Term Paper Review: Efficient Implementation of Vector Clocks

12CS10002 Aayush Singhal
12CS10042 Sabyasachi Behera
12CS30016 Bhushan Kulkarni
12CS30028 Rohit Agrawal

March 31, 2016

1 Introduction

In a distributed system, processes communicate via message passing. Events at a process may depend on the events at the same process and the events at other processes which communicate with it. Ordering of events is necessary for many applications like distributed mutual exclusion, distributed file consistency, causal broadcast etc. Due to distributed nature, events do not form a linear sequence but instead have a partial order. A partial order is defined on the set of all events, and two events are related to each other if one can potentially affect the other. The paper aims to reconstruct the partial order relationship, given a set of observed events.

2 Problem Statement

Directly happened before relation $<$ is defined as follows: for two events x and y, x directly happened before y, x$<$y iff (i) x and y are events in the same process and x occurs immediately before y or (ii) x is directly followed by the send of a message from process of x and y immediately follows the receipt of that message at process of y. The paper aims to extract the transitive closure of directly happened before relation $<_t$ over a set of events. Performance measures for algorithms:

- **Intrusion:** Intrusion measures the amount of information maintained at every process and the amount of extra information sent along with each message when a process communicates with another process. This information is later used to reconstruct the partial order relationship. If this information, maintained and sent, quantitatively is of the order of the number of processes, its strong intrusion, while if it is of constant size, then its weak intrusion.

- **Filtering:** Filtering measures the size of subset of events whose partial order can be found out. If for any subset partial order can be found, it is strong filtering.
- **Consistency:** Events for which partial order is to be determined may not be available at the process computing it. They may be sent to that process in any order. If for any subset of that set of events the partial order can be determined, it is strong consistency.

3 Solution Approach

In Lamport's method [1], a logical clock is maintained at every process that is incremented after each internal event. Every internal event and messages sent from a process, is timestamped with the current clock value. Upon receipt of the message the receiver process updates its clock value to the message timestamp if it is greater than its present clock value. Let C(x) denote logical clock value of event x. If $x <_t y$ then C(x)<C(y) but other way is not necessarily true. Concurrent events that do not affect each other might get different timestamps. Thus Lamport's approach only gives one of the possible orderings.

Fidge [2] used vector timestamps for clock values. Let n be the number of processes. An n dimensional logical clock T is kept at every process, all components initialised to zero. T(x) is the timestamp of an event x. At process i, T[i] is incremented for each internal event. T is appended to each message to be sent. Upon receipt of the message the receiver process updates T[j] to the message timestamp's j^{th} entry if it is greater than T[j]. At process P_i, $T_i[j]$ is the number of events that has happened at P_j before the last communication from P_j to P_i via any number of processes. For any two vector timestamps T(x) and T(y), $T(x) \neq T(y)$ iff $T(x)[k] \neq T(y)[k]$ for any k, and $T(x) \leq T(y)$ iff $T(x)[k] \leq T(y)[k]$ for all k. $T(x) < T(y)$ if $T(x) \leq T(y)$ and $T(x) \neq T(y)$. Ordering the events according to T gives the correct partial order relationship. But this method has strong intrusion since vector of size n in appended with each sent message. Present paper later discusses three approaches to implement vector clocks efficiently.

3.1 Transitive Dependency

At process P_i for each event x, transitive dependency approach maintains set of all predecessor events at each process P_j. Mukesh Singhal and Kshemkalyani's [3] approach reduces intrusion in vector clock implementation. The main intuition is to send only that information that has changed since the last message exchange. Every process maintains vectors T, LS and LU. For process P_i, $LS_i[j]$ is the local time when P_i last sent a message to P_j and $LU_i[j]$ is the local time when P_i updated $T_i[j]$. So if $LU_i[k] > LS_i[j]$, this means P_i has updated $T_i[k]$ after sending the last message to P_j and in the next message exchange with P_j , P_i should include $T_i[k]$ in the message overhead. $T_i[i]$ is incremented for every internal and send event. Upon receipt of a message, T is updated as before.

This approach requires FIFO channels since we always send along information that has changed since the last message exchange. So in the case of non FIFO channels, if newer messages reach the destination before older messages, the receiving process will miss out some information and incorrectly timestamp its events that occur between the receipt of the new and old message.

Since we maintain $<_t$ relationship at every event, we can reconstruct the partial order relationship for any subset of internal events. Therefore this method has strong filtering. Also since the timestamp of every event tells the number of events it depends upon at different processes, we only depend on the timestamp values and not on the order in which these events arrive for computing the partial order relationship. So we also have strong consistency. It has weak intrusion.

3.2 Direct Dependency

At process P_i for each event x, direct dependency approach stores a set of events y such that (i) y occurred before x at same process P_i or (ii) y occurred at process P_j and there exists a message sent from P_j after y which was received by P_i before x. Fowler and Zwaenepoel [4] approach maintains direct dependency among the events. While sending message, entry in T corresponding to self is appended with message. Upon receipt, entry in T corresponding to the sender is updated. To construct the entire partial order for event x at P_i, we invoke recursive call where the k^th recursive invocation updates the causal dependency vector if any process affects the given event through k hop message exchanges. After n recursions, where n is the number of processes, the causal dependency vector has the same value as the direct dependency vector associated with the most recent event at any process that affects the given event.

This approach suffers from both weak filtering and weak consistency. If there does not exist any internal event between receive and send of message exchange occurring between two events of two different processes, over more than one hop, partial order cannot be found out. This situation is called Invisible Interaction. This approach works for Non Invisible Interaction or NIVI. It has weak intrusion.

3.3 Pseudo-direct Dependency

At process P_i for each event x, pseudo-direct dependency approach stores a set of events y such that (i) y occurred before x at same process P_i or (ii) y occurred at process P_j and there exists a sequence messages, first sent from P_j after y, last was received by P_i before x and each intermediate receive directly happened before next send. Jard and Jourdan [5] method maintains the pseudo-direct predecessors for each event x. To get all the predecessors of an event x we have to recursively examine again the pseudo-direct predecessors of the events. When an internal event occurs, current dependencies are stored to that event and dependency record updated only to that event since we can look into records of that event to get previous dependencies. While sending message, current dependency records are appended to it. Upon receipt of message,

records are modified according to message record values. This method works without requirement of NIVI since message contains pseudo-direct dependencies information so that receiver process knows about other predecessors even if there is invisible interaction.

This method has strong filtering since partial order can be found out for any subset of events. It has weak consistency since the entire set of predecessors is not maintained. It has medium intrusion bounded by a number K since whenever upon the receipt of a message and after the union of timestamps, the size of dependency records becomes more than K, it makes a NULLEVENT to occur which brings down the size to one.

4 Comparison

	Filtering	Consistency	Intrusion
Transitive Dependency	Strong	Strong	Strong
Direct Dependency	Weak	Weak	Weak
Pseudo-direct Dependency	Strong	Weak	Medium

5 Conclusion

In distributed system, finding dependencies among the events is of crucial importance. In the present paper, authors have discussed different approaches to do the same by maintaining logical clock at different processes. They have also discussed the pros and cons of all the approaches.

6 Evaluation

6.1 Organization of the topics

Rating: 10/10
Topics are organized properly. The sequence introduction, problem statement with explanation about the concepts, solution approaches, comparison and conclusion is fine. Each topic contains relevant details.

6.2 Ease of understanding the algorithm from the description/psuedocode/example

Rating: 8/10
All Examples are explained properly. Pseudo-codes are cleanly written. Transitive dependency algorithm (section 3.1) is described properly. But direct dependency algorithm (section 3.2) needs more discussion to explain about the recursive function VisitEvent in pseudo-code. Also, pseudo-direct dependency approach (section 3.3) should contain the method to get all predecessors recursively. Other details about algorithms are explained properly.

6.3 Overall synergy between the different parts

Rating: 10/10
Paper looks homogeneous. All parts are written in similar way by following the same conventions and terminologies.

References

[1] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," Commun. ACM, vol. 21, pp. 558–565, July 1978.

[2] C. J. Fidge, "Timestamps in message-passing systems that preserve the partial ordering," Proceedings of the 11th Australian Computer Science Conference, vol. 10, no. 1, pp. 56–66, 1988.

[3] M. Singhal and A. Kshemkalyani, "An efficient implementation of vector clocks," Information Processing Letters, vol. 43, no. 1, pp. 47 – 52, 1992.

[4] J. Fowler and W. Zwaenepoel, "Causal distributed breakpoints," in Distributed Computing Systems, 1990. Proceedings., 10th International Conference on, pp. 134–141, May 1990.

[5] C. Jard, G. V. Jourdan, T. Jeron, and J. X. Rampon, "A general approach to trace-checking in distributed computing systems," in Distributed Computing Systems, 1994., Proceedings of the 14th International Conference on, pp. 396–403, Jun 1994.