Distributed Systems

P4: Synchronization

by
Asger Song Høøck Poulsen
Firas Harbo Saleh

A Distributed Systems
Project

December 5, 2023

Contents

1 Introduction **1**

2 Methods and Materials **1**
2.1 Lamport Timestamp 1
2.1.1 Happens-before 2
2.1.2 Lamport Timestamp Algorithm 2
2.2 Vector Clock 3
2.3 Development and Environment Tools 3

3 Experiments, Results and Discussion **3**
3.1 Discussion 3

4 Conclusion and perspectives **3**
4.1 Conclusion 3
4.2 Lessons Learned 3
4.3 Future Work 3

1 Introduction

In the expansive field of distributed systems, the current project embarks on an insightful journey to explore and elucidate the fundamental concepts of logical clock algorithms, specifically focusing on Lamport Timestamps and Vector Clocks. This project aims to design, implement, test, and compare these algorithms, emphasizing their ability to order events in a distributed system with accuracy and efficiency.

The core challenge of this project lies in the intricate analysis and optimization of two pivotal logical clock algorithms - Lamport Timestamps and Vector Clocks. Our focus is twofold: firstly, to ensure the correctness of event ordering, and secondly, to optimize the overhead in terms of time, space, and message complexities. The journey encompasses a thorough process that begins with a detailed understanding of the algorithms, followed by a robust implementation in Python. The project progresses with rigorous testing and evaluation, comparing these algorithms against each other and benchmarking them against the state of the art.

Through meticulous research, development, and analytical scrutiny, this project endeavors to contribute a comprehensive understanding of these algorithms. It seeks to provide clear insights into their operational mechanics, effectiveness in distributed environments, and the potential areas where they can be applied or further developed.

2 Methods and Materials

2.1 Lamport Timestamp

The concept of Lamport Timestamps, introduced by Leslie Lamport[1], serves as a cornerstone in the realm of distributed systems for establishing a partial ordering of events. At the heart of this algorithm lies a simple yet powerful idea: using logical clocks — counters that are not tied to physical time — to sequence events across different processes in a distributed environment.

Lamport Timestamps operate on the principle that each process in a distributed system maintains its own logical clock. When an event occurs, be it a message send or receive, or an internal event, the

clock is incremented. The elegance of this system is its relative simplicity and the minimal overhead it incurs, making it a foundational approach in the study of distributed systems.

2.1.1 Happens-before

To establish synchronization among logical clocks in distributed systems, Leslie Lamport introduced the fundamental concept of happens-before, a crucial relation in Lamport Timestamps. This relation, denoted by $a \rightarrow b$, defines a chronological order between two events, stating that event a happens before event b. This relation is transitive, meaning that $\forall a, b, c$ if $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$. The happens-before relation is also irreflexive, meaning that $\forall a$: $a \nrightarrow a$, and antisymmetric, meaning that $\forall a, b$: $a \neq b$, if $a \rightarrow b$ then $b \nrightarrow a$.

The happens-before relation is used to establish a partial ordering of events in a distributed system. The ordering is established by comparing the timestamps of two events. If $a \rightarrow b$, then $C(a) < C(b)$, where $C(a)$ denotes the timestamp of event a. However, it is important to note that the converse, if $C(a) < C(b)$, then $a \nrightarrow b$, is not necessarily true. This is because the happens-before relation is a partial ordering, meaning that it is not necessarily true that $a \rightarrow b$ or $b \rightarrow a$. When it is the cast that $a \nrightarrow b$ and $b \nrightarrow a$, the two events are said to be concurrent.

2.1.2 Lamport Timestamp Algorithm

Considering the happens-before relation, Lamport Timestamps can be defined as follows: *The timestamp of an event is the maximum of its own timestamp and the timestamps of all events that happen-before it, plus one.* This definition can be expressed as the following equation:

$$C(e) = max(C(e), C(e')) + 1 \tag{1}$$

where $C(e)$ denotes the timestamp of event e, and $C(e')$ denotes the timestamp of the event that happens-before e. This equation is used to update the timestamp of an event when it occurs. The timestamp of an event is initialized to zero, and is incremented by one when an event occurs. The timestamp of an event is also included in messages sent between processes, and is used to update the timestamp of the receiving process.

Consider the following example, where three processes, P_1, P_2 and P_3, communicate with each other depcited in figure 1. The processes run on different machines, and each process has its own logical clock. The clocks run at different rates. P_1 is incremented by 2 units, 5 units in process P_2, and 10 units in process P_3, respectively.

The first example in figure 1 shows the clocks of the three processes. Consider message m_3 which leaves P_2 at 80 and arrives at P_2 at 40. Similarly, m_4 from P_2 to P_1 leaves at 50 and arrives at 22. These values are clearly impossible in a real system, as the clocks are not synchronized. This is where Lamport Timestamps come into play. The second example shows the clocks after the Lamport Timestamp algorithm has been applied. The algorithm follows the happens-before relation to synchronize the clocks. Since m_3 left at 80, it must arrive at 81 or later. To ensure this, each message carries the sending time according to its sender's clock.

To implement the Lamport Timestamp algorithm, each process maintains a local counter C_i which are updated according to the following steps[2]:

1. When a process P_i sends a message, it increments its counter C_i by one, and attaches the new value to the message.

2. When a process P_i receives a message, it sets its counter C_i to the maximum of its current value and the value in the received message, and increments its counter by one.

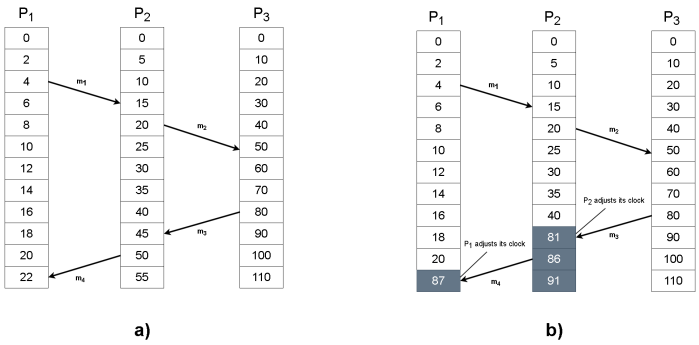

Figure 1: (a) Three processes, each with its own (logical) clock. The clocks run at different rates. (b) Lamport's algorithm synchronizes the clocks.

3. When a process P_i experiences an internal event, it increments its counter C_i by one.

2.2 Vector Clock

2.3 Development and Environment Tools

3 Experiments, Results and Discussion

3.1 Discussion

4 Conclusion and perspectives

4.1 Conclusion

4.2 Lessons Learned

4.3 Future Work

References

[1] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, 1978.

[2] M. Raynal and M. Singhal, "Logical time: Capturing causality in distributed systems," *Computer*, 1996.

[3] M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed. Maarten van Steen, 2023, ch. 5.

[4] Kubernetes, "Kubernetes documentation / service," 2023, last accessed 3 December 2023. [Online]. Available: https://kubernetes.io/docs/concepts/services-networking/service/#headless-services.