

# A Reliability-aware Value-based Scheduler for Dynamic Multiprocessor Real-time Systems

S. Swaminathan and G. Manimaran  
Dependable Computing & Networking Laboratory  
Dept. of Electrical and Computer Engineering  
Iowa State University, Ames, IA 50011, USA  
{swamis,gmani}@iastate.edu

**Abstract**—The growing needs for building complex real-time applications coupled with advancements in computing technology signify the importance of developing efficient algorithms for dynamic real-time systems. Dynamic real-time systems need to be designed not only to deal with expected load scenarios, but also to handle overloads by allowing graceful degradation in system performance. Value-based scheduling is a means by which graceful degradation can be achieved by executing critical tasks that offer high values/benefits/rewards to the functioning of the system. In value-based scheduling, each task is associated with a reward and penalty that is offered to the system depending on whether the task meets or misses its deadline. Some value-based scheduling defines “performance index” that captures not only the reward/penalty parameters, but also the trade-off between schedulability and reliability. In this paper, we propose a reliability-aware value-based dynamic scheduling algorithm for multiprocessor real-time systems, whose objective is to maximize the performance index of the system. The proposed scheduler selects a suitable redundancy level for each task so as to increase the performance index of the system. We have conducted simulation studies to evaluate the effectiveness of the proposed scheduler and its variants for a wide range of values of system parameters. Our studies show that the proposed scheduler offers a high “value ratio” (defined with respect to a near-optimal baseline algorithm) for non-trivial task sets.

## I. INTRODUCTION

In real-time systems the correctness of the results depend not only on the logical correctness, but also on the time at which the results are produced [1]. Multiprocessors and multicomputers based systems have emerged as a powerful computing means for real-time applications such as avionic control and nuclear plant control, because of their capability for high performance and reliability. The problem of scheduling real-time tasks in such systems is to determine when and on which processor a given task is executed. The allocation and scheduling can be performed either statically or dynamically. In static algorithms [1], the assignment of tasks to processors and the time at which the tasks start execution are determined *a priori*. Static algorithms are often used to schedule periodic tasks and are not applicable to aperiodic tasks whose arrival times and deadlines are not known *a priori*. Scheduling such tasks requires a dynamic scheduling algorithm [2], [3].

In dynamic scheduling, when a new set of tasks arrive at the system, the scheduler determines the feasibility of scheduling these new tasks without jeopardizing the guarantees that have been provided for the previously scheduled tasks. When the system is not overloaded, it is possible to ensure that all the tasks meet their deadlines by employing efficient scheduling algorithm. However, when the system is overloaded, it is impossible to meet the deadlines of all the tasks, which means that some tasks may have to be rejected and hence missing their dead-

lines. In such an overloaded system, it is often desirable to maximize the overall utility of the system by employing value-based scheduling. *Value-based scheduling* is a scheduling paradigm that schedules tasks such that overall value (i.e., utility) of the system is maximized. It is assumed that each task offers certain “value” to the system if it meets its deadline, otherwise, certain “penalty” is incurred. Thus, value-based scheduling is a decision problem involving the choice of a collection of services (tasks) to execute so that the “best possible” outcome ensues. The process of fixing the value for each task depends on the application and is a really complex process [4] which is beyond the scope of this paper. Examples of value function include Performance Index [5] and Value-Density [6]. These value functions capture the utility of tasks in different context with the former value function captures the importance of fault-tolerance while the latter emphasizing on the value for a computation time.

In this paper, we consider a fault-tolerance related value function that captures the tradeoff between schedulability and reliability, and propose a scheduling algorithm to maximize the overall performance of the system. This schedulability-reliability tradeoff in the context of value-based scheduling is motivated as follows. Due to the critical nature of tasks in a real-time system, it is essential that every task admitted in the system completes its execution even in the presence of faults. Therefore, fault-tolerance is an important requirement in such systems [5], [7], [8], [9]. Scheduling multiple versions of tasks on different processors provides fault-tolerance. In a fault-tolerant scheduling, increasing the redundancy level of a task decreases the probability that the task will fail after being accepted, at the same time decreasing the schedulability of the system, i.e., the number of tasks that get scheduled. This implies tradeoffs involving the fault-tolerance of the system, the rewards provided by guaranteed tasks that complete successfully, and the penalties due to tasks that fail after being guaranteed or that fail to be guaranteed.

In this paper, we consider the problem of value-based scheduling of a set of real-time tasks in a dynamic multiprocessor system, where the value function captures the reward (or penalty) obtained by meeting (or missing) a task’s deadline. We propose a dynamic scheduling algorithm that finds the right redundancy level for each task, satisfying the timing constraints, such that the performance index of the system is maximized.

## A. Related Work and Motivation

The problem of obtaining an optimal schedule for a set of real-time tasks in a multiprocessor system is NP-complete [10]. Moreover, it was shown in [11] that there does not exist an algorithm for optimally scheduling dynamically arriving tasks with or without mutual exclusion constraints. These negative results motivated the need for heuristic approaches for solving scheduling problems. Various heuristic algorithms, such as Myopic scheduling [2] and its variation ParMyopic [12], were proposed to solve the dynamic multiprocessor scheduling problems. The above said results also hold to the value-based scheduling problem addressed in this paper as being a derivative of dynamic multiprocessor scheduling. In recent years, there are some work on value-based scheduling algorithms [6]. These work assume a generic utility values as value functions. Also, there are some work which focuses on value-based scheduling with the value function aimed at maximizing the reliability of the system without sacrificing the schedulability.

*Spare-capacity scheduling:* The primary issue in the fault-tolerant related value based scheduling is to find the correct redundancy level for each task so that the system's reliability is maintained with high schedulability. In [9], an algorithm was proposed with fault-detection and location capabilities for real-time systems based on the myopic algorithm. The objective of the algorithm was to improve the schedulability and to use the spare capacity of idle processors for fault-detection and location. Thus, the algorithm's emphasis is on schedulability of the system than reliability. Further, this algorithm does not consider the value parameters into account for making a scheduling decision. Such an algorithm is inadequate when tasks have different rewards (or penalties) for their successful executions (or failures).

*Performance index scheduling:* In this context, an approach is proposed in [5] to determine the redundancy level for a given set of tasks so as to maximize the total performance index which is a performance-related reliability measure. This approach is used in dynamic planning based scheduling where decisions are taken as tasks arrive. In such a system with  $m$  processors,  $n$  tasks arrive at a particular point in time. The approach tries to find the best redundancy level for the tasks such that the overall performance index is maximized. Once a task redundancy level is determined, a task is said to be guaranteed if the given number of copies of the task are all scheduled to complete before the task's deadline. Suppose a task  $T_i$  provides a reward  $V_i$ , if it completes successfully once it is guaranteed, a penalty  $P_i$  if it fails after being guaranteed, and penalty  $Q_i$  if it is not guaranteed. Let  $R_i$  be the reliability of a task and  $F_i$  be its failure probability, where  $R_i = 1 - F_i$ . The redundancy level of a task  $T_i$  and the failure model of the processors affect  $R_i$ . The performance index  $PI_i$  for task  $T_i$  is defined as

$$PI_i = \begin{cases} V_i R_i - P_i F_i & , \text{if } T_i \text{ is guaranteed} \\ -Q_i & , \text{if } T_i \text{ is not guaranteed} \end{cases}$$

Then, the performance index for a task set containing  $n$  tasks is defined as

$$PI = \sum_{i=1}^n PI_i$$

In [3], another dynamic fault-tolerant scheduling algorithm was proposed, where the myopic algorithm was modified to schedule tasks aiming to maximize the overall  $PI$  of the system. However, the algorithm by its nature selects the maximum redundancy level (an input parameter) as the chosen redundancy level for each task. Such an approach of considering only one task at a time for determining the redundancy level for the task can lead to poor overall performance index, which is the motivation for our work.

The objective of our work is to propose an efficient dynamic scheduling algorithm that maximizes the overall performance index of the system is maximized by executing the right redundancy level for each task in the system. The fundamental difference in our algorithm is that the task schedule is extended by more than one task at a time, which can lead to better  $PI$  than the algorithms that consider only one task at a time.

The rest of the paper is organized as follows: Section II states the system model and definitions. Section III describes our proposed scheduling algorithm. The performance results of the proposed algorithm is explained in Section IV. Section V concludes the paper.

## II. SYSTEM MODEL AND TERMINOLOGY

- The system consists of  $m$  processors having identical reliability.
- Each task  $T_i$  is characterized by ready time ( $r_i$ ), worst-case computation time ( $c_i$ ), deadline ( $d_i$ ), reward value ( $V_i$ ) and penalty values ( $P_i$  and  $Q_i$ ). Further tasks are assumed to be aperiodic and non-preemptable.
- A task  $T_i$ 's earliest start time is denoted by  $EST(T_i)$ , which is computed as the maximum of its processor available time, resource available time and ready time.
- Let  $p$  be the reliability of a task (with one version),  $T_i$ , executing on a single processor. Then, the reliability ( $R_i$ ) of the task with  $r$  versions is given by  $R_i = 1 - (1 - p)^r$ .
- A task  $T_i$  is said to be feasible only if  $EST(T_{i1}) + c_i \leq d_i$ , where  $EST(T_{i1})$  denotes the earliest start time for task  $T_i$  with one version.
- Let  $A$  be the set of tasks in the feasibility check window, of size  $K$ , then the partial schedule obtained is *strongly feasible* if all the schedules obtained by executing the current schedule to the unscheduled task set is also feasible. i.e.,  $\forall i, T_i \in A, EST(T_{i1}) + C_i \leq d_i$
- The performance of a task  $T_i$  for a redundancy level  $j$  is defined as :

$$PI_{ij} = \begin{cases} V_i R_i - P_i F_i, & \text{if } j > 0 \\ -Q_i, & \text{if } j = 0 \end{cases} \quad (1)$$

where,  $F_i = (1 - p)^j$  and  $R_i = 1 - F_i$ .

## III. PROPOSED RELIABILITY-AWARE VALUE-BASED DYNAMIC SCHEDULER

In this section, we present a reliability-aware dynamic scheduling algorithm that aims at maximizing the overall value (performance index) of the system by finding the right redundancy level for each task. The proposed algorithm is a variant of

myopic algorithm. The myopic algorithm works on extending on a partial schedule  $P$ , by one task at a time. Initially it considers all the  $K$  tasks (in the set of tasks called feasibility check window) and checks if they are feasible. If it is strongly feasible then it extends the schedule by task  $T_i$  that offers the smallest heuristic value  $H(\cdot)$ , out of the all the tasks in its feasibility check window. If the current schedule is not strongly feasible, the algorithm backtracks and selects the task that offers the next best heuristic value.

The proposed scheduling algorithm, as given in Figure 1, starts of similar to that of the myopic scheduling algorithm, by considering  $K$  tasks in the feasibility check window,  $A$ . If the current schedule is strongly feasible then the algorithm selects the right combination of task with their appropriate redundancy level by the *Combination Selection Algorithm*, such that the overall performance index of the task set in  $A$  is maximized and the total number of tasks (including the redundant tasks) is less than or equal to  $L$ , the maximum schedule extension size. Then, the scheduling order of the tasks is determined by the *Order Selection Algorithm*, and the schedule is extended with specified order of the selected combination. If the selected order for a combination is not feasible to schedule, then task schedule is extended with different order until a feasible order is obtained. If all the orders are exhausted, then the task schedule is extended with next best combination that offers the next best  $PI$  value with the order being selected for the new combination. This process is being repeated until a feasible schedule is obtained or until all combinations are exhausted. The key difference between the proposed scheduler and the conventional myopic scheduler is that in the former, the schedule is extended by  $L$  tasks at a time, while it is extended by only one task at a time at the latter. Here, the value of  $L$  can be less than or equal to or more than the number of processors in the system (say  $m$ ). The value of  $L$  affects the effectiveness of the algorithm, with increased value of  $L$  reducing the total number of scheduling decisions. Similarly, the value of  $K$  also affects the effectiveness of the algorithm, as  $K$  represents the look-ahead nature of the scheduler. Higher the value of  $K$ , higher is the number of tasks considered in the search and hence the chance of the scheduler to come up with a task set giving a better value is increased.

#### A. Combination Selection Algorithm

The objective of the combination selection algorithm is to come up with right combination of tasks with their appropriate redundancy levels such that overall performance index of the system  $PI$  is maximized. Thus, the combination selection algorithm takes the task in the feasibility check window as input and extends the schedule upto  $L$  tasks, such that the selected combination gives the maximum  $PI$ . The combination selection algorithm can be considered as a search algorithm, which searches for the correct combination of redundancy levels of tasks that leads to the maximum  $PI$ , which are feasible to schedule. Hence the issue in this algorithm is the depth of search to be carried for every task-set scheduling to attain a good combination yielding the best or near-best  $PI$ . For example, if the number of tasks within the feasibility check window  $A$  is  $K$  and the maximum redundancy level of a task is  $r$ , then the combination selection at the maximum must search all combinations (of

#### Value-Based Scheduler()

Input: Task Set to be scheduled.

Output: Feasible Schedule maximizing the  $PI$  or failure.

1. Tasks (in the task queue) are ordered in non-decreasing order of deadline.
2. Start with an empty partial schedule.
3. Check if the set of tasks (set  $A$ ,  $|A|=K$ ), constituting the feasibility check window is strongly feasible.
4. **If** (strongly feasible)

##### Repeat

- *Combination Selection Algorithm*:  $\forall T_i \in A$ , find  $j (=r_i)$  for each task such that:

$$PI = \sum_{i=1, j=1}^{K, r} PI_{ij} \text{ is maximized.}$$

##### • Repeat

*Order Selection Algorithm*:

- $\forall T_i \in A$ , compute a heuristic value  $H(\cdot)$  for the selected redundancy level  $r_i$
- sort the tasks based on their  $H(\cdot)$ .
- The sorted order represents the dispatching order of the task set.

**Until** (Termination Condition 1)

**Until** (Termination Condition 2)

5. **if** (a feasible combination is obtained)
  - $\forall T_i \in A$ , extend the schedule by  $T_i$  with  $r_i$  versions
  - Move the feasibility check window  $A$ , to next  $K$  tasks.
- else**
  - Backtrack to the previous partial schedule.
  - Extend the schedule with the next best combination.

*Termination Condition 1*: Feasible order is obtained or the maximum number of orders have been searched.

*Termination Condition 2*: Feasible combination is obtained or the maximum number of combinations have been searched.

Fig. 1. Reliability-Aware Value-Based Scheduling Algorithm

order  $O(r^K)$ ) to guarantee an optimal solution. We propose two combination selection algorithms, *Exhaustive Search* and *Reduced Search*, in this section that can come up with an optimal or near-optimal solution with the first algorithm incurring high computation cost for optimal solution whereas the second algorithm incurs less computation cost for near-optimal solution.

#### A.1 Exhaustive Search Algorithm

The exhaustive search algorithm comes up with the best combination of tasks after an exhaustive search of all task combinations (within the feasibility check window). The algorithm is as follows:

1. Let the number of the tasks in the feasibility check window be  $K$  and the maximum redundancy level to be considered be  $r$ .
2. Calculate the performance index for all combinations of task sets, such as  $n_1$  version of  $T_1$ ,  $n_2$  version of  $T_2$ , ...,  $n_K$  versions of  $T_K$ , where  $0 \leq n_i \leq r$  and  $1 \leq i \leq k$  and  $\sum_{i=1}^k n_i = L$ , where  $L$  is the maximum schedule extension size.
3. Select the combination of  $(i, j)$  for each task  $T_i$  and its redundancy level  $j$  that offers the best overall performance index  $PI$ , where:

$$PI = \sum_{i=1, j=1}^{K, r} PI_{ij}$$

Though the algorithm is always guaranteed to come up with an optimal combination for the set of  $K$  tasks at any given time, its time complexity is  $O(r^K)$ , which makes it very expensive.

## A.2 Reduced Search Algorithm

This algorithm tries to reduce the search space by reducing the number of redundancy levels to be searched for each task, so that the number of combinations searched is less than that done by the *Exhaustive Search Algorithm*. This algorithm reduces the search space by taking smart decisions on the maximum redundancy level to be searched for each task. The algorithm finds the maximum redundancy level for each task, by finding the right redundancy level beyond which the gain in performance index for increasing redundancy is not high, which is determined by  $\Delta$  (an input parameter to the system). The reduced search algorithm is as follows:

1. Let the number of the tasks in the feasibility check window be  $K$  and the maximum redundancy level to be considered be  $r$ .
2. For each task  $T_i$ , fix the maximum level of redundancy level to be searched as  $j(=r_i)$ , such that  $PI_{i(j+1)} - PI_{ij} \leq V_i * \Delta/100$ , where  $\Delta$  is a fixed value (such as 1).
3. Calculate the performance index for all combinations ( $r_1 * r_2 * \dots * r_K$ ) of task sets, such as  $n_1$  version of  $T_1$ ,  $n_2$  version of  $T_2, \dots$ ,  $n_K$  versions of  $T_K$  (where  $0 \leq n_i \leq r_i$  and  $1 \leq i \leq k$ )

To illustrate the working of this algorithm let's consider the following example. Consider a task  $T_i$  for  $\Delta = 0.02$ , with following characteristics:  $\langle v_i, p_i, q_i \rangle = \langle 10, 100, 5 \rangle$  and with the reliability of the processor  $p = 0.9$ . For this task set, the PI values for different redundancy levels are:  $PI_{i1} = -1, PI_{i2} = 8.9, PI_{i3} = 9.98, PI_{i4} = 9.998$ . Since for this task,  $PI_{i4} - PI_{i3} = 0.018 \leq (v_i * \Delta = 10 * 0.02)$ , the algorithm eliminates all combinations of task  $T_i$  with redundancy level more than 3 in its search. The intuition behind pruning search in this manner is that the reward for searching for more than a redundancy level of 3 for the given is not high (as determined by the  $\Delta$  factor). Though, the reduced search algorithm need not always come up with a best possible PI for every task set, it incurs a less computation cost for every combination selection process for a task set. The amount of search executed and how close the solution is to the best combination offering the best PI, is controlled by the  $\Delta$  parameter. With increased value of  $\Delta$ , the number of redundancy levels searched for each task is also increased, thereby increasing the search space and improving the overall PI.

### B. Order Selection Algorithm

The objective of the order selection algorithm is to order the task execution sequence, for a given combination of a task set, such that selected combination of task set is feasible to schedule. The order selection algorithm is an important component in the proposed scheduler as the feasibility of scheduling a selected combination depends on the effective dispatching of the tasks. This is because in a dynamic scheduling environment the task dispatch order is not a trivial decision as it involves many factors such as the earliest available processor time, task ready time and earliest resource availability time. So the task dispatching order must take into account these constraints to increase the chances of meeting tasks deadlines. A mere dispatching of task with highest PI with their redundancy level is not desirable, as this can introduce "holes" (idle processor time) in the schedule,

TABLE I  
EXAMPLE TASK SET

task	$r_i$ $c_i$ $d_i$	$V_i$ $P_i$ $Q_i$
$T_1$	0 5 10	10 100 1
$T_2$	2 5 10	20 150 2
$T_3$	0 5 10	10 80 0.8
$T_4$	3 5 20	30 200 3
$T_5$	0 5 20	8 50 0.5

leading to missing of deadlines of many other tasks. In this section, we propose two order selection schemes, *Deadline Ordering* and *Heuristic Ordering*, which orders the selected task set combination based on deadline and heuristic functions respectively.

#### B.1 Deadline Ordering Scheme

The deadline ordering scheme determines the order of task execution, based on how close the tasks are to their deadline, for all the tasks in a given task combination. So, tasks having smaller deadlines will be scheduled first. The disadvantage of this scheme is that it does not take into account the Earliest Start Time (*EST*) of a task into account which can introduce a lot of holes. Further failure to consider *EST* can result in poor schedulability when the deadline for all the tasks in a task set are around the same time.

#### B.2 Heuristic Ordering Scheme

The heuristic ordering scheme overcomes the disadvantage of the deadline ordering scheme by taking into account the *EST* of a task for finding the appropriate task dispatch order. This scheme uses a heuristic function which captures the importance of deadline and also *EST* of a task, which is given by:

$$H(T_i) = d_i + W * EST(T_i)$$

The  $W$  factor determines the weight given to the *EST* of a task. So, lower the value, means lesser the importance given to the *EST*. So, the tasks are ordered based on their heuristic value  $H(\cdot)$  and dispatched in the sorted order. As explained in the *FT-Myopic algorithm*, if a selected order for a task set is not feasible, then a different order can be obtained by varying the  $W$  parameter. A special case of this scheme for  $W = 0$  is the deadline ordering scheme.

### C. Illustration for the proposed value-based scheduler

Consider a real-time system with four processors. Assume that five tasks  $\{T_1, T_2, T_3, T_4, T_5\}$  are to be scheduled with all tasks having the same computation time and deadline but different reward values, as given in Table 1. Assume the reliability of the processor to be 0.9,  $K = 3$ ,  $L=4$  and  $r=3$ .

Then the algorithm behaves as follows: First the combination selection algorithm selects  $T_2$  and  $T_3$  out of the first 3 tasks ( $T_1, T_2, T_3$ ). Then the ordering algorithm (Heuristic comes up with schedule of  $T_{21}$  on  $P_1, T_{22}$  on  $P_2, T_{31}$  on  $P_3, T_{32}$  on  $P_4$ . Then, in the next schedule, out of 3 tasks ( $T_1, T_4$  and  $T_5$ ),  $T_1$  and  $T_2$  are selected with a redundancy level of 2 each. The order of dispatch is of  $T_1$  first (if *EST* or *Deadline ordering* scheme

TABLE II  
SIMULATION PARAMETERS

Task Parameter	Value Range
$C_i$	10 ... 40
$V_i$	50 ... 200
$P_i$	(2 ... 4) * $V_i$
$Q_i$	$V_i/10$

is used) and  $T_4$  next and then  $T_5$  is scheduled with redundancy level of 4, as no other task is left to schedule.

#### IV. PERFORMANCE EVALUATION

In this section, we first discuss the method adopted for task generation and simulation and then present the simulation results.

##### A. Task Set Generation for Value-based Scheduling

The objective of the proposed scheduling algorithm in this paper is to obtain a feasible schedule for a set of tasks, if such a schedule exists, such that the overall value (performance index) of the system is maximized. Heuristic algorithms cannot be guaranteed to achieve this everytime but one heuristic algorithm can be considered better than another, if it obtains a better value and a feasible schedule, given a schedulable task set. This is the basis of our simulation study. Hence, in our simulation study we generate a tightly schedulable task set for  $m$  processor system, with appropriate redundancy level determined during the task generation phase, based on its value. Thus, the task generator generates tasks to guarantee almost maximum utilization of the processors and also determines the redundancy level of each processor based on its value. So, higher the value of a task, higher its redundancy level. The inputs to the task generator are given in Table 2.

The schedule generated by the task is in the form of a matrix  $M$  which has  $S$  (given by schedule length) columns and  $r$  rows. Each row represents a processor and column represents the time unit. The task generator starts with an empty matrix, then after selecting the value parameters and computation time of a task are chosen randomly using a uniform distribution between the values as shown in Table 2. Then, it determines the redundancy level for each task based on its value, such that it is proportional to the value of the system and fills up the appropriate columns and rows of the matrix. Thus, the task generator generates task until the utilization left is less than the minimum computation time of a task. Here, the task's deadlines are chosen to be their finish time of the generated task, which gives very little leeway for the scheduler. We believe that the task set generated by this task generator can evaluate various heuristics in a rigorous manner.

##### B. Simulation Method

In our simulation,  $N$  schedulable task sets are generated and the values obtained for the schedule during task generation is recorded. Performance of the the proposed algorithms, ES-D (Exhaustive Search and Deadline Ordering), ES-

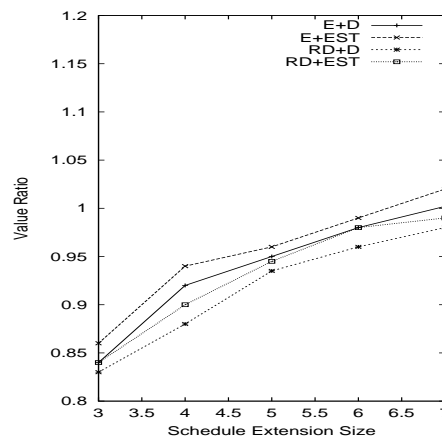


Fig. 2. Effect of  $L$

EST (Exhaustive Search and Earliest Start Time Ordering), RS-D (Reduced Search and Deadline Ordering) and RS-EST (Reduced Search and Earliest Start Time ordering) was evaluated based on the following metric:

- *ValueRatio*: It is defined as the ratio of the value obtained by the scheduler to the value obtained during task generation. It must be noted this value can vary from 0 to 1 (and also above 1 some cases, because the value of the schedule obtained during task generation need not be optimal).

$$ValueRatio = \frac{Value_{sched}}{Value_{generator}} \quad (2)$$

where,  $Value_{sched}$  = Value obtained by the proposed scheduler,  $Value_{generator}$  = Value obtained during task generation, this is used as the baseline value for comparison.

Thus, all the four algorithms are evaluated for varying values of maximum schedule extension size (say  $L$ ), i.e., the maximum number of tasks with which a schedule is extended for every scheduling decision, the feasibility check size window ( $K$ ),  $\Delta$  (for the Reduced Search Algorithm). The simulation was run for 10 sessions of feasible task sets, each consisting of 50 tasks and each simulation was run 5 times and the averages are plotted.

##### C. Effect of Schedule Extension Size ( $L$ )

In this experiment, the efficiency of all the schedulers were tested for varying values of Schedule Extension Size,  $L$ . Recall that one of the motivations for our work is to extend the task schedule by more than one task at a time in order to improve the overall  $PI$ . The results of the experiment conducted for a 4-processor system with  $K$  of 4 is given in Figure 2.

As it can be seen from the figure, increase in  $L$  gives a better performance with respect to the value ratio. This is due to the fact that the scheduler can make a more effective scheduling decision, when it is allowed to extend its schedule with more tasks (as the number of redundancy levels that can be searched for each task will also be high, thereby leading a higher value). However, it must be noted that higher values of  $L$  requires more computation for making a scheduling decision as the scheduler's search space increases. Comparing the performance of

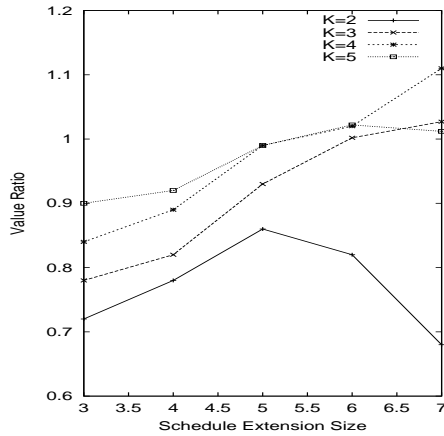


Fig. 3. Effect of  $K$  on 4-processor system for ES+EST Scheduler

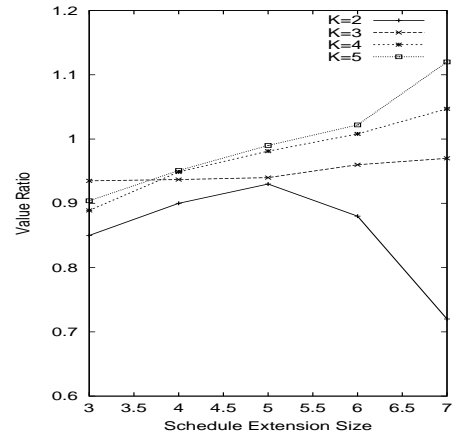


Fig. 4. Effect of  $K$  on 5-processor system for ES+EST Scheduler

the four algorithms it can be seen that the Exhaustive Search algorithms performing better than the reduced search counterparts. However, the decrease in value ratio due to the reduced search is not that high, hence there is a good reward for going for reduced search as it obtains a schedule that also has some good value ratio. It must be noted that in this simulation, the resource constraints have not been simulated, which would establish the superiority of EST ordering schemes over Deadline ordering schemes. However, still it can be observed that in the current simulation setup, the EST ordering algorithms do better than deadline ordering schemes as the amount of holes (idle processor time) created in the schedule is decreased.

#### D. Effect of Feasibility Check Window Size ( $K$ )

In this experiment, the efficiency of the ES+EST scheduler was tested for different values of  $K$  and for different values of  $L$ . The results of the experiment conducted for 4-processor and 5-processor system are given in Figure 3 and Figure 4. The effect of  $K$  was studied only for ES+EST scheduler as it is the one that gives best performance among all the schedulers and can measure the effect of  $K$  on *ValueRatio* effectively. The  $K$  is studied for varying  $L$  parameters as both of these parameters affect the scheduling performance together, as explained below.

As it can be seen from the figures, the increase in  $K$  increases the value ratio of the schedulers. This is due to the fact that the  $K$  represents the look-ahead nature of the scheduler and higher its value, the more effective is its scheduling decision, leading to higher value of the system. However, the performance gain obtained after increasing  $K$  from 4 to 5 is not really huge, hence a more complex search is not that rewarding for a small performance increase in this setup. Further, it can be seen from the figures that as for a given  $K$  PB (say 2 in the figures), the *ValueRatio* obtained decreases with increase in  $L$  beyond a value. For example, in the figures it can be seen that for  $K$  of 2, the *ValueRatio* obtained decreased beyond  $L=5$ . This is due to the fact that extending a task schedule with smaller number of tasks (determined by  $K$ ), each having more versions (determined by  $L$ ) yield less value. Thus, it is necessary that  $L$  parameter should be high if the  $K$  is also high to get a better per-

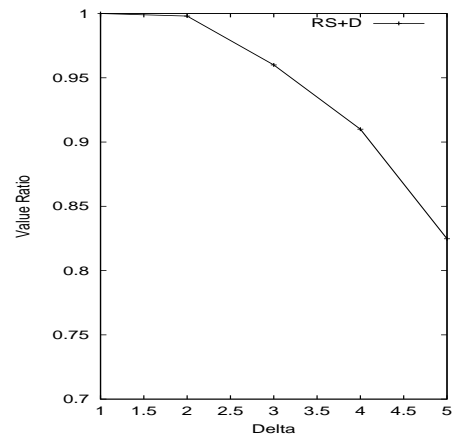


Fig. 5. Effect of  $\Delta$

formance. It must be noted that this problem of scheduling more versions of tasks due to small sized feasibility check window is eliminated by the reduced search algorithms as they determine the maximum redundancy level of a task and does not extend the schedule more than the determined levels, irrespective of the value of  $L$ .

#### E. Effect of $\Delta$

In this experiment, the efficiency of the RS+EST scheduler was tested for different values of  $\Delta$ . The results of the experiment, shown in Figure 5 conducted for 4-processor with  $L=7$ .

As it can be seen from the figure that with decreasing values of  $\Delta$ , the *ValueRatio* obtained increases. This is due to the fact that the search space increases with increase in  $\Delta$ , thereby leading a better performance. However, the performance gain obtained from  $\Delta = 3$  to  $\Delta = 1$  is not high. Hence, it can be clearly seen that the need for unnecessary searching is eliminated by the algorithm.

## V. CONCLUSIONS

In this paper, we propose an algorithm for value-based scheduling in multiprocessor real-time systems, which aims to

maximize the overall  $PI$  of the system. The proposed scheduler has two components: Combination Selection and Order Selection algorithms. We study the effectiveness of the proposed scheduling algorithm (for various combinations of the components) through simulation studies by comparing the value obtained by scheduling a feasible task set to the value generated during its generation, by varying  $K$  and  $L$  parameters. We find that a careful selection of  $K$  and  $L$  parameter pair can yield high performance. We find the reduced search algorithm maintains a good value ratio incurring less search cost, which is important for dynamic schedulers. Thus, we feel that the proposed dynamic value-based scheduler, can improve the overall utility ( $PI$  in this case) without having an a priori knowledge of the task value parameters.

#### REFERENCES

- [1] K. Ramamrithnam and J. Stankovic, "Scheduling Algorithms and operating systems support for real-time systems", *Proc. of IEEE*, vol. 82, no.1, pp.55-67, Jan. 1994.
- [2] K. Ramamrithnam, J. Stankovic, and P.F. Shiah, "Efficient Scheduling algorithms for multi-processor real-time systems", *IEEE Tran. Parallel and Distributed Systems*, vol. 1, no. 2, pp. 184-194, Apr. 1990.
- [3] J. Zhou, G. Manimaran and A.K. Somani, "A dynamic scheduling algorithm for improving performance index in multiprocessor real-time systems", *Proc. of ADCOMM'99*
- [4] A. Burns, D. Prasad, A. Bondavalli, F. Di Giandomenico, K. Ramamrithnam, J. Stankovic, and L. Strigini, "The Meaning and Role of Value in Scheduling Flexible Real-Time Systems", *Journal of Systems Architecture*, 1998.
- [5] F.Wang, K. Ramamrithnam, and J. Stankovic, "Determining redundancy levels for fault-tolerant real-time systems", *IEEE Trans. Computers*, vol. 44, no. 2, pp. 292-301, Feb. 1995.
- [6] S.A. Aldarmi and A. Burns, "Dynamic value-density for scheduling real-time systems", *Proc. the 11th Euromicro Conference on Real-Time Systems*, pp. 270-277, Jun. 1999.
- [7] S.Ghosh, R. Melhem, and D. Mosse, "Fault-Tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems", *IEEE Trans. Parallel and Distributed Systems*, vol. 8, no. 3, pp. 272-284, Mar. 1997.
- [8] A.K. Somani, and N.H.Vaidya, "Understanding Fault-tolerance and reliability", *IEEE Computer*, vol. 30, no. 4, pp.45-50, Apr. 1997.
- [9] K. Mahesh, G. Manimaran, C. Siva Ram Murthy, and A.K. Somani, "Scheduling algorithm exploiting spare capacity and task laxities for fault detection and location in real-time multiprocessor systems", *Jornal of Parallel and Distributed Computing*, vol. 52, no. 2, pp. 136-150, June 1998.
- [10] M.R.Garey and D.S. Johnson, "Computer and intractability, a guide to theory of incompleteness", W.H.Freeman Company, 1979.
- [11] M.L.Dertouzos and A.K.Mok, "Multiprocessor on-line scheduling of hard real-time tasks", *IEEE Software Engg.*, vol. 15, no. 12, pp. 1497-1506, Dec. 1989.
- [12] G. Manimaran and C. Siva Ram Murthy, "An efficient and dynamic scheduling algorithm for multi-processor real-time systems", *IEEE Trans. Parallel and Distributed Systems*, vol. 9, no. 3, pp. 312-319, Mar. 1998.