POSTGRESQL中子事务以及性能分析

作者：Laurenz Albe译者: 陈雁飞

**作者介绍**

**Laurenz Albe**：Cybertec的高级顾问和支持工程师。自2006年以来，一直与PostgreSQL合作并为其做出贡献。

**译者简介**

**陈雁飞**：开源PostgreSQL爱好者，一直从事PostgreSQL数据库运维工作

最近，在排查PostgreSQL性能问题的时候，两次遇到子事务相关问题。所以，我想这个话题非常适合作为博客内容。

## 什么是子事务？

每个人都了解数据库事务。在PostgreSQL中，事务是默认工作在自动提交模式下，多语句情况下，需要显示调用BEGIN或者START TRANSACTION来开启一个事务，最后使用END或者COMMIT结束它。如果用ROLLBACK中断一个事务（或者数据库会话结束的时候没有执行提交操作），那么在事务中的操作将成为没有完成的。

现在子事务允许你回滚部分已经在事务中完成的工作。可以使用下面的标准语句在一个事务中开启子事务：

SAVEPOINT name;

“name”表示一个子事务的标识符（没有单引号！）。不能在SQL中提交一个子事务（将和包含它的事务一起自动提交），但是可以使用下面的命令回滚：

ROLLBACK TO SAVEPOINT name;

## 子事务的使用

子事务在长事务中有非常大的作用。在PostgreSQL中，事务中任何一个错误都会中断整个事务：

|  |
| --- |
| test=> BEGIN;  BEGIN  test=\*> SELECT 'Some work is done';       ?column?  -------------------   Some work is done  (1 row)    test=\*> SELECT 12 / (factorial(0) - 1);  ERROR:  division by zero  test=!> SELECT 'try to do more work';  ERROR:  current transaction is aborted, commands ignored until end of transaction block  test=!> COMMIT;  ROLLBACK |

对于一个做了很多工作的事务来说，这是非常烦人的，因为这意味着失去到目前为止完成的所有工作。子事务可以帮助我们从这种情况中进行恢复

|  |
| --- |
| test=> BEGIN;  BEGIN  test=\*> SELECT 'Some work is done';       ?column?  -------------------   Some work is done  (1 row)    test=\*> SAVEPOINT a;  SAVEPOINT  test=\*> SELECT 12 / (factorial(0) - 1);  ERROR:  division by zero  test=!> ROLLBACK TO SAVEPOINT a;  ROLLBACK  test=\*> SELECT 'try to do more work';        ?column?  ---------------------   try to do more work  (1 row)    test=\*> COMMIT;  COMMIT |

注意ROLLBACK TO SAVEPOINT回滚一个旧事务a的时候，会重新开始一个新的子事务。

## PL/pgSQL中子事务

即使你从来没有使用过SAVEPOINT语句，但是你可能遇到过子事务。在PL/pgSQL中，上面的代码类似下面

|  |
| --- |
| BEGIN     PERFORM 'Some work is done';       BEGIN  -- a block inside a block        PERFORM 12 / (factorial(0) - 1);     EXCEPTION        WHEN division\_by\_zero THEN           NULL;  -- ignore the error     END;       PERFORM 'try to do more work';  END; |

每次输入带有EXCEPTION子句的语句块时，都会开启一个新的子事务。当离开这个块的时候会提交该子事务，进入异常处理分支的时候表示回滚。

## 数据库之间兼容性

其它数据库处理事务中错误的方式不尽相同。不会中止完整的事务，而是仅仅回滚导致错误的语句，从而使事务本身处于活动状态。

当从这样的数据库迁移或移植到PostgreSQL中时，你可能需要在子事务中包装每个语句，以模拟上面的行为。

PostgreSQL JDBC驱动程序中有一个连接参数“autosave”，如果将其设置为“always”，就会在每条语句之前自动设置一个保存点，方便在失败的时候回滚。

如下所示，这种转换技巧存在严重的性能瓶颈。

## 性能测试用例

为了说明由于过度使用子事务导致性能问题，创建下面的测试用例表

|  |
| --- |
| CREATE UNLOGGED TABLE contend (      id integer PRIMARY KEY,      val integer NOT NULL  )  WITH (fillfactor='50');    INSERT INTO contend (id, val)  SELECT i, 0  FROM generate\_series(1, 10000) AS i;    VACUUM (ANALYZE) contend; |

这个表数据量很少、不记录日志以及低的填充因子，这些都是为了尽可能降低I/O。这样，可以更好地观察子事务的影响。

我将使用pgbench（一个PostgreSQL附带的基准测试工具）来运行下面的自定义SQL脚本。

|  |
| --- |
| BEGIN;  PREPARE sel(integer) AS     SELECT count(\*)     FROM contend     WHERE id BETWEEN $1 AND $1 + 100;  PREPARE upd(integer) AS     UPDATE contend SET val = val + 1     WHERE id IN ($1, $1 + 10, $1 + 20, $1 + 30);    SAVEPOINT a;  \set rnd random(1,990)  EXECUTE sel(10 \* :rnd + :client\_id + 1);  EXECUTE upd(10 \* :rnd + :client\_id);    SAVEPOINT a;  \set rnd random(1,990)  EXECUTE sel(10 \* :rnd + :client\_id + 1);  EXECUTE upd(10 \* :rnd + :client\_id);    ...    SAVEPOINT a;  \set rnd random(1,990)  EXECUTE sel(10 \* :rnd + :client\_id + 1);  EXECUTE upd(10 \* :rnd + :client\_id);    DEALLOCATE ALL;  COMMIT; |

第一组测试用例将设置60个子事务，第二组测试用例将设置90个子事务。通过使用预备语句方式尽可能减少查询解析的影响。

在每个数据库会话中，pgbench将:client\_id替换成一个唯一的数字。所以只要没有不超过10个客户端，每个客户端的更新操作不会产生冲突，但是会查询其他客户端产生的数据行。

## 性能测试

由于机器只有8核，因此在测试中将使用6个并发客户端运行十分钟。

为了让“perf top”能查看到重要的信息，需要安装PostgreSQL调试符号信息。这在生产系统上也是推荐的。

**TEST 1（60个子事务）**

|  |
| --- |
| pgbench -f subtrans.sql -n -c 6 -T 600    transaction type: subtrans.sql  scaling factor: 1  query mode: simple  number of clients: 6  number of threads: 1  duration: 600 s  number of transactions actually processed: 100434  latency average = 35.846 ms  tps = 167.382164 (including connections establishing)  tps = 167.383187 (excluding connections establishing) |

下面是在测试运行中，使用“perf top --no-children --call-graph=fp --dsos=/usr/pgsql-12/bin/postgres”命令展示的信息

|  |
| --- |
| +    1.86%  [.] tbm\_iterate  +    1.77%  [.] hash\_search\_with\_hash\_value  1.75%  [.] AllocSetAlloc  +    1.36%  [.] pg\_qsort  +    1.12%  [.] base\_yyparse  +    1.10%  [.] TransactionIdIsCurrentTransactionId  +    0.96%  [.] heap\_hot\_search\_buffer  +    0.96%  [.] LWLockAttemptLock  +    0.85%  [.] HeapTupleSatisfiesVisibility  +    0.82%  [.] heap\_page\_prune  +    0.81%  [.] ExecInterpExpr  +    0.80%  [.] SearchCatCache1  +    0.79%  [.] BitmapHeapNext  +    0.64%  [.] LWLockRelease  +    0.62%  [.] MemoryContextAllocZeroAligned  +    0.55%  [.]\_bt\_checkkeys      0.54%  [.] hash\_any  +    0.52%  [.] \_bt\_compare       0.51%  [.] ExecScan |

Test2（90个子事务）

|  |
| --- |
| pgbench -f subtrans.sql -n -c 6 -T 600    transaction type: subtrans.sql  scaling factor: 1  query mode: simple  number of clients: 6  number of threads: 1  duration: 600 s  number of transactions actually processed: 41400  latency average = 86.965 ms  tps = 68.993634 (including connections establishing)  tps = 68.993993 (excluding connections establishing) |

下面是命令“perf top --no-children --call-graph=fp --dsos=/usr/pgsql-12/bin/postgres”得到的内容

|  |
| --- |
| +   10.59%  [.] LWLockAttemptLock  +    7.12%  [.] LWLockRelease  +    2.70%  [.] LWLockAcquire  +    2.40%  [.] SimpleLruReadPage\_ReadOnly  +    1.30%  [.] TransactionIdIsCurrentTransactionId  +    1.26%  [.] tbm\_iterate  +    1.22%  [.] hash\_search\_with\_hash\_value  +    1.08%  [.] AllocSetAlloc  +    0.77%  [.] heap\_hot\_search\_buffer  +    0.72%  [.] pg\_qsort  +    0.72%  [.] base\_yyparse  +    0.66%  [.] SubTransGetParent  +    0.62%  [.] HeapTupleSatisfiesVisibility  +    0.54%  [.] ExecInterpExpr  +    0.51%  [.] SearchCatCache1 |

即使考虑到test2都是长事务，与test1相比，仍然有60%性能差距。

## 子事务实现

要了解发生什么，我们需要了解事务和子事务实现方式。

当一个事务或者子事务中修改了数据后，会为该事务分配一个事务ID（transaction ID）。PostgreSQL在提交日志（commit log）中跟踪这些事务ID信息，日志信息持久化存储在数据目录下pg\_xact子目录中。

但是，事务和子事务之间有下面几点差异：

* 每个子事务包含一个事务或者子事务（“父亲”）
* 提交子事务不会刷新WAL
* 一个数据库会话中有且只能有一个事务，但是可以有多个子事务

存储给定子事务的父信息相关的（子）事务信息持久化存储在数据目录下的pg\_subtrans子目录。由于这些信息随着包含事务结束后立即变成过去时，因此不必在关闭或者崩溃期间保留这些数据。

## 子事务和可见性

PostgreSQL中行级版本（元组）可见性由xmin和xmax系统列决定的，分别表示创建和删除事务的事务ID。如果存储的事务ID是子事务信息，那么PostgreSQL还必须查询包含（子）事务的状态，以确定对该事务ID是否可见。

为了确定语句可以看到哪些元组，PostgreSQL在语句（或事务）开始的地方首先获取数据库的快照信息。快照主要包含如下信息：

* 最大事务ID：任何超过该事务ID都是不可见的
* 获取快照的时候处于活跃状态的事务和子事务
* 当前（子）事务中可见的最早命令号（commnad number）

快照通过查询进程数组（process array）信息来进行初始化，进程数组保存在共享内存中并包含有当前运行进程的相关信息。当前，它也包含后端进程的当前事务ID，并且每个会话最多可以容纳64个未中止的子事务。如果有超过64个这样的子事务，那么快照被标记为子事务溢出（suboverflowed）。

## 结果分析

一个子溢出的快照不会包含检测可见性的所有数据信息，所以PostgreSQL有时将不得不求助于pg\_subtrans。这些页缓存在共享内存中，但是在perf中可以看到SimpleLruReadPage\_ReadOnly函数排在前面输出。其它事务必须更新pg\_subtrans后才能注册子事务，可以在perf输出中看到如何与读进程争夺轻量级锁。

## 分析子事务太多问题

除了查看”perf top”，还有其它指向该问题方向的可疑点：

* 运行单个进程的时候负载表现很好，但是并发多个数据库会话后会变高
* 在pg\_stat\_activity视图中经常看到等待实践“SubtransControlLock”
* 如果使用“pg\_export\_snapshot()”函数导出快照信息，数据目录下的pg\_snapshots 子目录保存的结果文件中包含有“sof:1”信息，其表示子事务数组溢出

## 结论

子事务是一个很好的工具，但是需要合理使用它。如果需要并发，每个事务不要启动超过64个子事务。

本文中提供的分析方法应该可以帮助你确定是否存在类似问题。

找到问题的根因可能很棘手。例如：对于SQL语句的每个结果行（可能在触发器中）调用的带有异常处理程序的函数，启动新的子事务不会那么明显。

**原文地址**

<https://www.cybertec-postgresql.com/en/subtransactions-and-performance-in-postgresql/>