(19)国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 118092796 A
(43)申请公布日 2024.05.28

(21)申请号 202410210025.8

(22)申请日 2018.10.05

(30)优先权数据
62/585,062 2017.11.13 US
16/121,938 2018.09.05 US

(62)分案原申请数据
201880086379.9 2018.10.05

(71)申请人 维卡艾欧有限公司
地址 以色列特拉维夫

(72)发明人 马奥尔·本-达扬
奥姆里·帕尔蒙 利兰·兹维贝
卡纳尔·阿迪蒂

(74)专利代理机构 北京康信知识产权代理有限责任公司 11240
专利代理师 刘彬

(51)Int.Cl.
G06F 3/06(2006.01)
G06F 16/13(2019.01)

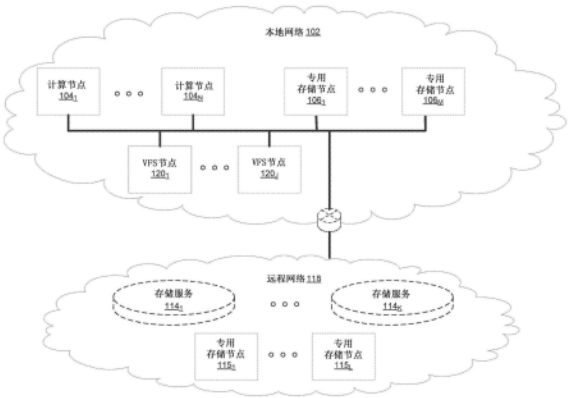
权利要求书2页 说明书14页 附图10页

(54)发明名称

分布式存储系统的目录结构

(57)摘要

本发明公开了分布式存储系统的目录结构。多个计算设备经由网络可通信地彼此耦接,并且多个计算设备中的每一个能够操作地耦接至快闪存储设备中的一个或多个。每个计算设备能够操作以访问快闪存储设备内的一个或多个存储块,并维护用于管理对存储器的访问的目录结构。可以根据添加或移除存储在存储器中的一个或多个相关文件来自适应地调整目录结构的大小。



CN 118092796 A

1.一种系统,包括:

计算设备,通信地耦接到包括多个存储块的存储网络,其中,所述计算设备被配置为:

与所述存储网络通信;

维护目录结构,所述目录结构包括哈希集,用于管理对所述多个存储块中的一个或多个存储块的访问;

根据所述一个或多个存储块的访问自适应地调整所述目录结构大小;以及

将一部分所述目录结构重新分布至另一计算设备以放大所述哈希集。

2.根据权利要求1所述的系统,其中,所述多个存储块包括非易失性存储器。

3.根据权利要求1所述的系统,其中,所述访问包括添加或删除与所述目录结构相关联的文件。

4.根据权利要求1所述的系统,其中:

所述目录结构包括一个或多个注册表块,

每个注册表块包括一个或多个键-值条目,并且

每个键-值条目对应于一个或多个文件。

5.根据权利要求4所述的系统,其中,键-值条目添加到所述一个或多个注册表块的注册表块中,直到键-值条目的数量超过预定容量为止,在键-值条目的数量超过所述预定容量时,所述注册表块被拆分并且新的注册表块被添加到所述一个或多个注册表块。

6.根据权利要求4所述的系统,其中,从所述一个或多个注册表块的注册表块中移除键-值条目,直到键-值条目的数量等于或低于预定级别为止,在键-值条目的数量等于或低于所述预定级别时,将所述注册表块与所述一个或多个注册表块中的另一个注册表块合并。

7.根据权利要求1所述的系统,其中,所述存储网络包括跨多个存储设备分布的故障弹性地址空间。

8.根据权利要求7所述的系统,其中,每个注册表块由一个或多个自适应索引标识。

9.一种方法,包括:

将计算设备通信地耦接到包括多个存储块的存储网络;

维护目录结构,用于管理对所述多个存储块中的一个或多个存储块的访问,其中,所述目录结构包括哈希集;

根据所述多个存储块中的所述一个或多个存储块的访问自适应地调整所述目录结构大小;以及

将一部分所述目录结构重新分布至另一计算设备以放大所述哈希集。

10.根据权利要求9所述的方法,其中,访问一个或多个存储块包括:添加或删除与所述目录结构相关联的文件。

11.根据权利要求9所述的方法,其中:

所述目录结构包括一个或多个注册表块,

每个注册表块包括一个或多个键-值条目,并且

每个键-值条目对应于一个或多个文件。

12.根据权利要求11所述的方法,其中,所述方法包括:

将键-值条目添加到所述一个或多个注册表块的注册表块中,以及

如果键-值条目的数量超过预定容量,则拆分所述注册表块并将新的注册表块添加到所述一个或多个注册表块。

13.根据权利要求11所述的方法,其中,所述方法包括:

从所述一个或多个注册表块的注册表块中移除键-值条目;以及

如果键-值条目的数量等于或低于预定级别,则将所述注册表块与所述一个或多个注册表块中的另一个注册表块合并。

14.根据权利要求9所述的方法,其中,所述存储网络包括跨多个存储设备分布的故障弹性地址空间。

15.根据权利要求14所述的方法,其中,每个注册表块由一个或多个自适应索引标识。

分布式存储系统的目录结构

[0001] 本申请是分案申请,其母案申请的申请号为201880086379.9,申请日为2018年10月5日,发明名称为“分布式存储系统的目录结构”。

[0002] 优先权要求

[0003] 本申请要求2017年11月13日提交的标题为“Directory Structure For A Distributed Storage System”的美国临时专利申请62/585,062以及2018年9月5日提交的标题为“Directory Structure For A Distributed Storage System”的美国专利申请第16/121,938的优先权。

背景技术

[0004] 通过将这样的方法与参照附图在本公开的其余部分中阐述的本方法和系统的一些方面进行比较,数据存储的常规方法的局限性和缺点对于本领域技术人员将变得显而易见。

[0005] 相关申请的交叉引用

[0006] 题为“Distributed Erasure Coded Virtual File System”的美国专利申请号15/243,519的全文通过将其整体引入而结合于本文中。

发明内容

[0007] 提供了用于分布式存储系统中的目录结构的方法和系统,该方法和系统基本上由至少一个附图示出和/或结合至少一个附图进行了描述,并在权利要求中更完整地阐述。

附图说明

[0008] 图1示出根据本公开的方面的虚拟文件系统的各种示例配置。

[0009] 图2示出根据本公开的方面的虚拟文件系统节点的示例配置。

[0010] 图3示出根据本公开的示例实现方式的虚拟文件系统的另一种表示。

[0011] 图4示出根据本公开的示例实现的使用盘上哈希来拆分快闪存储器注册表的块的示例。

[0012] 图5示出根据本公开的示例实现的使用盘上哈希来拆分快闪存储器注册表的存储桶的示例。

[0013] 图6A示出根据本公开的示例实现的使用盘上哈希来合并快闪存储器注册表的块的示例。

[0014] 图6B示出根据本公开的示例实现的从目录读取操作返回的条目的示例。

[0015] 图7是示出根据本公开的示例实现的用于利用盘上哈希添加快闪存储器注册表的块的方法的流程图。

[0016] 图8是示出根据本公开的示例实现的用于利用盘上哈希来访问快闪存储器注册表的块的方法的流程图。

[0017] 图9示出其中两个分布式故障弹性地址空间驻留在多个固态存储盘上的示例实现

方式。

[0018] 图10示出了根据本公开的示例实现方式的前向纠错方案,其可以用于保护存储到虚拟文件系统的非易失性存储器的数据。

具体实施方式

[0019] 本公开中的系统适用于小型集群,并且还可以扩展到许多成千上万个节点。讨论了关于非易失性存储器(NVM)(例如,以固态驱动器(SSD)形式出现的闪存)的示例实施例。NVM可以分为4kB“块”(block)和128MB“组块”(chunk)。“扩展区”(extent)也可以存储在易失性存储器中,例如用于快速访问的RAM,也可以由NVM存储备份。扩展区可以存储块的指针,例如,指向存储在块中的1MB数据的256个指针。在其他实施例中,也可以使用更大或更小的存储器划分。本公开中的元数据功能可以有效地分布在许多服务器上。例如,在大的负荷指向文件系统名称空间的特定部分的“热点”的情况下,则此负荷可以分布在多个节点上。

[0020] 图1示出根据本公开的方面的虚拟文件系统(VFS)的各种示例配置。图1中所示的是局域网(LAN)102,其包括一个或多个VFS节点120(由1至J的整数索引,$j \geq 1$),并且可选地包括(虚线所示):一个或多个专用存储节点106(由1至M的整数索引,$M \geq 1$);一个或多个计算节点104(由1至N的整数索引,$N \geq 1$);和/或将LAN 102连接到远程网络118的边缘路由器。远程网络118可选地包括一个或多个存储服务114(由1至K的整数索引,$K \geq 1$);和/或一个或多个专用存储节点115(由1至L的整数索引,对于$L \geq 1$)。

[0021] 每个VFS节点120_j(j为整数,其中$1 \leq j \leq J$)是网络计算设备(例如,服务器,个人计算机等),其包括用于运行VFS进程以及可选的客户端进程的电路(或者直接在设备104_n的操作系统上和/或在设备104_n中运行的一个或多个虚拟机中)。

[0022] 计算节点104是可以在没有VFS后端的情况下运行VFS前端的联网设备。计算节点104可以通过将SR-IOV放入NIC并使用完整的处理器内核来运行VFS前端。备选地,计算节点104可以通过经由Linux内核联网堆栈路由联网并且使用内核进程调度来运行VFS前端,因此不具有完整内核的要求。如果用户不想为VFS分配完整的核心,或者联网硬件与VFS要求不兼容,这将很有用。

[0023] 图2示出根据本公开的方面的VFS节点的示例配置。VFS节点包括VFS前端202和驱动器208、VFS存储器控制器204、VFS后端206和VFS SSD代理214。如本公开中所使用的,“VFS进程”是实现以下中的一个或多个的进程:VFS前端202、VFS存储器控制器204、VFS后端206和VFS SSD代理214。因此,在示例实现方式中,可以在客户端进程和VFS进程之间共享VFS节点的资源(例如,处理和存储器资源)。VFS的进程可以配置为要求相对少量的资源,以最大程度地减少对客户端应用程序的性能的影响。VFS前端202、VFS存储器控制器204和/或VFS后端206和/或VFS SSD代理214可以在主机201的处理器上或在网络适配器218的处理器上运行。对于多核处理器,不同的VFS进程可以在不同的内核上运行,并且可以运行服务的不同子集。从客户端进程212的角度来看,与虚拟文件系统的接口独立于运行VFS进程的特定物理机器。客户端进程只需要驱动器208和前端202存在就能为它们提供服务。

[0024] VFS节点可以被实现为直接在操作系统上运行的单个租户服务器(例如,裸机),或者被实现为裸机服务器中的虚拟机(VM)和/或容器(例如Linux容器(LXC))。VFS可以在LXC

容器中作为VM环境运行。因此,在VM内部,仅包含VFS的LXC容器可以运行。在经典的裸机环境中,存在用户空间应用程序,并且VFS在LXC容器中运行。如果服务器正在运行其他容器化的应用程序,则VFS可能在容器部署环境(例如Docker)的管理范围之外的LXC容器内运行。

[0025] VFS节点可以由操作系统和/或虚拟机监视器(VMM)(例如,管理程序)服务。VMM可以用于在主机201上创建并运行VFS节点。多个内核可以驻留在运行VFS的单个LXC容器内,并且VFS可以使用单个Linux内核在单个主机201上运行。因此,单个主机201可以包括多个VFS前端202、多个VFS存储器控制器204、多个VFS后端206和/或一个或多个VFS驱动器208。VFS驱动器208可以在LXC容器的范围之外的内核空间中运行。

[0026] 单个根输入/输出虚拟化(SR-IOV)PCIe虚拟功能可用于在用户空间222中运行网络堆栈210。SR-IOV允许隔离PCI Express,从而可以在虚拟环境上共享单个物理PCI Express,并且可以向单个物理服务器计算机上的不同虚拟组件提供不同的虚拟功能。I/O堆栈210使VFS节点能够绕过标准TCP/IP堆栈220并直接与网络适配器218通信。可以通过无锁队列将用于uniX(POSIX)VFS功能的便携式操作系统接口提供给VFS驱动器208。SR-IOV或完整的PCIe物理功能地址也可以用于在用户空间222中运行非易失性存储器Express(NVMe)驱动器214,从而完全绕开Linux IO堆栈。NVMe可以用于访问通过PCI Express(PCIe)总线连接的非易失性存储介质216。非易失性存储介质216例如可以是固态驱动器(SSD)形式的闪存或固态硬盘(SSD)或内存模块(DIMM)形式的存储类存储器(SCM)。其他示例可能包括存储类存储器技术,例如3D-XPoint。

[0027] 通过将物理SSD 216与SSD代理214和联网210耦接,可以将SSD实现为联网设备。可替代地,可以通过使用诸如NVMe-oF(基于结构的NVMe)的网络协议将SSD实现为附接网络的NVMe SSD 242或244。NVMe-oF可以允许使用冗余网络链路访问NVMe设备,从而提供更高级别或弹性。网络适配器226、228、230和232可以包含用于连接到NVMe SSD 242和244的硬件加速,以将它们转换为联网的NVMe-oF设备而无需使用服务器。NVMe SSD 242和244可以各自包括两个物理端口,并且可以通过这些端口中的任何一个来访问所有数据。

[0028] 每个客户端进程/应用程序212可以直接在操作系统上运行,或者可以在由操作系统和/或管理程序服务的虚拟机和/或容器中运行。客户端进程212可以在执行其主要功能的过程中从存储器读取数据和/或将数据写入存储器中。然而,客户端进程212的主要功能与存储无关(即,该过程仅关注其数据被可靠地存储并且在需要时可被检索,而不关注数据的存储位置、时间或方式)。引起这种过程的示例应用程序包括:电子邮件服务器、Web服务器、办公效率应用程序、客户关系管理(CRM)、动画视频渲染、基因组计算、芯片设计、软件构建和企业资源计划(ERP)。

[0029] 客户端应用程序212可以对与VFS驱动器208通信的内核224进行系统调用。VFS驱动器208将相应的请求放在VFS前端202的队列上。如果存在几个VFS前端,则驱动器可能会负荷均衡地对不同的前端进行访问,以确保始终通过同一前端访问单个文件/目录。这可以通过基于文件或目录的ID对前端进行“碎片化”(sharding)来完成。VFS前端202提供接口以用于基于负责该操作的存储桶将文件系统请求路由到适当的VFS后端。适当的VFS后端可以在同一主机上,也可以在另一主机上。

[0030] VFS后端206托管多个存储桶,其中每个存储桶服务于它接收的文件系统请求,并执行任务来另外地管理虚拟文件系统(例如,负荷平衡、日志记录、维护元数据、缓存、在层

之间移动数据、删除过时数据、更正损坏的数据等)。

[0031] VFS SSD代理214处理与相应的存储设备216的交互。这可以包括例如转换地址,以及生成发布给存储设备的命令(例如,在SATA、SAS、PCIe或其他合适的总线上)。因此,VFS SSD代理214用作虚拟文件系统的VFS后端206和存储设备216之间的中介。SSD代理214也可以与支持标准协议的标准网络存储设备通信,标准协议例如NVMe-oF(基于结构的NVMe)。

[0032] 图3示出根据本公开的示例实现方式的虚拟文件系统的另一种表示。在图3中,元件302表示虚拟文件系统驻留在的各个节点(计算,存储和/或VFS)的存储器资源(例如DRAM和/或其他短期内存)和处理(例如,x86处理器、ARM处理器、NIC、ASIC)资源,例如,如上关于图2所描述的。元件308表示提供虚拟文件系统的长期存储的一个或多个物理存储设备216。

[0033] 如图3所示,物理存储被组织为多个分布式故障弹性地址空间(DFRAS)518。每个分布式故障弹性地址空间包括多个组块310,组块310又包括多个块312。将块312组织成组块310仅是在一些实现中的便利,并且可能并非在所有实现中都进行。每个块312存储提交的数据316(其可以采取下面讨论的各种状态)和/或描述或引用提交的(committed)数据316的元数据314。

[0034] 将存储设备308组织成多个DFRAS使得能够从虚拟文件系统的许多(可能是全部)节点进行高性能并行提交(例如,图1的所有节点104_1至104_N,106_1至106_M和120_1至120_J可以并行执行并发提交)。在一个示例实现方式中,虚拟文件系统的每个节点可以拥有多个DFRAS中的相应一个或多个,并对其拥有的DFRAS具有独占读取/提交访问权限。

[0035] 每个存储桶拥有DFRAS,并因此在对其进行写入时不需要与任何其他节点进行协调。每个存储桶可能会在许多不同的SSD上的许多不同组块上建立条带,因此每个存储桶及其DFRAS可以基于许多参数选择当前要写入的“组块条带”,并且一旦将组块分配给该存储桶,就无需进行协调就可以进行写入。所有存储桶都可以有效地写入所有SSD,而无需协调。

[0036] 每个DFRAS仅由在特定节点上运行的其所有者存储桶所拥有和访问,从而允许VFS的每个节点控制存储设备308的一部分,而不必与任何其他节点进行协调(在初始化期间或在节点故障之后(重新)分配持有DFRAS的存储桶期间除外,例如,这可以与存储设备308的实际读取/提交操作异步执行)。因此,在这样的实现方式中,每个节点可以独立于其他节点正在做什么而读取/提交到其存储桶的DFRAS,而在读取并提交到存储设备308时不需要达成任何共识。此外,在特定节点发生故障的情况下,该特定节点拥有多个存储桶的事实允许更智能且更有效地将其工作负荷重新分配给其他节点(而不是将整个工作负荷分配给单个节点,可能会创建“热点”)。就这一点而言,在一些实现方式中,存储桶的数量相对于系统中的节点的数量可能较大,使得任何一个存储桶可能是放置在另一节点上的相对较小的负荷。这允许根据其他节点的能力和容量对发生故障的节点的负荷进行细粒度的重新分配(例如,具有更多功能和容量的节点可能会获得故障节点存储桶的较高百分比)。

[0037] 为了允许这种操作,可以保持元数据,该元数据将每个存储桶映射到其当前拥有的节点,使得可以将对存储设备308的读取和提交重定向到适当的节点。

[0038] 负荷分配是可能的,因为整个文件系统元数据空间(例如,目录、文件属性、文件中的内容范围等)可以分解(例如,切碎或碎片化)成均匀的小片(例如,“碎片”)。例如,具有30k服务器的大型系统可以将元数据空间分成128k或256k的碎片。

[0039] 每个这样的元数据碎片可以被保存在“存储桶”中。每个VFS节点可能负责几个存

储桶。当存储桶在给定的后端上正在服务元数据碎片时,该存储桶被视为该存储桶的“活动”或“领导者”(leader)。通常,存储桶比VFS节点多。例如,有6个节点的小型系统可以有120个存储桶,而有1000个节点的大型系统可以有8k个存储桶。

[0040] 每个存储桶可以在一小群节点上活动,通常5个节点组成该存储桶的五元组。集群配置使所有参与节点都保持关于每个存储桶的五元组分配的最新信息。

[0041] 每个五元组监视自己。例如,如果集群中有1万台服务器,并且每个服务器有6个存储桶,则每个服务器将只需要与30台不同的服务器进行通信即可维护其存储桶的状态(6个存储桶将具有6个五元组,因此6×5=30)。这比集中式实体必须监视所有节点并保持集群范围的状态要少得多。使用五元组可以使性能随着更大的集群而扩展,因为当集群大小增加时,节点不会执行更多的工作。这可能会带来一个缺点,即在“哑”模式下,一个小型集群实际上可以产生比物理节点更多的通信,但是可以通过它们共享所有存储桶而在两台服务器之间仅发送单个心跳来克服此缺点(随着集群的增长,这将更改为仅一个存储桶,但是如果您有一个5个服务器的小型集群,则它将仅在所有消息中包括所有存储桶,而每个服务器将仅与其他4个服务器通信)。五元组可以使用类似于Raft共识算法的算法进行决策(即达成共识)。

[0042] 每个存储桶可能都有可以运行它的一组计算节点。例如,五个VFS节点可以运行一个存储桶。但是,在任何给定时刻,组中只有一个节点是控制器/领导者。此外,对于足够大的集群,没有两个存储桶共享同一组。如果集群中只有5个或6个节点,则大多数存储桶可以共享后端。在一个相当大的集群中,可能有许多不同的节点组。例如,在26个节点的情况下,有超过$64000\ (\frac{26!}{5!*(26-5)!})$个可能的五节点组(即五元组)。

[0043] 组中的所有节点都知道并同意(即达成共识)哪个节点是该存储桶的实际活动控制器(即领导者)。访问存储桶的节点可能会记住(“缓存”)组中(例如五个)成员中该存储桶的领导者的最后一个节点。如果它访问存储桶领导者,则存储桶领导者执行所请求的操作。如果它访问的节点不是当前领导者,则该节点指示领导者“重定向”访问。如果访问缓存的领导者节点超时,则联系节点可以尝试使用同一五元组的另一个节点。集群中的所有节点共享集群的公用“配置”,这使节点可以知道哪个服务器可以运行每个存储桶。

[0044] 每个存储桶可都有负荷/使用率值,该值指示文件系统上运行的应用程序对存储桶的使用程度。例如,即使使用的存储桶数量不平衡,具有11个使用率较低的存储桶的服务器节点也可以接收另一个元数据存储桶,以在具有9个使用率较高的存储桶的服务器之前运行。可以根据平均响应等待时间、并发运行的操作数、内存消耗或其他指标来确定负荷值。

[0045] 即使VFS节点没有故障,也可能会发生重新分发。如果系统根据跟踪的负荷指标识别出一个节点比其他节点更忙,则系统可以将其一个存储桶移动(即“故障转移”)到另一台不太繁忙的服务器。但是,在实际将存储桶重新定位到不同主机之前,可以通过转移写入和读取来实现负荷平衡。由于每次写入可能在由DFRAS决定的不同的节点组上结束,因此具有较高负荷的节点可能不会被选择在要写入数据的条带中。系统还可能选择不提供来自高负荷节点的读取。例如,可以执行“降级模式读取”,其中,从同一条带的其他块重构高负荷节点中的块。降级模式读取是通过同一条带中的其余节点执行的读取,并且通过故障保护来

重建数据。当读取延迟过高时,可能会执行降级模式读取,因为读取的发起者可能会认为该节点已关闭。如果负荷足够高以创建更高的读取延迟,则集群可以恢复为从其他节点读取该数据,并使用降级模式读取来重建所需的数据。

[0046] 每个存储桶管理其自己的分布式擦除编码实例(即,DFRAS 518),并且不需要与其他存储桶协作来执行读取或写入操作。可能有成千上万个并发的分布式擦除编码实例同时工作,每个实例用于不同的存储桶。这是扩展性能不可或缺的一部分,因为它可以有效地将任何大型文件系统划分为不需要协调的独立部分,从而无论扩展规模如何都可以提供高性能。

[0047] 每个存储桶处理属于其碎片的所有文件系统操作。例如,目录结构,文件属性和文件数据范围将属于特定存储桶的管辖范围。

[0048] 从任何前端完成的操作开始于找出哪个存储桶拥有该操作。然后确定该存储桶的后端领导者和节点。可以通过尝试最近知道的领导者来执行该确定。如果最近知道的领导者不是当前领导者,则该节点可能知道哪个节点是当前领导者。如果最近知道的领导者不再是存储桶的五元组的一部分,则该后端将让前端知道应该回到配置中以找到存储桶的五元组的成员。操作的分布允许复杂的操作由多个服务器而不是标准系统中的单个计算机处理。

[0049] 盘上哈希(ODH,on-disk hash)可以管理在4k注册表页上实现的哈希表。这些哈希表可能具有各种大小的键和值(例如<KEY,VALUE>对,键-值条目),支持在存储桶内(以及跨存储桶)运行的许多其他更高级别的数据结构。

[0050] 所公开的系统可以支持存储在相同注册表上的相同种类的多个哈希表(具有不同的对象ID)或存储在相同注册表上的不同种类的哈希表。哈希可以基于密码哈希函数,并提供每种哈希类型和实例的唯一签名。ODH的注册表键可以基于哈希的类型,哈希实例的ID和块索引。因为注册表(包括例如块索引)存储在RAM中,所以寻找4k块并查明它是否存在是有效的。

[0051] ODH是在存储在注册表中的4k块上有效实现的可扩展(并且可折叠,collapsible)哈希。因此,使用的4k块数量与存储在ODH上的数据量成正比。

[0052] ODH具有基于KEY和VALUE类型的动态结构,并将哈希信息存储为排序的<KEY,VALUE>数组,从而允许进行有效的二进制搜索。排序是基于键的密码哈希签名而不是键本身完成的,以确保排序标准在所有位上的均匀分布。因为哈希集消耗的能量可能比单个4k块中的消耗更多,所以需要一种用于扩展和折叠哈希集的机制。

[0053] 因为潜在地有成千上万的并发DFRAS并发工作,所以利用了可伸缩的目录结构。当前公开的用于分布式存储系统的目录结构被分成不需要集中协调的独立目录片段,从而不管规模如何都提供高性能。

[0054] 使用ODH实现目录结构。每个ODH块的注册表键将是类型=Dir,索引节点ID,块索引。ODH中的每个<KEY,VALUE>条目将对应于在文件名(即“键”)和索引节点ID(即“值”)上计算的哈希。因此,文件名的哈希用于查找索引节点ID(如果存在)。

[0055] 图4示出了根据本公开的示例实施方式的使用ODH来拆分快闪存储器注册表的块的示例。可以在每个4kB ODH注册表块中实现400个条目和每个条目10个字节(例如,键两个字节,值8个字节)的表。为了说明的目的,图4中的注册表块被示出为具有最多四个<KEY,

VALUE>条目。除了<KEY,VALUE>对的本地哈希表之外,每个ODH注册表块还可以包括指示索引和拆分级别的报头信息。拆分级别可以存储在块内与“键”(或块的哈希)分开。

[0056] 每个ODH块的元素都按键排序,并且可以一个接一个地序列化。

[0057] ODH注册表块存储在DFRAS上。设置/获取操作由ODH实现,ODH比DFRAS“高”了几层。ODH假定存在一个正常运行的注册表,该注册表基于每个此类页的索引提供存储ODH 4k数据的服务。当更新ODH页时,这也实现了写均衡。

[0058] 只要条目的数量适合于一个块内,该代码可以通过搜索单个块来查找ODH。使用类型和实例ID,ODH可以枚举块中的<KEY,VALUE>对,直到找到正确的对。

[0059] 当块填满时,它将增加拆分级别,并根据密码哈希将某些条目移动到新块中。条目按照键的哈希的最低有效位(LSB)移动到块。保留与索引匹配的LSB,而与新块(添加“1”)匹配的LSB移至新块。因为排序和移动基于键的哈希,并且此类哈希生成哈希值的均匀分布,所以大约一半的条目将保留在原始块中,而一半的条目将被迁移到新创建的块中。

[0060] 在没有处于拆分级别0<索引=NIL,拆分=1>的哈希索引的情况下,可以使用一个注册表块401A创建ODH。前四个ODH条目中的键的LSB为0010、1010、1110和1111,且对应的值分别为value_x0,value_x1,value_x2和value_x3。

[0061] 当拆分块时,添加新块,并且现有块不改变其地址。这维护了用于查找数据的“锚点”。拆分后,锚块是索引以“0”开头的块,因为当将前导“0”添加到数字时,该数字的值不会更改,因此该块的地址实际上不会更改。

[0062] 在图4中,当拆分注册表块401A时,新块402A添加了报头信息<索引=1b,拆分=1>。现有块401A中以“1”结尾的<KEY,VALUE>对被移动到新块402A中。现有块401A中以“0”结尾的<KEY,VALUE>对保留在哈希表中,并且报头信息从<索引=NIL,拆分=0>更改为<索引=0b,拆分=1>。为了说明的目的,在块401A被拆分之后,尽管该块的物理存储器地址没有改变,但将其指定为块401B。

[0063] 在块401A被拆分之后,ODH注册表块401B接收<KEY,VALUE>对(0000,value_y0)。因为注册表块401B已达到其容量,所以将注册表块401B拆分,并添加一个带有报头信息<索引=10b,拆分=2>的新块403A。现有块401B中以“10”结尾的<KEY,VALUE>对被移动到新块403A中。现有块401A中以“00”结尾的<KEY,VALUE>对保留在哈希表中,并且报头信息从<索引=0b,拆分=1>更改为<索引=00b,拆分=2>。为了说明的目的,在块401B被拆分之后,将其指定为块401C。

[0064] 在从块401A拆分之后,ODH注册表块402A接收<KEY,VALUE>对(0001,value_y1),(0101,value_y2)和(1011,value_y3)。因为注册表块402A已达到其容量,所以将注册表块402A拆分,并为新块404A添加报头信息<索引=11b,拆分=2>。现有块402A中以“11”结尾的<KEY,VALUE>对被移动到新块404A中。现有块401A中以“01”结尾的<KEY,VALUE>对保留在哈希表中,并且报头信息从<索引=1b,拆分=1>更改为<索引=01b,拆分=2>。为了说明的目的,在块402A被拆分之后,将其指定为块402B。

[0065] 在从块401B拆分后,ODH注册表块403A接收<KEY,VALUE>对(0110,value_z3)。因为注册表块403A已达到其容量,所以将注册表块403A拆分并添加新的块405A。现有块403a中的以“110”结尾的<KEY,VALUE>对被移动到带有报头信息<索引=110b,拆分=3>的新块405A中。现有块403A中以“010”结尾的<KEY,VALUE>对保留在哈希表中,并且报头信息从<索

引＝10b,拆分＝2>更改为<索引＝010b,拆分＝3>。为了说明的目的,在块403A被拆分之后,将其指定为块403B。

[0066] 在块401B被拆分之后,ODH注册表块401C接收<KEY,VALUE>对(0100,value_z0)和(1000,value_z1)。由于注册表块401C尚未达到其容量,因此不需要拆分。

[0067] 图4示出了由单个后端(和单个CPU内核)处理的单个存储桶中的ODH。因为目录还跨越多个存储桶和多个服务器,所以不能允许单个目录成为性能瓶颈。如果要使用单个存储桶(并因此使用单个注册表和该注册表中的单个ODH)来处理整个目录,则目录的大小以及该目录可以存储的活动量将受到限制。因此,跨越了多个存储桶的目录操作是高效的并且有效的。

[0068] 图5示出了根据本公开的示例实现的使用盘上哈希来拆分快闪存储器注册表的存储桶的示例。该算法的工作方式类似于每个存储桶中的单个ODH,但是它不仅具有一个拆分级别,而且具有两个拆分级别。内部拆分级别是上面讨论的拆分级别,而外部拆分级别是跨存储桶的拆分级别。外部拆分级别和内部拆分级别从0开始。

[0069] “目录片段”是存储桶中的目录的一部分。系统决定单个存储桶(例如1M)中可以存储多少个文件条目。当外部拆分增加时(即,将目录片段推到其他存储桶),内部拆分会减少。

[0070] 在关于图4所述的拆分之后,ODH包括块401C,块403B,块405A,块402B和块404A。这全部五个ODH块可以位于单个存储桶500A上。为了说明的目的,存储桶500A和存储桶500B是相同的逻辑存储桶。存储桶500B表示存储桶拆分后的存储桶500A。

[0071] 当已经为ODH确定了存储桶拆分条件时,可以将存储桶500A的ODH拆分为存储桶500B和501A,其中存储桶501A是新的存储桶。

[0072] 此存储桶拆分的一个强大特性是,只有4k块的一半必须移至新的存储桶,而块中的剩下的一半仍然保留。对于拆分ODH和拆分整个存储桶都是如此。如果ODH如图4所示拆分,则半个块将重新定位到另一个现有存储桶。如果存储桶的所有ODH被拆分,则每个ODH会将<KEY,VALUE>条目的一半留给原始存储桶,而<KEY,VALUE>条目的一半将重新定位到新存储桶。

[0073] 可以根据每个块中的键的LSB来确定拆分。为了说明的目的,在拆分存储桶之后,用新的版本号来指定对应的块(例如,401C变为401D),尽管这些块的物理存储器地址可能不变。另外,减小了拆分级别(对应于索引宽度)。

[0074] 块401C,403B和405A中的键的LSB为“0”(对应于存储桶500)。这些块分别被重新标记为块401D,403C和405B,并在存储桶500A上的ODH被拆分后保留在相同的存储桶(即,存储桶500A被重新标记为存储桶500B)中。

[0075] 块402B和404A中的键的LSB为“1”(对应于新存储桶501)。这些块分别被重新标记为块402C和404B,并且在拆分存储桶500A之后将其移动到存储桶501A(存储桶“1”)。

[0076] 当目前驻留在存储桶500B和501A上的ODH进一步拆分时,将考虑键的两个LSB。块401D中的键的两个LSB为“00”(对应于存储桶500)。块401D被重新标记为块401E,并且在存储桶500B(存储桶“0”)上的ODH被拆分之后,保留在相同的存储桶(即,存储桶500B被重新标记为存储桶500C)中。

[0077] 块403C和405B中的键的两个LSB为“10”(对应于新存储桶502)。块403C和405B被重

新标记为块403D和405C,并且在存储桶500B(存储桶“0”)上的ODH被拆分之后被移动到存储桶502A(存储桶“10”)。

[0078] 在块402C中的键的两个LSB为“01”(对应于存储桶501)。块402C被重新标记为块402D,并且在存储桶501A(存储桶“1”)上的ODH被拆分之后,保留在相同的存储桶(即,存储桶501A被重新标记为存储桶501B)中。

[0079] 块404B中的键的两个LSB为“11”(对应于新存储桶503)。块404B被重新标记为块404C,并且在存储桶501A(存储桶“1”)上的ODH被拆分之后被移动到存储桶503A(存储桶“11”)。

[0080] 在哈希的最低有效位中找到外部拆分索引。外部拆分索引中的位数由外部拆分级别指示。内部拆分索引位于外部拆分索引的位随后的最低有效位中。例如,在块405A中,外部拆分级别为0(即,单个存储桶的情况),内部拆分级别为3,并且第一条目的哈希为“0110”,外部拆分索引为NIL(即,无位),内部拆分索引为“110”,且哈希索引为“1”。如果存储桶500A被拆分,则所有块(401C,403B,405A,402B和404A)的外部拆分级别都将增加1,而内部拆分级别将减小1。在拆分存储桶之后,块405A(重新标记为框405B)保留在相同的存储桶中,即,存储桶500A(重新标记为存储桶500B)。外部拆分索引从“NIL”(在块405A中)变为“1”(在块405B中)。块405B的内部拆分索引从“110”(块405A)变为“11”(块405B)。并且在块405B中第一条目的哈希索引保持为“1”。

[0081] 当增加外部拆分级别并且减小内部拆分级别时,一个或多个完整块被移动到下一存储桶。内部拆分索引以1结尾的的所有块(例如,块402B和404A)被移到新存储桶。内部拆分索引以0结尾的所有块(例如,块401C,403B和405A)都位于同一存储桶中。每个ODH块都会记住块ID,内部拆分索引(和/或内部拆分级别)和外部拆分索引(和/或外部拆分级别)。

[0082] 因为目录中的两个以上条目可能存在哈希冲突,所以每个条目都具有“防哈希冲突”键。如果没有两个条目具有相同的哈希值,则防哈希冲突键为0。因此,ODH将哈希映射到值和防哈希冲突(AHC)索引。如果存在冲突(即,另一个文件已经具有相同的哈希(键)),则如块405C所示,用递增的反哈希冲突键来存储该值。每个块条目都存储值和防冲突索引。

[0083] 为了支持非常大的哈希集,可以在多个进程(和几个存储桶)上支持哈希集处理。此外,为了实现负载平衡,可以将存储桶本身拆分和/或合并。例如,将新服务器添加到群集时,现有存储桶可能会拆分并重新分布在所有服务器上。

[0084] 如果条目的组合数量将适合块的足够小的部分(例如,小于4k块的2k),则同胞块(即两个处于相同拆分级别并且来自同一父块的块)可以合并。图6A示出了根据本公开的示例实现的使用盘上哈希来合并快闪存储器注册表的块的示例。

[0085] 在关于图4所述的拆分之后,ODH包括块401C,块403B,块405A,块402B和块404A。块403B和块405A处于相同的拆分级别(即,拆分级别3),并且来自相同的父块。例如,如果从框405A中移除两个<KEY,VALUE>条目,则条目的组合数量(即2)将适合于块的足够小的部分。因此,可以减小块403B的拆分级别和相应的索引宽度以形成块601(其可以与403B位于相同的地址)。

[0086] 同样,块402B和块404A处于相同的拆分级别(即,拆分级别2),并且来自相同的父块。例如,如果从块402B中移除<KEY,VALUE>条目,并且从块404A中移除两个<KEY,VALUE>条目,则条目的组合数量(即2)将适合块的足够小的部分。因此,可以减小块402B的拆分级别

和相应的索引宽度以形成块602(其可以与402B位于相同的地址)。

[0087] 图6B示出了根据本公开的示例实现的从目录读取操作返回的条目的示例。目录读取操作以常规遍历顺序遍历磁盘上的哈希块,并按其键的顺序返回每个块中的条目,就好像磁盘上的哈希被扩展到最大级别(即,内部级别+外部级别)一样。这就像从ODH块中创建树一样。如图6B所示,即使块(401C,601和601)都不包括四个级别,树的叶子也处于第四级别。

[0088] 返回目录中的文件条目,以其哈希的位以逆顺序排序。这也是文件条目将以最高拆分级别拆分的顺序。因为ODH条目是以这种方式排序的,所以目录读取操作返回一个值后,拆分将不会将其向前发送以再次返回。

[0089] 目录读取操作返回(恰好一次)遍历开始时存在并且在遍历结束时仍然存在的每个目录条目。目录读取操作可以选择性地返回(最多一次)遍历过程中创建或删除的每个目录条目。目录读取操作返回请求数量的目录条目和“连续令牌”,以允许遍历继续进行。

[0090] 图7是说明根据本发明的示例实例的用于利用盘上哈希添加快闪存储器注册表的块的实例方法的流程图。在块701,ODH接收要添加到注册表的<KEY,VALUE>对。

[0091] 在块703处,根据键的一个或多个LSB来定位对应的存储桶和块。LSB的数量可以根据当前的最大拆分级别来确定。如果一个或多个LSB与索引不匹配,则LSB的数量可以减少一。在具有第二大拆分级别的块上继续搜索匹配索引,依此类推。例如,如果找不到索引=“11”(假设最大拆分级别为2),则继续搜索拆分级别=1的索引=“1”。

[0092] 在705,如果在适当的表中添加额外的<KEY,VALUE>条目超过或接近块容量,则可能需要进行块拆分。在707,如果可以添加新块,则在711可能发生块拆分。在707,如果无法添加新块,则在709可能返回IO错误。无法添加新块的原因包括内存容量和/或处理器负载。

[0093] 在705,如果确定块容量足够或在711进行了块拆分,则在713添加新接收的<KEY,VALUE>对。在715处,如果DEC指示有必要,则可以进行如图5中所述的存储桶拆分。

[0094] 图8是示出根据本公开的示例实现的用于利用ODH来访问快闪存储器注册表的块的方法的流程图。通过枚举所有条目同时枚举所有块开始目录读取。

[0095] 在块801,对所请求文件的索引ID计算键哈希。VFS前端(FE)可以根据外部拆分级别选择存储桶。每个FE可以为每个目录存储正确的拆分级别。目录的索引节点也可以存储拆分级别。因此,FE可以从保存索引节点的存储桶中确定正确的拆分级别。此外,当持有ODH的存储桶决定拆分时,通过索引节点协调存储桶拆分。

[0096] 在块803处,根据键的一个或多个LSB来搜索对应的存储桶和块。LSB的数量可以根据当前的最大拆分级别来确定。如果一个或多个LSB与索引不匹配,则LSB的数量可以减少一。在具有第二大拆分级别的块上继续搜索匹配索引,依此类推。例如,如果未找到索引=“11”(假设最大拆分级别为2),则继续搜索拆分级别=1的索引=“1”。如果在803中未找到与键对应的条目,在805处可能返回IO错误。

[0097] 如果在803找到对应于键的条目,则在807访问哈希表中的<KEY,VALUE>条目。如果删除所访问的<KEY,VALUE>条目,则有可能合并块。如果访问表及其同胞表中<KEY,VALUE>条目的总数适合单个块,则将该表及其同胞表合并到锚定块中。

[0098] 目录读取操作可以迭代的方式请求少量文件。如果在809处请求多个文件,则对下一个条目的搜索可以在当前块处开始并且继续到同胞块(即,处于相同的拆分级别)。为了

找到下一个块,搜索可以反转索引的位,加1,然后再次反转位。

[0099] 在811处,目录读取操作允许根据连续令牌从任何条目中继续。重新调整到目录读取操作的结果可有效地在ODH中找到条目,即使在文件创建和删除过程中也是如此。如果提交的拆分级别在单个目录读取操作期间发生更改,则目录读取操作可能需要跳转到新的片段才能继续。连续令牌允许遍历从返回的最后一个条目继续,而不考虑拆分。连续令牌将是返回的最后一个键(或哈希)。返回的最后一个文件被赋予连续令牌以支持"无状态"列表,而目录可能由于拆分和合并而发生更改。每个条目都可以获取一个小程序(cookie),操作系统会将该小程序呈现回来,以支持列表的继续。这允许下一个目录读取操作在813处查找正确的ODH块并从那里继续。

[0100] 图9示出其中两个分布式故障弹性地址空间驻留在多个固态存储盘上的示例实现方式。组块$510_{1,1}$至$510_{D,C}$可以被组织为多个组块条带520_1至520_S(S是整数)。在示例实现方式中,使用前向纠错(例如,擦除编码)来分别保护每个组块条带520_s(s是整数,其中$1 \leqslant s \leqslant S$)。因此,可以基于期望的数据保护级别来确定任何特定组块条带520_s中的组块$510_{d,c}$的数量。

[0101] 为了说明的目的,假设每个组块条带520_s包括N=M+K(其中,N,M和K中的每个是整数)个组块$510_{d,c}$,则N个组块$510_{d,c}$中的M个可以存储数据码(对于当前存储设备通常是二进制数字或"位")且N个组块$510_{d,c}$中的K个可以存储保护码(再次通常是位)。然后,虚拟文件系统可以向每个条带520_s分配来自N个不同故障域的N个组块$510_{d,c}$。

[0102] 如本文所用,"故障域"是指一组组件,其中任何一个组件中的故障(组件掉电,变得无响应等)可能导致所有组件的故障。例如,如果机架具有单个机架顶部交换机,则该交换机的故障将使与该机架上的所有组件(例如,计算、存储和/或VFS节点)的连接断开。因此,对于系统的其余部分,这等效于该机架上的所有组件如同一起出现故障。根据本公开的虚拟文件系统可以包括比组块510更少的故障域。

[0103] 在对于每个这样的节点,虚拟文件系统的节点以完全冗余的方式仅与单个存储设备506连接和供电的示例实现方式中,故障域可以仅仅是该单个存储设备506。因此,在示例实现方式中,每个组块条带520_s包括位于存储设备506_1至506_D的N个中的每个上的多个组块$510_{d,c}$(因此,D大于或等于N)。这样的实现方式的示例在图7中显示。

[0104] 在图9中,D=7,N=5,M=4,K=1,并且存储设备被组织为两个DFRAS。这些数字仅用于说明,而无意作为限制。第一DFRAS的三个组块条带520被任意地调出以用于说明。第一组块条带520_1由组块$510_{1,1}$、$510_{2,2}$、$510_{3,3}$、$510_{4,5}$和$510_{5,6}$组成;第二组块条带520_2由组块$510_{3,2}$、$510_{4,3}$、$510_{5,3}$、$510_{6,2}$和$510_{7,3}$组成;第三组块条带5203由组块$510_{1,4}$、$510_{2,4}$、$510_{3,5}$、$510_{5,7}$和$510_{7,5}$组成。

[0105] 尽管在实际实现方式中在图9的简单示例中D=7和N=5,但是D可能比N大得多(例如,大于1的整数倍,并且可能高至多个数量级),并且可以选择两个值,以使单个DFRAS的任何两个组块条带520驻留在同一组N个存储设备506(或更一般地说,在同一组N个故障域)上的概率低于所需阈值。以此方式,任何单个存储设备506_d(或更普遍地,任何单个故障域)的故障将导致(期望统计可能基于以下各项确定:D和N的选定值,N个存储设备506的大小以及故障域的布置)任何特定条带520_s的至多一个组块$510_{b,c}$的丢失。更进一步,双重故障将导致绝大多数条带最多丢失单个块$510_{b,c}$,并且只有少量的条带(基于D和N的值确定)将从任

何特定的条带520_s中丢失两个组块(例如,两次故障的条带数量可能会比一次故障的条带数量成指数地减少)。

[0106] 例如,如果每个存储设备506d是1TB,并且每个组块是128MB,则存储设备506_d的故障将导致(期望统计可能基于以下各项确定:D和N的选定值,N个存储设备506的大小以及故障域的布置)7812(=1TB/128MB)个组块条带520丢失一个组块510。对于每个这样的受影响的组块条带520_s,可以使用适当的前向纠错算法和组块条带520_s的其他N-1个组块来快速地重建丢失的组块$510_{d,c}$。此外,由于受影响的7812个组块条带均匀分布在所有存储设备506_1至506_D中,因此重建丢失的7812个组块$510_{d,c}$将涉及(希望的统计可能基于以下项确定:D和N的选定值,N个存储设备506的大小以及故障域的布置)从每个存储设备506_1至506_D读取相同数量的数据(即,重建丢失的数据的负担均匀地分布在所有存储设备506_1-506_D中,以便从故障中非常快速地恢复)。

[0107] 接下来,转向两个存储设备506_1至506_D并发故障的情况(或更一般地说,两个故障域的并发故障),由于每个DFRAS的组块条带520_1至520_S在所有存储设备506_1至506_D上的均匀分布,只有极少数的组块条带520_1至520_S会丢失N个组块中的两个。虚拟文件系统可操作为基于元数据快速识别这种两次丢失的组块条带,该元数据指示组块条带520_1至520_S与存储设备506_1至506_D之间的映射。一旦识别出这样的两次丢失组块条带,虚拟文件系统就可以在开始重建一次丢失组块条带化之前优先考虑重建那些两次丢失组块条带。其余的组块条带将仅具有单个丢失的组块,并且对于它们(受影响的组块条带中的绝大多数),两个存储设备506_d的并发故障与仅一个存储设备506_d的故障相同。类似的原理适用于三个并发故障(在两个并发故障场景中,具有三个故障块的组块条带的数量将远远少于具有两个故障块的数量),依此类推。在示例实现方式中,可以基于组块条带520_s中的丢失数量来控制执行组块条带520_s的重构的速率。这可以通过例如控制执行用于重建的读取和提交的速率,执行用于重建的FEC计算的速率,传送用于重建的网络消息的速率等来实现。

[0108] 图10示出根据本公开的示例实现方式的前向纠错方案,其可以用于保护存储到虚拟文件系统的非易失性存储器的数据。所示为DFRAS的块条带530_1至530_4的存储块$902_{1,1}$至$902_{7,7}$。在图10的保护方案中,每个条带的五个块用于存储数据码,且每个条带的两个块用于保护码(即,M=5和K=2)的数据存储。在图10中,使用以下公式(1)至(9)计算保护码:

[0109] $$P1 = D1_1 \oplus D2_2 \oplus D3_3 \oplus D4_4 \oplus D5_4 \quad (1)$$

[0110] $$P2 = D2_1 \oplus D3_2 \oplus D4_3 \oplus D5_3 \oplus D1_4 \quad (2)$$

[0111] $$P3 = D3_1 \oplus D4_2 \oplus D5_2 \oplus D1_3 \oplus D2_4 \quad (3)$$

[0112] $$P4 = D4_1 \oplus D5_1 \oplus D1_2 \oplus D2_3 \oplus D3_4 \quad (4)$$

[0113] $$Z = D5_1 \oplus D5_2 \oplus D5_3 \oplus D5_4 \quad (5)$$

[0114] $$Q1 = D1_1 \oplus D1_2 \oplus D1_3 \oplus D1_4 \oplus Z \quad (6)$$

[0115] $$Q2 = D2_1 \oplus D2_2 \oplus D2_3 \oplus D2_4 \oplus Z \quad (7)$$

[0116] $$Q3 = D3_1 \oplus D3_2 \oplus D3_3 \oplus D3_4 \oplus Z \quad (8)$$

[0117] $$Q4 = D4_1 \oplus D4_2 \oplus D4_3 \oplus D4_4 \oplus Z \quad (9)$$

[0118] 因此,图10中的四个条带530_1至530_4是多个条带(在这种情况下为四个条带)FEC保

护域的一部分，并且任何块条带530_1至530_4中的任何两个或更少块的丢失通过使用以上等式(1)至(9)的各种组合来恢复。为了进行比较，单条带保护域的示例是：如果仅通过P1保护$D1_1$，$D2_2$，$D3_3$，$D4_4$，$D5_4$，并且将$D1_1$，$D2_2$，$D3_3$，$D4_4$，$D5_4$和P1全部写入条带530_1(530_1是单个条带FEC保护域)。

[0119] 根据本公开的示例实现方式，多个计算设备经由网络彼此通信地耦接，并且多个计算设备中的每一个包括多个存储设备中的一个或多个。多个故障弹性地址空间分布在多个存储设备上，使得多个故障弹性地址空间中的每一个跨越多个存储设备。多个故障弹性地址空间中的每一个被组织成多个条带(例如，如图9和图10所示的多个530)。多个条带中的每个或多个条带是多个前向纠错(FEC)保护域(例如，诸如图10中的多条带FEC域)中的相应一个域的一部分。多个条带中的每一个可以包括多个存储块(例如，多个512)。多个条带中的特定条带的每个块可以位于多个存储设备中的不同存储设备上。多个存储块的第一部分(例如，由图10的条带530_1的$902_{1,2}$至$902_{1,6}$组成的5个数量)可用于存储数据码，而多个存储块的第二部分(例如，图10的条带530_1的两个$902_{1,1}$和$902_{1,7}$的数量)可以用于存储至少部分地基于数据码计算出的保护码。多个计算设备可以可操作以对多个条带进行排序。该排序可以用于选择多个条带中的哪个条带用于对多个故障弹性地址空间之一的下一提交操作。排序可以基于在多个条带中的每个条带中有多少个受保护和/或不受保护的存储块。对于多个条带中的任何一个，该排序可以基于存储在具有多个条带中的特定一个的多个存储设备上的位图。该排序可以基于多个条带中的每个条带中当前存储数据的块的数量。该排序可以基于用于向多个条带中的每个条带进行提交的读取和写入开销。每个故障弹性地址空间可以在任何给定时间仅由多个计算设备之一拥有，并且多个故障弹性地址空间中的每个只能由其所有者读取和写入。每个计算设备可以拥有多个故障弹性地址空间。多个存储设备可以被组织成多个故障域。多个条带中的每个条带可以跨越多个故障域。每个故障弹性地址空间可以跨越所有多个故障域，使得当多个故障域中的任何特定故障域发生故障时，在多个故障域中的每个其他故障域之间分配用于重建丢失数据的工作量。多个条带可以分布在多个故障域上，使得在多个故障域中的两个故障域同时发生故障的情况下，多个故障域中的两个故障域上的多个条带中的任何一个特定条带的两个块出现的机率小于多个故障域中的两个故障域上的多个条带中的任何特定条带中只有一个块存在的几率。多个计算设备可用于首先重建具有两个故障块的多个条带中的任何一个，然后重建仅具有一个故障块的多个条带中的任何一个。多个计算设备可用于以比仅具有一个故障块的多个条带的重建速率更高的速率(例如，在专用于重建的CPU时钟周期中，百分比较大，专用于重建的网络传输机会占较大百分比，等等)来执行具有两个故障块的多个条带的重建。多个计算设备可用于在一个或多个故障域发生故障的情况下，基于丢失了多个条带中的同一条带的多少其他块来确定重建任何特定丢失块的速率。其中，多个故障域中的一个或多个包括多个存储设备。多个FEC保护域中的每一个可以跨越多个条带中的多个条带。多个条带可以被组织为多个组(例如，图9中的组块条带520_1至520_S)，其中，多个组中的每一个包括多个条带中的一个或多个，以及多个计算设备可操作用于对每个组，对组的多个条带中的一个或多个进行排序。多个计算设备可操作为：对多个组中的选定的一组执行连续的提交操作，直到组的多个条带中的一个或多个条带不再满足确定的标准为止，并且在多个组中选择的一个不再满足确定的标准时，选择多个组中的另一个。该标准可以基于有多少块可用于新数据写

入。

[0120] 尽管已经参考某些实现描述了本方法和/或系统,但是本领域技术人员将理解,在不脱离本方法和/或系统的范围的情况下,可以进行各种改变并且可以替换等同物。另外,在不脱离本发明范围的情况下,可以做出许多修改以使特定情况或材料适应本发明的教导。因此,意图是本方法和/或系统不限于所公开的特定实现,而是本方法和/或系统将包括落入所附权利要求的范围内的所有实现。

[0121] 如本文所使用的,术语“电路”和“电路系统”是指物理电子组件(即硬件)以及任何软件和/或固件(“代码”),其可以配置硬件,由硬件执行和/或与硬件相关联。如本文所使用的,例如,特定的处理器和存储器在执行第一行或多行代码时可以包括第一“电路”,并且在执行第二行或多行代码时可以包括第二“电路”。如本文所用,“和/或”是指列表中由“和/或”连接的任何一个或多个项目。作为示例,“x和/或y”表示三元素集{(x),(y),(x,y)}中的任何元素。换句话说,“x和/或y”是指“x和y之一或两者”。作为另一个示例,“x,y和/或z”表示七元素集{(x),(y),(z),(x,y),(x,z),(y,z),(x,y,z)}。换句话说,“x,y和/或z”是指“x,y和z中的一个或多个”。如本文所使用的,术语“示例性”是指用作非限制性示例,实例或说明。如本文所使用的,术语“例如”和“例如”引出一个或多个非限制性示例,实例或插图的列表。如本文所利用的,只要电路包含执行该功能所需的必要硬件和代码(如有必要),该电路就“可操作”以执行功能,而不管该功能的性能是否被禁用(例如,通过用户可配置的设置,出厂调整等)。

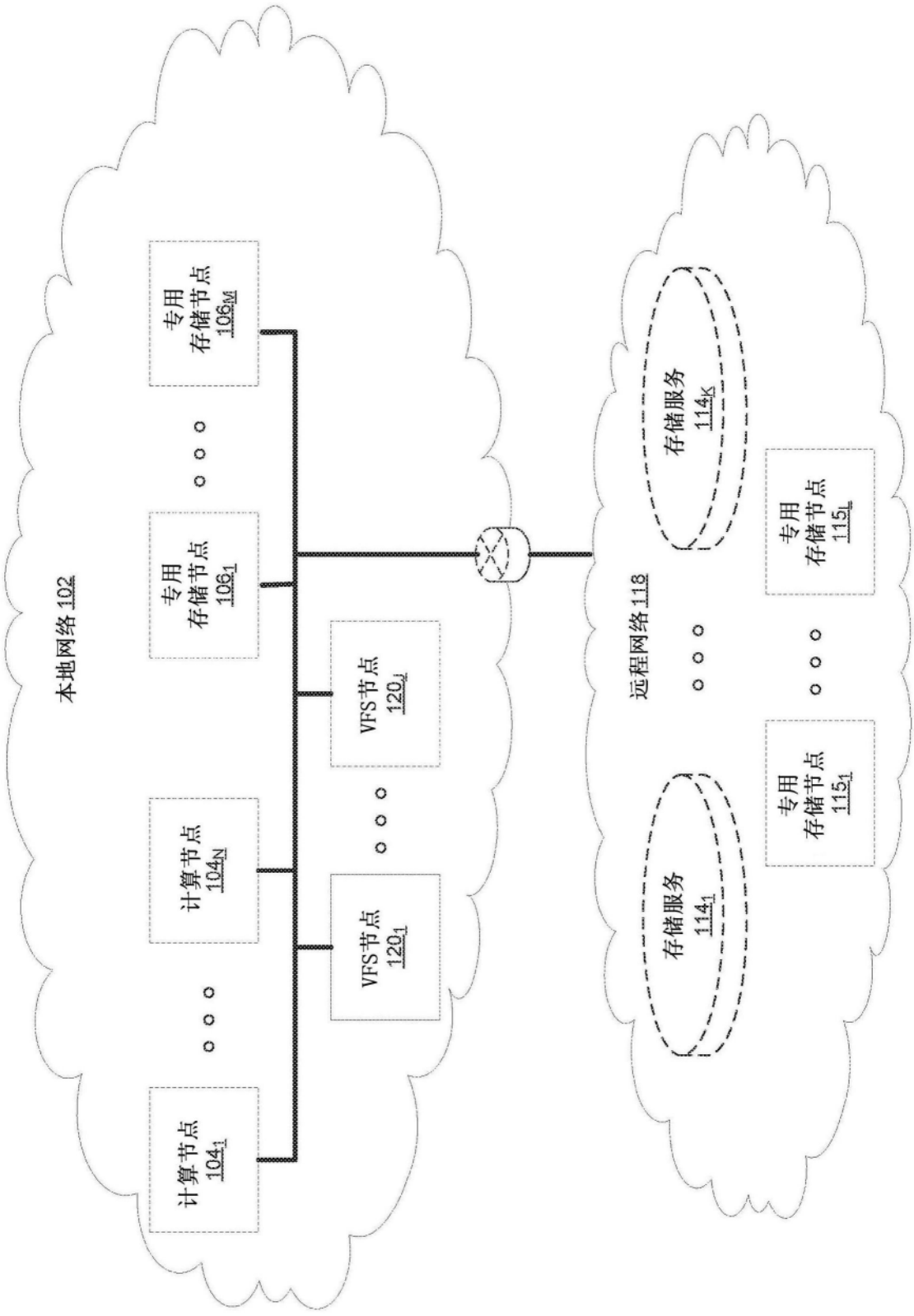

本地网络 102
计算节点 104_1
计算节点 104_N
专用存储节点 106_1
专用存储节点 106_M
VFS节点 120_1
VFS节点 120_J
远程网络 118
存储服务 114_1
存储服务 114_K
专用存储节点 115_1
专用存储节点 115_L

图1

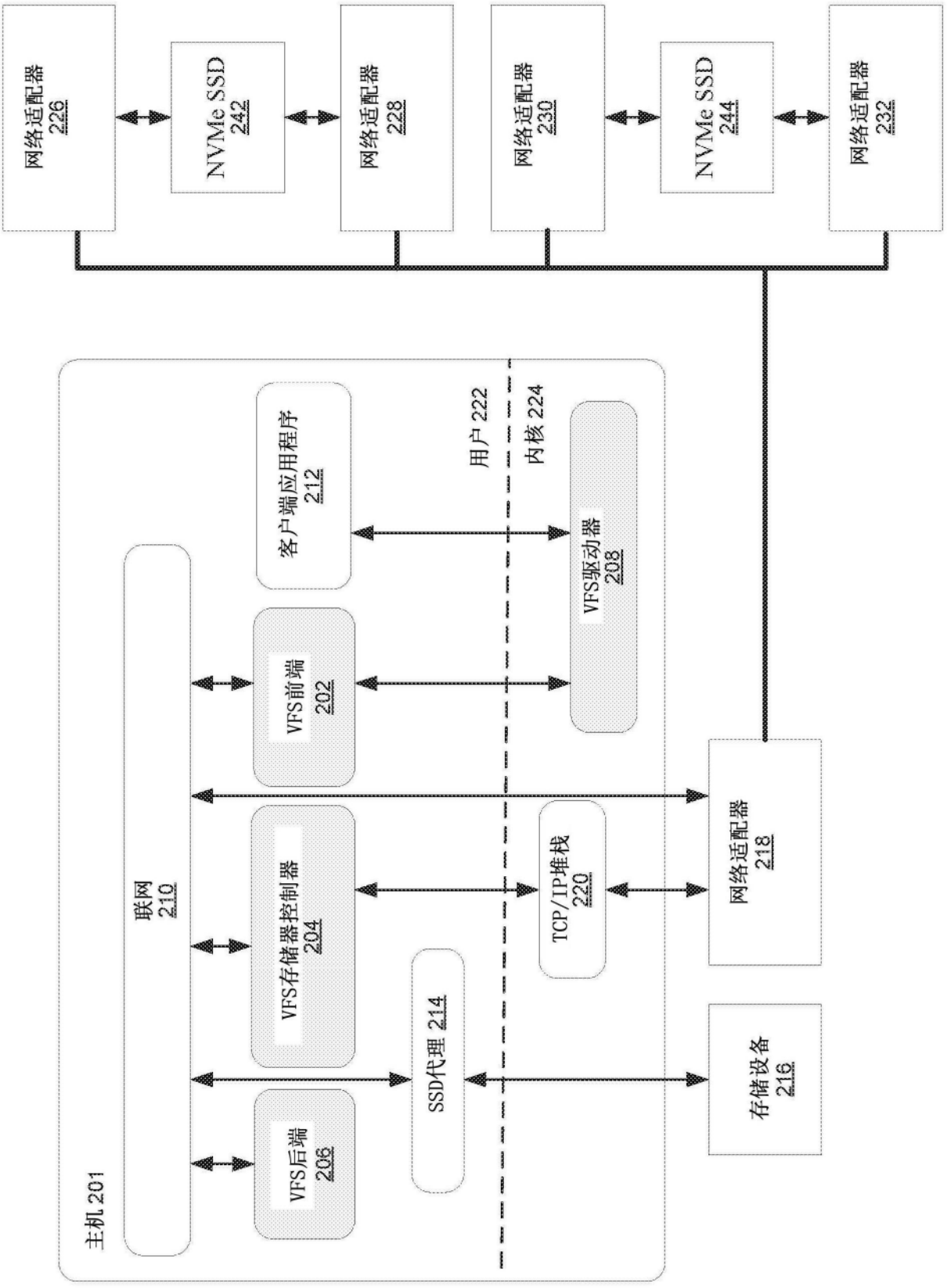

网络适配器 226
NVMe SSD 242
网络适配器 228
网络适配器 230
NVMe SSD 244
网络适配器 232
主机 201
联网 210
客户端应用程序 212
用户 222
内核 224
VFS驱动器 208
VFS前端 202
VFS存储器控制器 204
TCP/IP堆栈 220
网络适配器 218
VFS后端 206
SSD代理 214
存储设备 216

图2

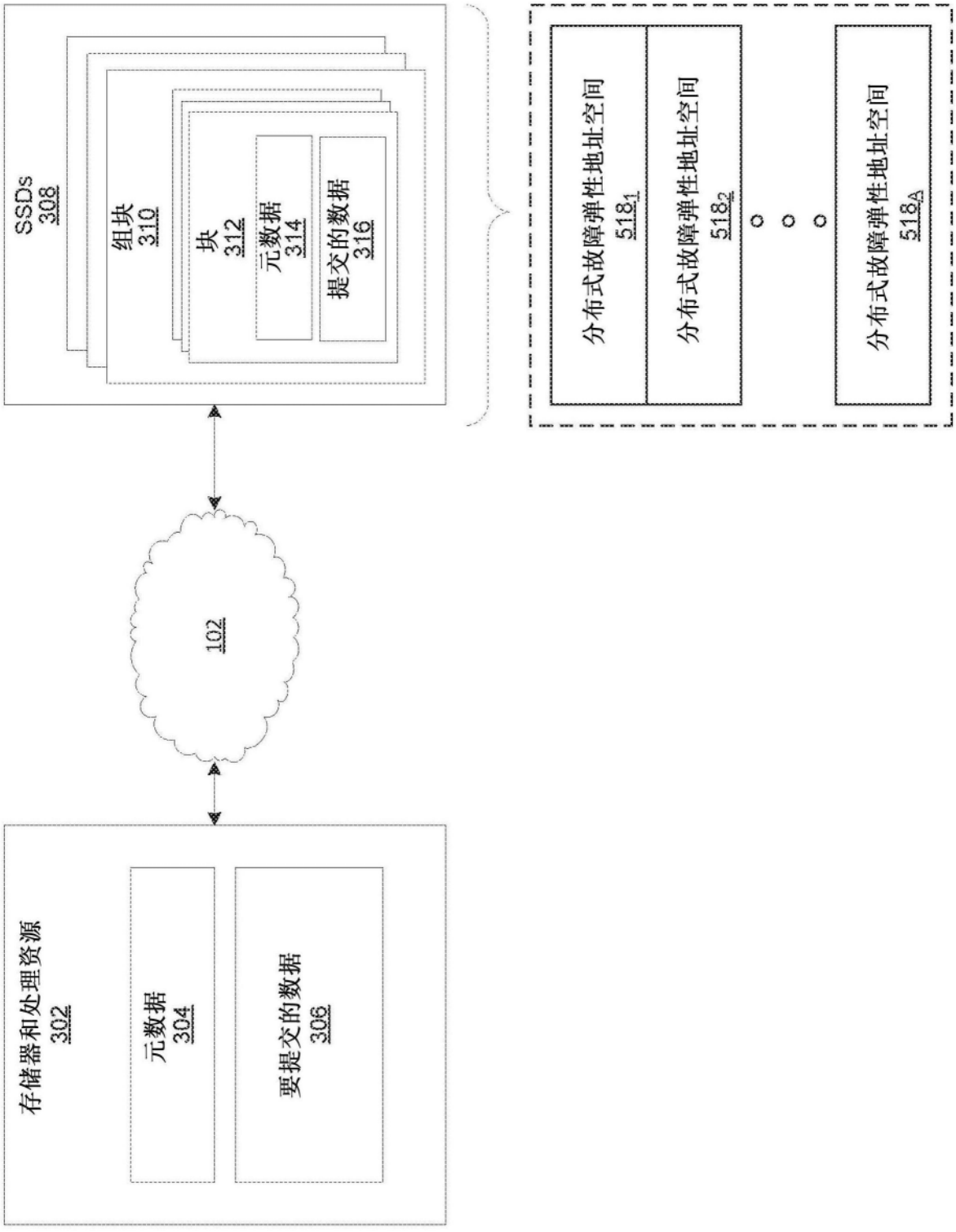

SSDs
308
组块
310
块
312
元数据
314
提交的数据
316
分布式故障弹性地址空间
518_1
分布式故障弹性地址空间
518_2
分布式故障弹性地址空间
518_A
102
存储器和处理资源
302
元数据
304
要提交的数据
306

图3

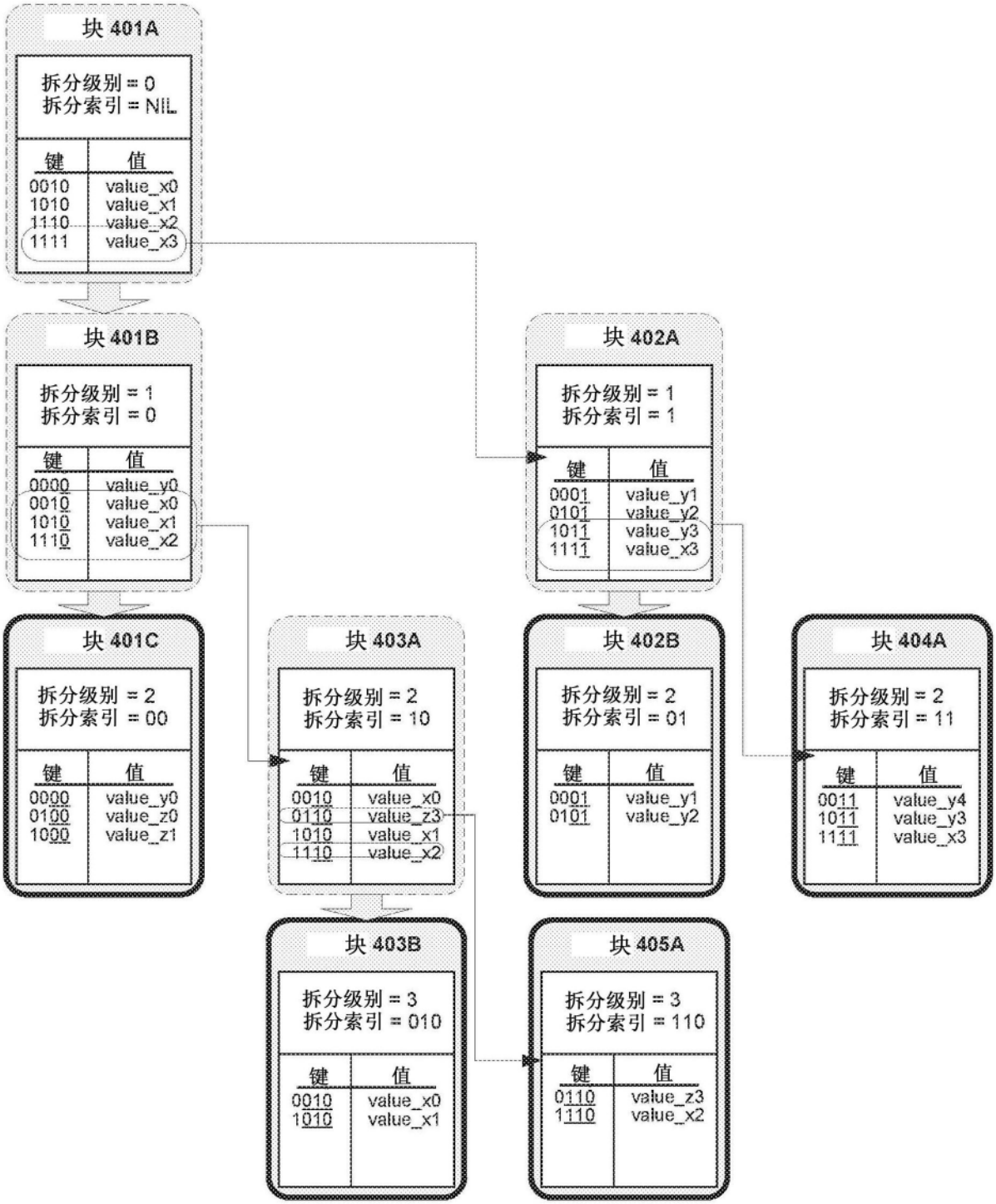

块 401A
拆分级别 = 0
拆分索引 = NIL
键 值
0010 value_x0
1010 value_x1
1110 value_x2
1111 value_x3
块 401B
拆分级别 = 1
拆分索引 = 0
键 值
0000 value_y0
0010 value_x0
1010 value_x1
1110 value_x2
块 402A
拆分级别 = 1
拆分索引 = 1
键 值
0001 value_y1
0101 value_y2
1011 value_y3
1111 value_x3
块 401C
拆分级别 = 2
拆分索引 = 00
键 值
0000 value_y0
0100 value_z0
1000 value_z1
块 403A
拆分级别 = 2
拆分索引 = 10
键 值
0010 value_x0
0110 value_z3
1010 value_x1
1110 value_x2
块 402B
拆分级别 = 2
拆分索引 = 01
键 值
0001 value_y1
0101 value_y2
块 404A
拆分级别 = 2
拆分索引 = 11
键 值
0011 value_y4
1011 value_y3
1111 value_x3
块 403B
拆分级别 = 3
拆分索引 = 010
键 值
0010 value_x0
1010 value_x1
块 405A
拆分级别 = 3
拆分索引 = 110
键 值
0110 value_z3
1110 value_x2

图4

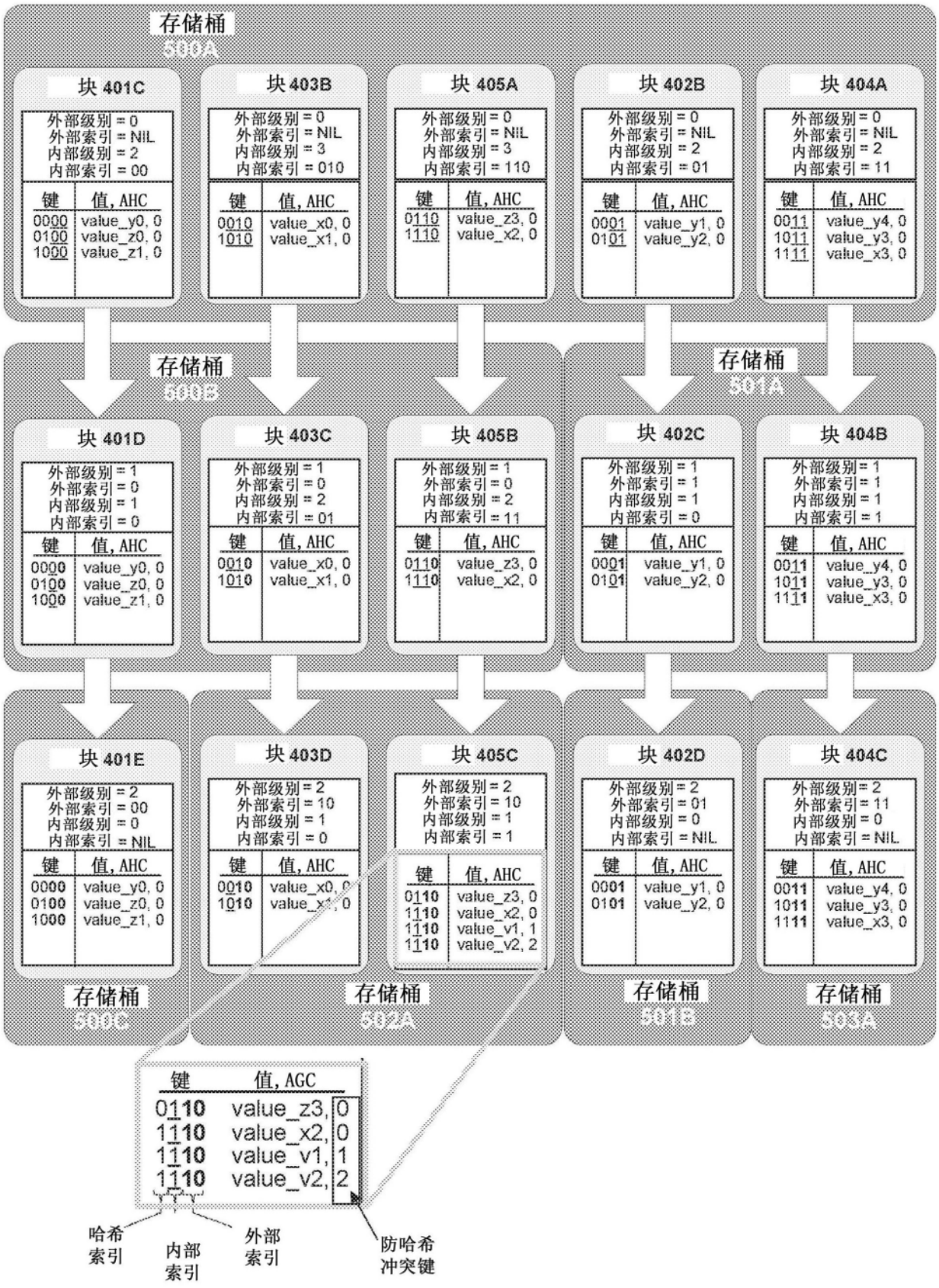

存储桶
500A
块 401C
外部级别 = 0
外部索引 = NIL
内部级别 = 2
内部索引 = 00
键 值, AHC
0000 value_y0, 0
0100 value_z0, 0
1000 value_z1, 0
块 403B
外部级别 = 0
外部索引 = NIL
内部级别 = 3
内部索引 = 010
键 值, AHC
0010 value_x0, 0
1010 value_x1, 0
块 405A
外部级别 = 0
外部索引 = NIL
内部级别 = 3
内部索引 = 110
键 值, AHC
0110 value_z3, 0
1110 value_x2, 0
块 402B
外部级别 = 0
外部索引 = NIL
内部级别 = 2
内部索引 = 01
键 值, AHC
0001 value_y1, 0
0101 value_y2, 0
块 404A
外部级别 = 0
外部索引 = NIL
内部级别 = 2
内部索引 = 11
键 值, AHC
0011 value_y4, 0
1011 value_y3, 0
1111 value_x3, 0
存储桶
500B
存储桶
501A
块 401D
外部级别 = 1
外部索引 = 0
内部级别 = 1
内部索引 = 0
键 值, AHC
0000 value_y0, 0
0100 value_z0, 0
1000 value_z1, 0
块 403C
外部级别 = 1
外部索引 = 0
内部级别 = 2
内部索引 = 01
键 值, AHC
0010 value_x0, 0
1010 value_x1, 0
块 405B
外部级别 = 1
外部索引 = 0
内部级别 = 2
内部索引 = 11
键 值, AHC
0110 value_z3, 0
1110 value_x2, 0
块 402C
外部级别 = 1
外部索引 = 1
内部级别 = 1
内部索引 = 0
键 值, AHC
0001 value_y1, 0
0101 value_y2, 0
块 404B
外部级别 = 1
外部索引 = 1
内部级别 = 1
内部索引 = 1
键 值, AHC
0011 value_y4, 0
1011 value_y3, 0
1111 value_x3, 0
块 401E
外部级别 = 2
外部索引 = 00
内部级别 = 0
内部索引 = NIL
键 值, AHC
0000 value_y0, 0
0100 value_z0, 0
1000 value_z1, 0
存储桶
500C
块 403D
外部级别 = 2
外部索引 = 10
内部级别 = 1
内部索引 = 0
键 值, AHC
0010 value_x0, 0
1010 value_x1, 0
块 405C
外部级别 = 2
外部索引 = 10
内部级别 = 1
内部索引 = 1
键 值, AHC
0110 value_z3, 0
1110 value_x2, 0
1110 value_v1, 1
1110 value_v2, 2
存储桶
502A
块 402D
外部级别 = 2
外部索引 = 01
内部级别 = 0
内部索引 = NIL
键 值, AHC
0001 value_y1, 0
0101 value_y2, 0
存储桶
501B
块 404C
外部级别 = 2
外部索引 = 11
内部级别 = 0
内部索引 = NIL
键 值, AHC
0011 value_y4, 0
1011 value_y3, 0
1111 value_x3, 0
存储桶
503A
键 值, AGC
0110 value_z3, 0
1110 value_x2, 0
1110 value_v1, 1
1110 value_v2, 2
哈希索引
内部索引
外部索引
防哈希冲突键

图5

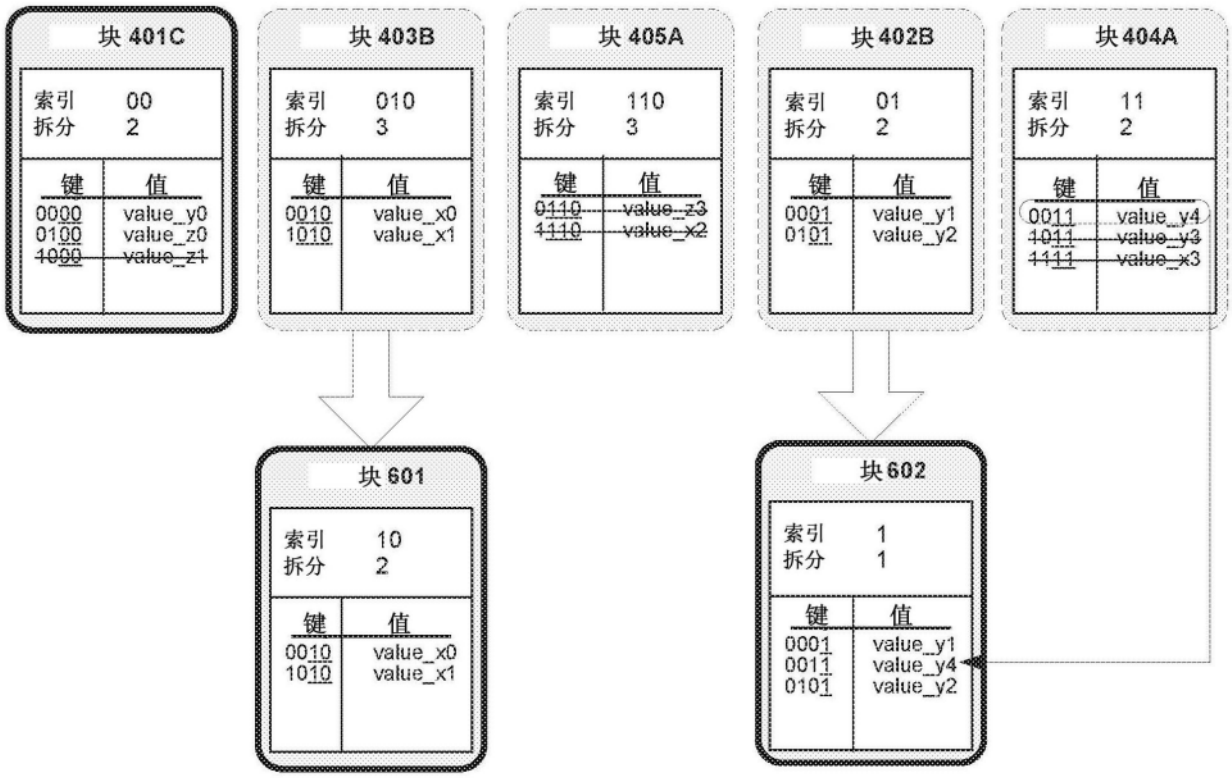

块 401C
索引 00
拆分 2
键 值
0000 value_y0
0100 value_z0
1000 value_z1
块 403B
索引 010
拆分 3
键 值
0010 value_x0
1010 value_x1
块 405A
索引 110
拆分 3
键 值
0110 value_z3
1110 value_x2
块 402B
索引 01
拆分 2
键 值
0001 value_y1
0101 value_y2
块 404A
索引 11
拆分 2
键 值
0011 value_y4
1011 value_y3
1111 value_x3
块 601
索引 10
拆分 2
键 值
0010 value_x0
1010 value_x1
块 602
索引 1
拆分 1
键 值
0001 value_y1
0011 value_y4
0101 value_y2

图6A

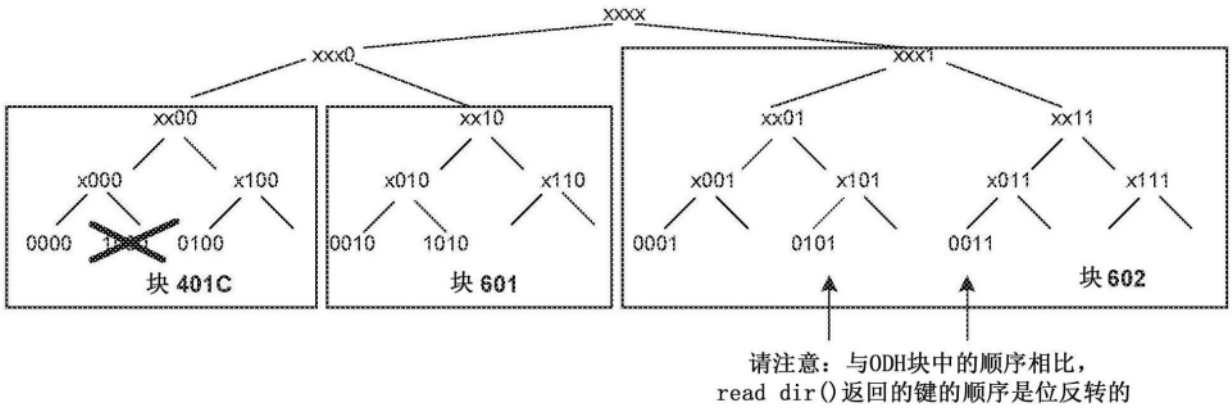

xxxx
xxx0
xxx1
xx00
xx10
xx01
xx11
x000
x100
x010
x110
x001
x101
x011
x111
0000
0100
0010
1010
0001
0101
0011
块 401C
块 601
块 602
请注意：与ODH块中的顺序相比，
read dir()返回的键的顺序是位反转的

图6B

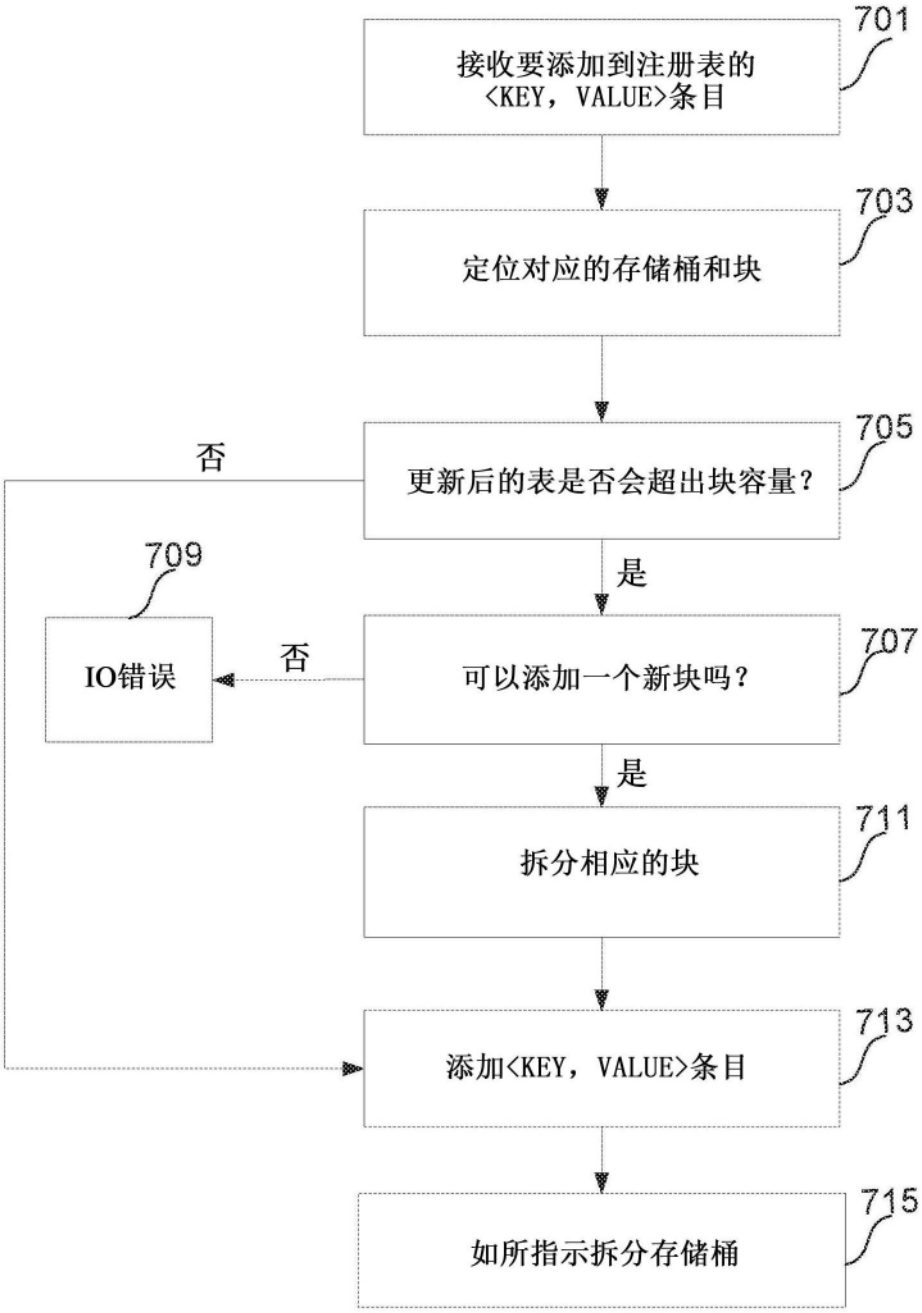

701
接收要添加到注册表的<KEY，VALUE>条目
703
定位对应的存储桶和块
705
更新后的表是否会超出块容量？
否
是
709
IO错误
否
707
可以添加一个新块吗？
是
711
拆分相应的块
713
添加<KEY，VALUE>条目
715
如所指示拆分存储桶

图7

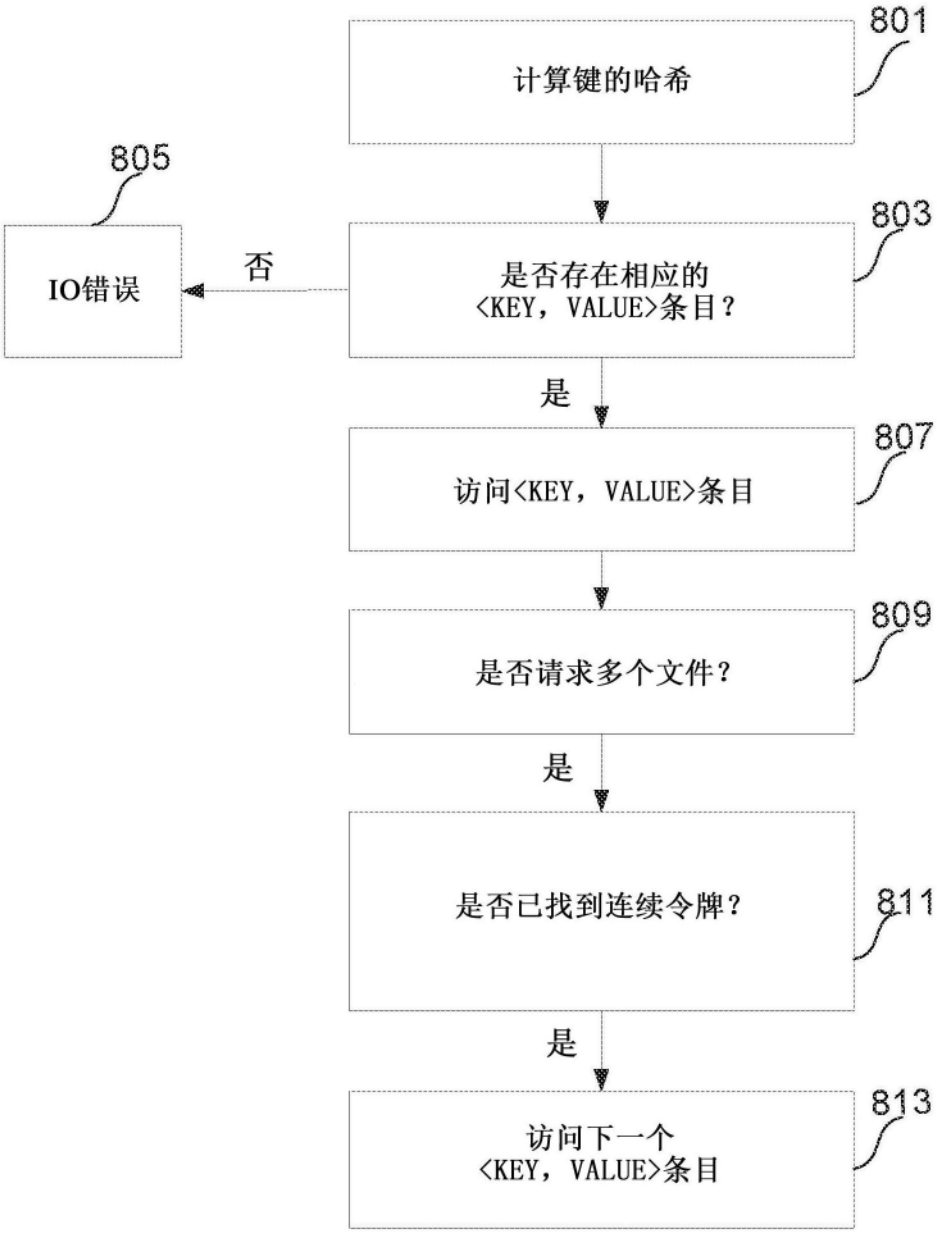

801
计算键的哈希
805
IO错误
否
803
是否存在相应的
<KEY，VALUE>条目？
是
807
访问<KEY，VALUE>条目
809
是否请求多个文件？
是
是否已找到连续令牌？
811
是
813
访问下一个
<KEY，VALUE>条目

图8

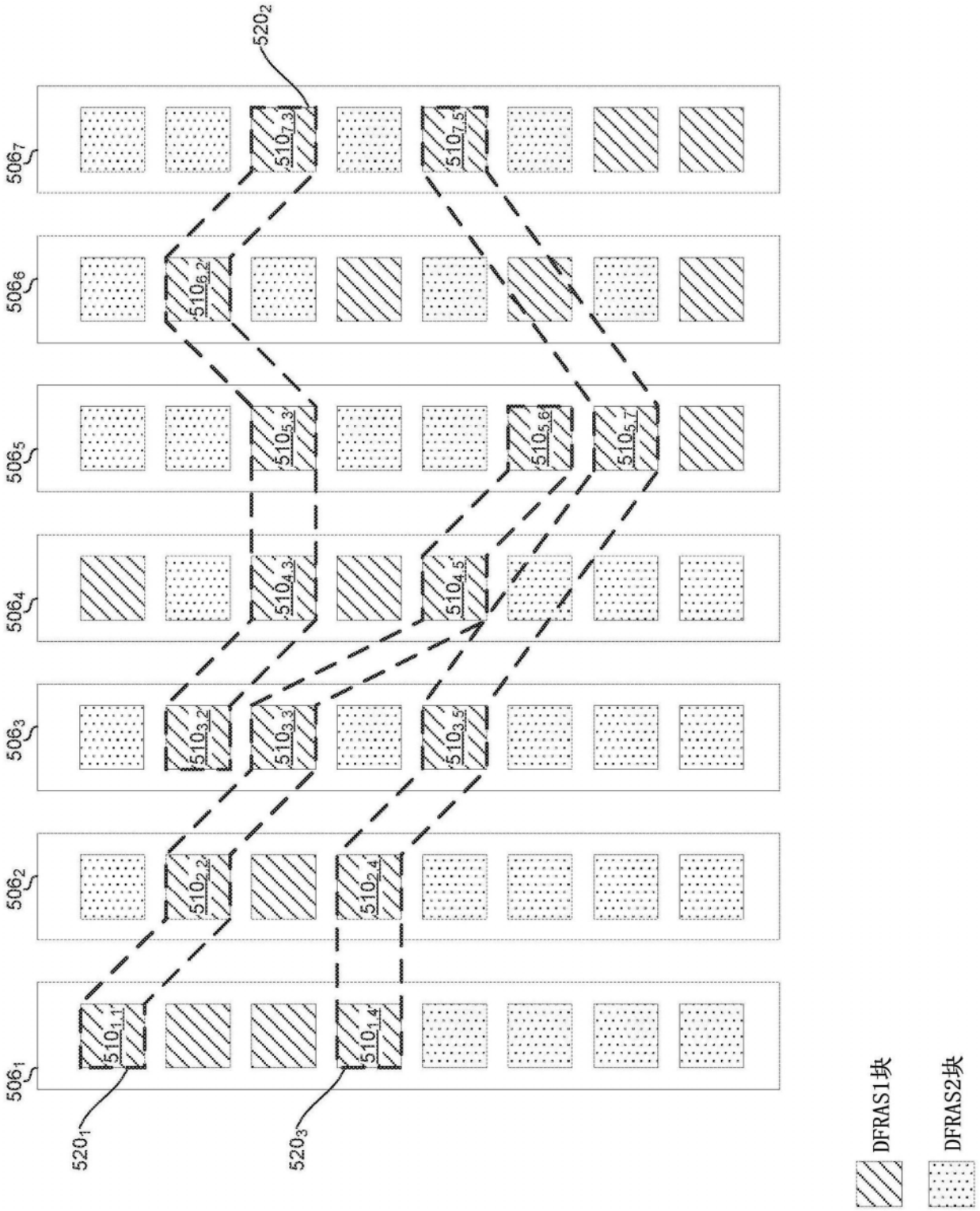

520_2
506_7
$510_{7,3}$
$510_{7,5}$
506_6
$510_{6,2}$
506_5
$510_{5,3}$
$510_{5,6}$
$510_{5,7}$
506_4
$510_{4,3}$
$510_{4,5}$
506_3
$510_{3,2}$
$510_{3,3}$
$510_{3,5}$
506_2
$510_{2,2}$
$510_{2,4}$
506_1
$510_{1,1}$
$510_{1,4}$
520_1
520_3
DFRAS1块
DFRAS2块

图9

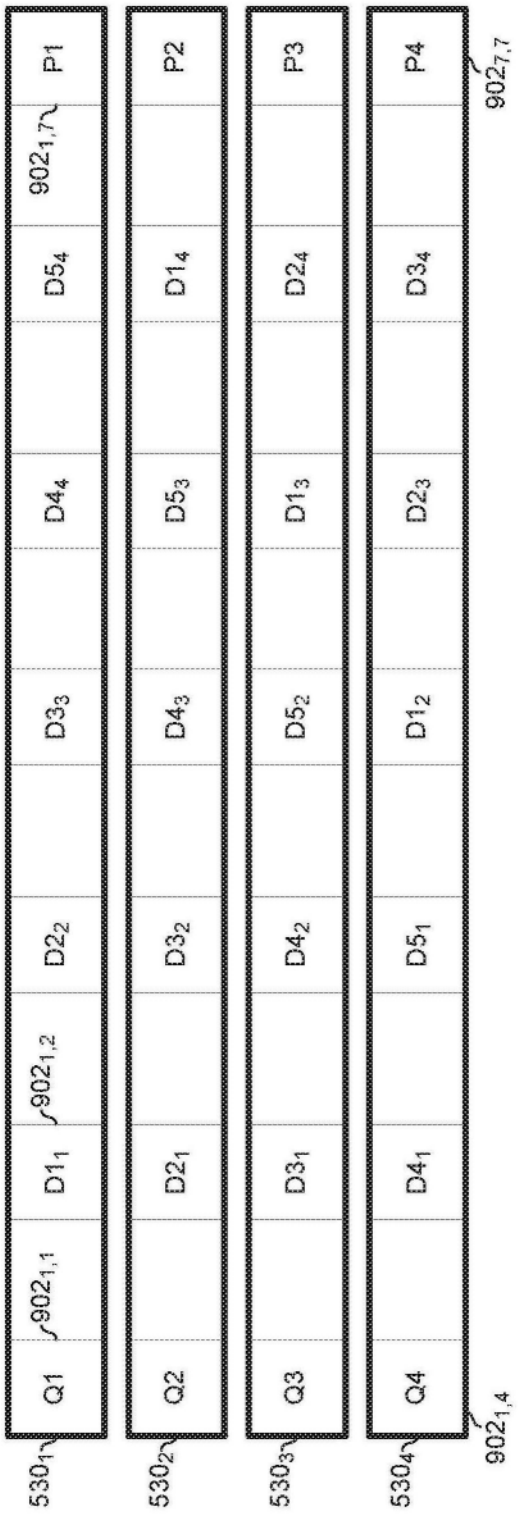

Q1
Q2
Q3
Q4
$902_{1,1}$
$902_{1,2}$
$902_{1,7}$
$902_{7,7}$
$902_{1,4}$
$D1_1$
$D2_1$
$D3_1$
$D4_1$
$D2_2$
$D3_2$
$D4_2$
$D5_1$
$D3_3$
$D4_3$
$D5_2$
$D1_2$
$D4_4$
$D5_3$
$D1_3$
$D2_3$
$D5_4$
$D1_4$
$D2_4$
$D3_4$
P1
P2
P3
P4
530_1
530_2
530_3
530_4

图10