2-Phase Commit

2.1 Overview

A **transaction** can be defined as a very small unit or task of a program which cannot be further divided in to smaller sub-tasks. A transaction must maintain Atomicity, Consistency, Isolation, and Durability – commonly known as **ACID** properties – in order to ensure accuracy, completeness, and data integrity. **Distributed transactions** are transactions in which two or more network hosts are involved. We will refer to these hosts as *COHORTS*.

The **Two-phase commit protocol (2PC)** is a distributed algorithm that coordinates all the *COHORTS* that participate in a distributed transaction on whether to **commit** or **abort** the transaction. The protocol is fault-tolerant i.e, the protocol achieves its goal even in cases of temporary system failure. We will begin by giving an informal definition of the service.

2.2 Intended Service Informally Stated

The read-write-lock service has four functions that can be called by local users in the environment. The service calls the inputted functions simultaneously. The three functions in the service are as follows:

- `put(key, value)` : creates or updates a key-value pair in the store
  - returns `COMMIT` or `ABORT`
- `get(key):` returns the value for the given key in the store
  - returns `(value, COMMIT)` or `ABORT` on completion
- `delete(key)` : deletes the key-value pair in the store
  - returns `COMMIT` or `ABORT`

2.3 Implementation `Imp_0`

Here class `Imp_0` that implements the service will be discussed. Here we assume the availability of a **failure-free store**.

class Imp_0 It has three functions: `put(key, value)` , `get(key)` , and `delete(key)` . The class maintains the following variables:

- `db_lock` (lock): lock that protects the database
- `store` (map): Store that holds the key-value pairs

In [6]:
```
from threading import Lock

class Imp:
```

```
    def __init__(self):
        self.store = dict()
        self.db_lock = Lock()

    def put(self, key, val):
        self.db_lock.acquire()
        self.store[key] = val
        print("put call", key)
        self.db_lock.release()

    def get(self,key):
        self.db_lock.acquire()
        try:
            rval = self.store[key]
        except:
            rval = "ABORT"
            print(key, " was not found")
        self.db_lock.release()
        return rval

    def delete(self,key):
        self.db_lock.acquire()
        print("delete call", key)
        try:
            self.store.pop(key)
        except:
            print(key, " was not found")
        self.db_lock.release()



if __name__ == '__main__':
    imp = Imp()
```

2.4 User Program

To run `Imp_0` run `python 2PC/user.py -num_threads <num_threads> -op_names <op_names> -imp "Imp_0"`

- `<num_threads>` : number of user threads making calls
- `<op_names>` : names of all the operations where each operation is separated by a space. For eg: `"put(k1,v1) put(k2,v2) delete(k1) get(k1)"`
- `<imp>` : name of the implementation you are running. For eg: `"Imp_0"`
- `<num_users>` : name of users running the implementation (only valid for `"Imp_2"`)

The program creates the number of threads and begins execution. For `"Imp_0"` and `"Imp_1"` , all the operations are stored in `op_names` . The threads dequeue the first operation in `op_names` , parse it and begin the operation. For `"Imp_2"` , the process is a little different as we work with multiple user instances. When using `"Imp_2"` , the compiler will prompt the sequence for each `User` . Then, the `Coordinator` (defined in `Imp_2`) will manage the processing of all the operations by dequeuing each `User` and executing its operations. Any invalid operation returns `Invalid Operation` .

In [ ]:
```
import threading
import sys,argparse
```

```
from Imp_1 import Imp as Imp_1
from Imp_0 import Imp as Imp_0
from coordinator import Coordinator
import multiprocessing
import time


class User():

    def __init__(self, num_threads, op_names, imp_name):
        self.num_threads = num_threads
        self.op_names = op_names
        self.user_threads = []

        if(imp_name == "Imp_0"):
            self.imp_name = imp
            self.imp = Imp_0()
            for j in range(self.num_threads):
                t = threading.Thread(target=self.perform_ops)
                self.user_threads.append(t)
                t.start()

            print("Database after ", self.imp.store)
        elif(imp_name == "Imp_1"):
            self.imp_name = imp
            self.imp = Imp_1()
            for j in range(self.num_threads):
                t = threading.Thread(target=self.perform_ops)
                self.user_threads.append(t)
                t.start()


            print("Database A after ", self.imp.store_A)
            print("Database B after ", self.imp.store_B)
        elif(imp_name == "Imp_2"):
            self.imp_name = imp_name

        else:
            print("Not a valid implementation")


    def perform_ops(self):
        if(self.imp_name == "Imp_2"):
            pid = multiprocessing.current_process().name
            print('Current process', pid)
        else:
            tid = multiprocessing.current_thread().name
            print('Starting user thread', tid)

        while(self.op_names):
            i = self.op_names.pop(0)
            s = i.split("(")

            func = s[0]
            key = "key"
            value = "value"


            if(func == "put"):
                param = s[1].split(",")
```

```
                key = param[0]
                value = param[1].replace(")", "")

                self.imp.put(key, value)

            elif(func == "get"):
                key = s[1].replace(")", "")
                self.imp.get(key)

            elif(func == "delete"):
                key = s[1].replace(")", "")
                self.imp.delete(key)

            else:
                print("Invalid Arguments")

if __name__ == '__main__':
    p = argparse.ArgumentParser(
      usage='python 2PC/user.py -num_threads -op_names -imp -num_users',
      description='create a user to drive the given implementation')

    p.add_argument("-imp", type=str, help='implementation running')
    p.add_argument("-num_users", type=int, help='number of users', default = 1)
    p.add_argument("-num_threads", type=int, help='number of user threads', defa

    args = p.parse_args(sys.argv[1:])
    print("Implementation args:", args)

    num_threads = args.num_threads
    imp_name = args.imp
    op_names = []
    num_users = args.num_users

    print("Starting ", imp_name)

    if(imp_name == "Imp_2"):
        users = []
        for j in range(num_users):
            print("Enter op_names for User ", j + 1 , ": ")
            op_name = input()
            user = User(1, op_name.split(" "), imp_name)
            users.append(user)

        coordinator = Coordinator(users)
        coordinator.manage_tasks()

    else:
        p.add_argument("-op_names", type=str, help='name of the operations')
        op_names = args.op_names
        op_names = self.op_names.split(" ") #list of all operations
        user = User(num_threads, op_names, imp)

        if hasattr(user.imp, '__end__'):
            for t in user.user_threads:
              t.join()
            user.imp.__end__()
```

Sample output on `python 2PC/user.py -num_threads 2 -op_names "put(k1,v1) put(k2,v2) delete(k1) put(k3,v3) delete(k3) get(k1) put(k2,v2)`

put(k4,v4)" :

```
 starting 2PC.user
('2PC args:', Namespace(num_threads=2, op_names='put(k1,v1) put(k2,v2)
delete(k1) put(k3,v3) delete(k3) get(k1) put(k2,v2) put(k4,v4)'))
('starting user thread', 'Thread-1')
('Operation being performed is ', 'put(k1,v1)', 'by ', 'Thread-1')
('put call', 'k1')
('Operation being performed is ', 'put(k2,v2)', 'by ', 'Thread-1')
('put call', 'k2')
('Operation being performed is ', 'delete(k1)', 'by ', 'Thread-1')
('delete call', 'k1')
('Operation being performed is ', 'put(k3,v3)', 'by ', 'Thread-1')
('put call', 'k3')
('Operation being performed is ', 'delete(k3)', 'by ', 'Thread-1')
('delete call', 'k3')
('Operation being performed is ', 'get(k1)', 'by ', 'Thread-1')
('k1', ' was not found')
('Operation being performed is ', 'put(k2,v2)', 'by ', 'Thread-1')
('put call', 'k2')
('Operation being performed is ', 'put(k4,v4)', 'by ', 'Thread-1')
('put call', 'k4')
('starting user thread', 'Thread-2')
('database after =', {'k2': 'v2', 'k4': 'v4'})
```

2.5 Implementation `Imp_1`

To run `Imp_1` run `python 2PC/user.py –num_threads <num_threads> –op_names <op_names> –imp "Imp_1"`

`Imp_1` is a single-process multi-threaded implementation. We have two distinct locks and stores in this implementation. *class* Imp_1 It has three functions: `put(key, value)` , `get(key)` , and `delete(key)` . The class maintains the following variables:

- `store_A` (map): Store that holds the key-value pairs from keys values starting with `ASCII` values betweem 32 and 79
- `store_B` (map): Store that holds the key-value pairs from keys values starting with `ASCII` values betweem 80 and 126
- `db_lock_A` (lock): lock that protects the database A
- `db_lock_B` (lock): lock that protects the database B

In [2]:
```
from threading import Lock

class Imp:

    def __init__(self):
        self.store_A = dict()
        self.store_B = dict()
```

```
        self.db_lock_A = Lock()
        self.db_lock_B = Lock()

    def put(self, key, val):
        if(chr(ord(key[0])) < 80):
            self.db_lock_A.acquire()
            self.store_A[key] = val
            print("put call in Database A", key)
            self.db_lock_A.release()
        else:
            self.db_lock_B.acquire()
            self.store_B[key] = val
            print("put call in Database B", key)
            self.db_lock_B.release()


    def get(self,key):
        if(chr(ord(key[0])) < 80):
            self.db_lock_A.acquire()
            try:
                rval = self.store_A[key]
            except:
                rval = "ABORT"
                print(key, " was not found")
            self.db_lock_A.release()
        else:
            self.db_lock_B.acquire()
            try:
                rval = self.store_B[key]
            except:
                rval = "ABORT"
                print(key, " was not found")
            self.db_lock_B.release()
        return rval

    def delete(self,key):
        if(chr(ord(key[0])) < 80):
            self.db_lock_A.acquire()
            print("delete call", key)
            try:
                self.store_A.pop(key)
            except:
                print(key, " was not found")
            self.db_lock_A.release()
        else:
            self.db_lock_B.acquire()
            print("delete call", key)
            try:
                self.store_B.pop(key)
            except:
                print(key, " was not found")
            self.db_lock_B.release()



if __name__ == '__main__':
    imp = Imp()
```

Sample output on python 2PC/user.py -num_threads 2 -op_names "put(k1,v1) put(k2,v2) delete(k1) put(k3,v3) delete(k3) get(k1) put(k2,v2)

put(B4,v4)" :

```
 starting 2PC.user
('2PC args:', Namespace(num_threads=2, op_names='put(k1,v1) put(k2,v2)
delete(k1) put(k3,v3) delete(k3) get(k1) put(k2,v2) put(B4,v4)'))
('starting user thread', 'Thread-1')
('Operation being performed is ', 'put(k1,v1)', 'by ', 'Thread-1')
('put call in Database B', 'k1')
('Operation being performed is ', 'put(k2,v2)', 'by ', 'Thread-1')
('put call in Database B', 'k2')
('Operation being performed is ', 'delete(k1)', 'by ', 'Thread-1')
 ('starting user thread', 'Thread-2')
('Operation being performed is ', 'put(k3,v3)', 'by ', 'Thread-2')
('put call in Database B', 'k3')
(('delete call', 'k1')
('Operation being performed is ', 'get(k1)', 'by ', 'Thread-1')
('k1', ' was not found')
('Operation being performed is ', 'put(k2,v2)', 'by ', 'Thread-1')
('put call in Database B', 'k2')
('Operation being performed is ', 'put(B4,v4)', 'by ', 'Thread-1')
('put call in Database A', 'B4')
('Database A', 'Thread-1', {'B4': 'v4'})
('Database B', 'Thread-1', {'k3': 'v3', 'k2': 'v2'})
'Operation being performed is ', 'delete(k3)', 'by ', 'Thread-2')
('delete call', 'k3')
('Database A', 'Thread-2', {'B4': 'v4'})
('Database B', 'Thread-2', {'k2': 'v2'})
```

2.6 Implementation `Imp_2`

For the distributed implementation, we introduce a new class called the **coordinator**. The `Coordinator()` object found in `coordinator.py` will process and manage the operations between the various users and the database. The Coordinator uses the Multiprocessing.Manager() object. The Manager() controls a server process which holds Python objects and allows other processes to manipulate them using proxies. The implementation used these managed proxies to make changes within the database. The Manager() object takes care of message passing and locks.

In [3]:
```
from Imp_2 import Imp as Imp_2
from multiprocessing import Process, Manager

class Coordinator:
    def __init__(self, users):
        self.users = users

    def manage_tasks(self):
        with Manager() as manager:
            storeA = manager.dict()
```

```
            storeB = manager.dict()

            self.imp = Imp_2(storeA, storeB)

            while(self.users):
                user1 = self.users.pop(0)
                user1.imp = self.imp

                try:
                    user2 = self.users.pop(0)
                except:
                    user2 = None
                    print("No more users")

                p1 = Process(target=user1.perform_ops)
                p1.start()

                if(user2 != None):
                    user2.imp = self.imp
                    p2 = Process(target=user2.perform_ops)
                    p2.start()
                    p2.join()

                p1.join()

            print("Database A", self.imp.store_A)
            print("Database B", self.imp.store_B)
```

`Imp_2` is a distributed implementation involving 2 processors and `num_users`. We have two distinct stores in this implementation, both managed by the Manager object explained above.

class `Imp_2`

It has three functions: `put(key, value)`, `get(key)`, and `delete(key)`. The class maintains the following variables:

- `store_A` (map): Store that holds the key-value pairs from keys values starting with `ASCII` values betweem 32 and 79
- `store_B` (map): Store that holds the key-value pairs from keys values starting with `ASCII` values betweem 80 and 126

This implementaion is very similar to the previous implementations we have seen, except that `Imp_2` receives a reference to the store through the `Coordinator()`.

In [ ]:

```
class Imp:

    def __init__(self, StoreA, StoreB):
        self.store_A = StoreA
        self.store_B = StoreB

    def put(self, key, val):
        if(ord(key[0]) < 80):
            self.store_A[key] = val
            print("put call in Database A", key)
        else:
            self.store_B[key] = val
            print("put call in Database B", key)
```

```
    def get(self,key):
        if(ord(key[0]) < 80):
            try:
                rval = self.store_A[key]
                print(key, " was found")
            except:
                rval = "ABORT"
                print(key, " was not found")
        else:
            try:
                rval = self.store_B[key]
            except:
                rval = "ABORT"
                print(key, " was not found")
        return rval

    def delete(self,key):
        if(ord(key[0]) < 80):
            print("delete call for ", key)
            try:
                self.store_A.pop(key)
            except:
                print(key, " was not found")
        else:
            print("delete call for ", key)
            try:
                self.store_B.pop(key)
            except:
                print(key, " was not found")


if __name__ == '__main__':
    imp = Imp()
```

Sample output on `python 2PC/user.py -num_threads 1 -imp "Imp_2" -num_user 6`

```
 ('Implementation args:', Namespace(imp='Imp_2', num_threads=1,
num_users=6))
('Starting ', 'Imp_2')
('Enter op_names for User ', 1, ': ')
"put(k1,v1) put(k2,v2) delete(k1) put(k3,v3) delete(k3) get(k1)
put(k2,v2) put(B4,v4)"
('Enter op_names for User ', 2, ': ')
"put(k3,v4) delete(k2)"
('Enter op_names for User ', 3, ': ')
"put(B4,v6) put(11,22) delete(k1)"
('Enter op_names for User ', 4, ': ')
"put(11,33) get(B4) delete(k1)"
('Enter op_names for User ', 5, ': ')
"put(11,33) get(B4) delete(k1)"
('Enter op_names for User ', 6, ': ')
"put(k3,5) delete(k2) get(11) put(k2,v5)"
```

```
('Current process', 'Process-2')
('Current process', 'Process-3')
('put call in Database B', 'k1')
('put call in Database B', 'k2')
('delete call for ', 'k1')
('put call in Database B', 'k3')
('delete call for ', 'k3')
('put call in Database B', 'k3')
('delete call for ', 'k2')
('k1', ' was not found')
('put call in Database B', 'k2')
('put call in Database A', 'B4')
('Current process', 'Process-5')
('Current process', 'Process-4')
('put call in Database A', '11')
('put call in Database A', 'B4')
('B4', ' was found')
('delete call for ', 'k1')
('put call in Database A', '11')
('delete call for ', 'k1')
('k1', ' was not found')
('k1', ' was not found')
('Current process', 'Process-6')
('Current process', 'Process-7')
('put call in Database A', '11')
('B4', ' was found')
('delete call for ', 'k1')
('k1', ' was not found')
('put call in Database B', 'k3')
('delete call for ', 'k2')
('11', ' was found')
('put call in Database B', 'k2')
```

In [ ]: