SCHEDULING IMPRECISE COMPUTATION TASKS WITH INTRA-TASK / INTER-TASK DEPENDENCE

R. S. Oliveira [*] **and J. S. Fraga** [+]

[*] *UFRGS, II, CP 15064, Porto Alegre-RS 91501-970, Brazil*
[+] *UFSC, LCMI, CP 476, Florianopolis-SC 88040-900, Brazil*

Abstract. This work presents two heuristics to be used as admission policy in imprecise task systems. It is assumed that imprecise tasks have intra-task and inter-task dependence and a 0/1 constraint. These heuristics are supposed to be used combined with off-line schedulability tests and on-line acceptance tests already described in the literature. The objective of the admission policy is to maximize system utility through the selection of optional parts for execution. Simulation is used to analyze the proposed heuristics.

Keywords. Real-time, scheduling algorithms, flexible, dynamic.

1. INTRODUCTION

It has been proposed (Audsley, *et al.*, 1994a; Davis, *et al.*, 1995) an approach that integrates off-line and on-line algorithms to achieve both guarantee for critical tasks and flexibility to improve system utility. An off-line schedulability test is used to guarantee that certain tasks will always meet their deadlines. Then, an on-line acceptance test is used to accept additional requests to improve system utility, without jeopardizing previously guaranteed tasks.

It is possible that, sometimes, the set of additional tasks available for execution is larger than what could be accepted. In this case, an admission policy is used to decide which tasks, among the available additional tasks, will be offered to the acceptance test. Those tasks that are approved by the admission policy and by the acceptance test will be executed. The admission policy is like a filter that discards additional tasks that are not able to add much utility.

The imprecise computation technique (Liu, *et al.*, 1994) allows some flexibility in the scheduling of real-time systems. Tasks are composed of a mandatory part and an optional part. In case of an overflow, optional parts may not be executed. Some imprecise task models consider that each task precisely executed adds some value to the system. Some other models consider that each task not completely executed adds some error to the system.

The multiple versions technique can be used to implement an imprecise task. In its simplest form, each imprecise task has two versions. The primary version is able to generate a precise result, but has a bigger worst-case execution time. The secondary version is able to generate an acceptable (imprecise) result, within a smaller worst-case execution time. The secondary version corresponds to the mandatory part of the task. The difference between the two versions defines the optional part. When multiple versions are used, it is necessary to decide before the task starts which version will execute. Once a task started its execution, it is no longer possible to review that decision. Also, the optional part will be completely executed (primary version) or not executed at all (secondary version). This requirement is called a 0/1 constraint.

Most imprecise computation studies assume that the value or the error associated with a task is already defined by the moment the task arrives. It is supposed not to be affected by what happens to other

tasks or to previous requests of the same task. However, there are works that assume some kind of dependence among tasks. In (Feng and Liu, 1994) the task model includes inter-task dependence. Error in the input data of a task may extend its own mandatory and/or optional worst-case execution times. Examples of tasks with inter-task dependence appear in applications such like speech recognition, radar tracking and image processing for autonomous navigation. The task model described by Chung, *et al*. (1990) includes intra-task dependence. It is assumed that every task must be precisely executed at least once every Q requests. Examples of tasks with intra-task dependence appear in applications such like radar tracking and control systems. In those two works, the error model was such that an optional part could become mandatory or increase the overall mandatory execution time, losing its "optional" character.

This paper presents a set of heuristics to be used as admission policy when the system is made of imprecise tasks. These heuristics are supposed to be used combined with off-line schedulability tests and on-line acceptance tests already described in the literature. The off-line test will guarantee that at least the mandatory part of each task will always be finished before its deadline. The on-line acceptance test checks if the execution of a specific optional part would jeopardize the execution of mandatory parts. The objective of the admission policy is to maximize system utility through the selection of optional parts. Simulation is used to analyze the proposed heuristics with respect to this purpose. It is also assumed that imprecise tasks have intra-task and inter-task dependence and are programmed with the multiple versions technique. Differently from the work of Feng and Liu (1994) and Chung, *et al*. (1990), in this paper optional parts remain always optional. Even if never executed, they neither increase the mandatory execution time of any task in the system nor become themselves mandatory in any sense.

The remainder of the paper is organized as follows: section 2 contains the task model and the scheduling problem. Section 3 describes heuristics that can be used as admission policies. Section 4 describes the simulations used to value the performance of the many heuristics. Section 5 shows the simulation results. Section 6 contains the closing remarks.

2. PROBLEM FORMULATION

It is assumed a system with N periodic or sporadic tasks that execute in a single processor. Every task has a fixed priority that is unique in the system. Without loss of generality, it is assumed that all tasks were named such that T_i has a higher priority than T_j if and only if i<j. Each task T_i is the source of an infinite number of requests. Each request of T_i is composed of a mandatory part and an optional part. The worst-case execution times of both parts are known off-line and denoted by M_i and O_i, respectively. It is also assumed a 0/1 constraint for the optional parts.

Each task T_i has a period (periodic task) or a minimum inter-arrival time (sporadic task) denoted by P_i. It also has a deadline D_i, relative to its arrival time. It is supposed that $\forall T_i$, $D_i \leq P_i$. It is necessary an off-line guarantee that the mandatory part of every task is always completed before its deadline. An optional part only adds value to the system when it is finished before the deadline of the task.

It is assumed that every task T_i has an associated nominal value V_i that represents the utility of task T_i to the system. The nominal value does not take into account the intra-task or inter-task dependence. At run-time, every release $T_{i,k}$ of task T_i is associated to an effective value $V_{i,k}$. The effective value is defined by the nominal value and the dependence of the tasks. The value that $T_{i,k}$ adds to the system will be zero when the optional part is not executed or $V_{i,k}$ when it is executed. The general goal of the scheduling solution is to maximize the total value of the system. The total value is the sum of the values added by every request of each task. Throughout this paper V(t) is used to denote the total value of the system at instant t. The value $\Phi(t)$ represents the total processor time, within interval [0,t), that was not used to run mandatory parts. The system average value density at instant t is denoted by $\Lambda(t)$ and given by V(t) divided by $\Phi(t)$.

The intra-task dependence between releases $T_{i,k}$ and $T_{i,k+1}$, of task T_i, is modeled by the assumption that an imprecise execution of $T_{i,k}$ will increase the effective value of the precise execution of $T_{i,k+1}$. The effective value of release $T_{i,k+1}$, denoted by $V_{i,k+1}$, will be:

V_i when $T_{i,k}$ is precisely executed;

$V_i + (\alpha_i . V_{i,k})$ when $T_{i,k}$ is imprecisely executed;

where α_i is the recovery rate of task T_i. It describes the amount of value that can be recovered after a specific request was not precisely executed.

It is supposed that inter-task dependence appears in subsets of tasks that are always released at the same time. The different priorities establish a precedence relation among the tasks of the subset. A task with higher priority is executed before and will be able to change the behavior of those tasks of the subset with lower priorities.

The inter-task dependence between releases $T_{j,k}$ and $T_{i,k}$, j<i, is modeled by assuming that a precise execution of $T_{j,k}$ will reduce the worst-case execution times of both mandatory and optional

parts of $T_{i,k}$. In other words, the worst-case execution time of $T_{i,k}$ will become:
$M_i + O_i$ when $T_{j,k}$ is imprecisely executed;
$\beta_{j,i} . M_i + \gamma_{j,i} . O_i$ when $T_{j,k}$ is precisely executed;
where $\beta_{j,i}$ and $\gamma_{j,i}$, $0<\beta_{j,i}<1$, $0<\gamma_{j,i}<1$, are the reducing factors linking T_j to T_i. When the execution time of T_i depends on more than one task, it is necessary to apply the reducing factors of each task that affects T_i and had its release k precisely executed.

The value density $\lambda_{i,k}$ of the optional part of release $T_{i,k}$ is defined as the effective value of $T_{i,k}$ divided by the worst-case execution time of its optional part, after the corrections due to inter-task dependence. In other words:

$$\lambda_{i,k} = \frac{V_{i,k}}{O_i \times \prod_{\forall j.T_j \in Pred(T_{i,k})} \gamma_{j,i}} \qquad (1)$$

where $Pred(T_{i,k})$ denotes the set of tasks that affect the worst-case execution time of T_i and which respective release k was completely executed. The product of $\gamma_{j,i}$ is due to the fact that many tasks can precede T_i and affect its behavior. The definition of $\lambda_{i,k}$ takes into account the possible reduction of the worst-case execution time of the optional part of T_i due to inter-task dependence.

3. DESCRIPTION OF HEURISTICS

This section contains the description of heuristics that can be used as admission policy in the task model considered. The first two policies (FCFS and AVDT) were described in (Davis, *et al.*, 1995). The other two policies are proposed in this paper.

3.1 First-Come-First-Serviced (FCFS)

When FCFS is used, every optional part is always considered for execution. Every time the execution of a request is about to start, the acceptance test is applied to its optional part. If it is accepted, then the precise version of the task is executed. This policy does not take into account the different task values.

3.2 Adaptive Value Density Threshold (AVDT)

In this policy the run-time support maintains an updated measure of the system average value density $\Lambda(t)$. The value $\Lambda(t)$ is used as a minimum limit for the value density of optional parts. Only optional parts that have a value density $\lambda_{i,k}$ higher than $\Lambda(t)$ will be considered. Those from this set that pass the acceptance test will be executed.

3.3 Compensated Value Density Threshold (CVDT)

This admission policy, introduced in this paper, applies the same basic ideas of AVDT. As before, only optional parts that have a value density $\lambda_{i,k}$ higher than a threshold will be considered for execution. Those from this set that pass the acceptance test will be executed. Differently from AVDT, the minimum limit used by policy CVDT, denoted by ζ, is given by:

$$\zeta = \Lambda(t) \times Min(5 \times \pi(t), 1.1) \qquad (2)$$

where $\pi(t)$ denotes the rejection rate of the acceptance test, that is, the number of rejected optional parts divided by the number of tested optional parts.

As before, the system average value density $\Lambda(t)$ is used. It is now multiplied by a correction factor that takes into account the rejection rate of the acceptance test. When the rejection rate is 20%, the value $\Lambda(t)$ itself is used. When the rejection rate is higher than 20%, signaling that the admission policy is admitting too many candidates, the threshold value is increased. When the rejection rate is lower than 20%, signaling that the acceptance test is receiving too little candidates, the threshold value is reduced. The number 1.1 appears in equation 2 only to limit the effect of the correction factor. The correction factor can vary from zero (when $\pi(t)=0$) to 1.1 (when $5\times\pi(t)\geq 1.1$). The values 1.1, as an upper limit for the threshold, and 20%, as an ideal rejection rate, were chosen based upon many experiences. It is possible that, for applications of another kind, a fine adjust of those values would result in a better performance.

3.4 Compensated Value Density Threshold for Inter-Task Dependence (INTER)

This policy is similar to the previous two in the sense that it compares a value density with a minimum limit. As before, only optional parts that have a value density higher than the threshold will be considered for execution. Policy INTER compares the minimum limit ζ (eq. 2) with $\lambda_{i,k}+\varepsilon_{i,k}$, that is, value density $\lambda_{i,k}$ plus a correction factor $\varepsilon_{i,k}$.

The definition of correction factor $\varepsilon_{i,k}$ takes into consideration that a precise execution of $T_{i,k}$ will reduce the mandatory execution times of its successors tasks. The importance of a precise execution of $T_{i,k}$ is augmented by the possible future execution of other optional parts. These future executions make use of the reduction of the worst-case mandatory execution times of the tasks that are successors of $T_{i,k}$. It is assumed that those future executions will observe a value density of $0.5\times\zeta$.

The definition of $\varepsilon_{i,k}$ also takes into account that a task may have more than one predecessor. The worst-case mandatory execution time $m_j(t)$, of release $T_{j,k}$, includes corrections due to those predecessors of T_j that are already precisely executed. It is calculated by:

$$m_j(t) = M_j \times \prod_{\forall h.T_h \in Pred(T_{j,k})} \beta_{h,j} \qquad (3)$$

The correction factor $\varepsilon_{i,k}$ is given by:

$$\varepsilon_{i,k} = \frac{0.5 \times \zeta \times \sum_{\forall j.T_j \in Succ(T_i)} [(1-\beta_{i,j}) \times m_j(t)]}{O_i \times \prod_{\forall h.T_h \in Pred(T_{i,k})} \gamma_{h,i}} \qquad (4)$$

where ζ is the minimum limit given by equation 2; $Succ(T_i)$ represents the set of tasks that may have its worst-case execution time modified by T_i; $Pred(T_{i,k})$ and $Pred(T_{j,k})$ denote the set of tasks that had their release k precisely executed and can affect the worst-case execution times of, respectively, T_i and T_j.

4. SIMULATION

The heuristics described in the previous section are evaluated by simulation. The performance of the heuristics proposed in this paper will be compared with the performance of two previously published policies, that is, FCFS and AVDT. In order to run a simulation, it is necessary to define which off-line schedulability test will be used to guarantee that mandatory parts are schedulable. It is also necessary to define which on-line acceptance test will be applied. It is important to note that the heuristics presented in this paper do not specifically require the use of the tests chosen for the simulation.

It is assumed in this paper that each task receives a fixed priority in concordance with the Deadline Monotonic policy (Leung and Whitehead, 1982). Ties are broken arbitrarily, so each task has a unique priority. It is used here the exact schedulability test described by Audsley, *et al*. (1993).

As in (Davis, *et al*., 1995), it is applied an acceptance test based on the "Dynamic Approximate Slack Stealing" (DASS), described by Audsley, *et al*. (1994b). This algorithm keeps a slack counter for each priority level. These counters are consulted to check if an optional part can be accepted. Note that the guarantee associated with the optional part of release $T_{i,k}$ is given by the time the task is about to start its execution, instead of at the moment of its release. In this way, the acceptance test can take advantage of any slack generated by higher priority tasks that execute and then are finished between the release and the start of $T_{i,k}$.

The task load was defined in a way similar to others works in the literature (Audsley, *et al*., 1994b; Davis, *et al*., 1995). In all simulations, the task set was composed of 18 periodic tasks. The task set comprised 6 tasks with period between 20 and 200 ticks, another 6 tasks with period between 200 and 2000 and the remaining 6 tasks with period between 2000 and 20000. The deadlines were random numbers, from an uniform distribution in the range limited by 20 and the period of the respective task.

The worst-case execution times of the mandatory parts were also randomly chosen, but in a way that the total mandatory utilization would be 30%, 60% or 90%. Only task sets of which the mandatory parts could be guaranteed off-line were simulated. The worst-case execution times of the optional parts were again randomly chosen, but this time to result a total utilization varying from 90% to 300%. There is not a previously defined ratio between the optional and the mandatory worst-case execution times of any task. For each combination of mandatory and optional utilization, next section shows the average over 20 different task sets

Nominal values V_i were chosen randomly from a uniform distribution in the range 1 to 10. Both the recovery rates α_i and the reduction factors $\beta_{i,j}$ were again chosen from a uniform distribution, this time between 0 and 1. The reduction factors $\gamma_{i,j}$ were assumed to be 1 for all tasks. In order to evaluate the effect of inter-task dependence, it was assumed that the 18 tasks were organized in 6 triples. Tasks that belong to the same triple have the same period and inter-task dependence.

5. SIMULATION RESULTS

In this section the simulation results are summarized by tables 1 to 9. The main concern of the simulations is the system total value generated by each heuristic. This total value is compared to that obtained by FCFS with the same task load. For example, a figure of 1.5 means that the heuristic at hand was able to generate 150% of the value generated by FCFS, both working with exactly the same task load. Obviously, the figure associated with FCFS will always be 1.

The simulations assume that the acceptance test overhead is zero. Since all simulated admission policies apply the same acceptance test, this overhead is the same for all policies. Also, The computational costs of the admission policies were not included in the simulations. It is important to note that the effective values and the worst-case execution times can be kept by the run-time support. They would be updated every time a task is completed. In this case, algorithms FCFS, AVDT and CVDT have a complexity of *O(1)*. Policy INTER

has a complexity of *O(N)* as a result of the sum that appears in the fractional part of equation 4.

Tables 1 to 3 show the simulation results when there is only intra-task dependence. Policies CVDT and INTER show the same behavior. That was expected since $\varepsilon_{i,k}=0$ in the absence of inter-task dependence and the two heuristics become the same. In all simulated loads CVDT and INTER were better than FCFS and AVDT. The better performance of CVDT and INTER with respect to AVDT is due to the capacity of the former two of detecting when the processor is underloaded and the threshold should be reduced. This advantage becomes less important as the mandatory load is increased.

Table 1: Intra-task only, 30% mandatory utilization.

Option.	AVDT	CVDT	INTER
60%	0.966	1.015	1.015
90%	1.063	1.159	1.159
120%	1.194	1.302	1.302
150%	1.265	1.397	1.397
180%	1.374	1.533	1.533
210%	1.449	1.610	1.610
240%	1.596	1.747	1.747
270%	1.513	1.692	1.692

Table 2: Intra-task only, 60% mandatory utilization.

Option.	AVDT	CVDT	INTER
60%	1.167	1.251	1.251
90%	1.343	1.460	1.460
120%	1.545	1.676	1.676
150%	1.537	1.638	1.638
180%	1.559	1.662	1.662
210%	1.702	1.788	1.788
240%	1.692	1.773	1.773

Table 3: Intra-task only, 90% mandatory utilization.

Option.	AVDT	CVDT	INTER
60%	1.924	1.931	1.931
90%	1.893	1.931	1.931
120%	1.713	1.735	1.735
150%	1.753	1.761	1.761
180%	1.713	1.718	1.718
210%	1.779	1.791	1.791

Tables 4 to 6 show the simulation results when there is only inter-task dependence. Again, CVDT and INTER show a better performance than AVDT. Although policy INTER takes explicitly into account the inter-task dependence, that results in higher figures than CVDT only with a high mandatory load (table 6). Policies CVDT and INTER show similar results with a mandatory load of 30% and 60%.

Table 4: Inter-task only, 30% mandatory utilization.

Option.	AVDT	CVDT	INTER
60%	0.917	1.004	1.004
90%	0.978	1.135	1.133
120%	1.175	1.364	1.364
150%	1.276	1.503	1.506
180%	1.465	1.739	1.733
210%	1.495	1.831	1.832
240%	1.701	1.964	1.965
270%	1.689	2.039	2.039

Table 5: Inter-task only, 60% mandatory utilization.

Option.	AVDT	CVDT	INTER
60%	0.943	1.076	1.079
90%	1.171	1.352	1.365
120%	1.433	1.669	1.681
150%	1.494	1.759	1.773
180%	1.630	1.906	1.925
210%	1.794	2.064	2.078
240%	1.710	1.937	1.976

Table 6: Inter-task only, 90% mandatory utilization.

Option.	AVDT	CVDT	INTER
60%	1.205	1.285	1.450
90%	1.469	1.543	1.884
120%	1.590	1.654	2.147
150%	1.909	1.942	2.111
180%	1.782	1.828	2.151
210%	1.682	1.623	1.986

Tables 7 to 9 show the simulation results when there are both types of dependence. They are similar to those observed when there is only inter-task dependence. Policies CVDT and INTER have better figures than AVDT. Also, policy INTER is better than CVDT only when the mandatory load is 90%.

Table 7: Intra-task and inter-task, 30% mandatory.

Option.	AVDT	CVDT	INTER
60%	0.954	1.002	1.002
90%	0.999	1.092	1.091
120%	1.137	1.246	1.244
150%	1.211	1.344	1.344
180%	1.325	1.478	1.478
210%	1.389	1.552	1.548
240%	1.542	1.684	1.680
270%	1.483	1.673	1.673

It is not possible to directly compare the results presented above with the results presented by Chung, *et al*. (1990) and by Feng and Liu (1994). In (Chung, *et al.*, 1990) there is an obligation to guarantee that the optional parts of all tasks would be eventually executed. In (Feng and Liu, 1994),

although a specific optional part could never be executed, it would then cause an increase of the execution times of mandatory parts. In the work presented here, neither dependence type is able to increase the system mandatory load.

Table 8: Intra-task and inter-task, 60% mandatory.

Option.	AVDT	CVDT	INTER
60%	0.981	1.059	1.060
90%	1.150	1.253	1.261
120%	1.358	1.487	1.497
150%	1.387	1.506	1.516
180%	1.390	1.512	1.549
210%	1.566	1.663	1.675
240%	1.564	1.665	1.679

Table 9: Intra-task and inter-task, 90% mandatory.

Option.	AVDT	CVDT	INTER
60%	1.213	1.252	1.398
90%	1.455	1.467	1.716
120%	1.516	1.546	1.820
150%	1.666	1.651	1.774
180%	1.546	1.575	1.731
210%	1.531	1.550	1.773

It is possible to compare the policies presented in this paper and AVDT. Both CVDT and INTER present, in general, better results than AVDT. However, the difference is small. The best results of CVDT compared to AVDT are due to the threshold used. In moments of low load, AVDT rejects optional parts with a low value density, even if the processor is idle. The minimum limit used by CVDT takes into account the processor utilization level.

Policy INTER explicitly considers the inter-task dependence. It shows better results than AVDT and CVDT in the presence of this kind of dependence. When both mandatory and optional loads are high, factors β and γ become more important. Policy INTER correctly takes into consideration the future gains made possible by executing an optional part that will reduce the execution times of other tasks.

6. CONCLUSION

This paper described two new heuristics that can be used as admission policy in an imprecise task system. Their behavior was analyzed through simulation and compared with two others algorithms already described in the literature. This paper considers those systems where there is dependence between different releases of the same task and/or there is dependence between different tasks. The proposed heuristics showed a performance that, in general, was better than the other two heuristics.

It is important to note the dynamic nature of many values considered by the heuristics, such that, $\Lambda(t)$ and $\pi(t)$. These values should be taken from the recent past and not from the whole life of the system. In this way the heuristics would be better prepared to deal with a mode change where the task values are changed. It is an open question how to determine the specific time interval to be considered.

Other questions remain opened. It was assumed $\gamma=1$ for all tasks. Probably, lower values of γ would increase the difference between the performance of INTER and the performance of the others heuristics. At the same time, if it was taken into account the overhead incurred by the admission policies, the most affected policy would also be INTER. Finally, it is not clear how imprecise computation concepts could be incorporated into software design.

REFERENCES

Audsley, N.C., A. Burns, M.F. Richardson, K. Tindell and A.J. Wellings (1993). Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling. *Soft. Eng. J.*,**8(5)**,284-292.

Audsley, N.C., A. Burns, R.I. Davis and A.J. Wellings (1994a). Integrating Best Effort and Fixed Priority Scheduling. *Proc. of Workshop on Real-Time Program.*, Germany, june 1994.

Audsley, N.C., R.I. Davis and A. Burns (1994b). Mechanisms for Enhancing the Flexibility and Utility of Hard Real-Time Systems. *Proc. IEEE Real-Time Sys. Symp.*, Puerto Rico, pp. 12-21.

Chung, J.-Y., J.W.-S. Liu and K.-J. Lin (1990). Scheduling Periodic Jobs That Allow Inprecise Results. *IEEE Trans. Comp.*, **39(9)**, 1156-1174.

Davis, R., S. Punnekkat, N. Audsley and A. Burns (1995). Flexible Scheduling for Adaptable Real-Time Systems. *Proc. IEEE Real-Time Tech. and App. Symp.*, may 1995, pp. 230-239.

Feng, W. and J.W.-S. Liu (1994). Algorithms for Scheduling Tasks with Input Error and End-to-End Deadlines. Tech. Rep. #1888, Dept. of Comp. Sci., Univ. of Ill. at Urbana-Champaign.

Leung, J.Y.T. and J. Whitehead (1982). On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks. *Perf. Eval.*, **2(4)**, 237-250.

Liu, J.W.-S., W.-K. Shih, K.-J. Lin, R. Bettati and J.-Y. Chung (1994). Imprecise Computations. *Proc. of the IEEE*, **82(1)**, 68-82.