文章编号: 1000-5641(2016)05-0144-09

面向 OceanBase 的存储过程设计与实现

祝 君, 刘柏众, 余晟隽, 官学庆, 周敏奇

(华东师范大学 数据科学与工程研究院, 上海 200062)

摘要: 存储过程是数据库管理系统的一个重要特性, 它是标准结构化查询语言(Structured Query Language, SQL)的一个扩展. OceanBase是一个新型的支持海量数据处理的分布式数据库系统, 但现有OceanBase的开源版本不支持存储过程功能, 影响了该系统在企业和机构中的推广和应用. 本文在深度分析存储过程原理以及OceanBase查询处理策略的基础上, 设计并实现了支持PL/SQL(Procedural Language/SQL)的存储过程机制.

关键词: 存储过程; SQL; OceanBase

中图分类号: TP392 **文献标识码**: A **DOI:** 10.3969/j.issn.1000-5641.2016.05.016

Designs and implementations of stored procedure in OceanBase

ZHU Jun, LIU Bai-zhong, YU Sheng-jun, GONG Xue-qing, ZHOU Min-qi

(*Institute for Data Science and Engineering, East China Normal University, Shanghai* 200062, *China*)

Abstract: As an extension of standard SQL(Structured Query Language), the stored procedure is an important feature in modern databases. OceanBase is a new type of distributed database system which supports massive data processing, but the open-sourced version OceanBase does not support stored procedure, which influences its adoption in enterprises. In this paper, we analyze the principle of stored procedure and the query processing mechinism of OceanBase in detail. Then, the complete design and implementation of stored procedure which supports PL/SQL are presented.

Key words: stored procedure; SQL; OceanBase

0 引 言

随着大数据时代和互联网时代的到来, 信息量呈井喷式爆发, 传统的集中式数据库管理系统难以处理如此巨大的数据. 在集中式系统架构下, 尽管可以依据业务特点对数据库进行拆分, 实现数据表的水平切分/垂直切分并存储到不同的数据库服务器上, 但随着业务和数据量不断增长, 需要不断地增加服务器以实现更细粒度的数据表切分, 这种方法需要大量的

收稿日期: 2016-05

基金项目: 国家自然科学基金(61332006); 国家863计划项目(2015AA015307)

第一作者: 祝 君, 男, 硕士研究生, 研究方向为分布式数据库. E-mail: zhujunxxxxx@163.com.

通信作者: 周敏奇, 男, 副教授, 硕士生导师, 研究方向为内存数据库. E-mail: mqzhou@sei.ecnu.edu.cn.

人工维护成本. 因此很多企业将目光转向了逐渐成熟的分布式数据库系统, 并利用其良好的可扩展性和容错性等来满足存储海量数据的应用需求. 近几年, 随着分布式数据库的快速发展, 各种分布式数据库如雨后春笋般出现, 如 Bigtable[1]、Spanner[2]、VoltDB[3],以及阿里巴巴(Alibaba)的 OceanBase[4]等. 虽然这些系统多数拥有存储和管理海量数据的能力, 但是在数据一致性、事务处理能力、SQL[5]功能上, 同传统的关系数据库系统相比还有差距, 很难被直接应用于银行业务系统等传统的大型信息系统中, 制约了分布式数据库在企业和机构中的推广和使用. 但是越来越多的企业开始关注于增强分布式数据系统功能, 如增加 SQL 支持、事务、二级索引、一致性等.

OceanBase 是 Alibaba 研发的关系型分布式数据库, 它实现了关系数据库的重要特征, 也支持 SQL 查询; 但是和主流的关系型数据库系统 PostgreSQL[6-7]、MySQL[8]等相比较, 功能上还存在一些不足的地方, 如不支持存储过程[9]和游标等. 存储过程在现代企业中应用十分广泛, 大多数企业的业务逻辑都采用存储过程实现, 而 OceanBase 作为关系型数据库想要在企业和机构中被广泛应用, 就需要支持存储过程, 以便企业和机构易于将存储过程所写的业务迁移到 OceanBase 数据库系统.

本文通过深入分析主流的开源数据库管理系统的存储过程功能, 对实现中的一些关键技术问题进行研究, 并结合 OceanBase 架构和其现有的查询引擎, 提出一种适合 OceanBase 数据库架构的存储过程设计和实现方案. 本文安排如下：第 1 节介绍 OceanBase 架构和存储过程; 第 2 节介绍 OceanBase 存储过程机制; 第3节介绍 PL/SQL[10]引擎架构; 第 4 节对 OceanBase 存储过程进行实验; 第 5 节总结全文.

1　背　　景

1.1　OceanBase概述

OceanBase 整机架构[11]分为 4 个模块：主控服务器 RootServer; 更新服务器 UpdateServer; 基线数据服务器 ChunkServer; 合并服务器 MergeServer. 如图 1 所示.

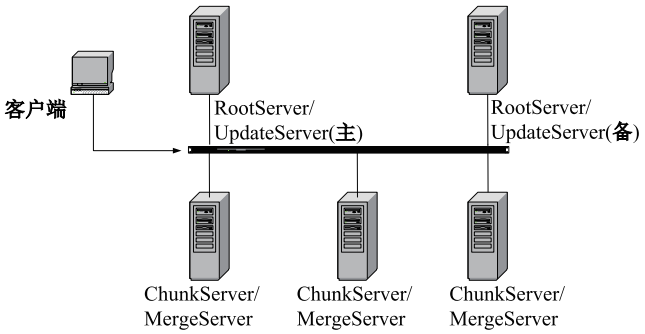

图 1　OceanBase架构

Fig. 1　The architecture of OceanBase

在 OceanBase 中, 和客户端直接连接的是 MergeServer. 客户端通过 MySQL (关于数据库管理系统)协议将 SQL 请求发送到 MergeServer 后, MergeServer 首先解析 MySQL 协议, 从中提取出用户发送的 SQL 语句, 接着通过 MS-SQL 模块, 使用 Flex 与 Bison[12]进行词法解析和语法分析[13], 生成语法树[13]; 然后生成 SQL 语句的逻辑查询计划和物理查询计划; 最后根据物理查询计划调用 OceanBase 内部的各种操作符. MergeServer 是 OceanBase 查询请求

的入口, 负责对收到的 SQL 进行处理和响应.

1.2 PL/SQL 和存储过程简介

PL/SQL 是 Oracle 数据库对 SQL 语句的扩展定义并实现的一种过程化 SQL 语言(Procedural Language/SQL), 也是一种程序语言, 在普通 SQL 语句的使用上增加了编程语言的特点. PL/SQL 引入多种数据类型、变量声明、条件控制、循环结构和子程序等过程语言要素来强化 SQL 语句的过程处理能力. 所以 PL/SQL 就是把数据操作和查询语句组织在 PL/SQL 代码的过程性单元中, 并通过逻辑判断、循环等操作实现复杂的功能或者计算的程序语言.

存储过程是一段被命名后保存在数据库服务器端的一段预先编译的代码. 上层应用给定存储过程名称和相应参数后, 解释执行预编译的代码, 能有效减少上层应用与数据库交互的次数和数据传输量.

2 OceanBase存储过程机制设计

2.1 OceanBase存储过程架构

在 OceanBase 中为实现存储过程, 选择使用 PL/SQL 作为存储过程的宿主语言. 因为 PL/SQL 最为常用, 用户更习惯 PL/SQL 的编程方式. 但 OceanBase 的原有架构中, MergeServer 只能对 SQL 语句进行解析, 不能解析存储过程使用的 PL/SQL 语句. 所以在原来的 MergeServer 架构[14]中, 增加 PL/SQL 引擎模块来对存储过程的 PL/SQL 语句进行解析处理. 增加 PL/SQL 引擎模块后的 MergeServer 架构如图 2 所示.

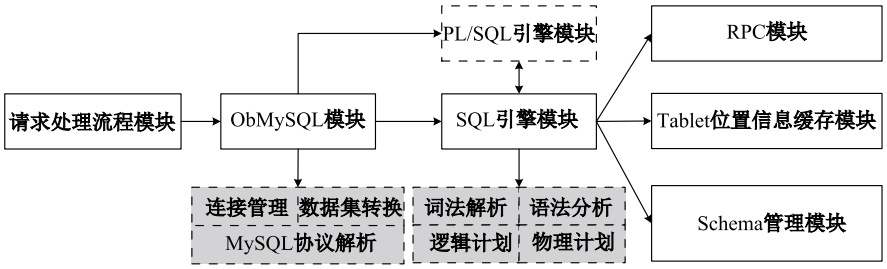

图 2 MergeServer架构图

Fig. 2 The architecture of MergeServer

增加了 PL/SQL 引擎模块后, 当 MergeServer 收到存储过程相关的 PL/SQL 语句后, 会将收到的 PL/SQL 交给 PL/SQL 引擎处理. PL/SQL 引擎[15]包括编译器和解释器两个部分. 编译器负责对 PL/SQL 进行编译. 解释器负责解释执行[16]编译器编译后的中间代码.

2.2 存储过程创建和存储

2.2.1 存储过程创建

存储过程的创建可以说是存储过程物理计划树[17]的生成过程. 物理计划树的生成分为 4 个步骤: 词法解析; 语法分析; 逻辑计划生成; 物理计划生成. 存储过程创建的流程如图 3 所示.

当存储过程语句发送到 MergeServer 后, PL/SQL 引擎对存储过程代码进行语法词法解析, 判断是否有语法错误, 如果有错误, 则将予以报错, 否则将会生成一棵语法树. 语法树生成后进入到逻辑计划生成部分, 对存储过程中的表、字段、属性、变量的合法性进行检查[17], 如果有错误则提示错误信息. 最后进入物理计划生成阶段, 会先判断存储过程在系统

表里面是否已经定义过, 如果存在相同名称的存储过程则提示用户, 否则将存储过程源码存入系统表 __all_procedure 里面, 并把生成的物理执行计划存储在数据库服务端; 然后返回创建成功的提示信息. 在创建存储过程的时候并不会解释执行生成的物理计划, 物理计划只会在调用的时候才会被解释执行.

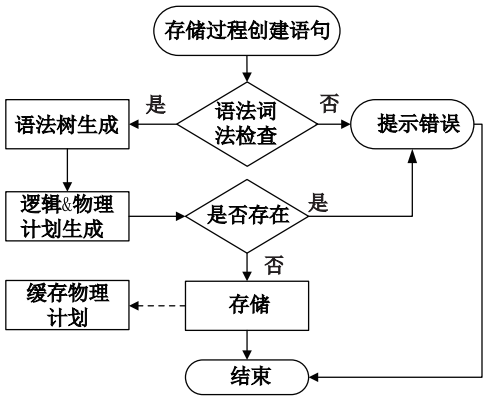

图 3　存储过程创建流程图

Fig. 3　The flowchart of stored procedure creating

2.2.2　存储过程的存储

存储主要分为两种方式：单源码方式和源码+预编译方式. 单源码方式是指只在数据库中保存存储过程的源码, 而后者是在数据库中保存了源码后, 并且也保存了预编译生成的执行单元. 在 OceanBase 的存储过程实现方案中存储过程的存储模式, 采用和 Oracle、PostgreSQL 类似的方案, 即源码+中间代码的存储方案, 不同的是在 OceanBase 中, 中间代码不是语法树, 而是物理计划树.

在 OceanBase 中新增一张名为 __all_procedure 的系统表, 用来保存存储过程源码. __all_procedure 表的基本字段如表 1 所示.

表 1　__all_procedure表的字段

Tab. 1　Schema of __all_procedure

字段	类型	描述
sp_name	varchar	存储过程名称
arg_number	int	参数个数
args_type	varchar	参数数据类型,使用逗号分隔
args_inout	varchar	参数的输入输出类型
sp_source	text	存储过程源码

在 OceanBase 中, 客户端连接到 MergeServer 的时候, 系统会为该连接分配一个 SESSION 用来表示这次会话. 在 SESSION 中建立一个缓存中间代码的队列, 为了不过多占用内存, 中间代码的缓存采用 FIFO(First Input First Output) (先进先出队列)策略, 如果当前的 SESSION 中缓存的中间代码过多, 可将最早进入缓存队列的对象移除.

2.3　存储过程的执行

调用存储过程的语句进入到查询处理器后, 首先会通过 PL/SQL 引擎进行词法、语法解析, 判断是否存在语法错误. 然后根据解析出来的存储过程名称和参数在缓存队列中进行查找, 当函数签名和调用参数一致时返回该存储过程的物理执行计划树; 如果没有在缓存队列中找到

物理执行计划树, 就到系统表 __all_procedure 中查找, 查询到记录则将查询到的存储过程源码使用 PL/SQL 引擎重新编译生成物理计划树, 否则提示存储过程不存在或错误. 最后将物理计划树使用 PL/SQL 解释器进行解释执行. 存储过程执行流程如图 4 所示.

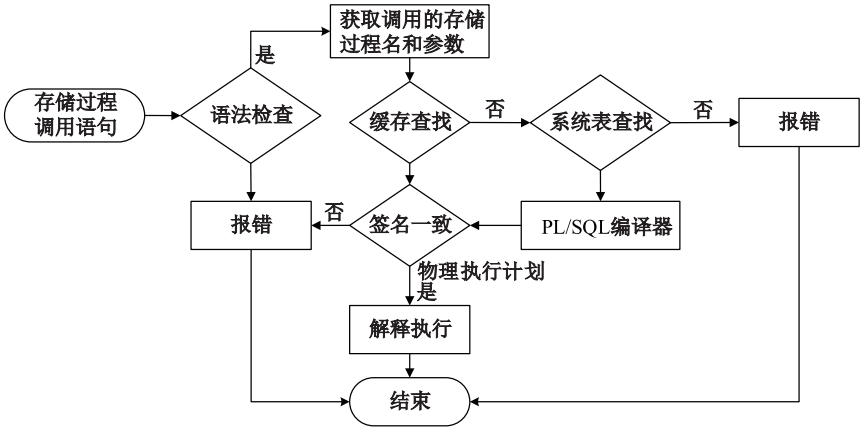

图 4 存储过程执行流程图

Fig. 4 The flowchart of stored procedure execution

存储过程是没有返回值的, 但是可以通过 INOUT 和 OUT 类型的输出参数来实现. 因此在执行存储过程前将 INOUT 和 OUT 类型的参数拷贝到运行时状态栈的最底层中, 当解释执行结束过后, 将 INOUT 和 OUT 类型的参数拷贝到用户的变量空间.

3 OceanBase PL/SQL引擎设计与实现

3.1 PL/SQL引擎架构

在 OceanBase 的 MergeServer 中没有 PL/SQL 处理引擎, 因此不能支持过程化语言. 根据 PL/SQL 兼有过程式语言和 SQL 语句的特点, 在 OceanBase PL/SQL 引擎设计的时候, 为了充分利用原有系统的 SQL 引擎, 把过程式语句和 SQL 语句分开处理[18].

如图 5 所示, 在对 PL/SQL 编译执行的时候, 首先读取 PL/SQL 语句块, 进行编译, 在编译的过程中如果是 SQL 语句, 调用 OceanBase 的 SQL 引擎进行编译生成语法树; 如果是过程语句则由 PL/SQL 引擎来进行编译生成语法树, 然后将两部分语法树结合生成一棵语法树. 在 PostgreSQL 系统中, PL/pgSQL 引擎只生成语法树作为中间代码, 然后解释器根据语法树解释执行遇到 SQL 语句时调用系统提供的内部接口进行执行. OceanBase PL/SQL 引擎将中间代码的层次从语法树提升到物理计划树, 物理计划树在解释执行的时候无需再次通过 SQL 编译成物理计划, 提高了存储过程中 SQL 语句的执行速度.

OceanBase PL/SQL 引擎在对 PL/SQL 代码编译后,生成了一种物理计划树形式的中间代码, 这种代码并不能直接执行, 需要一个解释器来根据每个节点的类型进行相应的解释执行. 据此可以将 OceanBase PL/SQL 引擎分为两部分: 编译器和解释器. 在 OceanBase 的 PL/SQL 引擎架构中, 词法分析器、语法分析器、逻辑计划生成器、物理计划生成器是属于编译器部分, 解释执行器属于解释器部分.

3.2 PL/SQL编译器

PL/SQL 的编译过程实际上就是语法树和物理计划树的生成过程. 语法树的生成需要经过词法分析和语法分析两个步骤, 形成抽象语法树结构[16].

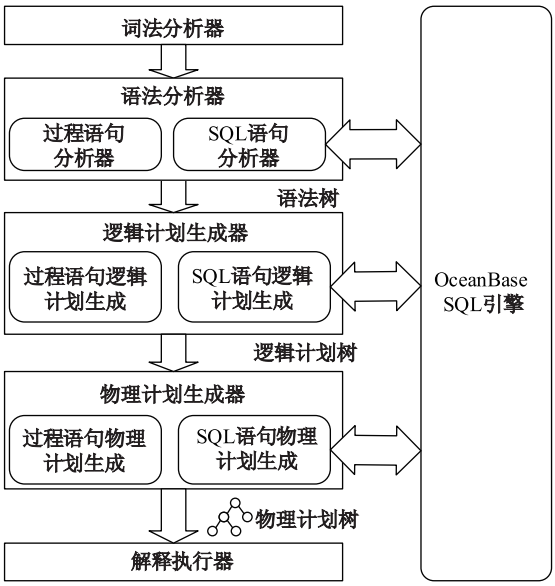

图 5　PL/SQL引擎架构

Fig. 5　The architecture of PL/SQL engine

词法分析程序是从程序源码中提取分析标识符(常量、数学符号、括号、变量名和保留字等), 以提供给后续的语法分析程序使用. 语法分析程序的作用是对词法分析后产生的标识符进行语法分析, 判断其是否符合语法, 形成抽象语法树结构. 在实践中, 语法分析过程中可能包括多个任务, 比如将不同的词法单元的信息收集到符号表[13]中、进行类型检查和其他类型的语义分析, 以及生成中间代码. 图 6 所示为 IF 节点的语法树.

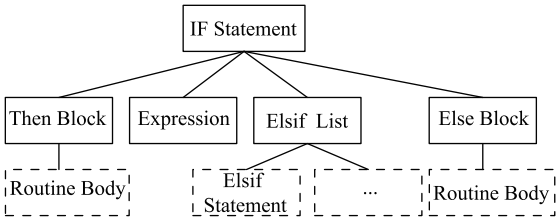

图 6　IF语句语法树结构

Fig. 6　The syntactic tree of IF statement

OceanBase PL/SQL 语句在生成了语法树后, 仅仅只能够判断 PL/SQL 的语法是否正确. 所以构建语法树后还需要生成逻辑计划, 而逻辑计划需要明确 SQL 语句中所涉及的表、字段和表达式等是否有效. PL/SQL 语法树在生成逻辑计划和物理计划的时候采用分治法: 对于 SQL 语句则调用 OceanBase 的 SQL 引擎生成对应的逻辑计划和物理计划, 否则使用 PL/SQL 引擎生成逻辑计划和物理计划.

3.3　PL/SQL 解释器

3.3.1　解释执行

OceanBase PL/SQL 引擎生成物理计划树表示的中间代码后, 并不能直接执行. 需要一个解释器对生成的中间代码进行解释执行.

在传统数据库的存储过程的实现方案中, 会区别对待控制流程语句和 SQL 语句, 会将 SQL 语句交给现有的 SQL 引擎来执行, 这样可以降低实现难度, 但同时这会降低存储过程的执行效

率. 但是在 OceanBase 中, 由于在生成中间代码的时候已经将其中的 SQL 语句生成为物理执行计划, 因此在执行 SQL 语句的时候无需对 SQL 再次编译, 这样做的好处是在执行 WHILE 和 LOOP 过程语句的时候减少 SQL 编译时间.

OceanBase PL/SQL 解释器的方案就是遍历物理计划树, 如图 7 所示, 从物理计划树的根节点开始, 在遇到 SQL 语句的物理计划的时候就直接交由 OceanBase 既有的执行引擎执行该物理计划, 而遇到流程控制语句的时候, 根据节点类型采用不用的算法进行执行, 在执行过程中如果对变量的操作部分由运行时状态完成、如果节点包含 Routine Body 的话, 则继续递归地去遍历访问这些节点直到结束.

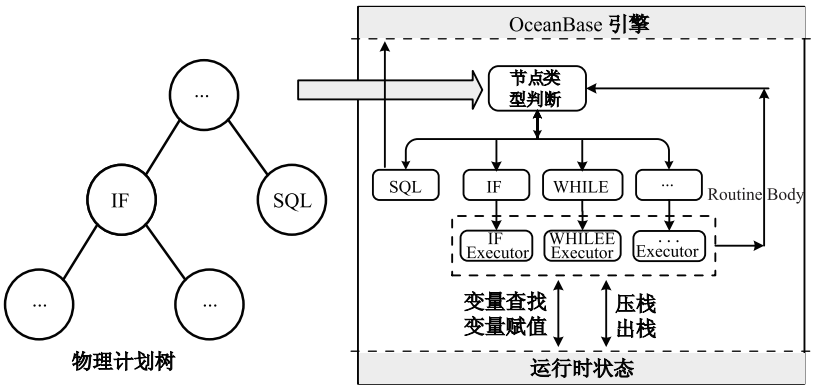

图 7 PL/SQL解释器

Fig. 7 The architecture of PL/SQL executor

3.3.2 运行时状态

由于解释器在解释执行存储过程中需要获取声明变量的类型和值, 为此需要为解释器维护一个运行时状态[13]. 运行时状态包括一个唯一标识符和一个符号表: 唯一标识符用来表示是否执行中有异常; 符号表用来存储变量信息. 在 OceanBase 中系统变量和用户变量都存储在 Session 里, 所以在设计运行时状态的时候需要考虑用户的自定义变量和运行时变量的重名问题. 在 OceanBase 中可以通过给运行时变量名添加前缀的方式来区别用户变量. 我们的方案是随机 16 位字符作为运行时变量名的前缀, 这样可以尽量防止变量名冲突.

PL/SQL 中语句块是可以相互嵌套的, 变量可以在不同的语句块中声明, 但是在相同域内不能重名. 为了实现对变量的作用域控制, 符号表的数据结构为一个栈, 每一个栈元素是一个 Hash 表, Hash 表里通过变量名来获取或设置变量的值. 每进入一个新的语句块就增加一个 Hash 表入栈, 当前层次的语句块可以访问栈顶到栈低的所有元素的时候以及当一个语句块结束的时候, 栈顶元素出栈. 如图 8 所示.

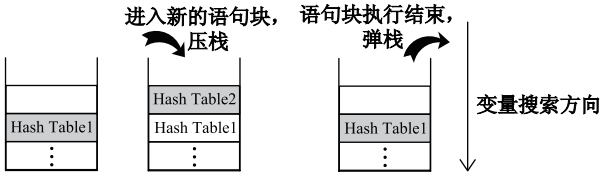

图 8 运行时状态堆栈

Fig. 8 Runtime stack

当定义一个变量时, 搜索方向为从当前层开始往栈底搜索. 如果在搜索过程中没有找到同名变量, 则可以允许声明该变量, 否则提示变量已声明. 同理在访问变量和变量赋值的时候也是

按照自顶向下的顺序进行遍历查找.

4　实　　验

4.1　实验环境

实验中把 OceanBase 数据库的 RootServer、UpdateServer、ChunkServer 和 MergeServer 都部署在同一台服务器上, 客户端用来运行测试程序. 服务器的配置如下：CPU E5606 2.13 GHz; 132 GB 内存; 2TB/7500 转硬盘; CentOS release 6.5 系统. 客户端的配置为: CPU E5-1603 2.80 GHz; 32 GB 内存; 1TB/7500 转硬盘; Windows7 64 位.

4.2　实验过程

本测试主要采用客户机/服务器模式下, 在客户端使用 JDBC 调用存储过程和完全通过 JDBC 完成数据库 DML 语句操作的性能对比. 实验中建立 1 张 TEST 表, 建表语句为

CREATE TABLE TEST(C1 int primary key,C2 int,C3 varchar);.

步骤为, 在数据库中建立名为 TEST 且有 1 个参数为 int 类型的存储过程. 相应的创建语句为

CREATE PROCEDURE TEST(X INT) BEGIN WHILE X > 0 THEN SET X = X – 1; INSERT INTO TEST VALUES(X, 1, 'TEST', 123.456, 'TEST'); END WHILE; END;.

分别采用 JDBC 调用存储过程和循环使用 JDBC 调用数据库完成多对 TEST 表的增删、改查操作, 通过参数控制存储过程执行的循环次数.

4.3　实验结果分析

通过存储过程 (Stored Procedure) 和 JDBC (Jave Data Base Connectivity) (Java 数据库连接), 完成对 TEST 表从 1 000—10 000 条记录的插入、更新、删除、查询等操作. 所用的时间如图 9 所示, 横轴表示 SQL 和存储过程的执行次数, 纵轴表示执行耗时.

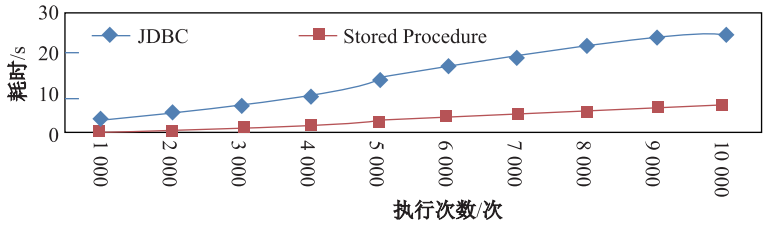

图 9　JDBC 与存储过程 SQL 性能对比

Fig. 9　SQL performance comparison JDBC with stored procedure

由图 9 的数据可知, 使用存储过程和 JDBC 执行查询操作的耗时和执行次数是呈线性增长的, 随着执行次数的增加, 执行时间也线性增加. 而且存储过程的执行时间明显小于 JDBC 的执行时间, 其性能大约是 JDBC 的 3 倍. 之所以有这样的结果, 主要是使用存储过程可以带来一些优势, 例如减少了网络上数据传递的通信量和通信的次数、提高了事务执行的速度、降低了事务响应应时间、减少了 SQL 语句编译的时间, 以及不用每次都对 SQL 进行语法解析、逻辑计划生成和物理计划生成等.

5　总结与展望

由于 OceanBase 发布的开源版本没有考虑对存储过程的支持, 影响了其在企业和机构的推广使用. 存储过程的实现取决于数据库系统的架构和实现方案. 本文基于 OceanBase 架构的特点及开源数据库存储过程的实现方案, 设计并实现了 OceanBase 的存储过程功能: 以增新增语

法结构为目标, 增加了 PL/SQL 解析引擎; 选择编译流程控制语句及 SQL 语句的实现方案, 减少了 OceanBase 因存储过程额外编译 SQL 语句的时间; 在实现过程中解决了面向过程的流程控制语句与面向集合的 SQL 语句的统一处理; 解释了执行存储过程时的状态维护以及运行时变量与用户变量的区分等难点.

实验表明, 存储过程能够减少客户端与数据库服务器的通信量, 并通过缓存生成的物理计划树提高执行性能. 但是还有一些不完善的地方, 如异常处理机制和重载存储过程都还未能实现, 在以后的工作中将对这两个问题进行研究.

[参 考 文 献]

[1] CHANG F, DEAN J, GHEMAWAT S, et al. Bigtable: A distributed storage system for structured data [C]//Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI). 2006: 205-218.

[2] CORBETT J C, DEAN J, EPSTEIN M, et al. Spanner: Google's globally distributed database [J]. ACM Transactions on Computer Systems, 2013, 31(3): 251-264.

[3] STONEBRAKER M, WEISBERG A. The voltDB main memory DBMS [J]. IEEE Data Eng Bull, 2013, 36(2): 21-27.

[4] OceanBase [EB/OL]. [2016-03-01]. https://github.com/alibaba/oceanbase/.

[5] HURSCH C J, HURSCH J L. SQL: The Structured Query Language [M]. [S.l.]: TAB books Inc, 1988.

[6] PL/pgSQL [EB/OL]. [2016-03-02]. http://www.postgresql.org/docs/8.3/static/plpgsql.html.

[7] 彭智勇, 彭煜玮. PostgreSQL数据库内核分析 [M]. 北京: 机械工业出版社, 2011.

[8] MySQL [EB/OL]. [2016-03-02]. http://www.mysql.com/.

[9] Stored Procedure [EB/OL]. [2016-03-02]. http://en.wikipedia.org/wiki/Stored_procedure.

[10] URMAN S. Oracle9i PL/SQL Programming [M]. 北京: 机械工业出版社, 2002.

[11] 杨传辉. 大规模分布式存储系统原理解析和架构实战 [M]. 北京: 机械工业出版社, 2013.

[12] LEVINE J, JOHN L. Flex & Bison [M]. 南京: 东南大学出版社, 2010.

[13] AHO A V, SETHI R, ULLMAN J D. Compilers: Principles, techniques, and tools [M]. Boston: Addison-Wesley Publishing Company, 1986.

[14] 阳振坤. OceanBase关系数据库架构 [J]. 华东师范大学学报(自然科学版), 2014(5): 141-148+163.

[15] 高朝瑞. GKD-Base PL/SQL存储过程和包的研究与实现 [D]. 长沙: 国防科学技术大学, 2004.

[16] 汪琦. 基于解释器的数据库存储过程研究 [D]. 武汉:华中科技大学, 2007.

[17] GARCIA-MOLINA H, ULLMAN J D, WIDOM J. 数据库系统实现 [M]. 杨冬青, 吴愈青, 包小源, 译. 2版. 北京: 机械工业出版社, 2010.

[18] 朱涛, 周敏奇, 张召. 面向 OceanBase 的存储过程实现技术研究 [J]. 华东师范大学学报(自然科学版), 2014(5): 281-289.

(责任编辑: 李 艺)