Response-time analysis and server design for hierarchical scheduling

Luís Almeida
DET/IEETA, Universidade de Aveiro
Aveiro, Portugal
lda@det.ua.pt

Abstract

Hierarchical scheduling has been generating a considerable interest, recently, due to its ability to separate the concerns of scheduling at the system and application levels. It allows sharing resources, e.g. a CPU or a communication network, among different complex applications, which in turn, are composed by several entities that must be scheduled within a given server. It is also an important brick in the current trend towards higher integration and flexibility that can be witnessed, for example, in embedded systems. This paper addresses the particular case of fixed priorities-based application-level schedulers together with a deferrable server model, at the system level. It starts with an adequate response time analysis based on the notion of server availability for a known server. Then it addresses the inverse problem of designing a server with minimum system-level resource requirements to fulfill the application time constraints. The proposed method seems to represent a positive trade-off concerning complexity and tightness when compared to another method presented recently in the literature. The impact of intra and inter-server blocking as well as the specific case of non-preemptive scheduling in the server design are the object of on-going work.

1. Introduction and related work

Hierarchical scheduling has been generating a considerable interest, recently, due to its ability to separate the concerns of scheduling at the system and application levels. It is a fundamental brick in the current trend towards higher integration and flexibility in embedded systems [8], which opens the way to higher efficiency and lower costs by means of resource sharing, as well as to higher resilience to hardware failures by means of dynamic reallocation of computing or communication entities [7].

Hierarchical scheduling is intimately connected with resource partitioning according to which a shared resource, e.g. CPU or network, is used by several complex applications each of which is composed of a set of entities, e.g. tasks or streams. These entities must be scheduled internally to the application inside one specific resource partition to which they were allocated. At a higher level, all resource partitions are scheduled using a given system-level policy. The concept of server is well adapted to this level, supporting temporal isolation among partitions.

The problem of hierarchical scheduling bears many resemblances with other scheduling problems that have been tackled in the past, such as those regarding exclusions [5] and inserted idle-time [9][2]. In fact, looking to the system from the perspective of one application executing within a server, the periods of time in which the respective server cannot execute, e.g. because another server has been scheduled at the system level, can be seen as exclusion periods or periods of inserted idle-time. Moreover, in specific scopes, such as real-time communication over shared media, some forms of hierarchical scheduling have long been used, e.g. TDMA (as in TTP/C), as well as the multi-phased cyclic framework (as in WorldFIP, FlexRay or FTT-CAN [1]). In both cases, either slots or phases can be taken as servers that are scheduled periodically.

However, recent work has brought to light new results that are more general. In [7] the authors use the concept of virtual resource and server supply function, and deduce utilization bounds for schedulability within partitions. In [6] the focus is on hierarchical resource partitioning using fixed priorities local schedulers and both deferrable and sporadic servers. It presents a response-time analysis as well as utilization bounds. In [4], the authors deduce a tighter schedulability test for partitions using fixed priorities local schedulers and address the issue of server design in order to meet the application time constraints.

This paper relates closely to the works referred above. We will consider a fixed priorities local scheduler together with a deferrable server model to manage a resource partition. Then, we will adapt a previously developed method [1] to derive upper bounds to the worst-case response time of tasks executing within a given server. We will also use the concept of server supply function as in [7] but we will refer to it as availability function for coherence with our previous work. The response-time analysis we propose is equivalent but different to that in [6] and we believe it is more flexible, being easily adapted to non-preemption, as in [1], and to irregular partitions as in [1] and [7]. Finally, we also address the problem of server design as in [4] but our method seems to represent a more favorable compromise between tightness and complexity. We are currently addressing the impact of intra and inter-server blocking in the server design problem.

2. Task model

In this work, we consider that a given active resource, say a CPU, is used to execute a set of independent applications.

This work was partially supported by the European Commission (accompanying measure ARTIST, IST-2001-34820).

Each application W is composed by a set G_W of N_W tasks, $G_W = \{\tau_i(C_i,T_i,D_i,J_i,P_i),\ i=1..N_W\}$, in which each task, at this point, will be considered independent and fully preemptive. Each task τ_i is characterized by a period T_i, a worst-case execution time C_i, a relative deadline D_i that is shorter than or equal to the respective period, a maximum release jitter of J_i and a fixed priority P_i that may possibly be derived from the period, the deadline, or any sub-optimal criterion. Also, the set of tasks G_W executes by priority order within a deferrable server S_W, which is characterized, at a system level, by a period T_S and a capacity C_S.

The system-level scheduler that schedules partitions, and the current system load, determine when the server is available to execute application tasks. We thus define the server *availability function*, $A_S(t)$, which returns for each instant the cumulative CPU time available for the application to execute, since an arbitrary time origin. However, if an on-line system-level scheduling policy is used, the specific pattern of $A_S(t)$ is difficult to predict. Therefore, for the sake of independence with respect to the system-level scheduling policy and load, it is helpful to use a lower bound that assures that, at a given instant, the total CPU time available for that application is never smaller than a given value.

In order to determine a lower bound to the availability function, we can use the same reasoning as in [4], or equivalently the one in [1]. Basically, it consists on considering the worst-case server availability pattern with respect to the arbitrary time origin, in which the server suffers maximum latency (D) in the beginning and then follows a periodic pattern with its capacity available at the end of each periodic instance (fig. 1). The value of D depends directly on the maximum jitter that the periods of server availability may suffer, as determined by the system scheduler. It may vary between T_S-C_S (no jitter), e.g. as in a TDMA schedule, and $2*(T_S-C_S)$ (absolute maximum jitter) as in fig. 1. For the sake of generality we will consider such initial latency as given by

$$D = (1+\beta)*(T_S-C_S)$$

where $0 \le \beta \le 1$ represents the actual maximum server availability jitter ($\beta=0 \rightarrow$ *no jitter*, $\beta=1 \rightarrow$ *max abs jitter*). To maintain independency from the system scheduling policy $\beta=1$ should be considered.

For simplicity, in the remainder of the paper we will use the same expression *availability function* for the lower bound function, and we will refer to it as $A\__S(t)$ (1). In [4] expression (1) is referred to as server characteristic function and in [7] as server supply function.

$$A\__S(t) = \begin{cases} 0, & t < \Delta \\ t - (\Delta + k*(T_S - C_S)), & \Delta + k*T_S \le t < \Delta + k*T_S + C_S \\ (k+1)*C_S, & \Delta + k*T_S + C_S \le t < \Delta + (k+1)*T_S \end{cases} \quad (1)$$
$$k = \lfloor (t-\Delta)/T_S \rfloor$$

3. Response-time analysis

A relatively simple but effective way of upper bounding the response-time for each task τ_i within a given server S is to use the same reasoning explained in [1]. The fact that we are now using a fully preemptive task model together with a deferrable server further simplifies the analysis therein presented, which was based on non-preemption with inserted idle-time together with a background server.

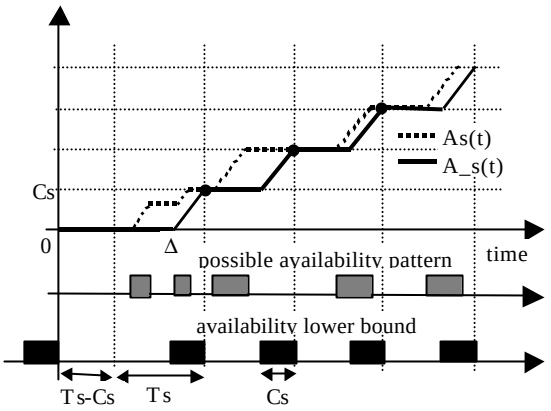

Figure 1 – Server availability functions $A_S(t)$ and $A\__S(t)$

Therefore, we compute for each task τ_i and for each instant t the maximum load submitted to the server by the task itself together with all higher priority tasks. We call this the *level i submitted load function*, $H_i(t)$ (2), which can be determined by the usual methods in fixed-priorities response time analysis [3]. In fact, the critical instant for each task is not changed by the presence of the server [6]. Expression (2) considers G_W sorted by decreasing priorities, $\forall_{i,j}\ i<j \Rightarrow P_i>P_j$.

$$H_i(t) = \sum_{j=1}^{i} \left\lceil (t+J_j)/T_j \right\rceil * C_j \quad (2)$$

An upper bound to the worst-case response time for task τ_i, referred to as R_i, is obtained by determining the earliest instant in which the maximum submitted load $H_i(t)$ matches the least server availability $A\__S(t)$ (fig. 2). A trivial schedulability test can then be implemented comparing R_i and D_i for all tasks.

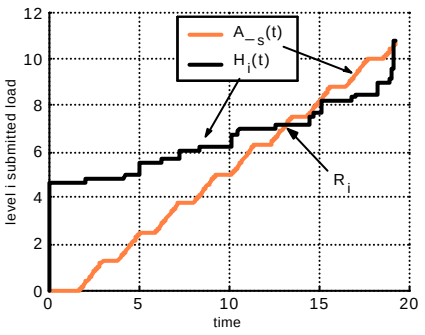

Figure 2 – Worst-case response time of task τ_i.

The fact that the application is restricted to run within the server S is already taken into account by using the availability function. In order to promptly determine R_i, we can make use of the inverse of the availability function, referred to as $A^{inv}{}_S(t)$ and formalized in (3). R_i can thus be expressed as in (4).

$$A^{inv}{}_S(u) = \Delta + m*T_S + u - m*C_S, \quad m = \lceil u/C_S \rceil - 1 \quad (3)$$

$$earliest\ R_i:\ R_i = A^{inv}{}_S(H_i(R_i)) \quad (4)$$

To solve equation (4) we can use the iterative procedure

$$R_i^0 = H_i(0) \quad and \quad R_i^{n+1} = A^{inv}{}_S(H_i(R_i^n))$$

that either converges to $R_i = R_i^{n+1} = R_i^n$ or grows beyond the deadline D_i, within a finite number of iterations. This can easily be demonstrated by the fact that, from iteration to iteration, the increment in $H_i(t)$ is lower bounded by $min_{j=1..i}(C_j)$. If the whole CPU is available to execute the application, then $A\_{S}(t) = A^{inv}{}_S(t) = t$ and expression (4) reduces to the usual response time analysis for fixed priorities scheduling for a similar task model [3].

4. Server design

In this section we tackle the opposite problem of the previous one, i.e. given an application W, which parameters should the server $S_W(C_S,T_S)$ have so that it requires the least system resources and still meets the application time constraints? In order to address this problem we will start by noticing that such a server should be as tight as possible concerning fulfilling the application time constraints, otherwise it would over consume system resources. Therefore, using the approach presented in the previous section, we will consider that, for each task τ_i the respective R_i is just on the deadline D_i. This means that the availability function $A\_{S}(t)$ should be such that intersects $H_i(t)$ exactly at $t=D_i$. To achieve this, we start by defining the set of *deadline points* $DP_W = \{DP_i(D_i,H_i(D_i)), i=1..N_W\}$ (fig. 3), which represent the lowest availability demand required for meeting all deadlines of the application. Our purpose, then, is to define $A\_{S}(t)$ so that it is higher then but as close as possible to the set of such points.

In this paper we follow a simplified approach similarly to the one in [4], in which the availability function $A\_{S}(t)$ is further lower bounded by a piecewise linear curve referred to as $A\_'_S(t)$, which has the same initial latency Δ as $A\_{S}(t)$ and then grows linearly with slope $\alpha = C_S / T_S$, i.e. the server bandwidth (fig. 3).

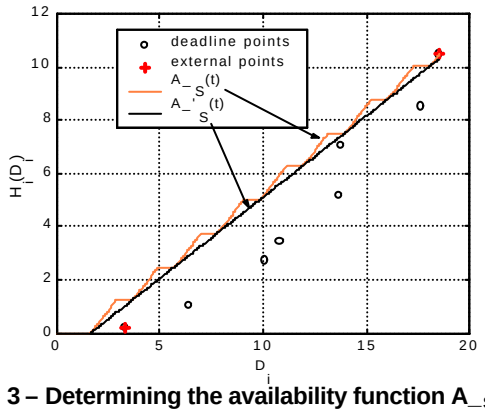

Figure 3 – Determining the availability function A_S(t).

Given this simplification, the problem is now reduced to finding the lowest $A\_'_S(t)$ curve that passes above or through all deadline points DP_i. For this we determine the subset of *external points*, $E_W \subseteq \{E_j(x_j,y_j), j=1..N_E\} \subseteq DP_W$ (fig. 3) through which a straight line can be drawn that passes over all other deadline points and fulfills the limitations of the availability function, namely $\alpha \leq 1$ (the server cannot use more than the total CPU bandwidth) and, for each E_j, $\alpha \geq x_j/y_j$ (otherwise, $\Delta<0$ and, since $\alpha>0$, it would lead to $T_S<0$ and $C_S<0$). The determination of the E_W subset is carried out using a simple algorithm that goes through all deadline points in ascending deadline order, starting with the assumption that the first deadline point is an external point. Notice that the slopes of the segments that join every two consecutive external points must be monotonically decreasing from the maximum $\alpha \leq 1$ to the minimum $\alpha \geq x_j/y_j$. Therefore, the algorithm removes from the set of deadline points all those that would violate this decreasing slope pattern. At the end, the points that remain are the external points. The time complexity of the algorithm varies with the characteristics of the task set between N_W and N_W^2. Also, in general, the number of external points N_E is substantially smaller than the number of deadline points N_W (i.e. the number of tasks), which contributes to decrease the complexity of the remaining part of the process.

The set of external points defines a solution space for the server design problem. In fact, all the $A\_'_S(t)$ curves that touch at least one external point with a valid slope are possible solutions. Valid slopes are those in between the slopes of the segments that join each point with the previous and with the next, considering the absolute maximum and minimum for the points in the extremes. This can be represented by the union of the following N_E subintervals, $\alpha \in \{[1\ \alpha(E_1,E_2)] \cup [\alpha(E_1,E_2)\ \alpha(E_2,E_3)] \cup \ldots \cup [\alpha(E_{N_E-1},E_{N_E})\ x_{N_E}/y_{N_E}]\}$. Each of these subintervals corresponds to $A\_'_S(t)$ curves that pass through one external point $E_j(x_j,y_j)$ ($j=1..N_E$) and can be characterized by a straight-line equation as in (5).

$$y = \alpha(x - x_j) + y_j \quad and \quad \Delta = x_j - y_j/\alpha \qquad (5)$$

The equation on the right also allows determining the respective server period T_S doing $T_S = \Delta/((1+\beta)(1-\alpha))$.

Finally, in order to find one specific solution, one can use the cost function proposed in [4], which considers both the bandwidth directly requested by the server, $\alpha = C_S/T_S$, as well as the overhead bandwidth implicitly used by the server in context switching between applications at the system level. This latter factor can be roughly computed as C_O/T_S where C_O is a system parameter representing the context switching time. The cost function can thus be expressed as $F = c_1*\alpha + c_2*C_O/T_S$. Minimizing F corresponds to finding the balance between minimizing α and maximizing T_S.

Considering the straight-line equation (5) of each α subinterval, i.e. each E_j external point, one can express T_S as a function of α and then use it within the cost function F. The α of minimum cost for each subinterval j ($\alpha_{j,min}$) can be easily determined with a closed formula (6) obtained by differentiating F with respect to α and calculating the respective root.

$$\alpha_{j,min} = y_j/x_j * \left(1 + \sqrt{1 - \frac{(y_j - (1+\beta)*C_O)/(x_j - (1+\beta)*C_O)}{y_j/x_j}}\right) \qquad (6)$$

Whenever $\alpha_{j,min}$ lies outside the respective subinterval, the closest extreme is considered for minimum. After having determined $\alpha_{j,min}$ for all subintervals ($j=1..N_E$) it is then just a

matter of selecting the one that generates the absolute minimum cost (α_{min}). It is also necessary to identify to which subinterval the α_{min} value belongs to, in order to use the respective equation (5) to determine Δ and then T_S.

The server parameters generated this way are not optimal due to several factors such as the pessimism included in the server initial latency Δ, the successive approximations of the effective server availability function $A_S(t)$ and the use of the deadline points that may lead to worse than necessary response time upper bounds. However, concerning the $A\_{S}(t)$ approximation by $A\_'_S(t)$, it is possible to carry out a simple final improvement step that consists in shifting the availability function $A\_{S}(t)$ to the right until it touches one deadline point. This is carried out increasing the server period T_S as stated in (7), which also leads to a further reduction in α. The amount of benefit, however, depends on the task set. With 1000 random task sets with uniformly distributed periods, we achieved an average (max) of 1.8% (8%) increase in T_S and 1.7% (7.4%) reduction in α.

$$T_{S,imp} = T_S + \min_{i=1..N_\Omega}\left(\frac{D_i - A^{inv}{}_S(H_i(D_i))}{\lfloor (D_i + (1+b)*C_S)/T_S \rfloor}\right) \qquad (7)$$

In [4] the final minimum cost server parameters are not determined, just the solution space in terms of possible (α,Δ) pairs. When using the same example case, i.e. $\Gamma_W=\{(C_i,T_i)=(1,4), (1,11), (3,25)$ and $(D_i=T_i, J_i=0, P_i=1/i)\}$ we obtain the set of deadline points $DP_W=\{(4,1), (11,4), (25,13)\}$ from which two are external points $E_W=\{(4,1), (25,13)\}$. Therefore, we obtain two α subintervals and using equation (5) we can establish the (α,Δ) solution space. Informally, we can see (fig. 4) that our solution space is contained in the one derived in [4] and thus it is worse, although for a small difference (less than 4% in the low values of α). On the other hand, the worst-case time complexity of the method in [4] may reach 2^{N_W}-1, contrasting with the simplicity of our method with a worst-case time complexity of N_W^2. Executing the final improvement step in (7) generates an operating point (fig. 4) that is better than both solution spaces.

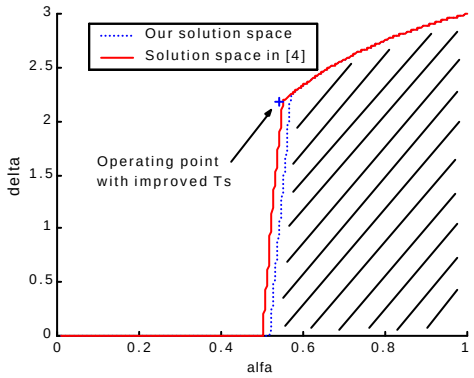

Figure 1 – Comparing the (α,Δ) solution space.

5. Conclusion and on-going work

This paper considered the case in which an application composed by several tasks executes within a deferrable server with a fixed priorities local scheduling policy. Two main results are presented, the response time analysis for such tasks and the design of the server to allow fulfilling the application time constraints using the least system resources.

The former contribution is a generalization of the well-known worst-case response time analysis for fixed-priority systems [3] that copes with the limited processor availability delivered by a server and it is based on the analysis previously developed for traffic scheduling within the asynchronous messaging system of FTT-CAN [1].

The latter contribution goes in the same direction as that of [4] but presents a different method that leads to a more favorable compromise between tightness of the solution (slightly lower) and complexity of the process (substantially lower). This fact supports the possibility of using the proposed method on-line, in dynamic systems in which new applications can join the system on-line and request a server for execution. We also believe that the proposed method is better suited to include other constraints such as inter and intra-server blocking as well as irregular partitions. This is subject of on-going work. We are also working to derive the server availability function directly from the deadline points of the application, without using the approximation of the availability to a piecewise linear curve. This is expected to improve the tightness of the server design method.

Acknowledgment

The author would like to thank Enrico Bini for the fruitful discussions concerning the server design problem that helped in improving the respective part of this paper.

References

[1] Almeida L., P. Pedreiras, J. A. Fonseca, The FTT-CAN Protocol: Why and How, *IEEE Transactions on Industrial Electronics*, **49(6)**, December 2002.

[2] Almeida L., J. Fonseca. Analysis of a Simple Model for Non-Preemptive Blocking-Free Scheduling. *Proc. of ECRTS'01 (EUROMICRO Conf. on Real-Time Systems)*. Delft, Holland. June 2001.

[3] Audsley, N., A. Burns, M. Richardson, K. Tindell and A. Wellings. Applying New Scheduling Theory to Static Priority Pre-Emptive Scheduling. *Software Engineering Journal*, **8(5)**: 285-292, 1993.

[4] Bini, E., G. Lipari. Resource Partitioning among Real-Time Applications. *Proc. of ECRTS'03 (EUROMICRO Conf. on Real-Time Systems)*. Porto, Portugal. July 2003.

[5] Xu, J., D.L. Parnas. Scheduling processes with release times, deadlines, precedence and exclusion relations. IEEE Trans. on Software Engineering, **16**:360-369 March 1990.

[6] Saewong, S., R. Rajkumar, J.P. Lehoczky, M.H. Klein. Analysis of hierarchical fixed priority scheduling. Proc. of ECRTS'02 *(EUROMICRO Conf. on Real-Time Systems)*. Vienna, Austria. June 2002.

[7] Mok, A., X. Feng. A model of hierarchical real-time virtual resources. Proc. of RTSS'02 (IEEE Real-Time Systems Symposium). Austin, USA. December 2002.

[8] Rushby, J., A Comparison of Bus Architectures for Safety-Critical Embedded Systems, *CSL Technical Report, SRI International*, September 2001.

[9] Howell, R. and M. Venkatrao. On Non-Preemptive Scheduling of Recurring Tasks Using Inserted Idle Times. *Information and Computation*, **117**, 1995.