Performance Analysis of Singhal-Kshemkalyani’s Implementation of Vector Clock

MD Iftakharul Islam
Computer Science Department
Kent State University
Kent, OH, 44242, USA
mislam4@kent.edu

Abstract—Implementation of vector clock is a fundamental problem in distributed system algorithms. Vector clock is strongly consistent, thus gives a total ordering of processes in terms of causality. But, state forward implementation of vector clock is not scalable. This paper studies Singhal-Kshemkalyani’s implementation of vector clock. Our analysis dictates that Singhal-Kshemkalyani’s implementation reduces the message size significantly over the naïve vector clock implementation.

Index Terms—Vector clock, Causality, Distributed System Algorithms.

I. INTRODUCTION

Vector clock is a strongly consistent logical clock, where each process stores timestamps for every process. When any local event occurs, it updates its timestamp. Again, while sending the message, the process has to attach the vector clock to the message. If number of process is very large, the message size also increases. That’s why vector clock is not as scalable as it should be. Singhal-Kshemkalyani (S-K) algorithm dictates another alternative implementation of vector clock. In addition to store timestamp of every other process, it stores Last Update matrix and Last Sent matrix. Last Update matrix contains the timestamp of a process when it was updated last. Last Sent matrix contains the timestamp when it was sent message for the last time. So, when sending message to any process, rather than sending timestamp of every processes, it only sends the timestamps that has been updated since last sent time. Using the rule, it decreases the message size significantly.

The concept of logical clock was introduced by Lamport [2]. He proposed scalar clock, which is not strongly consistent. Vector clock was also developed independently [3]. In [1], a more efficient implementation of S-K algorithm was proposed. This paper studies the S-K algorithm, and analyzed it’s performance in terms of number of integers to be sent in messages.

Section 2 describes the experimental setup. Section 3 shows the result. Future research is explained in section 4, and section 5 concludes the paper.

II. EXPERIMENTAL SETUP

We implemented both vector clock and Singhal-Kshemkalyani’s implementation of vector clock to compare the result. To simulate the causality among processes, we considered a ring topology of processes. Each processes have three guarded commands, such as, send message, receive message, and local event. Processes have to receive a message if it sends message or execute a local event for two times consecutively. This condition is added to simulate the causality. We implemented the simulation in Java. We implemented both the vector clock and S-K algorithm for same problem set. We calculated the number of integer needed for each scheme.

III. RESULT

We varied the number of processes from 5 to 50 by incrementing by 5. At each data point, we took 5 results, and we plotted the average of the result. We measured the number of integers needed to be sent in the messages. The graph is shown in the figure 1. It’s very clear from that graph that vector clock is not scalable, as the number of integers in the messages increase with number of processors. But, for S-K algorithm, the required number of integers is significantly less than that of vector clock.

It may apparent from the graph that number of integers in the message does not increase with number of processors. But, we tested the algorithm for very large number of processes, for example, for 3000 processes. We found that the number indeed increases but in very slow pace.

![Fig. 1. Effect on number of integers in the messages with increase of number of processes](image-url)

Again we found that for very small number of processes, the number of integers does not differ too much from that of S-K algorithm. In fact, if number of processes is three, most of
the time, S-K algorithm requires more messages than vector clock. This is because, in S-K implementation, we have to send process ID in addition to timestamp.

IV. FUTURE WORK

In the paper, we measured the performance of S-K algorithm for ring topology. Research should be done for other topologies, such as, mesh, tree, or completely connected topology. Research should be also conducted on how program behaviors affect the performance of S-K method. We also have seen that S-K algorithm may need more messages than vector clock algorithm. We need to formalize the situation when this situation might occur.

V. CONCLUSION

The paper shows that though vector clock is a very effective way to impose causality among processes, it’s not that scalable when number of processes is very high. S-K algorithm intelligently solves the problem of scalability. S-K algorithm needs significant small messages than vector clock algorithm. The saving of S-K method increases with number of processes. The message size of S-K algorithm increases very slowly with increase of number of processes, which make it a perfect solution for large distributed systems.

REFERENCES

