Blockchain

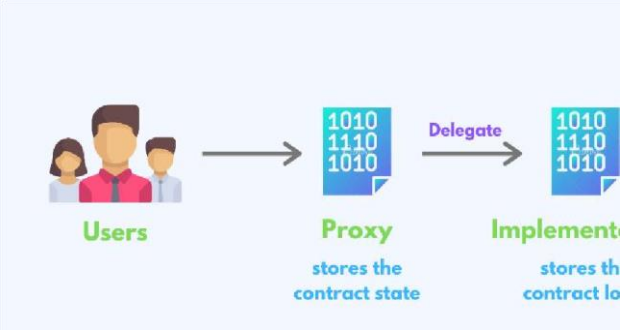
SMART CONTRACT STORAGE STATE MIGRATION COST ANALYSIS

Abstract:

In blockchain we deploy smart contracts to modify functionality on blockchain. Smart contracts can be deployed for various reasons, for example security, feasible to use, transactions, etc. We know that every operation on the blockchain takes some gas as a fee to operate on blockchain. Also, it costs gas if we use storage or update it. We know that one program can be useable till sometime after we need to update it, Similarly, smart contracts need to be updated. So, when we update smart contract, it will cost us heavily because we need to update its storage and storage can be large. So, we want to reduce the gas cost while updating the storage in contract migration. Traditionally people save data off chain and then using multiple methods update the new smart contract. While we want to let the storage be a separate part from smart contract, So, we don't need to update the storage every time. It is done by using the technique of
Proxy contract.

Introduction:

Smart contracts are there to improve the security of blockchain. Smart contract works as a program. So, after some time every program need to be supervised and if there is space of improvement new version of that program should be launched. Similarly, smart contracts also get their new versions because of many reasons like security breach, new policy, etc. So, we launched a new smart contract having all the things we need. Now the problem is how can we synchronize this smart contract with the previous smart contract. Which means we must read data from previous contract and write it for new contract (writing data on blockchain). We know that writing data on-chain cost gas. So, we must introduce new methods to reduce the gas cost of data migration. Proxy contracts allow you to inherit contracts or use a separate data contract so while updating smart contracts we don't need to update data, or it will be in the new smart contract as it is inherited from the previous smart contract.



Related Work:

There's a lot of work that is done for the decrease in the gas cost of the data migration. Using googles API to fetch data from blockchains and then upload that using contract which works as a middleman or use the functions to upload that data to blockchain.

Truffle also provide us the utility to migrate our contracts. Using truffle, we can optimize our migration which costs less gas and is better and easier as truffle will do all our work.

```
$ truffle migrate
```

After the migration we need to deploy our contract which also truffle do we just need to provide things.

```
// Stage deploying A before B
deployer.deploy(A);
deployer.deploy(B);
```

After deploying we get the address of the account using functions of truffle.

```
module.exports = function(deployer, network, accounts) {
  // Use the accounts within your migrations.
}
```

In proxy methods:

- Diamond pattern: EIP-2532
- Transparent proxy pattern
- Universal upgradeable proxy standard (UUPS): EIP-1822

1. These three techniques are used very much. Diamond pattern is rarely used because it's hard to implement as we use mapping (hashing) to find out the logical contract, this technique uses multiple logical contracts.

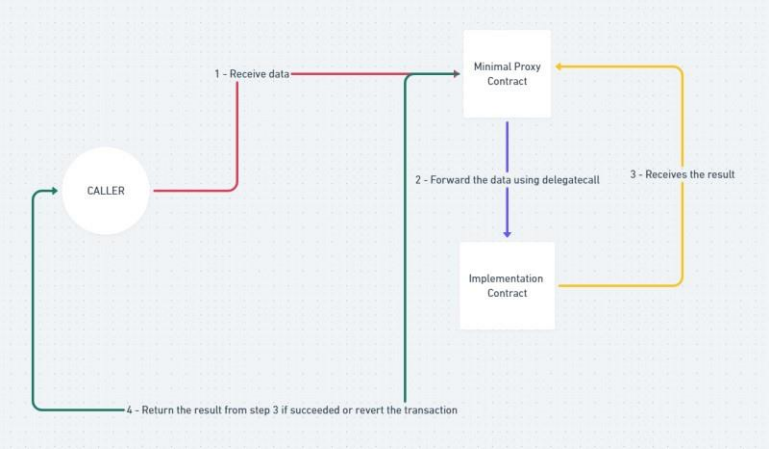
2. Transparent proxy pattern is mostly used because it's easy to implement and uses one or two logical contract (smart contracts).

3. The UUPS is used only when needed. In regular development transparent proxy pattern is used and works fine but UUPS will only be used if a very secure thing is needed to be implemented. Here the upgrade is done through the logical contract but not though the proxy contract as used in transparent proxy pattern.

Methods:

There are multiple methods in data migration few are describe below:

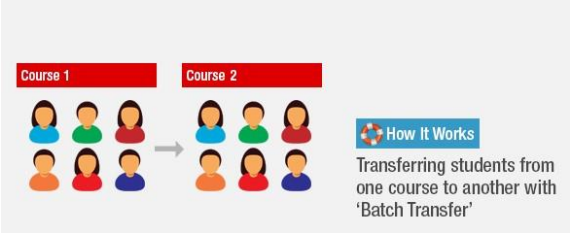
1. We can do data migration by using contract to contract migration. Means your previous or new contract has methods (functions) to access data from previous contract by request and store it in new contract or old contract get data and send it to new contract.



2. Secondly, we can also do it by using a middle contract as it will get all data from previous and then from that we can send it to new contract only if our

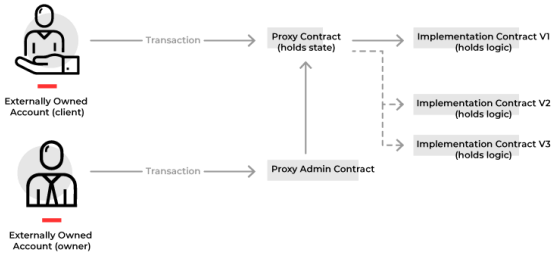
previous contracts have no means of sending data.

3. Thirdly, we can use **batch transfer function,** here we transfer multiple accounts amount in a single action.



Just like in this image, batch transfer function will transfer the data of multiple address at once instead of copying one address and its data then paste it in new smart contract.

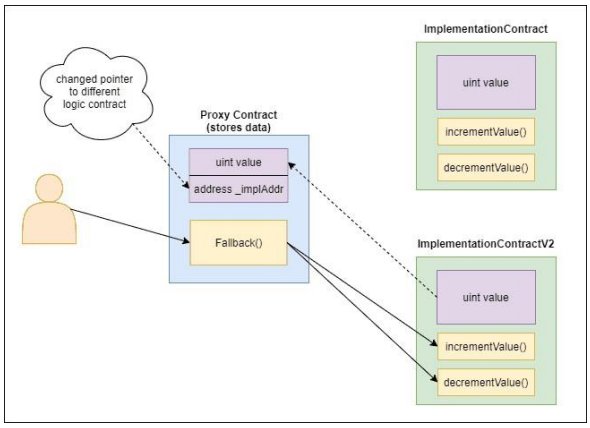
4. We can also use an upgradable smart contract with a common store. Here if user makes many smart contracts all smart contracts will have same storage so there will be no need of storage migration in case of same blockchain smart contract.



The most cost efficient is upgradable smart contract where the storage is fixed or it will be inherited from contract to contract, and smart contracts as different version of programs can be launched without even storage migration. As it will not add the cost of the migration of data to the blockchain for new smart contracts. It is achieved through **PROXY contract** in blockchain.

Proxy pattern	Pros	Cons
Transparent proxy pattern	Comparatively easy and simpler to implement; widely used	Requires more gas for deployment, comparatively
Diamond proxy pattern	Helps to battle the 24KB size limit via modularity; incremental upgradeability	More complex to implement and maintain; uses new terminologies that can be harder for newcomers to understand; as of this writing, not supported by tools like Etherscan
UUPS proxy pattern	Gas efficient; Flexibility to remove upgradeability	Not as commonly used as it is fairly new; extra care is required for the upgrade logic (access control) as it resides in the implementation contract

Let's suppose we deploy a smart contract without proxy and security buy is found, now we must write new smart contract and replace that with new one. Again, doing the data migration too. This all will take time. So, users' interest in this contract will not be like before. But we can control these things if we use proxy as admin can instantly freeze that activity. So, hackers or anyone cannot take advantage of that. And as the new contract is ready admin will reroute the proxy from old to new one.
In proxy contract, it is designed in such a way that it will contact another contract to do the tasks user ask. The contract which proxy contract will contact to do work is the latest version of smart contract which admin deployed on blockchain. With the change of target address in the proxy contract we can change the smart contract which we want to use. So, there will be no need for migration cost.



We need to write a proxy contract which will communicate with the deployed smart contract we will be using. After writing the proxy smart contract we can move towards our smart contract and build it. Now both the contracts are developed and ready to deploy. Deploy proxy contract and smart contract. We can deploy smart contracts in upcoming time with no need to transfer data.

Experimental Design:

First make a contract named Storage. It will hold all the storage items we need for our smart contracts.

```
pragma solidity ^0.4.21;

contract Storage {
    uint public val;
}
```

After that we need to make our proxy contract. In Proxy contract, we can make two functions, one for future upgradeable contract which access is only for admin (owner). The other function will be an anonymous function so that every call will go to that function no matter what is called. That function will fetch (load) the function we need from the smart contract and execute it.

Anonymous function is a callback function so it will come back to check if the execution was reverted or successful and result of that is returned to the user.

```
pragma solidity ^0.4.21;

import './Ownable.sol';
import './Storage.sol';

contract Registry is Storage, Ownable {

    address public logic_contract;

    function setLogicContract(address _c) public onlyOwner returns
(bool success){
        logic_contract = _c;
        return true;
    }

    function () payable public {
        address target = logic_contract;
        assembly {
            let ptr := mload(0x40)
            calldatacopy(ptr, 0, calldatasize)
            let result := delegatecall(gas, target, ptr,
calldatasize, 0, 0)
            let size := returndatasize
            returndatacopy(ptr, 0, size)
            switch result
            case 0 { revert(ptr, size) }
            case 1 { return(ptr, size) }
        }
    }
}
```

The final step is of the smart contract we need.

```
pragma solidity ^0.4.21;

import './Storage.sol';

contract LogicOne is Storage {
    function setVal(uint _val) public returns (bool success) {
        val = 2 * _val;
        return true;
    }
}
```

This design is for the data that is being separated from proxy and smart contract. The other design is using the a base contract that will be used as inherited while proxy and smart contract will be inherited from that contract. We need to build an abstract account like this.

```
pragma solidity 0.8.1;
//inheritance level proxy contract
abstract contract Upgradeable {
    mapping(bytes4 => uint32) _sizes;
    address _dest;

    function initialize() virtual public ;

    //IT IS USED AS A REPLACING FUNCTION WHERE WE CHANGE THE ADDRESS AND IT WILL CHANGE THE SMART CONTRACT
    function replace(address target) public {
        _dest = target;
        target.delegatecall(abi.encodeWithSelector(bytes4(keccak256("initialize()"))));
    }
}
```

After making this account we need to build the proxy contract that will reroute through the request to the smart contract and its specific function.

```
contract Dispatcher is Upgradeable {
//this will treat as a proxy contract

    //this will set's the target (the smart contract we will use)
    constructor(address target) {
        replace(target);
    }

    function initialize() override public{
        // Should only be called by on target contracts, not on the dispatcher
        assert(false);
    }

    //reroutes all the requests to the smart contract and give reaction as reverted or completed
    fallback() external {
        bytes4 sig;
        assembly { sig := calldataload(0) }
        uint len = _sizes[sig];
        address target = _dest;

        assembly {
            // return _dest.delegatecall(msg.data)
            calldatacopy(0x0, 0x0, calldatasize())
            let result := delegatecall(sub(gas(), 10000), target, 0x0, calldatasize(), 0, len)
            return(0, len) //we throw away any return data
        }
    }
}
```

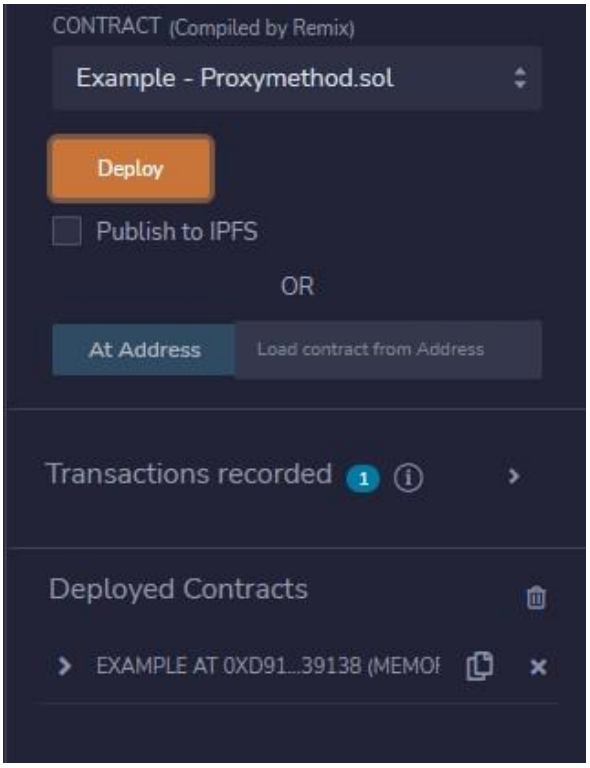
Now the dispatcher is a proxy contract that will change the smart contract address by using proxy contract's function. Now we have proxy contract all we need is the smart contract as shown below.

```
//this will treat as a smart contract
contract Example is Upgradeable {
    uint _value;

    function initialize() override public {
        _sizes[bytes4(keccak256("getUint()"))] = 32;
    }

    //SETTING THE UNIT FOR DATA
    function getUint() public view returns (uint) {
        return _value*6;
    }

    function setUint(uint value) public {
        _value = value;
    }
}
```

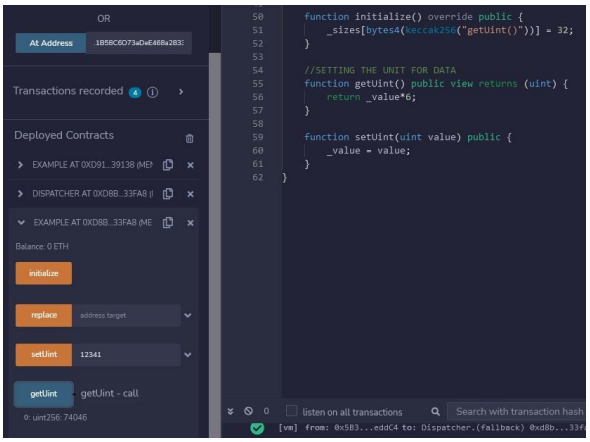
This is all we need for an inherited contract.

Experimental Result:

We now have deployed all the things we need for the formation of the inheritance storage proxy contract.



Now the base smart contract is developed. We will develop dispatcher and give that dispatcher the smart contract address.



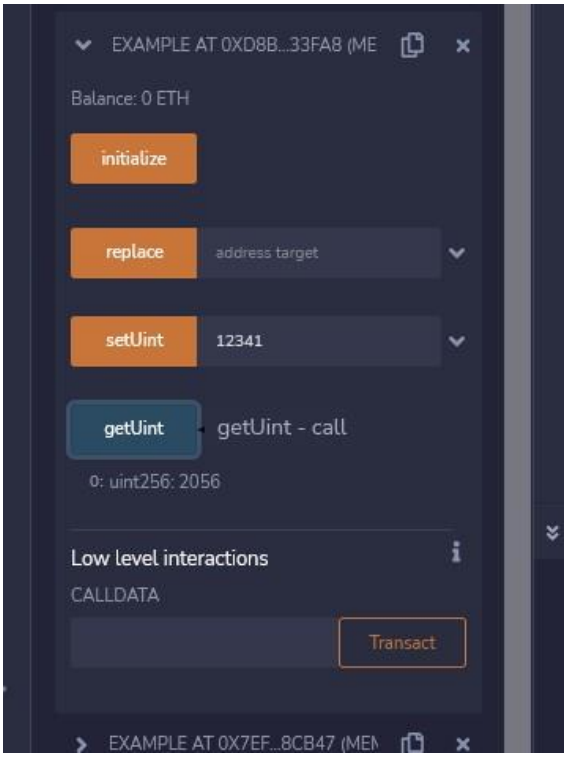
We can see that we deployed smart contract and give dispatcher the address of new smart contract and stored data in it which is shown above.

Now we will deploy new smart

contract what will divides value and change the address of target to that smart contract through dispatcher.

We can see that a new contract named "Example" is deployed at the end. We got that address placed that in dispatcher and replace the targeted smart contract, which is a successful operation.

Now we will check that if we request this will show us the multiplied value or the original value which we gave as **12341.**



So, it divided the value and gave **2056**.

- 1st smart contract o Value given is 12341 o Output was 12341*6 => 74,046
- 2nd smart contract o Value given is 12341 o Output was 12341/6 => 2056

So, we can say that after giving the address of 2nd contract to dispatcher it reroutes our request of **get Unit** to 2nd contract so we got 2056 answer instead of 444,276 which will be given if dispatcher was failed to change address of smart contract.

Conclusion:

We can conclude from the above experiment that proxy contract was a successful deploy and we can use that in real life after some addition of security checks as it will lead to secure and trustworthy contract. For the experiment on the external storage contract of proxy contract we need to develop environment in truffle.js. The given code will be enough for experiment in truffle.js. After a successful experiment in truffle.js we can add security check after that the contract will be ready to deploy.

Bibliography

[[Online]. Available: https://medium.com/@blockchain101/the-basics-of-upgradable-proxy1 contracts-in-ethereum-479b5d3363d6.
]
[[Online]. Available: https://medium.com/coinmonks/ethereum-smart-contract1 migration2 13f6f12539bd..
]
[[Online]. Available: https://mixbytes.io/blog/storage-upgradable-ethereum-smart2 3 contracts..
]
[[Online]. Available: https://docs.soliditylang.org/en/v0.5.4/introduction-to-smart3 4 contracts.html?highlight=delegatecall#delegatecall-callcode-and-libraries..
]
[[Online]. Available: 5 https://mirror.xyz/0x235ec6764650A2fa198AE6533cE323090Dcb3045/kxLfa4UsbLBG3] YYQLGvwmGpUmqrqBnH7IzO1nLJMPZ4..
[[Online]. Available: https://blog.chain.link/smart-contract-call-another-smart5 contract/. . 6
]
[[Online]. Available: https://ethereum.stackexchange.com/questions/114809/what6 exactly7 is-a-proxy-contract-and-why-is-there-a-security-vulnerability-invol. .
]
[[Online]. Available: https://research.csiro.au/blockchainpatterns/general7 8 patterns/migration-patterns/. .
]
[[Online]. Available: https://blog.trailofbits.com/2018/10/29/how-contract-migration8 9 works/. .
]