Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control
Journal homepage: http://kinetik.umm.ac.id
ISSN: 2503-2267
Vol. 7, No. 1, February, Pp. 55-68

CICM: a collaborative integrity checking blockchain consensus mechanism for preserving the originality of data in the cloud for forensic investigation

Omoniyi Wale Salami*[1], Muhammad Bashir Abdulrazaq[2], Emmanuel Adewale Adedokun[3], Basira Yahaya[4]
Department of Computer Engineering, Ahmadu Bello University, Zaria, Nigeria[1,2,3,4]

Article Info
Keywords:
Forensic Investigation, Blockchain Consensus Mechanism, Agreement Process, Data Originality, Data Integrity

Article history:
Received: December 12, 2021
Accepted: January 15, 2022
Published: February 28, 2022

Cite:
O. W. Salami, M. B. . Abdulrazaq, E. A. Adedokun, and B. Yahaya, "CICM: A Collaborative Integrity Checking Blockchain Consensus Mechanism for Preserving the Originality of Data the Cloud for Forensic Investigation", *KINETIK*, vol. 7, no. 1, pp. 55-68, Feb. 2022.
https://doi.org/10.22219/kinetik.v7i1.1378

*Corresponding author.
Omoniyi Wale Salami
E-mail address:
salamiow@gmail.com

Abstract
The originality of data is very important to achieve correct results from its forensic analysis for resolving an issue. Data may be analyzed to resolve disputes or review issues by finding trends in the dataset that can give clues to the cause of the issue. Specially designed foolproof protection for data integrity is required for forensic purposes. Collaborative Integrity Checking Mechanism (CICM), for securing the chain-of-custody of data in a blockchain is proposed in this paper. Existing consensus mechanisms are fault-tolerant, allowing a threshold for faults. CICM avoids faults by using a transparent 100% agreement process for validating the originality of data in a blockchain. A group of agreement actors check and record the original status of data at its time of arrival. Acceptance is based on general agreement by all the participants in the consensus process. The solution was tested against practical byzantine fault tolerant (PBFT), Zyzzyva, and hybrid byzantine fault tolerant (hBFT) mechanisms for efficacy to yield correct results and operational performance costs. Binomial distribution was used to examine the CICM efficacy. CICM recorded zero probability of failure while the benchmarks recorded up to 8.44%. Throughput and latency were used to test its operational performance costs. The hBFT recorded the best performance among the benchmarks. CICM achieved 30.61% higher throughput and 21.47% lower latency than hBFT. In the robustness against faults tests, CICM performed better than hBFT with 16.5% higher throughput and 14.93% lower latency than the hBFT in the worst-case fault scenario.

1. Introduction

Digital forensic analysis is used to investigate the trend of activities on data on a digital device to reconstruct the events that cause an issue and establish required facts. Like other forensic investigations, it is usually required to verify claims made by parties in a dispute or to review historical activities to predict the future. Digital devices are of various varieties. They include handheld devices like mobile smartphones, tabletop systems, laptop computers, wireless sensors, etc. Digital technology is mostly used for information processing including its preparation, manipulation, and dissemination. The big role digital technology is playing in information systems, especially online information exchange has made it a key component in dispute resolution. But the fragile nature of digital data, e.g., wireless sensor data [1], may make it difficult to be extracted successfully in a way that will not hamper its successful use to generate genuine information. This is because it can be easily damaged by improper handling, unethical inspection method [2], or an attack on the network [3]. Manipulations, made deliberately or by mistake, that alters the state or interpretation of data can render it unacceptable [4] for analysis. Keeping vital records of digital data and its metadata before its properties change is very essential to preserve its originality for future requirements. The solution proposed in this paper will use blockchain technology for ensuring prompt preservation of data properties as early as possible. This will prevent loss or manipulation of the features that may change its information contents.

Blockchains use a set of cryptographic algorithms to implement consensus mechanisms for enforcing trust and transparency [5] where a high level of trust among transacting parties is very essential. Blockchain technology is suitable for preserving trust among stakeholders in the chain of custody of digital data. It has been employed for different data security purposes. The security and fault tolerance of a blockchain depends highly on the effectiveness of its consensus mechanism [6]. Various consensus mechanisms are available which are used for different blockchain applications [7]. Each consensus mechanism has its preferential benefits and performance limitations because of the threat model and assumptions on which its design was based. Forensic analysis of data relies highly on the accuracy of data being analyzed to be able to trace the history of the data correctly and extract necessary trends of activities needed to infer the desired information. Existing consensus mechanisms do not have adequate facilities for capturing and preserving such accurate records of data attributes before incidence response as needed for digital forensic purposes. It is essential to preserve the forensic soundness of the originality of data from the time of its creation to avoid manipulating it after creation before incidence response. A new consensus mechanism that is suitable for mining forensically sound data attributes is proposed in this paper. The consensus mechanism provides a protocol for collaborative verification of data attributes at the time of creation on the storage device. It uses a 100% agreement by all participants in the consensus process for foolproof preservation of data integrity as suggested in [8]. The mechanism is called *Collaborative Integrity Checking Mechanism* (CICM). It uses elliptical curve cryptography (ECC) algorithms for security and cryptoanalysis. ECC is a key-based cryptography technique that uses points on an elliptic curve to create security parameters. It uses a smaller key to achieve higher security than other security systems like RSA [9]. CICM also verifies the responses of the participants in an agreement process with a consistency history token. The consistency token is unique for every participant. Blockchain was used in the proposed solution for keeping the records of data attributes as at the time it was created to make it immutable.

A new transparent collaboration method was used for the agreement process of CICM. The collaboration method enhanced better synchrony among participants in a consensus and highly reduced message overhead and protocols. It also introduced a new consensus mechanism with 100% agreement suitable for preserving data integrity for data analysis that requires accurate data integrity. A novel method was used for computing the consistency history of the agreement actors by the CICM.

Throughput and latency were used to analyze the operational costs of CICM as used in [10][11]. The binomial distribution used to prove the security of the solution proposed in [12] was used to analyze CICM efficiency. The results were compared with those of practical byzantine fault tolerant (PBFT) [11], Zyzzyva [13], and hybrid byzantine fault tolerant (hBFT) [10] that were used as benchmarks.

1.1 Related Works

Erbacher [4] identified important areas of forensic investigation where validation is very important to make forensic analysis reports valid. Among the significant areas identified are data generation, data collection, and data storage. The forensic validity requirements for these areas emphasize the originality of data as opposed to the typical security concerns that focus on the protection of privacy and other interests of the user in the information. Blockchain was proposed by Nakamoto [14] purposely for ensuring the validity of transactions conducted with cryptocurrency. It was first used for Bitcoin. It possesses necessary features for preserving transactions authenticity which has made it a technology of choice for applications requiring validity. Blockchain is a computer security tool that is adequate for guaranteeing transparency, authenticity, and audit of digital records [15].

The benefits of decentralized auditing and verifiable cryptography in blockchain as well as using blockchain's smart contract to automate time-consuming processes in IoT were investigated in [16]. It was discovered that employing those features of blockchain can enhance achieving significant time and cost savings. Yan *et. al.* [12] proposed a solution for securing the chain of custody of digital forensic evidence using blockchain technology to ensure traceability and preservation of the integrity of the digital evidence data [12]. Blockchain technology was also used for monitoring deliveries in the supply chain of petroleum products from depots to retail outlets [17].

Forensic-Chain [15] was built on top of Hyperledger. It is a permissioned blockchain solution for recording evidence during the digital forensic investigation process. CrowdBC [18] employed blockchain for securing crowdsourcing systems. CrowdBC conceptualized the use of blockchain to decentralize crowdsourcing storage and ensure fairness between parties in crowdsourcing transactions. Consensus mechanisms provide coherence protocols to a group of machines that are working collectively such that they can yield correct results even if some of its group members failed. Consensus mechanisms are an important component of reliable large-scale systems. Consensus protocols are used to solve byzantine and crash faults. *Validating actor*s in a distributed blockchain network use consensus mechanisms for reaching an agreement on the validity of a transaction to be accepted into the blockchain ledger [19]. It is a fundamental module in blockchain that provides tamper-free protection and also ensures all the actors agreed upon the same version of a value [5].

The main types of consensus algorithms are Byzantine Fault Tolerant (BFT) and Crash Fault Tolerant (CFT). Byzantine faults are failures that make a system yield wrong results because of wrong inputs [20]. Crash faults are caused by defects in the device that is executing a process [21][22]. Several consensus mechanisms available are developed as variants of these types. The classical BFT mechanism [23] is a consensus algorithm based on the byzantine generals' problem. It uses collective decision, called the agreement, in a distributed network to safeguard against system failures by reducing the influence of the faulty nodes. The agreement process ensures that all nodes implement the same instructions. It is designed to be resilient against some faulty nodes that are much lower in number than the quorum required for consensus. BFT algorithm tolerates up to n faulty nodes existing among 3n+1 nodes to work correctly, but the process is expensive both in terms of cost and effort [23].

Different consensus algorithms have been proposed based on the BFT principle intended to improve the BFT algorithm and make it applicable in other use cases. Among BFT consensus mechanisms is Practical Byzantine Fault Tolerance (PBFT) [11]. PBFT uses views to represent the period a primary coordinates a consensus process. Uncommitted requests are discarded during checkpoints when a view is changed. PBFT reduced quorum to 2f+1, where *f* is the number of faulty nodes. It introduced a view protocol for removing a faulty primary and checkpoint protocol for clearing old data that was pending in the view of a changed primary to synchronize system states in the previous view and the new view. Zyzzyva [13] is another consensus protocol that uses speculation to improve the cost of replication better than it was in the classical BFT mechanism. It was developed upon PBFT protocol. Zyzzyva does not discard uncommitted requests from the old view when transiting into a new view but retains them if they were not earlier than any request that had been committed. Zyzzyva uses speculation to agree on request and sends it to client. If the client detects faults it uses commit certificate to make loyal replicas that sent replies to commit to their replies. Zyzzyva recognized the possibility of an untrusted primary that can exploit speculative agreement protocol to prolong the commit process up to twice as may be necessary. Hybrid byzantine fault tolerance (hBFT) [10] proposed improvements to Zyzzyva and PBFT. Like Zyzzyva, hBFT is a speculative Byzantine fault-tolerant mechanism with minimum cost. It shifted some critical jobs to clients to reduce message complexity. It also tolerates an unlimited number of faults. hBFT uses a three-phase checkpoint protocol to resolve inconsistencies when replicas are out of order in speculative execution or when a primary triggers checkpoint through the panic system, and also to detect faulty clients' behaviour if they intentionally triggered checkpoint protocol. Moving critical jobs to clients introduced another vulnerability which resolving it could cause additional overhead. Stellar consensus protocol (SCP) [24] is another consensus mechanism of BFT class. It introduces federated byzantine agreement (FBA) and quorum slices. FBA allow validating peers to prioritize response of particular peers they consider important and wait for those prioritized responses before conceding to accepted values. FBA can make some participants not accept a correct value when their important peers have not agreed to it which may increase time cost for an agreement process.

The CFT consensus mechanisms are different from BFT based on their threat and consensus models. Two example mechanisms that are crash fault mechanisms are the Paxos [25] and the Raft [26]. Paxos is a CFT consensus mechanism in which acceptors monitor the order of messages to avoid accepting contradictory messages or accepting a message more than once. Paxos was considered to be complex and difficult to understand, so Raft [26] was proposed to address understandability and complexity issues in Paxos. Raft incorporated leadership election in the agreement process to reduce its protocols. It also uses joint consensus to address membership change issues and a reconfiguration algorithm to keep normal processing of requests uninterrupted during membership changes.

Performance of network protocols like consensus algorithms is often measured with throughput and latency. Throughput is a metric used to determine the number of completed tasks within a specific period. It can be used to determine the computational costs of an algorithm. Latency determines the time taken to complete a process. Throughput and latency are important metrics for determining the performance of a consensus algorithm to show if the algorithm will be able to meet up with the possible rate of arrival of values for validation in real-time. The metrics were used in [10][11] to test the scalability of their algorithm as the faults and users increased. The binomial distribution provides a means to view the trend of the probability of outcomes in a specific number of attempts. It was used to prove the security of the proposed consensus in [12].

2. Research Method

CICM is a partial leader-based consensus mechanism, it does not operate on views and does not change leaders. There is no leader election as is done in some consensus mechanisms. CICM receives messages from its cluster controller in a cloud network and sends a reply to a shared drive on its *evidence server*. Some mechanisms receive values from a leader/primary and send a reply to the client. The *agreement actor*s in CICM receive a message in one-to-many and reply in many to one collaboration file for agreements as opposed to the case wherein the agreement messages are sent many-to-many in some mechanisms. The rounds of message exchanges in CICM are 3 for fault-free but not more than 8 depending on the fault. There are mechanisms where the message exchanges are far more than the worst case of CICM because of their many protocols and protocol messages. The occurrence of faults that may be caused by compromised nodes is rare in CICM because of the transparency provided by shared buckets for collaboration where every *agreement actor* sees the same data as it arrived. The transparency is comparable to a presentation that is projected onto a screen where every audience can see it and each audience also has a copy of it opened on her/his digital device. Any disparity will be easily detected by all. In most other mechanisms the faults can only be seen and reported by replicas individually or by the client. In CICM, an *agreement actor* whose response is wrong may even see it and correct it before submitting its reply. CICM protects users' data from being modified after it left the users' hands. Other mechanisms ensure correct results of execution before being written to a blockchain but do not ensure its originality afterwards.

The components of the CICM and the methodology of its experimental implementation are explained in the following.

2.1 The Proposed Collaborative Integrity Checking Mechanism Model

The Collaborative Integrity Checking Mechanism, CICM, is a blockchain consensus algorithm for capturing attributes of data and storing it in a blockchain. It is a consensus mechanism specifically designed to preserve the soundness of digital data in the cloud for future forensic investigation requirements. It consists of a group of nodes called "*validating actor*s" that are involved in the agreement process. The *validating actor*s include the *edge node*, the replica nodes, the evidence server, and an optional artificial intelligence (AI) machine. The *edge node* may be a switch/router, cloud controller, *cluster controller*/node controller, depending on the type of network. It is the entrance through which messages from users enter the network. Replica nodes, the evidence server, and the AI machine that are involved in the agreement process are called "*agreement actors*". The *edge node* only shares the incoming messages with the *agreement actor*s. Every *agreement actor* has the *ID* and address of the *edge node*, and the *ID*s, signatures, addresses, and *consistency history* (CH) of other *agreement actor*s in a list arranged in the same order as the lists held by the other *agreement actor*s. Figure 1 shows interconnections between the CICM machines.

Collaboration service is now being used for increasing productivity. There are several collaboration tools and services available nowadays. Google Workspace and Microsoft SharePoint are among the popular collaboration services available. CICM uses a shared drive for collaboration to improve its operations. The evidence server hosts a shared drive where items for collaboration by the *agreement actor*s are located. The items on the shared drive are the main blockchain ledger, an auxiliary blockchain ledger, and 3 shared buckets namely; *message-hash bucket*, *vote bucket*, and *review bucket*. The items on the shared drive are accessed exclusively by the *agreement actor*s only. Each of the *agreement actor*s has read/write access to them. Except for the *agreement-hash* bucket to which the *cluster controller* also has read/write access. CICM neither tolerates any compromised nodes nor wrong responses. Acceptance is by 100% agreement by all the responding actors. This is in contrast to other agreement-based consensus mechanisms that allow a threshold for the number of untrusted nodes that may present wrong responses to requests, CICM takes into cognizance the peculiarities of digital forensic investigations which are premised on the possibility of extraction of genuine information. If information is falsified and the falsification is genuinely detected by the investigation it is acceptable for inclusion in the investigation report. The significance of the protection provided by the proposed CICM is post-execution because it preserves the originality of the data produced by execution processes. CICM model involves message validation by multiple actors. Every *agreement actor* in the trust model uses the same method as used by other actors for verifying transaction integrity. There is no election of leaders for the consensus protocol of this mechanism because there are nodes among the main actors in this mechanism, i.e., the evidence server and cluster controller, whose basic functions are similar to the roles played by leaders in other consensus mechanisms. A crypto-hash algorithm is used for authenticating the status of messages by *agreement actor*s. *Consistency history* is used to monitor the performance of a *validating actor*. Signature and ID are used to validate a *validating actor*'s submission or response. The process of the message inspection and the message record mining into the distributed blockchain ledger is explained in the following. All communications between *agreement actor*s and the *edge node*, in this case, a cluster controller, pass through a virtual private network (VPN) secured channel that connects the *validating actor*s as shown in Figure 1.

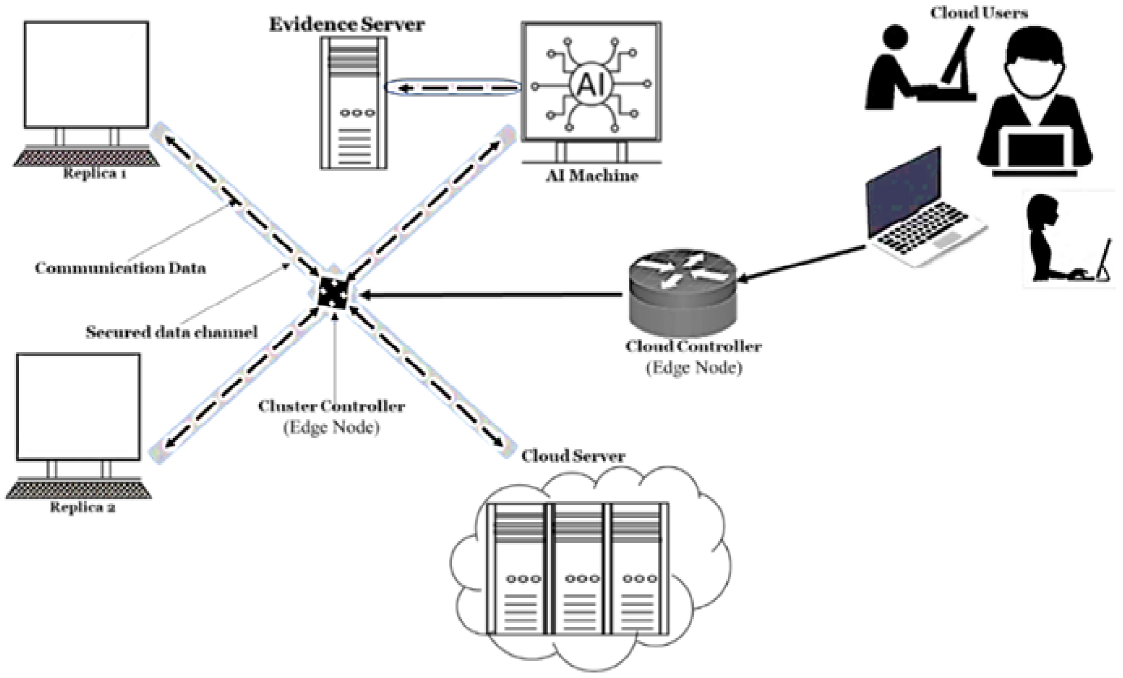


Figure 1. Interconnections and Communication between CICM Components

2.2 The CICM Protocol

CICM is a non-quorum, fault-free consensus protocol with unelected partial leaders. The nodes that perform functions similar to those performed by leaders in other consensus mechanisms are the edge node and the evidence server. The edge node provides the value to be validated to the *agreement actor*s as done by the primary [10]. The evidence server provides a collaboration platform where *agreement actor*s submit their responses for collaborative review similar to a leadership role described in [27]. They are not elected and are not recognized as leaders because they do not control the consensus process in any way. It is only their roles that are similar to some roles of a leader in a consensus mechanism. This is why they are referred to as unelected partial leaders. The protocol allows only active members of the *agreement actor*s to participate in an agreement process. It rejects byzantine faults because it does not tolerate traitors. Thus, it does not accept incorrect responses from any of the participants into the main blockchain. The mathematical function for modelling the CICM Protocol can be given as;

$$f\left(\boldsymbol{H}^{S}_{VOTE_J}\right) = \begin{cases} 1, \Leftrightarrow \forall \boldsymbol{H}^{S}_{VOTE_J} \in \boldsymbol{H}^{S}_{VOTE}: \boldsymbol{H}^{S}_{VOTE_J} = 1 \\ 0, \Rightarrow \exists \boldsymbol{H}^{S}_{VOTE_J} \in \boldsymbol{H}^{S}_{VOTE}: \boldsymbol{H}^{S}_{VOTE_{J\geq 1}} \neq 1 \end{cases} \quad (1)$$

In Equation 1, $\mathbf{H}^{S}_{\mathrm{VOTE}_J}$ is the individual signed response from each actor J, $\mathbf{H}^{S}_{\mathrm{VOTE}}$ are all the signed votes submitted by the participants. There is consensus if and only if all participants responses agreed on the value. Otherwise, if one or more participants' responses disagreed with others there is no consensus. The proof using probability theory to show that Equation 1 can provide bases for the high level of accuracy required for digital data forensic analysis results is given in the following.

2.3 Proof of Efficacy of CICM

The proof is based on the following rule of probability theory;

1) The sum of the probabilities of all possible outcomes is 1.
   The rule can be mathematically modelled for probabilities of outcomes A, B to N as;

$$P_{AB\ldots N} = P_A + P_B + \cdots + P_N = 1 \quad (2)$$

In Equation 2, the $P_{AB\ldots N}$ is the sum of probabilities of outcomes A, B, …, N, P_A, P_B and P_N are the probabilities of the individual outcomes A, B, …, N respectively.

The binomial distribution is used to examine the possible rate of producing wrong results by the CICM consensus algorithm in the following.

The formula for binomial distribution is;

$$P(x) = \frac{n!}{(n-x)! \times x!} \times P^{x} \times (1-c)^{(n-x)} \quad (3)$$

The formula for binomial distribution is given in Equation 3. The $P(x)$ is the probability of getting a total number of x outcomes, P is the probability of getting the outcome in one attempt, n is the number of attempts made.

Using Equation 3, the probability for the CICM to yield incorrect results x times in n attempts is calculated. Since CICM does not tolerate traitors, $P = 0$. If there would be a failure x cannot be 0, thus $x > 0$. Therefore,

$$P(x) = \frac{n!}{(n-x)! \times x!} \times 0^{x} \times (1-0)^{(n-x)} = 0$$
$$(n,\ x) > 0 \quad (4)$$

The results of Equation 4 will always be 0 for all values of n and x greater than 0. CICM has a probability of 0 to give incorrect results for values of n and x greater than 0. According to the rule of probability stated earlier, the sum of the probabilities of CICM giving incorrect results and CICM giving correct results is 1. Thus, the probability that CICM will give correct results x times in n attempts is 1 - $P(x)$, which is 1 – 0 = 1.

Equation 4 has mathematically shown that CICM can yield the high level of accuracy required for forensic analysis results from digital data which is what was aimed to achieve. The mathematical test presented here for proving the efficacy of CICM was carried out on the hBFT, PBFT, and Zyzzyva that were used as benchmark solutions as presented later.

CICM protocol records the status of data including the instances of faults that may affect the results of data analysis. The correct attributes of data are put in the main blockchain while records of both the correct and the faulty replies in a validation process where faults are discovered are put in the auxiliary blockchain. The purpose is to provide adequate accurate information about the data to an analyst/investigator that may need to extract information from the data in the future.

The components of the CICM protocol consist of (1) Registration, (2) Agreement, (3) Crash-Check, (4) Authentication.

2.4 Registration

Each member of the *agreement actor*s group registers with the cluster controller and with every other member of the group. A list of the identification certificates of other members that registered with each node is stored in a particular order agreed by the participants and the same with every participant. The *agreement actor*s group members' certificates contain the node ID, IP address, consistency history, and public key. A shared drive that hosts the main blockchain, auxiliary blockchain, and the 3 shared buckets; *message-hash bucket*, *vote bucket*, and *review bucket* is available on the evidence server for collaboration by *agreement actor*s. The first bucket, called the *message-hash bucket*, is used for submitting the message hash computed by each *agreement actor*. The second bucket, which is the *vote bucket*, is used for agreement voting. The third bucket, called the *review bucket*, is used for reviewing the message from nodes that computed a wrong message hash. All members of the *validating actor* group have *read/write access* on the *message-hash bucket*, but only the *agreement actors* have *read/write access* on the vote and review buckets. The *agreement actors* also have *Read-only* access to the main and auxiliary blockchain ledgers. Registration is done once but items on the certificate may change as may be necessary. Since the ECC algorithm is used the public key may be changed as may be agreed by the participants. Also, the consistency history token changes at the end of every successful agreement process.

The routing update setting should be configured to update routing tables of the actors frequently enough to minimize broken link challenges and ensure actors are always connected. The participating nodes should periodically check the links on their routing table by exchanging routing information periodically to make sure they are not down.

2.5 Agreement Algorithm

The agreement algorithm checks the properties of the message when it was received on the cloud and accurately records it in the appropriate ledger. The stages involve in a normal agreement process when all actors received the correct copy of the message is illustrated in Figure 2. A cloud user sends a message to the cloud (Figure 2: A). The cloud controller shares a copy with the *cluster controller* as **<msg, msg-id, timestamp>$_{cc}$**. The *cluster controller* forwards **<msg, msg-id, timestamp>$_{CC}$** to the *agreement actor*s and drops a copy in the *message-hash bucket* (Figure 2: A-B). The **msg** is the original message the user sent to the cloud, the **msg-id** is the ID of the message on the cloud, cc indicates that the forwarded message originated from the cloud controller. The *agreement actor*s compute and sign the hash of the message as $H(m)^S$, and the signed hash of the message metadata as $H(meta)^S$. The signed hash of the message and its metadata together with the *agreement actor*'s identity ID, its consistency history up to the last consensus process conducted before the present one, CH_{n-1}, and the nonce, N_{n-1}, that generated the CH_{n-1} are composed as **<$H(m)^S$, $H(meta)^S$, ID, CH_{n-1}, N_{n-1}>**. Each actor will send the details as **<$H(m)^S$, $H(meta)^S$, ID, CH_{n-1}, N_{n-1}>** to the *message-hash bucket* (Figure 2: B-C).

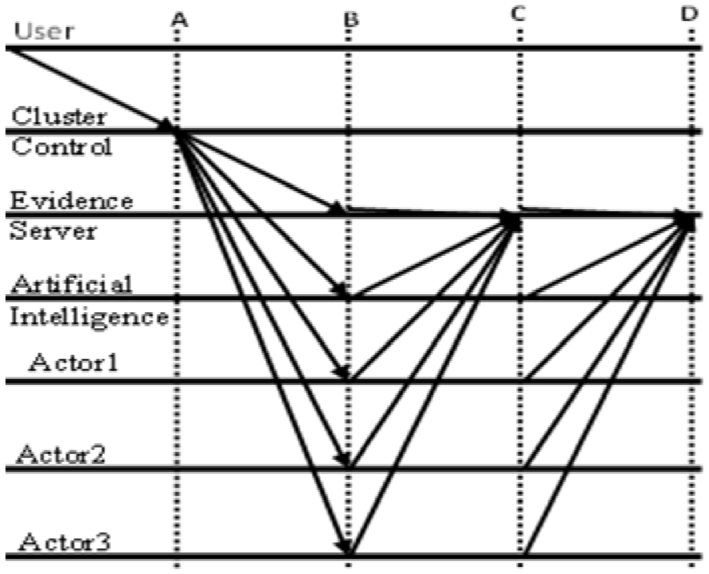


Figure 2. Agreement Process with No Faults

The algorithm for this stage is presented in Figure 3 and Figure 4. Figure 3 shows the pseudocode for the cluster controller role when forwarding user messages from the cloud controller to the agreement actors. Figure 4 contains the pseudocode for the message attributes scrutinization and recording by the agreement actors.

```
**Initialization:**
msg = original user message
msag_id = message ID on cloud
timestamp = cloud timestamp
On event <msg, msg-id, timestamp>cc Received
Forward <msg, msg-id, timestamp>cc to { Agreement Actors}
Put <msg, msg-id, timestamp>cc in message-hash bucket
```

Figure 3. The algorithm for Cluster Controller activies

```
Initialization:
ID = this actor's ID
$CH_{N-1}$ = CH from last session
$N_{n-1}$ = Nonce for $CH_{N-1}$
$RESP_{INDEX}$ = “'                    ////Serial number of an Actors response in the bucket
$H_x$ = “”                             ////Unsigned computed Hash of X
$H_x^S$ = “”                           ////Signed  computed Hash of X
Hash(.)                                ////Hash function
On event <msg, msg-id, timestamp>$_{CC}$ Received from cluster controller
$H_{msg}$ ← Hash(msg)
$H_{meta}$ ← Hash(meta)
Put <$H_{msg}^S$, $H_{meta}^S$, ID, $CH_{n-1}$, $N_{n-1}$> in message-hash bucket
$RESP_{INDEX}$ = INDEX + 1
Copy $RESP_{INDEX}$
```

Figure 4. The Algorithm for Agreement Actors Message Attributes Checking Process

CH_{n-1}, which is the CH for the last consensus conducted that all other actors have a copy of it, is used at this stage. The CH for the current consensus process, CH_n, will be computed at the end of this process using the final hash submitted for agreement.

Each *agreement actor* has the *consistency history* of other actors up to CH_{n-1} and can confirm CH_n with the message hash H(m) and the nonce N. ID and signature are used to authenticate the sender. When an *agreement actor* computed its response as explained it will put it in the *message-hash bucket* on the shared drive on the evidence server. It will copy the serial number of its response in the *message-hash bucket*, i.e., the element number/index of its response in the *message-hash bucket*. All *agreement actor*s are monitoring the addition of responses into the *message-hash bucket* and making a copy of it as it progresses. They know the number of members in the group, so when the last response is put in the *message-hash bucket*, they all know that the process has been completed. The *agreement actor*s would have gotten complete copies of the contents of the shared *message-hash bucket* to themselves by the time the last response was dropped in. They then individually authenticate each response with the identity certificate of the node that submitted it and also check if all responses give the correct hash of the message. Each *agreement actor* that confirmed that all responses sent gave the same correct hash of the message will generate and sign a hash of all the responses it copied from the shared *message-hash bucket* and inspected as $H(R)^S$. It will use the $H(R)^S$ as its vote for agreement and submit it to the shared *vote bucket* (Figure 2: C-D). When the voting for agreement is completed the original text of the message will be discarded and the accepted record of the message will be entered into the appropriate blockchain. This stage is executed with the pseudocode in Figure 5. The pseudocode in Figure 5 checks that every response from agreement actors comply with each other and agreed on the value. If there is anyone that does not comply the actor that sent the response that differs will be requested to correct the error. They recheck the submitted responses for compliance when voting is completed and compute consistency history tokens if everything is okay.

At the end of the integrity checking process, when the responses from the actors have been admitted, each actor will use the H(R) it submitted as its vote for agreement to compute CH as follows. It is computed as a signed hash, $H()^s$, using the previous CH_{n-1} and the H(R) in $\mathbf{H(CH_n)^S = <H(CH_{n-1}, H(R))>}$. The computed CH must have the hexadecimal digits indicating the number of times the particular actor's responses complied with the correct original message hash. The hexadecimal count must be at a position in the CH characters which is equivalent to the actor's index in the *agreement actor*s ID list. Example, counting from zero, an actor located at index 3 in the ID list who has had 158_{10} (equivalent to 0x9E or $9E_{16}$) correct responses will compute its CH as **<H(CH$_{n-1}$,** H(R)**)> = xxx9Exxxxxxxxxxxxxxxxxxxxxxxxxxxx** if SHA256 was used. The CH computation is implemented in the loop at the bottom of Figure 5.

```
Initialization:

Nonce = 0

H = {all submitted message hash from other actors}
$ID_J$ = This Actor's ID                                   ////This Actor is Actor J
$H_O$ = Message Hash from another actor                     ////Each of other Actor is Actor O
$CH_J$ = Consistency history of Actor J for present session
$ID_{INDEX,J}$ = Index of $ID_J$ on the list of Actors IDs
$H^S_{VOTE,J}$ = signed  vote from this Actor
$H^S_{VOTE}$ = {submitted signed votes in the vote-bucket}
$CH_{count,J}$ =    total replies from this actor that are consistent with others
On event message-hash bucket changed
If ($H \neq H_O$) send <"Request", review-bucket ← {(msg, msg-id)$_O$, ID$_O$, (CH$_{n-1}$, N$_{n-1}$)$_O$} > to Actor O
On event <(msg, msg-id)$_O$, ID$_O$, (CH$_{n-1}$, N$_{n-1}$)$_O$> Received from Actor O in  review-bucket
Each Actor Do
    Compare (msg, msg-id)$_O$ with (msg, msg-id)$_{cc}$
    If((msg-id)$_O$ ≡ (msg-id)$_{cc}$ Then
        If((msg)$_O$ ≡ (msg)$_{cc}$ or (msg)$_O$ ≈ (msg)$_{cc}$) send <"Request", Recompute> to O
        Record Transaction in Main Blockchain
        Else Blacklist O and Broadcast <"O suspected"> to other Actors
        Record Transaction in Auxilliary Blockchain
        End If
    Else  Blacklist O and Broadcast <"O suspected"> to other Actors
    Record Transaction in Auxilliary Blockchain
    End If
    If (Response Complete) Copy review bucket contents
    $H_{VOTE,J}$ ← Hash(review bucket contents)
    Put $H^S_{VOTE,J}$ in vote bucket
    On event vote bucket change
      If ($H^S_{VOTE} \neq H^S_{VOTE,O}$) send <"Request", Recompute> to O
    On vote complete
      If({$H^S_{VOTE}$} ≡ {$H^S_{VOTE,J}$ }) Do
      CH ← Hash(<H(CH$_{n-1}$, H(R))$^S$, Nonce>)
      For i = 0, i < length(CH), increment i
          If ( i = $ID_{INDEX,J}$ AND CH ∋ $0xCH_{count}$ @ $i$) Exit Do
      End For
      Nonce ← Nonce + 1
      End Do
    Broadcast CH
    End If
End Each Actor Do
```

Figure 5. The Algorithm for Comparing Agreement Actors Message Attributes Checking Results and Concluding the Agreement Process

The consistency history token is unique for the participant because the serial number of its ID on the list is indicated in it. It is linked to other participants' CH because the H(R) is the same for all participants that submitted the correct response in the session.

2.6 The Possible Conditions for Faults in CICM and their Possible Solutions

Some conditions that could cause errors in the operation protocols of CICM and the possible solution for resolving them are presented in the following. Figure 6 is used to explain how different errors are resolved and where the process will end in the session.

2.6.1 Incorrect H(m) Computed by one or more Actors

The actor that submitted a wrong message hash (Figure 6: Actor2, dash arrow, B-C) will be requested to submit the message it generated the hash from to the shared *review bucket* (Figure 6: C-D). *Agreement actors* that submitted the correct message hash will do a bit-wise comparison of the message that generated a different hash with their own that generated correct hash values (Figure 6: D-E). They will submit their votes for the correct responses into the *vote bucket* when submitting comparison results for the wrong message into the *review bucket* for all to see ((Figure 6: E-F).

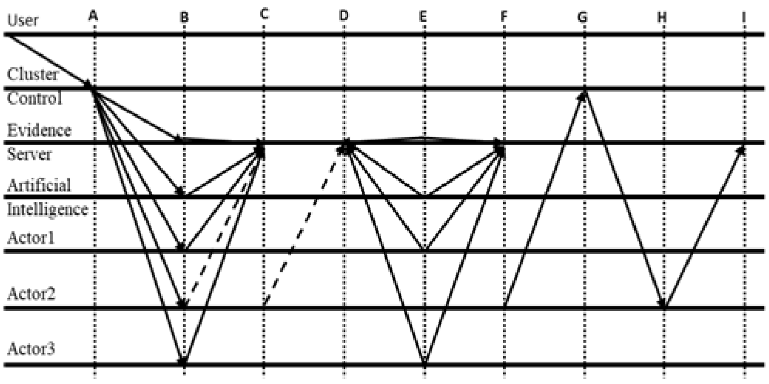
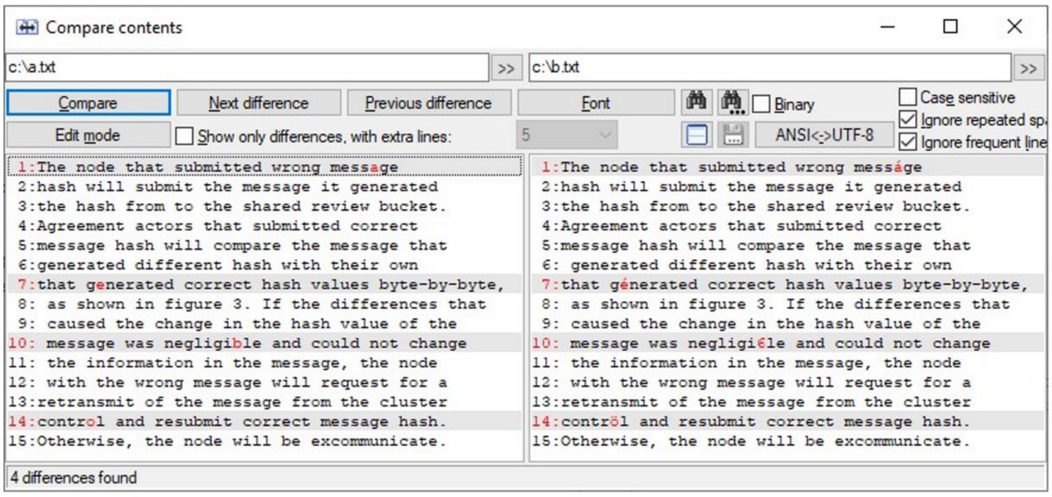


Figure 6. Agreement Process, a Mismatch from Actor2

If the differences that caused the change in the hash value of the message was negligible and could not change the information contained in the message (as shown in Figure 7), the node with the wrong message will use the message ID to request *cluster controller* to retransmit the message to it (Figure 6: F-G) when it received the retransmitted message from cluster controller (Figure 6: G-H), it will recompute a correct message hash and resubmit it (Figure 6: H-I). If the resubmitted corrected message hash is correct, it will be acknowledged to be okay and accepted into the main blockchain. Otherwise, if the corrected message hash resubmitted was still wrong or the modification found in the message in step Figure 6: E-F changes the information content of the message, the node will be excommunicated and the administrator will be alerted of the problem. Records of such faulty replies and the correct replies for the particular validation session are put in the auxiliary blockchain.

In Figure 7, the characters that changed may not make the message incomprehensible and a reader may still be able to get the correct information from the message. But if it were words that changed, then the information contained in the message might change as well.



```
 1:The node that submitted wrong message           1:The node that submitted wrong messáge
 2:hash will submit the message it generated       2:hash will submit the message it generated
 3:the hash from to the shared review bucket.      3:the hash from to the shared review bucket.
 4:Agreement actors that submitted correct         4:Agreement actors that submitted correct
 5:message hash will compare the message that      5:message hash will compare the message that
 6:generated different hash with their own         6: generated different hash with their own
 7:that generated correct hash values byte-by-byte,  7:that génerated correct hash values byte-by-byte,
 8: as shown in figure 3. If the differences that  8: as shown in figure 3. If the differences that
 9: caused the change in the hash value of the     9: caused the change in the hash value of the
10: message was negligible and could not change   10: message was negligi6le and could not change
11: the information in the message, the node      11: the information in the message, the node
12: with the wrong message will request for a     12: with the wrong message will request for a
13:retransmit of the message from the cluster     13:retransmit of the message from the cluster
14:control and resubmit correct message hash.     14:contröl and resubmit correct message hash.
15:Otherwise, the node will be excommunicate.     15:Otherwise, the node will be excommunicate.
```

4 differences found

Figure 7. Bit-wise comparison of messages

2.6.2 Incorrect H(R) was Computed by one or more Actors

If the actor that computed incorrect H(R) submitted a correct H(m) into the *message-hash bucket* before, the actor will be requested to recompute its H(R) because its H(m) had been used to generate vote tokens by other actors. This will make the process that has only this error to end at the point Figure 6: F.

2.6.3 One or more Actors Submitted Correct H(m) but did not Submit $H(R)^S$

The actor will be considered to have crashed after submitting the correct H(m) and it does not respond to pings. If its **H(m)** was correct it will be accepted as valid. When the actor comes up again it will update its *consistency history*. This process will end at the point in Figure 2: D because since its H(m) was correct, the absence of its vote does not have a negative effect.

2.6.4 One or more actors did not Submit Any Response

Crash check protocol as explained here later will be carried out by each active actor on the actors that did not submit a response in the review bucket when their threshold time elapsed. If the inactive actor does not respond it will be considered to have crashed and will be counted out of the agreement process for the session. The crashed node can join another agreement session whenever it becomes active again but not the session that it did not start with other actors.

All actors that did not participate, or submitted differing responses during a consensus process will not update their *consistency history*. The actors that submitted wrong responses and corrected its error will update their *consistency history*. When a record is updated in the auxiliary blockchain consistency history of all the participants will not be updated because the results are disputable.

2.6.5 Crash-Check

The active actors including the *evidence server* will start a timer when they received **<msg, msg-id, timestamp>$_{CC}$**. Each actor will take note of the duration between the time it received **<msg, msg-id, timestamp>$_{CC}$** and the time the first reply was received in the *message-hash* bucket. Twice that duration will be used as the threshold time between submission of the last and the subsequent replies in any buckets. The threshold time calculated by each actor may not be the same. Thus, an actor whose threshold time has elapsed and there were no new replies in a bucket where it expects replies, the actor will send a ping as a hello message to those that are supposed to submit a reply but have not done so. The actor that does not participate in an agreement process and did not respond to pings will be considered to have crashed. Only the consensus responses received will be used for the message validation. If it is the only crash error that occurred, the process will still end at the point in Figure 2: D.

2.7 Authentication

Authentication of participants is done on every response a participant submitted. A participant submits its ID and consistency history together with its signed response. Other actors in the agreement process confirm the signature on the response to be that of the owner of the ID on the response. The consistency history which is unique to a participant and has attributes that link it to other participants' consistency history is also used to authenticate a participant that submits a response.

2.8 Performance Evaluation of CICM Algorithm

The proposed collaborative integrity checking consensus mechanism is evaluated for operational performance and robustness against faults using throughput and latency as metrics. It was tested and its results were compared with three popular consensus mechanisms. The evaluation tests were simulated in NS3-Dev installed on Ubuntu 20.04 LTS, Python 3.8.10, GCC version 10.3.0 on WSL2 virtual machine of Windows 10 Pro 21H1, OS build 19043.1319. The simulation was done on Dell Latitude 7450, 16GB RAM, 1TB HDD. A set of 100 nodes were used comprising 60 users, a cloud server, a cloud controller, and 38 validating actors comprising the cluster controller, evidence server, AI, and other 35 *Agreement Actor*s for CICM. NS3 is a handy testbed that can be adapted for different use cases by adding modules and/libraries to it as may be needed. Nodes are assigned roles by writing scripts for executing the desired functions and installing the scripts on the node. NS3 supports C++ and Python scripting. C++ codes were used for the tests. Cryptopp library was added to the NS3 modules. The elliptic curve cryptography in the Cryptopp library was used for all the cryptographic operations. The ECC digital signature algorithm (ECDSA) was used for signing, and the elliptic curve Diffie–Hellman key exchange (ECDHKE) was used for sharing messages. MD5MAC of Cryptopp was used for MAC generation for the benchmarks that used MAC.

The benchmark solutions used are PBFT [11], Zyzzyva [13], and hBFT [10]. Because of the differences in the working process of CICM and the benchmarks, the settings for the best performances for the benchmarks as reported in [10] were used. According to [10] the three solutions perform better with batching than without batching. The performance peaked when a batch of 10 was used and clients were 30. The best microbenchmark for them was reported as 0/0 (when a client sent 0 KB and replicas replied with 0 KB) and the worst was 4/4 (when a client sent 4 KB and replicas reply with 4 KB). The benchmarks were tested with the following settings; batching b=10, f=1 (1 faulty replica), 64 nodes comprising 60 clients and 4 replicas among which primary is chosen, and 0/0 benchmark. CICM does not batch requests but enough cache size, 10MB, that can avoid packets drop was used to accommodate a large number of messages with a maximum size of 5KB each.

2.8.1 Comparison of CICM Efficacy Against Faults with those of the Benchmark Solution

The binomial distribution chart in Figure 8 shows the probabilities for different numbers of outcomes over the 100 attempts. The results for values of $n = 100$ and different values of x for the mechanisms are shown in Figure 8. Since the fault tolerance for the benchmark solutions is the same as 3f+1, f is the number of traitors. Therefore, their probability, P, of getting incorrect result in one attempt is $\frac{1}{3}$.

Each of the benchmark solutions has a very small probability of giving an incorrect result in a hundred attempts. Unfortunately, the probabilities increase with an increased number of possible incorrect outcomes in the number of attempts. The probability peaked at 0.0844 for 33 possible incorrect outcomes in hundred attempts in Figure 8.

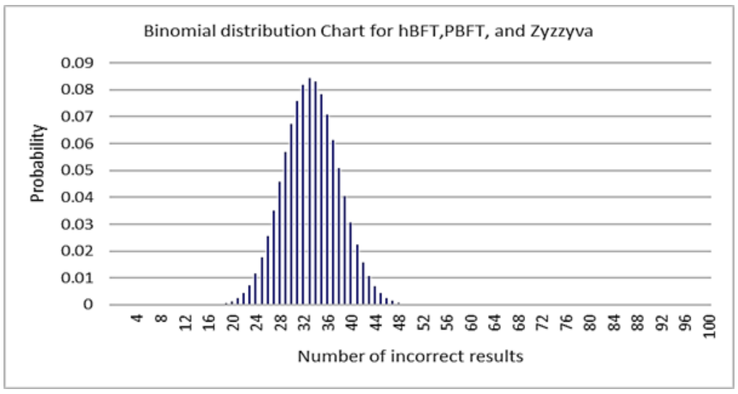


Figure 8. Binomial Distribution of the Probabilities of Getting a Specific Number of Incorrect Outcomes in 100 Attempts.

2.8.2 Throughput

Throughput is used to measure the rate of operation executions by the mechanisms. It determines the number of operations completed per unit of time. Figure 9 compares the results of throughput tests for each mechanism. The CICM operations were faster because the operations were mostly carried out locally. The shared drive provides local copies of the values to nodes as soon as it is available. This enhances faster synchronization. Also, in the benchmarks, the cryptographic operations are performed twice on the same values, the primary perform cryptographic operations on the values when it sends an order to the replicas, the replicas also carry out their cryptographic operations on their response values. But in CICM the cryptographic operations on the values are done only by *agreement actor*s before it is written to the appropriate buckets on the shared drive. CICM maintain a focus in every operation. The message is received from a single source at the same time, it is analyzed together and agreed upon in a single place.

Message overhead was highly reduced with great synchrony among the *validating actor*s in CICM than all other solutions considered. This is shown in the better performance of CICM in Figure 9 where it recorded an average of 30.61% better throughput than its nearest contender, the hBFT. It recorded 52.75% better throughput than PBFT and 72.69% better than zyzzyva.

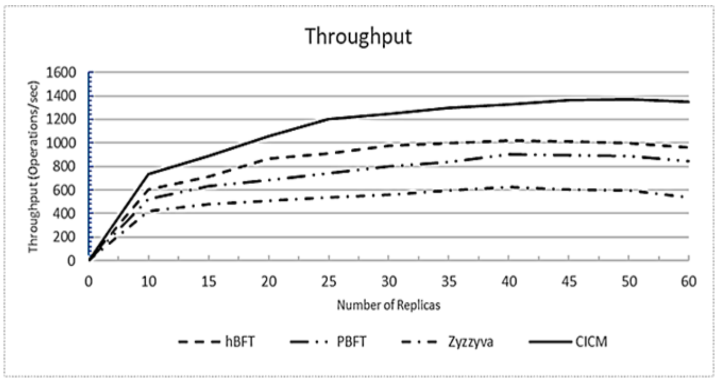


Figure 9. Throughput Plotted Against the Increasing Number of Clients

2.8.3 Latency

The latency test compared the time taken by a mechanism to reach an agreement from when its actors received a request. For the benchmarks, the time starts when the primary received a request from the client because the primary and the clients are active actors in the agreement process. The time starts when *agreement actor*s received **<msg,**

msg-id, timestamp>$_{CC}$ in CICM because the cluster controller does not participate in the agreement process. It ends when the message is committed in the benchmarks and when the agreement votes process submission is completed in CICM. Figure 10 shows the results of latency tests for each mechanism. CICM recorded a 21.47% faster process than hBFT, 35.2% faster than PBFT, and 50.72% faster than zyzzyva.

The speed advantage of CICM over the benchmark solutions was due to the high synchrony through its better collaboration method and reduced message overhead in its processes. These tests results show that CICM has better operational performance and lesser operational costs than the other solutions. A lower throughput and a higher latency are indications of poorer performance and higher costs.

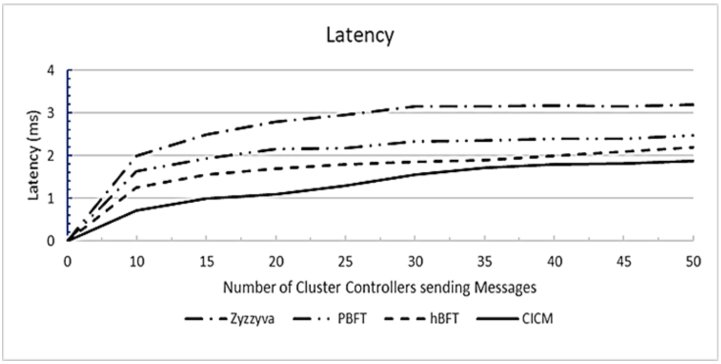
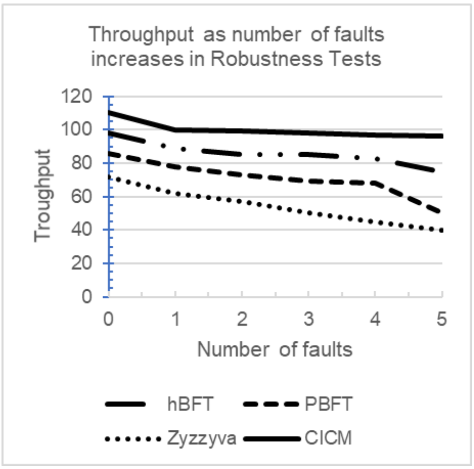


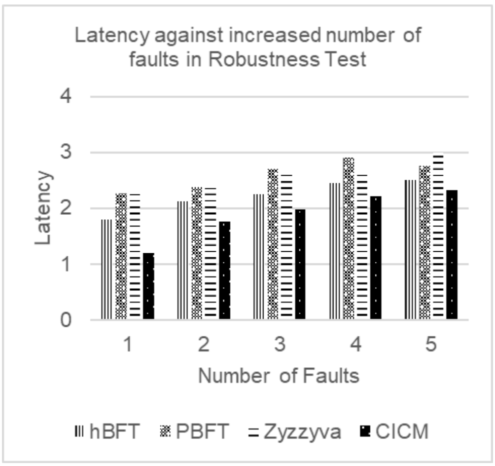
Figure 10. Operational Costs Test: Latency Plotted Against the Number of Clients

2.8.4 Robustness

The robustness test compared the scalability of the mechanisms as the number of faults increased. The benchmarks have thresholds for the number of faults they can tolerate. CICM operation is the same in the face of any faults as explained in section 5. If the fault is a minor one and can be reversed the node involved would be asked to review and recompute the correct value, otherwise, the value will be recorded in the auxiliary ledger. CICM writes disputable results in auxiliary blockchain other mechanisms discard such results. Discarding such results is not acceptable for forensic investigations because vital facts may be lost with the discarded records. The downward fall of CICM throughput and its increasing latency with the increase in faults was due to the repeated actions for every similar fault. The time spent on each fault was comparatively the same for the same type of fault. In the case of the benchmarks, cryptographic operations and message overheads per value increased with an increased number of faults. Figure 11 compares the robustness of each mechanism against the others. CICM recorded a quite more stable and higher throughput than the benchmarks (Figure 11(a)). Zyzzyva's throughput decreased faster than the other benchmarks when the faults increased. CICM recorded 16.5% better average throughput than hBFT, 41.51% better than PBFT, and 84.05% better than zyzzyva. All mechanisms recorded increasing latency as the number of faults increased but CICM recorded the least latency in each test followed by hBFT. Zyzzyva recorded lower latency than PBFT in the tests when *f* = 1, *f* = 2, *f* = 3, and *f* = 4 but recorded higher when *f* = 5 (Figure 11(b)). On average, CICM latency was 14.93% lower than hBFT, 27.12% lower than PBFT, and 26.09% lower than zyzzyva.



(a) Throughput for 0/0 benchmark



(b) Latency for 0/0 benchmarks

Figure 11. Throughput and Latency for benchmark 0/0 measured for the 4 mechanisms

3. Results and Discussion

The importance of the tests conducted on the proposed solution and the results obtained is discussed in this section.

3.1 Avoidance of Critical Consequences

The results of the analysis of the proposed solution and the benchmarks using binomial distribution theoretically show that the proposed CICM is ideally not prone to faults. The same test shows that the probability of the benchmarks producing wrong results could be as high as 8.44%. The challenges of having, even, such a small probability for wrong results are; 1) it does not mean that the possibility of the error occurring is eliminated. 2) the time of occurrence of such error is random which means that it can happen earlier than expected or at the most critical time. Such avoidable consequences are not acceptable in forensic analysis. The reason is that the results of the forensic analysis of digital data may be needed for resolving a life-threatening issue, e.g., to prove innocence in a murder case that carries a death penalty, or other very important instances, such as in business. CICM can address this challenge as shown by equation 4. It makes both records of flawless and faulty data validation processes available to the prospective data analyst to be able to make well-informed decisions. Thus, it can produce sound data for accurate analysis results that can be relied upon to make correct decisions.

3.2 CICM adequacy for Data Capturing for Forensic Analysis Purposes

The operational performance tests show that the performance of the proposed solution increases faster than the other benchmark solutions when the number of messages increases. It recorded a reasonably more stable higher throughput and lower latency than the benchmarks for large numbers of message inputs. This is attributable to the transparent collaborative method adopted for the consensus process which makes the agreement protocol faster. Also, the error handling by the CICM whereby the records of the process in which irreconcilable errors found are simply recorded in the auxiliary blockchain reduces the time spent on the error corrections. These results confirm the suitability of the proposed solution for processing message validations at a rate commensurate with their rate of arrival on the cloud.

3.3 The CICM better Faults Handling

The robustness tests show that the proposed solution is considerably stable irrespective of increasing faults. The steps taken to reconcile an erroneous response from a participant are simpler and fewer than the other mechanisms. Also, participants with crash faults are simply put aside so that the process will not be unnecessarily delayed. These account for the higher stability of the solution in the face of increasing faults.

4. Conclusion

This research proposed a solution for protecting the originality of data on cloud storage to make it suitable for extracting accurate information that can be used as a basis for making valid decisions. The solution implements a consensus mechanism using a 100% agreement for accepting values to be recorded in the main blockchain and records any values that received less than 100% consensus in an auxiliary blockchain. The agreement actors involved in the consensus process scrutinize the status of the message on arrival to the cloud and record its accurate attributes in the appropriate blockchain ledger based on 100% agreement by actors or otherwise. Transparent collaboration used for the agreement process reduced its steps, ensured higher accuracy, and increased synchrony among the actors. This work proposed a new mechanism for achieving a high level of accuracy required for forensic analysis results from digital data and other data analysis that requires very accurate results. The efficacy of the proposed mechanism for the intended uses was checked and compared with three existing solutions using the binomial distribution. Its operational performance costs and robustness against faults were also tested using throughput and latency as metrics. It recorded better results in all the tests than the benchmarks.

5. Limitations and Future Research

This mechanism does not confirm the correctness of the source of the messages as provided by the cluster controller. It only accepts that the message originated from the user whose ID and address were attached by the cloud controller as source and destination. So, the security of the mechanism does not cover activities performed on data by the cloud controller or the cluster controller before forwarding the message to the actors of the mechanism. If the evidence server is compromised the collaboration process may be disrupted. Although it is a rare possibility for all of the agreement actors to be compromised at the same time, but if they were all compromised together the proposed mechanism may not serve its purposes anymore. So, there is the need to secure the validating actors from being compromised. Future research work will focus on obviating the vulnerability and securing the actors.

References

[1] B. Yahaya, M. B. Mu'azu, and S. Garba, "Congestion Control Strategies on Integrated Routing Protocol for the Opportunistic Network: A Comparative Study and Performance Analysis," *Int. J. Comput. Appl.*, vol. 117, no. 4, pp. 975–8887, 2015. https://dx.doi.org/10.5120/20540-2906

[2] O. I. Ademu, C. O. Imafidon, and D. S. Preston, "A New Approach of Digital Forensic Model for Digital Forensic Investigation," *Int. J. Adv. Comput. Sci. Appl.*, vol. 2, no. 12, pp. 175–178, 2011. https://dx.doi.org/10.14569/IJACSA.2011.021226

[3] O. W. Salami, I. J. Umoh, E. A. Adedokun, and M. B. Muazu, "Implementing Flash Event Discrimination in IP Traceback using Shark Smell Optimisation Algorithm," *Kinet. Game Technol. Inf. Syst. Comput. Network, Comput. Electron. Control*, vol. 4, no. 3, pp. 259–268, 2019. https://doi.org/10.22219/kinetik.v4i3.740

[4] R. F. Erbacher, "Validation for digital forensics," *ITNG2010 - 7th Int. Conf. Inf. Technol. New Gener.*, pp. 756–761, 2010. https://doi.org/10.1109/ITNG.2010.18

[5] N. Chaudhry and M. M. Yousaf, "Consensus Algorithms in Blockchain: Comparative Analysis, Challenges and Opportunities," in *ICOSST 2018 - 2018 International Conference on Open Source Systems and Technologies, Proceedings*, Jan. 2019, pp. 54–63. https://doi.org/10.1109/ICOSST.2018.8632190

[6] Q. Wang, J. Huang, S. Wang, Y. Chen, P. Zhang, and L. He, "A Comparative Study of Blockchain Consensus Algorithms," *J. Phys. Conf. Ser.*, vol. 1437, no. 012007, pp. 1–8, Jan. 2020. https://doi.org/10.1088/1742-6596/1437/1/012007

[7] S. S. Hazari and Q. H. Mahmoud, "Comparative evaluation of consensus mechanisms in cryptocurrencies," *Internet Technol. Lett.*, vol. 2, no. 3, pp. 1–6, May 2019. https://doi.org/10.1002/itl2.100

[8] O. W. Salami, M. B. Abdulrazaq, E. A. Adedokun, and B. Yahaya, "Collaborative Integrity Verification for Blockchain-Based Cloud Forensic Readiness Data Protection," in *Informatics and Intelligent Applications. ICIIA 2021. Communications in Computer and Information Science, vol 1547*, M. Sanjay, O. Jonathan, D. Robertas, and M. Rytis, Eds. Springer International Publishing, Cham, 2022, pp. 138–152. https://doi.org/10.1007/978-3-030-95630-1_10

[9] J. Mo, Z. Hu, H. Chen, and W. Shen, "An efficient and provably secure anonymous user authentication and key agreement for mobile cloud computing," *Wirel. Commun. Mob. Comput.*, vol. 2019, no. Article ID 4520685, pp. 1–12, 2019. https://doi.org/10.1155/2019/4520685

[10] S. Duan, S. Peisert, and K. N. Levitt, "Hbft: Speculative Byzantine fault tolerance with minimum cost," *IEEE Trans. Dependable Secur. Comput.*, vol. 12, no. 1, pp. 58–70, Jan. 2015. https://doi.org/10.1109/TDSC.2014.2312331

[11] C. M and L. B, "Practical byzantine fault tolerance and proactive recovery[J]," ACM Trans. Comput. Syst., vol. 20, no. 4, pp. 398–461, 2002.

[12] W. Yan, J. Shen, Z. Cao, and X. Dong, "Blockchain Based Digital Evidence Chain of Custody," in *The 2nd International Conference on Blockchain Technology 2020*, 2020, pp. 19–23. https://doi.org/10.1145/3390566.3391690

[13] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: Speculative Byzantine Fault Tolerance," *ACM Trans. Comput. Syst.*, vol. 27, no. 4, pp. 7:1-7:39, Jan. 2009. https://doi.org/10.1145/1658357.1658358

[14] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008.

[15] A. H. Lone and R. N. Mir, "Forensic-chain: Blockchain based digital forensics chain of custody with PoC in Hyperledger Composer," *Digit. Investig.*, vol. 28, pp. 44–55, 2019. http://dx.doi.org/10.1016/j.diin.2019.01.002

[16] K. Christidis and M. Devetsikiotis, "Blockchains and Smart Contracts for the Internet of Things," *IEEE Access*, vol. 4. Institute of Electrical and Electronics Engineers Inc., pp. 2292–2303, 2016. https://doi.org/10.1109/ACCESS.2016.2566339

[17] L. A. Ajao, J. Agajo, E. A. Adedokun, and L. Karngong, "Crypto Hash Algorithm-Based Blockchain Technology for Managing Decentralized Ledger Database in Oil and Gas Industry," *Multidiscip. Sci. J.*, vol. 2, pp. 300–325, 2019. https://doi.org/10.3390/j2030021

[18] M. Li *et al.*, "CrowdBC: A Blockchain-based Decentralized Framework for Crowdsourcing," *IEEE Trans. Parallel Distrib. Syst.*, 2018. https://doi.org/10.1109/TPDS.2018.2881735

[19] L. S. Sankar, M. Sindhu, and M. Sethumadhavan, "Survey of consensus protocols on blockchain applications," in *2017 4th International Conference on Advanced Computing and Communication Systems, ICACCS 2017*, Aug. 2017, pp. 1–5. https://doi.org/10.1109/ICACCS.2017.8014672

[20] L. Lamport and D. Equipment, "The Part-Time Parliament," *ACMTransactionsonComputerSystems*, vol. 16, no. 2, pp. 133–169, 1998. https://doi.org/10.1145/279227.279229

[21] L. Tseng, Q. Zhang, and Y. Zhang, "Brief Announcement: Reaching Approximate Consensus When Everyone May Crash," in *34th International Symposium on Distributed Computing (DISC 2020)*, 2020, vol. 53, pp. 53:1–53:3. https://doi.org/10.4230/LIPIcs.DISC.2020.53

[22] H. Samy, A. Tammam, A. Fahmy, and B. Hasan, "Enhancing the performance of the blockchain consensus algorithm using multithreading technology," *Ain Shams Eng. J.*, vol. 12, no. 3, pp. 2709–2716, Sep. 2021. https://doi.org/10.1016/j.asej.2021.01.019

[23] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, 1982. https://doi.org/10.1145/357172.357176

[24] D. Mazières, "The Stellar Consensus Protocol : A Federated Model for Internet-level Consensus," pp. 1–45, 2015.

[25] L. Lamport, "Paxos Made Simple," ACM SIGACT News, vol. 32, no. 4, pp. 18–25, 2001.

[26] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in Proceedings of the 2014 USENIX Annual Technical Conference, USENIX ATC 2014, 2014, pp. 305–319.

[27] S. Chaisawat, C. V.-2020 17th I. Joint, and U. 2020, "Fault-Tolerant Architecture Design for Blockchain-Based Electronics Voting System," *ieeexplore.ieee.org*, 2020. https://doi.org/10.1109/JCSSE49651.2020.9268264