INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact factor: 4.295
(Volume 4, Issue 2)
Available online at: www.ijariit.com

Logical clock implementation in the Distributed system

Sanchit Kumar Shukla
sanchitshukla@yahoo.co.in
Babasaheb Bhimrao Ambedkar Bihar University, Muzaffarpur, Bihar

Dr. P. K Sharan
sanchitshu@gmail.com
Babasaheb Bhimrao Ambedkar Bihar University, Muzaffarpur, Bihar

ABSTRACT

A distributed system consisting of system on network. The network provide the facility of information exchange between the systems. The client server systems are the basic concept for this type of system. In distributed system the communication delay are finite and unpredictable. The process does not share common memory and communication happen by passing message over communication network. The message failure is also the common problems on distributed system. The network data link layer resolve this problems by various algorithms such as

(1) Stop and Wait
(2) Stop and Wait for Noisy channels
(3) Automatic repeat request and positive Acknowledgement with Retransmission (ARQ OR PAR)
(4) Sliding window protocols
(5) Go Back-N protocol
(6) Protocol using selected repeat

But in this paper we discuss the Logical Clock approach of Distributed system.

Keywords: *Distributed computing, Vector clock, Matrix clock, Event ordering, Clock synchronization, Logical clock.*

1. INTRODUCTION

In distributed system another fundamental need is to design a log or causally ordering as asynchronous distributed computation make a progress in spurts and it turn out a need of logical time clock

By these Logical there are following benefits:

- Knowledge of casual precedence among events:- This events help ensure liveness and fairness in mutual exclusiveness algorithems and help to maintain deadlock
- Tracking of dependent event:- It benefited into construct consistent state for resuming in failure condition.
- Knowledge about Progress:-Knowledge of dependency among event help measure the progress of process in the distributed computation,and also useful in discardind obsolete information garbage collection and termination detection.
- Concurrency Measure:-The Logical Clock is useful in measuring the amount of concurrency in computation by assigning the Timestamp to them.

To implement Distributed system there are two approaches

(1) Order event across process and try to synchronies every clock which impossible

(2) Second approaches is to assigned Timestamp to event such that it obey causality i.e. Happen before relationship

Example:- Consider " A" and " B " are two events for process P in both figure (i) and (ii)

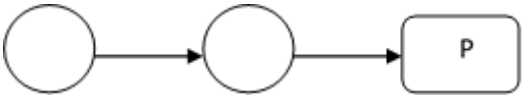


Fig (1)

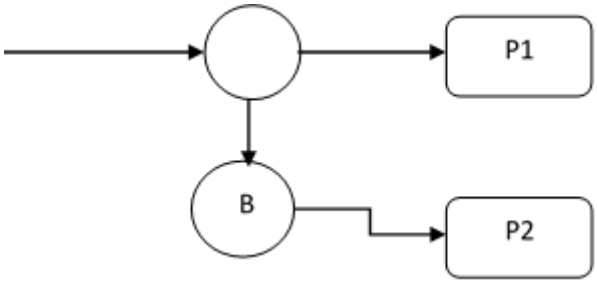


Fig (2)

Timestamp (A)< Timestamp(B)

[Ts(A) < Ts(B)]

Tuple for Logical Clock: Logical clock consisting of time Doman "T" and logical Clock " C "
The Doman element T consisting of partially order set over relation < which is called "Happen before or Causal Precedence which is Synonym with earlier than relation of Physical Time.

The Logical Clock C is a function that map an event "e" in distributed system to element in time doman T and denote as C(e) and it is called the Timestamp of e and noted as C:H T Such that following property satisfied

(1) → For Two event $\mathbf{e_i}$ and e_j ,$\mathbf{e_i} \rightarrow e_j \Rightarrow C(e_i) < C(e_j)$
"This is called clock consistent Condition"

(2) For two event $\mathbf{e_i}$ and $\mathbf{e_j}$; $\mathbf{e_i} \rightarrow e_j \Leftrightarrow C(e_i) < C(e_j)$

"This is called strong consistent Condition"

The happen before relation of Logical Clock follow

(a) Transitive Relation (b) causally orders relation (c) Concurrent Relation.

Transitive Relation:- Ts(A) < Ts(B) and Ts(B) < Ts(C)

Then Ts (A) < Ts(C)

Causally orders relation:- If A -> B i.e. if A happen before B then for any change in A its effect appear on B

Concurrent Relation:- If not A -> B i.e. if A not happen before B, And If not B->A i.e. if B not happen before A

Then **A || B i.e. A and B happen in Parallel.**

There are three ways Time Stamping for logical clock.

(1) Scalar/Linear Timestamp

(2) Vector Timestamp

(3) Matrix Timestamp

Scalar/Linear Timestamp:- The scalar time representation was proposed by Lamport in 1978

The Basic algorithms is as follow

- Each process use a local count integer whose initial value is Zero

- The process increment its value when it send or receive a instruction
- The counter is assigned to the event as its timestamp
- The send event carries its timestamp
- The reciver counter is updated as MAX((Local Clock,Msg Timestamp)+1)

Thus Rules **(1)** and **(2)** to update the clocks are as follows:
(1) Before executing an event (send, receive, or internal), process pi executes the following:
Ci = Ci+d where d > 0

In general, every time **(1)** is executed, *d* can have a different value, and this value may be application-dependent. However, typically *d* is kept at 1 because this is able to identify the time of each event uniquely at a process, while keeping the rate of increase of *d* to its lowest level.

(**2)** Each message piggybacks the clock value of its sender at sending time. When a process pi receives a message with timestamp Cmsg, it

Executes the following actions:

1. Ci = max(Ci,Cmsg);
2. Execute **(1)** ;
3. Deliver the message.

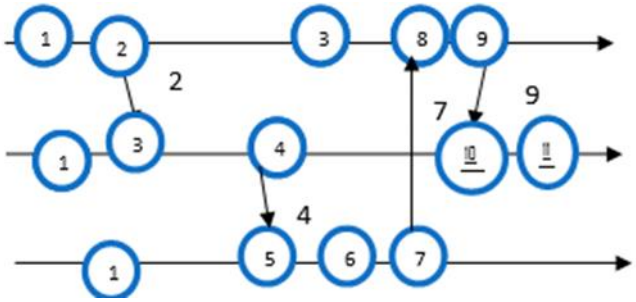


Fig (3)

Also the scalar clock can be used to total order event in distributed system. But the main problems occur when the timestamp of two event are equal

Also the scalar clock can be used to total order event in distributed system. But the main problems occur when the timestamp of two event are equal.

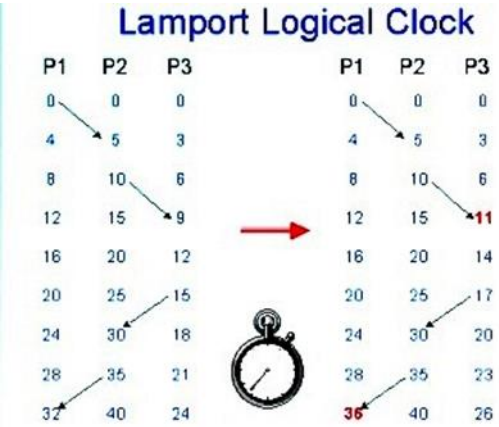


Fig (4)

For Two event e_i and e_j ; $e_i \longrightarrow e_j$ => $C(e_i) = C(e_j)$ => $e_i \| e_j$

That is the process P1 and Process P2 have equal Timestamp then this issue can be resolve on the basis of identifiers.The lower the identifier in ranking the higher the priority.As the timestamp is denoted by tuple(t,i)

T=time of occurrence and i = identifier of the process where it occur.The total order relation "$<$" on the two event X and Y with time Stamp(h,i) and (k,j) then it defined as follow $X<Y \Leftrightarrow (h<k$ or $(h=k$ and $I < j))$

Limitation of LAMPORT'S Clock

If A->B then it implies C(A) < C(B) But C(A) < C(B) does not A->B

Vector Clock

A number of researchers most notably the **MATTERN** and **FIDGE** Proposed the Vector Clock for the Time stamping mechanism for distributed computation.
Let us consider n parallel process each process having vector length (n x n).These vector are use to time stamping each process p main

Algorithms

1. Vector initialize to zero for each process Vi [j] = 0 for I,j=1234---N
2. Increment vector before Time stamp event Vi[i]= Vi[i]+1
3. Massage is sent from **Pi with Vi attached to it**
4. **When Pj recive message** VJ[i]=Max{Vi[i], VJ[i]} for all i

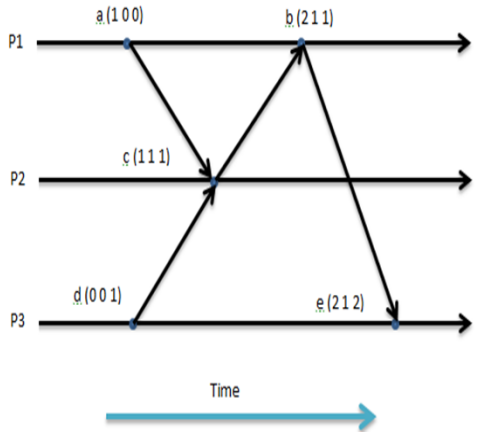
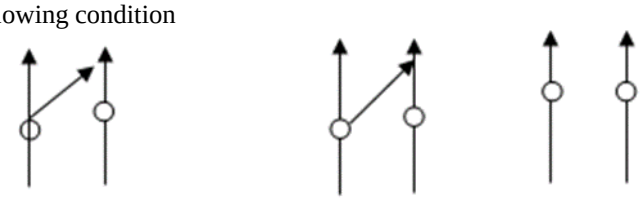


Fig (5)

Limitation of Vector Clock

1. For any two event a and b ;a->b
if and only if $V_i^a < V_j^b$
2. For all C such that **a->c** and **a->b** the it should follow $V_i^a < V_j^b$
That is no path from a to b in following condition



V_i^a =(2,0) V_i^b =(1,1) V_i^a =(1,0) V_i^b =(0,1) V_i^a =(1,0) V_i^b =(0,1)

Fig (6)

2. MATRIX CLOCK

Definition Matrix clocks is an extension of the vector Logical clocks that also contains the information about the other processes views of the system. In a system of matrix clocks, the time is represented by a set of n × n matrices of non-negative integers. A process p_i maintains a matrix mt_i [1...n, 1...n] mt_i [i,i] denotes the local logical clocks of P_j and

A. track the progress of the computation at process P_j. $mt_i[i,j]$ represent the latest knowledge that process p_i has about the local logical clock $mt_j(j,j)$ of process P_j that is knowledge of p_i has about the logical local clock of P_j.

B. The entire matrix mt_i [J,k] represent the knowledge that process p_i has about the last Knowledge that P_j contain about the Logical clock $mt_K(k,k)$ of P_k .

Note that row $mt_i[i,.]$ is nothing but the vector clock $vt_i[.]$ and exhibits all properties of vector clocks. Processes p_i uses the following rules Rule R1 and Rule R2 uses to update its clock

Rule R1 : Before executing an event and it update, its local logical time as follows: $mt_i[i,i] = mt_i[i,i]+d$ $(d > 0)$

Rule R2: Each message m is piggy-backed with matrix time $\mathbf{m_t}$. When $\mathbf{p_i}$ receives a message (m, m_t) from a process $\mathbf{p_j}$, $\mathbf{p_i}$ executes the following sequence of actions:

1. Update its logical global time as follows: $1 \leq k \leq n$: $\mathbf{mt_i}$ [i,k]=max: $\mathbf{mt_i}$ [i,k], : **mt**[j,k]) $1 \leq k,l \leq n$: $\mathbf{mt_i}$ [k,l]=max(:$\mathbf{mt_i}$[k,l],:**mt**[k,l])

2. Execute R1.

3. Deliver message m.

   Basic property: Clearly vector $\mathbf{mt_i}$ [i,.] contains all the properties of the vector clocks. In addition Matrix clocks have the following property: min($\mathbf{mt_i}$ [k,l]) ≥ t, where process pi knows that every other process p_k knows that p_i's local time has progressed till t If this is true, it is clear that process p_i know that all other processes know that pi will never send information with a local time ≤ t. In many applications, this implies that process will no longer require from pi certain information and can use this fact to discard obsolete information. If d is always 1 in the rule R1, then $\mathbf{mt_i}$ [k,l], denotes the number of events occurred at p_i and known at p_k as for as p_i's knowledge is concerned. Below show the detail implementation of Matrix Clock.

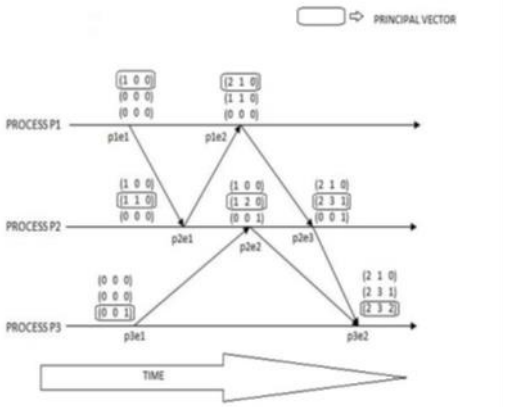


Fig. (7)

Now compute the feasibility of the proposed compaction technique. We define the efficiency of a compaction technique as the average percentage reduction in the size of the timestamp related information to be transferred in a message as compared to when sending the entire matrix timestamp. We also define the following Term **n**: the average number of entries in Ti that qualify for transmission in a message using the proposed technique.

b: the number of bits in a sequence number.
log2N: the number of bits needed to code N process ids.

The mattern / fidge clock require N.b bits of timestamp information, whereas the proposed technique require $\mathbf{(log2N+b).n^2}$ bits and vector timestamp technique require (log2N+b).n bits thus the efficiency of the technique is given by the following expression:

$$\left\{1 - \frac{(\mathbf{log2N+b}).n^2}{b.N}\right\} \times \mathbf{100\ \%}$$

Limitation: It is easy to see that the propped technique is beneficial only if n < N.b/(log2N+b)

3. CONCLUSION

In this paper we have described about the Logical Clock of distributed computations, and have examined its implementation and closure. We have lastly presented an efficient technique to maintain the matrix clock, which cuts down the communication overhead due to propagation of matrix timestamp by sending only incremental changes in the timestamp. The technique can reduce the communication overhead substantially if the interaction between processes is localised. The technique has a small memory overhead as memory is cheap now cheap it is not major problems. Thus this technique can use to reduce traffic on a communication network whose capacity is limited and is often the bottleneck.

4. REFRENCES

[1] K. Birman and T. Joseph, Reliable communication in presence of failures, *ACM Transactions on Computer Systems*, **3**, 1987, 47–76.
[2] K. M. Chandy and L. Lamport, Distributed snapshots: determining global states of distributed systems, *ACM Transactions on Computer Systems*, **3**(1), 1985, 63–75.
[3] A. Kshemkalyani, M. Raynal and M. Singhal, Global snapshots of a distributedsystem, *Distributed Systems Engineering Journal*, **2**(4), 1995, 224–233.
[4] L. Lamport, Time, clocks and the ordering of events in a distributed system,*Communications of the ACM*, **21**, 1978, 558–564.of agreement for asynchronous events, *Proceedings of the 5th Symposium on.Principles of Distributed Computing*, 1988, 197–209. S. M. Shatz, Communication mechanisms for programmingdistributed systems,*IEEE Computer*, 1984, 21–28.
[5] Matrix Clock Synchronization in the Distributed Computing Environment by Avaneesh Singh1, Neelendra Badal *Computer science and Engg. Department at Kamla Nehru Institute of Technology, Sultanpur, U.P.*

