SNAPSHOT ISOLATION 隔离级别功能开发文档

修订历史

版本	修订日期	修订描述	作者	备注
Cedar 0.2	2016-06-18	SNAPSHOT ISOLATION 隔离级别功能开发文档	郭进伟 肖冰	

1 总体设计

1.1 综述

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。在0.1版本中，Cedar仅实现了READ COMMITTED隔离级别。

为了满足更多的应用需求，需要在Cedar中实现快照隔离级别（SNAPSHOT ISOLATION）。快照技术是一种多版本并发控制的协议及实现，应用于多数商业和开源数据库，例如Oracle和PostgreSQL。Cedar中的UPS端已采用快照技术，我们可以利用该技术实现事务级的快照隔离级别。

在Cedar中，MS端负责接收客户端的SELECT请求。但是，MS端在处理SELECT请求的时候，并没有将事务信息传递给UPS端。所以UPS无法感知与SELECT操作相关的事务信息，也就是UPS无法根据事务信息获取数据：（1）无法读取本事务修改但未提交的数据；（2）无法根据隔离级别读取本事务指定的版本数据。

因此，为了实现SNAPSHOT ISOLATION，以及避免上述的问题，我们需要将事务信息嵌入到SELECT语句的整个执行流程中。

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

客户端设置事务隔离级别、显式地执行事务的流程如下：

1. 通过SET语句设置本会话中事务的隔离级别；
2. 通过START TRANSACTION开启一个新的事务；
3. 执行读操作（SELECT）或者写操作；
4. 通过COMMIT显式地提交事务，或者通过ROLLBACK显式地回滚事务，或者事务执行失败隐式地回滚事务。

根据上述的流程，需要实现的功能如下：

首先Cedar的SELECT执行流程不涉及其所属事务，即读取操作没有关联其所属事务相关信息。所以第一个要解决的问题是，在执行读取操作执行过程中，事务信息的传递。也就是说，MS端需要保存事务信息，并在执行读取操作时，将事务信息传递给UPS端。

其次，记录和传递事务信息是为了在不同的隔离级别下面对数据版本的读取有不同的控制。因此在UPS处理读取请求时，执行流程需要满足不同隔离级别需求。

最后，对于客户端而言，需要提供设置新的隔离级别的SQL支持，这里需要关系到客户端会话初始时分配的锁信息。

1.4 性能指标

当系统全局的事务隔离级别设置为SNAPSHOT ISOLATION时，系统的性能（包括系统的吞吐量和客户端的响应延迟）与原有隔离级别下的性能保持一致。

2 模块设计

根据功能需求，SNAPSHOT ISOLATION的功能开发分为以下三个子模块：

- 事务信息的传递
- SELECT语句在UPS端的执行
- SQL的支持

2.1 事务信息的传递

2.1.1 总体流程

为了实现SNAPSHOT ISOLATION，MS端在处理SELECT操作时，需要将事务信息传递给UPS，事务信息的获取及传递流程如下：

1. MS端收到客户端的START TRANSACTION请求后，将该请求转发给UPS端。UPS收到START TRANSACTION请求后，会为其分配新的事务（包括事务描述符、事务开始时间戳），并将事务信息返回给MS端，MS将该事务信息保存至本地；
2. MS端收到客户端的SELECT请求后，首先从本地获取该客户端对应的事务信息，然后查询需要访问的CS集合，将事务信息打包进数据请求中，发送给相应的CS。CS端接收到数据请求后，获得其中的事务标识符，并将其打包进增量数据的请求中，发送给UPS；
3. UPS端收到读取请求时，可以获取该读取请求相关的事务信息，根据事务信息获取指定的版本数据并返回。

通过以上的流程可以看出，事务信息的传递分为两种：

传递类型	**说明**
事务信息的反传	UPS分配事务信息，并将该信息返回给MS
事务信息的正传	MS从本地获取事务信息，该信息最终传递给UPS

2.1.2 事务信息的反传

Cedar的ObTransID为事务信息结构体，其中包含了部分事务信息，该结构体主要用于事务执行更新操作的时候。当UPS分配完事务信息后，将ObTransID返回给MS，MS将其保存至本地。为了满足SNAPSHOT ISOLATION，需要在ObTransID中增加新的成员变量：事务开始时间戳和隔离级别信息。并且补全相关重置、序列化和反序列化以及获取序列化大小的成员函数。新增成员变量如下所示：

```
struct ObTransID
{
  ObTransID(): descriptor_(INVALID_SESSION_ID),
               ups_(),
               start_time_us_(0),
               trans_start_time_us_(0),
               isolation_level_(READ_COMMITED) {}
  int64_t trans_start_time_us_;    // 事务开始时间戳（逻辑）
  int32_t isolation_level_;        // 隔离级别
};
```

2.1.3 事务信息的正传

ObTransID的正传分为两个阶段：（1）MS端向CS端的传递；（2）CS端向UPS端的传递。两个阶段均需要包含读取请求相关的事务信息。

第一个阶段：MS收到客户端的SELECT 请求后，首先从本地获取到该客户端对应的ObTransID，然后将ObTransID打包进CS的读取请求中。ObSqlReadParam为该请求中的重要参数，因此我们为其添加ObTransID类型的成员变量来标识事务，以及对应的set、get、序列化以及反序列化方法。部分代码如下所示：

```
class ObSqlReadParam
{
  // 设置成员变量trans_id_
  virtual inline int set_trans_id(const ObTransID& trans_id);
  // 获取成员变量trans_id_
  virtual const ObTransID &get_trans_id() const;
  // 序列化
  int ObSqlReadParam::serialize_basic_param();
  // 反序列化
  int ObSqlReadParam::deserialize_basic_param();
  // 获取序列化大小
  int64_t get_basic_param_serialize_size(void) const;

  // 新增成员变量trans_id_
  ObTransID trans_id_;
}
```

第二个阶段：当CS需要从UPS获取增量数据时，在其请求中添加事务相关信息。ObReadParam为该请求中的重要参数，因此我们为其添加ObTransID类型的成员变量来标识事务，以及对应的set、get、序列化以及反序列化方法。部分代码如下所示：

```
class ObReadParam
{
  // 设置成员变量trans_id_
  void set_trans_id(const ObTransID& trans_id);
  // 获取成员变量trans_id_
  ObTransID get_trans_id(void) const;

  // 新增成员变量trans_id_
  ObTransID trans_id_;
}
```

2.2 SELECT语句在UPS的执行

2.2.1 总体流程

在Cedar中，UPS负责增量数据以及事务信息的维护。UPS接收到CS发来的读取请求后，会首先判断该读取请求的类型，然后根据读取请求的类型执行相应的流程。

在UPS端，读取多行记录的流程类似于读取单行记录。因此我们在描述读取流程的时候，可以不用区分读取请求的类型。在读取请求中加入事务信息（ObTransID）后，读取的执行流程为：

1. UPS在接收到读取请求后，反序列化该读取请求，并获得该请求中的事务信息ObTransID；
2. 为读取请求分配新的只读会话（ROSessionCtx），该只读会话与请求读取的事务没有任何关系；
3. 判断ObTransID中的隔离级别。如果隔离级别是REPEATABLE-READ，并且请求读取的事务的开始时间戳为非0值，则将该只读会话的开始时间戳设置为ObTransID中的事务开始时间戳；否则认为隔离级别为READ-COMMITTED，将该只读会话的开始时间戳设置为当前已公开的最大事务号（该事务号为单调递增的时间戳）；
4. 访问内存表，获取相应的记录，并且遍历每一个记录的数据块链表，根据事务标识符和事务开始时间戳获取相应的数据块。

2.2.2 只读会话中事务信息的添加

为了实现UPS根据事务信息获取指定的版本数据，需要在只读会话（ROSessionCtx）中记录事务的开始时间戳以及事务标识符，因此需要在BaseSessionCtx类中修改或增加新的成员变量以及成员方法，如下所示：

```
class BaseSessionCtx
{
  public:
    virtual void reset();
    void set_trans_start_time(const int64_t trans_start_time);
    int64_t get_trans_start_time() const;
    uint32_t get_trans_descriptor() const;
    void set_trans_descriptor(const uint32_t trans_descriptor);
  private:
    int64_t trans_start_time_;    // 事务开始时间戳
    uint32_t trans_descriptor_;   // 事务描述符
};
```

> **注意**：在调用reset方法的时候，需要重置事务开始时间戳以及事务描述符。

2.2.3 MemTable相关方法的修改

当只读会话（ROSessionCtx）包含有读取请求的事务信息后，UPS便可以将该会话作为参数来获取数据。无论UPS处理读取单行记录的请求，还是处理读取范围记录的请求，都要使用MemTableGetIter来获取数据。由于在只读会话中新增了事务开始时间戳以及事务描述符，因此在获取和遍历每一条记录的数据块链表的时候，需要根据新增的成员变量来获取指定版本的数据。涉及修改的方法如下：

```
// 设置查询相关信息，传入记录
void MemTableGetIter::set_
(const TEKey &te_key,
 const TEValue *te_value,
 const ColumnFilter *column_filter,
 const bool return_rowkey_column,
 const BaseSessionCtx *session_ctx)

// 获取记录的数据块链表
ObCellInfoNode *MemTableGetIter::get_list_head_()

// 根据事务标识符来判断是否可以遍历未提交链表
bool MemTableGetIter::read_uncommited_data_()

// 遍历下一个数据块
int MemTableGetIter::next_cell()

// 根据事务开始时间戳来判断是否结束数据块链表的遍历
bool MemTableGetIter::trans_end_(const ObObj &value)
```

2.3 SQL的支持

2.3.1 设计概述

为了在用户支持上与大多数数据库系统保持兼容，对应ANSI定义的四种隔离级别，本文介绍Cedar0.2版本中新增的Snapshot Isolation，在提供服务时对应REPEATABLE-READ的SQL支持。

客户端需要通过SQL语句来设置隔离级别，以满足其业务需求。Cedar已支持隔离级别的设置，设置会话隔离级别为REPEATABLE READ的SQL语句如下：

```
> SET @@SESSION.tx_isolation='REPEATABLE-READ';
```

当开启一个新的事务的时候，我们需要对该事务进行初始化。在初始化的过程中，需要为事务分配必要的资源，例如，根据隔离级别分配相应的锁信息。因为事务在执行更新语句时，需要对记录加行锁，UPS需要根据不同的隔离级别，执行不同的加锁流程。而Cedar没有REPEATABLE READ隔离级别对应的锁信息。因此，我们需要添加相应的锁信息类。

2.3.2 部分实现

当客户端显式地开启一次事务时，需要显式地执行START TRANSACTION语句。当UPS收到OB_START_TRANSACTION请求后，则需要为客户端分配会话（session），用于保存事务信息，该过程包括为其分配锁信息。

UPS根据该事务的隔离级别，通过LockMgr的assign函数为其分配锁信息。因此在UPS中，不同的隔离级别通过实现ILockInfo接口，来实现不同的加锁机制。因此，我们需要为SNAPSHOT ISOLATION实现ILockInfo接口，该实现类为RRLockInfo。RRLockInfo类的方法及成员变量如下所示：

```
class RRLockInfo : public ILockInfo
{
  public:
    RRLockInfo(RWSessionCtx &session_ctx);
    ~RRLockInfo();
    int on_trans_begin();
    int on_read_begin(const TEKey &key, TEValue &value);
    int on_write_begin(const TEKey &key, TEValue &value);
    void on_trans_end();
    void on_precommit_end();
    int cb_func(const bool rollback, void *data, BaseSessionCtx &se
ssion);
  private:
    RWSessionCtx &session_ctx_;
    RowExclusiveUnlocker row_exclusive_unlocker_;
    CallbackMgr callback_mgr_;
};
```

3 模块接口

事务信息ObTransID从MS端向CS端的传递，需要在ObSqlReadParam添加相应的成员变量和成员方法，设置成员变量ObTransID及获取成员变量ObTransID的方法如下：

```
/**
 * @brief set the member variable ObTransID
 * @param[in] trans_id, input transaction ID
 */
int set_trans_id(const ObTransID& trans_id);

/**
 * @brief get the member variable ObTransID
 * @return the member variable ObTransID
 */
const ObTransID &get_trans_id() const;
```

事务信息ObTransID从CS端向UPS端的传递，需要在ObReadParam添加相应的成员变量和成员方法，设置成员变量ObTransID及获取成员变量ObTransID的方法如下：

```
/**
 * @brief set the member variable ObTransID
 * @param[in] trans_id, input transaction ID
 */
void set_trans_id(const ObTransID& trans_id);

/**
 * @brief get the member variable ObTransID
 * @return the member variable ObTransID
 */
ObTransID get_trans_id(void) const;
```

4 使用限制条件和注意事项

- 避免丢失更新采用Cedar原有的SELECT…FOR UPDATE来实现。
- 尚无START TRANSACTION WITH CONSISTENT SNAPSHOT语句的功能支持，该语句的表现形式与START TRANSACTION一致。
- 暂时只在显式事务中有效，不考虑AUTOCOMMIT（系统内部未做处理）。
- 暂时对应ANSI的REPEATABLE READ，即客户端设置隔离级别为REPEATABLE READ时，系统内部采用的是SNAPSHOT ISOLATION。
- 暂时全局的REPEATABLE-READ隔离级别设置需要在退出会话重新进入后方可生效。
- 当客户端会话隔离级别为REPEATABLE READ时，INSERT、UPDATE、DELETE、SELECT…FOR UPDATE的执行流程保持与READ COMMITTED一致。
- 隔离级别的设置要严格按照本文给出的表达形式，不能写成”READ-COMMITTED”、”REPEATABLE-READ”以外的输入形式，否则系统将进入不可服务状态。