truncate 功能开发文档

修改历史

版本	修订日期	修订描述	作者	备注
Cedar 0.3	2017-07-01	truncate功能开发文档	贺小龙	无

总体设计

1.1 综述

> Cedar 是由华东师范大学数据工程与科学学院基于OceanBase 0.4.2 研发的可扩展的关系数据库。在Cedar 0.2中，没有实现truncate table定义语言。为了满足某些应用的需求，需要在Cedar中实现truncate功能。该语句用于完全清空指定表，但是保留表结构。从逻辑上说，该语句与用于删除所有行的DELETE FROM语句相同。执行TRUNCATE语句，必须具有表的删除和创建权限。它属于DDL语句。TRUNCATE TABLE语句与DELETE FROM语句有以下不同：
>
> - 删减操作会取消并重新创建表，这比一行一行的删除行要快很多。
> - TRUNCATE TABLE语句执行结果显示影响行数始终显示为0行。
> - 使用TRUNCATE TABLE语句，表管理程序不记得最后被使用的
> - AUTO_INCREMENT值，但是会从头开始计数。
> - TRUNCATE语句不能在进行事务处理和表锁定的过程中进行，如果使用，将会报错。
> - 只要表定义文件是合法的，则可以使用TRUNCATE TABLE把表重新创建为一个空表，即使数据或索引文件已经被破坏。

1.2.名词解释

- 主控服务器（RootServer，RS）：Cedar集群的主控节点，负责管理集群中的所有服务器，以及维护tablet信息。
- 更新服务器（UpdateServer， UPS）：负责存储Cedar系统的增量数据，并提供事务支持。
- 基准数据服务器（ChunkServer， CS）：负责存储Cedar系统的基线数据。
- 合并服务器（MergeServer, MS）：负责接收并解析客户端的SQL请求，经过词法分析、语法分析、查询优化等一系列操作后发送到CS和UPS，合并基线数据和增量数据

1.3 功能简述

在Cedar中执行TRUNCATE TABLE语法如下：

```
truncate table [if exists] table_name1, table_name2, ..., table_name
N
    允许truncate多张表
    成功返回"0 rows affected"
    失败返回错误代码
```

因为设计方案的原因，在使用方面有如下几个限制：

- truncate之后表不允许更新直至下一次memtable冻结完成（不等合并完成即可恢复写）
- memtable 冻结手动触发

即在TRUNCATE TABLE操作执行之后,还需要手动执行小版本合并或者大版本合并操作才算真正完成TRUNCATE操作。具体操作命令如下：

```
    1. 主动发起大版本合并(工具在oceanbase/bin目录下)
        ./ups_admin -a ip -p port -t major_freeze
    2. 主动发起小版本合并
        ./ups_admin -a ip -p port -t minor_freeze
```

1.4 性能指标

> 实现truncate功能之后，系统在数据查询时首先向ups查询该表是否被truncate，如何被truncate，会将改变查询的数据范围。与没有实现改功能之前，多了一次交互，在性能上有一定影响。

2 模块设计

根据功能需求，truncate table 的功能实现涉及多个模块，具体每个模块分工如下:

模块	子模块	代码模块	备注
Truncate table执行模块	逻辑执行计划	sql	
	物理执行计划	sql	session environment需检查是否不在事务内、自动提交为真，如不满足则返回执行失败
	Updateserver上执行计划	Updateserver	Mutator提交到table_btree
基础数据结构	基础数据结构	updateserver	构建table_btree
Memtable模块	初始化、析构	updateserver	Memtable的初始化、析构
	createTime、modifyTime维护	updateserver	自动维护记录的createTime、modifyTime列值
	Checksum计算	updateserver	计算整个memtable的checksum
	更新操作执行	updateserver	Truncate操作后guard对该表的写操作，返回not allowed
	读操作执行	updateserver	Truncate操作后对该表的读一律返回row_deleted
Sstable模块	Dump memtable写入	sstable	冻结时sstable的写入，将truncate信息写入ObSStableSchema中。即使memtable中无行级增量数据，但只要table_btree中记录着truncate操作，也需要写入sstable中
	Dump memtable写入	sstable	
	Dump memtable读取	sstable	当ObSStableSchema中truncate_flag字段有效，则返回row_deleted。
锁机制实现	Table_btree互斥锁	updateserver	实现truncate操作之间的并发控制
日志写入回放	Ups日志写入、回放	updateserver	
内部表维护	记录某基础表最近一次被truncate的时间戳（包含相应索引表）		记录最近一次被清空时间

2.1 执行模块

执行模块总的分为三个部分，生成逻辑计划、物理计划并在ups上执行物理计划。在生成物理计划中session environment需要检查是否不在事务内、自动提交为真，如不满足则返回执行失败。执行的物理计划是将Mutator提交到已经初始化的table btree上。

2.3 Memtable模块

> 该模块主要是在ups上存储truncate信息，具体代码如下：

```
else if (is_trun_tab_(cell_info.value_))
{
    ret = ccw.tab_truncate();
    ccw.set_tab_truncated(true);
}
```

```
TEKey cur_table_key; //行主键
cur_table_key.table_id = cur_key.table_id;
TEValue * cur_table_value;
cur_table_value = table_engine_.get(cur_table_key);
if (cur_table_value != NULL && cur_table_value->cell_info_cnt != 0)
{
  ret = OB_TABLE_UPDATE_LOCKED;
  TBSYS_LOG(ERROR, "table_id %ld has been locked, wait for mem_free
ze", cur_key.table_id);
}
//add:e
```

2.4 Sstable模块

> 在冻结Memtable时，需要将truncate table信息写入到ObSStableSchema中，即使memtable中无行级增量数据，但只要table_btree中记录着truncate操作，也需要写入sstable中。其中schema增加了如下数据结构:

```
struct ObSSTableSchemaTableDef
{
  uint32_t table_id_;
  int64_t trun_timestamp_;
  NEED_SERIALIZE_AND_DESERIALIZE;
};
```

在table_btree中记录truncate操作的代码如下:

```
//将某表新增的行的信息添加到schema中
if (OB_SUCCESS != (ret = schema_mgr.build_sstable_schema(schema_han
dle_, sstable_schema)))
{
    TBSYS_LOG(WARN, "build sstable schema failed, err[%d]", ret);
}
//将truncate信息添加到schema中
else if (OB_SUCCESS != (ret = fill_truncate_info_(sstable_schema)))
{
    TBSYS_LOG(WARN, "fill truncate info into sstable schema failed,
err[%d]", ret);
}
else
{
    bret = true;
}
//mod:e
```

2.5 内部表维护

> 在系统中增加一张内部表来记录某基础表最近一次被truncate的时间戳。具体创建代码如下：

```
int ObExtraTablesSchema::all_truncate_op_info(TableSchema& table_sc
hema)
{
  int ret = OB_SUCCESS;
  table_schema.init_as_inner_table();
  strcpy(table_schema.table_name_, OB_TRUNCATE_OP_TABLE_NAME);
  table_schema.table_id_ = OB_ALL_TRUNCATE_OP_TID;
  table_schema.rowkey_column_num_ = 2;
  table_schema.max_rowkey_length_ = TEMP_ROWKEY_LENGTH;
  table_schema.max_used_column_id_ = OB_APP_MIN_COLUMN_ID + 7;
  int column_id = OB_APP_MIN_COLUMN_ID;
  ADD_COLUMN_SCHEMA("rs_trun_time",
  column_id++,
  1,
  ObPreciseDateTimeType,
  sizeof(int64_t),
  false);
  ADD_COLUMN_SCHEMA("table_id",
  column_id++,
  2,
  ObIntType,
  sizeof(int64_t),
  false);
  ADD_COLUMN_SCHEMA("table_name",
  column_id++,
  0,
  ObVarcharType,
  OB_MAX_TABLE_NAME_LENGTH+OB_MAX_DATBASE_NAME_LENGTH+1,
  false);
  ADD_COLUMN_SCHEMA("user_name",
  column_id++,
  0,
  ObVarcharType,
  OB_MAX_USERNAME_LENGTH,
  false);
  ADD_COLUMN_SCHEMA("info",
  column_id++,
  0,
  ObVarcharType,
  2 * OB_MAX_TABLE_NAME_LENGTH,
  false);
  return ret;
  }
```