Study and Analysis of Ho-Ramamoorthy 2-Phase Deadlock Detection Algorithm

Nitish Chaparala

Kent State University, Department of Computer Science, Kent, Ohio-44242
nchapara@kent.edu

Project Report for Advanced Operating Systems

***Abstract*— This paper describes the detailed study of the Ho-Ramamoorthy 2-Phase deadlock detection algorithm. The paper is based on the practical implementation of the algorithm and results are drawn from the experiment results. These algorithms are implemented on C++ platform. The message complexity and time complexity are used to measure the performance of the algorithm.**

I. Introduction

Ho-Ramamoorthy 2-phase algorithm detects deadlocks when it comes across a cycle in the wait-for-graphs. Cycles are checked for twice before it declares a deadlock.

Snapshots are used to implement this algorithm, to track the working. Snapshot of an algorithm returns the state of the system and message queue. It basically returns a status message giving details about the current state of individual nodes in the system graph.

Snapshot algorithm runs on a modification of the random flood algorithm to suit the needs of our desired deadlock detection algorithm. The random flood algorithm is modified such that a potential cycle is formed in a virtual tree to detect deadlocks. If the initiating node broadcasts the messages, and at some point receives a message back from the child node, a cycle is formed.

Various data points are inserted in the implementation of the algorithms to keep track on the number of messages and the time taken till that particular execution point. This data is collected over several runs and performance comparisons are presented based on the readings.

II. Experimental Setup

The experiment was conducted by modifying the random flood algorithm. Deadlocks scenario is obtained while processes send messages to each other. The initiating process becomes the father node, and upon receiving a message from the children nodes, a cycle is formed. This is considered to be the graph, and the cycle indicates a deadlock.

To measure the message complexity, forty processes were considered, and 10 runs per data point were considered, and the average number of these messages was considered.

To measure the time complexity, forty processes were considered, and the time taken to execute the algorithm (deadlock detection) was noted for 10 runs per data point.

III. Analysis of the Algorithm

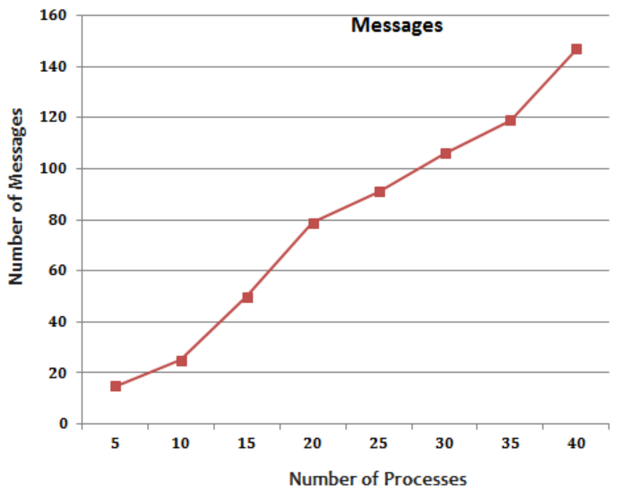
The analysis of the implemented algorithm started with the measurement of the number of messages (sent and received until the detection of a deadlock), to the number of processes.

Following is the table for the data used.

Message Complexity

Processes	**Messages**
5	15
10	25
15	50
20	79
25	91
30	106
35	119
40	147

Table 2.1



Graph. 2.1 Messages vs. Processes

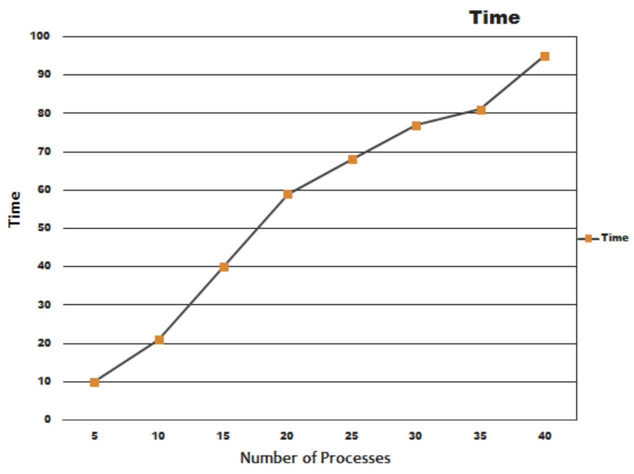
Graph 2.1 shows the time complexity of Ho-Ramammorthy's algorithm. Graph depicts a normal behavior. It shows gradual increase in number of messages as the number of processes increase.

For the second analysis, time (in ms) for the execution of algorithm is measured with the help of a physical clock. It is considered for the number of processes.

Time Complexity

Processes	Time
5	10
10	21
15	39
20	58
25	67
30	76
35	83
40	95

Table 2.2



Graph 2.2 Time vs. Processes

Graph 2.2 shows the increase in the time taken by the algorithm to execute as the number of processes increase. It shows normal behaviour up to 20 processes, where there is a constant increase until 30 processes. Then, there is an increase again. It is necessarily due to changes in the physical clock of the CPU.

IV. Implementation Details

The algorithm is implemented on the C++ platform. It essentially contains a queue module for the messages to be broadcasted and a modified random flood method which creates the necessary graphs to check for the deadlock. It is also responsible for taking the snapshots for the display of results.

Future Work and Conclusions

As part of the future work, firstly, the algorithms have to be tested on large distributed systems. The performance of the algorithm needs to be tested on various topologies as well. Ensuring that the algorithm is implemented in a more scalable manner should be the priority. Lastly, the algorithm implementation can be improved to reduce/remove the false deadlock detection possibility.

In conclusion, the Ho-Ramamoorthy 2-Phase deadlock detection algorithm was implemented with certain problems with the modularity of the code. It was observed that increase in the number of processes increases the time taken to detect the deadlock increases. Also, the number of messages before detecting the deadlock increases with an increase in the number of processes.

Acknowledgment

I would like to thank Dr. Mikhail Nesterenko, our professor for the course in guiding us throughout the project implementation and giving us an opportunity to showcase our skills.

References

[1] *G.S. Ho, C.V. Ramamorthy* "Protocols for deadlock detection in distributed database systems", IEEE Transactions on Software Engineering, 8(6), pp 554-557. Sep. 1982"

[2] *Singhal, Mukesh. And Shivaratri, Niranjan* [ISBN: 0-07-057572-X] "Advanced Concepts in Operating Systems". McGraw-Hill Publications, 1994, p. 156-158.