Food Supply Chain Management Using Hyperledger Fabric

Tanisha Fonseca 40266436
Concordia University

1 Abstract

In today's globalized world, food supply chains face increasing scrutiny over issues such as transparency, traceability, and product authenticity. Traditional supply chain systems suffer from inefficiencies, lack of real-time monitoring, and vulnerability to fraud, resulting in reduced consumer trust and operational delays. This project introduces a decentralized, transparent and immutable food supply chain management solution using Hyperledger Fabric. Using blockchain technology, the system ensures the secure tracking of food products from manufacturing to final delivery. It enables stakeholders, including manufacturers, wholesalers, distributors, retailers, and consumers, to interact on a tamper-proof platform that guarantees authenticity and improves operational efficiency. The architecture includes a React.js front-end, Node.js middleware with REST API, and smart contracts written in Go, backed by a multi-organization Fabric network with a robust certificate authority structure. This report outlines system design, implementation, and impact, demonstrating how blockchain can revolutionize modern supply chains.

2 Problem Statement

The traditional global food supply chain, involving various stakeholders such as manufacturers, distributors, and retailers, faces major challenges such as limited transparency, fraud risk, and inefficiencies from manual verification. These systems often lack real-time tracking, making it difficult to trace a product's origin or ensure its authenticity an increasing concern among consumers. Manual transaction processes, involving multiple intermediaries, typically take 15 to 20 minutes and are prone to errors and inconsistencies.

To address these issues, this project proposes a blockchain-based solution using Hyperledger Fabric. The goal is to create a decentralized and tamper-proof platform that offers real-time updates and end-to-end traceability throughout the supply chain. This system will improve transparency, reduce the potential for fraud, and streamline verification processes, ultimately improving consumer trust and operational efficiency.

3 Solution

To address the challenges in the food supply chain, we developed a full-stack blockchain-based application using Hyperledger Fabric. The system improves transparency, security, and efficiency by enabling real-time product tracking, secure data sharing, and immutable record keeping between all participants in the supply chain. Ensure end-to-end traceability from the manufacturer to the consumer, reduces the risk of fraud, and builds trust among stakeholders through verifiable transactions.

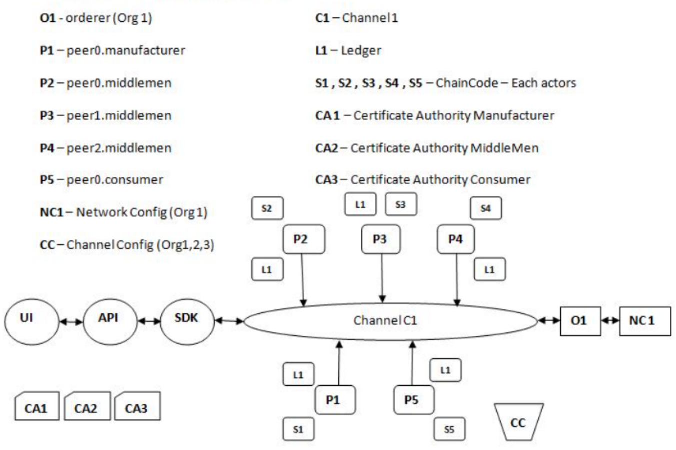

O1 - orderer (Org 1)
P1 – peer0.manufacturer
P2 – peer0.middlemen
P3 – peer1.middlemen
P4 – peer2.middlemen
P5 – peer0.consumer
NC1 – Network Config (Org1)
CC – Channel Config (Org1,2,3)
C1 – Channel1
L1 – Ledger
S1 , S2 , S3 , S4 , S5 – ChainCode – Each actors
CA1 – Certificate Authority Manufacturer
CA2 – Certificate Authority MiddleMen
CA3 – Certificate Authority Consumer
UI
API
SDK
Channel C1
O1
NC1
P1
P2
P3
P4
P5
L1
S1
S2
S3
S4
S5
CA1
CA2
CA3
CC

Figure 1: System architecture

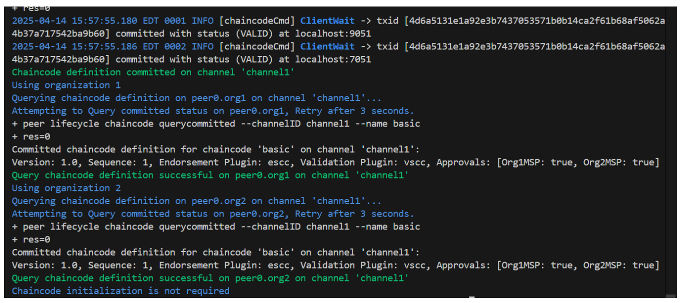
4 System Components

4.1 Network Structure

The Blockchain Network is a customized Hyperledger Fabric network, which is set up to support the food supply chain management system. It consists of three organizations: Manufacturer, Middlemen (Wholesalers, Retailers and Distributors) and Consumer. In total, there are five peers, one orderer, and one channel. This decentralized structure allows each participant to maintain a secure and transparent record of product movements throughout the supply chain. The inclusion of multiple peers ensures fault tolerance, while the orderer plays a central role in maintaining the consensus and integrity of transactions. The channel is used to isolate the network's data, providing a secure environment for transactions and communication among the organizations.

4.2 Chaincode Deployment

The Chaincode, written in Golang, serves as the smart contract layer in the Hyperledger Fabric network. It defines the business logic for the food supply chain, managing operations such as creating new products, updating product information, and tracking the product's journey from manufacturer to consumer. Chaincode is responsible for enforcing the rules and processes involved in the product life-cycle, ensuring that all actions are compliant with the defined business logic. For example, when a product moves from one supply chain actor to another, the chain code records and updates the status, ensuring that all participants have an accurate and immutable record of the transaction. It also enables the querying of product data based on various parameters such as product ID or user role, allowing stakeholders to track products at any given point in the supply chain.



```
+ res=0
2025-04-14 15:57:55.180 EDT 0001 INFO [chaincodeCmd] ClientWait -> txid [4d6a5131e1a92e3b7437053571b0b14ca2f61b68af5062a
4b37a717542ba9b60] committed with status (VALID) at localhost:9051
2025-04-14 15:57:55.186 EDT 0002 INFO [chaincodeCmd] ClientWait -> txid [4d6a5131e1a92e3b7437053571b0b14ca2f61b68af5062a
4b37a717542ba9b60] committed with status (VALID) at localhost:7051
Chaincode definition committed on channel 'channel1'
Using organization 1
Querying chaincode definition on peer0.org1 on channel 'channel1'...
Attempting to Query committed status on peer0.org1, Retry after 3 seconds.
+ peer lifecycle chaincode querycommitted --channelID channel1 --name basic
+ res=0
Committed chaincode definition for chaincode 'basic' on channel 'channel1':
Version: 1.0, Sequence: 1, Endorsement Plugin: escc, Validation Plugin: vscc, Approvals: [Org1MSP: true, Org2MSP: true]
Query chaincode definition successful on peer0.org1 on channel 'channel1'
Using organization 2
Querying chaincode definition on peer0.org2 on channel 'channel1'...
Attempting to Query committed status on peer0.org2, Retry after 3 seconds.
+ peer lifecycle chaincode querycommitted --channelID channel1 --name basic
+ res=0
Committed chaincode definition for chaincode 'basic' on channel 'channel1':
Version: 1.0, Sequence: 1, Endorsement Plugin: escc, Validation Plugin: vscc, Approvals: [Org1MSP: true, Org2MSP: true]
Query chaincode definition successful on peer0.org2 on channel 'channel1'
Chaincode initialization is not required
```

Figure 2: Chaincode Initialized

4.3 Frontend and Middleware Architecture

The front-end of the food supply chain management system is developed using React.js, offering a responsive and user-friendly interface for all supply chain participants to access real-time blockchain data such as product status, order history, and transaction details. The middleware, built with Node.js, uses REST APIs and the Hyperledger Fabric SDK to connect the front-end to the blockchain network, enabling smooth handling of transactions, product tracking, and user actions. This architecture ensures efficient, secure, and scalable communication throughout the system.

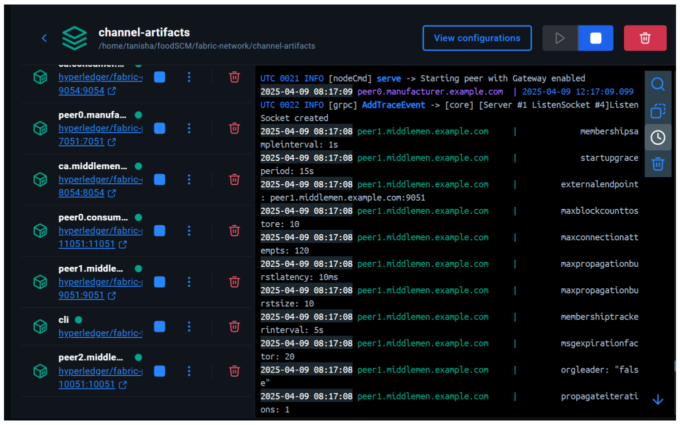
5 Network Setup

The network setup process for Hyperledger Fabric begins with the generation of certificates. First, a directory is created for the supply chain project, in this case, `Scm`. Once the necessary files are placed, including the `bin` folder from Fabric samples, the cryptographic configuration file (`crypto-config.yaml`) is written to define the identities and roles of each participant in the network. With this configuration in place, certificates are generated using the `cryptogen` tool, which ensures that each organization within the network (such as the Manufacturer, Middlemen and Consumer) has its own cryptographic identity. The certificates are saved in an output directory for later use.

After the certificates are generated, the next step is to create the configuration files, including the `configtx.yaml`, which defines the configuration settings for the channel and ordering service. With the configuration in place, the genesis block for the network is created using the `configtxgen` tool, initializing the ordering service that coordinates the flow of transactions in the network. The genesis block is essential for bootstrapping the network and is the starting point for the Fabric network setup.

Once the genesis block is created, the channel is configured and created using the `configtxgen` tool. The command configtxgen -profile SupplyChainChannel creates the `supplychainchannel.tx` file, which contains the channel's configuration and is used to establish the communication layer across the network. This file is crucial for defining how peers from different organizations interact within the channel. The channel ID `supplychainchannel` is then passed to all organizations for them to join.

Next, anchor peers are updated. Anchor peers are specific peers in an organization that are used to facilitate cross-organization communication within the channel. This step ensures that each organization's anchor peer is recognized within the channel. For example, the `configtxgen` command updates the anchor peer for the Middlemen organization, specifying its role and communication capabilities within the channel. The command ensures that the Middlemen organization's peer is appropriately set up to interact with other organizations in the network.



```
UTC 0021 INFO [nodeCmd] serve -> Starting peer with Gateway enabled
2025-04-09 08:17:09 peer0.manufacturer.example.com  | 2025-04-09 12:17:09.099
UTC 0022 INFO [grpc] AddTraceEvent -> [core] [Server #1 ListenSocket #4]Listen
Socket created
2025-04-09 08:17:08 peer1.middlemen.example.com     |            membershipsa
mpleinterval: 1s
2025-04-09 08:17:08 peer1.middlemen.example.com     |            startupgrace
period: 15s
2025-04-09 08:17:08 peer1.middlemen.example.com     |        externalendpoint
: peer1.middlemen.example.com:9051
2025-04-09 08:17:08 peer1.middlemen.example.com     |        maxblockcounttos
tore: 10
2025-04-09 08:17:08 peer1.middlemen.example.com     |        maxconnectionatt
empts: 120
2025-04-09 08:17:08 peer1.middlemen.example.com     |        maxpropagationbu
rstlatency: 10ms
2025-04-09 08:17:08 peer1.middlemen.example.com     |        maxpropagationbu
rstsize: 10
2025-04-09 08:17:08 peer1.middlemen.example.com     |        membershiptracke
rinterval: 5s
2025-04-09 08:17:08 peer1.middlemen.example.com     |        msgexpirationfac
tor: 20
2025-04-09 08:17:08 peer1.middlemen.example.com     |        orgleader: "fals
e"
2025-04-09 08:17:08 peer1.middlemen.example.com     |        propagateiterati
ons: 1
```

Figure 3: Docker containers

Finally, the Docker Compose network launch is initiated. This involves bringing up the Fabric network using

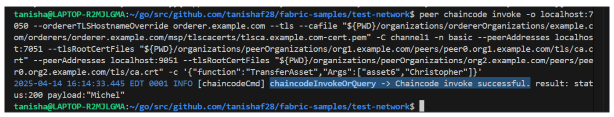
Docker Compose, which orchestrates the deployment of various services such as orderers, peers, and Certificate Authorities (CAs). Each peer corresponds to an organization, and the orderers are responsible for managing the consensus mechanism of the network. The network is launched with these components running in Docker containers, allowing the blockchain network to function effectively.

6 Chaincode Deployment

The deployment of chaincode involves several important steps, starting with installing the chaincode on all relevant peers. The chaincode contains the business logic that defines how products are created, updated, transferred, and queried within the supply chain system. To deploy the chaincode, it is first installed on each peer that will interact with it. This is done using the `peer chaincode install` command, which installs the chaincode package on the selected peers.

After the chaincode is installed, it needs to be instantiated on the channel, so that it is available for use. This step is executed using the `peer chaincode instantiate` command, specifying the version, channel name, and any initialization parameters required by the chaincode. Once instantiated, the chaincode is fully deployed and ready for interaction.

Once the chaincode is deployed and instantiated, the next step is to invoke its functions. This enables users to interact with the blockchain and updates the ledger. For instance, the createProduct function can be called by the Manufacturer to register a new product, while the updateProduct function can be used by other supply chain participants such as Wholesalers, Distributors, and Retailers to modify the product's status as it progresses through the supply chain. The peer chaincode invoke command initiates a transaction proposal, which is endorsed by peers and then recorded on the ledger. Invoking chaincode is crucial, as it activates the business logic and updates the blockchain's current state.



```
tanisha@LAPTOP-R2MJLGMA:~/go/src/github.com/tanishaf28/fabric-samples/test-network$ peer chaincode invoke -o localhost:7
050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.c
om/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C channel1 -n basic --peerAddresses localhos
t:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.c
rt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/pee
r0.org2.example.com/tls/ca.crt" -c '{"function":"TransferAsset","Args":["asset6","Christopher"]}'
2025-04-14 16:14:33.445 EDT 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: stat
us:200 payload:"Michel"
tanisha@LAPTOP-R2MJLGMA:~/go/src/github.com/tanishaf28/fabric-samples/test-network$
```

Figure 4: Chaincode Deployment

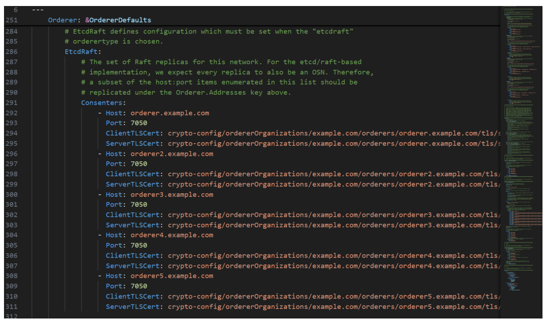
The Hyperledger Fabric network will include three organizations Manufacturer, Middlemen, and Consumer—with five peers, one orderer, and a dedicated channel. Each organization will have its certificate authority for secure identity management. This blockchain-based supply chain solution will enhance business efficiency, reduce fraud, and increase trust across stakeholders.

7 Application Architecture and Flow

The system begins with user onboarding, where an Admin enrolls all participants in the supply chain, including the Manufacturer, Wholesaler, Distributor, Retailer, and Consumer. Product creation is exclusive to the Manufacturer, who is the only entity authorized to add new products to the blockchain. Once a product is created, it undergoes a transition through the supply chain, moving from the Manufacturer to the Wholesaler, then to the Distributor, and finally to the Retailer. Consumers can then place orders for products and mark them as delivered once received, completing the product's journey through the supply chain. This ensures traceability, accountability, and transparency at every stage of the product's life cycle.

8 Results

The Hyperledger Fabric network for this supply chain system is configured with a RAFT consensus mechanism, providing fault tolerance and ensuring high availability for the network. The network consists of five orderers running on port 7050, which manage the consensus process and validate transactions across the network. The network is structured with three organizations: Manufacturer, Consumer, and Middlemen. Each organization plays a distinct role in the supply chain, with Middlemen having three peers to handle transactions and smart contract executions, while Manufacturer and Consumer each have a single peer. This setup ensures efficient transaction processing and enhances the scalability of the system, allowing seamless communication and coordination between all participants while maintaining data integrity and security across the blockchain network.



```
---
Orderer: &OrdererDefaults
        # EtcdRaft defines configuration which must be set when the "etcdraft"
        # orderertype is chosen.
        EtcdRaft:
            # The set of Raft replicas for this network. For the etcd/raft-based
            # implementation, we expect every replica to also be an OSN. Therefore,
            # a subset of the host:port items enumerated in this list should be
            # replicated under the Orderer.Addresses key above.
            Consenters:
                - Host: orderer.example.com
                  Port: 7050
                  ClientTLSCert: crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com/tls/s
                  ServerTLSCert: crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com/tls/s
                - Host: orderer2.example.com
                  Port: 7050
                  ClientTLSCert: crypto-config/ordererOrganizations/example.com/orderers/orderer2.example.com/tls/
                  ServerTLSCert: crypto-config/ordererOrganizations/example.com/orderers/orderer2.example.com/tls/
                - Host: orderer3.example.com
                  Port: 7050
                  ClientTLSCert: crypto-config/ordererOrganizations/example.com/orderers/orderer3.example.com/tls/
                  ServerTLSCert: crypto-config/ordererOrganizations/example.com/orderers/orderer3.example.com/tls/
                - Host: orderer4.example.com
                  Port: 7050
                  ClientTLSCert: crypto-config/ordererOrganizations/example.com/orderers/orderer4.example.com/tls/
                  ServerTLSCert: crypto-config/ordererOrganizations/example.com/orderers/orderer4.example.com/tls/
                - Host: orderer5.example.com
                  Port: 7050
                  ClientTLSCert: crypto-config/ordererOrganizations/example.com/orderers/orderer5.example.com/tls/
                  ServerTLSCert: crypto-config/ordererOrganizations/example.com/orderers/orderer5.example.com/tls/
```

Figure 5: Orderer Setup

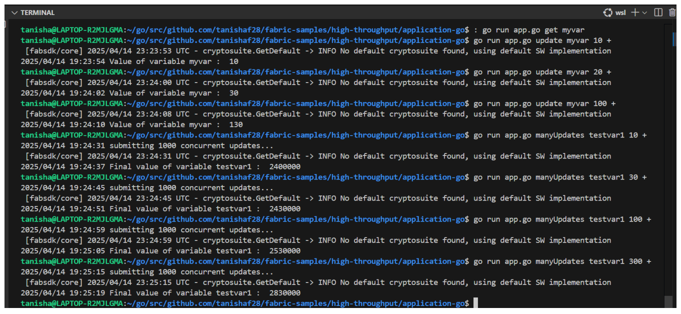
8.1 Unit Testing

The unit testing process for the food supply chain system ensures the functionality of various chaincode functions, validating that they perform as expected in different scenarios. For instance, the `deliveredProduct` function is tested with correct, missing, and non-existent product IDs, expecting it to update the product status to ”Delivered”, throw an error when no product ID is provided, and return an error when the product ID does not exist. Similar tests are conducted for functions like `sendToWholesaler`, `sendToDistributor`, `sendToRetailer`, and `sellToConsumer`, which check the correct flow of products between different supply chain participants. Each of these functions is tested with valid and invalid inputs, including missing IDs and incorrect user roles, ensuring that appropriate errors are raised in cases of incorrect or missing data. Additionally, the `queryAsset` and `queryAll` functions are tested to verify that product details are returned correctly when the right arguments are supplied, and that error messages are generated when the required arguments are missing. This thorough testing helps ensure that the chain code handles all possible input cases robustly, providing consistent and reliable behaviour in real-world use.

Function & Input	**Expected Outcome**
`deliveredProduct`[”ProductID”]	Status updated to ”Delivered”
`sendToWholesaler`[”ProductID”, ”WholesalerID”]	Product sent to wholesaler
`sendToDistributor`[”ProductID”, ”DistributorID”]	Product sent to distributor
`sendToRetailer`[”ProductID”, ”RetailerID”]	Product sent to retailer
`sellToConsumer`[”ProductID”]	Product sold to consumer
`queryAsset` [”ProductID”]	Returns product details
`queryAll` [”Product”]	Returns all products

Table 1: Unit Test Summary

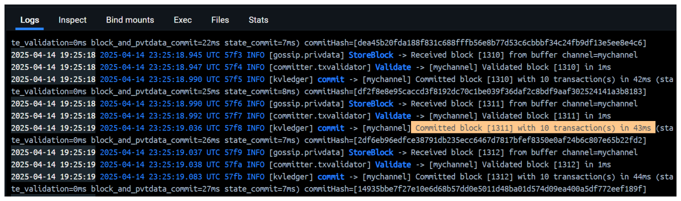
8.2 Additional Testing



```
tanisha@LAPTOP-R2MJLGMA:~/go/src/github.com/tanishaf28/fabric-samples/high-throughput/application-go$ : go run app.go get myvar
tanisha@LAPTOP-R2MJLGMA:~/go/src/github.com/tanishaf28/fabric-samples/high-throughput/application-go$ go run app.go update myvar 10 +
 [fabsdk/core] 2025/04/14 23:23:53 UTC - cryptosuite.GetDefault -> INFO No default cryptosuite found, using default SW implementation
2025/04/14 19:23:54 Value of variable myvar :  10
tanisha@LAPTOP-R2MJLGMA:~/go/src/github.com/tanishaf28/fabric-samples/high-throughput/application-go$ go run app.go update myvar 20 +
 [fabsdk/core] 2025/04/14 23:24:00 UTC - cryptosuite.GetDefault -> INFO No default cryptosuite found, using default SW implementation
2025/04/14 19:24:02 Value of variable myvar :  30
tanisha@LAPTOP-R2MJLGMA:~/go/src/github.com/tanishaf28/fabric-samples/high-throughput/application-go$ go run app.go update myvar 100 +
 [fabsdk/core] 2025/04/14 23:24:08 UTC - cryptosuite.GetDefault -> INFO No default cryptosuite found, using default SW implementation
2025/04/14 19:24:10 Value of variable myvar :  130
tanisha@LAPTOP-R2MJLGMA:~/go/src/github.com/tanishaf28/fabric-samples/high-throughput/application-go$ go run app.go manyUpdates testvar1 10 +
2025/04/14 19:24:31 submitting 1000 concurrent updates...
 [fabsdk/core] 2025/04/14 23:24:31 UTC - cryptosuite.GetDefault -> INFO No default cryptosuite found, using default SW implementation
2025/04/14 19:24:37 Final value of variable testvar1 :  2400000
tanisha@LAPTOP-R2MJLGMA:~/go/src/github.com/tanishaf28/fabric-samples/high-throughput/application-go$ go run app.go manyUpdates testvar1 30 +
2025/04/14 19:24:45 submitting 1000 concurrent updates...
 [fabsdk/core] 2025/04/14 23:24:45 UTC - cryptosuite.GetDefault -> INFO No default cryptosuite found, using default SW implementation
2025/04/14 19:24:51 Final value of variable testvar1 :  2430000
tanisha@LAPTOP-R2MJLGMA:~/go/src/github.com/tanishaf28/fabric-samples/high-throughput/application-go$ go run app.go manyUpdates testvar1 100 +
2025/04/14 19:24:59 submitting 1000 concurrent updates...
 [fabsdk/core] 2025/04/14 23:24:59 UTC - cryptosuite.GetDefault -> INFO No default cryptosuite found, using default SW implementation
2025/04/14 19:25:05 Final value of variable testvar1 :  2530000
tanisha@LAPTOP-R2MJLGMA:~/go/src/github.com/tanishaf28/fabric-samples/high-throughput/application-go$ go run app.go manyUpdates testvar1 300 +
2025/04/14 19:25:15 submitting 1000 concurrent updates...
 [fabsdk/core] 2025/04/14 23:25:15 UTC - cryptosuite.GetDefault -> INFO No default cryptosuite found, using default SW implementation
2025/04/14 19:25:19 Final value of variable testvar1 :  2830000
tanisha@LAPTOP-R2MJLGMA:~/go/src/github.com/tanishaf28/fabric-samples/high-throughput/application-go$
```

Figure 6: Updates per Transaction

In this experiment, we evaluate the throughput capabilities of a Hyperledger Fabric test network by testing a high-throughput chaincode model. The goal is to simulate network performance under thousands of concurrent transactions, particularly when updating the same asset. The test network uses a single channel and custom chaincode designed to avoid version conflicts, which occur in traditional models where asset data is stored under a single key. The high-throughput model addresses this by storing updates as deltas across multiple rows, reducing key contention and allowing more parallel transactions. A Go-based application is used to simulate both high-throughput and traditional models. In tests with 1000 concurrent transactions, the high-throughput model successfully aggregates updates, while the traditional model experiences version conflicts. This experiment highlights the importance of structuring chain code to handle high concurrency and improve throughput.



```
te_validation=0ms block_and_pvtdata_commit=22ms state_commit=7ms) commitHash=[dea45b20fda188f831c688fffb56e8b77d53c6cbbbf34c24fb9df13e5ee8e4c6]
2025-04-14 19:25:18 2025-04-14 23:25:18.945 UTC 57f3 INFO [gossip.privdata] StoreBlock -> Received block [1310] from buffer channel=mychannel
2025-04-14 19:25:18 2025-04-14 23:25:18.947 UTC 57f4 INFO [committer.txvalidator] Validate -> [mychannel] Validated block [1310] in 1ms
2025-04-14 19:25:18 2025-04-14 23:25:18.990 UTC 57f5 INFO [kvledger] commit -> [mychannel] Committed block [1310] with 10 transaction(s) in 42ms (sta
te_validation=0ms block_and_pvtdata_commit=25ms state_commit=8ms) commitHash=[df2f8e8e95caccd3f8192dc70c1be039f36daf2c8bdf9aaf302524141a3b8183]
2025-04-14 19:25:18 2025-04-14 23:25:18.990 UTC 57f6 INFO [gossip.privdata] StoreBlock -> Received block [1311] from buffer channel=mychannel
2025-04-14 19:25:18 2025-04-14 23:25:18.992 UTC 57f7 INFO [committer.txvalidator] Validate -> [mychannel] Validated block [1311] in 1ms
2025-04-14 19:25:19 2025-04-14 23:25:19.036 UTC 57f8 INFO [kvledger] commit -> [mychannel] Committed block [1311] with 10 transaction(s) in 43ms (sta
te_validation=0ms block_and_pvtdata_commit=26ms state_commit=7ms) commitHash=[2df6eb96edfce38791db235ecc6467d7817bfef8350e0af24b6c807e65b22fd2]
2025-04-14 19:25:19 2025-04-14 23:25:19.037 UTC 57f9 INFO [gossip.privdata] StoreBlock -> Received block [1312] from buffer channel=mychannel
2025-04-14 19:25:19 2025-04-14 23:25:19.038 UTC 57fa INFO [committer.txvalidator] Validate -> [mychannel] Validated block [1312] in 1ms
2025-04-14 19:25:19 2025-04-14 23:25:19.083 UTC 57fb INFO [kvledger] commit -> [mychannel] Committed block [1312] with 10 transaction(s) in 44ms (sta
te_validation=0ms block_and_pvtdata_commit=27ms state_commit=7ms) commitHash=[14935bbe7f27e10e6d68b57dd0e5011d48ba01d574d09ea400a5df772eef189f]
```

Figure 7: Commited Blocks

9 Discussion

To further demonstrate the efficiency of the high-throughput smart contract logic, we conducted a series of tests using both sequential and concurrent update methods. Initially, the variable myvar was updated sequentially using the update function with values 10, 20, and 100. As expected, the values were cumulatively added, resulting in a final value of 130. This confirms the correct functioning of standard transaction submission. Subsequently, we used the manyUpdates method to perform 1000 concurrent updates on a variable named testvar1 with different update values: 10, 30, 100, and 300. The final results: 2,400,000; 2,430,000; 2,530,000; and 2,830,000 respectively clearly show that the updates were successfully batched and processed in parallel, with no conflicts or data inconsistencies. These outcomes validate the system's capability to handle high transaction throughput efficiently, leveraging Fabric's ability to commit multiple independent writes in a single block.

The graph, titled ”Effective Throughput vs Updates Per Transaction”, illustrates how the system's throughput performance scales with the number of updates performed within a single transaction. The X-axis represents the number of updates per transaction, while the Y-axis denotes the effective throughput measured in updates per second.

From the graph, we observe a positive linear trend as the number of updates per transaction increases, the effective throughput also increases significantly. For example, with a small number of updates (around 10 to 30), the throughput remains relatively low (a few thousand updates per second). However, as the number of updates per transaction rises to

100, 200, and 300, the throughput correspondingly jumps to approximately 15,000, 35,000, and 75,000 updates per second.

This clearly demonstrates the efficiency of batching updates within a single transaction, especially in a high-throughput chaincode model. Instead of processing thousands of individual transactions each competing for ledger access grouping multiple updates into one transaction dramatically reduces contention, minimizes overhead, and significantly boosts the effective update rate. This finding is especially important in blockchain systems like Hyperledger Fabric where transaction conflict and commit latency can become major bottlenecks under high loads.

In summary, this graph supports the core design choice of aggregating multiple updates per transaction, as it leads to scalable performance and maximizes system throughput in scenarios involving frequent state modifications. In Food Supply chain, the graph supports aggregating multiple product updates per transaction, which improves scalability and throughput. This is crucial for handling frequent changes in ownership, location, or condition, making the system more efficient and responsive for real-time traceability in large-scale supply chains.

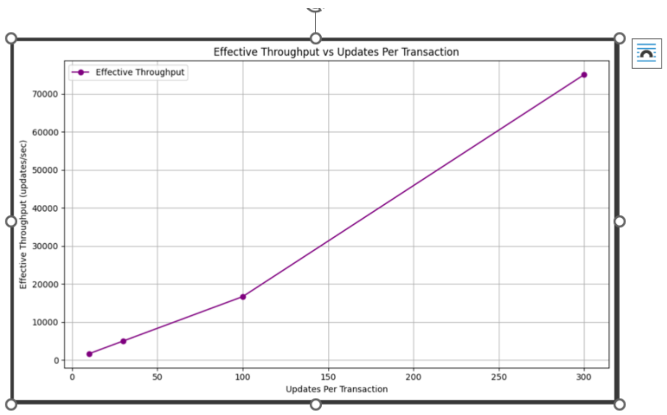


Figure 8: Effective throughput vs updates per Transaction

The following graph (Figure 9) compares the baseline and optimized Hyperledger Fabric setups. The traditional readings were taken from the default network configuration, while the optimized readings were obtained by applying enhancements like batching updates, improving chain code, and tuning network parameters. The results show that these optimizations significantly boost throughput as the number of updates per transaction increases.The x-axis denotes the number of updates per transaction, while the y-axis represents the effective throughput measured in updates per second. The optimized approach, indicated by the solid green line with circular markers, demonstrates a significantly higher throughput as the number of updates increases, suggesting better scalability and efficiency. In contrast, the traditional method, shown by the dashed red line with square markers, exhibits a more modest growth in throughput. This comparison highlights that the optimized approach delivers superior performance, especially at higher transaction volumes, making it a more effective solution for high-throughput applications.

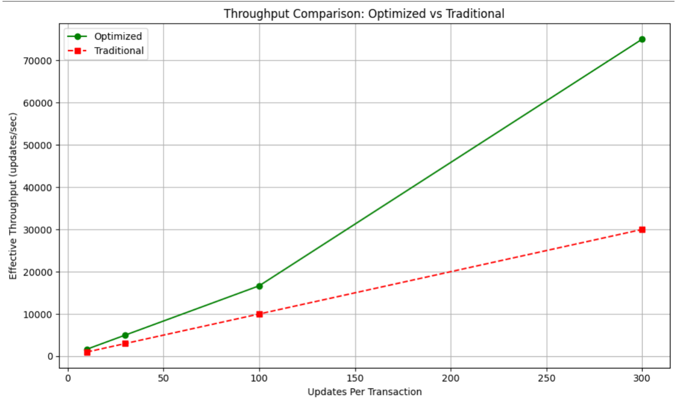


Figure 9: Optimized vs Traditional

10 Conclusion

In conclusion, the blockchain-based Food Supply Chain Management system using Hyperledger Fabric enhances transparency, traceability, and security. Leveraging decentralized ledger technology, it securely records transactions from production to consumption, reducing fraud and building trust among stakeholders. The system efficiently tracks product flow through roles like Manufacturer, Wholesaler, Distributor, Retailer, and Consumer. As future work, the system can be expanded to support large-scale deployment across multiple machines, integrate email notifications for key events, and enhance user experience through advanced frontend and API testing. Additionally, incorporating real-time data analytics could further improve the accuracy and responsiveness of the supply chain.

References

[1] Miguel Pincheira Caro; Muhammad Salek Ali;, ”Blockchain-based traceability in Agri-Food supply chain management,” IEEE IOT Vertical and Topical Summit Conference, 2018.

[2] Andreas Kamilaris, Agusti Fonts, ”The rise of blockchain technology in agriculture and food supply chains” Volume 91, September 2019

[3] Tanishaf28. ”FoodSupplyChain Repository” Available at: `https://github.com/tanishaf28/FoodSupplyChain.git` (Accessed: 2025-04-14).