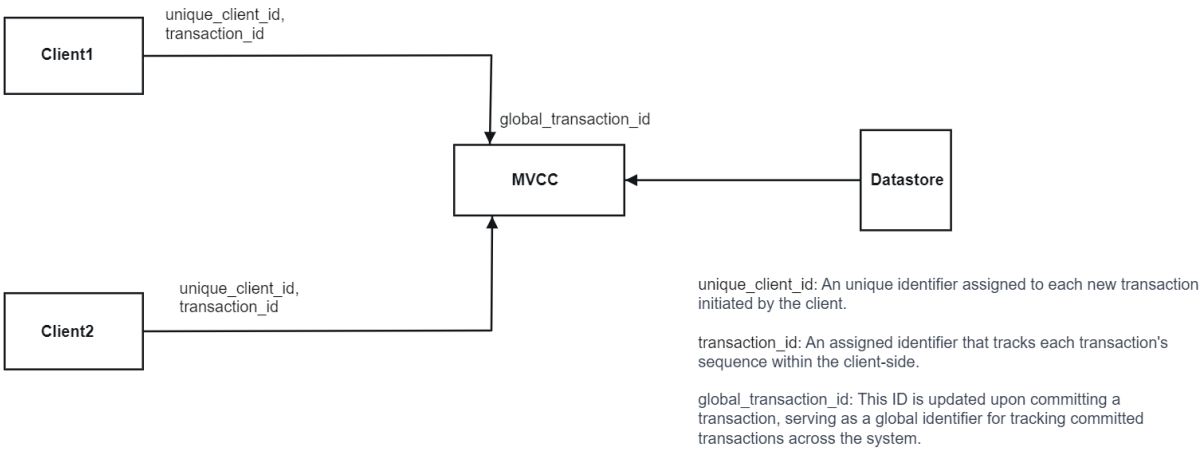
Implement MVCC for Python Pickle



Pickle

In the MVCC implementation, we used pickle to store and retrieve objects, employing them for versioning within the MVCC system.

ZMQ

Communication between client and server is established using ZMQ.

In our Multi-Version Concurrency Control (MVCC) implementation, we used ZeroMQ (ZMQ) for efficient communication between components.The MVCC system utilizes ZMQ sockets to establish communication between clients and the server (MVCC in this case) .

We employ specific ZMQ patterns, such as Request-Reply, to handle transaction initiation, data retrieval, and other interactions.

Additionally, we incorporated the use of JSON in the ZeroMQ (ZMQ) communication, enhancing interoperability and message serialization between different components of the system.

JSON request & response format for different operations

Start_transaction operation

request_data = {

"type": "start"

}

response_data = {

"transaction_id": transaction_id,

"unique_client_id": unique_client_id

}

Read operation

request_data = {

"type": "read",

"transaction_id": transaction_id,

"client_transaction_id": client_transaction_id

}

response_data = {

"value": value,

}

Write Operation

```
request_data = {
        "type": "write",
        "value": str(value),
        "transaction_id": transaction_id,
        "unique_client_id": client_transaction_id
}
response_data = {
        "value": received_data['value'],
      "transaction_id": received_data ['transaction_id'],
      "unique_client_id": received_data['unique_client_id']
}
```

Commit Operation

```
request_data = {
      "type": "commit",
      "transaction_id": transaction_id,
      "unique_client_id": client_transaction_id
}
response_data = {
      "value": "success"
}
```

For eg:- *{*

global_transaction_id : 0
transaction_id : 0,
Unique_client_id: ,1
Balance: 100
}

MVCC(Multiversion concurrency control)

- Avoid read delays
- Maintains consistency
- Maintains versioning
- Handle conflicts while concurrent transactions are committing

AVOID READ DELAYS

- If two transactions hitting the same Mvcc and datastore are reading the same data concurrently, no locks should be provided and a successful read should happen.

MAINTAINS CONSISTENCY

- The data will not be reflected to another transaction until it get successfully committed

MAINTAINS VERSIONING ON WRITE

- On each write of the transaction, a new version of data is created.

HANDLE CONFLICTS

- On each commit the MVCC is responsible to increment the Global Transaction Id , and whenever a new transaction is ready to commit , the transaction id is to be compared with the global transaction id and commits are handled.

TEST CASES

TESTCASE 1: Read, Write and Commit from a single client

TRANSACTION 1
Read *{* *global_transaction_id : 0* *transaction_id : 0,* *Unique_client_id: ,1* *Balance: 100* *}*
Write *{* *global_transaction_id : 0* *transaction_id : 0,* *Unique_client_id: ,1* *Balance: 110* *}*
Commit *{* *global_transaction_id : 1* *transaction_id : 1,* *Unique_client_id: ,1* *Balance: 110* *}*

TEST CASE 2 : Successful Read of consistent data

TRANSACTION 1	TRANSACTION 2
Read *{* *global_transaction_id : 0* *transaction_id : 0,* *Unique_client_id: ,1* *Balance: 100* *}*	
	Read *{* *global_transaction_id :0 ,* *transaction_id : 0*

	Unique_client_id: 2, *Balance: 100* *}*
Write *{* *global_transaction_id 0: ,* *transaction_id :0 ,* *Unique_client_id:1 ,* *Balance: 110* *}*	
	Read *{* *global_transaction_id :0 ,* *transaction_id : 0,* *Unique_client_id:2 ,* *Balance: 100* *}*
Commit *{* *global_transaction_id : 1,* *transaction_id :1 ,* *Unique_client_id:1 ,* *Balance: 110* *}*	
	Read *{* *global_transaction_id :1 ,* *transaction_id : 1,* *Unique_client_id: 2,* *Balance: 110* *}*

TESTCASE 3: Concurrent commit handling

TRANSACTION 1	TRANSACTION 2
Read *{* *global_transaction_id : 0* *transaction_id : 0,* *Unique_client_id: ,1* *Balance: 100* *}*	
	Read *{* *global_transaction_id : 0* *transaction_id : 0,* *Unique_client_id: ,2* *Balance: 100* *}*
Write *{* *global_transaction_id 0: ,* *transaction_id :0 ,* *Unique_client_id:1 ,* *Balance: 400* *}*	
	Write *{* *global_transaction_id 0: ,* *transaction_id :0 ,* *Unique_client_id:2 ,* *Balance: 500* *}*
Commit *{* *global_transaction_id : 1,* *transaction_id :1 ,* *Unique_client_id:1 ,* *Balance: 500* *}*	

	Commit *{* *global_transaction_id : ,* *transaction_id :,* *Unique_client_id:2 ,* *Balance:* *}* CONFLICT