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 three functions are as follows:

- `put(key, value)` : creates or updates a key-value pair in the store
  - can only be called if `start_process()` returns `COMMIT`
  - 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 2PC 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>`

- `<num_threads>` : number of user threads making calls
- `<op_names>` : names of all the operations where each operartion is seperated by a space. For eg: `"put(k1,v1) put(k2,v2) delete(k1) get(k1)"`

The program creates the number of threads and begins execution. All the operations are stored in `op_names` . The threads dequeue the first operation in `op_names` , parse it and begin the operation. Any invalid operation returns `Invalid Operation` .

In [ ]:
```
import threading
import sys,argparse
from Imp_0 import Imp

class User():

    def __init__(self, argv):
        print('starting 2PC.user')

        p = argparse.ArgumentParser(
```

```
      usage='python 2PC/user.py -num_threads -op_names',
      description='create a user to drive the given 2PC implementation')

    p.add_argument("-num_threads", type=int, help='number of user threads',

    p.add_argument("-op_names", type=str)


    args = p.parse_args(argv[1:])

    print("2PC args:", args)
    self.num_threads = args.num_threads
    self.op_names = args.op_names

    self.op_names = self.op_names.split(" ")

    self.user_threads = []
    self.imp = Imp()


    for j in range(self.num_threads):
      t = threading.Thread(target=self.perform_ops)
      self.user_threads.append(t)
      t.start()


    print("Committing...")
    print("database after commit =", self.imp.store)




def perform_ops(self):
    tid = threading.current_thread().name
    print('starting user thread', tid)

    while(self.op_names):
        i = self.op_names.pop(0)
        print('i is', i, tid)
        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 Operation")


if __name__ == '__main__':
  user = User(sys.argv)
  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')
 Committing...
('database after commit =', {'k2': 'v2', 'k4': 'v4'})
```

2.5 Implementation `Imp_1` (Distributed)

To run `Imp_1` run `python 2PC/user.py -num_threads <num_threads> -op_names <op_names>` and make a slight modification to the `User Program` by changing `from Imp_0 import Imp` in *line 3* to `from Imp_1 import Imp`

This implementation is *distributed*. 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 [ ]:

```
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')
('Thread-1', ' is Committing...')
('Database A after commit by', 'Thread-1', {'B4': 'v4'})
('Database B after commit by', 'Thread-1', {'k3': 'v3', 'k2': 'v2'})
'Operation being performed is ', 'delete(k3)', 'by ', 'Thread-2')
('delete call', 'k3')
('Thread-2', ' is Committing...')
('Database A after commit by', 'Thread-2', {'B4': 'v4'})
('Database B after commit by', 'Thread-2', {'k2': 'v2'})
```

In [ ]:

In [ ]: