COMBINED MANGO QUERY AND COMPOSITE KEY STRATEGY FOR QUERY OPTIMIZATION IN HYPERLEDGER FABRIC SUPPLY CHAIN SYSTEMS

A MASTER'S THESIS

Submitted to
Graduate School of Electrical Engineering

By

FABIANO ABBEY KARO SEKALI

201012410042

In partial fulfillment of the requirements
for the Degree of Master of Engineering

TELKOM UNIVERSITY
BANDUNG
2025

APPROVAL PAGE

MASTER'S THESIS

COMBINED MANGO QUERY AND COMPOSITE KEY STRATEGY FOR QUERY OPTIMIZATION IN HYPERLEDGER FABRIC SUPPLY CHAIN SYSTEMS

by

FABIANO ABBEY KARO SEKALI
201012410042

Approved and authorized to fulfil one of the requirements of
Program of Master of Electrical-Telecommunication Engineering
School of Electrical Engineering
Telkom University
Bandung

Bandung, 12th August, 2025

Supervisor	Co-Supervisor
Dr.Eng. Favian Dewanta, S.T., M.Eng	**Dr. Yudha Purwanto, S.T., M.T.**
NIP. 15870022	NIP. 02770066

SELF DECLARATION AGAINST PLAGIARISM

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct. I have full cited and referenced all materials and results that are not original to this work.

12th August, 2025
FABIANO ABBEY KARO SEKALI

Signature: 

ABSTRACT

Efficient data querying in Hyperledger Fabric-based systems remains a significant challenge due to the decentralized architecture and limited query flexibility. Even though the Hyperledger Fabric offers mechanisms such as Mango Query and composite keys, those mechanisms have their own limitations, in which Mango Query lacks performance on multicondition searches, while composite keys are rigid and context-dependent. Moreover, the reliance on historical or off-chain data poses consistency and trust issues in real-time applications. To overcome these limitations, this paper proposes a query strategy that dynamically chooses between Mango-based indexing and composite key access, depending on the number of query conditions, while operating entirely on a world-state database. Implemented in a simulated supply chain environment with 10,000 and 50,000 records, the proposed method achieves substantial latency improvements of over 90% in multicondition scenarios, while also supporting flexible query patterns including warehouse, timestamp, and responsible person. Compared to basic queries without indexing, this approach offers a scalable and efficient solution for permissioned blockchain environments, especially in supply chain systems where fast and accurate data retrieval is critical.

Keywords: Hyperledger Fabric, Data Retrieval, Supply chain, Composite Key, Mango Query.

ABSTRAK

Pencarian data yang efisien dalam sistem blockchain Hyperledger Fabric tetap memiliki tantangan signifikan pada arsitektur terdesentralisasi dan keterbatasan fleksibilitas pencarian. Meskipun Hyperledger Fabric menyediakan mekanisme seperti Mango Query dan kunci komposit, mekanisme tersebut memiliki keterbatasan masing-masing, di mana Mango Query kurang optimal dalam pencarian dengan kondisi ganda, sementara kunci komposit bersifat kaku dan bergantung pada konteks. Selain itu, ketergantungan pada data historis atau off-chain menimbulkan masalah konsistensi dan kepercayaan dalam aplikasi real-time. Untuk mengatasi keterbatasan ini, Tesis ini mengusulkan strategi pencarian yang secara dinamis memilih antara pengindeksan berbasis Mango dan akses Composite key, tergantung pada kondisi pencarian, sambil beroperasi sepenuhnya pada world state database. Dilaksanakan dalam lingkungan rantai pasokan simulasi dengan 10.000 dan 50.000 catatan, metode yang diusulkan mencapai peningkatan latensi yang signifikan lebih dari 90% dalam skenario multicondition, sambil juga mendukung pola kueri fleksibel termasuk gudang, *timestamp*, dan orang yang bertanggung jawab. Dibandingkan dengan kueri dasar tanpa pengindeksan, pendekatan ini menawarkan solusi yang skalabel dan efisien untuk lingkungan blockchain berizin, terutama dalam sistem rantai pasokan di mana pengambilan data yang cepat dan akurat sangat kritis.

Kata kunci: Hyperledger Fabric, Data Retrieval, Supply chain, Composite Key, Mango Query.

ACKNOWLEDGEMENTS

This thesis is compiled with the effort, help, and support from all supporting elements. The author would like to express the deepest gratitude and thanks to:

1. Allah SWT, for all the love, guidance and forgiveness in every mistake that the author has ever done and Rasulullah SAW, as role model who inspire writer in living life and trying to be better.

2. To my beloved parents and my younger sister for all the prayers and support that have been given to strengthen the author in completing this thesis.

3. Mr. Favian Dewanta as the first supervisor who has provided advice related to the thesis and has facilitated this thesis and paper.

4. Mr. Yudha Purwanto as the second supervisor who has provided advice related to this thesis and insight into Federated Learning.

5. Thanks to the MTE admin who has helped the author with the master's degree administration.

6. Thanks to Satria, Lutfi, Reynaldi as the author's friends who have become lecture friends and thesis comrades.

7. Thanks to the author's friends who have provided support and invited to play when the author was burned out.

   Perhaps that is all that the author can say. I apologize if there are misspelled of names. For the parties who were not mentioned in the above gratitude, the author apologizes profusely for the oversight. May Allah the All-Knowing, who has nothing to escapes His control, record your kindness and reward it with the best possible reward.

PREFACE

Alhamdu lillahi rabbil 'alamin, praise to Allah, the most gracious, the most merciful, with the mercy and guidance, the author has successfully finished this thesis with the title of "**COMBINED MANGO QUERY AND COMPOSITE KEY STRATEGY FOR QUERY OPTIMIZATION IN HYPERLEDGER FABRIC SUPPLY CHAIN SYSTEMS**". The author compiled this thesis to be filled in the graduation requirements in Program of Master of Electrical-Telecommunication Engineering, School of Electrical Engineering, Telkom University.

The suggestions for improving this thesis are highly appreciated. Hopefully, this thesis is expected to be improved and provided contributions for the reader and Indonesia especially for education and research of telecommunication on the future.

Bandung, 12th August, 2025

FABIANO ABBEY KARO SEKALI

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 Background

Blockchain Technology was Initially created as the basis for cryptocurrencies like Bitcoin, blockchain technology can now revolutionize a number of industries, particularly supply chain data sharing [1][2]. In the supply chain industry, timely access to data plays a critical role in ensuring efficiency and resilience of operations. Effective data sharing provides real-time informations into the supply chain information such as inventories, responsible person, and time. information obtained can be used by stakeholders for the benefit of the company.

However, conventional supply chain data systems often suffer from several limitations, including data inconsistency, siloed information, and inefficient query mechanisms when faced with large volumes of heterogeneous data. These limitations become more pronounced as organizations expand globally and the complexity of supply chain networks increases. As a result, stakeholders often struggle with delayed access to accurate data, leading to suboptimal inventory management and reduced responsiveness [3].

Blockchain-based platforms such as Hyperledger Fabric have been introduced to mitigate these issues by providing a decentralized, tamper-resistant infrastructure for data storage and sharing. Unlike public blockchain platforms, Hyperledger Fabric is designed for permissioned enterprise environments, where members are known and authenticated. It supports modular components such as pluggable consensus, chaincode (smart contract) execution, and configurable membership services, making it a flexible solution for a wide range of enterprise applications, including supply chain management [3].

Nevertheless, despite its modular and secure architecture, Hyperledger Fabric continues to face performance challenges in data retrieval, especially when operating over large-scale datasets. Querying the world state database—particularly in scenarios involving multi-condition filters or high data volume—can introduce substantial latency. The bottlenecks often occur during the endorsement and validation phases or as a result of inefficient query processing over unindexed fields [4][5].

Several studies have sought to address these performance limitations. For instance, Zhou et al. proposed LedgerData Refiner, an external data replication ap-

proach designed to improve query performance by mirroring blockchain data into off-chain databases [6]. Although effective in enhancing speed, this method introduces trust and consistency risks, as off-chain queries are no longer governed by the on-chain consensus and endorsement logic. On the other hand, Yan et al. presented an approach based on composite key design and LevelDB, demonstrating that structured keys can improve deterministic queries. However, their solution lacked support for rich query capabilities and was limited to simple key-value access patterns [7].

To overcome these limitations, this research proposes a hybrid query optimization strategy that combines Mango Query indexing and Composite Key construction within the CouchDB-based world state of Hyperledger Fabric. Mango Queries are well-suited for handling flexible, multi-field search conditions through JSON-based selectors and indexed fields. In contrast, composite keys enable high-speed, prefix-based lookups, particularly efficient for structured or hierarchical data access. By selectively applying the most suitable technique based on the complexity of the query—using Mango Queries for single-field filters and Composite Keys for multi-field lookups—this combined strategy seeks to enhance query performance while preserving data consistency and chaincode-level trust.

The proposed solution is evaluated using a simulated supply chain dataset of 10,000 and 50,000 records to reflect real-world blockchain data usage. By reducing query latency and improving scalability, the study aims to contribute to the development of more efficient data retrieval mechanisms for permissioned blockchain systems, enhancing their viability for large-scale industrial applications.

1.2 Theoretical Framework

This research is based on several main theories that underlie the development of query optimization strategies on Hyperledger Fabric blockchain systems. These theories include the concept of Distributed Ledger Technology, CouchDB database structure, Mango Query, and the use of Composite Key. The following is a description of the theories used.

1. Blockchain Theory and Distributed Technology (DLT)
   Blockchain is part of Distributed Ledger Technology (DLT), which allows recording transactions in a decentralized, transparent, and immutable manner. In the Hyperledger Fabric, DLT allows all licensed network members to participate in consensus and real-time tracking of asset [1]. This supports transparency and data integrity in supply chain management systems.

2. CouchDB and Mango Query on Hyperledger Fabric
   Hyperledger Fabric supports world state storage using CouchDB, a NoSQL document database that stores data in JSON format. Mango Query is a feature used to perform JSON selector-based searches, enabling flexible multi-attribute filters. Research by Zheng et al. (2022) showed that the performance of Mango Query is strongly influenced by the existence and structure of a suitable index [5].

3. Composite Key Theory in Chaincode
   Composite Key is a method of combining multiple attributes into one deterministic key to improve query efficiency. Hyperledger Fabric supports prefix-based search using the CreateCompositeKey function, which is proven to speed up data access when queries are performed on one specific attribute [3]. This approach is effective for simple search scenarios, such as by warehouse ID or product status.

1.3 Conceptual Framework/Paradigm

in this section there are 3 main points of the Conceptual Framework, such as Input, Process, and Output. Input is the data used for processing, which is in the form of inventory data containing *Product, Quantity, Warehouse, Status, Responsible person,* and *Timestamp*. In the process section, the data will be entered into the blockchain block using the private blockchain environment, Hyperledger Fabric. Next, is the output, the desired output is that when searching for data, the system will match the data with the Indices or Keys.

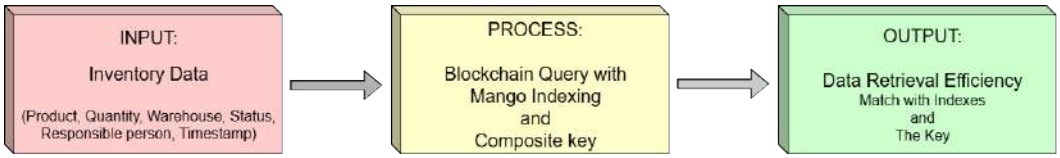

Fig. 1.1 Conceptual Framework.

1.4 Statement of the Problem

There are several problems with conventional Hyperledger Fabric query systems. This problem can certainly cause great disadvantages if it spreads to a larger scale. These problems are:

1. Low Data Retrieval Speed and Efficiency:
   In blockchain-based supply chain management systems such as Hyperledger Fabric, the efficiency of data retrieval is important because it directly affects the performance of the systems. Hyperledger Fabric has various query methods for data retrieval, including Mango Query and Composite Key. Each has advantages and limitations. Mango Query supports multi-attribute searches but is prone to performance degradation if the data volume is large or the index is not optimized, while Composite Key excels in single-attribute searches, but does not support the flexibility of complex queries. Reliance on any one method can lead to reduced performance, so an adaptive combination strategy is needed to overcome these limitations.

2. Limitations in data transparency:
   It cannot be denied that warehouse data is important because it includes data on goods, warehouses, and data on the person in charge of the warehouse. These data are very vulnerable to manipulation by irresponsible people.

1.5 Hypothesis

It is expected that the application of the combination strategy between Mango Query and Composite Key in the Hyperledger Fabric system can improve the efficiency of data retrieval compared to the use of each method separately. This strategy is expected to speed up query execution time, reduce resource consumption, and increase system scalability in supply chain management scenarios with large data volumes and various query complexities.

1.6 Assumption

1. The Hyperledger Fabric system has been properly configured and supports the use of CouchDB as the state database for Mango Query execution.

2. Mango indexes can be created and maintained regularly to support data search performance.

3. The key writing scheme for Composite Key has been designed with a consistent deterministic structure to support prefix-based search.

4. All experiments were run in a permissioned blockchain network environment with stable infrastructure specifications.

1.7 Scope and Delimitation

This research focuses on testing and evaluating the combination of Mango Query and Composite Key to improve data retrieval efficiency in the Hyperledger Fabric network. The scope of implementation is limited to the context of a supply chain management system, specifically related to inventory data that includes warehouse attributes, item status, person in charge, and recording time.

Experiments were conducted using one blockchain channel and one organization, as well as three organizations. The organization runs one peer instance connected to the CouchDB database as the state database. It should be noted that in the Hyperledger Fabric architecture, each peer has its own CouchDB, and as long as it is in the same channel and does not use private data, the contents of the world state in each CouchDB will be synchronous and identical.

This research also does not cover advanced security aspects such as private data collection, advanced encryption, or integration with external systems such as IoT. The main focus is on measuring query performance in terms of execution time, resource consumption, and efficiency of the data search strategy used.

1.8 Importance of the Study

This research aims to design, implement, and evaluate a combination strategy between Mango Query and Composite Key in data retrieval on Hyperledger Fabric permissioned blockchain network. This strategy is expected to improve data query efficiency in the context of supply chain systems, especially in scenarios with large data volumes and high complexity. The specific objectives of this research are:

1. Implement a data search function with a combination of Mango Query and Composite Key in a smart contract (chaincode).

2. Experimenting and measuring data retrieval performance based on time, CPU usage, and memory consumption.

3. Comparing the performance results of the combination strategy with conventional methods such as Mango-only and Composite-only.

1.9 Research Plan and Action Point

Month	Action Points
October 2024	Conduct initial research on Hyperledger Fabric, Mango Query, Composite Key, and blockchain-based inventory systems.
November 2024	Study the capabilities of CouchDB and Mango Query in Fabric; investigate prior optimization methods in rich queries.
December 2024	Set up Hyperledger Fabric testbed with CouchDB and implement chaincode for querying inventory data.
January 2025	Develop and test an adaptive query function combining Mango Query and Composite Key logic in Go chaincode.
February 2025	Design baseline functions; build workload scenarios for benchmark testing.
March 2025	Conduct single-user experiments; collect execution time, CPU usage, memory consumption, and result accuracy.
April 2025	Prepare experimental report and write research paper; analyze Mango vs Composite vs Adaptive performance.
May 2025	Extend experiment with Mango-only and Composite-only methods for deeper comparison and novelty enhancement.
June 2025	Finalize thesis document, including all results, analysis, discussion, and conclusions. Prepare for submission.

CHAPTER 2
BASIC CONCEPT

This chapter discusses the basic concepts of this thesis. These theories become introduction from the proposed design in this thesis, such as Mango Indexes, Composite Key, and Smart Contract.All explanations related to the theory used are explained in general terms.

2.1 Inventory Management System

Inventory Management System (IMS) is a system used to manage stock items in the warehouse. IMS serves to record the entry and exit of goods and ensure that stock quantities are monitored in order to minimize excess or shortage of inventory. IMS has become an important component in the supply chain. This is because IMS can provide efficiency and optimization in warehouse management [8]. This technology helps companies to simplify inventory tracking and stock auditing. With the use of IMS, companies can optimize other areas because the supply chain section has reached the desired level of efficiency [9].

With the development of technology, IMS has evolved to adopt Internet of Things (IoT) and Artificial Intelligence (AI) to improve tracking and optimize inventory data processing. For example, integration with IoT sensors allows the system to monitor the status of goods in real-time. Furthermore, AI is used to predict future stock based on the analysis of previous data[10]. With the integration of these two technologies, modern IMS is able to provide intelligent and automatic decisions, of course, this helps companies reduce errors [11].

In addition, the implementation of IMS can increase transparency and security in supply chain management. In conventional record-keeping, there is a potential for human error that results in mismatches between records and actual conditions in the field [12]. However, with the application of blockchain technology in IMS, inventory transactions can be recorded in a decentralized and immutable, thus providing the benefits of a better audit trail and reducing the potential for data manipulation from irresponsible parties[13][14].

2.2 Blockchain

Blockchain is a technology that was first introduced by Satoshi Nakamoto through Bitcoin. It provides secure, transparent, and immutable data recording. It is now expanding into various fields, including finance, logistics, and supply chain management. Unlike traditional centralized databases, blockchains store data in the form of blocks that are interconnected through nodes and cryptographic hash functions. This makes any changes to the data easily detectable by network participants[1].

Each block in the blockchain contains a transaction, a timestamp and, the hash of the previous block. Each block of many blocks will eventually form a chain that can no longer be changed at all [15]. The validation process on the blockchain is done through consensus mechanisms such as Proof of Work (PoW) or mining as used in the Bitcoin system, Proof of Stake (PoS) where the amount of crypto a user owns determines their ability to validate transactions as in Ethereum, or lighter consensus algorithms such as PBFT (Practical Byzantine Fault Tolerance) which is quite commonly used in permissioned blockchains such as Hyperledger Fabric due to the absence of mining mechanisms [16].

The correlation between supply chain management and blockchain is that blockchain technology can provide high traceability and data security. Various information related to the products, quantities, storage location, distribution time, and the identities of those persons in charge can be recorded and verified by the parties involved in the network in a transparent manner [17]. By adapting blockchain, companies can reduce dependence on third parties, minimize the risk of data manipulation, and human error. The ability of blockchain to create a system that can be trusted makes it a potential solution to improve security, efficiency, and most importantly, data integrity in logistics management systems.

2.2.1 Blockchain Structure

Blockchain structure consists of various components that work together to ensure the security and decentralization of the network. One of the main elements is the block, which stores a collection of transaction data. In order to efficiently verify the data in a block, blockchain utilizes a structure called Merkle Tree. This structure takes the form of a binary tree, where each node is a hash of the node below it. This allows the system to ensure the integrity of large amounts of data without having to read the entire block, an approach that is especially important in large-scale networks [18][15].

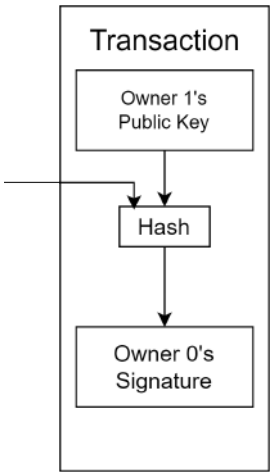

Fig. 2.1 Block Structure.

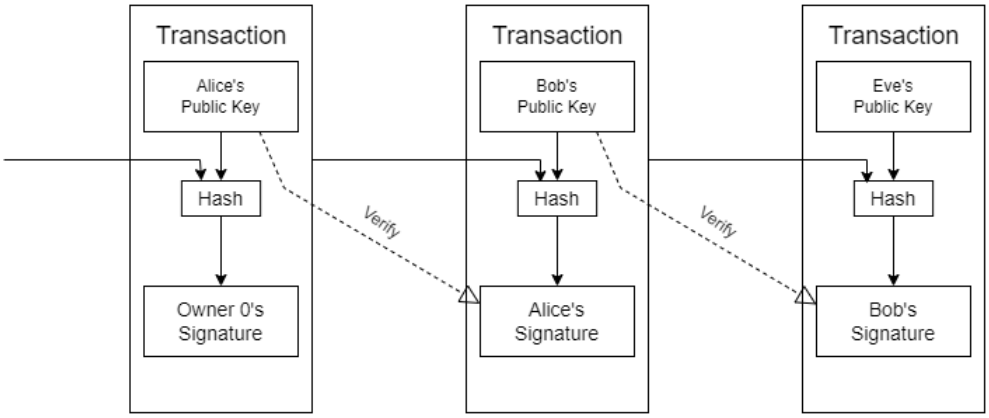

Fig. 2.2 Blockchain Structure.

2.2.2 Types of Blockchain

1. Public Blockchain: Public blockchain is an open, decentralized network where anyone can participate as a node, validate transactions, and access data. This type of blockchain uses distributed consensus, such as Proof of Work (PoW) or Proof of Stake (PoS) to verify transactions. The main examples of public blockchains are Bitcoin and Ethereum. Public blockchains are highly transparent, but they face challenges related to scalability and high energy costs as each transaction must be verified by many nodes around the world[19].

2. Private Blockchain: Private blockchains are closed distributed networks, in which only authorized parties can participate. Private blockchains are often used by enterprises for internal applications, such as supply chain management or sensitive data management, as they offer more control over data ac-

cess and participation. An example of a private blockchain implementation is Hyperledger Fabric, which is designed for enterprise applications. Private blockchains have higher efficiency than public blockchains due to the smaller number of nodes and simpler consensus mechanisms such as PBFT (Practical Byzantine Fault Tolerance)[19].

Table 2.1 Comparison of Public and Private Blockchain

Parameters	Public Blockchain	Private Blockchain
Access	Open to Anyone	Restricted by Permit
Decentralization	Fully decentralized	Restrictedly decentralized
Security	Managed by distributed consensus (PoW, PoS)	Managed by a trusted entity (PBFT)
Speed	Slower due to large number of nodes	Faster with fewer nodes
Usage	Cryptocurrency	Supply Chain
Example	Bitcoin and Ethereum	Hyperledger Fabric

2.3 Hyperledger Fabric

Hyperledger Fabric is a modular permissioned blockchain platform developed under the Hyperledger Project initiative by the Linux Foundation. In contrast to public blockchains such as Bitcoin or Ethereum, Hyperledger Fabric is intended to create a private network, where only entities that have been approved to enter the consortium can join, participate, and perform transactions. The Hyperledger Fabric architecture provides high flexibility in managing network components such as in terms of consensus, identity, and data storage, so that customization is very suitable for fulfilling enterprise needs such as supply chain management [20].

Hyperledger Fabric adopts a data model that separates transaction recording (ledger) and data current state (world state). For world state, Fabric supports various backend databases, one of which is often used is CouchDB. CouchDB allows document-based data search using JSON queries through “Rich Query”. With CouchDB, users can perform complex data searches using Mango Query. Mango Query provides flexibility in the customization of its index configuration, which

supports multi-attribute filtering and the use of indexes to improve performance. However, the performance of Mango Query is highly dependent on the availability and configuration of indexes that match the data query pattern [21].

Hyperledger Fabric also supports the use of smart contracts (chaincode) that can be programmed in various languages such as Go, JavaScript, and Java. Through chaincode, researchers can design condition-based search logic, including automatic selection strategies between Mango Query and Composite Key based on the type of incoming query. This feature makes Fabric an ideal platform for building blockchain systems with flexible yet efficient data search requirements.

2.3.1 Key Components: Peer, Orderer, Channel and Chaincode

The Hyperledger Fabric architecture is built from several main components that have specific and complementary functions to support transaction execution and data management in a decentralized yet controlled manner. The first and most central component is the peer, which is a network node in charge of storing the ledger and executing the chaincode. Peers also store the world state, which is the current status of all data that has been committed to the blockchain. In a network of multiple organizations, each organization generally has one or more peers that are independent, but synchronize their ledgers if they are in the same channel. Peers are divided into several types, such as endorsing peers that validate transactions through the execution of chaincode, and committing peers that receive transaction blocks from orderers to be saved to their local ledger.

The next component is the orderer, which is responsible for sorting and distributing transactions. The orderer acts as a coordination center to arrange validated transactions into blocks, and then distribute them to all peers in the network associated with a particular channel. Fabric supports several consensus algorithms for the orderer, such as RAFT and Kafka, which can be selected according to the availability of the network. With this approach, Fabric separates the process of validating transactions (endorsement) and determining the order of transactions (ordering), which is not commonly found on public blockchain platforms, thus improving the efficiency and scalability of the system.

Furthermore, channels are logical entities that facilitate communication and data exchange between organizations in the Fabric network. Channels enable the establishment of private data domains, where only organizations that are members of a particular channel can access the ledgers and transactions within it. This mechanism is the main foundation for the implementation of data privacy and isolation in the Hyperledger Fabric architecture.

In this research implementation, one channel is used to store all supply chain assets, assuming that all participating organizations are in the same data scope and have shared access to the ledger used.

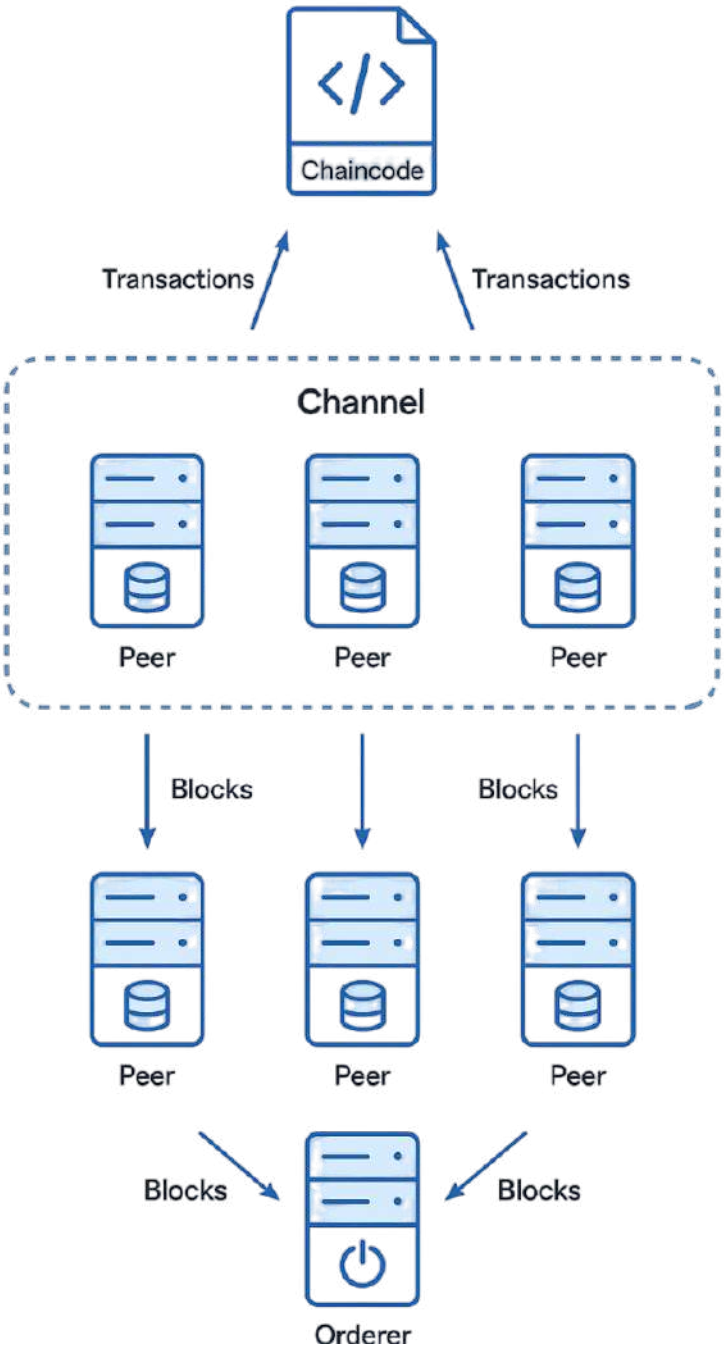

Fig. 2.3 Hyperledger Fabric Key Components.

2.3.2 World State Database

In the Hyperledger Fabric architecture, the ledger consists of two main parts, the blockchain and the world state. The blockchain serves as an immutable historical record of all transactions that occur in the network, stored in the form of cryptographically interconnected blocks. Meanwhile, the world state is a representation of the current status of the assets or data stored in the system. When a transaction is successfully committed, the input or changes that occur will be reflected in the world state, so users do not need to read the entire blockchain to find out the current value of an asset.

One of CouchDB features is Mango Query, a built-in query engine that enables JSON filter-based searches. Mango Query can be optimized by creating custom indexes (Mango indexes) for specific fields, which can improve query performance. However, the performance of Mango Query is highly dependent on the availability of indexes that match the query pattern. Without an index, the search can be slow, especially when the amount of data is large, as the system will perform a full scan of all [21] documents.

In the context of this research, CouchDB is used to store supply chain inventory data and perform various data search functions based on the fields in each asset document. The CouchDB-based world state feature is an important foundation for the implementation of flexible Mango Query strategies and can be dynamically combined with deterministic approaches such as Composite Key to produce optimal data retrieval system performance.

2.3.3 Mango Query

Mango Query is a document-based data search feature powered by CouchDB, the state database of choice in Hyperledger Fabric. Mango Query allows user to make queries in the form of JSON selectors, which can filter documents based on attribute values. This feature has flexibility as it can be used to query for data based on one or more attributes, supporting operators such as $eq, $gte, $lte, $and, $or, and so on. This makes Mango Query very suitable for complex data search scenarios, such as in supply chain management systems that involve many variables[22] [23].

Hyperledger Fabric supports full integration between chaincode and Mango Query. In practice, JSON selector queries can be written directly inside chaincode functions, and run against CouchDB to retrieve relevant assets. With this capability, developers can build multi-attribute search functions dynamically and efficiently.

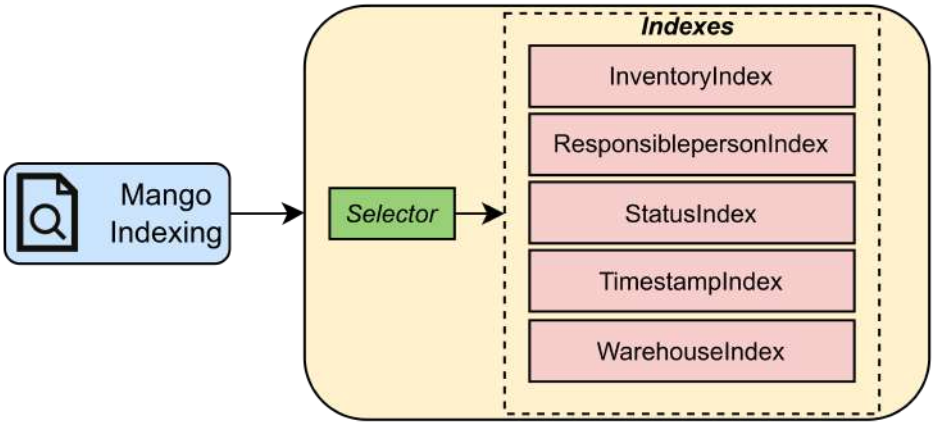

Fig. 2.4 Mango Query Strategy.

2.3.4 Composite Key

Composite Key is a mechanism to form a multi-part key from multiple attributes in an entity. Composite keys allow developers to combine two or more values into one deterministic key, which can be used to efficiently store and retrieve data from the ledger. The main advantage of Composite Key lies in its efficiency in one-dimensional or consistent pattern-based data searches, such as searches based on ID prefixes or specific entity categories. Since Hyperledger Fabric stores the ledger in the form of a key-value structure, searching with Composite Key utilizes lexicographic sorting of keys, so the system can retrieve all data that has the same prefix without having to scan the entire ledger.

However, Composite Key has limitations in flexibility. This approach does not support condition-based searches or combinations of fields that are not in key order[7]. For example, a search based on a combination of warehouse and status cannot be performed if only the warehouse is in the Composite Key prefix.

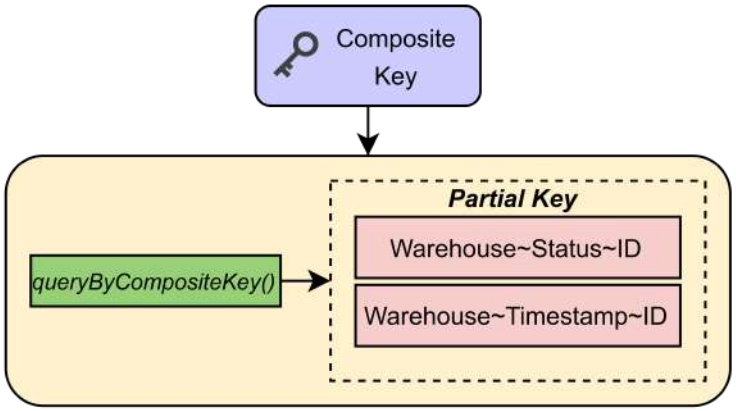

Fig. 2.5 Composite Key Strategy.

CHAPTER 3
THE PROPOSED CODES

This chapter discusses the system and the construction of the proposed combination of Mango Query and Composite key. Research design and research parameter is described in this chapter. Furthermore, all scenarios are desrcibed in this chapter.

3.1 Preliminary Research

Before designing the main experiments in this research, a series of preliminary studies were conducted to evaluate the technical feasibility and initial effectiveness of the combined approach between Mango Query and Composite Key in data retrieval on the Hyperledger Fabric. This research started with testing the Mango Query-based search function using fields such as warehouse, status, and timestamp. The test shows that Mango Query performance is strongly influenced by the structure and availability of Mango indexes. This finding is in line with the study of Zheng et al, who stated that search efficiency in CouchDB databases is highly dependent on index design and the number of fields involved in the query selector [21]. Without explicit index settings, query execution time increases significantly as data increases, especially when 5,000 entries are entered. Furthermore, the Composite Key approach was tested for prefix-based deterministic search scenarios such as rm, wh, and rs, corresponding to the asset ID structure used in the system. Preliminary test results show that this method is highly efficient for single-key attribute-based searches, as it leverages Fabric GetStateByPartialCompositeKey function which internally optimizes searches based on key prefixes. This is reinforced by the findings of Yan et al. (2022), who showed that composite key provides significant performance improvement for queries with one-dimensional deterministic structure [7]. However, this method does not support multi-attribute search or complex conditions, making it less flexible for advanced search scenarios.

From this exploration, a prototype input condition-based query strategy selection logic, where the system will select the use of Mango Query for single-field searches and Composite Key for deterministic multi-field searches. This approach aims to combine the advantages of each method, with the hope of improving the efficiency of the data retrieval system in the blockchain without sacrificing query flexibility.

3.2 System Model

The system proposed in this research, as shown in Fig 3.1, is designed to optimize the efficiency of data retrieval on permissioned blockchain networks using Hyperledger Fabric. The main focus of the system is on attribute-based inventory data retrieval, which is commonly used in supply chain management. The system utilizes the CouchDB-based world state to support flexible queries through Mango Query, and Composite Key strategy for key prefix-based deterministic search.

The data is stored as JSON documents with important attributes such as idPrefix, warehouse, status, responsiblePerson, and timestamp. Two main approaches are used for data retrieval, namely Mango Query and Composite Key, which are then combined in one adaptive strategy. The query method selection logic is built into the smart contract (chaincode), which automatically determines the execution path based on the number and type of parameters provided in the query request.

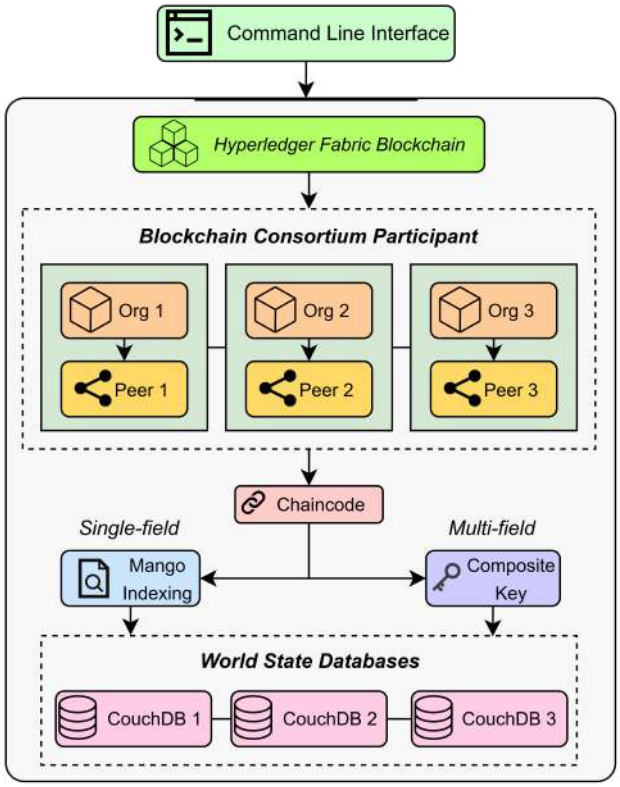

Fig. 3.1 Mango-Composite Architecture.

The system model was evaluated through a series of experiments involving single-field and multi-field search scenarios. Each approach is tested on the basis of execution time, CPU consumption, memory, and the number of data results obtained. With this approach, the system model is expected to demonstrate the superiority of the combination strategy over the use of a single query method, espe-

cially in large-scale and complex data scenarios. The architecture can be abstracted as a function $S(Q)$ that maps a conditional query Q to the appropriate query strategy. The decision is based on the number of attributes involved in the query. The system behaves as follows:

$$S(Q) = \begin{cases} \text{MangoQuery}(Q), & \text{if } |Q| = 1 \\ \text{CompositeKey}(Q), & \text{if } |Q| \geq 2 \end{cases}$$

3.2.1 Supply Chain integration in Hyperledger Fabric Topology

To represent the SME-based supply chain model more realistically, this study proposes a blockchain-based system architecture that integrates three main organizations, namely Procurement, Warehouse, and Retail & Sales, as illustrated in Fig 3.2. Each organization has a role in the distribution of goods and data, and has its own ledger and peers connected through the Hyperledger Fabric network.

In this topology, Organization 1 (Procurement) is responsible for procuring materials or products from suppliers or factories. This entity records information related to purchases, delivery schedules, and transaction documents. Next, Organization 2 (Warehouse) receives goods from Procurement and records data related to storage, condition status, and logistics movements within the warehouse. Finally, Organization 3 (Retail & Sales) is responsible for recording distribution activities to outlets or end customers, including transaction details and delivery status.

This topological approach is adopted to support realistic SME scenarios, which in practice have limited resources but still require a secure recording system.

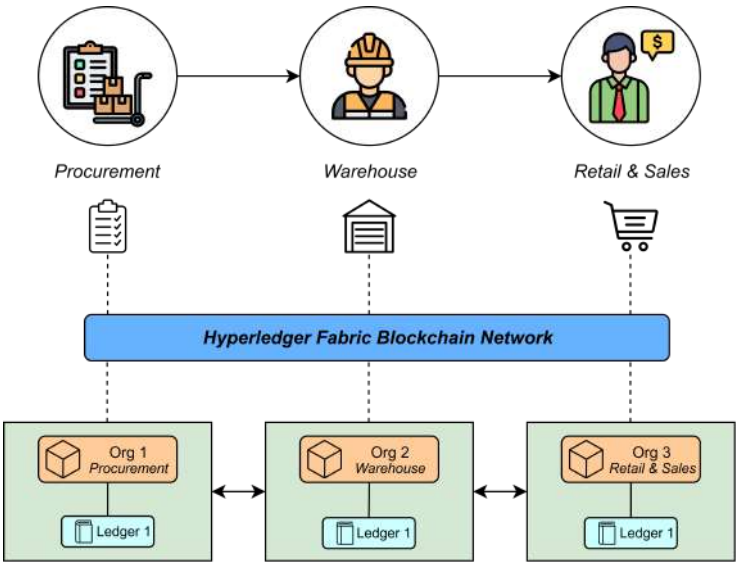

Fig. 3.2 Supply Chain Topology.

3.3 Experimental Setup

The experiments in this research were carried out in a controlled local environment using a virtualized Hyperledger Fabric network. The environment was deployed on an Ubuntu 20.04 LTS-based virtual machine managed by Vagrant and VirtualBox, with allocated hardware specifications of 2 vCPUs and 6 GB RAM. All services, including orderers, peers, and CouchDB instances, were run on a single VM to ensure consistent performance measurement without external network interference.

Hyperledger Fabric version 2.5 was used as the blockchain platform, configured with CouchDB as the state database to enable rich queries through Mango selectors. Chaincode was developed in Go (Golang), integrating both Mango Query and Composite Key strategies within the query logic. Query execution and benchmarking were performed using the Fabric CLI (`peer chaincode invoke` and `peer chaincode query`) in combination with the `/usr/bin/time -v` tool to capture CPU usage, memory consumption, and execution time for each query.

Table 3.1 System Configuration Parameters

Parameters	**Value**
Virtualization Platform	Vagrant + VirtualBox
Operating System	Ubuntu 20.04 LTS
vCPU	2 Cores
vRAM	6 GB
Blockchain Platform	Hyperledger Fabric v2.5
State Database	CouchDB (3 Instances)
Chaincode Language	Go (Golang)
Testing Method	Fabric CLI + `/usr/bin/time -v`

3.3.1 Experimental Scenario and Limitations

To evaluate query performance under varying operational loads, experiments were conducted using datasets of 10,000 and 50,000 inventory records. Four query strategies were tested:

1. **HLF Baseline** — native CouchDB Mango Query without optimization.

2. **Mango Query Only** — single-field and multi-field queries using explicit Mango selectors.

3. **Composite Key Only** — key-based lookups using predefined composite key structures.

4. **Mango-Composite (Proposed)** — switching mechanism between Mango Query (single-field) and Composite Key (multi-field).

Each strategy was executed across multiple concurrency levels: single-user, and multi-user scenarios with 3, 5, and 7 concurrent clients. Queries were divided into single-field lookups (e.g., by ID prefix, warehouse, status, responsible person) and multi-field lookups (e.g., warehouse + status, warehouse + timestamp). Each configuration was repeated three times, and average results were taken to reduce measurement variance.

Limitations of the Experimental Environment:

- All components were deployed on a single virtual machine, which does not fully represent a distributed production environment.
- Limited hardware resources (2 vCPU, 6 GB RAM) may impact scalability results when extrapolated to larger systems.
- No real network latency or inter-organizational delays were introduced, as all nodes run locally.
- Maximum dataset size used is 10,000 and 50,000 records, which may reflect on SMEs, not performance for millions of records.
- This experiment focuses on read/query performance.

This scenario and limitation description ensures that the results are interpreted within the context of the controlled environment and highlights potential differences if deployed in a real-world multi-node blockchain network.

3.4 Data Collection Mechanism

The simulation scenario in this research is designed to evaluate the performance of data retrieval in the Hyperledger Fabric using four search approaches, namely: Baseline Query (manual query using JSON selector), Mango Query Only, Composite Key Only, and Mango-Composite Key Strategy combination. These approaches

are used to test the effectiveness of data query strategies against various variations of query structures and large data scales in the context of supply chain systems.

The testing is performed on a dummy dataset of 10,000 and 50,000 supply chain entries by executing each query type defined in Table 3.2. Each entry has a JSON structure with ID, Warehouse, Status, ResponsiblePerson, and Timestamp attributes. Six types of query scenarios were tested, which were divided into two main categories: single-field query and multi-field query. For the single-field category, queries were performed based on idPrefix, warehouse, status, and responsiblePerson. As for multi-field, the system tested the combination of warehouse + status and warehouse + timestamp fields.

Table 3.2 Query Types and Method

Query Types	Method
`QueryByIDPrefix`	Mango Query + Index
`QueryByWarehouse`	Mango Query + Index
`QueryByStatus`	Mango Query + Index
`QueryByResponsiblePerson`	Mango Query + Index
`QueryByWarehouseAndStatus`	Composite Key
`QueryByWarehouseAndTimeStamp`	Composite Key

Each approach was tested under two user conditions: (1) single-user environment, where only one organization (Org1) queries the ledger, and (2) multi-user environment, where three organizations (Org1, Org2, and Org3) query the same channel in parallel. The purpose of these variations is to evaluate the stability and efficiency of query strategies in a collaborative context that resembles a real-world permissioned blockchain network. Each scenario is measured based on query execution time, CPU usage, memory consumption, as well as the number of results returned by the system.

3.5 Benchmark Scenario

The benchmark in this study is designed to evaluate the performance of data retrieval on the Hyperledger Fabric network by comparing four query strategy approaches as described in the data collection mechanism section. Each approach is tested under the same conditions, with a dataset of 10,000 and 50,000 entries. The

tests were conducted using a quantitative experimental approach, where each query performance metrics was measured objectively and holistically.

Each type of query, be it single-field or multi-field, was executed 10 times to obtain a stable average value. To get the system performance, the time -v command in Linux environment is used to record the execution time (second), CPU usage during the process (percent CPU usage), and memory consumption (maximum resident set size). These parameters were chosen because they reflect the system performance from various aspects, such as time efficiency, computational load, and resource usage effectiveness. Fig.3.3 illustrates the end-to-end benchmark workflow.

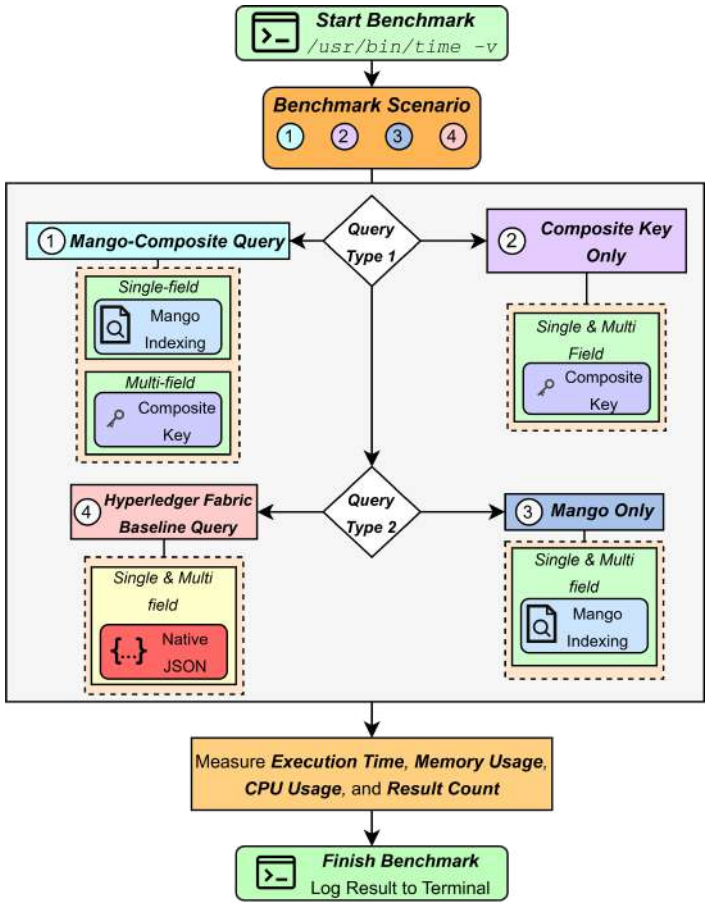

Fig. 3.3 Experimental Benchmarking Architecture

In addition to performance parameters, this test also records the number of search results or the amount of data successfully retrieved as an indicator of query success. This is important to ensure that each search method is not only efficient, but also accurate and consistent in producing data that match the desired criteria. The tests were conducted in two scenarios, which are single-user environment and multi-user environment as mentioned in the previous subsection. This aims to evaluate the stability and scalability of each strategy under different load conditions.

CHAPTER 4
PERFORMANCE EVALUATIONS

This chapter presents the results of simulation and benchmarking experiments conducted to evaluate the performance of four query strategies implemented in Hyperledger Fabric, such as Baseline Query, Mango Query Only, Composite Key Only, and the Combined Mango-Composite Strategy. The evaluation is carried out under two operational scenarios, namely single-user and multi-user environments, with each query strategy assessed using four performance metrics, namely execution time, CPU usage, memory consumption, and number of records retrieved. The tested queries are categorized into single-field and multi-field types to reflect different levels of query complexity commonly encountered in supply chain systems. Through this analysis, the research aims to determine the most efficient and scalable query strategy for blockchain-based inventory systems using CouchDB.

4.1 Overview of the Experiment

The experiments in this study are designed to evaluate the performance of four approaches to data retrieval strategies in the Hyperledger Fabric network, namely: Baseline Query, Mango Query Only, Composite Key Only, and Mango-Composite Strategy Combination. The main objective of this experiment is to measure the extent to which each approach can handle data retrieval in a blockchain-based supply chain management system scenario, both in terms of time efficiency, resource usage, and consistency of search results.

Each approach was tested against two main query categories, such as single-field queries and multi-field queries. Single-field queries include searches based on single attributes such as idPrefix, warehouse, status, and responsiblePerson, while multi-field queries combine two attributes, namely warehouse + status and warehouse + timestamp. The data used in the test consists of 10,000 and 50,000 inventory asset entries stored in a ledger with a uniform JSON structure.

The experiments were conducted in two user environments: a single-user environment, where only one of the three organizations queried, and a multi-user environment, where three organizations (Org1, Org2, and Org3) queried simultaneously.

4.2 10,000 Data Experiment: Single-user Environment

The first test was conducted in a single-user environment scenario, where a user executes 10,000 data lookup requests independently against the Hyperledger Fabric single organization. The purpose of this test is to measure the performance of each query strategy under conditions without resource contention, so that the results reflect the basic efficiency of each approach.

Six types of queries were tested, namely four single-field queries (idPrefix, warehouse, status, and responsiblePerson) and two multi-field queries (warehouse + status and warehouse + timestamp).

Table 4.1 Query Performance Comparison in Single-User Environment using 10,000 Data Entries

Query Strategy	**Query Type**	**10,000 Records**		
		Time (s)	**CPU (%)**	**Memory (MB)**
HLF Baseline	IDPrefix	2.91 s	3.67 %	28.248 MB
	Warehouse	4.30 s	2.67 %	25.396 MB
	Status	4.12 s	2.33 %	25.503 MB
	ResPerson	3.75 s	2.33 %	25.687 MB
	Ware + Stat	6.23 s	2.00 %	29.908 MB
	Ware + Time	8.04 s	2.00 %	25.737 MB
Mango Query	IDPrefix	2.90 s	3.33 %	28.654 MB
	Warehouse	4.08 s	2.33 %	25.684 MB
	Status	4.05 s	2.00 %	28.127 MB
	ResPerson	3.73 s	2.00 %	25.615 MB
	Ware + Stat	0.49 s	36.00 %	25.513 MB
	Ware + Time	0.20 s	68.00 %	26.594 MB
Composite Key	IDPrefix	1.25 s	8.33 %	28.895 MB
	Warehouse	1.31 s	10.67 %	27.403 MB
	Status	1.20 s	9.00 %	28.248 MB
	ResPerson	0.24 s	37.67 %	25.549 MB
	Ware + Stat	0.33 s	38.00 %	25.463 MB
	Ware + Time	0.44 s	45.33 %	25.635 MB
Mango-Composite	IDPrefix	2.87 s	3.00 %	28.824 MB
	Warehouse	3.92 s	2.33 %	25.556 MB
	Status	4.06 s	2.33 %	28.107 MB
	ResPerson	3.60 s	2.33 %	25.588 MB
	Ware + Stat	0.36 s	34.33 %	25.489 MB
	Ware + Time	0.25 s	51.00 %	25.359 MB

4.2.1 Single-user 10,000 Data Entries: Execution Time Result and Analysis

Table 4.1 and Figure 4.1 presents the average execution time of each data retrieval strategy in a single-user environment with 10,000 data entries. The data is taken for six query functions with varying levels of complexity, both single-field and multi-field. The results show that the Composite Key Only approach consistently provides the fastest execution time for deterministic queries such as *QueryByIDPrefix*, *QueryByWarehouse*, and *QueryByStatus*, with durations below 1.1 seconds. This is due to the efficiency of the *GetStateByPartialCompositeKey* function in performing key prefix-based searches.

In contrast, the Baseline and Mango Query Only approaches show higher execution times, especially for single-field queries that are not index-optimized. For multi-field queries such as QueryByWarehouseAndStatus and QueryByWarehouseAndTimestamp, the Mango Query Only and Mango-Composite Strategy approaches gave the best results, with average times below 0.5 seconds. The combination approach showed balanced adaptive performance, approaching Composite Key on simple queries, and resembling Mango Query on complex queries. This indicates that the combination strategy is able to maintain time efficiency while retaining flexibility.

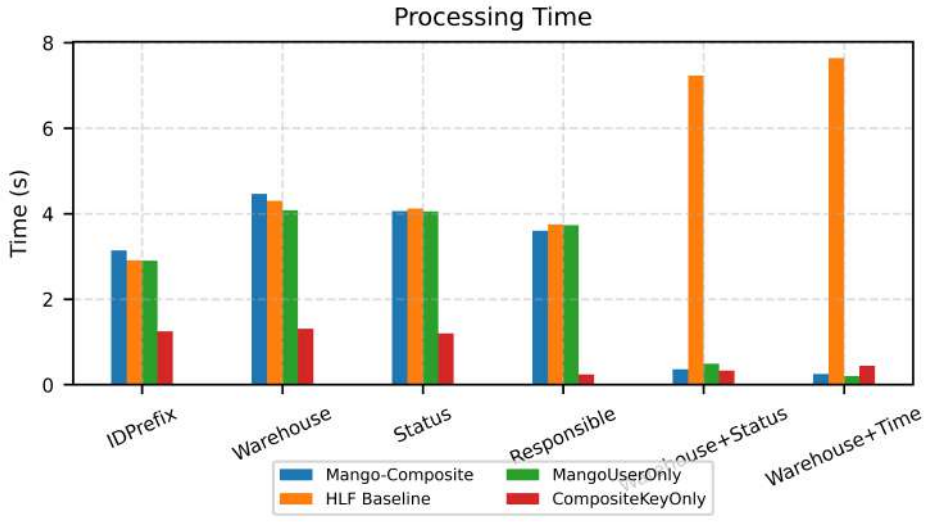

Fig. 4.1 Performance comparison of different query strategies on memory usage for single user — Mango-Composite, Baseline, MangoUserOnly, and CompositeKeyOnly.

4.2.2 Single-user 10,000 Data Entries: CPU Usage Result and Analysis

CPU utilization is an important indicator to assess the computational load required by each data retrieval strategy. This metric describes the average percentage of processor usage during the query execution process. In this single-user test, the CPU value is recorded using the *time -v* utility and shows how efficient a particular strategy is in utilizing computational resources, especially when accessing large volumes of data.

The results in Table 4.1 and Figure 4.2 show that the Composite Key Only approach tends to have a higher CPU consumption than the other approaches, despite its very short execution time. This suggests that this approach relies on computational performance to generate speed. In contrast, Baseline and Mango Query Only resulted in lower CPU usage, but with longer execution times. The Mango-Composite Strategy approach exhibits intermediate characteristics, with relatively efficient CPU usage, especially for multi-field queries such as *QueryByWarehouse-AndTimestamp*.

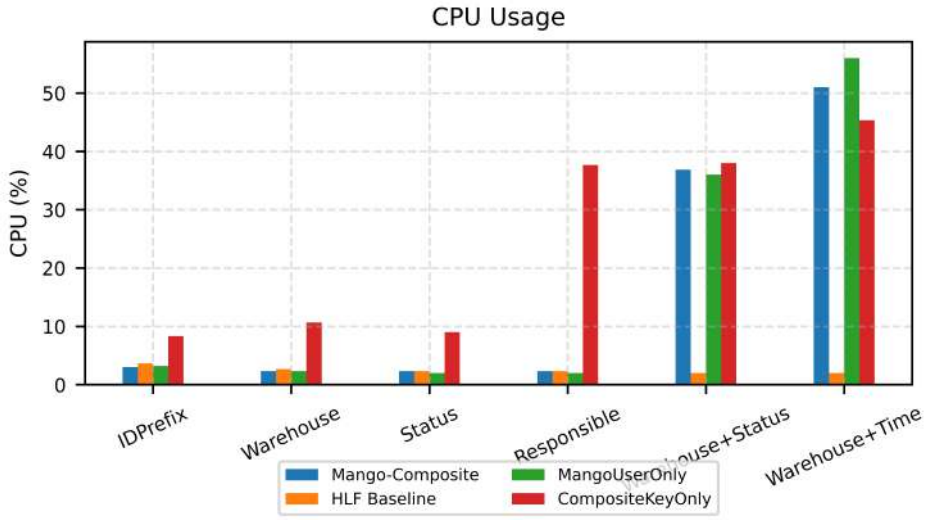

Fig. 4.2 Performance comparison of different query strategies on CPU usage for single user — Mango-Composite, Baseline, MangoUserOnly, and CompositeKeyOnly.

4.2.3 Single-user 10,000 Data Entries: Memory Usage Result and Analysis

Memory consumption measures the memory (in megabytes) that the system uses during the query execution process. This measurement gives an idea of how much memory space each strategy requires, which is important for considering resource efficiency, especially on RAM-constrained systems. This data is captured using the Maximum resident set size metric generated by the *time -v* command.

As shown in Table 4.1 and Figure 4.3, all approaches have relatively similar memory consumption with small fluctuations. The Composite Key Only approach tends to have the highest memory consumption in some functions, such as *QueryByIDPrefix* and *QueryByWarehouse*, but the difference is not significant overall. Mango-Composite Strategy and Mango Query Only are in about the same memory range, while Baseline has the most stable memory values, but is not always efficient in terms of time or CPU performance. These differences show that all strategies remain within acceptable memory efficiency limits for the scale of data used.

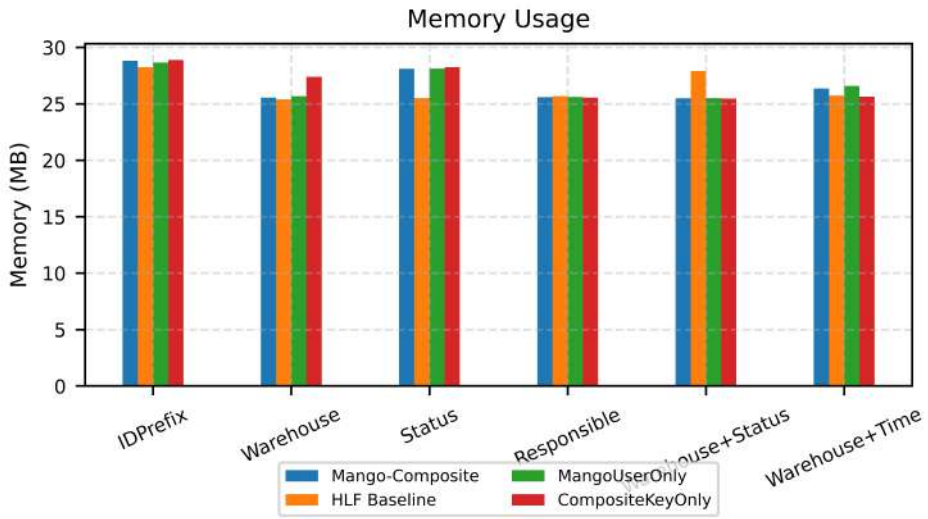

Fig. 4.3 Performance comparison of different query strategies on execution time for single user — Mango-Composite, Baseline, MangoUserOnly, and CompositeKeyOnly.

4.2.4 Single-user 10,000 Data Entries Experimental Result and Analysis

consistently outperformed other methods in query execution time in single-field and multi-field scenarios, as detailed in Table 4.2 and illustrated in Fig. 4.3, Fig.4.2, and Fig.4.1, which visualizes the comparative performance across all query strategies. For single-field queries, it completed the search in an average of 2.87 seconds, slightly faster than the *HLF Baseline* (2.91s) and *Mango Query Only* (2.90s), and reasonably close to the fastest and heavy method, *Composite Key Only* (1.25s). In multi-field queries with minimal data matches, *Mango-Composite* demonstrated a remarkable advantage, completing in just 0.24 seconds compared to *HLF Baseline* (8.04s), *Mango Query Only* (0.2s), and *Composite Key Only* (0.44s).

However, the strategy is not without trade-offs. In certain cases, particularly multi-field queries, it recorded noticeably higher CPU usage than the *HLF Baseline* and *Composite Key Only* approaches. For instance, the multi-field warehouse+timestamp query peaked at 51% CPU, surpassing *HLF Baseline* (2%) and even *Composite Key Only* (45.33%). This increase comes from managing conditional routing and maintaining index and key-based paths within the smart contract.

Nonetheless, this increase in resources is justified by the resulting increase in performance. *Mango-Composite* can route simple queries through the already indexed Mango selector and complex queries through Composite Key, thus avoiding costly full scans. Unlike *Mango Query Only*, which incurs higher CPU due to complex JSON evaluation, or *Composite Key Only*, which demands additional iterations. *Mango-Composite* offers a balanced trade-off between speed and resource efficiency that makes it well suited for permissioned blockchain environments with diverse data patterns, such as supply chains.

Table 4.2 Single-user Mango-Composite query performance by field type, including CPU, time, and memory metrics using 10,000 data entries.

Data Count	Field Type	Avg. CPU	Avg. Time	Avg. Memory
2790	*Single-field*	3%	3,87 s	28,824 MB
2091		2.33%	4.06 s	28.107 MB
569		2.33%	3.92 s	25.556 MB
287		2.33%	3.6 s	25.588 MB
267	*Multi-field*	34.33%	0.36 s	25.489 MB
1		51%	0.25 s	25.359 MB

4.3 50,000 Data Experiment: Single-user Environment

The second test was still conducted in a single-user environment scenario, where a user independently executed 50,000 data search requests against a single Hyperledger Fabric organization. The purpose of this test was to measure the performance of each query strategy in conditions without resource competition, so that the results reflect the use of large amounts of data from each approach.

Table 4.3 Query Performance Comparison in Single-User Environment using 50,000 Data Entries

Query Strategy	**Query Type**	**50,000 Records**		
		Time (s)	**CPU (%)**	**Memory (MB)**
HLF Baseline	IDPrefix	29.37 s	63.23 %	41.680 MB
	Warehouse	8.05 s	69.43 %	29.365 MB
	Status	8.77 s	58 %	28.283 MB
	ResPerson	7.63 s	68.8 %	29.414 MB
	Ware + Stat	2.53 s	42.37 %	27.140 MB
	Ware + Time	5.87 s	47.77 %	26.773 MB
Mango Query	IDPrefix	27.36 s	51.43 %	39.438 MB
	Warehouse	7.17 s	63.43 %	32.019 MB
	Status	8.10 s	51.73 %	32.245 MB
	ResPerson	7.06 s	52.67 %	28.443 MB
	Ware + Stat	1.13 s	77.97 %	27.276 MB
	Ware + Time	0.14 s	88.43 %	29.819 MB
Composite Key	IDPrefix	0.13 s	88.03 %	28.896 MB
	Warehouse	0.12 s	89.83 %	26.580 MB
	Status	0.17 s	86.13 %	26.533 MB
	ResPerson	0.11 s	93.67 %	28.380 MB
	Ware + Stat	0.13 s	85 %	28.191 MB
	Ware + Time	0.14 s	85.07 %	26.785 MB
Mango-Composite	IDPrefix	25.68 s	25.68 %	41.327 MB
	Warehouse	6.51 s	60.37 %	32.019 MB
	Status	8.30 s	39.33 %	38.651 MB
	ResPerson	6.75 s	52.77 %	29.48 MB
	Ware + Stat	0.12 s	84.03 %	34.344 MB
	Ware + Time	0.11 s	84.97 %	31.463 MB

4.3.1 Single-user 50,000 Data Entries: Execution Time Result and Analysis

Table 4.3 and Figure 4.4 presents the average execution time of each data retrieval strategy in a single-user environment with 50,000 data entries. The data is taken for six query functions with varying levels of complexity, both single-field and multi-field query. The results show that the Composite Key Only approach consistently provides the fastest execution time for deterministic queries such as *QueryByIDPrefix*, *QueryByWarehouse*, and *QueryByStatus*, with durations below 1 second. This is due to the efficiency of the *GetStateByPartialCompositeKey* function in performing key prefix-based searches.

On the other hand, the proposed system has balanced results even though the data volume has been increased to a total of 50,000 data entries. First, in single-column queries, Mango-Composite has a slightly shorter processing time compared to mango user only. This is due to the use of rich queries, which provide a slight advantage over pure mango queries, where the mechanism does not rely on doctype but instead directly indexes the dataset. Next, in multi-column queries, the proposed system performs slightly better than composite key only. Although not significant, it is sufficient to match the results of the comparison in terms of processing time.

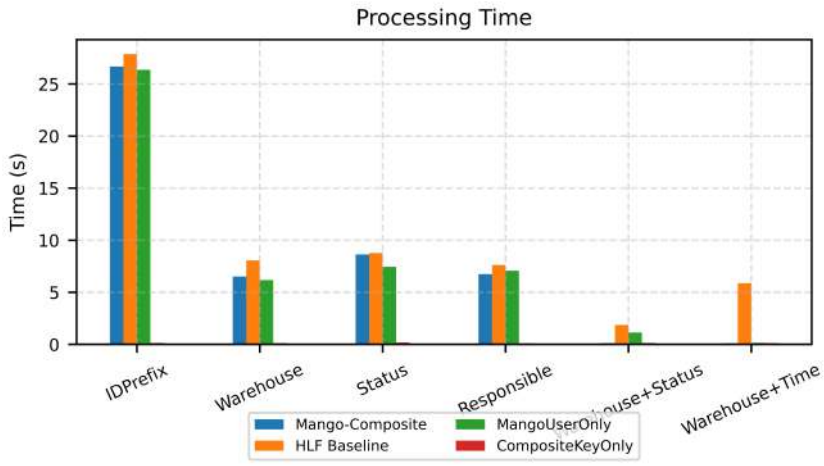

Fig. 4.4 Performance comparison of different query strategies on execution time for single user — Mango-Composite, Baseline, MangoUserOnly, and CompositeKeyOnly.

4.3.2 Single-user 50,000 Data Entries: CPU Usage Result and Analysis

The results and analysis of CPU usage in a single-user environment with 50,000 data points show significant differences between the four query strategies. The Composite Key Only approach consistently uses the highest CPU resources across all query types, ranging from 85% to 93.67%. This is understandable because this approach relies heavily on full key scans and direct iterations on the ledger state, without the assistance of CouchDB indexing.

In contrast, the Mango-Composite strategy demonstrated better efficiency, with CPU usage ranging from 39.33% to 84.97%. This reflects the effectiveness of adaptive selection between Mango Query and Composite Key based on the structure of the query being executed. For single-field queries such as Status and Responsible, the MangoUserOnly approach tends to use slightly higher CPU than Mango-Composite, reaching up to 63.43%, likely because the selector used is less optimal and involves a broader index scan process in CouchDB.

These results indicate that Composite Key can provide speed, but this comes at the cost of high CPU consumption. On the other hand, Mango-Composite achieves a balance between performance and resource efficiency by leveraging the advantages of both approaches.

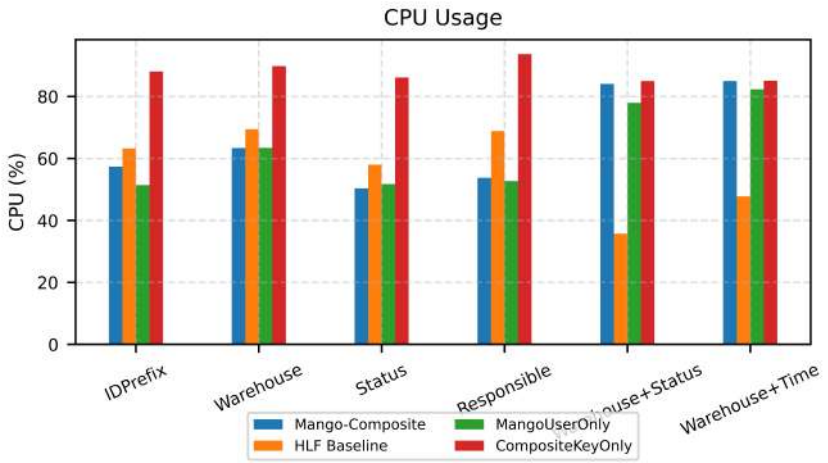

Fig. 4.5 Performance comparison of different query strategies on execution time for single user — Mango-Composite, Baseline, MangoUserOnly, and CompositeKeyOnly.

4.3.3 Single-user 50,000 Data Entries: Memory Usage Result and Analysis

Memory usage in testing 50,000 data points showed a rather interesting pattern. The Mango-Composite strategy generally uses the highest amount of memory, particularly for single-field queries such as IDPrefix and Status, reaching 41.33 MB and 38.65 MB, respectively. This can be explained because the Mango-Composite strategy tends to perform selector evaluation and indexing simultaneously, especially when the combination of Mango and Composite strategies complements each other.

Meanwhile, the MangoUserOnly approach shows fairly stable memory usage, averaging around 30-39 MB. This strategy relies entirely on Mango Query, so the use of CouchDB indexes makes data processing efficient but still requires a large space for index, especially when processing large query results like IDPrefix.

Conversely, the Composite Key Only strategy is the most memory-efficient, with ranging from 26-28 MB across nearly all queries. This indicates that the key traversal-based approach directly accessing the ledger state, despite its high CPU usage, does not impose significant memory overhead since it does not involve JSON selector processing or index parsing.

HLF Baseline strategy produced mixed results, with memory usage that was not very consistent. For some queries like IDPrefix, memory consumption reaches 41.68 MB, but drops significantly for other queries like Status (26.14 MB) and Warehouse+Time (26.57 MB).

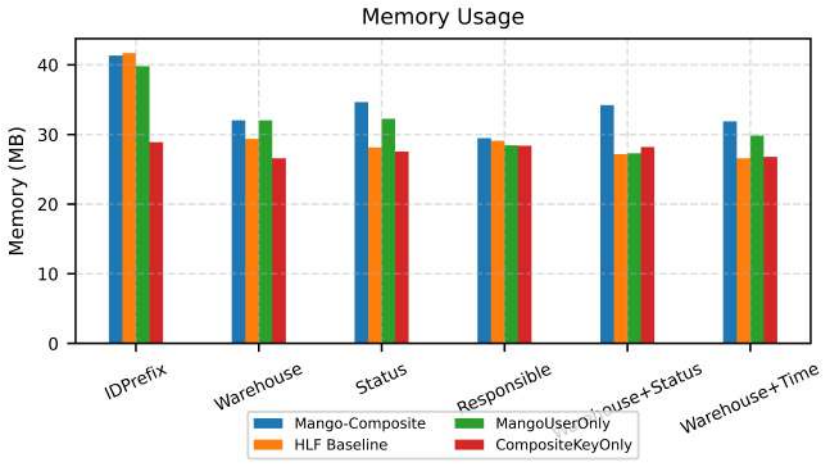

Fig. 4.6 Performance comparison of different query strategies on execution time for single user — Mango-Composite, Baseline, MangoUserOnly, and CompositeKeyOnly.

4.3.4 Single-user 50,000 Data Entries Experimental Result and Analysis

In single-user testing with 50,000 data entries, the *Mango-Composite* approach maintained its performance, particularly in query execution time. This approach outperforms the *HLF Baseline* and *Mango Query Only* in single-field queries, achieving faster results, such as 6.51 seconds for a data warehouse query compared to Mango-only (7.17 seconds) and Baseline (8.05 seconds). Although both Mango-based methods utilize indexing, the use of *doctype* in *Mango-Composite* effectively avoids the complexity of JSON, resulting in better processing time.

From the perspective of resource usage, memory consumption across all methods remained relatively moderate, with *Mango-Composite* maintaining stable usage between 26-41 MB. However, CPU usage showed a marked increase, particularly in complex multi-field queries like warehouse+timestamp, where it peaked at 84.97%. This suggests that while Mango-Composite achieves time efficiency, it trades off with a moderately higher processing load, yet still lower than Mango-only and Composite-only in most cases.

Overall, these results reinforce that *Mango-Composite* offers a scalable and balanced solution. It suits well even as the dataset size increases significantly, providing consistent performance benefits without severe overhead, making it suitable for real-world blockchain environments where query diversity and volume coexist.

Table 4.4 Single-user Mango-Composite query performance by field type, including CPU, time, and memory metrics using 50,000 data entries.

Data Count	Field Type	Avg. CPU	Avg. Time	Avg. Memory
14112	*Single-field*	43.33%	25.68 s	41.327 MB
2787		60.37%	6.51 s	32.019 MB
10609		39.33%	8.3 s	38.651 MB
326		52.77%	5.85 s	26.473 MB
267	*Multi-field*	84.03%	0.12 s	34.213 MB
1		94.97%	0.11s	31.859 MB

4.4 10,000 Data Experiment: Multi-user Environment

The third test was conducted in a multi-user environment scenario, where 3, 5, and 7 users execute data lookup requests independently against the Hyperledger Fabric network organizations (Org1, Org2, Org3). The purpose of this test is to measure the performance of each query strategy under conditions without resource contention, so that the results reflect the basic efficiency of each approach using 10,000 data entries.

Table 4.5 Query Performance Comparison in Multi-User Environment with 3 users testing and using 10,000 Data Entries

Query Strategy	Query Type	10,000 Records		
		Time (s)	CPU (%)	Memory (MB)
HLF Baseline	IDPrefix	4.48	2%	29.275
	Warehouse	4.43	2%	27.422
	Status	5.94	2%	28.310
	ResPerson	5.19	3%	27.592
	Ware + Stat	5.63	2%	28.567
	Ware + Time	5.94	2%	26.133
Mango Query	IDPrefix	3.01	3%	26.250
	Warehouse	4.07	2%	27.445
	Status	5.06	2%	28.992
	ResPerson	5.10	2%	27.210
	Ware + Stat	0.89	38%	27.596
	Ware + Time	0.21	69%	27.141
Composite Key	IDPrefix	1.09	15%	29.471
	Warehouse	1.02	12%	28.183
	Status	0.88	12%	28.057
	ResPerson	0.41	36%	27.218
	Ware + Stat	0.69	37%	28.155
	Ware + Time	0.15	71%	26.909
Mango-Composite	IDPrefix	2.93	3%	26.034
	Warehouse	4.06	3%	27.461
	Status	4.85	2%	29.194
	ResPerson	4.73	2%	27.762
	Ware + Stat	0.53	35%	27.389
	Ware + Time	0.15	72%	26.669

Table 4.6 Query Performance Comparison in Multi-User Environment with 5 users testing and using 10,000 Data Entries

Query Strategy	Query Type	10,000 Records		
		Time (s)	CPU (%)	Memory (MB)
HLF Baseline	IDPrefix	9.16	5.2%	32.916
	Warehouse	9.41	5.53%	29.558
	Status	12.8	7%	31.841
	ResPerson	14.61	7%	30.394
	Ware + Stat	14.6	9.4%	31.590
	Ware + Time	14.38	9.33%	28.993
Mango Query	IDPrefix	7.1	7.53%	28.491
	Warehouse	9.26	6.6%	30.385
	Status	10.8	8.27%	31.755
	ResPerson	12.72	8.2%	28.703
	Ware + Stat	1.5	42.53%	30.175
	Ware + Time	0.28	72%	29.047
Composite Key	IDPrefix	2.8	13.8%	29.546
	Warehouse	2.1	18.93%	28.928
	Status	1.74	20.87%	28.796
	ResPerson	1.13	40.6%	26.578
	Ware + Stat	1.29	42.28%	26.578
	Ware + Time	0.15	74.4%	28.246
Mango-Composite	IDPrefix	6.84	7.6%	28.337
	Warehouse	9.21	7.53%	30.842
	Status	10.6	10.4%	31.817
	ResPerson	11.3	8.2%	30.118
	Ware + Stat	0.66	35.4%	29.813
	Ware + Time	0.15	72.4%	30.178

Table 4.7 Query Performance Comparison in Multi-User Environment with 7 users testing and using 10,000 Data Entries

Query Strategy	Query Type	10,000 Records		
		Time (s)	CPU (%)	Memory (MB)
HLF Baseline	IDPrefix	13.60	27.2%	35.124
	Warehouse	17.12	28.4%	34.890
	Status	22.68	33.5%	31.164
	ResPerson	26.91	46.9%	31.426
	Ware + Stat	28.40	49.6%	33.714
	Ware + Time	27.31	53.6%	33.827
Mango Query	IDPrefix	9.52	31.3%	29.452
	Warehouse	15.33	33.3%	36.281
	Status	18.27	40.6%	36.883
	ResPerson	21.68	43.7%	35.641
	Ware + Stat	3.50	90.7%	35.941
	Ware + Time	3.68	96.1%	34.612
Composite Key	IDPrefix	3.70	61.8%	30.182
	Warehouse	3.10	68.3%	29.104
	Status	2.20	74.6%	29.582
	ResPerson	2.20	91.4%	28.621
	Ware + Stat	1.26	95.7%	30.179
	Ware + Time	1.10	99.4%	30.390
Mango-Composite	IDPrefix	8.13	33.8%	30.571
	Warehouse	14.07	36.1%	37.149
	Status	18.53	41.7%	32.478
	ResPerson	20.47	42.4%	33.438
	Ware + Stat	1.80	92.5%	30.518
	Ware + Time	1.30	97.2%	31.692

4.4.1 Multi-User 10,000 Data Entries: Processing Time Result and Analysis

Above in Table 4.5, Table 4.6, and Table 4.7 are results for processing time per query strategy in a multi-user environment for 3, 5, and 7 users. The execution time increases with the increased number of users thus increasing the processing time, meanwhile other factors like the complexity of queries or methods being used add up to it. For single-field queries such as *QueryByIDPrefix*, it was observed that the Composite Key Only method posted processing times of 1.09 seconds (for three users), 2.8 seconds (for five users), and 3.7 seconds (for seven users) being faster than Mango-Composite and Mango Query Only methods. On the other hand, HLF Baseline registered more method processing time compared with all others that registered up to a high of 13.6 seconds on seven user tests proving that direct access to ledger convention becomes inefficient under multiuser scenarios.

In multi-field queries such as *QueryByWarehouseAndStatus* and *QueryByWarehouseAndTime*, the Mango-Composite method shows improved efficiency. In a 7-user test, the execution time for *QueryByWarehouseAndTime* was only 1.3 seconds, significantly better than HLF Baseline (27.31 seconds) and even Composite Key Only (1.1 seconds), which is generally more optimal for conditional queries. This performance is consistent across all user counts, demonstrating the adaptive strategy ability to optimize execution paths for both simple and complex queries.

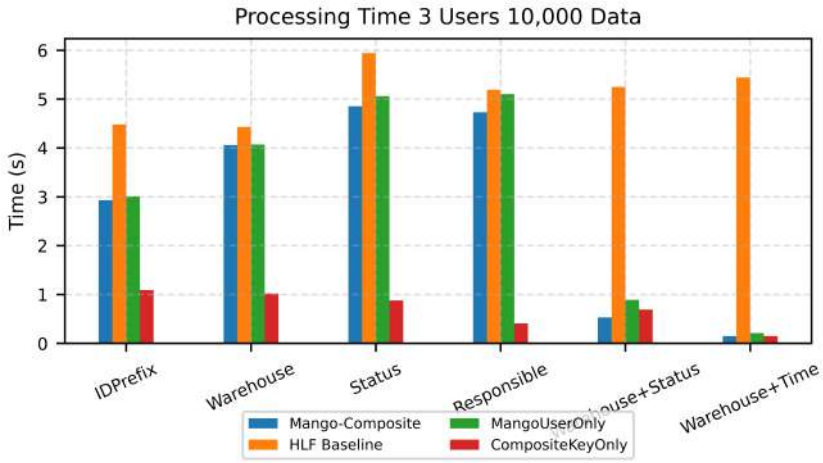

Fig. 4.7 Performance comparison of different query strategies on memory usage with 10,000 for 3 users — Mango-Composite, Baseline, MangoUserOnly, and CompositeKeyOnly.

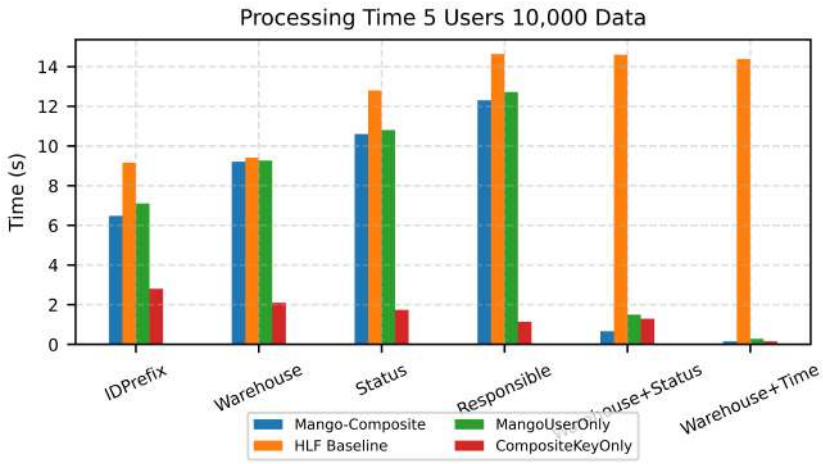

Fig. 4.8 Performance comparison of different query strategies on memory usage with 10,000 for 5 users — Mango-Composite, Baseline, MangoUserOnly, and CompositeKeyOnly.

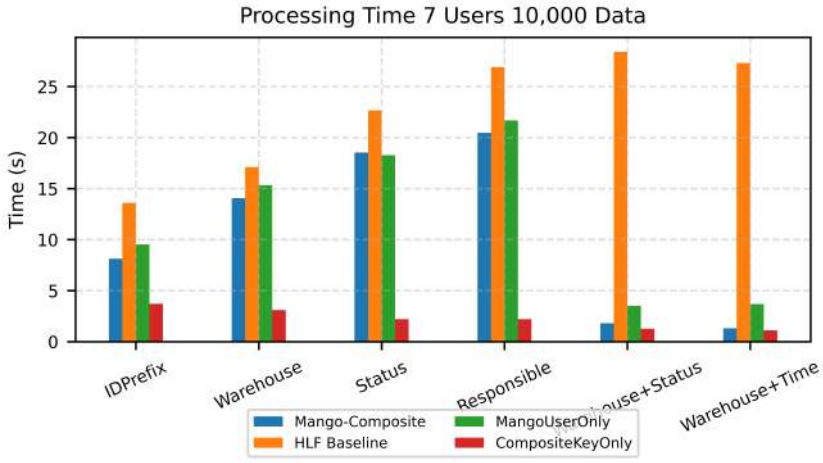

Fig. 4.9 Performance comparison of different query strategies on memory usage with 10,000 for 7 users — Mango-Composite, Baseline, MangoUserOnly, and CompositeKeyOnly.

This study concludes by stating that in a multi-user scenario with 10,000 data, the Mango-Composite strategy was able to maintain processing time, resulting in good performance. This strategy balances between being close to fast execution and allowing flexibility in chaincode architecture, thus making it an ideal for permissioned blockchain systems where dynamic query patterns exist like supply chains.

4.4.2 Multi-User 10,000 Data Entries: CPU Usage Result and Analysis

The CPU usage of each strategy in the multi-user scenario is shown in Table 4.5, Table 4.6, and Table 4.7. It can be seen that query complexity and the number of ac-

tive users have a direct impact on CPU consumption. Strategies involving JSON selector evaluation or composite key construction at runtime show a significant spike in CPU usage, especially as the number of users increases.

The *Composite Key Only* approach consistently shows the highest CPU usage in almost all scenarios. For example, for the *QueryByResponsiblePerson*, CPU usage rise from 36% (3 users) to 40.6% (5 users), then to 91.4% (7 users), which indicates a heavy computational load on key-based searches. Similarly, *Mango Query Only* experiences a significant CPU spike on conditional queries such as *QueryByWarehouseAndTime*, with usage reaching 72% at 5 users and 96.1% at 7 users.

On the other hand, the *Mango-Composite* query method shows better performance. Although it does not always have the lowest CPU usage, but this method has a moderate CPU load. For example, with the *QueryByWarehouseAndStatus* method, CPU usage remains below 36% with 5 users and rises to 92.5% with 7 users, still more efficient than the Mango query or Composite key approach in some query types. This shows that the architecture developed is successful in channeling queries to a more efficient execution path according to the context.

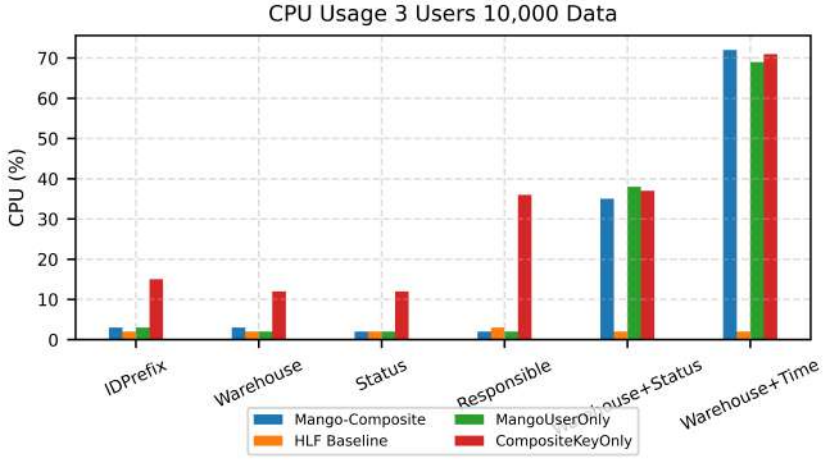

Fig. 4.10 Performance comparison of different query strategies on CPU usage with 10,000 for 3 users — Mango-Composite, Baseline, MangoUserOnly, and CompositeKeyOnly.

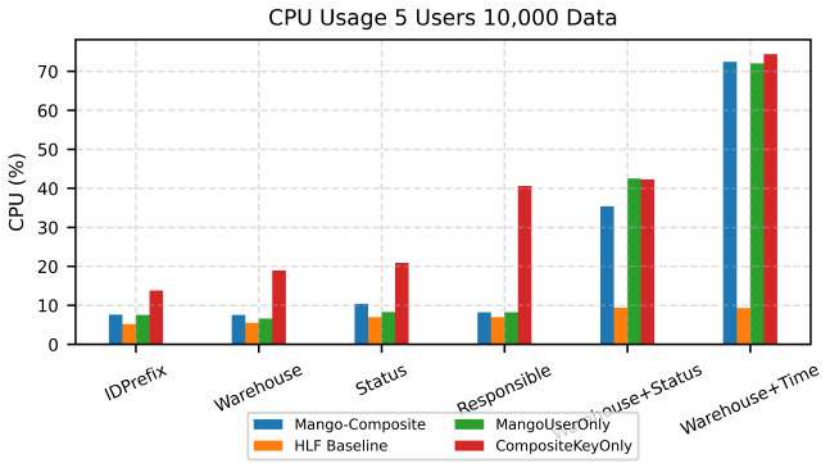

Fig. 4.11 Performance comparison of different query strategies on CPU usage with 10,000 for 5 users — Mango-Composite, Baseline, MangoUserOnly, and CompositeKeyOnly.

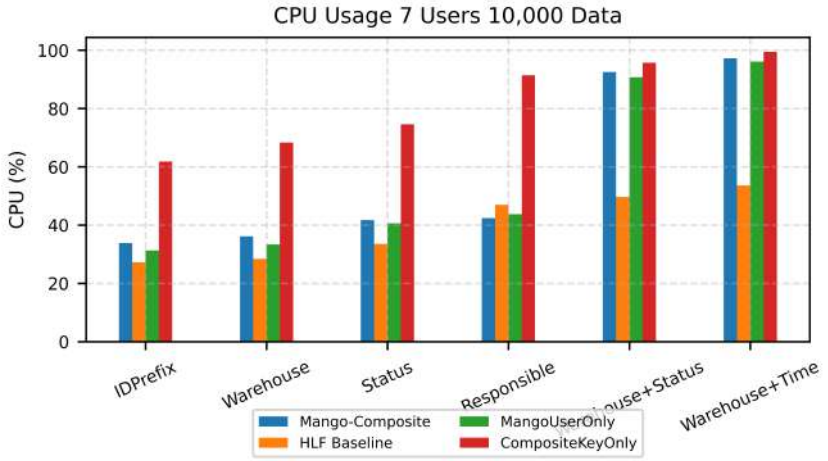

Fig. 4.12 Performance comparison of different query strategies on CPU usage with 10,000 for 7 users — Mango-Composite, Baseline, MangoUserOnly, and CompositeKeyOnly.

Thus, in terms of CPU usage, *Mango-Composite* shows a balance between performance and resource consumption. This strategy and method is able to adapt to multi-user loads and complex query types such as multi-field. This makes it a good and acceptable option in private blockchain systems such as digital supply chains.

4.4.3 Multi-User 10,000 Data Entries: Memory Usage Result and Analysis

Table 4.5, Table 4.6, and Table 4.7 provide data related to the memory consumption details of the four query strategies across various query types with different numbers of users. Memory usage tends to be more stable compared to CPU usage

test results, but still shows variations depending on the scenario used, query type and complexity, and the load of active users.

The *Composite Key Only* method consistently shows efficiency and low memory usage in almost all scenarios. For example, in the *QueryByWarehouse* function, memory consumption remains low from 28,183 MB (3 users) to 28,928 MB (5 users), and only 29,104 MB (7 users). This shows that the key-value-based method minimizes the load of complex data structures in the execution process, although on the other hand, it has the trade-off of considerable CPU usage.

In contrast, the *Mango Query Only* strategy experiences fluctuations in memory usage, especially in multi-field queries such as *WarehouseAndStatus* or *Warehouse-AndTime*. This increase in memory consumption illustrates the burden of parsing and evaluating JSON selectors in CouchDB, which tends to increase as the number of users increases.

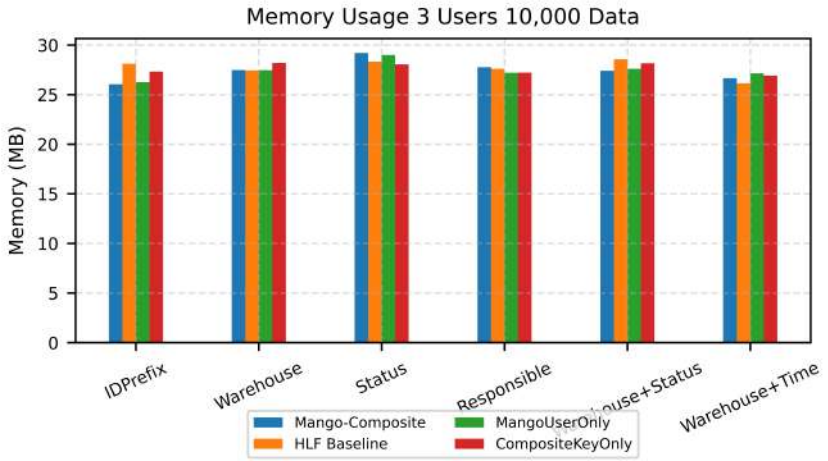

Fig. 4.13 Performance comparison of different query strategies on memory usage with 10,000 for 3 users — Mango-Composite, Baseline, MangoUserOnly, and CompositeKeyOnly.

The *Mango-Composite* method shows balance in testing. In the 7-user scenario, the average memory consumption remains within a reasonable and balanced range, such as 30,571 MB for *QueryByIDPrefix* and 31,692 MB for *QueryByWarehouse-AndTime*, nearly matching the efficiency of the Composite Key method but with greater flexibility. This shows that despite having two execution paths (Mango + Composite), this strategy is able to keep memory stable with query path retrieval in this 10,000 data test.

Thus, from a memory usage perspective, *Mango-Composite* is able to maintain memory usage stability while remaining flexible to query type variations and increases in the number of active users, making it a viable option for small to medium-scale permissioned blockchain systems.

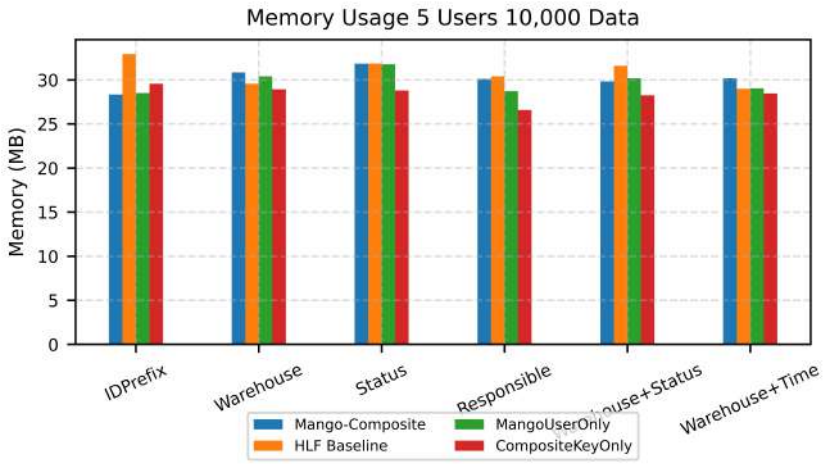

Fig. 4.14 Performance comparison of different query strategies on memory usage with 10,000 for 5 users — Mango-Composite, Baseline, MangoUserOnly, and CompositeKeyOnly.

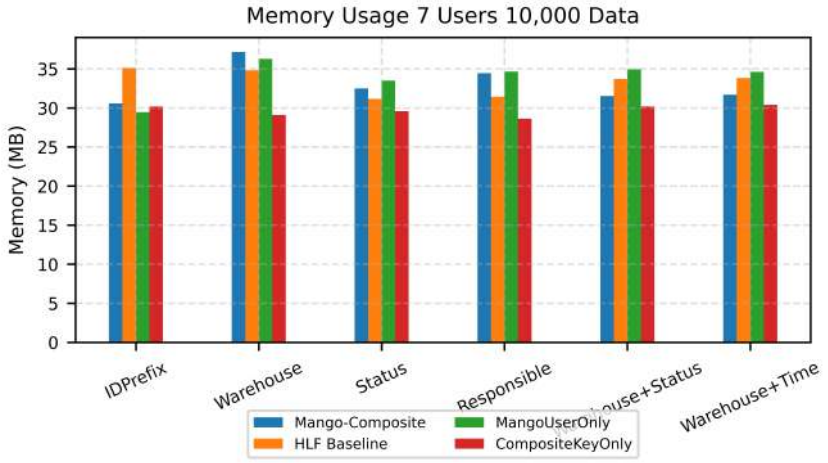

Fig. 4.15 Performance comparison of different query strategies on memory usage with 10,000 for 7 users — Mango-Composite, Baseline, MangoUserOnly, and CompositeKeyOnly.

4.4.4 Multi-User 10,000 Data Entries: Experimental Result and Analysis

The results of testing in a multi-user environment with 3, 5, and 7 users provide a comprehensive overview of the scalability and efficiency of each query method. The three main parameters are processing time, CPU usage, and memory usage. These three parameters were analyzed to identify trade-offs and the relative advantages of the proposed approach, namely the Mango-Composite combination.

The first parameter is processing time. The *Mango-Composite* method shows competitive performance in almost all query types, both single-field and multi-field. For multi-field queries such as *QueryByWarehouseAndStatus*, the proposed method was able to execute in an average time of 0.98 seconds, compared to the *HLF Base-*

line which required 16.21 seconds. This shows a reduction in time of 93.9%. For single-field queries like *QueryByIDPrefix*, *Mango-Composite* records an average time of 5.96 seconds, which is 54.2% faster than the *HLF Baseline* (13.04 seconds).

Next is the CPU usage parameter. CPU usage in the *Mango-Composite* combination method also successfully maintained efficiency. The average CPU usage is around 41%, lower than *Mango Query Only* (48.73%), and *Composite Key Only* (78.53%). This means CPU efficiency has improved by approximately 46.7% compared to Composite Key and 14% compared to Mango Query Only.

The last parameter is memory usage. Memory usage in the *Mango-Composite* method provides fairly competitive results. The average memory used is 30.48 MB, slightly lower than the *HLF Baseline* (31.73 MB), although not lower than the *Composite Key Only* method (28.18 MB). In one single-field query scenario such as *QueryByStatus*, *Mango-Composite* uses 33.09 MB, which is more efficient than Mango-only (34.96 MB or 5.3%) and HLF Baseline (33.77 MB or 2%).

Overall, *Mango-Composite* provides superior balance in a multi-user environment. By combining the strengths of selector index and prefix key search, this strategy achieves up to 93% improvement in execution time, up to 46% CPU efficiency, and up to 5% memory efficiency compared to the HLF baseline and conventional approaches. This demonstrates its reliability in permissioned blockchain ecosystems with high access loads, such as supply chain systems in the SME sector.

4.5 50,000 Data Experiment: Multi-user Environment

The last test was conducted in a multi-user environment scenario, where 3, 5, and 7 users execute data lookup requests independently against the Hyperledger Fabric network organizations (Org1, Org2, Org3). The purpose of this test is to measure the performance of each query strategy under conditions without resource contention, so that the results reflect the basic efficiency of each approach using 50,000 data entries.

Table 4.8 Query Performance Comparison in Multi-User Environment with 3 users testing and using 50,000 Data Entries

Query Strategy	Query Type	50,000 Records		
		Time (s)	CPU (%)	Memory (MB)
HLF Baseline	IDPrefix	29.65	61.33%	40.840
	Warehouse	8.37	69.79%	29.863
	Status	8.83	70.76%	29.691
	ResPerson	7.78	70.84%	28.721
	Ware + Stat	2.39	39.79%	27.918
	Ware + Time	6.47	48.77%	28.836
Mango Query	IDPrefix	28.72	53.34%	39.631
	Warehouse	7.40	66.57%	31.369
	Status	8.49	61.42%	32.204
	ResPerson	7.23	57.99%	28.056
	Ware + Stat	1.56	79.27%	28.673
	Ware + Time	0.14	89.46%	32.260
Composite Key	IDPrefix	0.13	85.69%	35.517
	Warehouse	0.12	89.41%	26.971
	Status	0.12	88.68%	26.887
	ResPerson	0.13	89.92%	28.305
	Ware + Stat	0.12	87.38%	27.844
	Ware + Time	0.12	88.33%	28.038
Mango-Composite	IDPrefix	27.64	49.14%	40.206
	Warehouse	6.46	64.83%	30.427
	Status	7.03	53.16%	33.250
	ResPerson	6.36	55.82%	27.473
	Ware + Stat	0.13	83.94%	28.401
	Ware + Time	0.12	85.88%	30.682

Table 4.9 Query Performance Comparison in Multi-User Environment with 5 users testing and using 50,000 Data Entries

Query Strategy	Query Type	50,000 Records		
		Time (s)	CPU (%)	Memory (MB)
HLF Baseline	IDPrefix	33.96	76.27%	43.817
	Warehouse	9.23	74.84%	36.447
	Status	9.63	77.56%	35.274
	ResPerson	11.87	80.41%	33.381
	Ware + Stat	7.61	85.86%	35.205
	Ware + Time	12.71	87.78%	33.192
Mango Query	IDPrefix	32.78	69.23%	44.128
	Warehouse	8.53	75.41%	36.258
	Status	9.23	74.93%	34.639
	ResPerson	10.25	77.73%	31.773
	Ware + Stat	3.93	88.23%	36.644
	Ware + Time	0.14	91.59%	37.183
Composite Key	IDPrefix	0.14	94.71%	38.682
	Warehouse	0.13	90.57%	35.614
	Status	0.14	94.05%	30.587
	ResPerson	0.14	93.36%	32.292
	Ware + Stat	0.14	94.37%	33.471
	Ware + Time	0.14	95.99%	35.144
Mango-Composite	IDPrefix	30.29	70.48%	46.111
	Warehouse	8.49	74.57%	35.381
	Status	8.71	71.16%	34.207
	ResPerson	8.29	73.07%	32.107
	Ware + Stat	0.13	89.89%	38.163
	Ware + Time	0.14	91.14%	37.386

Table 4.10 Query Performance Comparison in Multi-User Environment with 7 Users Testing and using 50,000 Data Entries

Query Strategy	Query Type	50,000 Records		
		Time (s)	CPU (%)	Memory (MB)
HLF Baseline	IDPrefix	68.34	98.18	47.571
	Warehouse	24.51	97.11	43.184
	Status	34.71	99.38	44.612
	ResPerson	62.13	100	46.306
	Ware + Stat	72.63	100	44.257
	Ware + Time	86.74	100	38.561
Mango Query Only	IDPrefix	53.52	98.48	48.442
	Warehouse	17.46	99.27	45.828
	Status	38.61	98.62	46.374
	ResPerson	30.65	100	48.542
	Ware + Stat	74.21	100	43.002
	Ware + Time	80.17	100	38.937
Composite Key Only	IDPrefix	3.18	100	47.140
	Warehouse	25.00	100	44.582
	Status	71.25	100	42.775
	ResPerson	74.93	100	46.284
	Ware + Stat	71.84	100	40.338
	Ware + Time	68.66	100	38.126
Mango-Composite	IDPrefix	59.36	99.39	48.942
	Warehouse	17.47	99.72	44.773
	Status	47.66	100	47.921
	ResPerson	29.02	99.48	49.249
	Ware + Stat	75.05	100	41.472
	Ware + Time	85.51	100	39.087

4.5.1 Multi-User 50,000 Data Entries: Processing Time Result and Analysis

Testing the query execution time in a multi-user environment (3, 5, and 7 users) with 50,000 entries shows that the Mango-Composite approach can provide competitive performance in almost all types of queries, although it is not always the fastest due to bottlenecks caused by virtualization configuration limitations, which only use 2vCPU and 6GB RAM. A summary of the test results can be seen in Table 4.8, Table 4.9, Table 4.10.

For simple or single-field queries such as QueryByIDPrefix with 5 users, the CompositeKeyOnly approach showed good performance, with processing times below 0.15 seconds for all numbers of users, and remained stable even as the user load increased. In contrast, the HLF Baseline method experienced a significant decline in performance, with execution time increasing from 29.65 seconds (3 users) to 76.34 seconds (7 users). Compared to the baseline, the Mango-Composite approach achieves execution time improvements of up to 61.5% faster on IDPrefix (3 users) and 59.9% faster on Responsible (5 users), reflecting the efficiency of this strategy.

Next, for queries with two attributes such as Warehouse+Status and Warehouse+Time, the Mango-Composite approach shows the best results at low concurrency levels. For example, with 3 users, the execution time for Warehouse+Status is only 0.12 seconds, compared to 2.39 seconds on the HLF Baseline (an improvement of 94.98%). However, when the number of users increases to 7, both Mango-Composite and MangoUserOnly experience a performance degradation due to CPU saturation. The execution time for Warehouse+Time even increases to 85.51 seconds, nearly matching the HLF Baseline at 86.74 seconds, indicating a bottleneck effect.

Interestingly, the CompositeKeyOnly approach remained fast and stable across all single-attribute queries, with execution times around 0.12-2 seconds even when tested with 7 users. However, for two-attribute queries like Warehouse+Status, execution times increased sharply (e.g., 83.59 seconds at 7 users), likely due to the overhead of composite key iteration.

Overall, the Mango-Composite approach with this scope and a load of 50,000 data points proved to be more scalable than the HLF Baseline and more flexible than CompositeKeyOnly. The execution time improvement achieved was up to 94.98% better depending on the query type and concurrency level. Although not the fastest in all scenarios, this approach provides a performance balance for various search patterns.

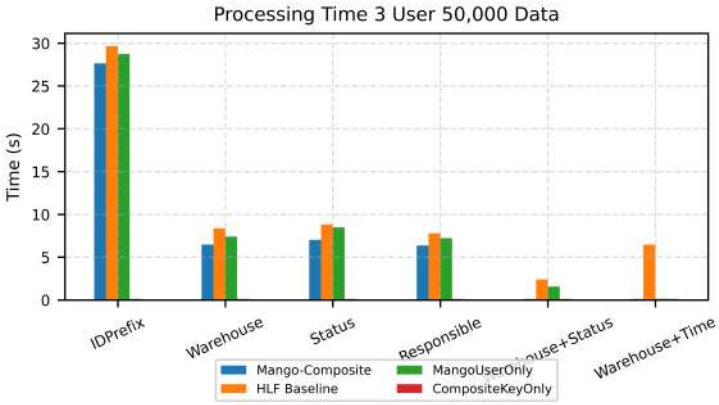

Fig. 4.16 Performance comparison of different query strategies on Processing Time with 50,000 for 3 users — Mango-Composite, Baseline, MangoUserOnly, and CompositeKeyOnly.

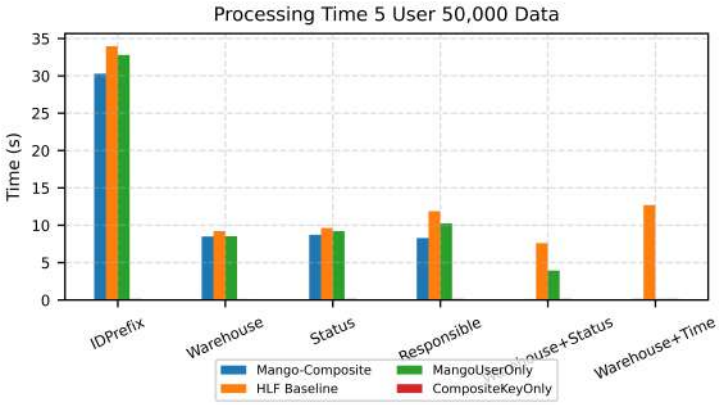

Fig. 4.17 Performance comparison of different query strategies on Processing Time with 50,000 for 5 users — Mango-Composite, Baseline, MangoUserOnly, and CompositeKeyOnly.

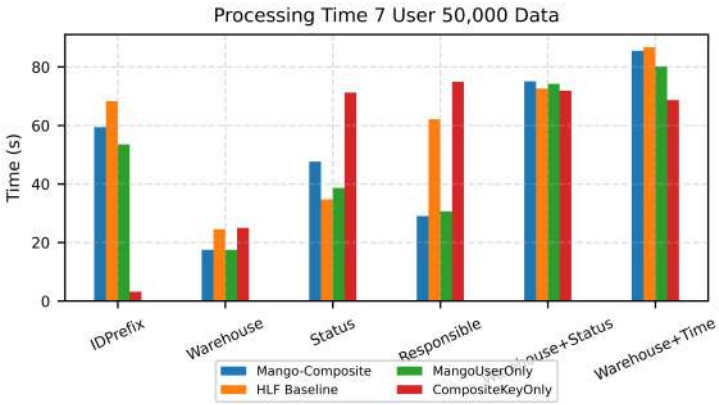

Fig. 4.18 Performance comparison of different query strategies on Processing Time with 50,000 for 7 users — Mango-Composite, Baseline, MangoUserOnly, and CompositeKeyOnly.

4.5.2 Multi-User 50,000 Data Entries: CPU Usage Result and Analysis

The results of testing CPU usage show significant differences between query methods, especially when the number of users increases from 3 to 7. In general, the CompositeKeyOnly approach experiences the highest CPU usage in all scenarios, often reaching 100%, indicating that although fast, this method places an extreme load on the processor and is computationally inefficient for large-scale simultaneous users.

Next, the Mango-Composite strategy showed more stable performance. The CPU usage range of Mango-Composite at 3 users was in the range of 40-86%, increasing to 69-91% for 5 users, and reaching 97-100% when tested with 7 users, which is close to the CPU usage of other methods. This indicates an increase in load, but with more controlled growth compared to other strategies. Compared to the HLF Baseline, which also reaches 100% for nearly all query types with 7 users, Mango-Composite maintains efficiency with fewer users while remaining competitive in high-performance scenarios.

However, with 7 users, all methods approached the maximum CPU limit (98-100%), indicating a bottleneck in computing resources on a larger scale. This opens up opportunities for further research into load balancing in the Hyperledger Fabric environment.

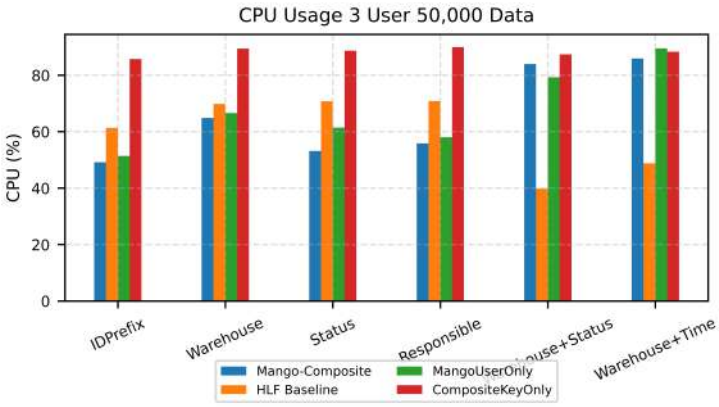

Fig. 4.19 Performance comparison of different query strategies on CPU Usage with 50,000 for 3 users — Mango-Composite, Baseline, MangoUserOnly, and CompositeKeyOnly.

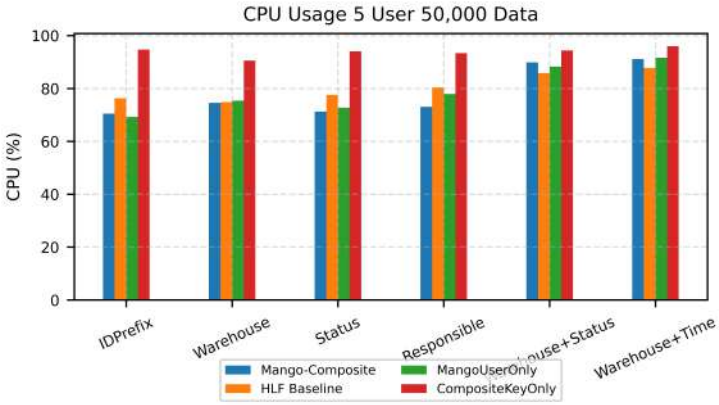

Fig. 4.20 Performance comparison of different query strategies on CPU Usage with 50,000 for 5 users — Mango-Composite, Baseline, MangoUserOnly, and CompositeKeyOnly.

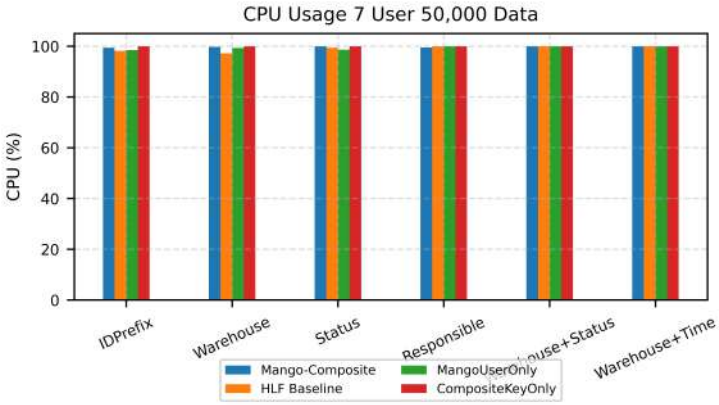

Fig. 4.21 Performance comparison of different query strategies on CPU Usage with 50,000 for 7 users — Mango-Composite, Baseline, MangoUserOnly, and CompositeKeyOnly.

4.5.3 Multi-User 50,000 Data Entries: Memory Usage Result and Analysis

The Mango-Composite and Composite Key Only strategies showed better memory efficiency than other strategies. This was particularly evident in the 3- and 5-user tests. With 7 users, the memory differences between strategies tended to even out.

In the 3-user test, the Mango-Composite strategy produced the lowest average memory usage in 4 out of 6 query types, with usage as low as 26.18 MB in QueryByStatus. This shows that the combination of Mango indexing with composite keys can reduce the memory footprint, especially in multi-field query scenarios. The CompositeKeyOnly method also shows consistently low results, although in some cases, such as QueryByWarehouseAndStatus, it is still slightly higher than Mango-Composite.

In tests with 5 users, the memory efficiency pattern is maintained. The Mango-Composite strategy is the most efficient for queries such as IDPrefix, Status, and Warehouse+Time. For example, in QueryByStatus, Mango-Composite only used 30.21 MB, while HLF Baseline and Mango Query Only reached 34.12 MB and 42.63 MB, respectively. This difference shows an efficiency of more than 29% compared to Mango Query Only.

When the number of users was increased to 7, all strategies began to show high memory consistency, although some methods such as Mango-Composite experienced large fluctuations, for example reaching 53,259 MB on QueryByResponsiblePerson. On the other hand, memory usage in the Mango-Composite method remained in the range of 41-49 MB, indicating stable memory usage.

Overall, Mango-Composite shows superiority in memory usage efficiency and stability in all user scenarios. Compared to HLF Baseline, memory savings can be better in certain queries. Meanwhile, CompositeKeyOnly competes closely, but in some cases exceeds Mango-Composite in memory usage, especially for simple queries. The reason why Mango-Composite sometimes has high memory usage is due to the switching method in query strategy selection based on two conditions, as explained in the previous chapter, such as single-field and multi-field.

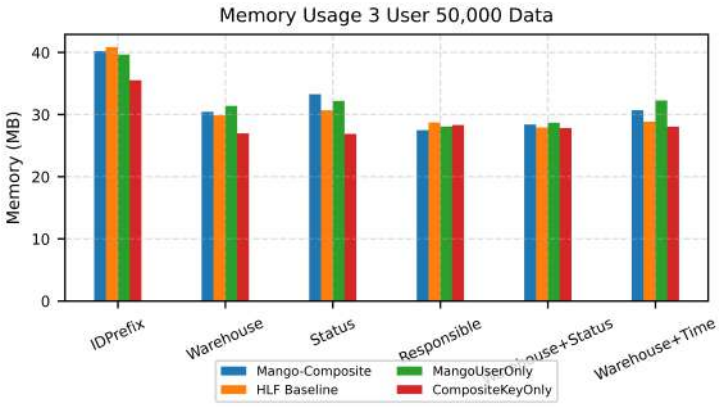

Fig. 4.22 Performance comparison of different query strategies on memory usage with 50,000 for 3 users — Mango-Composite, Baseline, MangoUserOnly, and CompositeKeyOnly.

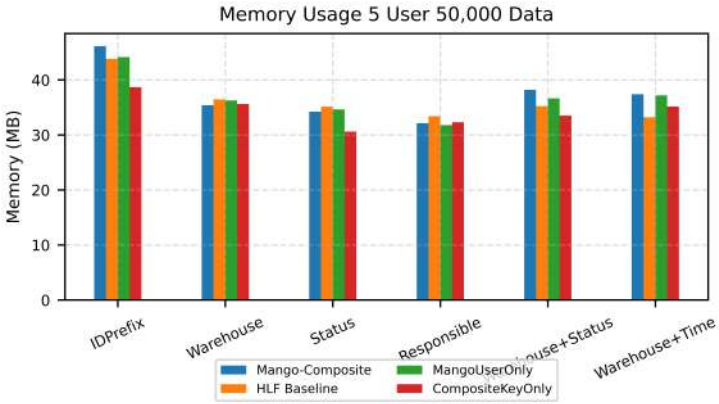

Fig. 4.23 Performance comparison of different query strategies on memory usage with 50,000 for 5 users — Mango-Composite, Baseline, MangoUserOnly, and CompositeKeyOnly.

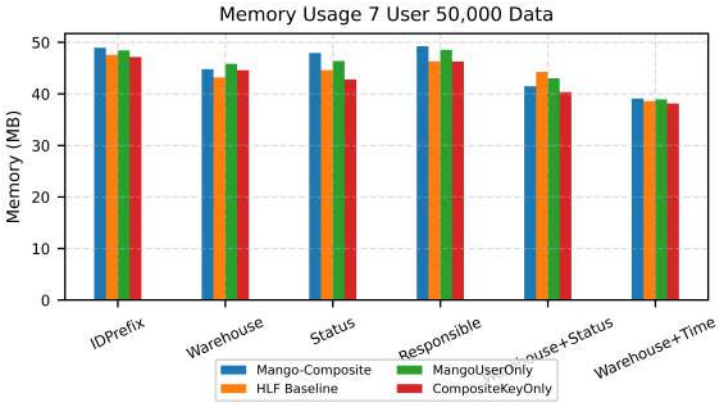

Fig. 4.24 Performance comparison of different query strategies on memory usage with 50,000 for 7 users — Mango-Composite, Baseline, MangoUserOnly, and CompositeKeyOnly.

4.6 Analysis of Memory Usage Fluctuations

Based on the tests that have been conducted, memory usage in various approaches shows a pattern that tends to be stable in most scenarios, with relatively small fluctuations between experiments. However, there are certain conditions where memory consumption shows a sudden increase (spike), especially in the Mango Query Only and Mango-Composite approaches. These spikes generally occur in high-complexity multi-field queries, such as QueryByWarehouseAndStatus and QueryByWarehouseAndTime, as well as in single-field queries like QueryByStatus that require more intensive selector processing in CouchDB.

These fluctuations can be explained by the nature of Mango Query, which scans documents based on indexes and the amount of data retrieved from the total amount of data in the database. Next, in the Mango-Composite strategy, although the use of Composite Key reduces the search load for some queries, integration with Mango Query still causes extra memory load on multi-field execution with large amounts of data.

Overall, despite fluctuations in certain cases, the peak memory consumption values observed are still within reasonable limits for the test system specifications (6 GB RAM). This shows that the proposed strategy remains feasible for use in a permissioned blockchain environment, with consideration for additional optimization if implemented at a production scale with higher loads.

CHAPTER 5
CONCLUSION

5.1 Conclusions and Future Works

This research proposes a query optimization strategy that combines the Mango Query and Composite Key mechanisms to improve data retrieval efficiency in Hyperledger Fabric systems, particularly in the context of blockchain-based supply chains. This strategy is designed with execution path selection based on the number of query conditions: Mango Query is used for single-field searches, while Composite Key is used for multi-field searches involving two or more attributes. This approach is implemented in test scenarios with datasets of 10,000 and 50,000 entries and multi-user workloads involving up to seven users.

The results of the experiment show that the Mango-Composite strategy is able to reduce processing time compared to the baseline approach significantly. In conditional (multi-field) queries, the execution time was recorded as low as 0.12 seconds, with a performance improvement of up to 90% compared to the baseline. For comparative validation, two additional approaches were also tested, namely Mango Query Only and Composite Key Only. Although Composite Key Only showed the fastest execution time in some cases, high CPU consumption was a major drawback due to the iterative process. On the other hand, Mango Query Only offered selector flexibility, but tended to be memory and CPU intensive in complex filters. The Mango-Composite strategy consistently balances speed and resource efficiency across all tests, both in single-user and multi-user scenarios.

Although this strategy incurs a slight overhead in multi-field conditions, its advantages in terms of responsiveness and efficiency make it suitable for implementation in permissioned blockchain systems with varying query patterns. In the future, this research can be further developed by integrating caching strategies and workload-aware query planning to reduce overhead in environments with high query frequencies.

REFERENCES

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Tech. Rep., 2008. [Online]. Available: www.bitcoin.org

[2] V. Buterin, "Ethereum: A next-generation smart contract and decentralized application platform." Tech. Rep., 2014. [Online]. Available: www.ethereum.org

[3] Y. F. Wen and C. M. Hsu, "A performance evaluation of modular functions and state databases for hyperledger fabric blockchain systems," *Journal of Supercomputing*, vol. 79, pp. 2654–2690, 2 2023.

[4] C. Wang and X. Chu, "Performance characterization and bottleneck analysis of hyperledger fabric," in *Proceedings - International Conference on Distributed Computing Systems*, vol. 2020-November. Institute of Electrical and Electronics Engineers Inc., 11 2020, pp. 1281–1286.

[5] S. Zhang, S. Hua, B. Pi, J. Sun, K. Yamashita, and Y. Nomura, "Performance diagnosis and optimization for hyperledger fabric," in *2020 2nd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, 2020, pp. 210–211.

[6] E. Zhou, H. Sun, B. Pi, J. Sun, K. Yamashita, and Y. Nomura, "Ledgerdata refiner: A powerful ledger data query platform for hyperledger fabric," in *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, Oct 2019, pp. 433–440.

[7] T. Yan, W. Chen, P. Zhao, Z. Li, A. Liu, and L. Zhao, "Handling conditional queries and data storage on hyperledger fabric efficiently," *World Wide Web*, vol. 24, pp. 441–461, 1 2021.

[8] J. Lohmer, E. R. da Silva, and R. Lasch, "Blockchain technology in operations and supply chain management: A content analysis," *Sustainability (Switzerland)*, vol. 14, 5 2022.

[9] A. E. Mohamed, *Inventory Management*. IntechOpen, 7 2024. [Online]. Available: https://www.intechopen.com/chapters/88430

[10] M. Seyedan and F. Mafakheri, "Predictive big data analytics for supply chain demand forecasting: methods, applications, and research opportunities," *Journal of Big Data*, vol. 7, 12 2020.

[11] M. A. B. M. A. T. Md Khyrul Islam, Hasib Ahmed, "Role of artificial intelligence and machine learning in optimizing inventory management across global industrial manufacturing and supply chain: A multi-country review," *GLOBAL MAINSTREAM JOURNAL*, pp. 1–14, 5 2024. [Online]. Available: https://globalmainstreamjournal.com/index.php/IJMISDS/article/view/105

[12] S. Holloway, "Impact of digital transformation on inventory management: An exploration of supply chain practices," *Preprints*, July 2024. [Online]. Available: https://doi.org/10.20944/preprints202407.0714.v1

[13] T. A. Alghamdi, R. Khalid, and N. Javaid, "A survey of blockchain based systems: Scalability issues and solutions, applications and future challenges," *IEEE Access*, vol. 12, pp. 79 626–79 651, 2024.

[14] S. A. Abeyratne and R. P. Monfared, "Blockchain ready manufacturing supply chain using distributed ledger," pp. 2321–7308. [Online]. Available: http://ijret.esatjournals.org

[15] Z. Liu, L. Ren, Y. Feng, S. Wang, and J. Wei, "Data integrity audit scheme based on quad merkle tree and blockchain," *IEEE Access*, vol. 11, pp. 59 263–59 273, 2023.

[16] P. Swathi and M. Venkatesan, "Scalability improvement and analysis of permissioned-blockchain," *ICT Express*, vol. 7, pp. 283–289, 9 2021.

[17] G. T. Ho, Y. M. Tang, K. Y. Tsang, V. Tang, and K. Y. Chau, "A blockchain-based system to enhance aircraft parts traceability and trackability for inventory management," *Expert Systems with Applications*, vol. 179, 10 2021.

[18] W. Zheng, X. Wang, Z. Xie, Y. Li, X. Ye, J. Wang, and X. Xiong, "Data management method for building internet of things based on blockchain sharding and dag," *Internet of Things and Cyber-Physical Systems*, vol. 4, pp. 217–234, 1 2024.

[19] S. Johar, N. Ahmad, W. Asher, H. Cruickshank, and A. Durrani, "Research and applied perspective to blockchain technology: A comprehensive survey," *Applied Sciences (Switzerland)*, vol. 11, 7 2021.

[20] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, S. Manevich *et al.*, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*. ACM, 2018, pp. 1–15.

[21] W. Zheng, X. Wang, Z. Xie, Y. Li, X. Ye, J. Wang, and X. Xiong, "Data management method for iot based on blockchain sharding and dag," *Internet of Things and Cyber-Physical Systems*, vol. 4, pp. 217–234, 2022.

[22] S. A. Razoqi, "Reasons of the transformed toward nosql databases and its data models," *Journal of Education & Science*, vol. 29, no. 2, 2020.

[23] A. Stoltidis, K. Choumas, and T. Korakis, "Performance optimization of high-conflict transactions within the hyperledger fabric blockchain," in *2024 6th Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, 2024, pp. 1–4.