



(21) 申请号 202280037946.8

(22) 申请日 2022.05.18

(30) 优先权数据

21175921.2 2021.05.26 EP

(85) PCT国际申请进入国家阶段日

2023.11.24

(86) PCT国际申请的申请数据

PCT/US2022/029722 2022.05.18

(87) PCT国际申请的公布数据

W02022/251009 EN 2022.12.01

(71) 申请人 微软技术许可有限责任公司

地址 美国华盛顿州

(72) 发明人 A·德拉戈耶维奇 刘俊义

(74) 专利代理机构 北京世辉律师事务所 16093

专利代理师 马明月

(51) Int.Cl.

G06F 16/901 (2006.01)

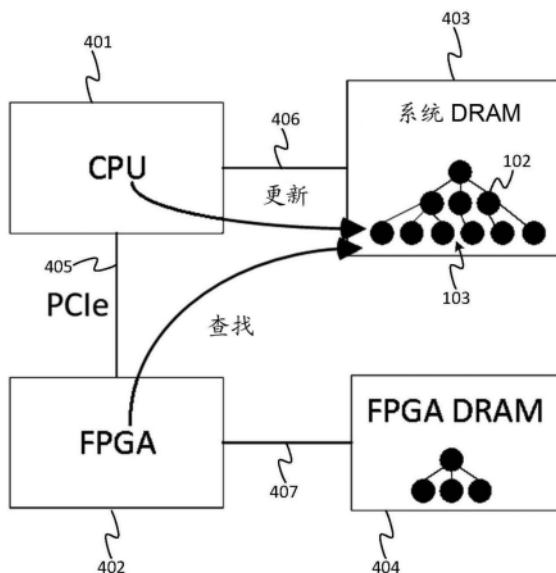
权利要求书3页 说明书15页 附图9页

(54) 发明名称

基于树的数据结构

(57) 摘要

写入器向树的叶节点写入项,并且读取器从叶节点读取项。每个节点包括相应的第一块和第二块,第一块包括按键的顺序排序的相应叶的多个项。当向叶写入新项时,写入器按写入的顺序将新项写入所标识的叶节点的第二块,而不是按键的顺序排序。当从叶读取一个或多个目标项时,读取器基于a) 在第一块中已经排序的项的顺序和b) 读取器通过相对于第一块的项的键对第二块的项进行排序,来搜索针对一个或多个目标项的叶。



1. 一种系统, 包括:

存储数据结构的存储器, 所述数据结构包括树结构, 所述树结构包括多个节点, 每个节点具有节点ID, 一些节点是叶节点, 其他节点是内部节点, 其中每个内部节点是所述树结构中一个或多个子节点的相应集合的父节点, 每个子节点是叶节点或另一个内部节点, 每个叶节点是子节点但不是父节点, 并且其中每个所述叶节点包括一个或多个项的相应集合, 每个项包括键-值对, 每个所述内部节点将其各个子节点中每个子节点的节点ID映射到由所述各个子节点涵盖的一系列键;

写入器, 所述写入器被布置为向所述叶节点写入项; 以及

读取器, 所述读取器被布置为从所述叶节点读取项;

其中每个所述叶节点包括相应的第一块和相应的第二块, 所述第一块包括相应叶节点的多个项, 所述多个项在存储器的地址空间中按键的顺序排序;

其中所述写入器被配置为, 当向所述树结构写入新项时, 通过遵循所述树结构中的键到节点ID的映射来标识要写入哪个叶节点, 并且按写入的顺序将所述新项写入所标识的叶节点的所述第二块, 而不是按键的顺序排序; 以及

其中所述读取器被配置为, 当从所述树结构读取一个或多个目标项时, 通过遵循所述树结构中的键到节点ID的映射来确定要从哪个叶节点读取, 然后基于a) 在所述第一块中已经排序的项的顺序和b) 所述读取器通过相对于所述第一块的所述项的键对所述第二块的所述项进行排序, 来搜索针对所述一个或多个目标项所确定的叶节点。

2. 根据权利要求1所述的系统, 其中所述写入器在软件中实现, 所述软件存储在所述系统的计算机可读存储器中, 并且所述写入器被布置为在所述系统的一个或多个处理器上运行。

3. 根据权利要求1或2所述的系统, 其中所述读取器在PGA、FPGA或专用硬件电路中实现。

4. 根据任一项前述权利要求所述的系统, 其中所述树结构采用B树的形式。

5. 根据权利要求4所述的系统, 其中B树是B+树, 由此, 项不存储在中间节点中。

6. 根据任一项前述权利要求所述的系统, 其中:

在至少一些叶节点和/或内部节点中, 所述相应的第一块被划分为段, 并且所述节点进一步包括快捷块, 所述快捷块包括多个捷径, 每个捷径包括i) 所述相应的第一块的对应的段的边界处的键的指示和ii) 到所述叶节点内对应的段的偏移量; 以及

所述读取器被配置为, 当执行第一块的所述搜索时, 使用所述捷径基于键仅搜索可以包含一个或多个项的段。

7. 根据任一项前述权利要求所述的系统, 其中:

所述写入器被配置为在写入时, 当写入至少一个所述叶节点时, 包括与所述相应的第二块中的每个项相关联的反向指针, 每个反向指针包括到所述第一块中的项的偏移量, 所述第一块中的项与所述第二块中的关联项相比具有下一个更大或下一个更小的键;

所述读取器被配置为, 当执行第二块的所述排序时, 使用所述反向指针通过相对于所述第一块的键对所述第二块的所述项进行排序。

8. 根据任一项前述权利要求所述的系统, 其中:

所述写入器被配置为在写入时, 当写入至少一个所述叶节点时, 包括顺序提示, 所述顺

序提示指示针对所述第二块中的每个项:当将所述项写入所述第二块时,项的键相对于所述第二块中当前键的顺序;以及

所述读取器被配置为,当执行第二块的所述排序时,使用所述顺序提示通过相对于所述相应的第一块的键对所述第二块的所述项进行排序。

9.根据权利要求8所述的系统,其中所述读取器被配置为通过以下方式使用所述顺序提示:

- 按存储顺序处理所述相应的第二块中的所述项;以及
- 使用所述顺序提示对数组中的所述项进行排序,其中所述数组存储所述第二块内的项的偏移量;

其中当每个具有顺序提示 i 的项被处理时,其在所述第二块中的偏移量被插入到所述数组中的位置 i ,并且位置 $j \geq i$ 处的所有项被向右移动,使得在整个第二块被处理之后,所述数组包含按排序顺序的项的偏移量。

10.根据任一项前述权利要求所述的系统,其中:

所述数据结构包括页表,所述页表将节点ID映射到相应的存储器地址;

所述读取器被配置为,当从确定的节点ID的节点读取时,使用所述页表来确定在存储器中确定的节点的存储器地址;以及

所述写入器被配置为,当更新节点以写入、拆分或合并所述节点时,将所述节点的更新版本写入新的存储器地址,然后一旦写入,更新所述页表中的所述相应的存储器地址,以指向所述新的存储器地址。

11.根据任一项前述权利要求所述的系统,其中:

每个第二块具有最大大小;以及

所述写入器被配置为,如果写入新项,所述新项将导致所标识的节点的所述相应的第二块超过最大大小,则将所述第二块与所述相应的第一块合并。

12.根据权利要求11的所述系统,其中:

所述数据结构包括页表,所述页表将节点ID映射到相应的存储器地址;

所述读取器被配置为,当从所确定的节点ID的节点读取时,使用所述页表来确定在存储器中确定的节点的存储器地址;以及

所述读取器被配置为,当更新节点以合并所述相应的第二块和第一块时,将包括合并的节点的更新版本写入新的存储器地址,然后一旦合并完成,更新所述页表中的所述相应的存储器地址以指向所述新的存储器地址。

13.根据任一项前述权利要求所述的系统,其中:

每个叶节点进一步指示一个或多个兄弟叶节点的节点ID,所述兄弟叶节点涵盖与所述相应叶节点的键相邻的一系列键;以及

所述读取器被配置为执行范围扫描以从一系列键中读取项,并且当执行从跨越多个叶节点的一系列项中读取的范围扫描时,以:

-通过遵循树结构中的键到节点ID的映射,确定叶节点之一,所述叶节点之一涵盖扫描范围中的键之一,然后

-通过使用在所述叶节点之一中指示的至少一个所述兄弟叶节点的ID来确定至少一个其他叶节点,所述其他叶节点涵盖所述扫描范围中的至少一个其他键。

14.一种方法,包括:

在存储器中存储数据结构,所述数据结构包括树结构,所述树结构包括多个节点,每个节点具有节点ID,一些节点是叶节点,其他节点是内部节点,其中每个内部节点是所述树结构中一个或多个子节点的相应集合的父节点,每个子节点是叶节点或另一个内部节点,每个叶节点是子节点但不是父节点,并且其中每个所述叶节点包括一个或多个项的相应集合,每个项包括键-值对,每个所述内部节点将其各个子节点中每个子节点的节点ID映射到所述各个子节点涵盖的一系列键;

向叶节点写入项;以及

从叶节点读取项;

其中每个所述叶节点包括相应的第一块和相应的第二块,所述第一块包括相应叶节点的多个项,所述多个项在存储器的地址空间中按键的顺序排序;

其中所述写入包括,当向所述树结构写入新项时,通过遵循所述树结构中的键到节点ID的映射来标识要写入到哪个叶节点,并且将按写入的顺序将所述新项写入所标识的叶节点的所述第二块,而不是按键的顺序排序;以及

其中所述读取包括,当从所述树结构读取一个或多个目标项时,通过遵循所述树结构中的键到节点ID的映射来确定要从哪个叶节点读取,然后基于a)在所述第一块中已经排序的项的顺序和b)读取器通过相对于所述第一块的所述项的键对所述第二块的所述项进行排序,来搜索针对所述一个或多个目标项所确定的叶节点。

15.一种被实施在非暂时性计算机可读介质或媒介上的计算机程序,包括被配置为当在一个或多个处理器上运行时执行权利要求14所述的写入和/或读取的代码。

基于树的数据结构

背景技术

[0001] 诸如B树之类的树结构可以用作索引数据结构,用于存储键值对(每对包括一个值,即被存储的内容,以及一个键,用于索引该值)。树的每个叶节点都包含多个项,这些项是键值对。从叶向上走,树的每个内部节点都包含其每个子节点(可以是叶或其他内部节点)所涵盖的一系列键值的指示。

[0002] 当一个新项被添加到树中时,写入器使用树结构来确定哪个叶包含新项的键所属的一系列键。如果这个叶没有满,新项可以添加到这个现有叶中。然而,如果这个叶已满(叶和内部节点通常具有最大字节大小),那么这个叶将被拆分。即创建一个新叶。这也意味着更新旧叶和新叶的父节点,以便正确引用两个叶的键范围。如果对父节点的更新也会使父节点超出其最大大小,那么父节点自身将被拆分,并更新祖父节点的引用,依此类推。如果项被删除,这也可能涉及合并叶或内部节点。

[0003] 在给定的应用程序中,写入器或读取器可能需要执行多个操作。读取器可以使用特定键读取单个项,也可以执行范围扫描以从一系列键中读取。写入器可以写入新条目、修改现有条目或删除条目。

[0004] 索引数据结构用于广泛的软件系统,例如数据存储和数据库、文件系统、操作系统等。索引数据结构存储键值对,并支持查找、扫描、插入和删除键值对等操作。B树就是这样一种索引数据结构。B树是按顺序索引,这意味着它也支持扫描操作。扫描操作返回存储在树中的所有键值对,并且键在指定范围内。

发明内容

[0005] 在基于树的数据结构(例如B树)中,需要平衡写入器和读取器之间的工作负载。例如,如果新条目仅按时间顺序添加,而不按键排序,那么这对写入器来说非常快。然而,读取器必须在读取时对项进行排序,以执行单个读取或范围扫描(至少范围扫描需要排序-单个读取可以,使用排序以相同的方式实现查找和范围扫描更简单)。另一方面,如果写入器每次执行写入时将所有新条目按排序顺序放置,这将使读取器的读取非常快,但在写入时给写入器带来更大的负担。期望提供在这两种方法之间的折衷方案。

[0006] 根据本文公开的一个方面,提供了一种系统,该系统包括存储数据结构的存储器,该数据结构包括树结构,该树结构包括多个节点,每个节点具有节点ID。一些节点是叶节点,其他节点是内部节点,其中每个内部节点是树结构中一个或多个子节点的相应集合的父节点。每个子节点是叶节点或另一个内部节点,每个叶节点是子节点但不是父节点。每个叶节点包括一个或多个项的相应集合,每个项包括键-值对,每个内部节点将其各个子节点中每个子节点的节点ID映射到各个子节点涵盖的一系列键。该系统还包括以软件、硬件或其组合实现的写入器,被布置为向叶节点写入项;以及以软件、硬件或其组合实现的读取器,被布置为从叶节点读取项。每个叶节点包括相应的第一块和相应的第二块,第一块包括相应叶节点的多个项,多个项在存储器的地址空间中按键的顺序排序。写入器被配置为,当向树结构写入新项时,通过遵循树结构中的键到节点ID的映射来标识要写入哪个叶节点,

并且按写入的顺序将新项写入所标识的叶节点的第二块,而不是键的顺序排序。读取器被配置为,当从树结构读取一个或多个目标项时,通过遵循树结构中的键到节点ID的映射来确定要从哪个叶节点读取。然后基于a)在第一块中已经排序的项的顺序和b)读取器通过相对于第一块的项的键对第二块的项进行排序,来搜索针对一个或多个目标项所确定的叶节点。

[0007] 在实施例中,第一块可以具有比第二块更大的最大大小,并且因此第一块可以被称为“大块”,以及第二块被称为“小块”。

[0008] 在实施例中,写入器可以用软件实现,但读取器可以用定制硬件实现,例如在可编程门阵列(PGA)或现场可编程门阵列(FPGA)中。然而,更一般地,写入器可以用硬件(例如,硬件加速器)来实现和/或读取器可以用软件来实现。

附图说明

[0009] 为了帮助理解本文公开的实施例,并说明如何将这样的实施例付诸实施,仅以示例的方式参考以下附图,其中:

[0010] 图1示意性地示出了针对示例键区间[14,22]的包含扫描形式的范围扫描,其中由返回项的键形成的区间是落在输入区间内的最大子集;

[0011] 图2示意性地示出了针对示例键区间[14,22]的覆盖扫描形式的范围扫描,其中由返回项的键形成的间隔是包含输入区间的最小超集;

[0012] 图3示意性地示出了针对示例键区间(14,22)的覆盖扫描形式的范围扫描;

[0013] 图4是根据本文公开的实施例的系统的示意框图;

[0014] 图5示意性地示出了包括树结构的数据结构;

[0015] 图6示意性地示出了包括小块和大块的节点;

[0016] 图7示意性地示出了将新项插入小块的方法;

[0017] 图8示意性地示出了包括顺序提示以对小块进行排序的方法;

[0018] 图9示意性地示出了基于顺序提示对小块进行排序的方法;

[0019] 图10是四元件小块排序器的示意电路图;

[0020] 图11示意性地示出了在插入操作期间合并大块和小块的方法;以及

[0021] 图12示意性地示出了在插入期间拆分叶节点的方法。

具体实施方式

[0022] 本公开提供了一种节点布局,其中每个节点由第一块和第二块组成。优选地,第一块具有比第二块更大的最大大小(以位或字节为单位),因此第一块和第二块可以分别被称为大块和小块。以下将通过参考大块和小块的示例来描述,但原则上可以更一般地分别用“第一块”和“第二块”替换。

[0023] 大块被排序,这允许高效地搜索节点。在上下文中,排序意味着项按键排序,使得项按键的顺序与项在存储器的地址空间中出现的顺序相同(取决于实现的虚拟或物理地址空间)。另一方面,小块包括节点的最新插入和删除,并按时间顺序排列,这允许快速更新。另一种选择是保持整个节点未排序,导致快速的更新性能,但搜索性能较差;或者整个节点排序,导致快速的搜索性能,但更新性能较差。

[0024] 可选地,快捷键可以在大块中使用。这允许节点搜索仅检查整个节点的一小部分,从而获得更好的性能。

[0025] 作为另一种备选或附加优化,在实施例,可以在小块中包括“反向指针”。这允许在不比较键的情况下在小块和大块中的项之间建立顺序。这导致有效的实现(例如在硬件中),因为不需要比较整个键。

[0026] 作为可以与快捷键和/或反向指针一起使用或独立于快捷键和/或反向指针使用的另一可选优化,在实施例,“顺序提示”可以包括在小块中。小块中的顺序提示允许非常有效的硬件在不比较键的情况下建立小块中项的排序顺序。

[0027] 另一个可选的优化是提供复杂更新的同步。在这种情况下,当节点被拆分或合并时,写入器通过创建子树的副本(例如线程私有副本),然后利用页表中的单个指针更新将子树交换来更新树。

[0028] 在实施例,写入器可以在运行在一个或多个处理器上的软件中实现,但读取器可以在专用硬件中实现,硬件可以包括FPGA或PGA,甚至专用(即固定功能)电路,例如ASIC(专用集成电路)。

[0029] 实现索引数据结构的专用硬件是具有专用管道来执行数据结构操作的硬件。这与实现指令集并执行使用该指令集实现数据结构操作的程序的通用CPU形成对比。专用硬件有几个优点:它具有更高的每瓦吞吐量,更可预测和更低的延迟,并且可以比在CPU上运行与软件相同的功能更便宜。使用专用硬件的主要缺点是它更难设计,构建比软件花费更多时间,并且在专用硬件的情况下,它在构建后无法更改,因此通常仅用于非常广泛部署的功能。

[0030] 然而注意,本公开的范围不限于在软件中实现写入和在硬件中实现读取,在其他可能性中,写入器可以在诸如FPGA或ASIC之类的某种形式的硬件中实现,和/或读取器可以在软件中实现。

[0031] 本文公开的实施例提供了一种从两全其美中获益的内存B树:所有操作都可以在CPU上的软件中执行,而查找和扫描操作可以在我们设计的专用硬件上执行。在许多工作负载中,查找和扫描是主要操作,这意味着大多数操作受益于硬件卸载。查找和扫描比更新操作更简单,这意味着整个系统可以更快地构建和部署。在一个示例实现中,存储器可以是基于DRAM的主机存储器,但这不是限制性的,在其他示例中,B树可以存储在例如存储NVM设备中,诸如SSD或3D-Xpoint驱动器。

[0032] 上述技术的示例将在参考图来更详细地描述。

[0033] 下面将根据B树来描述,但更一般地,本文公开的思想,诸如不同类型的读取和写入操作、大块和小块的使用、快捷键、反向指针、顺序提示和/或复杂更新(拆分和合并)的同步,可以应用于任何用于存储键值对的树结构中。

树结构

[0034] 诸如B树这样的键-值存储将数据存储为键值对(也称为项)。项的大小可以是可变的,但根据实现,数据结构可以限制项的总允许大小,例如,到512字节,或者在一些实现中可以使它们成为固定大小以简化实现。每个键与单个相应的值相关联;不允许多个键值对具有相同的键。

[0035] 图1、4和5以示例的方式示出了诸如B树的树结构103的元素。树103是数据结构的

一种形式,在存储器中实现,其可以用作键-值存储。树103包括多个节点102,其中一些是叶节点102L,另一些是内部节点102I。每个节点102具有相应的节点ID,其在本文中也可以称为逻辑ID(LID)。每个叶节点102L包括一个或多个项101的相应集合,每个项101包括键值对(作为示例,图1示出了正在接受范围扫描的节点102,但这不是限制,并且只是读操作类型的一个示例)。键值对包括键,该键充当项的索引,映射到值,该值是项的内容。图中项101内示出的数字是示例的键(在实践中,键可能会运行到更大的数字,但这只是为了说明)。

[0036] 每个内部节点102I指定内部节点的一个或多个子节点的相应集合。一旦树被构建起来,至少一个或一些内部节点将具有多个子节点(尽管当树中只有一个叶时,例如当它刚刚创建时,根节点只有一个子节点)。每个内部节点102I还指示其各个子节点所涵盖的一系列键。图5中示出的特定方案,其中键夹在指针之间,稍后讨论,只是指定此映射的一个可能示例。

[0037] 通过实现映射的任何方式,内部节点102I中的节点ID因此定义父节点和子节点之间的边,从而形成树结构。每个内部节点102I是至少一个相应子节点的父节点。一旦树被建立,至少一个或一些内部节点中的每个都是多个相应子节点的父节点(尽管如上所述,当树中只有一个叶时,根将具有单个子节点)。给定父节点的子节点可以是叶节点102L或另一个内部节点102I。叶节点102L只是子节点,而不是父节点(即叶是树的底部,或者换句话说说是树支的终点)。

[0038] 对于本文中要由子节点“涵盖”的键,意味着:子节点是包含具有该键的项(键-值对)的叶节点102L;或者子节点是另一个内部节点102I,经由其节点ID到键的映射,最终导致包含具有该键的项的叶节点102L在树的层次结构中向下一个或多个级别(或“代”)。

[0039] 树顶部的内部节点102I之一是根节点。如果写入器或读取器要写入或读取具有特定键的项,则首先在根节点中查询节点ID到指定的键范围的映射,以查找根的哪个子节点涵盖所需的键。如果该子节点本身是另一个内部节点102I,则写入器或读取器随后查询该节点中指定的映射,以查找下一代的哪个子节点涵盖所需的键,依此类推,直到找到包含具有所需键的项的叶节点102L。

[0040] 在实施例中,键比较可以遵循C memcmp函数的语义,这意味着整数键应该以大端格式存储以保持整数比较语义。然而,这只是一个可能的例子。

[0041] 在实施例中,B树可以采用B+树的形式,由此只有叶节点102L包含键值对,并且内部节点102I仅包含节点ID到键范围的映射。

读/写操作

[0042] 诸如B树之类的树结构可以支持读取操作类型:单个查找和/或范围扫描。它支持写入(更新)操作类型:插入、修改和/或删除。此类操作的示例语义如下所述。

[0043] 查找:这将一个键作为参数,如果在树中存在,则返回与该键关联的值。否则,它返回状态“未找到”。

[0044] 扫描:这将两个键作为输入参数,并从树中返回所有键位于闭合区间[低-键,高-键]的键值对。在实施例中,支持两种类型的扫描:包含扫描和覆盖扫描。图1、2和3说明了区别。(1,u)表示下限1和上限u即使在树中也不包含在结果中。[1,u]表示如果它们在树中,则它们被包含。组合[1,u),其中1被包含,但u不包含,以及(1,u],其中1不包含,但u可能包含。

[0045] 使用包含扫描(图1),由返回项的键形成的区间是落在输入区间内的最大子集。当

服务于数据库查询(如“获取所有键在36和80之间的项”)时,可以使用包含扫描。该结果不包括键为35的项,即使没有键为36的项。使用覆盖扫描(图2和3),由返回项的键形成的区间是包含输入区间的最小超集。在图2的情况下,键为22的项被返回,但在图3中不是。覆盖扫描在查询存储元数据(如“从文件开头获取存储偏移范围为550-800的数据的磁盘块”)时很有用。结果包括550之前的最高键和800之后的最小键(即,指定范围的马肩隆(wither)侧的键),这意味着返回请求范围内存储数据的所有磁盘块。

[0046] 插入:这需要插入一个新的键值对。如果树中不存在具有指定键的项,则将键值对插入树中。否则,该操作不会更新树并返回状态“已存在”。

[0047] 修改:这需要更新现有的键值对。如果树中存在具有指定键的项,则该值将更新为输入参数的值。注意,该值的大小可以与以前不同。否则,该操作不会更新树并返回状态“未找到”。

[0048] 删除:这需要删除项的键。如果树中存在该项,则将其删除。否则,操作返回状态“未找到”。

系统架构

[0049] 图4示出了根据实施例的示例系统。该系统包括写入器401、读取器402、系统存储器403以及可选地读取器402的本地存储器404。树结构103被存储在系统存储器403中。写入器401和读取器402可操作地耦合到系统存储器403。例如,读取器402可操作地经由诸如PCIe总线这样的总线405耦合到写入器401;而写入器401经由不需要经由总线通信的单独的、本地的或专用的连接406可操作地耦合到系统存储器403。写入器401可以被布置为经由本地连接406访问系统存储器403以向树103写入,而对于读取器402从树中读取,它可能必须经由总线405访问存储器403。读取器402可以经由其自己的本地连接407连接到其本地存储器404,并且可以被布置为从而将树103的一部分缓存在读取器的本地存储器404中。

[0050] 存储器403可以采取任何合适的形式,并且可以包括体现在一个或多个存储器单元中的一个或多个存储器介质。例如,存储器介质可以包括基于硅的电子介质,例如SRAM(静态运行内存)或DRAM(动态RAM)、EEPROM(电可擦除和可编程ROM)、闪存等;或者磁介质,例如磁盘或磁带;或者甚至更奇特的形式,例如可重写光学介质或合成生物存储器;或者这些和/或其他类型的任何组合。体现存储器403的存储器单元可以包括与写入器401位于同一芯片上的片上存储器单元,或者与写入器401位于同一板上的单独单元,或者外部单元,例如SSD(固态驱动器)或者可以在与写入器401相同的外壳或机架中实现或者甚至在单独的外壳或机架中的不同位置中实现的硬盘驱动器等;或者这些的任何组合。在一个特定实施例中,如图所示,存储器403包括可以在与写入器401相同的板上或芯片上实现的DRAM。例如,在一个特定示例中,系统使用插入CPU所在板上的DRAM板。但这不是限制性的。

[0051] 读取器的本地存储器404,如果使用的话,也可以采取任何合适的形式,包括任何一个或多个单元中的任何合适的介质。在一个特定的实施例中,如图所示,读取器的本地存储器404包括DRAM其可以集成到读取器402中或者实现在与读取器402相同的芯片或板上。

[0052] 在实施例中,写入器401以软件实现,被存储在系统的非暂时性计算机可读存储器中,并布置为在系统的一个或多个处理器(例如CPU)上运行;而读取器402以硬件实现,其可以采用FPGA、PGA或甚至专用(固定功能)硬件电路的形式,或其组合。可以根据这样的实施例来描述以下内容,但是应当理解,对于以下描述的任何技术,不排除写入器401可以在硬

件中实现和/或读取器402可以在软件中实现。

[0053] 用于存储写入器401的软件的存储器(和/或在备选实施例中以软件实现的读取器)可以采取任何合适的形式,诸如上文描述的那些中的任何一种,或者备选地是只读存储器(ROM),诸如基于硅的电子ROM或光学介质,诸如CDROM等。一个或多个处理器可以包括例如通用目的CPU(中央处理器);或者专用或加速器处理器,诸如GPU(图形处理器)、DSP(数字信号处理器)、密码处理器或AI加速器处理器等;或者这些和/或其它种类的任何组合。

[0054] 在一个特定实施例中,如图所示,写入器401在CPU上运行的软件中实现,而读取器402在FPGA中实现。下面的实施例可以根据这样的示例,但是应当理解,这不是限制性的。

[0055] B树将数据存储在系统存储器403中。在实施例中,这允许存储与系统存储器一样大的树(通常为数百GB),但代价是读取器402(例如FPGA)通过PCIe 405访问系统存储器。读取器(例如FPGA)402可以在其本地存储器(例如DRAM)404中保存树的上层的高速缓存,以加快执行速度。在一个示例实现中,读取器402在其本地寄存器文件中记录根节点的LID和树层级的数量。

[0056] B树也可以设计为允许在二级存储上存储数据,以允许存储更大的树。新兴的非常低延迟的NVMe(非易失性快速存储器)设备使其成为一个有吸引力的选择,即使对于低延迟场景也是如此。

[0057] 在实施例中,FPGA 402只执行查找和扫描操作,更新操作由CPU 401执行。CPU和FPGA之间的同步最好做成轻量级的。可以只有在B树节点被拆分或合并时才需要通过PCIe总线405进行通信(更多细节请参见数据结构部分)。仅支持FPGA上的读操作允许以读为主的用例从硬件卸载中受益,而无需在硬件中实现所有操作。

数据结构

[0058] 在实施例中,B树103是B+树,这意味着键值对101仅存储在叶102L中,而内部节点102I存储键指示符和指向子节点的指针。

[0059] 图5示出了一个两层树的示例,但这里描述的原则可以应用于具有任意数量层级的树(即任意数量的父级和子级)。

[0060] 包括树103的数据结构还包括页表501。页表501将节点ID(也称为逻辑ID,LID)映射到各个节点102存储在系统存储器403中的实际存储器地址。这样,写入器401或读取器402可以通过基于页表501中的节点ID查找地址,基于其节点ID(LID)在存储器403中定位节点102。

[0061] 如上所述,每个内部节点102I包括将其每个子节点ID映射到各个子节点所涵盖的一系列键的指示的信息。图5说明了如何实现这一点的一个示例。

[0062] 在该示例中,每个内部节点102I包括节点指针502的集合和键指示符503。每个节点指针502是指向根据节点ID(逻辑ID,LID)指定的子节点的指针。该示例中的每个键指示符503指定子节点所涵盖的范围的边界处的键之一。在每个内部节点102I中,具有键l在指针左边,键u在其右边的指示符503的指针502指向将键存储在区间 $[l, u)$ 中的子树。此外,每个内部节点102I存储指向子树的指针502,该子树的键低于内部节点中最低的键。这个最左边的指针存储在节点的头部603中(例如参见图6)。换句话说,每个键指示符被夹在一对指针之间,并且键指示符指定了键指示符两侧指针指向的子节点所涵盖的范围之间的边界。换句话说,内部节点存储最左边的子lid。它们基本上存储X个键和X+1个指针。最左边的是+

1个指针。在图5所示的特定示例中,将在中间叶中插入一个键为[22,65]的新项;将在最左边叶中插入键小于22的项;将在最右边叶中插入键大于或等于65的项。

[0063] 可以理解的是,图5中所示的将子节点映射到键范围的方案只是一个示例。另一个示例是例如,首先将所有边界键存储在节点中,然后使用所有LID跟随它们。

[0064] 在一些实施例中,每个叶102L还可以存储指向其左侧的叶和指向其右侧的叶的指针,以简化范围扫描的实现。

[0065] B树103从底部开始生长(就像任何其他B树一样)。如果在插入过程中,新项101无法放入它映射到的叶102中,则叶被分成两部分。旧叶的项被分成两半,指向新叶的指针被插入到父节点中。如果父节点变满,它也会被拆分。拆分可以递归地继续到树的根。如果根被拆分,则创建一个新的根,增加树的高度。当叶因为所有项都被删除而变为空时,它会从父节点中删除。备选地,另一种选择是在叶小于半满时合并叶,以保持B树的空间不变。

节点布局

[0066] 在实施例中,B树节点102的大小可以是8KB,以使它们与DRAM页大小对齐。它们可以被分配在操作系统不能移动的固定存储器中,以便允许读取器402(例如FPGA)通过使用它们的物理存储器地址来访问节点102,然而这不是必需的。

[0067] 内部节点102I优选地不直接存储子节点的地址。相反,它们存储每个节点的逻辑标识符(LID)。在一个示例中,LID的大小可以是6字节,将树的最大大小限制为 2^{61} 字节。从节点的LID到节点的虚拟和/或物理地址的映射被维护在页表501中。页表501既可以存储在系统存储器403中,也可以存储在FPGA附加存储器404中。当创建新节点的映射或者当节点的映射被改变时,写入器401(例如CPU)更新系统存储器403中的表501。如果在读取器侧保持高速缓存,则写入器401还向读取器402(例如FPGA)发出命令(例如通过PCIe)以更新读取器附加存储器404中的表的副本。备选地,读取器可以轮询树以获取更新。

[0068] 使用LID的寻址节点提供了一定程度的间接性,例如允许在NVMe设备上存储节点。它还有助于同步对树的访问,因为在某些情况下,更新操作可以执行节点的写时复制。

[0069] 在实施例中,内部节点102I和叶节点102L的布局是相同的。例如参见图6。节点头部603可以存储在,例如,节点102的前32个字节中。头部603可以包含:指定节点102的类型(内部或叶)、节点中使用的字节数、用于同步对该节点的访问的锁位和节点版本号的字段,和/或下面讨论的几个其他字段。在例如图5所示的示例中,内部节点102I的头部603还存储最左边的子节点的LID。叶节点102L的报头可以存储左兄弟节点和右兄弟节点的LID。键和值可以是可变大小的,因此它们可以存储为blob。每个blob都有一个2字节的头部,指定其大小。LID可以在没有头部的情况下存储,因为它们的大小是已知的。值可以内联存储在B树节点102L以提高顺序访问性能。大于512字节的值可以存储在B树103之外,并且指向该值的指针存储在叶102L中。

[0070] 不管选择如上例示的节点大小等特定实现参数,根据本文公开的实施例,每个叶节点102L包括两个块——大块601和小块602。为了实现这一点,可以将指向块之间边界的指针存储在节点的头部603中。例如,该指针可以表示为相对于大块601的开始的偏移量。大块601按排序顺序存储项,小块602存储最近更新的日志。当小块602变得大于阈值(例如设置为512字节)时,它与大块601合并。通过将节点102L分成大块和小块,读取操作受益于访问大部分排序数据,而没有在每次更新时对节点进行排序的开销。

[0071] 小块602中的条目可以是新插入的项或删除标记。作为可选的优化,小块602中的每个条目可以进一步存储指向大块601中的项的指针(例如2字节长)。作为方便的标签,这可以被称为“返回指针”(尽管术语“返回”不一定意味着限制到物理地址空间中的任何特定方向或类似方向)。在一个示例实现中,对于新插入的项101,指针指向具有大于添加到小块602的新项的键的大块601中的第一项。或者对于删除标记,指针指向已删除的项。指针可以表示为节点102内的偏移量。读取器402可以使用它来建立大块和小块中项目101之间的顺序,而不需要比较它们的键。这是一个有用的优化,特别是如果读取是在硬件中实现的。

[0072] 作为另一个可选的优化,小块602中的每个项可以包括一个字段(例如1字节长),该字段在项插入时将其索引存储在已排序的小块中。即,这是项在插入时的顺序。这可以被称为“顺序提示”。这些索引在搜索期间由读取器402在扫描小块时“重放”,以重建当前已排序的顺序。该顺序存储在读取器402的临时“间接”数组中,例如在FPGA寄存器中,用于访问已排序顺序的项。这使得排序更加高效,特别是如果在硬件中实现,并且不会引入显著的延迟。

[0073] 图7示出了将新项101插入到小块602中的步骤的示例。步骤a)示出了插入之前的节点102。在步骤b),项101被复制到小块602中。这对于读取器402还不可见。在步骤c),节点头部603中指示的数据的大小被改变以包括新项。

[0074] 作为另一个可选的优化,为了优化节点中对键的搜索,可以将大块601划分为大致相等大小的段。段边界处的键与指向边界的指针一起存储在快捷块604中,快捷块604可以紧跟在节点头部603之后存储。对键的搜索通过扫描快捷块并识别可能包含键的段来开始。搜索仅检查那些段和小块,这减少了从存储器中读取的数据量。例如,如果性能瓶颈是读取器(例如FPGA)402和系统存储器403之间的PCIe带宽,则此优化显著提高了性能。对于32字节的键和更小的键,搜索从8KB节点读取大约1.5KB的数据,与扫描整个节点的更直接的实现相比,性能提高了5倍。快捷键可以在大块601和小块602的合并期间选择。

节点高速缓存

[0075] FPGA 402可以有几个千兆字节的DRAM 404直接连接到它(例如4GB)。这个存储器可能不足以在所有用例中存储整个B树103,但对于前几个层级来说可能足够大,因此在实施例中它可以用作高速缓存来减少延迟和增加B树的吞吐量。例如,高速缓存404可以分配在例如8KB节点颗粒度以减少元数据量,但可以以256字节的块来获取数据,以减少通过PCIe 405读取的不需要的数据量。FPGA 402可以为每个高速缓存节点102保留32位占用位图,以跟踪高速缓存404中的块。可以通过每当更新页映射时使高速缓存页无效来维持高速缓存404和系统存储器403之间的页的一致性。当页映射被更新时,CPU 401上的软件可以通过PCIe 405发送页表更新命令,并且FPGA 402可以通过清除占用位图来使页面无效。

页表

[0076] 读取器402(例如FPGA)可以维护存储在其本地存储器404(例如DRAM)中的页表501的本地副本,本地存储器404例如可以附接到FPGA设备402。在实施例中,表中的每个条目在FPGA存储器中为8字节,但在系统DRAM中为16字节。这是因为在实施例中,系统DRAM中的副本存储两个地址(虚拟和物理),而FPGA DRAM存储器中的副本存储一个地址(仅物理)。

[0077] 页表501中的条目存储映射到它们各自LID的节点102的存储器地址。通常,这些地

址可以是物理地址或虚拟地址,具体取决于实现。在实施例中,写入器采用虚拟存储器映射,写入器侧的页表501存储映射到它们各自ID的节点的虚拟地址。在页表副本保存在FPGA 402中的实施例中,FPGA页表副本中的条目可以存储映射到它们各自LID的节点的物理地址。例如,通过8字节的条目和4GB的DRAM,系统支持高达4TB的树,这对于今天服务器上的主存储器树来说足够大。可以通过使读取器402(例如FPGA)访问系统存储器403或NVMe设备中的页表501,或者通过增加B树节点大小来支持更大的树。

[0078] 页表501在所有可能的实施例中不是绝对必要的,较不优选的备选方案是直接使用存储器地址作为内部节点102I中的节点指针。

查找

[0079] 查找从根开始,遍历每一级的内部树节点102I,找到指向下一级节点102的指针,每次查找终止于叶102L。如果叶包含键,则返回关联的值,否则返回指示键未存储在树中的状态。

[0080] 当访问内部节点102I时,查找可以最初获取节点102的第一部分(例如第一个512字节),该部分包含节点头部603和快捷块604。查找开始于搜索快捷块604以找到小于或等于目标键的最后一个键。它跟随快捷项的指针找到大块601的段。搜索从存储器中获取大块并对其进行搜索以找到小于或等于目标键的最后一个键。接下来(或并行),该查找从内存中获取小块602并对其进行搜索以找到小于或等于目标键的最后一个键。如果使用反向指针,查找遵循存储在小块和大块中找到的较大项101中的指针,而不比较键-如果小块项的反向指针指向大块项之后,则它遵循小块中的指针;否则,它遵循大块中的指针。如果目标键小于大块和小块中的第一个键,则查找遵循最左边的指针。

[0081] 当查找到达叶102I时,它以类似于内部节点的方式进行搜索。主要区别在于它在小块和大块中查找精确匹配,这意味着不需要对大块和小块中的项进行排序。

[0082] 小块搜索:小块602的搜索可以通过在检查它们之前对项101进行排序来开始。通过排序,小块的搜索可以在遇到具有键大于目标键的第一个项时立即停止。在实施例中,小块排序不比较键。相反,它使用存储在每个小块项中的顺序提示字段来建立排序。顺序提示字段在项插入时将项的索引存储在排序的小块中。在扫描小块以重建当前排序顺序的同时“重放”索引。建立的顺序存储在临时间接数组中,例如在FPGA寄存器中,而不复制项。间接数组用于访问按排序顺序中的项。排序不会引入显着的延迟,尤其是在硬件中,并且可以在搜索快捷块604和大块601的同时并行执行。在软件中,排序顺序可以保存在小的间接数组中。

[0083] 图8示出了插入节点102的示例序列。每个项101中示出的数字表示其键。为了清楚起见,大块601中的元素被省略。项上方的数字表示其顺序提示。在步骤a)中,节点中的小块602最初是空的。在步骤b)中,作为示例,插入了键是90的项,并为其分配了顺序提示0,因为它是小块中的第一个项。接下来在步骤c)中,插入了键是60的项。它是小块中最小的项,因此为其分配了顺序提示0。现有项中的顺序提示没有改变。顺序提示是根据小块的排序顺序确定的,这是在插入新项之前建立的。接下来,在步骤d),插入一个键为30的项目。它再次是小块中最小的,因此它的顺序提示也是0。在步骤e),最后一个项的键是45,它的顺序提示是1。

[0084] 图9示出了对小块602的排序。该图示出了间接数组901的状态。间接数组存储小块

中项的偏移量,但在该图中,用于说明目的,每个偏移量由其键表示。

[0085] 为了对小块602进行排序,项101按照它们存储的顺序进行处理,并使用它们的顺序提示对间接数组中的项进行排序。间接数组901存储小块中项的偏移量。当处理具有顺序提示 i 的项时,它在该小块中的偏移量被插入间接数组中的位置 i 。位置 $j \geq i$ 的所有项向右移动。在整个小块被处理后,间接数组包含按排序顺序的项的偏移量。图9示出了针对图8所示小块的间接数组排序。在一个实现中,图中的每个步骤都在FPGA上的单个循环中完成,并行与键比较。在第一步a)中,项90被插入到第一个槽中。在下一步b)中,所有项都向右移动,并且具有顺序提示0的项60被插入到第一个槽中。在接下来的步骤c)中,所有项再次向右移动,为具有顺序提示0的项30腾出空间。在最后的步骤d)中,项60和90向右移动,具有顺序提示1的项45被插入到索引为1的插槽中。最终状态e)示出了步骤a)-d)之后间接数组的状态。

[0086] 图10示出了间接数组排序器的示例实现。它示出了一个4项宽的数组。该排序器包括9位宽的寄存器。寄存器的输入是其输出,寄存器左侧的输出或输入偏移量值。通过将输入索引与寄存器的(常数)索引进行比较来选择寄存器输入。如果它们相等,则选择输入偏移量;如果输入索引小于寄存器的索引,则选择左侧寄存器的输出;如果输入索引更大,则将寄存器的输出写入自身。此示例实现不覆盖在小块中删除项的情况,但为了适应寄存器的输入可以进行调整,以便也将寄存器的输出带到右侧。

[0087] 当然可以理解,图10中所示的布局仅仅是一种可能的硬件实现方式,一旦给出本文的公开内容,本领域技术人员就可以使用传统的设计工具来设计其他。此外,读取侧401上的软件实现也是可能的。

[0088] 在CPU 401上实现排序可以遵循类似的方法。间接数组很小,很容易放入L1缓存中,因此操作很快。主要区别在于将数组中的项向右移动需要多个周期。

范围扫描

[0089] 范围扫描操作从根遍历树103,如上所述,以找到目标范围的开始键。然后,它从开始键向前扫描,直到到达范围的结束键或树的结尾。

[0090] 在实施例中,为了在叶102L之间移动,范围扫描可以使用兄弟指针。在这样的实施例中,每个叶102L都有一个指向其左兄弟节点和右兄弟节点的指针,使得可以在两个方向上执行范围扫描。然而,兄弟指针的使用并不是必需的。如果不使用兄弟指针,那么扫描将返回树以找到下一个叶。在大多数情况下,它可以只查看第一个父节点并获取下一个兄弟节点的LID。但是,如果它已经走到父节点的末尾,它将前往它的父节点以找到它的兄弟节点。扫描可能必须一直走到根。

[0091] 范围扫描返回按其键排序的项101。为了保持项的排序,叶102L的扫描可以一起处理大块601和小块602。在覆盖扫描中,扫描首先找到具有最高键的项,该键小于或等于大块和小块中范围的开始。“标准”包含扫描是类似的,只是它将从大于或等于范围开始的最低键开始。无论哪种方式,扫描都会向前扫描大块和小块中的节点,寻找区间的结束,并返回在区间中的项。当在大块和小块中都找到具有大于右边界的键的项时,扫描停止。扫描时,如果下一个小项和下一个大项都在区间内,搜索可以使用小项中的返回指针来决定接下来返回哪个:如果小项指向大项,则返回小项,并且搜索移动到下一个小项;否则,返回大项,并且搜索移动到下一个大项。为了处理快捷键604,扫描可以跟踪当前大段的结束。在这种情况下,在下一个段的开头,从快捷块中检索键,从大块中检索值。在段的中间,键和值在大

块中。

插入

[0092] 插入操作从树遍历开始,找到要在其中插入新项101的节点120L。遍历类似于查找过程中的遍历。不同之处在于,在实施例,它读取所有项,而不管它们的版本如何,以实现插入和修改操作的语义。在更新节点之前,写入器401可以用写锁锁定它,确保节点的状态是遍历期间观察到的状态(注意,在实施例,存储器403可以支持不同类型的锁、写锁和读锁,使用写锁并不一定意味着使用读锁)。例如。写锁可以存储在节点头部的32位锁字中,其中锁字由锁位和31位节点序列号组成,节点序列号当节点更新时递增。写入器401可以通过使用锁字上的原子比较并交换指令原子地设置锁位来获取写锁,同时保持序列号与遍历期间相同。如果比较并交换成功,这意味着自遍历以来节点没有改变,因此可以继续插入。如果比较并交换失败,则自遍历以来节点已经改变,因此写入器通过再次执行树遍历来重试操作。

[0093] 快速路径插入:在一种常见情况下,插入不需要拆分节点102L或合并大块601和小块602。这种常见情况的插入可以就地执行(图7)。写入器401通过首先在小块602的末尾之后复制新项,然后调整小块的大小以包括新项101来原子地附加新项。为此,写入器401可以在获取写锁后执行以下步骤:在获取锁后将项复制到小块;更新节点的大小;增加节点的序列号;并解锁节点。在实施例,锁字和头部中的小块的大小可以共享相同的64位字,因此写入器401可以递增节点版本,更新节点的大小,并用单个指令释放锁。并发读取器在复制新项时不观察新项,因为它存储在头部603中当前指定的小块末尾之外。它们观察没有该项的节点,或者观察完全写入该项的节点。

[0094] 在实施例,读取器402(例如FPGA)不高速缓存叶节点102L。如果是,写入器401将不得不通过PCIe 405发出命令来使高速缓存404无效,这将为常见情况的插入引入额外的开销。

[0095] 大-小合并:如果小块602变得太大,写入器401合并大块601和小块602(见图11)。它为节点102L分配新的存储器缓冲区,并将节点中的所有项101排序到新缓冲区中的大块中。在实施例,新项被添加到合并的大块中(而不是新的小块)。即,新节点的小块在合并后是空的。原则上可以采用另一种方式(即新项是添加新小块的第一个项),但是将新项存储在大块中会立即导致随着时间的推移减少的大-小合并。

[0096] 在使用快捷键604的实施例,写入器401在对项进行排序时选择快捷键(它在该点选择它们,因为大块是不可变的)。对于每个处理过的项,写入器401根据例如到目前为止复制的字节数、要复制的剩余字节数和/或键和值的平均大小来决定是将其放入快捷键还是大块。它最好最大化快捷键的数量,同时保持大块段的类似大小。

[0097] 当所有项都复制到新缓冲区中时,写入器401原子地将页表501中节点LID的映射替换为新缓冲区的地址。为了更新LID映射,在实施例,写入器401更新CPU和页表的FPGA副本中的LID条目。它获取LID条目上的软件锁,向FPGA发出页表更新命令,在FPGA命令完成后释放软件锁。父节点不需要更改,因为节点的LID保持不变。最后,写入器将旧存储器缓冲区放入垃圾回收列表中,对其解锁,并在其头部设置“已删除”标志,以确保并发写入器不会更新旧缓冲区。正在进行的操作仍然可以从旧缓冲区中读取,但如果一个操作需要更新已删除的节点,它会重试。重试时,它将使用新的节点映射。

[0098] 所使用的垃圾回收机制可以是任何合适的方案,例如用于并发系统的标准显式垃圾回收,例如RCU、基于世代的存储器管理器或风险指针。

[0099] 图11示出了在插入操作期间大块与小块合并的示例。将新项插入节点102L将导致小块602超过其最大大小,因此小块现在将与新项一起合并到大块601中。步骤a)示出插入到最右侧节点102L之前的树103。步骤b)示出了新缓冲区的分配并合并到其中。插入指向节点的旧指针(虚线)。在步骤c),页表被更新以指向新缓冲区,一旦正在进行的任何操作不需要旧缓冲区,旧缓冲区将被放入垃圾回收(GC)。

[0100] 这种方法的一个好处是,可以执行复杂的更新,例如合并,而不必在原始存储器地址中对节点的旧实例放置读锁(尽管一些操作仍然可能使用写锁,这只防止多个写入器,但允许一个写入器和一个或多个读取器访问节点)。如果读取器402在写入器401在新缓冲区中更新节点102L时但在页表更新之前尝试读取节点102L,则读取器402只需从页表501中当前指定的现有存储器地址读取旧实例。一旦完成对新缓冲区的写入(即新内存位置),写入器401然后可以快速更新页表501,以将节点的LID的相应页表条目切换到新存储器地址。在实施例中这可以在单个原子操作中完成。任何后续读取都会基于更新后的页表条目从新地址读取节点的新实例。类似的方法可用于其他复杂的更新,例如节点拆分或节点合并。

[0101] 当为LID分配新的存储器缓冲区时,它的序列号可以安全地设置为0,因为在删除和重用它之前,它确保缓冲区是任何操作都无法访问的。以一些示例大小为例,如果在小块大于512字节并且小块条目的大小至少为10字节(删除条目大小为10字节,插入条目大小至少为13字节)时合并大块601和小块602,则缓冲区的最大版本为41,这意味着31位序列号永远不会环绕。事实上,在一种实现中,最小大小实际上平均为6字节,但即使使用1字节条目,序列号也不会溢出。

节点拆分

[0102] 如果叶102L中没有足够的空间用于插入,则写入器401将叶一分为二,并将指向新项的指针插入到父节点中(图12)。如果父节点已满,则写入器401也会拆分父节点。拆分可以通过这种方式向上传播到根。更新但未拆分的内部节点102I称为拆分的根。写入器通过更新页表501中拆分的根的映射来替换拆分的根下方的整个子树。在更新任何数据之前,写入器可以在通过操作正在更新的所有节点102上获取写锁(这些是防止其他写入器干扰的锁)。如果由于节点版本不匹配而导致锁定失败,则写入器释放所有获取的写锁并重新启动操作。写入器还分配所有需要的存储器缓冲区来完成操作。它为它拆分的每个节点分配两个新节点(具有新LID和新缓冲区),并为拆分的根分配一个存储器缓冲区。如果存储器分配失败,因为系统存储器不足,写入器可以放弃并将适当的状态代码返回给调用者。在获取任何锁并分配存储器后,写入器从叶向上处理节点,将它们拆分进新分配的缓冲区。拆分产生的两个节点中的每一个都最终从原始节点获得大约一半的数据。写入器在拆分时合并大块和小块。当到达拆分的根时,写入器将其复制到一个新的存储器缓冲区中。它修改新缓冲区以包括指向新子节点的指针和节点之间边界处的键。要交换两个子树,写入器用其新的存储器缓冲区的地址更新页表中拆分的根的映射。以这种方式交换子树确保读取器观察旧子树,或观察新子树。然后,写入器将所有被拆分的节点的存储器缓冲区、它们的LID和拆分的根的旧的存储器缓冲区放入垃圾回收列表。最后,它解锁所有存储器缓冲区并标记它们已删除。

[0103] 图12示出了在插入过程中拆分节点的示例。虚线框表示新分配的节点缓冲区。填充框表示具有新LID和存储器缓冲区的新节点。步骤a) 示出了拆分之前的树103。最右边的节点正在被拆分。拆分的根是树的根。在步骤b), 为拆分分配了两个新节点, 为拆分的根分配了一个新缓冲区。在步骤c), 更新拆分的根的映射, 并将节点放入垃圾回收 (GC) 列表。

[0104] 在实施例中, 写入器还更新兄弟叶指针, 这些指针在扫描操作期间使用。它锁定兄弟叶, 并在新子树中交换后更新指针。即使兄弟指针和拆分的根可能不是原子更新的, 读取器观察到树的一致状态。

[0105] 如果树的根不能容纳一个新项, 写入器将树的根拆分并创建一个新的根, 增加树的高度。写入器如上执行拆分, 它为新的根分配LID和存储器缓冲区, 而不仅仅是为拆分的根分配存储器缓冲区。它用最左边的指针设置为旧的根的左半部分和一个指向旧的根右半部分的单个项来初始化新的根。然后, 它可以向读取器 (例如FPGA) 发送命令以更新根和树的高度。在一个示例实现中, 读取器402在其本地寄存器文件而不是在高速缓存404中, 记录根节点的LID和树级别的数量。高速缓存中的数据副本, 例如较旧的根节点, 当其LID重新用于另一个物理节点时, 最终将失效, 或者可能会被逐出, 因为短时间后不会访问旧数据。所以高速缓存只需要专注于维护缓存节点数据的一致性。

删除

[0106] 删除项101类似于插入新项目。写入器401将删除条目插入到小块602中。该条目指向它正在删除的项, 因此在查找操作期间忽略已删除的项。在大块601和小块602的合并期间, 回收已删除项占用的空间。如果节点102变为空, 则将其删除。节点删除可以类似于节点拆分从叶102L沿树103向上进行。在实施例中, 使用与上述相同的技术确保其原子性。

修改

[0107] 修改操作可以作为插入和删除的组合来执行。写入器401将旧项的删除条目和新项附加到小块602, 并通过更新节点的头部603来原子地发布它们。

总结

[0108] 可以理解的是, 以上实施例仅通过示例的方式进行了描述。

[0109] 更一般地, 根据本文公开的一个方面, 提供了一种系统, 包括: 存储数据结构的存储器, 该数据结构包括树结构, 该树结构包括多个节点, 每个节点具有节点ID, 一些节点是叶节点, 其他节点是内部节点, 其中每个内部节点是树结构中一个或多个子节点的相应集合的父节点, 每个子节点是叶节点或另一个内部节点, 每个叶节点是子节点但不是父节点, 并且其中每个叶节点包括一个或多个项的相应集合, 每个项包括键-值对, 每个内部节点将其各个子节点中每个子节点的节点ID映射到各个子节点涵盖的一系列键。该系统还包括: 写入器, 该写入器被布置为向叶节点写入项; 以及读取器, 该读取器被布置为从叶节点读取项。每个叶节点中包括相应的第一块和相应的第二块, 第一块包括相应叶节点的多个项, 多个项在存储器的地址空间中按键的顺序排序。写入器被配置为, 当向树结构写入新项时, 通过遵循树结构中的键到节点ID的映射来标识要写入哪个叶节点, 并且按写入的顺序将新项写入所标识的叶节点的第二块, 而不是按键的顺序排序。读取器被配置为, 当从树结构读取一个或多个目标项时, 通过遵循树结构中的键到节点ID的映射来确定要从哪个叶节点读取, 然后基于a) 在第一块中已经排序的项的顺序和b) 读取器通过相对于第一块的项的键对第二块的项进行排序, 来搜索针对一个或多个目标项所确定的叶节点。

[0110] 在实施例中,写入器可以在软件中实现,该软件存储在系统的计算机可读存储器中,并且写入器被布置为在系统的一个或多个处理器上运行。在实施例中,读取器可以在PGA、FPGA或专用硬件电路中实现。然而,备选地,写入器可以在硬件中实现,和/或读取器可以在软件中实现,或者可以在软件和硬件的组合中实现。

[0111] 在实施例中,树结构可以采取B树的形式;例如B+树,由此,项不存储在中间节点中。

[0112] 在实施例中,在至少一些叶节点和/或内部节点中,相应的第一块可以被划分为段,并且该节点进一步包括快捷块,快捷块包括多个捷径,每个捷径包括i) 相应的第一块的对应的段的边界处的键的指示和ii) 到叶节点内的对应的段的偏移量。在这样的实施例中,读取器被配置为,当执行第一块的搜索时,使用捷径基于键仅搜索可以包含一个或多个项的段。

[0113] 在实施例中,写入器被配置为在写入时,当写入至少一个叶节点时,包括与相应的第二块中的每个项相关联的反向指针,每个反向指针包括到第一块中的项的偏移量,第一块中的项与第二块中的关联项相比具有下一个更大或下一个更小的键。在这样的实施例中,读取器被配置为,当执行第二块的排序时,使用反向指针通过相对于相应的第一块的键对第二块的项进行排序。

[0114] 在实施例中,写入器被配置为在写入时,当写入至少一个叶节点时,包括顺序提示,顺序提示指示针对第二块中的每个项:当将项写入第二块时,项的键相对于第二块中当前键的顺序。在这样的实施例中,读取器被配置为,当执行第二块的排序时,使用顺序提示通过相对于相应的第一块的键对第二块的项进行排序。

[0115] 例如,读取器被配置为通过以下方式使用顺序提示:按存储顺序处理相应的第二块中的项;以及使用顺序提示对数组中的项进行排序,其中数组存储第二块内的项的偏移量。当每个具有顺序提示i的项被处理时,其在第二块中的偏移量被插入到数组中的位置i,并且位置 $j \geq i$ 处的所有项被向右移动,使得在整个第二块被处理之后,数组包含按排序顺序的项的偏移量。

[0116] 在实施例中,数据结构包括页表,页表将节点ID映射到相应的存储器地址。在该情况下,读取器被配置为,当从确定的节点ID的节点读取时,使用页表来确定在存储器中确定的节点的存储器地址。在一些实施例中,写入器被配置为,当更新节点以写入、拆分或合并节点时,将节点的更新版本写入新的存储器地址,然后一旦写入,更新页表中的相应的存储器地址,以指向新的存储器地址。

[0117] 在实施例中,每个第二块具有最大大小。写入器被配置为,如果写入新项,该新项将导致所标识的节点的相应的第二块超过最大大小,则将第二块与相应的第一块合并。

[0118] 在实施例中,数据结构包括页表,页表将节点ID映射到相应的存储器地址,读取器被配置为,当更新节点以合并相应的第二块和第一块时,将包括合并的节点的更新版本写入新的存储器地址,然后一旦合并完成,更新页表中的相应的存储器地址以指向新的存储器地址。

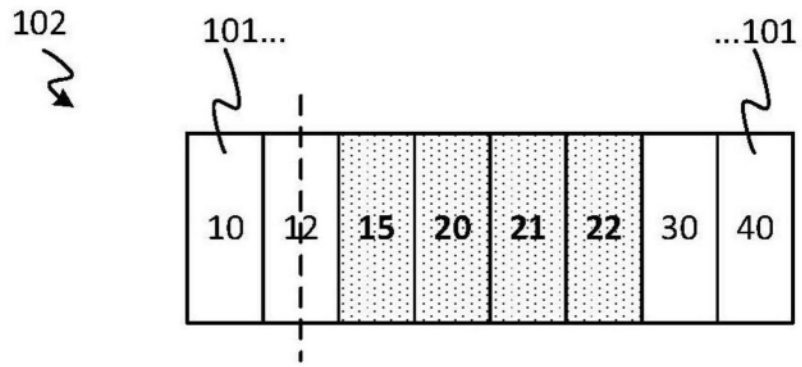
[0119] 在实施例中,每个叶节点进一步指示一个或多个兄弟叶节点的节点ID,该兄弟叶节点涵盖与相应叶节点的键相邻的一系列键。并且读取器被配置为执行范围扫描以从一系列键中读取项。在这样的实施例中,读取器被配置为当执行从跨越多个叶节点的一系列项

中读取的范围扫描时,以:通过遵循树结构中的键到节点ID的映射,确定叶节点之一,该叶节点之一涵盖扫描范围中的键之一,然后通过使用在叶节点之一中指示的至少一个兄弟叶节点的ID来确定至少一个其他叶节点,其他叶节点涵盖扫描范围中的至少一个其他键。

[0120] 根据本文公开的另一方面,提供了一种方法,包括根据本文公开的任何实施例的存储器、写入器和/或读取器的操作。

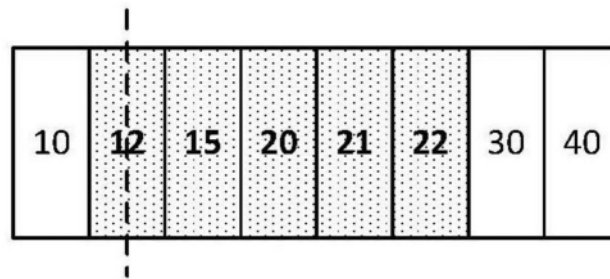
[0121] 根据另一方面,可以提供一种实施在非暂时性计算机可读介质或媒介上的计算机程序,包括被配置为当在一个或多个处理器上运行时执行写入器的写入和/或读取器的读取的代码。

[0122] 一旦给出本文的公开内容,所公开技术的其它变型或应用对本领域技术人员来说可能变得显而易见,本公开的范围不限于所描述的实施例,而仅限于所附的权利要求。



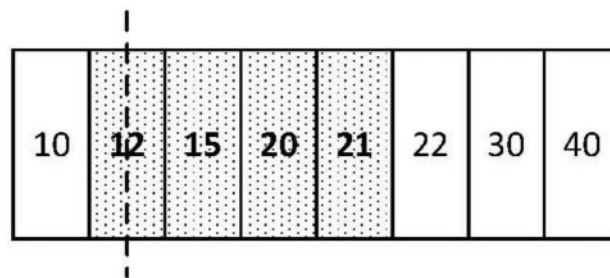
针对 [14,22] 的包含扫描

图1



针对 [14,22] 的覆盖扫描

图2



针对 (14,22) 的覆盖扫描

图3

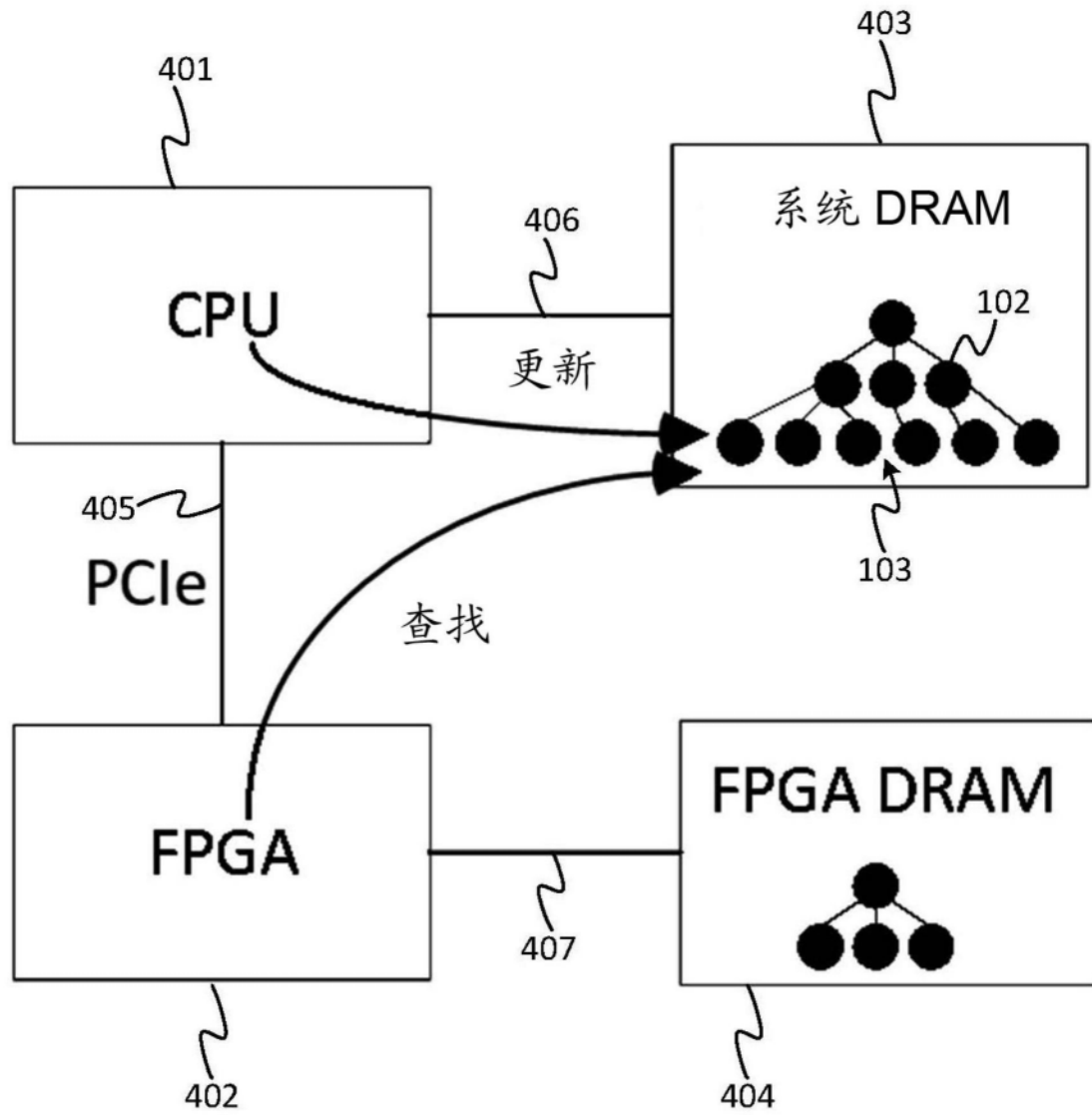


图4

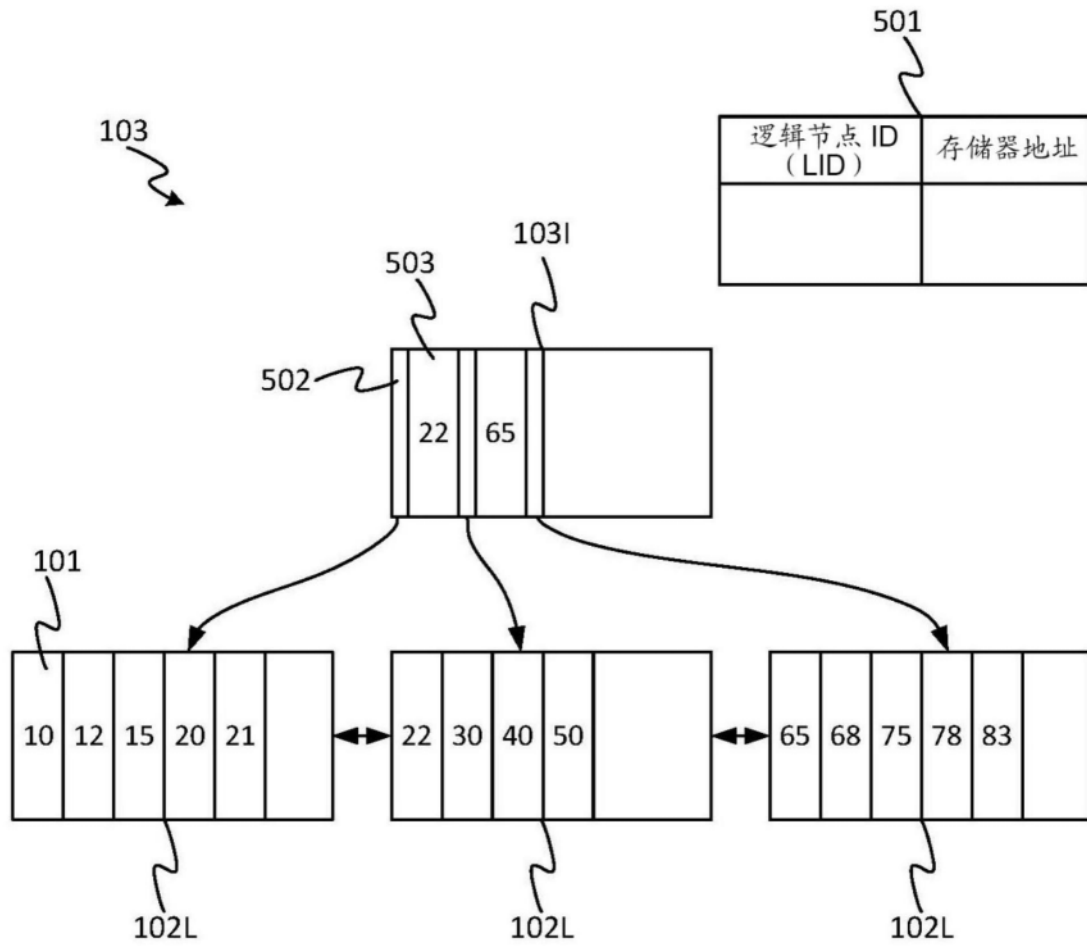


图5

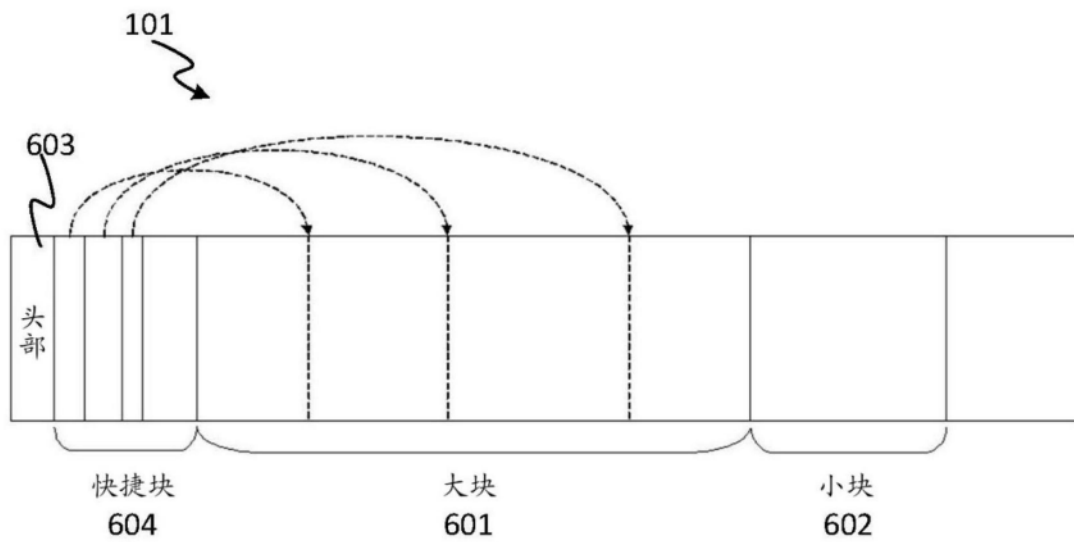


图6

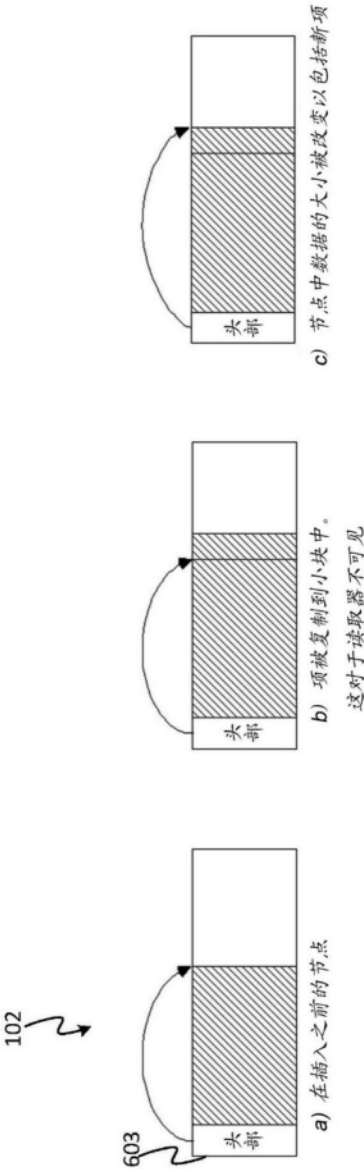


图7

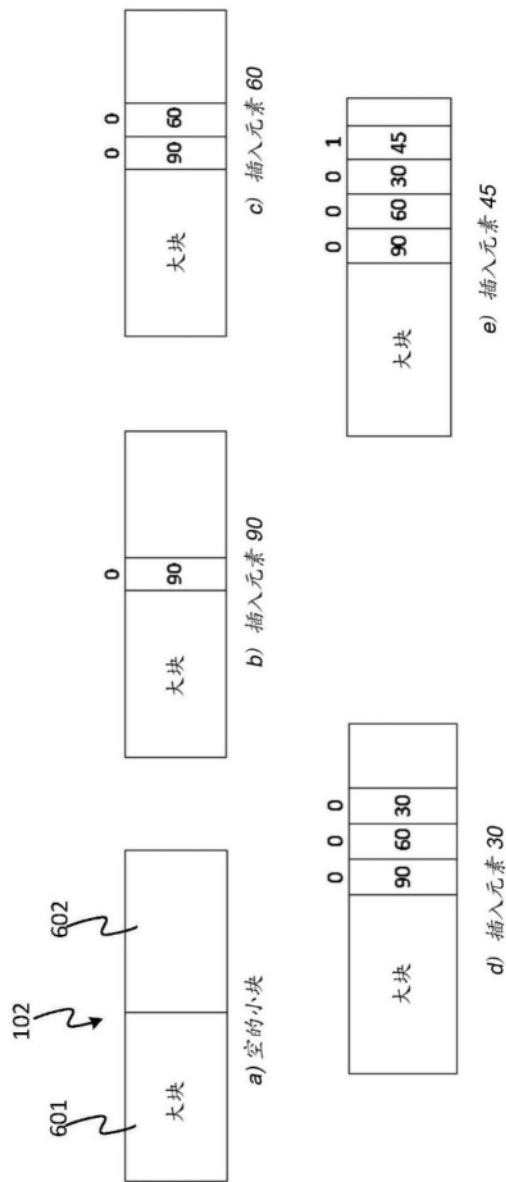


图8

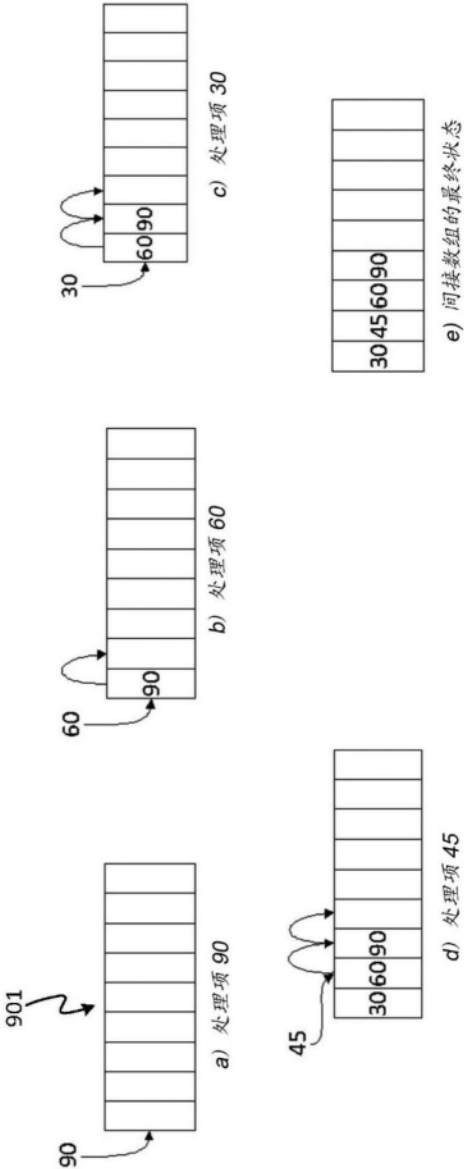


图9

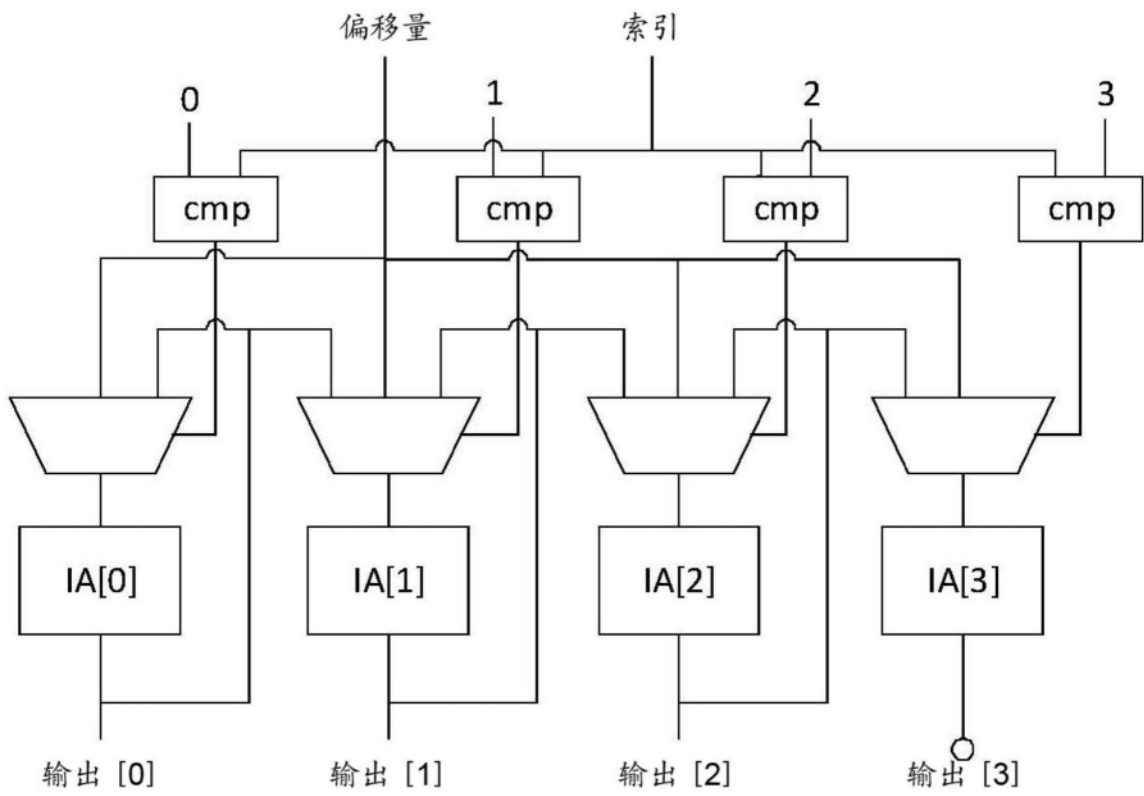


图10

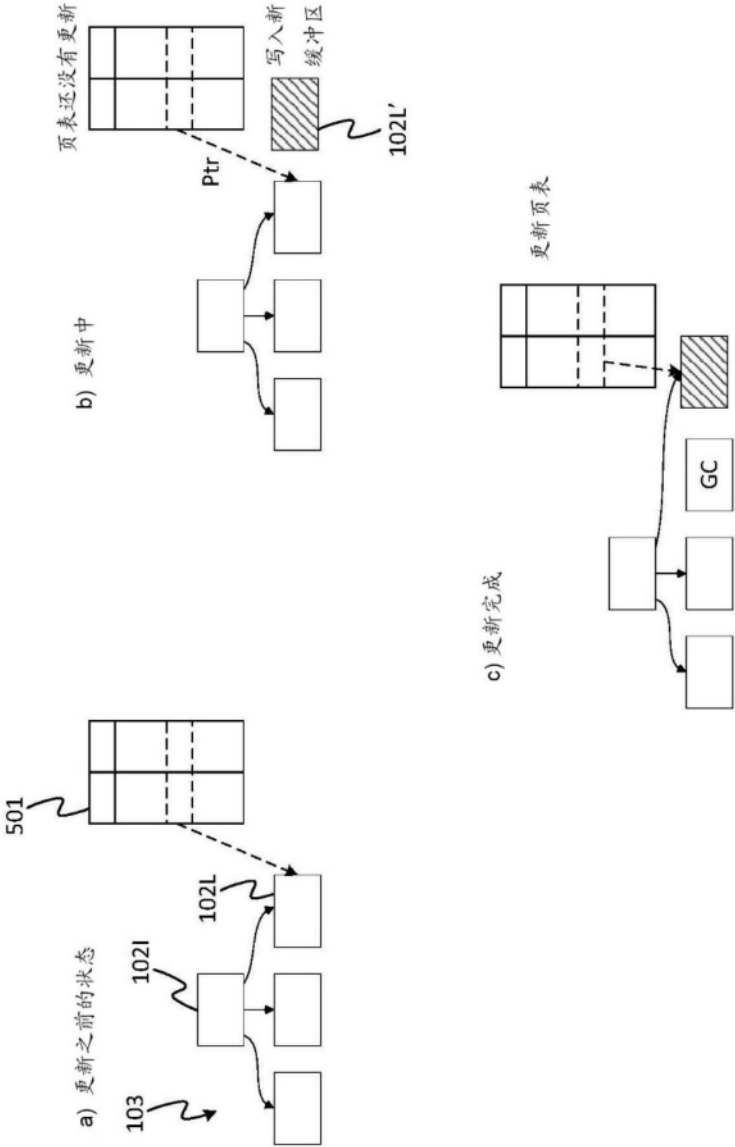


图11

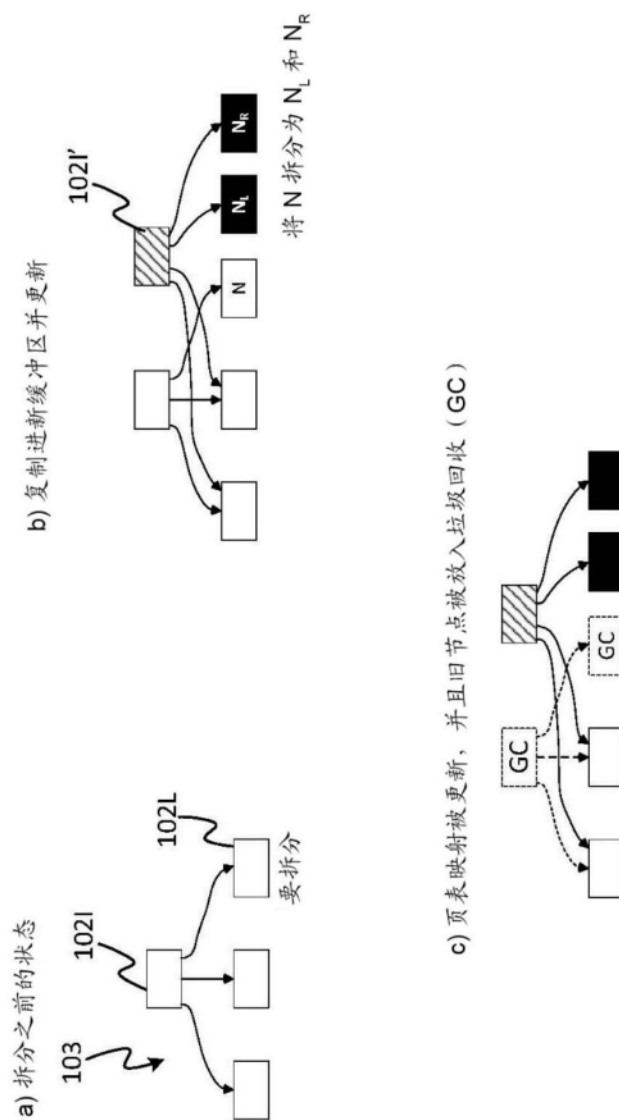


图12