### **Objective**

This report documents a multi-phase project focused on deploying and using Apache Cassandra in a distributed setting and enhancing its integrity through blockchain integration. The project has two major parts:

1. **Part 1**: Set up and operate a Cassandra cluster, import log data, and analyze it using Python and CQL.
2. **Part 2**: Implement a blockchain-assisted verifiable data query system using Merkle Trees and Ethereum.

## **Part 1: Distributed Log Analysis Using Apache Cassandra**

### **A. Setting Up Cassandra Cluster (50 Points)**

**Goal:** Configure a multi-node Apache Cassandra cluster across three virtual machines.

**Steps Performed:**

1. Installed Java 8 and Python 3 on all nodes.
2. Added Apache Cassandra repository and imported keys.
3. Installed Cassandra using apt and configured cassandra.yaml on all nodes.
4. Configured seeds, listen\_address, and rpc\_address per node.
5. Started Cassandra and validated setup using nodetool status.
6. Created a test keyspace and table:

CREATE KEYSPACE patient WITH replication = {'class': 'SimpleStrategy', 'replication\_factor': 1};  
CREATE TABLE patient.exam (patient\_id int, id int, date timeuuid, details text, PRIMARY KEY (patient\_id, id));

1. Inserted and verified sample records in CQLSH.

### **B. Importing Log Data into Cassandra (25 Points)**

**Goal:** Efficiently import structured log data into Cassandra using DSBulk.

**Steps Performed:**

1. Created log\_keyspace and a structured log table.
2. Used DataStax Bulk Loader (DSBulk) with CSV file and config file to import log data.

dsbulk load -k log\_keyspace -t log\_data\_small\_log -url output\_log\_file\_New.csv -header true -f dsbulk.conf -h 10.254.3.108 --port 9042

### **C. Querying Log Data with Python and CQL (25 Points)**

**Goal:** Answer analytical queries on log data stored in Cassandra.

**Sample Queries Implemented:**

* Hits to "/administrator/index.php"
* Hits from IP 96.32.128.5
* Most accessed URL and IP
* Firefox and Mozilla accesses
* GET request ratio on 02/Apr/2022
* Requests with size <= 404 bytes
* IPs with >10 404s or the top IP with most 404s

## **Part 2: Blockchain-Assisted Verifiable Cassandra**

### **Goal:**

Implement a verifiable query architecture that ensures trustworthiness in outsourced databases by using Merkle Trees and Ethereum smart contracts.

### **Components:**

* **Data Owner (DO):** Prepares KV dataset and constructs Merkle Tree
* **Service Provider (SP):** Stores data in Cassandra
* **Query Client (C):** Queries and verifies results using Merkle Proof
* **Malicious Client (MC):** Simulates data tampering
* **Ethereum Blockchain:** Stores immutable Merkle root for verification

### **Setup:**

1. Installed Python dependencies:

pip3 install cassandra-driver merkletools py-solc-x web3==5.31.4

1. Started Ethereum test environment using Ganache
2. Ran Cassandra and inserted data from DO
3. Executed driver.py to simulate full flow of upload, query, and verification

### **Verification Modes:**

* **No Attack:** Queried value matches Merkle Root
* **With Attack:** Tampered value fails verification (Merkle Root mismatch)

### **Summary of Part 2:**

This portion demonstrates how blockchain can be used to establish trust in a distributed database system. The proof-of-concept integrates Merkle Tree generation, smart contract interaction, and tamper detection.

## **Conclusion**

This project successfully integrates two powerful paradigms:

* **Cassandra** for distributed, scalable, and fault-tolerant data storage
* **Blockchain + Merkle Trees** for verifiable and tamper-proof queries