2021年9月

openGauss MOT（内存表）DBA指南

作者:
王鹏 博士
Vinoth Veeraraghavan
Vladi Vexler

议程

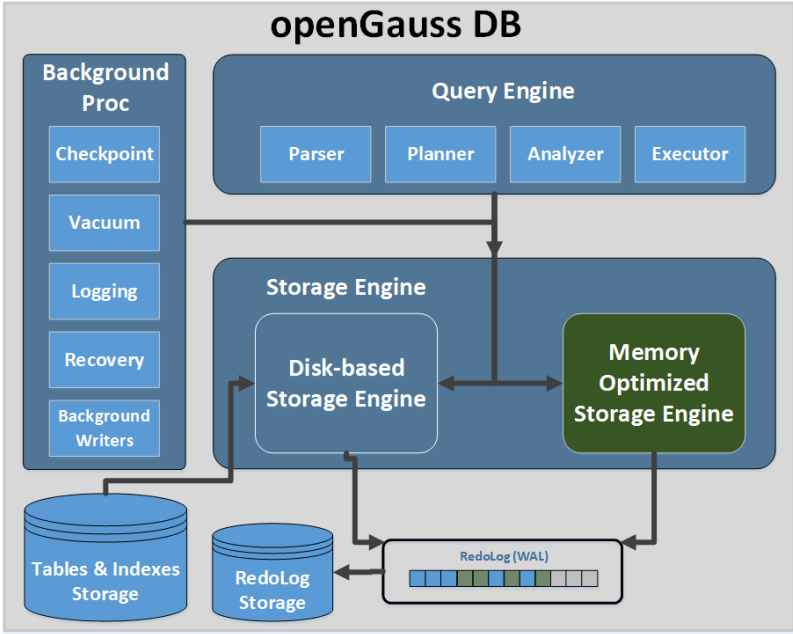
- MOT介绍
- MOT部署与配置
- MOT使用情况
- 测验和家庭作业

OPENGAUSS MOT技术介绍

MOT介绍

内存表：

- 基于事务的行存储引擎，与传统磁盘引擎并排的基于磁盘的存储引擎
- 针对多核和大内存服务器优化：近线性可扩展性
- 符合ACID要求+严格可持久化+高可用性
- 完全集成到openGauss中，支持绝大部分SQL特性（存储过程、函数...）
- 支持x86和ARM64鲲鹏
- **优点：**
  - **高吞吐量：3倍于磁盘表，6倍于PG 12.2**
  - **低延迟：事务加速3倍至5.5倍**
  - **严格一致性保障的HA和RTO**

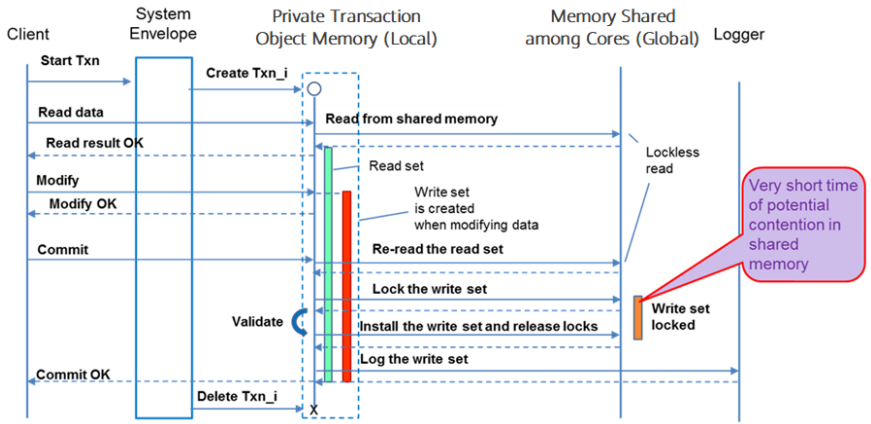


关键技术：

- 内存优化数据结构
- 无锁事务管理
- 无锁索引
- NUMA感知，事务本地内存
- 高效、可靠的持久化
- 查询本机编译(JIT)

基于Silo的事务管理以及乐观并发控制

- **事务本地内存与全局内存：**
  - 全局内存（shared memory）是所有CPU核心共享的长期内存，主要用于存储所有表数据和索引
  - 本地内存是短期的私有内存，主要用于在会话中处理事务，并将数据更改存储在本地内存中，直到提交阶段。
- **无锁事务管理：**
  - 所有相关数据都会从全局内存复制到本地内存。
  - 基于OCC算法，最小化全局内存上的争用时间。
  - 事务完成后，此数据将从本地内存推回全局内存。



扩展FDW

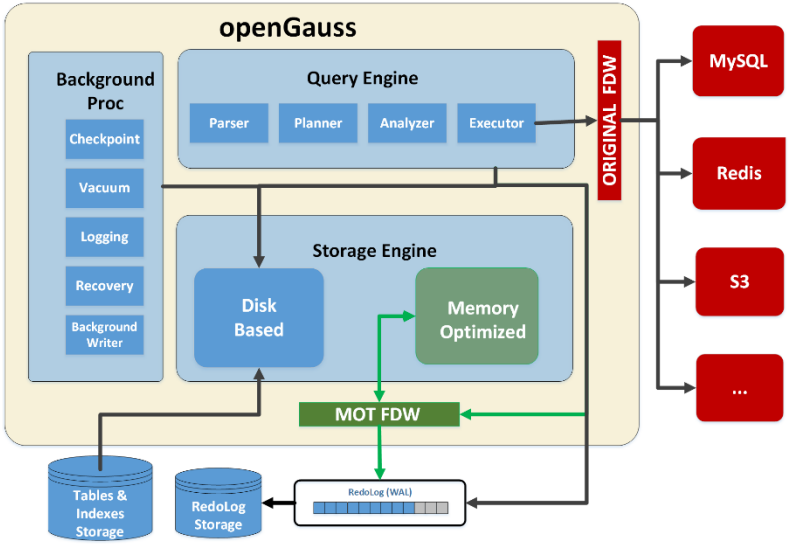
- 为了将MOT存储引擎集成到openGauss中，使用和扩展了现有的外部数据封装（FDW）机制。
- MOT存储引擎嵌入在openGauss内部，内存表由它管理。MOT表的访问由openGauss规划器和执行器控制。
- MOT从openGauss获取日志记录和检查点服务，并参与openGauss的恢复过程。

简单DDL：

```
create FOREIGN table test(x int) [server mot_server];
```

标准SQL

```
select * from test;
```



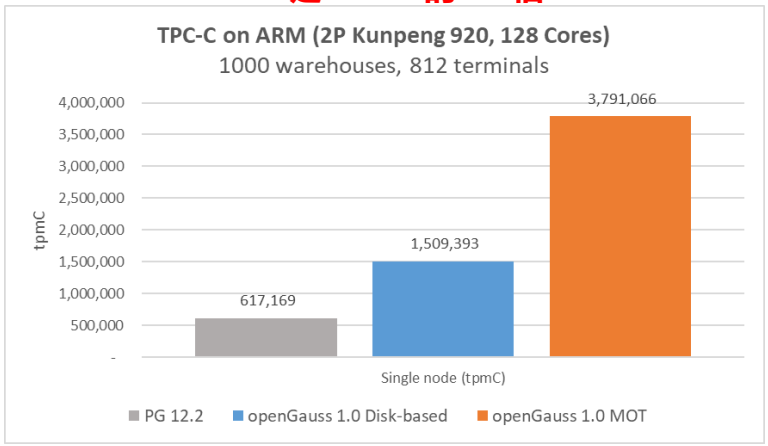
用于完全MOT集成的FDW扩展

- 创建表和索引
- 计划和执行的索引使用情况
- 持久化（日志记录、检查点）和高可用性
- 垃圾回收和删除表/索引
- 查询本机编译(JIT)

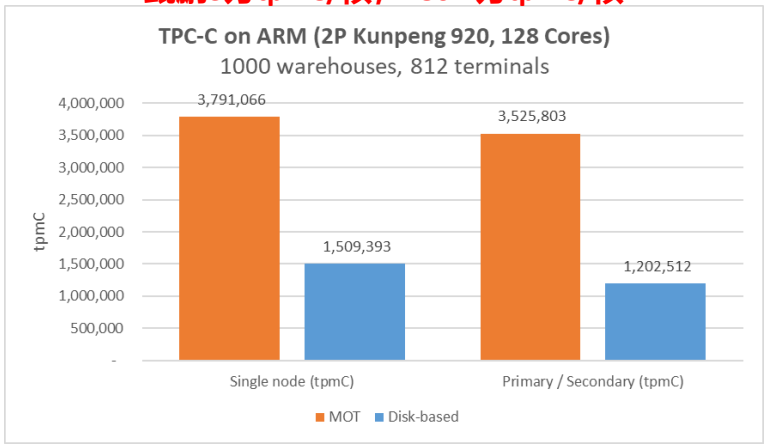
OPENGAUSS MOT性能基准（TPC-C）概述

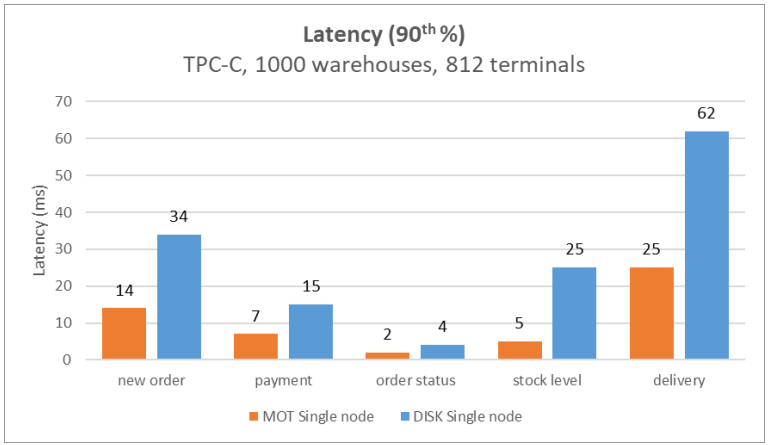
极致性能

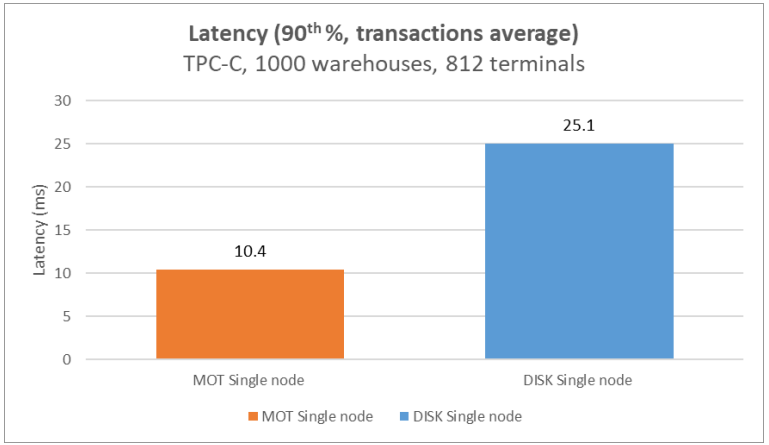
MOT是PG 12.2的6.15倍



鲲鹏3万tpmC/核，x86 4万tpmC/核







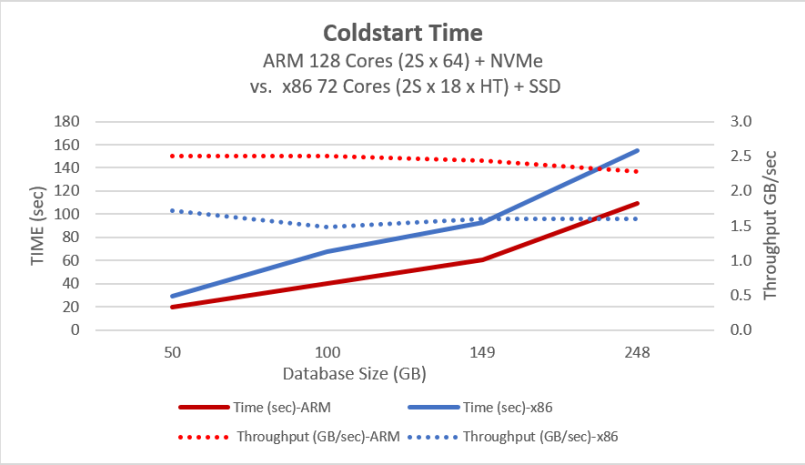
快速冷启动

冷启动恢复

系统在停止模式后完全运行所需的时间。

在40秒内加载100 GB数据库检查点（2.5 GB/秒）。

由于MOT索引不持久化，等于40秒内加载的数据+索引容量150 GB（3.75 GB/秒）。



OPENGAUSS MOT部署与配置

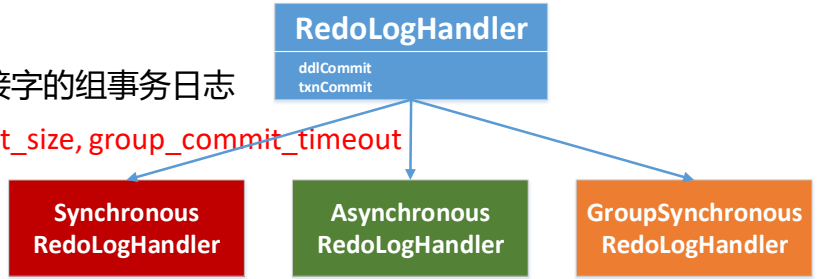
配置文件

- **postgresql.conf**
  1. **enable_increment_checkpoint=off//**如果在中启用了增量检查点，则无法创建MOT表。
  2. **max_process_memory = xxxGB//**根据应用程序需要（更多详细信息请参见内存规划幻灯片）。

- **mot.conf**
  - openGauss提供了预配置的mot.conf文件（与postgresql.conf位于同一路径）。为了获得最佳效果，建议根据应用程序的特定要求和首选项自定义其设置。
  - 8个设置组（总共<50个设置）:REDO日志、检查点、恢复、统计、错误日志、内存、垃圾收集、JIT（查询代码生成）

持久化和高可用：集成到openGauss中

- **日志记录：异步、同步和组提交（MOT独有）MOT优化：**
  - 每个事务的日志缓冲和无锁事务准备
  - 更新增量记录，这意味着仅记录更改
  - 支持NUMA的组提交：减少I/O，提高性能。基于NUMA套接字的组事务日志
  - ➢ 仅适用于组同步提交：enable_group_commit, group_commit_size, group_commit_timeout



- **检查点–基于最先进的CALC检查点算法**
  - 内存使用率低：任何时候最多存储两个记录副本。
  - 低开销。小于其他异步检查点算法。
  - 使用虚拟一致性点。不需要停止数据库就可以实现物理一致性。
  - ➢ 配置： checkpoint_dir, checkpoint_segsize, checkpoint_workers

Low-Overhead Asynchronous Checkpointing in Main-Memory Database Systems

Kun Ren, Thaddeus Diamond, Daniel J. Abadi, Alexander Thomson
Yale University
kun.ren@yale.edu; thaddeus.diamond@aya.yale.edu; {dna,thomson}@cs.yale.edu

- **冷启动恢复**
  - ➢ 配置：checkpoint_recovery_workers
- **复制和HA配置：**
  - ➢ 使用openGauss gs_ctl工具用于HA控制和操作群集。
  - ➢ 包括：gs_ctl switchover, gs_ctl failover, gs_ctl build

MOT内存规划

- 与PG一样，openGauss数据库进程的内存由max_process_memory设置中的上限控制，该设置在postgres.conf文件中定义。
- **MOT中的全局和本地内存：**
  - **全局内存：长期内存池，包含MOT表的数据和索引。全局内存均匀分布在所有NUMA节点上，并由所有CPU内核共享。**
  - **本地内存：短期对象的内存池，主要用于处理事务（也称为事务私有内存）和存储数据更改。在提交时，数据更改将移动到全局内存。内存对象分配以NUMA本地方式执行，以实现尽可能低的延迟。**
  - **规则1：** (max_mot_global_memory + max_mot_local_memory) < max_process_memory – 2GB
- **配置： min/max_mot_global_memory 和 min/max_mot_local_memory 设置。**
  - **规则2：可配置为max_process_memory的百分比（%）或单位（KB/MB/GB/TB）。**

```
min_mot_global_memory = 1 GB
max_mot_global_memory = 80%
```

如果MOT全局内存使用量太接近此定义的最大值，则为了保证MOT系统不崩溃，MOT将不接受新数据。超过此限制分配内存的尝试将被拒绝，并向用户报告错误。

MOT内存规划-#2

- **规则3：实际全局内存需求规划（从磁盘表迁移至内存表）**
  - 确定特定磁盘表的大小（包括其数据和所有索引）。以下统计查询可用于通过标准pg_relation_size函数确定customer 表的数据大小和 customer_pkey 索引大小。表大小=数据大小+索引大小：

```
select pg_relation_size('customer');
select pg_relation_size('customer_pkey');
```

  - 加上60%，在典型的OLTP场景（80:20读写比）中，每个表的MOT内存使用率比基于磁盘的表高出60%，因为MOT使用了更高性能的数据结构和算法，以实现更快的访问，具有CPU缓存感知和内存预取。
  - 加上预期的增长。例如5%月度增长= 80%年增长率(1.05^12)

- **规则4：实际本地内存需求规划**
  - 并发会话数x平均会话大小的函数（典型的OLTP工作负载为8 MB）。

```
SESSION_COUNT * SESSION_SIZE (8 MB) + SOME_EXTRA (一般为100MB)
```

- **规则5：单个事务必须限制为1GB大小。示例：**

```
delete from SOME_VERY_LARGE_TABLE;
```

OPENGAUSS MOT使用情况

授予权限，创建表，创建索引

1. **允许特定用户创建和访问MOT表（DDL、DML、SELECT）-仅运行一次以下语句：**

```
GRANT USAGE ON FOREIGN SERVER mot_server TO [user];
```

2. **创建MOT表：**

```
create FOREIGN table Mot_Table1 (id integer […]) [server mot_server];
```

- 始终使用“FOREIGN ”关键字引用MOT表进行CRUD操作。
- server mot_server 部分是可选的。

3. **删除MOT表**

```
drop FOREIGN table Mot_Table1;
```

- 始终使用“FOREIGN ”关键字引用MOT表。

4. **创建MOT表索引**

```
create index Mot_Table1_index1 on Mot_Table1(id [...]);
```

查询本机编译(JIT)

- **即时编译（JIT，有时也称为codegen）通过LLVM将完整查询编译为本机格式，以减少多个处理层次，性能明显更好，更加轻量化。**
- **编译后的代码会产生一个C函数指针，以后可以由许多会话并发调用以直接执行（Lite执行）**
- **针对OLTP进行了优化**
- **性能提升超过30%**

```
conn = DriverManager.getConnection(connectionUrl, connectionUser, connectionPassword);

// Example 1: PREPARE without bind settings
String query = "SELECT * FROM getusers";
PreparedStatement prepStmt1 = conn.prepareStatement(query);
ResultSet rs1 = pstatement.executeQuery())
while (rs1.next()) {…}
```

- **MOT JIT编译缓存计划**
  - 在同一会话中跨不同参数重用JIT结果
  - 在不同会话中重用JIT结果

OCC和重试

- **MOT使用乐观并发控制（OCC）事务机制。**
  - 在事务期间（使用任何隔离级别），直到提交阶段，不会对记录加锁。优点：性能更高。缺点：可能被中止。如果另一个会话尝试更新相同的记录，则更新可能会失败，MOT在提交时通过版本检查机制检测冲突。
  - 注意：当使用串行化或可重复读取隔离级别时，在使用悲观并发控制的引擎上也会发生类似的中止。

➢ **解决方案：重试代码。示例**

```
int commitAborts = 0;
while (commitAborts < RETRY_LIMIT) {
    try {
              stmt =db.stmtPaymentUpdateDistrict;
              stmt.setDouble(1, 100);
              stmt.setInt(2, 1);
              stmt.setInt(3, 1);
              stmt.executeUpdate();
              db.commit();
              break;
    } catch (SQLException se) {
              if(se != null && se.getMessage().contains("could not serialize
access due to concurrent update")) {
                  log.error("commmit abort = " + se.getMessage());
                  commitAborts++;
                  continue;
              }else { db.rollback(); }
              break;
   }
}
```

MOT监控

- https://opengauss.org/en/docs/1.1.0/docs/开发指南/mot-monitoring.html
- **MOT全局内存** `mot_global_memory_detail();`
- **MOT本地内存** `mot_local_memory_detail();`

```
> select mot_global_memory_detail();

 numa_node | reserved_size      | used_size
-----------+--------------------+-----------
-1         | 194716368896       | 25908215808
0          | 446693376          | 446693376
1          | 452984832          | 452984832
2          | 452984832          | 452984832
3          | 452984832          | 452984832
...
```

- **当前会话内存**

```
> Select * from mot_session_memory_detail() where
  sessid = pg_current_sessionid();

 sessid       | total_size         | free_size          | used_size
--------------+--------------------+--------------------+----------
123761.1238   | 6291456            | 1861992            | 4429464
```

2020年7月

代码：https://gitee.com/opengauss

文档：

https://opengauss.org/zh/docs/1.1.0/docs/DeveloperGuide/内存表特性.html

wangpeng.mail@huawei.com

Vinoth.Veeraraghavan@huawei.com

Vladi.Vexler@huawei.com