2024年全国大学生计算机系统能力大赛PolarDB数据库创新设计赛（天池杯）

初赛方案报告

学校： 南开大学

队伍： 数据都队

队员： 岳志鑫 王荣熙 张耕嘉

填写说明：

- 正文、标题格式已经在本文中设定，请勿修改；
- 本文档应结构清晰，描述准确，不冗长拖沓；
- 提交文档时，以PDF格式提交；
- 本文档内容是初赛内容的组成部分，务必真实填写。如不属实，将导致奖项等级降低甚至终止参加比赛。

目　　录

1. 代码及描述文档

主要修改文档及方向如下:

polardb_build.sh：	修改相关参数
tpch_copy.sh：	修改并行化、参数、
guc.c(src/backend/utils/misc/guc.c)：	开启预分配
heapm.c(src/backend/access/heap/heapam.c)：	修改多线程
heapm.h(src/include/access/heapam.h)：	修改多线程
postgresql.conf.sample.polardb_pg(src/backend/utils/misc/)：	修改参数
postgresql.conf.sample(src/backend/utils/misc/)：	修改参数
pg_hba.conf.sample（src/backend/libpq/pg_hba.conf.sample）	开启 unix 连接

2. 性能优化设计

2.1 优化目标

提高数据导入速度、降低磁盘 I/O 开销、减少内存使用

2.2 优化策略

（1）将大数据拆分为小数据再导入，并行执行 COPY 导入，逐一启动任务并始终保持 300 个并行的连接，减少导入时长。

（2）开启 PolarDB 预分配功能，减少 IO 次数，降低性能损耗。

（3）通过设置参数寻找更为高效的导入策略，增大导入效率。

（4）通过使用 unix socket 代替 tcp 连接，减少导入时长。

（5）通过设置多线程开启预分配，加快数据导入速度

3. 性能优化实现路径

步骤 1:拆分数据并开始并行导入

toch_copy.sh

```
DATA_DIR="/data"
SPLIT_DIR="$DATA_DIR/splitfiles"
# 创建目标目录（如果不存在）
mkdir -p $SPLIT_DIR
# 表名列表
tables=( nation region customer orders lineitem partsupp part supplier)
# 拆分并导入每个表
for table in "${tables[@]}"; do
  #echo "正在拆分 $table.tbl ..."
  # 使用 split 按 10000 行拆分文件，并将拆分后的文件放入 $SPLIT_DIR 目录
  split -l 100000 $DATA_DIR/$table.tbl $SPLIT_DIR/${table}_
  # 启动并行 COPY 操作，每个拆分后的文件都并行导入
```

```
      for file in $SPLIT_DIR/${table}_*; do
        # 限制并发数量为 100
        ((i=i%300)); ((i++==0)) && wait
        #echo "开始导入 $file ..."
        psql -h /tmp -p 5432 -U $DB_USER -d $DB_NAME -c "COPY $table FROM
'$file' WITH (FORMAT csv, DELIMITER '|');" &
      done
      wait
      echo "$table.tbl 拆分并导入完成"
    done
    echo "所有表导入完成！"
```

步骤 2:开启预分配

guc.c

```
    //yzx 修改预分配
    #define MAX_CONFIG_VARS 100
    struct config_bool config_pool[MAX_CONFIG_VARS];
    int pool_index = 0; // 当前使用到的预分配池的索引
    void DefineCustomBoolVariable(const char *name,
                                 const char *short_desc,
                                 const char *long_desc,
                                 bool *valueAddr,
                                 bool bootValue,
                                 GucContext context,
                                 int flags,
                                 GucBoolCheckHook check_hook,
                                 GucBoolAssignHook assign_hook,
                                 GucShowHook show_hook)
    {
        // 检查是否还有可用的预分配空间
        if (pool_index >= MAX_CONFIG_VARS) {
            // 如果超出预分配数量，可以选择扩展池或报错
            elog(ERROR, "Exceeded maximum number of custom bool variables");
            return;
        }

        // 从预分配池中获取下一个可用的 config_bool
        struct config_bool *var = &config_pool[pool_index++];
        // 初始化字段
        var->gen.name = name;
        var->gen.short_desc = short_desc;
        var->gen.long_desc = long_desc;
        var->gen.context = context;
        var->gen.flags = flags;
        var->variable = valueAddr;
        var->boot_val = bootValue;
        var->reset_val = bootValue;
        var->check_hook = check_hook;
        var->assign_hook = assign_hook;
        var->show_hook = show_hook;

        // 调试输出
        if (var->assign_hook != NULL
&& !is_session_dedicated_guc(&var->gen)) {
            ELOG_PSS(DEBUG1, "conf with assign_hook is shared '%s'",
var->gen.name);
        }

        // 注册该变量
        define_custom_variable(&var->gen);
    }
```

步骤 3:修改参数

polardb_build.sh、postgresql.conf.sample.polardb_pg、postgresql.conf.sample

```
    shared_buffers = 6GB
    autovacuum = off
    checkpoint_timeout = 1d
    max_wal_size = 128GB
    min_wal_size = 64GB
    checkpoint_completion_target = 0.9
    bgwriter_delay = 10ms     # 10-10000ms between rounds
    bgwriter_lru_maxpages = 500   # max buffers written/round, 0 disables
    bgwriter_lru_multiplier = 2.0   # 0-10.0 multiplier on buffers
scanned/round
    bgwriter_flush_after = 512kB    # measured in pages, 0 disables
    wal_level=minimal #yzx 修改 确保 WAL 级别为 minimal，禁用 WAL 写入
    max_wal_senders = 0        # 禁用 WAL 发送器
    hot_standby = off           # 禁用热备份
    synchronous_commit = off   # 禁用同步提交
    max_wal_size=128GB
    min_wal_size=64GB
    bgwriter_delay=10ms
    bgwriter_flush_after=512 #1MB
    bgwriter_lru_maxpages=500
    max_connections = 300
    unix_socket_directories='/tmp'
    autovacuum_naptime = 100min
    max_worker_processes = 32
```

步骤 4：使用 unix 代替 tcp 连接

pg_hba.conf.sample

```
    # TYPE  DATABASE        USER            ADDRESS                 METHOD
    # "local" is for Unix domain socket connections only
    local   all             all                                     trust
    local   postgres        postgres                                trust
    # IPv4 local connections:
    host    all             all             127.0.0.1/32            trust
    # IPv6 local connections:
    host    all             all             ::1/128                 trust
    # Allow replication connections from localhost, by a user with the
    # replication privilege.
    local   replication     all                                     trust
    host    replication     all             127.0.0.1/32            trust
    host    replication     all             ::1/128                 trust
```

步骤五：开启多线程

heapam.h

```
    #include <pthread.h>
    #include <stdlib.h>
    #include <stdio.h>
    // 定义线程参数结构
    typedef struct
    {
        Relation relation;
        HeapTuple *inputTuples;
        HeapTuple *outputTuples;
        int start;
        int end;
        TransactionId xid;
        CommandId cid;
        int options;
    } ThreadArgs;
    void *prepare_insert_range(void *args);
    void parallel_prepare_insert(Relation relation, HeapTuple *inputTuples,
HeapTuple *outputTuples,
                                int ntuples, TransactionId xid, CommandId
cid, int options, int numThreads);
```

heapam.c

```
    // 单线程处理函数
    void *prepare_insert_range(void *args)
    {
        ThreadArgs *threadArgs = (ThreadArgs *)args;

        for (int i = threadArgs->start; i < threadArgs->end; i++)
        {
            threadArgs->outputTuples[i] = heap_prepare_insert(
                threadArgs->relation,
                threadArgs->inputTuples[i],
                threadArgs->xid,
                threadArgs->cid,
                threadArgs->options);
        }

        return NULL;
    }

    // 并行处理函数
    void parallel_prepare_insert(Relation relation, HeapTuple *inputTuples,
HeapTuple *outputTuples,
                                int ntuples, TransactionId xid, CommandId
cid, int options, int numThreads)
    {
        pthread_t threads[numThreads];
        ThreadArgs threadArgs[numThreads];
        int batchSize = ntuples / numThreads;

        // 创建线程
        for (int i = 0; i < numThreads; i++)
        {
            int start = i * batchSize;
            int end = (i == numThreads - 1) ? ntuples : start + batchSize;

            threadArgs[i].relation = relation;
            threadArgs[i].inputTuples = inputTuples;
            threadArgs[i].outputTuples = outputTuples;
            threadArgs[i].start = start;
            threadArgs[i].end = end;
            threadArgs[i].xid = xid;
            threadArgs[i].cid = cid;
            threadArgs[i].options = options;

            if (pthread_create(&threads[i], NULL, prepare_insert_range,
&threadArgs[i]) != 0)
            {
                perror("Failed to create thread");
                exit(EXIT_FAILURE);
            }
        }

        // 等待线程完成
        for (int i = 0; i < numThreads; i++)
        {
            if (pthread_join(threads[i], NULL) != 0)
            {
                perror("Failed to join thread");
                exit(EXIT_FAILURE);
            }
        }
    }
```

4. 性能提升效果

4.1 测试结果

- 优化前：数据导入时间为 2280.596 秒。
- 优化后：数据导入时间为 1753.614 秒。

4.2 性能提升

- 数据导入时间减少了 23.107%。

5. 指导老师的贡献

指导老师：温延龙
贡献：

1. 对我们的优化方向提供建议，制定优化的计划。

2. 对每个阶段的步骤进行评审，检验是否符合正规逻辑。