(19)国家知识产权局

(12)发明专利

(10)授权公告号 CN 113821177 B
(45)授权公告日 2024.06.18

(21)申请号 202111179858.5

(22)申请日 2021.10.11

(65)同一申请的已公布的文献号
申请公布号 CN 113821177 A

(43)申请公布日 2021.12.21

(73)专利权人 中山大学
地址 510275 广东省广州市海珠区新港西路135号

(72)发明人 庄铸滔 陈志广

(74)专利代理机构 广东南北知识产权代理事务所(普通合伙) 44918
专利代理师 李思坪

(51)Int.Cl.
G06F 3/06(2006.01)

(56)对比文件
CN 112395212 A,2021.02.23

审查员 梁静静

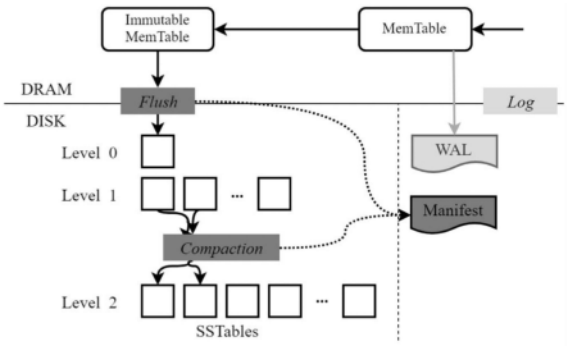
权利要求书2页 说明书6页 附图3页

(54)发明名称

一种基于NVM的LSM树的存储结构及其数据存储方法

(57)摘要

本发明公开了一种基于NVM的LSM树的存储结构及其数据存储方法,包括:DRAM层,用于将数据插入到数据结构WriteBatch并构建写入对象队列,根据写入对象队列将数据结构WriteBatch的数据写入MemTable;DISK层,用于将MemTable转换为SSTable并基于NVM对SSTable进行合并,完成写操作。本发明将NVM作为SSTabl的存储载体,以提升系统的吞吐速率。本发明作为一种基于NVM的LSM树的存储结构及其数据存储方法,可广泛应用于存储设备领域。



CN 113821177 B

1.一种基于NVM的LSM树的存储结构,其特征在于,将NVM当作SSD或HDD的替代设备的LSM树设计,包括:

DRAM层,用于将数据插入到数据结构WriteBatch并构建写入对象队列,根据写入对象队列将数据结构WriteBatch的数据写入MemTable;

DISK层,用于将MemTable转换为SSTable并基于NVM对SSTable进行合并,完成写操作;

所述基于NVM的LSM树存储结构的数据存储方法,包括写操作:

将数据插入到数据结构WriteBatch并构建写入对象队列;

根据写入对象队列将数据结构WriteBatch的数据写入MemTable;

将MemTable转换为SSTable;

基于NVM对SSTable进行合并,完成写操作;

还包括,读操作:

在Mutable MemTable中查找待索引的键;

在Mutable MemTable中未查找到待索引的键,在Immutable MemTable中查找待索引的键;

在Immutable MemTable中未查找到待索引的键,在SSTable中查找待索引的键。

2.根据权利要求1所述一种基于NVM的LSM树的存储结构,其特征在于,所述将数据结构WriteBatch的数据写入MemTable具体还包括:

判断当前MemTable是否填满;

判断到MemTable填满,将MemTable转为Immutable MemTable并新创建一个MenTable。

3.根据权利要求2所述一种基于NVM的LSM树的存储结构,其特征在于,所述将MemTable转换为SSTable这一步骤,其具体包括:

将MemTable转换为SSTable;

遍历一遍Immutable MemTable,得到Immutable MemTable的数据项数量;

新建一个数据项数量与Immutable MemTable相同的SSTable,将Immutable MemTable的数据项在SSTable的最尾部插入。

4.根据权利要求3所述一种基于NVM的LSM树的存储结构,其特征在于,所述基于NVM对SSTable进行合并,完成写操作这一步骤,其具体包括:

创建Compaction对象和SSTable数组,所述Compaction对象包括SSTable合并的记录信息,所述SSTable数组设有可容纳键值对的数目阈值;

对将要合并的SSTable构建一个迭代器,所述迭代器将将要合并的SSTable的键值对从按键的大小从小到大依次返回;

遍历该迭代器,对于每个返回的键值对,根据预设规则判断该键值对是否可丢弃;

判断到返回的键值对不可丢弃,将该键值对加入到新创建的SSTable;

当一个SSTable填满时,新创建一个SSTable,将新创建的SSTable压入SSTable数组的末端;

判断到所有不可丢弃的键值对已插入到SSTable数组中,执行安装替换,完成写操作。

5.根据权利要求4所述一种基于NVM的LSM树的存储结构,其特征在于,所述在MutableMemTable中查找待索引的键这一步骤,其具体包括:

创建一个SkipList的迭代器并通过调用该迭代器的查询功能来查找待索引的键;

返回查找结果。

6.根据权利要求5所述一种基于NVM的LSM树的存储结构,其特征在于,所述在ImmutableMemTable中未查找到待索引的键,在SSTable中查找待索引的键这一步骤,其具体包括:

在Mutable MemTable和Immutable MemTable中均没有查找到待索引的键;

从L0层开始往上层查找,直至最高层;

查找到待索引的键,输出待索引的键并结束;

没有查找到待索引的键,则输出没有找到结果。

一种基于NVM的LSM树的存储结构及其数据存储方法

技术领域

[0001] 本发明涉及存储设备领域,尤其涉及一种基于NVM的LSM树的存储结构及其数据存储方法。

背景技术

[0002] 在数据存储领域中,基于LSM树存储结构实现对数据的存储是较为常见的,目前的LSM树存储结构是以传统的块存储设备作为SSTable的存储载体,其存在的问题是灵活度不足,需要将硬盘上的数据先读入DRAM再进行操作,从而使得整个系统的读写性能一般,吞吐速率较低。

发明内容

[0003] 为了解决上述技术问题,本发明的目的是提供一种基于NVM的LSM树的存储结构及其数据存储方法,将NVM作为SSTabl的存储载体,以提升系统的吞吐速率。

[0004] 本发明所采用的第一技术方案是:一种基于NVM的LSM树的存储结构,包括以下步骤:

[0005] DRAM层,用于将数据插入到数据结构WriteBatch并构建写入对象队列,根据写入对象队列将数据结构WriteBatch的数据写入MemTable;

[0006] DISK层,用于将MemTable转换为SSTable并基于NVM对SSTable进行合并,完成写操作。

[0007] 本发明所采用的第二技术方案是:一种基于NVM的LSM树存储结构的数据存储方法,包括写操作:

[0008] 将数据插入到数据结构WriteBatch并构建写入对象队列;

[0009] 根据写入对象队列将数据结构WriteBatch的数据写入MemTable;

[0010] 将MemTable转换为SSTable;

[0011] 基于NVM对SSTable进行合并,完成写操作。

[0012] 进一步,所述将数据结构WriteBatch的数据写入MemTable具体还包括:

[0013] 判断当前MemTable是否填满;

[0014] 判断到MemTable填满,将MemTable转为Immutable MemTable并新创建一个MenTable。

[0015] 进一步,所述将MemTable转换为SSTable这一步骤,其具体包括:

[0016] 将MemTable转换为SSTable;

[0017] 遍历一遍Immutable MemTable,得到Immutable MemTable的数据项数量;

[0018] 新建一个数据项数量与Immutable MemTable相同的SSTable,将Immutable MemTable的数据项在SSTable的最尾部插入。

[0019] 进一步,所述基于NVM对SSTable进行合并,完成写操作这一步骤,其具体包括:

[0020] 创建Compaction对象和SSTable数组,所述Compaction对象包括SSTable合并的记

录信息,所述SSTable数组设有可容纳键值对的数目阈值;

[0021] 对将要合并的SSTable构建一个迭代器,所述迭代器将将要合并的SSTable的键值对从按键的大小从小到大依次返回;

[0022] 遍历该迭代器,对于每个返回的键值对,根据预设规则判断该键值对是否可丢弃;

[0023] 判断到返回的键值对不可丢弃,将该键值对加入到新创建的SSTable;

[0024] 当一个SSTable填满时,新创建一个SSTable,将新创建的SSTable压入SSTable数组的末端;

[0025] 判断到所有不可丢弃的键值对已插入到SSTable数组中,执行安装替换,完成写操作。

[0026] 进一步,还包括读操作:

[0027] 在Mutable MemTable中查找待索引的键;

[0028] 在Mutable MemTable中未查找到待索引的键,在Immutable MemTable中查找待索引的键;

[0029] 在Immutable MemTable中未查找到待索引的键,在SSTable中查找待索引的键。

[0030] 进一步,所述在Mutable MemTable中查找待索引的键这一步骤,其具体包括:

[0031] 创建一个SkipList的迭代器并通过调用该迭代器的查询功能来查找待索引的键;

[0032] 返回查找结果。

[0033] 进一步,所述在Immutable MemTable中未查找到待索引的键,在SSTable中查找待索引的键这一步骤,其具体包括:

[0034] 在Mutable MemTable和Immutable MemTable中均没有查找到待索引的键;

[0035] 从L0层开始往上层查找,直至最高层;

[0036] 查找到待索引的键,输出待索引的键并结束;

[0037] 没有查找到待索引的键,则输出没有找到结果。

[0038] 本发明方法及系统的有益效果是:本发明是将NVM作为传统块设备的替代设备的LSM树设计,将NVM作为SSTabl的存储载体,对SSTable的读写操作是直接在NVM上进行的,消除了从硬盘读入数据进DRAM的操作,大大提升了系统的吞吐速率。

附图说明

[0039] 图1是本发明一种基于NVM的LSM树的存储结构的系统框架图;

[0040] 图2是本发明具体实施例的写操作;

[0041] 图3是本发明具体实施例的读操作。

具体实施方式

[0042] 下面结合附图和具体实施例对本发明做进一步的详细说明。对于以下实施例中的步骤编号,其仅为了便于阐述说明而设置,对步骤之间的顺序不做任何限定,实施例中的各步骤的执行顺序均可根据本领域技术人员的理解来进行适应性调整。

[0043] 我们设计并实现了基于NVM的LSM树的新的存储架构,命名为NVLSM(Novel LSM)。NVLSM可以有效的利用NVM的字节可寻址特性,来减少读和写的延迟,从而可以实现较高的吞吐速率。NVLSM不采用传统的块存储设备作为SSTable的存储载体,而是将NVM作为

SSTable的存储载体,因此NVLSM的对SSTable基本数据操作单位不是块,而是字节,因此NVLSM相比传统的LSM设计有更大的灵活度,同时也充分利用的NVM的字节可寻址特性。同时,由于SSTable是存储在NVM上的,因此,不同于以往需要将硬盘上的数据先读入DRAM,再进行操作,NVLSM对SSTable的读写操作是直接在NVM上进行的,这消除了从硬盘读入数据进DRAM的操作,大大提升了系统的吞吐速率。最后,NVLSM改变了以往基于块设备的SSTable的数据结构,使得存储于NVM上的SSTable有额外的空间可以存储上层SSTable的数据,这样减少了SSTable在合并时需要对NVM的写操作次数,提升了LSM的写性能,从而提升整个系统的吞吐速率。

[0044] 参照图1,本发明提供了一种基于NVM的LSM树的存储结构,包括:

[0045] DRAM层,用于将数据插入到数据结构WriteBatch并构建写入对象队列,根据写入对象队列将数据结构WriteBatch的数据写入MemTable;

[0046] DISK层,用于将MemTable转换为SSTable并基于NVM对SSTable进行合并,完成写操作。

[0047] 一种应用于上述基于NVM的LSM树的存储结构的存储方法,参照图2,包括写操作:

[0048] 将数据插入到数据结构WriteBatch并构建写入对象队列;

[0049] 根据写入对象队列将数据结构WriteBatch的数据写入MemTable;

[0050] 具体地,在对NVLSM的写操作中,首先我们会将需要写入的数据插入到数据结构WriteBatch中,这样做的目的是可以将对数据库的多次写操作都一次性写入到WriteBatch,然后再写入到MemTable中,这样可以提高写入的速率。由于可能同时会有多个WriteBatch需要写入MemTable中,所以NVLSM有一个写入对象的队列,当需要写入数据时,当前的写入者会等待在队列里的前面的所有写入者写完,然后再进行写入。

[0051] 将MemTable转换为SSTable;

[0052] 基于NVM对SSTable进行合并,完成写操作。

[0053] 进一步作为本方法的优选实施例,所述将数据结构WriteBatch的数据写入MemTable具体还包括:

[0054] 判断当前MemTable是否填满;

[0055] 判断到MemTable填满,将MemTable转为Immutable MemTable并新创建一个MenTable。

[0056] 具体地,在对MemTable进行插入操作时,我们会给当前的写入操作预留空间,即判断当前的MemTable是不是已经填满了,如果填满了,则需要将MemTable变为Immutable MemTable,然后再新创建一个MenTable使得当前的WriteBatch中的数据项可以插入到MemTable中。在插入操作完成后,持有锁的写入者会通知其他写入者,使得下一个等待锁的写入者可以继续写入。

[0057] 进一步作为本方法的优选实施例,所述将MemTable转换为SSTable这一步骤,其具体包括:

[0058] 将MemTable转换为SSTable;

[0059] 遍历一遍Immutable MemTable,得到Immutable MemTable的数据项数量;

[0060] 新建一个数据项数量与Immutable MemTable相同的SSTable,将Immutable MemTable的数据项在SSTable的最尾部插入。

[0061] 在上述的为新插入的数据预留空间的操作中,包含了可能将Immutable MemTable转换为SSTable的操作,以及可能将SSTable的合并操作压入后台线程队列的操作。在将Immutable MemTable转换为SSTable的过程中,我们会新建一个SSTable,先遍历一遍Immutable MemTable,获得Immutable MemTable的数据项的数量,然后新建一个数据项数量与其相同的的SSTable,从而将Immutable MemTable的数据项在SSTable的最尾部插入。这样相当于把Immutable MemTable的数据项都复制进了SSTable中,因此,区别于以往要将Immutable MemTable的数据项先写入缓冲区,然后再调用系统的Write操作将缓冲区的数据写入磁盘中,NVLSM通过有效利用NVM的类似与DRAM的字节可寻址特性消除了序列化内存数据的开销。

[0062] 由于LSM树存储结构的特性,所有对NVLSM的插入和更新操作都会先放到内存中,然后随着内存的MemTable的不断填充,达到预设的上限时,才将其中的数据转存到NVM中,这一步为Flush操作。

[0063] 进一步作为本方法优选实施例,所述基于NVM对SSTable进行合并,完成写操作这一步骤,其具体包括:

[0064] 创建Compaction对象和SSTable数组,所述Compaction对象包括SSTable合并的记录信息,所述SSTable数组设有可容纳键值对的数目阈值;

[0065] 具体地,所述记录信息包括本次合并操作所在LSM的层数Lx,以及包含从Lx层选中的一个SSTable和Lx+1层的所有与被选中的SSTable键范围有重叠的SSTable的数组。

[0066] 从Lx层选取一个SSTable的策略如下:首先NVLSM会为每一层SSTable都维护一个合并指针,指针所在的位置表示下一次合并操作所要选取的SSTable的位置。合并指针会循环移动,即当指针移动到当前层的最后一个SSTable时,下一次合并指针会移动到当前层的第一个SSTable。

[0067] 对将要合并的SSTable构建一个迭代器,所述迭代器将将要合并的SSTable的键值对从按键的大小从小到大依次返回;

[0068] 遍历该迭代器,对于每个返回的键值对,根据预设规则判断该键值对是否可丢弃;

[0069] 判断到返回的键值对不可丢弃,将该键值对加入到新创建的SSTable;

[0070] 当一个SSTable填满时,新创建一个SSTable,将新创建的SSTable压入SSTable数组的末端;

[0071] 判断到所有不可丢弃的键值对已插入到SSTable数组中,执行安装替换,即将更新后的NVLSM替换当前的NVLSM版本,完成写操作。

[0072] 具体地,新建一个新版本的NVLSM,然后将没有进行合并操作的层的SSTable复制到新的NVLSM版本中,对进行合并操作的两层,我们遍历所有上述创建的SSTable数组以及原有的两层的SSTable数组,通过归并操作,将SSTable有序排放,从而完成构建一个新的版本的NVLSM。在完成上述完成SSTable的有序排放之后,我们会更新每一层的SSTable的合并值Compaction Score,即查看Lx+1层是不是需要再进行新的合并操作,同时我们也会更新每一层的合并指针,使得合并指针在键的范围空间中旋转。

[0073] 另外,还包括一种情况:在Lx层只有一个SSTable参与合并而且Lx+1层没有SSTable参与合并的情况下,即只有一个SSTable参与合并操作,简单的移动Lx层的SSTable至Lx+1层。

[0074] 进一步作为本方法优选实施例,参照图3,还包括读操作:

[0075] 在Mutable MemTable中查找待索引的键;

[0076] 具体地,Mutable MemTable的意思是当前的MemTable仍然可以进行插入操作,当Mutable MemTable的数据量超过阈值时,Mutable MemTable会变为Immutable MemTable,此时MemTable无法再进行插入或者更新操作,只能够进行读操作。

[0077] 在Mutable MemTable中未查找到待索引的键,在Immutable MemTable中查找待索引的键;

[0078] 具体地,如果当前DRAM中存在Immutable MemTable,则会在ImmuTable MemTable中进行查询,查询的过程和Mutable MemTable的一样。

[0079] 在Immutable MemTable中未查找到待索引的键,在SSTable中查找待索引的键。

[0080] 进一步作为本方法优选实施例,所述在Mutable MemTable中查找待索引的键这一步骤,其具体包括:

[0081] 创建一个SkipList的迭代器并通过调用该迭代器的查询功能来查找待索引的键;

[0082] 返回查找结果。

[0083] 进一步作为本方法优选实施例,所述在Immutable MemTable中未查找到待索引的键,在SSTable中查找待索引的键这一步骤,其具体包括:

[0084] 在Mutable MemTable和Immutable MemTable中均没有查找到待索引的键;

[0085] 从L0层开始往上层查找,直至最高层;

[0086] 查找到待索引的键,输出待索引的键并结束;

[0087] 没有查找到待索引的键,则输出没有找到结果。

[0088] 具体地,区别于传统面向SSD或HDD等块设备的SSTable查找会用Bloom Filter来加速查找过程,由于NVM本身具有较快的读速度(读速度与DRAM基本一致),因此在对SSTable的查找中,我们采用简单的二分查找,即首先,由于除L0层外,其他层的SSTable的键的范围不会重合,因此,我们采用二分查找确定待索引的键在一层的哪SSTable中,然后,对该SSTable,我们采用二分查找到第一个键大于或者等于给顶用户给的键的键值对(因为在内部存储的键值对的键额外加上了键的类型即是否删除和键的序列号),然后比较待索引的键,如果相同,则根据键的类型即是否被删除而返回未找到或者返回值。对于位于L0层中的SSTable,由于它们的键的范围之间可能会相交,因此我们会对每一个SSTable都采用二分查找判断待索引的键是不是在当前的SSTable中,由于L0层的SSTable数量并不是很多,因此这是可是忍受的开销。由于NVLSM对SSTable的查找全部在NVM中进行,因此在SSTable中的查找不需要对磁盘的解序列化操作,从而消除了原有LSM的序列化/解序列化操作,也不需要先对索引块进行查找,然后再对数据块进行读取查找的两次读入块的时间。

[0089] 本发明NVLSM是完全将NVM当作SSD或HDD等块设备的替代设备的LSM树设计,不同于以往的研究,大部分都是将NVM作为DRAM的缓冲,将NVM用于缓冲本来应该位于DRAM的MemTable数据结构,然后利用NVM的大容量特性使得后台线程可以不被阻塞的完成SSTable合并。大多数对NVM的利用都是将DRAM,NVM和SSD三种存储介质结合起来,形成从上到下的存储层次,这样的做法增加了不同层次的数据迁移的复杂度,增加了编程的难度。而NVLSM则只有两层结构,减少了存储体系结构的复杂性。

[0090] 在写数据方面,NVLSM充分利用NVM的字节可寻址特性。在将位于内存层的

MemTable的键值对写入NVM时,不同于LevelDB等传统LSM设计实现需要先将Immutable MemTable的所有数据添加的缓冲中,然后再写入SSD等块存储设备,NVLSM会利用NVM的字节可寻址特性将Immutable MemTable中的数据直接拷贝到NVM上,这类似于memcpy操作,这样的做法极大地提高了写入速度,减少了Compaction线程被阻塞的时间。

[0091] 同时由于NVM与DRAM都具有字节寻址特性,NVLSM不再需要对索引块的读取和数据块的读取,在NVM的SSTable上对待索引键的查找只需要在NVM上进行,消除了以往的基于块设备的需要在磁盘和DRAM间的I/O操作时间,极大的提高了读性能。

[0092] 以上是对本发明的较佳实施进行了具体说明,但本发明创造并不限于所述实施例,熟悉本领域的技术人员在不违背本发明精神的前提下还可做作出种种的等同变形或替换,这些等同的变形或替换均包含在本申请权利要求所限定的范围内。

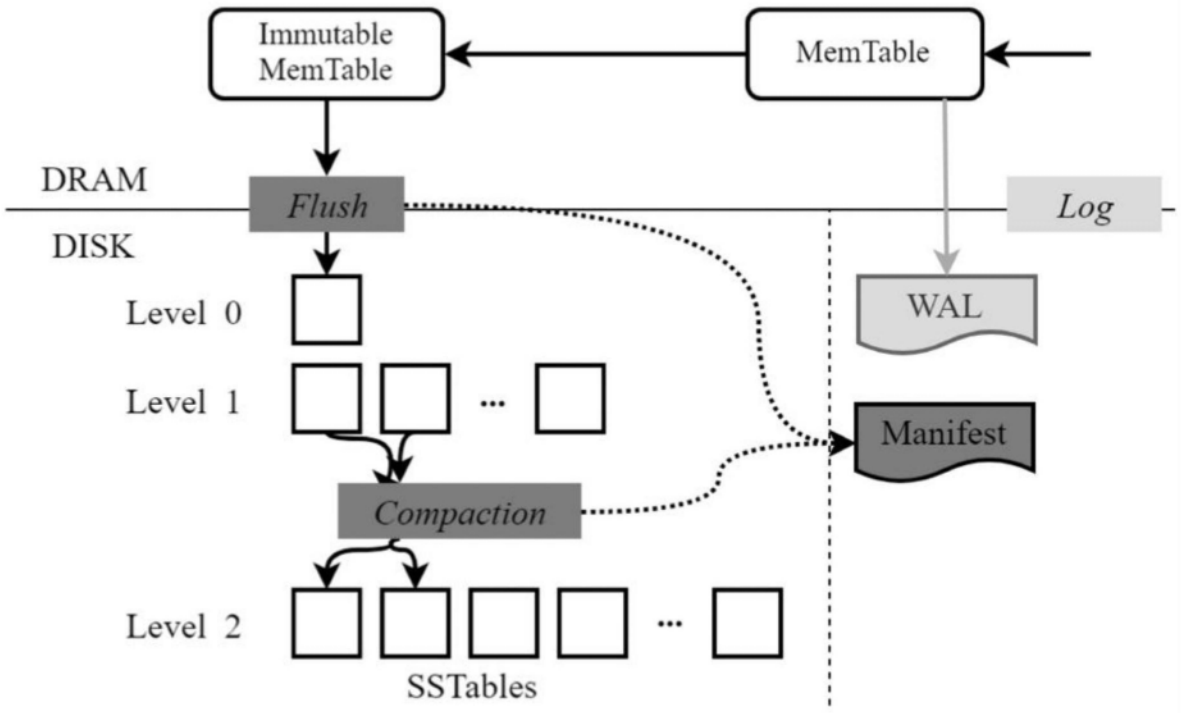

Immutable
MemTable
MemTable
DRAM
Flush
Log
DISK
Level 0
WAL
Level 1
Manifest
Compaction
Level 2
SSTables

图1

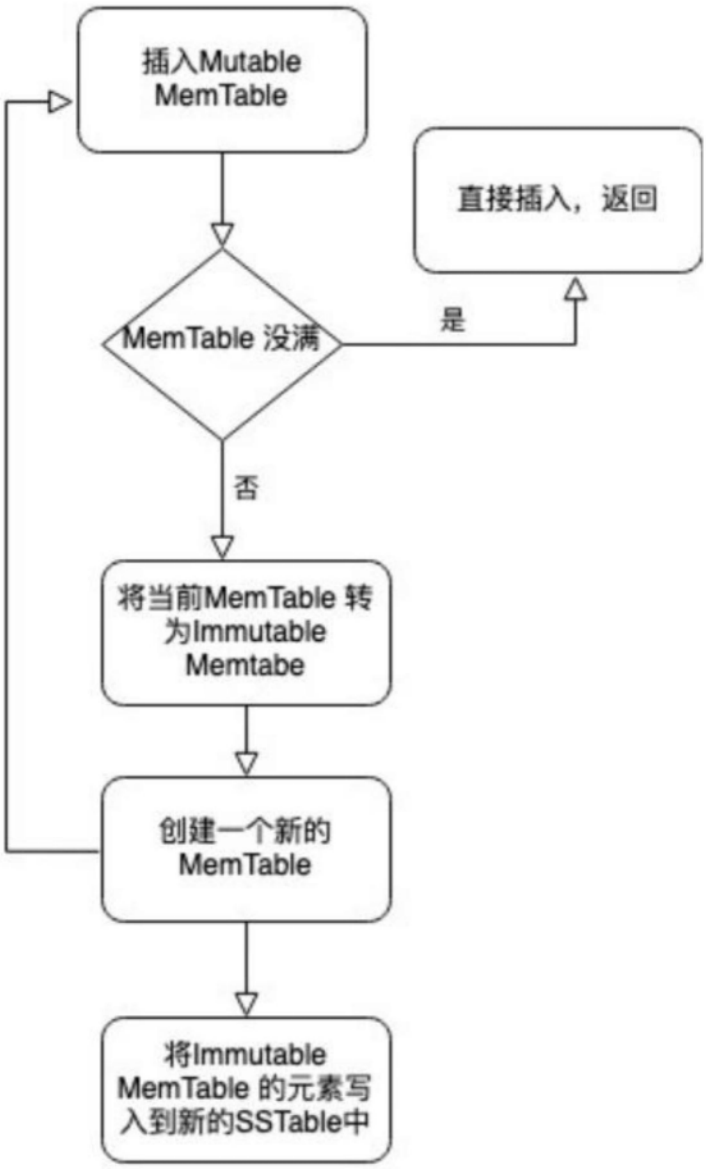

插入Mutable MemTable
直接插入，返回
MemTable 没满
是
否
将当前MemTable 转为Immutable Memtabe
创建一个新的MemTable
将Immutable MemTable 的元素写入到新的SSTable中

图2

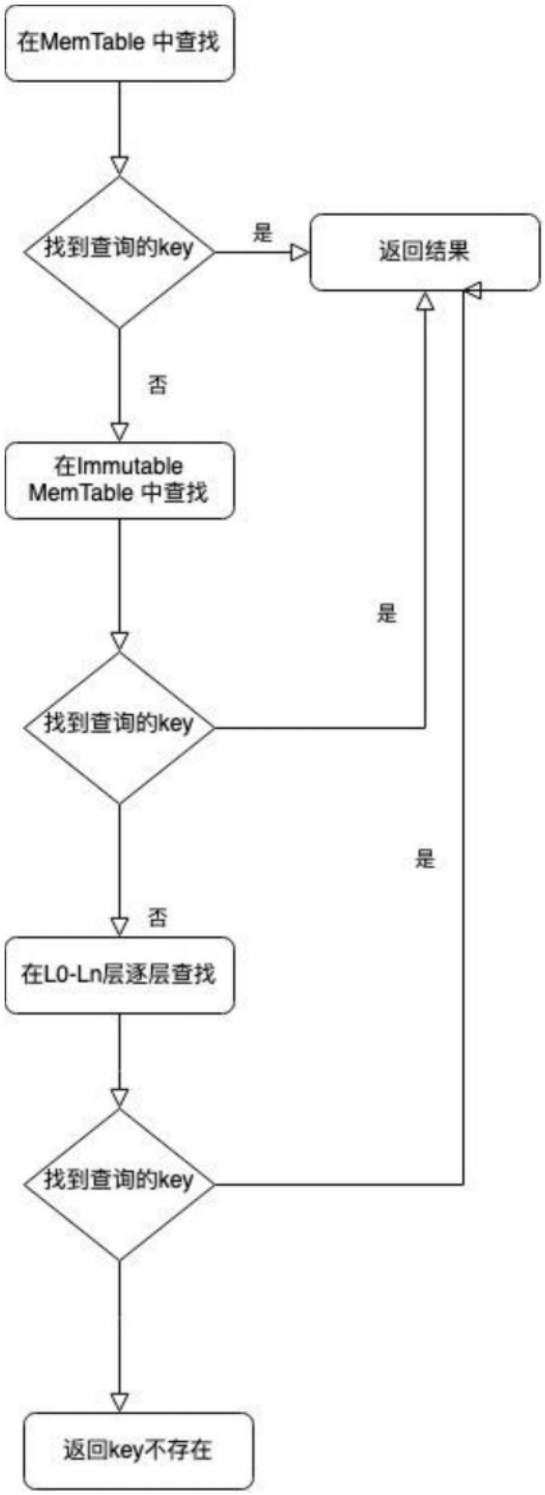

在MemTable 中查找
找到查询的key
是
返回结果
否
在Immutable MemTable 中查找
是
找到查询的key
是
否
在L0-Ln层逐层查找
找到查询的key
返回key不存在

图3