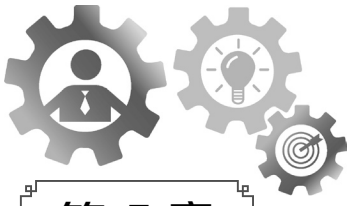

第 5 章

数据库实例优化

本章主要介绍在数据库实例层面进行性能分析的方法和策略，从数据库时间模型入手，分析数据库性能问题的主要瓶颈和可能采用的调整策略，并通过 TPCC 的性能测试演示对于该工作负载在数据库实例方面调优的整个过程。

5.1 性能分析概述

如果定位系统的性能问题是由数据库引起的，或用户响应时间的全链路只有数据库可优化，数据库的性能问题分析就显得尤为重要。

传统数据库通常都提供统计信息供用户分析，但各子系统的度量指标却各有不同而且丰富度不够。例如，数据缓冲区使用命中率度量，I/O 系统使用读写延迟度量、CPU 使用利用率度量。这些度量标准各自独立，缺乏统一的度量单位，使得难以从全局视角对性能问题进行分析，也很难量化不同子系统对整体系统性能的影响。

为了解决这一问题，KES 引入了时间模型，旨在明确各子系统对系统性能的影响。基于这一时间模型，KES 构建了一个多维度的分析框架，涵盖执行阶段、SQL 查询、数据库操作、报文传输等多个层面，帮助进一步定位性能问题的根本原因。图 5-1 给出了 KingbaseES 中每个数据库服务进程在处理 SQL 请求时的时间开销模型。

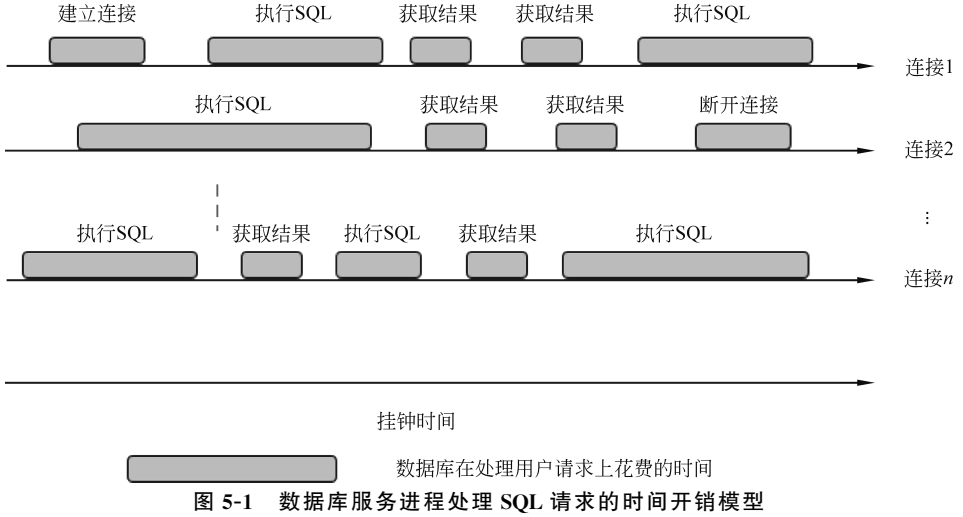

图 5-1 数据库服务进程处理 SQL 请求的时间开销模型

数据库的性能问题最终表现是用户感知到系统的处理变慢，每个请求需要更多的处理时间，因此分析数据库性能问题的核心思路是基于数据库时间模型做自顶向下的多维度时间拆解，看看时间到底花费在哪里，如图 5-2 所示。

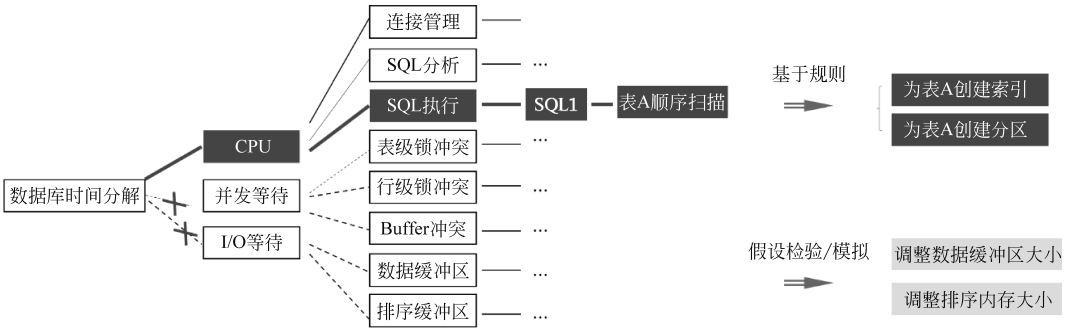

图 5-2 数据库性能分析的时间分解

数据库系统的服务进程在处理用户的请求时，要么 CPU 在工作，要么该服务进程由于不能获得所需要的资源而处于等待状态，因此首先看系统的数据库时间主要花费在 CPU 上还是花费在等待事件上，然后再进一步分析原因。第 4 章中介绍了数据库的性能诊断工具，我们从中可以获得这些信息。针对数据库的性能问题，分析思路一般如下。

(1) 查看动态性能视图或 KWR 中的等待时间，看是否存在等待较多的等待事件。可能的等待事件包括 I/O、网络、封锁等。

(2) 对于等待事件占比较高的(5%以上)情况，根据等待事件的含义，采取具体的动作。

(3) 如果没有明显的等待事件，则主要是 CPU 计算的问题，可从实例效率、慢 SQL、硬件与架构层面等方向进行分析与优化。

(4) 性能视图/KWR 等报告提供了大量的其他报告，可以进一步分析性能问题的原因。

5.2 等待事件瓶颈分析与优化

在 KingbaseES 中，等待事件是每个后台进程在执行过程中可能遇到的各种等待状态的标识。这些状态反映了数据库在处理查询时，进程由于等待某些资源、锁、I/O 操作等而被挂起的情况。

在性能分析任务中，等待事件可以发挥重要作用，它能帮助数据库管理员识别并诊断系统中的性能瓶颈。通过分析不同的等待事件，管理员可以针对性地优化锁竞争、I/O 性能、自动化任务以及应用程序的查询效率，从而提升数据库的整体性能和响应速度。

常见的等待事件包括轻量级锁、常规锁、I/O、缓冲区等待等。下面介绍一些常见的等待事件，分析其含义并探讨可能的优化方向。

5.2.1 I/O 等待事件分析与优化

磁盘 I/O 涉及机械操作，相对 CPU 是耗时的操作，对磁盘读写时会触发 I/O 等待事件。数据库服务进程需要进行读写的磁盘文件包括数据文件和临时文件，I/O 类等待事件分为两大类：数据文件的读写和临时文件的读写。

数据文件的读写的I/O等待事件是DataFileRead和DataFileWrite。KingbaseES数据库服务进程访问一个关系表或者索引数据页时，若数据页已经存在于共享缓冲区中，则直接返回该数据页；若不存在，则需要从存储中读取该数据页，这会触发DataFileRead等待事件。几乎所有的SQL语句在执行的过程中都可能触发DataFileRead或DataFileWrite等待事件。

数据文件的读写的I/O等待事件产生的主要原因是数据库服务进程需要进行太多的物理磁盘操作，进一步分析通常有以下几种情况。

(1) 共享缓冲区设置得过小，这样不能缓存更多的数据页面，需要到磁盘上物理读取所需要的页面。

(2) SQL语句在执行过程中需要访问大量的数据：

a. 访问表的方式是全表顺序扫描，而不是索引扫描；

b. 执行一些DDL语句，例如CREATE TABLE AS SELECT和CREATE INDEX，需要扫描全表的数据；

c. 执行vacuum或analyze语句时，需要扫描全表的数据。

针对这种情况，可以从操作系统和数据库两个层面调整I/O资源的使用。

操作系统层面可以调整磁盘调度算法。Linux操作系统常见的I/O调度策略包括以下几种。

(1) CFQ：该策略将I/O请求按进程进行排队，为每个进程分配公平的I/O带宽。在多用户、多任务的系统中，如果各个进程都有随机读写需求，cfq就能确保每个进程都能获得合理的I/O资源，防止某个进程独占I/O带宽，但对于单个进程的随机读写性能提升可能不如deadline策略。

(2) Deadline：该策略为每个I/O请求设置了最后期限，保证了请求能在一定时间内得到处理，避免了某些请求长时间等待。在数据库等对随机读写响应时间要求较高的场景中，该策略可以显著减少读写延迟，提高随机读写性能。

(3) NOOP：该策略仅对I/O请求进行简单排序，减少了调度的开销，对于随机读写频繁的场景，如使用固态硬盘(SSD)时，能让SSD的随机读写优势充分发挥，提升随机读写性能。因为SSD本身不存在机械寻道时间，所以简单的调度可以更快地响应随机请求。

对于机械磁盘来说，deadline是数据库系统的最佳选择，例如tpcc的工作负载通常采用deadline磁盘调度策略，固态硬盘一般不做调整。

使用操作系统命令可以查看和修改磁盘设备的I/O策略，例如：

```
--查看系统中的磁盘设备
[root@localhost ~]# lsblk
NAME            MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda               8:0    0   200G  0 disk
├─sda1            8:1    0   500M  0 part /boot
└─sda2            8:2    0 199.5G  0 part
  ├─centos-root 253:0    0    50G  0 lvm  /
  ├─centos-swap 253:1    0   3.9G  0 lvm  [SWAP]
  └─centos-home 253:2    0 145.6G  0 lvm  /home
sr0              11:0    1     4G  0 rom

--查看磁盘设备sda的I/O调度策略，输出中在[ ]内的策略是目前正在使用的策略
```

```
[root@localhost ~]#cat /sys/block/sda/queue/scheduler
noop deadline [cfq]

--修改磁盘设备 sda 的 I/O 调度策略
[root@localhost ~]#echo deadline > /sys/block/sda/queue/scheduler
[root@localhost ~]#cat /sys/block/sda/queue/scheduler
noop [deadline] cfq

[root@localhost ~]#
```

这是临时修改磁盘设备 I/O 调度策略的方式，如果操作系统重新启动，还会恢复原来的设置。如果需要永久修改该设置，可以修改/etc/default/grub 文件，找到 GRUB_CMDLINE_LINUX 这一行，在引号内添加 elevator=＜调度策略名＞。

在数据库系统层面可以做以下调整。

(1) 根据数据库系统的硬件配置情况，如果有可能，适当扩大共享缓冲区大小，可以缓存更多的数据页，以减少物理磁盘 I/O 的次数。在配置文件 Kingbase.conf 中重新设置配置参数 shared_buffer 的值，修改共享缓冲区大小，系统需要重新启动才能生效。

(2) 查看系统的 checkpoint 的设置是否合理。Checkpoint 事件会增加系统的 I/O 压力，但是当系统出现故障时，可缩短系统恢复的事件，根据自己的需求设置配置参数 checkpoint_timeout 来调整 Checkpoint 的执行频率。

(3) 把数据文件、日志文件部署到不同的 I/O 设备上，或者使用 RAID 设备，以此利用多个设备的 I/O 能力分担 I/O 压力，提升每次 I/O 的速度。

(4) 识别 DataFileRead 占比高的 SQL 语句，并运行 EXPLAIN 命令分析其输出的执行计划中的扫描类型是否包含顺序扫描，例如使用 sys_stat_sqlwait 视图识别顺序扫描高的关系表(sys_stat_all_tables)，分析包含该关系表的 SQL 语句执行计划是否合理。

(5) 分析执行时间长的 SQL 语句中 WHERE 子句包含的列是否使用了索引，如果没有，请考虑为此列创建索引。

如果等待事件 BufFileRead 和 BufFileWrite 的时间开销比较大，则说明数据库系统进行了大量的临时文件读写。数据库服务进程在处理 SQL 语句的过程中经常需要对大批量数据进行排序，例如 ORDER BY 子句、CREATE INDEX 语句等，如果可以使用的内存不够大，则需要采用外排序算法，将临时数据写入和读取磁盘文件而触发物理 I/O 操作。或者在进行大数据量表的哈希连接操作时，哈希表在内存中放不下，也会创建临时文件。SQL 语句在执行过程中如果需要创建大量的临时文件，在其执行计划中就可以看到这些操作。

由于数据库系统是一个多用户系统，单个服务进程如果占用过多的内存资源，就会影响其他服务进程的运行，因此 KingbaseES 使用系统参数 work_mem 控制每个服务进程可以使用的工作内存大小，默认是 4MB。适当扩大该参数可以增加内部排序和哈希计算的可用内存，这会改善性能。但是，由于每个服务进程都有各自的工作内存，如果该值设置得过大，在客户端会话数量很多的情况下，数据库实例会消耗非常大的内存，这容易引起内存不足的严重问题。对于这种情况，可以使用连接池控制并发的客户端连接数量。

CREATE INDEX 和 CLUSTER 语句在执行过程中需要对数据排序，它们在执行过程中可以使用的内存的大小由系统参数 maintenance_work_mem 设置。如果维护性操作仅在个别客户端连接上进行，可以在会话级设置按需扩大该值来改善性能。

5.2.2　轻量级锁等待事件分析与优化

KingbaseES 中的轻量级锁(LightLock)主要用于规避多个服务进程在访问共享的数据结构时的并发冲突,对于所有可能并发访问的共享内存结构,一般都由轻量级锁进行保护。轻量级锁分读写两种,对共享内存结构的读读操作不冲突,对读写和写写操作冲突。轻量级锁的加锁和放锁由 KingbaseES 根据访问请求内部控制。如果轻量级锁的等待事件占用了过长时间,则说明对内存共享结构的访问是热点,需要进一步分析其原因。

KingbaseES 中通常容易形成热点的轻量级锁包括 buffer_content、lock_manager、wal_insert、WALWriteLock 等,下面分别详细介绍。

1. buffer_content

轻量级锁 buffer_content 用于保护对数据缓冲区中数据页面的访问,包括表数据页面和索引数据页面。服务进程对数据页的读写操作一般通过共享缓冲区,读数据页之前会对目标数据页加上共享锁,写数据页之前需要加上独占锁。数据页加了共享锁时,多个读请求仍然可以访问该数据页。但是,一旦加了独占锁,则其他读写请求都不能访问该数据页,直到独占锁被释放。

当某个服务进程需要读取或写入共享数据缓冲区中的某个页面,而另一个服务进程正锁定该页面以进行内容更改时,会发生 buffer_content 等待事件。当要查询的数据页不在共享缓冲区时,服务进程需要更多的时间进行磁盘 I/O,将数据页加载到内存中,这会导致试图读取该数据页的其他服务进程等待更长的时间,尤其是较多的并发更新相同的缓冲区数据页容易触发 buffer_content 等待事件,在有大量索引的关系表情况下,因为一个索引页面对应数个数据页面,对于索引页面的访问其冲突相对数据页面会放大。大量的外键约束在读取目标数据页的时候,可能还需要检查外键约束引用的数据页,这会增加额外的数据页锁定操作。

如果观察到轻量级锁 buffer_content 等待事件的时间比较长,根据对应的 SQL 语句判断原因可能如下。

(1) 同时并发访问比较大的表,在索引页面上引起 buffer_content 等待事件,可以考虑进行分区,索引也自然分区,让关键热点数据页尽量分散。

(2) 删除不必要的索引。

2. lock_manager

轻量级锁 lock_manager 用于保护共享内存中的锁(常规锁)表结构。当数据库服务进程访问某个数据库对象时,为了保证事务处理的正确性,会在需要访问的数据库对象上加锁,事务结束时放锁,这些锁的信息都保存在共享内存中的锁表结构中,所有的加锁和放锁操作都会访问锁表结构,而必须持有轻量级锁 lock_manager,出现并发冲突时就会出现 lock_manager 等待事件。

lock_manager 等待事件多发生在以下几种情况:

1) 高并发

并发的 DDL 语句比较多或者长事务比较多,同一事务访问了过多的数据库对象,导致 KingbaseES 中对于封锁的优化措施不能使用,造成 lock_manager 等待事件。

建议调整应用程序的逻辑，尽可能减少长事务。也可以使用连接池，控制连接数量，减少并发度，但要避免可能发生的连接风暴。

2）表有大量分区

当访问具有大量分区的表时，在查询优化和执行阶段都可能打开每个分区表，导致表锁申请的次数异常多，触发 lock_manager 等待事件。对于这种情况，可以参考 10.2.5 节中对分区表的优化措施，调整系统参数 partition_table_limit 的值，减少查询优化期间的系统冲突。

3）临时表的不合理使用

当大量并发的服务进程频繁执行创建、删除表操作（临时表），也可能导致 lock_manager 等待事件。如果是这种情况，需要检查应用的逻辑，尽量规避该情况。

如果 lock_manager 排在主要等待事件列表中的第一个或第二个，请检查列表中是否也显示了 Lock 分类的等待事件 Relation、transactionid 和 tuple，如果前面的事件显示在列表的前面，可以考虑首先优化这些等待事件，这些事件可能是 lock_manager 的驱动因素。

3. wal_insert 和 WALWriteLock

wal_insert 和 WALWriteLock 是与数据库事务日志相关的等待事件。事务日志是数据库实现事务的原子性和持久性的核心机制，遵循“日志优先写（WAL）”的原则。任何对数据库中常规数据的修改，都必须生成相应的日志记录，日志记录先写入内存中的日志缓冲区，当事务提交或数据页面写入磁盘时，日志缓冲区中对应的日志记录必须写到磁盘上的日志文件中。轻量级锁 wal_insert 用于保护对日志缓冲区的并发写入，WALWriteLock 用于保护对日志文件的并发写。

wal_insert 等待事件通常在以下几种情况下发生。

（1）多个并发的服务进程需要同时向日志缓冲区写日志记录；

（2）日志缓冲区（wal_buffers）满或者事务提交需要同步把日志缓冲区的内容写入磁盘上的日志文件，这时插入 WAL 记录可能需要等待 I/O 完成，导致较长的 wal_insert 等待。

当发现 wal_insert 等待事件占用时间占比较高时，可以开启 KingbaseES 提供的向日志缓冲区写日志记录的无锁化优化，将系统配置参数 enable_xlog_insert_lock_free 设置为 on，然后重启数据库，可以消除 wal_insert 等待事件，实现性能的提升。但是，该参数要求操作系统支持 128 位原子操作，并不是所有的平台都支持。

WALWriteLock 等待事件通常是因为产生了巨大的日志量，或者进行了大量的小事务时，每次提交动作触发日志写操作，如果磁盘的 I/O 压力很大，则更加剧了 WALWriteLock 等待事件的时间。减少该等待事件的手段通常是减少日志数量或减少日志写盘的次数或提升日志写盘速度，因此可以采用以下的优化方法。

（1）适当增加日志缓冲区的大小，对于一些长事务，可以减少日志写盘的次数。例如，修改系统配置参数 wal_buffers = 512M。

（2）对于并发执行的事务比较多的情况，可以使用成组提交的方式降低日志写盘的次数。设置下面的系统配置参数：

```
commit_delay = 10
commit_siblings = 5
```

表示每次事务提交时，且当前至少有5个(不含自己)活跃事务时，需等待10微秒，以实现多个事务同时提交。

(3) 根据业务逻辑的情况设置参数，减少系统的日志量、日志的写盘方式、事务的同步方式等，参见2.3.2节中日志系统的配置说明。

WALWriteLock等待事件时间偏高表示事务提交繁忙，还有可能是日志文件所在的磁盘性能不足，导致日志刷盘时持有锁过久。可以考虑将日志文件目录切换到性能更好的磁盘上。若已安装完成，可以在停库的情况下，首先在高性能磁盘上创建日志文件的目录sys_wal，然后将原目录sys_wal下的wal日志复制过去，最后在原位置创建软连接即可。

5.2.3 常规锁等待事件分析与优化

常规锁通常用于保护数据库对象的并发访问，以实现正确的事务处理。常规锁通常由KingbaseES在处理SQL语句的过程中自动加锁和放锁，用户不需要干预，在一些特殊的情况下，用户也可以使用LOCK语句直接对数据库中的表加锁。

容易发生的常规锁等待事件包括以下几种。

1. relation

relation等待事件表示正等待获得一个表或索引上的锁。KingbaseES的服务进程在处理SQL语句的过程中访问每个表或索引时，首先在表或索引上加锁，根据不同的操作加相应类型的锁，在事务提交时释放锁。2.2.2节中给出了数据库中锁的使用方式以及可能发生的锁冲突，当对同一个表或索引加锁时出现冲突，就会发生relation等待事件。

下面是常见的发生relation等待事件的情况。

(1) 同一个表上的DDL语句与增加、删除、修改、查询语句通常不能并发执行。例如，有些ALTER TABLE语句会影响DML和查询语句的执行。CREATE INDEX语句也会阻塞DML和查询语句的执行，使用concurrently选项，并发创建索引，可能执行时间更长。

(2) 系统的维护操作与日常的工作负载也会冲突。例如，在表上执行VACUUM语句回收表中的空闲空间，如果使用FULL选项，会阻塞该表上几乎所有的操作，因此系统的维护操作可以放在工作负载比较低的时间段，维护操作尽可能使用封锁强度比较低的选项，例如日常使用不带FULL选项的VACUUM语句，根据需要再使用FULL选项。

(3) 数据库中的锁通常在事务结束时释放锁，因此如果系统中有非常多的长事务或没有提交的空闲事务，就会增加relation等待事件的时间。

(4) 主备物理复制集群，备机读取事务长期不结束，relation锁长期持有，导致删除表操作触发relation等待事件。

2. transactionid

transactionid等待事件常见于并发的服务进程对同一个元组操作的情况，例如并发更新同一个元组，这时后来的服务进程就要等待正在操作该元组的事务结束，触发transactionid等待事件。在KingbaseES中，通常对同一个元组并发的读写操作并不冲突，只对并发的写写操作才冲突，但是当查询语句中有FOR UPDATE、FOR KEY SHARE等选项时，与对该元组的更新操作是冲突的。

无论是relation还是transactionid等待事件，发生的根本原因都是并发冲突，因此优化

这种情况的基本思路是减少冲突的机会或减少冲突的时间。

为了减少阻塞 SQL 语句的影响，可以采用下面的手段进行优化。

(1) 使用配置参数 lock_timeout 设置超时，以限制 SQL 语句等待获取关系锁的时间。如果未在指定的超时内获取锁定，则请求锁定的事务将被取消，可以在会话级别设置这些参数的值。

(2) 中断系统中空闲没有提交的事务。

在应用程序层面可以采用下面的优化方式：

(1) 在保证应用逻辑正确的前提下，一些 SQL 语句可以使用 NOWAIT 选项，如果无法立即获取锁定，则 NOWAIT 指令将取消请求锁定的查询；

(2) 设计应用程序或数据模型以避免更新语句与 SELECT … FOR UPDATE 语句冲突；

(3) 使用连接池以降低应用程序中的并发率；

(4) 在保证应用逻辑正确的前提下，不要使用太长的事务。

3. extend

在 KingbaseES 中引发封锁等待事件，除上面提到的用于事务处理的锁外，还有一个 extend 锁等待事件，该等待事件表示正等待扩展一个关系。KingbaseES 采用操作系统管理数据文件的磁盘空间使用情况，每个表或索引对应一个或多个操作系统的文件，当表或索引需要更多的磁盘空间存放数据时，就会扩展其数据文件(extend)的大小，为了规避多个服务进程同时扩展数据文件，则需要加 extend 锁，因此系统会生成 extend 等待事件。INSERT、COPY 和 UPDATE 的并发操作可以生成此事件。

extend 等待事件通常出现在磁盘性能略差，或者存储服务器的带宽不足的环境中，如果该等待事件占用的时间比较多，则可以采用下面的优化策略。

(1) 通过批量异步扩展，降低操作磁盘的次数和延迟，提升性能：

a. 系统参数 max_extend_num 代表单次扩展的最大页面数。

b. 参数 wait_extend_lock_factor 代表为等待者扩展页面数的系数，单次扩展页面数由等待者数 * wait_extend_lock_factor 计算生成。

c. 增大 max_extend_num 和 wait_extend_lock_factor 的值尝试解决该问题。

(2) 升级磁盘或升级连接磁盘网络的带宽。

5.2.4 通信等待事件分析与优化

KingbaseES 采用客户端/服务器的体系结构，客户端/服务器通过 TCP/IP 通信。数据库应用程序通过数据库编程接口(如 JDBC 等)访问数据库。

通信类等待事件包括 ClientRead 和 ClientWrite。ClientRead 等待事件表示服务进程正等待从客户端读取数据，ClientWrite 表示服务进程正等待向客户端发送数据，出现该类等待事件通常很可能网络负载会比较高。

如果通信类等待事件占用了过多的事件，则可以考虑从以下几个角度进行优化。

(1) 检查和优化网络连接是否正常，使用工具诊断网络延迟，并确保网络连接稳定。考虑使用更高效的网络配置，例如增加带宽或减少网络跳数以降低延迟。

(2) 监控客户端的 CPU 和内存使用情况，确保客户端有足够的资源处理数据传输。可

以通过调整客户端应用程序的负载均衡策略或优化代码缓解高负载问题。

(3) 如果需要传输大量数据(如大型结果集),可考虑是否通过分批读取或写入减轻客户端和服务器之间的压力。例如,在接口端配置 defaultRowFetchSize,使用批量返回模式。需要注意的是,对于应用,确实需要较多的返回数据,批量返回的模式可能带来额外的开销。

(4) 在应用层修改 SQL 语句或者逻辑,根据实际应用需求,选择通过 LIMIT 等分页方式限制返回行数。

(5) 错峰执行占用网络资源较多的语句或者操作。

5.3　CPU 瓶颈分析与优化

当数据库系统没有明显等待事件时,说明影响数据库性能的主要问题在于 CPU 的计算瓶颈。对于 CPU 的瓶颈,可以从 SQL 语句的优化开始,按照 Top SQL 时间自高向低分析,选择更优的执行计划,以减少对数据库系统资源的消耗。有关 SQL 语句的优化技术后面的章节中有详细描述。在数据库实例层面可以采用的优化手段包括绑核技术和使用执行计划缓存。如果软件算法优化已经无法解决问题,则需要通过增加硬件的算力或者在架构层面进行优化。

5.3.1　绑核技术

绑核技术是指将 KingbaseES 的服务进程绑定在固定的 CPU 核号上,可以增加 CPU 的 cache 命中率,以此提升性能指标。

KingbaseES 提供了系统配置参数 bindcpulist,用来指定如何将服务进程与 CPU 核绑定。例如:

(1) bindcpulist ='0-95'表示将数据库服务进程绑定到 0 到 95 号 CPU 核心。

(2) bindcpulist ='0-3,10,20-30,50',表示将数据库服务进程绑定到 0 到 3 号核、10 号核、20 到 30 号核、50 号 CPU 核心上。

创建 KingbaseES 服务进程时,根据此参数给出的 CPU 核号列表信息,将当前服务进程绑定在一个固定的 CPU 核号上。绑核是按参数列表顺序进行的。

5.3.2　使用执行计划缓存

OLTP 类型的工作负载经常使用大量的不太复杂的 SQL 语句,执行非常快,但是 SQL 语句的编译阶段生成执行计划在总时间的占比较高,如果将之前生成的执行计划缓存起来,下次接收到相同的 SQL 语句时,直接执行缓存的执行计划,而不重新计算生成,可以有效降低数据库响应时间,从而提高系统的效率。

KingbaseES 支持执行计划缓存来提升系统的性能。数据库服务器收到客户端发送的 SQL 语句后,进行查询编译,生成执行计划,然后把执行计划缓存到共享内存中,缓存的 SQL 执行计划通过根据 SQL 语句计算出的 SQLID 标识。KingbaseES 中后面的需要执行的 SQL 语句如果可以匹配,则可以使用缓存的执行计划执行,从而节省了生成执行计划的时间开销。

KingbaseES 使用系统参数 simple_plan_cache_mode 控制执行计划缓存的行为,该参

数用于设置是否开启执行计划的缓存，具体有下面 3 个取值。

(1) OFF：不开启执行计划缓存，是默认值；

(2) EXACT：数据库服务器根据 SQL 语句字符串生成 SQLID，因此只有 SQL 语句完全相同，才可以匹配到缓存中的执行计划；

(3) FORCE：数据库服务器对 SQL 语句进行词法分析，将 SQL 常量变为参数，同时过滤掉空格、注释等干扰元素，重新生成 SQL 语句字符串，并使用该字符串 SQLID。

重新设置该参数时，需要重新启动数据库服务器。

当 simple_plan_cache_mode 参数的取值为 FORCE 时，有以下几个特点。

(1) 对于只有常量不同的 SQL 语句也是可以匹配的，这样匹配的范围更大，但是在做执行计划缓存时需要做更多的工作。

(2) 把 SQL 语句中的常量变成参数后生成执行计划，可能存在的问题是：对于不同的参数值，该执行计划有可能不太合适，执行起来会耗费更多的资源。为了解决这个问题，KingbaseES 并不是缓存每个 SQL 的执行计划，而是评估该执行计划是否合理，如果它适用于很多参数值，则认为是一个合适的执行计划。KingbaseES 设置系统参数 simple_plan_cache_custom 为大于或等于 0 的整数，默认值为 5，即对于同一个 SQL 语句，前 5 次生成的执行计划不缓存，根据前 5 次执行计划中估算的执行代价计算一个平均值，后面再生成的执行计划中估算的执行代价不大于该值才缓存，否则不缓存该 SQL 语句的执行计划，通过这种方式评估执行计划的合理性。

(3) 如果 SQL 语句中包含临时表、interval、Coalesc 关键字或查询条件中包含 LIKE 等查询结果不稳定的情况，则不会缓存该 SQL 语句的执行计划。

当用户不希望缓存一些 SQL 语句的执行计划时，可以配置 simple_plan_cache_notcache_sql 指定这些 SQL 的列表。该参数的取值为以分号分割的 SQL 字符串。因为 KingbaseES 采用了模糊匹配的方式，所以对每个 SQL 可以只写出 SQL 的一部分。

使用函数 get_plan_cache()显示所有的缓存执行计划的 SQL 的列表：

```
select * from get_plan_cache();
```

返回的结果集包括以下信息：①sqlid 是根据 SQL 字符串生成的 HASH 值；②sqlowner 和 sqldb 对应用户 ID 和数据库 ID；③usesize 和 usecount 是该缓存计划占用的共享内存空间和重用的次数；④sqltext 是 SQL 语句的文本；⑤costs 是缓存的执行计划代价，当没有缓存计划时，代价为－1。

KingbaseES 的 SQL 语句执行计划缓存在共享内存中，也是耗费系统资源的。系统配置参数 simple_plan_cache_size 控制支持的缓存执行计划的数量，取值为大于或等于 0 的整数，默认值为 1000，即最多缓存 1000 个执行计划。

当 SQL 语句涉及的对象(如表的定义、函数的定义、统计信息等)发生改变时，系统会自动删除其对应的执行计划。用户也可以通过系统提供的函数管理执行计划。

(1) SELECT remove_all_plan_cache()：删除所有缓存的执行计划；

(2) SELECT remove_plan_cache(sqlid,sqlowner,sqldb)：删除指定的缓存执行计划，其中，sqlid 为根据 SQL 字符串计算出的整数值，sqlowner 为数据库用户 OID，sqldb 为数据库 OID，这 3 个值都可以为 NULL。