Scheduling Real Time Tasks: A Performance Study

F. Panzieri  L. Donatiello  L. Poretti

Technical Report UBLCS-93-10

May 1993

Laboratory for Computer Science
University of Bologna
Piazza di Porta S. Donato, 5
40127 Bologna (Italy)
The University of Bologna Laboratory for Computer Science Research Technical Reports are available via anonymous FTP from the area ftp.cs.unibo.it:/pub/TR/UBLCS in compressed PostScript format. Abstracts are available from the same host in the directory /pub/TR/UBLCS/ABSTRACTS in plain text format. All local authors can be reached via e-mail at the address last-name@cs.unibo.it.

UBLCS Technical Report Series

93-7 Distributed Programming with Logic Tuple Spaces, by P. Ciancarini, April 1993.
93-8 Coordinating Rule-Based Software Processes with ESP, by P. Ciancarini, April 1993.
93-14 Guard Compilation in Logic Shared Dataspace Languages, by M. Garpari, June 1993.
Scheduling Real Time Tasks:  
A Performance Study

F. Panzieri\textsuperscript{2}   L. Donatiello\textsuperscript{2}   L. Poretti\textsuperscript{2}

Technical Report UBLCS-93-10

May 1993

Abstract

This paper describes a performance study of a number of task scheduling algorithms conventionally used in the design of distributed real-time systems. That study, based on a simulation model, allows one to assess the adequacy of those algorithms with respect to different parameters that can characterize the distributed system load and its communication costs.

\textsuperscript{1} An earlier version of this work appeared in Proc. International Conference on Modeling and Simulation, Pittsburgh (Pa), May 10 - 12 1993. Partial support for this work was provided by the Commission of European Communities under ESPRIT Programme Basic Research Project 6360 (BROADCAST), and by the Italian National Research Council (CNR) under contract n. 92.00069.CT12.115.25585.

\textsuperscript{2} Dipartimento di Matematica, Università di Bologna, Piazza Porta S. Donato 5, 40127 Bologna (Italy)
1 Introduction

The purpose of this paper is to discuss the results of a simulation study aimed at assessing the adequacy of a number of task scheduling policies usually deployed in the design of distributed real-time (RT) systems.

In general, the principal responsibility of a RT system consists of producing logically correct results, while meeting predefined deadlines in doing so. Hence, the computational correctness of the system depends on both the logical correctness of the results it produces, and the timing correctness (i.e. the ability to meet deadlines) of its computations.

The design complexity of any such system is typically influenced by such issues as the application timing and resource requirements, and the system resource availability. In addition, that complexity can be exacerbated if the RT system is constructed out of geographically dispersed resources, interconnected by some communication network, so as to form a distributed RT system (DRTS). In fact, in a DRTS, the network (and the protocols used for communications over it) can introduce overheads that may affect the computational correctness of the system.

DRTS resources operate under the control of a distributed real-time operating system. That operating system may have to support RT tasks that concurrently compete for cpu, I/O, and network resources. Thus, that operating system has to incorporate appropriate task scheduling policies that meet the RT task requirements.

In order to assess the adequacy of those scheduling policies, we have developed a discrete and stochastic simulation model of a generic DRTS. This model captures the probabilistic nature of various parameters that can characterize the DRTS load, and its communication costs. These parameters include (i) task deadline, (ii) communication overheads, (iii) context switch time, (iv) number of tasks, and (v) scheduling policy.

The principal performance indices that we evaluate include: (i) the resource breakdown utilization [10], and (ii) the normalized mean response time for each task (both these indices are defined in Section 3).

This paper is structured as follows. In the next Section we characterize the class of RT systems we have considered. In Section 3, we describe the parameters and the performance measures of the simulation, and discuss the performance results we have obtained. Finally, in Section 4, we provide some concluding remarks.

2 System Description

A generic real-time system can be thought of as consisting of the following three principal subsystems [5]: a controlled subsystem, that represents the application which dictates the real-time requirements; a control subsystem, that controls some computing and communication equipment for use from the controlled subsystem, and an operator subsystem, initiating and monitoring the entire system activity.

The controlled subsystem consists of tasks (termed application tasks, in the following) that execute using the equipment governed by the control subsystem. This latter subsystem can be constructed out of a possibly very large number of processors, equipped with such local resources as memory and mass storage devices, and interconnected by a high speed network (e.g. a Fiber Distributed Data Interface (FDDI) network [4]). Those processors and resources are mastered by a distributed RT operating system. The principal responsibility
of this operating system is to guarantee that the execution of all the application tasks meet the real-time requirements of those tasks.

The application tasks can be classified, according to their timing requirements, as hard real time (HRT), soft real time (SRT), not real time (NRT) tasks. A HRT task is a task whose timely (and logically correct) execution is deemed as critical for the operation of the entire system. The deadline associated to a HRT task is conventionally termed hard deadline, owing to the critical nature of that task. As a consequence, it is assumed that missing a hard deadline can result in a catastrophic system failure. A SRT task, instead, is characterized by an execution deadline whose adherence is indeed desirable, although not critical, for the functioning of the system (hence, the SRT task deadline is usually termed soft deadline). NRT tasks are those tasks which exhibit no real-time requirements (e.g. system maintenance tasks that can run occasionally in the background).

Application tasks can be further classified as periodic, aperiodic, and sporadic tasks. Periodic tasks are those tasks that enter their execution state at regular intervals of time, i.e. every T time units. These tasks, generally used in such applications as signal processing and control, are typically characterized by hard deadlines [7]. Aperiodic tasks are those tasks whose execution time cannot be anticipated, as their execution is determined by the occurrence of some external event. These tasks are usually characterized by soft deadlines. Finally, aperiodic tasks characterized by hard deadlines are termed sporadic tasks [10] (e.g. tasks dealing with the occurrence of system failures).

2.1 System Parameters
In view of this latter task classification, we have developed a DRTS simulation model that incorporates a variety of scheduling algorithms, suitable for the scheduling of periodic, aperiodic, and sporadic tasks [2]. In addition, in order to support the simulation of I/O request management, our model implements different I/O controller management policies (note that the network interface itself is modeled as an I/O device). Finally, our model embodies a number of task synchronization protocols that implement concurrency control.
mechanisms, and solve (or prevent [1]) the priority inversion problem [8] (see below).

For the purposes of this paper, we shall not describe here the algorithms mentioned above (they are discussed at length in the already cited references). However, it is worth mentioning that these algorithms have been chosen as they are representative, to the best of our knowledge, of the most widely used algorithms (for their relative scope) in RT environments. Hence, in the following, we shall introduce the principal characteristics of these algorithms, only.

**Scheduling Algorithms.** Our model implements the Rate Monotonic (RM), the Earliest Deadline First (EDF), and the Least Slack Time First (LSTF) algorithms [7] for the scheduling of periodic tasks. Each of these algorithms is priority driven and preemptive. However, the RM algorithm is a static scheduling algorithm, as the priority assigned to a task is maintained unaltered during entire life time of that task. Instead, the EDF and the LSTF algorithms are dynamic algorithms, as the task priorities can be changed at run time. Aperiodic task scheduling can be supported, in our model, by means of the Background (BG), the Polling (PL), the Deferrable Server (DS), and the Sporadic Server (SS) [6] algorithms. The BG scheduling algorithm is implemented by executing aperiodic tasks in those time intervals in which no periodic tasks are active. The PL, DS, and SS algorithms are implemented by periodic servers that schedule aperiodic tasks at regular intervals of time, provided that no periodic task be in execution. Finally, the scheduling of the sporadic tasks is simulated by implementing a periodic server, fully dedicated to the scheduling of those tasks, that is enabled sufficiently frequently to guarantee not to miss the sporadic task hard deadlines.

**I/O Management.** In our model, the scheduling of tasks accessing I/O resources can be governed by one of the preemptive scheduling algorithms mentioned above (i.e. the RM, the EDF, and the LSTF algorithms). In addition, our model allows its user to choose a FIFO discipline for I/O resource management, and to specify arbitrary network delays.

**Task Synchronization and Priority Inversion.** The phrase ‘priority inversion’ is used to indicate the situation in which the execution of a higher priority task is delayed by lower priority tasks [3]. With priority driven RT schedulers, this problem can occur when there is
3 Simulation Model

In order to simulate the mastering and control of that problem, our model implements the Basic Priority Inheritance (BPI), the Priority Ceiling (PC), the Priority Limit (PL), and the Semaphore Control (SC) protocols [9]. The principal scope of each of these four protocols is to minimize the so-called Worst Case Blocking Time, i.e. the time interval in which the execution of a higher priority task can be delayed by lower priority tasks. An alternative approach to the solution of the priority inversion problem has been proposed in [1], and is based on preventing the occurrence of that problem. In order to assess the effectiveness of that approach, our model incorporates a particular priority prevention protocol described in [1].

3 Simulation Model

Our simulation model has been implemented, using the C programming language, so as to accept in input a description of the DRTS to simulate, and to produce, as output, statistical results of the simulation experiments.

The input DRTS description consists of the specification of both system load, and operating system parameters. The former include the following random variables: number of periodic (PT) and aperiodic tasks (AT) that may request execution, the task period (P), the cpu request (CR) and the deadline (D) of each task, and their probability distribution. The latter include the scheduling and task synchronization policies the operating system is to use, and the random variables: operating system preemption cost (PrC), and network overhead (NO).

The output produced by our implementation is intended to allow one to evaluate the performance of the various algorithms mentioned above. In particular, our model provides the following two figures of merits:

- **Breakdown Utilization** (BU): as defined in [10], this is the degree of resource utilization at or below which the operating system can guarantee that all the task deadlines will be met. This figure provides a measure of the effectiveness of the selected scheduling
policy, as the larger the breakdown utilization, the larger the cpu time devoted to task execution;

- **Normalized Mean Response Time** (NMRT): this is the ratio between the time interval in which a task becomes ready for execution and terminates, and the actual cpu time consumed for the execution of that task. Yet again, this figure provides a measure of the effectiveness of the selected scheduling policy as, the larger the Normalized Mean Response Time, the larger the task idle time.

### 3.1 Simulation Study

In our simulation study, we have examined the performance of the algorithms introduced in section 2.1, under a variety of different system load conditions, and operating system characteristics. The results we have obtained are discussed below.

#### 3.1.1 Periodic Tasks

To begin with, the BU obtained with the RM, EDF and LSTF algorithms, for the scheduling of periodic tasks, have been examined as a function of the operating system preemption cost. In the following, we shall assume that the task deadline coincide with the task period, and that the cpu request of a generic task \(i\) is ‘generated’ from the uniform distribution in the interval \([0, p_i]\), where \(p_i\) denotes the task period. The simulation results discussed in this Subsection have been obtained by using the method of independent replications (300 independent runs for each experiment), and 95% confidence intervals have been constructed for the performance indices.

Assuming that:

1. PrC is the same for each one of these three algorithms,
2. PT is a constant, equal to 10,
3. P is uniformly distributed in the interval \([1, 100]\),

our results show that the EDF and LSTF dynamic algorithms perform better than the RM static algorithm, as illustrated in Figure 1. However, in practice, the above assumption 1
can be unrealistic, as the dynamic algorithms must compute and assign the task priorities at run time, thus introducing additional overheads to the preemption cost; hence, the use of the RM algorithm can be favored to that of the dynamic algorithms, as its implementation is simpler, and the preemption cost it entails is lower.

This observation has led us to concentrate our investigation on the RM algorithm, as far as periodic task scheduling is concerned. Thus, we have examined its behavior as the number of tasks in execution grows. In addition, we have considered the following four different probability distributions of the random variable $P$:

1. Uniform distribution in $[1, 100]$ (variance = 816.8),
2. Beta distribution with parameters $a = 15$ and $b = 15$ (variance = 79.4), and parameters $a = 0.5$ and $b = 0.5$ (variance = 1862.2),
3. Normal distribution with parameters mean = 50.5 and variance = 78.4,
4. Exponential distribution with parameter $a = 0.5$.

The Beta, Normal and Exponential distributions are scaled in the interval $[1, 100]$. The results produced by our simulation model are illustrated in Figure 2.

This Figure shows that the RM algorithm is extremely sensitive to the variance of the random variable $P$. In particular, low variance can notably degrade the RM scheduling performance. In essence, this can be explained as follows. The RM algorithm assigns higher priority to tasks with shorter periods. Thus, if the task period random variable has low variance, the different task periods are characterized by short time intervals between the periods’ terminations. Owing to this observation, we have developed an algorithm that allocates independent tasks to the DRTS cpus, so as to provide a high variance for $P$ on each of these cpus.

Figure 3 depicts the result provided by our model as a function of the number of cpus. This Figure illustrates that a conventional task allocation algorithm (indicated as Normal in Figure 3), that ignores the task distribution issue by, for example, polling each cpu in the system until it finds one available for task execution, produces very low BU values compared to our allocation algorithm (indicated as Enhanced in Figure 3).
3.1.2 Aperiodic Tasks

NMRT is the most relevant figure of merit when aperiodic tasks are introduced in a DRTS, and coexist with the periodic tasks. The experiment we have carried out consist of simulating the presence (on the same cpu) of both periodic and aperiodic tasks. We assume that:

- the scheduling algorithm used is the RM algorithm,
- the periodic task load is about 69%, and the number of periodic tasks is 10, with period uniformly distributed in the interval [1,100],
- the number of aperiodic tasks is 10,
- the time between consecutive activations of each aperiodic task is exponentially distributed with mean equal to 20,
- the aperiodic task server is the task with highest priority.

The NMRT simulation results we have obtained, as a function of the aperiodic task load, show that the bandwidth preserving algorithms (i.e the DS, SS, IS algorithms) perform better than such traditional algorithms as polling and background, as depicted in Figure 4.

Essentially, this is because the aperiodic task execution can start any time during the server period. Thus complex algorithms, such as DS, SS, and IS, allow the scheduler to start rapidly the execution of the aperiodic tasks. Compared with easier methods, such as polling, these algorithms meet effectively the execution requirements of those aperiodic tasks that request short execution time (even if these requests are very frequent). However, we have observed that, when an aperiodic task requires an amount of cpu execution time close to that of the most time consuming task of the system, the differences among the various methods tend to disappear.

3.1.3 I/O Scheduling

As pointed out in [11], I/O requests are scheduled, in general, with FIFO discipline. However, this can lead to a low resource utilization, as illustrated in Figure 5. The results shown in this Figure have been obtained by simulating a system characterized as follows:
1. a variable number of periodic tasks, with period $P$ uniformly distributed in the interval $[1,100]$, are concurrently running in the system,

2. every task is divided in three parts: input, processing and output. We assume that the time spent during the I/O phase is the same consumed for processing data.

We consider both preemptive and non preemptive controllers. A non preemptive controller is one that cannot interrupt an I/O operation once this has been started. With a preemptive controller, instead, a high priority I/O operation can preempt a lower priority one. Consequently, the use of a preemptive controller may appear to be more appropriate in a Real Time system. However, if the RM algorithm is implemented in order to assign priorities to the tasks (and hence to the task I/O requests) the following non obvious results can be observed.

We have carried out a number of simulations that show the performance differences (in terms of Resource Breakdown Utilization) between the preemptive and the non preemptive controller.

We have examined the behavior of these two controllers when the task periods are generated with a variety of different distributions. Figure 6 show the Resource Breakdown utilization obtained when the task periods are uniformly distributed in the interval $[1,B]$. It can be seen that the preemptive controller can lead to a greater resource Breakdown Utilization for a limited number of values of $B$, only. Using a low variance distribution (i.e. the normal distribution with variance equal to 78.4) for the period random variable $P$, we have obtained that for all values of $B$, the Breakdown Utilization achieved by the non preemptive controller is always greater than that achieved by the preemptive controller. In contrast, using a high variance distribution (i.e. a beta distribution with variance equal to 1862.2) the preemptive controller exhibits its superiority, as illustrated in Figure 7. Moreover, we have noted that the difference in terms of performance between the two kinds of controllers tend to disappear as the number of tasks grows.

Thus, in conclusion, the benefits that can be obtained using a preemptive controller cannot be considered as absolute, as these benefits depend upon the system load.
3.1.4 Priority Inversion

The priority inversion problem can typically occur when RT tasks share data. Concurrent accesses to those data can be handled by means of concurrency control mechanisms such as semaphores. However, if a low priority task locks a semaphore, higher priority tasks which require that semaphore are forced to wait its release, thus incurring in a so-called blocking time overhead. Priority control protocols that limit this overhead in a RT system have been developed in order to guarantee the tasks deadlines (i.e. the BPI, SC, PL, and PC protocols already mentioned). As illustrated in Figure 8, these protocols exhibit different performance; in particular, as the blocking time grows, the BPI degrades notably. Instead, the performance of the PC, the SC, and the PL protocols maintain values which are very close to each other (the PL protocol performance results are omitted from Figure 8). However, the SC protocol is an optimal but hard to implement protocol; hence, a number of recent RT system implementations (e.g. Real Time MACH [13]) favor the use of the PC protocol.

Finally, our simulation model implements a recently proposed priority prevention protocol [1]. This protocol differs from the priority control protocols previously examined as it is capable of eliminating the priority inversion problem. Using this protocol, the analysis of a RT system is indeed easier, as less effort is required to construct a feasible schedule for that system. However, the performance of this priority prevention protocol turns out to be lower than that obtained with the priority control protocols discussed above, as illustrated in Figure 9.

4 Concluding Remarks

In this paper we have discussed the results we have obtained from a simulation study of real time task scheduling algorithms. These results indicate the effectiveness of the various scheduling algorithms we have considered, under different system load and preemption cost conditions.
We have obtained these results using a simulation model that we have implemented on HP 700 workstations; the programming language we have used is the C language. The principal motivations for using this particular programming language are its efficiency and flexibility.

Finally, we would like to mention that we are currently working on the definition of an analytical model in order to express rigorously our results.

Acknowledgements We wish to thank our colleague Ozalp Babaoglu for his useful comments concerning our results on the priority prevention protocol, and Hewlett-Packard Italiana S.p.A. for providing the hardware and software infrastructure used for the development of the simulation model described in this paper.

References


