(19)国家知识产权局

(12)发明专利

(10)授权公告号 CN 115905246 B
(45)授权公告日 2023.05.09

(21)申请号 202310238897.0
(22)申请日 2023.03.14
(65)同一申请的已公布的文献号
申请公布号 CN 115905246 A
(43)申请公布日 2023.04.04
(73)专利权人 智者四海(北京)技术有限公司
地址 100000 北京市海淀区学院路甲5号1幢三层1#厂房3-011
(72)发明人 高天一 代晓磊 姜诚 谢丹博
(74)专利代理机构 北京超凡宏宇专利代理事务所(特殊普通合伙) 11463
专利代理师 赵兴
(51)Int.Cl.
G06F 16/22(2019.01)
G06F 16/215(2019.01)
G06F 16/2455(2019.01)
G06F 9/50(2006.01)
审查员 张骞

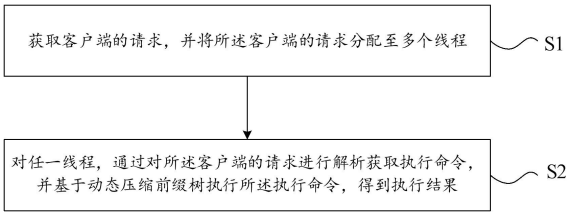
权利要求书2页 说明书8页 附图2页

(54)发明名称

基于动态压缩前缀树的KV缓存方法与装置

(57)摘要

本申请提供了一种基于动态压缩前缀树的KV缓存方法与装置,所述方法包括:获取客户端的请求,并将所述客户端的请求分配至多个线程;对任一线程,通过对所述客户端的请求进行解析获取执行命令,并基于动态压缩前缀树执行所述执行命令,得到执行结果。本申请中的动态压缩前缀树使数据存储全局有序,提高了查询效率,节省了内存空间,提高了缓存局部性,可以更好地支持区间扫描;通过增设prefix key节点进一步确保了数据完整性;通过网络层的请求分配,实现了底层数据结构的多核多线程处理;通过并发管理单元构建了并发安全的索引树,从而能够在海量的并发操作下提供强大的性能保证;通过内存回收单元实现了多线程场景下的内存释放。



CN 115905246 B

1.一种基于动态压缩前缀树的KV缓存方法,其特征在于,所述方法包括:

获取客户端的请求,并将所述客户端的请求分配至多个线程;

对任一线程,通过对所述客户端的请求进行解析获取执行命令,并基于动态压缩前缀树执行所述执行命令,得到执行结果;其中,所述动态压缩前缀树包括并发管理单元、内存回收单元、内存压缩单元;

所述并发管理单元用于为所述动态压缩前缀树的各节点调整版本号,所述版本号包括读写阻塞标志位和重试标志位;所述为所述动态压缩前缀树的各节点调整版本号,包括:

若某节点存在写操作,则在进行写操作之前和写操作之后,均对所述节点的版本号进行第一类加处理,以调整所述读写阻塞标志位的值;

若某节点存在删除操作,则在进行删除操作时,对所述节点的版本号进行第二类加处理,以调整所述重试标志位的值;

所述内存回收单元用于支持所述动态压缩前缀树实现多线程的内存回收;其中,所述支持所述动态压缩前缀树实现多线程的内存回收,具体包括:

针对任一线程,将所述线程的代计数设为最大值;

对全局预删除数据列表进行上锁操作,并遍历所有线程确定最小的代计数;

对所述全局预删除数据列表进行解锁操作,并删除所述线程对应的预删除数据列表中小于所述最小的代计数的数据。

2.根据权利要求1所述的方法,其特征在于,所述读写阻塞标志位用于判断各节点是否存在写操作以及是否阻塞其他线程的读写操作,相应的,基于所述读写阻塞标志位判断各节点是否存在写操作以及是否阻塞其他线程的读写操作的步骤具体包括:

针对任一线程当前访问的节点,获取所述节点的版本号;

基于所述节点的版本号,确定所述节点的读写阻塞标志位的值;若所述读写阻塞标志位的值为1,则表示所述节点存在写操作且已阻塞其他线程的读写操作;若所述读写阻塞标志位的值为0,则表示所述节点不存在写操作且未阻塞其他线程的读写操作。

3.根据权利要求1所述的方法,其特征在于,所述重试标志位用于判断访问各节点是否需要进行重试操作,相应的,基于所述重试标志位判断访问各节点是否需要进行重试操作的步骤具体包括:

针对任一线程当前访问的节点,获取所述节点的版本号;

基于所述节点的版本号,确定所述节点的重试标志位的值;若所述重试标志位的值为1,则表示访问所述节点需要进行重试操作;若所述重试标志位的值为0,则表示访问所述节点不需要进行重试操作。

4. 根据权利要求1所述的方法,其特征在于,所述动态压缩前缀树设有prefix key节点,所述prefix key节点用于存储与前缀相同的数据。

5.根据权利要求1所述的方法,其特征在于,所述内存压缩单元用于采用路径压缩和惰性扩展对所述动态压缩前缀树进行压缩处理。

6.一种基于动态压缩前缀树的KV缓存装置,其特征在于,所述装置包括:

线程分配模块,用于获取客户端的请求,并将所述客户端的请求分配至多个线程;

线程执行模块,用于对任一线程,通过对所述客户端的请求进行解析获取执行命令,并基于动态压缩前缀树执行所述执行命令,得到执行结果;其中,所述动态压缩前缀树包括并

发管理单元、内存回收单元、内存压缩单元；

所述并发管理单元用于为所述动态压缩前缀树的各节点调整版本号，所述版本号包括读写阻塞标志位和重试标志位；所述为所述动态压缩前缀树的各节点调整版本号，包括：

若某节点存在写操作，则在进行写操作之前和写操作之后，均对所述节点的版本号进行第一类加处理，以调整所述读写阻塞标志位的值；

若某节点存在删除操作，则在进行删除操作时，对所述节点的版本号进行第二类加处理，以调整所述重试标志位的值；

所述内存回收单元用于支持所述动态压缩前缀树实现多线程的内存回收；其中，所述支持所述动态压缩前缀树实现多线程的内存回收，具体包括：

针对任一线程，将所述线程的代计数设为最大值；

对全局预删除数据列表进行上锁操作，并遍历所有线程确定最小的代计数；

对所述全局预删除数据列表进行解锁操作，并删除所述线程对应的预删除数据列表中小于所述最小的代计数的数据。

7.一种电子设备，其特征在于，所述电子设备包括存储器和处理器，所述存储器存储有计算机程序，所述处理器运行所述计算机程序时执行权利要求1至5中任一项所述的基于动态压缩前缀树的KV缓存方法。

8.一种可读存储介质，其特征在于，所述可读存储介质中存储有计算机程序，所述计算机程序在处理器上运行时执行权利要求1至5中任一项所述的基于动态压缩前缀树的KV缓存方法。

基于动态压缩前缀树的KV缓存方法与装置

技术领域

[0001] 本申请涉及数据处理技术领域，尤其涉及一种基于动态压缩前缀树的KV缓存方法与装置。

背景技术

[0002] 目前著名的中文问答社区已拥有千万级的问题、亿级的回答，而各社区对缓存的依赖规模更加庞大，日均需处理过亿缓存请求。而海量的数据请求的性能、扩展性，对线上用户体验、公司决策、产品策略等非常重要。

[0003] 为了解决复杂度较高的缓存请求，通常通过横向扩展充分利用计算机的CPU性能，业界中最常见的解决方案为Redis，但Redis仍存在不足之处，例如，Redis虽然支持I/O线程的多核，但计算逻辑一般多为单核处理，对于复杂的计算命令，多核收效甚微，并且其多核的扩展存在上限；Redis底层基于哈希表实现Key的存储，无法保证Key的有序；此外，Redis对于按照区间的范围查找效果很差，需要扫描全集。

发明内容

[0004] 本申请提供一种基于动态压缩前缀树的KV缓存方法与装置，以实现横向扩展，解决复杂度高的缓存请求，充分利用现代计算机的CPU性能。

[0005] 第一方面，本申请提供一种基于动态压缩前缀树的KV缓存方法，所述方法包括：

[0006] 获取客户端的请求，并将所述客户端的请求分配至多个线程；

[0007] 对任一线程，通过对所述客户端的请求进行解析获取执行命令，并基于动态压缩前缀树执行所述执行命令，得到执行结果；

[0008] 其中，所述动态压缩前缀树包括并发管理单元、内存回收单元、内存压缩单元，所述并发管理单元用于为所述动态压缩前缀树的各节点调整版本号，所述版本号包括读写阻塞标志位和重试标志位。

[0009] 根据本申请提供的一种基于动态压缩前缀树的KV缓存方法，所述为所述动态压缩前缀树的各节点调整版本号，包括：若某节点存在写操作，则在进行写操作之前和写操作之后，均对所述节点的版本号进行第一类加处理，以调整所述读写阻塞标志位的值；若某节点存在删除操作，则在进行删除操作时，对所述节点的版本号进行第二类加处理，以调整所述重试标志位的值。

[0010] 根据本申请提供的一种基于动态压缩前缀树的KV缓存方法，所述读写阻塞标志位用于判断各节点是否存在写操作以及是否阻塞其他线程的读写操作，相应的，基于所述读写阻塞标志位判断各节点是否存在写操作以及是否阻塞其他线程的读写操作的步骤具体包括：针对任一线程当前访问的节点，获取所述节点的版本号；基于所述节点的版本号，确定所述节点的读写阻塞标志位的值；若所述读写阻塞标志位的值为1，则表示所述节点存在写操作且已阻塞其他线程的读写操作；若所述读写阻塞标志位的值为0，则表示所述节点不存在写操作且未阻塞其他线程的读写操作。

[0011] 根据本申请提供的一种基于动态压缩前缀树的KV缓存方法，所述重试标志位用于判断访问各节点是否需要进行重试操作，相应的，基于所述重试标志位判断访问各节点是否需要进行重试操作的步骤具体包括：针对任一线程当前访问的节点，获取所述节点的版本号；基于所述节点的版本号，确定所述节点的重试标志位的值；若所述重试标志位的值为1，则表示访问所述节点需要进行重试操作；若所述重试标志位的值为0，则表示访问所述节点不需要进行重试操作。

[0012] 根据本申请提供的一种基于动态压缩前缀树的KV缓存方法，所述内存回收单元用于支持所述动态压缩前缀树实现多线程的内存回收；其中，所述支持所述动态压缩前缀树实现多线程的内存回收，具体包括：针对任一线程，将所述线程的代计数设为最大值；对全局预删除数据列表进行上锁操作，并遍历所有线程确定最小的代计数；对所述全局预删除数据列表进行解锁操作，并删除所述线程对应的预删除数据列表中小于所述最小的代计数的数据。

[0013] 根据本申请提供的一种基于动态压缩前缀树的KV缓存方法，所述动态压缩前缀树设有prefix key节点，所述prefix key节点用于存储与前缀相同的数据。

[0014] 根据本申请提供的一种基于动态压缩前缀树的KV缓存方法，所述内存压缩单元用于采用路径压缩和惰性扩展对所述动态压缩前缀树进行压缩处理。

[0015] 第二方面，本申请还提供一种基于动态压缩前缀树的KV缓存装置，所述装置包括：

[0016] 线程分配模块，用于获取客户端的请求，并将所述客户端的请求分配至多个线程；

[0017] 线程执行模块，用于对任一线程，通过对所述客户端的请求进行解析获取执行命令，并基于动态压缩前缀树执行所述执行命令，得到执行结果；

[0018] 其中，所述动态压缩前缀树包括并发管理单元、内存回收单元、内存压缩单元，所述并发管理单元用于为所述动态压缩前缀树的各节点调整版本号，所述版本号包括读写阻塞标志位和重试标志位。

[0019] 第三方面，本申请实施例还提供了一种电子设备，所述电子设备包括存储器和处理器，所述存储器存储有计算机程序，所述处理器运行所述计算机程序时，执行上述的基于动态压缩前缀树的KV缓存方法中的任一实现方式中的步骤。

[0020] 第四方面，本申请实施例还提供了一种可读存储介质，所述可读取存储介质中存储有计算机程序，所述计算机程序在处理器上运行时，执行上述的基于动态压缩前缀树的KV缓存方法中的任一实现方式中的步骤。

[0021] 综上所述，基于动态压缩前缀树的KV缓存方法与装置，实现了横向扩展，解决了复杂度高的缓存请求，充分利用了现代计算机的CPU性能。通过网络层的请求分配，实现了底层数据结构的多核多线程处理方法；通过构建动态压缩前缀树，不仅使数据存储全局有序，提高了查询效率，还节省了内存空间，提高了缓存局部性，可以更好地支持区间扫描；通过为动态压缩前缀树增设prefix key节点，进一步确保数据的完整性和准确性；通过并发管理单元调整读写阻塞标志位和重试标志位实现对各节点读写操作的并发管理，构建了并发安全的索引树，从而能够在海量的并发操作下提供强大的性能保证；通过内存回收单元实现了多线程场景下的内存释放；通过内存压缩单元进一步节省内存空间。

附图说明

[0022] 为了更清楚地说明本申请或现有技术中的技术方案，下面将对实施例或现有技术描述中所需要使用的附图作一简单地介绍，显而易见地，下面描述中的附图是本申请的一些实施例，对于本领域普通技术人员来讲，在不付出创造性劳动的前提下，还可以根据这些附图获得其他的附图。

[0023] 图1是本申请提供的基于动态压缩前缀树的KV缓存方法的流程示意图；

[0024] 图2是本申请提供的为所述动态压缩前缀树的各节点调整版本号的方法的流程示意图；

[0025] 图3是本申请提供的基于动态压缩前缀树的KV缓存装置的结构示意图；

[0026] 图4是本申请实施例提供的电子设备的结构示意图。

[0027] 图标：300-KV缓存装置；310-线程分配模块；320-线程执行模块；400-电子设备；410-存储器；420-处理器；430-总线。

具体实施方式

[0028] 为使本申请的目的、技术方案和优点更加清楚，下面将结合本申请中的附图，对本申请中的技术方案进行清楚、完整地描述，显然，所描述的实施例是本申请一部分实施例，而不是全部的实施例。基于本申请中的实施例，本领域普通技术人员在没有作出创造性劳动前提下所获得的所有其他实施例，都属于本申请保护的范围。

[0029] 对实施例中涉及的术语进行解释：

[0030] 动态压缩前缀树(Adaptive Radix Tree, ART)：以二进制位串为关键字的前缀树，是一种多叉树形结构，同时又类似多层索引表，每个中间节点包含指向多个子节点的指针数组，叶子节点包含指向实际的对象的指针。

[0031] Key值：用于检索的字符串或字节数组。

[0032] Value：Key值对应的数据。

[0033] KV缓存：KV是Key-Value的缩写，KV存储也叫键值对存储。

[0034] 根节点：树的最顶端的节点，且仅有一个。

[0035] 子节点：除根节点外，自身还有分支的节点。

[0036] 叶子节点：无分支的节点为叶子节点，也称终端节点。

[0037] 图1是本申请提供的基于动态压缩前缀树的KV缓存方法的流程示意图。如图1所示，所述基于动态压缩前缀树的KV缓存方法包括：

[0038] S1，获取客户端的请求，并将所述客户端的请求分配至多个线程。

[0039] 其中，所述获取客户端的请求，并将所述客户端的请求分配至多个线程，具体包括：

[0040] 步骤S11，获取客户端的请求，并发送至网络层；

[0041] 步骤S12，将所述客户端的请求分配至多个线程均匀的分配到不同的线程。

[0042] 具体地，由于Libuv网络库一个高性能、事件驱动的I/O网络库，其本身并不支持多核多线程的请求处理，故本申请实施例在网络层基于Libuv网络库进行了开发，并利用uv_write2函数和SO_REUSEPORT将IPC(Inter-Process Communication)信号发送给多个线程，从而将所述客户端的请求分配至多个线程均匀的分配到不同的线程，且各个线程之间互不

干扰。

[0043] 需要说明的是,为了降低数据库的压力,通常将热点数据作为缓存,并在索引层采用动态压缩前缀树进行缓存,以更好地支持网络层的多核多线程的缓存请求处理,解决复杂度较高的缓存请求,提高数据库的读写性能。其中,所述热点数据可以是为大部分用户频繁访问的数据。

[0044] S2,对任一线程,通过对所述客户端的请求进行解析获取执行命令,并基于动态压缩前缀树执行所述执行命令,得到执行结果。

[0045] 其中,所述动态压缩前缀树包括并发管理单元、内存回收单元、内存压缩单元。

[0046] 具体地,各线程对所述客户端的请求进行解析,获取对应自身线程的数据流并不断封装成一个完整命令,以得到各线程对应的执行命令;然后,各线程基于其各自的执行命令,向索引层的动态压缩前缀树中拉取或写入数据,完成对各线程的执行命令的处理,得到执行结果。值得的注意的是,不同的线程的执行命令可以同时进行,且可以去相关的索引层中拉取或写入数据。

[0047] 在一些实施例中,所述动态压缩前缀树设有prefix key节点,所述prefix key节点用于存储与前缀相同的数据,即,prefix key是指此Key值既是一个前缀树中的一个前缀,又是一个单独且完整的key值。例如,向某个空索引树中插入abca、abcc、abcd,此时的公共前缀为abc,但若需要在此索引树再插入key值“abc”,需增设prefix key节点,以存储与公共前缀“abc”相同的key值“abc”。

[0048] 在一些实施例中,所述动态压缩前缀树每个节点可以容纳的Key值是动态变化的,所述动态压缩前缀树的节点类型包括node4、node16、node48、node256,其中,node4中包含了4个固定长度的指针空间以及4个固定长度的索引位;node16中包含16个固定长度的指针空间以及16个固定长度的索引位;node48中包含48个固定长度的指针空间和256个索引位;node256中包含256个固定长度的指针空间和256个索引位。

[0049] 在一些实施例中,所述动态压缩前缀树中的节点类型为node16的节点还支持向量化计算,具体地,通过SIMD(Single Instruction Multiple Data, 单指令多数据流)指令对多数据同时进行操作,实现并行加速以提升效率,例如,通过SIMD指令一次读取[1 2 5 5],再一次读取[2 3 7 1],并执行加法操作计算出两次读取的值的结果[3 5 12 6],这个过程即为向量化计算。

[0050] 需要说明的是,所述动态压缩前缀树为全局有序,具体地,对于节点类型为node4、node16的每个节点,在数据写入时需要按照ASCII码的顺序进行有序写入,例如,针对某一索引树中存储的数据acd,若此时需要插入b,则存储时需将b插入c和d之间;对于节点类型为node48、node256的每个节点,由于数据本身包含一个有序的且长度为256的数组,所以无需调整就可按序存储所有字符。

[0051] 针对上述步骤S2,可以理解的是,所述动态压缩前缀树的并发管理单元用于为所述动态压缩前缀树的各节点调整版本号,所述版本号包括读写阻塞标志位和重试标志位。其中,所述读写阻塞标志位用于判断各节点是否存在写操作以及是否阻塞其他线程的读写操作;所述重试标志位用于判断访问各节点是否需要进行重试操作。

[0052] 在一些实施例中,所述并发管理单元为基于乐观锁设计的管理单元,通过为所述动态压缩前缀树的各节点引入版本号,并基于版本号的读写阻塞标志位和重试标志位进一

步管理各个节点的数据，解决多线程场景下读取数据时读操作与写操作同时存在导致读操作正确性降低、程序中断等问题。具体地，在某一线程执行命令的过程中，先获取当前访问的节点的版本号，再通过所述读写阻塞标志位的值判断当前访问的节点是否存在写操作以及当前访问的节点是否阻塞其他线程的读写操作，即，当前线程是否可以正常读取当前访问的节点的数据；通过所述重试标志位的值判断是否需要进行重试操作。

[0053] 针对上述步骤S2，还可以理解的是，所述动态压缩前缀树的内存回收单元用于支持所述动态压缩前缀树实现多线程的内存回收。其中，多线程的内存回收主要是为了解决多线程场景下，由于某一删除操作使其余线程无法读取或者读取到空值等导致的线程无法进行的问题。在本申请的实施例中，所述内存回收单元可以采用EBR(epoch based reclamation)技术，支持所述动态压缩前缀树实现多线程的内存回收。由于考虑到写操作所需要插入的数据可能使某一节点的节点类型发生变化，产生新的节点，以及删除操作可能需要删除部分节点对应的数据，此时，需要对预删除的数据进行标记，而不是立刻删除，例如，暂时记录并存储在全局预删除数据列表。

[0054] 在一些实施例中，针对任一线程，若其存在写操作或者删除操作，在线程初始化时，需先在本地创建线程信息文件thread_info，并通过deletion_map_lock对所述全局预删除数据列表deletion_map进行上锁操作，以防止其他线程修改所述全局预删除数据列表deletion_map，同时将各线程对应的线程信息文件thread_info加入全局预删除数据列表deletion_map中。

[0055] 在上述实施例中，所述全局预删除数据列表deletion_map为各线程的预删除数据列表deletion_list的合集；所述线程信息文件thread_info包括线程标识、全局的代计数gobal_epoch、预删除数据信息。其中，所述线程标识可以使用phread_self函数获取，所述全局的代计数gobal_epoch从初始的0值依次递增，所述预删除数据信息为各线程需要删除的节点信息，所述将各线程对应的线程信息文件thread_info加入全局预删除数据列表deletion_map中包括：针对任一线程，将所述预删除数据信息同步到所述全局预删除数据列表deletion_map中的该线程对应的预删除数据列表deletion_list中。

[0056] 在一些实施例中，针对任一线程，若其存在写操作或者删除操作，在每次执行命令时为其设置一个线程的代计数local_epoch，且各线程在一个代的周期结束时会执行thread Exit Epoch And Cleanup函数，同时采用EBR技术实现多线程的内存回收，实现线程清理。针对存在写操作或者删除操作的任一线程，所述采用EBR技术实现多线程的内存回收，具体包括以下步骤：

[0057] 步骤a1，将线程的代计数设为最大值；

[0058] 具体地，线程的代计数local_epoch、全局的代计数gobal_epoch均为uint64类型，通过将线程的代计数local_epoch设为uint64的最大值，从而确保在首次轮询中进行内存回收的过程中清除掉代计数为第一代的数据。

[0059] 在一些实施例中，还可以基于全局的代计数gobal_epoch修改线程的代计数local_epoch；具体地，若线程的代计数local_epoch小于全局的代计数gobal_epoch，则保持线程的代计数local_epoch不变；若线程的代计数local_epoch大于全局的代计数gobal_epoch，则将线程的代计数local_epoch修改为全局的代计数gobal_epoch，以保持与全局同步。

[0060] 步骤a2,对所述全局预删除数据列表进行上锁操作,并遍历所有线程确定最小的代计数;

[0061] 具体地,通过deletion_map_lock对所述全局预删除数据列表deletion_map进行上锁操作,以防止其他线程修改全局预删除数据列表deletion_map;然后,遍历所述全局预删除数据列表中的所有线程,确定最小的代计数epoch_min。

[0062] 步骤a3,对所述全局预删除数据列表进行解锁操作,并删除各线程对应的预删除数据列表中小于所述最小的代计数的数据;

[0063] 具体地,对所述全局预删除数据列表deletion_map进行解锁操作;遍历所述全局预删除数据列表deletion_map中的当前线程对应的预删除数据列表deletion_list,筛选出小于所述最小的代计数epoch_min的代的数据,并进行删除。例如,针对线程A,设其local_epoch为4,而epoch_min为3,则在deletion_map中线程A对应的deletion_list进行遍历,将标记的预删除数据中代小于3的节点的数据进行删除。

[0064] 步骤a4,结束退出。

[0065] 针对上述步骤S2,还可以理解的是,所述动态压缩前缀树的内存压缩单元用于采用路径压缩和惰性扩展对所述动态压缩前缀树进行压缩处理。

[0066] 在一些实施例中,所述采用路径压缩和惰性扩展对所述动态压缩前缀树进行压缩处理,包括:针对所述动态压缩前缀树的叶子节点和非叶子节点,对符合第一条件的节点进行移除和合并;其中,所述第一条件为某个节点只有单一的子节点。例如,在向某一空索引树插入BAR、BAZ时,即,父节点B只有一个子节点A,此时将子节点A合并至其父节点B,并移除子节点A。

[0067] 在一些实施例中,所述采用路径压缩和惰性扩展对所述动态压缩前缀树进行压缩处理,还包括:针对所述动态压缩前缀树的叶子节点,对符合第二条件的节点进行省略;其中,所述第二条件为某个节点的子节点数少于两个。例如,在向某一空索引树插入FOO时,即,父节点F只有一个子节点O,子节点O只有一个子节点O,此时为了节省空间,可以直接省去两个内部子节点的创建,直接由将父节点F直接指向存储后缀OO的节点。

[0068] 本申请实施例提供的基于动态压缩前缀树的KV缓存方法,实现了横向扩展,解决了复杂度高的缓存请求,充分利用了现代计算机的CPU性能。通过网络层的请求分配,实现了底层数据结构的多核多线程处理方法;通过构建动态压缩前缀树,不仅使数据存储全局有序,提高了查询效率,还节省了内存空间,提高了缓存局部性,可以更好地支持区间扫描;通过为动态压缩前缀树增设prefix key节点,进一步确保数据的完整性和准确性;通过并发管理单元调整读写阻塞标志位和重试标志位实现对各节点读写操作的并发管理,构建了并发安全的索引树,从而能够在海量的并发操作下提供强大的性能保证;通过内存回收单元实现了多线程场景下的内存释放;通过内存压缩单元进一步节省内存空间。

[0069] 图2是本申请提供的为所述动态压缩前缀树的各节点调整版本号的方法的流程示意图。其中,所述版本号为uint64类型的变量,且该变量初始的64位均为0,所述读写阻塞标志位位于所述版本号的倒数第二位,用于判断各节点是否存在写操作以及是否阻塞其他线程的读写操作;所述重试标志位位于所述版本号的倒数第一位,用于判断访问各节点是否需要进行重试操作。值得注意的是,所述读写阻塞标志位的优先级高于所述重试标志位,即,在获取某一节点的版本号后,先通过所述读写阻塞标志位判断是否可以正常读取,若不

能正常读取再进一步通过所述重试标志位判断是否需要进行重试。

[0070] 如图2所示,所述为所述动态压缩前缀树的各节点调整版本号,包括:

[0071] S21,若某节点存在写操作,则在进行写操作之前和写操作之后,均对所述节点的版本号进行第一类加处理,以调整所述读写阻塞标志位的值;

[0072] 其中,第一类加处理可以是对所述节点的版本号进行+2操作。假设某一节点存在写操作,则在进行写操作之前,对该节点的版本号进行+2操作,使版本号的后三位由初始的000变为010,此时,将所述读写阻塞标志位由0变为1;在进行写操作之后,对该节点的版本号再次进行+2操作,使版本号的后三位由初始的010变为100,此时,将所述读写阻塞标志位又由1变为0。

[0073] 在一些实施例中,基于所述读写阻塞标志位判断各节点是否存在写操作以及是否阻塞其他线程的读写操作的步骤具体包括:

[0074] 步骤b1,针对任一线程当前访问的节点,获取所述节点的版本号;

[0075] 步骤b2,基于所述节点的版本号,确定所述节点的读写阻塞标志位的值;若所述读写阻塞标志位的值为1,则表示所述节点存在写操作且已阻塞其他线程的读写操作;若所述读写阻塞标志位的值为0,则表示所述节点不存在写操作且未阻塞其他线程的读写操作。

[0076] 需要说明的是,若某一节点对应的版本号的读写阻塞标志位为1,此时除正在对该节点进行写操作的线程外,其余线程并不能对该节点进行读写操作,处于等待状态,但其余线程可以通过自旋对该节点进行重复的读写尝试,在读写阻塞标志位变为0时,即可立刻进行写或者读操作。

[0077] S22,若某节点存在删除操作,则在进行删除操作时,对所述节点的版本号进行第二类加处理,以调整所述重试标志位的值。

[0078] 其中,第二类加处理可以是对所述节点的版本号进行+3操作。假设某一节点存在删除操作,则对该节点的版本号进行+3操作,使版本号的后三位由初始的000变为011,或者由100变为011,此时,将所述读写阻塞标志位由0变为1,所述重试标志位由0变为1。

[0079] 在一些实施例中,基于所述重试标志位判断访问各节点是否需要进行重试操作的步骤具体包括:

[0080] 步骤c1,针对任一线程当前访问的节点,获取所述节点的版本号;

[0081] 步骤c2,基于所述节点的版本号,确定所述节点的重试标志位的值;若所述重试标志位的值为1,则表示访问所述节点需要进行重试操作;若所述重试标志位的值为0,则表示访问所述节点不需要进行重试操作。

[0082] 需要说明的是,所述删除操作包括删除节点操作、删除路径操作,所述重试操作是指从当前访问的节点跳转到根节点重新下沉,由于乐观锁在操作数据时非常乐观,即所述重试操作的次数相对较少,不会一直进行重试,或者也可以理解成,所述第二类加处理为低频操作。

[0083] 本申请实施例提供的为所述动态压缩前缀树的各节点调整版本号的方法,通过对版本号进行第一类加处理调整所述读写阻塞标志位的值,从而实现对其他线程的读写操作的阻塞控制;通过对版本号进行第二类加处理调整所述读写阻塞标志位和所述重试标志位的值,不仅可以实现对其他线程的读写操作的阻塞控制,还可以实现对此线程是否需要进行重试操作进行控制;此外,上述通过版本号管理节点,实现对节点进行上锁操作,相对于

对整个索引树进行上锁操作而言开销较小，可以极大地提高运行效率，且还实现了对动态压缩前缀树的并发安全管理。

[0084] 图3是本申请提供的基于动态压缩前缀树的KV缓存装置的结构示意图，可以用于实现上述实施例所描述的方法。如图3所示，所述装置包括：

[0085] 线程分配模块310，用于获取客户端的请求，并将所述客户端的请求分配至多个线程；

[0086] 线程执行模块320，用于对任一线程，通过对所述客户端的请求进行解析获取执行命令，并基于动态压缩前缀树执行所述执行命令，得到执行结果；其中，所述动态压缩前缀树包括并发管理单元、内存回收单元、内存压缩单元，所述并发管理单元用于为所述动态压缩前缀树的各节点调整版本号，所述版本号包括读写阻塞标志位和重试标志位。

[0087] 上述基于动态压缩前缀树的KV缓存装置的详细描述，请参见上述实施例中相关方法步骤的描述，重复之处不再赘述。以上所描述的装置实施例仅仅是示意性的，其中所使用的作为分离部件说明的“模块”可以是实现预定功能的软件和/或硬件的组合，可以是或者也可以不是物理上分开的。可以根据实际的需要选择其中的部分或者全部模块来实现本实施例方案的目的。本领域普通技术人员在不付出创造性的劳动的情况下，即可以理解并实施。

[0088] 图4是本申请提供的电子设备的结构示意图，如图4所示，所述电子设备包括存储器410和处理器420，所述存储器410存储有计算机程序，所述处理器420运行所述计算机程序时执行基于动态压缩前缀树的KV缓存方法中的步骤。

[0089] 本申请实施例还提供了一种可读存储介质，所述可读取存储介质中存储有计算机程序，所述计算机程序在处理器上运行时，执行基于动态压缩前缀树的KV缓存方法中的步骤。

[0090] 应当理解是，所述电子设备可以是个人计算机、平板电脑、智能手机等具有逻辑计算功能的电子设备；所述可读存储介质可以是ROM(Read-Only Memory, 只读存储器)、RAM(Random Access Memory, 随机存取存储器)、磁碟、光盘等。

[0091] 最后应说明的是：以上实施例仅用以说明本申请的技术方案，而非对其限制；尽管参照前述实施例对本申请进行了详细的说明，本领域的普通技术人员应当理解：其依然可以对前述各实施例所记载的技术方案进行修改，或者对其中部分技术特征进行等同替换；而这些修改或者替换，并不使相应技术方案的本质脱离本申请各实施例技术方案的精神和范围都应涵盖在本发明的保护范围之内。

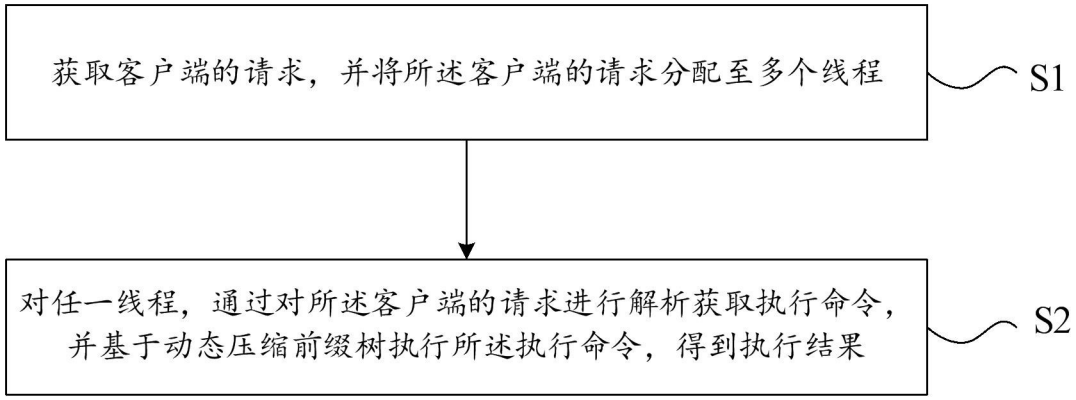

获取客户端的请求，并将所述客户端的请求分配至多个线程
S1
对任一线程，通过对所述客户端的请求进行解析获取执行命令，并基于动态压缩前缀树执行所述执行命令，得到执行结果
S2

图1

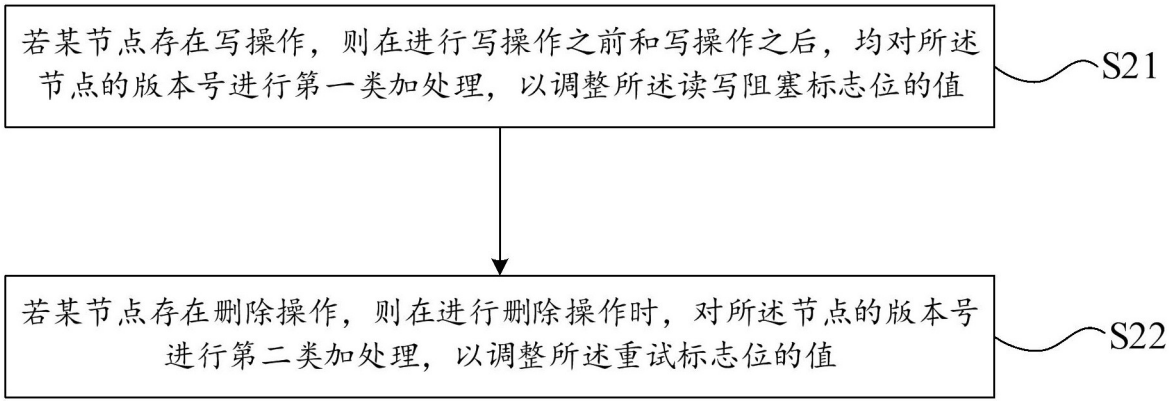

若某节点存在写操作，则在进行写操作之前和写操作之后，均对所述节点的版本号进行第一类加处理，以调整所述读写阻塞标志位的值
S21
若某节点存在删除操作，则在进行删除操作时，对所述节点的版本号进行第二类加处理，以调整所述重试标志位的值
S22

图2

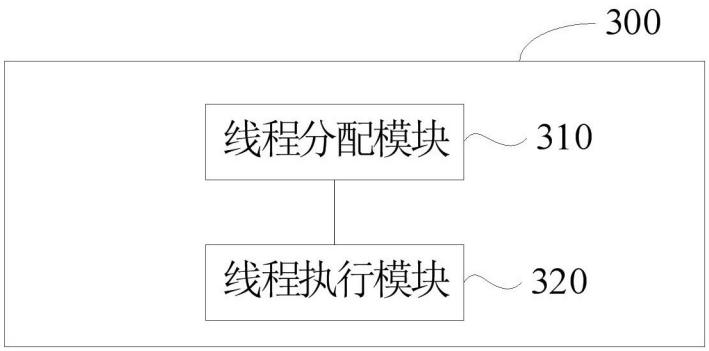

300
线程分配模块
310
线程执行模块
320

图3

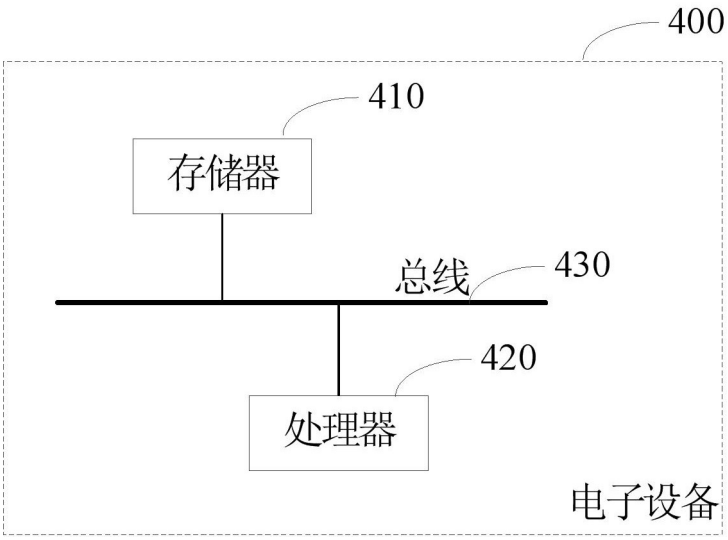


图4