

数据库线程池控制方法和装置、电子设备、计算机介质

相关申请的交叉引用

本专利申请要求于 2023 年 7 月 28 日提交的、申请号为 202310942845.1、发明名称为“数据库线程池控制方法和装置、电子设备、计算机介质”的中国专利申请的优先权，该申请的全文以引用的方式并入本申请中。

技术领域

本公开涉及计算机技术领域，具体涉及数据库线程池控制方法和装置、电子设备以及计算机可读介质。

背景技术

MySQL 数据库默认的并发连接模型采用“一线程一连接”，即为每个连接创建一个线程，这种数据库的一大缺陷，就是在高并发场景下，频繁创建/销毁线程会导致数据库卡死或崩溃。此外，即使使用长连接模式，大量的线程也会极大增加任务调度负担。

发明内容

本公开的实施例提出了数据库线程池控制方法和装置、电子设备、计算机可读介质。

第一方面，本公开的实施例提供了一种数据库线程池控制方法，该方法包括：在数据库的操作系统中加载能够将线程替换为协程的动态运行库，生成至少一个协程调度线程，协程调度线程用于调度至少一个协程；接收客户端向数据库发送的连接请求；检测数据库创建新线程的请求是否为处理连接请求的请求；响应于检测到数据库创建新线程的请求为处理连接请求的请求，创建与连接请求对应的协程，并通过至少一个协程调度线程调度协程完成连接请求对应的访问任务。

在一些实施例中，上述在数据库的操作系统中加载能够将线程替换为协程的动态运行库，生成至少一个协程调度线程包括：响应于数据库无法修改源代码且不具有连接管理插件机制时，启动数据库；加载并初始化动态运行库，生成至少一个协程调度线程，动态运行库具有创建线程的钩子函数，创建线程的钩子函数

用于采用协程替换数据库的原生运行库生成的连接处理线程；在数据库初始化时，重载创建线程的钩子函数，以通过创建线程的钩子函数判断当前创建的线程是否为连接处理线程，在当前创建的线程是连接处理线程时，将该连接处理线程转换为协程。

在一些实施例中，上述在数据库的操作系统中加载能够将线程替换为协程的动态运行库，生成至少一个协程调度线程包括：响应于数据库具有连接管理插件机制时，启动数据库；加载并初始化动态运行库，生成至少一个协程调度线程；初始化数据库；加载并初始化插件，以使插件接收向数据库发送的连接请求，并数据库的线程构建请求为用于处理连接请求的请求时，控制动态运行库中的创建线程的钩子函数工作，创建线程的钩子函数用于判断当前创建的线程是否为连接处理线程，在当前创建的线程是连接处理线程时，将该连接处理线程转换为协程。

在一些实施例中，上述方法还包括：将至少一个协程调度线程划分为 IO 密集型调度线程和计算密集型调度线程；上述通过至少一个协程调度线程调度协程完成连接请求对应的访问任务包括：检测与连接请求对应的协程是否处于第一状态与第二状态之间的状态，其中，第一状态为协程处于等待执行数据库命令的数据包，数据包已到达，且该协程需要被唤醒时的状态；第二状态为协程位于计算密集型调度线程，且该协程发起对特定文件的特定写入请求的状态；若检测到与连接请求对应的协程处于第一状态与第二状态之间的状态，采用至少一个协程调度线程中的计算密集型调度线程调度与连接请求对应的协程；若检测到与连接请求对应的协程未处于第一状态与第二状态之间的状态，采用至少一个协程调度线程中的 IO 密集型调度线程调度与连接请求对应的协程。

在一些实施例中，上述若检测到与连接请求对应的协程未处于第一状态与第二状态之间的状态，采用至少一个协程调度线程中的 IO 密集型调度线程调度与连接请求对应的协程包括：响应于与连接请求对应的协程处于创建连接的阶段，将协程放置在当前的 IO 密集型调度线程，当前的 IO 密集型调度线程基于负载均衡算法确定；响应于与连接请求对应的协程处于对特定文件的特定写入操作，且完成写特定文件 IO 请求的提交以及让出协程进入 IO 等待之前，将协程迁移至当前的 IO 密集型调度线程。

在一些实施例中，上述数据库为 MySQL 数据库，上述响应于与连接请求对应的协程处于对特定文件的特定写入操作，且完成写特定文件 IO 请求的提交以

及让出协程进入 IO 等待之前，将协程迁移至当前的 IO 密集型调度线程包括：响应于与连接请求对应的协程处于对二进制日志文件的写操作，且完成写文件 IO 请求的提交以及让出协程进入 IO 等待之前，将协程迁移至当前的 IO 密集型调度线程。

在一些实施例中，上述方法还包括：检测 IO 密集型调度线程中是否具有驱动批量提交的协程；若 IO 密集型调度线程中具有驱动批量提交的协程，将被该驱动批量提交协程唤醒的协程迁移至 IO 密集型调度线程，该驱动批量提交协程唤醒的协程迁移入的 IO 密集型调度线程基于负载均衡算法确定。

在一些实施例中，上述数据库为 MySQL 且开启组提交的数据库，上述检测 IO 密集型调度线程中是否具有驱动批量提交的协程包括：检测 IO 密集型调度线程中是否具有对二进制日志文件进行写操作的驱动组提交过程的组领导协程；IO 密集型调度线程中具有驱动批量提交的协程，将被该协程唤醒的协程迁移至 IO 密集型调度线程包括：若检测到具有对二进制日志文件进行写操作的组领导协程，将该组领导协程唤醒的组成员协程迁移至 IO 密集型调度线程，各个组成员协程迁移入的 IO 密集型调度线程基于负载均衡算法确定。

第二方面，本公开的实施例提供了一种数据库线程池控制装置，该装置包括：加载单元，被配置成在数据库的操作系统中加载能够将线程替换为协程的动态运行库，生成至少一个协程调度线程，协程调度线程用于调度至少一个协程；接收单元，被配置成接收客户端向数据库发送的连接请求；检测单元，被配置成检测数据库创建新线程的请求是否为处理连接请求的请求；创建单元，被配置成响应于检测到数据库创建新线程的请求为处理连接请求的请求，创建与连接请求对应的协程，并通过至少一个协程调度线程调度协程完成连接请求对应的访问任务。

在一些实施例中，上述加载单元被配置成：响应于数据库无法修改源代码且不具有连接管理插件机制时，启动数据库；加载并初始化动态运行库，生成至少一个协程调度线程，动态运行库具有创建线程的钩子函数，创建线程的钩子函数用于采用协程替换数据库的原生运行库生成的连接处理线程；在数据库初始化时，重载创建线程的钩子函数，以通过创建线程的钩子函数判断当前创建的线程是否为连接处理线程，在当前创建的线程是连接处理线程时，将该连接处理线程转换为协程。

在一些实施例中，上述加载单元被配置成：响应于数据库具有连接管理插件

机制时，启动数据库；加载并初始化动态运行库，生成至少一个协程调度线程；初始化数据库；加载并初始化插件，以使插件接收向数据库发送的连接请求，并数据库的线程构建请求为用于处理连接请求的请求时，控制动态运行库中的创建线程的钩子函数工作，创建线程的钩子函数用于判断当前创建的线程是否为连接处理线程，在当前创建的线程是连接处理线程时，将该连接处理线程转换为协程。

在一些实施例中，上述装置还包括：划分单元，被配置成将至少一个协程调度线程划分为 IO 密集型调度线程和计算密集型调度线程；上述创建单元进一步被配置成：检测与连接请求对应的协程是否处于第一状态与第二状态之间的状态，其中，第一状态为协程处于等待执行数据库命令的数据包，数据包已到达，且该协程需要被唤醒时的状态；第二状态为协程位于计算密集型调度线程，且该协程发起对特定文件的特定写入请求的状态；若检测到与连接请求对应的协程处于第一状态与第二状态之间的状态，采用至少一个协程调度线程中的计算密集型调度线程调度与连接请求对应的协程；若检测到与连接请求对应的协程未处于第一状态与第二状态之间的状态，采用至少一个协程调度线程中的 IO 密集型调度线程调度与连接请求对应的协程。

在一些实施例中，上述创建单元进一步被配置成：响应于与连接请求对应的协程处于创建连接的阶段，将协程放置在当前的 IO 密集型调度线程，当前的 IO 密集型调度线程基于负载均衡算法确定；响应于与连接请求对应的协程处于对特定文件的特定写入操作，且完成写特定文件 IO 请求的提交以及让出协程进入 IO 等待之前，将协程迁移至当前的 IO 密集型调度线程。

在一些实施例中，上述数据库为 MySQL 数据库，上述构建单元进一步被配置成：响应于与连接请求对应的协程处于对二进制日志文件的写操作，且完成写文件 IO 请求的提交以及让出协程进入 IO 等待之前，将协程迁移至当前的 IO 密集型调度线程。

在一些实施例中，上述装置还包括：批驱单元，批驱单元被配置成：检测 IO 密集型调度线程中是否具有驱动批量提交的协程；若 IO 密集型调度线程中具有驱动批量提交的协程，将被该驱动批量提交协程唤醒的协程迁移至 IO 密集型调度线程，该驱动批量提交协程唤醒的协程迁移入的 IO 密集型调度线程基于负载均衡算法确定。

在一些实施例中，上述数据库为 MySQL 且开启组提交的数据库，上述批驱

单元进一步被配置成：检测 IO 密集型调度线程中是否具有对二进制日志文件进行写操作的驱动组提交过程的领导协程；若检测到具有对二进制日志文件进行写操作的组领导协程，将该组领导协程唤醒的组成员协程迁移至 IO 密集型调度线程，各个组成员协程迁移入的 IO 密集型调度线程基于负载均衡算法确定。

第三方面，本公开的实施例提供了一种电子设备，该电子设备包括：一个或多个处理器；存储装置，其上存储有一个或多个程序；当一个或多个程序被一个或多个处理器执行，使得一个或多个处理器实现如第一方面中任一实施例描述的方法。

第四方面，本公开的实施例提供了一种计算机可读介质，其上存储有计算机程序，该程序被处理器执行时实现如第一方面中任一实施例描述的方法。

附图说明

通过阅读参照以下附图所作的对非限制性实施例所作的详细描述，本公开的其他特征、目的和优点将会变得更明显：

图 1 是根据本公开的数据库线程池控制方法的一个实施例的流程图；

图 2 是本公开的一种数据库结构示意图；

图 3 是根据本公开的数据库线程池控制方法的另一个实施例的流程图；

图 4 是根据本公开的数据库线程池控制装置的一个实施例的结构示意图；

图 5 是适于用来实现本公开的实施例的电子设备的结构示意图。

具体实施方式

下面结合附图和实施例对本公开作进一步的详细说明。可以理解的是，此处所描述的具体实施例仅仅用于解释相关发明，而非对该发明的限定。另外还需要说明的是，为了便于描述，附图中仅示出了与有关发明相关的部分。

需要说明的是，在不冲突的情况下，本公开中的实施例及实施例中的特征可以相互组合。下面将参考附图并结合实施例来详细说明本公开。

传统数据库技术中，Percona 推出了线程池的方案，通过在线程池中分时执行不同连接中的 SQL（Structured Query Language，结构化查询语言）指令，有效控制线程个数，减少线程切换，从而在一定程度上保证系统的性能和稳定性。

在线程池开启的情况下，客户端连接会按照任务优先级策略的设定，根据其

将要执行的 SQL 指令类型，被放入不同的优先级队列。当高优先级队列中有任务时，就不会执行低优先级队列中的任务。

一般情况下，当任务队列中有任务可被执行，但没有空闲工作线程时，就会启动新的工作线程调度这些任务。工作线程的个数不会超过线程池指定的上限；而参数“thread_pool_max_threads”则设定整个线程池被允许创建的最大线程数；当工作线程持续空闲超过“thread_pool_idle_timeout”参数设定的时间，即会自动退出。

但是，如果正在被调度的任务执行时间都普遍较长，而与此同时工作线程个数已经达到上限；那么此时位于任务队列中的任务将长时间得不到调度执行。另一种类似情况，即高优先级任务队列中持续涌入大量任务时，低优先级队列中的任务亦将长时间无法被执行。这种情况在并发任务调度中是一种常见现象，被形象的称为任务“饿死”。

现有数据库线程池技术中，MySQL 的线程池存在“饿死”问题。要缓解“饿死”，就必须放宽对超限线程个数的限制，或者通过“extra port”创建连接。但是，如果过分使用超限线程和 extra port，线程池所谓“线程个数上限”的限制就变得形同虚设。因而，现有公开技术中的数据库线程池实现，普遍存在“饿死”和线程个数上限不受控的二难问题。而即便在线程池个数不受限的情况下，“thread_pool_stall_limit”参数（默认 10ms）控制的超时响应机制，也会把一条原本仅需几个微秒的 SQL 延长到 10ms 以上（触发超时使用超限线程的情况），进而引发 SQL 响应速度的剧烈抖动。

之所以出现上述“饿死”现象，其核心问题实际上是：一个长时间运行的 SQL，或者一个长时间处于等待状态（如锁等待造成的线程挂起）的 SQL，在这个 SQL 执行完毕之前，都无法从当前运行这个 SQL 的线程中被换出。也就是说，即使工作线程在执行某个 SQL 过程中处于挂起（锁等待）状态，也无法复用此线程调度执行其他的 SQL 任务；要想同时执行其他的 SQL，只能使用其他的线程。

为了更好地理解传统技术，下面对一些名词进行介绍：

MariaDB: 目前最流行的开源关系型数据库之一，可以看作是 MySQL 数据库的一个分支。它由 MySQL 的原始开发者制作，并保证永远保持开源。

Percona: 一种开源关系型数据库，Percona 的线程池是在移植 MariaDB 线程池源码的基础上，对线程池做了功能增强和性能优化。

线程池 (thread pool): 线程池顾名思义就是事先创建若干个可执行的线程放入一个池 (容器) 中, 需要执行任务时, 不是为这个任务创建一个新的线程, 而是使用线程池中已经创建好的线程。通过上述方案, 减少线程创建、销毁次数, 降低调度负担, 从而提升系统性能。

binlog: binlog 是一种二进制格式的日志文件, 用于记录用户对数据库更新操作的历史, 常用于实现数据库主从架构下的增量复制。其中, 主库不断生成 binlog 日志, 并将增量部分同步至从库; 而从库则通过不断应用这些日志, 达到主从一致的目的。

semi-sync: 半同步复制, MySQL 的一种主从复制模式, 其性能和可靠性均介于完全同步复制和完全异步复制之间。不同于完全同步复制, semi-sync 不追求主从之间数据可见性的实时一致, 而是仅保证了主从之间不会发生日志丢失的情况 (即, 保证主从间数据可见性的最终一致性)。通过弱化一致性要求, 达到提升主从复制性能的目的。

中继日志 (relay log): 基于 binlog 的数据库主从同步方案中, 从库 (即 binlog 接收方) 可以通过一种叫做 “中继日志” 的策略实现日志接收和应用的异步化。从库接收主库发来的 binlog 时, 可以将日志流落地成本地文件; 而后台的日志应用线程则可通过不断读取这些本地日志文件, 实现异步化的日志应用。上述从库生成的本地日志文件, 就被称为 “中继日志”。

组提交 (group commit): 数据库执行提交操作时需要写入日志, 而日志文件需要串行写入。在大批量并发写入的场景下, 如果每个事务都做一次日志写入, 实际的写入速度就被拖慢至日志文件串行写入的速度。组提交是一种针对上述问题的性能优化方案。大量并发写入时, 可以将多个几乎同时提交的事务的日志合并, 这样这一组事务只需要一次日志写入, 从而提升了并发性能。在 MySQL 中, 每个 binlog 组提交都会选择本组中事务号最小的事务为组 leader, 并由这个组 leader 负责驱动整个提交流程。

本公开提供了一种数据库线程池控制方法, 通过动态运行库注入, 在数据库无感知的情况下, 可以将数据库中连接处理线程透明替换为协程, 可以保证执行连接请求的线程数不会超过调度线程个数, 提升了数据库线程池的并发性能。如图 1, 示出了根据本公开的数据库线程池控制方法的一个实施例的流程 100, 该

数据库线程池控制方法包括以下步骤：

步骤 101, 在数据库的操作系统中加载能够将线程替换为协程的动态运行库, 生成至少一个协程调度线程。

本实施例中, 协程调度线程用于调度至少一个协程。

本实施例中, 动态运行库是支持数据库运行的运行库, 可以为数据库在运行时与操作系统间的交互提供编程接口。

本实施例中, 动态运行库是一种动态运行库, 编译好的函数存储于动态运行库中, 动态运行库通过函数向数据库提供 API 接口, 数据库通过该 API 接口调用动态运行库中的函数。在原生运行库的基础, 通过对原生运行库中与线程调度相关的库函数进行 hook, 可以得到动态运行库, 并且得到动态运行库在加载之后, 可以具有与线程等价的调度能力。

本实施例中, 可以全面对 C 运行库进行 hook, 得到动态运行库, 通过动态运行库提供运行时虚拟化环境, 数据库在该运行时虚拟化环境下运行, 实现以协程方式模拟原生运行库中实现的线程, 达到协程模拟线程以及通过协程调度线程调度协程的目的。

本实施例中, 加载动态运行库是在操作系统启动数据库时, 向该数据库动态注入动态运行库, 而动态运行库的运行时注入, 就是在操作系统启动一个数据库, 并将该程序向内存中进行加载之前, 先将某个动态运行库加载进来, 并运行这个动态运行库中的初始化代码。通过这种程序加载方式, 这个被预先载入的动态运行库就有机会提前构建出一个运行时的虚拟化环境, 在该运行时的虚拟化环境下, 数据库或者操作系统可以实时调用动态运行库中的函数, 本实施例中, 数据库启动之后, 相应可以启动协程调度线程。

本实施例中, 协程调度线程包括: 调度器以及用于放置至少一个协程的无锁队列, 协程调度线程调度至少一个协程的具体步骤包括: 调度器从无锁队列中获取至少一个协程中处于就绪状态的协程; 同步该协程的协程上下文与线程控制块中的线程上下文; 在该协程主动让出之后, 调度器从无锁队列中获取下一个处于就绪状态的协程, 并同步下一个协程的协程上下文与线程控制块中的线程上下文; 在下一个协程主动让出之后, 调度器继续从无锁队列中获取协程并同步线程控制块, 直至无锁队列中无任何协程为止。

本实施例中, 线程控制块是原生运行库中线程的数据结构, 叫做“线程控制

块”(Thread Control Block, 简称 TCB), 也就是线程的上下文结构。dtv 和 specific array, 就是 TCB 里面存储 TLS 数据的数据结构。dtv: 用于存储编码阶段就定义好的 TLS 变量。这个数据会随着数据库加载新的动态运行库而产生增量, 因为动态运行库中也有可能存在 TLS 变量。

specific: 动态创建的 TLS (其中, 创建和删除 TLS key 的库函数分别为: pthread_key_create 和 pthread_key_delete; 而读取和修改 TLS 值的库函数分别为: pthread_getspecific 和 pthread_setspecific)。这类 TLS 用一组指针数组维护(specific arrays), 因而个数存在上限 (Linux 最多支持创建 1024 个动态 TLS)。

本实施例中, 在数据库的源代码是开源的代码时, 通过直接修改和重新编译数据库的源代码得到动态运行库, 在加载动态运行库之后, 可以使数据库运行新的虚拟化环境中。可选地, 在数据库支持插件化开发时, 可以采用插件的插件接口实现连接响应, 连接管理插件在响应新连接的时候, 调用创建协程的钩子函数, 实现连接线程转协程的目的, 其中, 创建协程的钩子函数是在数据库启动时注入的, 加载插件的时机可以在用户希望的任何阶段。

步骤 102, 接收客户端向数据库发送的连接请求。

本实施例中, 连接请求是客户端向运行动态运行库之后的数据库发送的连接请求。

本实施例中, 为了将协程调度线程与数据库其他后台线程分开, 在数据库启动阶段挂载动态运行库时, 需要通过环境变量设定默认线程创建模式, 将默认行为设成启动系统原生线程。

步骤 103, 检测数据库创建新线程的请求是否为处理连接请求的请求。

本实施例中, 加载动态运行库之后, 动态运行库中的钩子函数的程序直接替换数据库中的连接管理器模块 (connection handler), 该钩子函数运行可以使数据库新建的连接处理线程 (session) 直接转换为协程。当采用插件的形式加载动态运行库时, 通过挂载线程池插件替换连接处理程序, 并在线程池插件替换连接管理器模块之后, 数据新建的连接处理线程直接被该线程池插件转换为协程, 在卸载该线程池插件时, 需要保证所有协程化的连接处理线程都已关闭, 否则会报“插件资源繁忙”错误。

本实施例中, 创建新线程的请求是数据库发出的创建线程的请求, 数据库接收到客户端的连接请求之后, 会为每个客户端连接创建一个对应的连接处理线

程，而创建连接处理线程的请求即为创建新线程的请求，也即向动态运行库的应用程序接口发出创建新线程的请求，本实施例数据库线程池控制方法运行于其上的执行主体可以在得到创建新线程的请求为处理连接请求的请求之后，构建连接处理线程，并通过构建的连接处理线程与客户端建立连接。可选地，执行主体还可以在得到创建新线程的请求为处理连接请求的请求之后，采用协程的代替构建的连接处理线程，并通过协程调度线程调度执行协程的方式与客户端建立连接。

步骤 104，响应于检测到数据库创建新线程的请求为处理连接请求的请求，创建与连接请求对应的协程，并通过至少一个协程调度线程调度协程完成连接请求对应的访问任务。

本实施例中，通过注入动态运行库，生成协程调度线程，可以使数据库中所有线程完全通过协程调度线程调度，同时，利用协程可让出性以及分时复用调度线程时间片的性质，可以保证数据库既不会出现任务“饿死”，也不会出现由于任务长时间无法被调度造成的响应时间抖动的问题。

本实施例中，将处理一个用户的连接处理线程的过程（从连接创建、到不断接收并执行 SQL、一直到连接关闭）看作一个访问任务，创建的与连接请求对应的协程就是这个任务的载体。针对连接请求，协程是可以不断创建的，唯一不变的是协程调度线程，为此，本公开需要通过有限的协程调度线程调度不断创建的协程。

如图 2 所示，示出了本公开数据库的一种结构示意图，在图 2 中，创建数据库中新连接的方式为：在接收到用于处理连接的线程构建请求之后，通过原生运行库中对创建线程的库函数的 HOOK（如图 2 中的 `pthread_create()`hook），将数据库的连接处理线程转换为协程，生成的协程可用于处理数据库中的连接，并且生成的协程可以被多个协程调度线程调度，达到了数据库“线程池”的效果，本实施例中，用于处理连接的线程构建请求是数据库处理连接请求的创建新线程的请求，在该请求被同意之后，数据库可以创建处理客户端连接请求的连接处理线程，而连接处理线程是处理客户端的连接请求的线程；进一步地，通过动态运行库中的钩子函数可以在具有用于处理连接的线程构建请求之后，直接构建代替连接处理线程的协程。

本公开的实施例提供的数据库线程池控制方法，首先，在数据库的操作系统中加载能够将线程替换为协程的动态运行库，生成至少一个协程调度线程，协程

调度线程用于调度至少一个协程；其次，接收客户端向数据库发送的连接请求；再次，检测数据库创建新线程的请求是否为处理连接请求的请求；最后，响应于检测到数据库创建新线程的请求为处理连接请求的请求，创建与连接请求对应的协程，并通过至少一个协程调度线程调度协程完成连接请求对应的访问任务。由此，本公开通过动态运行库的注入方式，将数据库中用于处理连接的处理线程透明替换为协程，该创建的与连接请求对应的协程以分时调用的形式运行，当某个协程中出现挂起等待等耗时操作时，可以主动让出当前协程调度线程，并由协程调度线程换入另一个可执行的协程，可以解决低优先级任务的延迟加剧和任务响应速度抖动的问题，提高了数据库的并发性能。

针对上述实施例，通过注入运行动态运行库，可以使数据库所有连接处理线程转化为连接处理协程，并且连接处理协程完全被协程调度线程调度，实现了与通过原生运行库的线程池方案类似的线程模型，但是数据库在运行时可能存在两种不同类型的负载，该负载一般可分为计算密集型和 IO 密集型，如图 3 所示，一主（Z）两从（C1、C2）架构的三节点 MySQL 数据库集群中，主节点上 SQL 执行的时间序列包括：接收 SQL 指令 -> 执行 SQL 指令-> 将修改内容写入 binlog -> 将 binlog 增量同步至两个从节点-> 完成 SQL 提交。当前部署 MySQL 主从复制大多采用 MySQL 集群及高可用-半同步模式，这种模式下，主从复制阶段的流程（对于开启组提交（binlog group commit）时，下面描述的是组 leader 的流程）步骤如下：

- 1) 主库将事务写入 binlog;
- 2) 主库将 binlog 传输至从库;
- 3) 从库接收主库发来的 binlog，并将其内容落地到中继日志（图 3 中接收中继日志），而后向主库发送确认应答;
- 4) 主库等待从库的确认字符应答，直到所有从库均已应答;
- 5) 主库完成事务提交，并向客户端提交返回“commit OK”。

一个 MySQL 的连接处理协程（通过对连接处理线程进行协程转化得到），会周而复始的重复上述过程，执行客户端发起的 SQL 语句。从整个流程中可以看出，只有执行 SQL 指令阶段是计算密集型的，此时 CPU 占用率较高；其他阶段都是发起 IO 或等待 IO 完成，即 IO 密集型负载。而 IO 密集型负载本身 CPU 消

耗很少，但由于需要频繁的进行 IO 等待，因而需要触发大量的协程换出换入动作。如果把 IO 密集型负载和计算密集型负载放到同一组调度线程中执行，IO 密集型负载频繁的换入换出动作就会打乱计算密集型任务的调度节奏，增加调度负担，降低整体性能。因为，协程是依靠分时复用调度线程的时间片实现模拟并行，调度线程处理换入换出也要消耗 CPU 时间；所以，调度线程在换入换出上耗费的 CPU 时间越多，用在执行计算任务的 CPU 时间就越少。

本实施例中，动态运行库加载后对所有数据库中线程的管理，默认将所有线程都转为协程，导致不同类型的协程（如，主从复制的 IO 协程、接收应答信号的协程等），以及数据库命令中不同类型的负载，都由同一批协程调度线程调度，导致不同类型协程负载的混合调度，IO 密集型负载需要以换出/换入方式进行 IO 等待，会打乱计算密集型负载的调度节奏，增加线程处理协程换入换出的负担。

为解决上述性能问题，本公开提供了一种数据库线程池控制方法的另一个实施例，该数据库线程池控制方法包括以下步骤：在数据库的操作系统中加载能够将线程替换为协程的动态运行库，生成至少一个协程调度线程，协程调度线程用于调度至少一个协程；将至少一个协程调度线程划分为 IO 密集型调度线程和计算密集型调度线程；接收客户端向数据库发送的连接请求，检测连接请求是否用于处理连接的线程构建请求；响应于检测到连接请求是用于处理连接的线程构建请求，创建与连接请求对应的协程，检测与连接请求对应的协程是否处于第一状态与第二状态之间的状态，其中，第一状态为协程处于等待执行数据库命令的数据包，数据包已到达，且该协程需要被唤醒时的状态；第二状态为协程位于计算密集型调度线程，且该协程发起对特定文件的特定写入请求的状态；若检测到与连接请求对应的协程处于第一状态与第二状态之间的状态，采用至少一个协程调度线程中的计算密集型调度线程调度与连接请求对应的协程；若检测到与连接请求对应的协程未处于第一状态与第二状态之间的状态，采用至少一个协程调度线程中的 IO 密集型调度线程调度与连接请求对应的协程。

本实施例中，将协程调度线程划分成两组：计算密集型调度线程和 IO 密集型调度线程，计算密集型调度线程用于调度执行计算密集型负载，IO 密集型调度线程用于调度执行 IO 密集型负载。

本实施例中，可以基于预设的划分规则，将至少一个协程调度线程划分为 IO 密集型调度线程和计算密集型调度线程，其中，预设的划分规则为：IO 密集型调

度线程与计算密集型调度线程的数量占比为固定值（如 1:4）。可选地，上述预设的划分规则还可以是：确定数据库的类型，基于数据库的类型，确定 IO 密集型调度线程与计算密集型调度线程的数量占比。对于 IO 操作较多的数据库，在划分规则的设置，可以将 IO 密集型调度线程的数量占比设置为较大的数值；对于计算操作较多的数据库，通过划分规则的设置，可以将计算密集型调度线程的数量占比设置为较大的数值。

本实施例中，与连接请求对应的协程是执行连接请求的协程，该协程可以处于执行连接请求的不同阶段，例如，接收数据库指令、执行数据库指令，向其他库提交指令处理结果，向客户端返回数据库指令执行结果阶段。其中，第一状态与第二状态之间的状态属于执行数据库指令阶段，在该执行数据库指令阶段需要大量的计算操作，将协程迁移至计算密集型调度线程进行调度，可以实现与 IO 密集型负载的分离。

本实施例中，第一状态与第二状态之间状态用于限定从等待执行数据库命令的数据包的操作即将被唤醒的点，到对特定文件发生特定写操作的点之间的这个阶段。在等待执行数据库命令的数据包的状态下，该数据包已经到达，将要唤醒该协程的时候，需要放入计算密集型调度线程。

对于 MySQL 数据库，第一状态与第二状态之间的状态是指：已经有 SQL 请求包的数据到达，等待 SQL 请求包的 IO 操作即将从等待状态被唤醒的状态。通过监控第一状态与第二状态之间的状态，可以将在该状态下的协程迁移到计算密集型调度线程，当该协程 resume 的时候，就正好在计算密集型调度线程中执行，并且开始处理这个刚接收到的 SQL 请求包。

以 MySQL 数据库为例，说明本公开数据库线程池控制方法的运行于其上的执行主体的工作原理如下：

1) 对新创建的连接，由于要执行握手确认和连接上下文初始化的工作（即创建连接的阶段），期间需要与客户端往复通信数次，属于 IO 密集型操作，因而该新连接对应的连接处理协程需要交由 IO 密集型调度线程处理，将该连接处理协程放置在 IO 密集型调度线程，使 IO 密集型调度线程调度该新连接对应的协程。

2) 当连接建立完成后，MySQL 的连接处理协程需要等待接收操作接收客户端发来的 SQL 指令数据包（MySQL 通过 poll 等待实现）；当 poll 结束等待（即，有数据包可接收时）并即将唤醒该连接处理协程时，即可将此连接处理协程迁移

至计算密集型调度线程。执行 SQL 运算过程中，当前协程始终保持在计算密集型调度线程中。

3) 当连接处理协程调用写操作开始写入 binlog 文件时，意味着连接处理协程开始进入主从同步和事务提交阶段（如开启 binlog group commit，则表示此协程为“组 leader”）；此时，在完成写文件 IO 请求的提交、并让出协程进入 IO 等待状态之前，将当前连接处理协程迁移至 IO 密集型调度线程。

从 binlog 写入、binlog 发送、ack 等待、以及完成事务提交，一直到有新的 SQL 指令可接收之前，当前连接处理协程始终保持在 IO 密集型调度线程中。

基于上述流程 1) -3)，周而复始，直到连接处理线程退出，即可实现 IO 密集型负载和计算密集型负载的分离。

本实施例中，可以通过动态运行库的回调接口注入调度线程迁移的逻辑，具体逻辑如下：

首先，假设挂载动态运行库时启动了 40 个协程调度线程；由于 IO 密集型任务 CPU 占用率很低，因而需要的线程数较少；将协程调度线程 ID 从 0~31 这 32 个调度线程划定为计算密集型调度线程，将协程调度线程 ID 从 32~39 这 8 个调度线程划定为 IO 密集型调度线程。

其次，将协程迁移至不同的协程调度线程时，线程池插件需采用负载均衡算法均衡迁移的协程调度线程，避免出现负载偏斜现象，可选地，在 MySQL 数据库，可以采用连接处理线程 ID 的 hash 值作为调度线程的选择因子，实现负载均衡算法。

基于上述前提，本公开了只需实现唤醒协程操作的回调函数，和写操作的回调函数，这两个回调函数，即可实现前文的协程分组调度机制。

在执行文件写入操作时，如果判断出当前协程写入的是 binlog 文件，则将其调度线程 ID 设为 32~39 之间的某个值（即，IO 密集型）。而在唤醒协程之前，如果判断出该协程即将进入 SQL 指令执行阶段，则将其调度线程 ID 设为 0~31 之间的某个值。

本公开的实施例提供的数据库线程池控制方法，通过将协程调度线程划分为两组，可以将 IO 密集型的调度性负载引流至专门的 IO 调度线程，减少了计算密集型调度线程的调度资源消耗，提高了数据库的并发性能。

在公开的一些可选实现方式中，上述在数据库的操作系统中加载能够将线程替换为协程的动态运行库，生成至少一个协程调度线程包括：响应于数据库无法修改源代码且不具有连接管理插件机制时，启动数据库；加载并初始化动态运行库，生成至少一个协程调度线程，动态运行库具有创建线程的钩子函数，创建线程的钩子函数用于采用协程替换数据库的原生运行库生成的连接处理线程；在数据库初始化时，重载创建线程的钩子函数，以通过创建线程的钩子函数判断当前创建的线程是否为连接处理线程，在当前创建的线程是连接处理线程时，将该连接处理线程转换为协程。

本实施例中，在创建线程的钩子函数中，监听当前创建的线程是否为连接处理线程，在当前创建的线程是连接处理线程时，采用协程上下文结构替换线程描述符，并通过带有线程本地约束的全局变量标记当前运行线程或协程，通过同步管理模块管理协程上下文结构中的线程本地存储信息，同步管理模块用于在协程调度线程调度协程时，同步该协程的线程本地存储。

本实施例中，协程上下文结构（以下称为 COCTX 结构）是放置协程上下文（以下称为 COCTX）的结构，用于维护协程的上下，其类似于线程描述符（或线程句柄）的作用。为了提供与线程行为的兼容性，需要 hook 原生运行库中创建线程、以及获取线程描述符的库函数，用 COCTX 替代线程描述符。以 Linux 为例，其线程描述符被称为“TCB”，其实现是一个名为“pthread”的结构体。Linux 中创建线程和获取线程描述符的调用分别是：

pthread_create: 创建线程并返回线程描述符，这个线程描述符的值，就是 TCB 结构的地址。本实施例中，创建线程函数的钩子就以返回 COCTX 的指针取而代之。

pthread_self: 获取当前线程的描述符，同样以 COCTX 的指针取代。

本实施例中，可以采用一个全局 TLS 变量（带有线程本地约束的全局变量）获取当前协程的上下文，其声明如下：

```
extern __thread COCTX* _current_coctx;
```

本实施例中，COCTX 有以下 4 种类型：

协程 COCTX: 对应一个实际的协程；

Global COCTX: 主线程的 _current_coctx 会被置为 global COCTX，这个 COCTX 在动态运行库刚刚被加载时就会被创建；

Scheduler COCTX: 每个协程调度线程（就是运行协程的线程）都会有一个默认的 COCTX，没有协程处于就绪状态，或者两个协程切换的中间状态，当前线程的 `_current_coctx` 就会被置为 scheduler COCTX;

Single COCTX: 创建系统原生线程时，会为其创建一个虚拟的 COCTX。

如上，`pthread_self` 的 hook 函数，实际上就是返回 `_current_coctx` 的值。程序中，通过判断当前 COCTX 的类型，就可以得知当前处于哪种线程、哪种运行状态。

本实施例中，COCTX 结构中存储的内容包括：协程栈以及 TLS。动态运行库为每个协程分配一个独立的栈，因而可以完整的模拟一个线程的行为（包括在栈中分配的内存大小和调用栈深度）。

如果协程不能完备的模拟线程的 TLS 机制，将会导致协程不能安全的访问带有 TLS 约束的变量。本实施例，通过同步管理模块（以下称为 `co_tls`）实现了对 TLS 管理的封装，`co_tls` 提供的能力对于“透明”模拟线程至关重要。以 Linux 为例，其 TCB 中存储以下两类 TLS：`dtv` 和 `specific`。

本公开的 `co_tls` 对 `specific` 的支持：`co_tls` 中需要维护一个与 TCB 中 `specific` arrays 类似的数据结构，当一个新创建的协程第一次被换入执行时，会复制其所属调度线程 TCB 中的 `specific`。此外，还需要 hook 管理 `specific` 的库函数，前文列举了 Linux 的库函数，Windows 中也有类似函数，分别是：`TlsAlloc`、`TlsFree`、`TlsGetValue`、和 `TlsSetValue`。

本公开的 `co_tls` 对 `dtv` 的支持：同样，新创建的协程第一次被调度执行时，也需要复制 TCB 中的 `dtv`。然而与 `specific` 不同的是，`co_tls` 中 `specific` 的内容无需再与 TCB 同步；但由于运行时可能动态加载其他动态运行库，因而 `co_tls` 的 `dtv` 需要与 TCB 中的 `dtv` 进行增量同步。在 scheduler COCTX 中存储一个对 TCB `dtv` 的增量备份，用以描述每次加载动态运行库时，`dtv` 产生的增量部分。同时，需要对加载动态运行库的库函数调用进行 hook（Linux 中该库函数叫做 `dlopen`，而 Windows 中则对应于 `LoadLibrary`）。以 Linux 为例，一个叫做“`link_map`”的数据结构描述了所有动态运行库信息，结合 `link_map` 和 `dtv` 增量备份，就可以解析出当前协程与 `dtv` 增量之间的差异。对于 `dtv` 增量的合并，采取“懒”同步方案，即在协程即将换入时再判断是否需要同步 `dtv`。如果 `dlopen` 是在某个协程中调用的，则需要在 `dlopen` 的 hook 中识别出 `_current_coctx` 为协程类型，并立即为

当前协程同步 dtv。

本实施例中，通过创建线程的钩子函数，即可模拟出与线程能力完全一致的协程。协程的切换过程如下：首先，协程调度线程在初始状态下，`_current_coctx` 为 scheduler COCTX；而后，当需要换入（resume）一个协程时，检查该协程与 dtv 增量备份之间是否存在差异（若存在则需同步），完成差异同步后将 `_current_coctx` 置为当前协程的上下文；最后，当该协程要求换出时，再把 `_current_coctx` 还原成 scheduler COCTX。整个协程切换过程完全在用户态执行，无需与操作系统发生任何形式的交互，从而保持了协程的轻量化特征。

本实施例实现了纯用户态的协程 TLS 切换，保持协程切换轻量化特征的同时，维系了 TLS 的跨协程安全性（兼顾高性能和“透明”虚拟化）。

本实施例提供的生成协程调度线程的方法，在数据库无法修改源代码且不具有连接管理插件机制时，通过向启动的数据库中注入具有创建线程的钩子函数的动态运行库，在数据库初始化时，重载创建线程的钩子函数，可以使创建线程的钩子函数截取创建连接处理线程的请求，并将连接处理线程转化为协程，本实施例提供的方法，为连接处理线程转换为协程提供了一种可靠的实现方式。

在公开的一些可选实现方式中，上述在数据库的操作系统中加载能够将线程替换为协程的动态运行库，生成至少一个协程调度线程包括：响应于数据库具有连接管理插件机制时，启动数据库；加载并初始化动态运行库，生成至少一个协程调度线程；初始化数据库；加载并初始化插件，以使插件接收向数据库发送的连接请求，并数据库的线程构建请求为用于处理连接请求的请求时，控制动态运行库中的创建线程的钩子函数工作，创建线程的钩子函数用于判断当前创建的线程是否为连接处理线程，在当前创建的线程是连接处理线程时，将该连接处理线程转换为协程。

本实施例中，连接管理插件机制是为了不修改数据库的前提下，通过插件对数据库功能进行扩展的机制，其也是一种不修改数据库源码的一种方案。

本实施例中，插件是一种位于数据库和动态运行库中的第三方设备，插件既可以调用动态运行库的应用接口，又可以调用数据库的应用接口，插件通过调用数据库的应用接口接收连接请求和数据库的线程构建请求，检测数据库的线程构建请求是否为用于处理连接请求的请求，并在数据库的线程构建请求为用于处理

连接请求的请求时，调取动态运行库的应用接口，通过调用动态运行库的应用接口，使创建线程的钩子函数工作，创建线程的钩子函数判断连接处理线程，并将连接处理线程转化为协程。

本实施例中，创建线程的钩子函数是通过 HOOK 机制处理 C 库中创建线程的函数得到的函数，创建线程的钩子函数的具体内容如上面实施例所描述，此处不再赘述。

本实施例提供的生成协程调度线程的方法，在数据库具有连接管理插件机制时，接入插件，通过插件接收连接请求，在连接请求为用户处理连接的线程构建请求时，调取动态运行库的创建线程的钩子函数工作，将连接处理线程转化为协程，本实施例提供的方法，为连接处理线程转换为协程提供了另一种可靠的实现方式。

在本公开的一些可选实现方式中，上述若检测到与连接请求对应的协程未处于第一状态与第二状态之间的状态，采用至少一个协程调度线程中的 IO 密集型调度线程调度与连接请求对应的协程包括：响应于与连接请求对应的协程处于创建连接的阶段，将协程放置在当前的 IO 密集型调度线程，当前的 IO 密集型调度线程基于负载均衡算法确定；响应于与连接请求对应的协程处于对特定文件的特定写入操作，且完成写特定文件 IO 请求的提交以及让出协程进入 IO 等待之前，将协程迁移至当前的 IO 密集型调度线程。

本实施例中，特定文件以及特征文件的写入操作可因数据库的类型而不同，例如，对于 MySQL 数据库，特定文件为 binlog 文件；对于 openGauss 数据库，特定文件为 xlog 文件，对于各种特定文件，各种类型数据具有相应的特定的写操作。

本可选实现方式提供的调度协程的方法，在数据库中的协程处于创建连接的阶段时，将协程迁移至 IO 密集型调度线程；在协程处于对特定文件的特定写入操作且完成特定文件 IO 请求的提交以及协程进入 IO 等待之前，将协程迁移至 IO 密集型调度线程，从而将处于 IO 密集型的任务引流至专门的 IO 调度线程，减少了计算密集型调度线程调度的资源消耗，提高了数据库的并发性能。

在本公开的一些可选实现方式中，当数据库为 MySQL 数据库时，上述响应

于与连接请求对应的协程处于对特定文件的特定写入操作,且完成写特定文件 IO 请求的提交以及让出协程进入 IO 等待之前,将协程迁移至当前的 IO 密集型调度线程包括:

响应于与连接请求对应的协程处于对二进制日志文件的写操作,且完成写文件 IO 请求的提交以及让出协程进入 IO 等待之前,将协程迁移至当前的 IO 密集型调度线程。

本实施例提供的迁移协程的方法,在数据库为 MySQL 数据库时,基于 MySQL 数据库中特定的二进制日志文件的写操作,将协程迁移至当前 IO 密集型调度线程,为 MySQL 数据库的协程的分配提供了一种可靠的实现方式。

在本公开的一些实施例中,上述数据库线程池控制方法还包括:检测 IO 密集型调度线程中是否具有驱动批量提交的协程;若 IO 密集型调度线程中具有驱动批量提交的协程,将被该驱动批量提交的协程唤醒的协程迁移至 IO 密集型调度线程,该驱动批量提交协程唤醒的协程迁移入的 IO 密集型调度线程基于负载均衡算法确定。

本实施例中,可以在动态运行库中,增加一个唤醒协程回调函数,该回调函数中具有相应的判断逻辑,判断逻辑为:正在执行唤醒其他协程操作的协程,如果其自身处于 IO 密集型调度线程中,则被其唤醒的协程也需要被迁移至 IO 密集型调度线程。

本实施例提供的数据库线程池控制方法,在 IO 密集型调度线程中具有驱动批量提交的协程时,将该驱动批量提交的协程唤醒的协程迁移至当前的 IO 密集型调度线程,可以提高协程调度效率。

在本公开的一些可选实现方式中,上述数据库为 MySQL 且开启组提交的数据库,检测 IO 密集型调度线程中是否具有驱动批量提交的协程包括:检测 IO 密集型调度线程中是否具有对二进制日志文件进行写操作的驱动组提交过程的组领导协程;IO 密集型调度线程中具有驱动批量提交的协程,将被该协程唤醒的协程迁移至 IO 密集型调度线程包括:若检测到具有对二进制日志文件进行写操作的组领导协程,将该组领导协程唤醒的组成员协程迁移至 IO 密集型调度线程,各个组成员协程迁移入的 IO 密集型调度线程基于负载均衡算法确定。

本实施例中，在 IO 密集型任务中，只有二进制日志文件组提交的组领导协程会在接收到从库的确认消息后，唤醒本提交组中的所有其他任务；而此时，被唤醒的任务也必然处于提交状态，因而需要被切换至 IO 密集型调度线程。

本实施例提供的迁移协程的方法，在数据库为 MySQL 数据库时，基于 MySQL 数据库中特定的二进制日志文件的写操作确定组领导协程，将该组领导协程唤醒的组成员协程迁移至当前 IO 密集型调度线程，为 MySQL 数据库的组成员协程的分配提供了一种可靠的实现方式。

进一步参考图 4，作为对上述各图所示数据库线程池控制方法的实现，本公开提供了数据库线程池控制装置的一个实施例，该装置实施例与图 1 所示的方法实施例相对应，该装置具体可以应用于各种电子设备中。

如图 4 所示，本公开的实施例提供了一种数据库线程池控制装置 400，该装置 400 包括：加载单元 401、接收单元 402、检测单元 403、创建单元 404。其中，上述加载单元 401，可以被配置成在数据库的操作系统中加载能够将线程替换为协程的动态运行库，生成至少一个协程调度线程，协程调度线程用于调度至少一个协程；上述接收单元 402，可以被配置成接收客户端向数据库发送的连接请求；上述检测单元 403，可以被配置成检测数据库创建新线程的请求是否为处理连接请求的请求；上述创建单元 404，可以被配置成响应于检测到数据库创建新线程的请求为处理连接请求的请求，创建与连接请求对应的协程，并通过至少一个协程调度线程调度协程完成连接请求对应的访问任务。

在本实施例中，数据库线程池控制装置 400 中，加载单元 401、接收单元 402、检测单元 403、创建单元 404 的具体处理及其所带来的技术效果可分别参考图 1 对应实施例中的步骤 101、步骤 102、步骤 103、步骤 104。

在一些实施例中，上述加载单元 401 被配置成：响应于数据库无法修改源代码且不具有连接管理插件机制时，启动数据库；加载并初始化动态运行库，生成至少一个协程调度线程，动态运行库具有创建线程的钩子函数，创建线程的钩子函数用于采用协程替换数据库的原生运行库生成的连接处理线程；在数据库初始化时，重载创建线程的钩子函数，以通过创建线程的钩子函数判断当前创建的线程是否为连接处理线程，在当前创建的线程是连接处理线程时，将该连接处理线程转换为协程。

在一些实施例中，上述加载单元 401 被配置成：响应于数据库具有连接管理插件机制时，启动数据库；加载并初始化动态运行库，生成至少一个协程调度线程；初始化数据库；加载并初始化插件，以使插件接收向数据库发送的连接请求，并数据库的线程构建请求为用于处理连接请求的请求时，控制动态运行库中的创建线程的钩子函数工作，创建线程的钩子函数用于判断当前创建的线程是否为连接处理线程，在当前创建的线程是连接处理线程时，将该连接处理线程转换为协程。

在一些实施例中，上述装置还包括：划分单元（图中未示出），上述划分单元被配置成将至少一个协程调度线程划分为 IO 密集型调度线程和计算密集型调度线程；上述创建单元 404 进一步被配置成：检测与连接请求对应的协程是否处于第一状态与第二状态之间的状态，其中，第一状态为协程处于等待执行数据库命令的数据包，数据包已到达，且该协程需要被唤醒时的状态；第二状态为协程位于计算密集型调度线程，且该协程发起对特定文件的特定写入请求的状态；若检测到与连接请求对应的协程处于第一状态与第二状态之间的状态，采用至少一个协程调度线程中的计算密集型调度线程调度与连接请求对应的协程；若检测到与连接请求对应的协程未处于第一状态与第二状态之间的状态，采用至少一个协程调度线程中的 IO 密集型调度线程调度与连接请求对应的协程。

在一些实施例中，上述创建单元 404 进一步被配置成：响应于与连接请求对应的协程处于创建连接的阶段，将协程放置在当前的 IO 密集型调度线程，当前的 IO 密集型调度线程基于负载均衡算法确定；响应于与连接请求对应的协程处于对特定文件的特定写入操作，且完成写特定文件 IO 请求的提交以及让出协程进入 IO 等待之前，将协程迁移至当前的 IO 密集型调度线程。

在一些实施例中，上述数据库为 MySQL 数据库，上述构建单元 404 进一步被配置成：响应于与连接请求对应的协程处于对二进制日志文件的写操作，且完成写文件 IO 请求的提交以及让出协程进入 IO 等待之前，将协程迁移至当前的 IO 密集型调度线程。

在一些实施例中，上述装置还包括：批驱单元（图中未示出），上述批驱单元被配置成：检测 IO 密集型调度线程中是否具有驱动批量提交的协程；若 IO 密集型调度线程中具有驱动批量提交的协程，将被该驱动批量提交协程唤醒的协程迁移至 IO 密集型调度线程，该驱动批量提交协程唤醒的协程迁移入的 IO 密集型调

度线程基于负载均衡算法确定。

在一些实施例中，上述数据库为 MySQL 且开启组提交的数据库，上述批驱单元进一步被配置成：检测 IO 密集型调度线程中是否具有对二进制日志文件进行写操作的驱动组提交过程的领导协程；若检测到具有对二进制日志文件进行写操作的组领导协程，将该组领导协程唤醒的组成员协程迁移至 IO 密集型调度线程，各个组成员协程迁移入的 IO 密集型调度线程基于负载均衡算法确定。

本公开的实施例提供的数据库线程池控制装置，首先，加载单元 401 在数据库的操作系统中加载能够将线程替换为协程的动态运行库，生成至少一个协程调度线程，协程调度线程用于调度至少一个协程；其次，接收单元 402 接收客户端向数据库发送的连接请求；再次，检测单元 403 检测数据库创建新线程的请求是否为处理连接请求的请求；最后，创建单元 404 响应于检测到数据库创建新线程的请求为处理连接请求的请求，创建与连接请求对应的协程，并通过至少一个协程调度线程调度协程完成连接请求对应的访问任务。由此，本公开通过动态运行库的注入方式，将数据库中用于处理连接的处理线程透明替换为协程，该创建的与连接请求对应的协程以分时调用的形式运行，当某个协程中出现挂起等待等耗时操作时，可以主动让出当前协程调度线程，并由协程调度线程换入另一个可执行的协程，可以解决低优先级任务的延迟加剧和任务响应速度抖动的问题，提高了数据库的并发性能。

下面参考图 5，其示出了适于用来实现本公开的实施例的电子设备 500 的结构示意图。

如图 5 所示，电子设备 500 可以包括处理装置（例如中央处理器、图形处理器等）501，其可以根据存储在只读存储器（ROM）502 中的程序或者从存储装置 508 加载到随机访问存储器（RAM）503 中的程序而执行各种适当的动作和处理。在 RAM 503 中，还存储有电子设备 500 操作所需的各种程序和数据。处理装置 501、ROM 502 以及 RAM 503 通过总线 504 彼此相连。输入/输出（I/O）接口 505 也连接至总线 504。

通常，以下装置可以连接至 I/O 接口 505：包括例如触摸屏、触摸板、键盘、鼠标、等的输入装置 506；包括例如液晶显示器（LCD，Liquid Crystal Display）、扬声器、振动器等的输出装置 507；包括例如磁带、硬盘等的存储装置 508；以及

通信装置 509。通信装置 509 可以允许电子设备 500 与其他设备进行无线或有线通信以交换数据。虽然图 5 示出了具有各种装置的电子设备 500，但是应理解的是，并不要求实施或具备所有示出的装置。可以替代地实施或具备更多或更少的装置。图 5 中示出的每个方框可以代表一个装置，也可以根据需要代表多个装置。

特别地，根据本公开的实施例，上文参考流程图描述的过程可以被实现为计算机软件程序。例如，本公开的实施例包括一种计算机程序产品，其包括承载在计算机可读介质上的计算机程序，该计算机程序包含用于执行流程图所示的方法的程序代码。在这样的实施例中，该计算机程序可以通过通信装置 509 从网络上被下载和安装，或者从存储装置 508 被安装，或者从 ROM 502 被安装。在该计算机程序被处理装置 501 执行时，执行本公开的实施例的方法中限定的上述功能。

需要说明的是，本公开的实施例的计算机可读介质可以是计算机可读信号介质或者计算机可读存储介质或者是上述两者的任意组合。计算机可读存储介质例如可以是——但不限于——电、磁、光、电磁、红外线、或半导体的系统、装置或器件，或者任意以上的组合。计算机可读存储介质的更具体的例子可以包括但不限于：具有一个或多个导线的电连接、便携式计算机磁盘、硬盘、随机访问存储器 (RAM)、只读存储器 (ROM)、可擦式可编程只读存储器 (EPROM 或闪存)、光纤、便携式紧凑磁盘只读存储器 (CD-ROM)、光存储器件、磁存储器件、或者上述的任意合适的组合。在本公开的实施例中，计算机可读存储介质可以是任何包含或存储程序的有形介质，该程序可以被指令执行系统、装置或者器件使用或者与其结合使用。而在本公开的实施例中，计算机可读信号介质可以包括在基带中或者作为载波一部分传播的数据信号，其中承载了计算机可读的程序代码。这种传播的数据信号可以采用多种形式，包括但不限于电磁信号、光信号或上述的任意合适的组合。计算机可读信号介质还可以是计算机可读存储介质以外的任何计算机可读介质，该计算机可读信号介质可以发送、传播或者传输用于由指令执行系统、装置或者器件使用或者与其结合使用的程序。计算机可读介质上包含的程序代码可以用任何适当的介质传输，包括但不限于：电线、光缆、RF (Radio Frequency, 射频) 等等，或者上述的任意合适的组合。

上述计算机可读介质可以是服务器中所包含的；也可以是单独存在，而未装配入该服务器中。上述计算机可读介质承载有一个或者多个程序，当上述一个或者多个程序被该服务器执行时，使得该服务器：在数据库的操作系统中加载能够

将线程替换为协程的动态运行库，生成至少一个协程调度线程，协程调度线程用于调度至少一个协程；接收客户端向数据库发送的连接请求；检测数据库创建新线程的请求是否为处理连接请求的请求；响应于检测到数据库创建新线程的请求为处理连接请求的请求，创建与连接请求对应的协程，并通过至少一个协程调度线程调度协程完成连接请求对应的访问任务。

可以以一种或多种程序设计语言或其组合来编写用于执行本公开的实施例的操作的计算机程序代码，程序设计语言包括面向对象的程序设计语言—诸如Java、Smalltalk、C++，还包括常规的过程式程序设计语言—诸如“C”语言或类似的程序设计语言。程序代码可以完全地在用户计算机上执行、部分地在用户计算机上执行、作为一个独立的软件包执行、部分在用户计算机上部分在远程计算机上执行、或者完全在远程计算机或服务器上执行。在涉及远程计算机的情形中，远程计算机可以通过任意种类的网络——包括局域网(LAN)或广域网(WAN)—连接到用户计算机，或者，可以连接到外部计算机（例如利用因特网服务提供商来通过因特网连接）。

附图中的流程图和框图，图示了按照本公开的各种实施例的系统、方法和计算机程序产品的可能实现的体系架构、功能和操作。在这点上，流程图或框图中的每个方框可以代表一个模块、程序段、或代码的一部分，该模块、程序段、或代码的一部分包含一个或多个用于实现规定的逻辑功能的可执行指令。也应当注意，在有些作为替换的实现中，方框中所标注的功能也可以以不同于附图中所标注的顺序发生。例如，两个接连地表示的方框实际上可以基本并行地执行，它们有时也可以按相反的顺序执行，这依所涉及的功能而定。也要注意的，框图和/或流程图中的每个方框、以及框图和/或流程图中的方框的组合，可以用执行规定的功能或操作的专用的基于硬件的系统来实现，或者可以用专用硬件与计算机指令的组合来实现。

描述于本公开的实施例中所涉及到的单元可以通过软件的方式实现，也可以通过硬件的方式来实现。所描述的单元也可以设置在处理器中，例如，可以描述为：一种处理器，包括加载单元、接收单元、检测单元、创建单元。其中，这些单元的名称在某种情况下并不构成对该单元本身的限定，例如，加载单元还可以被描述为“被配置成在数据库的操作系统中加载能够将线程替换为协程的动态运行库，生成至少一个协程调度线程”的单元。

以上描述仅为本公开的较佳实施例以及对所运用技术原理的说明。本领域技术人员应当理解，本公开的实施例中所涉及的发明范围，并不限于上述技术特征的特定组合而成的技术方案，同时也应涵盖在不脱离上述发明构思的情况下，由上述技术特征或其等同特征进行任意组合而形成的其它技术方案。例如上述特征与本公开的实施例中公开的(但不限于)具有类似功能的技术特征进行互相替换而形成的技术方案。

权利要求书

1、一种数据库线程池控制方法，所述方法包括：

在数据库的操作系统中加载能够将线程替换为协程的动态运行库，生成至少一个协程调度线程，所述协程调度线程用于调度至少一个协程；

接收客户端向所述数据库发送的连接请求；

检测所述数据库创建新线程的请求是否为处理所述连接请求的请求；

响应于检测到所述数据库创建新线程的请求为处理所述连接请求的请求，创建与所述连接请求对应的协程，并通过所述至少一个协程调度线程调度所述协程完成所述连接请求对应的访问任务。

2、根据权利要求1所述的方法，其中，所述在数据库的操作系统中加载能够将线程替换为协程的动态运行库，生成至少一个协程调度线程包括：

响应于所述数据库无法修改源代码且不具有连接管理插件机制时，启动数据库；

加载并初始化动态运行库，生成至少一个协程调度线程，所述动态运行库具有创建线程的钩子函数，所述创建线程的钩子函数用于采用协程替换所述数据库的原生运行库生成的连接处理线程；

在所述数据库初始化时，重载所述创建线程的钩子函数，以通过所述创建线程的钩子函数判断当前创建的线程是否为连接处理线程，在当前创建的线程是连接处理线程时，将该连接处理线程转换为协程。

3、根据权利要求1所述的方法，其中，所述在数据库的操作系统中加载能够将线程替换为协程的动态运行库，生成至少一个协程调度线程包括：

响应于所述数据库具有连接管理插件机制时，启动所述数据库；

加载并初始化动态运行库，生成至少一个协程调度线程；

初始化所述数据库；

加载并初始化插件，以使所述插件接收向所述数据库发送的连接请求，并在所述数据库的线程构建请求为用于处理所述连接请求的请求时，控制所述动态运行库中的创建线程的钩子函数工作，所述创建线程的钩子函数用于判断当前创建的线程是否为连接处理线程，在当前创建的线程是连接处理线程时，将该连接处理线程转换为协程。

4、根据权利要求 1-3 之一所述的方法，其中，所述方法还包括：

将所述至少一个协程调度线程划分为 IO 密集型调度线程和计算密集型调度线程；

所述通过所述至少一个协程调度线程调度所述协程完成所述连接请求对应的访问任务包括：

检测与所述连接请求对应的协程是否处于第一状态与第二状态之间的状态，其中，所述第一状态为协程处于等待执行数据库命令的数据包，所述数据包已到达，且该协程需要被唤醒时的状态；所述第二状态为协程位于计算密集型调度线程，且该协程发起对特定文件的特定写入请求的状态；

若检测到与所述连接请求对应的协程处于第一状态与第二状态之间的状态，采用所述至少一个协程调度线程中的计算密集型调度线程调度与所述连接请求对应的协程；

若检测到与所述连接请求对应的协程未处于第一状态与第二状态之间的状态，采用所述至少一个协程调度线程中的 IO 密集型调度线程调度与所述连接请求对应的协程。

5、根据权利要求 4 所述的方法，其中，所述若检测到与所述连接请求对应的协程未处于第一状态与第二状态之间的状态，采用所述至少一个协程调度线程中的 IO 密集型调度线程调度与所述连接请求对应的协程包括：

响应于与所述连接请求对应的协程处于创建连接的阶段，将所述协程放置在当前的 IO 密集型调度线程，当前的 IO 密集型调度线程基于负载均衡算法确定；

响应于与所述连接请求对应的协程处于对特定文件的特定写入操作，且完成写特定文件 IO 请求的提交以及让出协程进入 IO 等待之前，将所述协程迁移至当前的 IO 密集型调度线程。

6、根据权利要求 5 所述的方法，其中，所述数据库为 MySQL 数据库，所述响应于与所述连接请求对应的协程处于对特定文件的特定写入操作，且完成写特定文件 IO 请求的提交以及让出协程进入 IO 等待之前，将所述协程迁移至当前的 IO 密集型调度线程包括：

响应于与所述连接请求对应的协程处于对二进制日志文件的写操作，且完成写文件 IO 请求的提交以及让出协程进入 IO 等待之前，将所述协程迁移至当前的 IO 密集型调度线程。

7、根据权利要求5所述的方法，其中，所述方法还包括：

检测所述IO密集型调度线程中是否具有驱动批量提交的协程；

若所述IO密集型调度线程中具有驱动批量提交的协程，将被该驱动批量提交协程唤醒的协程迁移至IO密集型调度线程，该驱动批量提交协程唤醒的协程迁移入的IO密集型调度线程基于负载均衡算法确定。

8、根据权利要求7所述的方法，其中，所述数据库为MySQL且开启组提交的数据库，所述检测所述IO密集型调度线程中是否具有驱动批量提交的协程包括：

检测所述IO密集型调度线程中是否具有对二进制日志文件进行写操作的驱动组提交过程的组领导协程；

所述IO密集型调度线程中具有驱动批量提交的协程，将被该协程唤醒的协程迁移至IO密集型调度线程包括：

若检测到具有对二进制日志文件进行写操作的组领导协程，将该组领导协程唤醒的组成员协程迁移至IO密集型调度线程，各个组成员协程迁移入的IO密集型调度线程基于负载均衡算法确定。

9、一种数据库线程池控制装置，所述装置包括：

加载单元，被配置成在数据库的操作系统中加载能够将线程替换为协程的动态运行库，生成至少一个协程调度线程，所述协程调度线程用于调度至少一个协程；

接收单元，被配置成接收客户端向所述数据库发送的连接请求；

检测单元，被配置成检测所述数据库创建新线程的请求是否为处理所述连接请求的请求；

创建单元，被配置成响应于检测到所述数据库创建新线程的请求为处理所述连接请求的请求，创建与所述连接请求对应的协程，并通过所述至少一个协程调度线程调度所述协程完成所述连接请求对应的访问任务。

10、根据权利要求9所述的装置，其中，所述加载单元被进一步配置成：

响应于数据库无法修改源代码且不具有连接管理插件机制时，启动数据库；

加载并初始化动态运行库，生成至少一个协程调度线程，动态运行库具有创建线程的钩子函数，创建线程的钩子函数用于采用协程替换数据库的原生运行库生成的连接处理线程；

在数据库初始化时，重载创建线程的钩子函数，以通过创建线程的钩子函数判断当前创建的线程是否为连接处理线程，在当前创建的线程是连接处理线程时，将该连接处理线程转换为协程。

11、根据权利要求 9 所述的装置，其中，所述加载单元被进一步配置成：

响应于数据库具有连接管理插件机制时，启动数据库；

加载并初始化动态运行库，生成至少一个协程调度线程；

初始化数据库；

加载并初始化插件，以使插件接收向数据库发送的连接请求，并数据库的线程构建请求为用于处理连接请求的请求时，控制动态运行库中的创建线程的钩子函数工作，创建线程的钩子函数用于判断当前创建的线程是否为连接处理线程，在当前创建的线程是连接处理线程时，将该连接处理线程转换为协程。

12、根据权利要求 9-10 之一所述的装置，其中，所述装置还包括：

划分单元，被配置成将至少一个协程调度线程划分为 IO 密集型调度线程和计算密集型调度线程；

所述创建单元进一步被配置成：检测与连接请求对应的协程是否处于第一状态与第二状态之间的状态，其中，第一状态为协程处于等待执行数据库命令的数据包，数据包已到达，且该协程需要被唤醒时的状态；第二状态为协程位于计算密集型调度线程，且该协程发起对特定文件的特定写入请求的状态；若检测到与连接请求对应的协程处于第一状态与第二状态之间的状态，采用至少一个协程调度线程中的计算密集型调度线程调度与连接请求对应的协程；若检测到与连接请求对应的协程未处于第一状态与第二状态之间的状态，采用至少一个协程调度线程中的 IO 密集型调度线程调度与连接请求对应的协程。

13、根据权利要求 12 所述的装置，其中，所述创建单元进一步被配置成：

响应于与连接请求对应的协程处于创建连接的阶段，将协程放置在当前的 IO 密集型调度线程，当前的 IO 密集型调度线程基于负载均衡算法确定；

响应于与连接请求对应的协程处于对特定文件的特定写入操作，且完成写特定文件 IO 请求的提交以及让出协程进入 IO 等待之前，将协程迁移至当前的 IO 密集型调度线程。

14、根据权利要求 13 所述的装置，其中，所述数据库为 MySQL 数据库，所述构建单元进一步被配置成：响应于与连接请求对应的协程处于对二进制日志文

件的写操作，且完成写文件 IO 请求的提交以及让出协程进入 IO 等待之前，将协程迁移至当前的 IO 密集型调度线程。

15、根据权利要求 13 所述的装置，其中，所述装置还包括：

批驱单元，被配置成：检测 IO 密集型调度线程中是否具有驱动批量提交的协程；若 IO 密集型调度线程中具有驱动批量提交的协程，将被该驱动批量提交协程唤醒的协程迁移至 IO 密集型调度线程，该驱动批量提交协程唤醒的协程迁移入的 IO 密集型调度线程基于负载均衡算法确定。

16、根据权利要求 15 所述的装置，其中，所述数据库为 MySQL 且开启组提交的数据库，所述批驱单元进一步被配置成：

检测 IO 密集型调度线程中是否具有对二进制日志文件进行写操作的驱动组提交过程的领导协程；

若检测到具有对二进制日志文件进行写操作的组领导协程，将该组领导协程唤醒的组成员协程迁移至 IO 密集型调度线程，各个组成员协程迁移入的 IO 密集型调度线程基于负载均衡算法确定。

17、一种电子设备，包括：

一个或多个处理器；

存储装置，其上存储有一个或多个计算机程序；

当所述一个或多个计算机程序被所述一个或多个处理器执行，使得所述一个或多个处理器实现如权利要求 1-8 中任一项所述的方法。

18、一种非瞬时性计算机可读介质，其上存储有计算机程序，其中，该计算机程序被处理器执行时实现如权利要求 1-8 中任一项所述的方法。