DDS PROJECT REPORT

Authors (In alphabetical order):
Josef Emanuele Zerpa Ruiz, zerparuiz.1837394@studenti.uniroma1.it
Silvio d'Antonio, dantonio.2145048@studenti.uniroma1.it
Simona De Risi, derisi.2081525@studenti.uniroma1.it

Abstract

Distributed ledger technologies are modern and innovative solutions that are currently finding their way to popular awareness, thanks to relevant breakthroughs in the financial sector. They ensure transparency, reliability, and public access to data. Some applications are trying to bring these features to the agricultural sector. As to provide safer mechanisms of food supply chain management. Our work aims at exploring this topic, understanding the technologies, study present solutions and test both deployment and performances of a distributed ledger technology application.

Contents

Project Assignment

Distributed Ledger Technology is emerging as a solution to support traceability in the agricultural sector, namely as a technological tool to support the reliable storage of information about exchanges in the supply chain.

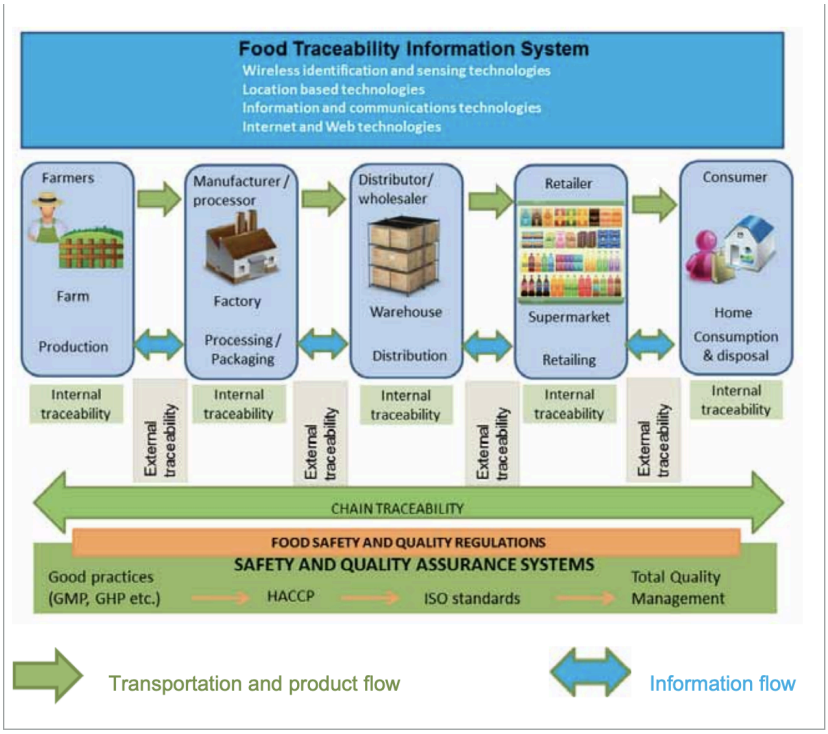
The aim of the project is to identify a solution already defined in the scientific literature (through Google Scholar or Scopus) for a specific traceability use case (e.g. milk, pasta, tomato sauce, etc.) based on a DLT, to test its deployment and to perform a dependability evaluation (both from a functional and performance point of view). If the identified solution seems too complex to be reimplemented/tested, a simplified version can be considered.

There are no constraints on the environment of analysis (local machine, docker, cloud free tiers, simulator).

The final report should include:

- Presentation of the traceability use case in the agricultural sector
- Presentation of the identified solution in the literature
- Description of the technologies used to develop the project
- Description of the evaluation design and results collected for the dependability evaluation
- Comments on the obtained results and highlights of potential limitations
- Link to the source code repository
- Installation instructions for configuring and running the software

Traceability use case in the agricultural sector



Source: Traceability in a food supply chain: Safety and quality perspectives Myo Min Aung, Yoon Seok Chang

In the agricultural sector, traceability has become an indispensable tool for enhancing food safety, ensuring quality control, and meeting regulatory and market demands. The traceability use case involves the systematic tracking of agricultural products throughout their entire lifecycle, from the initial stages of production to their final delivery to consumers. This process begins at the farm level, where inputs such as seeds, fertilizers, and pesticides are carefully recorded, and cultivation practices are documented. As the products move through the supply chain each step must be logged to create a comprehensive record of the product's journey.

This end-to-end traceability not only helps to enhance food safety and identify potential issues, but it is vital in a lot of situations:

- It helps to manage disease outbreaks that could impact entire product batches. For instance, if a particular issue is detected in a shipment, traceability allows for the precise identification of its source, enabling swift corrective actions and minimizing the impact on public health and market confidence.
- It is important to meet the standards of global markets, where consumers demand transparency and assurance about the origins and safety of their food.
- It offers benefits in terms of supply chain efficiency and sustainability. For example it allows for better inventory management and helps minimizing waste by ensuring that products reach the market in optimal conditions.

The solution in the literature

In the paragraph above it is stated that in the agricultural supply chain transparency, security and efficiency are requested, using Distributed Ledger Technology and Blockchain is a popular solution in the literature. Blockchain’s decentralized nature ensures that all transactions and data entries are immutable, meaning they cannot be altered or tampered with once recorded. This feature is particularly beneficial in agriculture, where trust and authenticity are critical. By leveraging blockchain, stakeholders across the supply chain—from farmers to distributors to retailers—can access a single, immutable source of truth that records every transaction and process step. This enhances transparency, allowing consumers to trace the origin and journey of their food products with confidence. Additionally, blockchain reduces the need for intermediaries by enabling direct, peer-to-peer transactions, which can streamline operations and reduce costs. Smart contracts, a feature of blockchain, can further automate and enforce agreements between parties, ensuring that terms are met without the need for third-party oversight. Moreover, the security provided by blockchain helps protect sensitive data from breaches and fraud, a growing concern in digital agriculture.

The source code of the solution analyzed and tested can be found at the following link: https://github.com/ac12644/Supply-Chain-Smart-Contract.

The solution makes use of Blockchain and Smart Contracts. The supply chain is simplified in this way:

There are 4 roles: the *farmer*, the *distributor*, the *retailer* and final *consumer*.
The core functionality is to be able to trace the steps of a given product, that will be called "Item", from the production by the farmer to the purchase of the final consumer.

Seven function can be identified:

1. Farmer creates a product and lists it to be purchased by Distributor
2. Farmer ships the product
3. Distributor receives the product, process it, package it and put it on sale
4. Retailer buys the product from Distributor
5. Distributor ships the product to Retailer
6. Retailer receives the product and put it on sale
7. Consumer purchase the product
8.

More in detail the struct Item is defined with various fields giving information about the stock unit, the product code (generated by the farmer), the product ID, the farmer that produced it, the price and the state.

The state of the product must pass the following 12 steps of the supply chain, and each function will emit a corresponding event:

1. *produceItemByFarmer*, that allows a farmer to produce an item will emit the event *ProduceByFarmer*
2. *sellItemByFarmer*, that allows the farmer to sell the product, will emit the event *ForSaleByFarmer*
3. *purchaseItemByDistributor*, that allows distributor to purchase product, will emit the event *PurchasedByDistributor*
4. *shippedItemByFarmer*, that allows farmer to ship product purchased by distributor, will emit the event *ShippedByFarmer*
5. *receivedItemByDistributor*, that allows distributor to receive product, will emit the event *ReceivedByDistributor*
6. *processedItemByDistributor*, that allows distributor to process product, will emit the event *ProcessedByDistributor*

7. *packageItemByDistributor*, that allows distributor to package product, will emit the event *PackageByDistributor*
8. *sellItemByDistributor*, that allows distributor to sell product, will emit the event *ForSaleByDistributor*
9. *purchaseItemByRetailer*, that allows retailer to purchase product, will emit the event *PurchasedByRetailer*
10. *shippedItemByDistributor*, that allows the distributor to ship the product purchased by the retailer, will emit the event *ShippedByDistributor*
11. *receivedItemByRetailer*, that allows retailer to receive the product, will emit the event *ReceivedByRetailer*
12. *sellItemByRetailer*, that allows the retailer to sell the product, will emit the event *ForSaleByRetailer*
13. *purchaseItemByConsumer*, that allows the final consumer to purchase the product, will emit the event *PurchasedByConsumer*

Technologies used in the project

Ethereum and Solidity

Ethereum is a particular implementation of the blockchain model. It is more modern than other implementations like Bitcoin, and thanks to its active community it undergoes continuous improvements, like the recent switch from Proof of Work schema to Proof of Stake less power hungry schema.

Throughput is higher on the Ethereum blockchain, since the computational complexity of mining a new block is set at an average of 1 every 12 seconds, compared to that of Bitcoin's 1 every 10 minutes approximately.

The Ethereum network also relies on the power of the Turing complete Ethereum Virtual Machine. It runs programs written in Solidity, among other options, handling the operations and transactions.

These are some of the reasons that contribute to the current high popularity of the Ethereum technology, the Solidity programming language, and the motives for our selection of these technologies in our project architecture.

We used Ethereum as implementation of a blockchain, hence a distributed ledger technology. The application described in the previous section is written in Solidity.

Kurtosis

Kurtosis is a handy and powerful architecture simulator and deployer. It wraps on top of the docker engine, starting docker machines, handling connection between nodes, and deploying a whole network infrastructure.

Kurtosis is specifically designed to handle blockchain networks. It manages both execution clients and consensus clients, allowing you to deploy your own local private network, test execution, and measure performances.

It elegantly provides you with a development environment made up of command line tools, inspection functions, management and configuration files.

We used Kurtosis to launch a local Ethereum network.

Hardhat

Hardhat is the developer view on the software aspect of an Ethereum network. The previous technology handles the “hardware” side of the network, the nodes. Hardhat is the module that interacts with the execution client, injects requests, transactions, and obtains information.

The Hardhat framework allows you to interact with your Smart contracts written in Solidity, perform operations using Javascript, define unit tests, and debug your network.

Once having built an Ethereum network, with Hardhat it is easy to connect to it, deploy your Smart contracts and test them.

The Hardhat framework comes with many plugins and libraries to simplify the testing work.

We used Hardat to interact with the local Ethereum network. With the framework, we were able to deploy contracts to the blockchain, send transactions, execute tests and measure performances.

Performance Evaluation

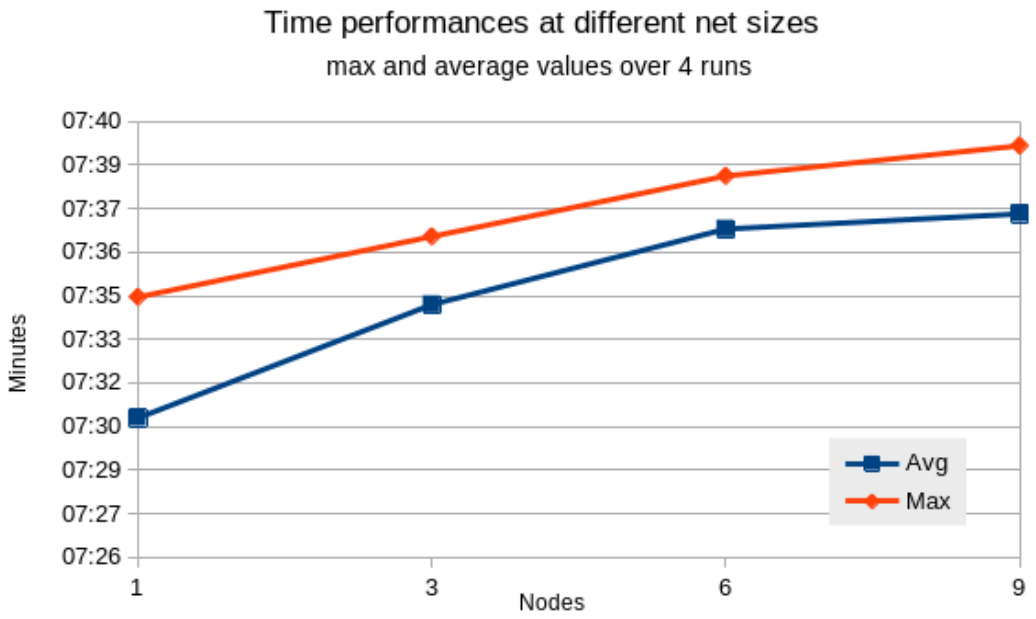
Performance tests with different network sizes

The purpose of these tests is to measure the time taken to fulfill a fixed number of requests issued by a node, and check how the execution time varies as the Ethereum net size increases.
The results of this test could give us some insights on the performance trend as the Ethereum net grows larger.

Methodology

We measured the time taken to run a batch of tests on the smart contracts at different network sizes, using the tools provided by the Hardhat development framework.
The execution of these tests first involves the deployment of the contracts used by the DAPP on a test net. Every function of these smart contracts is then tested by issuing a transaction and comparing the end result against the expected one.
Every test issues a total of 32 calls to smart contract functions.
As a test net, we used a local Ethereum network managed by Kurtosis. In this way we could easily change the number of nodes between executions.
For running this experiment we used different networks composed of 1, 3, 6, and 9 nodes, with one issuing node using the ‘lighthouse’ client. We are unfortunately limited to a maximum net size of 9 nodes due to hardware (RAM) limitations.
The tests were timed using the GNU tool ‘time’. We ran these tests 4 times for each network size then we extracted both the average and the maximum values.

Results



We can see that increasing the number of nodes from 1 to 9 increases the maximum time taken to fulfill the requests, even if marginally. We can see that there's an increase of about 4 seconds between the largest and the smallest network. Assuming a linear trend, this means that adding a node increases the maximum time of roughly 0.4 seconds, starting from the baseline of 1 node.
This increase is most likely due to the higher number of participants during the consensus phase.

Performance tests with different client nodes

We want to record the change in performance brought by a changing number of client nodes issuing transactions to the distributed network.

Methodology

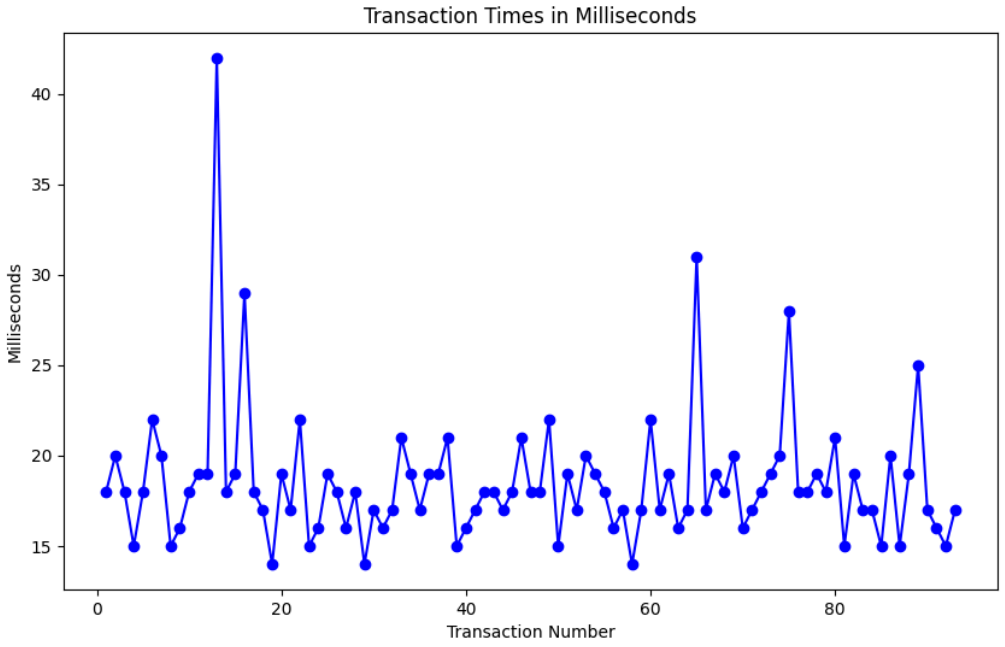
With Kurtosis we were able to fire up a fixed size blockchain network. Hardhat comes with a 'task' functionality. It allows you to write custom program executions. We created a new task on the hardhat project issuing function calls on a pre-deployed contract. To do that, we had to define the deploying recipe for the Solidity contracts. Through the 'ignition' module of the Hardhat framework, contract deployments can be automated. In our case we used ignition to deploy the "SupplyChain" contract to the local test net. We then took the address of the deployed address and passed it as argument to the custom defined task. Such task would then be able to call functions of that deployed contract, perform operations and receive responses.

By firing up different instances of a client server, executing the same task, we would then measure how the network responds to the growing number of clients issuing contract calls. Measuring the time delays between transaction requests.

Results

We were able to implement the above described methodology. Deploy contract on the blockchain, pass the deployment address to a custom task, make function contract calls from the custom task to a deployed contract. We faced a technological problem at the moment of executing the task on different clients. By trying to execute the tasks on different terminal windows of the same pc we encountered a kind of “Same origin” error on the requests. As if the Hardhat instance firing up the client environment was actually the same one, even on different terminal windows, different processes. We tried creating a new Hardhat project, connecting it to the same blockchain local network, and executing the same task. The same problem would arise.
More work should be done to address this issue and possibly find a working solution.

In the meantime we measured the timings of the different transaction calls executed by a single client. As a proof of concept.



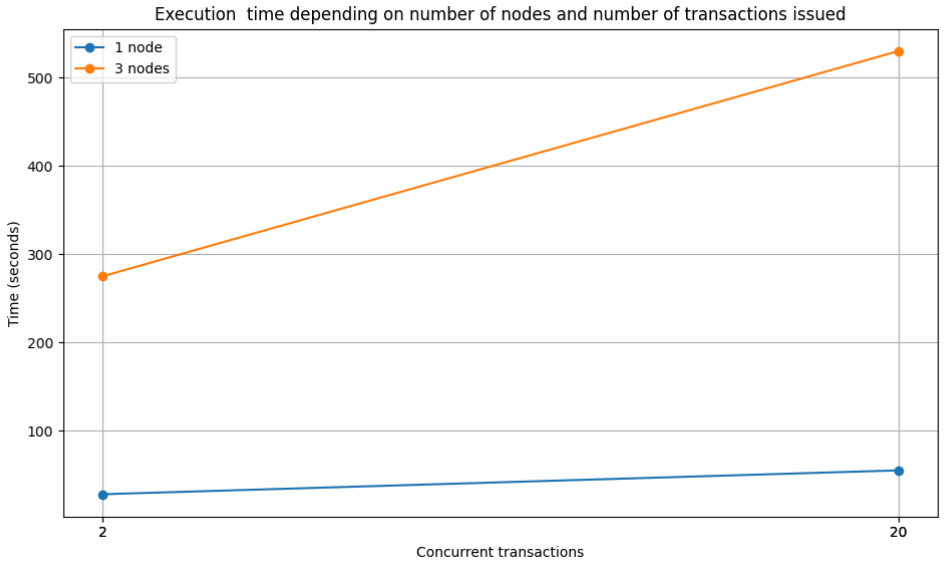
Performance test with concurrent transactions

We want to record the change in performance brought by a changing number of concurrent transactions issued by a single node to the network.

Methodology

The experiment has been done by writing a code that would issue multiple concurrent transactions by a single node. Actually the transactions are not really concurrent because they're issued in a for cycle, but the cycle doesn't wait for the resolution of the transactions because they're called in asynchronous mode. Then, once they've been issued, it is waited until all the transactions are present in the blocks of the chain.
The experiment has been evaluated on a test net of 1 or 3 nodes (brought up with Kurtosis) with the same modalities of the first experiment. It has been tested for 2 or 20 concurrent transactions.

Results



It is observed that in the case of 3 nodes the execution time is much higher, which was expected since a consensus protocol must be run while with only 1 node there is no actual consensus. In both cases the time grows with the growth of the number of concurrent transactions.

Comments on tests and results

The tests have been run on commercial laptops, the available hardware was strongly limited, for this reason not all the tests could have been run on big test nets. Probably running the same tests on more powerful PCs would lead to different execution times.

However we have still observed that increasing the load on the net, either due to a high number of nodes in the net or an increase of concurrent transactions, generally leads to an increase in response time.

Source code

The source code can be pulled from the following github repository: https://github.com/simonaderisi/Supply-Chain-Smart-Contract

Installing, configuring and running the software

To configure and run the software follow the following step:

1. Install Kurtosis as in https://docs.kurtosis.com/install/#ii-install-the-cli
2. Set up Hardhat as in https://hardhat.org/tutorial/setting-up-the-environment
3. Install all the packages with the `npm install` command (in alternative use `yarn install`)
4. Start kurtosis with the command `kurtosis run --enclave local-eth-testnet github.com/ethpandaops/ethereum-package --args-file eth-network-params.yaml`
5. Define local network connection in hardhat config file as in https://docs.kurtosis.com/how-to-local-eth-testnet#configure-hardhat-to-use-the-local-testnet
6. Compile the contracts with `npx hardhat compile`
7. Test the contracts with `npx hardhat test —-network localnet`
8. Deploy contracts with `npx hardhat ignition deploy ./ignition/modules/SupplyChain.js --network localnet`
9. Execute custom task with `npx hardhat --verbose —-network localnet fire --address <deployed_contract_address>`

Conclusions

Distributed ledger technology is a promising asset. Creative applications of this technology can bring its many benefits to different aspects of our economy. Not only financially. But also qualitatively. The technology is quite complex and the learning curve seems to be quite steep. Though effort and patience take a long way forward. We were able to find, deploy, execute and test a solution in the literature. It has been interesting learning methodologies, technologies and frameworks. We have learned through this project the complexities of developing DLTs. Surely distributed ledger technology is a powerful and valuable innovation.