Estimation of worst case latency of periodic tasks in a real time distributed environment

[1]RAMESH BABU NIMMATOORI, [2]Dr. VINAY BABU A, [3]SRILATHA C*

[1] Research Scholar, Department of CSE, JNTUH, Hyderabad, India-500085
[2]Professor, Department of CSE, JNTUH, Hyderabad, India-500085
[3] Assoc Prof. and Head, Department of ECE, ASTRA, Hyderabad, India-500008

ABSTRACT

Real-time computations require exact bounded response times. For relatively simple models of computation, it is possible to determine conditions under which it is theoretically possible to guarantee that an invocation of a task will complete its execution. But it is very much complicated to guarantee a response time for the periodic tasks that execute in a distributed system application. In a distributed system, the applications are aligned as a set of dependent tasks. Here, the time required for a message to pass from one task to the other is a function of the individual timelines. Each task further constitutes of subtasks related by data dependencies. Also, tight bounds are essential for the tasks because of the dynamic nature of the distributed systems. As the distributed systems have the hard real time deadlines, the task executions have to be completed before the specified deadline. Hence the estimation of the worst case latency for a task link is essential. And, as the task structure and the application system model are well known in advance, for the synthesis and verification of application-specific distributed systems this estimation is quite important. This paper proposes an approach to estimate the worst case latency of periodic tasks in a real time distributed system.

Keywords: *Real time, distributed system, task, periodic, latency.*

1. INTRODUCTION

A real time distributed system environment basically consists of multiple computing processors. These processors interact with each other for computation and control of the system. Thus, the real time application consists of a set of periodic tasks. Wherein, each of these tasks has to get executed before the stipulated deadline. Any Real-time Distributed system has high dependability, where a system failure during execution because of either software or hardware faults can cause huge damages. As it is practically difficult to guarantee for all the tasks to be executed within the stipulated deadline, many application systems impose a small extra portion of the time limit for them to get executed. But, because of the complexity and the volatile nature of the distributed systems, it is a challenging task for the programmer to see to that the time limit is not compromised. Though unexpected issues like the resource sharing, paralleling computing, buffer overflow, communication medium etc may result in the increase in the task latency, it is a highly desirable character for a real time distributed system to be stable and thus avoids violations of the latency constraints.

These systems are expected to function with high availability. Unlike in normal operating systems, the task characteristics are predefined in a real time system. It reduces the system overhead. The worst case latency of a periodic task can be estimated to observe whether the task meets its deadline or not. Thus the worst case latency estimation is very important in a real time environment for the schedulability of the periodic tasks.

The worst case latency is defined as the difference between the times at which input data is first made available to an application task and the time at which an application task performs an output operation based on the input data. The latency estimation can be carried out by either simulation or by analytical methods. The simulation method depends on the various test inputs given, while the analytical method is application system independent. The worst-case latency is a conservative abstraction of all the possible situations but it could be too pessimistic in many applications, since all the mentioned worst cases do not occur at the same time.

In this paper, a model based analysis is proposed which is a combination of both simulation and analytical methods. This approach is exhaustive as it is obtained directly from the system design but not from the inputs.

The paper is organized as follows:

Section 2 briefly reviews some previous work done in the area of real time distributed systems. The proposed methodology is described in Section 3. Section 4 gives experimental results using our design. Section 5 concludes the described work.

2. RELATED WORK

[2] formulated methodologies for scheduling of real time distributed system. It is based on the fact that individual application is schedulable or not. While the proposed approach is less pessimistic as it is more system based. Though [3] approached the simulation method, the speed enhancement was observed at the cost of increased number of test vectors. Certain schedulability analysis approaches such as Rate monotonic analysis and Earliest Deadline First provide sufficient conditions for schedulability [4] in predictable periodic realtime systems. but fail to address the dynamics of events, race conditions, and the non deterministic execution order of tasks.

[5] Showed a new methodology using the lack conditions but it depends on the concurrent code. [6] [7] showed the performance analysis of the distributed systems using the upper bound of the task model. Concept of shared memory resource in distributed systems was presented in [8].

Time-triggered approaches [9] are becoming common in mission-critical applications.. For distributed system architecture based on time triggered architecture [10], the performance analysis is easy. But for the event triggered architectures it is very much complicated. The proposed approach not only supports the event triggered architecture but also the periodic task transactions.

3. APPROACH

In a real time distributed environment, tasks play a major role because all the computational and interaction part are being carried out by them. A task is enabled by the release of the events that may be received by other tasks. Each task can be in different states depending upon the instance.

Consider a distributed system with a set of tasks being executed on multiple processors. These tasks are said to be periodic as they are raised continuously for communication and control. The initial offset, time period and the deadline are predefined. The system functionality F is defined as:

$$F = (T, R, D) \text{ -------- Eq (1)}$$

Where: T = set of tasks in the system

R = relation between the tasks in the system. Ie $R \in T \text{ XT}$

D = deadline

Each of $t_i \in T$, wherein t_i one of the tasks of the system. The t_i functionality is defined as:

$$t_i = (I_i, P_i, E_{imin}, E_{imax}) \text{ ------ Eq (2)}$$

where: I_i = Initial offset

P_i = Time period

E_{imin} = Shortest execution time

E_{imax} = Longest execution time (Worst case execution time)

The edge factor E is defined as the flow output of t_i as input to t_j.

$$E = (t_i, t_j) \text{ ------- Eq (3)}$$

The task link (task chain) L is defined as the sequence of sub tasks;

$$L = (t_{i1}, t_{i2}, t_{i3}, t_{i4} \text{ .....}) \text{ ------ Eq (4)}$$

When initially tasks are created, all possible relations among the tasks are established automatically by the operating system. When a message is sent from t_i to t_j, the deadline D is calculated with the help of ti invocation time. This is because, both these tasks are scheduled. When tj receives a message from any other process, then t_j deadline is calculated on the basis of the arrival time of the message. Thus, the worst case latency in a periodic task model is dependant on the individual deadlines rather than the task structure. A task is said to be periodic if it is executed only once in every period and is characterized by its time period. A periodic task model can have N number of periodic tasks. In real time distributed systems it is necessary and important to determine an upper bound of time in that the program block is executed. Also, it is necessary to implement

the application code as transactions that incorporate processing elements on more number of processors and communicates across more networks.

We assume non-overwrite, FIFO semantics for buffers. Hence a message may have to wait for more than one invocation of the next task, depending on the number of earlier messages in the buffer. As the system has a limited number of task links, the worst case latency of the system is the maximum of the worst-case latencies of the individual task link. Because of this observation, we henceforth focus on analyzing the worst-case latency of a single task links. The worst case latency is defined as the difference between the times at which input data is first made available to an application task and the time at which an application task performs an output operation based on the input data. For a system F, with set of tasks T the worst case latency WCL up to the nth task in a task chain consisting of tasks from 1 to n, is given by:

$$WCL = t_{in}^{k} - t_{i1}^{k} + E_i \text{ ------- Eq (5)}$$

Where: t_{in} = nth task of the task link t_i
t_{i1} = 1st task of the task link t_i
k = k^{th} incocation
t_{in}^{k} = absolute time of the 1st task of the task link t_i
E_i = some value between E_{imin} and E_{imax}

Once the WCL is calculated for a single task link, other WCLs can be calculated and the system level WCL is the maximum of the WCLs of the individual task links. For an ideal distributed system, the WCL should be zero. But practically WCL will be equal to or less than the deadline D.

Consider an example of a transition of a message from task t_i to tj. This being a distributed system, assume that these two tasks run on two different processors. Also, it establishes a communication network from N, from t_i to tj. Task t_i inputs data from the environment, does some initial processing and then passes its 'result' to the link. Task t_j takes this result, undertakes further processing and produces an output for the environment. The following is the algorithm for the message transmission.

Step 1: StartTime is the clock rate
Step 2: Write StartTime to the communication network
Step 3: Next release would be at a time equal to StartTime
Step 4: Take input from environment
Step 5: Process the input
Step 6: Write result to the communication network (t_j)
Step 7: next release would be at a time equal to start time + Time period of t_i
Step 8: delay until next release
Step 9: go to step 4 untill the ti links is completed. Ie. t_{i1} to t_{in}.

Using equation 5, the worst case latency can be calculated.

The following is the algorithm for the message reception by t_j

Step 1: read Start Time from communication network
Step 2: next release would be at Start Time + WCL
Step 3: delay until next release
Step 4: read from communication network (from t_i to t_j)
Step 5: undertake processing
Step 6: write result to the environment
Step 7: next release would be at a time equal to next release + Time period of t_j
Step 8; go to step 3 untill the tj links is completed

The advantage of the above algorithm is that the task t_j executes at the right period and is guaranteed) to have data available on the link within the bounds of the analysis, when it executes the read operation. The arrival of the message at t_j sets up the period for the task t_j. Initially, if the data arrives early in the cycle, the read operation will subsequently block and the effective 'period' of the task will be greater than t_i. But once the maximum latency for the data has been experienced t_j will behave as a purely periodic task. Hence, the arrival of the message at t_j sets up the period for t_j. From the above algorithms and the equations, the following tabular column is constructed:

Assume that the start time is 0 and the transaction period is 20:

Event no	Response time of ti	Transmission time of ti	Data arriving at tj (period of transaction+ response time of ti + transmission time of ti)	Next release at tj (arrival time + transaction period)
Event 1	5	12	0+ 5+12 = 17	17+ 20 = 37
Event 2	7	13	20+ 7+13 = 40	40+20 = 60
Event 3	7	14	40+7+14 = 61	61+20 = 81
Event 4	6	14	60+6+14 = 80	81+20 = 101.
Event 5	8	12	101 +8+12 = 121	121

*Because already a delay of 81 has been observed, which is greater than 80,the read operation at time 81 now does not block and the task loops with a fixed period of 20 at times 101, 121 etc. (ie. an offset of 21). Time units in msec.

Once the worst case latency is observed, the loop becomes purely periodical and after that all the operations will be non blocking. From the above table it can be observed that the period was characterized by the periods of 37, 23, 21, and 20 before this 20 value became fixed.

The worst case latency of a message entering into the model is stored in a clock variable V. its value is continuously checked for the deadline comparison when the corresponding output comes out. The number of messages being transmitted are stored in a buffer variable. Randomly, the worst case latency of any message transmission can be calculated. The index of the selected message is assigned as I. I is verified to be non zero to ensure that this is the only input message for which *V* is being modified. Again when the task ti finishes the message transmission I value is decremented. In the execution period, we check the value of V against the specified deadline D. The worst case latency can be verified using the following equation:

WCL = (Clock rate = En && V > D && I >=1)

4. RESULTS

To validate the above algorithms and the equations, simulation experiments were performed. The UPPAAL tool was employed for the verification and thus obtained are the latencies for various combinations of the time period and the transaction time.

Event no	Time period	Transmission time	WCL
1	10	3	233
2	10	2	242
3	7	1	250
4	15	2	267
5	12	5	277
6	10	3	283
7	5	3	288

* Time units in msec

In the simulations once the worst-case is reached the simulations continue the latency. If the real upper bounds are below the worst-case values used in the static analysis, the periodic model will stabilize at a value below the theoretical worst-case. It shows only the worst-case situations experienced by the system.

5. CONCLUSIONS

This paper proposed an approach for the periodic tasks whose transactions are dynamically set by its own characteristics. We have observed that the task with maximum latency for its input profile will execute as regular periodic tasks with a fixed period. The worst case latency of the periodic task model is quite important to design because it needs the exact boundaries for the validation of correctness with respect to the functionality of the real time distributed system. The worst case latency thus estimated is further used to check whether the

predefined deadline is being met by the system or not. Also this methodology aims at estimation of the latency with minimum transactions and thus not requiring setting much offset for the later transactions

REFERENCES:

[1] L. Waszniowski and Z. Hanz´alek. Analysis of real-time operating system based applications. *FORMATS 2003*.
[2] J. Kr´akora and Z. Hanz´alek. Timed automata approach to CAN verification. *INCOM 2004*, 2004.
[3] K. Lahiri, A. Raghunathan, and S. Dey. System-Level Performance Analysis for Designing On-Chip Communication Architectures. *IEEE Transactions on Computer Aided-Design of Integrated Circuits and Systems*, 20:768{783, 2001.
[4] G. C. Buttazzo. Rate Monotonic vs. EDF: Judgment Day. *Real-Time Systems*, 29:5{26, January 2005
[5] E. A. Lee. The problem with threads. *Computer*, 39(5):33{42, 2006.
[6] J. Palencia and M. G. Harbour. Offset-based response time analysis of distributed systems scheduled under edf. *ECRTS*, 00:3, 2003.
[7] R. Pellizzoni and G. Lipari. Improved schedulability analysis of real-time transactions with earliest deadline scheduling. *RTAS*, 00:66{75, 2005.
[8] S. F. Fahmy, B. Ravindran, and E. D. Jensen. On scalable synchronization for distributed embedded real-time systems. In *SEUS*, October 2008.
[9] H. Kopetz and G. Bauer. The Time-Triggered Architecture. *Proceedings of the IEEE, Special Issue on Modeling and Design of Embedded Software*, Oct. 2001.
[10] S.K. Baruah and A. Burns. Sustainable schedulability analysis. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 159–168, 2006.