DOI: https://doi.org/10.26565/2304-6201-2024-64-01

УДК (UDC) 004.7:004.65

Berezovskyi Oleksandr
PhD student, Faculty of Applied Mathematics and Informatics
Ivan Franko National University of Lviv, 1, Universytetska St., Lviv, Ukraine, 79000
e-mail: oleksandr.berezovskyi@lnu.edu.ua
https://orcid.org/0009-0003-2241-3324

Terletskyi Mykola
PhD student, Faculty of Applied Mathematics and Informatics
Ivan Franko National University of Lviv, 1, Universytetska St., Lviv, Ukraine, 79000
e-mail: Mykola.Terletskyi@lnu.edu.ua
https://orcid.org/0009-0002-6369-0793

Distributed Data Storing Based on Distributed Transaction Ledger

The primary trend in the development of modern information technologies is the migration of computations to the cloud, making distributed computing the dominant strategy for information processing. In particular, this poses the challenge of reliable distributed data storage. A well-known approach to solving the problem of distributed data storage is blockchain or, more generally, distributed ledger technology. A key challenge of this technology is creating an effective mechanism for the global numbering of registry records. The complexity of solving this problem results from the fundamental limitations of distributed computing — the inability to accurately synchronize distributed computing processes and the limitations resulting from the CAP theorem for distributed data stores. The authors attempt to circumvent the mentioned limitations based on the hypothesis that such limitations can be overcome by considering both the network topology and narrowing the class of distributed systems to distributed registers. The work is based on methods of modeling distributed computing, particularly the model of space-time diagrams proposed by L. Lamport. This model allows us to introduce such a tool as logical clocks, including Lamport's logical clock algorithm. Unfortunately, Lamport's logical clock algorithm allows assigning a common timestamp to different events if they are concurrent. The paper proposes an algorithm that is a composition of Lamport's clock algorithm and the wave algorithm, which is not only a logical clock but also assigns different timestamps to different events. Thus, this algorithm provides a mechanism for the global numbering of entries of distributed ledger replicas. A problematic issue remains gaps in the series of ledger entry numbers. Thus, the paper proposes an effective mechanism for the global numbering of records of a distributed register and identifies a shortcoming of this mechanism. Further study is to identify specific conditions in terms of network topology that would ensure the absence of the mentioned shortcoming.

Keywords: *ledger, distributed ledger, network topology, distributed computation, logical clock, Lamport's clock*.

Як цитувати: Berezovskyi O., Terletskyi M. Distributed Data Storing Based on Distributed Transaction Ledger. *Вісник Харківського національного університету імені В. Н. Каразіна, серія Математичне моделювання. Інформаційні технології. Автоматизовані системи управління*. 2024. вип. 64. С.6-12. https://doi.org/10.26565/2304-6201-2024-64-01

How to quote: O. Berezovskyi, and M. Terletskyi, "Distributed Data Storing Based on Distributed Transaction Ledger", *Bulletin of V. N. Karazin Kharkiv National University, series Mathematical modelling. Information technology. Automated control systems*, vol. 64, pp. 6-12, 2024. https://doi.org/10.26565/2304-6201-2024-64-01

Introduction

The exponential growth of data has necessitated innovative approaches to data storage and management. While efficient for structured data, traditional centralized database systems face scalability and security challenges as data volumes increase and distribution becomes critical [1]. Distributed data storage systems, on the other hand, offer scalability and fault tolerance but often compromise on data consistency and reliability.

A fundamental challenge in distributed data storage systems is the CAP theorem, which states that it is impossible to simultaneously guarantee Consistency, Availability, and Partition Tolerance for a distributed system [2].

Here, we understand consistency as strong or weak consistency more precisely, we say strong consistency whenever all nodes in the system see the same data at the same time; in contrast, if all nodes will see the same data eventually but there may be delays then we say about weak consistency.

Availability is understanding the system must always be available to process requests, even in the face of failures.

Partition Tolerance means the system must continue to operate correctly even if a system part loses the ability to communicate with its other parts.

The trade-off of these requirements forces system designers to decide which properties to prioritize.

Understanding the CAP theorem is crucial for designing and implementing distributed data storage systems. By carefully considering the trade-offs involved, system architects can make informed decisions about prioritizing these properties depending on their importance for their specific use case.

This research proposes a hybrid approach that combines the strengths of Acidic Database Management Systems (ACID DBMS) and Distributed Ledger Technology (DLT) [3]. By integrating the ACID properties of databases with the distributed and immutable nature of DLTs, we aim to create a robust, scalable, and secure distributed data storage solution [1].

Our approach is based on the assumption of a loss-free transmission of information in the system [4].

We are not expecting the proposed approach to be the panacea for any system development context. Moreover, one of our research goals is to prove that such an expectation is unjustified. But we hope this approach is fruitful for special network topologies under some fairness guarantees.
In this paper, we demonstrate the grounds of our expectations.

1. Distributed Computation

A distributed computation is an information processing performed by at least two processing units interacting via a network [1]. The key constraints on processing unit behavior are that each unit performs some sequential computation, and that message passing is the only interaction between units .

Thus, a distributed computation is determined by a network, whose nodes perform local sequential processes, by these local processes, and by features of the message-passing by processes. Below, we give the mathematical model of a distributed computation presented by O. Barskyi, I. Illin, and A. Zozulia on PhD Symposium of ICTERI 2024 Conference.

Typically, a processing unit that performs a particular local process of a distributed computation, is identified with that process and is called a computation participant.

Considering that, one can classify distributed computation as follows [4].

1. The local behavior of each computation participant corresponds completely to the computation specification, and each sent message will be eventually received.

2. The local behavior of a computation participant can violate the computation specification, but each sent message will be eventually received.

3. The local behavior of each computation participant completely corresponds to the computation specification but there is a nonzero probability of losing a message under transmission.

4. The local behavior of a computation participant can violate the computation specification and there is a nonzero probability of losing a message under transmission.

Usually, we refer to case 1 as a reliable computation, case 2 as a computation with Byzantine failures, case 3 as a computation with non-Byzantine failures [5], and case 4 as a general-kind computation.

Note that non-Byzantine failures are rather associated with the unreliability of network equipment than with incorrect behavior of computing participants. Therefore, we focus our attention on studying case 1 in the paper.

Model of a Network [6]**.** As it is generally accepted, we understand a network [7] as an oriented or unoriented graph with nodes that refer to participants and edges that indicate the possibility of transmitting data directly from the source of the edge to the target of the edge. The standard constraints for the network graph are that

- There is no loop in the graph. This means that there is no way to transmit data directly to itself.
- For any two nodes, there exists a route connecting these nodes. This constraint is referred to as the connectivity.

Model of a Participant Behavior. The next key component of distributed computation is the set of the computation participants' behaviors. However, if we have n participants in the computation it does

not mean that each participant has unique behavior. As a rule, we can divide all participants into classes of participants having the same behavior. Usually, the number m of these classes is less than n. The references to these classes are called roles as usual. Of course, the number of roles can be equal to one. For example, all participants of a distributed ledger have the same behavior which means the number of roles is one.

Local behavior is a sequence of computation actions and two kinds of additional actions: sending and receiving messages. These kinds of action provide interaction of the computation participants. So, for modeling local behaviors of a distributed computation, one can use any model of single-process computation extended with the mentioned two behavior primitives.

Events of a Computation. It is generally accepted that any action in a distributed computation is atomic. This assumption provides the ability to consider an action as an event, that is, some instantaneous change in the configuration of the system. According to L. Lamport [xx], the set of computational events E is equipped with a partial order called a causal order or a "happen-before" relation. The fact "an event e_1 has happened before an event e_2" is denoted by $e_1 \to e_2$. This relation is defined [8, Definition at p. 559] as follows:

1) if e_1 and e_2 are events corresponding to local actions of the same participant, which are neither sending nor receiving events, and e_1 precedes e_2 in the local ordering of this participant's events then $e_1 \to e_2$;
2) if e_1 is a message sending event, and e_2 is the receiving event of this message then $e_1 \to e_2$;
3) if $e_1 \to e_2$ and $e_2 \to e_3$ then $e_1 \to e_3$.

If both $e_1 \nrightarrow e_2$ and $e_2 \nrightarrow e_1$ are true, then events e_1 and e_2 are called concurrent. In this case, the denotation $e_1 \parallel e_2$ is used.

Logical Clock of a Computation. The concept of a logical clock was introduced for providing a manner to order events of a computation [8]. Firstly, we need a set T of timestamps which are surrogates of physical timepoints. The natural requirement is such a set should be a well-ordered set of the order type ω. It means that $T = \{t_0 < t_1 <\ldots< t_{n-1} < t_n < t_{n+1} <\ldots\}$.

A logical clock is a function $c: E \to T$ such that $c(e_1) < c(e_2)$ whenever $e_1 \to e_2$.

Let us discus a simple but important proposition.

Proposition. Let $c: E \to T$ be a logical clock then if $c(e_1) = c(e_2)$ then $e_1 \parallel e_2$ for any $e_1 \neq e_2 \in E$.

Proof. Indeed, there the following variants only for any $e_1, e_2 \in E$:

1) $e_1 \to e_2$ and $e_2 \to e_1$ is impossible because it contradicts to irreflexivity of the causal order.
2) $e_1 \to e_2$ and $e_2 \nrightarrow e_1$ is impossible because it contradicts to $c(e_1) = c(e_2)$ due to $c(e_1) < c(e_2)$.
3) $e_1 \nrightarrow e_2$ and $e_2 \to e_1$ is impossible because it contradicts to $c(e_1) = c(e_2)$ similarly to the previous.
4) and only $e_1 \nrightarrow e_2$ and $e_2 \nrightarrow e_1$ i.e. $e_1 \parallel e_2$ does not contradict to $c(e_1) = c(e_2)$.

Qed.

Lamport's clock algorithm. L. Lamport proposed the following algorithm to equip each computational event with a timestamp for $T = N$ where N is an ordinal of natural numbers. All participants of this algorithm behave similarly. In other words, the role set of this algorithm is a singleton.

Algorithm of Lamport's clock.

Assume

$p_1, \ldots, p_n (n \geq 2)$ be participants of a computation,

e_i^k denote the *k*-th event in the event sequence of the participant p_i.

Ensure

a logical clock $Lamport: E \to N$.

The algorithm requires to equip each participant p_i with a counter ts_i and initializes it by zero.

Behavior of p_i.

Catch an event e.

If e is an event corresponding to a local action of p_i then $c(e) \leftarrow ts_i$ and $ts_i \leftarrow ts_i + 1$.
If e is a message m sending event then $c(e) \leftarrow ts_i$, $ts_i \leftarrow ts_i + 1$, add ts_i to m and resend m.
If e is a message m receiving event, then $ts_i \leftarrow max(ts, ts_i)$, $c(e) \leftarrow ts_i$, and $ts_i \leftarrow ts_i + 1$.
Jump to the first line.

It is known that this algorithm computes a logical clock, and this clock referred to as $Lamport$ satisfies the following condition

$$Lamport(e) \leq c(e) \text{ for any clock } c: E \to N \text{ and } e \in E. \quad (1)$$

It is because of (see [Tel, p. 62])

$Lamport(e) = 0$ if and only if there is no $e' \in E$ such that $e' \to e$;
$Lamport(e) = n > 0$ if and only if $n = max\big(Lamport(e') \big| e' \in P(e)\big) + 1$
where $P(e) = \{e' \in E \vee e' \to ebutthereisnoe'' \in Esuchthate' \to e'' \to e\}$.

2. Model of Distributed Ledger

Before considering a distributed ledger (DL), we construct the model of a ledger, which is understood as a data store that allows only writing new data units and reading stored data units. In this context, this store can be represented by a finite set S of pairs $(timestamp, data_{unit})$ where the timestamp is a key. We do not fix a set of data units D and consider it as a model parameter. Also, we restrict the set T of timestamp values by ordinals of the order type ω. The concrete ordinal is selected depending on the studied problem. It can be the ordinal of natural numbers with the natural ordering or the ordinal of strings with lexicographic ordering [8]. For storing new data unit d, we choose the key

$$ts = min\{ts' \vee \forall x \in D, (ts', x) \notin S\} \quad (2)$$

and then write the pair (ts, d). Of course, formula (2) is suitable only for specifying a new key value, and not for its calculation, the algorithm of which depends on the specific choice of the set T.

A distributed ledger (DL) is an information system that stores data across network nodes (belonging to a set P) and allows only two types of transactions: writing new data and retrieving written data. It is suggested that each node in the network has its ledger called a replica. It ies assumed that each node in the network has its own ledger, called a replica. A distributed ledger exists physically only as a collection of replicas, so the consistency of these replicas is important.

Replica consistency can be formalized as follows

$$\forall ts \in T, p, q \in P, x_p, x_q \in D, (ts, x_p) \in S_p \to (ts, x_q) \in S_q \to x_p = x_q \quad (3)$$

In formula (3), S_p and S_q refer to the replicas owned by nodes p and q respectively.

Formula (3) does not require that replicas match for all nodes, it only requires that replica keys be not only local but also global keys.

Of course, due to the CAP theorem [2], we cannot expect the existence of an algorithm to record data to all local ledgers consistently. In more detail, DL nodes receive write requests and interact to execute these requests by the nodes' local ledgers modifying consistently.

So, we need a method for assigning the correct timestamp to any request to write a new data unit.

More details, let $p_1, \dots, p_n$ $(n \geq 2)$ be nodes of a DL being studied, w_i^k be k-th request for writing obtained by node p_i, and e_i be another event observed by node p_i (if we need to consider several such events, we use superscripts for these events distinguishing).

Of course, we can use Lamport's clock algorithm to determine the local keys of each DL node.

This approach, of course, ensures the inequality $Lamport(w_i^k) < Lamport(w_i^l)$ if $k < l$.

But, if we order events w_i^k and w_j^l, we can obtain $Lamport(w_i^k) = Lamport(w_j^l)$ due to $w_i^k \parallel w_j^l$.

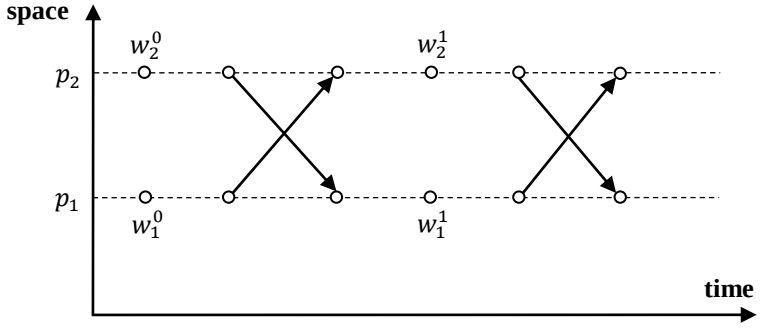

Fig.1. Lamport's timestamp conflict

In Fig. 1 we illustrate Lamport's timestamp conflict. In this figure, the unlabeled circles correspond to the events of sending/receiving notifications concerning timestamps. One can easily see those events w_1^0 and w_2^0 and w_1^1 and w_2^1 of receiving requests for writing a data unit have the same Lamport's timestamp equal respectively 0 and 3.

3. DL Timestamping Algorithm

The example from the previous section not only illustrates an anomaly in the Lamport clock algorithm for distributed ledgers but also allows us to understand how to modify this algorithm to avoid this anomaly.

The algorithm is applied if a wave algorithm is available for the network topology [6, 7]. Moreover, the use of wave algorithms specific to a particular network topology can significantly change the efficiency of the algorithm.

DL timestamping algorithm.

Assume:

$p_1, \dots, p_n$ be nodes of a DL where $n \geq 2$;

$T = N \times \{1, \dots, n\}$ be the timestamp set with the lexicographic order that turns it into a well-ordered set of order type ω

Ensure:

add a data unit with correct timestamp

Behavior of p_i.

Catch an event e.

If e is a request for writing data unit d, then

$ts \leftarrow (Lamport(e), i)$

write d with key ts

start wave by sending notification (ts, d)

If e is an event of notification receiving, then

if a message with ts from (ts, d) has received firstly, then write d with key ts

execute step of the wave algorithm adding Lamport's timestamp to all necessary messages

If e is an event of other kinds, then ignore it.

Jump to the first line.

The proposed algorithm guarantees that timestamps for writing requests received by the same ledger node are ordered by their arrival at this node. Due to Lamport's clock algorithm and the concurrency of the writing requests obtained by different ledger nodes, we can conclude that Lamport's timestamps of such requests can be equal.

Therefore, we add the identifier of the ledger node that has obtained a writing request to Lamport's timestamp of this request. This addition allows us to avoid the anomalies of logical clocks like the anomaly from the example.

Returning to the example, request event w_1^0 obtains the timestamp (0,1), event w_2^0 obtains the timestamp (0,2), event w_1^1 obtains the timestamp (3,1), and event w_2^1 obtains the timestamp (3,2). In other words, we have the following order of request events: w_1^0, w_2^0, w_1^1, w_2^1. However, we must note that the series of timestamp values has gaps. In the example, we do not have entries with timestamps (1,1), (1,2), (2,1), and (2,2) between the entries with the timestamps (0,2) and (3,1) respectively.

4. Conclusion

The previous consideration demonstrated that, under the crucial assumptions of network reliability and strict adherence to the algorithm by all participants in the distributed ledger, the composition of a wave algorithm and Lamport's clock algorithm gives a viable mechanism for establishing global numbering of ledger entries. This mechanism provides a crucial foundation for many distributed ledger applications, offering a consistent and ordered view of the transaction history.

However, the proposed algorithm, as outlined in the "DL Timestamp Algorithm" section, exhibits a significant drawback: it generates a series of timestamps with gaps. This gap, while not necessarily fatal to all applications, can pose challenges in certain scenarios. For instance, in a distributed transaction registry for a complex of ACID databases, consistent ordering and the absence of gaps in timestamps are paramount for maintaining data integrity and ensuring the correct execution of transactions across multiple nodes.

The presence of gaps in the timestamp series necessitates further investigation and refinement of the algorithm. This leads to several key open questions.

1) Robustness against Network Failures: the current algorithm's behavior in the face of network disruptions, both non-Byzantine (e.g., temporary network partitions) and Byzantine (e.g., malicious attacks), needs thorough evaluation. In the context the following are interesting:

- Can the algorithm be modified to maintain its functionality and ensure consistent timestamping even if such failures are present?
- What are the trade-offs between robustness and performance in such scenarios?

2) Algorithmic Complexity: assessing the computational complexity of the algorithm is crucial for practical implementation. In the context the following are interesting:

- How does the complexity scale with the number of nodes in the distributed ledger network?
- Are there optimization techniques for reducing the computational overhead while maintaining accuracy and consistency?

3) Physical Time Estimation: while the algorithm provides a logical ordering of events, it's essential to understand the relationship between the generated timestamps and physical time; in other words

- Can we estimate the physical time of execution of a round of the algorithm with reasonable accuracy?

This information is crucial for performance analysis, debugging, and potentially for applications that require real-time constraints.

Addressing these open questions is crucial for further development and practical application of the proposed global numbering mechanism.

Now let us outline some points for future research directions:

- Exploring alternative approaches to global numbering that may be more resilient to network failures.
- Investigating techniques for minimizing the occurrence of gaps in the timestamp series.
- Developing analytical models to predict the performance and behavior of the algorithm in different network environments.

By addressing these challenges and refining the proposed algorithm, we can pave the way for more robust and efficient distributed ledger technologies with applications in various domains, from financial systems to e-government systems.

REFERENCES

1. Françoise Baude, Denis Caromel, Nathalie Furmento, David Sagnol, Optimizing remote method invocation with communication–computation overlap. Future Generation Computer Systems, Vol. 18, Issue 6, pp. 769-778б ISSN 0167-739X, 2002.
   URL: https://www.sciencedirect.com/science/article/pii/S0167739X02000493
2. Brewer, Eric A. “CAP twelve years later: How the "rules" have changed.” Computer 45 (2012): 23-29. URL: https://ieeexplore.ieee.org/document/6133253
3. Hussein, Z., Salama, M.A. & El-Rahman, S.A. Evolution of blockchain consensus algorithms: a review on the latest milestones of blockchain consensus algorithms. Cybersecurity 6, 30 (2023). URL: https://doi.org/10.1186/s42400-023-00163-y
4. N. Olifer, V. Olifer, Computer Networks: Principles, Technologies and Protocols for Network Design. Wiley, 2006, 1008 p.
   URL: https://www.wiley.com/en-br/Computer+Networks%3A+Principles%2C+Technologies+and+ Protocols+for+Network+Design-p-9780470869826
5. Gong, Li, Patrick Lincoln and John M. Rushby. “Byzantine Agreement with Authentication: Observations and Applications in Tolerating Hybrid and Link Faults”. In: Dependable Computing and Fault Tolerant Systems. Vol. 10 (1995), pp. 139-157.
   URL: https://www.csl.sri.com/papers/dcca95/dcca95.pdf
6. I. V. Stetsenko, Systems modeling: textbook. Cherkasy: CSTU, 2010, 399 p. [in Ukrainian]

7. U.I. Losev, K.M. Rukkas, S.I. Shmatkov. Computer Networks: textbook. Kharkiv: V.N. Karazin Kharkiv National University, 2013, 248 p. [in Ukrainian] http://www.its.kpi.ua/itm/lgloba/Lists/publications/Attachments/15/(11)--TUD_IBIS_Shill_Globa_NTUU_KPI_camera_ready.pdf
8. L. Lamport. “Time, clocks, and the ordering of events in a distributed system.” CACM, 21(7) (1978), pp. 558-565. URL: https://dl.acm.org/doi/pdf/10.1145/359545.359563
9. W. Fokkink. “Distributed Algorithm: An Intuitive Approach. The MIT Press, 2013. URL: https://mitpress.mit.edu/9780262037662/distributed-algorithms/
10. G. Tel. “Introduction to Distributed Algorithms”. The Cambridge University Press, 2000. URL: https://www.amazon.com/Introduction-Distributed-Algorithms-Gerard-Tel/dp/0521470692

Березовський Олександр Валерійович *Аспірант*
Львівський національний університет імені Івана Франка, вул. Університетська, 1, Львів, 79000

Терлецький Микола Віталійович *Аспірант*
Львівський національний університет імені Івана Франка, вул. Університетська, 1, Львів, 79000

Розподілене зберігання даних на основі технології розподіленого реєстру

Основною тенденцією розвитку сучасних інформаційних технологій є перенесення обчислень в хмару, що робить розподілені обчислення домінуючою стратегією обробки інформації. Зокрема, це ставить задачу надійного розподіленого зберігання інформації. Відомим підходом до вирішення проблеми розподіленого зберігання даних є блокчейн або, у більш загальному випадку, технологія розподіленого реєстру. Ключовою проблемою цієї технології є створення ефективного механізму глобальної нумерації записів реєстру. Складність вирішення проблеми є наслідком фундаментальних обмежень розподілених обчислень - відсутністю можливості точної синхронізації процесів розподіленого обчислення та обемежень, що є наслідками CAP теореми для розподілених сховищ даних. Виходячи з гіпотези про те, що такі обмеження можуть бути подолані шляхом врахування як особливостей топології мережі, так і звуженням класу розподілених систем до розподілених реєстрів, автори намагаються обійти зазначені обмеження. В основі роботи лежать методи моделювання розподілених обчислень, зокрема, модель просторово-часових діаграм, запропонована Л. Лемпортом. Ця модель дозволяє ввести такий інструмент, як логічні годинники, включно з алгоритмом логічного годинника Л. Лемпорта. Нажаль, алгоритм логічного годинника Л. Лемпорта допускає приписування спільної позначки часу для різних подій за умови їх конкурентності. В роботі запропановано алгоритм, який є композицією алгоритму годинника Л. Лемпорта та хвильового алгоритму, який не тільки є логічним годинником, але й приписує різні позначки часу різним подіям. Таким чином, цей алгоритм дає механізм глобальної нумерації записів реплік розподіленого реєстру. Проблемним питанням залишається наявність лакун серед номерів записів реєстру. Отже, в роботі запропонований ефективний механізм глобальної нумерації записів розподіленого реєстру та виявлений недолік цього механізму. Подальшим розвитком дослідження є з’ясування специфічних умов в термінах топології мережі, які забезпечували б відсутність зазначеного недоліка.

Ключові слова: *реєстр, розподілений реєстр, мережева топологія, розподілене обчислення, логічний годинник, годинник Лемпорта.*

Надійшла у першій редакції 12.08.2024, в останній - 19.09.2024.
The first version has been received on 12.08.2024, the final version - on 19.09.2024.