

数据分片处理方法、装置、设备、存储介质及程序产品

本申请要求于 2023 年 11 月 06 日提交中国专利局、申请号为 2023114701631、申请名称为“数据分片处理方法、装置、设备、存储介质及程序产品”的中国专利申请的优先权，其全部内容通过引用结合在本申请中。

5 技术领域

本申请涉及分布式存储技术领域，特别涉及数据分片处理技术。

背景技术

在支持分布式事务的系统中，及时进行数据分布的调度，能够分散突发热点，实现负载均衡。

10 在相关技术中，可以由管理员手动修改数据分布策略；也可以通过在阻塞事务期间分裂过载数据分片的方式，自动实现负载均衡。

然而，相关技术中的手动方式涉及大量的数据搬迁，很容易出现失误，对事务处理造成影响；相关技术中的自动方式对过载数据分片进行分裂处理，将会延长阻塞事务时间，容易对事务的读写性能和提交性能造成影响。

15 发明内容

本申请提供了一种数据分片处理方法、装置、设备、存储介质及程序产品，可以在保证分布式事务的执行效率的同时，缩短阻塞事务的时间；所述技术方案内容如下。

根据本申请的一方面，提供了一种数据分片处理方法，由计算机设备执行，所述方法包括：

20 响应于对数据库中的第一数据分片的分裂指令，创建第二数据分片；所述分裂指令用于指示将所述第一数据分片中的数据分裂到所述第二数据分片；

控制所述第一数据分片中未提交的事务暂停进入提交阶段；

响应于所述第一数据分片中已进入所述提交阶段的事务提交完毕，控制所述第一数据分片中未读写的事务暂停进入读写阶段；

25 将第一事务数据从所述第一数据分片转移到所述第二数据分片；所述第一事务数据是第一事务的数据中对应于分裂后的所述第二数据分片的数据；所述第一事务是所述第一数据分片的活跃事务中需转移数据至所述第二数据分片的事务，所述活跃事务为处于所述读写阶段的事务；

30 响应于所述第一事务数据转移完成，控制所述第一数据分片中的事务恢复进入所述读写阶段和/或所述提交阶段。

根据本申请的一方面，提供了一种数据分片处理装置，所述装置包括：

创建模块，用于响应于对数据库中的第一数据分片的分裂指令，创建第二数据分片；所述分裂指令用于指示将所述第一数据分片中的数据分裂到所述第二数据分片；

第一暂停模块，用于控制所述第一数据分片中未提交的事务暂停进入提交阶段；

第二暂停模块，用于响应于所述第一数据分片中已进入所述提交阶段的事务提交完毕，控制所述第一数据分片中未读写的事务暂停进入读写阶段；

转移模块，用于将第一事务数据从所述第一数据分片转移到所述第二数据分片；所述第一事务数据是第一事务的数据中对应于分裂后的所述第二数据分片的数据；所述第一事务是所述第一数据分片的活跃事务中需转移数据至所述第二数据分片的事务，所述活跃事务为处于所述读写阶段的事务；

恢复模块，用于响应于所述第一事务数据转移完成，控制所述第一数据分片中的事务恢复进入所述读写阶段和/或所述提交阶段。

根据本申请的另一方面，提供了一种计算机设备，所述计算机设备包括处理器和存储器，所述存储器中存储有至少一条指令、至少一段程序、代码集或指令集，所述至少一条指令、所述至少一段程序、所述代码集或指令集由所述处理器加载并执行以实现如上方面所述的数据分片处理方法。

根据本申请的另一方面，提供了一种计算机可读存储介质，所述可读存储介质中存储有至少一条指令、至少一段程序、代码集或指令集，所述至少一条指令、所述至少一段程序、所述代码集或指令集由处理器加载并执行以实现如上方面所述的数据分片处理方法。

根据本申请的另一方面，提供了一种计算机程序产品，所述计算机程序产品包括计算机指令，所述计算机指令存储在计算机可读存储介质中，处理器从所述计算机可读存储介质读取并执行所述计算机指令，以实现上述如上方面所述的数据分片处理方法。

本申请实施例提供的技术方案可以包括如下有益效果：

本申请实施例所示的方案，响应于对数据库中的第一数据分片的分裂指令，创建第二数据分片；控制第一数据分片中未提交的事务暂停进入提交阶段；响应于第一数据分片中已进入提交阶段的事务提交完毕，控制第一数据分片中未读写的事务暂停进入读写阶段；将第一事务数据从第一数据分片转移到第二数据分片；响应于第一事务数据转移完成，控制第一数据分片中的事务恢复进入读写阶段和/或提交阶段。本方案在阻塞事务之前，先进行第二数据分片的构建；同时，将阻塞事务分为控制未提交的事务暂停进入提交阶段、以及控制未读写的事务暂停进入读写阶段这两个阶段执行，在阻塞事务提交时，不影响事务的读写以及即将提交完成的事务。本申请能够在保证分布式事务的执行效率的同时，缩短阻塞事务的时间。

附图说明

为了更清楚地说明本申请实施例中的技术方案，下面将对实施例描述中所需要使用的附图作简单地介绍，显而易见地，下面描述中的附图仅仅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。

图1是本申请涉及的一种传统分库分表架构的分布式数据库架构图；

图2是本申请涉及的一种分布式数据库架构图；

图3是本申请一个示例性实施例提供的的数据分片处理方法流程图；

- 图 4 是本申请另一个示例性实施例提供的数据分片处理方法的流程图；
图 5 是本申请又一个示例性实施例提供的数据分片处理方法的流程图；
图 6 是本申请再一个示例性实施例提供的数据分片处理方法的流程图；
图 7 是本申请一个示例性实施例提供的数据分片处理方法中暂停第一数据分片对应的
5 的新事务进入读写阶段的流程图；
图 8 是本申请另外一个示例性实施例提供的数据分片处理方法的流程图；
图 9 是本申请涉及的另一分布数据库架构图；
图 10 是本申请一个示例性实施例提供的一种数据分片处理方法的实施流程图；
图 11 是本申请一个示例性实施例提供的一种分裂流程图；
10 图 12 是本申请涉及的一种 2PC 协议的流程图；
图 13 是本申请一个示例性实施例提供的一种 2PC 案例示意图；
图 14 是图 13 的 2PC 案例的提交示意图；
图 15 是本申请一个示例性实施例提供的一种 2PC 流程图；
图 16 是本申请一个示例性实施例示出的数据分片处理装置的方框图；
15 图 17 是本申请一个示例性实施例提供的计算机设备的结构框图。

此处的附图被并入说明书中并构成本说明书的一部分，示出了符合本申请的实施例，并与说明书一起用于解释本申请的原理。

具体实施方式

为使本申请的目的、技术方案和优点更加清楚，下面将结合附图对本申请实施方式作
20 进一步地详细描述。

这里将详细地对示例性实施例进行说明，其示例表示在附图中。下面的描述涉及附图时，除非另有表示，不同附图中的相同数字表示相同或相似的要素。以下示例性实施例中
所描述的实施方式并不代表与本申请相一致的所有实施方式。相反，它们仅是与如所附权
利要求书中所详述的、本申请的一些方面相一致的装置和方法的例子。

25 在本公开使用的术语是仅仅出于描述特定实施例的目的，而非旨在限制本公开。在本
公开和所附权利要求书中所使用的单数形式的“一种”、“所述”和“该”也旨在包括多数形式，
除非上下文清楚地表示其他含义。还应当理解，本文中使用的术语“和/或”是指并包含一个
或多个相关联的列出项目的任何或所有可能组合。

需要说明的是，本申请所涉及的用户信息（包括但不限于用户设备信息、用户个人信
30 息等）和数据（包括但不限于用于分析的数据、存储的数据、展示的数据等），均为经用户
授权或者经过各方充分授权的信息和数据，且相关数据的收集、使用和处理需要遵守相关
国家和地区的相关法律法规和标准。例如，本申请中涉及到的攻击操作等对象行为都是在
充分授权的情况下获取的。

应当理解，尽管在本公开可能采用术语第一、第二等来描述各种信息，但这些信息不
35 应限于这些术语。这些术语仅用来将同一类型的信息彼此区分开。例如，在不脱离本公开
范围的情况下，第一参数也可以被称为第二参数，类似地，第二参数也可以被称为第一参
数。取决于语境，如在此所使用的词语“如果”可以被解释成为“在……时”或“当……时”或“响

应于确定”。

下面介绍本申请涉及的一些名词释义：

数据库 (Database, DB)：是按一定结构组织并长期存储在计算机内的、可共享的大量数据的有机集合。每个数据库都有一个或多个应用程序编程接口 (Application Programming Interface, API)，用于创建、访问、管理、搜索和复制数据库中保存的数据。

数据分片：是一个数据的最小管理单位，可以用来灵活地做数据调度，比如当某个节点负载特别高的时候，可以将其中一部分数据分片迁移到其他节点。

写容器 (writebatch)：是一个用来存放活跃事务数据的数据结构，每个事务都会有自己的 write batch，一个 write batch 中存放的数据只属于一个事务。对于分布式事务，其中每个事务在每个 Region 的事务上下文中都会有自己的 write batch。

事务数据：是事务上下文的一部分。分裂事务时，将 Old Region 对应的事务数据发送到 NewRegion 后，会在 NewRegion 上构建出事务上下文，确保 Old Region 和 NewRegion 都有对应的事务上下文。

Raft 组 (Raft Group)：数据同步组的基本单位，包含多个副本，其中一个副本为主副本 (Leader)，其余一个或者多个副本为从副本 (Follower)。Leader 负责将数据同步给 Follower，具体做法是将一条 Raft 日志 (Raft Log) (也称为计划 Proposal) 发给所有 Follower，当这条 Raft Log 被多数 Follower 持久化在本地磁盘后，副本就可以执行 (Apply) 这条 Raft Log，即根据 Raft Log 中包含的具体内容执行对应的操作。

两阶段提交 (2 Phase Commit, 2PC)：是一种保证分布式事务提交原子性的通用做法。2PC 将事务的提交分成两个阶段：准备阶段 (Prepare Phase, P) 和提交阶段 (Commit Phase, C)。事务的发起者称为协调者 (Coordinator)，通常只有一个，负责决策整个分布式事务的提交和回滚；事务的执行者称为参与者 (Participants)，通常有多个，负责自己本地事务的提交和回滚。

远程过程调用协议 (Remote Procedure Call, RPC)：一种通过网络从远程计算机程序上请求服务，而不需要了解底层网络技术的协议。RPC 是一种模块间通信的方式，通过网络向远程节点请求服务。

每秒请求数 (Query Per Second, QPS)：又称每秒查询率，是一台服务器每秒能够响应的查询次数，或者是一台服务器每秒能够处理的请求数。QPS 是对一个特定的查询服务器在规定时间内所处理流量多少的衡量标准，也是最大吞吐能力。

请参考图 1，其示出了本申请涉及的一种传统分库分表架构的分布式数据库架构图。

如图 1 所示，传统分库分表架构的分布式数据库，是一种简单的数据分布方案。底层的每个存储节点本质都是一个单机数据库。对于业务的一张大数据量的表而言，底层会将其按照用户指定的分库键或者分表键，将原表拆分成多张表，散布在不同存储节点上。所有存储节点的上层有一个代理 (proxy) 节点做路由及数据聚合，使得用户不需要关心底层数据实际的分布情况，即可做数据查询。

请参考图 2，其示出了本申请涉及的一种分布式数据库架构图。

—5—

如图 2 所示，该分布式数据库架构将整个分布式数据库拆分成以下三个模块：

存储集群：存储 KV 数据，会根据 Key 值对数据进行数据分片（Region），分散数据热点。其中，每个数据分片背后对应一个 Raft Group，实现主备数据同步。

计算集群：将用户数据的 SQL 语句转化为 KV 请求，发送给存储集群。

- 5 中央管控集群：检测存储集群的数据分布情况，指挥存储集群进行数据分片的分裂、合并、迁移，保证负载均衡，同时也提供全局时间戳授时功能。

存储集群中的每个节点会定期通过心跳方式，将本地数据分片的信息上报给中央管控集群中的节点。中央管控集群中的节点如果发现某个数据分片的大小超过了设定的阈值，就会向该数据分片的 Leader 下发分裂数据分片的任务，任务中带有拆分后数据分片的范围（包含原有数据分片的范围，和新分裂出来的数据分片的范围）。Leader 收到任务后，会在 Raft Group 中提交一条 Proposal。当这条 Proposal 被同步到多数副本，各个副本去 Apply 这条 Proposal 的时候，会各自创建出一个新的数据分片（New Region）。当多数副本都创建了 New Region 的时候，New Region 底层对应的 Raft Group 也就达到了 Raft 的多数派要求，此时即可选举出 Leader，并提供服务。

- 15 存储集群通过“关键字”来区分每一条数据。例如，对于行存数据库，存储集群所存储的每一条记录，都对应表中的一行数据。对于传统分库分表架构的分布式数据库，数据的“关键字”可以是主键或者分表键的值；而对于基于 KV 存储引擎的分布式数据库，数据的“关键字”可以是编码后得到的 Key 值。数据的“关键字”决定了数据被划分到的区间，目前，常见的数据划分方案可大致分为“基于关键字哈希值划分”和“基于关键字区间划分”。

- 20 “基于关键字哈希值划分”的划分方式基于一个关键字哈希函数，该函数的输入为“关键字”的值，输出为一个无符号整型。通过这种方式，数据库系统可以构建一个从“键值”到“哈希值”的映射关系。在分区的时候，每个分区对应一段无符号整型的范围，每个分区的范围不存在交集。如果某条数据的关键字值经过哈希函数所计算出的哈希值位于某分区的整型范围内，则该条数据会被存储在该分区中。

- 25 这种划分数据的好处在于，只要哈希函数计算的足够合理，存储系统就可以将大量数据均分在多个分区中，保证每个分区数据量几乎相同，因此这种数据划分方式具有良好的分散热点的效果。

“基于关键字区间划分”的划分方式，让每个分区直接映射到一段连续的键值区间，每个分区的范围不存在交集。这种划分方式不依赖于哈希函数，较为简单。

- 30 “基于关键字区间划分”的优点在于，当在做范围查询时，只需要找到查询范围所包含的若干个数据分区，然后仅对这几个分区做数据查询即可。这种查询方式缩小了数据查询的范围，提高了查询效率，避免了不必要的资源消耗。

分布式事务的生命周期，可以大致分为两个阶段：读写阶段和提交阶段。

一、读写阶段

- 35 在这个阶段中，用户会通过向数据库系统下发 SQL 请求，来进行数据的读取和写入。读写阶段写入的数据，在事务提交前，属于未提交数据，根据事务的隔离性要求，这部分数据不允许被其他事务访问。

在一些分布式数据库系统中，未提交的事务数据被缓存在计算层，直到事务进入提交阶段前，计算节点才会将事务数据发送给存储集群的对应数据分片上。由于存储集群上不缓存未提交的事务数据，因此存储集群中的数据分片分裂、合并等调度，不会对正在执行的事务造成影响，一定程度上保证了数据调度对用户无感知。

5 二、提交阶段

当用户向数据库系统下发提交（COMMIT）SQL 语句时，事务进入提交流程。此时数据库系统会针对事务做一些检测，判断事务提交是否会破坏数据的一致性（比如针对基于乐观并发控制的事务，检测其是否和其他事务有数据冲突），并通过日志（比如落盘 WAL 日志，或者向 Raft Group 中提交 Proposal），将事务所写的数据持久化下来。如果事务检测通过，则事务可以成功提交，此后事务所写的数据即可对其他事务可见。

在一些分布式数据库系统中，出于简单考虑，分布式事务的原子性由计算层保证（如利用提交协议 Percolator）。在这种架构下，存储引擎是不需要具备事务的概念的，即存储引擎不保证多条 KV 数据写入的原子性。这样一来，存储引擎针对 KV 数据写入和数据分片调度的并发处理，就会更加方便，如果有某条 KV 因为数据分片调度的缘故写入失败，只需要针对单条 KV 重试即可，不需要对事务的所有数据做重试。

从上述数据划分策略的分析来看，“基于关键字哈希值划分”的划分方式，具有良好的分散热点的效果，但是缺点在于范围查询需要访问所有的数据分区，效率不高。而“基于关键字区间划分”缺点在于数据热点依然存在，业务对数据的访问如果集中在某一小段范围内，则需要频繁地触发数据分片分裂，才可以拆分热点。

考虑到数据查询效率应该是最优先需要保证的，本申请实施例的技术方案优先采用“基于关键字区间划分”的方式。

针对数据热点问题，一是业务侧可以对数据做一些随机化处理，比如，对于容易产生数据热点的业务，可以由业务侧在写入数据时，在主键列上进行哈希函数计算，保证插入数据时，能够将数据分散到不同的数据分片上。二是只要在存储层将数据分片分裂对事务的影响优化得足够低，就能通过加快数据分片分裂的频率来降低数据热点的影响。本申请实施例的技术方案提供的数据分片分裂思路，做到了数据分裂对业务无感知，可以应对这一问题。

传统分库分表架构的分布式数据库，虽然实现起来相对简单，但是由于缺乏良好的数据分布检测手段，在数据划分方面控制得不够灵活，不能对突发的热点数据做迅速的调度。比如，在建表的时候，需要由数据库管理员（Database Administrator，DBA）人工指定分表键。而对于突发的热点，DBA 也需要手动修改数据分布策略，来将数据分摊到相邻的节点。这种数据分散的方式不仅需要消耗人力资源，而且涉及到大量的数据搬迁，很容易对业务造成影响。

所以，一种可选的数据分片策略是：基于分布式数据库新架构，通过中央管控集群来控制数据分片的分裂、合并和迁移，不再依靠 DBA 的支持，即可自动实现负载均衡。

但是，现有的数据分片方案还是会对业务造成影响。如上面提到的，通过在 Raft Group 中同步 Proposal 的方式，需要等待 Proposal 同步到多数派后，才能构建新的 Raft Group，

再经过一轮选举之后，新的数据分片才能对外提供数据写入的服务。这一周期跨越了一次 Raft Group 选举超时时间，通常需要花费数秒。而旧的数据分片在同步了 Proposal 之后，将不再支持向新的数据分片对应的范围写入数据，这也就意味着新的数据分片对应的数据范围在这数秒内是不可用的，容易造成 QPS 的下滑。

- 5 对于基于 Percolator 事务协议的分布式数据库，由于存储集群不需要保证多条数据写入的原子性，因此在处理数据写入和数据分片分裂的并发时会更加简单。但是这种实现的代价是，事务最终要落盘到每一条 KV 数据，在提交过程中，必须要额外写两条 KV 来保证事务的原子性（根据 Percolator 协议的要求，这三条 KV 数据，分别写到数据列族 data column family、锁列族 lock column family、写列族 write column family）。这意味着如果使用 Percolator
- 10 来保证事务的原子性，则计算集群和存储集群之间的数据通信量、存储集群 Raft Log 同步的 Proposal 数量、数据落盘的输入/输出（Input/Output, I/O）开销都会提高至 2~3 倍，提交性能较差。

- 为了保证分布式事务的提交效率，本申请后续实施例提供的技术方案将分布式事务完全下沉至存储集群。对于读写阶段的事务，其未提交的数据被维护在事务上下文中，由数
- 15 据所在的分片在内存中管理。在这种设计下，本方案依然可以保证当触发数据分片的分裂时，事务可以继续执行，不会被回滚，QPS 也不会有波动。在保证分布式事务的执行效率前提下，做到了数据分片分裂对业务无感知。

- 请参考图 3，其示出了本申请一个示例性实施例提供的的数据分片处理方法流程图。该方法由计算机设备执行。如图 3 所示，该方法可以包括以下步骤 310、步骤 320、步骤 330、
- 20 步骤 340 以及步骤 350。

步骤 310：响应于对数据库中的第一数据分片的分裂指令，创建第二数据分片；分裂指令用于指示将第一数据分片中的数据分裂到第二数据分片。

- 在本申请实施例中，当收到针对数据库中的第一数据分片的分裂指令时，计算机设备
- 25 可以在第一数据分片的存储集群节点创建新的第二数据分片。

 其中，分裂指令用于指示将第一数据分片中的数据分裂到新的第二数据分片，即将第一数据分片中的部分数据转移到第二数据分片中。

 其中，上述第一数据分片可以是一个 Raft Group 中的主副本，计算机设备可以将上述分裂指令同步到第一数据分片对应的各个从副本。

- 30 上述分裂指令用于指示数据分片的分裂，比如一个节点上有一个数据分片包含了特别多的数据，计算机设备可以通过分裂指令指示将其中的一部分迁移到其他节点，此时，需要在当前节点上进行数据分片的分裂。

 一方面，上述分裂指令，可以由中央管控集群检测到需要迁移的热点数据后，自动下发至该热点数据对应的数据分片（Region）所在的存储集群节点。

- 35 另一方面，上述分裂指令，也可以由开发者用户向中央管控集群发送超文本传输协议（Hypertext Transfer Protocol, HTTP）请求，中央管控集群基于 HTTP 请求，向对应的数据分片（Region）所在的存储集群节点下发分裂指令。

另外，开发者用户也可以通过向中央管控集群发送 HTTP 请求，来手动开启或关闭自动分裂功能。关闭后，中央管控集群将不会自行下发分裂指令。

在一些实施例中，上述分裂指令可以指示第一数据分片的 ID、第一数据分片的范围，以及新的第二数据分片的范围，包含分裂点 (split key)、起点 (start key)、终点 (end key)、数据分片标识号 (Region ID) 等。

具体比如，第一数据分片的范围是[A, D)，以 split key 为 B 分裂出新的第二数据分片，那么，第二数据分片的范围可以是[B, D)，即 start key 为 B，end key 为 D；相应的，分裂后的第一数据分片的范围可以为[A, B)。

在一些实施例中，上述步骤 310 之前包括计算 split key 的前置步骤，具体包括：
S11：中央管控集群向存储集群中待分裂的第一数据分片下发一个计算 split key 的任务，第一数据分片计算出 split key 后，将 split key 上报给中央管控集群；
S12：中央管控集群得到第一数据分片计算的 split key 之后，根据该 split key 生成对应的分裂指令，进而向第一数据分片以及第一数据分片的所有从副本下发分裂指令。

步骤 320：控制第一数据分片中未提交的事务暂停进入提交阶段。
在本申请实施例中，通过步骤 310 响应于对数据库中的第一数据分片的分裂指令，创建第二数据分片之后，计算机设备可以控制第一数据分片中未提交的事务暂停进入提交阶段。

在本申请实施例中，在完成第二数据分片的构建之后，再控制未提交的事务暂停进入提交阶段，而处于读写阶段的事务以及新开启的事务不受影响。上述方案能够缩短阻塞基于第一数据分片执行的事务进入提交阶段的时间，降低数据分片的分裂对事务的影响。

在一些实施例中，步骤 320 之前可以包括以下步骤：
S21：构建第一数据分片的分裂任务上下文；其中，分裂任务上下文包含一些任务信息，比如第一数据分片 ID、第二数据分片 ID、split key 等等；
S22：设置分裂任务状态为 REGION_SPLIT_INIT；
S23：发送 RPC 确认第一数据分片的从副本都进入 REGION_SPLIT_INIT 状态。

步骤 330：响应于第一数据分片中已进入提交阶段的事务提交完毕，控制第一数据分片中未读写的事务暂停进入读写阶段。

在本申请实施例中，第一数据分片中已进入提交阶段的事务提交完毕后，计算机设备可以控制第一数据分片对应的新事务暂停进入读写阶段，即暂停新事务的开启，该新事务即为第一数据分片中尚未进入读写阶段的事务，但上述操作对于已处于读写阶段的事务不产生影响。

在本申请实施例中，在进行步骤 340 数据转移之前，将阻塞事务的过程分为两个阶段，包括步骤 320 阻塞第一数据分片中未提交的事务进入提交阶段，以及步骤 330 在已进入提交阶段的事务提交完毕后，再暂停第一数据分片对应的新事务进入读写阶段。上述方案能够降低数据分片的分裂对事务的影响。

步骤 340：将第一事务数据从第一数据分片转移到第二数据分片；第一事务数据是第一事务的数据中对应于分裂后的第二数据分片的数据；第一事务是第一数据分片的活跃事务

中需转移数据至第二数据分片的事务，活跃事务为处于读写阶段的事务。

在本申请实施例中，计算机设备可以遍历第一数据分片对应的活跃事务（即处于读写阶段的事务），通过 RPC 将第一事务数据从第一数据分片转移到第二数据分片。

在本申请实施例中，第一数据分片的活跃事务中，可能只有部分活跃事务的事务数据
5 需要全部或者部分转移到第二数据分片，因此，需要根据分裂指令做出判断，即对第一数据分片的每个活跃事务的数据进行遍历，以便筛选出需要转移到第二数据分片的事务数据。

具体比如，第一数据分片的活跃事务包括事务 1、事务 2 和事务 3，分裂指令用于指示将 split key 之后的数据转移到第二数据分片。若事务 1 的数据在 split key 之前，则事务 1 的数据不需要转移；若事务 2 的数据在 split key 之后，则事务 2 的数据需要全部转移到第二数据分片；若事务 3 的数据一部分在 split key 之前，一部分在 split key 之后，则需要将
10 事务 3 中在 split key 之后的部分数据转移到第二数据分片。

步骤 350：响应于第一事务数据转移完成，控制第一数据分片中的事务恢复进入读写阶段和/或提交阶段。

在本申请实施例中，通过步骤 340 成功完成数据转移后，实现了第一数据分片上数据
15 热点的分散，计算机设备可以恢复第一数据分片对应的新事务进入读写阶段，以及恢复读写阶段的事务进入提交阶段。

综上所述，本申请实施例所示的方案，响应于对数据库中的第一数据分片的分裂指令，创建第二数据分片；控制第一数据分片中未提交的事务暂停进入提交阶段；响应于第一数据分片中已进入提交阶段的事务提交完毕，控制第一数据分片中未读写的事务暂停进入读写阶段；将第一事务数据从第一数据分片转移到第二数据分片；响应于第一事务数据转移完成，控制第一数据分片中的事务恢复进入读写阶段和/或提交阶段。本方案在阻塞事务之前，先进行第二数据分片的构建；同时，将阻塞事务分为控制未提交的事务暂停进入提交阶段、以及控制未读写的事务暂停进入读写阶段这两个阶段执行，在阻塞事务提交时，不影响事务的读写以及即将提交完成的事务。本申请能够在保证分布式事务的执行效率的同时，
20 缩短阻塞事务的时间。

请参考图 4，其示出了本申请另一个示例性实施例提供的的数据分片处理方法的流程图。该方法由计算机设备执行。如图 4 所示，上述图 3 所示实施例中的步骤 340 可以实现为步骤 340-10、步骤 340-20、步骤 340-30、步骤 340-40 以及步骤 340-50。

步骤 340-10：暂停活跃事务的写操作。

在本申请实施例中，通过步骤 330 控制第一数据分片中未读写的事务暂停进入读写阶段之后，计算机设备可以暂停第一数据分片对应的活跃事务的写操作；以便后续进行数据分裂。

其中，在禁写期间，如果存储集群收到计算集群下发的写请求，则返回重试错误码。

步骤 340-20：在第一数据分片中建立第一事务的第一容器和第一事务的第二容器。

在本申请实施例中，步骤 340-10 暂停活跃事务的写操作之后，计算机设备可以在第一数据分片中建立第一事务的第一容器和第一事务的第二容器，以便将事务数据写入对应的

容器；也就是说，上述第一容器和第二容器分别用于存放第一事务中的 split key 之前和 split key 之后的数据。

在另一些实施例中，计算机设备也可以对第一数据分片上第一事务原有的写容器进行拆分，将其拆分成了第一容器和第二容器，分别用于存放第一事务中的 split key 之前和 split key 之后的数据。

步骤 340-30：将第二事务数据写入第一容器，并将第一事务数据写入第二容器；第二事务数据是第一事务的数据中对应于分裂后的第一数据分片的数据。

在本申请实施例中，步骤 340-20 建立第一事务的第一容器和第一事务的第二容器之后，计算机设备遍历处于读写阶段的事务数据，将第二事务数据写入第一容器，并将第一事务数据写入第二容器。

也就是说，split key 之后的事务数据为第一事务数据，split key 之前的事务数据为第二事务数据。若事务 1 的数据在 split key 之前，则事务 1 的数据属于第二事务数据，需要写入第一容器；若事务 2 的数据在 split key 之后，则事务 2 的数据属于第一事务数据，需要写入第二容器；若事务 3 的数据一部分在 split key 之前，一部分在 split key 之后，则事务 3 中在 split key 之后的数据属于第一事务数据，需要写入第二容器，在 split key 之前的数据属于第二事务数据，需要写入第一容器。

步骤 340-40：将第二容器复制到第二数据分片，并在第二数据分片上构建第一事务的事务上下文。

在本申请实施例中，步骤 340-30 将第二事务数据写入第一容器，并将第一事务数据写入第二容器之后，计算机设备可以将第二容器复制到第二数据分片，并在第二数据分片上，基于第二容器中的第一事务数据构建第一事务的事务上下文；以便第二数据分片能够基于第一事务的事务上下文提供相应的对外服务。

在一些实施例中，计算机设备可以将第二容器的数据通过 RPC 发送至第二数据分片，并在第二数据分片上构建出事务上下文。

步骤 340-50：响应于第二数据成功复制到第二数据分片，且第二数据分片上第一事务的事务上下文成功构建，恢复活跃事务的写操作。

在本申请实施例中，步骤 340-40 将第二容器成功复制到第二数据分片，并在第二数据分片上成功构建第一事务的事务上下文，即可认为分裂成功，计算机设备可以恢复第一数据分片对应的活跃事务的写操作，以便继续对外服务。

在本申请实施例中，提供了转移第一事务数据的一种示例性方案，包括在活跃事务的禁写阶段，在第一数据分片中建立第一事务的第一容器和第一事务的第二容器；再将第二容器复制到第二数据分片，并在第二数据分片上构建第一事务的事务上下文；本方案能够在活跃事务的禁写阶段，将需要转移的第一事务数据从第一数据分片转移到第二数据分片，避免事务数据转移与事务写操作的冲突。

请参考图 5，其示出了本申请又一个示例性实施例提供的的数据分片处理方法的流程图。该方法由计算机设备执行。如图 5 所示，上述图 4 所示实施例中的步骤 340-50 之前，还包括步骤 340-42 和步骤 340-44。

步骤 340-42: 响应于第二数据成功复制到第二数据分片, 且第二数据分片上第一事务的事务上下文成功构建, 将第一数据分片中的第一容器更新为第一事务在第一数据分片中的写容器。

5 在本申请实施例中, 通过步骤 340-40 将第二容器成功复制到第二数据分片, 并在第二数据分片上成功构建第一事务的事务上下文, 即可认为分裂成功, 计算机设备可以将第一数据分片中的第一容器更新为第一事务在第一数据分片中的写容器; 以便通过第一容器为第一事务提供服务。

在一些实施例中, 分裂成功后, 计算机设备可以根据 split key 等分裂指令, 修改第一数据分片的 end key 信息; 将第二数据分片移动到对外服务的数据分片列表中。

10 步骤 340-44: 释放第一数据分片中的第二容器、以及第一事务在第一数据分片中原有的写容器。

在本申请实施例中, 通过步骤 340-42 将第一数据分片中的第一容器更新为第一事务在第一数据分片中的写容器之后, 计算机设备可以释放第一数据分片中的第二容器、以及第一事务在第一数据分片中原有的写容器; 能够及时释放资源, 避免内存过多消耗。

15 在本申请实施例中, 对于第一数据分片上的第一事务, 将第一容器作为该第一事务的新的写容器。由于通过上述步骤 340-40 已将第二容器复制到第二数据分片, 因此, 没有必要在第一数据分片上再保留第二容器, 可以直接释放掉第二容器。

20 在本申请实施例中, 提供了上述步骤 340-50 之前的补充方案, 包括在恢复活跃事务的写操作之前: 响应于第二数据成功复制到第二数据分片, 且第二数据分片上第一事务的事务上下文成功构建, 将第一数据分片中的第一容器更新为第一事务在第一数据分片中的写容器; 释放第一数据分片中的第二容器以及第一事务在第一数据分片中原有的写容器; 本方案能够在分裂成功后, 及时更新第一事务的写容器并释放冗余的容器, 节约第一数据分片的内存空间。

在一些实施例中, 该方法还包括:

25 响应于第二数据成功复制到第二数据分片, 且第二数据分片上第一事务的事务上下文成功构建, 修改第一数据分片的信息;

将第二数据分片移动到对外服务的数据分片列表中。

30 在本申请实施例中, 分裂成功后, 计算机设备可以修改第一数据分片的信息, 具体比如, 根据 split key 修改第一数据分片的 end key; 并将第二数据分片移动到对外服务的数据分片列表中, 以便新建的第二数据分片可以对外服务。

在本申请实施例中, 提供了分裂成功后的补充方案, 包括修改第一数据分片的信息, 将第二数据分片移动到对外服务的数据分片列表中; 以便第二数据分片能够对外提供服务, 保证分布式事务的执行效率。

在一些实施例中, 该方法还包括:

35 响应于第二容器复制失败和/或第二数据分片上第一事务的事务上下文构建失败, 删除第一容器以及第二容器;

恢复第一数据分片中的活跃事务的写操作;

清理第二数据分片。

在本申请实施例中，若上述步骤 340-30 中将第二事务数据写入第一容器，并将第一事务数据写入第二容器的过程失败，或者上述步骤 340-40 将第二容器复制到第二数据分片的过程失败，或者在第二数据分片上构建第一事务的事务上下文的过程失败；计算机设备可以认为分裂失败，此时需要及时删除第一容器以及第二容器，同时恢复第一数据分片中的活跃事务的写操作，并清理第二数据分片。

在本申请实施例中，提供了分裂失败后的补充方案，包括删除第一容器以及第二容器，恢复第一数据分片中的活跃事务的写操作，清理第二数据分片；本方案能够及时清理分裂失败了的相关数据，避免无用数据占用数据库的资源。

10

基于上述图 3 或图 4 所示的方案，请参考图 6，其示出了本申请再一个示例性实施例提供的的数据分片处理方法的流程图。该方法由计算机设备执行，如图 6 所示，上述图 3 或图 4 所示实施例中的步骤 330 可以实现为步骤 330a、步骤 330b、步骤 330c 以及步骤 330d。

步骤 330a：响应于第一数据分片中已进入提交阶段的事务提交完毕，建立第一数据分片的分裂日志。

在本申请实施例中，第一数据分片中已进入提交阶段的事务提交完毕之后，计算机设备可以将第一数据分片的状态由 L_WORKING 更新为 L_SPLIT；等待第一数据分片从副本的分裂日志；建立第一数据分片的分裂日志。

在本申请实施例中，将第一数据分片的从副本的分裂日志同步到第一数据分片的分裂日志，目的是确保所有副本都回放到同一个 split index 之后，再让第二数据分片提供服务。

提交完毕的事务数据，属于非活跃事务的数据，将被写到对应节点的存储实例里。由于同一节点的所有 Region 共享存储实例，因此，新建的第二数据分片也可以访问到第一数据分片的非活跃事务数据，这部分事务数据不需要转移。

而活跃事务的数据没有被写到存储实例中，只能被当前 Region 访问到，所以需要从第一数据分片转移到新建的第二数据分片，这样对应的事务才可以继续执行。

其中，存储实例（RocksDB）是存储集群内部用于存数据的一个模块，包括数据结构 Memtable 和数据结构 SSTable。其中，Memtable 中的数据存储在内存中，没有被持久化。

步骤 330b：将第一数据分片中缓存的数据持久化存储到磁盘中。

在本申请实施例中，步骤 330a 建立第一数据分片的分裂日志之后，计算机设备将第一数据分片中缓存的数据持久化存储到磁盘中。

在本申请实施例中，第一数据分片中缓存的数据包含活跃事务的数据、以及非活跃事务中没有被刷到 SSTable 中的数据。

可选的，磁盘中的数据不需要分裂，因为第一数据分片和新建的第二数据分片在同一个存储实例物理节点上，同一个存储实例物理节点上的所有 Region 会共享磁盘中的数据，因此磁盘中的数据没有分裂的必要。

步骤 330c：对第一数据分片保存快照。

在本申请实施例中，步骤 330b 将第一数据分片中缓存的数据持久化存储到磁盘之后，

计算机设备对第一数据分片保存快照，便于后续的回放处理。

其中，第一数据分片以 split index 保存快照，以便第一数据分片后续不再回放 split index 之前的日志，从而防止第一数据分片从日志中回放出来旧的数据，同时覆盖了第二数据分片所写的新数据。

- 5 以范围是 [A, H) 的 Region 1 为例，在分裂之前，Region 1 上事务提交时所同步的日志可能涉及到 [A, H) 中的任意数据。

假设有个事务 T1 提交了 E = 1 的修改，然后 Region 1 分裂出 Region 2，Region 1 的范围变为 [A, C)，Region 2 的范围为 [C, H)。此时，另外一个事务 T2 在 Region 2 上提交了 E = 2 的修改。

- 10 从上述描述来看，事务 T2 写的 E = 2 应当是最新的版本。

但是，假设允许 Region 1 回放 split index 之前的日志，也就相当于允许了 Region 1 回放事务 T1 所写的日志。由于节点启动时，Region 1 和 Region 2 的日志是并发回放的，一旦 Region 1 上事务 T1 的日志回放在 Region 2 上事务 T2 的日志回放之后，就会导致事务 T1 的 E = 1 后于事务 T2 的 E = 2 写入到 RocksDB 中，进而导致 E = 1 反而成了最新的版本，

- 15 导致数据出错。

因此，需要避免 Region 1 回放 split index 之前的日志，这样事务 T1 的日志就不会被回放了，也就保证了 E = 2 是最新的版本。而达到该目的的途径，就是对 Region 1 以 split index 保存快照。

- 20 在一些实施例中，对第一数据分片保存快照之后，计算机设备可以设置分裂任务状态为 REGION_SPLIT_CREATE_NEW_REGION；并等待从副本进入 REGION_SPLIT_CREATE_NEW_REGION 状态。

步骤 330d：控制第一数据分片中未读写的事务暂停进入读写阶段。

在本申请实施例中，计算机设备暂停第一数据分片对应的新事务进入读写阶段，即阻止基于第一数据分片执行的新事务开启。

- 25 在本申请实施例中，提供了上述步骤 330 的补充方案，包括第一数据分片中已进入提交阶段的事务提交完毕之后，建立第一数据分片的分裂日志，将第一数据分片中缓存的数据持久化存储到磁盘中，对第一数据分片保存快照；能够便于第一数据分片所有副本日志的同步，以及第一数据分片的回放。

在一些实施例中，该方法还包括：

- 30 响应于分裂指令，创建第三数据分片对应的第四数据分片；第三数据分片是第一数据分片的各个从副本中的一个；

获取第一数据分片的分裂日志；

对第三数据分片保存快照。

- 35 在本申请实施例中，响应于对数据库中的第一数据分片的分裂指令，计算机设备创建第三数据分片对应的第四数据分片；获取第一数据分片的分裂日志，并对第三数据分片保存快照。

其中，第三数据分片是第一数据分片的从副本中的任意一个，第三数据分片与第一数

据分片同步响应于分裂指令，创建第三数据分片对应的第四数据分片。

在本申请实施例中，提供了分裂过程中从副本的执行任务方案，包括获取第一数据分片的分裂日志，对第三数据分片保存快照；以便从副本及时备份主副本的相关数据。

5 基于上述实施例，请参考图 7，其示出了本申请一个示例性实施例提供的的数据分片处理方法中暂停第一数据分片对应的新事务进入读写阶段的流程图。该方法由计算机设备执行，如图 7 所示，上述步骤 330 可以实现为步骤 3301：

10 步骤 3301：响应于基于第一数据分片执行的已进入提交阶段的事务提交完毕，且针对第一数据分片的各个从副本分别保存了快照，控制第一数据分片中未读写的事务暂停进入读写阶段。

在本申请实施例中，在上述步骤 330 中，基于第一数据分片执行的已进入提交阶段的事务提交完毕，并且，针对第一数据分片的各个从副本（如第三数据分片）分别保存了快照，计算机设备才可以暂停基于第一数据分片执行的新事务进入读写阶段。

15 在本申请实施例中，提供了分裂过程中从副本的执行任务方案，包括响应于基于第一数据分片执行的已进入提交阶段的事务提交完毕，且第一数据分片的各个从副本分别保存了快照，暂停基于第一数据分片执行的新事务进入读写阶段；以便从副本及时备份主副本的相关数据，并同步更新各个从副本的状态。

在一些实施例中，该方法还包括：

20 响应于第一数据分片和第二数据分片分裂成功，修改第三数据分片的信息；
将第四数据分片移动到对外服务的数据分片列表中。

25 在本申请实施例中，基于上述创建第三数据分片对应的第四数据分片的实施例，若第一数据分片和第二数据分片分裂成功，即第一事务数据成功从第一数据分片转移至第二数据分片，计算机设备可以修改第三数据分片的信息，如根据 split key 修改第三数据分片的 end key；并将第四数据分片移动到对外服务的数据分片列表中，以便新建的第四数据分片能够对外服务。

在本申请实施例中，提供了分裂成功后从副本的执行任务方案，包括修改第三数据分片的信息，将第四数据分片移动到对外服务的数据分片列表中；以便第四数据分片能够对外提供服务，保证分布式事务的执行效率。

在一些实施例中，该方法还包括：

30 响应于第一数据分片和第二数据分片分裂失败，清理第四数据分片。

在本申请实施例中，基于上述创建第三数据分片对应的第四数据分片的实施例，若第一数据分片和第二数据分片分裂失败，即第一事务数据从第一数据分片至第二数据分片转移失败，计算机设备可以清理第四数据分片。

35 在本申请实施例中，提供了分裂失败后从副本的执行任务方案，包括清理第四数据分片；本方案能够及时清理分裂失败的相关数据，避免无用数据占用数据库的资源。

基于上述图 4 所示的方案，请参考图 8，其示出了本申请另外一个示例性实施例提供

的数据分片处理方法的流程图。该方法由计算机设备执行，如图 8 所示，上述图 4 所示实施例中的步骤 340-50 之前，还包括步骤 340-46。

步骤 340-46：将第一事务标记为已分裂的事务，并在第一数据分片的事务上下文中，将第二数据分片的标识添加到第一事务对应的数据分片标识。

5 在本申请实施例中，在步骤 340-50 恢复活跃事务的写操作之前，计算机设备可以将第一事务标记为已分裂的事务，并将第二数据分片的 ID 记录在第一数据分片对应的第一事务的上下文中；避免对应于第二数据分片的事务提交时遗漏新的第二数据分片，保证事务提交的原子性。

该方法还包括：

10 响应于第一数据分片接收到第三事务的预备请求，且第三事务是被标记为已分裂的事务，在第一数据分片的事务上下文中查询第三事务对应的数据分片标识；

响应于查询到第三事务对应的数据分片标识包括第二数据分片的标识，向第三事务的协调者数据分片返回预备响应，预备响应中包含第二数据分片的标识，以便协调者数据分片将第二数据分片的标识添加为第三事务的参与者数据分片标识。

15 在本申请实施例中，当第一数据分片接收到第三事务的预备请求时，且第三事务是被标记为已分裂的事务：计算机设备可以在第一数据分片的事务上下文中查询第三事务对应的数据分片 ID；

当查询到第三事务对应的数据分片 ID 包括第二数据分片的 ID 时：计算机设备可以向第三事务的协调者数据分片返回预备响应。

20 其中，预备响应中包含第二数据分片的 ID，以便协调者数据分片将第二数据分片的 ID 添加为第三事务的参与者数据分片的 ID。

在本申请实施例中，提供了被标记为已分裂的事务的提交方案，包括在恢复活跃事务的写操作之前，将第一事务标记为已分裂的事务，并在第一数据分片的事务上下文中，将第二数据分片的 ID 添加到第一事务对应的数据分片 ID；当第一数据分片接收到已分裂的第三事务的预备请求，在第一数据分片的事务上下文中查询第三事务对应的数据分片 ID；当查询到第三事务对应的数据分片 ID 包括第二数据分片的 ID，向第三事务的协调者数据分片返回预备响应，预备响应中包含第二数据分片的 ID，以便协调者数据分片将第二数据分片的 ID，添加为第三事务的参与者数据分片 ID；本方案能够在已分裂的事务的提交流程中，查找当前待提交的事务涉及的所有数据分片，避免已分裂的数据分片的遗漏，保证事务提交的完整性。

30

本申请实施例的上述技术方案，可以应用于另一种分布式数据库。请参考图 9，其示出了本申请涉及的另一种分布式数据库架构图。如图 9 所示，该分布式数据库包括计算集群（SQLEngine）、存储集群和中央管控集群（MetaCluster，MC）。

35 SQLEngine：计算引擎节点，主要完成结构化查询语言（Structured Query Language，SQL）层计算功能。

存储集群：存储引擎节点，主要完成事务以及键值（Key-Value，KV）存储功能。

MC：集群管理节点，管理整个集群的数据调度，也负责分配全局唯一的时间戳。

其中，基于上述图 9 所示的数据库进行数据分片的分裂过程可以如下：

MC 自动下发数据分片的分裂

当 MC 检测到热点数据，需要迁移热点数据时：

- 5 先向数据分片（Region）所在的存储集群节点下发指示分裂任务（split region job）的分裂指令，job 中带有分裂点（split key）；

存储集群收到 job 后，会将当前数据分片（Old Region）的右边界左移至 split key，同时会创建出一个新的数据分片（New Region），New Region 的左边界为 split key，右边界为 Old Region 原本的右边界。

- 10 这样一来，热点数据就被拆分到了两个数据分片上。

随后 MC 再下发迁移任务（migrate region job），将其中一个 Region 迁移到其他的比较空闲的存储集群节点上，这样就完成了负载均衡。

用户也可以通过向 MC 发送 HTTP 请求，来手动开启或关闭自动分裂功能。关闭后，MC 将不会自行下发 split region job。

- 15 该 HTTP 请求例如如下：

curl-XPUT

http://{mc_ip}:{mc_port}/meta-cluster/api/cluster/enable-split-region/{cluster_id}/{enable}

其中，参数说明如下：

- mc_ip 代表 MC 所在的节点的 IP 地址，mc_port 代表 MC 接收 HTTP 请求的端口号，
20 enable 代表是否开启自动检测 region 状态并下发分裂任务的功能，0 代表关闭，其他正整数值代表开启。

通过如下 HTTP 请求，可以查询自动分裂功能的开关状态：

curl-XGET

http://{mc_ip}:{mc_port}/meta-cluster/api/cluster/is-enable-split-region/{cluster_id}

- 25 手动触发数据分片的分裂

通过向 MC 发送 HTTP 请求，用户也可以手动要求 MC 下发 split region job，控制将某个 Region 按照特定的 split key 进行分裂。

该 HTTP 请求例如如下：

curl-XPUT

- 30 http://{mc_ip}:{mc_port}/meta-cluster/api/region/split-region/{cluster_id}/{region_id}/{is_force}?key="{key}"

其中，API 的作用为手动触发 Region 分裂，因此不受 split-region-enabled 配置参数的影响。

- 其中，可以通过 key 参数传入指定的分裂点，若不指定，则会由存储集群计算出一个
35 分裂点，key 为十六进制编码表示的字符串。

分裂任务主流程（如何保证不杀事务）

一、前置步骤

- 17 -

1. MC 向存储集群要分裂的 Region 的 Leader 副本下发一个计算 split key 的任务, Leader 副本计算出 split key 后, 将其上报给 MC (如果存储集群发现找不到一个合适的分裂点, 则返回失败)。

2. MC 得到存储集群计算的 split key 之后, 向存储集群上该 Region 的所有副本下发 split region job。

该 split region job 的格式例如为:

```
message SplitRegionJob {  
    uint64 job_id = 1;  
    RegionMeta old_region_meta = 2;  
    RegionMeta new_region_meta = 3;  
}
```

其中, job_id 唯一标识一次数据分片任务; RegionMeta 中包含了 Region 的 region_id、start_key、end_key 等信息; old_region_meta 中包含的是 Old Region 分裂前的信息 (start_key 和 end_key 都对应该分裂前的值); new_region_meta 中包含的是 New Region 的信息 (start_key 即 split_key, end_key 应当和 old_region_meta 的 end_key 相同)。

二、Leader 副本收到 MC 下发的分裂任务流程

请参考图 10, 其示出了本申请一个示例性实施例提供的一种数据分片处理方法的实施流程图。请参考图 11, 其示出了本申请一个示例性实施例提供的一种分裂流程图。如图 10、图 11 所示, 分裂任务流程如下。

S1001: 根据 split key 等元信息创建、启动 New Region; 其中, 元信息可以是分裂任务上下文中包含的信息, 或者是分裂任务包含的信息; 分裂任务上下文包含一些任务信息, 比如 Old Region ID、New Region ID、split key 等等。参照图 11 中「S1101: 创建新的数据分片」一步。

S1002: 构建分裂任务上下文, 并设置任务状态为 REGION_SPLIT_INIT, 发送 RPC 确认其它 Old Region 副本都进入该状态。

S1003: 通知事务模块, 阻止所有未进入 2PC 的活跃事务进入提交阶段 (即参与者不准 Prepare)。

S1004: 等待进入 2PC 的事务结束, 处于读写阶段的事务以及新开启的事务不受影响。

S1005: 将当前 Old Region 的状态由 L_WORKING 更新为 L_SPLIT。

S1006: Old Region 同步分裂日志 (region split raft log), 目的是确保所有副本都回放到同一个分裂索引 (split index) 之后, 再让 New Region 提供服务。

S1007: 将 Old Region 残留在内存中的数据持久化存储到磁盘中, 并以 split index 对 Old Region 的 Raft Group 进行保存快照 (save snapshot)。

S1008: 设置分裂任务状态为 REGION_SPLIT_CREATE_NEW_REGION。

S1009: 发送 RPC 确认其它副本都进入了步骤 S1008 的上述状态。

S1010: 阻止新事务开启, 并遍历所有活跃事务, 对每个事务进行如下操作:

a) 设置标记, 不允许 put/delete 等写操作。如果后续收到 SQL Engine 下发的写请求,

则返回重试错误码；

b) 新建 `old_region_writebatch` 和 `new_region_writebatch`，并遍历事务 `writebatch` 中未提交的数据，将其按分裂点分别写入前面新建的两个 `writebatch` 中；

- 5 c) 将 `new_region_writebatch` 的数据通过 RPC 发送到 New Region 的 Leader 上面，并在 New Region 的 Leader 上构建出事务上下文；参照图 11 中「S1102：根据分裂点，向新的数据分片传输部分的事务数据」一步，通过 RPC 将事务 A (Trans A) 的部分数据传输至新的数据分片。

S1011：上报 MC 本节点分裂任务完成。MC 综合所有节点上报的任务状态后，回复各个节点任务的状态，以决定继续还是回滚清理环境。

- 10 S1012-1：如果 MC 决定继续执行，分裂数据分片生效，并遍历所有事务分裂生效并继续提供服务：

a) 本节点收到信息后，则根据 `split key` 等元信息修改 Old Region 的 `end key` 信息，将 New Region 移动到对外服务的数据分片列表中；

b) 再次遍历所有活跃事务，并对每个事务进行如下操作：

- 15 i. 将 `old_region_writebatch` 更新成 `writebatch` 并生效，并释放掉 `new_region_writebatch` 和旧的 `writebatch`。参照图 11 中「S1103：若分裂成功，删除新的数据分片上分裂点后的数据」一步；

ii. 设置本事务为分裂过的事务，并记录好分裂的标识 (New Region ID)，这一步是为了保证事务提交的原子性；

- 20 iii. 允许事务进行 `put/delete` 等写操作，允许事务进入提交状态。

S1012-2：遍历完所有事务后，允许新事物开启，并修改 Region 状态为 `L_WORKING`，完成分裂，结束流程。

S1013-1：如果 MC 决定回滚，清理分裂事务，并遍历所有事务使得分裂不生效，并继续对外提供服务：

- 25 本节点收到信息后，清理掉 New Region，并遍历所有活跃事务，并对每个事务进行如下操作：

i. 释放掉新创建的 `old_region_writebatch` 及 `new_region_writebatch`，参照下图 11 中「S1104：若分裂失败，直接删除新的数据分片全部数据」一步；

ii. 允许事务进行 `put/delete` 等写操作，允许事务进入提交状态。

- 30 S1013-2：遍历完所有事务后，允许新事务开启，并修改 region 状态为 `L_WORKING`，完成分裂回滚，结束流程。

上述步骤 S1001、S1006、S1007 中，如果发生异常，则设置分裂任务状态为 `REGION_SPLIT_FAILED`，并进入上报 MC 异常流程。

- 35 上述步骤 S1010 中如果发现其它副本异常，直接进入上报 MC 异常流程。由于 Leader 副本后面对每个事务遍历的操作代价较大，要尽量避免无效执行。

三、Follow 收到 MC 下发的分裂任务流程

如图 10 所示，分裂任务流程如下：

S101: 根据 split key 等元信息创建、启动 New Region。

S102: 构建分裂任务上下文, 并设置任务状态为 REGION_SPLIT_INIT, 确认其它 Old Region 副本都进入该状态;

S103: 将当前 Old Region 的状态由 F_WORKING 更新为 F_SPLIT;

5 S104: 等待 Apply Leader 的分裂日志;

S105: 对当前 Old Region 进行保存快照;

S106: 设置分裂任务状态为 REGION_SPLIT_NEW_REGION;

S107: 上报 MC 本节点分裂任务完成。MC 综合所有节点上报的任务状态后, 回复各个节点任务的状态, 以决定继续还是回滚清理环境;

10 S108: 如果 MC 决定继续执行, 本节点收到信息后, 则根据 split key 等元信息修改 Old Region 的 end key 信息, 将 New Region 移动到对外服务的数据分片列表中, 并修改 Region 状态为 F_WORKING, 完成分裂, 结束流程;

S109: 如果 MC 决定回滚, 本节点收到信息后, 清理掉 New Region, 并修改 Region 状态为 F_WORKING, 完成分裂回滚, 结束流程。

15 综上, 整个流程从 Region 角度来看, 事务的生命周期可以跨越分裂, 但是 Region 分裂动作和事务的提交在时间上是互斥的。因此, Region 需要阻止读写事务进入提交状态, 并等待已进入提交状态的事务完成提交后, 才能进入分裂动作; 且在分裂动作结束后, 才允许读写事务继续进入提交状态。

存在这个限制的原因是, 目前存储集群在重启 (recover) 的时候, 多 Region 会同时并发回放, 这里的数据正确性依赖分裂时保存的 raft 快照元 (snapshot meta), 这个 snapshot meta 对应的 snapshot index 即为 split index。由于 Region 启动后, 会从 snapshot index 的位置开始回放日志, 如果有事务的日志跨越 split index (例如在 split index 前同步了 prepare raft log, 在 split index 后同步了 commit raft log), 那么 Region 启动后, 就会遗漏掉事务的一部分日志无法回放, 导致事务数据丢失。所以同步 split region log 的时候, 必须保证提交过程中的事务都结束了。

20 而针对上述「分布式数据库架构选型」中提到的传统分布式数据库分裂数据分片时 QPS 下滑的问题, 本申请实施例的技术方案对应的解决方法是, 在 Old Region 上事务禁写之前, 就先创建出了 New Region 的所有副本, 保证 New Region 选举出 Leader 副本之后, 再将 Old Region 的事务数据拆分到 New Region 上, 这样可以尽可能降低禁写的时间, 做到对业务无感知。

事务提交完整性的实现方式如下:

35 对于一个分布式事务, 在存储集群的多个 Region 上都会有事务上下文。为了保证多个 Region 上提交事务的原子性 (要么全提交, 要么全回滚), SQL Engine 在提交事务的时候, 会向其中某个 Region 下发提交请求 (commit request), request 中带有 SQL Engine 访问过的 Region 列表。收到 commit request 的 Region 担任协调者的职务, 通过 2PC 保证事务提交的原子性。

请参考图 12，其示出了本申请涉及的一种 2PC 协议的流程图。如图 12 所示，2PC 协议由协调者和参与者共同完成。

协调者进入准备（perpare）阶段，向每个参与者发送准备请求；

5 参与者接收准备请求后，执行提交响应或者不提交响应，提交响应包括重新写入以及写入准备后进入准备阶段；

协调者收集每个参与者的响应，判断 perpare 是否失败：

若失败，则中止 perpare；将中止反馈发送至所有参与者后，所有参与者写入中止；

若成功，则进入提交（commit）阶段，向每个参与者发送提交请求；

每个参与者接收提交请求后，执行响应，包括写入提交后进入提交阶段；

10 协调者收集每个参与者的提交反馈，向每个参与者发送确认请求；

参与者接收确认请求后，执行响应，包括写入确认后进入确认阶段，并清除事务上下文；

协调者收集每个参与者的确认反馈后，清除事务上下文。

如果事务提交之前，某个 Region A 分裂出了 Region B，然而 SQL Engine 没有感知到
15 Region B 的存在，这个时候 commit request 中就不会带有 Region B，Region B 上的数据也就不会在 2PC 流程中被提交，这样就破坏了事务的原子性要求。

在上述「Leader 副本收到 MC 下发的分裂任务流程」中，S1012-1 要求在 Old Region 上所有活跃事务上下文中「设置本事务为分裂过的事务，并记录好分裂的 New Region ID」。

协调者下发的 2PC Prepare 请求中，会带有全量的参与者列表。如果 Old Region 发现 2PC
20 Prepare 请求中没有包含这个新分裂出的 New Region ID，就会返回错误码，要求协调者补充参与者列表之后，再重新下发 2PC Prepare 请求。

请参考图 13，其示出了本申请一个示例性实施例提供的一种 2PC 案例示意图。如图 13 所示，给出了一种案例，事务 T 在读写阶段写了 A、B、D、G 四个 Key，其中 A、B 在 Region 1 上面，D、G 在 Region 2 上面，这样在事务 T 的 Region Cache 中会包含 Region 1
25 和 Region 2。假设此时事务 T 停止读写了一会，然后直接执行了 commit，巧合的是就在事务 T 停止读写的这个空隙中，Region 2 发生了分裂，分裂产生了新的 Region 3，注意此时 Region 2 上关于事务 T 的子事务空间被标记上了 is_split=true，new_region=3。分裂后，事务 T 才发起 commit 请求，并指定 Region1 成为协调者。

请参考图 14，其示出了图 13 的 2PC 案例的提交示意图。如图 14 所示，当 Region 1 收到
30 SQL Engine 发过来的事务提交请求（其中 region_list 为 1、2）后，处理流程如下。

S1：协调者收到事务提交请求，知道事务的参与者有 Region 1 和 Region 2，于是向 Region 1 和 Region 2 发送 prepare 请求。prepare 请求中会带上协调者已知的所有参与者的列表 Region 1 和 Region 2。

S2：Region 1 收到 prepare 请求后，查看事务 T 在本 Region 的子事务空间中的 is_split
35 变量，该变量值为 false，说明在子事务的生命周期内 Region 没有发生过分裂，所以直接执行 prepare（写日志同步到其它副本），并在成功后，应答协调者准备完毕（prepare ok）。

S3：Region 2 收到 prepare 请求后，查看事务 T 在本 Region 的子事务空间中的 is_split

-21-

变量，该变量值为 true，说明在子事务的生命周期内 Region 发生过分裂，进一步查看分裂后的 New Region 是否在 prepare 请求的成员列表中，new_region 的值为 3，并不在成员列表中，于是返回协调者为未准备完毕（prepare not ok），且把 New Region 的值告诉协调者。其中，需要设定对应的错误码，让协调者感知并特殊处理这种错误。具体比如，错误码是

5 EC_TDS_TRANS_PREPARE_NEED_MORE_PART，即 transaction prepare need more participants，表示的含义是需要添加新的参与者到参与者列表中。当协调者收到这个错误码时，会从响应（response）中读取 more_participants 字段，其中包含的信息，即 New Region 的 ID 为 3。

10 S4：协调者收到 Region 2 的回复之后，发现对应的错误码为 Region 发生了分裂，则从请求中获取 New Region 的 ID 为 3，并将 Region 3 加入到参与者列表中，并重启一轮新的 prepare 阶段：同时向 Region 1、Region 2 和 Region 3 发送 prepare 请求。prepare 请求中会带上协调者已知的所有参与者的列表 Region 1、Region 2、Region 3。

15 S5：Region 1 再次收到 prepare 请求之后，发现事务 T 在本 Region 的子事务空间的状态已经是 prepare 状态，对比请求中的参与者个数和自己记录的参与者个数，发现有新增长，于是将新增的 Region 3，记录下来并提交日志，成功后再次应答协调者 prepare ok。

S6：Region 2 再次收到 prepare 请求之后，查看事务 T 在本 Region 的子事务空间中的 is_split 变量，该变量值为 true，说明在子事务的生命周期内 Region 发生过分裂，进一步查看分裂后的 New Region 是否在 prepare 请求的成员列表中，new_region 的值为 3，在成员列表中，执行 prepare（写日志同步到其它副本），并在成功后应答协调者 prepare ok。

20 S7：Region 3 收到 prepare 请求后，查看事务 T 在本 Region 的子事务空间中的 is_split 变量，该变量值为 false，说明在子事务的生命周期内 Region 没有发生过分裂，所以直接执行 prepare（写日志同步到其它副本），并在成功后应答协调者 prepare ok。

S8：协调者收到 Region 1、Region 2 以及 Region 3 的 prepare ok 回复后，继续推动 2PC 事务向 commit 状态前进。

25 从上述过程中，可以看出当协调者的参与者列表中的 Region 发生了分裂，且分裂的 New Region 不在参与者列表中时，会多一轮补齐参与者的请求。

如果协调者补齐参与者列表后，New Region 又发生了分裂，出现了新的 Region，是否需要多一轮补齐参与者的请求，从流程设计上是需要，所以可以在这里要限制住实际上发生这种情况的可能性。

30 一个可行的方案是，严格限制 New Region 从产生到分裂之间的最小保护时间，因为事务进入 2PC 后通常很快就会结束，这样当最小保护时间远远大于一般事务进入 2PC 之后的提交时间时，就几乎不会遇到这种需要进行两轮以上参与者补齐的情况了。

上述最小保护时间，指的是当分裂产生的 New Region A 被创建出来后，至少经过这个最小保护时间后，这个 New Region A 才允许处理分裂任务，分裂出 New Region B。其中，

35 这个时间范围在秒级别即可，例如 3s，只需要保证远大于 2PC 的提交所需时间即可（在负载压力不大的情况下，一般 2PC 的提交时间，都在 1s 以内）。

设置上述最小保护时间的目的是，防止协调者频繁收到

EC_TDS_TRANS_PREPARE_NEED_MORE_PART 错误码。具体比如：

当 Region A 分裂出 Region B 后，Region A 收到了 prepare request，这个时候会设置该错误码，将 Region B 放到 more_participants 字段中，告知协调者 Region B 也需要作为参与者中的一员并提交。结果协调者向 Region B 发送 prepare request 前，Region B 又分裂出了 Region C，导致 Region B 也要设置该错误码，将 Region C 放到 more_participants 字段里，这就导致在扩充参与者列表上花费了过多轮次的通信。

最小保护时间能够让 Region B 等待一段时间，再处理分裂请求。因此，当 Region B 分裂出 Region C 的时候，事务的 2PC 流程已经结束，可以降低对事务提交的影响了，让事务可以快速走完 2PC 流程。

另外，在上面例子中的 Region 1 在这个过程中是不会发生分裂，原因是事务已经成功在这个 Region 上进入了 prepare 状态，这样就会阻止 Region 1 分裂了。

请参考图 15，其示出了本申请一个示例性实施例提供的一种 2PC 流程图。

如图 15 所示，2PC 准确请求的事务参与者清单中仅包含数据分片 1，但数据分片 1 分裂出了数据分片 2，则返回错误，告知协调者缺少遗漏的参与者，但不代表事务提交失败；

协调者更新参与者列表，重发准备请求，事务参与者清单中包含数据分片 1 和数据分片 2。

综上，本申请实施例提供一种对事务无感知的分裂数据分片的方法，该方法能够在支持分布式事务的系统中，快速对数据分片做拆分，分散热点数据。这种数据分裂的方法在保证事务提交的完整性的同时，具有较快的分裂速度，可以保证对业务的写入没有任何影响，不会造成性能波动以及写失败，能够帮助分布式存储系统在用户没有感知的情况下实现丝滑的负载均衡。

本申请实施例提供了一种做到对数据写入无感知的数据分片分裂方法，这种方法被应用于支持事务功能、或者批量原子写入功能的存储系统中，可以保证数据写入和数据分片分裂并发的正确性，事务提交的完整性也可以保证。

这种方法使得阻塞事务的时间更短，不会造成 QPS 的下降，更不会造成事务回滚，可以保证数据分片分裂，对用户的写入是无感知的。

这种方法不需要运维人员手动调整分区大小，管控节点会自动结合各个分区大小，评估是否应该下发分裂任务，自动做到负载均衡。

另外，本申请实施例提到的数据分片分裂策略，不局限应用于分布式数据库系统，其底层存储对应的基本的数据单位也不限于 KV 数据。例如文件存储、基于 B-Tree 的表数据存储等都可以使用类似的方法实现数据分片分裂，只要系统有办法可以将数据唯一映射到对应的数据分片即可。

本申请实施例中提到的“同步 raft log”仅作为基于 Raft Group 做主备数据同步的一种手段。对于分布式一致性 (Paxos) 协议实现主备同步的存储系统，可以替换成“同步 Paxos 决议”；对于单机数据库，可以替换成“落盘一条预写式日志 (Write Ahead Log, WAL) 日志”。

本申请实施例使用了“基于关键字区间划分”的数据划分方式，通过指定一个 split key 来将单个 region 拆分成两个。但是对于使用“基于关键字哈希值划分”技术的存储系统，也可

以指定一个 split hash value，对数据分片做拆分。不管怎样的数据划分方式，只要能将其一分为二，就可以使用本方案的做法。

图 16 其示出了本申请一个示例性实施例示出的数据分片处理装置的方框图，该装置可以用于执行如图 3、图 4、图 5、图 6、图 7 或图 8 所示方法中，由计算机设备执行的全部或部分步骤如图 16 所示，该装置包括：

创建模块 1601，用于响应于对数据库中的第一数据分片的分裂指令，创建第二数据分片；分裂指令用于指示将第一数据分片中的数据分裂到第二数据分片；

第一暂停模块 1602，用于控制第一数据分片中未提交的事务暂停进入提交阶段；

第二暂停模块 1603，用于响应于第一数据分片中已进入提交阶段的事务提交完毕，控制第一数据分片中未读写的事务暂停进入读写阶段；

转移模块 1604，用于将第一事务数据从第一数据分片转移到第二数据分片；第一事务数据是第一事务的数据中对应于分裂后的第二数据分片的数据；第一事务是第一数据分片的活跃事务中需转移数据至第二数据分片的事务，活跃事务为处于读写阶段的事务；

恢复模块 1605，用于响应于第一事务数据转移完成，控制第一数据分片中的事务恢复进入读写阶段和/或提交阶段。

在一些实施例中，转移模块 1604，用于，

暂停活跃事务的写操作；

在第一数据分片中建立第一事务的第一容器和第一事务的第二容器；

将第二事务数据写入第一容器，并将第一事务数据写入第二容器；第二事务数据是第一事务的数据中对应于分裂后的第一数据分片的数据；

将第二容器复制到第二数据分片，并在第二数据分片上构建第一事务的事务上下文；

响应于第二数据成功复制到第二数据分片，且第二数据分片上第一事务的事务上下文成功构建，恢复活跃事务的写操作。

在一些实施例中，装置还包括：更新模块，用于，

响应于第二数据成功复制到第二数据分片，且第二数据分片上第一事务的事务上下文成功构建，将第一数据分片中的第一容器更新为第一事务在第一数据分片中的写容器；

释放第一数据分片中的第二容器、以及第一事务在第一数据分片中原有的写容器。

在一些实施例中，装置还包括：修改模块，用于，

响应于第二数据成功复制到第二数据分片，且第二数据分片上第一事务的事务上下文成功构建，修改第一数据分片的信息；

将第二数据分片移动到对外服务的数据分片列表中。

在一些实施例中，装置还包括：删除模块，用于，

响应于第二容器复制失败和/或第二数据分片上第一事务的事务上下文构建失败，删除第一容器以及第二容器；

恢复活跃事务的写操作；

清理第二数据分片。

在一些实施例中，装置还包括：建立模块，用于，
响应于所述第一数据分片中已进入所述提交阶段的事务提交完毕，建立第一数据分片的分裂日志；

将第一数据分片中缓存的数据持久化存储到磁盘中；

5 对第一数据分片保存快照。

在一些实施例中，装置还包括：获取模块，用于，

响应于分裂指令，创建第三数据分片对应的第四数据分片；第三数据分片是第一数据分片的各个从副本中的一个；

获取第一数据分片的分裂日志；

10 对第三数据分片保存快照。

在一些实施例中，第二暂停模块 1603，用于，

响应于基于第一数据分片执行的已进入提交阶段的事务提交完毕，且针对第一数据分片的各个从副本分别保存了快照，控制第一数据分片中未读写的事务暂停进入读写阶段。

在一些实施例中，装置还包括：移动模块，用于，

15 响应于第一数据分片和第二数据分片分裂成功，修改第三数据分片的信息；

将第四数据分片移动到对外服务的数据分片列表中。

在一些实施例中，装置还包括：清理模块，用于，

响应于第一数据分片和第二数据分片分裂失败，清理第四数据分片。

在一些实施例中，装置还包括：添加模块，用于，

20 将第一事务标记为已分裂的事务，并在第一数据分片的事务上下文中，将第二数据分片的标识，添加到第一事务对应的数据分片标识；

装置还包括：查询模块，用于，

响应于第一数据分片接收到第三事务的预备请求，且第三事务是被标记为已分裂的事务，在第一数据分片的事务上下文中查询第三事务对应的数据分片标识；

25 响应于查询到第三事务对应的数据分片标识包括第二数据分片的标识，向第三事务的协调者数据分片返回预备响应，预备响应中包含第二数据分片的标识，以便协调者数据分片将第二数据分片的标识，添加为第三事务的参与者数据分片标识。

图 17 示出了本申请一个示例性实施例示出的计算机设备 1700 的结构框图。该计算机设备可以实现为本申请上述方案中的服务器。该计算机设备 1700 包括中央处理单元(Central Processing Unit, CPU) 1701、包括随机存取存储器(Random Access Memory, RAM) 1702 和只读存储器(Read-Only Memory, ROM) 1703 的系统存储器 1704，以及连接系统存储器 1704 和中央处理单元 1701 的系统总线 1705。该计算机设备 1700 还包括用于存储操作系统 1709、应用程序 1710 和其他程序模块 1711 的大容量存储设备 1706。

30 该大容量存储设备 1706 通过连接到系统总线 1705 的大容量存储控制器(未示出)连接到中央处理单元 1701。该大容量存储设备 1706 及其相关联的计算机可读介质为计算机设备 1700 提供非易失性存储。也就是说，该大容量存储设备 1706 可以包括诸如硬盘或者只读光盘(Compact Disc Read-Only Memory, CD-ROM)驱动器之类的计算机可读介质(未

35

示出)。

不失一般性，该计算机可读介质可以包括计算机存储介质和通信介质。计算机存储介质包括以用于存储诸如计算机可读指令、数据结构、程序模块或其他数据等信息的任何方法或技术实现的易失性和非易失性、可移动和不可移动介质。计算机存储介质包括RAM、ROM、可擦除可编程只读存储器(Erasable Programmable Read Only Memory, EPROM)、电子抹除式可复写只读存储器(Electrically-Erasable Programmable Read-Only Memory, EEPROM)闪存或其他固态存储其技术，CD-ROM、数字多功能光盘(Digital Versatile Disc, DVD)或其他光学存储、磁带盒、磁带、磁盘存储或其他磁性存储设备。当然，本领域技术人员可知该计算机存储介质不局限于上述几种。上述的系统存储器1704和大容量存储设备1706可以统称为存储器。

根据本公开的各种实施例，该计算机设备1700还可以通过诸如因特网等网络连接到网络上的远程计算机运行。也即计算机设备1700可以通过连接在该系统总线1705上的网络接口单元1707连接到网络1708，或者说，也可以使用网络接口单元1707来连接到其他类型的网络或远程计算机系统(未示出)。

该存储器还包括至少一条计算机程序，该至少一条计算机程序存储于存储器中，中央处理单元1701通过执行该至少一条计算机程序来实现上述各个实施例所示的方法中的全部或者部分步骤。

在示例性实施例中，还提供了一种芯片，芯片包括可编程逻辑电路和/或程序指令，当芯片在计算机设备上运行时，用于实现上述方面的数据分片处理方法。

在示例性实施例中，还提供了一种计算机程序产品，该计算机程序产品包括计算机指令，该计算机指令存储在计算机可读存储介质中。计算机设备的处理器从计算机可读存储介质读取该计算机指令，处理器从计算机可读存储介质读取并执行该计算机指令，以实现上述各方法实施例提供的数据分片处理方法。

在示例性实施例中，还提供了一种计算机可读存储介质，该计算机可读存储介质中存储有计算机程序，计算机程序由处理器加载并执行以实现上述各方法实施例提供的数据分片处理方法。

本领域普通技术人员可以理解实现上述实施例的全部或部分步骤可以通过硬件来完成，也可以通过程序来指令相关的硬件完成，所述的程序可以存储于一种计算机可读存储介质中，上述提到的存储介质可以是只读存储器，磁盘或光盘等。

本领域技术人员应该可以意识到，在上述一个或多个示例中，本申请实施例所描述的功能可以用硬件、软件、固件或它们的任意组合来实现。当使用软件实现时，可以将这些功能存储在计算机可读介质中或者作为计算机可读介质上的一个或多个指令或代码进行传输。计算机可读介质包括计算机存储介质和通信介质，其中通信介质包括便于从一个地方向另一个地方传送计算机程序的任何介质。存储介质可以是通用或专用计算机能够存取的任何可用介质。

以上所述仅为本申请的可选实施例，并不用以限制本申请，凡在本申请的精神和原则之内，所作的任何修改、等同替换、改进等，均应包含在本申请的保护范围之内。

权利要求

- 1.一种数据分片处理方法，由计算机设备执行，所述方法包括：
- 响应于对数据库中的第一数据分片的分裂指令，创建第二数据分片；所述分裂指令用于指示将所述第一数据分片中的数据分裂到所述第二数据分片；
- 5 控制所述第一数据分片中未提交的事务暂停进入提交阶段；
- 响应于所述第一数据分片中已进入所述提交阶段的事务提交完毕，控制所述第一数据分片中未读写的事务暂停进入读写阶段；
- 将第一事务数据从所述第一数据分片转移到所述第二数据分片；所述第一事务数据是第一事务的数据中对应于分裂后的所述第二数据分片的数据；所述第一事务是所述第一数据分片的活跃事务中需转移数据至所述第二数据分片的事务，所述活跃事务为处于所述读写阶段的事务；
- 10 响应于所述第一事务数据转移完成，控制所述第一数据分片中的事务恢复进入所述读写阶段和/或所述提交阶段。
- 2.根据权利要求 1 所述的方法，所述将第一事务数据从所述第一数据分片转移到所述第二数据分片，包括：
- 15 暂停所述活跃事务的写操作；
- 在所述第一数据分片中建立所述第一事务的第一容器和所述第一事务的第二容器；
- 将第二事务数据写入所述第一容器，并将所述第一事务数据写入所述第二容器；所述第二事务数据是所述第一事务的数据中对应于分裂后的所述第一数据分片的数据；
- 20 将所述第二容器复制到所述第二数据分片，并在所述第二数据分片上构建第一事务的事务上下文；
- 响应于所述第二数据成功复制到所述第二数据分片，且所述第二数据分片上所述第一事务的事务上下文成功构建，恢复所述活跃事务的写操作。
- 3.根据权利要求 2 所述的方法，其特征在于，在所述恢复所述活跃事务的写操作之前，
- 25 还包括：
- 响应于第二数据成功复制到所述第二数据分片，且所述第二数据分片上所述第一事务的事务上下文成功构建，将所述第一数据分片中的所述第一容器更新为所述第一事务在所述第一数据分片中的写容器；
- 释放所述第一数据分片中的所述第二容器、以及所述第一事务在所述第一数据分片中
- 30 原有的写容器。
- 4.根据权利要求 2 或 3 所述的方法，所述方法还包括：
- 响应于第二数据成功复制到所述第二数据分片，且所述第二数据分片上所述第一事务的事务上下文成功构建，修改所述第一数据分片的信息；
- 将所述第二数据分片移动到对外服务的数据分片列表中。
- 35 5.根据权利要求 2 至 4 任一项所述的方法，所述方法还包括：
- 响应于所述第二容器复制失败和/或所述第二数据分片上所述第一事务的事务上下文构建失败，删除所述第一容器以及所述第二容器；

-27-

恢复所述活跃事务的写操作；

清理所述第二数据分片。

6.根据权利要求1至5任一所述的方法，所述响应于所述第一数据分片中已进入所述提交阶段的事务提交完毕，控制所述第一数据分片中未读写的事务暂停进入读写阶段，包

5 括：

响应于所述第一数据分片中已进入所述提交阶段的事务提交完毕，建立所述第一数据分片的分裂日志；

将所述第一数据分片中缓存的数据持久化存储到磁盘中；

对所述第一数据分片保存快照；

10 控制所述第一数据分片中未读写的事务暂停进入所述读写阶段。

7.根据权利要求1至6任一项所述的方法，所述方法还包括：

响应于所述分裂指令，创建第三数据分片对应的第四数据分片；所述第三数据分片是所述第一数据分片的各个从副本中的一个；

获取所述第一数据分片的分裂日志；

15 对所述第三数据分片保存快照。

8.根据权利要求1至7任一项所述的方法，所述响应于所述第一数据分片中已进入所述提交阶段的事务提交完毕，控制所述第一数据分片中未读写的事务暂停进入读写阶段，包括：

20 响应于基于所述第一数据分片执行的已进入所述提交阶段的事务提交完毕，且针对所述第一数据分片的各个从副本分别保存了快照，控制所述第一数据分片中未读写的事务暂停进入所述读写阶段。

9.根据权利要求7或8所述的方法，所述方法还包括：

响应于所述第一数据分片和所述第二数据分片分裂成功，修改所述第三数据分片的信息；

25 将所述第四数据分片移动到对外服务的数据分片列表中。

10.根据权利要求7至9任一项所述的方法，所述方法还包括：

响应于所述第一数据分片和所述第二数据分片分裂失败，清理所述第四数据分片。

11.根据权利要求2至10任一项所述的方法，在所述响应于所述第二数据分片上所述第一事务的事务上下文的成功构建，恢复所述活跃事务的写操作之前，还包括：

30 将所述第一事务标记为已分裂的事务，并在所述第一数据分片的事务上下文中，将所述第二数据分片的标识添加到所述第一事务对应的数据分片标识；

所述方法还包括：

响应于所述第一数据分片接收到第三事务的预备请求，且所述第三事务是被标记为已分裂的事务，在所述第一数据分片的事务上下文中查询所述第三事务对应的数据分片标识；

35 响应于查询到所述第三事务对应的数据分片标识包括所述第二数据分片的标识，向所述第三事务的协调者数据分片返回预备响应，所述预备响应中包含所述第二数据分片的标识，以便所述协调者数据分片将所述第二数据分片的标识添加为所述第三事务的参与者数

据分片标识。

12.一种数据分片处理装置，所述装置包括：

创建模块，用于响应于对数据库中的第一数据分片的分裂指令，创建第二数据分片；
所述分裂指令用于指示将所述第一数据分片中的数据分裂到所述第二数据分片；

5 第一暂停模块，用于控制所述第一数据分片中未提交的事务暂停进入提交阶段；

第二暂停模块，用于响应于所述第一数据分片中已进入所述提交阶段的事务提交完毕，
控制所述第一数据分片中未读写的事务暂停进入读写阶段；

转移模块，用于将第一事务数据从所述第一数据分片转移到所述第二数据分片；所述
10 第一事务数据是第一事务的数据中对应于分裂后的所述第二数据分片的数据；所述第一事
务是所述第一数据分片的活跃事务中需转移数据至所述第二数据分片的事务，所述活跃事
务为处于所述读写阶段的事务；

恢复模块，用于响应于所述第一事务数据转移完成，控制所述第一数据分片中的事务
恢复进入所述读写阶段和/或所述提交阶段。

13.一种计算机设备，所述计算机设备包括处理器和存储器，所述存储器存储有至少一
15 条计算机指令，所述至少一条计算机指令由所述处理器加载并执行以实现如权利要求 1 至
11 任一所述的数据分片处理方法。

14.一种计算机可读存储介质，所述计算机可读存储介质中存储有至少一条计算机指令，
所述计算机指令由处理器加载并执行以实现如权利要求 1 至 11 任一所述的数据分片处理
方法。

20 15.一种计算机程序产品，所述计算机程序产品包括计算机指令，所述计算机指令存储
在计算机可读存储介质中；所述计算机指令由计算机设备的处理器读取并执行，以实现如
权利要求 1 至 11 任一所述的用于数据分片处理方法。