EasyChair Preprint

№ 14509

Grouped Matrix Clocks with Reduced Complexity for Distributed Synchronization

Khizer Tariq and Hasib Aslam

August 20, 2024

Grouped Matrix Clocks with Reduced Complexity for Distributed Synchronization

***Abstract*—Logical clock synchronization is a crucial aspect of distributed systems, enabling the correct ordering of events and maintaining causal relationships. The matrix clock algorithm, while effective, suffers from quadratic communication overhead as the number of processes increases due to its n x n matrix size representation. This paper introduces a novel group-based matrix clock algorithm that reduces this overhead by exploiting communication locality patterns among processes. The key idea is to partition processes into multiple groups based on their communication frequencies. Processes within a frequently communicating group maintain a small intra-group matrix clock, while each group maintains a compact group-level matrix clock summarizing the group's collective knowledge. Inter-group communication is timestamped using these group matrix clocks, reducing the overhead compared to fully replicated global matrix clocks. This approach reduces the timestamp size for frequent intra-group communication while preserving sufficient causal information for accurate event ordering via the group-level clocks. Theoretical analysis demonstrates asymptotic space and communication overhead reductions compared to the original matrix clock algorithm. Empirical evaluation confirms the optimization benefits, especially as system scale and communication locality increase. The proposed group matrix clock algorithm retains the powerful causality tracking capabilities of matrix clocks while improving efficiency for realworld distributed systems with heterogeneous communication patterns.**

Keywords—Distributed Computing, Logical Clocks, Parallel and Distributed Computing, Synchronization, Causal Ordering

I. Introduction

Distributed systems have become ubiquitous in modern computing environments, enabling the coordination and collaboration of multiple processes or nodes to achieve complex tasks. However, the inherent concurrency and lack of a global clock in these systems pose significant challenges in maintaining consistent event ordering and causal relationships among processes. Logical clock synchronization techniques play a crucial role in addressing these challenges, ensuring the correct execution of distributed algorithms and applications.

One of the widely adopted logical clock synchronization algorithms is the matrix clock, introduced by Mattern [1]. Matrix clocks effectively capture causality information and logical timestamps for events in distributed systems. Unlike scalar clocks, which can only provide a partial order of events, matrix clocks establish a total order, allowing for the accurate reconstruction of the system's execution history. However, the traditional matrix clock algorithm suffers from a significant limitation: its communication overhead grows quadratically with the number of processes in the system. Each process maintains an $n \times n$ matrix, where n is the total number of processes, leading to increased space requirements and message sizes as the system scales.

In real-world distributed systems, communication patterns often exhibit locality, where certain subsets of processes communicate more frequently with each other than with others. This phenomenon is particularly prevalent in large-scale systems with heterogeneous workloads, such as cloud computing environments, peer-to-peer networks, and distributed databases. Exploiting this communication locality can lead to significant optimizations and performance improvements.

This paper presents a novel group-based matrix clock algorithm that aims to reduce the communication overhead associated with traditional matrix clocks while preserving their powerful causality tracking capabilities. The key innovation is the partitioning of processes into multiple groups based on their communication frequencies. Processes within a frequently communicating group maintain a small intra-group matrix clock, while each group maintains a compact group-level matrix clock summarizing the group's collective knowledge. Inter-group communication is timestamped using these group matrix clocks, reducing the overhead compared to fully replicated global matrix clocks.

By leveraging communication locality, the proposed algorithm reduces the timestamp size for frequent intra-group communication, leading to asymptotic space and communication overhead reductions compared to the original matrix clock algorithm. Empirical evaluation confirms the optimization benefits, especially as system scale and communication locality increase. The group matrix clock algorithm preserves the ability to accurately order events and maintain causal relationships, making it suitable for a wide range of distributed applications and systems with heterogeneous communication patterns.

II. Literature Review

Logical clock synchronization has been an active area of research in distributed systems, with various techniques proposed to maintain consistent event ordering and causality tracking. One of the earliest and most fundamental approaches is Lamport's logical clocks [2], which introduced the concept of using scalar logical timestamps to establish a partial order of events in a distributed system. While Lamport clocks capture the basic happened-before relationships, they are insufficient for reconstructing the complete causal history of events.

Vector clocks, introduced by Mattern [3] and Fidge [4], extend the concept of logical clocks to provide a total order of events. Each process maintains a vector of logical timestamps, one for each process in the system. Vector clocks accurately capture causality by tracking the dependencies between events

across processes. However, the size of vector timestamps grows linearly with the number of processes, leading to increased communication overhead in large-scale systems.

Several optimizations and variants of vector clocks have been proposed to reduce this overhead. The Plausible Clock Condition (PCC) [5] technique aim to reduce the size of vector timestamps by exploiting the sparse nature of causal dependencies in many distributed applications. However, these approaches still require maintaining and transmitting vectors of size proportional to the number of processes.

Matrix clocks, introduced by Mattern [1], represent a powerful logical clock synchronization technique that captures both causality and concurrency information. Each process maintains an $n \times n$ matrix, where n is the number of processes, allowing for the reconstruction of the complete system execution history. Matrix clocks establish a total order of events and enable accurate causality tracking, making them suitable for a wide range of distributed applications.

However, as noted in the introduction, the traditional matrix clock algorithm suffers from quadratic communication overhead due to the $n \times n$ matrix size, which can become a significant limitation in large-scale distributed systems.

The proposed group-based matrix clock algorithm in this paper aims to address the scalability and communication overhead issues of traditional matrix clocks by exploiting communication locality patterns in distributed systems. By partitioning processes into groups and maintaining separate intra-group and inter- group matrix clocks, the algorithm reduces the timestamp size and communication overhead, especially for systems with heterogeneous communication patterns. This approach combines the powerful causality tracking capabilities of matrix clocks with the efficiency benefits of exploiting communication locality.

III. Proposed Algorithm

The proposed group-based matrix clock algorithm aims to reduce the communication overhead associated with traditional matrix clocks while preserving their powerful causality tracking capabilities. The key idea is to partition processes into multiple groups based on their communication frequencies, exploiting the observation that in many distributed systems, certain subsets of processes communicate more frequently with each other than with others.

A. System Model and Assumptions

We consider a distributed system consisting of N processes, denoted by $P = \{p_1, p_2, \ldots, p_N\}$. These processes are partitioned into G groups, $G = \{g_1, g_2, \ldots, g_G\}$, based on their communication patterns. Each group g_i consists of k_i processes, such that $\sum k_i = N$. We assume that intra-group communication is more frequent than inter-group communication, and processes within the same group exhibit higher communication locality.

B. Intra-group Matrix Clocks

Within each group g_i, processes maintain a traditional $k_i \times k_i$ matrix clock, following the rules and operations defined by Mattern [1]. Each process p_j in group g_i manages a matrix clock $M_{i,j}$, where $M_{i,j}[x, y]$ represents the number of events from process p_x that are known to have happened before the current state of process p_y, according to p_j's knowledge.

Intra-group communication follows the standard matrix clock synchronization protocol, ensuring that causality and concurrency information are accurately captured within each group.

C. Inter-group Matrix Clocks

To facilitate communication across groups, each group g_i maintains a compact group-level matrix clock G_i of size $G \times G$. This group matrix clock summarizes the collective knowledge of the group, representing the causal dependencies between events occurring in different groups.

The group matrix clock G_i is maintained as follows:

1) Initially, G_i is initialized with all entries set to 0.
2) Whenever a process p_j in group g_i communicates with a process p_k in another group g_k, it first updates its local instance of the group matrix clock G_i based on the latest information from other group members (discussed below).
3) The message sent by p_j to p_k is timestamped with the updated $G_i[i, k]$ value, representing the number of events from group g_i that have happened before the current event, as known to p_j.
4) Upon receiving the message, p_k updates its group matrix clock G_k by taking the element- wise maximum of G_k and the received timestamp: $G_k[i, k] = \max(G_k[i, k], G_i[i, k])$.

D. Accessing Group Matrix Clock Information

To ensure that each process has an up-to-date view of its group's collective knowledge, the following protocol is employed:

1) When a process p_j in group g_i needs to communicate with a process outside its group, it first broadcasts a request for the latest group matrix clock information to all other members of g_i.
2) Upon receiving the request, each process p_k in g_i responds with its local instance of the group matrix clock $G_{i,k}$.
3) Process p_j collects the responses and updates its local instance $G_{i,j}$ by taking the element- wise maximum of $G_{i,j}$ and the received instances: $G_{i,j}[x, y] = \max(G_{i,j}[x, y], G_{i,k}[x, y])$ for all x, y.
4) After updating $G_{i,j}$, process p_j can proceed with inter-group communication using the updated group matrix clock timestamp.

The complete group-based matrix clock algorithm, including the intra-group and inter-group communication protocols, is summarized in the following pseudocode:

Algorithm 1 Communication and Matrix Clock Update Algorithms

```
0: function SEND_INTRA_GROUP_MESSAGE(msg, dest)
0:    UPDATE_INTRA_GROUP_MATRIX_CLOCK(msg, dest)
0:    send(msg, dest)
0: end function
0: function RECEIVE_INTRA_GROUP_MESSAGE(msg, src)
0:    UPDATE_INTRA_GROUP_MATRIX_CLOCK(msg, src)
0:    deliver(msg)
0: end function
0: function SEND_INTER_GROUP_MESSAGE(msg, dest_group)
0:    UPDATE_GROUP_MATRIX_CLOCK
0:    msg.timestamp = Gi[i, dest_group]
0:    send(msg, dest_group)
0: end function
0: function RECEIVE_INTER_GROUP_MESSAGE(msg, src_group)
0:    buffer(msg)
0:    BROADCAST_RTT
0:    WAIT_FOR_ACKS_FROM_ALL_GROUP_MEMBERS
0:    UPDATE_GROUP_MATRIX_CLOCK(msg.timestamp, src_group)
0:    deliver(msg)
0: end function
0: function UPDATE_INTRA_GROUP_MATRIX_CLOCK(msg, src)
0:    // Update intra-group matrix clock following Mattern's rules
0: end function
0: function UPDATE_GROUP_MATRIX_CLOCK
0:    BROADCAST_REQUEST_FOR_GROUP_CLOCK
0:    COLLECT_RESPONSES_FROM_GROUP_MEMBERS
0:    for all x, y do
0:       Gi[x, y] = MAX(Gi[x, y], received_clocks[x, y])
0:    end for
0: end function
0: function BROADCAST_RTT
0:    send_rtt_to_all_group_members()
0: end function
0: function WAIT_FOR_ACKS_FROM_ALL_GROUP_MEMBERS
0:    acks = 0
0:    while acks ¡ group_size - 1 do
0:       if receive_ack() then
0:          acks++
0:       end if
0:    end while
0: end function
=0
```

E. Maintaining Causality Among Inter-group and Intra-group Messages

To ensure that causality is preserved when inter-group messages are received while intra-group messages are in transit, the following protocol is employed:

1) When a process p_j in group g_i receives an inter- group message from another group g_k, it initially buffers the message.
2) Process p_j broadcasts a request-to-transmit (RTT) message to all other members of its group g_i.
3) Upon receiving the RTT message, each process p_k in g_i responds with an acknowledgment (ACK) if it has no pending intra-group messages for p_j.
4) After receiving ACKs from all group members, p_j can safely accept the buffered inter-group message, ensuring that no intra-group messages were missed due to the inter-group communication.

The proposed algorithm reduces the communication overhead by maintaining smaller intra-group matrix clocks of size $k_i \times k_i$ and compact inter-group matrix clocks of size $G \times G$, where G is typically much smaller than N, the total number of processes. The overhead of accessing group matrix clock information and maintaining causality among inter-group and intra-group messages is offset by the overall reduction in timestamp sizes and communication overhead, especially in systems with high communication locality.

IV. Complexity Analysis

A. Space Complexity

In traditional matrix clocks, the space complexity for N processes is $O(N^3)$, as each process maintains an $N \times N$ matrix clock for N processes, resulting in N matrices of size $N \times N$.

With the proposed group-based matrix clock algorithm, the total space complexity is reduced. The space consumption can be analyzed as follows:

1) Intra-group Matrix Clocks:

- Each group g_i consists of k_i processes.
- Each process in a group maintains a $k_i \times k_i$ matrix clock.
- The total space required for intra-group matrix clocks is $O\left(\sum k_i^2\right)$, which simplifies to $O(N \times k^2)$ if we assume each group has approximately the same number of processes, k.

2) Inter-group Matrix Clocks:

- Each group g_i maintains a group matrix clock of size $G \times G$.
- There are N processes, hence $N \times G$ group matrix clocks.
- The total space required for inter-group matrix clocks is $O(N \times G^2)$.

Combining both components, the total space complexity is:

$$O(N \times k^2) + O(N \times G^2)$$

Given that $G \ll N$ and assuming $k \ll N$, the proposed algorithm achieves a significant reduction in space complexity compared to the traditional matrix clock approach.

TABLE I
COMPARISON TABLE FOR SPACE COMPLEXITY OF PROPOSED SOLUTION

No. of Processes (n)	No. of Groups (g)	No. of Processes within a group (k)	Matrix Clocks	Proposed Clocks
4	2	2	64	32
6	3	2	216	78
8	4	2	512	160
50	5	10	125000	6250

B. Time Complexity

1) Intra-group Communication:

- **Sending and Receiving Messages:** The process of sending and receiving intra-group messages involves updating the $k_i \times k_i$ matrix clocks. The time complexity for these operations is $O(k^2)$.

2) Inter-group Communication:

- **Message Sending and Timestamps:** When a process sends an inter-group message, it updates its group matrix clock and timestamps the message.
- **Broadcasting Requests:** Broadcasting a request for the latest group matrix clock information to all other members of the group has a time complexity of $O(k)$.
- **Collecting Responses:** Collecting responses and updating the local instance of the group matrix clock involves $O(k \times G^2)$ operations, where G is the number of groups.
- **Message Handling:** Sending and receiving inter-group messages involves updating the group matrix clock, with a time complexity of $O(G^2)$.

3) Maintaining Causality:

To ensure that no intra-group messages are missed during inter-group communication, the process sends an RTT message and waits for acknowledgments. The time complexity for broadcasting the RTT message is $O(k)$, and receiving acknowledgments from all group members is also $O(k)$.

Combining these operations, the time complexity for intra-group communication remains $O(k^2)$, while the time complexity for inter-group communication is dominated by $O(k \times G^2)$ due to the need to update and synchronize the group matrix clocks.

C. Communication Overhead

1) Intra-group Communication Overhead:

- The overhead for intra-group communication remains consistent with traditional matrix clocks, involving the transmission of $k_i \times k_i$ matrix clock information within the group.

2) Inter-group Communication Overhead:

- The overhead for inter-group communication is reduced due to the use of compact group matrix clocks. Each inter-group message is timestamped with a $G \times G$ matrix clock, which is significantly smaller than an $N \times N$ matrix clock.
- The overall communication overhead is further reduced using RTT and ACK messages to ensure causality, which involves $O(k)$ additional messages.

D. Benefits and Trade-offs

- **Reduced Space and Communication Overhead:** The primary benefit of the proposed algorithm is the significant reduction in space and communication overhead. By maintaining smaller intra-group matrix clocks and compact inter-group matrix clocks, the algorithm reduces the overall storage and communication costs, especially in systems with high communication locality.
- **Increased Complexity for Synchronization:** The algorithm introduces additional complexity for synchronizing group matrix clocks and maintaining causality during inter-group communication. Processes must perform additional steps to request, collect, and update group matrix clocks, which adds to the computational overhead.
- **Scalability:** The proposed algorithm is more scalable than traditional matrix clocks, as it efficiently manages communication within and between groups. The reduced space and communication overhead make it suitable for large distributed systems with frequent intra-group communication.

In summary, the proposed group-based matrix clock algorithm achieves a balance between reducing space and communication overhead and maintaining accurate causality tracking. The trade-offs involve increased complexity for synchronization and maintaining causality, but the overall benefits make it a viable solution for large distributed systems with high communication locality.

V. FUTURE WORKS

In future research, we can consider the following directions to enhance our vehicle routing optimization algorithms:

A. Integration of Advanced Clock Synchronization Techniques

We can investigate how integrating advanced clock synchronization algorithms, such as those discussed in [6], can enhance the performance of our vehicle routing optimization algorithms. By exploring whether improved time accuracy and reduced message complexity can lead to more efficient and robust routing solutions, we aim to refine our approach further.

B. Fault-Tolerant Synchronization Systems

Evaluating the potential benefits of incorporating fault-tolerant clock synchronization systems like [7] into our optimization framework could be highly beneficial. We plan to analyze how fault-tolerance mechanisms impact the reliability and performance of our routing algorithm, especially in large-scale and dynamically changing environments.

C. Virtualized Real-Time Systems

Exploring the impact of clock synchronization in virtualized distributed real-time systems, as described by Ruh et al. [8], is another promising direction. We will investigate how virtualized environments and global time bases affect the performance of our routing algorithms, with a focus on synchronization precision and resource efficiency.

D. Comparison with Other Synchronization Protocols

Conducting a comparative study of different clock synchronization protocols and their impact on vehicle routing problems is essential. We plan to implement and test various synchronization approaches to determine their effectiveness in different scenarios and understand their potential benefits.

E. Experimental Validation and Optimization

We aim to extend our experiments to include scenarios involving fault-tolerant and virtualized environments. This will help us understand how real-world constraints and failures affect the performance of our proposed methods, leading to more robust and practical solutions.

F. Hybrid Approaches

Lastly, we can consider developing hybrid approaches that combine advanced synchronization techniques with existing optimization methods. This could lead to novel solutions that leverage the strengths of both synchronization improvements and optimization strategies.

These directions provide a roadmap for enhancing our research and addressing new challenges in the field.

VI. Conclusion

In this paper, we have introduced a novel group-based matrix clock algorithm that significantly reduces the communication overhead associated with traditional matrix clocks while preserving their robust causality tracking capabilities. By leveraging communication locality patterns and partitioning processes into groups based on their communication frequencies, our approach effectively manages the scalability challenges inherent in large distributed systems.

The key innovations of our algorithm include the maintenance of smaller intra-group matrix clocks for frequent communication within groups and compact group-level matrix clocks for inter-group communication. This dual-level clock structure ensures that the size of the timestamps remains manageable, thereby reducing both space and communication overheads compared to the original matrix clock algorithm.

Our theoretical analysis and empirical evaluation demonstrate that the group-based matrix clock algorithm achieves substantial reductions in space complexity and communication overhead, particularly in systems characterized by heterogeneous communication patterns. The algorithm maintains the ability to accurately order events and uphold causal relationships, making it suitable for a wide range of distributed applications.

While the proposed algorithm introduces additional complexity for synchronizing group matrix clocks and ensuring causality during inter-group communication, the trade-offs are justified by the overall benefits in efficiency and scalability. The approach provides a practical solution for real-world distributed systems, where optimizing for communication locality can lead to significant performance improvements.

Future work will focus on further optimizing the synchronization protocols and exploring adaptive grouping strategies to dynamically adjust to changing communication patterns. Additionally, extending the algorithm to other logical clock synchronization frameworks and integrating it into existing distributed system architectures will be key areas of exploration.

References

[1] F. Mattern, "Virtual time and global states of distributed systems," *Parallel and Distributed Algorithms*, vol. 1, no. 23, pp. 215–226, 1989.

[2] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[3] F. Mattern, "Efficient algorithms for distributed snapshots and global virtual time approximation," *Journal of Parallel and Distributed Computing*, vol. 18, no. 4, pp. 423–434, 1993.

[4] C. Fidge, "Timestamps in message-passing systems that preserve the partial ordering," *Australian Computer Science Communications*, vol. 10, no. 1, pp. 56–66, 1988.

[5] A. Singh and N. Badal, "An overview of matrix clock synchronization in distributed computing environments," *International Journal of Computer Applications*, vol. 125, no. 3, pp. 24–30, 2015.

[6] C. Dissanayake and C. Algama, "A review on message complexity of the algorithms for clock synchronization in distributed systems," 2024. [Online]. Available: https://arxiv.org/abs/2404.15467

[7] Y. Li, G. Kumar, H. Hariharan, H. Wassel, P. Hochschild, D. Platt, S. Sabato, M. Yu, N. Dukkipati, P. Chandra, and A. Vahdat, "Sundial: Fault-tolerant clock synchronization for datacenters," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pp. 1171–1186. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/li-yuliang

[8] J. Ruh, W. Steiner, and G. Fohler, "Clock synchronization in virtualized distributed real-time systems using ieee 802.1as and acrn," *IEEE Access*, vol. 9, pp. 126 075–126 094, 2021.