https://doi.org/10.1051/jnwpu/20244220303

基于 LSM-Tree 的分布式数据库异步融合机制研究与实现

杜轶德, 刘文洁

(西北工业大学 计算机学院, 陕西 西安 710072)

摘 要:信息技术的不断发展,使得分布式数据库成为研究热点。由于 NoSQL 架构的分布式数据库对 SQL 支持有限且在事务处理及一致性方面存在缺陷,基于 LSM-Tree 的 NewSQL 数据库逐渐成为应用的主流,例如 TiDB、OceanBase 等。分布式 LSM-Tree 的存储架构将数据分为基线数据与增量数据,通过合并操作将不同分区的增量数据与基线数据不断融合,并存储在磁盘,从而减少内存压力。但合并会占用大量系统资源,严重影响系统可用性。因此提出了一种基于 LSM-Tree 架构的异步融合机制,通过细分合并流程,将数据融合异步化,有效地缩短了单次数据合并的时间。实验表明,提出的异步融合机制可显著缩短数据合并时间,提高系统在高频写入场景下的鲁棒性和可用性。

关 键 词:分布式数据库;LSM-Tree;数据合并;异步融合;数据分区
中图分类号:TP311 **文献标志码**:A **文章编号**:1000-2758(2024)02-0303-07

随着大数据和云计算技术的发展,传统的集中式数据库已经无法满足大数据量的需求,逐渐被分布式数据库所取代。分布式数据库可以将物理上分散的多个数据库单元连接起来组成逻辑上统一的数据库,具有可扩展性、高可用性、数据一致性等特点[1]。2006 年,Google 在分布式领域发表了 3 篇具有代表性的论文,包括分布式文件系统 GFS、分布式 KV 存储数据库 BigTable 以及分布式数据处理模型 MapReduce。这些论文为分布式数据库的发展奠定了基础。

数据库系统的发展与硬件发展息息相关,磁盘仍然是主流的存储设备,要提高数据库性能,需要结合磁盘的结构,优化上层数据访问逻辑。经典的数据库存储引擎如 InnoDB 等,使用 B+树的存储结构,带来了很好的查询性能,但对于修改操作而言,其采用原地更新的方式,直接将新的数据覆盖到之前的位置,这样会导致大量的随机 I/O,极大地降低了写性能,同时,大量的更新和删除操作也会导致磁盘页面碎片化的问题,一定程度上降低了存储设备的空间利用率。

O'Neil 等[2] 于 1996 年提出了 LSM-Tree(log-structured merge-tree)的存储结构,并被广泛应用于现在数据库系统的存储层中,如 BigTable、HBase、LevelDB、RocksDB、TiDB[3]、OceanBase[4] 等。LSM-Tree 是基于读写分离架构,写操作会将数据写入内存,当内存中的数据逐渐增长到内存上限时,会对内存中的数据进行持久化到磁盘,形成 SSTable 文件。读操作会同时获取磁盘和内存数据,从而得到最新数据。并且为了最小化 I/O 代价,磁盘上的 SSTable 文件通常采用分层存储的方式,但随着时间的积累,相同键的无用数据不断积累,数据范围互相交叠的 SSTable 文件越来越多,会引起严重的空间膨胀以及用户读取性能下降。为了解决上述问题,LSM-Tree 引入了合并的解决方法[5],合并操作会将磁盘中的 SSTable 文件通过多路归并的方式进行数据融合,清除无用的历史数据,减少数据文件的个数,同时也优化了用户的读取性能。

基于 LSM-Tree 存储的数据库系统,一次合并操作会带来大量的磁盘 I/O 操作[6],在写极度密集的

收稿日期:2023-04-06
基金项目:国家自然科学基金重点项目(61732014)与华为合作研究项目(D5204220342)资助
作者简介:杜轶德(1999—),硕士研究生
通信作者:刘文洁(1976—),副教授 e-mail:liuwenjie@nwpu.edu.cn

业务场景中,系统的合并速度往往跟不上用户的写入速度,会造成“写阻塞”的问题,严重影响系统的可用性。而对于分布式数据库而言,一次合并操作还会带来短时间大量的网络数据流量传输,占用了大量处理用户请求的带宽。

本文在深入研究 LSM-Tree 模型的基础上[7],结合实际的数据库产品和交易场景,分析现有合并算法瓶颈[8-10],提出了异步融合的数据合并机制,从 CPU 计算、磁盘 I/O 等多方面对现有合并流程进行优化。本文提出的融合机制在面向金融应用的分布式数据库系统 CBase 上进行了验证,实验表明,异步融合机制可大幅减少系统合并时间,降低合并对系统可用性的影响。

1 LSM-Tree 存储结构

LSM-Tree(log-structed-merge tree)是一种基于磁盘设计的有序、分层数据存储结构,采用了读写分离的思想,将随机写转化为批量的顺序写。在一段时间内数据的更新仅会以顺序写入的方式持久化到日志文件以及写在内存空间的数据结构中去,当内存使用超过指定阈值后,将内存中的数据批量顺序写入到磁盘中去,顺序写可以减少磁盘磁头的反复移动,极大地提高了数据写入速度。其架构如图 1 所示。

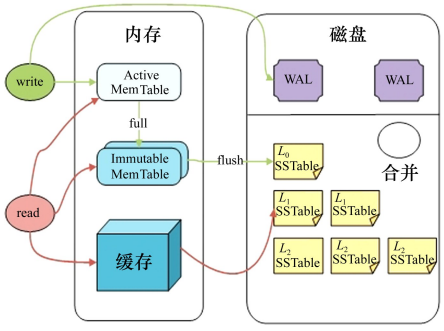

图 1 LSM-Tree 架构

其中,Active MemTable 是数据在内存的存储结构,其具体的数据结构表现方式因系统而异,常用的实现方式如 B+树、跳表、哈希表等。用户最新的写入会按序插入到该数据结构中。由于内存的易失性,在发生断电等系统故障后,Active MemTable 中的数据便会丢失,因此为了保障数据的持久性和可靠性,一次写操作需要将其对应的操作日志先持久化到磁盘上再写入 Active MemTable 中才视为提交成功,这一过程称为预写日志(write ahead log, WAL)。由于日志仅记录重做日志,是以 Append-Only 的顺序写方式写入,因此 LSM-Tree 中,一次写操作仅包括一次内存写入和磁盘顺序写入,其速度是远快于一次随机写入的。由于内存空间的有限性,当 Active MemTable 的大小达到一定的内存限制后,其会转化成 Immutable MemTable,该 MemTable 仅仅提供读取不提供写入,并开辟一个新的 Active MemTable 支持前台继续写入。Immutable MemTable 是一种中间状态,后台线程会将 Immutable MemTable 持久化到磁盘上形成 SSTable 文件,而不阻塞前台业务处理。SSTable(sorted string table)是 LSM 存储引擎在磁盘上的数据存储形式,是有序键值对的集合,作为只读数据,工程实现上通常引入布隆过滤器以及各级缓存以加速 SSTable 文件的查找速度。SSTable 的层数也因系统不同而异,如 OceanBase 采用了 L_0+$L_1$2 层管理的方式,而 RocksDB 则支持更多层的数据组织方式。为了清除历史数据以及加快读操作的速度,磁盘上的 SSTable 文件会通过后台合并操作进行合并,逐级下沉,使最上层的数据处于较热的状态。由于数据可能分布在内存中的各个 MemTable、磁盘上的各层 SSTable 上,一次读操作需要按序对上述空间进行搜索,直到搜到最新的数据,带来了不少的读放大。

2 基于 LSM-Tree 的分布式数据库 CBase

LSM-Tree 在写密集的业务场景下拥有良好的表现,因此工业上许多分布式数据库的底层存储引擎均是基于 LSM-Tree 实现的。CBase 是西北工业大学联合交通银行研发的面向金融应用的分布式数据库[11-13],该数据库融合了 LSM-Tree 增量聚集思想,采用了新型的 NewSQL 数据库架构,在保证了传统数据库事务 ACID 的特性并兼容传统 SQL 语法的基础上,实现了分布式数据库的可扩展和高可用特性,已在交通银行多个核心业务中应用。CBase 的架构如图 2 所示。

在 CBase 分布式数据库中,主要有几种角色:

1) Client 客户端 由于 CBase 很好地兼容了 MySQL 协议,上层应用可以将后台数据库平滑地迁移到 CBase。Client 通过 ODBC/JDBC 的方式接入

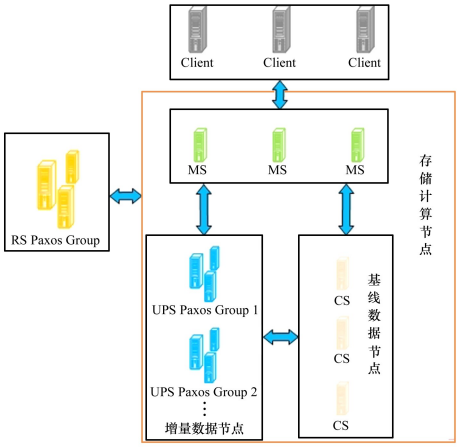

图 2　CBase 系统架构

CBase,负责处理并发送用户的请求以及接收数据库返回的数据或结果。

2) MS(MergeServer)　MS 为 CBase 中的计算节点,负责接收客户端发来的 SQL 请求,对其进行合法性检查、语法树生成、物理算子生成填充等步骤后,生成一棵该 SQL 语句对应的执行计划树,并根据执行计划,与集群中的存储节点或总控节点进行交互,将相关的请求进行转发并对分布式执行的结果进行汇总处理。该节点仅负责计算转发和结果合并,不保存实际数据,因此是无状态的数据节点,可以通过增加服务器而不需额外操作就可提升整个集群的计算能力。

3) CS(ChunkServer)　CS 为 CBase 中的基线数据节点,负责存储 L_1 层的基线数据,并为计算节点 MS 提供相应的读接口实现。L_1 层的数据按范围进行划分,按多副本冗余存储,均匀分布在不同的 CS 服务器上,保证出现单点故障时数据的完整可靠性,共同维护了基线数据的统一版本的快照。CS 也实现了数据的合并逻辑,在特定的时刻所有 CS 会将自身维护的所有基线数据与最新的增量数据融合,必要时会触发 SSTable 的分裂,重写 L_1 层的所有基线数据并提升数据版本。

4) UPS(UpdateServer)　UPS 为 CBase 中的增量数据节点,负责管理存储 MemTable、WAL 以及 L_0 层的数据及其合并过程,为其他服务器提供增量数据的读写接口,以及提供事务的完整 ACID 保障。其中,MemTable 的实现借助了 B+树的组织结构。UPS 按组进行组织,每一组节点负责管理一部分增量数据,组与组之间没有数据重叠,实现了可扩展性,消除了单点瓶颈。每一组内的多个节点一主多备,按 Raft 协议保证数据的一致性,在主 UPS 故障后可以自动切换到备机提供服务,提高了系统的可用性。

5) RS(RootServer)　RS 为 CBase 中的集群管理节点,负责整个集群的节点管理和数据管理。在某个节点上线时,会先向 RS 进行注册,RS 会维护整个集群的节点拓扑关系,并和集群中上述所有节点保持周期性的心跳,用于感知节点的实时状态。RS 维护着数据库中数据分布的元数据信息,计算节点 MS 从而实现根据范围定位一次查询涉及到的数据存储服务器,用于后续的请求分发。RS 的后台负载均衡线程会定期分析元数据的分布情况,按数据子表 SSTable 为单位及时补全副本、负载均衡等。为保障 RS 本身的可用性,一般部署多台 RS,形成 Paxos 组,并借助 Raft 选举算法进行可用性的保障。

3　异步融合机制

由于 CBase 架构采用了 LSM-Tree 的增量融合机制,每天在业务低峰期,会对 UPS 的增量数据和 CS 的基线数据进行融合,合并后数据落盘,释放内存资源,并删除无用数据,减少磁盘存储空间。但现有的合并机制针对 CS 上每一个 SSTable,合并流程都是在单独的线程中顺序串行地执行,即 SSTable 首先读取本地基线数据,然后拉取增量数据,与基线数据进行融合,随后将融合后的新基线数据进行写盘。合并过程会占用大量系统资源,包括磁盘 I/O 和网络资源等,且合并时间过长、合并效率不高,导致内存资源释放缓慢,严重影响系统可用性[14]。

针对该 CBase 架构的数据合并策略,本文提出并实现了一种异步数据融合策略[15],并利用该策略减少合并时数据融合代价和冗余磁盘读写 I/O 的开销,优化了 SSTable 合并的平均时间。

3.1　合并代价分析

由于目前 CBase 系统的合并流程采用多线程多轮同步合并,每一轮选择任一还未合并的 SSTable 进行调度,具体执行时间不易量化。本文将其量化为各个任务执行时间相近的平均值以便阐述,代表一种平均情况,主要用于说明对于单台 CS 而言合并的主要影响因素,故有(1)式所示关系。

$$C_{\mathrm{CS}} \propto \lceil \frac{n}{k} \rceil \times \frac{\sum_{i=0}^{n} C_i}{n} \tag{1}$$

式中：k 为合并的线程数；n 为这台 CS 上所管理的 SSTable 的数量，C_i 为编号为 i 的 SSTable 执行单线程合并所需的时间。根据(1) 式可以得出，提高并发度或者缩短每个 SSTable 合并所需的时间都可对单个 CS 服务器合并所需的时间产生积极影响。

一个实际物理集群中包含多台 CS，在合并过程中并行地进行合并，故系统最终合并的时间取决于合并时间最长的 CS，即有(2)式所示关系

$$C_{\text{Compaction}} = \max(C_{\text{CS1}}, C_{\text{CS2}}, \cdots, C_{\text{CS}n}) \quad (2)$$

在集群中的 CS 全部合并完后，即 $C_{\text{Compaction}}$ 时间后，UPS 会释放这一版本增量数据所占用的内存及磁盘资源。需要注意的是，当上一版本还未合并完成时，下一版本的合并就不会开始。例如，当基线数据版本为 3 时，UPS 上保存着版本为 4 的增量数据，如果在基线数据版本合并到 4 的过程中涌入了大量新数据，则会在 UPS 上的 L_0 层进行数据堆积(如 5，6，7 版本的多个稀疏 SSTable)，随着 L_0 层文件的增多，系统会限制写操作，甚至导致 write stall 写暂停，并且对于用户的所有查询请求，均需要扫描 L_0 层的全量数据进行数据合并，降低了系统的读性能，带来了不小的读放大。由于每日增量数据的庞大，会连续发生多次大版本合并，且单次合并时间的数量单位为小时级，严重影响了系统的可用性。因此合并时间过长也成了 CBase 中亟待优化的问题。

3.2 数据异步融合

数据异步融合的主要思想就是将合并流程做更精细化的模块管理，将必要的操作仍同步进行，一些非必要的操作尽可能地做异步处理，延后进行，以此尽快完成全局数据合并，释放相关资源。其流程如图 3 所示。

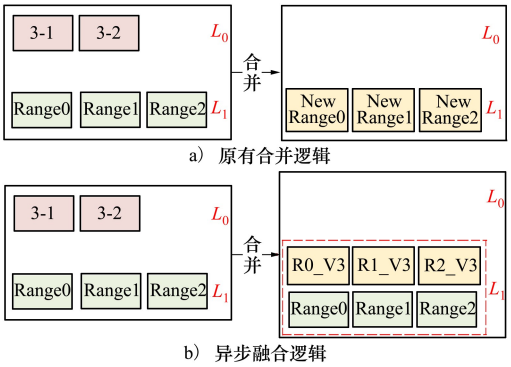

图 3 异步融合流程

如图 3 所示，一次合并行为在之前的系统中需要整个 L_1 层的参与，且 L_1 层的所有 SSTable 都会进行重写，对于基线数据而言，进行了一次冗余的磁盘读写操作。而在异步数据融合机制中，弱化了每一层全部有序的限制，以多版本的方式管理 L_1 层，合并仅仅意味着 L_0 层数据在 L_1 层的持久化，不涉及原基线数据的冗余读写，将数据融合做了异步化处理。通过异步化相关操作的优化，缩短增量数据下落的整个过程，使得增量数据节点的资源释放更快，保证了增量数据节点的连续可写性及读取性能稳定。

异步融合将数据融合落盘的时机做延后处理，相较于原有的数据合并机制，异步融合的策略主要从磁盘 I/O 和 CPU 两方面缩短合并的代价与时间。首先，异步数据融合磁盘 I/O 仅发生在增量数据的持久化过程中，减少了 L_1 层数据重写的磁盘带宽浪费；其次，在异步数据融合中，数据融合仅发生在基线数据拉取的多版本增量数据之间，增量数据相较于基线数据数据量很少，因此减少了基线数据服务器上占用 CPU 进行数据融合的代价。

目前系统的合并流程实现中，增量数据的拉取与基线数据的读取以及后续的数据融合部分较为耦合，在拉取增量前便会读取基线数据并放到内存中。为了减少这部分读取开销，进行更精细化的控制以及模块的解耦合设计，异步数据融合将增量数据拉取模块独立出来，作为 ChunkServer 开始合并的第一步，并设计独立的线程进行增量数据拉取，拉取完成后，将多分区的增量数据直接在增量拉取模块内进行数据融合，将融合后的全量增量数据交给下层模块进行持久化，生成磁盘上的单个数据文件。增量数据的提前融合减少了合并开销。

4 实验结果

实验的目的是验证异步数据融合策略对合并性能的提升效果，主要针对 CBase 相同分区规则和相同数据量场景下优化前后的数据合并时间进行验证。

4.1 实验环境

本文所有实验节点采用同一软硬件配置，保证外部条件完全一致。本实验所采用的详细服务器硬件配置信息如表 1 所示。

表 1　测试用服务器硬件配置表

配置	对应参数
CPU	Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30 GHz x 2(物理核心)
操作系统	Linux OBDHDFS52 2.6.32-431.20.3.el6.x86_64 #1 SMP Fri Jun 6 18:30:54 EDT 2014 x86_64 x86_64 x86_64 GNU/Linux
网卡 1	Intel Corporation I350 Gigabit Network Connection (rev 01)
网卡 2	Intel Corporation 82599EB 10-Gigabit SFI/SFP+Network Connection (rev 01)
网卡网速	10 000 Mbit/s
硬盘	535G SSD×1
内存	504G(16×32G)

4.2　异步数据融合有效性测试

本次实验共使用了若干张数据测试表,这些表具有相同的表 schema 结构,这些测试表的测试数据由 Sysbench 工具自动生成。每张表的分区情况在不同的实验中有所不同。

通过引入异步数据融合前后合并的耗时时间差异测试优化的提升空间,具体实验场景为:

实验 1:在本实验中,包含一张测试表,且为表分区。首先为该表准备 100 万行的数据并进行合并,形成一定量的基线数据(单副本 200 MB)。合并结束后,再次导入 100 万行均匀的增量数据,随后观察优化前后的增量合并所需时间。

实验 2:在本实验中,包含一张测试表,且为行哈希分区,按主键分在 2 个分区组中。其余对比操作同实验 1。

实验 3:在本实验中,包含 50 张测试表,且均为表分区。首先为该每张表准备 10 万行的数据并进行合并,形成一定量的基线数据(单副本 1 GB)。合并结束后,再次为每张表导入均匀的 10 万行增量数据,随后观察优化前后的增量合并所需时间。

在每组实验中,存在优化前后的对比设计,同时,实验 1 和实验 2 的设计测试了异步数据融合对不同分区规则表的优化空间差异,实验 1 和实验 3 的设计测试了不同数据量下异步数据融合策略的表现情况。实验结果如图 4 所示。

由图 4 可以看出,多分区场景下由于增量数据分散引入了额外网络开销,其合并速度会略慢于单分区场景,但不论是单分区还是多分区的增量场景,相较于优化前的合并时间,异步数据融合的策略均带来了更短的合并时间,时间缩短了 20%。而在基线数据更大的场景下,优化效果会更加明显。对于有一定量基线数据的数据库集群而言(常态下均满足此假设),同步重写基线的代价往往造成了读写 I/O 的浪费,并且随着基线数据的膨胀这部分代价会越来越大,而优化后合并时间的缩短,主要归功于数据融合异步化带来的读写基线的开销削减,优化后更短的合并时间可以更好地支持增量数据的连续写入。

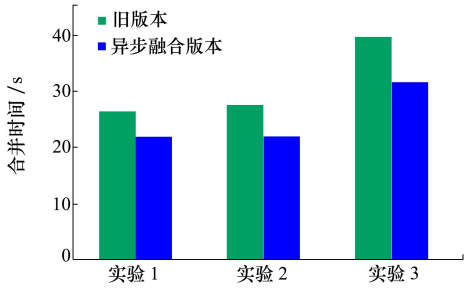

图 4　异步数据融合合并性能测试

4.3　基线数据量对合并时间的影响测试

系统合并时间随着基线数据量的增多而逐渐上升,采用异步数据融合后,对 L_1 层的写操作进行了优化,本实验就是验证优化后的策略在基线数据量增加的情况下合并时间是否减少。

实验 4:在本实验中,包含一张测试表,且为表分区。为该表准备不同量的基线数据,分别为 50 万,100 万,150 万,200 万,250 万,进行合并,随后导入固定均匀的 100 万行增量数据。观察不同测试组中优化前后的增量数据合并时间。

如图 5 所示,优化前旧版本的系统合并时间随着基线数据量的增多而逐渐上升,这是由于在之前的合并流程中,一次合并意味着基线数据的全部重

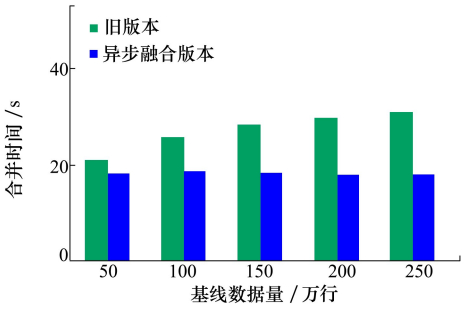

图 5 基线数据量对合并时间的影响测试

写,而重写所需的读写 I/O 开销与基线数据量正相关。而优化后的合并策略不需要对 L_1 层进行全部重写,仅需对增量数据进行落盘,在避免了额外的读写开销后发现优化后的系统合并时间随着基线数据数据量的变化大致保持稳定,符合设计预期。

5 结 论

数据库系统作为组织、存储、管理数据的介质,其底层存储架构随着数据量以及真实场景的变化而不断迭代演进。为了解决数据写入性能瓶颈以及满足存储海量数据的需要,基于 LSM-Tree 存储的分布式数据库应运而生,并被广泛应用于金融、互联网等行业,服务其写密集的业务场景。合并行为是为了优化 LSM-Tree 存储引擎的读取性能,释放内存资源以支持连续写入的常用操作,但合并本身的资源代价十分昂贵,一次合并行为的耗时往往在小时的数量级,这严重影响了资源的释放复用,降低了数据库系统的可用性。

本文在分析了 CBase 系统合并代价的基础上,设计了一种异步数据融合模型,该模型主要针对基线数据节点,通过在细分合并行为基础上的模块化设计,将增量数据的持久化和增量数据与基线数据的数据融合解耦,异步且有选择地进行耗时的数据融合,在缩短了整体合并时间的基础上,也为用户提供了多样的选择。

本文所提出的异步融合策略已经在分布式数据库 CBase 上进行了验证,实验结果表明,该策略能大幅降低合并时间,提供系统可用性。

参考文献:

[1] LU W, ZHAO Z H, WANG X Y, et al. A lightweight and efficient temporal database management system in TDSQL[J]. Proceedings of the VLDB Endowment, 2019, 12(12): 2035-2046

[2] O′ NEIL P, CHENG E, GAWLICK D, et al. The log-structured merge-tree(LSM-tree)[J]. Acta Informatica, 1996, 33: 351-385

[3] HUANG D, LIU Q, CUI Q, et al. TiDB: a raft-based HTAP database[J]. Proceedings of the VLDB Endowment, 2020, 13(12): 3072-3084

[4] YANG Z, YANG C, HAN F, et al. OceanBase: a 707 million tpmC distributed relational database system[J]. Proceedings of the VLDB Endowment, 2022, 15(12): 3385-3397

[5] LUO C, CAREY M J. LSM-based storage techniques: a survey[J]. The International Journal on Very Large Data Bases, 2020, 29(1): 393-418

[6] YAN B, CHENG X, JIANG B, et al. Revisiting the design of LSM-tree based OLTP storage engine with persistent memory[J]. Proceedings of the VLDB Endowment, 2021, 14(10): 1872-1885

[7] SARKAR S, STARATZIS D, ZHU Z, et al. Constructing and analyzing the LSM compaction design space[J]. Proceedings of the VLDB Endowment, 2021, 14(11): 2216-2229

[8] BINDSCHAEDLER L, GOEL A, ZWAENEPOEL W. Hailstorm: disaggregated compute and storage for distributed LSM-based databases[C]//Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems, 2020: 301-316

[9] HUANG H, GHANDEHARIZADEH S. Nova-LSM: a distributed, component-based LSM-tree key-value store[C]//Proceedings of the 2021 International Conference on Management of Data, 2021

[10] DAYAN N, WEISS T, DASHEVSKY S, et al. Spooky: granulating LSM-tree compactions correctly[J]. Proceedings of the VLDB Endowment, 2022, 15(11): 3071-3084

[11] 张晨煜, 刘文洁, 庞天泽, 等. 基于分布式数据库的相关子查询优化[J]. 西北工业大学学报, 2021, 39(4): 909-918

ZHANG Chenyu, LIU Wenjie, PANG Tianze, et al. Optimization of correlate subquery based on distributed database[J]. Journal of Northwestern Polytechnical University, 2021, 39(4): 909-918 (in Chinese)

[12] 景荟弘, 刘文洁, 高锦涛, 等. 面向分布式数据库的 HTAP 研究与实现[J]. 西北工业大学学报, 2021, 39(2): 430-438
JING Changhong, LIU Wenjie, GAO Jintao, et al. Research and implementation of HTAP for distributed database[J]. Journal of Northwestern Polytechnical University, 2021, 39(2): 430-438 (in Chinese)

[13] 刘文洁, 李戬勃, 李战怀, 等. 一种面向金融应用的海量分布式关系数据库[J]. 华中科技大学学报, 2019, 47(2): 121-126
LIU Wenjie, LI Jianbo, LI Zhanhuai, et al. A massire distribnted relational database for financial application[J]. Journal of Huazhong University of Science and Technology, 2019, 47(2): 121-126 (in Chinese)

[14] DAYAN N, IDREOS S. Dostoevsky: better space-time trade-offs for LSM-tree based key-value stores via adaptive removal of superfluous merging[C]//Proceedings of the 2018 International Conference on Management of Data, 2018

[15] RAJU P, KADEKODI R, CHIDAMBARAM V, et al. Pebblesdb: building key-value stores using fragmented log-structured merge trees[C]//Proceedings of the 26th Symposium on Operating Systems Principles, 2017

Research and implementation of asynchronous compaction mechanism of distributed database based on LSM-Tree

DU Yide, LIU Wenjie

(School of Computer Science, Northwestern Polytechnical University, Xi'an 710072, China)

Abstract: With the continuous development of information technology, distributed database has become a research hotspot. Due to the limit support for SQL and defects in transaction processing and consistency of distributed databases based on NoSQL architecture, NewSQL databases based on LSM-Tree become gradually the mainstream of applications, such as TiDB and OceanBase. The distributed LSM-Tree storage architecture divides the data into baseline data and incremental data. Through the compaction operation, the incremental data of different partitions and the baseline data are continuously merged and stored on the disk, thereby reducing memory pressure. However, compaction will occupy a large amount of system resources and seriously affect system availability. This paper proposes an asynchronous compaction mechanism based on LSM-Tree architecture. By subdividing the compaction process, the data merging is asynchronous, which effectively shortens the time for a single compaction operation. Experiments show that the asynchronous compaction mechanism proposed in this paper can significantly shorten the data merging time and improve the robustness and usability of the system in high-frequency writing scenarios.

Keywords: distributed database; LSM-Tree; data merging; asynchronous compaction; data partitioning

引用格式: *杜轶德, 刘文洁. 基于 LSM-Tree 的分布式数据库异步融合机制研究与实现[J]. 西北工业大学学报, 2024, 42(2): 303-309*

DU Yide, LIU Wenjie. Research and implementation of asynchronous compaction mechanism of distributed database based on LSM-Tree[J]. Journal of Northwestern Polytechnical University, 2024, 42(2): 303-309 (in Chinese)