(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 110347336 B
(45)授权公告日 2020.07.10

(21)申请号 201910497294.6
(22)申请日 2019.06.10
(65)同一申请的已公布的文献号
申请公布号 CN 110347336 A
(43)申请公布日 2019.10.18
(73)专利权人 华中科技大学
地址 430074 湖北省武汉市洪山区珞喻路1037号
(72)发明人 万继光 谢长生 胡皓胜 程志龙 王中华
(74)专利代理机构 华中科技大学专利中心 42201
代理人 李智 曹葆青
(51)Int.Cl.
G06F 3/06(2006.01)
G06F 16/22(2019.01)
审查员 曹野

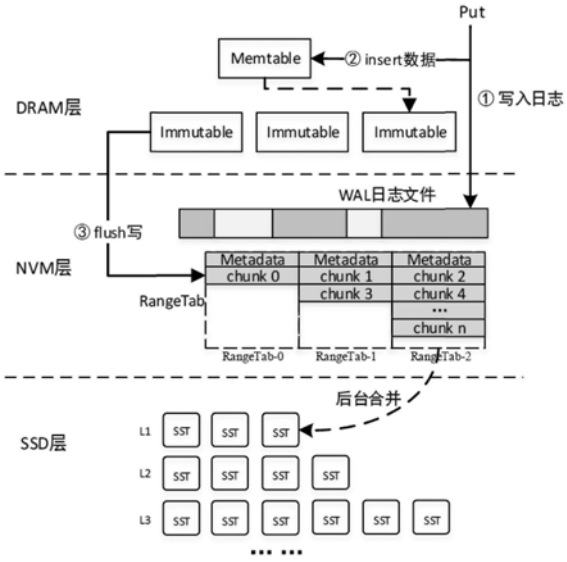
权利要求书2页 说明书9页 附图3页

(54)发明名称

一种基于NVM与SSD混合存储结构的键值存储系统

(57)摘要

本发明公开了一种基于NVM与SSD混合存储结构的键值存储系统，属于数据存储技术领域。本发明提出NVM与SSD混合存储的键值存储系统，在NVM上使用多个RangeTab结构来组织LSM-Tree的第0层键值数据，传统LSM-Tree结构的0层中各SSTable键范围是无序的，且键范围允许重叠，合并时因为较大的键范围容易读写更多的数据量，增加合并操作的时延，而本发明在第0层使用RangeTab结构重新组织键值数据，将多个RangeTab映射到不同且互不重叠的键范围中，Memtable键值数据由键范围将键值数据写入相应的RangeTab结构中，以缩短该层结构数据的合并时延；通过适当增加RangeTab结构的数量来增大所有RangeTab的数据容量。在相邻层容量之比保持不变的条件下，本发明每层能够容纳更多的数据量，LSM-Tree结构层数减少，系统合并的次数也会降低。



CN 110347336 B

1.一种基于NVM与SSD混合存储结构的键值存储系统，其特征在于，所述键值存储系统采用LSM-Tree结构作为存储引擎，由上到下分为：

内存，负责存储上层应用插入或更新到键值存储系统的最新键值数据于表Memtable中，若Memtable大小达到第一阈值，将其转变成只读状态的表Immutable后，将其键值数据写入到NVM；

NVM，负责使用RangeTab结构来组织LSM-Tree结构L_0层的键值数据，每个RangeTab结构容量大小相等，将各个RangeTab映射到不同且互不重叠的键范围，Immutable键值数据由键范围将键值数据写入相应的RangeTab结构中；若NVM中所有RangeTab结构的键值数据达到第二阈值，选取部分RangeTab结构与SSD中L_1层中键范围有重叠的所有SSTable文件合并到L_1层；

SSD，负责存储LSM-Tree结构L_1层以及所有下层SSTable文件数据，若L_i层SSTable文件数量达到第三阈值，将L_i层的部分SSTable文件合并到L_{i+1}层，$i \geqslant 1$。

2.如权利要求1所述的键值存储系统，其特征在于，所述RangeTab结构分成三部分：Metadata区、Index区与Data区；

Metadata区用于记录RangeTab的元数据；

Index区包含cur和seq，cur用于记录定位到Data区的索引信息，seq初始值为0，在RangeTab每次写入数据时其值自动累加1，而当键值数据全部合并到下层后其值又重新设置为0；

Data区只负责存储键值数据。

3.如权利要求2所述的键值存储系统，其特征在于，所述Data区由多个物理地址相连的Chunk组成，每个Chunk记录着键值数据内容Chunkdata、数据的大小Chunksize、数量、相对偏移地址这些元数据信息，每个Chunk的Chunkdata内存储的所有键值对有序排列，然后依次记录每条键值对的相对偏移地址，最后记录该Chunk中键值对的数量。

4.如权利要求1至3任一项所述的键值存储系统，其特征在于，使用哈希地址空间结构记录NVM上RangeTab结构存储的每个key的偏移地址，将含有足够多键值数据的每相邻若干个Chunk分成一组，以组为单位分批次构造哈希映射；查找某个Chunk内的键值对时使用哈希映射，在哈希地址空间中索引到键值数据的偏移地址，由偏移地址在Chunk结构中找到键值数据。

5.如权利要求1至3任一项所述的键值存储系统，其特征在于，在Immutable键值数据写到NVM之前，将key前缀相同的多个键值对以Chunk结构组织起来，每个Chunk内按照key的大小有序排列，然后根据key的前缀映射到有相同前缀的RangeTab，并将数据追加写到该RangeTab的Data区内最后一个Chunk的末尾。

6.如权利要求1至3任一项所述的键值存储系统，其特征在于，NVM上的数据合并到SSD上L_1层时，选取部分RangeTab结构，具体如下：

(1)若存在某个RangeTab存储的数据量与该RangeTab容量的比值超过阈值β，则对这些RangeTab按照数据量与总容量的比值从高到低排序，依次从比值最高的RangeTab开始将数据合并到下层；

(2)对于那些数据量与容量的比值达到阈值α的RangeTab，计算各个RangeTab的数据量与相邻下层存在键范围重叠的SSTable数据量的比值，并选取其中比值最大的RangeTab参

与合并；

(3)当所有RangeTab数据与容量比值的平均值大于μ，只考虑数据量与容量的比值小于α但均大于μ的RangeTab，随机选取键范围相邻且数据量尽可能多的多个RangeTab一起参与合并，且这些RangeTab总数据量与单个RangeTab容量比值不超过阈值γ；

(4)如果不满足前三个条件，则RangeTab不会触发合并；

其中，$0<\mu<\alpha<\beta<1$。

7.如权利要求2所述的键值存储系统，其特征在于，采用双缓存区结构，每个RangeTab在初始化分配Data区空间时，先分配一块物理空间，当一个缓冲区因为数据写满而进行合并流程时，动态分配一块同样大小的新物理空间，Chunk数据在新缓冲区完成写入过程，当合并流程结束后前一个缓冲区释放，只留下新缓冲区。

一种基于NVM与SSD混合存储结构的键值存储系统

技术领域

[0001] 本发明属于数据存储技术领域，更具体地，涉及一种基于NVM与SSD混合存储结构的键值存储系统。

背景技术

[0002] 使用LSM-Tree(日志结构合并树，LogStructuredMerged-Tree)作为存储引擎的有LevelDB、HBase与RocksDB等，该类数据存储引擎将写入的数据有序插入到内存中已排好序的跳表结构中，同时在内存中数据量达到足够多时就会将其批量写入持久化的存储介质，能为系统带来足够优秀的写性能，另外在向内存写入数据之前在持久化介质中追加写日志的操作可以保证数据的可靠性，防止系统断电时内存数据丢失等，但是LSM-Tree在将相邻层数据合并写到下层时存在较为明显的数据写放大问题。

[0003] 针对LSM-Tree的键值系统存在数据写放大的问题，现有技术中，第一种是直接降低LSM-Tree的层次来降低系统的写放大，该方法虽然能够减少数据写入系统的整体写放大次数，但是也会使得相邻层的写放大系数逐渐增大，具体地，相邻层数据之间的合并时参与的数据量会变得异常庞大而无法控制，在写入键值数据库的数据量足够多时，系统的写入操作由于合并耗时过长而长时间阻塞。第二种是在不修改LSM-Tree结构的基础上对数据读写做些优化，比如使用多线程并行读取不同层的数据加快读操作以及使用新型的非易失存储介质(Non-VolatileMemory，NVM)与DRAM交替存储Memtable中的键值数据等，这些方法能够提升部分读写性能，而最终写放大系数仍然没有显著降低。第三种是使用NVM存储Memtable数据，例如NoveLSM方案中，使用NVM和DRAM两类存储介质轮流作为Memtable数据存储的介质，并且设置NVM中Memtable数据大小为4GB(默认DRAM的Memtable为64MB)，这样可以降低系统因Memtable数据尚未写入固态硬盘(SolidStateDisk，SSD)而导致的阻塞时间。同时该方案对NVM介质中Memtable跳表结构进行了字节寻址访问的优化，提升数据写入的速度；为了提高读流程效率，使用双线程分别在Memtable和SSTable((SortedStringSSTable，有序字符串表))文件中并行查找数据。该方案通过增大NVM的容量来存储更多的Memtable数据，合并过程中L_0层数据量明显高于下层SSTable文件的数据量，能够减少L_0层合并的写放大比例，但是写入NVM的4GB键值数据会使得后续其它层合并的数据量骤增，系统写性能的曲线波动幅度更大，可能会导致系统更长的阻塞时间而无法对外提供正常服务。

发明内容

[0004] 针对现有技术的缺陷，本发明的目的在于解决现有技术无法兼顾读写性能和降低系统写放大的技术问题。

[0005] 为实现上述目的，本发明实施例提供了一种基于NVM与SSD混合存储结构的键值存储系统，所述键值存储系统采用LSM-Tree结构作为存储引擎，由上到下分为：

[0006] 内存，负责存储上层应用插入或更新到键值存储系统的最新键值数据于表

Memtable中，若Memtable大小达到第一阈值，将其转变成只读状态的表Immutable后，将其键值数据写入到NVM；

[0007] NVM，负责使用RangeTab结构来组织LSM-Tree结构L_0层的键值数据，每个RangeTab结构容量大小相等，将各个RangeTab映射到不同且互不重叠的键范围，Immutable键值数据由键范围将键值数据写入相应的RangeTab结构中；若NVM中所有RangeTab结构的键值数据达到第二阈值，选取部分RangeTab结构与SSD中L_1层中键范围有重叠的所有SSTable文件合并到L_1层；

[0008] SSD，负责存储LSM-Tree结构L_1层以及所有下层SSTable文件数据，若L_i层SSTable文件数量达到第三阈值，将L_i层的部分SSTable文件合并到L_{i+1}层，$i \geq 1$。

[0009] 具体地，所述RangeTab结构分成三部分：Metadata区、Index区与Data区；

[0010] Metadata区用于记录RangeTab的元数据；

[0011] Index区包含cur和seq，cur用于记录定位到Data区的索引信息，seq初始值为0，在RangeTab每次写入数据时其值自动累加1，而当键值数据全部合并到下层后其值又重新设置为0；

[0012] Data区只负责存储键值数据。

[0013] 具体地，所述RangeTab结构分成三部分：Metadata区、Index区与Data区；

[0014] Metadata区用于记录RangeTab的元数据；

[0015] Index区包含cur和seq，cur用于记录定位到Data区的索引信息，seq初始值为0，在RangeTab每次写入数据时其值自动累加1，而当键值数据全部合并到下层后其值又重新设置为0；

[0016] Data区只负责存储键值数据。

[0017] 具体地，所述Data区由多个物理地址相连的Chunk组成，每个Chunk记录着键值数据内容Chunkdata、数据的大小Chunksize、数量、相对偏移地址这些元数据信息，每个Chunk的Chunkdata内存储的所有键值对有序排列，然后依次记录每条键值对的相对偏移地址，最后记录该Chunk中键值对的数量。

[0018] 具体地，使用哈希地址空间结构记录NVM上RangeTab结构存储的每个key的偏移地址，将含有足够多键值数据的每相邻若干个Chunk分成一组，以组为单位分批次构造哈希映射；查找某个Chunk内的键值对时使用哈希映射，在哈希地址空间中索引到键值数据的偏移地址，由偏移地址在Chunk结构中找到键值数据。

[0019] 具体地，在Immutable键值数据写到NVM之前，将key前缀相同的多个键值对以Chunk结构组织起来，每个Chunk内按照key的大小有序排列，然后根据key的前缀映射到有相同前缀的RangeTab，并将数据追加写到该RangeTab的Data区内最后一个Chunk的末尾。

[0020] 具体地，NVM上的数据合并到SSD上L_1层时，选取部分RangeTab结构，具体如下：

[0021] (1)若存在某个RangeTab存储的数据量与该RangeTab容量的比值超过阈值β，则对这些RangeTab按照数据量与总容量的比值从高到低排序，依次从比值最高的RangeTab开始将数据合并到下层；

[0022] (2)对于那些数据量与容量的比值达到阈值α的RangeTab，计算各个RangeTab的数据量与相邻下层存在键范围重叠的SSTable数据量的比值，并选取其中比值最大的RangeTab参与合并；

[0023] (3)当所有RangeTab数据与容量比值的平均值大于μ,只考虑数据量与容量的比值小于α但均大于μ的RangeTab,随机选取键范围相邻且数据量尽可能多的多个RangeTab一起参与合并,且这些RangeTab总数据量与单个RangeTab容量比值不超过阈值γ;

[0024] (4)如果不满足前三个条件,则RangeTab不会触发合并,0<μ<α<β<1。

[0025] 具体地,采用双缓存区结构,每个RangeTab在初始化分配Data区空间时,先分配一块物理空间,当一个缓冲区因为数据写满而进行合并流程时,动态分配一块同样大小的新物理空间,Chunk数据在新缓冲区完成写入过程,当合并流程结束后前一个缓冲区释放,只留下新缓冲区。

[0026] 总体而言,通过本发明所构思的以上技术方案与现有技术相比,具有以下有益效果:

[0027] 1.本发明提出NVM与SSD混合存储的键值存储系统,在NVM上使用多个RangeTab结构来组织LSM-Tree的第0层键值数据,传统LSM-Tree结构的0层中各SSTable键范围是无序的,且键范围允许重叠,合并时因为较大的键范围容易读写更多的数据量,增加合并操作的时延,而本发明在第0层使用RangeTab结构重新组织键值数据,将多个RangeTab映射到不同且互不重叠的键范围中,Memtable键值数据由键范围将键值数据写入相应的RangeTab结构中,以缩短该层结构数据的合并时延;通过适当增加RangeTab结构的数量来增大所有RangeTab的数据容量。在相邻层容量之比保持不变的条件下,本发明每层能够容纳更多的数据量,LSM-Tree结构层数减少,系统合并的次数也会降低。

[0028] 2..本发明将存储在NVM上RangeTab待查找的键值数据预先构造哈希索引以减少访问存储介质的次数,而存储在SSD上的键值数据仍以SSTable文件组织,其查找方式保持不变。哈希索引只需记录每个key以及该key在RangeTab结构的偏移地址,在查找时将哈希函数计算key后得出的哈希值作为在哈希地址空间的地址来读取key及偏移地址,就可得到key在NVM上的实际物理地址。RangeTab的哈希索引方式查找数据可以大量减少访问NVM的次数,加快查找效率,间接提升系统的读性能。

[0029] 3.本发明在NVM中RangeTab合并时采用双缓存结构,能够降低系统写入阻塞的时延,为了减少合并写放大,优先选取写放大比例最低的RangeTab数据参与合并,以提升合并效率,进一步提升系统写性能。

附图说明

[0030] 图1为本发明实施例提供的一种基于NVM与SSD混合存储结构的键值存储系统结构示意图;

[0031] 图2为本发明实施例提供的RangeTab结构示意图;

[0032] 图3为本发明实施例提供的Data结构示意图;

[0033] 图4为本发明实施例提供的Chunk分组哈希映射过程示意图;

[0034] 图5为本发明实施例提供的RangeTab的键范围映射过程示意图;

[0035] 图6为本发明实施例提供的RangeTab双缓存结构示意图。

具体实施方式

[0036] 为了使本发明的目的、技术方案及优点更加清楚明白,以下结合附图及实施例,对

本发明进行进一步详细说明。应当理解，此处所描述的具体实施例仅仅用以解释本发明，并不用于限定本发明。

[0037] 如图1所示，本发明提供了一种基于NVM与SSD混合存储结构的键值存储系统，所述键值存储系统采用LSM-Tree结构作为存储引擎，由上到下分为：

[0038] 内存，负责存储上层应用插入或更新到键值存储系统的最新键值数据于表Memtable中，若Memtable大小达到第一阈值，将其转变成只读状态的表Immutable后，将其键值数据写入到NVM；

[0039] NVM，负责使用RangeTab结构来组织LSM-Tree结构L_0层的键值数据，将各个RangeTab映射到不同且互不重叠的键范围中，Memtable键值数据由键范围将键值数据写入相应的RangeTab结构中；若NVM中RangeTab结构的键值数据达到第二阈值，选取部分RangeTab结构与SSD中L_1层中键范围有重叠的所有SSTable文件合并到L_1层；

[0040] SSD，负责存储LSM-Tree结构L_1层以及所有下层SSTable文件数据，若L_i层SSTable文件数量达到第三阈值，将L_i层的部分SSTable文件合并到L_{i+1}层，$i \geq 1$。

[0041] 内存(DRAM)，负责存储上层应用插入或更新到键值存储系统的最新键值数据于表Memtable中，键值数据在DRAM中以跳表结构的形式有序组织。用户在将写入的键值对插入到Memtable之前，先将相应的log信息写入存储介质(NVM)上的日志文件，以便于系统断电后能够恢复内存中的键值数据。若Memtable大小达到第一阈值(RocksDB设置的默认值为64MB)，将其转变成只读状态的表Immutable后，再也无法对其做任何的插入、删除或修改操作。然后将表Immutable的键值数据flush写到NVM。

[0042] 本发明中存储介质采用NVM与SSD混合存储结构。

[0043] NVM，负责使用RangeTab结构来组织LSM-Tree结构L_0层的键值数据，将各个RangeTab映射到不同且互不重叠的键范围中，Memtable键值数据由键范围将键值数据写入相应的RangeTab结构中；若NVM中RangeTab结构的键值数据达到第二阈值，选取部分RangeTab结构与SSD中L_1层中键范围有重叠的所有SSTable文件合并到L_1层。另外，NVM还负责存储预写式日志(WriteAheadLogging，WAL)，键值存储系统在调用Put接口Put(key，value)向Memtable跳表内插入或更新键值数据之前，会先向NVM写入日志，防止机器发生如断电、宕机等故障而导致DRAM中Memtable数据丢失。通过预写式日志可以恢复DRAM跳表结构中的所有键值数据信息，而且使用NVM替代SSD存储WAL日志可以显著加快系统处理写请求的效率，提升系统对写请求的响应速度。

[0044] 由于NVM存储介质具有比NAND寿命长，容量大，非易失存储以及接近于DRAM读写性能的优势，它能够替代DRAM、硬盘等存储设备。本发明选择比SSD的读写性能更好的非易失存储介质NVM，例如，相变存储器PCM，其具有非易失性、可长期保存数据、而且数据不需刷新等优点。

[0045] RangeTab结构

[0046] RangeTab结构负责组织、映射与存储键值数据，其结构如图2所示。

[0047] RangeTab结构分成三部分：Metadata区、Index区与Data区。

[0048] Metadata区用于记录RangeTab的元数据，例如，该RangeTab在NVM上占用的容量大小capacity、已使用物理空间大小usedsize与起始物理偏移地址Chunk-Ikeyrange等信息。

[0049] Index区cur记录着定位到Data区的索引信息，用于辅助获取待查找键值数据的偏

移地址，而seq初始值为0，在RangeTab每次写入数据时其值自动累加1，而当键值数据全部合并到下层后其值又重新设置为0，在键值系统重新启动时可以比较seq值是否等于0来判断系统是否需要重新构建RangeTab结构。

[0050] Data区只负责存储键值数据，由多个物理地址相连的Chunk组成，如3所示，每个Chunk记录着键值数据内容Chunkdata及这些数据的大小Chunksize等元数据信息。Chunkdata内存储的所有键值对(keysize|key|valuesize|value)有序排列，然后依次记录每条键值对的相对偏移地址，最后记录该Chunk中键值对的数量。

[0051] 在读取Chunk键值数据时，先获取该Chunk键值对的数量(KV项数量)，然后依次读取并解析每个键值对的偏移地址(KV偏移地址)，最后从解析后的地址中读取相应的键值数据。在后台线程将Immutable数据flush写到RangeTab之前，会先将待写入的若干个键值对在内存中以Chunk结构形式封装起来，封装完毕后再将整个Chunk的数据一次性追加写到RangeTab的Data区，并在操作结束后修改Metadata区的元数据信息。

[0052] 单个RangeTab结构的数据容量接近SSTable文件的最大容量(64MB)时，性能最佳。第二阈值根据实际需要设置，取值范围为2GB-32GB。

[0053] 哈希索引

[0054] 在RangeTab的Data区中，每个Chunk包含若干条键值对，其内部的键值对有序排列。每个Chunk内部Chunkdata的键值数据部分都会对应着记录着其相对偏移地址的索引结构，每一条记录键值对相对偏移地址的索引结构大小固定，占8个字节。Metadata区记录着每次新写入Chunk的键范围，在RangeTab中根据key查找键值对时，需要从Metadata区获取该RangeTab中所有Chunk的键范围信息，通过比较待查找的key是否在相应的键范围内，得到可能存有key的所有Chunk集合。同时，为了保证最终查找到键值数据的正确性，需要从最新写入RangeTab的Chunk数据开始往前查找，避免出现查找到的key对应的value值为旧值或无效值的情况。

[0055] 为了提升查找Chunk内键值对的效率，本发明设计哈希地址空间结构，记录每个key的偏移地址。在查找某个Chunk内的键值对时，只需要在哈希地址空间由哈希函数计算后得到的地址处获取到键值对所处的偏移地址值，然后根据该偏移地址在Chunk中读取对应待查找的键值数据。

[0056] 读取键值对整个过程的时间复杂度为O(1)，而原先在Chunk中需要经过二分查找法多次访问NVM并对比数据，以确定Chunk内待查找的键值对，该过程的时间复杂度为O(logN)。与二分查找法对比，使用哈希索引方式查找键值数据的偏移地址，能显著减少查找过程中访问NVM的次数，提升RangeTab键值数据的查找效率。

[0057] 为了避免最糟糕情况导致RangeTab查找性能下降，本发明设计将含有足够多键值数据的每相邻若干个Chunk分成一组，以组而不是单个Chunk为单位来构造哈希索引，其哈希索引的构造方式如图4所示。每组Chunk内的每条键值对使用哈希函数计算key在哈希空间的位置L，该位置L所在的数据块内记录着键值对在Chunk内的偏移地址Addr，哈希函数计算公式可以用下式表示：

[0058] L＝hash(key)％N

[0059] N代表键值对的个数，哈希地址空间大小等于该组若干个Chunk中所有键值对的数量，哈希函数映射产生地址冲突的key会暂存在冲突标记空间。一组Chunk中不同Chunk之间

可能存在key相同的键值对，后写入RangeTab的Chunk键值数据相对来说较新些。因此在构造哈希地址空间时，计算key的哈希值从最先写入RangeTab的Chunk开始往后遍历键值对，哈希索引过程中在遇到相同key的情况下较新value的地址与较旧的地址出现冲突时，在哈希地址空间中用新键值对替换已存在的旧键值对；或者从最后写入RangeTab的Chunk开始遍历键值对，若当前正处理的key已经在哈希地址空间或冲突标记空间中存在，则直接跳过该键值对不作处理。

[0060] 等一组Chunk的全部键值对哈希映射完成后，再处理冲突标记空间的数据。使用位图标识哈希地址空间每个数据块是否空闲或已被占用，将冲突标记空间中的第1～X个键值对依次存储在哈希地址空间的第1～X个空闲数据块中。在查找时，若key可能哈希映射到冲突标记空间，则在该冲突标记空间中使用二分查找法快速定位冲突标记空间中的key，另外依据键值对数量的不同选取合适的哈希函数也可以减少发生哈希冲突的次数。

[0061] 键范围映射

[0062] L_0层RangeTab准备合并时，会选取下层与该RangeTab的键范围存在重叠的所有SSTable文件参与合并。若L_0层中实际参与合并的只有一个RangeTab，则合并的实际键范围会相对更宽些，下层会有更多的SSTable文件参与合并，在极端情况下甚至会选取下层所有SSTable文件。这样在将RangeTab相同数据量写到下层时，额外写入下层的数据量更多，系统前台线程处理请求的性能会进一步降低。本发明设计L_0层使用多个RangeTab存储数据，每个RangeTab结构容量大小相同且不同RangeTab的键范围互不重叠，并分析多个RangeTab的设计相比单个RangeTab的优势以及如何影响系统写性能。

[0063] (1)单个RangeTab的设计在每次合并时键值数据的键范围可能变得很大，大多数情况下几乎能与下层所有SSTable文件的键范围发生重叠，也就更容易让下层所有SSTable文件都参与合并，进而使得合并时写放大对应的数据量骤增，系统在更长时间内发生阻塞而无法对外提供正常的读写服务。

[0064] (2)而多个RangeTab的设计，能够缩小每次参与合并RangeTab的键范围以减少下层存在键范围重叠的SSTable文件数量，继而减少合并过程涉及的数据量，降低系统前台线程处理请求的阻塞时延，使得系统写性能曲线表现得更加平稳。

[0065] (3)多个RangeTab的设计使得每次参与合并RangeTab的键范围相对较小，在每次合并到下层的数据量基本不变的情况下，增加了L_0层容纳键值数据的容量。在相邻层写放大系数AF保持不变条件下，有效减少LSM-Tree结构的层数，降低系统数据的写放大比例与合并次数，提升系统的随机写性能。

[0066] 内存Immutable的键值对写到NVM各个RangeTab时，为了尽可能让相邻的键值对分布在物理地址相邻的位置，本发明设计按照key前缀为RangeTab划分不同的键范围，保证写到相同RangeTab的每个Chunk内所有键值对均有相同的前缀。如图5所示，假设key的大小为16个字节，前缀占用前4个字节，由这前缀对应4个字节表示的区间范围确定了RangeTab的键范围。

[0067] 在Immutable数据写到NVM之前，将key前缀相同的多个键值对以Chunk结构组织起来，然后根据key的前缀映射到有相同前缀的RangeTab，并将数据追加写到该RangeTab的Data区。

[0068] 第一合并策略

[0069] RangeTab处于LSM-Tree结构的L_0层，其数据合并到下层的次数占系统总合并次数的大部分。因此RangeTab的合并效率成为键值系统写性能的主要瓶颈之一。

[0070] 当Immutable数据正准备向某个正在合并的RangeTab中flush写数据时，为了尽可能地减少RangeTab数据合并导致系统flush写入的阻塞时延，本发明方案针对合并策略提出了双缓存结构设计。预先为RangeTab分配两块大小相同的物理空间，在将该RangeTab某块物理空间存储的数据合并到下层过程中，flush写的数据则存储到另一块物理空间，能够避免flush写数据到RangeTab时发生阻塞的情况，同时降低了flush线程引起系统写请求阻塞的概率。

[0071] 另外，合并策略还提出了RangeTab在哪些情况下可以触发合并以及如何选取RangeTab与下层SSTable数据参与合并的具体方案，以缩短由合并引起的Immutable数据flush写的阻塞时延。同时，当相同数据量合并到下层时，合并策略尽可能多地降低合并的写放大，提升合并效率。

[0072] 双缓存结构

[0073] 在某个RangeTab被选取参与合并流程的过程中，可能存在系统的flush线程准备向该RangeTab写入Chunk数据，一般情况下有两种处理方式:阻塞系统的flush线程直到该RangeTab所有数据的合并流程结束，或者临时在NVM上分配一块额外的物理空间，然后将待flush写的键值数据暂时写到该物理空间，等到RangeTab数据合并结束后，再将该物理空间的键值数据迁回到RangeTab。

[0074] 对于前者，阻塞flush线程写入数据的做法虽然比较容易解决代码实现的问题，但是由于系统flush线程一直阻塞而无法完成数据flush写入流程，严重降低了系统前台线程处理请求的效率。而对于后者，额外分配一块物理空间暂存待写入的键值数据虽然能解决系统写入数据时引起阻塞的问题，但合并结束后再次将该键值数据迁回RangeTab的过程只会徒增不必要的数据迁移开销，降低flush线程flush写数据到RangeTab的效率。

[0075] 为了解决这个问题，本发明采取双缓存结构，其结构设计如图6所示。每个RangeTab在初始化分配Data区空间时，预先分配原先两倍大小的物理空间，分别命名为buffer-1和buffer-2两个缓存区，RangeTab初始化时buffer-1和buffer-2均未存放任何Chunk数据，其状态设置为init_state。

[0076] 当flush线程准备向RangeTab写Chunk数据时，由于buffer-1和buffer-2的状态均是init_state，因此需要先修改buffer-1状态为write_state，表明其当前状态可以存储Chunk数据，并将Chunk数据写入buffer-1;而当buffer-1剩余的物理空间快要耗尽时，其将由write_state状态转变成compaction_state，表明该块物理空间的数据将要或者正在合并，若此时有新的Chunk数据要flush写入该RangeTab，则不能继续写入buffer-1。此时若buffer-2状态为init_state则将其转换成write_state，只有当缓存区的状态为write_state才能将Chunk数据写入其对应的物理空间中，否则只能阻塞flush线程的写流程，直到存在某个缓存区的状态重新变为write_state。

[0077] 参与合并的buffer-1或buffer-2在合并结束后，其状态由compaction_state变为init_state，只有当没有缓存区的状态为write_state，才需要将其中某一个缓存区的状态由init_state变成write_state。

[0078] 本发明采用预先分配两块同等大小物理空间的缓存区结构，解决RangeTab数据合

并过程中可能存在flush线程flush写数据到该RangeTab的系统阻塞问题。也可使用按需分配的方式，即只有在RangeTab需要额外缓存区空间时才会为其分配，而当其键值数据合并结束后就立刻回收对应的物理空间，该方法能有效节省NVM上RangeTab占用的物理空间。但是RangeTab处于LSM-Tree结构的最上层，在负载不断地向系统写入键值数据的过程中系统会频繁地触发合并，按需分配在实际需要该空间时才为其分配虽然在一定程度上能够节省NVM物理存储空间，但是在每次遇到需要分配大量物理空间的情况下可能会增加额外的时间开销，间接降低flush线程flush写数据的效率。考虑到分配大量物理空间带来的时间开销，本发明采取预先分配的方式为RangeTab结构分配相应的物理空间。

[0079] 合并触发策略

[0080] 合并选取的数据量过多时，可能会导致flush线程flush写数据与系统写数据的阻塞时延过长，系统写性能波动起伏过大。而合并数据量过少会导致合并次数增多，其存在两个劣势：一是每次针对少量数据的读写操作无法充分利用NVM的高I/O带宽；二是下层以SSTable文件为单位选取参与合并的键值数据，RangeTab合并的数据量较少并不会缩短实际合并时数据的键范围，反而可能会提高下层参与合并的SSTable文件数据的比例，进而提高该次合并操作的写放大比例。因此如何选择参与合并的数据，决定了本次合并的效率，间接影响系统的写性能。

[0081] 本发明为每个RangeTab存储数据量占用物理空间的比例大小设有3个阈值：μ，α与β，且β>α>μ。

[0082] 由上所述，本发明设计满足以下三个条件中的任意一个就可以触发RangeTab数据合并：

[0083] (1)存在某个RangeTab可用的物理空间快要耗尽，例如数据量比例超过β；

[0084] (2)所有RangeTab数据量占容量比例的平均值(或数据总量占总容量的比值)达到某个较高值α(α<β)；

[0085] (3)当所有RangeTab数据量与容量比值的平均值(或数据总量与总容量的比值)达到某个较高值μ(μ<α)。

[0086] 因此，RangeTab选取算法步骤如下：

[0087] (1)若存在某个RangeTab存储的数据量与该RangeTab容量的比值超过某个阈值β(β>α)例如90％时，则对这些RangeTab按照数据量与总容量的比值从高到低排序，依次从比值最高的RangeTab开始将数据合并到下层。其原因是这些RangeTab更容易阻塞flush线程写数据的流程，需要最优先处理；

[0088] (2)其次对于那些数据量与容量的比值达到阈值α的RangeTab，计算各个RangeTab的数据量与相邻下层存在键范围重叠的SSTable数据量的比值，并选取其中比值最大的RangeTab参与合并，以尽可能地减少该次合并实际的写放大；

[0089] (3)最后在所有RangeTab数据与容量比值的平均值大于μ时，这时只考虑数据量与容量的比值小于α但均大于μ(例如μ≥60％)的RangeTab(此时不存在比值大于α的RangeTab)。随机选取键范围相邻且数据量尽可能多的多个RangeTab一起参与合并，实际参与合并的RangeTab数量不能过多，以便合并的键范围始终保持在较小范围内。而且这些参与合并的RangeTab总数据量与单个RangeTab的容量上限之比不能超过阈值γ例如2，避免合并的数据量过大；

[0090] (4)如果不满足前三个条件,则RangeTab不会触发合并。

[0091] 由RangeTab选取算法选择合适的RangeTab后,然后从下层选取键范围存在重叠的SSTable文件参与合并流程。该选取算法通过缩短合并数据的键范围以及减少合并实际涉及到的数据量,尽可能提升RangeTab数据量与下层SSTable的比值,降低LSM-Tree合并的写放大比例,提高合并效率,最终提升系统的写性能。

[0092] SSD,负责存储LSM-Tree结构L_1中以及所有下层SSTable文件数据,若L_i层SSTable文件数量达到第三阈值,将L_i层的部分SSTable文件合并到L_{i+1}层,$i \geq 1$。SSD层还负责存储其它如MANIFEST、CURRENT等记录键值系统配置信息相关的文件,而且所有合并过程中新创建的SSTable文件均存储在SSD。

[0093] 将L_i层的部分SSTable文件合并到L_{i+1}层,具体包括:

[0094] 1)读取L_i层SSTable文件和L_{i+1}层中与该SSTable键范围存在重叠的所有SSTable文件;

[0095] 2)将这些读取的SSTable文件数据在内存中做归并排序;

[0096] 3)将这些有序的数据以SSTable形式重新写入存储介质,而合并之前旧SSTable文件在合并结束后因为失效会被删除。

[0097] 以上,仅为本申请较佳的具体实施方式,但本申请的保护范围并不局限于此,任何熟悉本技术领域的技术人员在本申请揭露的技术范围内,可轻易想到的变化或替换,都应涵盖在本申请的保护范围之内。因此,本申请的保护范围应该以权利要求的保护范围为准。

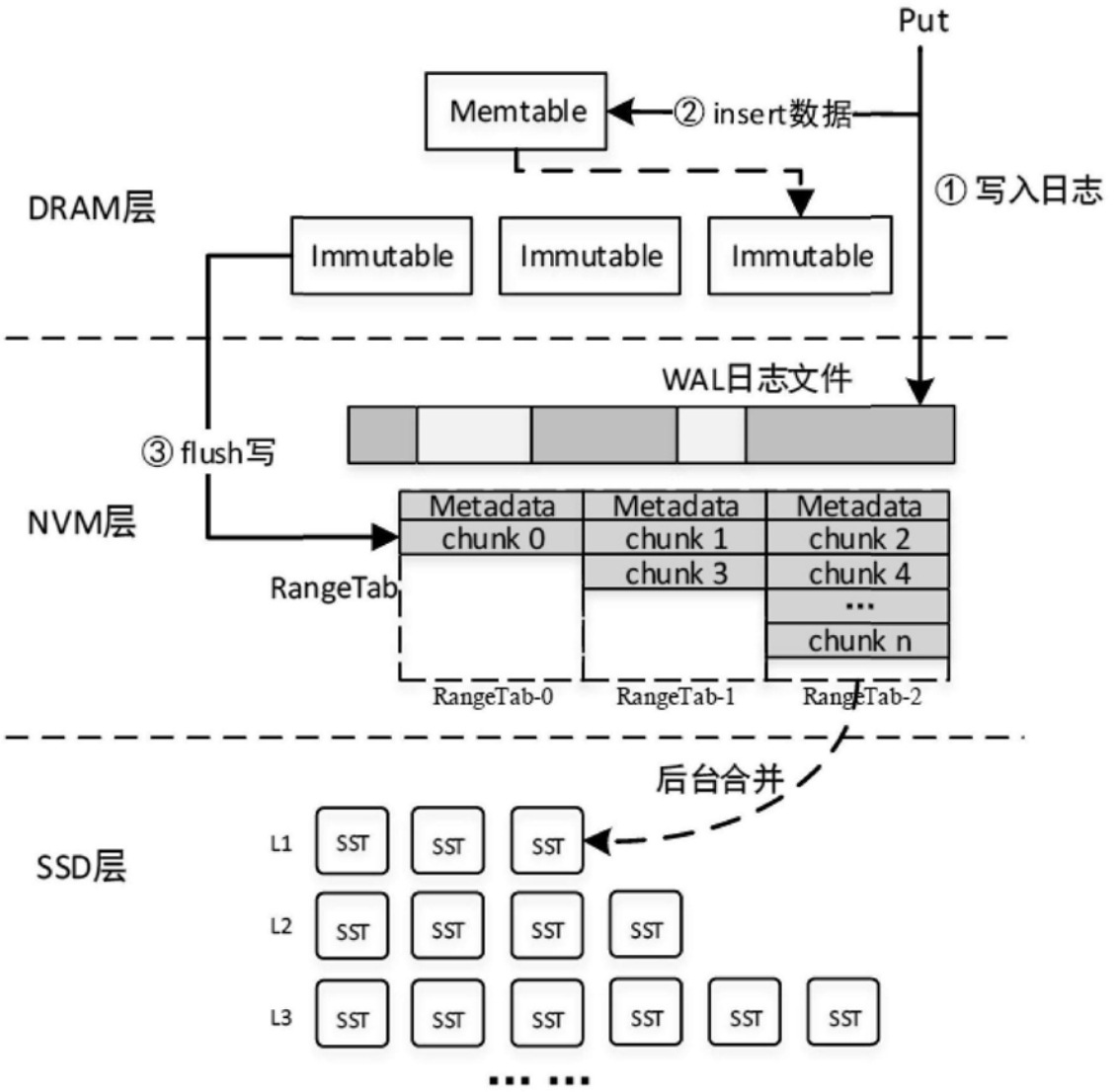

Put
Memtable
② insert数据
① 写入日志
DRAM层
Immutable
Immutable
Immutable
WAL日志文件
③ flush写
NVM层
Metadata
Metadata
Metadata
chunk 0
chunk 1
chunk 2
chunk 3
chunk 4
...
chunk n
RangeTab
RangeTab-0
RangeTab-1
RangeTab-2
后台合并
SSD层
L1
L2
L3
SST

图1

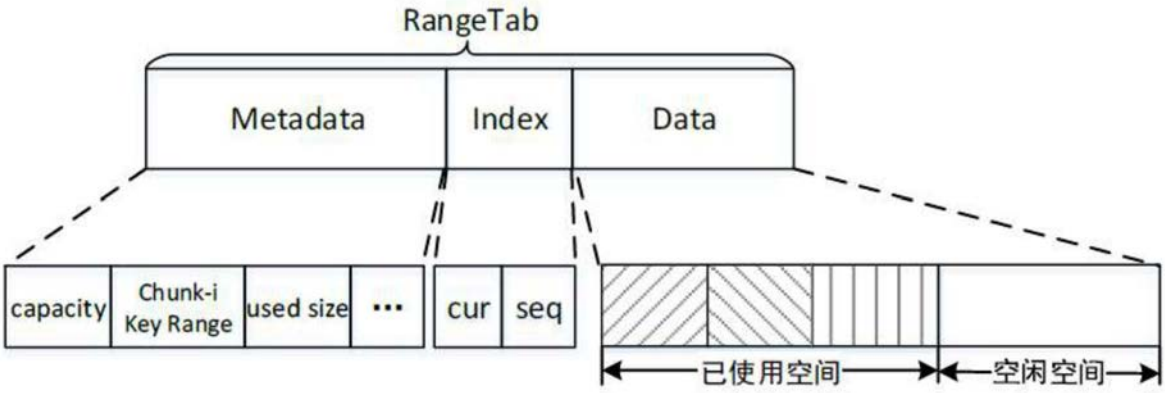

RangeTab
Metadata
Index
Data
capacity
Chunk-i
Key Range
used size
...
cur
seq
已使用空间
空闲空间

图2

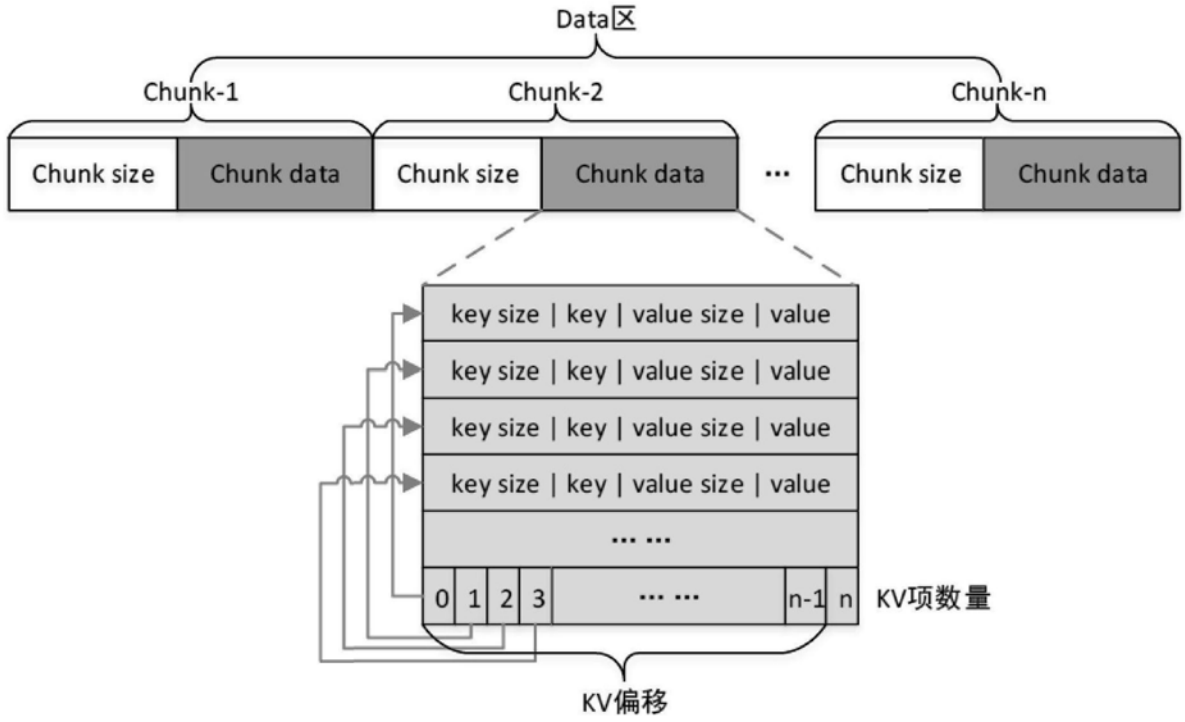

Data区
Chunk-1
Chunk-2
Chunk-n
Chunk size
Chunk data
Chunk size
Chunk data
...
Chunk size
Chunk data
key size | key | value size | value
key size | key | value size | value
key size | key | value size | value
key size | key | value size | value
... ...
0
1
2
3
... ...
n-1
n
KV项数量
KV偏移

图3

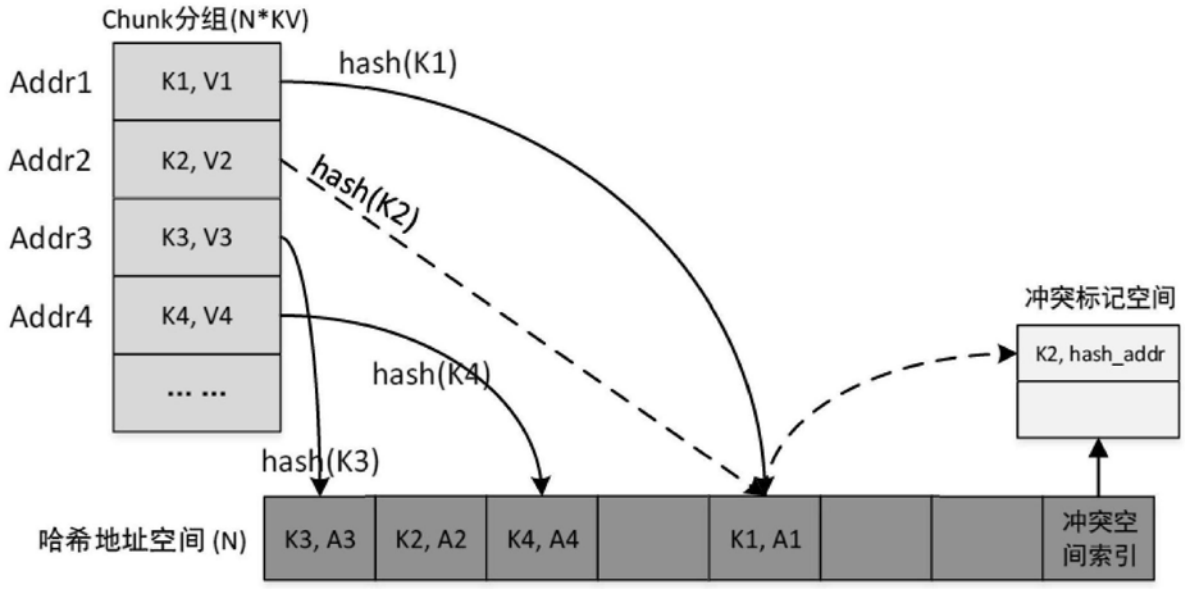

Chunk分组(N*KV)
Addr1
K1, V1
Addr2
K2, V2
Addr3
K3, V3
Addr4
K4, V4
... ...
hash(K1)
hash(K2)
hash(K4)
hash(K3)
冲突标记空间
K2, hash_addr
哈希地址空间 (N)
K3, A3
K2, A2
K4, A4
K1, A1
冲突空间索引

图4

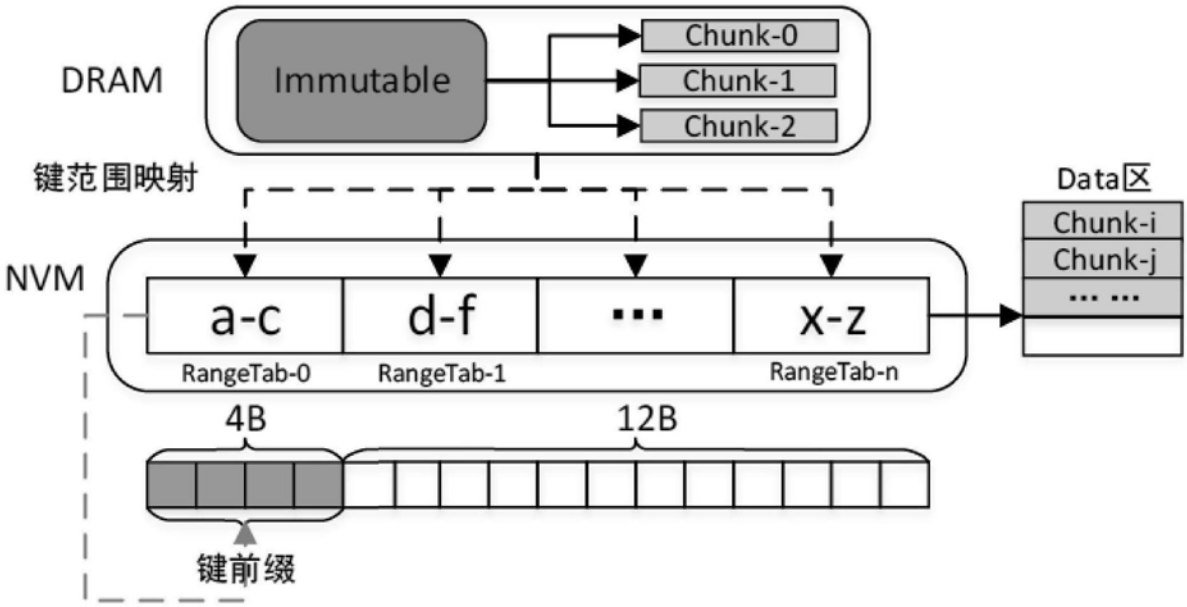

DRAM
Immutable
Chunk-0
Chunk-1
Chunk-2
键范围映射
Data区
Chunk-i
Chunk-j
NVM
a-c
d-f
x-z
RangeTab-0
RangeTab-1
RangeTab-n
4B
12B
键前缀

图5

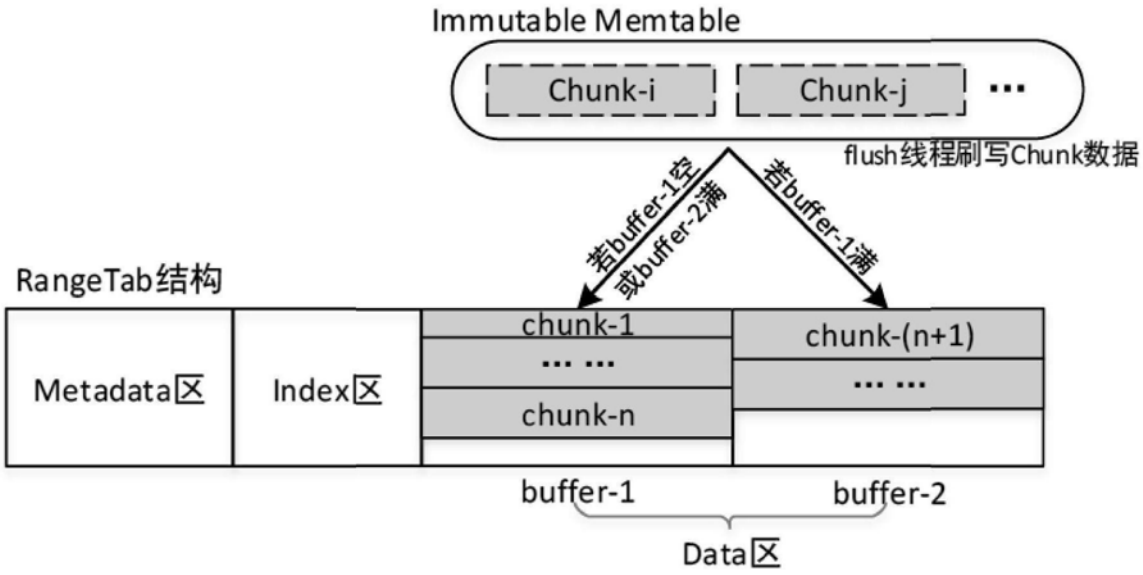

Immutable Memtable
Chunk-i
Chunk-j
flush线程刷写Chunk数据
若buffer-1空
或buffer-2满
若buffer-1满
RangeTab结构
Metadata区
Index区
chunk-1
chunk-n
chunk-(n+1)
buffer-1
buffer-2
Data区

图6