Time Complexity Analysis for Termination Detection Using weighted Protocol

Shreya Verma
Computer Science Department
MS Ramaiah Institute of Technology
Bangalore
shreyaverma27@gmail.com

Siddhartha
Computer Science Department
MS Ramaiah Institute of Technology
Bangalore
targetsiddu@gmail.com

Sadhvik Reddy
Computer Science Department
MS Ramaiah Institute of Technology
Bangalore
sadhvikreddy1997@gmail.com

Sini Anni Alex
Computer Science Department
Assistant professor
MS Ramaiah Institute of Technology
Bangalore
sinialex@msrit.edu

***Abstract*—Termination detection in the study of distributed systems is a popular topic for research and a major problem of study. It involves checking if all the nodes have successfully terminated their execution. There are a large number of methods to implement termination detection. In this paper termination detection using weighted protocol is used that is described in Imran Riaz Hasrat, Muhammad Atif, "Formal Specification and Analysis of Termination Detection by Weight-throwing Protocol",IJACSA Vol. 9, No. 4, 2018. We will apply time complexity technique to find out its measure in the algorithm used in the paper and also determine how to make the time complexity better. Our results show the time complexity of the proposed algorithm along with measures to improvise it in future.**

Keywords—Weight throwing protocol, Termination Detection, Time complexity, Model checking

I. INTRODUCTION

Termination detection is an essential area of concern in the distributed systems. In distributed system, process state formulates the basis for termination detection. There are two possible states of a process i.e, alive state or dead state. An alive state means that the process is still executing its computation or activity whereas, a dead state means the computation or activity has ended or terminated. The synonym for dead state is passive and for alive state is active. In the beginning all the process are in the active state. Processes can take the following actions:
•Only an active state process will send basic messages to all other processes.
• Any process may enter passive state at any instance.
•On arrival of basic messages, passive process will become active again.

It is very important for the previous phase of computation to terminate in order for the other phase to begin. In multiphase algorithms [2], one phase always depends on completion of another phase. A major flaw called deadlock is also caused due to improper termination detection[3]. different termination detection algorithms have been proposed by many researchers as described in [7]–[15]. In the referred paper, formal model is used based on mathematical tools and a software called UPPAL. The formal verification of the termination detection algorithm using weight throwing is done using this software as a model checker. UPPAAL has a simulator that was used in the paper to develop the model[19]. The verifier present in UPPAAL is equipped with ability to keep track of property fails by creating traces of action sequences. To investigate this situation, simulator replays the action sequences.

A. *Our contribution*

In the paper, two models have been devised. Model 1, is a fault free system in which only two types of messages are sent across the communication channel i.e, basic message and control message. The basic messages are sent from one process to another. These messages are sent to MessageBuffer first, until the receiver is ready to receive them. Control messages are sent from MessageBuffer to the leader. In Model 2, a fault system is taken into consideration hence, there are two types of messages sent across the communication channel along with the previously two mentioned i.e, failRequest and failReply. We will formally devise the time complexities of both the models and compare and contrast them. We present a formal specification to reduce the time complexity of both the models. We also present an analyzed summary of both the models in the paper.

B. *Road Map*

The rest of this paper is organized as follows. In Section 2, we describe the previous work on termination detection using weight throwing protocol using two models. In Section 3, we describe both the system models assumed by the referred paper. In Section 4, we state the time complexity meaning. In Section 5, we deduce the time complexity of Model 1. In Section 6, we deduce the time complexity of Model 2. In section 7 both the time complexities are compared and methods to improve them are mentioned. Section 8, circumferences the results and the paper paper is concluded in Section 8.

II. RELATED WORK

In the early times the termination detection algorithm was adopted only for systems which had a fixed number of nodes but, the paper analyzes the existence of systems in which nodes can be removed or added dynamically in an asynchronous system. The system also finds out the complexity of the system and helped our paper to find the latency and complexity.

In this paper in order to reduce latency and complexity, a tree based algorithm is used for dynamic, asynchronous vast systems proposed by Dijkstra and Scholten in [1]. The difference is this paper was that the nodes could leave or join the tree even during the computation. This paper gave us an idea on how to reduce complexity.

This paper helped analyze termination detection for global snapshots. This paper was referenced in order to determine if the complexity can be reduced using local snapshot or global snapshot. The referred paper uses spanning tree algorithm to take a global snapshot such that at least one node in the system is aware if all the other nodes have terminated or not. In the system the underlying structure used is such that the labels or vertices and edges is modified. All the relabellings are local which provide an abstraction for distributed computing. This paper also proved local termination is not stable.

The deadlock paper was referenced in order to reduce time complexity by taking into consideration the occurrence of deadlock. In this paper, deadlock detection and recovery methods were portrayed. As the biggest flaw in distributed systems i.e, deadlock occurrence is identified, the latency and time complexity can be automatically reduced. The following paper also determined detection for global snapshots.

III. SYSTEM MODEL

The referred paper utilizes two models in order to devise the theory of weighted protocol being invalid in a few cases. There are two models. The first model is fault-free and the second model is with faults. The models are described in the sections following. The protocol has used the following channels mentioned below. To perform the functionality in termination detection of a distributed system which is fault-free, hand-shake protocol in the channels are used. The functioning of the channels is described as follows:-

A. Model 1 (fault-free)

1. BasicMessageS(BMS) :The channel is necessary as this this channel is used for the BMS communications. BMS is sent to BufferMessage which stores all the messages until the receiver is ready for receiving.

2. BasicMessageR(BMR) :This channel is used to move messages from BufferMessage to the process receiving messages.

3. ControlMessageS(CMS) :This channel is used by the system to send CMS to BufferMessage.

4. ControlMessageR(CMR) :This channel is used to move stored CMR from BufferMessage to a leader.

B. Explanation of model

- There are parallel processes running which get activated through the channels discussed in the previous section. Hence, the activation process comprises of four processes, sending BMS, receiving BMR, sending CMS, receiving CMR. The complete termination, functionality between active and A1 (process) is explained.
- A BMS is sent to the BufferMessage(which stores the data till the time the receiver is ready to receive it such that the weights are divided into equal halves. One weight constitutes the numerator and other, the denominator in order to prevent floating point errors. The weights when combined together form a single weight.
- When the process moves from A1 to active, updateOut() (called when basic or control message is being sent) function is called which updates Out_arr[] (stores all the messages for outgoing transit) for recording weights.
- The first weight(w) i.e, w[1] is multiplied with two because multiplying the denominator with two will divide the overall value by two.
- When the action takes place from active to A2(process) , a basicMessageR is sent. Then the function call for updateIn() (call when BMR or CMR is accepted) happens which updates In_arr[] (stores all the messages for incoming transit) for recording weights.
- In the next step, w[0] and w[1] is equated to 0 in order to put the complete weight to the leader.
- A message transfer from idle(no messages received or sent) to A4(process) is similar to busy(when messages are sent or received) to A2 except that CMR is the activation channel.
- The leader aggregates all the weights from all the busy processes through CMR. If the weight/cost is equted to the predefined weight which is announces at the start then the leader establishes termination.

C. Model 2 (with faults)

Other Channels (that are not in mentioned in model 1) are

1)Fail Report : tells other processes about it's failed status & process receives status of failed processes using this channel .

2)Fail Request & Fail Request R : sends snapshots to message buffer and from there to termination process .

3)Fail Reply & Fail Reply R : snapshot reply to snap buffer then to recipient termination process .

D. Explanation of model

Declarations :

1. FIn[] and FOut[] : arrays store all the weights entering in and exiting out of processes .
2. FI FO Diff[]: array stores the difference between all the weights entering in and exiting out of failed process .

3. S[]: stores ids of termination process of all instances
4. F[] : records unsucessful processes known to every other process.
5. Ftemp[] : actual record of failed processes .

There are ten communicative choices in which four are same as the model 1 and others are :

- Snapshot of request message that is sent and received .
- Reply message
- Send & Receive final report message .

- Send & Receive final report message .

We discuss about actions between F1 to F4 :

- F1 is a FR(fail Report) Channel which detects unsucessful process and adds to F[] and Flush[] . The Leader() function is called to know current leader , if it is not declared leader then it becomes active and if it's a leader then it reaches Snapstate ,until the receiver is ready the snapshot is stored in snapbuffer and once it is ready then snapshot is sent from snapbuffer .
- In F2 , process receives snapshot request from snap buffer in fail Request R and sends snap shot reply to snap buffer for leader in fail Reply channel . F[] is updated for maintaining all failed processes .
- In F3 , leader gets the stored snapshot from snap buffer in fail Reply R channel. Where the difference between F[] of receiver and sender is calculated . If difference between them is more than zero then snapshot is inconsistant . If snapshot is processed then process is sent to active state or it's sent to snap state .
- In F4 ,similar to F1 detects failed process . This ensures that the progress of the snapshot is not terminated and the failed process is added into F[] and Flush[]. Also process is removed from S[] if it's still not empty then snapshot is taken.
- At failure, process updates FTemp[] into an array and gets to a fail state , Fail Report informs all the other process that it is failed .

IV. ANALYSIS OF TIME COMPLEXITY

We compare time complexity of termination detection algorithm in two models which are fault-free and faulty system.

Time complexity is representation of time taken to compute any algorithm. Time complexity of any algorithm can be found easily. Time complexity can be used to its utmost efficiency in case of recursive functions. Time complexity of declarations are minimal as the are executed only once per compilation. However loops, recursive functions, goto statements and logical jump statements take up majority of time in computation. However, single application of these statements wont take up much of the computation time. But combined with multiple statements of control statements can result in costly time complexity. "big O notation" is how typically time complexity is measured. O(Nn) is the representation of Time complexity, where "N" is the number of inputs and "n" is number of looping/logical expressions.

In paper which we choose which is Formal Specification and Analysis of Termination. Detection by Weight-throwing Protocol by Imran Riaz Hasrat, Muhammad Atif and Muhammad Naeem. There are two methods which are produced, one is which detects termination in fault-free system which is model 1 and another is faulty system which is referred as model 2.

V. TIME COMPLEXITY OF MODEL 1 (fault-free)

For N number of messages sent in a Distributed systems, This distributed algorithm calls a updateOut() function (O (G(n))), Which updates Out_arr[] which takes O(N) for Best case and O (N2 (G(N))) plus O(N) for function call and Out_arr[] respectively.

For N number of messages received in a Distributed systems, This distributed algorithm calls a updateIn() function (O (G(n))), Which updates In_arr[] which takes O(N) for Best case and O (N2 (G(N))) plus O(N) for function call and Out_arr[] respectively.

The message Buffer is used to to store messages until the receiver is free to receive the message. For this to happen, The Algorithm uses function called updateBBuffer() which keeps record of all incoming messages which takes same time as earlier mentioned update function. This however is used for only incoming messages and finally there is control buffer function for all the control messages which calls function updateCBuffer().

So, Time complexity of the Model 1 can be summarized as Follows:

For N messages Sent:

Best Case: O(N + G(N)).

Worst Case: O(N(1+N G(N)))

For N messages Received:

Best Case: O(2(N + G(N))).

Worst Case: O(2N(1+N G(N)))

For N Control messages received:

Best Case: O(G(N-1))

Worst Case: O((N-1)2 G(N-1))

VI. TIME COMPLEXITY OF MODEL 2 (fault)

Faulty system follows same procedure as the Fault-less system. So, The time complexity of Model 2 is build on time complexity of model 1. However, there are many functions which are called according to occurrence of fault in distributed system. For N failures in the distributed system, there is an Update of arrays F[] and Flush[] which takes up O(N) time for array. Assuming leader is chosen once per failure which takes up O(N(G(N))). The function calSN() calls the calcDiff() function. for every process id,The calcDiff() function calls isAvailable() function. If the process is available in array of F[], TRUE value is returned. This rules out the necessity to send snapshot Request Message for that specific process.this can be approximately translate to O(log(N))., Leader process is followed by AddIn() function. In AddIn() function, For all failed processes known to current leader, Total incoming weights are computed. This Function call adds up to O(NG(N)) and Incoming weights are added with complexity of O(N). Similarly, In AddOut() function, For all failed processes known to current leader, Total Outgoing weights are computed. This Function call adds up to O(NG(N)) and Outgoing weights are added with complexity of O(N) and To keep track difference between incoming and outgoing weights, A function FIin_FOut_Diff() is called. This function uses two array namely FIn[] and FOut[] which stores incoming and outgoing weights respectively. The difference is moved to FI_FO_Diff[].

Hence calculating time complexity will be addition of updation of three array access and function call which constitutes to O(3N+N(G(N))). This is followed by constants checks of snapshots requests and access of SnapBuffer process. When leader sends a snapshot request message, SnapBuffer has to store it. This Snapshot message further forwarded to termination process. For which, Termination process sends a snapshot reply message which is stored by SnapBuffer. This stored snapshot reply message is sent to leader.

Sending and receiving snapshots happens with linear time but time constraint here is indefinite waiting time for the receivers to be able to send the snapshot and clear the buffer.

So, Time complexity of the Model 2 can be summarized as Follows:

For N messages Sent (Same as Model 1) :

Best Case: O(N + G(N)).

Worst Case: O(N(1+N G(N)))

For N messages Received (Same as Model 1) :

Best Case: O(2(N + G(N))).

Worst Case: O(2N(1+N G(N)))

For N Control messages received (Same as Model 1) :

Best Case: O(G(N-1))

Worst Case: O((N-1)2 G(N-1))

Leader Election :

Best Case: O(2N G(N))

Worst Case: O(2N(N G(N)))

Local Declarations:

Best Case: O(log(N))

Worst Case: O(N2 log(N))

Incoming Weights:

Best Case: O(3N+N(G(N)))

Worst Case: O(3N+N(N G(N)))

Outgoing Weights:

Best Case: O(3N+N(G(N)))

Worst Case: O(3N+N(N G(N)))

VII. TIME COMPLEXITY COMPARISON OF BOTH THE MODELS

Table given below explains the time comparison between model one and model two:

Conditions	Computational time for model 1	Computational time for model 2
Requirement 1	57600m23.345s	06.307s**
Invariant 1	7210m34.453s	7215m33.873s
Invariant 2	7206m12.560s	11520m21.212s
Invariant 3	7002m19.350s	20.353s**

**Conditions NOT SATISFIED by the model.

For Model 1:
Total Number of Termination Process Instances = 3
Total Weight of the System = 1
Weight of Each Instance = ⅓
For Model 2:
Total Number of Termination Process Instances = 4
Total Weight of the System = 1
Weight of Each Instance = 1/4

Requirement 1: System is terminated if and only if all the weights in the system are collected successfully.So, Deadlock in the system should only occur when the leader is in announce state and all the other processes are kept in passive state.

Invariant 1: This Invariant is applied for all the processes in the system except Leader process. Invariant 1 says that a process should be in passive state, only if its weight is zero and Vice versa.

Invariant 2: Invariant 2 is about message passing where all the processes are allowed to send basic messages to each other and control messages to leader. A Non-Zero weight is attached before sending control and basic messages.

Invariant 3: The third invariant states that weights of current weight and initial weight should be equal along with sum of weights of Incoming and outgoing messages should be equal as well.

VIII. CONCLUSION

.We have presented with the time complexity of major operations in termination detection algorithm. For the simplicity, We have only considered Major Operations of distributed systems rather than computations and declarations. These declarations run only once which takes O(1) time. All the Big-Oh notations are only rough estimates as the execution of algorithms in distributed systems are variable. In this paper, there is a fault-free system and faulty system which are referred as model 1 and model 2 respectively.
In case of terminal detection, both the models are same if No fault occurs, however, non occurrence of fault is not practical. So model 2 is more applicable for applied distributed systems even though model 1 has better time complexity.

References

[1] *Formal Speocofication and analysis of termination detection using weighted protocol , vol9,4,2018*

[2] K. M. Chandy and J. Misra, “Asynchronous distributed simulation via a sequence of parallel computations,” Commun. ACM, vol. 24, no. 4,pp. 198–206, Apr. 1981. [Online]. Available: http://doi.acm.org/10.1145/358598.358613

[3] K. M. Chandy, J. Misra, and L. M. Haas, “Distributed deadlock detection,” ACM Trans. Comput. Syst., vol. 1, no. 2, pp. 144–156, May1983.[Online].Available: http://doi.acm.org/10.1145/357360.357365

[4] K. M. Chandy and J. Misra, “Asynchronous distributed simulation via a sequence of parallel computations,” Commun. ACM,vol.24, no. 4, pp. 198–206, Apr. 1981. [Online].

[5] Y. Tseng, “Detecting termination by weight-throwing in a faulty distributed system,” J. Parallel Distrib. Comput., vol. 25, no. 1, pp. 7–15,1995. [Online]. Available: https://doi.org/10.1006/jpdc.1995.1025

[6] X. Wang and J. Mayo, “A general model for detecting distributed termination in dynamic systems,” in 18th International Parallel and Distributed Processing Symposium, 2004. Proceedings., April 2004.

[7] G. Tel and F. Mattern, “The derivation of distributed termination detection algorithms from garbage collection schemes,” ACM Trans. Program. Lang. Syst., vol. 15, no. 1, pp. 1–35, Jan. 1993. [Online].Available: http://doi.acm.org/10.1145/151646.151647

[8] F. Mattern, H. Mehl, A. A. Schoone, and G. Tel, “Global virtual time approximation with distributed termination detection algorithms,”Tech.Rep., 1991.

[9] S. Chandrasekaran and S. Venkatesan, “A message-optimal algorithm for distributed termination detection,” J. Parallel Distrib. Comput.,vol. 8, no. 3, pp. 245–252, mar 1990. [Online]. Available: http://dx.doi.org/10.1016/0743-7315(90)90099-B [12] J. Pang, Analysis of a Security Protocol in μCRL, C. George and H. Miao, Eds.Berlin, Heidelberg: Springer Berlin Heidelberg, 2002.

[10] termination detection algorithm,” Journal of Parallel and Distributed Computing, vol. 67, no. 10, pp. 1047–1066, 2007. [Online].Available http://www.sciencedirect.com/science/article/pii/S0743731507000998