Cloud Computing : Increasing Trustworthiness and Accuracy of Cloud Transactions

V. Venkatesa Kumar
Assistant Professor
Dept. of Computer Science & Engg.
Anna University Regional Centre,
Coimbatore, India

Newlin Rajkumar
Assistant Professor
Dept. of Computer Science & Engg.
Anna University Regional Centre,
Coimbatore, India

N. Venkatakrishnan
PG Scholar
Dept. of Computer Science & Engg.
Anna University Regional Centre,
Coimbatore, India

***Abstract: O*ne of the primary goals of Cloud service providers is to provide assurance that the information being shared on cloud computing environment is secure. For transactions to be secure, we need to address various constraints from an end-user and Cloud service provider's point of view. The end-user is primarily concerned with the provider's security policy, how and where their data is stored and who has access to that data. On the other hand, concerns for the Cloud service provider can range from the physical security of the infrastructure and the access control mechanism of cloud assets, to the execution and maintenance of security policy. In this paper, we analyze the methodologies used to authorize users who access distributed database systems and the risks faced by these methodologies. To increase the trustworthiness of the transactions and also to ensure its accuracy, a combination of Two-Phase Validation Commit Protocol and Blow-fish algorithm is proposed. We analyze this approach through simulation method and the results are shared.**

Keywords – Cloud Computing, Secure Transactions, User Authorization, Validation Commit Protocol, Blow-fish Algorithm

I. INTRODUCTION

Cloud Computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services. Cloud service providers have the advantages of greatly simplified software installation and maintenance and centralized control over versioning; end users can access the service “anytime, anywhere”, share data and collaborate more easily, and keep their data stored safely in the infrastructure. From an economic perspective, end users can save huge IT capital investments and be charged on the basis of a pay-only-for-what-you-use pricing model.

To provide scalability and elasticity, cloud service provider’s often make heavy use of replication of data to ensure consistent performance and availability. A transaction, in the context of a database, is a logical unit that is independently executed for data retrieval or updates. Transactions are powerful abstractions that facilitate the structuring of database systems and distributed systems in a reliable manner.

In systems that host sensitive resources, accesses are protected via authorization policies that describe the conditions under which users should be permitted access to resources. These policies describe relationships between the system principles, as well as the certified credentials that users must provide to attest to their attributes. In a transactional database system that is deployed in a highly distributed and elastic system such as the cloud, policies would typically be replicated—very much like data— among multiple sites, often following the same weak or eventual consistency model. It therefore becomes possible for a policy-based authorization system to make unsafe decisions using stale policies.

Replications in Cloud are designed to ensure that replicated databases remain consistent, even in the case of hardware faults, server restarts, or network failures. Despite these efforts, however, it is possible that hardware failures (disk errors, memory errors) or software errors (causing memory corruption) might lead to inconsistent databases. The system may suffer from *policy inconsistencies* during policy updates and *user credential inconsistencies*.

In this paper, we address the inconsistency problem in transactional database systems which are deployed on the Cloud. We present the following information:

- Need for safe transactions that do not violate policy inconsistencies and user credential inconsistencies. We also provide details on the ACID properties that transactions have to meet
- Explanation of the approaches that guarantee the trustworthiness of transactions executing on cloud servers
- Proposal to use Two-Phase Validation Commit (2PVC) protocol that ensures that a transaction is safe by checking policy, credential, and data consistency during transaction execution and implementing Blowfish Algorithm for rectifying privacy issues
- Evaluation outcomes based on the approaches implemented on an experimental system.

II. PROBLEM DEFINITION

A. Database Replication

Database replication is the frequent electronic copying data from a database in one computer or server to a database in another so that all users share the same level of information. The result is a distributed database in which users can access data relevant to their tasks without interfering with the work of others. The implementation of database replication for the purpose of eliminating data ambiguity or inconsistency among users is known as normalization.

Database replication can be done in at least three different ways:

- Snapshot replication: Data on one server is simply copied to another server, or to another database on the same server.
- Merging replication: Data from two or more databases is combined into a single database.
- Transactional replication: Users receive full initial copies of the database and then receive periodic updates as data changes.

In the Cloud environment, a distributed database management system (DDBMS) ensures that changes, additions, and deletions performed on the data at any given location are automatically reflected in the data stored at all the other locations. Therefore, every user always sees data that is consistent with the data seen by all the other users.

B. Data Integrity

Data corruption can happen at any level of storage and with any type of media, So Integrity monitoring is essential in cloud storage which is critical for any data center. Data integrity is easily achieved in a standalone system with a single database. Data integrity in such a system is maintained via database constraints and transactions.

ACID (Atomicity, Consistency, Isolation, and Durability) is a set of properties that guarantee that database transactions are processed reliably.

Atomicity requires that each transaction be "all or nothing": if one part of the transaction fails, the entire transaction fails, and the database state is left unchanged. An atomic system must guarantee atomicity in each and every situation, including power failures, errors, and crashes. To the outside world, a committed transaction appears (by its effects on the database) to be indivisible ("atomic") and an aborted transaction does not happen.

The *consistency* property ensures that any transaction will bring the database from one valid state to another. Any data written to the database must be valid according to all defined rules, including constraints, cascades, triggers, and any combination thereof. This does not guarantee correctness of the transaction in all ways the application programmer might have wanted (that is the responsibility of application-level code) but merely that any programming errors cannot result in the violation of any defined rules.

The *isolation* property ensures that the concurrent execution of transactions result in a system state that would be obtained if transactions were executed serially, i.e. one after the other. Providing isolation is the main goal of concurrency control. Depending on concurrency control method, the effects of an incomplete transaction might not even be visible to another transaction.

Durability means that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors. In a relational database, for instance, once a group of SQL statements execute, the results need to be stored permanently (even if the database crashes immediately thereafter). To defend against power loss, transactions (or their effects) must be recorded in a non-volatile memory.

C. Problem Identification

Users interact with the system by submitting queries or update requests encapsulated in ACID transactions. Since transactions are executed over time, the state information of the credentials and the policies enforced by different servers are subject to changes at any time instance. A transaction is safe if it is a trusted transaction that also satisfies all data integrity constraints imposed by the database management system. A safe transaction is allowed to commit, while an unsafe transaction is forced to rollback.

The system may suffer from policy inconsistencies during policy updates due to the relaxed consistency model that allows data to be inconsistent among some replicas during the update process, but ensures that updates will eventually be propagated to all replicas underlying most cloud services. For example, it is possible for several versions of the policy to be observed at multiple sites within a single transaction, leading to inconsistent (and likely unsafe) access decisions during the transaction. Second, it is possible for external factors to cause user credential inconsistencies over the lifetime of a transaction. For instance, a user's login credentials could be invalidated or revoked after collection by the authorization server, but before the completion of the transaction.

D. System Model

To carry out the experimental aspects of the proposed approaches in the paper, we developed a banking system with following modules:

- Client Authentication
- Banking Services
- Coordinator Services
- Account Services
- Debit Process
- Funds Transaction
- Log Maintenance

In Client Authentication module, online customers must have access to a computer and a method of payment. In our

system, the user interactions are login, registration, communication, online payments and transaction. User details are handled in backend common database. In computer security, a login or logon is the process by which individual access to a computer system is controlled by identifying and authenticating the user referring to credentials presented by the user. A user can log in to a system to obtain access and can then log out or log off when the access is no longer needed. To log out is to close off one's access to a computer system after having previously logged in.

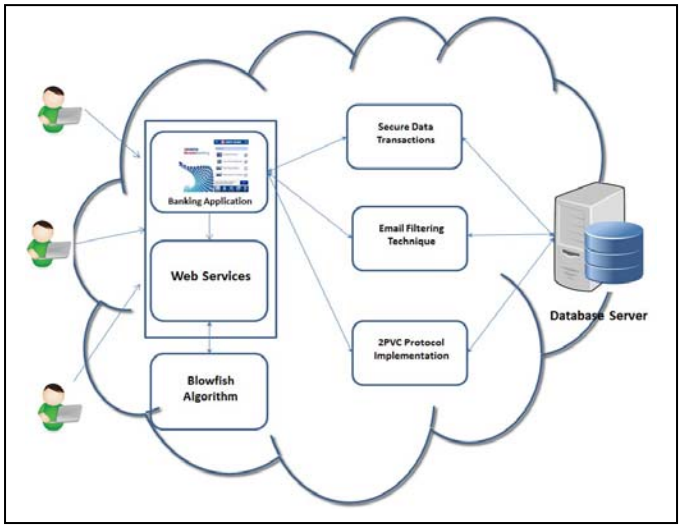

Figure 1. Simulation System

In Banking Services module, each and every funds transaction process proceeded through the banking services. In banking service constrain funds transaction support and verifying authorized person and so on.

On the Coordinator-side, the services comprises of:

- Activation service: The Activation service creates a Coordinator object and a transaction context for each transaction. Essentially, the Activation service haves like a factory object that creates Coordinator objects.
- Registration service: The Registration service allows the Participants and the Initiator to register their endpoint references.
- Completion service: The Completion service allows the Initiator to signal the start of the distributed commit.
- Coordinator service: The Coordinator service runs the 2PC protocol, which ensures atomic commitment of the distributed transaction.

Account service comprises:

- CompletionInitiator service. The CompletionInitiator service is used by the Coordinator to inform the Initiator of the final outcome of the transaction, as part of the Completion protocol.
- Participant service. The Participant service is used by the Coordinator to solicit votes from, and to send the transaction outcome to, the Participant.

A Debit Process is an electronic payment method that allows cardholders to access funds from their bank account at the point of sale using a debit card and a Personal Identification Number. The cardholder must enter a PIN to authorize payment of goods, and once the transaction is authorized, the money is debited from the cardholder's bank account and credited to the merchant. This service allows customers to securely purchase products and services using their existing bank account while allowing businesses to expand their online payment options beyond credit cards and significantly reduce their processing fees. Advantages of offering PIN based debit transactions:

- Reduced Processing Fees - Merchants are charged a flat fee for each on-line (PIN based) debit transaction instead of a percentage rate (discount fee) plus transaction fee.
- Reduced Fraud - Purchases are authorized at the time of sale, reducing the potential losses inherent with other types of payments, such as paper checks.
- Eliminates credit card chargebacks - PIN based debits are not subject to chargebacks.
- Transactions cannot be downgraded - PIN based debits are not subject to the downgrade fees that sometimes apply to credit card transactions that did not qualify for the best rate.

Funds Transaction facilitates the client to transfer the money from one account to the other. It includes crediting and debiting money in to and from the account by verifying the account number given by the user with decrypted account number from database.

The log maintenance consists of historical information that allow the admin to monitor the Existing information, Appending information to its Existing information, verifying new customers and acknowledging them. Report section allows the admin to generate report dynamically. It is usable for all other processes mentioned above since each and every process stored in a log format in our database.

III. IMPLEMENTING TRUSTWORTHY TRANSACTIONS

Processing a transaction often requires a sequence of operations that is subject to failure for a number of reasons. A trustworthy transaction satisfies the correctness properties of proofs of authorization and also satisfies the data integrity constraints. To enforce these properties and constraints, we propose to implement the following algorithms:

- Validation Based Concurrency Control Algorithm
- Two-Phase Commit Protocol Algorithm
- Blowfish Algorithm

We will now look at each of these algorithm details.

A. Validation Based Concurrency Control Algorithm

In optimistic concurrency control techniques, also known as validation or certification techniques, no checking is done while the transaction is executing. In this scheme, updates in the transaction are not applied directly to the database items until the transaction reaches its end. During transaction execution, all updates are applied to local copies of the data items that are kept for the transaction (Note 6). At the end of transaction execution, a validation phase checks whether any of the transaction's updates violate serializability. Certain information needed by the validation phase must be kept by the system. If serializability is not violated, the transaction is committed and the database is updated from the local copies; otherwise, the transaction is aborted and then restarted later.

There are three phases for this concurrency control protocol:

- Read phase: A transaction can read values of committed data items from the database. However, updates are applied only to local copies (versions) of the data items kept in the transaction workspace.
- Validation phase: Checking is performed to ensure that serializability will not be violated if the transaction updates are applied to the database.
- Write phase: If the validation phase is successful, the transaction updates are applied to the database; otherwise, the updates are discarded and the transaction is restarted.

The idea behind optimistic concurrency control is to do all the checks at once; hence, transaction execution proceeds with a minimum of overhead until the validation phase is reached. If there is little interference among transactions, most will be validated successfully. However, if there is much interference, many transactions that execute to completion will have their results discarded and must be restarted later. Under these circumstances, optimistic techniques do not work well. The techniques are called "optimistic" because they assume that little interference will occur and hence that there is no need to do checking during transaction execution.

The optimistic protocol we describe uses transaction timestamps and also requires that the write_sets and read_sets of the transactions be kept by the system. In addition, start and end times for some of the three phases need to be kept for each transaction. Recall that the write_set of a transaction is the set of items it writes, and the read_set is the set of items it reads. In the validation phase for transaction Ti, the protocol checks that Ti does not interfere with any committed transactions or with any other transactions currently in their validation phase. The validation phase for Ti checks that, for each such transaction Tj that is either committed or is in its validation phase, one of the following conditions holds:

- Transaction Tj completes its write phase before Ti starts its read phase.
- Ti starts its write phase after Tj completes its write phase, and the read_set of Ti has no items in common with the write_set of Tj.
- Both the read_set and write_set of Ti have no items in common with the write_set of Tj, and Tj completes its read phase before Ti completes its read phase.

When validating transaction Ti, the first condition is checked first for each transaction Tj, since (1) is the simplest condition to check. Only if condition (1) is false is condition (2) checked, and only if (2) is false is condition (3)—the most complex to evaluate—checked. If any one of these three conditions holds, there is no interference and Ti is validated successfully. If none of these three conditions holds, the validation of transaction Ti fails and it is aborted and restarted later because interference may have occurred.

B. Two-Phase Commit (2PC) Algorithm

The 2-phase commit (2PC) protocol is a distributed algorithm to ensure the consistent termination of a transaction in a distributed environment. Thus, via 2PC a unanimous decision is reached and enforced among multiple participating servers whether to commit or abort a given transaction, thereby guaranteeing atomicity. The protocol proceeds in two phases, namely the prepare (or voting) and the commit (or decision) phase, which explains the protocol's name.

The protocol is executed by a coordinator process, while the participating servers are called participants. When the transaction's initiator issues a request to commit the transaction, the coordinator starts the first phase of the 2PC protocol by querying—via prepare messages—all participants whether to abort or to commit the transaction. If all participants vote to commit then in the second phase the coordinator informs all participants to commit their share of the transaction by sending a commit message. Otherwise, the coordinator instructs all participants to abort their share of the transaction by sending an abort message. Appropriate log entries are written by coordinator as well as participants to enable restart procedures in case of failures.

As long as a transaction is still executing ordinary operations, coordinators as well as all participants operate in the Initial state. When the coordinator is requested to commit the transaction, it initiates the first phase of the 2PC protocol: To capture the state of the protocol's execution (which needs to be available in case of protocol restarts as explained below), the coordinator first forces a begin log entry, which includes a transaction identifier as well as a list of the transaction's participants, to a stable log. Afterwards, the coordinator sends a prepare message to every participant, enters the Collecting state and waits for replies.

Upon receiving a prepare message, a participant decides whether it is able to commit its share of the transaction. In either case, suitable log entries for later recovery operations as well as a prepared log entry indicating the vote ("Yes" or "No") are forced to a stable log, before a response message containing the vote is sent back to the coordinator. In case of a No-vote, the participant switches into the Aborted state and immediately aborts the transaction locally. In case of a Yes-vote, the participant moves into the Prepared state. In the latter case the participant is said to be in doubt or blocked as it has now given up its local autonomy and must await the final

decision from the coordinator in the second phase (in particular, locks cannot be released yet).

Once the coordinator has received all participants' response messages it starts the second phase of the 2PC protocol and decides how to complete the global transaction: The result is "Commit" if all participants voted to commit and "Abort" otherwise. The coordinator then forces a commit or aborts log entry to the stable log, sends a message containing the final decision to all participants, and enters the corresponding state (Committed or Aborted).

Upon receipt of the decision message, a participant commits or aborts the local changes of the transaction depending on the coordinator's decision and forces suitable log entries for later recovery as well as a commit or abort log entry to a stable log. Afterwards, it sends an acknowledgment message to the coordinator and enters the corresponding final state (Committed or Aborted).

Once the coordinator has received all acknowledgment messages it ends the protocol by writing an end log entry to a stable log to enable later log truncation and enters the final state, Forgotten. (For multiple participants, the actions simply have to be duplicated; in case of abort, at least one of the participants votes "No", which implies that all occurrences of "commit" are replaced with "abort".)

C. *Blowfish Algorithm*

Blowfish Algorithm is used in the data transformation process for Encryption and Decryption, respectively. The details and working of the algorithm are given below.

Blowfish is a symmetric block cipher that can be effectively used for encryption and safeguarding of data. It takes a variable-length key, from 32 bits to 448 bits, making it ideal for securing data. Blowfish was designed in 1993 by Bruce Schneier as a fast, free alternative to existing encryption algorithms. Blowfish is unpatented and license-free, and is available free for all uses.

Blowfish Algorithm is a Feistel Network, iterating a simple encryption function 16 times. The block size is 64 bits, and the key can be any length up to 448 bits. Although there is a complex initialization phase required before any encryption can take place, the actual encryption of data is very efficient on large microprocessors.

Blowfish is a variable-length key block cipher. It is suitable for applications where the key does not change often, like a communications link or an automatic file encryptor. It is significantly faster than most encryption algorithms when implemented on 32-bit microprocessors with large data caches.

In our proposed work, we combine the validation protocol and the 2PC protocol to form the 2 phase validation commit protocol (2PVC). 2PVC can be used to ensure the data and policy consistency requirements of safe transactions. Specifically, 2PVC will evaluate the policies and authorizations within the first, voting phase. That is, when the TM sends out a Prepare-to-Commit message for a transaction, the participant server has three values to report: 1) the YES or NO reply for the satisfaction of integrity constraints as in 2PC, 2) the TRUE or FALSE reply for the satisfaction of the proofs of authorizations as in 2PV, and 3) the version number of the policies used to build the proofs as in validation protocol.

IV. SIMULATION RESULTS

Simulation experiments were carried out to identify the performance of the proposed solution by calculating the "processing time" for different inputs. The simulation environment is given input sizes of 2kb, 5kb, 10kb, 20kb and 50kb. The running time before the implementation of the proposed solution and after implementation are captured. We also calculate the Speed-up ratio based on the running time. Speed-Up Ratio is defined as the ratio of mean processing time using the proposed solution to the mean processing prior to implementation of the solution. The environment was run multiple times with each input size and the mean value was used for calculations in each case.

Also, we carry out an analysis on the accuracy of the policy updates. 1,000 transactions were run multiple times to gather average statistics on the false rate with the updates. The randomized transactions were randomly composed of database reads and writes with equal probability. To simulate policy updates at different servers, the master policy server picks a random participating server to receive the updates.

The Mean Processing Time is calculated in milliseconds and the Input size is taken in kilobytes.

TABLE I. A COMPARISON OF MEAN PROCESSING TIME

Input Size	Processing Time (Before)	Processing Time (After)
2kb	540.2	610.2
5kb	580.6	700.9
10kb	620.3	785.8
20kb	748.2	942.1
50kb	935.4	1174.3

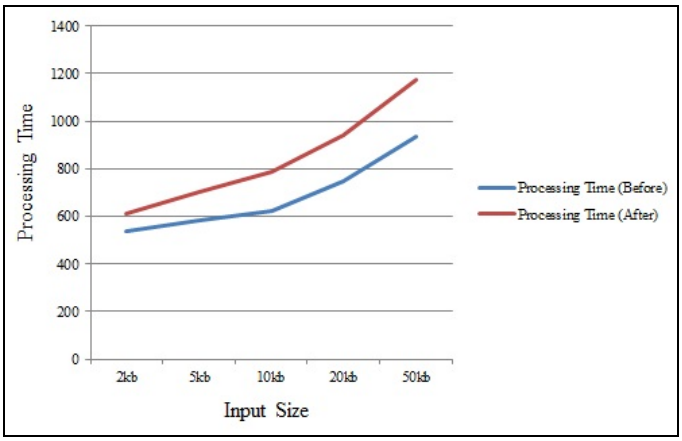

Figure 2. Comparison of the processing times

TABLE II. A COMPARISON OF FALSE RATES

Experiment #	False Rate (Before)	False Rate (After)
1	14	6
2	12	7
3	12	6
4	13	5
5	14	8

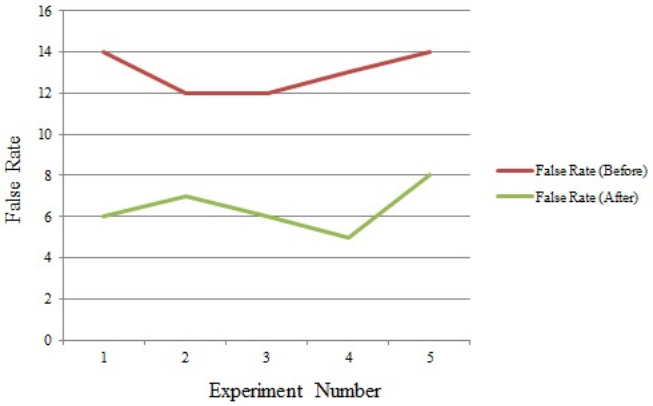

Figure 3. Comparison of the false rates

From the tabular results above, the following observations and inferences can be made. As the size of the input increases, the processing times are expected to increase. When we compare the 2 processing times, we notice that the processing time has slightly increased in the case of proposed solution. But when we look at the false rate, there is a drastic reduction in the number of incorrect policy updates.

V. CONCLUSIONS

Benefits of cloud computing are many as compared to the traditional computing. But more and more end users will move to Cloud environment only when they get the confidence that their data and transactions are secure. A combination of algorithms that will enforce consistency, accuracy and precision of the authorization policies that increases the trustworthiness of the transactions has been identified. An attempt has been made to determine if the proposed approach will guarantee safe transactions.

We have analyzed the Mean Processing Time of the proposed solution on the Cloud for different input sizes and we observed the variation in speedup ratio and mean processing time. With simulated workloads, we have experimentally evaluated the implementations relative to transaction processing performance and accuracy. We found that higher values for these metrics come at higher processing time and it is also dependent on implementing them correctly.

REFERENCES

[1] Marian K. Iskander, Tucker Trainor, Dave W. Wilkinson, Adam J. Lee, Member and Panos K. Chrysanthis, (2014) Balancing Performance, Accuracy, and Precision for Secure Cloud Transactions, 417-426

[2] Rabi Prasad Padhy, Manas Ranjan Patra and Suresh Chandra Satapathy, (2011) Cloud Computing: Security Issues and Research Challenges

[3] Bupesh Mansukhani and Tanveer A. Zia, (2011) An empirical study of challenges in managing the security in cloud computing

[4] Farhad Soleimanian Gharehchopogh and Meysam Bahari, (2013) Evaluation of the Data Security Methods in Cloud Computing Environments

[5] Martin Kuehnhausen, Victor S. Frost and Gary J. Minden, (2012) Framework for Assessing the Trustworthiness of Cloud Resources, 142-145

[6] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia, (2009) Above the Clouds: A Berkeley View of Cloud Computing

[7] Adam J. Lee and Marianne Winslett, Safety and Consistency in Policy Based Authorization Systems

[8] Dr. Rao Mikkilineni and Vijay Sarathy, (2009) Cloud Computing and the Lessons from the Past, 57-62

[9] Abdullah Abuhussein, Harkeerat Bedi and Sajjan Shiva, Evaluating Security and Privacy in Cloud Computing Services:A Stakeholder's Perspective

[10] Sang-Ho Na, Kyoung-Hun Kim and Eui-Nam Huh, (2013) A Methodology for Evaluating Cloud Computing Security Service-Level Agreements, 235-242

AUTHORS PROFILE

V. Venkatesa Kumar is working as Assistant Professor in Anna University Regional Centre, Coimbatore. He has PhD Computer Science & Engineering from Anna University, Coimbatore. He is an active member of various workshops and conferences at different levels. His areas of interest are Web Structure Mining, Data networks, Web Security.

Newlin Rajkumar is working as Assistant Professor in Anna University Regional Centre, Coimbatore. He has PhD Computer Science & Engineering from Anna University, Coimbatore. He is an active member of various workshops and conferences at different levels. His areas of interest are Computer networks and Digital Image Processing.

Venkatakrishnan is pursuing his Master in Engineering in Computer Science from Anna University Regional Centre, Coimbatore. His areas of interest includes cloud computing, software engineering, Network security, and Computer network.