Deadline fault tolerance in a networked real-time system

Peng WEN [a] and Yan LI [b]
[a]Faculty of Engineering and Surveying, [b]Department of Mathematics and Computing
The University of Southern Queensland, Toowoomba 4350, QLD, Australia
Email: [a]pengwen@usq.edu.au; [b]liyan@usq.edu.au

Abstract. This paper applies a well-developed algorithm in real-time computing to a networked real-time system, and model the system as a periodic real-time computing one. In this model, each process is named as a task and implemented in two versions: the primary and the alternate. The primaries might fail but the alternates are guaranteed. A scheduling algorithm manages to execute all primaries if possible. Otherwise it guarantees the executing of each task either the primary or the alternate before their deadlines. The algorithm is verified in simulation. The result shows that in cases with high failure probability, the low priority tasks have a lower completion rates. In cases with low failure probability, both high priority and low priority tasks can be completed very well.

Keywords. Fault tolerance, Real-time system, Networked system, Distributed system

1. Introduction

In some networked real-time systems, such as computer-integrated manufacturing and industrial process control, a number of tasks are periodically invoked and executed in order to collectively accomplish a common mission/function and each of them must be completed by a certain deadline. Most timing constraints in real-time systems are deterministic, that is, non-statistical, in nature. Deadlines and other assertions involving time are expressed in terms of exact values, rather than aggregate measures such as averages. The reason, of course, is that failures to meet deterministic guarantees often mean mission failures. For example, a railway crossing gate on a road must always be closed by the time a train reaches the crossing. These kinds of deterministic constraints can be contrasted with other network based system performance and timing measures that are usually treated as governed by some stochastic process [1-2].

In real-time computing systems in cases both timing and computation constraints can not be met, one way of meeting the timing constraints is to trade computation quality for timeliness. Liestman and Campbell *et al* [3] proposed a mechanism in real-time computing systems. In this mechanism, two versions of programs are provided for each real-time task: primary and alternate. The primary version contains all the necessary functions and produces good quality results, but its execution is more prone to failure because of its high level of complexity and resource usage. The alternate version, on the other hand, contains only minimum required functions and minimum resource usage and, produce less precise but acceptable results. Its failure rate comparing to the primary version is considered as zero [2-4].

In this paper, we address the model of periodic tasks invoked in a networked real-time system, and apply a real-time computing algorithm to schedule these tasks with time deadline fault tolerance. The objective of our scheduling algorithm is to guarantee either the primary or the alternate version of each task to be correctly completed before the corresponding deadline while trying to complete as many primaries as possible. However, if the primary of a task fails during its execution or if its successful completion cannot be guaranteed, we must activate the alternate of the task. The rest of the paper is organised as follows: Section 2 briefly reviews the periodic tasks invoked in a networked real-time system, and model these tasks as a real-time periodic system. Then we discuss the fault tolerance algorithm, and provide the simulation and experiment studies in section 3. The paper is concluded in section 4.

2. Networked real-time system and modelling

Most networked real-time systems are much complicated and need to be implemented in real-time programming [1, 4-6]. A networked real-time system generally has two types of processes: periodic and sporadic. Periodic processes are activated on a regular basis between fixed time intervals. They are used typically for systematically monitoring, polling, sampling information from sensors, computing control signal and outputting control actions. For example, one may employ a periodic process to read all the variables in refinery every 500 ms or to scan the airspace every 2 seconds. In contrast, sporadic processes are event driven; they are activated by an external signal or a change of some relationships. A sporadic process could be used to react to an event indicating a fault of some piece of equipment or a need to change modes. For a real-time control system, most of the processes are periodic. For example, in an output feedback control for a linear system the controller can be represented as:

$$x(k+1) = Fx(k)+G\,y(k)+G_c\,u_c(k)$$
$$u(k) = Cx(k)+G\,y(k)+D_c\,u_c(k)$$

where x(k) is the controller variable, $u_c(k)$ is the system input and u(k) is the controller output.

This controller can be implemented in a computer as

```
Begin
        A-D, $y$ & $u_c$ Input
        $U: = C\times x+D\times y+D_c\times u_c$
        $X: = F\times x+G\times y+G_c\times u_c$
        D-A Output
End
```

For the above implementation, the computation delay is not optimised. It is obvious that we can output the control signal before all these computations are completed. Actually, the control signal u is available after executing the second line of the code, the D-A conversion can be done before the state is updated. The computation delay may be reduced further by calculating the product C×X after the D-A conversion. The implementation in the above can be modified as the follow:

```
Begin
        A-D, $y$ & $u_c$ Input
        $U := u_1 + D \times y + D_c \times u_c$
        D-A Output
        $X := F \times x + G \times y + G_c \times u_c$
        $u_1 = C \times x$
End
```

The processes discussed in the above section can be considered as real-time periodic tasks in real-time computing system. A model for this system is given as the follow: A real-time periodic task system consists of a set of n periodic task $\tau = [\tau_1, \tau_2, \dots \tau_n]$. Each task τ_i must be executed once every T_i time unit, where T_i is the period of τ_i. Each execution of a periodic task is called a job (or request), and every job of τ_i has a computation (or execution) time e_i. The jth job of τ_I is denoted as J_{ij} for all $n \geq i \geq 1$ and $n_i \geq j \geq 1$. J_{ij} is ready for execution at time $(j-1)T_i$ and must be completed by the time of the next job period of the same task, which is equal to jT_i. We define $r_{ij} = (j-1)T_i$ to be the request time (or release time) and $d_{ij}=jT_i$ the deadline of J_{ij}.

The fault-tolerant real-time periodic task system considered in this paper is formally defined as the follows: Consider a set of n real-time periodic task $\tau = [\tau_1 , \tau_2 , \dots \tau_n]$, each task τ_I has a period T_i and two independent versions of computation program: the primary P_i and the alternate A_i. The primary contains all the necessary functions and, when executed correctly, produces perfect results for this task, but its reliability cannot be guaranteed because of its complicated functions that are difficult to test or verify. On the other hand, the alternate is reliable due to its simple functions that are easy to test and produces acceptable results for this task. P_i has a computation time p_i, A_i has a computation time a_i, and, usually, $p_i \geq a_i$ for $n \geq i \geq 1$. Let the planning cycle, T, be the least common multiple (LCM) of $T_1, T_2, \dots T_n$. Then, $n_i = T/T_i$ is the number of jobs of τ_I in each planning cycle. Since the task invocation behaviour repeats itself for every planning cycle, we only need to consider all task invocations during any one planning cycle. Thus, without loss of generality, we can consider the problem of scheduling tasks for the first planning cycle [0 T]. The primary and the alternate of the jth job J_{ij} of τ_I are denoted by P_{ij} and A_{ij}, respectively. For each J_{ij} of τ_I, either P_{ij} and A_{ij} must be completed by its deadline jT_i. Since P_{ij} provides a better computation quality, we would prefer the execution of P_{ij} to that of A_{ij}. However, in case P_{ij} fails, we must ensure A_{ij} to be completed by its deadline, thus providing an acceptable, though possibly degraded, computation result. That is, we want to complete as many primaries as possible while guaranteeing either primary or the alternate of each task to be successfully completed by its deadline.

3. Real-time periodic task system scheduling

For a given real-time periodic task set τ, all the alternates are scheduled using a fixed priority-driven scheduling algorithm to reserve time intervals as late as possible in a planning cycle before runtime. At runtime, if there are primaries pending during the time intervals that were not reserved by alternates, the scheduler chooses the primaries to execute. The primaries can be scheduled by an online scheduling algorithm, such as a (fixed or dynamic) priority-driven pre-emptive scheduling scheme with the RM or

EDF priority assignment. A primary may fail at any time during its execution or take too long to complete. If a primary fails, its corresponding alternate must be executed. Moreover, when the notification time, v_{ij}, of alternate A_{ij} is reached, yet its corresponding primary P_{ij} has not been completed or has failed, A_{ij} is activated. That is, pre-empting the execution of any primary, including P_{ij}, or other lower-priority alternates. For the primary P_{ij}, if it has not been finished, it will be aborted since its alternate, A_{ij}, is chosen to be executed. If the primary is not P_{ij}, it will be suspended and resumed later. Every alternate, if activated on or after its notification time, has higher priority than all primaries and the activated alternates are executed according to their priorities assigned by the offline fixed-priority algorithm.

In calculating the notification times, we consider only the alternates, A_{ij}s. We use a_i as the computing time of τ_I and use a fixed–priority algorithm F to construct a schedule (for example, by using the slack interval method to calculate it backward from time T to time 0, and find the finish time v_{ij} of A_{ij} in the schedule, for each $n \geq i \geq 1$ and $n_i \geq j \geq 1$. We will call this algorithm the backward-F algorithm for given fixed-priority scheduling algorithm F. We then use v_{ij} as the notification time of A_{ij} at runtime. That is, the notification times are simply the finish times of the alternates if they are scheduled by a fixed-priority algorithm backward from time T. Note that the notification times force the alternates to be scheduled as late as possible and, hence, the scheduling algorithm leaves the largest possible room for executing the primaries before executing the alternates.

The objective of our fault-tolerant scheduling algorithm is to guarantee either the primary or alternate version of each job to be successfully completed before its corresponding deadline while trying to complete as many as primaries as possible. Therefore, the following two metrics: PctSucci, which indicates the percentage of successfully completed primaries for each task, and W, the processor time wasted by executing unsuccessful primaries during the whole time span of the schedule, are employed [2]

3.1 Simulation

To verify this algorithm, a simulation is carried out. The task set used in the simulation is $\tau=[\tau_1, \tau_2, \tau_3, \tau_4]$ with task (T_i, p_i, a_i)=(15, 4, 2), (24, 5, 3), (40, 10, 5), and (160, 30, 15) and the planning cycle is T=LCM(15, 24, 40, 160)=2400. We also test this algorithm for different primary failure rate. The simulation results are presented in the table below.

Table 1 Simulated $P_{ct}S_{ucc}$ results

	FP=0.1	FP=0.05	FP=0.01	FP=0
Task 1	87.18%	92.31%	97.44%	100%
Task 2	84.62%	91.03%	97.23%	100%
Task 3	64.1%	82.05%	93.59%	100%
Task 4	23.1%	58.97%	84.62%	100%

Table 2 Simulated W results

	FP=0.1	FP=0.05	FP=0.01	FP=0
W	94.42%	51.35%	18.92%	0%

In the above table, the two metrics: $P_{ct}S_{ucc}$ and W are calculated as follows. Suppose FP (failure probability) is 0.1 and there are 20 jobs, then there are two primary failures because of FP $\times$ 20=0.1$\times$ 20=2, and 18 of the 20 jobs will complete successfully. If the actual schedule accommodates only nine successful primaries, then $P_{ct}S_{ucc}$=9/18=50%. That is, $P_{ct}S_{ucc}$ is the percentage of actual successful primaries among the maximum possible successful primaries, thus representing how many subsequent primaries are affected by the early failures and how well the scheduling algorithm deals with early primary failures. If the execution of a primary is aborted when the corresponding notification time is reached, the amount of time that has already been consumed by the primary is regarded as wasted. W, the wasted processor time, is calculated by summing up the time slots wasted by all unsuccessful primaries.

The simulation results in the above figure show that the lower priority task suffers failure more significantly. For example, for FP=0.1, the percentage of successful primaries for task 4 degraded to only 23.1%, while that of task 1 is greater than 87.18%. The primary of task 4 gets less of a chance to be executed because of its lower priority.

3.2 Experimental testing

Incitec Limited is a large chemical manufacturing company with facilities located Australia wide. It is Australia's largest fertilizer manufacturer, producing products such as ammonia, ammonium nitrate and urea. It has two main manufacturing sites, one at Gibson Island in Brisbane and the other at Kooragang Island in Newcastle. To maintain competitive the plant has undergone a number of upgrades to maximise capacity and efficiency with minimum personal or equipment cost [8]. We propose to monitor and control this mixed large system through Internet which is available almost every where.

This network based real-time monitoring and control system can be modelled as a real-time periodic task system. As this is a distributed system, there are many computers and micro-controllers included. It is hard to do the scheduling as these tasks are executed in parallel by different processors. We have to normalise these processor time, task periods and execution times into an equivalent uniprocessor time, task periods and execution times. The outcomes for this normalisation are listed in the table 3 below

Table 3 Task Set

Task	**Period** (10ms)	**Primary**(10ms)	**Alternate**(10ms)
Data acquisition	30	6	2
Signal conditioning	30	3	2
Control algorithm	30	4	2
System updating	60	6	3
Display	50	5	2
Communication	100	8	5

An inspection to the above table, we can see that the processor utilization factor for this system

$$U(\tau) = \sum_{i=1}^{n} e_i / T_i = \frac{6}{30} + \frac{3}{30} + \frac{4}{30} + \frac{6}{60} + \frac{5}{50} + \frac{8}{100} = 0.713$$

The least upper bound for this system is

$$n(2^{1/n} - 1) = 6(2^{1/6} - 1) = 0.735.$$

Obviously, $U(\tau) = 0.713 < 0.735$, the system is schedulable both by RM and EDF.

A prototype system is set up in our control lab through Internet. There are two industrial PC computers in this prototype system. One computer is supposed to be located in the workshop end and connected to the PLCs, transducers, sensors and actuators and another computer is supposed to be in the operational end to perform the displaying and operation functions. They are connected through the Internet in the university. In the testing operation, the switches are turned on and off at the workshop end. These changes are detected by PLCs, collected by the computer at the workshop end and are sent to the operational computer at the operational end. A sequence control command are sent out at the operational end, and executed correctly at the workshop end. The analog signal measurement and other functions are also tested. All the real-time constraints can be met perfectly.

4. Conclusion

In this paper, we study a networked real-time system, and propose to model it as a periodic real-time computing system. For this system, with efficient scheduling algorithms and software deadline fault-tolerance mechanism, we can design a system that meet its task timing constraints with the tolerating of system faults.

The simulation results developed in this paper shown that, in cases with high failure probability, the lower priority tasks suffer a lot in completion rate. For example, when failure rate FP=0.1, the lowest priority task can reach only 23.1% completion rate. When failure rate FP decreases to 0.01, the lowest priority task can reach 84.62% in completion rate. These results show that an efficient real-time periodic task system heavily depends on its task failure rates. We should keep its task failure rates to the minimum in our design. In fact, most of real world systems their task failure rates are required less than 10^{-3}. Therefore, this algorithm will be very helpful to guide our system design. Later on, we developed a real-time supervision control and data acquisition system using this method

References

[1] A. C. Shaw, Real-time systems and software, John Wiley & Sons, Inc, US, 2001.

[2] C. C. Han, K. G. Shin & J. Wu, A fault-tolerance scheduling algorithm for real-time periodic tasks with possible software faults, IEEE Trans. Computer, vol. 52, no.3, pp. 363-372, Mar. 2003.

[3] A. L. Liestman & R. H. Campbell, “A fault-tolerance scheduling problem,” IEEE Trans. Software Eng., vol.12, no.11, pp. 1089-1095, Nov. 1986.

[4] A. M. K. Cheng, Real-time systems scheduling, analysis, and verification, A John Wiley & Sons., Inc., New Jersey, US, 2002.

[5] K. J. Astrom & B Wittenmark, Computer-controlled systems theory and design, Prentice-Hall International, Inc., NJ, US, 1997.

[6] B. Hoseph, Real-time personal computing for data acquisition and control, Prentice Hall, New Jersey, US, 1989.

[7] C. L. Liu & J. W. Layland, “Scheduling algorithms for multi-programming in a hard real-time environment, “J. of the ACM, vol.20, no.1, pp. 46-61, Jan 1973.

[8] D. G. Dugdale, “Controller optimisation of a tube heat exchanger”, Dissertation, Faculty of Engineering and Surveying, The University of Southern Queensland, Queensland, Australia, 2001.