Ledger as an auditing solution for Cloud Database

Akash Malla
Computer Science
and Engineering Dept.
Santa Clara University
amalla1@scu.edu

Sharon Subathran
Computer Science
and Engineering Dept.
Santa Clara University
ssubathran@scu.edu

Neha Soma
Software
Engineering Dept.
Santa Clara University
nsoma@scu.edu

Sneha Mule
Computer Science
and Engineering Dept.
Santa Clara University
smule@scu.edu

https://github.com/SomaNeha/COEN241CloudChainTeamProject

***Abstract*—NoSQL databases, being non-relational, distributed and horizontally scalable are able to satisfy most of the needs of the present day applications. They provide high performance and scalability even when handling large volumes of unstructured data. This paper aims to implement a system that captures and aggregates data from server logs, providing a platform from which data can be easily reviewed and analyzed.In this project, we capture all long-running database queries that are requested by applications deployed on AWS EC2 instance. We maintain a list of all such query requests and call it a ledger. This ledger can be used to carry out performance analysis of such queries.**

Keywords: Distributed Ledger, Digital Signature, Virtual Machine(VM), NoSQL(MongoDB)

I. Introduction

As data keeps increasing in size and has an unstructured form, there is a dire urgency to store big data efficiently and in a secure manner. Traditional RDBMS databases have limitations in terms of non-linear query execution time and static relational schema model. As a workaround to increase performance, a primary and secondary index could be created on selected table columns in a schema in order to retrieve data faster. The addition of indexes has a cost to it as it requires more space, possibly terabytes of data, which in turn degrades performance drastically. The cost of hardware to store data and maintenance to support are major drawbacks. Although RDBMS has a fair share of benefits that include robustness, simplicity, flexibility, and performance, it is not the right solution to handle the current scale of data warehousing, data inflow from cloud applications, social media, and IoT devices. As a result of problems from RDBMS, NoSQL database has been developed to be schemaless and provide better performance for big data storage. NoSQL(MongoDB) databases are capable to scale horizontally to ensure high availability and scalability. It supports dynamic schema to store semi-structured or unstructured data.[3].

The main reason why many organizations still use RDBMS is solely due to inadequate security in place of NoSQL databases. In 2013, Cloud Security Alliance (CSA) alluded that NoSQL databases have poor authentication and no protection for data integrity. A common solution to protect sensitive data is through data encryption[4]. MongoDB is a popular choice to store application logs thanks to its schema-free design making it flexible enough to use for storing data whose schema tends to change from time-to-time.

The following Table-I, gives the comparison between Distributed Ledger based CryptDB and other databases

TABLE I
Comparison between various Databases

Name	Description	Pros	Cons
RDBMS	Structured	Complies all ACID properties and has some security	Slow to run query for big data. Requires creation of index on data to improve performance
NoSQL	Schemaless, Scalable, unstructured database	Good performance to run queries to retrieve big data compared to RDBMS	Lacks proper security. Eventual consistency can let update, insert and deletion of records take time
CryptDB	Schemaless, Scalable, unstructured database	Provides strong security as data is encrypted	Eventual consistency can let update, insert and deletion of records take time.
Distributed Ledger (Our proposed solution)	Schemaless, Scalable, unstructured database that uses distributed ledger for validation of queries	Improves current performance by monitoring active MongoDB operations	Additional Space for ledger and requires daemon process to run in order to capture current operations for an interval

Servers generate a large number of events/logs (i.e. logging,) that contain information about their operation including errors, warnings, and users behaviour. Clients extract data from these database logs to carry out analysis on the performance. By default, this data is stored in plain text log files which are time-consuming, difficult to review, reference, and visualize without an efficient system for aggregating and storing this data. This is the system that we implement- a way to eliminate the heavy lifting of formatting and capture all active operation-related data in a single collection, making it easy to reference and analyze for a client to be able to query. The project was implemented in public cloud, it could implement as a database component in private cloud models.

A. *Motivation/Background*

The use of Blockchain technology[2] was first introduced by Bitcoin. The chronic feature of this technology enhances trust through transparency and traceability within any transaction of data and financial resources. It has a set of benefits: decentralized control, there is no central authority that owns or controls the network thus avoiding a single point of failure; immutability, where written data is tamper-resistant forever; and the ability to create and transfer assets on the network, without reliance on a central entity Decentralized Ledger is a key feature of this technology and Bitcoin meant for it to behave as a distributed transaction management system to avoid spending the coin multiple times. So there is a list which consists of various transactions, said to be a ledger or block. Each block/ledger is generated by proof-of-work and appended to the existing blockchain. The concept of distributed ledger provides an incentive platform for distributed database since it exists across a decentralized network of machines that allows transparent transaction mechanisms in various businesses [Fig 1].

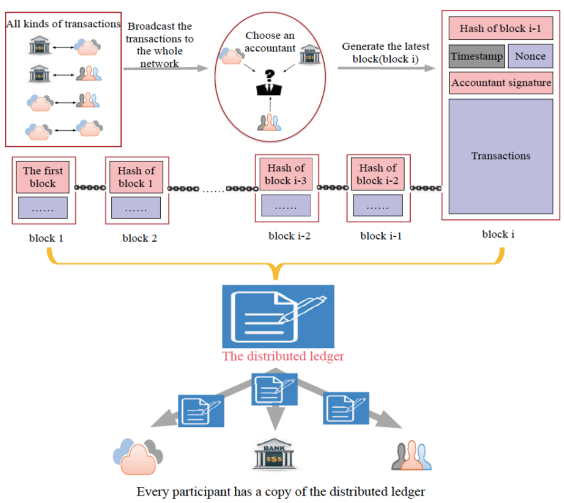

Fig. 1. Demonstrates blockchain for financial institutions

As VM management system plays a central role in a distributed ledger for cloud database operations that involve important information, a protection mechanism is built. This mechanism guarantees only authorized users can access corresponding data on the ledger. We also evaluate the proposed system from performance and security perspectives to demonstrate its usefulness in the real world environment.

II. Problem Analysis

Virtual Machine refers to the software implementation of any computing device or machine which is used to execute the program as physical machines. When a user tries to work on virtual machines they acquire the resources of the virtualization installed in the remote machines which are been accessed by related protocols so as to behave like a local machine. We used public cloud in our project as private clouds often cost much more than public clouds. In order to implement and work on shared systems so that the resources and workload are been shared among the systems.

There are many ways to isolate workloads; such as are separate machines, physically separate machines, virtualization, containerization (OS-level virtualization). It is said that virtualization can be used on physically separated machines, or by running containers inside hardware virtualization which says it is one of the strongest isolation mechanisms in the cloud environment with inbuilt cloud security concerns to protect from data breaches.A strong form of isolation is to not connect machines at all through any network. This has its advantages for security. It means that the vulnerabilities in a computer, or an operating system's networking stack, or the OS low-level services, cannot be exploited over a network.In virtualization. For more security concerns instead of using virtualization, we used VM instead Docker- container to avoid a large number of data leakages that damage the organizations.

As we have log and Operations in the database so that we can have a history of queries running and keep track of them by time. The log is based on changes that are made to data in the tables that you track. Also, the log consists of, entries are which chronologically ordered and show changes that are made to the fields on the specified tables.

In our project we would like to have the collection of all logs when and what changes with respective time, date and updated queries that are made in the ledger. As Servers generate a large number of events/logs (i.e. logging,) that contain information about their operation including errors, warnings, and users behavior. Clients extract data from these database logs to carry out analysis on the performance. By default, this data is stored in plain text log files which are time-consuming, difficult to review, reference, and visualize without an efficient system for aggregating and storing this data. This is the system that we implement a way to eliminate the heavy lifting of formatting and capture all active operation-related data in a single collection, making it easy to reference and analyze for a client to be able to query.

A. *Contributions*

Our contributions in this work are summarized as follows:

- We propose a solution that employs the concept of Distributed Ledger Technology (DLT) to enhance the security and consistency of data.
- This paper focuses typically to help, many clients who have to extract data from database logs to analyze performance issues.
- Reviewing and formatting log data is time-consuming, if not, more time consuming than analyzing formatted data using a visualization utility or machine learning algorithm
- We wanted to eliminate the heavy lifting of formatting and capture all active operation-related data in a single collection for any client to be able to query.

III. Proposed Solution

We started off the initial phase of out project researching blockchain concepts and wanted to implement them in such a way so as to improve the security and consistency of data in a NoSQL DB. With further research, we felt that we really could not make relevant contributions to existing solutions. So with the knowledge that we had compiled from our research on NoSQL databases and the concept of ledgers used in blockchains, our new aim was to provide a concise and clean platform to audit logs in a NoSQL database. In a real-world scenario, there will be multiple applications that contact the database to either make changes or retrieve some data. Servers generate a large number of events/logs (i.e. logging,) that contain information about their operation including errors, warnings, and users behavior. Typically, many clients have to extract data from database logs to analyze performance issues.Reviewing and formatting log data is time-consuming if not, more time consuming than analyzing formatted data using a visualization utility or machine learning algorithm. We wanted to eliminate the heavy lifting of formatting and capture all active operation-related data in a single collection for any client to be able to query.This is the problem that we aim to solve with this solution. The core component of our solution is to capture the data related to longer running queries. We stored data in the ledger in the MongoDB. MongoDB is installed in Amazon AWS EC2 instance (VM)[15]. we propose a solution with a ledger that captures all operations related to queries which are taking more time to run in MongoDB. For a simulation of a number of queries generated, in this solution, we used to stress test which inserts, update data for every second. We wrote a python script which can fetch important data into ledger about MongoDB long-running queries.we used mongo compass tool to visualize data in the ledger.

The output of above queries will provide operation IDs of the current running or queued operations in mongoDB. The output is analyzed and appended to ledger if the operation ID does not exist. Ledger is a collection in mongoDB that stores operation ID, timestamp, and a status flag as a document in the ledger collection. By default Status Flag is set to FALSE. The daemon process interval is set to 1 minute, so we collect and maintain the output of the run by updating the ledger every time. Executed operations can be determined by comparing the operation IDs that are present in the ledger against the Daemon process's result. If they are not present, it implies that the operation has been successfully executed and therefore the status flag for those operations is set to "TRUE". The ledger will hold operations for an interval of one hour.

IV. Implementation

This section discusses how ledger as a concept could be integrated into any NoSQL database. Ledger, as previously mentioned, can be thought of data that consists of a list of transactions in cryptocurrencies such as bitcoin. Similarly, keeping track of certain transactions that occur in NoSQL database(for example MongoDB)[11] provides great insight for improving performance and analyzing what types of operations are performed. A ledger could easily answer questions like what and why is the database performing the way it is. Just like how customers currently create and maintain their own tables or collections to hold their data, we propose to create and maintain a ledger that would hold database related statistics added continuously. MongoDB is not only the most popular NoSQL databases in the current market but also the fastest growing database today. That is why MongoDB has been chosen as the database to do a test and present the model. As you can see in [Fig 2], one box consists of multiple collections that hold customer data and the other box consists of the ledger.

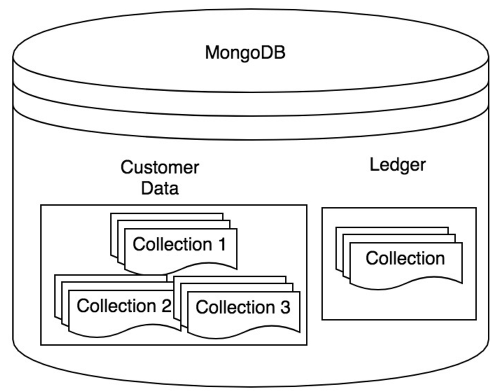

Fig. 2. Customer Data will consist of multiple collections and ledger is a single collection

A. *Ledger and its parameters*

Ledger is a single collection that is appended with data as a JSON object which is a document in MongoDB for a set interval. An operation related details are added to the ledger if and only if the operation is active, which means it is currently running, and if the operation is running longer than 50ms. The reason for maintaining a separate collection for auditing purposes provide benefits such as:

- Being able to query the ledger using MongoDB query syntax.
- The flexibility of adding or removing parameters or key-value pairs easily in the ledger per each customers requirement.
- Each document in the ledger is constructed by fetching real-time database statistics using commands such as current and server status. In other words, it is a central location to find all necessary auditing details.

The parameters or fields chosen to be added to the ledger comprises of the utmost important data that can generally help for analysis purpose. Below is a list of parameters and why they would be useful to keep track of [Fig 3]

- Operation ID: It is a unique id that each active MongoDB operation is associated with.

```
_id: ObjectId("5aaaf698d49ad729c08f58db")
opid: 291619
operation: "update"
microsecs_running: 330
client: "127.0.0.1:36720"
namespace: "POCDB.POCCOLL5"
time: 2018-03-15 08:41:28.322
current_connections: 9
available_connections: 51191
active_clients: 16
current_queue: 0
network_bytesIn: 86200665
network_bytesOut: 56528302
network_numRequests: 291360
opcounters_insert: 144862
opcounters_query: 144414
opcounters_update: 143942
extra_info_page_faults: 193
memory_virtual: 1029
memory_resident: 738
```

Fig. 3. Demonstration of our Ledger

  - If the python script[13] is run every second and a duplicate operation ID is found in the ledger, then we know for a fact that the same operation has been running for at least a second now.
- Operation Type: Shows the type of operation (insert, update, delete, query etc).
- Microseconds running: Execution time of an operation in milliseconds.
- Client IP Address: This information can allow how many clients are sending requests to the MongoDB instance for an interval of time.
- Namespace: Collection on which the operation is being executed.
- Current and available connections: Helps track whether there needs to be an increase in available connections.
- Network byte input and byte output: Provides network level data transmission from and to the MongoDB instance.
- The network number of requests
- Total number of inserts, queries, updates at the moment Number of page faults
- A virtual and resident memory of a process: Virtual memory stands for total memory that a program is able to access and resident memory stands for physical memory that is being consumed by the process.

B. Results

For this implementation, first, an AWS EC2 Ubuntu instance has been deployed to install a MongoDB instance. The core component of the implementation is to have a utility that can load test MongoDB instance during the time python script is collecting and adding objects to the ledger for logging. There is an open source tool called POCDriver.jar (https://github.com/johnlpage/POCDriver) that provided a simple interface to provide different parameters such as how many collections should be used, number of threads, whether the only insert, only updates, only query or a combination of the different operations that need to be executed during the load test. This flexibility provided by POCDriver allowed us to test various scenarios. Below [Fig 4] is an example of running the load test by asking it to simulate, inserts, updates, queries, use 3 threads and 10 collections, run for a 10-minute duration.

```
ubuntu@ip-172-31-27-35:~$ java -jar POCDriver.jar -k 20 -i 20 -u 20 -d 600 -t 3 -e -y 10
MongoDB Proof Of Concept - Load Generator
Worker thread 0 Started.
Worker thread 1 Started.
Worker thread 2 Started.
------------------------
After 10 seconds, 15165 new records inserted - collection has 15165 in total
1387 inserts per second since last report 0.00 % in under 50 milliseconds
144 keyqueries per second since last report 100.00 % in under 50 milliseconds
1453 updates per second since last report 90.48 % in under 50 milliseconds
0 rangequeries per second since last report 100.00 % in under 50 milliseconds

------------------------
After 20 seconds, 36645 new records inserted - collection has 36645 in total
2148 inserts per second since last report 0.00 % in under 50 milliseconds
210 keyqueries per second since last report 100.00 % in under 50 milliseconds
2157 updates per second since last report 90.36 % in under 50 milliseconds
0 rangequeries per second since last report 100.00 % in under 50 milliseconds
```

Fig. 4. Working on ubuntu shell

```
ubuntu@ip-172-31-27-35:~$ mongostat
insert query update delete getmore command % dirty % used flushes vsize   res qr|qw ar|aw netIn netOut conn                 time
    *0    *0     *0     *0       0    14|0     0.0    0.1       0  280M 53.0M   0|0   0|0 1.99k  52.7k    4 2018-03-15T21:09:53Z
    *0    *0     *0     *0       0    18|0     0.0    0.1       0  280M 53.0M   0|0   0|0 1.67k   286k    3 2018-03-15T21:09:54Z
    *0    *0     *0     *0       0    19|0     0.0    0.1       0  280M 53.0M   0|0   0|0 1.74k   267k    2 2018-03-15T21:09:55Z
    *0    *0     *0     *0       0    14|0     0.0    0.1       0  280M 53.0M   0|0   0|0 1.99k  52.1k    4 2018-03-15T21:09:56Z
   339   416    331     *0       0    34|0     0.1    1.9       0  304M 73.0M   0|0   1|0  216k   343k    5 2018-03-15T21:09:57Z
  1016  1100   1181     *0       0    26|0     0.5    5.2       0  339M  106M   0|0   1|0  636k   551k    7 2018-03-15T21:09:58Z
  1429  1258   4193     *0       0    10|0     1.1    7.7       0  364M  130M   0|0   0|0 1.08m   389k    9 2018-03-15T21:09:59Z
   864  1046  13980     *0       0    24|0     1.3    8.6       0  374M  140M   0|0   0|1 1.67m   592k    5 2018-03-15T21:10:00Z
  1854  1709   1949     *0       0    10|0     2.0   10.3       0  391M  157M   0|0   0|1 1.10m   452k    7 2018-03-15T21:10:01Z
  1621  1652   1667     *0       0    47|0     2.5   11.5       0  405M  170M   0|0   0|0  977k  1.11m    5 2018-03-15T21:10:02Z
insert query update delete getmore command % dirty % used flushes vsize   res qr|qw ar|aw netIn netOut conn                 time
  1896  1828   1793     *0       0    28|0     3.1   12.7       0  418M  182M   0|0   0|0 1.11m   692k    9 2018-03-15T21:10:03Z
  1792  1822   1636     *0       0    24|0     3.7   13.9       0  430M  194M   0|0   0|0 1.05m   817k    7 2018-03-15T21:10:04Z
  2032  1783   1997     *0       0    20|0     4.3   15.0       0  443M  206M   0|0   0|0 1.17m   585k    7 2018-03-15T21:10:05Z
  1727  2079   1809     *0       0    18|0     5.0   16.2       0  456M  218M   0|0   0|0 1.08m   744k    8 2018-03-15T21:10:06Z
  2029  1815   2035     *0       0    10|0     5.2   17.4       0  468M  229M   0|0   0|0 1.18m   459k    5 2018-03-15T21:10:07Z
  2080  2143   2043     *0       0    27|0     5.3   18.6       0  482M  242M   0|0   0|0 1.24m   822k    5 2018-03-15T21:10:08Z
  2220  2159   2177     *0       0    28|0     5.3   19.9       0  495M  256M   0|0   0|0 1.31m   861k    5 2018-03-15T21:10:09Z
  2185  2003   2160     *0       0    10|0     4.9   21.0       0  508M  268M   0|0   0|0 1.28m   503k    8 2018-03-15T21:10:10Z
  2090  2282   2148     *0       0    24|0     5.1   22.3       0  522M  281M   0|0   1|0 1.28m   877k    5 2018-03-15T21:10:11Z
  2063  1924   2063     *0       0     6|0     4.8   23.4       0  535M  293M   0|0   1|0 1.21m   480k    6 2018-03-15T21:10:12Z
insert query update delete getmore command % dirty % used flushes vsize   res qr|qw ar|aw netIn netOut conn                 time
  2050  2102   2067     *0       0    30|0     4.8   24.6       0  548M  304M   0|0   0|0 1.23m   826k    5 2018-03-15T21:10:13Z
  2049  2059   2007     *0       0    16|0     4.7   25.7       0  561M  316M   0|0   0|0 1.22m   548k    8 2018-03-15T21:10:14Z
  2142  2141   2173     *0       0    21|0     4.7   26.8       0  573M  327M   0|0   0|0 1.28m   706k    7 2018-03-15T21:10:15Z
  2425  2218   2210     *0       0    18|0     5.2   28.2       0  588M  341M   0|0   0|0 1.39m   658k    5 2018-03-15T21:10:16Z
  1994  2165   2052     *0       0    32|0     5.2   29.2       0  600M  352M   0|0   1|0 1.22m   832k    5 2018-03-15T21:10:17Z
  2723  2444   2723     *0       0    22|0     6.2   30.6       0  614M  364M   0|0   0|0 1.59m   846k    9 2018-03-15T21:10:18Z
  2403  2460   2451     *0       0    15|0     5.7   31.9       0  627M  376M   0|0   0|0 1.44m   656k    5 2018-03-15T21:10:19Z
  2651  2668   2548     *0       0    17|0     5.6   33.2       0  642M  389M   0|0   0|0 1.57m   661k   11 2018-03-15T21:10:20Z
  2629  2664   2664     *0       0    23|0     5.5   34.6       0  657M  402M   0|0   0|0 1.57m  1.03m    5 2018-03-15T21:10:21Z
  2690  2701   2720     *0       0    31|0     5.3   35.9       0  672M  416M   0|0   1|1 1.60m   936k    7 2018-03-15T21:10:22Z
```

Fig. 5. Demonstration of Mongostats

C. Evaluation

For evaluating the results of running a python script to populate ledger collection, mongo compass[12] tool has been used. Mongo compass allows to query and manage MongoDB instance. It provides data visualization automatically by connecting to the ledger collection, whether or not you provide a filter. This is a start for any customer to analyze the data captured in the ledger. As you can see in [Fig 6], it shows bar charts to understand long running operations and for example, tells how much portion of the data is of a particular operation type. In this figure, there has been a filter applied to show the visualization of insert operation only. While running the utility with 3 threads, the CPU utilization reaches high levels very quickly. But with further analysis, it was found that this was due to the inherent property of the working of the load balancing tool and our proposed system suffers no separate performance impact.

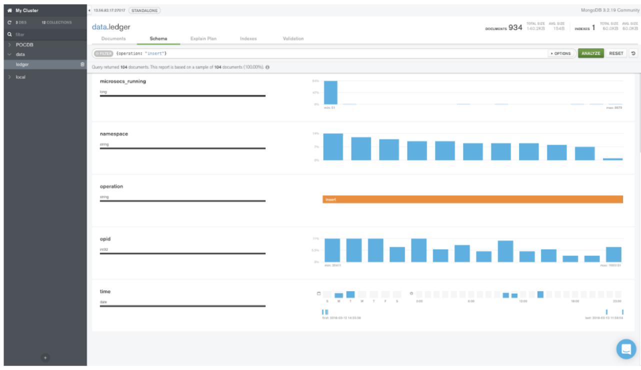

Fig. 6. Visualization of data

V. Conclusion

In our project, we tried to implement a solution which is to enhance the data storage about the queries which takes a longer time to execute in a systematic format. We used a ledger to store MongoDB log data. The customer can choose an appropriate time interval to insert into ledger (For example every second, 30 seconds, a minute or more). The customer has the flexibility to keep or remove ledger data depending on how much space available in their cloud environment. It is a concise version of typical NoSQL log entries. Ledger stores data related to the operation and can be useful for data analysis purpose. By using this solution in the data analysis, the service provider can take action in advance to prevent long-running query delay. The service provider can use machine learning algorithms to find common patterns or performance of each operation type which create long-running queries and take precautionary action to avoid delay in the queries. It can give the ability to monitor data in heavy traffic (For example Thanksgiving weekend) and Low Traffic time during the course of a year. This solution is advisable assuming space is not a constraint for a client. In a cloud environment, space constraint can be resolved by simply adding more machines (horizontally Scaling), however, if adding more machines leads to going over a clients budget, then this solution may not be advisable in that scenario.

VI. Future Work

This solution can be useful for the data analysis purpose. Using this data service provider can Use machine learning algorithm to analysis common pattern to find common patterns or performance of each operation type which create long-running queries and take precautionary action to avoid delay in the queries. It can be used for visualization of clustering data, error detection, graphical representation.

The solution of maintaining system logs in the form of a ledger can be extended to other NoSQL databases like Cassandra and Redis with a prediction of similar results.

VII. Related Work

In this section,[5] we briefly review existing works on using Crypt-NoSQL, the first prototype to support execution of query over encrypted data on NoSQL databases with high performance. CryptDB utilized a proxy server to translate and rephrase the client's query before it goes into the database, making this query executable over encrypted contents.It applies several encryption schemes that are called SQL-aware to data, which means the execution of some specific queries over the encrypted contents will become possible by using their encryption schemes. However, CryptDB has many limitations like it require more storage space. CryptDB cannot perform server-side computations on values encrypted for different principals because the ciphertexts are encrypted with different keys. And it only has the worst security guarantee, and it requires too much space.

This paper is an approach which[8] Corda R3 enables the technologies by using distributed ledger or a blockchain technology. Corda platform in AWS which uses a shared smart contract to encapsulate thebusiness logic of a transaction between organizations. This is all achieved with cryptographic technologies that provides an immutable record on a shared ledger system where data is ensured by validated users, which deals to enable strongest privacy assurance in the industry.

References

[1] A. Brinckman et al, *"A Comparative Evaluation of Blockchain Systems for Application Sharing Using Containers,"* 2017 IEEE 13th International Conference on e-Science (e-Science), Auckland, 2017, pp. 490-497.

[2] Z. Zheng, S. Xie, H. Dai, X. Chen and H. Wang, *"An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends,"* 2017 IEEE International Congress on Big Data (BigData Congress), Honolulu, HI, 2017, pp. 557-564.

[3] S. Chickerur, A. Goudar and A. Kinnerkar, *"Comparison of Relational Database with Document-Oriented Database (MongoDB) for Big Data Applications,"* 2015 8th International Conference on Advanced Software Engineering and Its Applications (ASEA), Jeju, 2015, pp. 41-47.

[4] M. H. Shih and J. M. Chang, *"Design and analysis of high performance crypt-NoSQL,"* 2017 IEEE Conference on Dependable and Secure Computing, Taipei, 2017, pp. 52-59.

[5] M. Ahmadian, *"Secure query processing in cloud NoSQL,"* 2017 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, 2017, pp. 90-93. doi: 10.1109/ICCE.2017.7889242

[6] L. Xu, L. Chen, Z. Gao, Y. Lu and W. Shi, *"CoC: Secure Supply Chain Management System Based on Public Ledger,"* 2017 26th International Conference on Computer Communication and Networks (ICCCN), Vancouver, BC, 2017, pp. 1-6.

[7] J. Garay, A. Kiayias, and N. Leonardos, *The bitcoin backbone protocol: Analysis and applications,* in Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, 2015, pp. 281310.

[8] https://www.r3.com/blog/2017/12/05/r3s-corda-becomes-one-of-the-first-dlt-platforms-available-on-aws-marketplace/

[9] Building Enterprise-Grade Blockchain Databases with MongoDB white paper, 2017. https://www.mongodb.com/white-papers

[10] A. Stanciu, "Blockchain Based Distributed Control System for Edge Computing," 2017 21st International Conference on Control Systems and Computer Science (CSCS), Bucharest, 2017, pp. 667-671.

[11] MongoDB Documentaion https://docs.mongodb.com/manual/tutorial/rotate-log-files/

[12] MongoDB Compass https://www.mongodb.com/products/compass

[13] PyMongo Documentation https://api.mongodb.com/python/current/
[14] Apache Jmeter https://jmeter.apache.org/
[15] https://docs.aws.amazon.com/quickstart/latest/mongodb/architecture.html