Implementation of Lamport's Scalar clocks and Singhal-Kshemkalyani's VC, Algorithms (May 2007)

Saleh.M.Alnaeli
Kent State University
Computer Science Department
salnaeli@kent.edu

***Abstract*—The causality between events in distributed systems is one of the essential concepts for analyzing distributed operating systems. This relation in distributed systems is tracked using the logical time. This paper discusses some ways used for implementing the logical time, Lamport's Scalar clocks and Singhal-kshemkalyani's vector clock. Experiments were made on S-K vector clock and regular vector clocks to study their behavior under some different arguments such like, exchanged messages number, processes number, and involved processes number. We show that S-k achieves high efficiency with relatively low number of exchanged messages and low number of involved processes in the computation. We also show that the way a computation is constructed plays big role in choosing the technique that must be used. Moreover we introduce a new approximated equation that can help determining the success in using S-K technique.**

Index Terms—Involved processes, bandwidth, events, and computation

I. INTRODUCTION

CAUSALITY RELATION BETWEEN EVENTS in distributed systems is a strong concept for analyzing and drawing interfaces about a computation. The knowledge of that relation between events helps solve a variety of problems in distributed systems (e.g. Algorithms design, progress monitoring, concurrency measuring [1].) In distributed systems the occurrence of events is higher while the events execution time is smaller. Consequently, the using of physical clocks may not help producing an accurate capture of the causality relation between events [1]. The method of using physical clocks suffers from pitfalls like clock drift. Therefore logical clocks are used [3]. In a distributed system that uses logical clock, each process has its own logical clock that increased according some rules. Every events will be assigned a timestamp such that the event happens before will have a smaller timestamp than the one happens after if they are causally related. There are many techniques for implementing logical clocks, some of them, Lamport's scalar, vector clocks and S-k, will be shown briefly, and some experiments are done as well.

II. IMPLEMENTING LOGICAL CLOCKS

Logical clocks can be implemented using a local data structure at every process in the distributed system to represent logical time and set of rules that used to update the data structure. Using this time other process can synchronize themselves with it, and relay on that logical time to order the events. Ordering of events is important to be known to analyze the computations. Systems of logical clock differ in their representation of logical time and in the set of rules used to update the time.

A. LAMPORT'S SCALAR TIME.

It was proposed by Lamport in 1978 trying to order events in distributed system. Each a process P has its own logical clock variable C that value is assigned with each event at that process. One of the major problems in scalar clocks is that partially consistency. We mean by that, if we know the clock value it is not guaranteed that we can know the events order. In other words, if we have events a, b, if $a \rightarrow b$, then $C(a) < C(b)$ but if $C(a) < C(b)$ we can determine weather $a \rightarrow b$ or not. There are two rules used to update the clock value: Rule, before executing event update Ci so, Ci := Ci + d (d>0). Rule2, attach timestamp of the send event to the transmitted message when received, timestamp of receive event is computed as follows: Ci := max(Ci , Cmsg) and then execute Rule1.

B. VECTOR TIME

Was developed by Fidge , Mattern and Schmuck 1988 [2]. In vector clock system, the time is represented by one dimension vector of positive integer numbers. Each process P maintains a vector v such that v[i] is the value of clock of the local process P. Other vector elements reflex the most recent knowledge of the local process about other processes. Like Scalar time, vector time system has a set of rules. Rule1: the local value is increased by one after a local event execution. Rule2: if the event is send, the local value is increased and a copy of the vector is attached with the message being sent. Rule3: once the message is received, the recipient of the message updates its local vector such that each element

will update its value by the coming one if the coming is grater, otherwise it keeps the current value. Then the recipient will update its value by taking the maximum value among the whole vector including it self and increment it by one. A new

technique has been introduced for more efficient implementation to the vector clock by singhal and Kshemkalyani.

C. SINGHAL- KSHEMKALYANI VC IMPLEMENTATION (S-K.)

It was introduced by Mukesh Singhal and Ajay D.Kshemkalyani in 1991 as an efficient implementation of vector clocks. S-K states that, instead of sending the whole vector with the message, only changed elements should be sent with their ID's, keeping the same update rules that used in vector clocks. In addition, each process maintains two vectors: LS[1..n] to keep the value of the local process when the last message was sent to another process, and LU[1..n] to hold the most recent value of the sender of coming message. So, each sender to a message needs to send with the message only the elements that meet the condition: {(x,vti[x])| LSi[j] < LUi[x]} such that x is all processes but the recipient of the message being sent j.

The sent vector contains the processes' Id's and Clock values of changed processes.

III. IMPLENTATION OF SCALAR'S AND S-K CLOCKS

A. SCALAR LOGICAL CLOCKS APPLICATION

An application was developed in C++ and was verified in different ways: 1. checking its consistency using a function compares the new value of previous one locally and with the sender in receive event. 2. results were compared with vector clock application. Computations were entered from input text file Generated manually and using a computation generator developed in C++ randomly (random sender and receiver). Each event is constructed according the following scheme: ***EventType,SenderID,ReceiverID*** such that: EventType is 1 for internal event, 2 for send event and 3 for receive event. (Example:3,7,8 means an event to receive a SMS was sent by Pcocess 7 to 8.) Also order of the events can be changed in the InputFile, just we need to make sure that the receive event is preceded by send event. Sending to a process it self is assumed as internal event. (Example 2,5,5.)

B. VECTOR CLOCKS:

An application was also developed in C++ and was verified using pre-known computation. Computations were entered from input text file generated manually and using a computation generator developed in C++ randomly (random sender and receiver). In similar way that used for scalar events format with an extra field:

EventType,SenderID,ReceiverID,EventId such that:

EventType is 1 for internal event, 2 for send event and 3 for receive event. EventId is number of the event when the message has been sent. (Example:3,7,8,4 means an event to receive a SMS was sent by Process 7 to 8 and the event was the fourth send event.) Order of the events can be changed in the InputFile just care is needed to ensure that the receive event is preceded by a send event and sending to a process it self is an internal event. (Example 2,5,5,4). In addition, Scalar application was embedded in this application for verification purposed.

C. S-K APPLICATION

It is implemented in C++ and was verified in different ways:

Results were compared with others generated by the combined Scalar and vector clock application. And, Known examples and random computations were used for that purpose. Computations were entered from input text file and were generated using a computation generator developed in C++.

IV. EXPERIMENTATION

In this section, we provide performance evaluations of S-K technique under various parameters and against regular vector clock application. The simulations were performed using our applications which were developed by C++ and executed in local machine with windows vista installed. We have generated our computation using two computation random generators. The computations were constructed in a way that enforces the algorithms uses as much as resources needed to make more pressure on the application. Performance metrics evaluated include Stamps size used (Bandwidth) in units (1 unit=32 bytes) and conditions of varying processes number, messages number, and number of the involved processes in the computation. It's expected that S-K in the worst case will perform as VC.

Parameter	Default Values
Processes numbers	**50-100 // constant 50**
Simulation Cycles	**15**
Messages Number	**500-2500**
Involved processes in computation	**100%-50% // 100%-20% (10-50 of 50)**
Events sequence 1	**All Send to all and all receive (50 lost)**
Event sequence 2	**Randomly send and direct receive**

Table 1. Default Simulation Parameters

A. EVALUATION OF THE EFFECT OF NUMBER OF MESSAGES AND NUMBER OF PROCESSES WITH SEQUENCE1 COMPUTATIONS

The computation were constructed by sequence1 where the send events where performed first and then the receipt events. 2500 messages have been used with 50 lost. Figure1

shows that S-K out performance regular VC even with changing the number of processes and involved processes as

well which contradicts with the equation that were introduced by S-K in their paper.

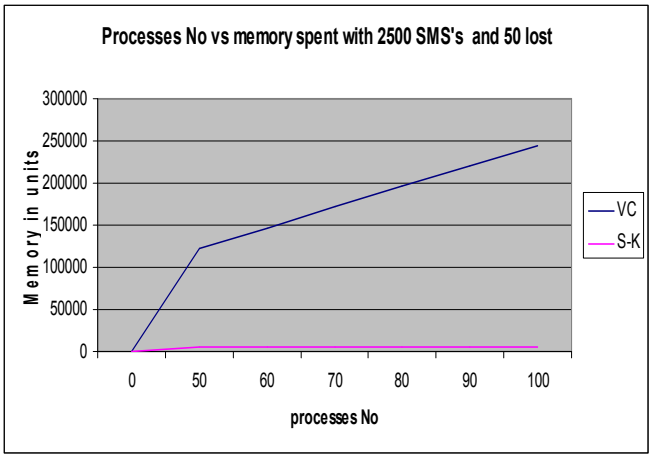

Figure1. the effect of #involved processes and #messages. With varied number of processes systems.

B. NUMBER OF MESSAGES AND NUMBER OF INVOLVED PROCESSES WITH SEQUENCE2 COMPUTATIONS AND CONSTANT NUMBER OF PROCESSES 50.

The computations were constructed by sequence2 where Sender and receiver were randomly picked and events executed in away the after sending, message is directly received to btain as much updates I the vector elements as possible. A constant number of processes have been used, 50, and each time we change the number of the involved processes in the computa_ tion from 10-50, and number of the sent messages in whole system from 500-2500. Number of the cycles were used was 15, to get accurate results since random computation is used.

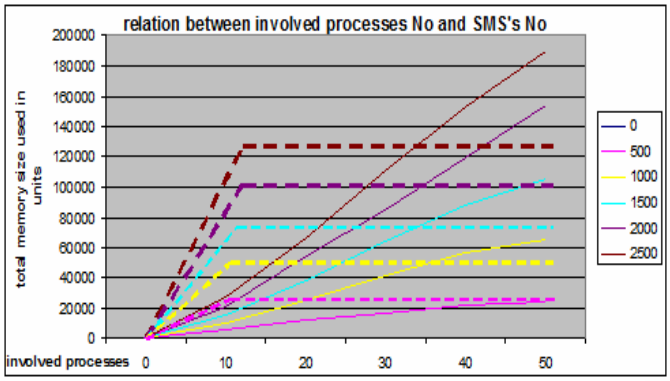

Figure2. the effect of #involved processes and #messages. With 50 processes system. Dots lines for the regular vector clocks and continues line for S-K

As we can see from figure2, when the number of the involved processes is less than about 70%, S-K outperforms the regular VC. After increasing number of the involved processes, S-K becomes inefficient. Also, number of the message looks involved in this observation, when number of the messages sent is less than 500, S-k does well the send events where performed first and then the receipt events. 2500 messages have been used with 50 lost.

V. SUMMARY AND CONCLUSION

In this paper, we presented the concept of logical time clocks and some technique used to represent it. Also, how can S-K can be used to drop the need to send all the clock vector in vector clock for efficient use of the bandwidth. Our experimentations have shown also interesting observations summarized as follows:

A. The sequence of the events plays big role in determining the efficiency of S-K.
B. Number of the involved processes in the computation can affect S-K performance.
C. For low number of messages, S-K seems fine.
D. When number of involved processes is about 70% and number of messages close to 20 times of all system processes then S-K becomes a weak.
E. The efficiency equation is not always applicable event in localized computations.

VI. FUTURE WORK

Implementation of Singhal-Kshemkalyani's technique should be studied more using more complex computation to derive adequate equation that might be more precise in valuating the situation where that technique can help.
Also, developing of the application it self can play big role for getting an optimal advantages of this technique.

APPENDIX

The source code of the applications used in the experimentations including the computation generators, and our presentation of this project.

ACKNOWLEDGMENT

Thanks to Prof.Nesterenko for his nice manner, and my family, and friends for their patience and support.

References

[1] http://web.cecs.pdx.edu/~black/CS410ds/papers/raynalOSRLogicalClocks.pdf
Michel RAYNAL IRISA, Campus de Beaulieu
[2] Logical clock, Adv OS course slides. Prof. **Mikhail Nesterenko** http://deneb.cs.kent.edu/~mikhail/classes/aos.s10/
[3] Manas Hardas . a paper in same subject 2007. Kent State University.