Smart Contract Monitoring using Cloud Driven Technology

Dawid Pionk   
*South East Technological University*Carlow, Republic Of Ireland  
Dawidpionk01@gmail.com

*Abstract*—This paper explores the possible use of an integrated blockchain and cloud computing system. It proposed a way to use both systems in a complementary manner. The blockchain side of the system is used to carry out the transactions, provide security, and data protection. In contrast, the cloud computing side is responsible for easier data access and monitoring of smart contract execution. This way, you gain additional protection and easier access to your data. The gap between the 2 technologies is bridged using a lambda function that listens to specific changes on the blockchain and then updates the table accordingly. The game state is then updated based on the entries in the table. This makes it so that your system backup data is not centralized, allowing for increased security against cyberattacks. It also protects against fraudulent activity since, unlike traditional databases that could be hacked, the blockchain is immutable. This approach enhances security, transparency, and data integrity, as well as opens new venues for monetization through blockchain fees. The primary goal of this system is to show how you can improve data access and security using a hybrid system.

Keywords—Blockchain, Cloud Computing, Smart Contracts Monitoring, Game Transactions

# Introduction

Blockchain technology has proven itself to be a very secure way of storing data; it allows for a secure, transparent, and tamper-resistant way of handling data. Cloud computing tends to be an easy, quick, and cost-efficient way of managing data. This paper explores how a hybrid system that takes advantage of the strengths of both of these systems could be used for an online game.

A player in this game would purchase an item through the interaction with a smart contract on the Ethereum blockchain. The fees for the purchase of a newly created item or transactions between players would be applied. The game state is updated using a cloud table, which is updated after interactions on the blockchain.

In this paper, I am going to cover the system I mentioned above in more detail, the research I needed to take to understand blockchain and cloud computing technologies. I will show code for an experiment I ran to examine the validity of the proposed system. Also, I will discuss limitations and potential improvements to the proposed system.

# Literature Review

In this section, I give an overview of the research I did for this project, what implementation of integrated blockchain and cloud services I found. I also discuss other people’s research and how it aided me throughout this project. In each paragraph, I discuss a unique paper, survey, article, etc. that aided me in this paper.

An interesting case study for a cloud system that integrates cloud computing is the IBM Food Trust. Under this system, blockchain was used to enhance transparency and efficiency. It allowed for a secure way of tracking transactions and food products to avoid waste, fraud and to track the origin of food products that spread diseases. This system also allows for the creation of a standard ledger that could be used by all members of a particular supply chain. How it works is by only permitting trusted participants to enter data, like farmers or shopkeepers. Then those shopkeepers, instead of entering their data into a paper or Excel ledger, would enter it into the food trusts distributed ledger on the blockchain, allowing everyone easy access to view. [1]

In a paper about Blockchain-Based Cloud Computing, Bharathi Murthy et al. analyzed how integrating cloud computing with blockchain technologies can be used to mitigate certain key limitations in the field of cloud computing by surveying previous papers that have been written on this subject. It looks at how things like privacy, data security, and management could be improved through the blockchain. It explains each of the technologies in great detail and proposes a hybrid system using both technologies. This paper proved useful towards my proposed system cause it showed how I can combine the 2 systems. [2]

In a survey done by Jinglin et al. called “Integrated Blockchain and Cloud Computing Systems: A Systematic Survey, Solutions, and Challenges,” they explore the combination of blockchain and cloud computing systems. It is focused on how a hybrid system works, the different combinations, and improvements to security. It also explores other subjects such as consensus protocols, recent industry trends, and different models. It also explores different gaps in research and some challenges, such as performance issues, energy costs, and compliance with different regulations. This survey helped me explore different models of blockchain and cloud computing that I could implement into my research, as well as some interesting implications of it. [3]

An interesting article by the Swiss cyber institute talks about how blockchain technologies can enhance security in cloud environments. This article talks about how blockchains security systems such as immutable blocks and consensus protocols could be used to provide additional security. It also talks about how its decentralized nature allows for greater resistance against failure or attacks. What I gained from this article was the benefits of security blockchain systems provide [4].

I also looked into other potential applications of blockchains to give me an idea of how to best apply my project. I looked into an article to better understand the benefits of an integration of the technologies could provide for scalability and security. [5]

How  I plan to imply the research I done into the project I’m creating is by looking through this and choosing what particular aspects from each paper I want to apply. How I am going to integrate the traditional cloud system and see how I can apply it to my project.

# System Architecture

In this section I give an overview of the technologies I interacted with for this project. I give a high level overview of how the project was set up. I explain each of the individual components that were used to implement the project and its technologies, I also include a diagram to aid in the explanation of this system.

## Technologies

### Blockchain

A blockchain is a type of distributed digital ledger. It is comprised of “blocks,” which store information such as transactions, people involved, rewards for creating blocks, and other information for security or building programs. Its is called a blockchain since all those blocks point towards the previous one forming a chain. The key concept of blockchains is that they are decentralized, unlike traditional cloud systems, they are not stored in a specific location instead, they could be hosted on thousands of machines across the world connected through the internet.

The blockchain technology I will be using for this project will be Ethereum.

### Smart Contracts

Smart Contracts are a concept in distributed ledger technology where two parties can agree on a certain action that will be executed. It is implemented as a piece of code that self-executes once a certain criterion is met. This is how you create functions and implement applications on the blockchain. To implement this concept, I will write code on the Ethereum virtual machine using Solidity.

### Cloud Computing

Cloud computing is the process of delivering certain useful services over the “cloud”, which in this case, the cloud refers to the internet. There are 3 main categories for cloud services. Infrastructure as a service(IaaS), which means it provides resources/infrastructure over the internet. Platform as a service(PaaS) provides vital development services, like development environments over the cloud, and software as a service(SaaS), which means they provide user applications over the internet, like Office 365 or games. For this project, I’m going to focus on the concept of IaaS.

IaaS essentially allows you to outsource resources, such as RAM, Processing power or secondary storage. This allows you to safe your time and resources on setting up your equipment and in essence just allows you to rent out others people hardware and receive the output or input through the internet. For this purpose, I will be using AWS servers.

## Overview of the system

The system I propose works by having players buy in-game items through the interaction with a smart contract and then a cloud system that monitors changes on the blockchain. How this works is, for example, a player buys a sword in an online game, they would purchase it using Ethereum using a smart contract, then the data from that transaction would be extracted onto a table stored on the cloud. This is especially useful in case of data being lost on a centralised data centre due to several reasons, such as natural disasters or a hacker stealing user data. The advantage of this system is that no matter what happens to the database stored by the company, the data will be safe, also since this system allows purchases of items, the creation of blocks could be funded by players spending money.  Also, this could potentially give another way to bring in revenue since you could tax transactions between these items.

The purpose of this system is to investigate how you can use the cloud to aid in the monitoring of events that take place on the blockchain. This could be useful since the blockchain itself is more difficult to interact with the data than a simple database.

How this system works is by creating a smart contract using Solidity, that would be deployed onto the Ethereum testing blockchain called Seploia. Solidity is a high-level language used to interact with Ethereum code.

This smart contract can track what players own which items, which allows a secure and reliable way of storing information about the game state. This smart contract was deployed using Infura, which provides you with a node that you can use to connect directly to the Ethereum blockchain. If a user makes a transaction using this smart contract, a listener function written in JavaScript would pick up on it and send this update to the table stored on AWS. Then the game code could pick up on changes from this table and update the game state.

## Components

* Cloud Server: This is where the data can be stored or retrieved from. Examples of these include AWS or Azure
* Ethereum Blockchain: Used to store the transaction data and currency.
* Smart Contracts: The code for the transaction
* Lamba function: This listens out for the changes on the blockchain.
* Game: Here the user initiates the transactions.

## Architecture diagram

A – User purchases an item using Ethereum blockchain.

B – The lambda listener function catches that a new event fired

C – The lambda function updates the AWS database table to reflect the new purchase.

D – The game checks and sees a new entry meaning the game state should be updated.

E – The player can now use their new item

## Advantages of this system

Blockchains themselves offer enormous security benefits but have many drawbacks, such as performance issues. This system bridges that gap. You can easily query, search, and display what items belong to whom, which are listed for sale, etc., using the cloud database, while still having the security provided by the blockchain. This ensures that each item can’t be duplicated, your system will be more fraud-proof as well, as if your database gets wiped due to an error, you can easily work your way through the blockchain and rebuild your system.

# Experiment setup and performance evaluation

In this section, I will walk you through how I set up a partial implementation of the proposed system. I will go through how I set up the experiment step by step, and then I will evaluate the results, whether it was successful, what I accomplished, etc.

So for this experiment I created a simple program that checks whether it would be possible to create a smart contract tracking game items and then having them updated on an AWS server. I didn’t focus on the AWS to game aspect of it since it would be much simpler to implement.

Also in this experiment I didn’t place any owner address since I assumed the players account would be linked to their eth address in the game.

## Environmental setup

How this project was setup was through several steps:

1. Created a smart contract where a developer can create Items, players can then purchase each Item from the developer or from each other. They could also set their own prices so the players can make profit, also the developers would take a portion of these earnings as a fee. Code Listing A shows this function.
2. I tested out the smart contract using foundry to see if it’s functioning properly.
3. I created a project-id using infura so I could connect to an Sepolia node.
4. Deployed the contract using my MetaMask account, infura and foundry. The deployed smart contract can be view in Figure 2.
5. Created a table on DynamoDB on AWS services called itemEvents that tracks each item on the smart contract. This table can be viewed on Figure 3.
6. Created a lambda function in JavaScript to listen out for events omitted by my smart contract and update the cloud table. The function was deployed using ether library. Code Listing B shows this function.
7. The system was tested to see if it works properly. .

## Evaluation

I managed to achieve several functionality goals:

* The smart contract was successfully deployed onto the blockchain.
* The event listener successfully detects itemCreated() events.
* The table was successfully updated after an item was created.
* 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.
* Overall, the experiment proof to be a success

## Metrics

Below are the prices for each of the functions on the blockchain. The gas price at the time of the tests on 16/05/2025 is 3.015 gwei on average. (The gas fee is basically the payment to execute something on the eth blockchain). The price of 1 eth at the time of this project was €2,318.03. The price of the transaction was gotten by gas fee \* gas price \* cost of 1 eth in euro.

* Cost for the testBuyItem() function: gas 211670, €1.4803.
* Cost of BuyItemFromPlayer(): gas 268888, €1.879
* Cost of testCreateItem(): gas 161250, €1.1272.

This show that there would be roughly a surcharge between 1 to 2 euros per transaction. Which makes it not too expensive.

# Discussion

## Limitations

### Cost Constraints

Implementing an additional system on the blockchain for security could add additional costs related to the users spending extra money on gas fees instead of on the game store.

### Privacy

This system does come at the cost of the users’ transactions being visible. So other people could view how much they are buying, what they’re buying and when. To some users this could prove like a too big of an invasion of privacy stopping them from interacting with the system.

### Price volatiliy

Due to a lack in a centralized entity controlling the price of coins, the value of your items could vary drastically day to day therefore your income would vary a lot over time even if your sales remain the same.

## Possible Additions to the system

### Cloud Gaming

As the internet’s speeds continue to improve, it has become increasingly popular to host games on the cloud. You could apply this system to pay for resources from the cloud on the blockchain instead of a subscription model using local currency like is common place now. This would involve a similar technology like in the proposed system.

### NFTs

A user could get NFTs as well with each sold item or for each participated event or for getting achievements. This NFTs could be traded as well and the platform could take fees for each transaction, expanding the proposed marketplace.

# Conclusion

In this I presented a model for gaming system interacting and taking advantage of both the blockchain and cloud services. It solves issues of security and fault resistance in products that could be wiped out by a faulty server or stolen if not secure. The experiment performed showed that the proposed system is feasible. This research also opens the door to other potential areas that this system could be applied to such as aiding supply chain systems and other areas. Overall the findings have the potential to aid future cloud services to be more secure.

##### Acknowledgment

I would like to express my thanks to my supervisor Shei Lei for their guidance through this project as well as the staff at UNUM for their classes that enhanced my understanding of cloud computing,

##### References

1. High, M. (2020). Supply chain insight: Inside IBM’s Food Trust Blockchain system. [online] supplychaindigital.com. Available at: https://supplychaindigital.com/technology/supply-chain-insight-inside-ibms-food-trust-blockchain-system
2. Bharathi Murthy, Ch.V.N.U., Lawanya Shri, M., Kadry, S. and Lim, S. (2020). BLOCKCHAIN BASED CLOUD COMPUTING: ARCHITECTURE AND RESEARCH CHALLENGES. IEEE Access, pp.1–1. doi:https://doi.org/10.1109/access.2020.3036812.
3. Jinglin Zou, Debiao He, Sherali Zeadally, Neeraj Kumar, Huaqun Wang, and Kkwang Raymond Choo. 2021. Integrated Blockchain and Cloud Computing Systems: A Systematic Survey, Solutions, and Challenges. ACM Comput. Surv. 54, 8, Article 160 (November 2022), 36 pages. https://doi.org/10.1145/3456628
4. Swiss Cyber Institute. (2021). What is Blockchain Security in Cloud Computing? [online] Available at: https://swisscyberinstitute.com/blog/blockchain-security-in-cloud-computing/.
5. ‌WebClues Infotech (2025). Blockchain Meets Cloud Computing: A Powerful Integration for Scalability & Security ☁️🔗. [online] Medium. Available at: https://medium.com/coinmonks/blockchain-meets-cloud-computing-a-powerful-integration-for-scalability-security-%EF%B8%8F-331bb33cd88b [Accessed 15 May 2025].

##### aPPENDIX

*Code Listing A. GameState Smart Contract*

// SPDX-License-Identifier: UNLICENSED

pragma solidity ^0.8.13;

contract GameState {

    struct Item {

        uint256 id;

        string name;

        uint256 price;

        uint256 attack;

        uint256 defense;

        address owner;

        bool forSale;

    }

    event ItemBought(

        address player,

        uint256 quantity

    );

    event ItemCreated(

        uint256 indexed itemId,

        string name,

        uint256 price,

        uint256 attack,

        uint256 defense

    );

    Item[] public itemsList;

    modifier onlyDeveloper() {

        address Developer = 0x1234567890123456789012345678901234567890; // Replace with the actual developer address

        Developer = msg.sender; // This here just for the sake of testing

        require(msg.sender == Developer, "Not the contract owner");

        \_;

    }

    modifier onlyPlayer(uint256 itemId) {

        require(msg.sender == itemsList[itemId].owner, "Not the player");

        \_;

    }

    function createItem(

        string memory name,

        uint256 price,

        uint256 attack,

        uint256 defense

    ) public onlyDeveloper returns (uint256 itemId){

        itemId = itemsList.length;

        Item memory newItem = Item({

            id: itemId,

            name: name,

            price: price,

            attack: attack,

            defense: defense,

            owner: address(0),

            forSale: true

        });

        itemsList.push(newItem);

        emit ItemCreated(itemId, name, price, attack, defense);

    }

    function buyItem(uint256 itemId) public payable {

        require(itemId < itemsList.length, "Item does not exist");

        Item storage item = itemsList[itemId];

        require(msg.value >= item.price, "Not enough Ether sent");

        require(item.owner == address(0), "Item already owned");

        if (msg.value > item.price) {

            payable(msg.sender).transfer(msg.value - item.price);

        }

        // Below section I send the money to the developer in this case I set it as the sender

        // just for the sake of testing

        address developer = msg.sender;

        payable(developer).transfer(item.price);

        item.owner = msg.sender;

        item.forSale = false;

        emit ItemBought(msg.sender, itemId);

    }

    function buyItemFromPlayer(uint256 itemId) public payable {

        require(itemId < itemsList.length, "Item does not exist");

        Item storage item = itemsList[itemId];

        address seller = item.owner;

        require(seller != address(0), "Item not owned by anyone");

        require(item.forSale == true, "Item not for sale");

        require(seller != msg.sender, "Cannot buy your own item");

        require(msg.value >= item.price, "Not enough Ether sent");

        /\* Here the service fee would be deducted

        This section removed for the sake of the experiment

        (bool platformSuccess, ) = payable(developer).call{value: platformFee}("");

        equire(platformSuccess, "Payment to platform failed");

        \*/

        // Refund excess Ether

        if (msg.value > item.price) {

            payable(msg.sender).transfer(msg.value - item.price);

        }

        // Pay the seller

        payable(msg.sender).transfer(item.price);

        // Transfer ownership

        item.owner = msg.sender;

        item.forSale = false;

        emit ItemBought(msg.sender, itemId);

    }

    function getItem(uint256 itemId) public view returns (Item memory) {

        require(itemId < itemsList.length, "Item does not exist");

        return itemsList[itemId];

    }

    function setItemPrice(uint256 itemId, uint256 newPrice) public onlyPlayer(itemId) {

        require(itemId < itemsList.length, "Item does not exist");

        Item storage item = itemsList[itemId];

        item.price = newPrice;

    }

    function setItemForSale(uint256 itemId, bool forSale) public onlyPlayer(itemId) {

        require(itemId < itemsList.length, "Item does not exist");

        Item storage item = itemsList[itemId];

        item.forSale = forSale;

    }

    function getItems() public view returns (Item[] memory) {

        return itemsList;

    }

    function getItemCount() public view returns (uint256) {

        return itemsList.length;

    }

}

*Code Listing B. listener.js*

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

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

// AWS Region 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("https://sepolia.infura.io/v3/12be543cebb045c3b94f5820ef89dc7b");

// Contract address and ABI

const contractAddress = "0x80a712A480292f5EFd3aBb387B6fC56d9B8fdf29";

const abi = [

    "event ItemCreated(uint256 indexed itemId, string name, uint256 price, uint256 attack, uint256 defense)"

];

// Setup contract

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

console.log("Listening for ItemCreated events...");

// Event listener

contract.on("ItemCreated", async (itemId, name, price, attack, defense) => {

    console.log(`ItemCreated event: ID=${itemId}, Name=${name}, Price=${price}, Attack=${attack}, Defense=${defense}`);

    const params = {

        TableName: "itemEvents",

        Item: {

            itemId: itemId.toString(),

            name: name,

            price: price.toString(),

            attack: attack.toString(),

            defense: defense.toString(),

            timestamp: new Date().toISOString()

        }

    };

    try {

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

        console.log("Stored item event in DynamoDB.");

    } catch (err) {

        console.error("Error storing event in DynamoDB:", err);

    }

});