Cloud computing interaction with blockchain systems

# Introduction

In this paper, we will present a review of how cloud computing can be interacted with on the blockchain. More specifically we explored the area of Product as a service. This brought several limitations such as security etc into play. We think this is a valuable field of research since it could offer more security and cheaper ways of storing data. Some other interesting parts include decentralized access to data and some real word use case research. With this paper we contribute to the field by researching hybrid systems and developing a prototype for a cloud to blockchain application.

# Literature Review

## Technology Review

Bla bla bla

## Similar Projects

Filecoin is a decentralized storage network designed to store information. How it works is a user can rent out space on their computer for other users. This effectively transforms the users device into a cloud storage center. It is a cloud alternative to centralized solutions like AWS. The big issue with this type of solution is that after a user stops mining/storing data the data is lost. To solve that issue there is a degree of redundancy with multiple copies, but this just causes additional overhead.

Arweave is another decentralized storage network designed to store information.

Stori

Amazon Web Services (AWS) Blockchain Solutions

## Their limitations

# System Architecture

## High level overview

So how cloud computing interacts can be bit complicated, but has many applications. In this report I decided to focus how decentralized blockchain could interact with centralized cloud data centers. This process can be complicated since blockchains don’t have a direct way to interact with cloud data centers.

## Components

Below I explain how each component interacts with the overall system:

* Cloud Server: This is where the data can be stored or retrieved from. Examples of these include AWS or Azure
* Blockchain: Here is the system where the underlying Ethereum is.
* Smart Contracts: They are hosted on the blockchain, these are programs where the users can interact with the blockchain or different data.
* Web3 programs: Here the user can use as a point to interact with blockchain from.
* UI: Here is the actual interface the user interacts with
* Lamba function: This listens out for the changes on the blockchain.

## Architecture diagram

# Experiment setup and performance evaluation

## Environmental setup

How this project was setup was through several steps:

1. Created a database on the aws servers where the data be stored
2. Created a table on the said database.
3. Created a smart contract using solidity that deals with creating vaults.
4. Created an project key using infuria so I would be able to interact with the sepolia testnet.
5. Deployed smart contract onto the testnet.
6. Created an event listener using javascript and node.js that listens out for any events emitted by the smart contracts address.
7. Tested it out by creating a vault and seeing if the table was updated.

## Use cases

The primary use case for this specific experiment would be users who want an easy way to view and store data produced on the blockchain. This system shows how a proposed hybrid blockchain to cloud data storage system could work. When a vault is created the data is stored both on the blockchain and on a AWS database. This has benefits such as users having easy access to view data, as well as data stored on servers being easy to audit since the blockchain can’t be altered. This system could be expanded into areas that need to be easy to access and couldn’t be altered such as secure document storage or encrypted medical data.

## Evaluation

We managed to achieve several functionality goals:

* The smart contract was successfully deployed onto the blockchain.
* The event listener successfully detects addVault() events.
* The table on the AWS database was successfully updated after an added vault.
* The system doesn’t miss any events when it is actively listening.
* The system takes under 6 seconds to go from emitted event to updated table.

## Screenshots

# Limitations and Challenges

## Technical Limitations

## Cost Constraints

Implementing an additional system that stores data centrally could prove more expensive since one data is placed on the blockchain it is effectively free to host.

## Security and Privacy Concerns

The largest issue with the system is not having any encryption technology for any data being placed in it. An alternative experiment for this would be a user created a key and encrypting all data before placing it on the aws and blockchain, since blockchain data is publicly visible.

# Conclusion

# References

const { ethers } = require("ethers");

const AWS = require("aws-sdk");

// AWS Regon setup

AWS.config.update({ region: "eu-north-1" });

// DynamoDB client

const dynamoDb = new AWS.DynamoDB.DocumentClient();

// Ethereum provider (Sepolia via Infura)

const provider = new ethers.JsonRpcProvider("XXXXXXXX");

// Contract address and ABI

const contractAddress = "0x9d58134Dd3fba0B3dB67264E73195186E28BafA7";

const abi = [

    "event VaultAdded(uint256 vaultId, address owner)"

];

// This section listens for sepolia vaultAdded events

const contract = new ethers.Contract(contractAddress, abi, provider);

console.log("Searching for events");

contract.on("VaultAdded", async (vaultId, owner) => {

    console.log(`VaultAdded event: Vault ID ${vaultId.toString()}, Owner ${owner}`);

    const params = {

        TableName: "UserVaults",

        Item: {

            vaultId: vaultId.toString(),

            ownerAddress: owner

        }

    };

    // Store the event in DynamoDB or error handling

    try {

        await dynamoDb.put(params).promise();

        console.log("Successfully stored event in DynamoDB:", params.Item);

    } catch (err) {

        console.error("Error with storing data", err);

    }

});