Soft Real-Time Task Response Time Prediction in Dynamic Embedded Systems

Cássia Yuri Tatibana, Carlos Montez, and Rômulo Silva de Oliveira

Universidade Federal de Santa Catarina, Pós-Graduação em Engenharia Elétrica,
Caixa Postal 476,
Florianópolis-SC, 88040-900, Brazil

Abstract. The hardware infrastructure that provides the support of ubiquitous embedded computing may be shared by different applications. Many of those applications have real-time requirements, where events from the environment require the reaction of the computing system. The meeting of deadlines is hindered by the fast system dynamics. At the same time, the embedded system must deal with overload situations. In this paper we assume that an embedded application receives aperiodic requests with soft deadlines. Other unknown applications are executed simultaneously. The goal of this paper is to discuss algorithms to estimate the probability of a deadline to be met. The prediction of a deadline miss at the request arrival allows actions for damage control.

1 Introduction

The accelerated growth of the use of embedded systems in a huge diversity of products, towards the implementation of ubiquitous systems in different environments, creates the demand for an always larger diversity of applications. In this context, physical and financial constraints will require the sharing of the hardware infrastructure (processors, memory, network, etc) that provides the support of ubiquitous embedded computing among different applications, at different times. Although nowadays most embedded systems are developed through co-design of the hardware and of the software, in the future the infrastructure embedded in the environment will need to support the download and the execution of applications designed much before or after the hardware was designed.

Many embedded system and ubiquitous system applications have soft real-time requirements, where events in the environment generate the need of a reaction from the computing system. Most of the time, those timing requirements are not critical.

In this context, the meeting of timing requirements is hindered by the dynamics of the system (applications are started and finished at any moment), by the lack of knowledge about internal characteristics of the other applications (worst-case computation time, types of existent threads, etc), and by resource constraints. At the same time, the need for autonomy requires embedded systems to deal with overload situations without the aid of human supervision.

R. Obermaisser et al. (Eds.): SEUS 2007, LNCS 4761, pp. 273–282, 2007.

In this paper we consider the situation where an embedded application receives requests from clients that can be physical devices (by hardware interrupts), tasks in other processors (by messages) or tasks in the same processor. Such requests are accompanied by a soft deadline, in an explicit way (for example, a data field in the received message) or in an implicit way (all request of that type always have the same deadline). The application is capable of working with a pre-defined group of request types, that is to say, it implements a pre-defined group of services. There is a local thread pool responsible for serving those requests. It is used at the most a single thread for each service type, in order to reduce the overhead associated with context switching and synchronization mechanisms. Other applications of unknown nature are executed simultaneously.

Requests arrival is aperiodic, characterizing the system as dynamic and subject to overloads. Deadlines are necessarily not critical. During overload there is the formation of request queues for one or more of the supported services. However, the occurrence of an overload in the processor doesn't affect the requests generation directly. That is because requests are generated due to external events and they are not subject to control flow.

The objective of this paper is to discuss algorithms to predict the probability of meeting a deadline, in dynamic systems that are not controlled by the algorithm and there is limited information about the state of the system. The prediction of a deadline missing must be made at the arrival of the service request, since it will allow actions of damage control, such as the signaling of alarms (timing exception), the use of an alternative computer or the use of some mechanism for load reduction (admission control).

The remaining of the paper is organized as follows: section 2 presents some of the related work, section 3 describes the problem, section 4 has the proposed approach, an example application is described in section 5 and section 6 contains the final remarks.

2 Related Work

Most of the real-time scheduling theory considers systems where no deadline may be missed.On the other hand, there are applications that tolerate the eventual miss of a deadline, as long as it doesn't happen too often. Several works present schedulability tests that include the possibility of eventual misses of deadlines. For example, Tia et al. [11] present Probabilistic Time Demand Analysis, which was extended by Diaz et al. [5]. Gardner et al. [7] present Stochastic Time Demand Analysis. Other stochastic analysis methods are the one presented by Manolache [10] for uniprocessor systems and the one presented by Leulseged et al. [9] for multiprocessor systems. Also, the Real-Time Queuing theory presented by Lehoczky [8] can provide stochastic guarantees for system with a high traffic load. Several results have been presented requiring a specific scheduler, such as those by Abeni [1] for reservation based systems, and the Statistical Rate Monotonic Scheduling by Atlas et al. [2].

Some works in the literature deal with the response time of aperiodic tasks. In [3] the existence of periodic tasks is assumed, whose hyper-period is analyzed, the moments of idle processor are identified and these moments form the base of the analysis, which is better when the computation time of the aperiodic task is small compared to the hyper-period. In [4] it is supposed FCFS as the scheduling policy, for tasks whose control flow is described by a probabilistic graph. The service requests are characterized by a probabilistic distribution. In [6] it is supposed that each application can be characterized by a workload that expresses the amount of resources (processor, disk and network) necessary for that application. The modeling of the system is base on this demand description. The estimate of the latency average is used in [12] to predict the response time of a new request sent to a HTTP server. It combines the average and variance of previously observed latencies into a single metric.

3 Problem Description

Consider an embedded computer where several programs execute simultaneously and a program is capable to accomplish a certain group S of services, $S = \{S_1, S_2, \ldots, S_{ns}\}$, where ns is the number of services supported by the program. The execution of a service may correspond to the execution of a single or several software functions, or even the interaction with physical devices.

Along the execution of this program, aperiodic requests arrive for the execution of its services. We will call task the execution of a service. Without loss of generality, we can number the requests in the growing order of their arrival instants, being T_1 the first received request. Each task T_k is characterized by the request of a specific service S_k, $S_k \in S$, and by a relative deadline D_k.

All service requests are executed according to the scheduling algorithm of the program and of the underlying operating system. Tasks whose response time is bigger than the deadline are executed to the end. This avoids the occurrence of internal inconsistency in the data structures of the program. Also, requests for the same service are always executed in the order of arrival. The scheduling among different services may vary.

We want an algorithm that, at the arrival of task T_k, it determines the probability of deadline D_k of task T_k to be met, i.e., $P(R_k \leq D_k)$, where R_k is the response time associated with the execution of T_k. As an additional constraint, the algorithm has to be implemented at the application level, it can not depend of specific support from the underlying operating system.

It is necessary to define the metric used to compare the quality of the response time predictions done by several algorithms in a given system. In this work we use the relative error rate $E(z)$ observed for each considered algorithm, where z indicates the algorithm used. This rate supplies a measure valid only for a given system and a given pattern of request arrivals, but it allows a comparison among different algorithms.

At each arrival of a task T_k in the system, each response time prediction algorithm z under evaluation calculates the probability $P_k(z)$ of this task to

meet its deadline, $0 \leq P_k(z) \leq 1$. Task T_k is executed and its effective response time R_k is measured.

The relative error $E_k(z)$ is the error associated with the prediction of the response time done by algorithm z for the task T_k, it is defined as:

$$E_k(z) = \begin{cases} 1 - P_k(z) & \text{case} \quad R_k \leq D_k \\ P_k(z) & \text{case} \quad R_k > D_k \end{cases}$$

The value of E_k is necessarily between 0 and 1. The situations below illustrate the behavior of this metric:

- The algorithm z determines that there is a chance of 80% for task T_k to meet its deadline, $P_k(z) = 0.8$, and it meets its deadline, we have the relative error $E_k(z) = 1 - 0.8 = 0.2$;
- The algorithm z determines that there is a chance of 80% of task T_k to meet its deadline, $P_k(z) = 0.8$, but the task misses its deadline, we have the relative error $E_k(z) = 0.8$;
- The algorithm z determines that there is a chance of 50% of the task T_k to meet its deadline, $P_k(z) = 0.5$, and in this case we will have $E_k(z) = 0.5$ independently of the task meeting $(1 - 0.5)$ or not (0.5) its deadline.

The relative error rate of a given algorithm z is defined as:

$$E(z) = \frac{\sum_{\text{all k}} E_k(z)}{n_k}$$

4 Proposed Approach

A simple approach is to use the response time of tasks in the past to predict the probability of meeting the deadline of a task that arrives. So, the program history should be maintained with the response time of each task executed before, for each service of the program. The history can be used as the PMF (Probability Mass Function) of the random variable R_k, and a simple inspection of the history supplies the probability $P(R_k < D_k)$ of task T_k. However, this simple approach has some disadvantages. Due to the fast system dynamics, only recent response times are important to predict the response time of a new task. Besides, the task type may have not been requested for a long period, and the prediction algorithm has no data to work.

In fact, the complete history contains the PMF of the response time of invocations of a given service, observed on long periods of time. In order to decide about the probability of the next execution of this service to meet its deadline, it is necessary the PMF of this next activation. The PMF of the next request for service S depends on the current state of the system, which is defined by several factors, such as: (i)The parameters supplied in the call of the service; (ii) The current value of the permanent variables of the program; (iii) The computer load due to the other tasks of that application; (iv) The size of the queues associated with services of this program; (v) The computer load due to other applications;

(vi) Changes in the relative demand for the several resources in the system, what affects the many services differently.

The number of possible system states is huge, if not infinite. However, it is possible to merge those states, according to some properties that are easy to measure and capable of indicating the response time approximately to be expected in the next execution of the requested service.

A form of attacking the problem is to use, to predict the response time of a service S_1, not only the old executions of S_1, but also the historical data about old executions of all the services supported by the program. We assume that an overload in the node will increase the response time of all the services, and not only of S_1. Besides, the response time of S_1 may be influenced by the current state of the permanent variables of the program (size and content of the data structures of the program). Thus, the recent behavior of all the services can be used for the calculation of the response time prediction.

An analysis of the crossed correlation among the response times of the several services supported by the program would allow the identification of which services are important for the estimate of the response time of which services. Assuming that the program implements n_s services, it would be necessary n_s^2 correlation studies, including the auto-correlations. Such analysis would need to be done continually, given the non stationary nature of the considered system. For example, a variation in the size of a data structure of the program may change the correlation between two specific services. The processing cost of this solution is prohibitive for many systems.

4.1 Approach Description

The approach proposed in this paper tries to merge all the possible system states in just two: normal and high loaded. Initially the system is in state normal. At the moment a task T_k finishes, its response time R_k is compared with the average response time R_s associated with service S. The new state of the system is defined as State Normal, in case $R_k \leq R_s$, or State High Loaded, in case $R_k > R_s$.

The value R_s is defined as a running average. It is updated whenever a task concludes the execution of service S. The new value of R_s is denoted by $R_s(i+1)$ and it is given by:

$$R_s(i+1) = \alpha \times R_k + (1-\alpha) \times R_s(i)$$

where $R_s(i)$ is the previous value of R_s, R_k is the last response time observed for service S and α is a constant between 0 and 1. The purpose of keeping R_s as a running average is twofold: to discard old response time values and to reduce the computing cost, since R_s is used by the prediction algorithm every time a new request for service S arrives.

A record of historical data is maintained with the recent response times of each method, together with the information on the system state at the moment of the arrival of that request. In fact, there are two historical records, one with the response times observed with the system in normal state, and another observed with the system in high-load state.

Whenever a new request arrives for service S, the historical data specific for service S response times is used. It is considered only that response time that was observed in the past, when the system was in the same state that it is now. By using this historical record as a PMF, the conditional probability of the task meeting its deadline can be calculated, given that the system is identified as in a given state, that is to say, $P(R_k \leq D_k|\text{system state})$. A simple inspection of the historical record supplies this information.

The historical record is implemented as a circular list, with a finite size. Only the last T conclusions of each service type are maintained in the record. The size T depends on the kind of system, this will be discussed in the next section.

5 Example Application

An application was implemented as a proof of concept for the approach proposed in this paper. Since the objective of the proposed approach is to deal with dynamic systems, whose behavior of the operating system and of the other applications is unknown, the application was implemented in Java and executed on a desktop computer using Windows XP, together with other programs. The availability of Java in the versions Micro-Edition (J2ME) and Real-Time (RTSJ) will increase its use in embedded and ubiquitous systems.

The program to be analyzed is composed by 7 types of services. Simple and quite common functions were used: insert in a list at the end, sequential search and delete from the list, calculation of average through a complete scan of the list, calculation of standard deviation through two complete scans of the list, binary search in a sorted array, generation of two square matrices with pseudo-random numbers and finally the multiplication of two square matrices. The Java class Vector was used for the implementation of the list. The list starts empty and it grows along the execution of the program. The matrix used by the generation service and by the multiplication service has a random size between 50x50 and 100x100, defined at each new arrival of the task of matrix generation. It is important to observe that this variation in the parameters of this service also generates a variation in the computing effort associated with the service. The Java Garbage Collector execution was not controlled neither requested.

Except for the requests for the matrix multiplication, all the others are generated with intervals between arrivals that follow an uniform distribution, within the following intervals: 10mS to 20mS for Insert; 10mS to 40mS for Remove; 10mS to 45mS for Average Computation; 10mS to 45mS for Standard Deviation Computation; 10mS to 20mS for Binary Search; 30mS to 70mS for Matrix Generation.

The matrix multiplication is always requested by the service of matrix generation. Therefore, the interval between arrivals of the requests of matrix multiplication is approximately equal to the interval between arrivals of the requests for matrix generation.

As an additional disturbance in the system, a Java thread was included that generates two matrices of size 100x100, then it makes their multiplication. This thread doesn't use the service queue mentioned before, but it implements its own

functionality. This extra load is activated once approximately every 4 seconds. It does not represent a disturbance controlled by the program.

Regarding the priorities of the Java threads, it is important that the request service load does not interfere with the requests generation. So, the threads "source" receive the highest priority, the thread "extra load" has an intermediary priority and the threads "server" have the lowest priority. The scheduling policy among the 7 threads of type "server" is described together with the experiment results, in the next section.

The deadlines for the requests were chosen in a way to be spread along the interval of values observed as their response times. For all the service types, the values of the deadlines were randomly generated, with uniform distribution, inside of an interval defined by 1 millisecond and approximately twice the value of the response time average for that service. Each experience lasted 60 seconds, but the data about the quality of the predictions were collected only during the final 40 seconds. A value of 0.01 was used for α.

5.1 Experiment Results

For each experiment, the relative error rates obtained for each tested algorithm are shown. In each experiment, the following algorithms are considered:

- ALWAYS: it always indicates that the deadline will be met;
- NEVER: it always indicates that the deadline will be missed;
- SOSO: it always indicates a probability of 0.5 of meeting the deadline;
- SAMELAST100: it uses a single history record, size 100 for each service type;
- SAMELAST30: it uses a single history record, size 30 for each service type;
- SAMELAST3: it uses a single history record, size 3 for each service type;
- SAMELAST1: it uses a single history record, size 1 for each service type;
- DUALLAST100: it uses a history record with annotation about the system state, with size 100 for each service type;
- DUALLAST30: it is similar to DUALLAST100, size 30 for each service type;
- DUALLAST3: it is similar to DUALLAST100, size 3 for each service type;
- DUALLAST1: it is similar to DUALLAST100, size 1 for each service type.

A first group of experiences used scheduling COLABORATIVE among the threads that the program uses for servicing the requests. When concluding the servicing of a request, the server thread yields the processor to the next server thread, in a circular way. Results are presented in Table 1.

The algorithms ALWAYS and NEVER present a relative error rate close to 0.5, it indicates that around half of the service requests missed their deadlines during the experiment. This was expected because of the way the deadline values were generated. The algorithm SOSO presents a relative error rate of exactly 0.5.

Two conclusions can be easily noticed from the Table 1. Firstly, the modeling of the system state in two levels brings benefits, because the relative error rate of the algorithms DUALLAST are always smaller than those of the algorithms SAMELAST. Secondly, it can be observed that the size of the response time log is not important. A record with only 3 entries was enough to obtain similar

Table 1. Results with colaborative scheduling

Algorithm	Error rate	Algorithm	Error rate
ALWAYS	0.594		
NEVER	0.406		
SOSO	0.500		
SAMELAST100	0.257	DUALLAST100	0.205
SAMELAST30	0.256	DUALLAST30	0.206
SAMELAST3	0.258	DUALLAST3	0.204
SAMELAST1	0.258	DUALLAST1	0.244

results to a record with 100 entries. That is explained by the PMF of the task response times, which presents an enormous tail to the right. Table 2 shows, for each service, the average response times, their standard deviation and their medians. The fact that the average is so much larger than the median indicates the concentration of values to the left of the curve and the existence of some very big values to the right of the curve.

A second group of experiences was done, now using fixed priorities among the threads that service the requests. Lower priority was defined for threads associated with requests "Remove", "Double scan" and "Matrix multiplication." Table 3 shows the results.

A third group of experiences was done with the same program described before, in the same conditions of the first group of experiences, but now executing simultaneously in the computer an anti-virus program, making a complete scan of the main disk. The program anti-virus represents an additional load to the system. The results were very similar to those of Table 1.

An important question is to know if the relative error rate keeps constant for all the services, or the algorithm makes better predictions for one service than for another. Table 4 shows the relative error rate of the algorithms SAMELAST3 and DUALLAST3, in the conditions of the first experiment, according to each service type. It can be noticed that, although difference exists in the quality of the predictions for the different service types, the prediction of DUALLAST3 is always better than the prediction of SAMELAST3.

Table 2. Statistics about response times

Service type	Average (uS)	Standard dev. (uS)	Median (uS)
Insert	900.1	4814.7	44.0
Remove	1099.4	4186.9	219.0
Scan	941.3	4659.1	126.0
Double scan	948.9	4056.0	184.0
Binary search	862.0	4540.4	36.0
Matrix generation	3173.6	6540.7	2090.0
Matrix multiplication	9408.4	7262.8	7827.0

Table 3. Results with priority scheduling

Algorithm	Error rate	Algorithm	Error rate
ALWAYS	0.699		
NEVER	0.301		
SOSO	0.500		
SAMELAST100	0.255	DUALLAST100	0.216
SAMELAST30	0.253	DUALLAST30	0.216
SAMELAST3	0.249	DUALLAST3	0.210
SAMELAST1	0.251	DUALLAST1	0.272

Table 4. Relative error rate by service type

Service Type	Number of Tasks	SAMELAST3	DUALLAST3
Insert	2421	0.317	0.225
Remove	1501	0.176	0.172
Scan	1384	0.226	0.193
Double scan	1377	0.097	0.080
Binary search	2430	0.328	0.225
Matrix generation	803	0.279	0.279
Matrix multiplication	803	0.332	0.298

6 Conclusions

In this paper we considered algorithms to predict the probability of a deadline to be met, in dynamic systems that are not controlled by the algorithm, which has only limited information on the state of the system. The early prediction of a deadline miss allows actions of damage control, such as the signaling of alarms, the use of an alternative computer or some load reduction mechanism.

The relative error rate was defined as the metric to be used for the comparison among different algorithms. Considering the existence of processing and memory limitations, we looked for algorithms that don't demand a great computing power. The paper proposed the use of an algorithm based on the record of previous response times, maintained separately according to the system state being Normal or High-Loaded. The implementation of an example showed the feasibility of the approach and also that the records used don't need to be big.

An open question is how to integrate, in the definition of the system state, the state of the service request queues. The service request queues are an excellent indication of sudden overload, even when this fact doesn't still shows itself in the response times of the services, because the requests are still being serviced. Another important subject is the size of the record to be maintained. The record of response times represents a time window to the past. Since the requests are aperiodic, a record of fixed size represents a time window with variable size.

References

1. Abeni, L., Buttazzo, G.: Stochastic Rate Monotonic Scheduling. In: WPDRTS'01. Proceedings of the 9th International Workshop on Parallel and Distributed Real-Time Systems (April 2001)
2. Atlas, A., Bestavros, A.: Statistical Rate Monotonic Scheduling. In: RTSS'98. Proceedings of the 19th IEEE Real-Time Systems Symposium, Madrid-Spain, pp. 123–132 (December 1998)
3. Binns, P.: Statistical Estimation of Aperiodic Response Times when Scheduled on top of Static Timelines. In: PARTES'04. 1st International Workshop on Probabilistic Analysis Techniques for Real-Time and Embedded Systems (2004)
4. Chu, W.W., Leung, K.K.: Task Response Time Model and Its Applications for Real-Time Distributed Processing Systems. In: Proceedings of the Real-Time Systems Symp., pp. 225–236 (December 1984)
5. Diaz, J.L., Garcia, D.F., Kim, K., Lee, C.G., LoBello, L., Lopez, J.M., Min, S.L., Mirabella, O.: Stochastic Analysis of Periodic Real-Time Systems. In: RTSS'02. Proceedings of the 23rd IEEE Real-Time Systems Symposium, Austin-USA, pp. 289–300 (December 2002)
6. Ferdean, C., Makpangou, M.: Exploiting Application Workload Characteristics To Accurately Estimate Replica Server Response Time. In: Proceedings of DOA (2005)
7. Gardner, M.K., Liu, J.W.: Analyzing Stochastic Fixed-Priority Real-Time Systems. In: Proc. of the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (March 1999)
8. Lehoczky, J.P.: Real-Time Queuing Theory. In: RTSS'96. Proc. of 17th IEEE Real-Time Systems Symposium, Los Alamitos-USA, pp. 186–195. IEEE Computer Society Press, Los Alamitos (1996)
9. Leulseged, A., Nissanke, N.: Probabilistic Analysis of Multi-processor Scheduling of Tasks with Uncertain Parameter. In: Proc. of the 9th International Conference on Real-Time and Embedded Computing Systems and Applications (February 2003)
10. Manolache, S.: Schedulability Analysis of Real-Time Systems with Stochastic Task Execution Times. Licentiate Thesis No. 985, Dept. of Computer and Information Science, IDA, Linköping University, Sweden (December 2002)
11. Tia, T.S., Deng, Z., Shankar, M., Storch, M., Sun, J., Wu, L.C., Liu, J.S.: Probabilistic Performance Guarantee for Real-Time Tasks with Varying Computation Times. In: RTAS'95. Proc. of the 1st IEEE Real-Time Technology and Applications Symposium, Chicago-USA, pp. 164–173. IEEE Computer Society Press, Los Alamitos (1995)
12. Vingralek, R., Breitbart, Y., Sayal, M., Scheuermann, P.: Web++: A System For Fast and Reliable Web Service. In: Proc. of the USENIX Annual Technical Conference Monterey, California-USA (June 1999)