Smart Contract Data Monitoring and Visualization

Seng Kuang Yap[1], Zhongli Dong[1,2], Mark Toohey[1,4], Young Choon Lee[3], Albert Y. Zomaya[2]

***Abstract*—Blockchain technology has attracted significant industry, academic, and governmental attention since its emerged in 2008. Blockchain use cases are now being explored by traditional, transaction-oriented businesses in the finance, insurance, logistics and healthcare sectors to name a few. This has expanded further with the widespread use of Internet of Things (IoT) devices. Massive amounts of data are generated by IoT devices and are recorded in the blockchain. While blockchain provides many advantages, such as immutability and transparency, its serialized nature makes impossible to read in a single query. Multiple requests are required even for simple tasks, such as displaying an account's transaction history. This further leads to the difficulty in understanding the data in the blockchain. In this paper, we address the problem of smart contract visualization in a real-time manner. To this end, we design a visualization dashboard for smart contracts. A visual aid for massive amounts of data helps users understand the blockchain's overall activities, uncover operational risks and provide critical intelligence by visualising unusual activities and connections. Such insights may enable the user to investigate and predict any anomalies or reveal any network vulnerabilities. Cattle farm selected as a use case because the voluminous data can be acquired from IoT sensors on the farm cattle. Our dashboard has been proven to help visualize the life cycle of animals, the distribution of activities and time factor analysis. This visualization can give a user a better perspective of the token functions and results as well as animal management issues.**

***Index Terms*—Blockchain, Ethereum, Smart Contract, NFT, Internet of Things, Supply Chain**

I. Introduction

Blockchain is a distributed ledger system that replicates across multiple computers that are assembled in a peer-to-peer network that follows a cryptographic protocol. The members of the network are called nodes and each node has its own copy of the blockchain that digitally records and stores the data in unchangeable packages called blocks. [1].

The next significant development was the Ethereum blockchain, which combined a distributed ledger and a Turing-complete programming language used to write smart contracts.

Smart contracts were initially defined as automated legal contracts that are, at least partly, capable of being expressed and implemented in software [2]. Ethereum implemented a global digital computer to execute peer-to-peer smart contracts that can't be shut down [1]. The Ethereum blockchain runs these smart contracts using a virtual machine called the Ethereum Virtual Machine (EVM). There are many usage of smart contracts, such as digital identity, trade finance, insurance, data recording and supply chain management.

In the meantime, Internet of Things (IoT) devices have been increasingly adopted in data gathering for blockchain. In particular, blockchain and related technologies enable IoT devices to record immutable append-only data and make that data publicly available for all of the network's participants. Massive amounts of data are generated by IoT devices and are recorded in the blockchain by logging smart contract events. Due to the serialized nature of blockchain, it is impossible to read the event logs in a single query. This makes users struggle to easily and timely understand the data in blockchain.

There have been several studies on blockchain visualization, e.g., [3], [4], [5]. The systematic review presented by Natkamon et al. in [3] shows that there have been various attempts to visualise blockchain data [6]. However, the current focus is on the visualization of token transactions. We are not aware of any works that have addressed the visualization of Ethereum smart contract data.

In this paper, we address the problem of smart contract data monitoring and visualization in a real-time manner using the cattle supply chain data that record on Ethereum blockchain. We have chosen a cattle farm as a use case because the voluminous data can be acquired from IoT sensors on the farm cattle (Figure 1). Data visualization involve the generating of graphical representation of data that aims to identify the patterns, trends, correlations and outliers of the data. Blockchain provide several benefits for data visualization, thank to the high quality data, traceability, built-in anonymity as well as large data volumes. However, challenges are faced on data aggregations and real time visualization from the blockchain due to the nature of data storage on blockchain.

To this end, we design a smart contract monitoring and visualization dashboard that can visualize data in a real-time manner. In particular, our dashboard solution incorporates the monitoring of events and the recording of new data as soon as the relevant new block was mined. Our smart contracts do not record data in variables or in a data structure. Data is instead stored in logs, that emit an event when the function is called. This reduces the amount of gas required and is a cost-effective means of storing data on an Ethereum blockchain.

Our event monitoring and dashboard system shown in Figure 2 uses the extract, transfer and load (ETL) method and also retrieves events emitted from the blockchain and insert them into databases. The ETL process is streamlined to reduce the time required to filter data from the historical

[1] Seng Kuang Yap, Zhongli Dong and Mark Toohey are with Aglive Research, PO Box 196 Geelong VIC 3220, Australia.

[2] Zhongli Dong and Albert Y. Zomaya is with Center for Distributed and High Performance Computing, School of Computer Science, The University of Sydney, NSW 2006, Australia.

[3] Young Choon Lee is with School of Computing, Macquarie University, Sydney NSW 2109, Australia.

[4] Mark Toohey is a law lecturer in the Master's in Blockchain and Digital Currency course at the University of Nicosia, Cyprus.

978-8-3503-1019-1/23/$31.00 ©2023 IEEE

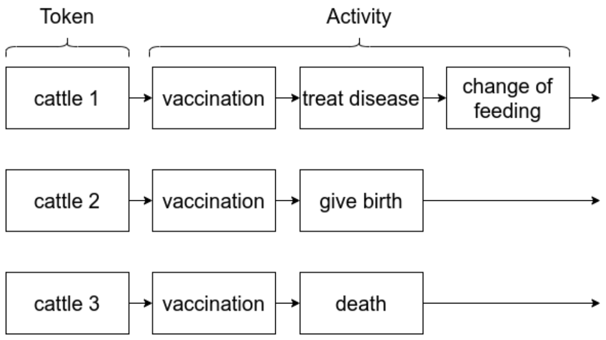


Fig. 1: Examples of the smart contract data logging.

blocks. Storing the event data in databases helps reduce the time taken to constantly query the blockchain in order to update the dashboard. Pseudo-data was generated to simulate the flow of IoT data from the cattle farm.

Our dashboard has been proven to help visualize the life cycle of tokens (and the linked animals), the distribution of activities and time factor analysis. This visualization can give a user a better perspective of the token functions and results as well as animal management issues.

The remainder of this paper is structured as follows. Section II gives background. Section III presents the proposed system architecture and gives insights into the various tools and technologies used to build the proposed system. Section IV presents the dashboard results and discusses workflow methodologies and issues regarding building the dashboard. Section V reviews and discusses related work. Section VI concludes the paper.

II. Background

Blockchains are an immutable append-only database, where each block consists of the header, the transactions, and the previous block header. A new block header is formed by hashing information in both the current block and the previous block header. Due to the serialized nature of blockchain, it is impossible to read the event logs from the blockchain in a single query. Usually, multiple requests are required even for simple tasks, such as displaying an account's transaction history. In contrast, a relational database stores information in a structure that rarely changes. A cattle management database can more easily record farm data e.g birth date; vaccinations; death; diseases; veterinary treatments; pregnancy; and calving etc. We can use the following query to obtain the cattle vaccination history for the period between two different dates:

```
SELECT * FROM TABLE_NAME WHERE
    DATE_OF_VACCINATION
BETWEEN 'STARTING_DATE_TIME' AND '
    ENDING_DATE_TIME';
```

However, such a simple query is more difficult on a blockchain because a blockchain sequentially stores event log. As shown below, a query on cattle vaccinations during a certain period would have to search the blockchain logs block by block. The build date recorded in the blockchain would also have to be compared with the STARTING_DATE_TIME and ENDING_DATE_TIME to determine if cattle vaccinations were recorded within that date range.

```
myContract.events.MyEvent({filter: {
    DATE_OF_BUILD},fromBlock:0},
function(error, event){
   console.log(event);
   // Compare DATE_OF_VACCINATION with
       STARTING_DATE_TIME and
       ENDING_DATE_TIME
   // Record cattle detail fulfil the above
        requirement.
       })
```

Clearly, considerable time is required to search the blockchain's event logs as we need to iterate through each of the blocks in order to filter the query data and aggregrate data for visualization.

Smart contracts and IoT devices can combine to monitor and verify a product's entire life cycle and its traceability. This can be used to validate a food product's source, its quality, and even how it was handled. Stated another way, the technology can be used to confirm the identity, location, and condition of a particular item or shipment. It can also help verify whether a certain product was ethically produced and check the item's sustainability credentials. Safeguarding and improving product integrity will help improve supply chain transparency by aiding the monitoring of food safety and quality. Of course, IoT data stored in a smart contract is also available for public verification.

There are two ways to record data on to the smart contracts. One is in a variable state where values are permanently stored and can be used to change smart contract functions. Another way is to log it as smart contract events as an unchangeable data structure that cannot be accessed directly by smart contracts. The former storage cost 20 000 [gas] per 32 bits and the latter costs 8 [gas] per byte [7]. In this study, data will be stored as smart contract events to reduce gas costs. However, to visualized data stored in smart contract events pose operational difficulties because of the blockchain's sequential nature [7].

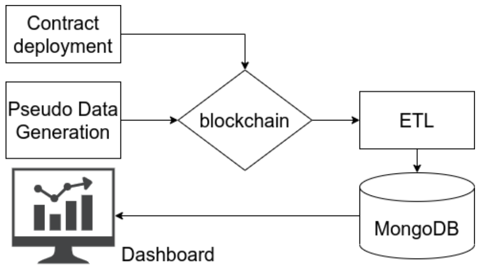


Fig. 2: The Overall Design of Event Monitoring and Dashboard System.

III. Architecture of Monitoring and Visualization System

Figure 3 illustrates the system architecture of a monitoring and visualization system for Smart Contracts. It consist of a front-end user interface that collects raw data from IoT devices, and interacts with the back-end services. Data cleaning aggregates and formats data and fixes or removes any incorrect, corrupt, duplicate or incomplete data. A clean daily data file will then be temporarily stored in the server awaiting next step.

Parallel data flows from Daily Transaction to Blockchain and Data Extraction and storage, have the advantage of enabling data verification. This ensures all blockchain transactions are also recorded in databases. In the proposed architecture, it is assumed that data is cleaned and arranged as per daily logs. Cleaning Data from IoT devices is outside the scope of this paper as we are primarily focused on Daily Transactions.

Our cattle use case adopts a modified version of an ERC721 smart contract to record on-farm IoT data and visualize the smart contract events. It does not include activities related to either the transfer of cattle along the supply chain or to the tracking of beef products.

A. Daily Logs

We considered it important to protect the cattle producer's privacy. So, we have only used simulated data. Algorithms were used to generate simulated data for token creation and the recording of events that were common farm activities. We also added in information insights gained from our Aglive platform and various industry standards. Developers can generate their own data to simulate other use cases.

Data acquisition methods and raw data cleaning methods will not be addressed in this paper. Instead, it is assumed that the data is cleaned, verified, and filtered before being written into the blockchain.

The data generation application will generate daily data and store the data as a daily log. Random simulated data will induct new cattle onto the farm and also record some departures.

Each time an animal was inducted onto the platform a new token was created. When cattle ceased to exist, the relevant token was burned.

Eighteen activities were included in the algorithm including record of birth; arrival; feeding; movements; and weight etc. Those transactions were fed into the smart contract.

Other token activities will be discussed in detail in below.

B. Daily Transaction

The Daily Transaction task is to write daily logs into smart contract. Daily Transaction consist of two parts: 1. transaction signing; and 2. key management.

Transaction Signing: checks daily data entries and evoke the corresponding smart contract functions. The daily transaction application ensures the creation of new tokens or the addition of activities to a new token will not be transacted in the same block as a transaction will fail if the token does not already exist. The application will only add the new transaction after for several new blocks to be created.

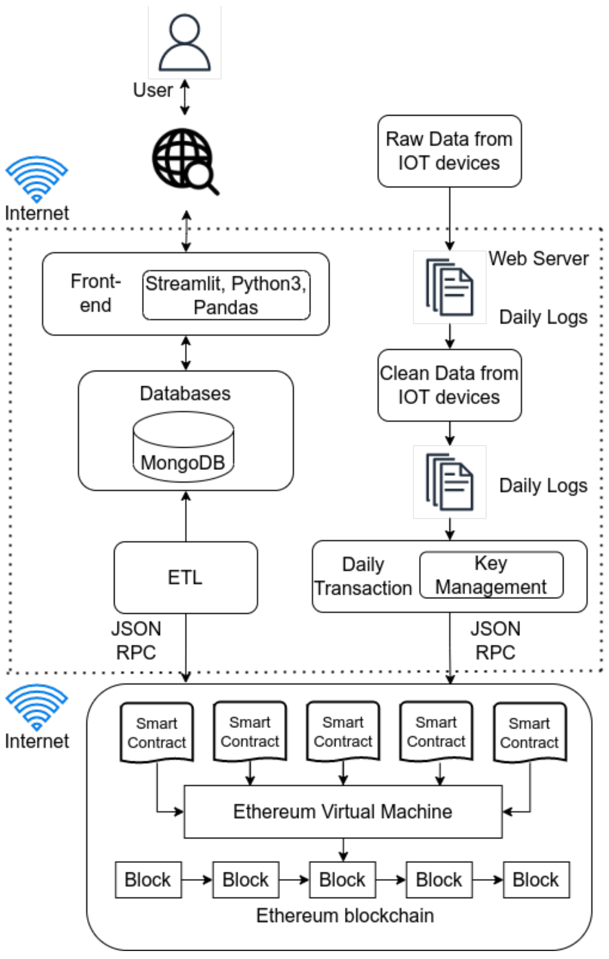


Fig. 3: System Architecture.

Key management is the application for managing a nonce in a transaction signing account that is used for all relevant cattle farm transactions. Key management tracks the current nonce value of the signing account. In this project, only one account was used for transaction signing. However, key management could be easily be expanded to enable signing by multiple accounts.

C. Data Verification

Data verification ensures the transaction is successfully recorded on the private Ethereum blockchain. Transaction hashes created when the transaction is signed, can be used to verify that data by retrieving transaction receipts from the private Ethereum blockchain.

The two steps are: 1. the transaction hashes are checked to ensure they match the number of transaction receipts. 2. the status of the transaction receipts are checked to ensure their 'True' status.

D. Ethereum Blockchain

Standard ERC721 token contracts were used in the proposed system. The only modification was that instead of using mapping to store activities related to each ERC721 token, we emitted an event when any new activity was added to the relevant token.

As mentioned, the induction of new cattle resulted in a token being created and the death of any cattle resulted in the burning of a token. Corresponding events were also emitted. This method helped reduced our gas usage for data storage.

Three custom functions were added into the standard ERC721 smart contract: 1. Create a token; 2. Add a token activity; and 3. Deactivate or 'burn' a token.

Token Creation

A standard mint function from an Openzeppelin ERC721 contract was used when any new cattle were inducted. The token count was increased and an event was emitted.

Token Deactivation

A standard burn function from an Openzeppelin ERC721 contract [8] was used as this method stops the token from appending any new activity and emits a token deactivation event.

1) Private Ethereum Blockchain

Vitalik Buterin's paper [9] introduced Ethereum and addressed several limitations of Bitcoin's scripting language. The main contribution of Ethereum over Bitcoin is that Ethereum's scripting language is Turing-complete. A private Ethereum [10] blockchain and proof of authority consensus system was achieved using Clique (Geth implementation). This method provided availability and partition tolerance and helped guarantee consistency.

Proof of authority consensus algorithms use minimal computation because there is no mining required. [11]. The algorithm depends on one or more trusted authorities ('validators') who collect transactions from the transaction pool, and then bundle them into blocks and add them into the chain.

In some instances only a sole validator may be assigned. If multiple validators are assigned, one of those validators will be selected as the leader and their block proposal will have the highest priority compared to the proposals by the other validators. The other validators will check the validity of the received block and after that process, each valid block will be added into the blockchain. The leader can be changed every round as the cycle continues [12]. Proof of authority consensus is therefore considered suitable for a private Ethereum blockchain.

2) Smart Contract

Smart contracts are created using Etherum's Turing-complete scripting language. They are an abstract layer that enable anyone to create their own rules for ownership, transaction format, and state transition functions. [13].

Ganache is a local Ethereum blockchain that is used to test and deploy smart contracts. It does not require connection with a real blockchain or use of an Ethereum client like Geth or Parity. The Ganache-GUI which provides GUI app and Ganache-CLIis a command line tools.

Ganache-Cli was used due to its ability to time travel. It hasa special utility that allows time dependent or stateless tests on a local Ethereum Blockchain.

A smart contract is also a self-executable block of a code that is deployed on the blockchain. The code specifies the rules that govern the interaction between the particular blockchain's participating parties. It is guaranteed to run in the same way on all peer nodes.

While several languages are used for writing smart contracts, the most common are: Solidity, Vyper, Yul and Yul+ [14]. All of these languages are compiled into bytecode before being deployed on the EVM.

3) ERC721

Ethereum tokens are digital assets written by the smart contract. Developers can build tokens on top of the Ethereum blockchain. This allows developers to use the existing Ethereum blockchain as the infrastructure that creates new tokens instead of building a new blockchain for new tokens. ERC721 is an open standard that enables a smart contract to create unique (non-fungible) tokens. ERC721 provides a minimum interface (abstract class in other programming languages) for managing, owning, and trading unique tokens. The ERC721 interface does not restrict the addition of extra functions or token metadata.

4) Smart Contract's Event

An event is a means of understanding changes in a smart contract's state. Ethereum allows the contract to log a change of state to the blockchain in a specific format that allows the EVM to easily retrieve and filter these events. An event can carry data about any state changes.

E. Extract, Transform, and Load

The Extract, Transform, and Load application will monitor the latest block, and filter blocks for events related to the smart contract's function call. These filtered events were stored on a MongoDB database. Finally, the web application read data from MongoDB to update the table in the front-end and push the update to the dashboard so it could be shown to end users. The database contained three tables for token creation, token burning and for recording any activities added to a specific token.

Web3py is used to build Extract, Transform, and Load applications that extracted data from the private Ethereum blockchain and stored it into MongoDB.

Two filters were created to monitor these respective events in the latest block. Both filters sleep every two seconds in between checking the block. "add_token_activity_event" events and "transfer" events are the two events emitted by the smart contract. These two events were monitored asynchronously.

"Transfer" events are part of the events that are inherited from standard ERC721 contract, while the "add_token_activity_event" event is user defined in the cattle contract. "Transfer" events were also emitted by either the create token or the deactivate token functions, depending on the ”from” and ”to” addresses.

The create token function assigns a zero account to ”from” and the user address to ”to” in the ”transfer” event.

The deactivate token function assigns the user address to ”to” and zero account is assigned to ”from” in the ”transfer” function.

F. Databases

A database is used in the system to mitigate the need to repeatedly filter the data in all the blocks.

Time stamps also play an important role in enabling data visualization. By using databases, data can be selected based on duration required by the user. With the ever-increasing size of the chain's data blocks, scanning al, the blocks for data would take too long. Because the data is extracted from the database, this modified system architecture is scaleable. even if many users are simultaneously accessing the dashboard web page.

MongoDB is document-based database. It is classified as NoSQL as it does not have a predefined schema. It works flexibly with different document types, such as JSON, BSON, XML and BLOBs. However, there are a few drawbacks as it does not support JOIN querying and it consumes memory because key names have to be stored for each document. A MongoDB database was used because it is the database used by the Aglive platform.

For the purpose of this paper, three collections are created in MongoDB. A collection is the equivalent of an Relational Database Management System(RDBMS) table, as shown in Table Ia, Table Ib and Table Ic.

token_id	timestamp	block_number
int	int	int

(a) create_token_event

token_id	timestamp	block_number
int	int	int

(b) deactive_token_event

token_id	activity	timestamp	block_number
int	string	int	int

(c) add_token_activity_event

TABLE I: Schema for three collections and data type

Table Ia and Table Ib are used to record the creation and deactivation of tokens, respectively.

All cattle were each given a unique token ID. A ”timestamp” was recorded when a particular block was sealed or authorized by a sealer. It represents the time when the token was created or deactivated in the blockchain. It is important to note that a timestamp is not the actual time when the event happened. The ”block_number” is the block where this token was created or deactivated.

Table Ic creates a new row when a new activity is attached to any active token. Activities are not created as an array as this enables us to query a specific activity. Activities are recorded along with the related ”timestamp” and ”block_number” or when an activity event emitted by the smart contract.

The ”block_number” is important for data verification as activities are not stored in the smart contract's variables. Gas use was reduced by storing activity data when an event was emitted instead of storing a variable in the smart contract. The ”block_number” is also used to verify whether the data stored in the database also exists on the private Ethereum Blockchain without having to filter the whole blockchain.

G. Front-end

Front-end was built using Python 3 libraries as they contain a number of useful data manipulation and chart plotting packages and that led to easier and faster development.

1) Python 3

Python 3 is a popular high-level programming language that can handle various programming tasks such as numerical computation, web development, database programming, network programming, parallel processing, etc. It is an interpreted language. Portions of the code can be tested on the command line before it is incorporated into the program. There is no need for compiling or linking.

Python 3 was also used for data extraction, data manipulation, data analysis, and data plotting. Multiple standard and external Python3 packages such as Web3py, Plotly and Streamlit were used. These packages will be discussed below.

2) web3js and web3py

Web3js is a collection of libraries that allow users to interact with the Ethereum blockchain. Three Ethereum node deployment methods are available for Web3js to interact with Ethereum blockchain: HTTP; web socket; and inter-process communication. Web3py, a Python implementation of Web3js, was used on this project.

3) plotly

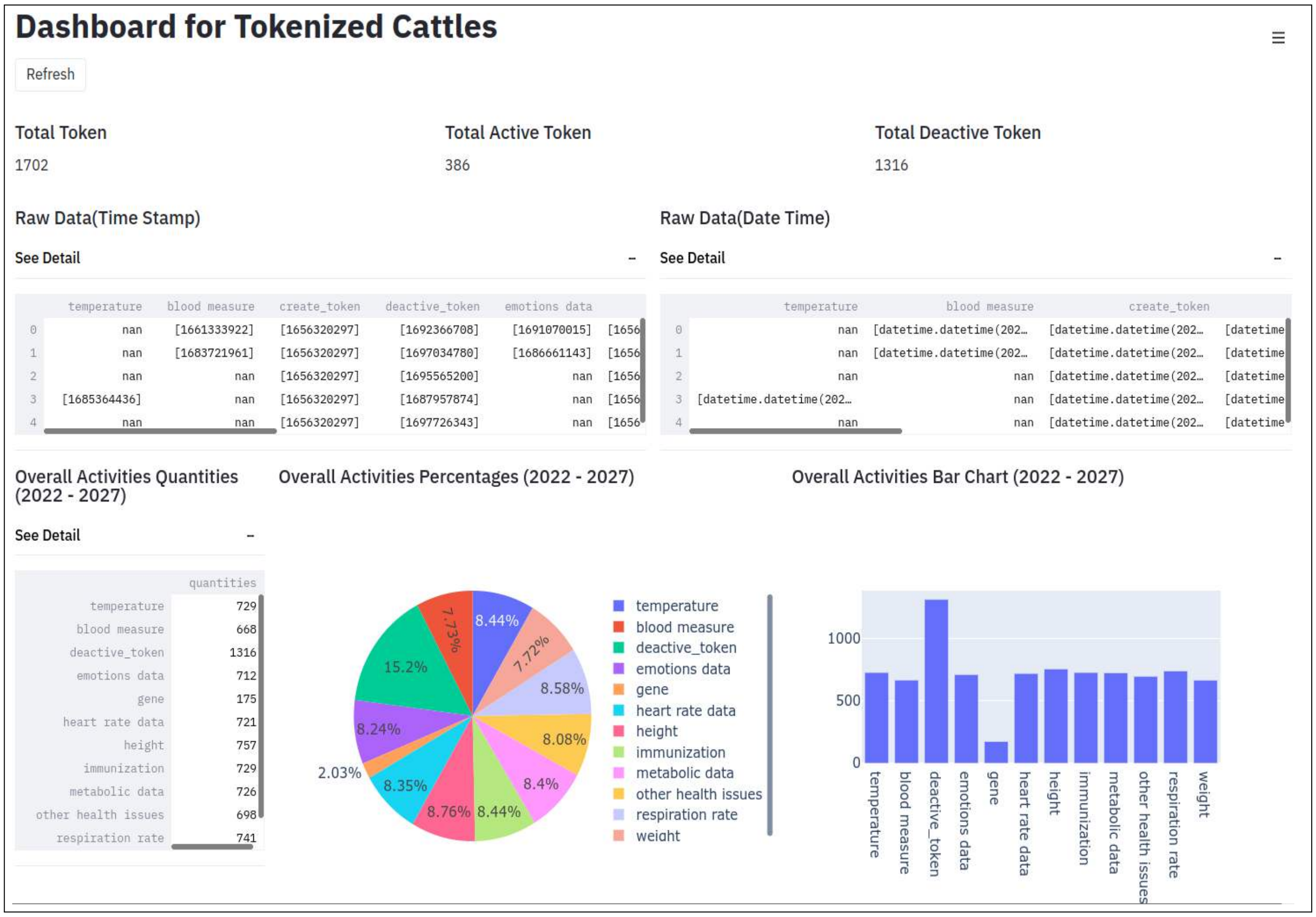
The Plotly library is an interactive, open-source plotting library that supports over 40 unique chart types covering a wide range of statistical, financial, geographic, scientific, and three-dimensional use-cases.

Plotly is built on top of the plotly.js JavaScript library as this enables Python users to easily create interactive web-based visualization. These visualization can be displayed in Jupyter Notebooks, as a stand alone HTML files, or they can be served as part of web applications by using the Dash framework. The Plotly library is sometimes referred to as ”plotly.py” to differentiate it from the JavaScript library.

4) streamlit

Many open source web frameworks are available, such as Flask, Django, Tornado, etc. These web frameworks provide flexibility when building the web user interface design. However, incorporating visualization into these frameworks, requires integration of the web framework with the plotting framework.

Streamlit on the other hand provides a convenient way to create web applications when it is integrated with popular visualization frameworks. Streamlit is a fast development



Dashboard for Tokenized Cattles

Total Token 1702

Total Active Token 386

Total Deactive Token 1316

Raw Data(Time Stamp)

	temperature	blood measure	create_token	deactive_token	emotions data	
0	nan	[1661333922]	[1656320297]	[1692366708]	[1691070015]	[1656
1	nan	[1683721961]	[1656320297]	[1697034780]	[1686661143]	[1656
2	nan	nan	[1656320297]	[1695565200]	nan	[1656
3	[1685364436]	nan	[1656320297]	[1687957874]	nan	[1656
4	nan	nan	[1656320297]	[1697726343]	nan	[1656

Raw Data(Date Time)

	temperature	blood measure	create_token	
0	nan	[datetime.datetime(202…	[datetime.datetime(202…	[datetime
1	nan	[datetime.datetime(202…	[datetime.datetime(202…	[datetime
2	nan	nan	[datetime.datetime(202…	[datetime
3	[datetime.datetime(202…	nan	[datetime.datetime(202…	[datetime
4	nan	nan	[datetime.datetime(202…	[datetime

Overall Activities Quantities (2022 - 2027)

	quantities
temperature	729
blood measure	668
deactive_token	1316
emotions data	712
gene	175
heart rate data	721
height	757
immunization	729
metabolic data	726
other health issues	698
respiration rate	741

Overall Activities Percentages (2022 - 2027)

Overall Activities Bar Chart (2022 - 2027)

Fig. 4: Dashboard

tool for proof of concept that supports various data manipulation and visualization frameworks, such as pandas, matplotlib, seaborn, altair, plotly, Dash and bokeh. They enable presentable front-end dashboards to be created with only limited front-end development knowledge. The disadvantage is that Streamlit does not have the flexibility of a dynamic dashboard panel.

We used Streamlit for our dashboard due to its simplicity and short learning curve. We simplified the design by creating a refresh button on the dashboard to enable a user to query new data from database directly into the dashboard.

IV. Evaluation

In this section, we present various dashboard data to demonstrate the efficacy of our visualization dashboard.

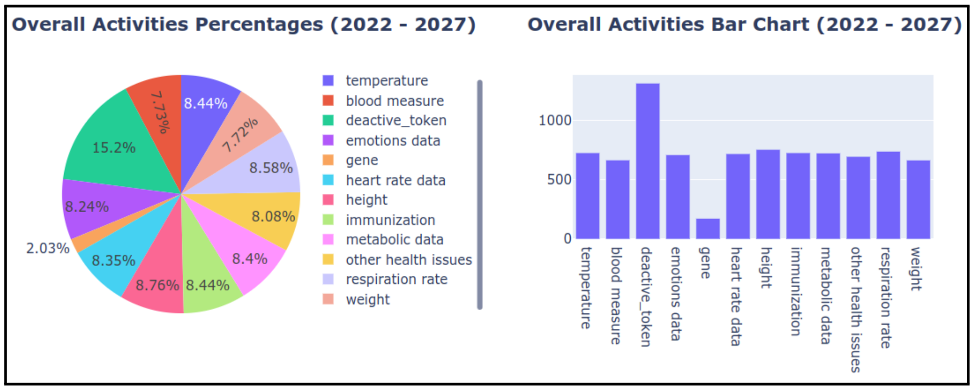
The use of ETL application and database data has eliminated the need to search and filter the blockchain - a process that would have required multiple passes just to perform a simple data query. This method has reduced dashboard query data required to plot charts and greatly improved the user experience, especially by reducing the dashboard loading times.

The architectural design responding to user requests was showcased using multiple charts and data tables. Some selected dashboard panels are discussed in this section (Figures 4 and 5). The dashboard is interactive with real-time data with the use of a refresh button that reduces CPU usage. However, an automatic refresh can be implemented.

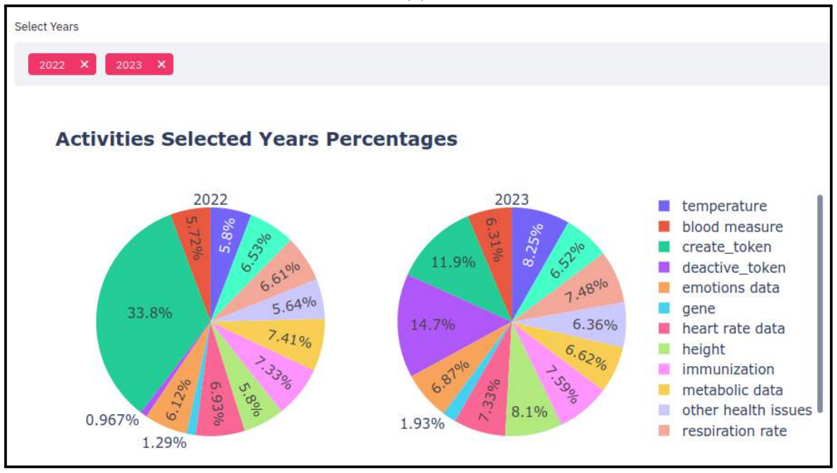
Every plot has a zoom feature to enable the user to easily increase the plot size. The modular form of Streamlit [15] allow users to organize the dashboard chart and table based on their preferences.

The dashboard 4 highlights three important token statistics. Each token represents and actual animal of the particular farm. Total Tokens is the total number of tokens minted in the smart contract. Total Active Token is the number of cattle currently on the farm. Total Deactived Tokens are the token that have been burned or tokens for animals that are no longer alive. Together these statistics give a snapshot of farm activity.

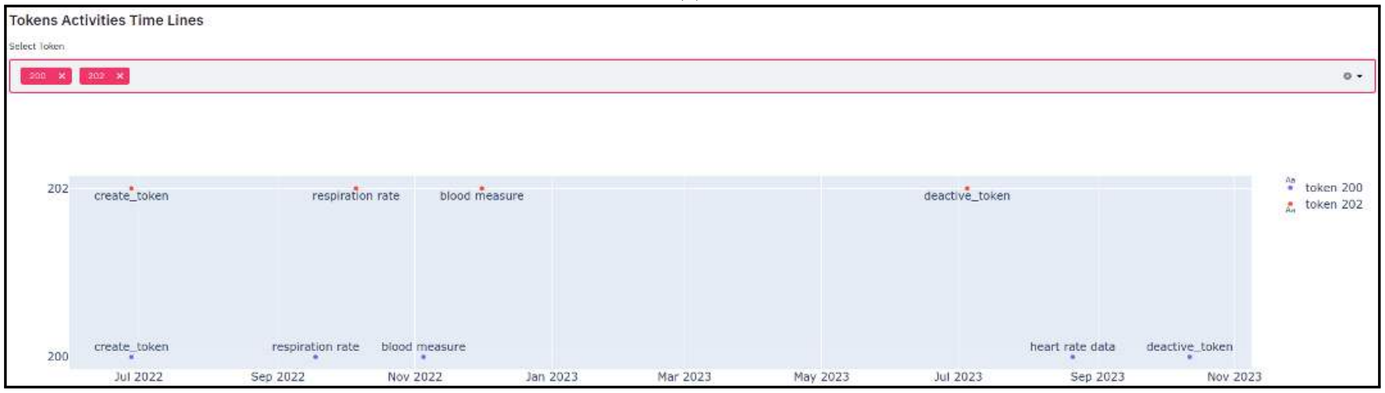
Other cattle events can be recorded and shown: e.g. genomic testing, health treatments, animal movements, custodial transfers of animals to sale-yards etc, as well as feeding event where animals are grain fed. It should be noted that we disabled ”feeding” and ”moving” activities in our test plots as



(a)



(b)



(c)

Fig. 5: Other Panels of the Dashboard:(a) Overall Activities Pie Plot and Overall Activities Bar Chart. (b) Pie Plot of Activities with Selected Years. (c) Tokens' Activities Timeline

they are high frequency activities. Once a token is deactivated, no further event data can be stored against that token.

When you are reviewing the charts, it may be helpful to review the raw data used to plot the chart. Our design include a raw data table with Unix time stamp format and date time format. Data fields with "NaN" are activities that did not happen to the cattle. You may notice that some cell entries list a timestamp, as such activities happened numerous times. Only the top five rows of the table are shown in order to save space.

The Overall Quantities (2021-2027) table in Figure 4 shows all activities over the years since the smart contracts were deployed. These figures can be combined with the farm's cost structure to determine the operating costs. Of course, it also highlights the most costly activities.

Figure 5a shows the percentages pie chart and quantity bar chart for various activities. These charts are plotted using data from Table Overall Activities Quantities (2021-2027).

Comparing activities between years is helping as an understanding trends may inform farm management decisions.

Figure 5b allows user to select and compare annual activity. Due to the fixed panel size, the chart size will be reduced if additional annual data is included.

Cattle life span and health data is crucial operational information. It could highlight animals that should be culled due to poor health, failure to conceive, or that have conformation problems.

Our project in Figure 5c also show a token's activity timeline. Users can select the token of interest and view the the recorded activities.

V. Related Work

Sundara et al. [4] examined eight bitcoin blockchain visualization attempts and gave a short description of the visual representations and implementations. Sundara et al. used a systematic mapping process to gather data through internet-based data collection. All of the blockchain visualization attempts we reviewed focused on monitoring real-time bitcoin transactions and the data was displayed using using either tree diagrams, hubs, or nodes. Python, WebGL, Javascript, and other libraries were used as visualization tools.

Natkamon et al. [6] systematically reviewed seventy six blockchain visualization attempts and classified them based on analysis tasks and visual representations. Furthermore, Natkamon et al. [6] extended their work to include other kind of blockchains (e.g. Ethereum).

Natkamon et al. [6] divided blockchain data visualizations into six different classifications: charts; time series; graphs; multi-dimension visualizations; maps; and casual. The majority of these visualizations involve: a transaction; flow of money; account balances; account addresses; node distribution; hash rates etc. [6].

Zhong et al. [5] realized that traditional blockchain visualization tools just show the information in tabular and textual forms. These forms are limited in their ability to help a novice user understand cryptocurrency transactions, mainly because they provide insufficient information. [5]. Zhong et al. introduced the multiple visualization schemes in their web page for visualization block transactions and their SilkViser paper. [5]. They separated the blockchain visualization into block, transaction, and address pages. Questionnaire results show that SilkViser helped novice users to understand important concepts in bitcoin transaction systems and provided advanced information for experienced users.

Nonetheless, none of the existing blockchain visualization research covered streamlined smart contract data manipulation and visualization.

VI. Conclusion

In this paper, we presented a method to visualize and analyze events emitted from smart contracts, along with proposed dashboard architecture and the related development process. Our technology stack and development methodology was also detailed. Cattle farming was selected as our use case due to the high volume of data acquired from IoT devices. We have also shown how cattle data can be managed on an Ethereum blockchain.

The dashboard we have designed can help visualize the life cycle of tokens (and the linked animals), the distribution of activities, and how time factor analysis can be conducted. This visualization can give a user a better perspective of the token functions and results as well as animal management issues.

References

[1] J. Kehrli, "Blockchain explained," *Netguardians [en lınia].[Data de consulta: 25 de juny de 2017]¡ https://www. netguardians. ch/news/2016/11/17/blockchain-explained-part-1*, 2016.

[2] W. Zou, D. Lo, P. S. Kochhar, X.-B. D. Le, X. Xia, Y. Feng, Z. Chen, and B. Xu, "Smart contract development: Challenges and opportunities," *IEEE Transactions on Software Engineering*, vol. 47, no. 10, pp. 2084–2106, 2019.

[3] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," tech. rep., Manubot, 2019.

[4] T. Sundara, I. Gaputra, and S. Aulia, "Study on blockchain visualization," *JOIV: International Journal on Informatics Visualization*, vol. 1, no. 3, pp. 76–82, 2017.

[5] Z. Zhong, S. Wei, Y. Xu, Y. Zhao, F. Zhou, F. Luo, and R. Shi, "Silkviser: A visual explorer of blockchain-based cryptocurrency transaction data," in *2020 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pp. 95–106, IEEE, 2020.

[6] N. Tovanich, N. Heulot, J.-D. Fekete, and P. Isenberg, "Visualization of blockchain data: A systematic review," *IEEE Transactions on Visualization and Computer Graphics*, 2019.

[7] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[8] openzeppelin, "openzeppelin-contracts." https://github.com/OpenZeppelin/openzeppelin-contracts, 2020.

[9] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," *white paper*, vol. 3, no. 37, 2014.

[10] V. Buterin *et al.*, "Ethereum white paper," *GitHub repository*, vol. 1, pp. 22–23, 2013.

[11] S. Kaur, S. Chaturvedi, A. Sharma, and J. Kar, "A research survey on applications of consensus protocols in blockchain," *Security and Communication Networks*, vol. 2021, 2021.

[12] P. K. Singh, R. Singh, S. K. Nandi, K. Z. Ghafoor, D. B. Rawat, and S. Nandi, "An efficient blockchain-based approach for cooperative decision making in swarm robotics," *Internet Technology Letters*, vol. 3, no. 1, p. e140, 2020.

[13] D. Vujičić, D. Jagodić, and S. Ranjić, "Blockchain technology, bitcoin, and ethereum: A brief overview," in *2018 17th international symposium infoteh-jahorina (infoteh)*, pp. 1–6, IEEE, 2018.

[14] Ethereum Foundation, "Solidity documentation release 0.8.5." https://buildmedia.readthedocs.org/media/pdf/solidity/develop/solidity.pdf, 2021.

[15] streamlit, "Streamlit." https://github.com/streamlit/streamlit, 2021.