Extending Rate Monotonic Analysis when Tasks Share Buffers

F. Singhoff, J. Legrand, L. Nana, L. Marcé
EA 2215 Team, University of Brest
20, av Le Gorgeu, CS 93837
29238 BREST Cedex 3
{singhoff,jlegrand,nana,marce}@univ-brest.fr

Abstract

In this paper, we study applications composed of periodic tasks and buffers. We assume that tasks are scheduled according to Rate Monotonic. We also assume that tasks read messages from buffers. Messages are received from network at an unknown rate. Checking feasibility of such applications consists in checking task deadlines with classical RMA feasibility tests but also checking that no buffer overflow occurs. We present a RMA extension which makes it possible to study this kind of application. From a queueing system model, a maximum and an average buffer analysis are proposed. This performance analysis is implemented in a tool : the Cheddar Real Time Simulator.

1 Introduction

This paper deals with performance analysis of distributed real time systems. Real time systems we study are composed of processors connected by a network. On each processor, tasks are scheduled according to a preemptive fixed priority scheduler [12]. Tasks are periodically activated and share buffers. Buffer producers and consumers are not synchronized. A task produces or consumes data at its own rate : a consumer can be awaked without having a message to read.

Since 1980, many models, methods and tools were proposed to study performance of this kind of real time systems (eg. Petri Net [13], Synchronous languages [6], ...). One of them, usually called "Rate Monotonic Analysis" (or RMA), provides a set of quantitative methods which helps the system designer to predict the timing behavior of real time tasks. With RMA, task temporal constraints can be checked with scheduling simulation and feasibility tests such as bound on processor utilization factor [12] or task response times [7, 2].

The first ideas about RMA were proposed 30 years ago [12]. The method was strongly extended to cope with many application requirements [8] and was successfully used in many projects [14]. Nevertheless, RMA is still unusable in many cases.

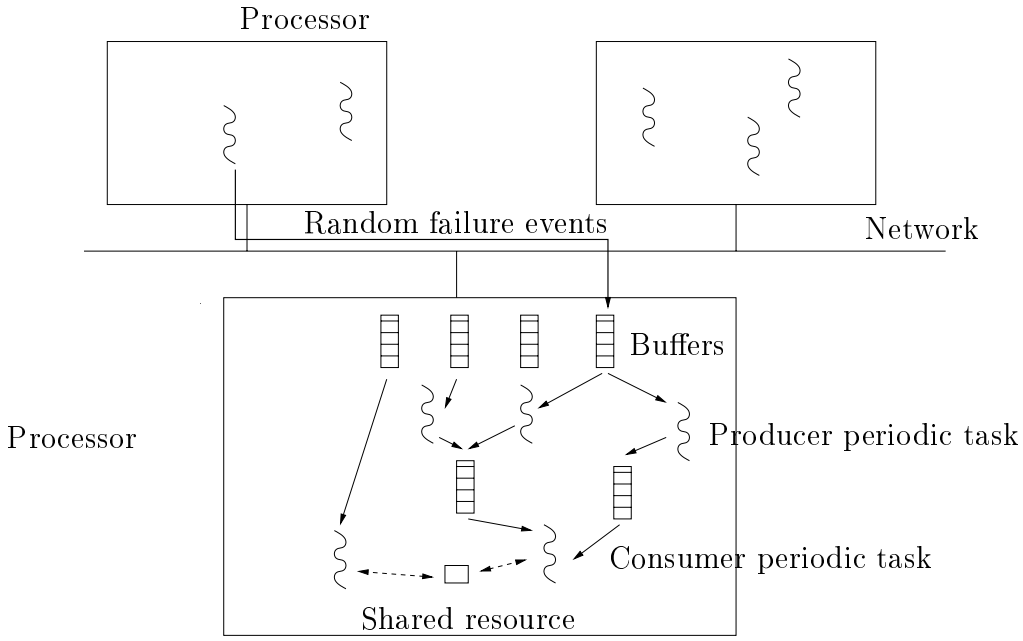

Figure 1: A Distributed Real Time System

First, even if buffers are common operating system functionalities (such as queueing ports in ARINC 653[1]), it seems that few results exist about buffer performance analysis in the case of tasks scheduled according to RMA.

Secondly, with RMA, tasks and messages should most of the time be activated in a periodic way[4]. Unfortunately, in large real time distributed systems, some parts may not be fully periodic. It's the case of monitoring services in aeronautical systems [3] which help operators to detect and diagnose failures (see Figure 1). This kind of real time system can be made of many different equipments, and then, its failure probability can be high. Most of the time, these systems require regular maintenance periods. Sometimes, in order to decrease maintenance cost, they provide services to collect data during the life of the system, to analyze them and to store analysis results into a persistent memory. These analysis results are finally presented at maintenance period. Since failures occur in a random way, this kind of application has to handle random arrival rates of data.

This paper extends RMA to cope with applications composed of buffers where events/messages are delivered to the system in a random way. We propose new

measures of buffer performances. These propositions are implemented in a simulation tool which allows the designer to perform analysis of applications built according to Rate Monotonic Analysis.

RMA applications made of buffers can then be studied using both worst case and average case analysis. We'll show that worst case analysis can be performed if assumptions are made on event arrival rate. In this case, the system is checked assuming that a smallest period of event arrival rate exists. Otherwise, if no worst case assumption is made, we'll show that an average analysis can be realized.

In section 2, we first give a short description of our buffer performance propositions. Section 3 is devoted to the presentation of a simulator which implements these propositions. Finally, in section 4, we conclude and present ongoing works.

2 Performance analysis of buffers shared by tasks scheduled with RMA

Let us go back to the application example of Figure 1. In this monitoring application, we suppose that two kinds of inter-tasks communication exist. First, periodic tasks receive messages from the network. These messages, which provide failure information, are stored in buffers such as ARINC 653 queueing ports. Secondly, tasks share data through buffers and shared resources. In the two cases, we suppose that at most one message/data can be produced or consumed during a periodic task activation. All the tasks are independent and are scheduled with RMA[12].

To check the feasibility of this kind of application, we have to check task deadlines. This can be done with RMA feasibility tests. We also have to check buffer performance. This section deals with buffer performance and mainly focus on mean/maximum waiting time of one message/data in a buffer and on mean/maximum buffer utilization factor.

2.1 Worst case buffer analysis

We first show how to perform buffer analysis if assumptions on worst case failure arrival rate are made. To find a bound on buffer utilization factor in the case of a buffer shared by tasks scheduled with RMA, we can study the case of voice transmission service provided by the AAL2 layer of ATM networks.

In AAL2/ATM, a producer sends audio packets at a fixed rate d. This throughput is expressed in cells per second, the protocol data unit of ATM networks. A bounded variable delay is required by each cell to go from the sender to the receiver. In an AAL2 communication service, the consumer should receive the cell at the same rate the producer sends it. Each received cell

is then buffered during a sufficient amount of time to hide this variable transmission delay. In [5], it has been shown that the size of the buffer used to hide variable transmission delay is bounded by :

$$B = \left\lceil \frac{\delta}{d} \right\rceil \tag{1}$$

Where δ is the maximum delay a cell stays in the buffer. We call this delay the maximum memorization delay.

The systems we study in this paper are similar to the one described above and we can apply equation (1) to find bound on buffers shared by RMA scheduled periodic tasks. Let now suppose that the failure arrival rate is bounded by a period, the smallest period between two successive failures. For a buffer shared by N periodic producers and 1 periodic consumer, the buffer bound is [11] :

$$B = \max_{\forall y \geq 0} \left(\sum_{prod \in PROD} \left\lceil \frac{\delta + O_{prod}}{P_{prod}} \right\rceil - y \right) \tag{2}$$

where $PROD$ is the set of producers, P_{prod} the period of the producer $prod$ and O_{prod} the maximum delay between the wake up time of the consumer and the wake up time of the producer $prod$. This bound is based on the maximum memorization delay which is equal to $\delta = 2(y+1).P_{cons}$ for a given message i (y is the number of messages that may already be in the buffer before the message i is inserted; P_{cons} is the period of the buffer consumer). From (2) and for all possible values of y, it may be proven that for a buffer shared by 1 periodic consumer and N periodic producers, the buffer bound is :

$$B = 2.N$$

if tasks are harmonics[1] and

$$B = 2.N + 1$$

in the other case.

[1] A task set is said to be harmonic if and only if each task period is a positive integer multiple of all smaller task periods.

2.2 Average buffer analysis

The previous section describes how to perform a worst case analysis of buffers shared by RMA scheduled tasks, let's see now the average case.

We can model buffers of our monitoring applications with queueing models [9]. A queueing model is a model which makes it possible to compute measures of performances on a system. The modeled system is composed of servers and customers. Servers run customer requests : a server is awaked when a customer requires it. If new customers arrive in the system when a server is busy, their requests are stored in a queue. By defining the rate of customer request arrivals and the rate of requests that the server can handle, a queueing system model makes it possible to predict the average queue utilization factor and the average customer waiting time.

Different kind of customer arrivals and service time definition exists. The most usual are deterministic (D), Markovian (M) or general (G). D means constant delays (constant service time for the server side or constant delay between two customer arrivals for the customer side). M describes a customer arrival or a service time where delays follow an exponential probability distribution. Finally, if no assumption on the probability distribution is made, G is used. G is defined by an average rate and its variance. Following the Kendall notation, a queueing system is described by at least 3 parameters : $a|b|c$. The a parameter is the customer arrival rate. b describes the service time rate. Finally, c is the number of servers. For instance, a system with one server, a constant service time and an exponential customer arrival is a M/D/1 queueing system.

Let's see now how our monitoring application can be modeled with a queueing system model (a detailed explanation can be found in [10]). Each buffer can be modeled by a queue. Messages from the network are customers and periodic consumer tasks are servers.

The service time is determinist but, due to the RMA scheduling, the service time is different from the one modeled by the D service time distribution :

- A customer request does not require a constant delay of the server. Indeed, the response time of a consumer task depends on its fixed priority.
- Since tasks are periodically activated, a consumer task is not necessary ready to run when a customer request arrives.
- Finally, a consumer task can be awaked even if the buffer is empty.

Then, to perform buffer analysis with a queueing model, we have to define a new kind of deterministic service rate. This new service rate will be called P in the remaining of the paper.

As usual service rates, P is described by a mean service time W_s and its variance σ_s^2. Let's define the mean service time. The message service time is the time used by the server to handle

a message (remains being computed by the server). When there are one or several messages in the buffer, the service time is the time between 2 subsequent consumer activations. On the other hand, when the buffer is empty, the service time is the time between one message arrival and the next consumer activation.

Due to the fact that a consumer task could be awaked when the buffer is empty, we have to study two cases :

1. The production rate is very low compared to the consumption rate. No message is consumed for most of consumer awakening times. In this case, the mean service time is equal to $P_{cons}/2$.
2. The production rate is close to the consumption rate. In the case of heavy queueing utilization factor, each time a consumer is awaked, it has a message to consume. In such case, the mean service time is P_{cons}.

By combining these two cases, an approximation of P mean service time can be found. In order to compute σ_s^2, the response time of each consumer task activation is needed. This information can be obtained with an extension of the classical RMA worst case response time algorithm [7]. Finally, if customer arrival rate is Markovian, the service time for the corresponding M/P/1 queueing system is :

$$W_s = \frac{P_{cons}}{2(1 - \lambda\frac{P_{cons}}{2})} = \frac{P_{cons}}{2}(1 + \rho) \tag{3}$$

Where $\rho = \lambda W_s$ and λ is the mean arrival rate of data in the buffer

The variance on average service time is :

$$\sigma_s^2 = \left(\frac{1 + 3\rho}{4}\right)\frac{1}{n}\sum_{i=1}^{n} W_i - W_s \tag{4}$$

Where W_i is the ith service time of the queueing system server (which is also the ith consumer task response time).

From (3) and (4) and with the help of M/G/1 theoretical results, we can compute a theoretical average message waiting time and a theoretical average number of messages in the buffer. These last measures of performances are computed with these M/G/1 usual equations [9] :

$$W = W_s + \frac{\lambda(W_s^2 + \sigma_s^2)}{2(1 - \rho)} \tag{5}$$

and

$$L = \lambda W \tag{6}$$

Where W is the average message waiting time and L, the average number of messages in the buffer.

3 Cheddar, a real time scheduling simulator

We are developing a tool which helps designers to study RMA applications. This tool, Cheddar, provides features for describing a system or an application composed of tasks, processors, buffers and shared resources. Tasks can be activated periodically or randomly and can run on different processors.

Cheddar is composed of two independent parts : a graphical editor and a framework. The editor is used to describe the real time application. The framework provides a set of feasibility tests and a flexible scheduling simulation engine.

Cheddar runs on Solaris, Linux and win32 systems and the framework is distributed under the GNU General Public License (see http://beru.univ-brest.fr/~singhoff/cheddar). An exhaustive list of services provided by Cheddar is given on the Cheddar's user guide Web pages [15].

Many RMA feasibility tests are provided by the framework (most of the feasibility tests in the uniprocessor case and some of them in the distributed and in the multiprocessor cases). The main feasibility tests are based on response time and processor utilization factor for most of usual schedulers [12, 7]. Of course, if buffers are defined, buffer performance analysis can be performed. These performance analysis are based on results presented in section 2 of this paper.

The simulation engine allows users to display a scheduling in a graphical way. From this simulation, much information can be computed by Cheddar : response times, shared resource blocking times, free time units ... Since feasibility tests are only available for a few well known schedulers, the Cheddar's simulation engine is flexible enough to simulate systems with specific task models or schedulers[16].

We propose the use of a small language to express scheduler behaviors. Schedulers expressed with this language are not compiled but interpreted by the framework at simulation time.

Figure 2 shows an example of an ARINC 653 partitions scheduling with Cheddar [1]. An ARINC 653 system is composed of several partitions. A partition is an unit of program and is itself composed of processes and memory spaces. A processor can host several partitions. Two levels of scheduling exist in an ARINC 653 system : partitions scheduling and processes scheduling.

1. **Processes scheduling.** In one partition, processes are scheduled according to their fixed priority. The scheduler is preemptive and always gives the processor to the highest fixed priority ready task of the partition. When several tasks of a partition have the same priority level, the oldest one is elected.

2. **Partitions scheduling.** Partitions share the processor in a predefined way. On each processor, partitions are activated according to an activation table. This table is built at design time and defines a cycle of partitions scheduling. The table describes for each partition when it has to be activated and how much time it has to run.

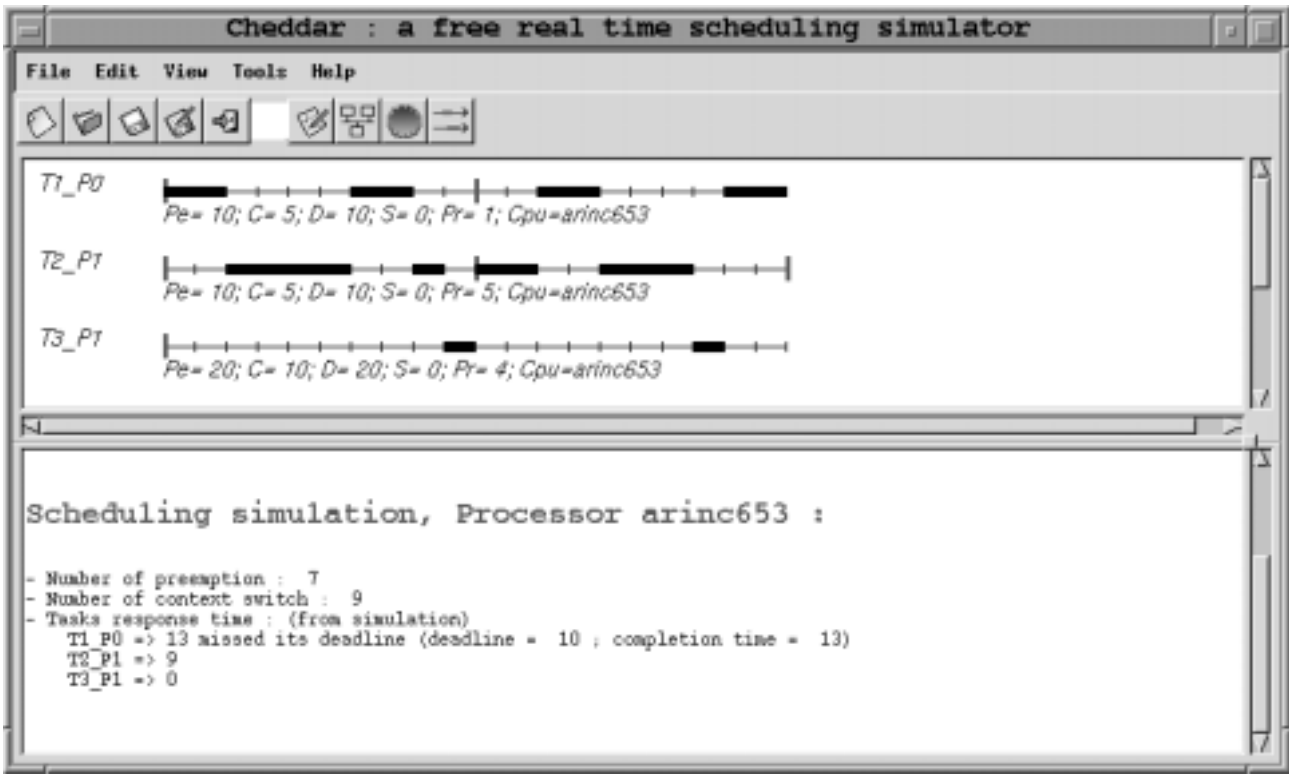

Figure 2: An example of ARINC 653 scheduling with Cheddar

Figure 2 displays a system made of 3 tasks hosted by one processor. The processor owns 2 partitions : partitions P0 and P1. The task T1 runs on the partition P0 and the two others run on the partition P1. Tasks have a fixed priority : T2 is the highest priority level task and T1 is the lowest one. The cyclic partitions scheduling has to be done so that P0 runs before P1. In each cycle, P0 should run during two units of time and P1 should run during four units of time.

This kind of specific scheduling can be easily designed with Cheddar. To achieve this scheduling, a possible user-defined scheduler could be the one given in Figure 3. This user-defined Cheddar scheduler is interpreted at simulation time and is made of several "sections" :

- The **start section** provides variable declarations.
- The **priority section** computes task priorities. The code given here is called each time a scheduling decision has to be taken (at each unit of time for preemptive schedulers and when a task stops running for the non preemptive case).
- The **election section**. In this section, the scheduling simulator engine decides which ready task should receive the processor for the next units of time.

4 Conclusion and future work

This paper proposes extensions to RMA in the case of buffers shared by periodic tasks. The applications we consider are composed of periodic tasks running on uniprocessor systems. These tasks share buffers and receive messages from the network in a random way. Checking feasibility of such applications consists in checking task deadlines with usual RMA feasibility tests but also checking that no buffer overflow occurs.

We have proposed tests to find the maximum bounds on buffer sizes which hold when assumptions on worst case message rate arrival are made. We have also proposed a performance analysis to compute the average size of these buffers if no minimum inter-arrival time between 2 messages can be found.

These propositions are implemented in a GNU GPL real time scheduling simulator. This simulator provides most of usual feasibility tests for Rate Monotonic Scheduling.

Future work should study applications with randomly activated tasks. Today, few results exist for checking temporal constraints of such tasks. We aim at providing feasibility tests (response time and shared resources blocking time) for such tasks running on uniprocessor systems with fixed priority schedulers.

References

[1] Arinc. *Avionics Application Software Standard Interface*. The Arinc Committee, January 1997.

[2] A. N. Audsley, A. Burns, M. Richardson, and K. Tindell. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, pages 284–292, 1993.

[3] B. Burchell. A3XX Maintenance : A First Look. *Overhaul and Maintenance Revue. URL : www.aviationnow.com*, August 2000.

[4] F. Cottet, J. Delacroix, C. Kaiser, and Z. Mammeri. *Scheduling in Real Time Systems.* John Wiley and Sons Ltd editors, 2002.

[5] M. Gagnaire and D. Kofman. *Réseaux Haut Débit : réseaux ATM, réseaux locaux, réseaux tout-optiques.* Masson-Inter Editions, Collection IIA, 1996.

[6] P. Le Guernic, T. Gautier, M. Le Borgne, and C. Le Maire. Programming real time applications with SIGNAL. INRIA-RENNES, Rapport numéro 1446, 1991.

[7] M. Joseph and P. Pandya. Finding Response Time in a Real-Time System. *Computer Journal*, 29(5):390–395, 1986.

[8] M. H. Klein, T. Ralya, B. Pollak, R. Obenza, and M. G. Harbour. *A Practitioner's Handbook for Real Time Analysis.* Kluwer Academic Publishers, 1994.

[9] L. Kleinrock. *Queueing System : theory.* Wiley-interscience, 1975.

[10] J. Legrand, F. Singhoff, L. Nana, and L. Marcé. Performance Analysis of Buffers Shared by Independent Periodic Tasks. Submitted to the IEEE International Real-Time Systems Symposium (RTSS), July 2004.

[11] J. Legrand, F. Singhoff, L. Nana, L. Marcé, F. Dupont, and H. Hafidi. About Bounds of Buffers Shared by Periodic Tasks : the IRMA project. In the 15th Euromicro International Conference of Real Time Systems (WIP Session), Porto, July 2003.

[12] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environnment. *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.

[13] J. L. Peterson. *Petri Net theory and the Modelling of Systems.* Prentice Hall, 1981.

[14] SEI. The Rate Monotonic Analysis. Technical report, In the Software Technology Roadmap. http://www.sei.cmu.edu/str/descriptions/rma_body.html, September 2003.

[15] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Cheddar User's Guide. Technical report, Technical report number singhoff-01-2003, Available at http://beru.univ-brest.fr/~singhoff/cheddar/docs/ug.html, September 2003.

[16] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Cheddar : a Flexible Real Time Scheduling Framework. International ACM SIGADA Conference, Atlanta, November 2004.

```
start_section:
   partition_duration : task array of integer;
   dynamic_priority : task array of integer;
   number_of_partition : integer :=2;
   current : integer :=0;
   time_partition : integer :=0;
   – The activation partition table
   –
   partition_duration[0]:=2;
   partition_duration[1]:=4;
   time_partition:=partition_duration[current];
priority_section:
   if time_partition=0
      then current:=(current+1)
         mod number_of_partition;
      time_partition:=partition_duration[current];
   end if;
   forall i loop
      if task_partition[i]=current
         then dynamic_priority[i]:=priority[i];
         else dynamic_priority[i]:=0;
           ready[i]:=false;
      end if;
   end loop;
   time_partition:=time_partition-1;
election_section:
   return max_to_index(dynamic_priority);
```

Figure 3: Process and partition scheduling in an ARINC 653 system