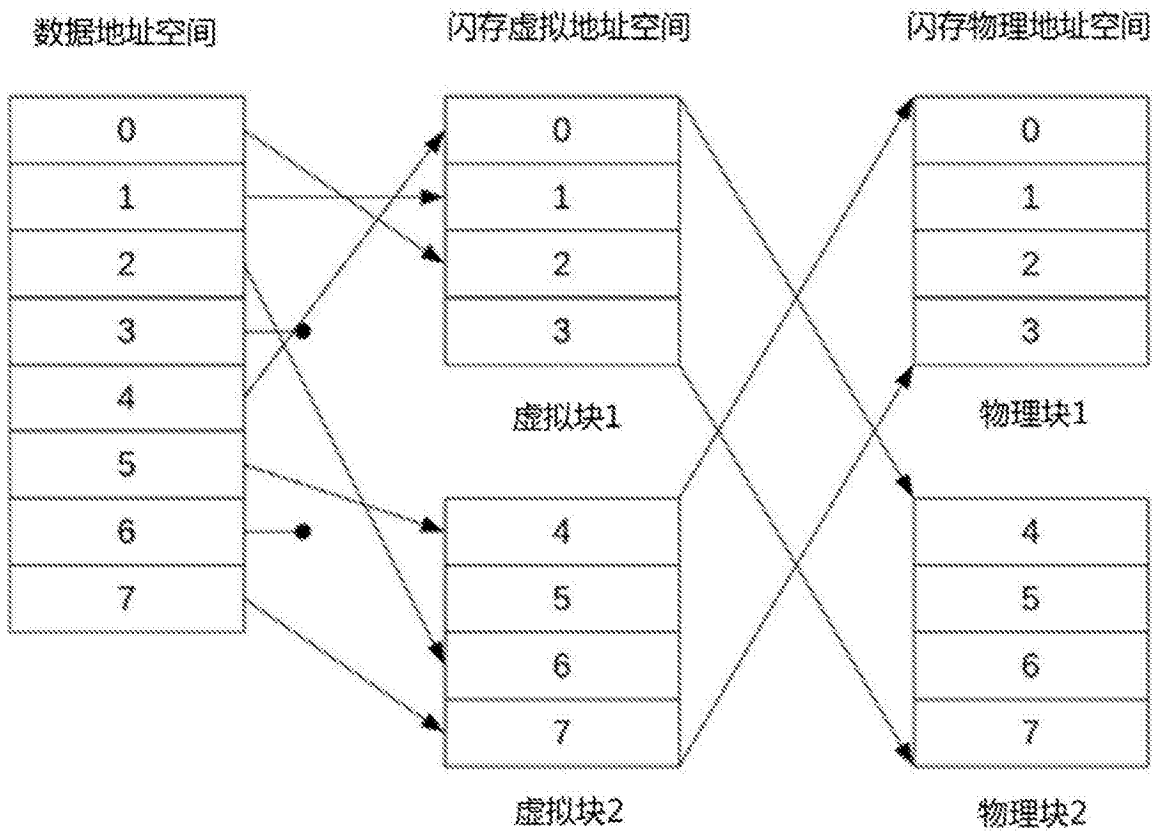

数据地址空间
闪存虚拟地址空间
闪存物理地址空间
0
1
2
3
4
5
6
7
0
1
2
3
虚拟块1
4
5
6
7
虚拟块2
0
1
2
3
物理块1
4
5
6
7
物理块2

图17

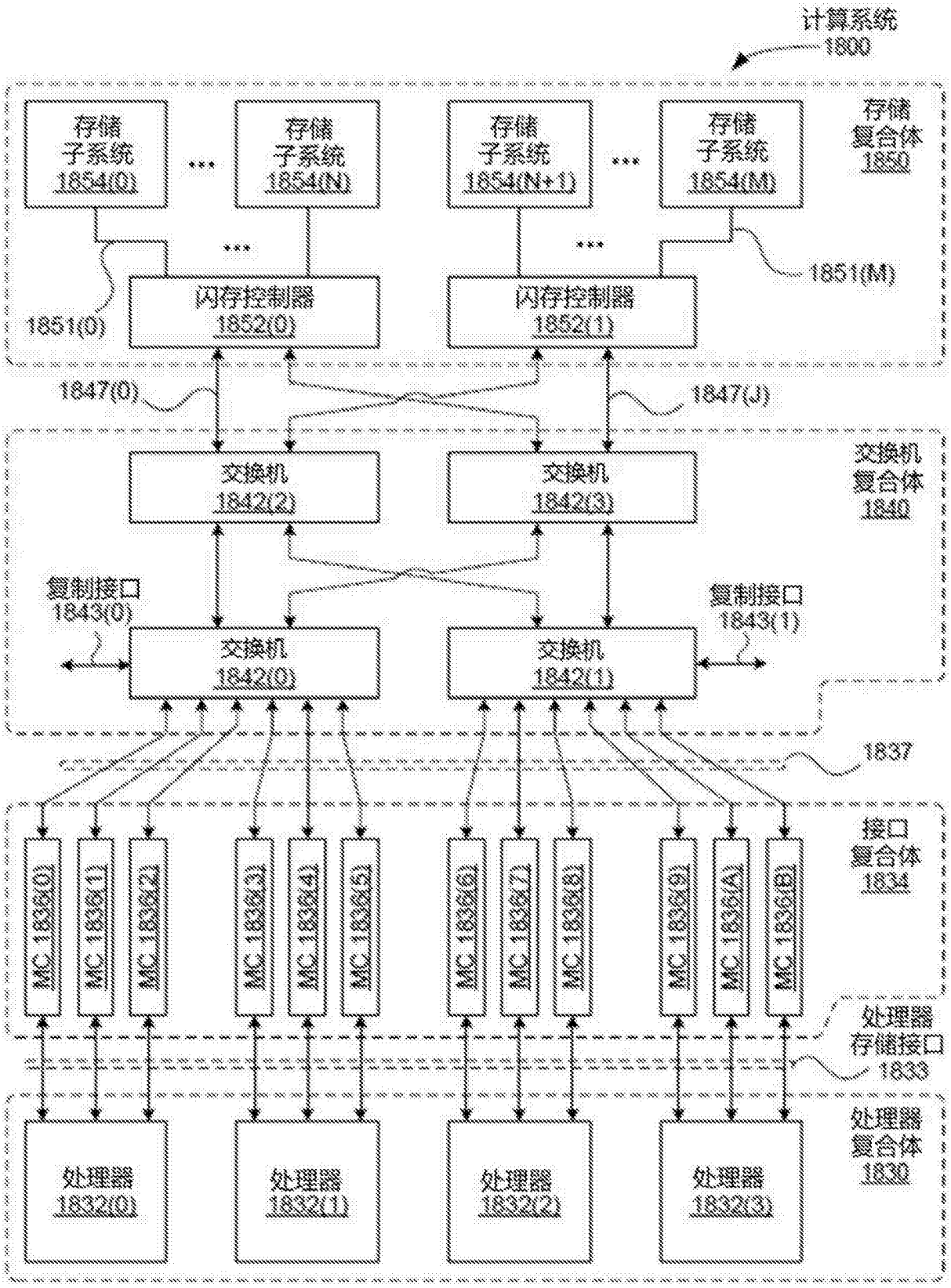

计算系统
1800
存储
子系统
1854(0)
存储
子系统
1854(N)
存储
子系统
1854(N+1)
存储
子系统
1854(M)
存储
复合体
1850
1851(0)
1851(M)
闪存控制器
1852(0)
闪存控制器
1852(1)
1847(0)
1847(J)
交换机
1842(2)
交换机
1842(3)
交换机
复合体
1840
复制接口
1843(0)
复制接口
1843(1)
交换机
1842(0)
交换机
1842(1)
1837
MC 1836(0)
MC 1836(1)
MC 1836(2)
MC 1836(3)
MC 1836(4)
MC 1836(5)
MC 1836(6)
MC 1836(7)
MC 1836(8)
MC 1836(9)
MC 1836(A)
MC 1836(B)
接口
复合体
1834
处理器
存储接口
1833
处理器
1832(0)
处理器
1832(1)
处理器
1832(2)
处理器
1832(3)
处理器
复合体
1830

图18

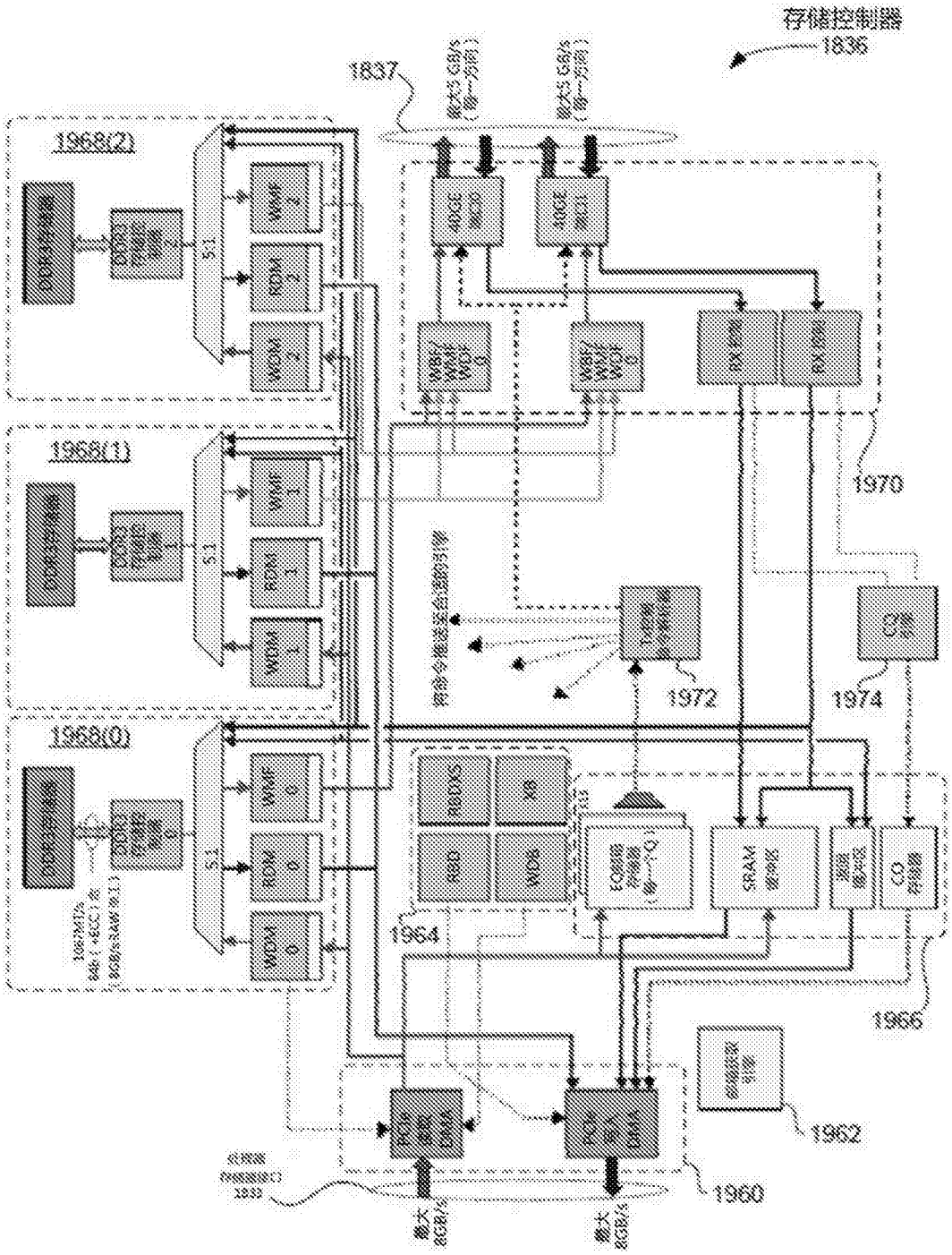

存储控制器
1836
1837
1968(2)
1968(1)
1968(0)
1970
1972
1974
1964
1966
1962
1960

图19

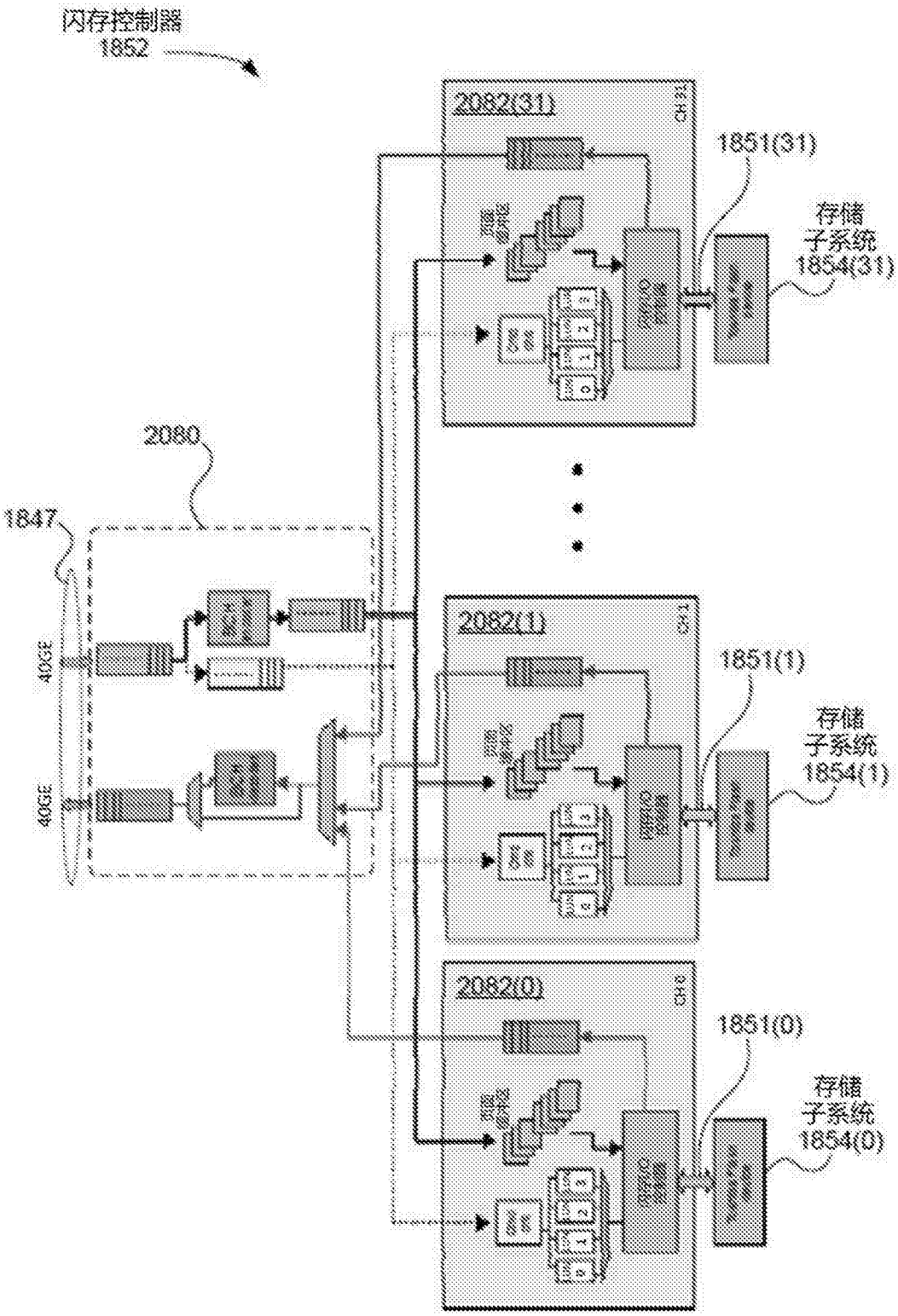

闪存控制器
1852
2082(31)
1851(31)
存储
子系统
1854(31)
2080
1847
40GE
40GE
2082(1)
1851(1)
存储
子系统
1854(1)
2082(0)
1851(0)
存储
子系统
1854(0)

图20

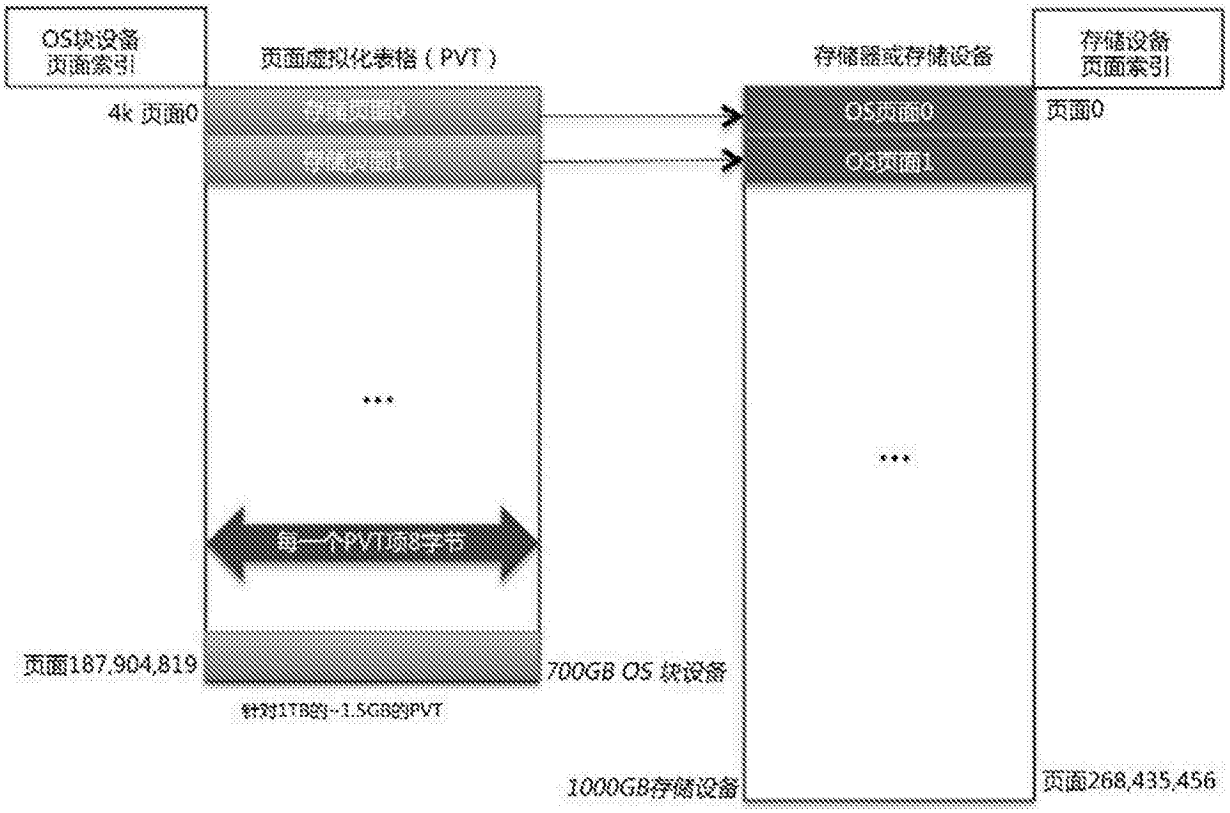

OS块设备
页面索引
页面虚拟化表格（PVT）
存储器或存储设备
存储设备
页面索引
4k 页面0
页面0
OS页面0
OS页面1
...
...
页面187,904,819
700GB OS 块设备
1000GB存储设备
页面268,435,456

图21A

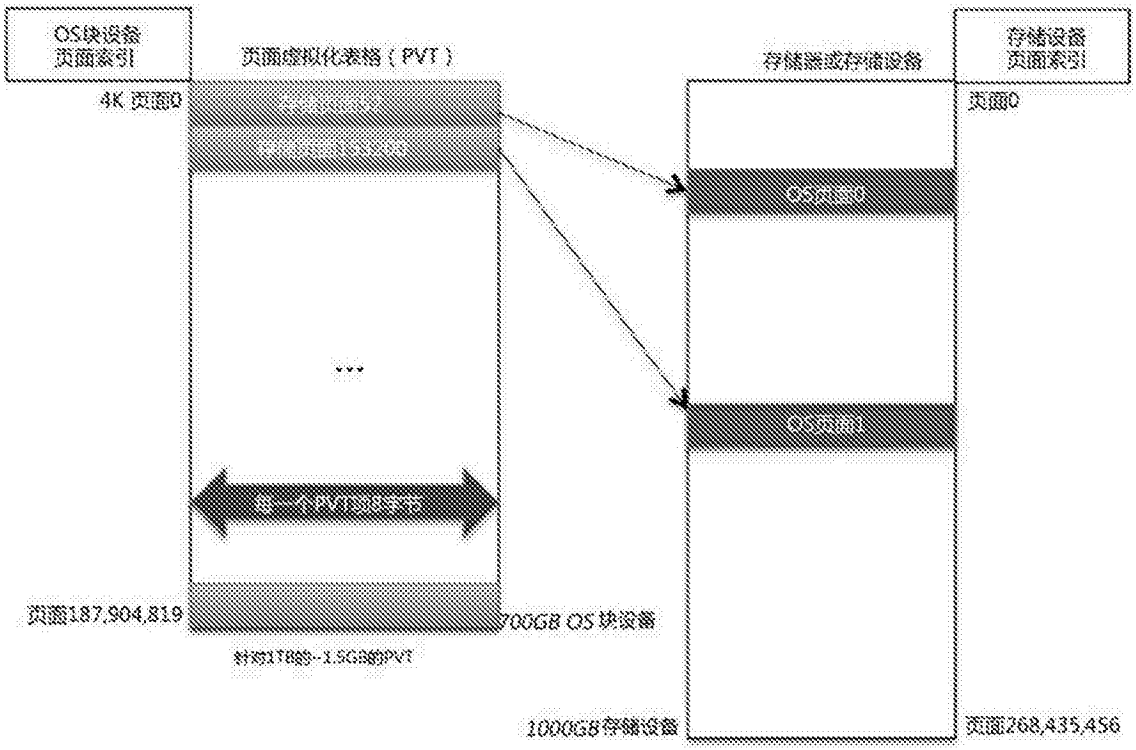

OS块设备
页面索引
页面虚拟化表格（PVT）
存储器或存储设备
存储设备
页面索引
4K 页面0
页面0
OS页面0
OS页面1
...
...
页面187,904,819
700GB OS 块设备
1000GB存储设备
页面268,435,456

图21B

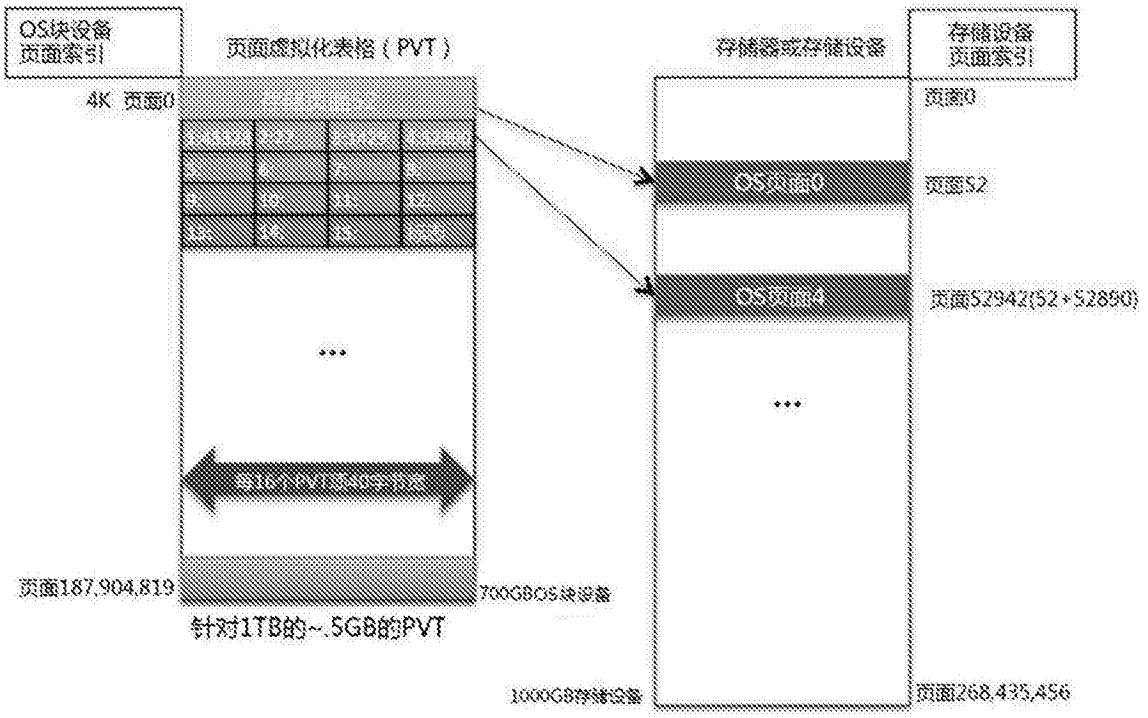

OS块设备
页面索引
页面虚拟化表格（PVT）
存储器或存储设备
存储设备
页面索引
4K 页面0
页面0
OS页面0
页面52
OS页面4
页面52942(52+52890)
...
页面187,904,819
700GBOS块设备
针对1TB的~.5GB的PVT
1000GB存储设备
页面268,435,456

图21C

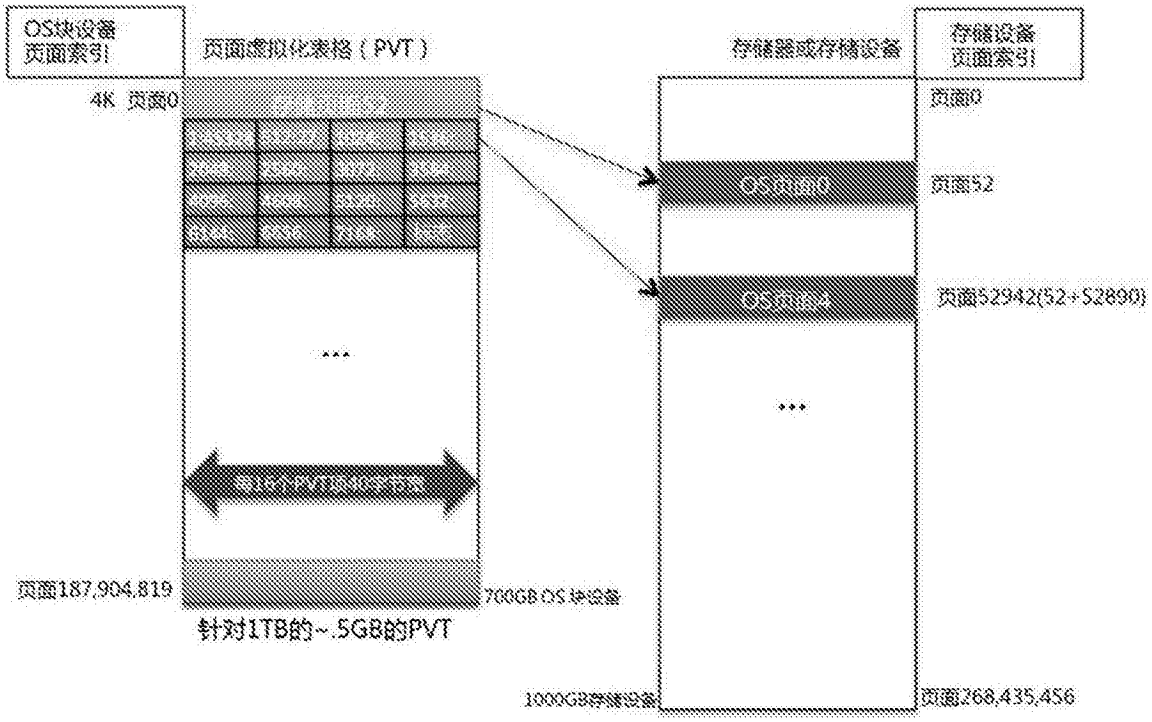

OS块设备
页面索引
页面虚拟化表格（PVT）
存储器或存储设备
存储设备
页面索引
4K 页面0
页面0
OS页面0
页面52
OS页面4
页面52942(52+52890)
...
页面187,904,819
700GB OS 块设备
针对1TB的~.5GB的PVT
1000GB存储设备
页面268,435,456

图21D

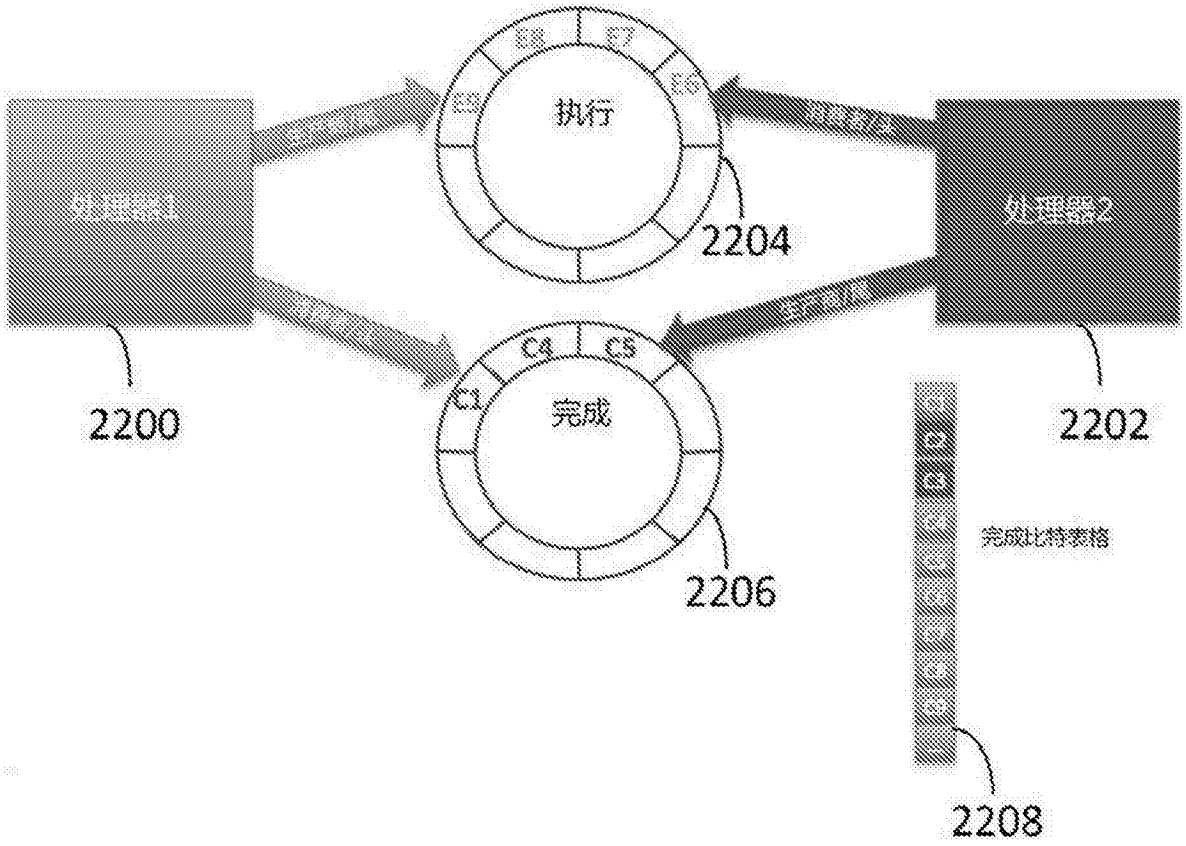

执行
2204
处理器1
处理器2
2200
2202
完成
2206
完成比特表格
2208

图22

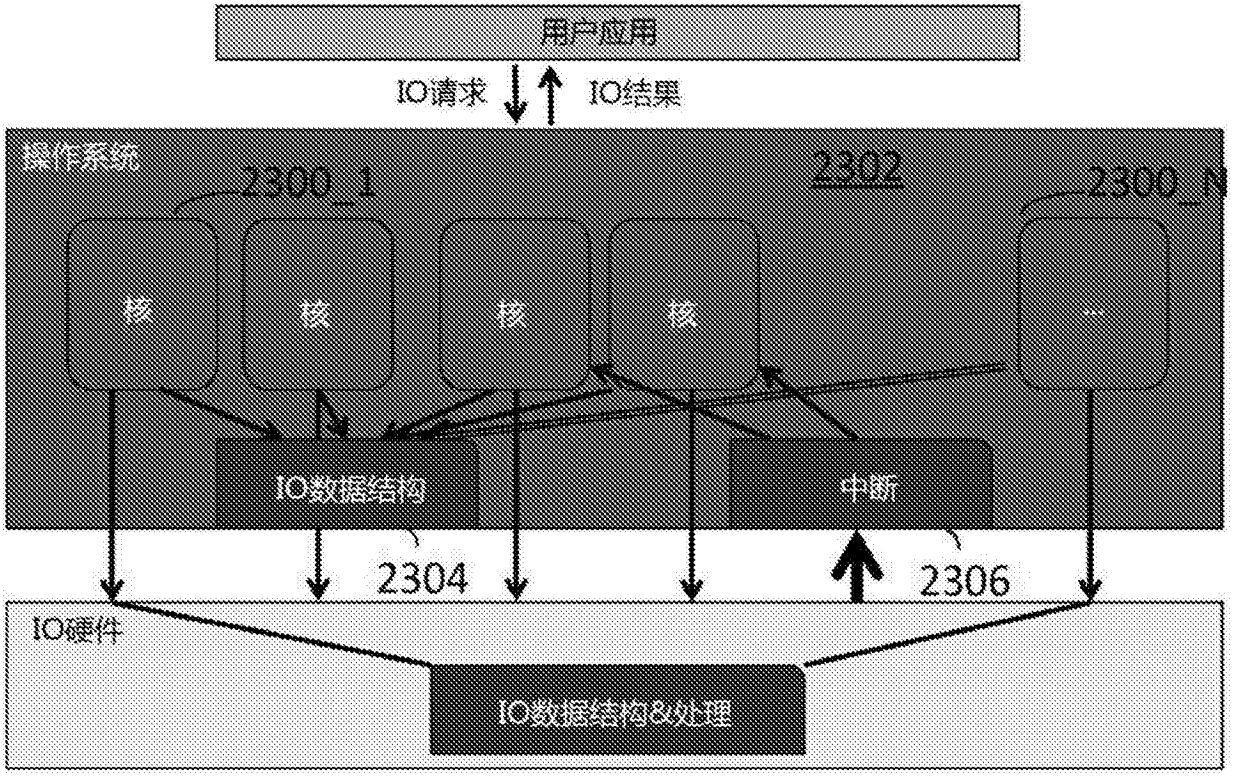

用户应用
IO请求
IO结果
操作系统
2300_1
2302
2300_N
核
核
核
核
...
N
IO数据结构
中断
2304
2306
IO硬件
IO数据结构&处理

图23(现有技术)

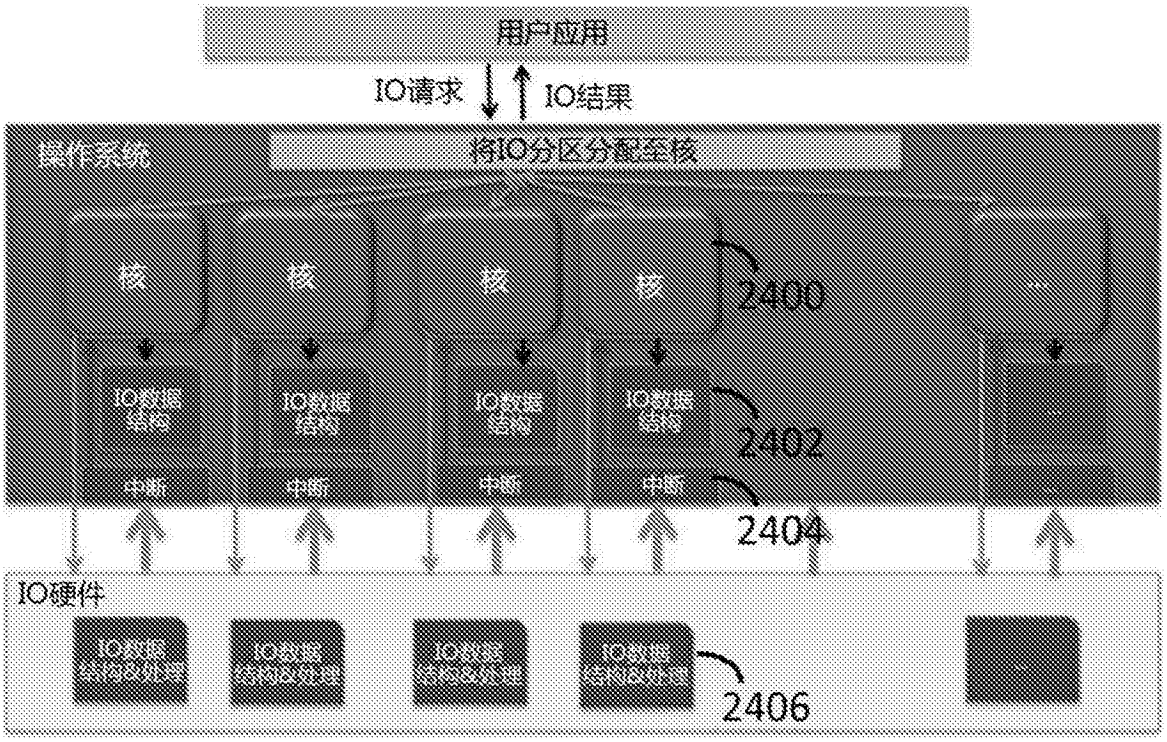

用户应用
IO请求
IO结果
操作系统
将IO分区分配至核
核
核
核
核
2400
IO数据结构
IO数据结构
IO数据结构
IO数据结构
2402
中断
中断
中断
中断
2404
IO硬件
IO数据结构的处理
IO数据结构的处理
IO数据结构的处理
IO数据结构的处理
2406

图24

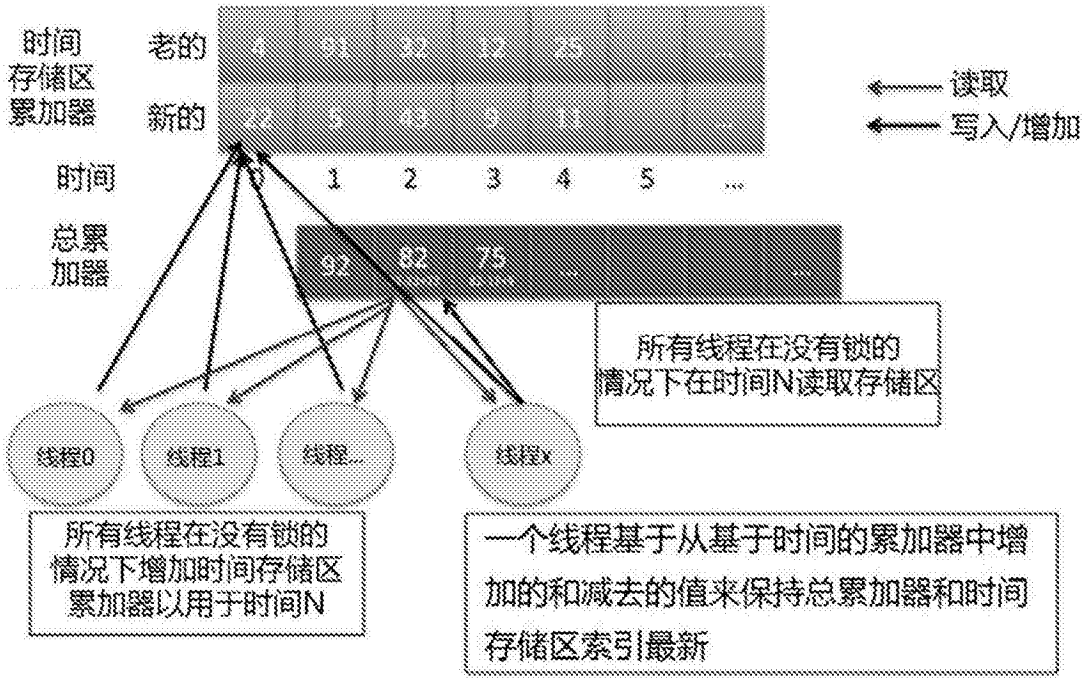

时间存储区累加器
老的
新的
时间
总累加器
读取
写入/增加
0
1
2
3
4
5
...
92
82
75
所有线程在没有锁的情况下在时间N读取存储区
线程0
线程1
线程...
线程x
所有线程在没有锁的情况下增加时间存储区累加器以用于时间N
一个线程基于从基于时间的累加器中增加的和减去的值来保持总累加器和时间存储区索引最新

图25

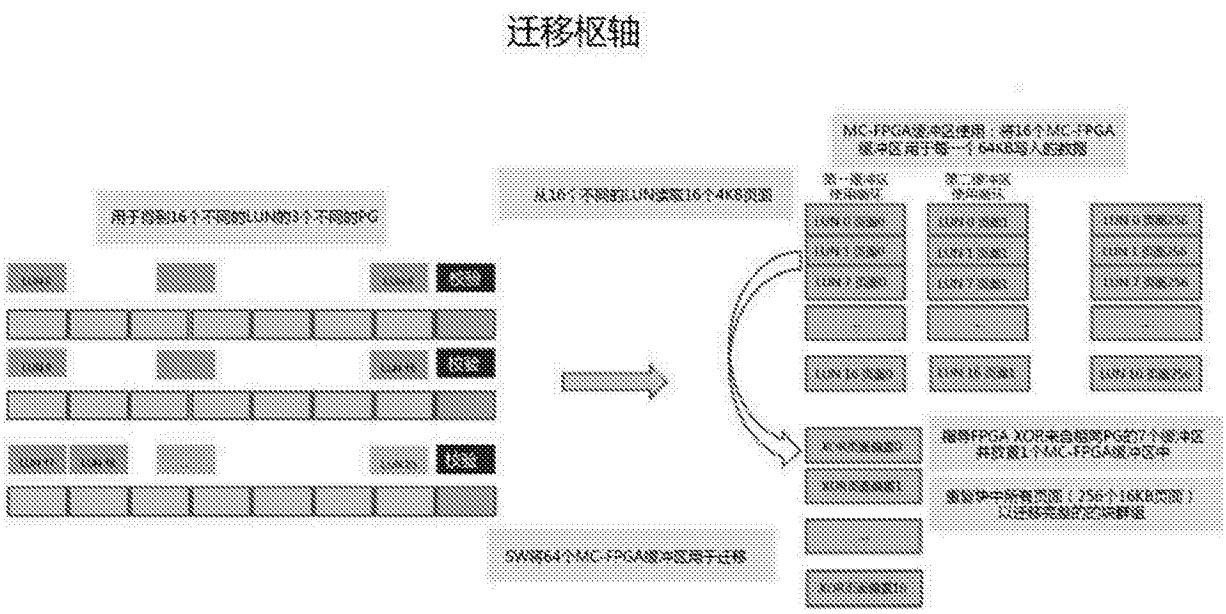

迁移枢轴

图26

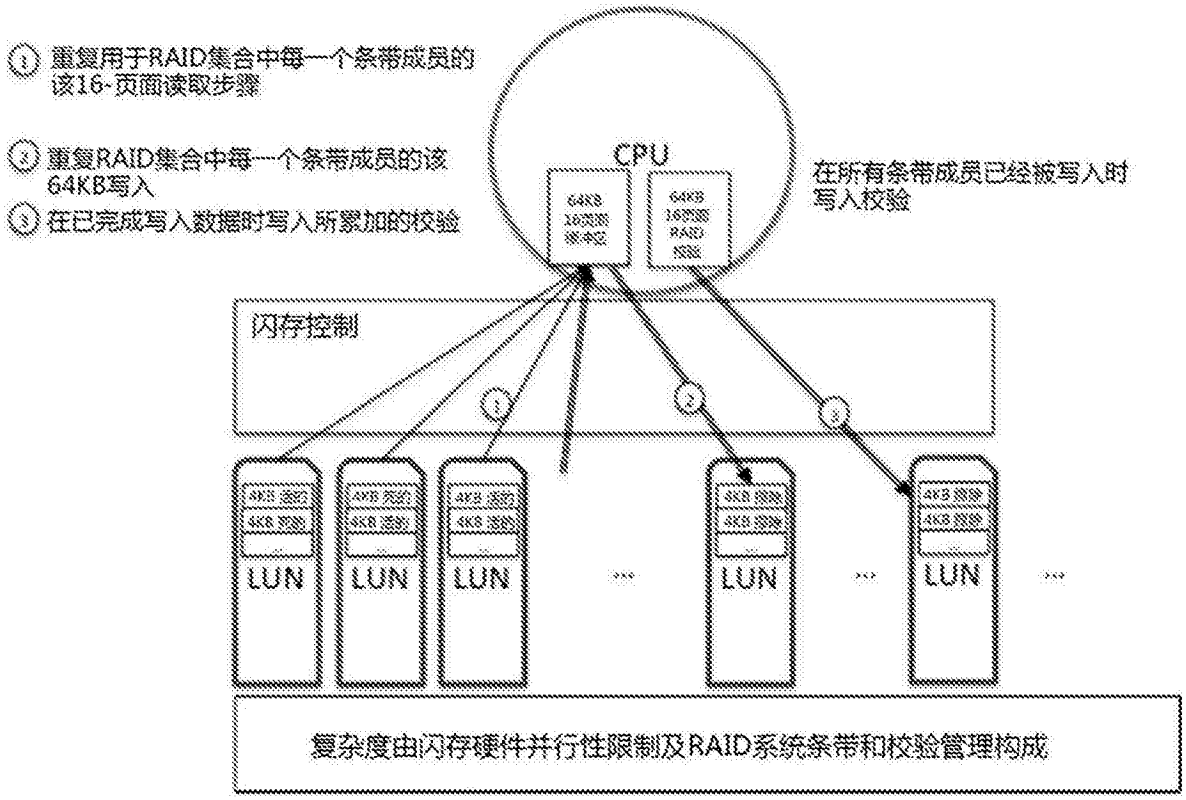

① 重复用于RAID集合中每一个条带成员的该16-页面读取步骤
② 重复RAID集合中每一个条带成员的该64KB写入
③ 在已完成写入数据时写入所累加的校验
CPU
在所有条带成员已经被写入时写入校验
闪存控制
LUN
LUN
LUN
LUN
LUN
复杂度由闪存硬件并行性限制及RAID系统条带和校验管理构成

图27(现有技术)

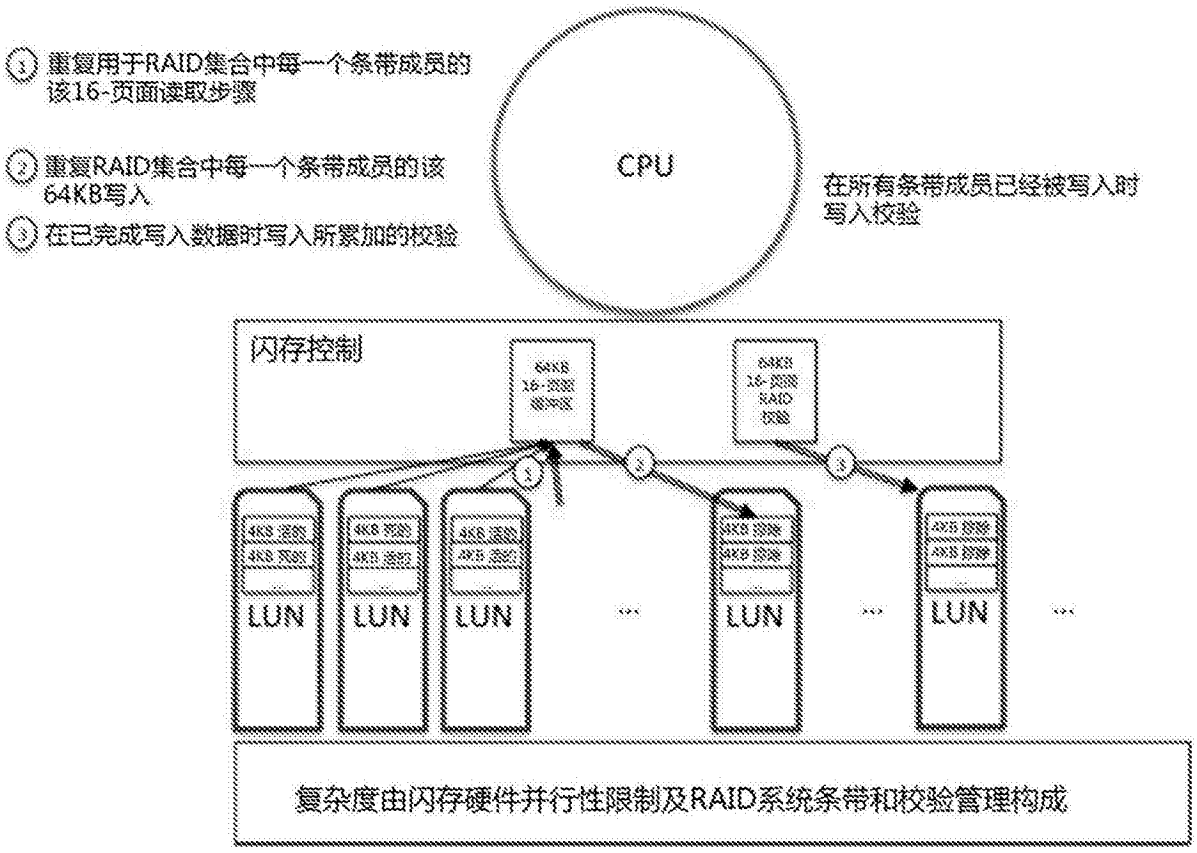

① 重复用于RAID集合中每一个条带成员的该16-页面读取步骤
② 重复RAID集合中每一个条带成员的该64KB写入
③ 在已完成写入数据时写入所累加的校验
CPU
在所有条带成员已经被写入时写入校验
闪存控制
LUN
LUN
LUN
LUN
LUN
复杂度由闪存硬件并行性限制及RAID系统条带和校验管理构成

图28

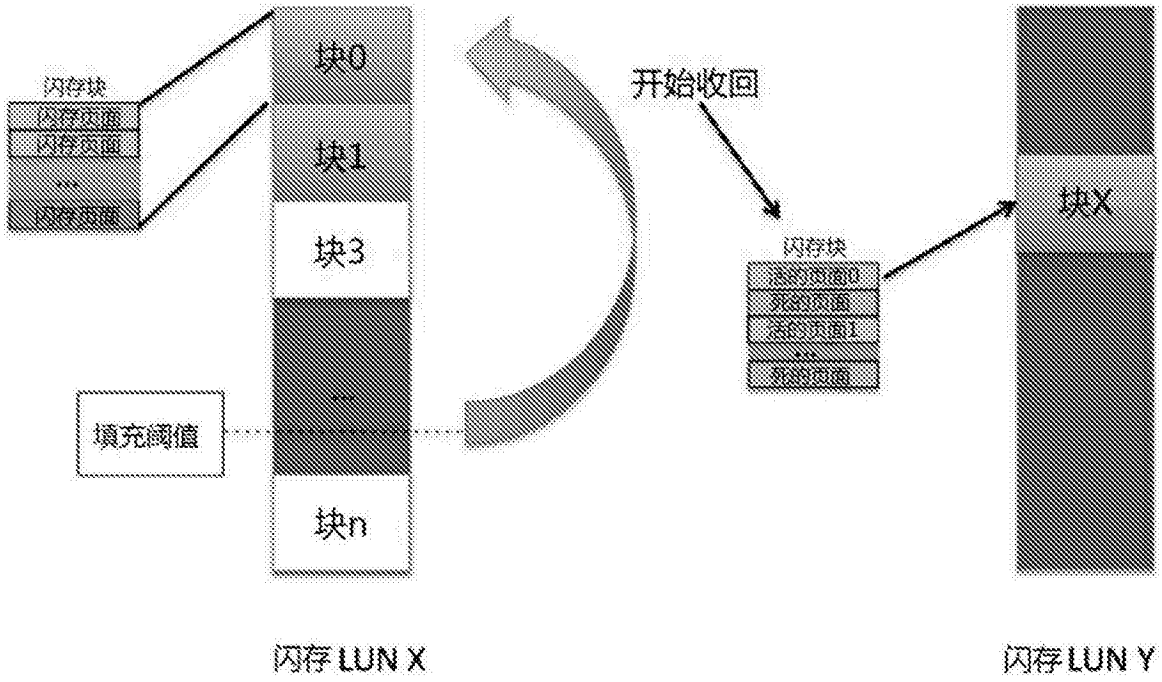

闪存块
块0
块1
块3
块n
填充阈值
开始收回
闪存块
块X
闪存 LUN X
闪存 LUN Y

图29

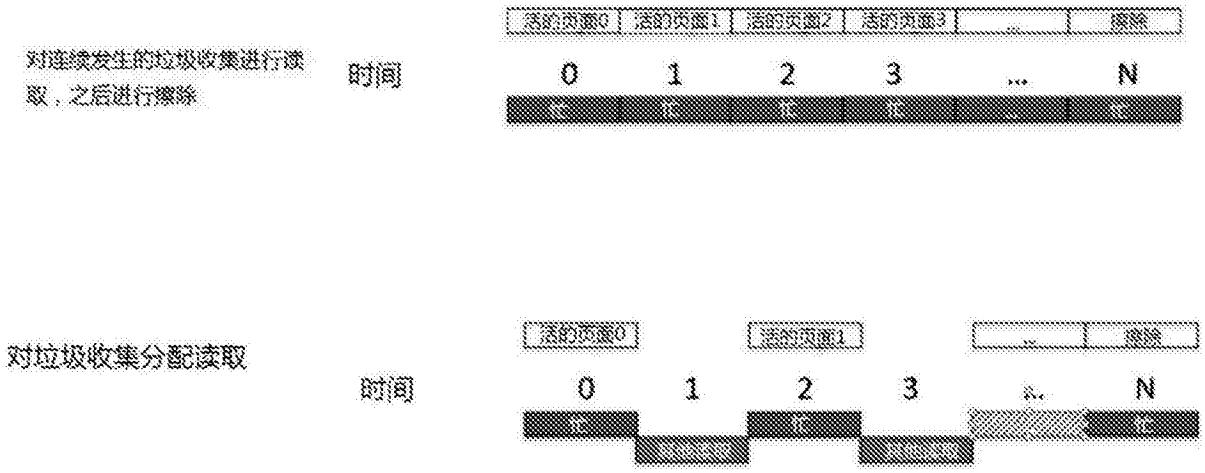

对连续发生的垃圾收集进行读取，之后进行擦除
时间
活的页面0
活的页面1
活的页面2
活的页面3
擦除
0
1
2
3
...
N
对垃圾收集分配读取
时间
活的页面0
活的页面1
擦除
0
1
2
3
N

图30

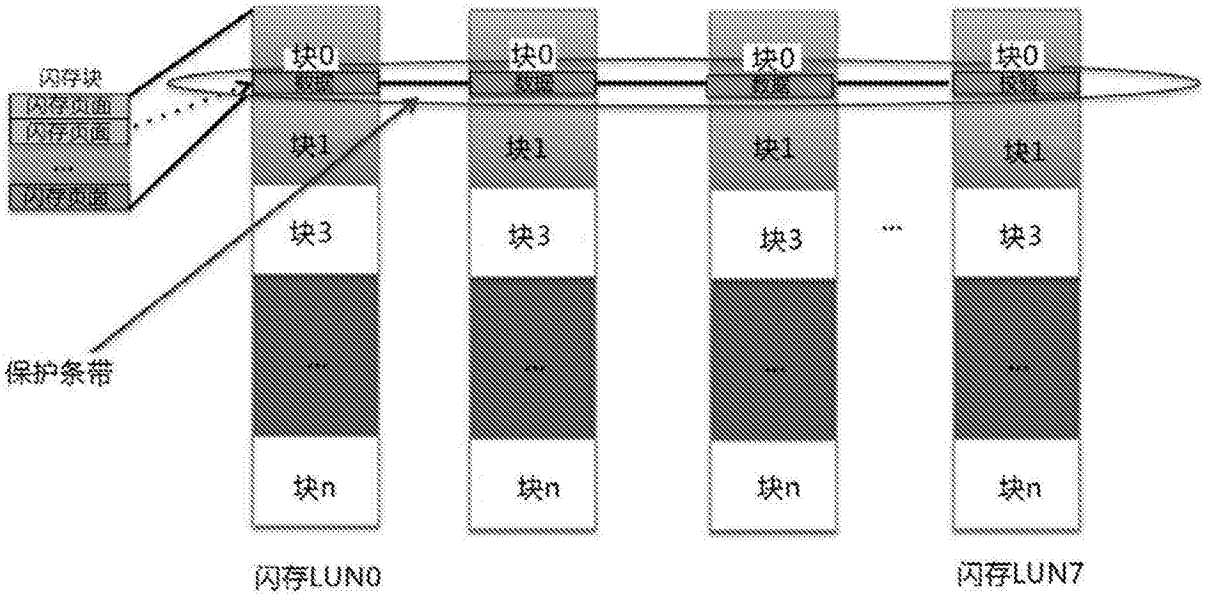

闪存块
闪存页面
闪存页面
闪存页面
保护条带
块0
数据
块1
块3
块n
闪存LUN0
闪存LUN7

图31

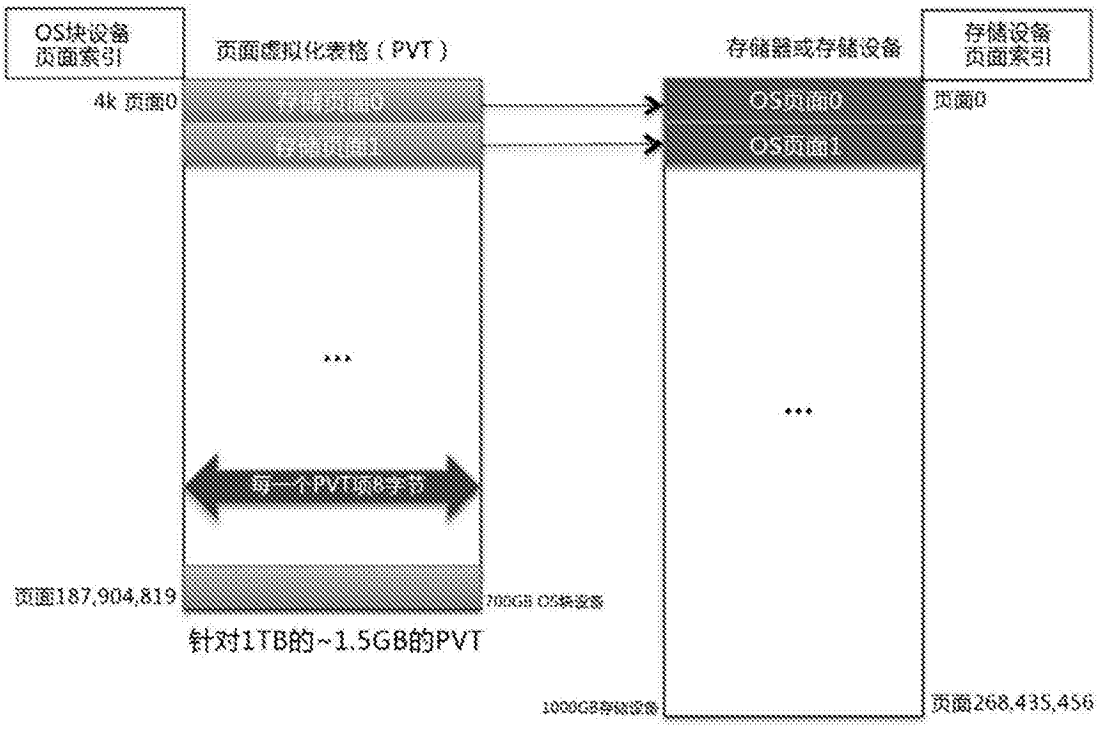

OS块设备
页面索引
页面虚拟化表格（PVT）
存储器或存储设备
存储设备
页面索引
4k 页面0
页面0
OS页面0
OS页面1
...
...
页面187,904,819
700GB OS块设备
针对1TB的~1.5GB的PVT
1000GB存储设备
页面268,435,456

图32

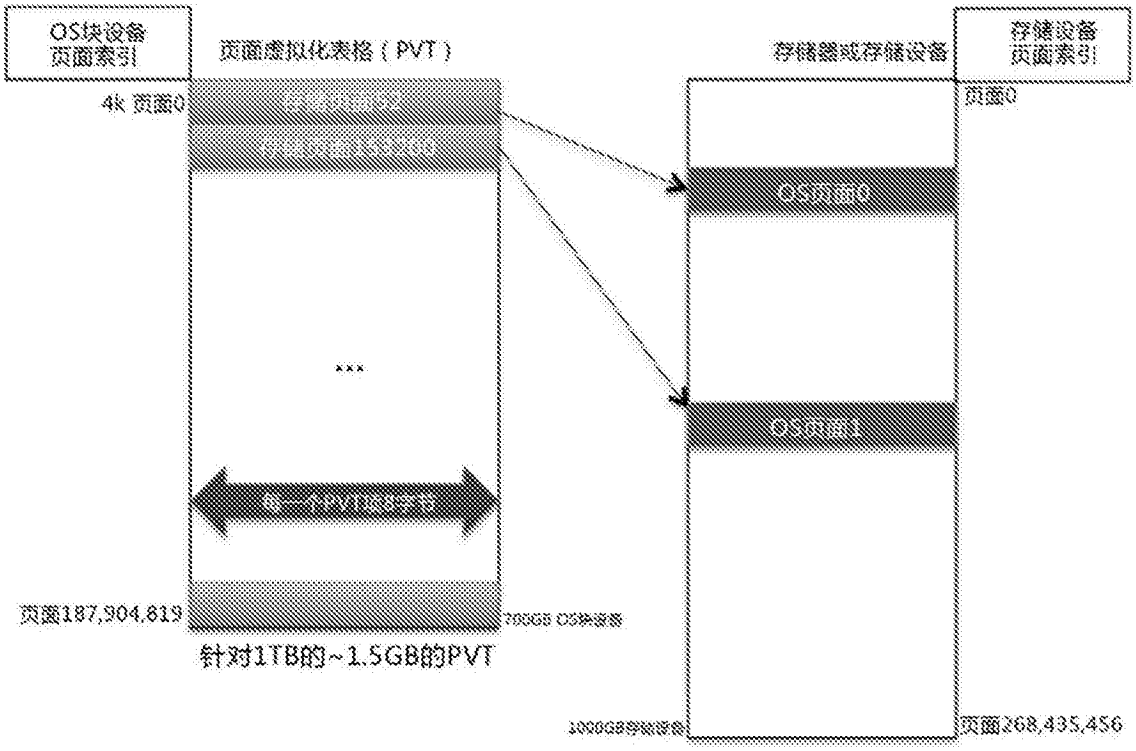

OS块设备
页面索引
页面虚拟化表格（PVT）
存储器或存储设备
存储设备
页面索引
4k 页面0
页面0
OS页面0
OS页面1
...
...
页面187,904,819
700GB OS块设备
针对1TB的~1.5GB的PVT
1000GB存储设备
页面268,435,456

图33

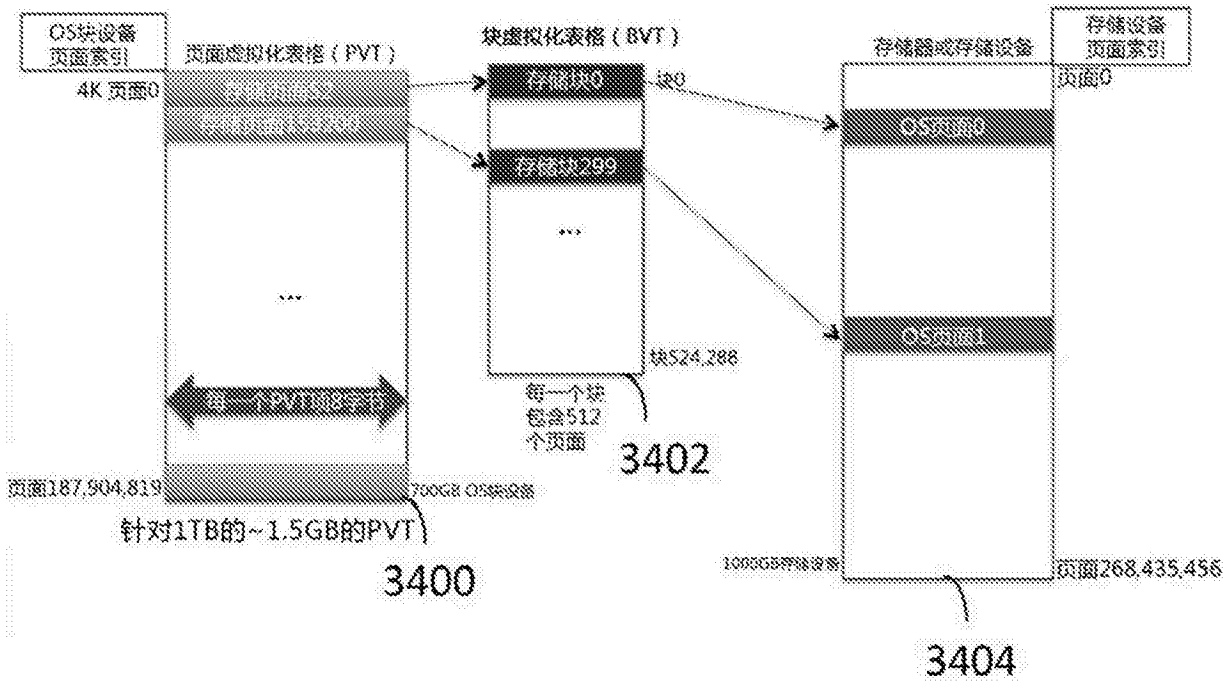

OS块设备
页面索引
页面虚拟化表格（PVT）
4K 页面0
页面187,904,819
700GB OS块设备
针对1TB的~1.5GB的PVT
3400
块虚拟化表格（BVT）
块0
块524,288
每一个块
包含512
个页面
3402
存储器或存储设备
存储设备
页面索引
页面0
页面268,435,456
3404

图34

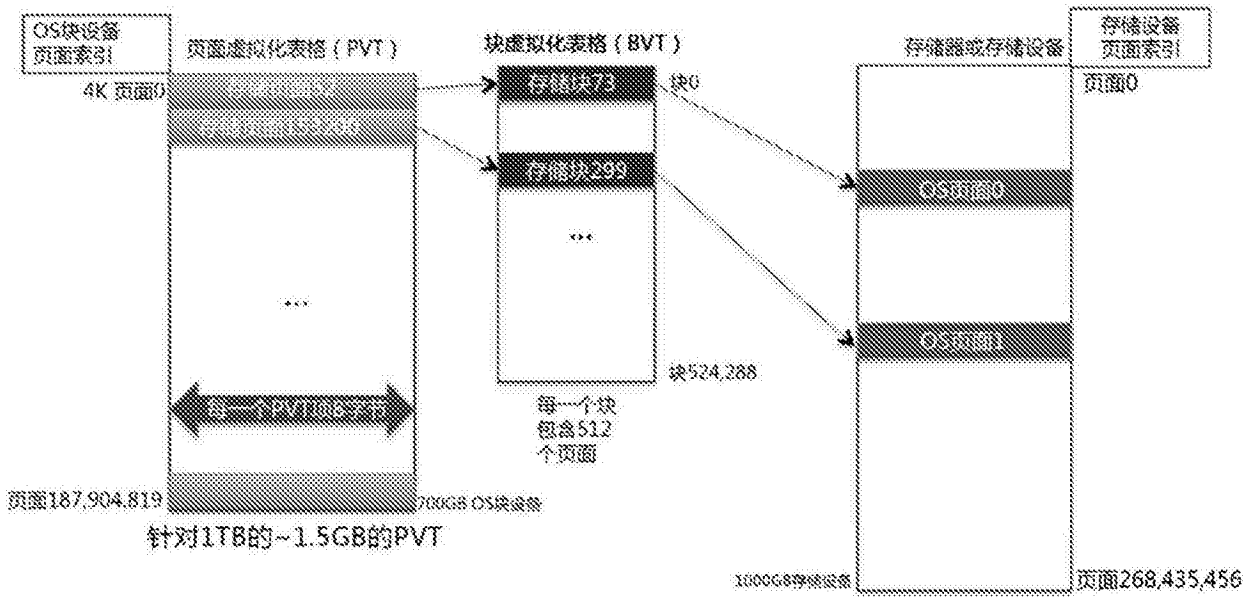

OS块设备
页面索引
页面虚拟化表格（PVT）
4K 页面0
页面187,904,819
700GB OS块设备
针对1TB的~1.5GB的PVT
块虚拟化表格（BVT）
块0
块524,288
每一个块
包含512
个页面
存储器或存储设备
存储设备
页面索引
页面0
页面268,435,456

图35

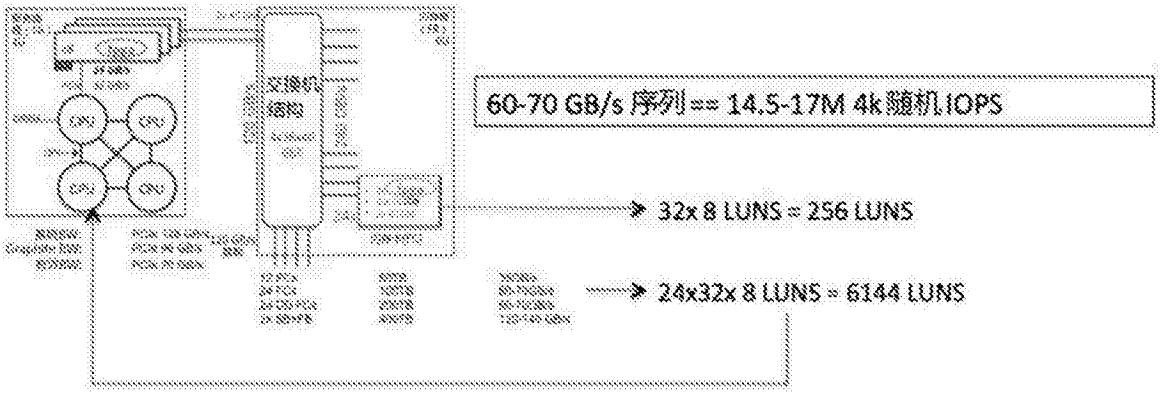

60-70 GB/s 序列 == 14.5-17M 4k 随机 IOPS
32x 8 LUNS = 256 LUNS
24x32x 8 LUNS = 6144 LUNS
我们向OS驱动器揭示了每一个独立可编程LUN
• 大多数供应商具有随机读取IOP性能，该性能是它们的序列BW读取性能的1/4至1/10。
• Graphite 1 IOP性能等同于其BW性能。

图36

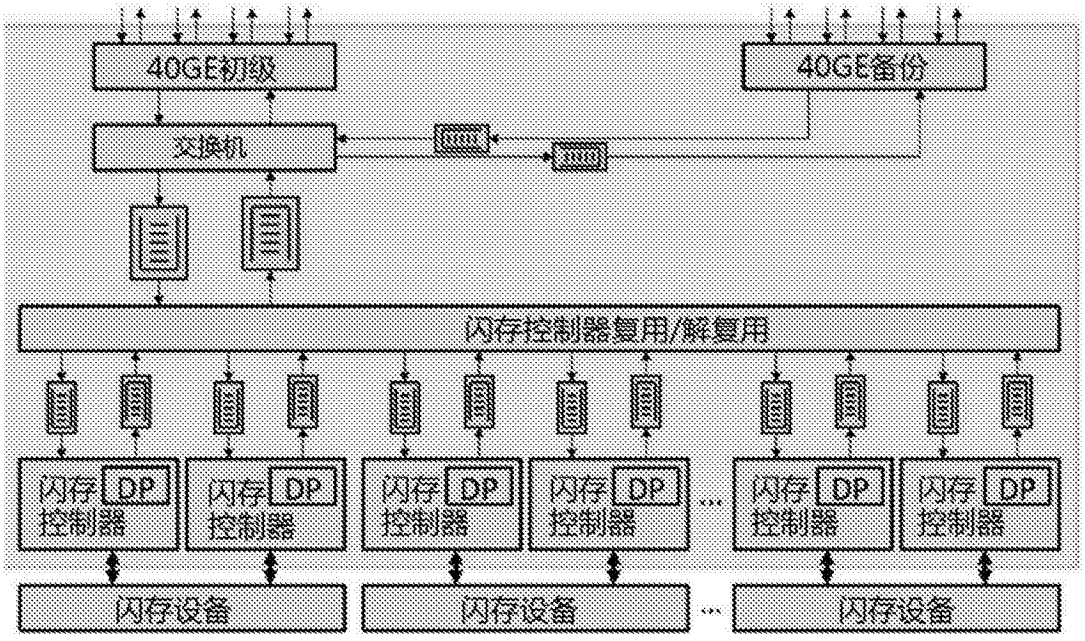

40GE初级
40GE备份
交换机
闪存控制器复用/解复用
闪存 DP 控制器
闪存设备

图37

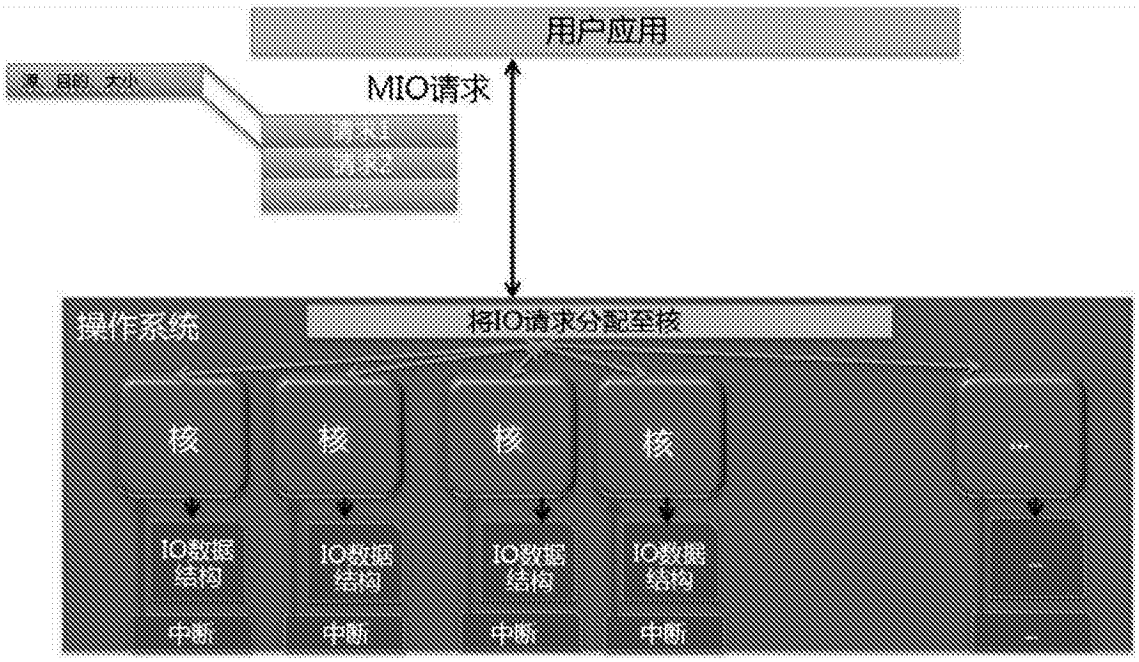

用户应用
MIO请求
操作系统
将IO请求分配至核
核
核
核
核
IO数据结构
IO数据结构
IO数据结构
IO数据结构
中断
中断
中断
中断

图38

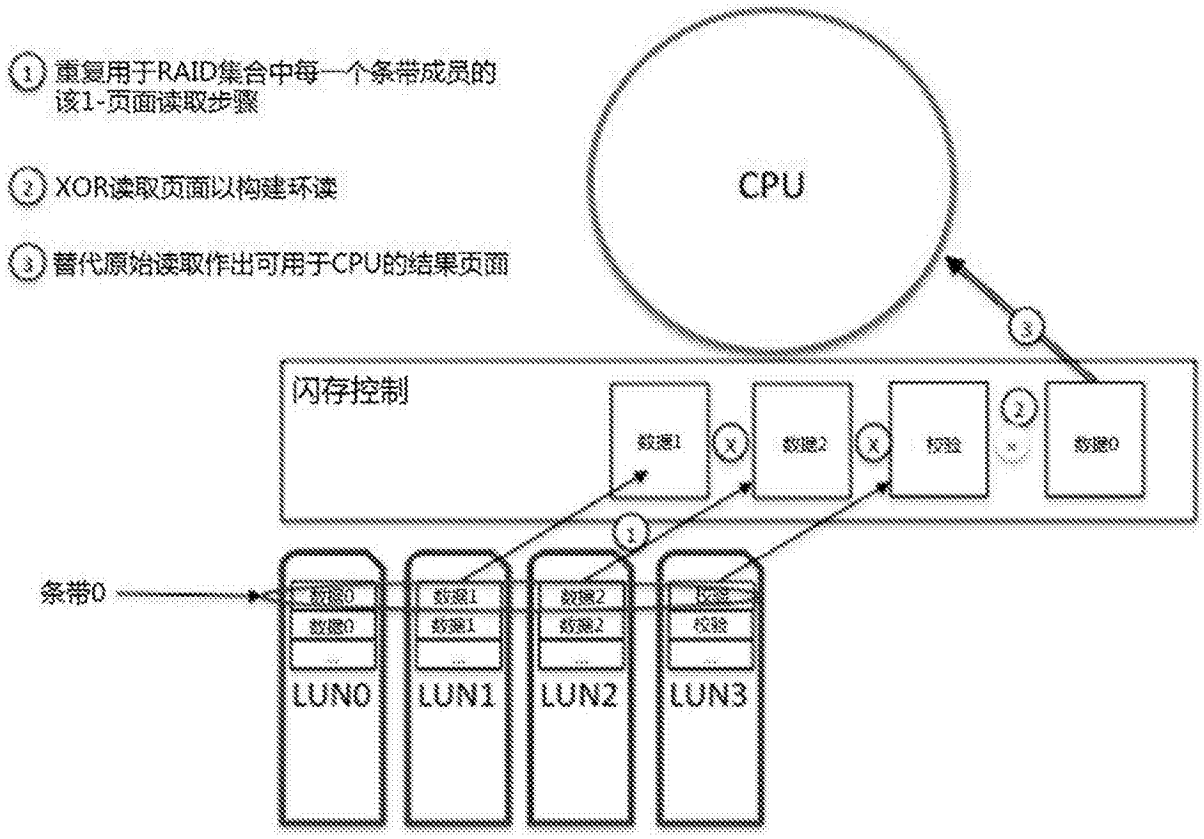

1 重复用于RAID集合中每一个条带成员的该1-页面读取步骤
2 XOR读取页面以构建环读
3 替代原始读取作出可用于CPU的结果页面
CPU
闪存控制
数据1
数据2
校验
数据0
条带0
LUN0
LUN1
LUN2
LUN3

图39

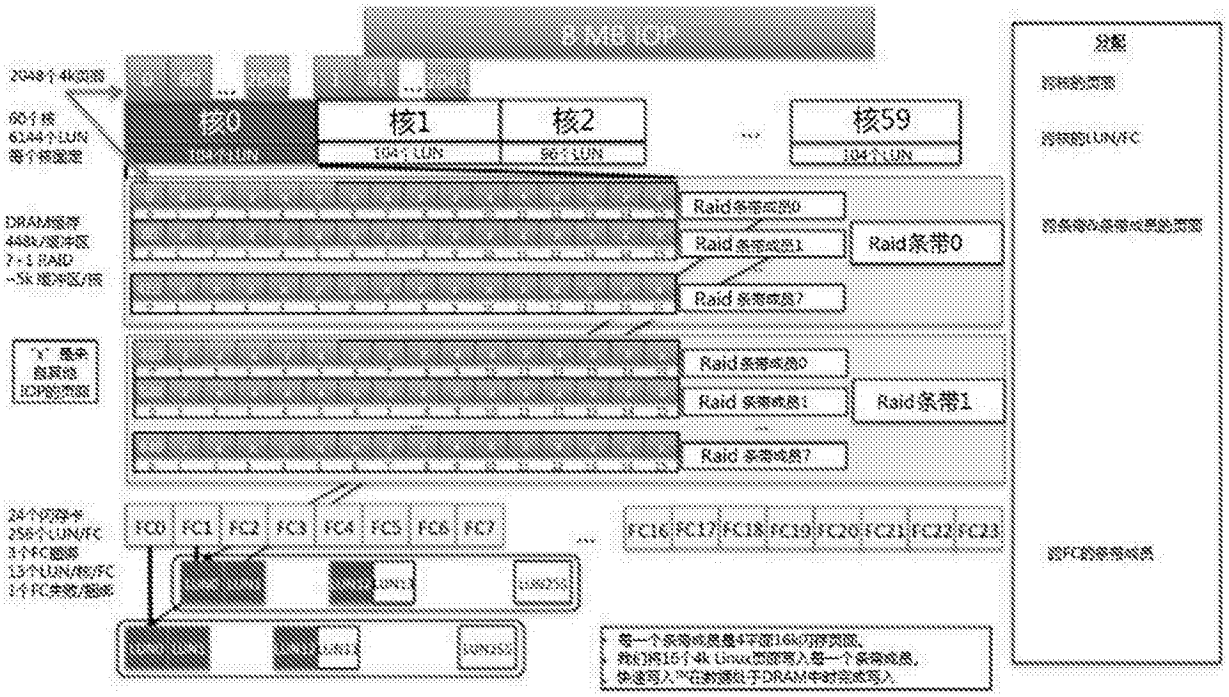

核0
核1
核2
核59
Raid条带0
Raid条带1
FC0
FC1
FC2
FC3
FC4
FC5
FC6
FC7

图40

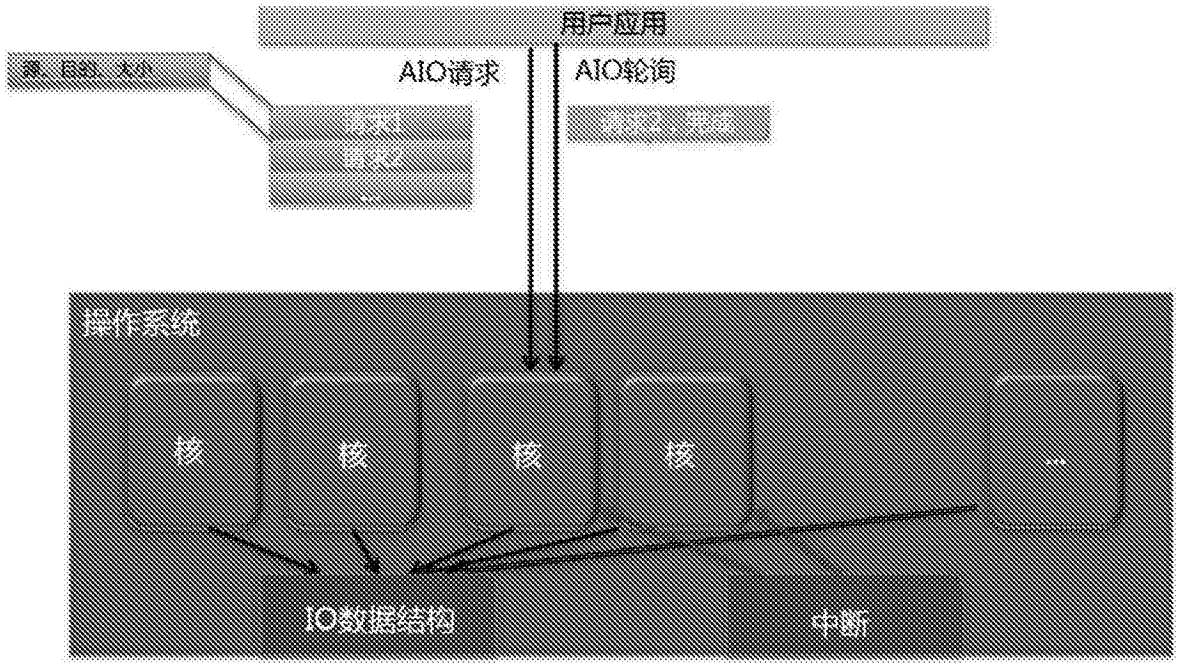

用户应用
AIO请求
AIO轮询
操作系统
核
核
核
核
IO数据结构
中断

图41

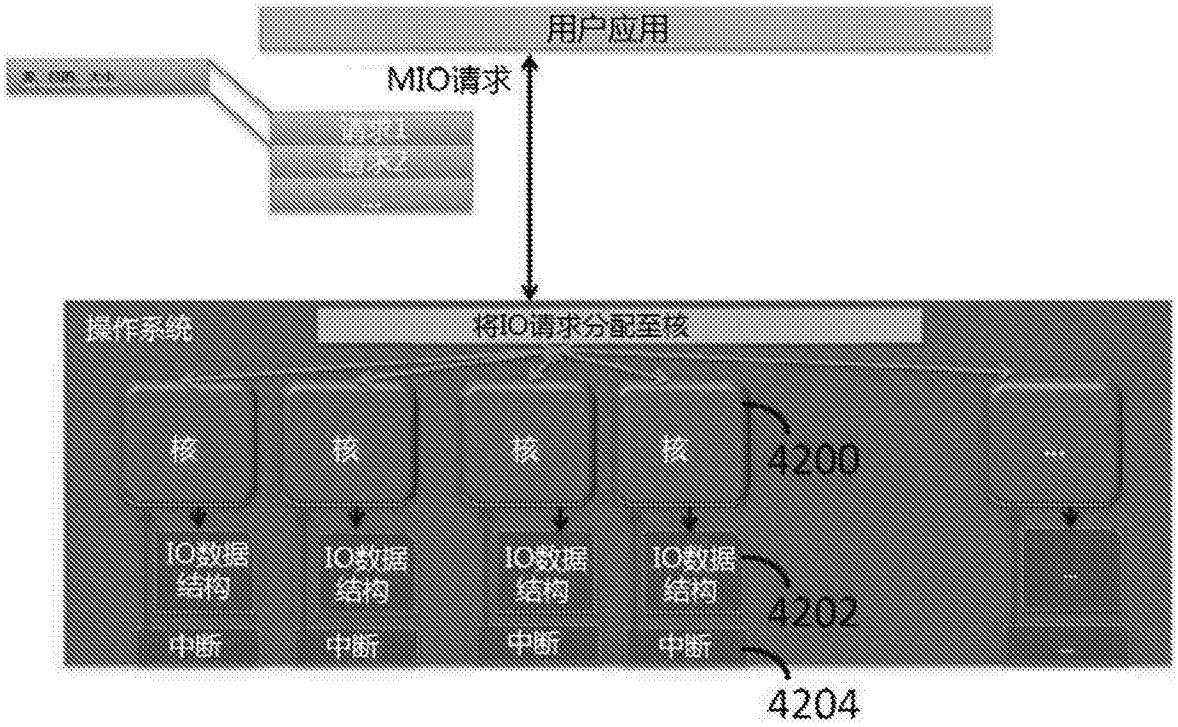

用户应用
MIO请求
操作系统
将IO请求分配至核
核
核
核
核
4200
IO数据结构
IO数据结构
IO数据结构
IO数据结构
4202
中断
中断
中断
中断
4204

图42

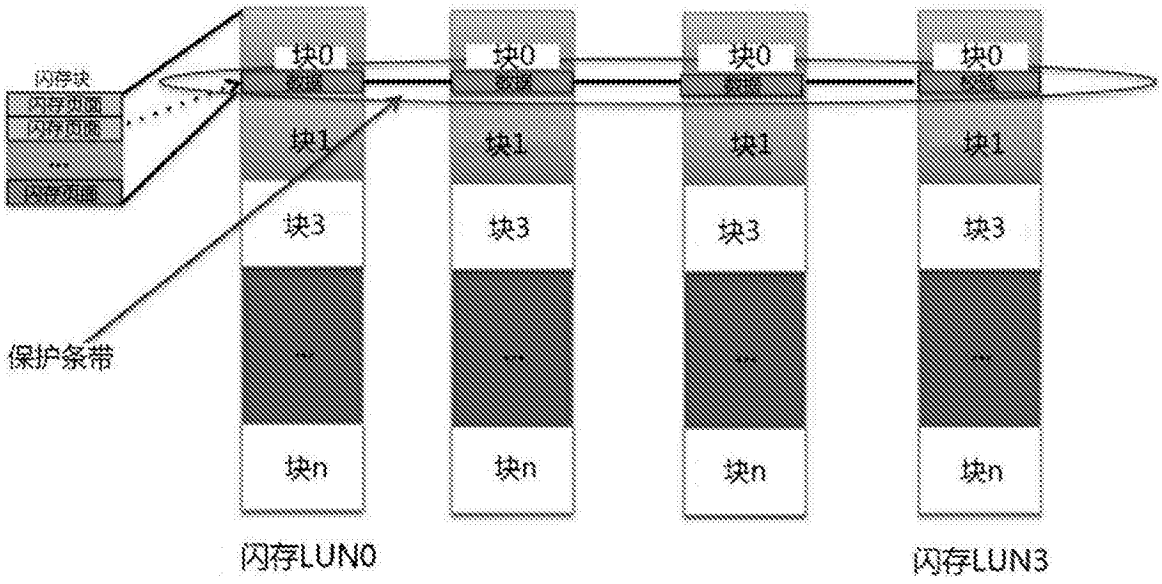

闪存块
闪存页面
闪存页面
闪存页面
保护条带
块0
块1
块3
块n
闪存LUN0
闪存LUN3

图43

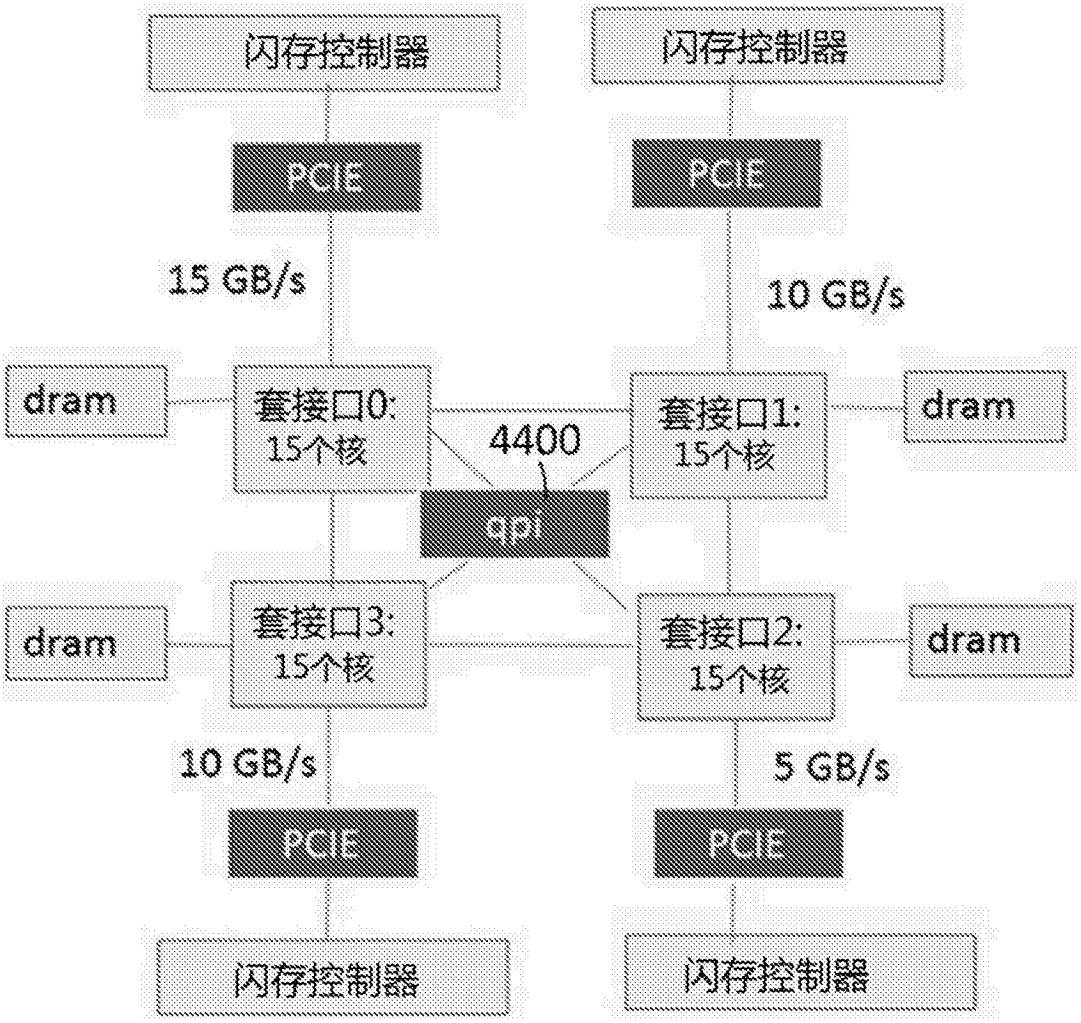

闪存控制器
闪存控制器
PCIE
PCIE
15 GB/s
10 GB/s
dram
套接口0:
15个核
4400
套接口1:
15个核
dram
qpi
dram
套接口3:
15个核
套接口2:
15个核
dram
10 GB/s
5 GB/s
PCIE
PCIE
闪存控制器
闪存控制器

图44

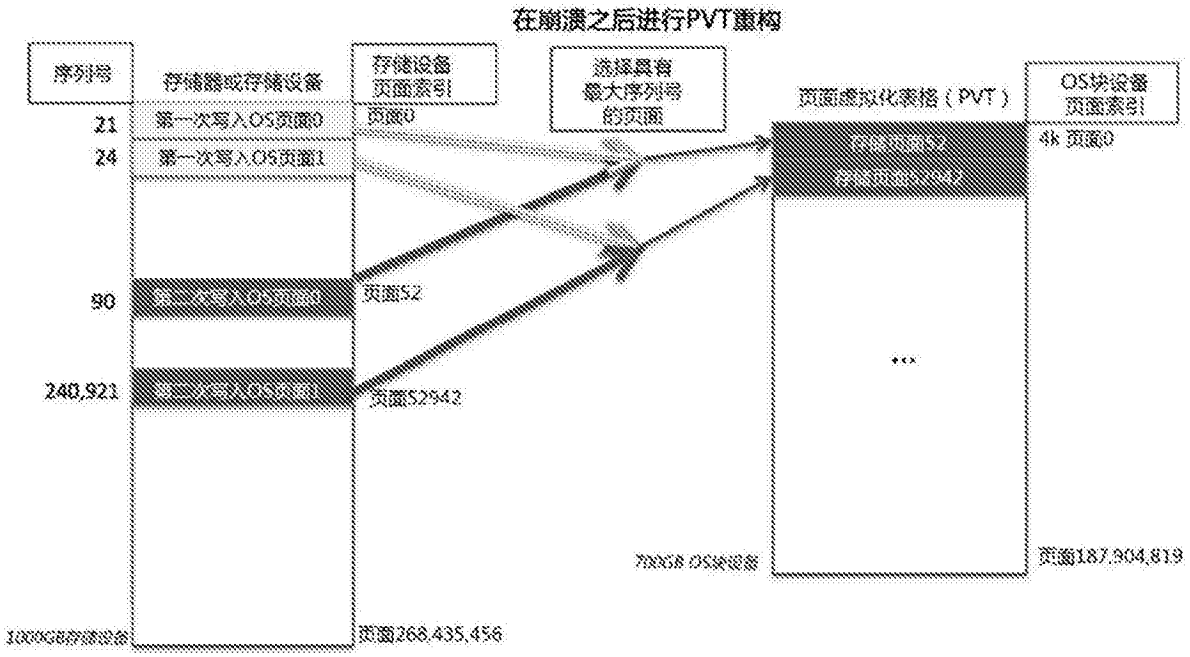

在崩溃之后进行PVT重构
序列号
存储器或存储设备
存储设备
页面索引
选择具有
最大序列号
的页面
页面虚拟化表格（PVT）
OS块设备
页面索引
21
第一次写入OS页面0
页面0
24
第一次写入OS页面1
4k 页面0
90
页面52
240,921
页面52942
...
页面187,904,819
页面268,435,456

图45

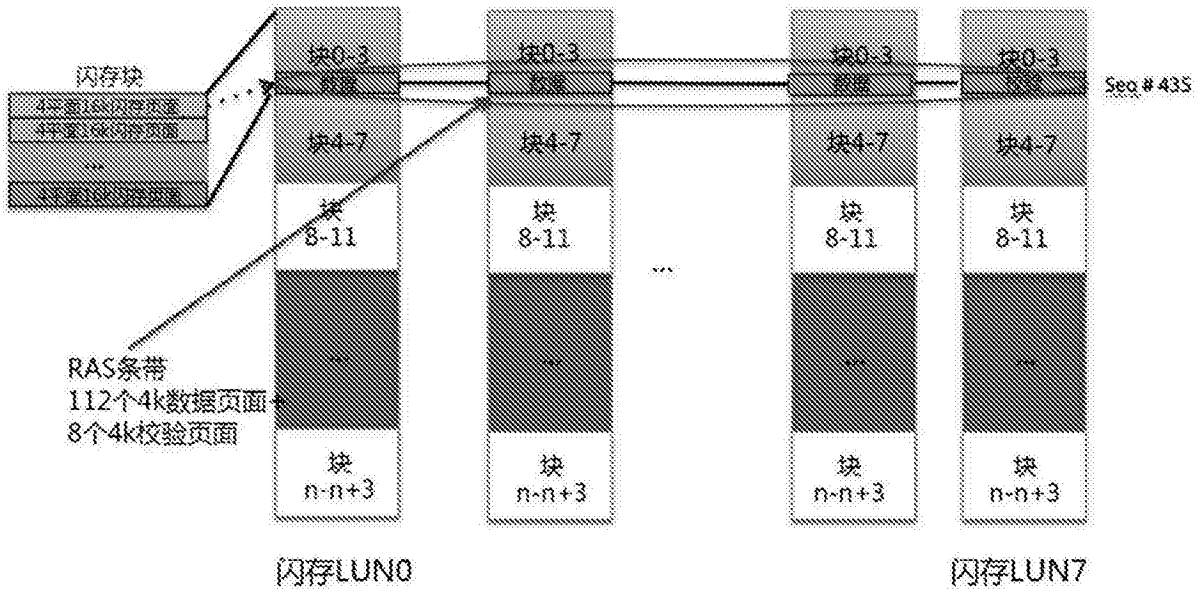

闪存块
块0-3
块4-7
块
8-11
...
块
n-n+3
Seq # 435
RAS条带
112个4k数据页面+
8个4k校验页面
闪存LUN0
闪存LUN7

图46

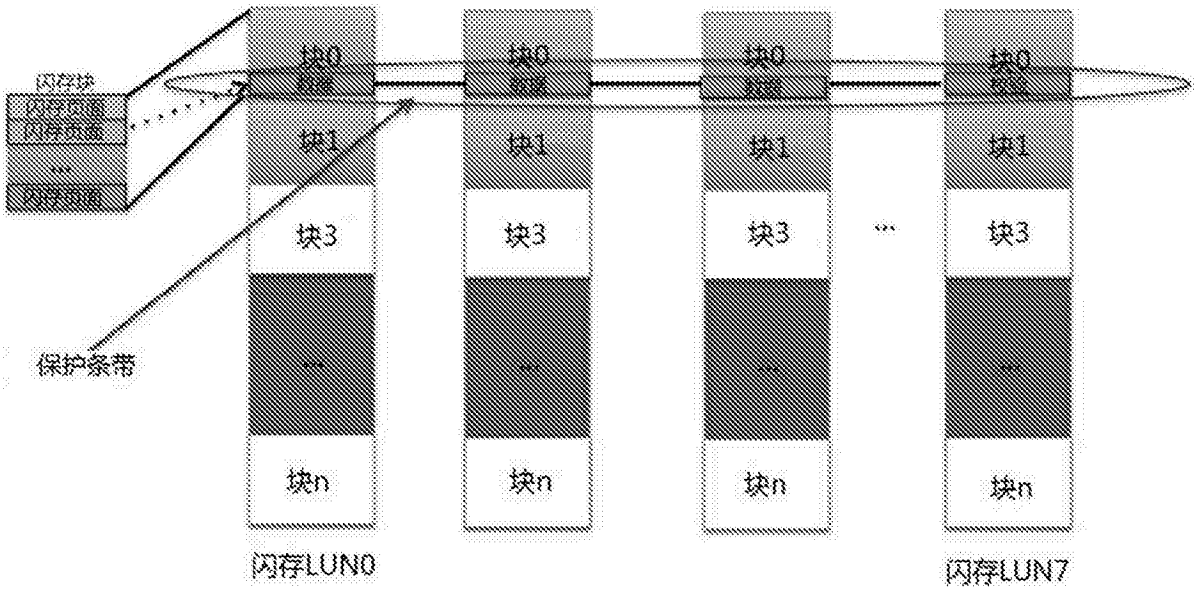

闪存块
闪存页面
闪存页面
闪存页面
保护条带
块0
数据
块1
块3
...
块n
闪存LUN0
闪存LUN7

图47

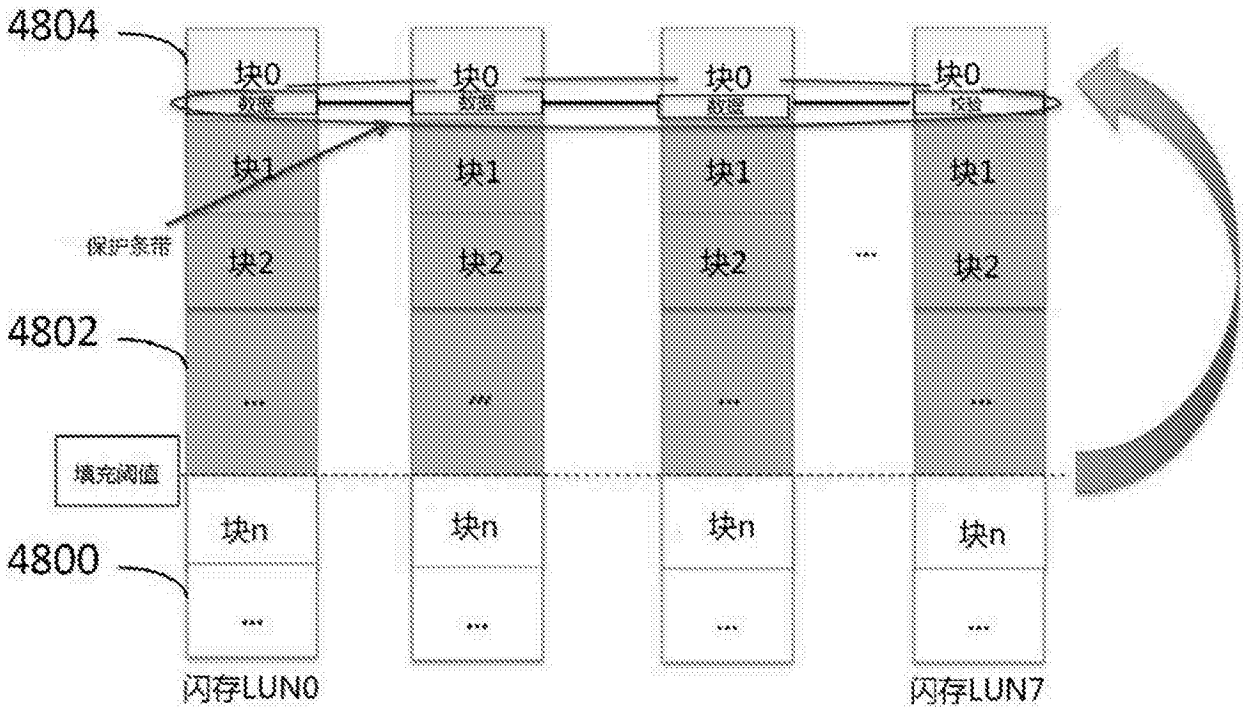

4804
4802
4800
保护条带
块0
数据
块1
块2
...
填充阈值
块n
闪存LUN0
闪存LUN7

图48

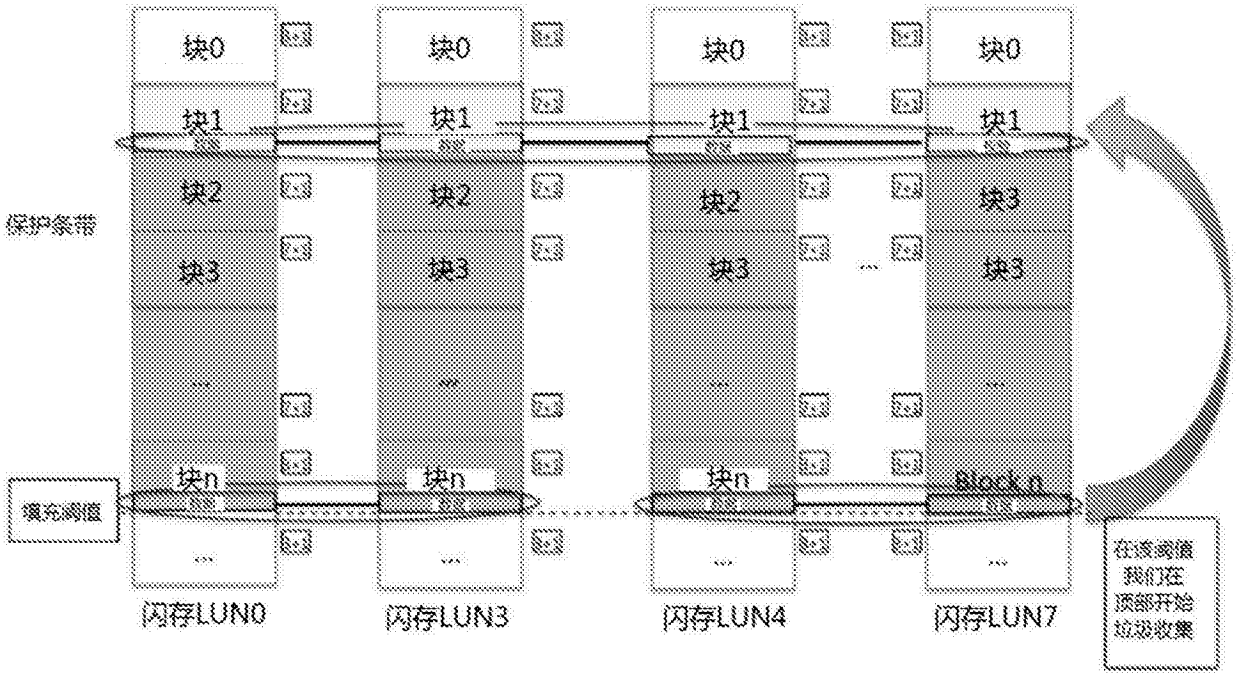

块0
块1
块2
块3
...
块n
保护条带
填充阈值
闪存LUN0
闪存LUN3
闪存LUN4
闪存LUN7
Block n
在该阈值
我们在
顶部开始
垃圾收集

图49

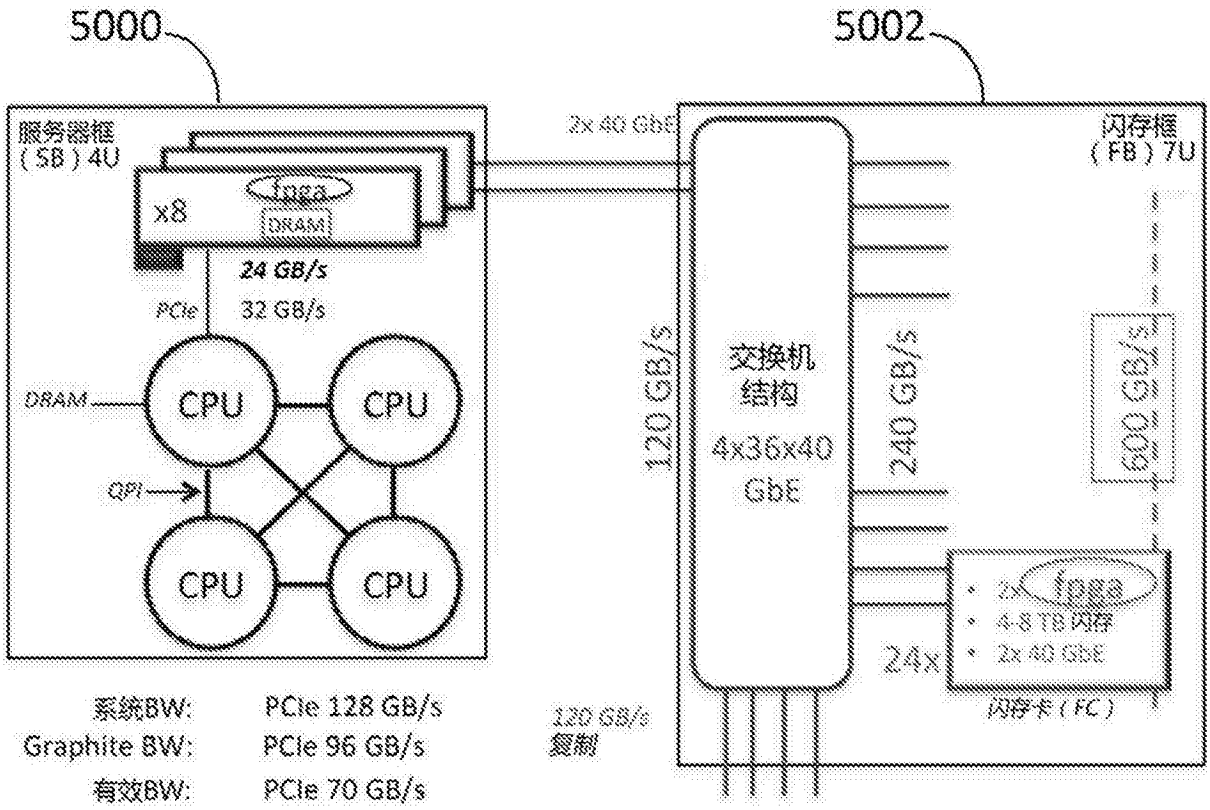

5000
5002
服务器框
(5B)4U
x8
fpga
DRAM
24 GB/s
PCIe
32 GB/s
CPU
CPU
CPU
CPU
DRAM
QPI
2x 40 GbE
120 GB/s
交换机
结构
4x36x40
GbE
240 GB/s
闪存框
(FB)7U
600 GB/s
24x
4-8 TB 闪存
2x 40 GbE
闪存卡(FC)
120 GB/s
复制
系统BW: PCIe 128 GB/s
Graphite BW: PCIe 96 GB/s
有效BW: PCIe 70 GB/s

图50

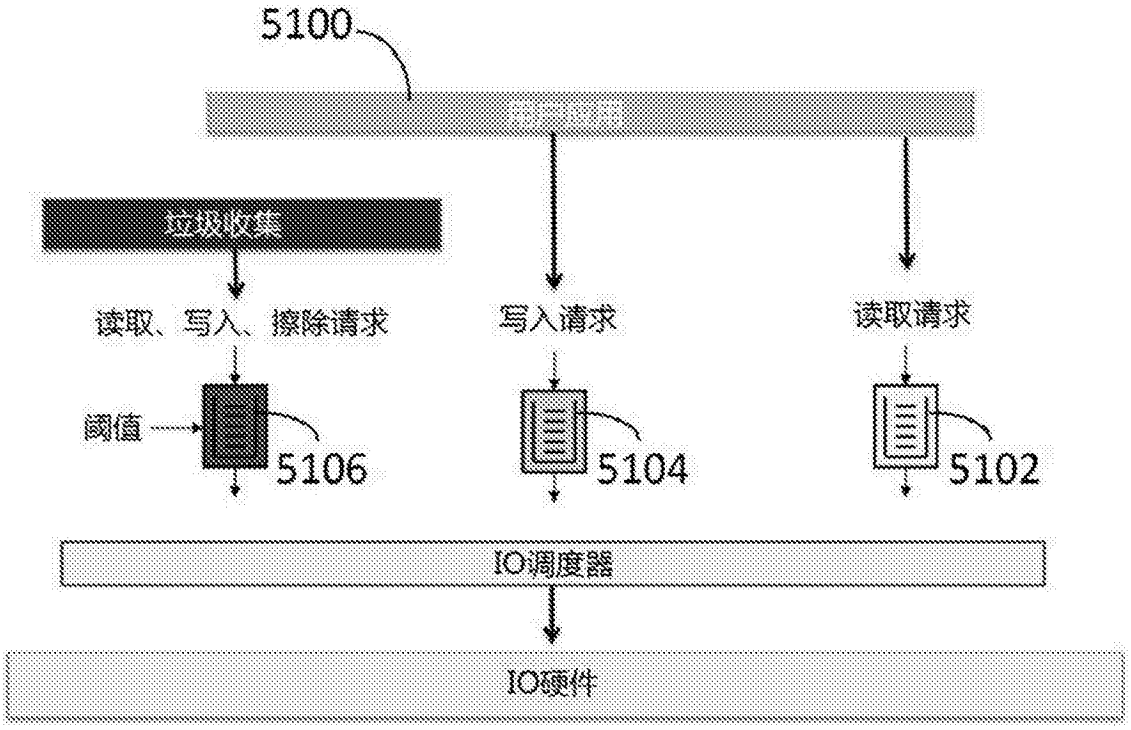

5100
垃圾收集
读取、写入、擦除请求
写入请求
读取请求
阈值
5106
5104
5102
IO调度器
IO硬件

图51

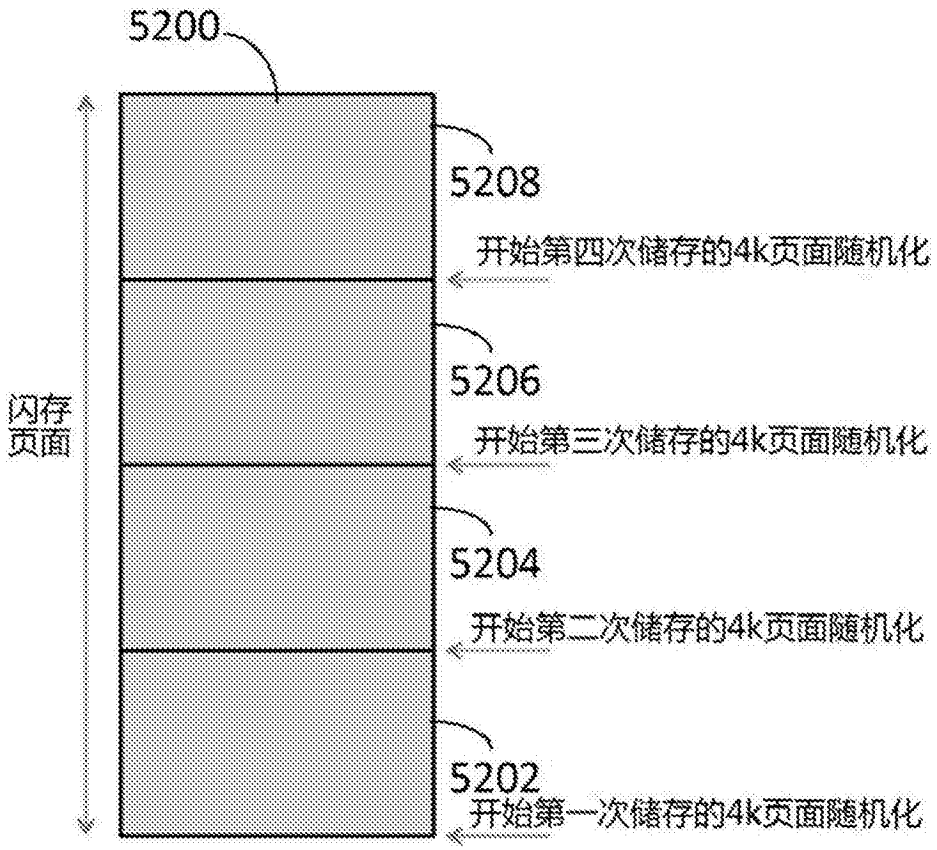

5200
5208
开始第四次储存的4k页面随机化
5206
开始第三次储存的4k页面随机化
闪存
页面
5204
开始第二次储存的4k页面随机化
5202
开始第一次储存的4k页面随机化

图52

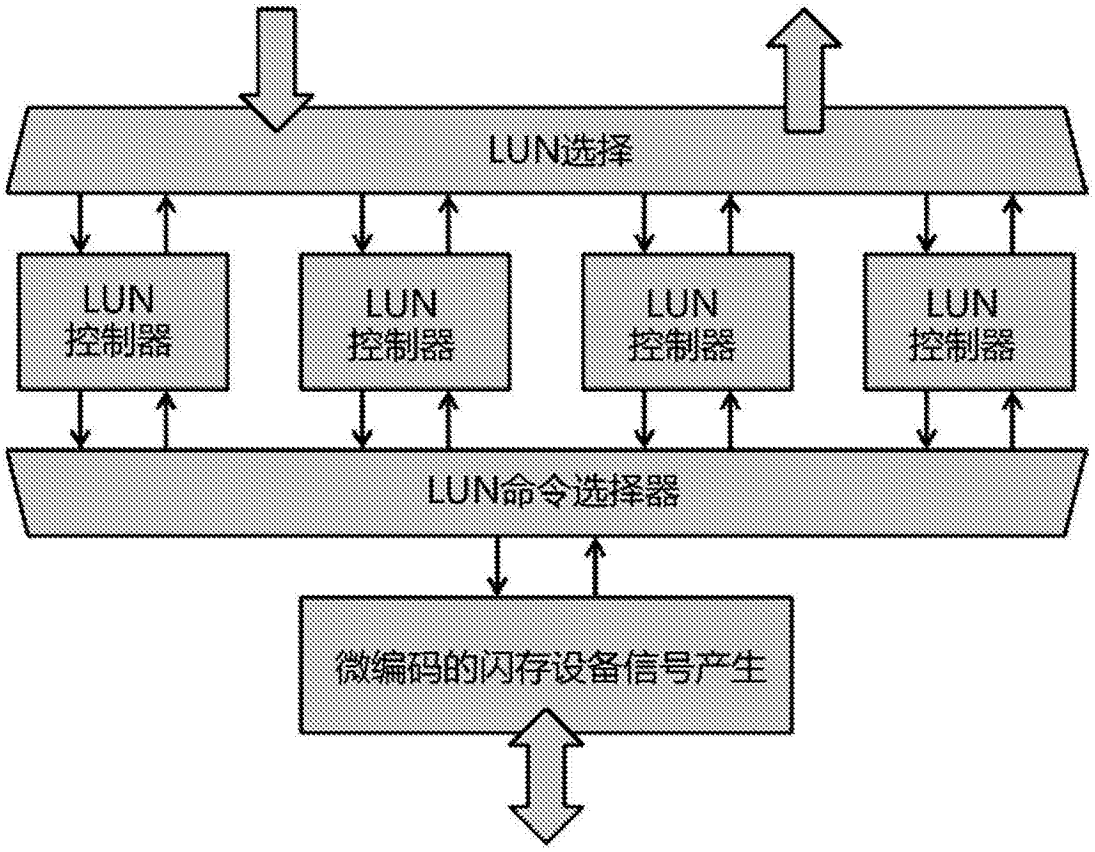


图53

擦除2	写入1	写入2	读取
开始擦除	开始写入	开始写入	开始读取
等待准备好	等待准备好	等待准备好	等待准备好
开始擦除		开始写入	完成读取
等待准备好		等待准备好	
得到状态		得到状态	

图54

开始擦除	得到状态	开始写入	开始读取	完成读取	重置	设置特征
CMD-60	CMD-78	CMD-80	CMD-00	CMD-06	CMD-FF	CMD-EF
ADDR-A1	ADDR-A1	ADDR-A1	ADDR-A1	ADDR-A1		ADDR-01
ADDR-A2	ADDR-A2	ADDR-A2	ADDR-A2	ADDR-A2		Din
ADDR-A3	ADDR-A3	ADDR-A3	ADDR-A3	ADDR-A3		Din
CMD-D0	Dout	ADDR-A4	ADDR-A4	ADDR-A4		Din
		ADDR-A5	ADDR-A5	ADDR-A5		Din
		Din (4k)	CMD-30	CMD-E0		
		CMD-10		Dout (4k)		

图55

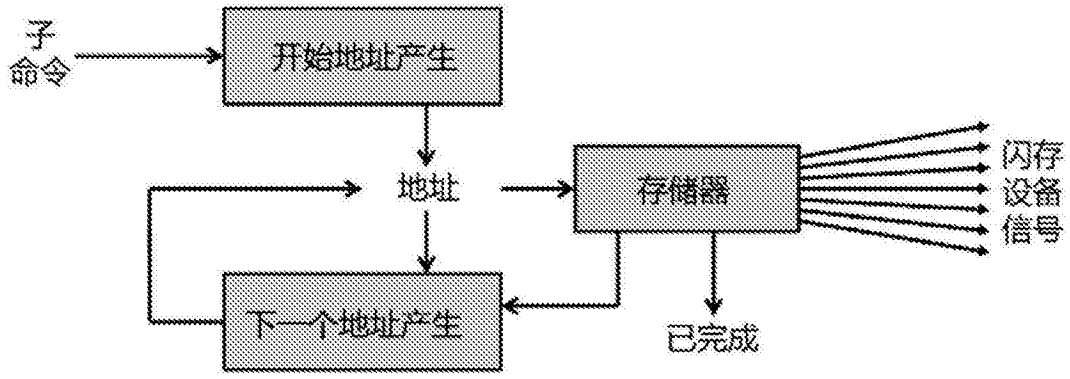


图56

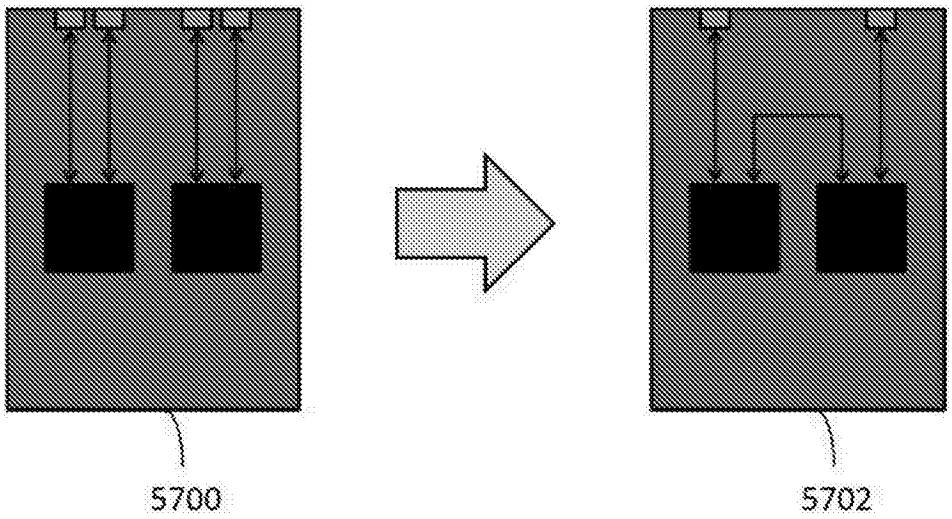


图57