LOGICAL AND VECTOR CLOCKS USED IN DISTRIBUTED SYSTEMS

Dr. Shamsudeen E

Assistant Professor of Computer Applications,
EMEA College of Arts and Science, Kondotty, Kerala, India.

Abstract

In distributed systems, there is no global clock exists rather it uses logical clocks to synchronize the events in the system. When designing or analyzing distributed systems, the logical nature of time is much significant.This concept put forward by Lamport to deal with the logical time. It is fully depend upon causality relation among events. This paper also look into the vector time which is also proposed to capture causality of between events of distributed system. In order to explain things in detail this paper also look into the implementation details of the Lamport's algorithm.

Key words : distributed systems,message passing, causality, logical time, happened before, vector time.

Introduction

A distributed system[1] as one in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages[2]. This simple definition covers the entire range of systems in which networked computers can usefully be deployed. Computers that are connected by a network may be spatially separated by any distance. They may be on separate continents, in the same building or in the same room.

The definition of distributed systems has the following significant consequences:

Concurrency[3]: In a network of computers, concurrent program execution is the norm. I can do my work on my computer while you do your work on yours, sharing resources[4] such as web pages or files when necessary. The capacity of the system to handle shared resources can be increased by adding more resources (for example, computers) to the network.

No global clock[5]:

For thousands of independent machines running concurrently[6] that may span multiple time zones and continents, some types of synchronization[7] or coordination must be given for these machines to collaborate efficiently with each other (so that they can appear as one). When programs need to cooperate they coordinate their actions by exchanging messages. Close coordination often depends on a shared idea of the time at which the programs' actions occur. But it turns out that there are limits to the accuracy with which the computers in a network can synchronize their clocks – there is no single global notion of the correct time.

In Distributed systems there is no centralized time server present. It prefers logical clock to synchronize the events that happened when the message passing among processors happens.

Logical Clocks

Logical Clocks[8] refer to implementing a protocol on all machines within your distributed system, so that the machines are able to maintain consistent ordering of events within some virtual time span. A logical clock is a mechanism for capturing chronological and causal relationships in a distributed system. Distributed systems may have no physically synchronous global clock[9], so a logical clock allows global ordering on events from different processes in such systems.

Suppose, we have more than 10 PCs in a distributed system and every PC is doing it's own work but then how we make them work together. There comes a solution to this i.e. LOGICAL CLOCK.

Approach 1

To order events across process, try to sync clocks in one approach.

This means that if one PC has a time 2:00 pm then every PC should have the same time which is quite not possible. Not every clock can sync at one time. Then we can't follow this method.

Approach 2

Another approach is to assign Timestamps to events.

In 20 computers, if we give each PC their individual number than it will be organized in a way that 1st computers will complete its process first and then second computer and so on.

But, this Timestamps method will only work as long as they obey causality.

Causality is fully based on HAPPEN BEFORE RELATIONSHIP[10].

- Taking single PC only if 2 events A and B are occurring one by one then TS(A) < TS(B). If A has timestamp of 1, then B should have timestamp more than 1, then only happen before relationship occurs.
- Taking 2 PCs and event A in P1 (PC.1) and event B in P2 (PC.2) then also the condition will be TS(A) < TS(B). Taking example- suppose you are sending message to someone at 2:00:00 pm, and the other person is receiving it at 2:00:02 pm.Then it's obvious that TS(sender) < TS(receiver).

Properties Derived from Happen Before Relationship

- **Transitive Relation**
  If, TS(A) <TS(B) and TS(B) <TS(C), then TS(A) < TS(C)
- **Causally Ordered Relation**
  a->b, this means that a is occurring before b and if there is any changes in a it will surely reflect on b.
- **Concurrent Event**
  This means that not every process occurs one by one, some processes are made to happen simultaneously i.e., A || B.

Lamport's Logical Clock

It is a procedure to determine the order of events occurring. It provides a basis for the more advanced Vector Clock Algorithm. Due to the absence of a Global Clock in a Distributed Operating System Lamport Logical Clock[11] is needed and widely used.

Algorithm:

Happened before relation(→): a → b, means 'a' happened before 'b'.

Logical Clock: The criteria for the logical clocks are:

[C1]: Ci (a) < Ci(b), [Ci → Logical Clock, If 'a' happened before 'b', then time of 'a' will be less than 'b' in a particular process.]

[C2]: Ci(a) < Cj(b), [Clock value of Ci(a) is less than Cj(b)]

Reference:

Process: Pi

Event: Eij, where i is the process in number and j: **j^{th}** event i the **i^{th} process**.

tm: vector time span for message m.

Ci vector clock associated with process **Pi**, the **j^{th}** element is **Ci[j]** and contains **Pi's** latest value for the current time in process **Pj**.

d: drift time, generally d is 1.

Implementation Rules[IR]:

[IR1]: If a → b ['a' happened before 'b' within the same process] then, **Ci(b) =Ci(a) + d**

[IR2]: Cj = max(Cj, tm + d) [If there's more number of processes, then tm = value of Ci(a), Cj = max value between Cj and tm + d]

For Example:

Take the starting value as 1, since it is the 1st event and there is no incoming value at the starting point (see in figure 1):

e11 = 1

e21 = 1

The value of the next point will go on increasing by d (d = 1), if there is no incoming value i.e., to follow [IR1].

e12 = e11 + d = 1 + 1 = 2

e13 = e12 + d = 2 + 1 = 3

e14 = e13 + d = 3 + 1 = 4

e15 = e14 + d = 4 + 1 = 5

e16 = e15 + d = 5 + 1 = 6

e22 = e21 + d = 1 + 1 = 2

e24 = e23 + d = 3 + 1 = 4

e26 = e25 + d = 6 + 1 = 7

When there will be incoming value, then follow **[IR2]** i.e., take the maximum value between **Cj** and **Tm + d**.

e17 = max(7, 5) = 7, [e16 + d = 6 + 1 = 7, e24 + d = 4 + 1 = 5, maximum among 7 and 5 is 7]

e23 = max(3, 3) = 3, [e22 + d = 2 + 1 = 3, e12 + d = 2 + 1 = 3, maximum among 3 and 3 is 3]

e26 = max(5, 6) = 6, [e24 + 1 = 4 + 1 = 5, e15 + d = 5 + 1 = 6, maximum among 5 and 6 is 6]

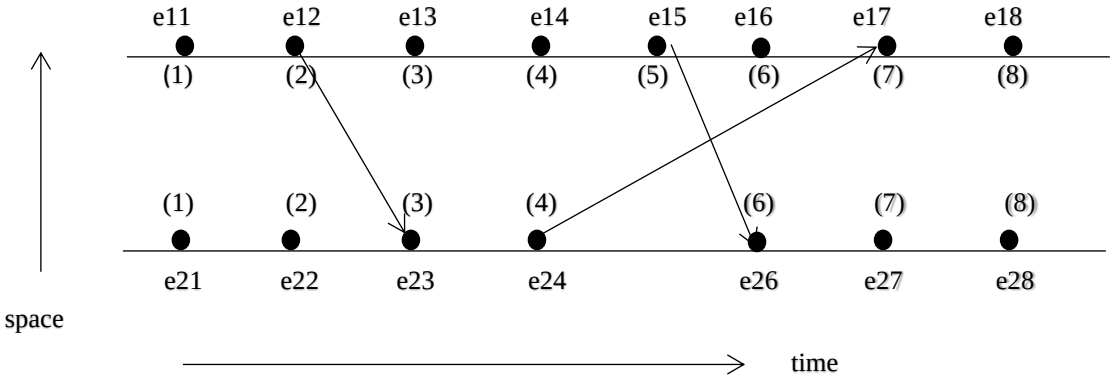

Figure 1

Limitation:

In case of [IR1], if a → b, then C(a) < C(b) → true.

In case of [IR2], if a → b, then C(a) < C(b) → May be true or may not be true.

Merits

Logical Clocks refer to implementing a protocol on all machines within your distributed system, so that the machines are able to maintain consistent ordering of events within some virtual times span. A logical clock is a mechanism for capturing chronological and causal relationships in a distributed system. Distributed systems may have no physically synchronous global clock, so a logical clock allows global ordering on events from different processes in such systems.

Demerits

There is no way to have perfect knowledge on ordering of events – A "true" ordering may not exist..

Vector Clock

It is an algorithm that generates partial ordering of events and detects causality violations in a distributed system. These clocks expand on Scalar time[12] to facilitate a casually consistent view of the distributed system, they detect whether a contributed event has caused another event in the distributed system. It essentially captures all the casual relationships. This algorithm helps us label every process with a vector(a list of integers) with an integer for each local clock of every process within the system. So for N given processes, there will be vector/ array of size N.

Vector Clock algorithm

- Initially, all the clocks are set to zero.
- Every time, an Internal event occurs in a process, the value of the process's logical clock in the vector is incremented by 1
- Also, every time a process sends a message, the value of the process's logical clock in the vector is incremented by 1.

Every time, a process receives a message, the value of the process's logical clock in the vector is incremented by 1, and moreover, each element is updated by taking the maximum of the value in its own vector clock and the value in the vector in the received message (for every element).

Example 1

Consider a process (P) with a vector size N for each process (see in figure 2): the above set of rules

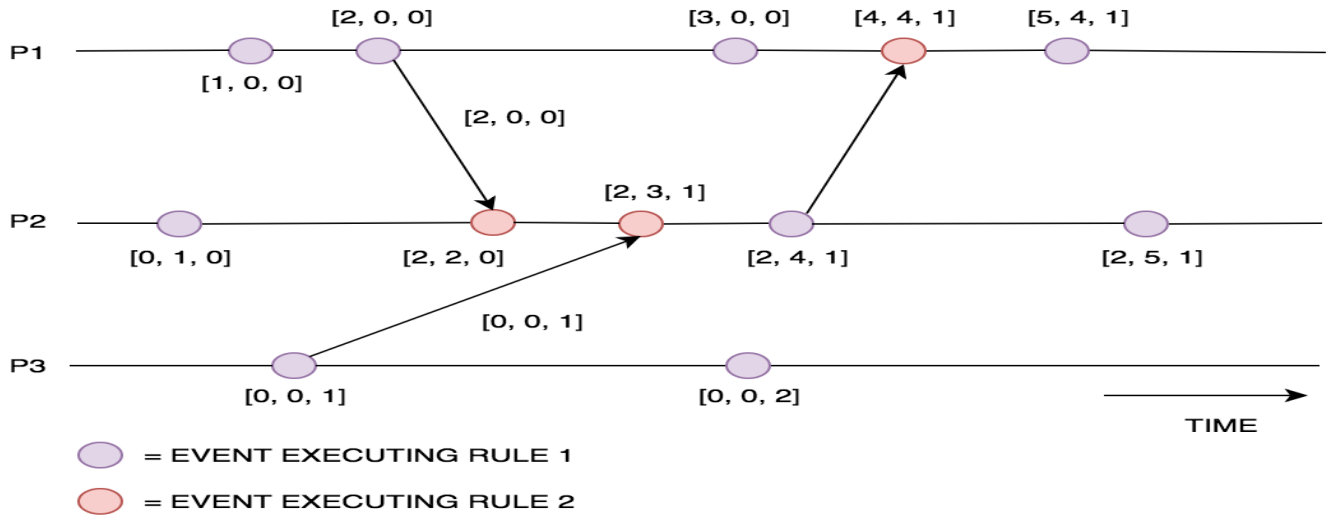

Figure 2.

mentioned are to be executed by the vector clock:
The above example depicts the vector clocks mechanism in which the vector clocks are updated after execution of internal events, the arrows indicate how the values of vectors are sent in between the processes (P1, P2, P3).

Example 2
To sum up, Vector clocks algorithms are used in distributed systems to provide a **casually consistent** ordering of events but the entire Vector is sent to each process for every message sent, in order to keep the vector clocks in sync.

Conclusion
In conclusion, we can see that when we go to a distributed system maintaining order becomes a difficult problem. In order to solve this we have to create an artificial view of time, known as **Logical Time**. There are many solutions that provide an implementation of **Logical Time**. **Vector Clocks** then build upon this technique by separating local and global clocks of processes, meaning event ordering can be made causally consistent. Furthermore, improvements are made upon **Vector Clocks** to improve their performance at the cost of needing greater storage space or requiring analysis of causality to be performed offline.

References

[1] Pradeep K Sinha: Distributed Systems: Concepts and Design. 2012
[2] Maarten van Steen, Andrew S. Tanenbaum: Distributed Systems. 2017.
[3] G. F. Coulouris, J. Dollimore, Distributed Systems. Concepts and Design. Addison-Wesley Publishing Company.
[4] A. Silberschatz, Operating System Concepts, Addison-Wesley Publishing Company, 1985.
[5] Barbara Liskov, Liuba Shrira, John Wroclawski: Efficient At-Most-Once Messages Based on Synchronized Clocks. 1991.
[6] A. Tanenbaum, R. Van Renesse, Distributed Operating Systems, Computing Surveys, Vol. 17, No.4, December 1985
[7] Barbara Liskov: Practical uses of synchronized clocks in distributed systems. 1993.
[8] Michel Raynal:About logical clocks for distributed systems:ACM SIGOPS Operating Systems ReviewJanuary 1992
[9] Ricardo Gusella, Stefano Zatti: The Accuracy of the Clock Synchronization Achieved by TEMPO in Berkeley UNIX 4.3BSD. 1989.
[10] Cary G. Gray, David R. Cheriton: Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency. 1989
[11] Leslie Lamport: Time, Clocks, and the Ordering of Events in a Distributed System. 1978.
[12] C. Fidge: Logical Time in Distributed Computing Systems: Computer Vol. 24. No. 8. Aug. 1991. pp 28-33.