Microprocessing and Microprogramming 39 (1993) 119–124
North-Holland

Determining effective multiprocessor scheduling policies for repetitive real–time tasks

S. Papadimitriou[a] A. Kameas[b] and G. Pavlides[c]

[a]Dept. of Computer Engineering & Informatics, University of Patras, 26110 Patras, Greece

[b]Dept. of Computer Engineering & Informatics, University of Patras, 26110 Patras, Greece

[c]Dept. of Computer Engineering & Informatics, University of Patras, 26110 Patras, Greece

This work presents an improved solution to the problem of determining effective multiprocessor scheduling policies for real-time repetitive tasks. In order for this approach to be applicable to some task, the probability density function of its execution time, a value function and a criticality factor are required. When these parameters are available, the proposed method constructs effective schedules for repetitive tasks that yield acceptable intermediate results; in addition, these schedules tend to degrade gracefully under "heavy" load conditions. A three–phase process is proposed: first a balanced allocation of tasks is attempted using the rate–monotone first–fit algorithm; then the schedule for each processing cluster is constructed by using a linear programming formulation of the task system. This formulation considers many important parameters not considered by previous approaches, and constitutes the proposed improvement over previous work. Finally, the schedule for the entire system is derived from the partial schedules.

1. INTRODUCTION

In designing a CASE tool that supports the interactive specification and prototyping of distributed real–time applications [4], the problem of optimizing the execution of periodic real–time tasks which are scheduled on multiprocessors came up. Applications of this kind (for example, real–time signal processing, real–time decision support etc) impose large processing requirements. Furthermore, since the tasks of these applications must be executed repetitively within a certain time limit, intermediate imprecise results are produced.

In this paper, a new approach to solve this problem is presented, which differs from traditional ones in that it tries to avoid timing faults by allowing the return of intermediate results of acceptable (though not optimal) quality. This method resulted from the application of the methodology presented in [4, 5] to scheduling. Prerequisites for its application are the availability for every task T_i of the probability density function $f_{X_i}(t)$ that describes its execution time, of a value function $V(T_i, t)$ that gives the earned value from allocating t time units to T_i and also of a criticality factor $\mathcal{K}_i$. When this information is available the proposed method produces effective schedules which in addition tend to degrade gracefully under heavy load conditions. It is important to note that the static nature of the periodic tasks allows extensive (off-line) optimizations to be performed, a property that suits our case.

Related work in this field [2] considers only the criticalities of some task, which are represented with the use of weights. Our work considers in addition to the task criticalities:

- the slope of the probability distribution curve, which expresses the benefits from allocating additional time to the task, and
- value functions, which express the importance of better results for a task; this factor clearly depends upon the role of the task in the particular application.

In the next section, a formulation of the scheduling problem is given together with a model of the system where the tasks will be executed. Then the proposed solution is described.

2. MODELING THE PROBLEM

Let us assume that there are N real-time tasks $(T_1, T_2, \ldots, T_N)$ running at a tightly coupled shared memory multiprocessor system which consists of M processing clusters interconnected by a high-speed bus [8]. In addition, each processing cluster includes a tightly coupled shared-memory multiprocessor with m_c processors. Due to the large and usually non-deterministic overheads associated with task migrations, the tasks are executed only at the processing clusters on which they were initially loaded. The existence, however, of fast hardware support for context switch operations and of a versatile mechanism by which one processor can continue a task left by another after locating it in shared memory, can make the preemption overheads very low even if the task is restarted on another processor (of the same cluster). This low overhead permits the use of *preemptive* scheduling. Note however that a task cannot be in execution in more than one processors at the same time.

For each task T_i, its ready time t_{r_i} (the time where the task is ready for execution) and its deadline t_{d_i} (the time where execution of the task must terminate) are defined. Then, the feasibility interval of the task is the time between its ready time and its deadline. Denote by P_i, $i = 1, 2, \ldots, N$ the periods of the N periodic tasks. After a time interval equal to the least common multiple S of all the tasks, all of them are invoked again and the order of invocations in $[t_s + S, t_s + 2 \cdot S]$ is the same as in $[t_s, t_s + S]$, where t_s is the *start time*, that is the time of the first invocation. A planning cycle [6] is defined as any interval of the form $[t_s + i \cdot S, t_s + i \cdot S + S)$, $i \in N$. In figure 1, three tasks (T_1, T_2, T_3) are depicted, with periods 30, 20 and 10 time units, respectively. The planning cycle is 60 time units, and contains six instances of T_1, three instances of T_2 and two instances of T_1 (namely, $T_{1.1}$ and $T_{1.2}$).

To solve the scheduling problem we must find for each processor i, $i = 1, 2, \ldots, M \star m_c$ at interval $[0, PC)$ (first planning cycle) a set of tuples $S_i = (t, T_j, c)$ meaning that at time t, processor i schedules task T_j and allows time c to it (a preemptive schedule). Furthermore, a task can be scheduled only in the processing cluster in which it is allocated. The execution times assigned to the task by the schedule must be within its feasibility interval.

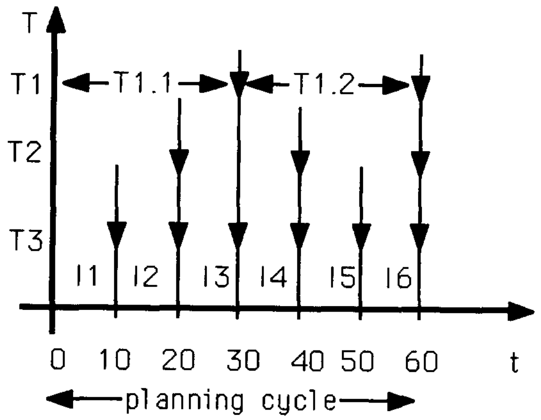

Figure 1. Example of a planning cycle

For each task T_i the distribution of its execution time can be taken either by the solution of the appropriate formal models (e.g. Continuous Time Markov Chain models [4, 5]) or by performing statistical analysis [7] of a large number of measurements. The first method is particularly suitable for systems that can be modelled with Petri-Net or equivalent (e.g. Data-Flow Graphs) structures, as is the case being modeled here. The Probability Distribution Function (PDF) of the execution time of the task represents the probability for the task to return a final result depending on how long it is being executed; it can be defined as $\Pr_i = F_{X_i}(C_i) = \Pr(X_i \le C_i)$, where C_i is the computation time assigned by the schedule to task i, with execution time described by the random variable X_i and $\Pr_i$ the probability of completion of its execution within this time. Repetitive tasks, however, may converge too slowly to a final result; that is why, levels of interme-

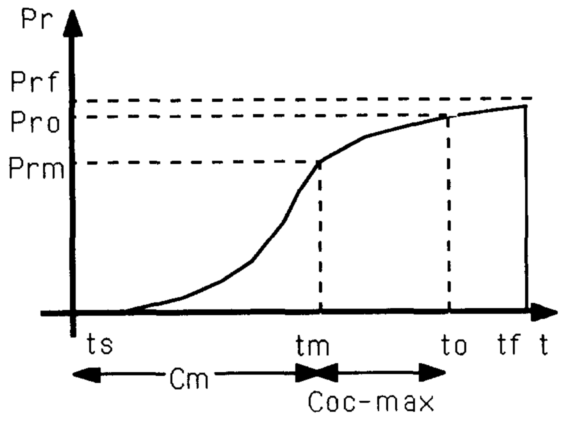

Figure 2. The PDF of a task's execution time

diate acceptable results must be defined. To represent such levels, two probabilities are used: the *mandatory probability* $\Pr_{m_i}$ and the *optimized probability* $\Pr_{o_i}$, $\Pr_{o_i} \geq \Pr_{m_i}$. The mandatory probability represents a good probability to complete successfully the execution of a task instance or, at least, return an acceptable intermediate result, under non-contending conditions; each task must be executed until its mandatory probability is reached. The optimized probability is the probability for the task to return an otpimal (final) result. It is $\prec 1$ for tasks that do not converge, or converge too slowly after a long time. In such cases, execution of the task must stop when an acceptable optimal result has been produced (refer to figure 2).

In actual cases, additional workload may be added due to several reasons (e.g sporadic task arrivals). In that case the scheduling policy must continue to allocate sufficient processing time to reach the mandatory probability of the task. To service the sporadic tasks, a fraction of the optimized probability time should be used. Depending on the availability of processing time, the execution of a task may proceed in order to improve the probability of successful execution. Thus, the system continues to operate correctly as the load increases, but its performance degrades gracefully. When all the optimized parts have been allocated, the scheduler does not accept other jobs, since the mandatory times must be preserved.

The implicit assumption that the performed computations are *monotone* (the result quality does not deteriorates as computation proceeds) is made. Given $\Pr_{m_i}$, $\Pr_{o_i}$, the corresponding computation times C_{m_i} (*mandatory*) and C_{o_i} (*optimized*) are easily derived from the PDF definition.

The quantities $C_{oc\_max_i} = C_{o_i} - C_{m_i}$, $i = 1, 2, \ldots, N$ are the *optimized computation times* that the schedule must allocate partially (totally, if sufficient computation time exists) on the m processing clusters in order to optimize the quantity

$$\sum_{i=1}^{N} K_i \cdot [V(T_i, C_{m_i} + C_{OC_i}) - V(T_i, C_{m_i})] \qquad (1)$$

where $C_{OC_i} \leq C_{oc\_max_i}$ is the part of the task's T_i optimized time assigned by the schedule for execution. The factor K_i is dependent both on the task's criticality and on the slope of the task's PDF curve; it is of the form $\gamma_i \cdot D + \mathcal{K}_i$, where γ_i is a proportionality constant that expresses the significance of the curve's slope for the particular task and $\mathcal{K}_i$ expresses the task's criticality. Tasks with curves having fast convergence to $\Pr_{o_i}$ in $[C_{m_i}, C_{o_i}]$ must be preferred since the optimized probability can be obtained with smaller computation time. The slope of the PDF curve is approximated by making approximations to the density $f_{X_i} = F'_{X_i}$ at every k (e.g. $k = 5$) points in the interval (C_{m_i}, C_{o_i}) starting with C_{m_i}. That is

$$f_{X_i}(k_i) = F'_{X_i}(k_i) \approx \frac{F_{X_i}(k_i + dc) - F_{X_i}(k_i)}{dc}$$

where k_i is the point where the density is approximated and dc is the distance between two consecutive points at which F_{X_i} was evaluated (clearly $k_{i+1} = k_i + dc$). Then the sum

$$D = \frac{\sum_{j=0}^{n} F'_{X_i}(C_{m_i} + j \cdot k \cdot dc)}{n}$$

(where $n = \lfloor \frac{C_{o_i} - C_{m_i}}{k \cdot dc} \rfloor$) indicates the speed of the convergence (larger means faster convergence).

The expression $V(T_i, C_{m_i}+C_{OC_i})-V(T_i, C_{m_i})$ expresses the benefits obtained from the allocation of additional C_{OC_i} time to the task. In order to compute this quantity every task must be considered separately, since it represents the value of each intermediate result of a task. Thus, at every time point the probability with which a task produces intermediate results and the quality of those results is considered. The value curves that obtained in this way, can be very different from the probability distribution ones. This is due to the fact that the probability distribution curve expresses only the probability of completion up to a time point and not the value of the produced results up to that point.

3. PROPOSED SOLUTION

The problem of finding effective schedules for the N periodic tasks can be divided into three subproblems: balanced distribution of the workload among the processing clusters, optimal allocation of the available computation time on each processing cluster to its tasks and construction of the schedule on every processing cluster (i.e the assignment of each task's allocated time to specific cluster's processors).

3.1. Workload distribution

For the allocation of tasks, the rate-monotone first-fit algorithm [3] can be used, according to which the task set is sorted by increasing periods and it is assigned to the clusters on a first-fit basis. A task *fits* on a cluster if it and those already assigned to the cluster can be feasibly scheduled with the rate-monotone algorithm. In the decision of whether a task fits on a cluster only its mandatory time is considered. If n_a tasks have already been assigned to a cluster and their mandatory parts have a total *mandatory utilization factor* $u = \sum_{i=1}^{n_a} \frac{c_{m_i}}{P_i}$ then the (n_a+1)th task can be assigned if $u + \frac{c_{m_{n_a+1}}}{P_{n_a+1}} \leq (n_a+1)(2^{\frac{1}{(n_a+1)}}-1)$, where c_{m_i} is the expectation of X_i ($c_{m_i} = E[X_i]$). Under this condition the existence of a feasible schedule is guaranteed. Since the above inequality indicates that the mandatory utilization factor is always less than 1 ($\ln 2 < n_a(2^{\frac{1}{n_a}}-1) < 1$, $\forall\ n_a > 1$) there always remains time for the optional parts.

3.2. Time allocation

After the task allocation is accomplished the schedule for each processing cluster can be constructed independently. That schedule must have the following properties:

a. At any time instant each processor executes at most one task.

b. Each task is assigned processing time only within its feasibility interval.

c. The completion of the execution of all the tasks at least until the points where their mandatory probabilities Pr_{m_i} are reached must be guaranteed.

d. After the satisfaction of property (c), any remaining time on the $M \star m_c$ processors is given to the tasks with the larger criticality in order to maximize expression 1.

Let $r_i = \frac{PC}{P_i}$ be the number of instances of task i in the planning cycle. The problem of scheduling N periodic tasks in one planning cycle can be transformed into an equivalent one of scheduling $N' = \sum_{i=1}^{N} r_i$ tasks in the same interval. To do this, for every task T_i, $i = 1, 2, \ldots, N$, each of its r_i instances can be treated as a separate task with ready time for the k'th instance, $k \cdot P_i$ and deadline $(k+1) \cdot P_i$ (feasibility interval $[k \cdot P_i, (k+1) \cdot P_i)$). Denote by $T_1', T_2', \ldots, T_{N'}'$ the N' derived tasks. The ready times of the N' tasks are sorted in increasing sequence. The sorted times split the planning cycle into $k'(k' < N')$ intervals (six such intervals, I_1 to I_6 are shown in figure 1).

This phase can be carried out optimally by using a linear programming formulation. Denote by C_{i_j} the computation time assigned by the schedule to task T_i' in the interval $j (1 \leq j \leq k')$ of length W_j. A matrix C with entries C_{i_j} (which give the optimal processing time in each interval j and for every task T_i') is derived in this phase, with $\sum_{j=1}^{k'} C_{i_j} = C_{m_i} + C_{OC_i}$. This matrix does not describe the processor on which a task is going to run and the exact time of its execution.

A linear programming formulation will be used to find the optimal allocations. The expression 1 must be maximized under the following constraints:

- The assigned time for every task T_i' in the planning cycle (at all the k' intervals) must be less than its optimized execution time C_{o_i} and greater than its mandatory execution time C_{m_i}.

  $$\sum_{j=1}^{k'} C_{i_j} \leq C_{o_i} \text{ and } \sum_{j=1}^{k'} C_{i_j} \geq C_{m_i},$$

  $i = 1, 2, \ldots, N'$

- Each task is assigned processing time only within its feasibility interval. Thus, $C_{i_j} = 0$ for each interval of T_i', $i = 1, 2, \ldots, N'$.

- The assigned processing time by the schedule on each interval must not be greater than the time available at that interval on all the processing elements of the cluster,

  $$m_c \cdot W_j \geq \sum_{i=1}^{N'} C_{i_j}, \; j = 1, \ldots, k'$$

  Note that the schedule is constructed for each cluster separately.

- Each task is assigned on at most one processor at any time,

  $$0 \leq C_{i_j} \leq W_j, \qquad (2)$$

  where $j = 1, \ldots, k'$, $i = 1, \ldots, N'$. Actually, the above constraints ensure that the allocated time to every task, at every interval can be fullfilled by servicing it with a single processor at a time (not necessarily the same).

By solving the above linear program the optimal C_{i_j} entries are obtained. In order to prevent implementation problems, these are rounded to the largest integer.

The complexity of one efficient algorithm for linear programming [1] is $O((n + m)m^2 + (n + m)^{1.5} m)$ where n is the number of inequalities and m is the number of variables. Replacing for the particular problem $n = 2N' + k' + 2k'N', m = k'N'$ and after simplification, a complexity of $O(k'^3 N'^3)$ can be obtained for the particular problem.

3.3. Schedule Construction

During the time allocation phase the parameters C_{i_j} for the optimal processing times in each interval j and for every task T_i' are derived and are rounded appropriately. The purpose of this phase is to construct the schedule by assigning these times to specific processors. Clearly, in the considered case of serial tasks the assignment of the same task on more than one processor at the same time must be avoided.

This phase constructs the final schedules for each of the k' intervals separately. Consider the interval I_j of length W_j (all the others are treated similarly). Let $T_1', T_2', \ldots, T_k'$ be the k of the N' tasks that are allocated non-zero execution time C_{i_j}, $i = 1, 2, \ldots, k$ in the I_j interval (the renumbering of the tasks does not affect the generality of the approach). Clearly, $C_{i_j} \leq W_j$ (by the constraint expressed with equation 2).

Two cases can now be considered. The first one is when $k \leq m_c$. This case is trivial since we can assign each task for execution on its own processor. In order to treat the other case ($m_c < k$) we use a simple and efficient heuristic. Although this heuristic does not yield optimal schedules, its complexity is linear with the number of tasks and it yields at most one preemption for each task. This heuristic works as follows (for a w_j interval): Let the tasks be numbered $T_1', T_2', \ldots, T_k'$ and the processors $p_1, p_2, \ldots, p_{m_c}$. Assign task T_1' to p_1. Task T_2' continues from the point T_1' ended and in the case that it does not fit in p_1 the remaining time is allocated starting from the start of W_j at p_2. Since $C_{i_j} \leq W_j$, $\forall\, i, \forall\, j$, (from equation 2) overlap of execution on two processors at the same time is guarranted to be avoided. Further, since all the available time on the processors is used and by the time allocation phase we have $m_c \cdot W_j \geq \sum_k C_{i_j}$ the construction of the schedule by the heuristic is guaranteed.

4. CONCLUSIONS AND FUTURE WORK

We have presented a three-phase process for the construction of effective multiprocessor schedules for repetitive real-time tasks. That process considers a wide range of important parameters in order to construct those schedules. Specifically, it considers for every task its criticality, the distribution of its execution time, the value it has for a particular application and the precedence constraints between the tasks. During the first phase the rate-monotone first-fit algorithm is applied. The second phase is carried out optimally by using a linear programming formulation, while for the third phase, the application of a new algorithm is proposed.

In the future, attempts to make any possible improvements at these phases will be made. Then the process will be applied to the construction of effective schedules during the design and production of distributed real-time applications composed of repetitive tasks (e.g transportation of large multimedia objects between the nodes of a distributed authoring system).

REFERENCES

1. N. Karmarker, *A new polynomial-time algorithm for linear programming.* Combinatorica, 4(4) (1984), 373-395.
2. Jane W.S. Liu, Kwei-Jay Lin, Wei-Kuan Shih, Albert Chuang-Shi, Jen-Yao Chung, Wei Zhao, *Algorithms for Scheduling Imprecise Computations.* IEEE Computer (1991), 58-68.
3. C. L. Liu and J. W. Layland, *Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment.* J. ACM, 20(1) (1973), 46-61.
4. S. Papadimitriou, A. Kameas, P. Fitsilis, G. Pavlides, *A new compression technique for tools that use data-flow graphs to model distributed real-time applications.* Proceedings of the 5th international conference on software engineering & its applications, Toulouse December 7-11, 1992.
5. S. Papadimitriou, A. Kameas, G. Pavlides, *The derivation of near-optimal CTMC models for real-time tasks modelled by data-flow graphs.* TR 92.09.14, Computer Technology Institute, 3 Kolokotroni str., Patras, Greece, 1992.
6. D. Peng and K.G. Shin, *Modeling of concurrent task execution in a distributed system for real-time control.* IEEE Trans. on Computers, C-36(4) (1987), 500-516.
7. K. S. Trivedi, *Probability and Statistics with Reliability, Queueing and Computer Science Applications.* Prentice-Hall, 1982.
8. Michael H. Woodbury, Kang G. Shin, *Performance Modeling and Measurement of Real-Time Multiprocessors with Time-Shared Buses.* IEEE Trans. on Computers, C-37(2) (1988), 214-223.