LAB3-DOC

Part 1: Simple Database without Transaction

第一部分的实现较为简单，主要是为了后面的两个部分提供基础，多种实现可以直接调用extent_client的接口，在此部分定义并实现了key2id的hash函数如下

```
    extent_protocol::extentid_t ydb_server::key2id(const std::string key)
{
    unsigned int seed = 131;
    extent_protocol::extentid_t hash = 0;
    for (unsigned int i = 0; i < key.length(); i++)
    {
        hash = hash * seed + key[i];
    }
    return (hash % 1022) + 2;
}
```

Part 2: Two Phase Locking

这一部分要实现2PL，所以定义了一部分数据类型

```
class transaction_entry
{
public:
    entry_map entrymap;
    string_set lockmap;
};
```

这是一个transaction的信息，entrymap储存了将要对数据库的操作，lockmap储存了其所拥有的锁。

Part 2A: Simple Design

普通实现中不需要考虑死锁，所以只需要在每次操作前检查一下锁是否在本transaction内即可。

Part 2B: Deadlock Detecting

这部分中要考虑多线程竞争中的死锁问题，所以增加了两个全局变量。

```
    std::map<std::string,int> global_lock;
    bool waiting_graph[GRAPH_SIZE][GRAPH_SIZE];
```

其中global_lock存储了锁在哪个transaction手中,waiting_graph存储了一个等待图,在acquire中通过分析等待图中是否有环来确定是否让transaction abort,其代码如下:

```
ydb_protocol::status ydb_server_2pl::acquire(ydb_protocol::transaction_id id,
std::string key)
{
    printf("acquire: transaction:%d key:%s\n",id,key.c_str());
    lc->acquire(GLOBAL_LOCK_LOCK);
```

```
    std::map<int, transaction_entry>::iterator iter =
transaction_cache.find(id);
    if (iter == transaction_cache.end())
    {
        printf("no transaction: %d\n", id);
        lc->release(GLOBAL_LOCK_LOCK);
        return ydb_protocol::TRANSIDINV;
    }
    transaction_entry transaction = iter->second;
    string_set local_lock = transaction.lockmap;
    // the transaction doesn't have lock now
    if (local_lock.find(key) == local_lock.end())
    {
        // others don't have lock too
        if (global_lock.find(key) == global_lock.end())
        {
            printf("transaction %d need acquire lock %s\n",id,key.c_str());
            lc->release(GLOBAL_LOCK_LOCK);
            lc->acquire(key2id(key));
            lc->acquire(GLOBAL_LOCK_LOCK);
            printf("transaction %d acquire lock %s
successfully\n",id,key.c_str());
            iter->second.lockmap.insert(key);
            global_lock[key] = id;
        }
        //others has lock
        else
        {
            int depend_id = global_lock.find(key)->second;
            waiting_graph[id][depend_id] = 1;
            if (detect_cycle())
            {
                printf("cycle, abort transaction:%d\n", id);
                int a;
                lc->release(GLOBAL_LOCK_LOCK);
                transaction_abort(id, a);
                lc->acquire(GLOBAL_LOCK_LOCK);
                for(int i = 0;i<GRAPH_SIZE;i++)
                {
                    waiting_graph[id][i]=0;
                    waiting_graph[i][id]=0;
                }
                lc->release(GLOBAL_LOCK_LOCK);
                return ydb_protocol::ABORT;
            }
            else
            {
                printf("transaction %d need wait for lock %s in transaction
%d\n",id,key.c_str(),depend_id);
                lc->release(GLOBAL_LOCK_LOCK);
                lc->acquire(key2id(key));
                lc->acquire(GLOBAL_LOCK_LOCK);
                waiting_graph[id][depend_id] = 0;
                printf("transaction %d acquire lock %s
successfully\n",id,key.c_str());
                iter->second.lockmap.insert(key);
                global_lock[key] = id;
            }
```

```
        }
    }
    else{
        printf("transaction %d has lock %s \n", id, key.c_str());
    }
    lc->release(GLOBAL_LOCK_LOCK);
    return ydb_protocol::OK;
}
```

其中大部分操作都有注释,GLOBAL_LOCK_LOCK是为了保护全局变量而设立的锁,在此处lab中取1025.

Part 3: Optimistic Concurrency Control

这一部分要实现OCC，所以定义了一部分数据类型

```
class transaction_occ
{
public:
    entry_map entrymap;
    string_map readset;
};
```

这是一个transaction的信息，entrymap储存了将要对数据库的操作，readset存储了其所读出的key和value.

重点在于commit阶段,代码如下:

```
ydb_protocol::status
ydb_server_occ::transaction_commit(ydb_protocol::transaction_id id, int &)
{
    lc->acquire(COMMIT_LOCK);
    // lab3: your code here
    printf("transaction_commit: %d\n", id);
    transaction_occ transaction;
    if (find_transaction(id, transaction) == ydb_protocol::TRANSIDINV)
    {
        printf("transaction_commit: no transaction: %d\n", id);
        return ydb_protocol::TRANSIDINV;
    }

    entry_map entrymap = transaction.entrymap;
    string_map readset = transaction.readset;

    // validation
    if (validation(readset))
    {
        //commit
        entry_map::iterator iter = entrymap.begin();
        for (; iter != entrymap.end(); iter++)
        {
            std::string key = iter->first;
            if (iter->second.kind == EDIT)
            {
                ec->put(key2id(key), iter->second.value);
            }
            else
```

```
            {
                ec->put(key2id(key),"");
            }
        }
    }
    else
    {
        //abort
        printf("validation false, abort transaction:%d\n", id);
        int a;
        if(transaction_abort(id, a)==ydb_protocol::OK)
        {
            lc->release(COMMIT_LOCK);
            return ydb_protocol::ABORT;
        }
        else{
            printf("abort error");
        }
    }

    if (remove_transaction(id) == ydb_protocol::TRANSIDINV)
    {
        printf("transaction_commit: no transaction: %d\n", id);
        return ydb_protocol::TRANSIDINV;
    }
    lc->release(COMMIT_LOCK);
    return ydb_protocol::OK;
}
```

其中关键部分均有注释,validation函数会遍历readset并确定中的值是否和数据库中相同,若相同commit,反之abort.

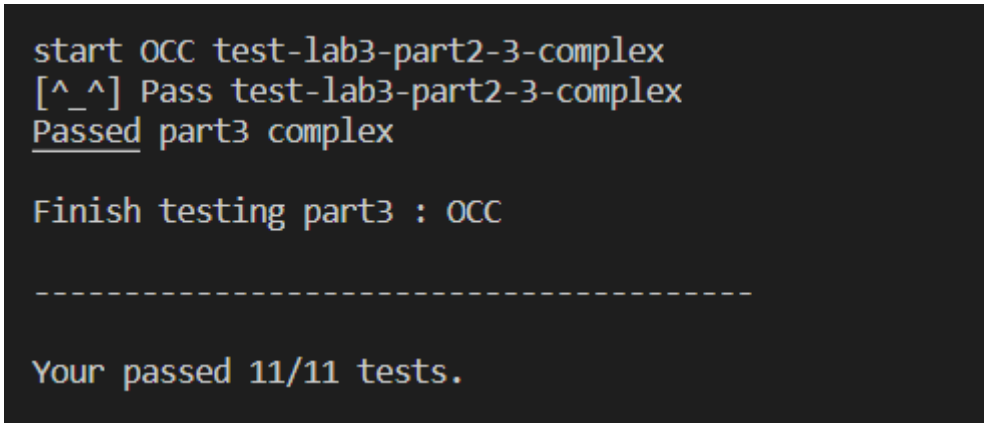
最终结果

grade_lab2.sh

```
stu@988e6025217d:~/devlop/lab1$ ./grade_lab2.sh
starting ./lock_server 23482 > lock_server.log 2>&1 &
starting ./extent_server 23476 > extent_server.log 2>&1 &
starting ./yfs_client /home/stu/devlop/lab1/yfs1 23476 23482 > yfs_client1.log 2>&1 &
starting ./yfs_client /home/stu/devlop/lab1/yfs2 23476 23482 > yfs_client2.log 2>&1 &
Passed part1 A
Passed part1 B
Passed part1 C
Passed part1 D
Passed part1 E
Passed part1 G (consistency)
Lab2 part 1 passed
Concurrent creates: OK
Concurrent creates of the same file: OK
Concurrent create/delete: OK
Concurrent creates, same file, same server: OK
Concurrent writes to different parts of same file: OK
Passed part2 A
Create/delete in separate directories: tests completed OK
Passed part2 B
yfs_client: no process found

Score: 120/120
```

grade_lab3.sh



Extra

- 更换了librpc64.a后，同样能通过测试。
- 主要优化点在于减小了全局变量锁的粒度。