存储过程功能设计文档

修订历史

版本	修订日期	修订描述	作者	备注
Cedar 0.1	2016-01-03	存储过程功能设计文档	祝君	
Cedar 0.2	2016-06-15	存储过程功能设计文档	朱涛 王冬慧	

1 需求分析

存储过程（Stored Procedure）是在大型数据库系统中，一组为了完成特定功能的SQL语句集，存储在数据库中，用户通过指定存储过程的名字并给出参数（如果该存储过程带有参数）来执行它。存储过程是数据库中的一个重要对象，任何一个设计良好的数据库应用程序都应该用到存储过程。

Cedar是华东师范大学信息科学与工程研究院基于OceanBase 0.4.2 研发的可扩展的关系数据库，实现了巨大数据量上的跨行跨表事务。但开源的OceanBase 0.4 版本中缺乏对存储过程的支持。Cedar 0.1版本在此基础上添加了存储过程的支持。此外，Cedar 0.2 版本将存储过程定义为事务，事务的执行通过存储过程的执行来进行。因此，重新规范了Cedar 0.1 版本关于存储过程部分的语法，重构了物理操作符的实现方式。 并通过成组执行和缓存的方式对事务处理进行了优化。存储过程SQL的解析流程可参考Cedar0.1版本存储过程设计文档，本文主要介绍存储过程语法规范和事务处理优化模块的设计。

2 功能简述

存储过程是数据库系统的一个功能拓展，将一组带有特定功能的SQL语句以及控制流程语句存储在服务器端，用户通过存储过程的名称以及相应参数进行调用。存储过程支持变量，顺序，条件和循环等控制结构，存储过程可以定义参数这使得存储过程具有很强的模块性同时具有批处理性。虽然存储过程没有返回值，但是存储过程可以通过定义输出参数来实现返回结果的功能。这些特性都使得存储过程成为数据库系统必不可少的功能。由于存储过程的源码是存储在服务器的，因此客户端可以通过存储过程名以及参数调用，不需要把这些SQL语句集合发送到服务器，这样的话可以减少网络中传输的数据量，降低网络负荷。

- 目前的版本中支持的语法有，CREATE PROCEDURE、DROP PROCEDURE、CALL、IF、WHILE、FOR LOOP、LOOP、CASEWHEN、SELECT INTO、DECLARE、SET。不支持cursor。详细的语法使用见存储过程语法规范及使用手册。

- 目前存储过程支持的类型为Cedar 0.1版本中现有的类型，其他类型不支持。不支持varchar的加操作。

我们将事务封装在存储过程中，这样做的优势在于事务的执行不会受到外界的干扰。本版本在事务执行前对事务进行了编译优化，分析指令之间的依赖关系，调整指令的执行顺序，将同种类型指令进行成组执行，使得事务执行能够获得更高的效率。此外，在本版本中还添加了存储过程缓存管理，通过缓存存储过程的物理计划，使得每次执行存储过程时减少编译次数，提升整体执行效率。

3 设计思路

事务优化主要分为两个功能模块：1）缓存管理；2）编译优化成组执行。

3.1 缓存管理

缓存能够减少存储过程物理计划编译的次数，从而减少执行时间。将存储过程的缓存管理根据功能区分主要分为以下几个模块：存储过程的创建；存储过程的删除；存储过程物理计划缓存的创建；存储过程物理计划缓存的删除。

- 在存储过程创建时，MS将存储过程源码发送给RS，由RS与UPS交互将其插入到系统表__all_procedure中去，并通知全局的MS在本地保存源码，以便于MS读取源码。
- 在存储过程删除时，RS与UPS交互将其从__all_procedure中删除，并通知全局的MS将本地保存的源码删除。
- 在每个客户端链接的时候会产生一个session，在session中调用一个存储过程，首先根据存储过程名判断在session本地有没有缓存物理计划，若没有，去MS缓存中读取源码，然后重新编译，编译成功后将物理计划缓存在session中以备重复调用。若有，则判断该存储过程的物理计划是否失效，若失效，则重新编译。否则，则不需要编译存储过程源码，直接调用物理计划。

> 缓存管理的优势在于，一方面是避免每次调用的时候都要从RS端获取源码，造成RS的压力，以及增大延迟； 另外一方面是降低每次调用时MS编译存储过程的开销。

3.2 编译优化成组执行

在存储过程中，每个指令存在三种执行模式：

- local processing
- cs-rpc
- ups-rpc

优化目标是在事务编译阶段分析指令，调整不同类型指令的执行次序，将 ups-rpc 合并在一起执行。因为目前UPS的处理能力是系统的性能瓶颈。该优化主要分为三步：1）事务编译。分析依赖关系，建立指令依赖图。2）根据指令依赖图，调整指令执行次序。3）将ups-rpc指令合并。在事务编译阶段，我们分析每个指令访问的变量集合以及表集合，并据此判定指令之间是否存在访问依赖。我们将依赖类型主要为以下四种（假设指令a 先于指令b 执行）：

- True Dependence: 指令b读取了指令a的写入
- Anti Dependence: 指令a读取了指令b将要写入的对象
- Output Dependence: 指令a与指令b均修改同一个对象。
- Control Dependence: 指令a的执行决定了指令b是否执行。

其中 Control Dependence 存在于分支控制结构中。其他的几种依赖类型通过分析每个指令的读写集合是否相交进行判定。根据指令之间的依赖关系建立一张有向图。在执行次序调整的优化过程中，我们建立的是一张反向图，即若 a先于b执行并且它们之间存在依赖，那么建立一条从b指向a的边。执行次序调整的步骤如下所示：

> Step 1: 首先，将非cs-rpc类的根节点压入执行栈中，并删除相关的边。持续该步骤，直到剩余的节点都是cs-rpc类的根节点。
> Step 2: 然后，判断非ups-rpc类的根节点，如果该节点指向了某个ups-rpc类的节点，那么就将该节点压入执行栈中，并删除相关的边。持续该步骤，直到找不到这节点。
> Step 3: 重复执行Step 1 和 Step 2。直到两个步骤都无法从图中删除节点。
> Step 4: 将剩余节点按照拓扑顺序压入执行栈中。
> Step 5: 将执行栈中的节点依次出栈，形成最终的执行次序。

在正确调整执行次序后，我们需要将连续的 ups-rpc 合并。合并方法如下：

> Step 1: 按照执行次序，找到第一个ups-rpc，然后继续探查直到遇到一个cs-rpc，记录最后一个ups-rpc
> Step 2: 如果第一个ups-rpc和最后一个ups-rpc相同的，那么不进行成组，若该指令是一个循环指令，那单独成组该指令；如果不相同，那么将这段指令成组在一起。
> Step 3: 分配一个 SpGroupInst，将需要成组的指令按次序插入到 SpGroupInst中。

4 参考文献

[1] Freedman, C., Ismert, E., & Larson, P. (2014). Compilation in the Microsoft SQL Server Hekaton Engine. IEEE Data Eng. Bull., 22–30. Retrieved from ftp://131.107.65.22/pub/debull/A14mar/p22.pdf.

[2] Kennedy, K., & McKinley, K. S. (1892). Loop distribution with arbitrary control flow. Proceedings SUPERCOMPUTING '90, 407–416. doi:10.1109/SUPERC.1990.130048.

[3] Stonebraker, M., Madden, S., Abadi, D. J., Harizopoulos, S., Hachem, N., & Helland, P. (2007). The End of an Architectural Era (It’s Time for a Complete Rewrite). Vldb, 12(2), 1150–1160. doi:10.1080/13264820701730900.