I.J. Information Technology and Computer Science, **2025, 4, 28-37**
Published Online on August 8, 2025 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2025.04.03

Methods of Increasing the Efficiency of Data Consistency in Information Systems

Nikitin Valerii*
Igor Sikorsky Kyiv Polytechnic Institute, National Technical University of Ukraine, 37 Prosp. Peremohy, Kyiv, Ukraine
E-mail: nikitin.valerii@lll.kpi.ua
ORCID iD: https://orcid.org/0000-0002-4509-1204
*Corresponding author

Krylov Ievgen
Igor Sikorsky Kyiv Polytechnic Institute, National Technical University of Ukraine, 37 Prosp. Peremohy, Kyiv, Ukraine
E-mail: ekrylov1965@gmail.com
ORCID iD: https://orcid.org/0000-0003-4313-938X

Anikin Volodymyr
Igor Sikorsky Kyiv Polytechnic Institute, National Technical University of Ukraine, 37 Prosp. Peremohy, Kyiv, Ukraine
E-mail: vkanikin@gmail.com
ORCID iD: https://orcid.org/0000-0001-6356-9788

Received: 22 January 2025; Revised: 16 March 2025; Accepted: 17 May 2025; Published: 08 August 2025

Abstract: The article is devoted to special methods for distributed databases that allow to accelerate data reconciliation in information systems, such as IoT, heterogeneous multi-computer systems, analytical administrative management systems, financial systems, scientific management systems, etc. A method for ensuring data consistency using a transaction clock is proposed and the results of experimental research for the developed prototype of a financial system are demonstrated. The transaction clock receives transactions from client applications and stores them in appropriate queues. The queues are processed based on the transaction priority. The highest priority queue is processed before the lowest priority queue. This allows you to determine which important data (such as financial transactions) should be processed first. The article justifies the replacement of the Merkle tree with a hashing algorithm and the use of the Bloom spectral filter to improve the Active Anti-Entropy method to accelerate eventual consistency. For its effective use, the filter generation algorithm is modified, which allowed to increase the speed of its generation and maintain a sufficient level of collision resistance.

Index Terms: Database, Distributed Systems, NoSql, Data Consistency, Consistency, Transactions, Transactional Clock, Active Anti-entropy, Bloom Filter.

1. Introduction

Modern information systems have high requirements for load resistance due to a large number of users, since availability is not limited to the borders of one region, but can go global. This can be solved in several ways: vertical or horizontal scaling. Vertical scaling requires high-performance hardware. In some situations, this is possible and is a solution, but it is a matter of time before you have to repeat the procedure again. Horizontal scaling solves the load problem by distributing the load between multiple service instances. This allows not only to distribute the load, but also to increase the availability of the system in case of failures [1].

This necessitates the creation of a distributed database, as with only one instance of the database, such a system may perform as well or worse in terms of availability due to network latency and processing a large number of requests [2].

A distributed database solves the above problems, but creates new ones related to the relevance of data versions. Write operations require the implementation of special methods and ways to maintain system consistency. Inconsistent data negatively affects the performance of the functions assigned to the information system. End users cannot make the right decisions operating on irrelevant data, which can lead to financial and, in some cases, human losses [3].

This work is open access and licensed under the Creative Commons CC BY 4.0 License.

On the other hand, implementing such mechanisms can affect system availability, since to ensure the highest level of consistency, special actions must be performed that can block newly created transactions. Blocking causes a certain delay during which the end user has to wait [4].

2. Problem Statement

Each information system operates with its own set of data, but in any of them, it is possible to distinguish such data that should be processed first. For example, there is a financial information system that contains a variety of information: personal data of users, user settings, transactions etc. Priority processing of financial transactions is key for such a system, as it directly affects the ability to perform the assigned functions and procedures. User personal data is also important, but its processing can be postponed and the impact of the delay will not be critical. User settings can be classified as service data, since they do not have a direct impact on the tasks set and can be postponed [5].

To improve performance, you can use a transaction clock, which will allow you to discard intermediate transactions and perform CRUD operations on the highest priority transactions [6]. This approach is not suitable for all information systems, but can significantly improve performance in situations where the main emphasis is on data relevance. These can be automated systems that operate in real time and are directly related to the characteristics of real objects (coordinates, temperature, pressure, etc.).

Also, one of the problems is the probability of collisions during data reconciliation on replicas of a distributed database [7]. This is due to the use of hashing for a large set of data, which cannot guarantee uniqueness. This is addressed by implementing the Active Anti-Entropy mechanism that reconciles data during write and read operations. This can be improved by adding background reconciliation, which will allow the consistency of the distributed database to be maintained [8]. To do this, you can use a modified Bloom filter based on prime numbers and the PH-2 hashing algorithm, which will increase resistance to collisions when identifying inconsistencies.

3. A Method for Improving Data Consistency Efficiency using a Transaction Clock with Prioritization

In a complex distributed system, there can be dependent and independent databases. Databases are dependent if they store replicated data. For example, data about a single person's bank account or hotel room reservation data that can be viewed and modified on different nodes in the distributed system.

Dependent databases require reconciliation, while independent databases do not. To distinguish them, it is necessary to introduce the concept of a dependency matrix MS, which is described as follows [9]:

$$MS_{i,j} = \begin{cases} 1, if\ database\ is\ dependent \\ 0,\ if\ database\ is\ independent \end{cases} \quad (1)$$

where i – instance number, j – record number.

A transaction clock is an intermediary between services and a distributed database. Its main tasks include combining data that relates to a single record and managing a transaction queue to speed up operations related to critical data.

Data reconciliation between different system nodes is performed according to the following algorithm:

- All data update and delete operations on dependent databases are executed in a transaction and transferred to the transaction clock queue as quickly as possible.
- All transactions that modify replicated data are stored in a corresponding queue and processed using transactional logic. After that, a resulting transaction is obtained and transmitted to all dependent nodes.
- To speed up the process of data reconciliation of critical data and to take into account the possible race condition, each database record is given its own priority. In this case, critical data will always have the highest priority. The sequence of processing of complex transactions will be performed according to the priorities of the changed records.

The weak point is the node on which the transaction clock will run, due to centralized write and delete operations. If it is unavailable, the possibility of creating the resulting transaction is lost. Read operations are not affected because they are executed on dependent databases, and transactions remain in a local queue on each node until the node with the transaction clock is restored or a failover occurs using the Paxos or Raft protocols.

The logic of the transaction clock is to create a single transaction from a certain set of transactions. If there is a delete operation among the transactions, all others are ignored and only that one remains. If the set consists of update operations, they are merged into the resulting transaction. In the context of document-oriented databases, this means that recursive copying of keys and values will occur. The merging occurs from the older transaction to the newer one. All transactions can be divided into three types: CREATE, UPDATE, DELETE.

A graphical representation of this process can be seen in Fig. 1.

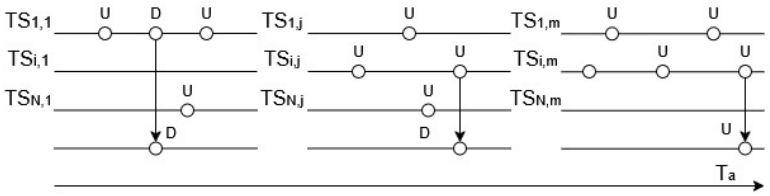


Fig.1. Transaction clock logic [2]

In Fig. 1, $TS_{i,j}$ represents a transaction that occurred on the i-th node, j-th record. During processing in the transaction clock, there can be several of them. For example, Update, Delete, Update operations occurred sequentially for transaction $TS_{1,1}$. Transaction logic must analyze all complex transactions of different nodes for one record and prepare the resulting transaction.

It should be noted that transactions are accumulated on the node as they arrive and are transmitted to the master node if the transaction priority value is the highest.

Of particular note is the Delete operation, which has a higher priority because the deletion occurs on all nodes of the system and because of this, its priority is higher than that of transactions with other operations.

Mathematically, the transaction clock can be described as follows:

$$\mathrm{TS_{i,j}} = f(\mathrm{P_{g_n}}, \mathrm{N}, \mathrm{T_a}, \mathrm{O}, \mathrm{SZ}, \mathrm{NZ}), \tag{2}$$

where $\mathrm{P_{g_n}}$ – dynamic priority n-transaction, N – record number, $\mathrm{T_a}$ – absolute transaction time vector, O – transaction operation, SZ – vector of old values, NZ – vector of new values.

The absolute transaction time vector of a particular node can be calculated using the formula:

$$\mathrm{T_{a_i}} = \mathrm{T_{S_i}} - \mathrm{T_{O_i}}, \tag{3}$$

where $\mathrm{T_{S_i}}$ – transaction time of the i-th node, $\mathrm{T_{O_i}}$ – relative error of the i-th node.

The dynamic transaction priority can be represented by the following formula:

$$P_{g_n} = f(P_{S_n}, T_{a_n}), \tag{4}$$

where P_{S_n} – static priority of the n-th entry, T_{a_n} – absolute time vector of the n-th transaction.

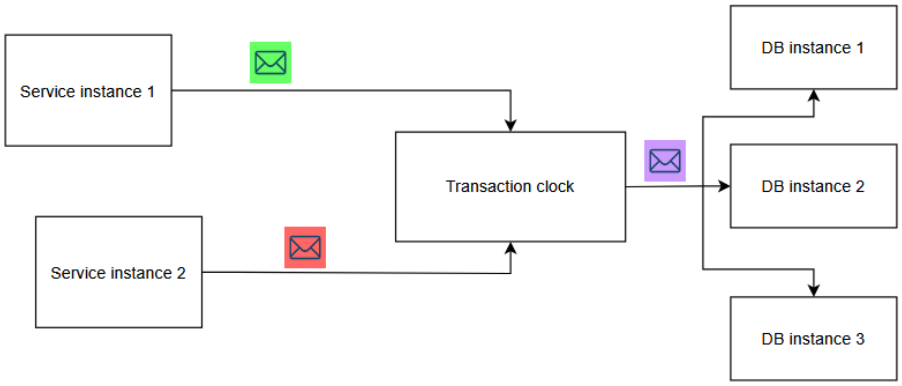


Fig.2. Example of a general work flow [10]

Fig. 2 shows messages sent by different clients to the transaction clock. In this case, the creation time of red messages is earlier than that of green messages. However, the transport time of these messages through the network is different: green messages arrive faster than red ones.

The following rules apply to transactions stored in the transaction clock:

- If there is a DELETE transaction in the transaction array, then all transactions are discarded and only it remains.
- If the set of transactions contains only transactions of type UPDATE, then the newer transaction is merged into the previous one.
- If the transaction set contains CREATE, then transactions of other types will be absent.

To order transactions and, if possible, avoid such situations, each transaction is transmitted with its creation timestamp. This timestamp is created according to the UTC standard, which allows to avoid problems with different time zones. When the process of merging transactions occurs, they are ordered in increasing order and the resulting transaction is created.

After merging the transactions into the resulting transaction, the latter is sent to all nodes of the distributed database. Fig. 3 shows the general scheme of work.

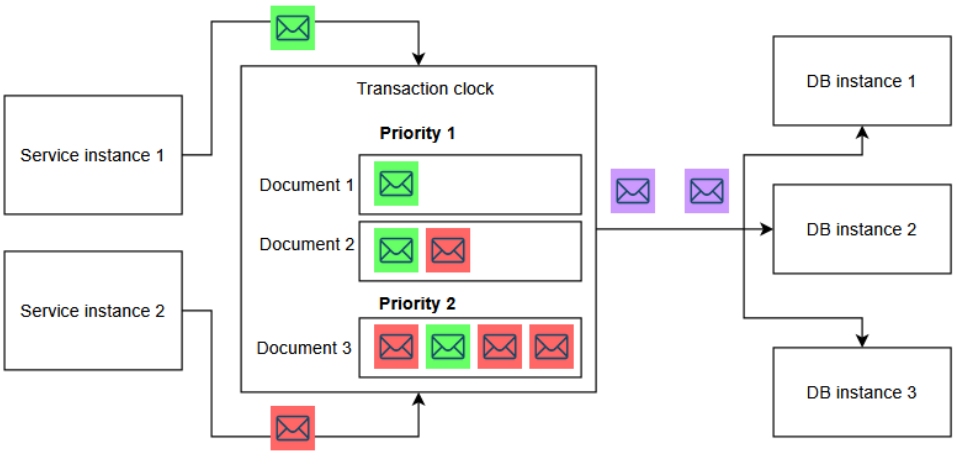


Fig.3. Example of a general diagram of the transaction clock subsystem [10]

Read operations are performed directly to replicas, which allows load balancing between nodes. This is especially useful when structural components are geographically dispersed.

The architecture itself can also be used to create a transaction clock for other databases, as it represents a general-purpose scheme. One possible solution could be the direct integration of the transaction clock into the MongoDB source code. However, this approach requires a deep understanding of the source code, is limited to this specific database, and does not allow for the development of a solution suitable for heterogeneous systems. Based on these considerations, it was decided to develop a separate service with its own software interface.

The service can logically be divided into three layers: endpoints, processing, and data access.

The endpoints layer contains the entry points to the service. For standardization and ease of use, these endpoints are implemented as a REST API on top of the HTTP application-layer protocol.

For convenience, some key parameters are passed in the HTTP request headers. The headers may differ depending on the type of operation:

- Database – the name of the database associated with the operation. This is a string data type. For example, when working with a database of a particular shop, the value “shop” should be provided.
- Collection – the name of the collection associated with the operation. This is also a string value. If the operation is related to user data, the value “users” may be passed.
- Id – the identifier of the record associated with the operation. The value depends on the specific database, as each may use different formats. MongoDB uses a 12-byte string-type identifier. For example, in the context of MongoDB, it may be necessary to pass the value “39aa9936da9c5d99f870ad38”.
- CreatedAt – the timestamp of the operation in UTC format. It is passed as a string value. For example: “2023-04-04T05:00:30.001000+09:00”.
- Priority – the priority level of the operation. It can take any positive integer value. For example, a priority value of 50 can be passed. It should be noted that a priority of 1 represents the highest priority. The larger the number, the lower the operation's priority. The default value is 50.

Each operation has its own set of required and optional headers [10].

4. Modernization of the Data Consistency Process using the Active Anti-entropy Mechanism

The process of data reconciliation using Active Anti-Entropy can be conditionally divided into the following steps:

- Inconsistency identification – establishing the fact that there are certain records on the nodes that differ from each other.
- Localization – determining which objects on the nodes differ.
- Correction – data reconciliation according to a procedure that should decide which data has higher priority. One option could be to use real timestamps or logical clocks.

To reduce the impact of hashing shortcomings, the reconciliation process can be performed for individual specific records on different nodes. In this case, there is no need to deal with localization, since we know at the very beginning the data that is being verified. It also eliminates the risk that mutual compensation will occur when hashing data sets, which will lead to a collision situation.

To identify inconsistencies at different nodes, it is necessary to use a specially calculated snapshot that is minimized in size to reduce the time of its generation and transmission through the network.

This allows to speed up the process, although it may create certain difficulties in scope of implementation and require additional load on the server hardware. The system itself in this case can be represented as a graph. Each node in the system continuously sends a snapshot of a certain record for verification to neighboring nodes. If the node that received the snapshot sees a difference as a result of comparison with the local version, it sends back a timestamp. The node that received the timestamp compares it with the local timestamp. If the current node has a newer version, it sends the record. Otherwise, the node does nothing. Fig. 4 shows a diagram of the mechanism [11].

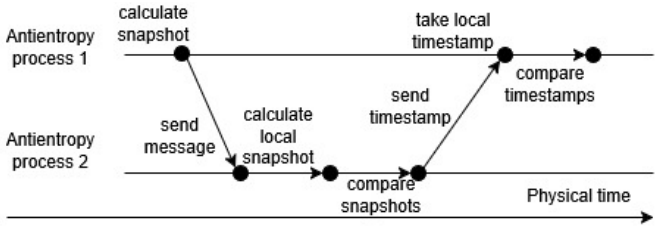


Fig.4. An example of the active anti-entropy mechanism working

Since message exchange occurs continuously, the following requirements arise for messages:

- The image must be collision-resistant.
- The speed of calculating the snapshot should be maximum.
- The size of the snapshot should be minimal to minimize the time required for transmission between nodes over a computer network.

Collision resistance is a critically important indicator, as it determines how quickly the mechanism can detect inconsistencies and begin the data reconciliation process.

Alternatively, cryptographic hashing algorithms could be used, but they have more security-related properties. This is not relevant for maintaining consistency, but may negatively affect the requirements for fast computation and size. In other words, they do not meet the second and third requirements. In this case, non-cryptographic hashing algorithms could be used, but the question of collision resistance remains open for them [12].

It should be noted that the described process occurs in parallel on all nodes without exception. That is, if node 1 sends a snapshot for verification, then node 2 can also send a snapshot for verification at the same time, but the verification may not necessarily refer to the same record (Fig. 5). Each node communicates with its neighbors.

One possible option for creating a snapshot is to use a Bloom spectral filter. In its classical form, it is an array of counters that is formed from the input data. The counter is an unsigned integer. Each input data block is hashed, a digest is obtained, and the position in the array is calculated using it. When accessing a specific element of this array, the counter value is incremented by one [10].

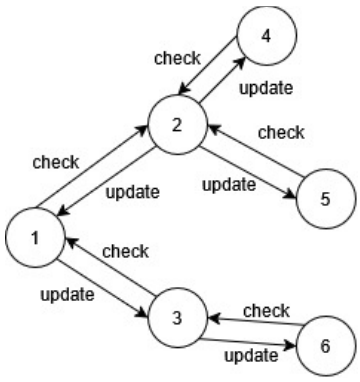


Fig.5. Example of node communication with active anti-entropy

The use of the Bloom spectral filter is due to the fact that this data structure allows for insertion and deletion of elements, which can provide flexibility during implementation for different types of non-relational databases. For example, in document-oriented databases, this allows only document values to be inserted into the filter, while ignoring static keys.

The ability to delete an element can be useful in key-value database applications to check and exclude consistent data. A snapshot is a set of bytes that contain metadata about a record (e.g., record identifier) and the calculated values of the Bloom spectral filter and hashing algorithm. The structure may also include other fields, depending on the type of NoSQL database and some specific tasks. For example, document-oriented databases operate on collections and an additional field with the collection name may be included in the structure. If the collection name is not of fixed size, the size of the collection name may be added for correct deserialization. An example of the structure is shown in Fig. 6 [13].

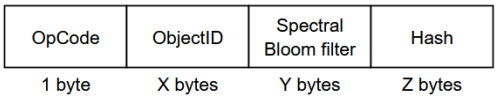


Fig.6. Example message to check records

One of the nodes, having received the snapshot and compared them, sends, if necessary, a response message with the structure as in Fig. 7.

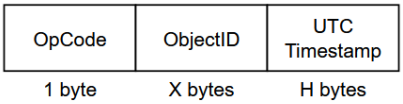


Fig.7. Example of response message structure after snapshot verification

The operation identifier is 1 byte in size and can take three values:

- If a message is sent to terminate the process, the field takes the value 0.
- If a message with a picture is sent, the field takes the value 1.
- If a message with a timestamp is sent, the field takes the value 2 [13].

These identifiers allow to determine the type of message and process it in the right way. The figures show the ObjectID, which identifies a specific record in the database. Its size and format depend on the specific database. In Fig. 7, there is a UTC Timestamp, which is a timestamp when the last changes were made to the document. It is needed to determine which document is newer on the replicas. Its size and format can also be represented differently and depend on the precision with which the time is described. For example, you can use a timestamp with only seconds, or you can also include micro- or nanoseconds. Since not all databases have a special API to obtain this value, the mechanism must store it separately or developers must add it to those documents that are consistent [13].

The last field is used to transmit a hash value. Transmitting a hash value is necessary to avoid collision situations when using the Bloom filter. It should be noted that a message with a data reconciliation termination operation may also be sent, but the message will consist only of the operation code, which is equivalent to 1 byte in size.

The mechanism uses UDP and TCP transport layer protocols for communication, which reduces message processing time on the sender and receiver sides. The UDP protocol is used for all operations. The TCP protocol ensures reliable document transmission when inconsistencies are detected. For the mechanism to work, it is necessary to listen two ports simultaneously to ensure operation [13].

For a mathematical description of the mechanism, a cellular automaton can be used, which resembles the propagation of updates in a system using Active Anti-Entropy. A cellular automaton is a simple computational model consisting of memory elements. It has a defined topology that can be described using a complex network graph $G = (V, E)$, in which V is a set of nodes and E is a set of edges. Thus, each entry is a node G: $k_i \in V_i$ for $1 \leq i \leq n$, where n is the number of elements of the set V and local interactions between these cells are modeled using edges: there is a connection between k_i and k_j if $(k_i, k_j) \in E_i$ [10].

Each cell can have a certain state at each time step t: $s \in S$, where S is a finite set of states. This state changes using a local transition function whose variables are the states of the root element and its neighbors at the previous time step.

Susceptible nodes are those that have not yet received an update. Updated nodes are characterized by the fact that they have received an update and can transmit it to neighboring nodes. The coefficient of the cellular apparatus is the transmission parameter $0 \leq p \leq 1$, which is the probability that a node will become updated and will be able to transmit the update to other connected nodes. This parameter is affected by the state of the network. If the network is stable and the probability of loss of UDP packets used to transmit the document snapshot is discarded, then this parameter is equal to 1.

Main characteristics of the distribution model:

- Each node $k_i \in V$ of the graph G represents a cell and belongs to the cellular apparatus.
- A set of states is formed by two states: susceptible (S), updated (U): $S = \{S, U\}$.
- A cell whose state is $s = U$ can only transmit to its associated nodes $\{k_j \mid (k_i, k_j) \in E \wedge s = S\}$.
- The local transition function between states is the transition from receptive to updated: if $s_t = S$, then $s_{t+1} = U$

with probability p.

The approximate time for the system to transition to a consistent state can be calculated using the formula:

$$T = N \cdot t_s, \tag{5}$$

where N — the number of iterations required for the system to be consistent, t_s — the time it takes for an update to propagate from node to node.

5. Experimental Studies of Data Reconciliation Processes

The consistency of the information system data can be considered synonymous with the consistency of the distributed database, since any information that users work with is stored there. The consistency of the database determines how quickly users tied to another node will see changes. To test and compare the effectiveness of the implemented methods, a special module with an API implementation has been developed, which allows interaction with the information system and the data storage, processing and replication subsystem. The module is written in the Python programming language, uses both an existing library for executing queries, a third-party the pymongo driver for working with MongoDB instances, and written drivers for the created software product.

The business logic begins with the requirement that a person must be registered in the system. To do this, the user provides some personal data, such as first name, last name, date of birth, and other relevant information. After that, an account is created to which this information is linked. The account is necessary for linking bank accounts and for activating or deactivating service features. This enables the creation of credit cards, which can then be associated with the account. To register a card, the following information must be provided: card number, CVV2 code, and expiration date. The initial balance of a new card is zero and will increase or decrease depending on the type of transaction. In general, there are two types: crediting when withdrawing funds or transferring to another account, and debiting when funds are received into the account. When a card is used, regardless of the operation type, a transaction is created in the system that cannot be altered. That is, the transaction is an atomic unit that is preserved over time. This ensures maximum reliability and linearity. In case of a failure on the user's side, a compensating transaction will be created, allowing the system to roll back to the state before the transaction occurred. Such situations may arise due to network issues or errors during interactions between the corresponding subsystems [10].

Additionally, service information includes data about self-service terminals and branches of the financial institution. A self-service terminal is linked to a specific branch and includes operational status and the amount of available cash as service-related data. The branch contains information about its operating hours, physical address, and which self-service systems are located there.

The information system will consist of an application for potential users, an application for internal department staff, and a distributed database composed of eight nodes. It should be noted that the structure of the distributed database and its interconnections will vary depending on the methods being used. When a centralized approach is applied, there will be a primary node with which interactions occur. All write operations will pass through this node, while read operations will be handled by the replicas.

The structural scheme will differ slightly when using a transaction clock. Both applications will interact with the transaction clock and read data from various nodes of the distributed database.

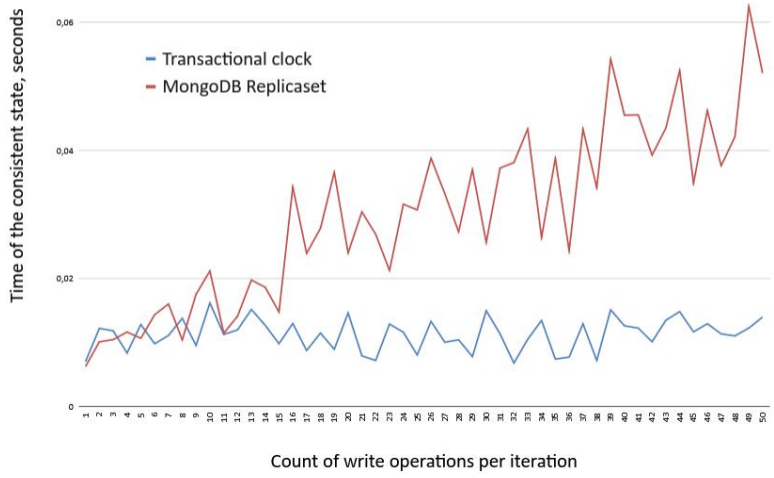


Fig.8. Comparing transactional clock and mongodb replicaset

The final scheme will differ significantly from the previous two, as it will employ the active anti-entropy method. In this setup, the applications connect to different replicas, and the replicas interact with each other.

In Fig. 8, the red line represents the operation of the information system using the built-in consistency mechanisms in MongoDB ReplicaSet. As the number of updates per iteration increases, the data consistency time increases and crosses the 0.06 second mark. The blue line indicates the data consistency time when using the transaction clock. The time spent on system consistency is less and does not increase over time. This is because transactions are combined into a final transaction, and only this transaction is sent to replicas. The built-in MongoDB ReplicaSet mechanism writes records one at a time, which requires additional time for network transmission and writing to the hard disk.

A comparison of the developed Active Anti-Entropy mechanism and MongoDB ReplicaSet is shown in Fig. 9.

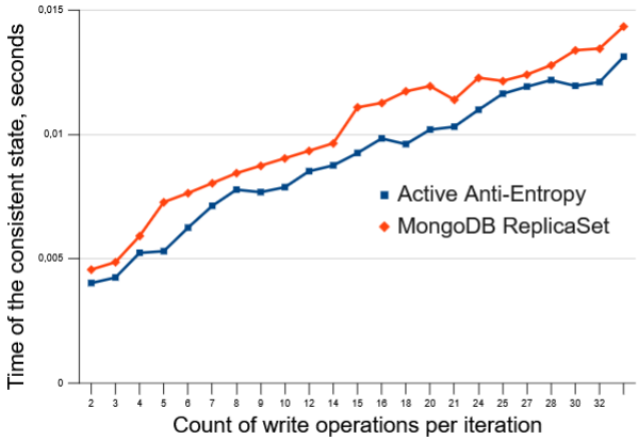


Fig.9. Comparison of the active anti-entropy mechanism and mongodb replicaset

The graph shows that the Active Anti-Entropy mechanism is faster than the MongoDB ReplicaSet mechanisms, but there is also a gradual increase in the required time. This is because the inconsistency identification procedure accumulates transactions and incurs additional communication costs with neighboring nodes [14].

6. Conclusions

The data reconciliation method using a transaction clock has been improved, which allows to increase the speed of synchronization of critical data in a distributed database with a large number of write operations by implementing transaction priority and using transactional logic. The active anti-entropy process has been improved, which differs from the existing one by using the PH2 hashing algorithm [15] instead of the Merkle tree and using a modified Bloom spectral filter, which allows to increase the speed of the data reconciliation process in information systems.

Using Active Anti-Entropy proven effective in areas with relatively few updates and allowed to increase the speed of reconciliation by 11%. Using the transaction clock proved effective in the section with a large number of updates and allowed to increase the speed of reconciliation of critical data by 77%.

While the present study introduces the transaction clock concept, upcoming investigations will emphasize the construction of a formal algorithmic model and its implementation in a fully operational prototype. Such an implementation will enable systematic validation and allow for deeper insight into how the proposed approach performs under varying system architectures and workloads. This phase will also include a more detailed methodological description, including dataset characteristics, benchmarking scenarios, and deployment configurations, to enhance reproducibility and transparency.

Another important direction involves broadening the scope of evaluation metrics beyond speed and collision resistance to encompass dimensions such as scalability, reliability under stress, and resilience to system faults. Investigating how the approach responds to adverse conditions—such as high-latency networks, node failures, and diverse hardware environments—will help assess its suitability for industrial-scale systems. Additionally, comparative studies with established techniques for ensuring data consistency will be conducted to contextualize performance trade-offs and innovation value. Attention will also be given to analyzing potential sources of error, particularly in scenarios involving delayed synchronization or data loss, with the goal of developing robust mitigation strategies that ensure integrity and stability in distributed dataflows.

References

[1] V. Nikitin, E. Krylov, Y. Kornaga, and V. Anikin, “Combined indexing method in NoSQL databases,” Adaptive Systems of Automatic Control, vol. 1, no. 38, pp. 3–9, 2021. [Online]. Available: https://doi.org/10.20535/1560-8956.38.2021.232948.

[2] V. Mukhin, V. Zavgorodnii, V. Nikitin, Y. Kornaga, I. Fartushnyi, and A. Stepanov, “Method of determining the required number of database nodes in a distributed data processing system,” in Proc. 2021 IEEE 3rd Int. Conf. Advanced Trends in

Information Theory (ATIT), 2021. [Online]. Available: https://doi.org/10.1109/ATIT54053.2021.9678569.
[3] S. Gupta, K. Saroha, and Bhawna, "Fundamental research of distributed database," IJCSMS International Journal of Computer Science and Management Studies, vol. 11, no. 2, pp. 138, Aug. 2011. [Online]. Available: https://www.researchgate.net/publication/266892710_Fundamental_Research_of_Distributed_Database.
[4] A. Wibowo and M. Subekti, "Building scalable and resilient database system to mitigate disaster and performance risks," Procedia Computer Science, vol. 135, pp. 25–34, 2018.
[5] R. Alt, R. Beck, and M. T. Smits, "Fintech and the transformation of the financial industry," Electronic Markets, vol. 28, no. 3, pp. 235–243, 2018. [Online]. Available: https://doi.org/10.1007/s12525-018-0310-9.
[6] C.-O. Truică, A. Boicea, and I. Trifan, "CRUD operations in MongoDB," in Proc. 2013 Int. Conf. Advanced Computer Science and Electronics Information (ICACSEI), Beijing, China, Jul. 2013, pp. 347. [Online]. Available: https://doi.org/10.2991/icacsei.2013.88.
[7] C. Leng, J. Wu, J. Cheng, X. S. Zhang, and H. Lu, "Hashing for distributed data," in Proc. Int. Conf. Machine Learning (ICML), Lille, France, Apr. 2015.
[8] R. van Renesse, D. Dumitriu, V. Gough, and C. Thomas, "Efficient reconciliation and flow control for anti-entropy protocols," in Proc. 2nd Workshop on Large-Scale Distributed Systems and Middleware (LADIS '08), Sep. 2008, Art. no. 6, pp. 1–7. [Online]. Available: https://doi.org/10.1145/1529974.1529983.
[9] V. Nikitin and E. Krylov, "Usage of transaction clock to speed up the data consistency process in distributed systems," Scientific Bulletin of Uzhhorod University. Series of Mathematics and Informatics, vol. 42, no. 1, pp. 188–192, 2023. [Online]. Available: https://doi.org/10.24144/2616-7700.2023.42(1).188-192.
[10] V. Nikitin and E. Krylov, "Methods of increasing the efficiency of data consistency in information systems," National Technical University of Ukraine 'Igor Sikorsky Kyiv Polytechnic Institute, [Online]. Available: https://discovery.kpi.ua/Record/dspace.elakpi-123456789-67224.
[11] V. Nikitin and E. Krylov, "Comparison of hashing methods for supporting consistency in distributed databases," Adaptive Systems of Automatic Control, vol. 1, no. 40, pp. 48–53, 2022. [Online]. Available: https://doi.org/10.20535/1560-8956.40.2022.261646.
[12] V. Nikitin, E. Krylov, Y. Kornaga, and V. Anikin, "Modification of hashing algorithm to increase rate of operations in NoSQL databases," Adaptive Systems of Automatic Control, vol. 2, no. 39, pp. 39–43, 2021. [Online]. Available: https://doi.org/10.20535/1560-8956.39.2021.247395.
[13] V. Nikitin and E. Krylov, "Active anti-entropy mechanism based on Spectral Bloom filter and PH-2 hash algorithm for reconciliation of replicas of NoSQL distributed document-oriented databases," Information Technology and Society, vol. 3, no. 9, pp. 63–67, 2023. [Online]. Available: https://doi.org/10.32689/maup.it.2023.3.8.
[14] V. Nikitin and E. Krylov, "Primary-based Spectral Bloom filter for ensuring consistency in distributed document-based NoSQL databases using Active Anti-Entropy mechanism," Computer Systems and Information Technologies, no. 3, pp. 75–80, 2023. [Online]. Available: https://doi.org/10.31891/csit-2023-3-9.
[15] V. Nikitin and E. Krylov, "A collision-resistant hashing algorithm for maintaining consistency in distributed NoSQL databases," Adaptive Systems of Automatic Control, vol. 2, no. 41, pp. 45–57, 2022. [Online]. Available: https://doi.org/10.20535/1560-8956.41.2022.271338.

Authors' Profiles



Valerii Nikitin holds a PhD in Information Systems from the National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute," where his research focused on improving data consistency in distributed systems. His work spans computer vision, deep learning, UAV detection, robotics, and IoT. He develops algorithms for image analysis and object tracking using both classical methods and modern architectures. As a researcher and lecturer, he teaches courses in computer vision, robotics, and networks, integrating his research into academic programs and supervising applied student projects.



Ievgen Krylov holds the academic title of Associate Professor and the degree of Candidate of Sciences (equivalent to PhD). His teaching activities include courses on parallel programming methods and technologies, web programming, and the design of post-relational and object-oriented databases. His research interests lie in real-time computer systems, modern web technologies, and SEO optimization of web-based projects.



Volodymyr Anikin is a faculty member involved in teaching courses on operating systems and decision-making theory. His primary research interests focus on the design and development of software systems, with a particular emphasis on system architecture and applied methodologies in software engineering.

How to cite this paper: Nikitin Valerii, Krylov Ievgen, Anikin Volodymyr, "Methods of Increasing the Efficiency of Data Consistency in Information Systems", International Journal of Information Technology and Computer Science(IJITCS), Vol.17, No.4, pp.28-37, 2025. DOI:10.5815/ijitcs.2025.04.03